Windows Vista Network Attack Surface Analysis

Dr. James Hoagland, *Principal Security Researcher*
Matt Conover, *Principal Security Researcher*, Tim Newsham, *Independent Contractor*, Ollie Whitehouse, *Architect*
Symantec Advanced Threat Research

Abstract

A broad analysis was performed on the network-facing components of the Microsoft Windows Vista™ release version. Our analysis explores how it affects network security and how it differs from previous versions of Microsoft Windows. Windows Vista features a rewritten network stack, which introduces a number of core behavior changes. Windows Vista also introduces a number of new protocols, most importantly IPv6, its supporting protocols, and several IPv4 to IPv6 transition protocols. As a client operating system, Windows Vista will be widely deployed and as such is an important topic for security research. We studied the following protocols and technologies: LLTD, IPv4, IPv6, Teredo, TCP, SMB2 named pipes, MS-RPC, and the Windows Firewall. We also studied ARP, NDP, IGMP, MLD, ICMPv6, and UDP.

CONTENTS

I. Introduction

WINDOWS Vista™ is Microsoft's long anticipated, new client operating system. It is due to replace Windows XP as Microsoft's premier desktop operating system. Windows Vista represents a significant departure from previous Windows systems, both in terms of its emphasis on security and its many new features. As security has grown in importance, Microsoft has paid increasing attention to it, evidenced by the significant investment of resources that has been made. Windows Vista provides Microsoft with the opportunity to introduce security into the design process of the core operating system. Microsoft has also chosen Windows Vista as the platform on which to introduce many newly developed technologies.

The Windows Vista network stack is particularly interesting because many of its components are new. The TCP/IP network stack has been rewritten and represents a significant departure from previous versions of Windows. The new stack was written to allow easier maintenance, important new performance enhancements, and improved stability[55]. It integrates support for IPv6 and IPv4 into a single network stack, and provides IPv6 support in the default configuration for the first time in the history of Windows. Many other new protocols are implemented and supported in Vista, either as part of the network stack or as separate components of the Windows operating system. These new protocols support features such as topology discovery, serverless name resolution and NAT traversal. Even SMB, one of Microsoft's oldest technologies, received a revision with the introduction of the SMB2 variant. The amount of new code present in Windows Vista provides many opportunities for new defects. Each new protocol introduces its own set of security implications, which must be understood and considered.

Symantec evaluated the security-related aspects of the Windows Vista network stack[1]. Our investigation was broad and, in places, deep, aiming to provide key intelligence in a timely manner. Obviously, it is impossible to do a complete analysis of the entire Windows network-facing environment, so we focused on the most common configuration (initial installed state) and conducted only the most productive research, given the time investment required. We hope you find this report useful as a Windows Vista network reference: we hope you find value in both the detailed security analysis and in the broad overview. If you believe that some information presented in this paper is inaccurate, we would appreciate hearing from you.

We describe our testing environments in Appendix I. The majority of the results presented in this report are obtained by using the release build of Vista: the build that will be widely installed. However, the LLTD-focused results are obtained from build 5472, and the Teredo-focused results are from RC2. No Microsoft source code was used during this analysis, although public Microsoft documentation was used when it was available.

[1]This is the second edition of this Symantec Response whitepaper. The first edition[49], entitled *W*indows Vista Network Attack Surface Analysis: A Broad Overview, covered Windows Vista Beta 2 builds 5270, 5231, and 5384 and was released July 2006.

This paper details our analysis of the Windows Vista network stack. The following sections give an overview of our research and findings. The details of our testing scope, testing methodology, and results are in the appendices. The information is organized by network layer. In section II we discuss link layer protocols. Section III covers network layer protocols, and section IV covers transport layer protocols. Section V covers Windows Firewall, a component whose design encompasses many protocol layers. Section VI covers the servers and clients that operate at the application layer, and section VII covers unsolicited traffic. Sections VIII and IX present our conclusions and suggestions for future work.

II. Link Layer Protocols

Windows Vista supports protocols at the Link Layer for transporting IP packets, for performing address resolution and auto configuration tasks, and for providing topology information for network diagnostics. For transporting IPv4 and IPv6 packets, Windows Vista uses protocols such as Ethernet, PPP and PPPoE. In support of the IPv4 and IPv6 protocols, Vista includes ancillary protocols such as Address Resolution Protocol (ARP) and Neighbor Discovery Protocol (NDP), which are necessary to support the transmission of IPv4 and IPv6 packets. Windows Vista also introduces support for the new Link Layer Topology Discovery (LLTD) protocol, which is used to provide network maps to assist in diagnosing networking problems.

We analyzed the ancillary support protocols ARP and NDP to determine how they responded to redirection attacks and address conflict situations. We also performed an analysis of the LLTD protocol. As for all link layer protocols, attacks against these protocols are limited to the local network. We did not perform an analysis of Ethernet due to its simplicity, nor of PPP or PPPoE, since those protocols are typically used on private links to which an attacker is unlikely to have access. Analysis of PPP and PPPoE may be warranted in the future.

A. Link Layer Topology Discovery protocol

The Link Layer Topology Discovery (LLTD) protocol is a newly developed protocol, designed by Microsoft for discovering the topology of the local network, and to serve as a tool for diagnosing quality of service issues. LLTD is a core component of Microsoft's network diagnostic strategy. By providing high quality topology information to end users, Microsoft hopes to make it easier for users to manage their home networks. LLTD is also part of Windows Rally[44]. Subsequent to the first edition of this paper[49], Microsoft has provided complete documentation for LLTD[33] and a development kit[45].

LLTD on Vista logically consists of two components: a client program (mapper) that initiates and directs topology discovery, and a server (responder), implemented as a kernel driver, which responds to requests. The client program is invoked when a user requests that a network map, such as that in Figure 1, be generated from the networking control panel. The responder is a standard part of a Vista installation, and is always running unless explicitly disabled; upon request,

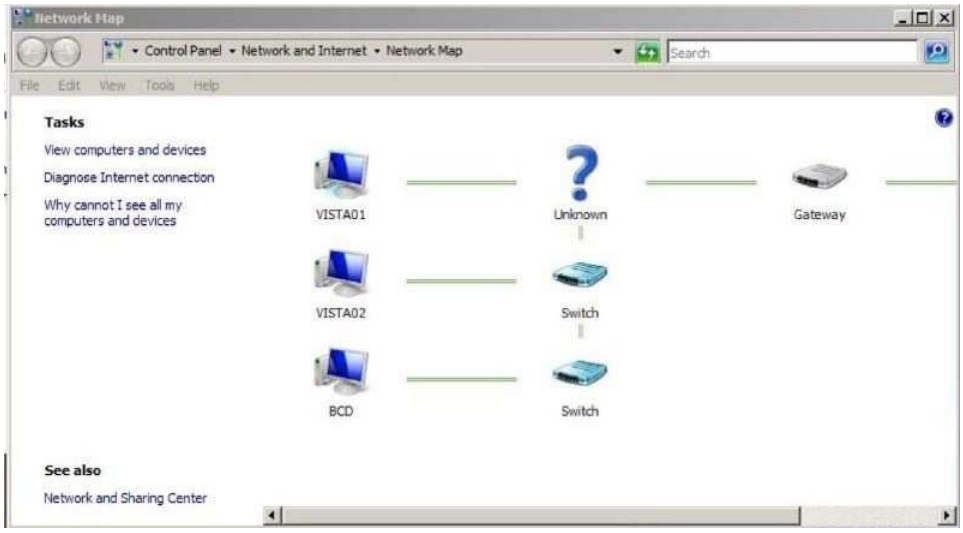

Fig. 1. Sample topology map within Vista

the responder provides information such as host name and Ethernet, IPv4, and IPv6 address.

The protocol sits directly on top of the Ethernet frame and has three layers of headers. As a non-routable protocol that elicits only basic information from hosts, minimal requirements exist for authentication, authorization and confidentiality. LLTD's security model is primarily designed to thwart denial-of-service attacks that are triggered by the usage of LLTD traffic against a LAN. We provide many more details on LLTD and its security model in Appendix II.

As a short research project, we investigated the security model, and the as-yet unknown vulnerabilities in the LLTD protocol and its Microsoft Windows Vista implementation. This research was performed on the latest Beta 2 build available at the time (5472). The development kit had not yet been released. We provide details on this analysis and our findings in Appendix III.

We found no major concerns, but some lesser ones. We found that we could spoof information and conduct a denial-of-service attack though simple packet injection. We were able to spoof another host, and even set up the user to go to an unintended external web page, instead of the local management interface they expected. We could also spoof being located on a bridge near the target, and it was possible to entirely impersonate another device. We found a number of ways to cause the mapping operation to fail. We also noted that devices (including an attacker) can provide an icon image that displays on the network map, which could support attacks. As a result of these findings, we recommend that users regard the information presented in the network map as unreliable.

In conclusion, LLTD is a simple non-routable protocol which — assuming the existence of some vulnerabilities — would require an attacker to have a presence on the local network, in order to exploit those vulnerabilities. From the research, it is clear that Microsoft utilized a secure development life-cycle to implement security at every stage, from design through implementation of the LLTD protocol. From a design perspective, this approach is evidenced in the restrictions around the use of the broadcast address, as well as the use of a credit model as guard against denial-of-service. Evidence at the implementation stage includes the use of compiler security features (/GS) combined with secure coding practices such as the use of the safe equivalent functions for string handling. Additionally, the diverse set of test cases applied resulted in findings of relatively small significance, which demonstrates Microsoft's success at preempting potential types of LLTD attacks. That said, many vendors may implement LLTD as part of the Windows Rally program, which may put responders on a network that — in terms of security development and adherence to the specification — are less well-implemented.

B. Address Resolution

The ARP[52] and NDP[48] protocols provide Ethernet address resolution for IPv4 and IPv6, respectively. ARP operates at the link layer and provides mechanisms for querying the link layer address of an IPv4 node and for propagating address changes to other hosts on the link. NDP is implemented using ICMPv6[8] packets above the IPv6 layer, but provides necessary services to transmit packets at the link layer: querying for the link layer address of an IPv6 node, propagating address changes, and address and route auto-configuration. NDP makes use of well-defined IPv6 multicast addresses[20] with fixed link layer addresses[10] to avoid bootstrapping problems. Both protocols are integral to the operation of the IP network stack and are enabled during installation.

ARP packets are sent in Ethernet frames, but NDP communicates using ICMPv6. However, NDP is presumably invulnerable to remote attack (assuming RFCs are followed) since link-local addresses are used, and the network stack verifies that an ND packet has a hop limit of 255 before processing its contents. The hop limit is decremented by one every time a packet is forwarded, and the packet discarded when its hop limit becomes zero. Since 255 is its maximum value, ND packets cannot be received from a remote network with a hop

limit of 255[2].

ARP is susceptible to a redirection attack when an attacker sends a "gratuitous" ARP packet to a target host. (Such packets are normally used to propagate address changes.) After receiving such a packet, the Vista stack forwards any packets that are waiting for the MAC address for the IPv4 host to the attacker's node, rather than to the intended target. The association between the MAC address and the attacker's IPv4 address is stored and used for future packets if there is either an existing entry (which is overwritten) or if the the ARP is sent directly to the host, rather than to the Ethernet broadcast address. When an existing ARP table entry is overwritten, no warnings are displayed to the user.

In the case that a Vista machine receives a conflicting directed or broadcast message for the statically configured IP address, that address becomes unusable and a pop-up message announces the conflict (similarly to Windows XP). Attempts to use the network result in an error until the network interface is reset.

One function of NDP is its namesake, Neighbor Discovery (ND). This function provides a similar link-layer address-to-IP address mapping as ARP. We found ND to be more resistant to attacks than the ARP implementation. We observed that Windows Vista hosts will not process unsolicited Neighbor Advertisements (NAs) unless they update an existing neighbor cache entry. However, it is still possible to perform a redirect attack by sending spoofed NAs in response to actual queries, or by blindly sending out NAs periodically[3]. We observed that Vista automatically configures a replacement RFC 3041 address[47] in the event of an apparent conflict, such as an attacker could simulate.

We provide more details on our ARP and ND spoofing tests in Appendix V and VI, respectively.

III. Network Layer

Microsoft chose to rewrite the Windows Vista IP stack rather than derive it from the previous Windows XP stack. This new stack integrates support for IPv4 and IPv6 into a single network stack and, according to Microsoft, is easier to maintain, gives increased performance, and is more stable than their previous network stack[55].

Windows Vista is the first Windows operating system to enable both IPv4[53] and IPv6[10] during installation. The Vista stack integrates IPv4 and IPv6 into a single network stack whereas previous implementations offered a separate IPv6 stack as an optional component. Many implementation characteristics are shared between both stacks as a result of this tight integration.

The inclusion of IPv6 support in Windows Vista is a major departure for Microsoft. IPv6 provides significant functionality, backed by code that is not tried by extensive use in a hostile environment. To make matters worse, many of the defenses that are relied on to protect today's IPv4 networks either do not yet support IPv6 or are similarly immature. As IPv6 is more widely deployed, we expect IPv6 to be heavily scrutinized by those with malicious intent.

In the following subsections, we discuss Vista's IP protocol behavior, supported upper level protocols, and tunneling protocols (especially Teredo). We also report our observations on ICMP, IGMP/MLD, and attacks.

A. IP Behavior

We measured implementation characteristics of the IPv4 and IPv6 protocol layer and, where relevant, compared them to previous implementations. The characteristics we measured were IP ID generation, IP fragment reassembly behavior, and source routing.

1) IPv4 ID Generation: Based on more than 600,000 data points, we observed that the Windows Vista stack generates IPv4 packet identifiers (used in IPv4 fragment reassembly) sequentially. Windows XP's stack also generates these sequentially. However, on Windows Vista, these wrap around starting at 0x7FFF, but on XP we have observed IPv4 IDs above this maximum. This can potentially be used for differentiation; a packet from a host with an IP ID that is 0x8000 or above cannot be Vista. Full details on this testing are available in Appendix VII.

Sequential IDs can be used to measure the network activity of a host. When two packets are received from a host, the amount of traffic that was sent in the intervening time is the difference between the IDs in each packet. Sequential IDs are also useful in counting hosts behind a NAT firewall [5].

There are a few differences in the use of the IPv6 ID field compared to this field in IPv4: (1) the field grows from 16 to 32 bits, (2) the field is moved to the Fragmentation extension header, (3) the field is normally only present when there is actual fragmentation, and (4) fragmentation is less common, since on-route IPv6 fragmentation is prohibited. Since it is seen less often, we did not study how IDs are generated under IPv6.

2) IP Fragmentation Reassembly: The Windows Vista networking stack behaved differently to the previous XP stack and other popular networking stacks when reassembling IPv4 and IPv6 fragments. Based on 64 test cases, we observe that, in most cases, Vista appears to discard an entire fragmented packet (the set of fragments with the same IP ID) if it contains partial overlaps. However, if the partial overlap fits within the leading part of the packet that could be reassembled based on fragments already sent, the overlapping fragment is ignored. In cases of exact overlap, newer fragments are discarded in favor of previously received fragments. This behavior was observed both for IPv4 and IPv6. (We describe this more fully, and give a primer on fragmentation, in Appendix VIII.) Given this behavior, this reassembly policy seems more likely to be the result of the data structures used for reassembly, than the result of explicit design. Hence, this behavior is more likely to change, possibly as a result of even small code changes.

The Windows XP stack allows for partial overlaps for IPv4. As a result of these differences, identical traffic sent to XP and Vista targets may be interpreted differently. Ambiguities in the interpretation of traffic provide opportunities for

[2]Tunneling protocols may provide a way around this restriction. We have not investigated any ND attacks used in conjunction with tunneling.

[3]The IETF has defined SEND[2] as a way to provide secured Neighbor Discovery, though Windows does not currently support it.

confusing network intrusion detection devices, unless handled appropriately[50].

Our IPv4 testing included sending fragmented UDP packets and observing the result on the recipient system. As described in Appendix VIII-B, for IPv6 we needed to take a different approach. We observed the reassembly via an ICMPv6 error message, which we induce by sending an unknown destination option. In both cases, we observed that an ICMP fragmentation timeout message would usually be produced after a one minute delay, but in some cases this error was not produced; the conditions required to produce the error message are not clear to us. We also observed ICMPv4 parameter problem messages sent in response to certain IPv4 overlap cases (specifically, when two fragments overlap at the end of the fragment, but the second one has More Fragments (MF) set).

3) Source Routing: Source routing describes a process where the packet originator predefines a series of "hops" to take on the way to a destination. Source routing is available in both IPv4 and IPv6. This function can be used by an attacker in a number of types of attacks, including bypassing access control. A best practice is to block it unless it is absolutely necessary. Windows XP never supported serving as an intermediate hop for IPv4 source routing; however, versions prior to SP2 would accept packets that were source-routed. By examining settings and observing actual behavior (Appendix IX), we confirmed that the Vista release continues this behavior for IPv4. In IPv6, Vista accepts packets that had been source-routed with type 0 for source routing[4].

B. IPv4 and IPv6 Supported Protocols

We explored what protocols Vista supports on top of IPv4 and IPv6. If a network stack responds to a probe with an ICMP message indicating that it does not support a protocol, then it becomes possible map out supported protocols.

As for the Windows XP stack, by default, the Windows Vista stack (in which the firewall is enabled by default) does not respond to received IPv4 packets that have an unsupported protocol number. However, it does respond to unsupported IPv6 Next Header values with an ICMPv6 Parameter Problem message about the Next Header value. (The IPv6 Next Header field is the same as the IPv4 protocol field except that it also encodes extension headers; the same number-space is used for both.) To get a more complete picture, we decided to test both IP protocol versions, with and without the Windows Firewall turned on (see Appendix X and XI). We summarize the results in Figure 2.

There are a number of noteworthy results here. It is surprising that protocols 43 and 44 appear to be supported under IPv4; in IPv6, these numbers denote IPv6 extension headers, but they have no meaning in IPv4[5]. We had seen 43 and 44 with IPv4 in beta builds (where sending packets to those would cause problems)[49], but had expected them to be removed by the release build. These protocol numbers no longer appear to cause problems in Vista, but due to limited time we did not explore if these are actually usable in some manner.

Vista seems to support a large number of tunneling options. We have not yet tested which, if any, could be used by default, and have not explored their security implications. Over IPv4, it appears that the Vista stack can support IPv4, IPv6, and GRE[6]. Here, IPv4 and IPv6 are directly encapsulated on IPv4; IPv6 over IPv4 is used for ISATAP and 6to4. GRE[15] has been historically used to tunnel IPv4 and non-IP protocols over IPv4.

Over IPv6, it appears that direct IPv4 and IPv6 encapsulation is supported. IPv6 over IPv6 is used in Mobile IPv6 (RFC 3775[26]), and IPv4 over IPv6 is a transition mechanism required in the future for IPv4 traffic, when a network only supports IPv6. Noteworthy here is that the apparent support for IPv4 over IPv6 is only present when the firewall is enabled; this may be an intentional policy decision similar to requiring an IPv6 firewall for Teredo to be available (see Appendix XIII-F).

C. Teredo and Other Tunneling Protocols

Teredo supports many tunneling protocols. We mentioned several in Section III-B that we detected that are apparently supported by Vista. Though it may be fruitful, we have not had the opportunity to look into most of these.

Windows Vista employs IPv6 transition technologies, which allow IPv6 to be used in an IPv4 environment that has limited or no IPv6 infrastructure[11]. Microsoft documentation ([32]) describes Teredo, ISATAP, 6to4, IPv6 automatic tunneling, and 6over4, as available in Windows 2003, so these may be presumed to be available on Vista as well. Among these, we have done a thorough investigation of Teredo (Appendix XII and XIII), and we have seen evidence of ISATAP support (Appendix XXVII and section III-B).

Teredo, defined in RFC 4380[23], is an IPv4–IPv6 transition technology, which Windows Vista uses if there are no neighboring IPv6 routers or ISATAP servers. We expect this to be the most common environment among Windows Vista users, until IPv6 sees wider network support.

Teredo works by carrying IPv6 packets inside of UDP packets sent over IPv4 networks. What makes Teredo unique among IPv6 transition mechanisms is its NAT traversal features. Teredo hosts establish and maintain a connection to one of a set of public Teredo servers. The IPv6 address assigned to a Teredo host encodes the public Teredo server that assigned it, as well as the public address and port assigned to the host (its address as seen outside of the NAT). A NAT-protected host can establish a direct connection to another such host with the assistance of the peer's Teredo server. The host can notify its peer that it wants to establish a connection by sending the packet to the peer's Teredo server, which is forwarded on to the peer host. The two peers may then send packets to each other,

[4]This could be related to future mobile IPv6 support[38], although that does not use type 0 source routing, and there are no direct signs that mobile IPv6 is supported by the release build of Vista.

[5]The protocols 249 and 251, which we noted in earlier builds[49], have been explained (and hidden by the stack). Reportedly, these protocol codes are used internally by the stack for communication with the IPsec offload module; these were not intended to be visible externally, but Windows Firewall failed to treat these as externally unsupported.

[6]This tentative conclusion is based solely on the lack of an ICMP protocol unreachable when probing those.

Protocol/EH code	Description	IPv4 + firewall	IPv4 - firewall	IPv6 + firewall	IPv6 - firewall
0	Hop-by-Hop Options EH	filtered	not supported	supported	supported
1	ICMPv4	filtered	supported	not supported	not supported
2	IGMP	filtered	supported	not supported	not supported
4	IPv4 over IPv4/IPv6	filtered	supported	supported	not supported
6	TCP	filtered	supported	supported	supported
17	UDP	filtered	supported	supported	supported
41	IPv6 over IPv4/IPv6	filtered	supported	supported	supported
43	Routing EH	filtered	supported	supported	supported
44	Fragment EH	filtered	supported	supported	supported
47	GRE	filtered	supported	not supported	not supported
50	IPsec ESP	filtered	supported	supported	supported
51	IPsec AH	filtered	supported	supported	supported
58	ICMPv6	filtered	not supported	supported	supported
59	IPv6 No Next Header	filtered	not supported	supported	supported
60	Destination Options EH	filtered	not supported	supported	supported
(unsupported protocols)		filtered	proto unreachable	param problem	param problem

Fig. 2. Vista's supported IPv4 and IPv6 protocols and extension headers. Results are shown for both with the firewall on (the default) and with the firewall off. EH stands for IPv6 extension header.

opening up mutual holes in their NAT gateways for return traffic to flow through. The two peers can maintain these NAT mappings indefinitely by periodically exchanging traffic.

Teredo restores global addressability and routing to hosts using private IPv4 addresses. This is a huge benefit in terms of functionality, but also has serious security implications. Many individuals and companies use private addresses as a key part of their defense strategy, and they will be left unexpectedly exposed to the Internet when Vista is installed, unless strict egress filtering is in place.

Furthermore, network based-security controls, such as NIPS and firewalls, are bypassed by the Teredo tunnels unless they are specifically Teredo-aware (and examine all Teredo traffic: clients and relays do not use fixed ports). This means that not all the intended security controls are applied as expected to Teredo tunneled IPv6 traffic. At a minimum, defense-in-depth is lost and, at worst, an important security mechanism is not applied[7]. We explore this and other Teredo security implications in more detail and describe how Teredo works, in a spin-off paper titled, *The Teredo Protocol: Tunneling Past Network Security and Other Security Implications*[22].

In investigating Vista's Teredo implementation on the RC2 Vista build (full details are in Appendix XIII), we found that, in order for Vista to use Teredo, an IPv6-capable firewall must be registered, and that Windows Firewall and other firewalls that use the Windows Filtering Platform[43] (such as Symantec's Vista-enabled NIS/NAV 2007) should apply protection equally to native IPv6 traffic and to Teredo tunneled IPv6 traffic.

We found no faults in the protocol parsing in Microsoft's Teredo stack implementation. However, relative to what is provided for in the Microsoft-sponsored, standard document[23], we found that some of the security features in the Windows Vista Teredo implementation are implemented minimally. In at least one situation (length of the ping test nonce), the implementation of these security features is sub-par to that which is recommended (32 bits instead of at least 64 bits), and the value "0" was used repeatedly.

As we describe in Appendix XIII-A, it is not entirely clear or easy to describe the circumstances under which Teredo will be used under Vista. However, when on two occasions we briefly connected our typically isolated network hosts (the network in Appendix I-A, which was not intended for use in Teredo testing) to an Internet-connected network, we found that they had configured a Teredo address (Appendix XXVIII). This occurred during Windows Activation and during Vista installation, when a host was accidentally connected to the wrong network. Thus we expect Teredo will be frequently used under Vista.

Organizations should pay attention to Teredo. We recommend that organizations and individuals wanting to use IPv6 do so properly by upgrading their security mechanisms to support native IPv6, and then acquire a native IPv6 Internet connection. Teredo and other IPv6 transition mechanisms should be disabled on the client (see Appendix XIII-E) and blocked on the network unless the security implications have been carefully considered and found acceptable.

D. ICMP

Vista supports ICMPv4 in association with IPv4 and ICMPv6 in association with IPv6. The basic ICMPv4 and ICMPv6 headers are syntacticly identical, though there are differences in the meaning of ICMP types and codes and in how the two versions of ICMP are used.

The testing described in Appendix XVI-A gave an opportunity to assess how much of the original packet the Vista network stack includes in ICMPv6 error messages. We found that it follows RFC 2463[8] and includes as much of the original packet as can fit in a 1280-octet return packet. Assuming no extension headers, this corresponds to 1232 octets of the original IPv6 packet.

[7]However, if Teredo proves to be more manageable and becomes used for a situation instead of a less manageable form of tunneling, there has been some benefit. The ideal, however, would be have no tunneling.

We observed that Vista rate limits ICMPv4 and ICMPv6 error messages, as required for ICMPv6 by RFC 2463[8]. Vista appears to suppress error messages when another was sent within the previous second. This caused a degree of difficulty in our testing, as we could not impute any significance to the absence of an error message, when one had been seen in the previous second. This required our UDP, IP, and other scanning to be run more slowly.

ICMPv4 and ICMPv6 echo services are available on Vista, but must have a firewall exception configured in order for them to be usable remotely.

E. IGMP and MLD

MLD is the Multicast Listener Discovery Protocol. Its job for IPv6 is equivalent to the job of IGMP for IPv4; that is, — to keep the network informed of the types of multicast traffic hosts on the network require. IGMPv3[7] and MLDv2[59] bear a strong resemblance to each other; the main difference is that IGMP sits directly on top of IPv4, whereas MLD (like NDP) sits on top of ICMPv6.

Vista uses IGMPv3 and MLDv2, often in tandem, to provide corresponding services for IPv4 and IPv6, respectively. On a clean install of Vista, we saw them used together to subscribe or unsubscribe from multicast addresses for IPv4 and IPv6 LLMNR (Link Local Multicast Name Resolution [1]) addresses and for SSDP (Simple Service Discovery Protocol) and UPnP addresses.

F. Defect Testing

We conducted tests to assess the stability of the Windows Vista TCP/IP stack:

1) We ran several historic attack scripts that have affected network stacks in the past. We found (Appendix XV) that the only attacks with a noticeable impact are those that produced a large number of packets per second. In the case of opentear, the host GUI became unresponsive until the attack was stopped, but we conclude that this was due to the overwhelming packet volume sent. The udp and udp2 attacks caused network congestion and interfered with our reverse ping.
2) We retested the three defects that we reported in the previous edition of this report[49], and which we found through the use of ISIC[16] (the crash1.py, crash2.py and crash3.py scripts); we observed no resultant impact.
3) Appendix XVI details tests that we conducted by generating IPv6 destination options with a malformed payload. These varied from totally random to precisely crafted tests. We have observed no persistent effects from any of these[8].

IV. Transport Layer

Windows Vista supports the TCP and UDP transport protocols over IPv4 and IPv6. We investigated the implementation characteristics of these protocols.

[8]As these tests were run mostly unsupervised in the background, we would be unlikely to notice any resultant short term or transient problems.

A. Ephemeral Ports

We found that the default ephemeral port range for both UDP and TCP has changed with Vista. It is now 49152–65535, though it can be adjusted using netsh. TCP and UDP are controlled separately, although the ephemeral port settings for each protocol under IPv6 and IPv4 are shared. In fact, we infer that the "next port to use" state for each of TCP and UDP applies to both IPv4 and IPv6. Thus, unless a socket creation call is made that applies to both IPv4 and IPv6, the same port number will not be used for both IPv4 and IPv6. We have observed the former case with TCP and the latter for UDP. We describe this in more detail in Appendix XVII.

B. TCP

We measured Windows Vista's TCP ISN generation, segment reassembly, and "fingerprint" behaviors and observed several behavioral differences from the earlier Windows XP stack.

The choice of the Initial Sequence Number to be used when establishing a TCP connection has a profound impact on the security of a TCP connection[4], [46], [51]. We measured Windows Vista's ISN generation and found that it appears to follow the generation algorithm recommended by RFC 1948[3], which generates ISN values by adding a system-wide counter to a secret hash of the connection identifier. This generation scheme offers strong protection against TCP attacks relying on poor ISN generation. We used two techniques to plot differences in Vista's sequential ISNs for both IPv4 and IPv6 and found what appears to be a uniform distribution (see Appendix XVIII).

We measured the TCP reassembly behavior of Windows Vista (Appendix XIX). When reassembling a TCP stream, Windows Vista resolved any conflicts in overlapping TCP segments by preferring data received in earlier segments over data received in later segments. This behavior differs from the behavior observed in the earlier Windows XP networking stack and in other popular stacks. Due to these differences, identical traffic sent to XP and Vista targets may be interpreted differently. Ambiguities in the interpretation of traffic provide opportunities for confusing network intrusion detection devices, unless handled appropriately[50].

There are many network stack fingerprinting methods which identify an operating system through its network stack implementation details[19]. We looked at the TCP behavior measured by the Nmap[18] utility. We observed that the Windows Vista networking stack behaves distinctly differently to the previous Windows XP version and other popular network stacks. The details of these differences are noted in Appendix XX. Because most incoming TCP traffic is filtered by Windows Firewall, this testing was done with the firewall turned off. However, there are likely other techniques that could be used to fingerprint Vista, even with its firewall turned on.

For both IPv4 and IPv6, we found that Windows Firewall filtered RST messages that would normally result from closed TCP ports.

C. UDP

For both IPv4 and IPv6, we found that Windows Firewall filtered ICMP port unreachable messages that would normally result from probes to unused UDP port numbers. Thus, in the default configuration, one cannot remotely map out currently used UDP ports, at least using the most obvious technique.

V. Firewall

A new and extended version of Windows Firewall comes with Vista. This provides protection against malicious attack by filtering out incoming packets before they are processed. Windows Vista configures Windows Firewall during installation, and Windows Firewall is running on all Windows Vista machines unless explicitly disabled. We measured the firewall configuration of a Windows Vista machine after installation, and after several common configuration changes. We also noted methods that could be used to detect the presence of a Windows Vista host, even when protected by Windows Firewall.

We provide full details in Appendix XXI and present our main findings here.

A. Firewall Rules

In Vista, all network interfaces are—at any given time—a part of one of three pre-defined profiles[12]: public, private, or domain. The public profile (the default) is for when the interface is connected to an untrusted network, for example, at a coffee shop. The private profile is for when the user is at home or work, and the domain profile is for cases where the host is part of a Windows domain. Windows Firewall makes use of these profiles to essentially maintain three different sets of firewall exceptions; this makes sense, as some networks are more trusted than others. This reflects Microsoft adapting to an increasingly mobile user base.

We found that each entry in the exceptions list in the Windows Firewall control panel corresponds to a similarly named group of firewall rules. These individual firewall rules—which can be observed using the application Windows Firewall with Advanced Security, in the inbound table—can apply to one or more profiles. Although these can be controlled individually and manually, they are normally enabled or disabled as a group, either for all profiles or for specific profiles. There are more general capabilities, but in our testing, enabling a rule means creating an exception for the circumstances that are governed by the rule. These rule parameters affect the circumstances in which an exception applies: protocol, local port, remote port, local network, remote network, and (local) program; these all must match for an exception to apply.

The application, Windows Firewall with Advanced Security, also contained outbound firewall settings and IPsec configuration in separate sections, but these were not within the scope of the research.

B. Initial State

We list Windows Firewall's inbound initial state in Table I (page 83). This consists of 166 rules.

In its default configuration, Windows Vista has three firewall rule groups that are enabled for at least one profile. The "Core Networking" group is active for all profiles, and the "Network Discovery" and "Remote Assistance" are active for the private profile only; thus by default there is considerably less exposure than if the network interface is set to private or domain. Core Networking covers ICMP errors, DHCP, IGMP, MLD, NDP, and Teredo. Network Discovery covers LLMNR, Netbios, web services discovery, SSDP and UPnP. Remote Assistance covers SSDP, UPnP, RPC Endpoint mapper, raserver.exe, and msra.exe. The firewall exceptions only matter if there is a program listening on a socket behind it, and that is not always the default case for these groups. For example, we notice that raserver.exe and msra.exe are not initially running (see Figure 48).

All of the TCP and UDP firewall rules present in the initial state that have "Any" as a remote port have a specific program that they are bound to, apparently limiting the breadth of exposure.

C. Configuration

Several common Windows configuration changes introduce filtering exceptions into Windows Firewall configuration; turning on File and Print sharing (CIFS), opting into People Near Me, using Windows Meeting Space, and enabling Windows Media Sharing are all examples of such changes. These changes must be authorized by the user using the Windows Vista consent mechanism. In Figure 3 we show what groups and profiles are enabled or disabled in some tests we conducted. The details of these firewall configuration changes can be found in Appendix XXI-C.

One surprising result, noted in many cases (as can be seen in the table), is that firewall rules are not disabled upon turning off the Vista function that causes the rules to be enabled (i.e., the rules were "sticky"). The exceptions persist even across a system restart; thus, until they are manually disabled, a legacy of firewall exceptions accumulates on a system. One of the conceivable, negative effects of this is that a malicious application could communicate through the exception without a consent prompt, and that a listener or service could remain exposed. The sticky situations we observed are the following: People Near Me, Windows Meeting Space, and partly for media sharing.

We examined the socket listener changes (as seen through netstat) that correspond to the configuration changes. We noticed that for a duration after Windows Meeting Space was enabled, a meeting was created, a meeting was closed, and the Windows Meeting Space was closed, leftover processes continued. One of these, DFSR.exe (TCP port 5722 on IPv4 and IPv6), could be reached remotely even after its reason for existence discontinued; however, this only lasted for a few minutes, so the extra exposure is limited.

In previous builds of Vista[49], Teredo quietly created firewall exceptions; retesting this was out of the scope of this project.

Firewall group	Profile	In initial state	Enable File Sharing or Public Folder Sharing	Disable File Sharing or Public Folder Sharing	Enable Media Sharing	Disable Media Sharing	Disable Network Discovery	Set up People Near Me	Sign into People Near Me	Sign out of People Near Me and reboot	Sign into Windows Meeting Space	Create Windows Meeting Space meeting	End Windows Meeting Space meeting, sign out, and reboot
Core Networking	private	✓											
	domain	✓											
	public	✓											
Remote Assistance	private	✓											
	domain												
	public												
Network Discovery	private	✓					−						
	domain												
	public												
File and Printer Sharing	private		+	−	+	−							
	domain												
	public												
Windows Media Player Network Sharing Service	private				+	−							
	domain				+	−							
	public												
Windows Media Player	private				+								
	domain				+								
	public												
Windows Peer to Peer Collaboration Foundation	private								+		+		
	domain								+		+		
	public								+		+		
Windows Meeting Space	private										+		
	domain										+		
	public										+		
Network Projector	private										+		
	domain										+		
	public										+		

Fig. 3. The enabling or disabling of firewall groups for profiles as series of actions were taken. Note that some never became disabled. The initially enabled groups are also listed. Firewall groups not involved are omitted.

D. Discovery

Windows Vista hosts that are protected by Windows Firewall can be discovered in several ways, even though ICMP echoes (pings) are filtered over both IPv4 and IPv6. Hosts on the same network can effectively "ping" a host by querying for the host's hardware address using the ARP or ND protocols, by requesting all neighbors to respond to a LLTD request, or by simply listening on the network. Detection using LLTD is particularly attractive because it returns the host's Ethernet, IPv4 and IPv6 addresses and host name. Hosts that are not on the same local network can elicit responses from a Windows Vista host remotely by using routable IPv4 and IPv6 packets. As previously mentioned, Windows Vista responds to IPv6 packets with unknown Next Header values. TCP port 5357 can also be attempted. Vista responds to packets received using an unhandled protocol or with certain malformed fields[20] with ICMP errors.

E. Tunneling

The tunneling protocols supported by Windows Vista have implications for firewalls protecting Vista hosts. If not blocked, tunnels may provide an attacker with an avenue to bypass all firewall restrictions. The tunneling protocols may also provide avenues for bypassing Windows Firewall. Exploring tunnel-based attacks was outside the scope of this project.

VI. Network Services

A. Active TCP Ports

We applied standard techniques to remotely enumerate the network services using the TCP transport over IPv4 and IPv6 in Windows Vista (Appendix XXII). We observed that in Vista's default configuration for the private profile, Windows Firewall allowed access to TCP port 5357 (Web Services on Devices, WSD) and there was a live service running on that port; therefore this provides a point of exposure to Vista hosts.

TCP port	description	IPv4 + firewall	IPv4 - firewall	IPv6 + firewall	IPv6 - firewall
135	RPC endpoint mapper	filtered	open	filtered	open
139	NBT	filtered	open	filtered	closed
445	SMB	filtered	open	filtered	open
5357	WSD	open	open	open	open
49152	RPC ephemeral	filtered	open	filtered	open
49153	RPC ephemeral	filtered	open	filtered	open
49154	RPC ephemeral	filtered	open	filtered	open
49155	RPC ephemeral	filtered	open	filtered	open
49156	RPC ephemeral	filtered	open	filtered	open
49157	RPC ephemeral	filtered	open	filtered	open
(closed ports)		filtered	RST	filtered	RST

Fig. 4. Vista's open TCP ports over IPv4 and IPv4, with and without the firewall on.

UDP port	description	IPv4 + firewall	IPv4 - firewall	IPv6 + firewall	IPv6 - firewall
123	NTP	open or filtered	open	open or filtered	open
137	NetBIOS name service	open or filtered	open	open or filtered	closed
138	NetBIOS datagram	open or filtered	open	open or filtered	closed
500	ISAKMP	open or filtered	open	open or filtered	open
1900	UPnP/SSDP	open or filtered	open	open or filtered	open
3702	Web Services Discovery	open or filtered	open	open or filtered	open
4500	IPsec	open or filtered	open	open or filtered	closed
5355	LLMNR	open or filtered	open	open or filtered	open
3–4 ephemeral ports	clients	filtered	open	open or filtered	open
(unused ports)		filtered	port unreachable	filtered	port unreachable

Fig. 5. Vista's used UDP ports over IPv4 and IPv4, with and without the firewall on.

By scanning with the firewall turned off and by using netstat, we found other active ports that the firewall filtered, in the default configuration (see Figure 4). All the ports that have services on IPv4, also have services on IPv6, except for port 139 (NBT). These filtered ports would need to be exposed (for example, by enabling corresponding Windows functionality) to be used by an attacker.

B. Active UDP Ports

Through remote enumeration, we found that Windows Firewall filtered access to all closed UDP ports over both IPv4 and IPv6. With the firewall turned off and with netstat, we found eight well-known ports in use (five through IPv6) and three or four varying ephemeral ports in use. We shows these in figure 5. (Due to the nature of UDP, no protocol-independent way exists to determine which of these ports offer services and which are purely clients.) The ephemeral ports and NTP are likely clients. In either case, for packets to reach the port in order to potentially attack it, the firewall must allow access.

We provide more details on this testing in Appendix XXIII.

C. File Sharing

It is common for computer users with several machines to turn on File and Printer Sharing, and we expect many Windows Vista users will do this. Windows Vista supports the SMB protocol and introduces the new SMB2[60] variant of the protocol.

SMB2 is a new implementation of the SMB protocol that provides a clean slate for Microsoft. It eliminates many of the legacy SMB calls that are no longer used. It supports high performance marshaling with fixed header sizes and better alignment rules, and it provides larger field widths for many of the protocol fields to ensure support for larger disks and faster computers in the future. SMB2 is the preferred protocol when supported by both client and server (two Windows Vista hosts, for example), but support is included for legacy interoperability.

File sharing allows remote access to named pipes. These pipes are often used as a transport mechanism for application protocols. We enumerated the named pipes (Appendix XXV) that could be accessed over both null (anonymous) and authenticated sessions, and summarized the results at the start of Figure 6. We could successfully access the netlogon, lsarpc and samr pipes without any authentication. All of these pipes are aliases and refer to the pipe named "lsass." This pipe is used as a transport for several RPC based interfaces. With the authenticated access, we found that success sometimes depended on whether the connection was coming from XP (which uses SMB) or Vista (SMB2), which may mean that the two protocols are handled by different implementations.

We enumerated the accessible RPC interfaces available over the named pipes. The interfaces that we could successfully use via null or authenticated sessions are listed in Figure 6. We identified which procedures could be called, on each of these interfaces, under the different access circumstances. Our calls were made without knowledge of the proper parameters, so we regarded a BAD_STUB_DATA as successful. In Figure 7 we present the 102 procedure calls that successfully completed over a null session. The names on the list are based on

Interface	lsarpc,samr,netlogon	lsass,protected_storage	W32TIME_ALT	wkssvc	atsvc,ROUTER,browser	srvsvc	trkwks	keysvc	InitShutdown	eventlog	ntsvcs	scerpc	LSM_API_service, tapsrv,plugplay	epmapper	MsFteWds
can be opened	any	A	A	A	A	A	A	A	A	A	A	A	A	A	A
achieved success or BAD_STUB_DATA	any	A	V-A	A	A	X-A	A	A	A	A	A	X-A	V-A	A	
afa8bd80-7d8a-11c9-bef4-08002b102989[v1.0] (rpcmgmt (ifids))	any	A	V-A	A	A	X-A	A	A	A	A	A	X-A	V-A	A	
12345778-1234-abcd-ef00-0123456789ab[v0.0] (LSA access (lsarpc))	any	A													
12345778-1234-abcd-ef00-0123456789ac[v1.0] (samsrv)	any	A													
3919286a-b10c-11d0-9ba8-00c04fd92ef5[v0.0] (LSA DS access (lsarpc))	any	A													
c681d488-d850-11d0-8c52-00c04fd90f7e[v1.0] (efsrpc)	any	A													
300f3532-38cc-11d0-a3f0-0020af6b0add[v1.2] (trkwks)				X-A	X-A	X-A	A	X-A		X-A	X-A	X-A			
6bffd098-a112-3610-9833-46c3f87e345a[v1.0] (wkssvc)			V-A	A	X-A	X-A	X-A	X-A							
8fb6d884-2388-11d0-8c35-00c04fda2795[v4.1] (w32time)				X-A	X-A	X-A	X-A	X-A							
1ff70682-0a51-30e8-076d-740be8cee98b[v1.0] (atsvc)				X-A	A	X-A	X-A	X-A							
4b324fc8-1670-01d3-1278-5a47bf6ee188[v3.0] (from srvsvc.dll, Netr*)				X-A	A	X-A	X-A	X-A							
6bffd098-a112-3610-9833-012892020162[v0.0] (from browser.dll, I_Browserr*, NetrBrowser*)				X-A	A	X-A	X-A	X-A							
82273fdc-e32a-18c3-3f78-827929dc23ea[v0.0] (eventlog, from wevtsvc.dll)										A	X-A	X-A			
367abb81-9844-35f1-ad32-98f038001003[v2.0] (Services Control Manager (SCM))										X-A	A	X-A			
93149ca2-973b-11d1-8c39-00c04fb984f9[v0.0] (scesrv)										X-A	X-A	X-A			
3dde7c30-165d-11d1-ab8f-00805f14db40[v1.0] (BackupKey)										X-A	X-A	X-A			
8d9f4e40-a03d-11ce-8f69-08003e30051b[v1.0] (umpnpmgr)										X-A	X-A	X-A			
894de0c0-0d55-11d3-a322-00c04fa321a1[v1.0] (InitShutdown)									X-A						
0b0a6584-9e0f-11cf-a3cf-00805f68cb1b[v1.1] (localpmp)														X-A	
e1af8308-5d1f-11c9-91a4-08002b14a0fa[v3.0] (epmapper)														A	

Fig. 6. Successful calls to UUIDs from pipes under different circumstances. Values are listed for a UUID and named pipe when we observed a successful call (resulting in "success" or "BAD_STUB_DATA") to a procedure in the UUID using the named pipe. "X-A" means the interface only succeeded from Windows XP (SMB) with an authenticated session, "V-A" means the interface only succeeded from Windows Vista (SMB2) with an authenticated session, "A" means it succeeded from both sources of authenticated session, and "any" means that it succeeded from both XP and Vista and both null and authenticated sessions. Note that some columns represent multiple pipes that behaved identically for UUID access. Also listed are the circumstances under which pipes could be opened and successfully used. The faint dotted grid lines on every fifth row and column are depicted only to facilitate reading the rows and columns; no grouping is implied.

available symbols; static analysis of system executables also helped inform the list of UUIDs to look for. More details on this and on the procedures that could be reached in other configurations are available in Appendix XXVI.

We saw ACCESS_DENIED appearing at a per-procedure level, suggesting that access control is being employed.

D. RPC Services Over TCP

In the release build of Vista there are no RPC TCP ports available in the initial configuration. In fact, we know of no remote RPC access available initially. However, with file sharing enabled, the endpoint mapper could be reached. We document the results of an enumeration of this service in Appendix XXIV.

Among other results, the endpoint mapper tells us that the six ephemeral TCP ports we saw active earlier (49152–49157) are associated with RPC. These are not accessible with just file sharing enabled. However, the endpoint mapper port (TCP 135) is and, as we did across named pipes, we enumerated the RPC interfaces supported on this port using a brute force enumeration technique. We provide more complete results in Appendix XXVI-B, but there were seven procedures between two interfaces that we could successfully call. Within the IOXIDResolver interface, the ResolveOxid, SimplePing, ComplexPing, ServerAlive, ResolveOxid2, and ServerAlive2 procedures were accessible. On the RPC remote management interface, only the fourth procedure (rpc_mgmt_inq_princ_name) was callable. We observed that the TCP port 135 had fewer interfaces that could be successfully accessed than using anonymous named pipes. Similarly, the interfaces in common had fewer procedures that could be successfully called using port 135 than anonymous named pipes.

Not all interfaces available on a network port are actually usable; there are RPC mechanisms for blocking requests arriving over the network ([30], [28]). This is useful for services that do not wish to be available over the network but share a process with another service that uses a network transport. This fact could explain some of the ACCESS_DENIED errors received when calling certain procedures.

VII. Unsolicited Traffic

To get a feel for the active services and protocols on a default Windows Vista installation, we observed the packets that Vista sends out during certain transitions and looked for unsolicited traffic. Traffic captures from Vista starting up,

`afa8bd80-7d8a-11c9-bef4-08002b102989[v1.0]` (rpcmgmt (ifids)):

- rpc_mgmt_inq_if_ids
- rpc_mgmt_inq_stats
- rpc_mgmt_is_server_listening
- rpc_mgmt_stop_server_listening
- rpc_mgmt_inq_princ_name

`12345778-1234-abcd-ef00-0123456789ab[v0.0]` (LSA access (lsarpc)):

- LsarClose
- LsarEnumeratePrivileges
- LsarQuerySecurityObject
- LsarChangePassword
- LsarOpenPolicyRPC
- LsarQueryInformationPolicy
- LsarSetPolicyReplicationHandle
- LsarEnumerateAccounts
- LsarEnumerateTrustedDomains
- LsarLookupNames
- LsarLookupSids
- LsarOpenAccount
- LsarEnumeratePrivilegesAccount
- LsarGetQuotasForAccount
- LsarGetSystemAccessAccount
- LsarOpenTrustedDomain
- LsarQueryInfoTrustedDomain
- LsarLookupPrivilegeValue
- LsarLookupPrivilegeName
- LsarLookupPrivilegeDisplayName
- LsarEnumerateAccountsWithUserRight
- LsarEnumerateAccountRights
- LsarQueryTrustedDomainInfo
- LsarOpenPolicy2
- LsarGetUserName
- LsarQueryInformationPolicy2
- LsarQueryTrustedDomainInfoByName
- LsarEnumerateTrustedDomainsEx
- LsarSetPolicyReplicationHandle
- LsarQueryDomainInformationPolicy
- LsarOpenTrustedDomainByName
- LsarLookupSids2
- LsarLookupNames2
- LsarLookupNames3
- LsarQueryForestTrustInformation

`12345778-1234-abcd-ef00-0123456789ac[v1.0]` (samsrv):

- SamrConnect
- SamrCloseHandle
- SamrQuerySecurityObject
- SamrLookupDomainInSamServer
- SamrEnumerateDomainsInSamServer
- SamrOpenDomain
- SamrQueryInformationDomain
- SamrEnumerateGroupsInDomain
- SamrEnumerateUsersInDomain
- SamrEnumerateAliasesInDomain
- SamrGetAliasMembership
- SamrLookupNamesInDomain
- SamrLookupIdsInDomain
- SamrOpenGroup
- SamrQueryInformationGroup
- SamrGetMembersInGroup
- SamrOpenAlias
- SamrQueryInformationAlias
- SamrGetMembersInAlias
- SamrOpenUser
- SamrQueryInformationUser
- SamrChangePasswordUser
- SamrGetGroupsForUser
- SamrQueryDisplayInformation2
- SamrGetDisplayEnumerationIndex
- SamrGetUserDomainPasswordInformation
- SamrQueryInformationDomain2
- SamrQueryInformationUser2
- SamrQueryDisplayInformation2
- SamrGetDisplayEnumerationIndex2
- SamrQueryDisplayInformation3
- SamrOemChangePasswordUser2
- SamrUnicodeChangePasswordUser2
- SamrGetDomainPasswordInformation
- SamrConnect
- SamrConnect3
- SamrConnect4
- SamrUnicodeChangePasswordUser3
- SamrConnect5
- SamrRidToSid

`3919286a-b10c-11d0-9ba8-00c04fd92ef5[v0.0]` (LSA DS access (lsarpc)):

- DsRolerGetPrimaryDomainInformation

`c681d488-d850-11d0-8c52-00c04fd90f7e[v1.0]` (efsrpc):

- EfsRpcOpenFileRaw
- EfsRpcReadFileRaw
- EfsRpcWriteFileRaw
- EfsRpcCloseRaw
- EfsRpcEncryptFileSrv
- EfsRpcDecryptFileSrv
- EfsRpcQueryUsersOnFile
- EfsRpcQueryRecoveryAgents
- EfsRpcRemoveUsersFromFile
- EfsRpcAddUsersToFile
- EfsRpcSetFileEncryptionKey
- EfsRpcNotSupported
- EfsRpcFileKeyInfo
- EfsRpcDuplicateEncryptionInfoFile
- EfsUsePinForEncryptedFiles
- EfsRpcAddUsersToFileEx
- EfsRpcFileKeyInfoEx
- EfsRpcGenerateEfsStream
- EfsRpcGetEncryptedFileMetadata
- EfsRpcSetEncryptedFileMetadata
- EfsRpcFlushEfsCache

Fig. 7. The names of the procedures we either successfully called or received BAD_STUB_DATA via null session named pipe access, when file sharing was enabled. There were no procedures that we discovered we could call via null session that we do not know the name of.

shutting down, and changing IP addresses were analyzed, and the protocol and protocol uses that we saw for each are listed in Appendix XXVII. We also collected a few weeks of traffic captures and identified the traffic that was apparently not the result of a direct user request. We summarize that data in Appendix XXVIII. Many of these messages represent requests to which a well-placed attacker could reply, in order to conduct an attack or to gather information.

We saw new protocols in use, including Teredo, IPv6, ICMPv6, NDP, MLD, Web Service Discovery (WS-Discovery), and LLMNR. Several of the protocols were used in apparent efforts to find Internet access on our isolated network, including SSDP, LLMNR, and NBNS, plus a Router Solicitation message. Similarly, LLMNR, NBNS, SSDP, WS-Discovery are used to automatically discover devices and services. We saw Vista clients that reside on the same network share information with each other; the protocols were WS-Discovery and UPnP (both resolve and probe).

VIII. Conclusion

The network stack in Windows Vista was rewritten from the ground up. By rewriting the stack, Microsoft has removed a large body of tested code and replaced it with newly written code, possibly introducing new corner cases and defects. This will provide for a more stable networking stack in the long term, but it will not be immune to attack, especially in the short term. A networking stack is a complex piece of software that typically takes many years to mature, though Microsoft seems to have successfully accelerated this process and reduced the timeframe, at least to a degree, by extensive testing and forethought.

Microsoft chose Vista as the platform to introduce new protocols and new implementations of old protocols. IPv6 and its supporting protocols are enabled during installation for the first time in Windows Vista. The IPv6 protocol is not new, but it has yet to see widespread deployment. To support the process of transitioning from IPv4 networks to IPv6 networks, and to increase the usefulness of peer-to-peer technologies, Microsoft has also enabled IPv6 tunneling support in Windows Vista; IPv4-based tunneling also appears to be available.

Some of these tunneling protocols, including Teredo, restore global addressability to hosts behind NAT firewalls, increasing the exposure of many users. We found that Vista requires a capable firewall to be running in order for Teredo tunneling to be active; this also appears to be the case for IPv4 over IPv6. These are sensible security precautions but cannot compensate for all of Teredo's problematic security implications. Tunneling methods can be used to evade security controls, and that

is what Teredo does (though that was not the intent). Unless network firewalls and IDSs are specifically aware of this protocol, they will not be applying the appropriate filtering to the IPv6 packet and its contents; this reduces defense-in-depth, and may result in a failure to apply important security controls. Compounding this issue is if Teredo will often be used on. Teredo is enabled by default and we found that it was readily used, despite Microsoft's apparently inaccurate statements[36] that downplay its level of activity. In addition, in our study of Vista's Teredo implementation, we found that some security features recommended by the Microsoft-developed standard were implemented minimally or at a strength less than that recommended by the standard.

Firewalls and IDSs will have to consider the presence of new Vista machines on their networks. If left unhandled and unchecked, IPv6 and its accompanying transition technologies allow an attacker access to hosts on private internal networks without the administrator expecting this global accessibility. Unwanted access can be prevented by analysis of IPv6 protocols in the firewall or IDS or by completely blocking all IPv6 protocols. Implementation-specific behavior of the new Vista stack allows an attacker to create ambiguous traffic that may be improperly interpreted by a passive intrusion detection device. IDSs will have to faithfully replicate Vista's behavior when analyzing data destined for Vista hosts. IDSs will also have to analyze new protocols and new versions of existing protocols or face being blind to their traffic.

In support of peer-to-peer communications, such as People Near Me and Windows Meeting Space and other local network discovery, Microsoft Vista supports the new serverless host information protocols LLTD, LLMNR, and PNRP. Taken together, these technologies provide mechanisms to discover and deliver payloads between peers. These features are critical to the success of Microsoft's peer-to-peer initiative, but are also the same features an attacker needs to deliver malicious content to his victims. We expect IPv6 and the new peer-to-peer protocols to play an increasing role in the delivery of malicious payloads as these technologies see wider deployment.

In our review of LLTD and its implementation on Vista, we found that the display of the network topology map that Vista provides is not entirely trustworthy. As of the 5472 build, it was possible with simple packet injection to achieve different kinds of spoofing on the map (possibly even tricking a user to visit an unintended web site as a result) and to cause the mapping to completely fail. We did find definite signs that Microsoft used a secure development life-cycle though, from design to implementation.

Vista has a new version of Windows Firewall which may contain unexpected quirks. One case, that at least users may find surprising, is that when firewall exceptions become enabled due to a feature such as People Near Me and Windows Meeting Space, they are not disabled as a result of turning off the feature; this can keep functionality such as ping active or make it easier for an attacker. Although the accessible ports and built-in firewall exceptions seem generally reasonable, we did find that in Vista's default configuration, TCP port 5357 was usable from off-host.

IX. Future Work

Our analysis of the networking technologies available in Windows Vista was both broad and sometimes deep. Due to the finite amount of time available, we could not test and review all areas of interest. In this new stack, with new protocols, many potential areas of interest still exist.

At the link layer, we did not investigate the Ethernet, PPP or PPPoE protocols, nor any of the link layer tunneling protocols, and we did not investigate all of the features of the ND protocol. We did look into the LLTD protocol, and we could look at the reference and third-party implementations; we could also update our results with a release or later build. We did not study precisely when ARP broadcast replies would be used.

At the network layer, we did not look into the ISATAP or 6to4 tunneling protocols. Tunneling protocols often invalidate some of the assumptions made in the design of other protocols, which may have security consequences. An analysis of attacks that could be performed in conjunction with tunneling protocols would likely be fruitful. We did not get an opportunity to try the different tunneling methods that appear to be implemented (IPv4 over IPv6, IPv4 over IPv4, IPv6 over IPv6, IPv6 over IPv4 and GRE) in order to understand what is required for them to be usable; nesting these might also prove interesting. All the tunneling protocols should be tested more thoroughly for implementation flaws. It would also be useful to map out the IPv6 options and option lengths that Vista supports.

At the higher levels of the protocol stack we left the LLMNR, PNRP, SSDP, and UPnP protocols completely untouched. These protocols should be analyzed and their security implications understood. The SMB2 protocol was covered summarily, but could also benefit from a deeper analysis. The firewall could also use more testing, particularly to determine how the firewall behaves when the same port is reused in IPv4 and IPv6 by different programs (or if that is even possible).

References

[1] Aboda, B., D. Thaler, L. Esibov, "Link-local Multicast Name Resolution (LLMNR)," RFC 4795, January 2007, ⟨http://www.ietf.org/rfc/rfc4795.txt⟩

[2] Arkko, J., J. Kempf, B. Zill, and P. Nikander, "SEcure Neighbor Discovery (SEND)," RFC 3971, March 2005, ⟨http://www.ietf.org/rfc/rfc3971.txt⟩

[3] Bellovin, S., "Defending Against Sequence Number Attacks," RFC 1948, May 1996. ⟨http://www.ietf.org/rfc/rfc1948.txt⟩

[4] Bellovin, S., "Security Problems in the TCP/IP Protocol Suite," *Computer Communications Review* 2:19, pp 32–48, April 1989. ⟨http://www.cs.columbia.edu/~smb/papers/ipext.pdf⟩

[5] Bellovin, S., "A Technique for Counting NATted Hosts," Proc. *Second Internet Measurement Workshop*, November 2002. ⟨http://www.cs.columbia.edu/~smb/papers/fnat.pdf⟩

[6] British Telecom, "BT Exact IPv6 Tunnel Broker". ⟨https://tb.ipv6.btexact.com/⟩

[7] Cain, B. and S. Deering, I. Kouvelas, B. Fenner, A. Thyagarajan, "Internet Group Management Protocol, Version 3", RFC 3376, October 2002. ⟨http://www.ietf.org/rfc/rfc3376.txt⟩

[8] Conta, A. and S. Deering, "Internet Control Message Protocol (ICMPv6) for the Internet Protocol Version 6 (IPv6) Specification," RFC 2463, December 1998. ⟨http://www.ietf.org/rfc/rfc2463.txt⟩

[9] Cover Pages, “Microsoft Releases Web Services Dynamic Discovery Specification (WS-Discovery)”, *Cover Pages*, October 2004. ⟨http://xml.coverpages.org/ni2004-02-17-b.html⟩

[10] Crawford, M., “Transmission of IPv6 Packets over Ethernet Networks,” RFC 2464, December 1998, ⟨http://www.ietf.org/rfc/rfc2464.txt⟩

[11] Davies, J. “Changes to IPv6 in Windows Vista and Windows Server ‘Longhorn’”, The Cable Guy, October 2005. ⟨http://www.microsoft.com/technet/community/columns/cableguy/cg1005.mspx⟩

[12] Davies, J. “Network Location Types in Windows Vista”, The Cable Guy, Sept 2006. ⟨http://www.microsoft.com/technet/community/columns/cableguy/cg0906.mspx⟩

[13] Deering, S. and R. Hinden, “Internet Protocol Version 6 (IPv6) Specification,” RFC 2460, December 1998, ⟨http://www.ietf.org/rfc/rfc2460.txt⟩

[14] Denis-Courmont, R., “Miredo : Teredo IPv6 tunneling for Linux and BSD”. ⟨http://www.simphalempin.com/dev/miredo/⟩

[15] Farinacci, D., T. Li, S. Hanks, D. Meyer, and P. Traina, “Generic Routing Encapsulation (GRE)”, RFC 2784, March 2000. ⟨http://www.ietf.org/rfc/rfc2784.txt⟩

[16] Frantzen, M. and S. Xiao, “ISIC - IP Stack Integrity Checker 0.06,” ⟨http://www.packetfactory.net/Projects/ISIC/⟩

[17] Frost, G., “Quality Windows AV Experience (qWAVE) and Network Quality of Service (QoS)”, Microsoft presentation. ⟨http://download.microsoft.com/download/9/8/f/98f3fe47-dfc3-4e74-92a3-088782200fe7/TWMO05011_WinHEC05.ppt⟩

[18] Fyodor, “Nmap (Network Mapper),” ⟨http://www.insecure.org/nmap/⟩

[19] Fyodor, “Remote OS detection via TCP/IP Stack FingerPrinting,” October 1998, ⟨http://www.insecure.org/nmap/nmap-fingerprinting-article.html⟩

[20] Hauser, V., “Attacking the IPv6 Protocol Suite,” CORE05, November 2005. ⟨http://www.thc.org/papers/vh_thc-ipv6_attack.pdf⟩

[21] Hinden, R. and B. Haberman, “Unique Local IPv6 Unicast Addresses”, RFC 4193, October 2005. ⟨http://www.ietf.org/rfc/rfc4193.txt⟩

[22] Hoagland, J., “The Teredo Protocol: Tunneling Past Network Security and Other Security Implications”, Symantec Response whitepaper, November 2006. ⟨http://www.symantec.com/avcenter/reference/Teredo_Security.pdf⟩

[23] Huitema, C., “Teredo: Tunneling IPv6 over UDP through Network Address Translations (NATs)”, RFC 4380. February 2006. ⟨http://www.ietf.org/rfc/rfc4380.txt⟩

[24] Internet Assigned Number Authority, “Protocol Numbers,” IANA registry. ⟨http://www.iana.org/assignments/protocol-numbers⟩

[25] Internet Assigned Number Authority, “Port Numbers,” IANA registry. ⟨http://www.iana.org/assignments/port-numbers⟩

[26] Johnson, D., C. Perkins and J. Arkko, “Mobility Support in IPv6”, RFC 3755. June 2004. ⟨http://www.ietf.org/rfc/rfc3755.txt⟩

[27] Marchand, J-B, “Network Store Interface”, Windows Network Services Internals, May 2006. ⟨http://www.hsc.fr/ressources/articles/win_net_srv/msrpc_winnsi.html⟩

[28] Marchand, J-B, “RPC services protection”, Windows Network Services Internals, May 2006. ⟨http://www.hsc.fr/ressources/articles/win_net_srv/rpc_services_protection.html⟩

[29] Microsoft, “Appendix A: TCP/IP Configuration Parameters”. ⟨http://technet2.microsoft.com/WindowsServer/en/library/db56b4d4-a351-40d5-b6b1-998e9f6f41c91033.mspx⟩

[30] Microsoft, “Be Wary of Other RPC Endpoints Running in the Same Process,” Platform SDK. ⟨http://msdn2.microsoft.com/en-us/library/aa373564.aspx⟩

[31] Microsoft, ”/GS (Buffer Security Check)”, MSDN, ⟨http://msdn2.microsoft.com/en-us/library/8dbf701c.aspx⟩

[32] Microsoft, “IPv6 Transition Technologies,” November 2002. ⟨http://www.microsoft.com/windowsserver2003/techinfo/overview/ipv6coexist.mspx⟩

[33] Microsoft, “Link Layer Topology Discovery Protocol Specification”, Sept 2006.⟨http://www.microsoft.com/whdc/Rally/LLTD-spec.mspx⟩

[34] Microsoft, “Microsoft Security Bulletin MS06-032”, June 2006. ⟨http://www.microsoft.com/technet/security/Bulletin/MS06-032.mspx⟩

[35] Microsoft, “Quality Windows Audio-Video Experience - qWave”, August 2002. ⟨http://www.microsoft.com/whdc/device/stream/qWave.mspx⟩

[36] Microsoft. “Teredo Overview.” ⟨http://www.microsoft.com/technet/prodtechnol/winxppro/maintain/teredo.mspx⟩.

[37] Microsoft, “Tracing”, MSDN. ⟨http://msdn2.microsoft.com/en-us/library/ms764325.aspx⟩

[38] Microsoft, “Understanding Mobile IPv6”, January 2007. ⟨http://www.microsoft.com/downloads/details.aspx?FamilyID=f85dd3f2-802b-4ea3-8148-6cde835c8921⟩

[39] Microsoft, “Windows Rally: Web Services on Devices”. ⟨http://www.microsoft.com/whdc/rally/rallywsd.mspx⟩

[40] Microsoft, “Teredo”, MSDN. ⟨http://msdn2.microsoft.com/en-us/library/aa965909.aspx⟩

[41] Microsoft, “Link Layer Topology Discovery (LLTD) Responder”, KB 922120. ⟨http://www.microsoft.com/downloads/details.aspx?familyid=4F01A31D-EE46-481E-BA11-37F485FA34EA⟩

[42] Microsoft, “Windows Driver Kit: Network Devices and Protocols: NET_BUFFER”. ⟨http://msdn2.microsoft.com/en-us/library/ms798961.aspx⟩

[43] Microsoft, “Windows Filtering Platform”. ⟨http://www.microsoft.com/whdc/device/network/WFP.mspx⟩

[44] Microsoft, “Windows Rally: Connectivity Technologies for Devices”. ⟨http://www.microsoft.com/whdc/rally/default.mspx⟩

[45] Microsoft, “Windows Rally Development Kit”, January 2007. ⟨http://www.microsoft.com/whdc/rally/rallykit.mspx⟩

[46] Morris, R., “A Weakness in the 4.2 BSD Unix TCP/IP Software,” Bell Labs Computer Science Technical Report 117, February 1985. ⟨http://pdos.csail.mit.edu/~rtm/papers/117.pdf⟩

[47] Narten, T. and R. Draves, “Privacy Extensions for Stateless Address Autoconfiguration in IPv6”, RFC 3041, January 2001. ⟨http://www.ietf.org/rfc/rfc3041.txt⟩

[48] Narten, T., E. Nordmark, and W. Simpson, “Neighbor Discovery for IP Version 6 (IPv6),” RFC 2461, December 1998. ⟨http://www.ietf.org/rfc/rfc2461.txt⟩

[49] Newsham, T. and J. Hoagland, “Windows Vista Network Attack Surface Analysis: A Broad Overview” (first edition), Symantec Response whitepaper, July 2006. ⟨http://www.symantec.com/avcenter/reference/ATR-VistaAttackSurface.pdf⟩

[50] Newsham, T. and T. Ptacek, “Insertion, Evasion and Denial of Service: Eluding Network Intrusion Detection,” Secure Networks, Inc. Technical Report, January 1998. ⟨http://citeseer.ist.psu.edu/ptacek98insertion.html⟩

[51] Newsham, T., “The Problem with Random Increments,” Guardent Technical Report, February 2001. ⟨http://www.thenewsh.com/~newsham/random-increments.pdf⟩

[52] Plummer, D., “An Ethernet Address Resolution Protocol,” STD 37, RFC 826, November 1982. ⟨http://www.ietf.org/rfc/rfc0826.txt⟩

[53] Postel, J., “Internet Protocol,” STD 5, RFC 791, ⟨http://www.ietf.org/rfc/rfc791.txt⟩

[54] Ricketts, M. “SendIP”, Project Purple. ⟨http://www.earth.li/projectpurple/progs/sendip.html⟩

[55] Scobleizer, “Abolade Gbadegesin and team - Networking in Windows Vista,” Channel 9 interview, ⟨http://channel9.msdn.com/Showpost.aspx?postid=116349⟩

[56] Seki, H., “Fingerprinting through RPC,” BlackHat 04, July 2004. ⟨http://www.blackhat.com/presentations/win-usa-04/bh-win-04-seki-up2.pdf⟩

[57] Song, D., DSniff 2.3, December 2000. ⟨http://naughty.monkey.org/~dugsong/dsniff/⟩

[58] The UPnP Forum, “UPnP Forum”. ⟨http://www.upnp.org/⟩

[59] Vida, R., L. Costa, “Multicast Listener Discovery Version 2 (MLDv2) for IPv6”, RFC 3810, June 2004. ⟨http://www.ietf.org/rfc/rfc3810.txt⟩

[60] The Wireshark Project, “SMB2,” ⟨http://wiki.wireshark.org/SMB2⟩

[61] The Wireshark Project, “Wireshark: The World’s Most Popular Network Protocol Analyzer”. ⟨http://www.wireshark.org/⟩

[62] Zalewski, M., “Strange Attractors and TCP/IP Sequence Number Analysis,” BindView Technical Report, April 2001. ⟨http://www.bindview.com/Services/Razor/Papers/2001/tcpseq.cfm⟩
[63] Zhang, J. “Diagnosing SideBySide Failures”, Junfeng Zhang’s Windows Programming Notes, April 2004. ⟨http://blogs.msdn.com/junfeng/archive/2006/04/14/576314.aspx⟩

APPENDIX I
TEST NETWORKS

Three different test network were used for the results presented in this paper. The main one (section I-A) used the release build of Vista; unless otherwise noted, this is the network that is in use. Our LLTD-related research was completed using build 5472 on the LLTD test network (section I-B) and our dedicated Teredo-related research was completed using Vista RC2 on the Teredo test network (section I-C).

A. Main Test Network

Our main test network consists of three physical hosts and three additional virtual hosts, as shown in Figure 8. The virtual hosts run under VMware Workstation 5.5 running on Windows XP, and are connected to the test network via bridged mode. The test network is isolated and has no routers or switches or any access outside the network; the three physical hosts are connected via a 10/100 Mbps Ethernet hub. Inside Vista, the network is typically configured as private network[12]; however, Vista sometimes automatically changes this setting (e.g. when it does not recognize the network), so this may not be the case consistently.

All four Vista hosts (two virtual, two laptops) are running clean installations of the Release To Manufacturing (RTM) build of Vista (build 6000, November 2006). Minimal configuration changes were made, except where otherwise noted in this report. Static IPv4 addresses from the 192.168.0.0/24 address space were configured on Vista. No IPv6 configuration was conducted, so the Vista machines have a link-local address (fe80::/64); these addresses are RFC 3041 privacy addresses[47], which we never saw change (except as a result of neighbor discovery spoofing, see section VI). A single account was created on all the hosts, and that was the result of Vista's guided setup.

A virtual Debian Linux host served as our analysis platform. On this host, we used shell variables to represent the IPv4, IPv6, and MAC addresses of various hosts, and used those in place of the actual addresses on the command line; we found this method improves the accuracy of testing and makes it simpler. Those shell variables are shown in this paper as part of Linux command lines (though we show the actual addresses for Windows command prompt and in the output of scripts).

We list the usual addresses and associated shell variables for the hosts in the following table:

Hostname	**IPv4 address**	**IPv6 address**	**MAC address**
acervista	`192.168.0.200 ($acerIP4)`	`fe80::ed59:b7ac:fc61:f865 ($acerLL6)`	`00:c0:9f:d2:0c:f8 ($acerMAC)`
hpvista	`192.168.0.201 ($hpIP4)`	`fe80::45dc:1fa6:3777:a480 ($hpLL6)`	`00:14:c2:d5:7e:96 ($hpMAC)`
vmvista	`192.168.0.203 ($vmIP4)`	`fe80::19e6:47a7:f579:3dfc ($vmLL6)`	`00:0c:29:72:e4:82 ($vmMAC)`
vmvista2	`192.168.0.204 ($vm2IP4)`	`fe80::f426:13fe:e8e7:720c ($vm2LL6)`	`00:0c:29:1b:50:aa ($vm2MAC)`
linux	`192.168.0.102 ($linuxIP4)`	`fe80::20c:29ff:fecd:b316 ($linuxLL6)`	`00:0c:29:cd:b3:16 ($linuxMAC)`

B. LLTD Test Network

The computing environment used to perform the analysis of LLTD was as follows:

- 1 x Linksys ADSL Hub and Router
- 1 x Windows XP SP1 Host
- 2 x Windows Vista Beta 2 (32-bit), Build 5472 Hosts

The two Microsoft Windows Vista hosts were running as guest operating systems within VMware Server 1.0.0. A logical network diagram can be seen in Figure 9.

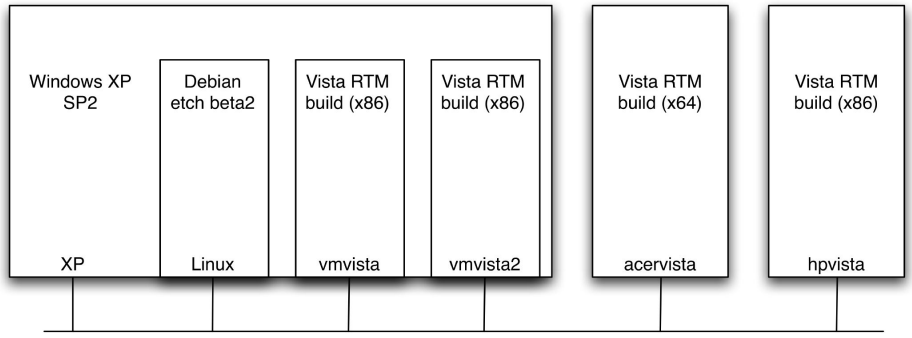

Fig. 8. The logical layout of our main test network. Three virtual hosts run under VMware on Windows XP.

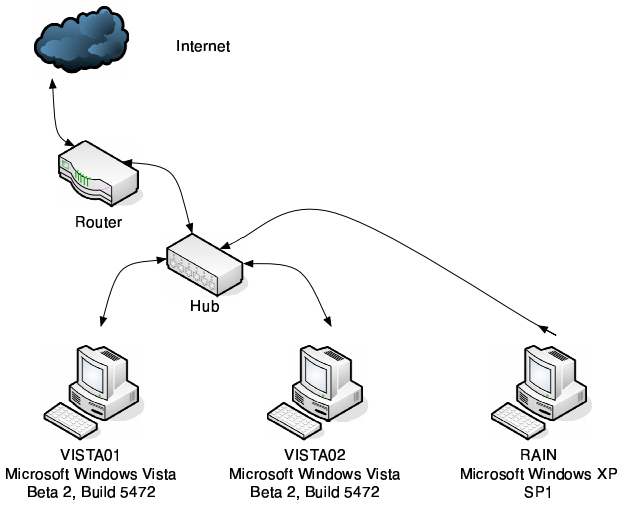

Fig. 9. The logical layout of LLTD test network.

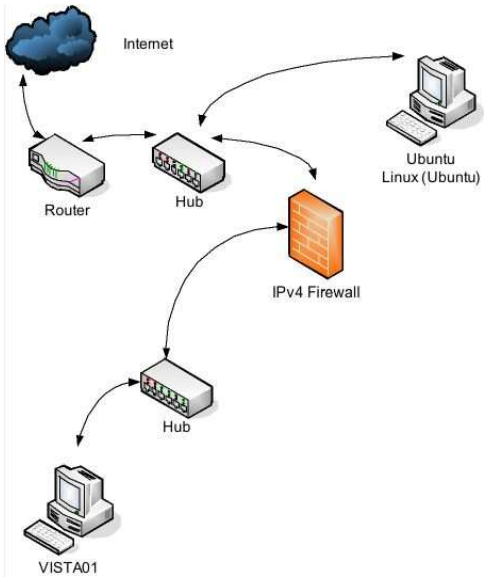

Fig. 10. The logical layout of Teredo test network.

C. Teredo Test Network

The computing environment used to perform the analysis of Teredo was as follows:

- 1 x Linksys ADSL Hub and Router
- 1 x Linux Host
- 1 x Windows Vista Host

A logical network diagram can be seen in Figure 10. This Vista host ran the latest Vista builds available at the time of testing, which was Beta 2 (5536), RC1 (5600), or RC2 (5744).

All testing was done with a restricted cone NAT.

APPENDIX II
LLTD INTRODUCTION

As part of Symantec's research into the attack surface presented to the network by Windows Vista, we studied the Link Layer Topology Discovery (LLTD) protocol. We looked into its security model and attempted to find previously unknown vulnerabilities in the LLTD protocol and its implementation in Microsoft Windows Vista. This research was performed on the latest Beta 2 build available at that time, which was 5472. This has not been updated for the release build of Vista: the Vista build that is the subject of most of the rest of this report. We introduce LLTD in this appendix and present our analysis and findings in Appendix III.

A. Background

The Link Layer Topology Discovery (LLTD) protocol is a layer 2 protocol that operates over 802.3 (Ethernet) and 802.11 (Wireless Ethernet) to aid in the discovery and documentation of network topology of small networks. In addition, LLTD can also be used in the detection and location of network bottlenecks which result in a lower quality of service (QoS). LLTD was originally designed and developed as part of the Windows Rally set of technologies which is described in the following terms by Microsoft, "designed to provide manufacturers of network-connected devices with an architecture that enables effortless setup, more secure and manageable connectivity to other devices and computers, and rich end-user experiences."[44].

In [33], Microsoft describes LLTD as follows: "... the LLTD protocol operates at Layer 2 in the OSI reference model and as such is not routable. The protocol is suitable only for networks comprising a single subnet, such as a small office network or a home network."

Microsoft goes on to describe the services which are available in LLTD:

- Quick discovery
- Topology discovery
- Quality of service diagnostics for network test
- Quality of service diagnostics for cross-traffic analysis

The purpose of Symantec's research was to identify generic attacks against the protocol, as well as Microsoft's implementation within Microsoft Windows Vista.

In January 2007, Microsoft released a Windows Rally development kit and sample code[45]. This first release is designed to help in the development of the LLTD protocol. It was not available as a source of insight in time for this research.

B. LLTD Protocol Overview

LLTD messages use Ethernet type 0x88D9 and are compromised of two base headers which are present in all packets. A third upper-layer header varies depending on the type of LLTD service and function. There are four layers of headers: Ethernet, the Demultiplex Header, the Base Header, and an upper layer header.

Ethernet (802.3 or 802.11)	Topology Upper Layer Headers:	QoS Upper Layer Headers:
Demultiplex Header:	Discover	QoSInitalizeSink
LLTD Version	Hello	QoSReady
Type of Service	Query	QoSProbe
LLTD Function	Query Response	QoSQuery
Base Header:	Query Large	QoSQueryResp
Real Source MAC	Query Large Response	QoSReset
Real Destination MAC	Emit	QoSError
LLTD Sequence Number	Probe	QoSAck
	Flat	QoSCounterSnapshot
	Train	QoSCounterResult
	Ack	QoSCounterLease

The typical protocol flows can be seen in Figure 11.

C. LLTD Security Model

As LLTD is a non-routable protocol which elicits only basic information from hosts there are minimal requirements with regards to authentication, authorization and confidentiality. LLTD's security has been primarily designed to thwart Denial of Service attacks being triggered by the usage of LLTD traffic against a LAN.

The only exception to this is that certain packets can be broadcast. An attacker can conceivably send these to the LAN's broadcast address and affect many different hosts if a vulnerability were discovered. The packets that can be sent to the broadcast address are shown in tables 12 and 13 (both based on [33]).

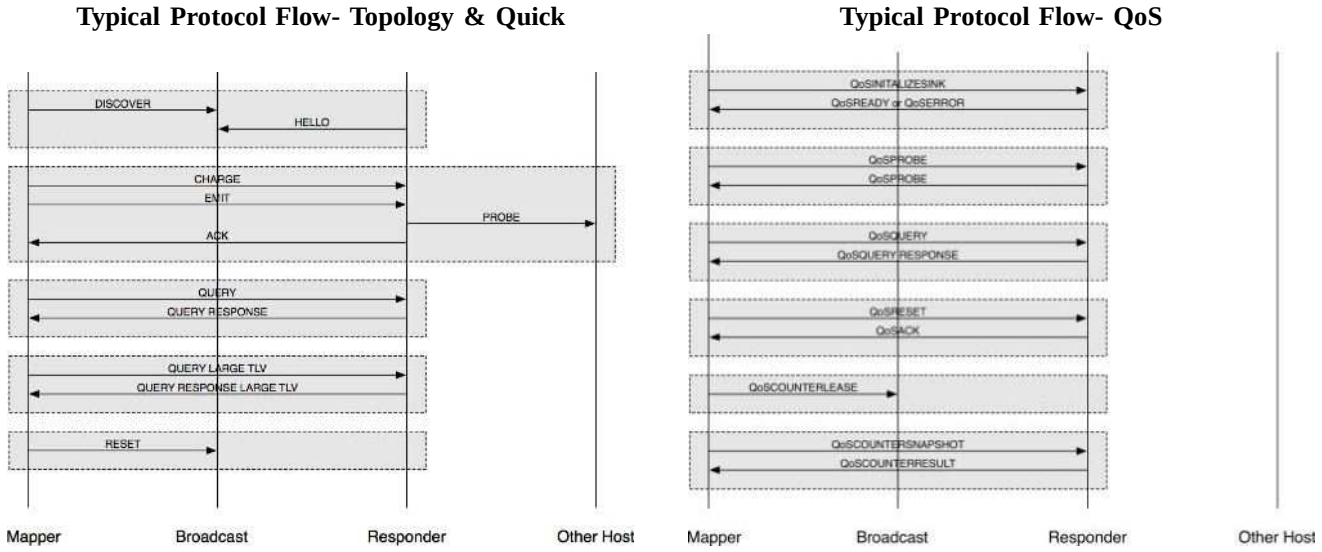

Fig. 11. The typical communication that occurs during a topology discovery and QoS analysis and how they relate.

Function	Value	Broadcast?
Discover	0x00	Required
Hello	0x01	Required
Emit	0x02	No
Train	0x03	No
Probe	0x04	No
Ack	0x05	No
Query	0x06	No
QueryResp	0x07	No
Reset	0x08	Permitted
Charge	0x09	No
Flat	0x0A	No
QueryLargeTlv	0x0B	No
QueryLargeTlvResp	0x0C	No

Fig. 12. The LLTD topology related packet types and their broadcast state.

As previously mentioned, LLTD's security features are designed to mitigate denial-of-service conditions. Within the specification ([33]), Microsoft makes explicit exactly how and when data should be sent, received, and processed.

The security mechanisms within LLTD can be broadly described as follows.

- Minimal traffic sent to the broadcast address.
- No traffic with a source address of the broadcast address should be replied to.
- A mapping host must accrue credit with a responder in order for the responder to send a PROBE in response to an EMIT command. Credit is based upon both size of data and number of packets. Credit is accrued through the mapping host sending CHARGE packets to the responder.
- Accrued credit is valid only within a strict time limit; this restriction prevents malicious hosts accruing vast amounts of credit in order to unleash on a target.
- Strict time limits are enforced when processing certain protocol states and packets, such as QUERY and QUERY RESPONSE packets.

The emit phase of the protocol has the following security precautions, which are performed on all triplets within the EMIT packet[33]:

> "For security reasons, a responder must perform the following checks before placing Train or Probe frames on the wire:
>
> - The Emit request must not have been sent to the broadcast address.
> - Train and Probe src field must equal the Responder's normal address or be within the range of the organizationally unique identifier (OUI) that is allocated to Microsoft for this protocol.
> - Trains and Probe dst field must not be Ethernet broadcast or multicast."

Function	Value	Broadcast?
QosInitializeSink	0x00	No
QosReady	0x01	No
QosProbe	0x02	No
QosQuery	0x03	No
QosQueryResp	0x04	No
QosReset	0x05	No
QosError	0x06	No
QosAck	0x07	No

Fig. 13. The LLTD QoS related packet types and their broadcast state.

These security mechanisms mitigate the impact of a whole range of different attacks previously seen with other protocols, such as amplification attacks.

APPENDIX III
LLTD ANALYSIS AND FINDINGS

A. Vista LLTD Implementation

The implementation of LLTD on Vista is broken down into a number of operating system components; these are described in Figure 14.

The key client and server components (lltdio.sys, rspndr.sys, lltdsvc.dll and lltdapi.dll) were all verified as compiled with the /GS flag under Microsoft Visual Studio[31]. The mechanism enabled by this flag is designed to mitigate against the successful exploitation of stack based buffer overflows.

A small amount of function use analysis was conducted against lltdsvc.dll. The result confirmed that all string handling functions are performed with safe equivalents, such as the following:

- StringCbCopyW(ushort *,uint,ushort const *)
- StringCbPrintfW(ushort *,uint,ushort const *,...)
- StringCchCatA(char *,uint,char const *)
- StringCchCatW(ushort *,uint,ushort const *)
- StringCchCopyA(char *,uint,char const *)
- StringCchCopyW(ushort *,uint,ushort const *)
- StringCchLengthA(char const *,uint,uint *)
- StringCchLengthW(ushort const *,uint,uint *)
- StringCchPrintfA(char *,uint,char const *,...)

This result, combined with our analysis of the implementation through fuzzing and test case creation, demonstrates that Microsoft has made a concerted effort to protect against common vulnerabilities that may yield a remote compromise of the host.

It should be noted that there were examples of unsafe functions calls such as memcpy() and memmove() used in a number of the kernel drivers. However, the instances analyzed did not reveal any exploitable conditions.

We also note is that the QoS component of LLTD in Microsoft Windows Vista appears to be taken from the qWave (Quality Windows Audio-Video Experience) framework[17], [35], which is designed specifically for QoS services in A/V rich wireless environments.

B. Disabling LLTD Within Vista

LLTD is covered by the “Network Discovery” item in the “Sharing and Discovery” section component of the “Network and Sharing Center” control panel of Microsoft Windows Vista. It should be noted that disabling this item disables the “Network Discovery” firewall group (see Appendix XXI-C.1), and hence this action affects UPnP, SSDP, NetBIOS, WS-Discovery, and LLMNR, in addition to LLTD.

Microsoft provides the option to disable this on a per-network-interface basis, regardless of its demarcation as public or private (see Figure 15).

It was observed that if “Network Discovery” is turned off while the Network interface is configured as a “Private Network” and the user selects “View full map”, then no network traffic will originate from the host. The result is that only the user’s

File	Role
lltdio.sys	Kernel I/O driver which handles all NDIS interactions for discovery; presents to the operating system as \Device\lltdio; responsible for discovery of other hosts on the network as well as determining network bandwidth.
rspndr.sys	Kernel I/O driver which handles all NDIS interactions for responding; presents to the operating system as \Device\rspndr; responsible for answering mapper requests to enable the host to be discovered on the network.
qwavedrv.sys	Kernel I/O driver related to qwave.dll.
lltdsvc.dll	Service which runs as “Local Service” and communicates with the LLTD I/O kernel driver lltdio.sys.
lltdapi.dll	Used by the user’s explorer process to interact with the lltdsvc service via COM.
qwave.dll	Used for QoS related services; holds its configuration parameters in the registry, which allows the turning on of tracing.
networkmap.dll	Used by the user’s Explorer process to draw the network topology map and interacts with lltdapi.dll.

Fig. 14. The LLTD Windows Vista components and their roles.

Fig. 15. Option to enable or disable Network Discovery on 5472 Vista

host, network, and Internet objects are included in the final topology map. However, while in this state, if another host sends an LLTD DISCOVER packet, then the host with "Network Discovery" disabled responds with a HELLO packet. The result is that the responding host is included in the topology map of the network.

The above is not true if the network interface is configured as a "Public Network". In this scenario, no information will be solicited from the target host.

Another method of disabling LLTD on a per-network-interface basis is to manually unbind the kernel drivers from the specific interface. This can be achieved via the "Network Connections" control panel applet (See Figure 16).

C. Topology Map in Vista

The topology map, as displayed on Microsoft Windows Vista (Figure 1), is actually produced by the user's Explorer process with the support of two DLLs (NetworkMap.dll and XMLLite.dll), both of which reside in the System32 directory on the host. The Explorer process loads NetworkMap.dll; NetworkMap.dll is itself a COM object that operates as a Control Panel plug-in. NetworkMap.dll, in turn, uses the LLTDAPI to communicate with the LLTDSVC service via COM to initiate a mapping session. LLTDAPI and LLTDSVC create the map which is then passed back to the NetworkMap COM component for parsing and display.

D. Hosts with Multiple Interfaces

We found that hosts with multiple interfaces do not retransmit LLTD packets received on one interface over a different interface. That is, any LLTD traffic (including probes) received on an interface for one network is not retransmitted to a second

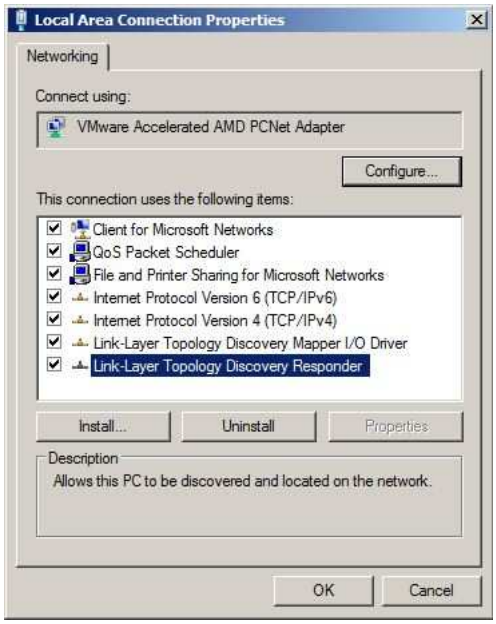

Fig. 16. Protocol Bindings Under Vista

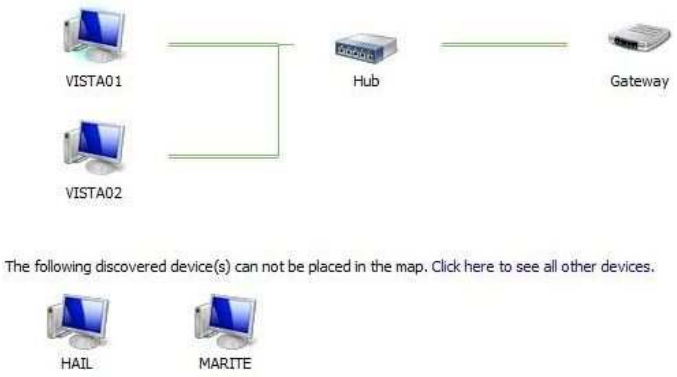

Fig. 17. Additional hosts discovered via NetBIOS

network, even if there is another interface that connects to it. Machines that are multi-homed, and that wish to perform a network map, must do so separately for each interface present.

E. Interaction with Other Protocols

LLTD does not interact with other protocols directly; however, the Microsoft Windows Vista Network Map does. Figure 17 shows two additional hosts discovered on the network: “HAIL” and “MARITE”. The Network Map learned of these additional hosts from the NetBIOS name table (Figure 18). Due to the fact that neither of these hosts are running LLTD responders, the Network Map has no way of knowing where they exist within the topology, and thus they are placed at the bottom of the diagram. Hosts that are discovered by way of the NetBIOS name table become clickable within the map (Figure 19). If the user left-clicks on the host’s icon, a connection attempt is made via NetBIOS using its UNC name. If the user right-clicks upon the hosts icon, they are presented with the menu seen in Figure 19.

The other protocol which the Network Map uses to discover and interact with hosts is UPnP[58]. UPnP-enabled devices also become clickable in a similar fashion to those discovered via NetBIOS. However, instead of attempting a connection via NetBIOS when the user left-clicks on the device, it will instead automatically open Internet Explorer and connect the host on port 80. When the user right-clicks on an UPnP-enabled device, they are able to display a properties dialog box (see Figure 20). This provides a wealth of information about the device obtained via UPnP.

F. Policy Controls

Microsoft provides group policy settings to allow administrators to control the usage of LLTD within an enterprise environment. All of the policy settings exist under the registry key: HKLM\SOFTWARE\Policies\Microsoft\Windows\LLTD.

```
C:\Users\Ollie>nbtstat -r

    NetBIOS Names Resolution and Registration Statistics
    ----------------------------------------------------

    Resolved By Broadcast     = 19
    Resolved By Name Server   = 0

    Registered By Broadcast   = 6
    Registered By Name Server = 0

    NetBIOS Names Resolved By Broadcast
-----------------------------------------------
           MARITE         <00>
           HAIL
           MARITE
           HAIL
           MARITE
           HAIL
           MARITE
           VISTA02
```

Fig. 18. NBTSTAT output

Fig. 19. Right-click menu for NetBIOS discovered host

Each setting is a DWORD, which should be set to either 0 or 1 to enable or disable, respectively. Each of the group policy settings is explained in the following:

Policy Registry DWORD	Comment
EnableLLTDio	If set to 0 will disable the networking mapping ability under “Network and Sharing Center”
ProhibitLLTDioOnPrivateNet	If set to 1 will disable LLTD on interfaces marked as on a “Private” network.
AllowLLTDioOnPublicNet	If set to 1 will enable LLTD on interfaces marked as on a “Public” network.
AllowLLTDioOnDomain	If set to 1 will enable LLTD on a network interface where the domain controller for the domain the host is a member of can be communicated with.

G. Mapper and Responder Relationship

Within LLTD topology discovery are the concepts of “Mapper” and “Responder”. The Mapper, as the name implies, is the host that initiates a mapping session for the network, and is the host on which the topology map is displayed. The Responder is an agent that is present on one or more hosts, which when requested interacts with the Mapper[9].

A Mapper may communicate with one or more responders at any one time, however only one Mapper may be active on the network at a time.

H. Generation and Sequence Numbers

LLTD has the concepts of “generation” and “sequence numbers”. These are not designed to provide security as the number is only a 32-bit number that is typically incremented. For more information, refer to the sections “Base Header Format”, “Generation Numbers”, and “Sequence Number Management” in the LLTD specification document[33].

[9]An LLTD responder is available for Windows XP[41], but we have not studied it.

Fig. 20. UPnP output from Gateway

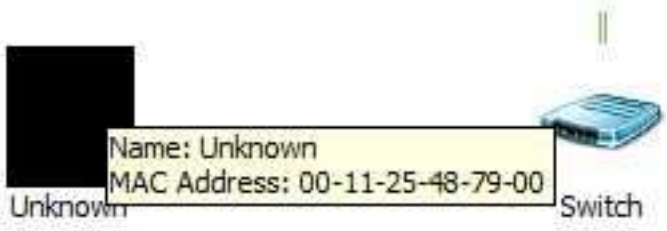

Fig. 21. Symantec LLTD Responder supplied icon

I. Device Supplied Images

This following does not describe an attack and it is only included for informational purposes. Within LLTD, two of the optional TLV types available within the HELLO packet are 0x0E (icon image) and 0x18 (detailed icon image). These allow a responder to supply their own Windows Icon to be shown in the Network Map.

The following restrictions are placed upon these images[33]:

> "..... ensure that the Icon Image TLV is set in the Hello frame if the image is smaller than or equal to 32,768 octets. Otherwise, set only this Detailed Icon Image TLV in the Hello frame if the icon image is greater than 32,768 octets and smaller than or equal to 262,144 octets."

Symantec implemented this feature within its LLTD responder, the result of which is shown in Figure 21. We simply supply a very small black bitmap icon file, which is shown in the network map.

This feature currently poses no threat to Microsoft Windows Vista. However, if an icon file format parsing bug is discovered in the future, this would be a vector for exploitation. Conceivably, this feature can be used in pranks or as part of a deception-based attack.

It should also be noted that two formats are supported for the icon images: the traditional bitmap and the Portable Network Graphics (PNG) format. Research into Microsoft's handling of PNG parsing was outside the scope of this project, so may represent an uninvestigated attack surface.

```
+--------+--------+--------+--------+--------+--------+
|00000010|00000100|          Characteristics          |
+--------+--------+--------+--------+--------+--------+
Type=0x02 Length=4

This property allows a responder to report various simple characteristics of its host or the
network interface that it is using.
MSB
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|N|N|F|M|L| | | | | | | | | | |
|P|X|D|W|P|0|0|0|0|0|0|0|0|0|0|
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| | | | | | | | | | | | | | | |
|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
                                  LSB
Bits 0-28: Reserved, must be zero.
Bit 27: (LP) 1  =   Interface is looping back outbound packets.
Bit 28: (MW) 1  =   Device has management Web page accessible via HTTP protocol. The mapper
constructs a URL from the reported IPv6 address. If one is not available, the IPv4 address is
used instead. The URL is of the form: http://<ip-address>/
Bit 29: (FD) 1  =   Interface is in full duplex mode.
Bit 30: (NX) 1  =   Interface is NAT-private side.
Bit 31: (NP) 1  =   Interface is NAT-public side.
```

Fig. 22. LLTD Hello packet TLV characteristics, from [33].

J. Internal XML Representation

Within the EXPLORER process that displays the Network Map on Microsoft Windows Vista, XML is used to describe the network elements, their properties, and their connections. This is then parsed and the network map drawn. During the course of the research, it was hypothesized that it may be possible to perform an XML injection attack to influence the network map.

In order to either prove or disprove this hypothesis, it is important to understand the internal XML schema that is used by the Network Map application. We show the results in Appendix IV.

The research showed that XML injection is not possible due to the fact that the less than (<) and greater than (>) characters, which are used to delimit an XML tag, are encoded as their equivalent XML entities (“<” and “>”, respectively). For example, a hostname of “</>A” would be converted to “</>A”. The result of this encoding is that the characters become benign in the context of the XML.

K. Attack: Spoof and Management URL IP Redirect

One of the attacks discovered during the course of the research into LLTD was the “Spoof and Management URL IP Redirect” attack.

Within LLTD HELLO packets, one of the TLV fields contains the characteristics of the device in question (see Figure 22). One of these characteristics (located at bit 28) is the whether or not the device has a management web interface. This “MW” characteristic can be combined with two other TLV types 0x07 (the device’s “IPv4 Address”) and with 0x0F (the device’s “Machine Name”) in order to spoof another device on the network.

The result of this attack can be seen in Figure 23: two hosts appear on the network with the same name. The key difference between the two hosts displayed is that the second is fake (our malicious host), which is to say while the MAC address is valid (i.e. the attacker’s), the real hostname is “RAIN” and its IP address is 192.168.1.102. Microsoft Windows Vista’s mapper does not appear to verify that the MAC and IP address combination contained in an LLTD packet relate to actual hostnames or IP addresses on the network.

The result of this is that when the second VISTA02 host (lower of the two) network map item is clicked (the method of gaining access to other hosts) or the user right-clicks and selects “Management URL”, the user’s Internet Explorer is automatically loaded and directed to the external IP address of 216.239.113.101 (www.news.com). This relationship can be seen in Figure 24. This popup displays for a few seconds when the user hovers their mouse over the network device’s icon. Thus an attacker can cause a user to go to a different web site than they expected.

L. Attack: Spoof on Bridge

The “Spoof on Bridge” attack works by making the Microsoft Windows Vista’s mapper modify its network diagram to make our attacking host appear physically closer to the spoofed host. In the previous example it was easy to differentiate between the two hosts due to their different locations within typology.

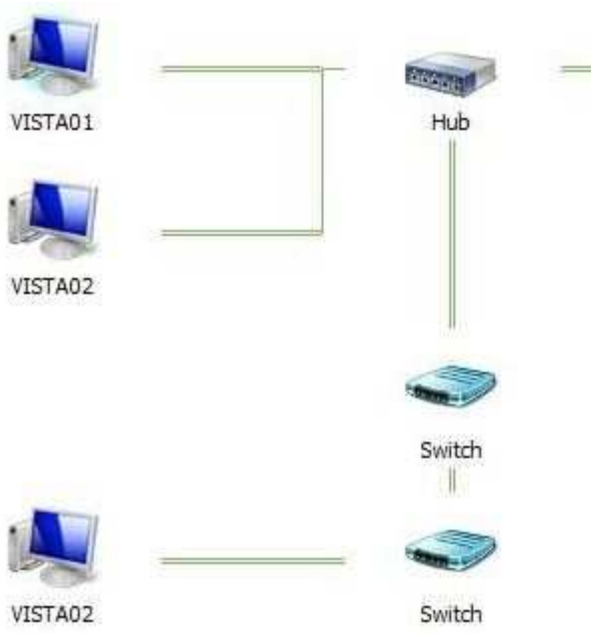

Fig. 23. Two hosts with same name, different IPs and MACs

The “Spoof on Bridge” attack works by spoofing the Layer 2 MAC address, providing the MAC address of the spoofed host while maintaining a different LLTD BASE header MAC. The BASE header “Real Source” field under normal circumstances in a HELLO packet would be same MAC as that found at Layer 2 (Ethernet / Wireless Ethernet). In instances where this is not the case it is assumed that the spoofed host is acting as a bridge and that the spoofing host is actually behind it. The result of the attack can be seen in Figure 25. This attack can be extended to cause many fake hosts to appear in the network topology map.

One interesting characteristic of this attack is that the attacking host cannot be seen by the spoofed host when it performs a network map. One advantage of this type of attack for the attacker, is that the attacking host only has to send a HELLO packet; all other packets, such as the PROBE and QUERY packets, are sent for processing to the Layer 2 MAC of the host that is being spoofed. Switches that enforce Layer 2 security controls to mitigate ARP spoofing would mitigate the risk of this attack as well.

M. Attack: Total Spoof

The Total Spoof attack, as the name implies, allows an attacker to completely spoof another host on the network. This is essentially a race condition. When the attacking host sees the DISCOVERY packet from the mapper, they must be able to generate a HELLO packet with the spoofed host’s MAC address in the Layer 2, BASE header, and HELLO TLV parts of the packet.

If the attacking host’s reply is received by the mapper before that from the spoofed host, then the details in the attacking host’s reply will be used in the network map instead of the details provided by the spoofed host. If the attacking host’s reply is observed by the spoofed host (which it should be, as it is sent to the broadcast address), before the spoofed host transmits its own reply, it will not transmit. Even though the spoofed host does not respond to the mapper’s initial DISCOVERY packet, it will still reply to QUERY and PROBE packets allowing the mapping operation to complete successfully. The result of this attack can be seen in Figure 26.

We can see in Figure 26 that the details from the spoofed HELLO packet sent by “RAIN” are used on the network diagram when describing VISTA02. This can be used by an attacker to redirect traffic for known hosts in certain circumstances, such as for a management URL to an external IP address. This should not impact users who use this method to connect to file and print sharing, as that will use the UNC path and not the IP address.

Switches that enforce Layer 2 security controls to mitigate the risk of ARP spoofing would also mitigate the risk of this attack as well.

N. Denial of Service

There are a number of ways to cause the mapping process to totally fail. The research showed that the mapping process could be made to fail reliably by not sending ACKs in response to EMIT packets from the mapper. This result can be seen in Figure 27. A single malicious responder on the network can achieve this.

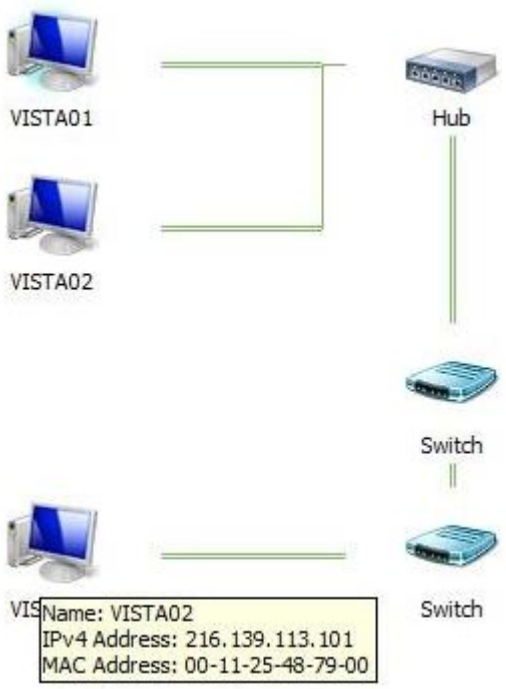

Fig. 24. Vista network map information popup

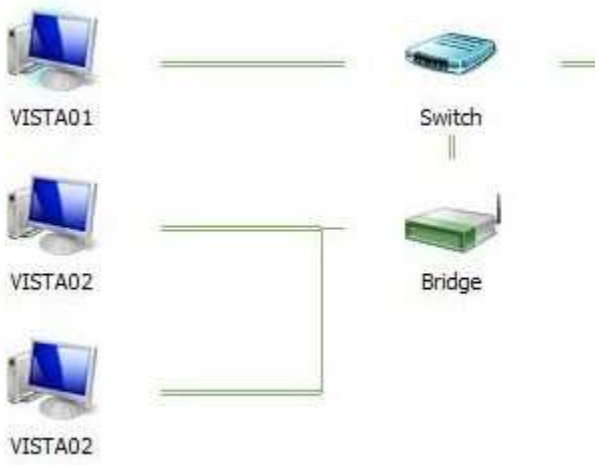

Fig. 25. “Spoof on Bridge” attack

Fig. 26. Result of the “Total Spoof” attack

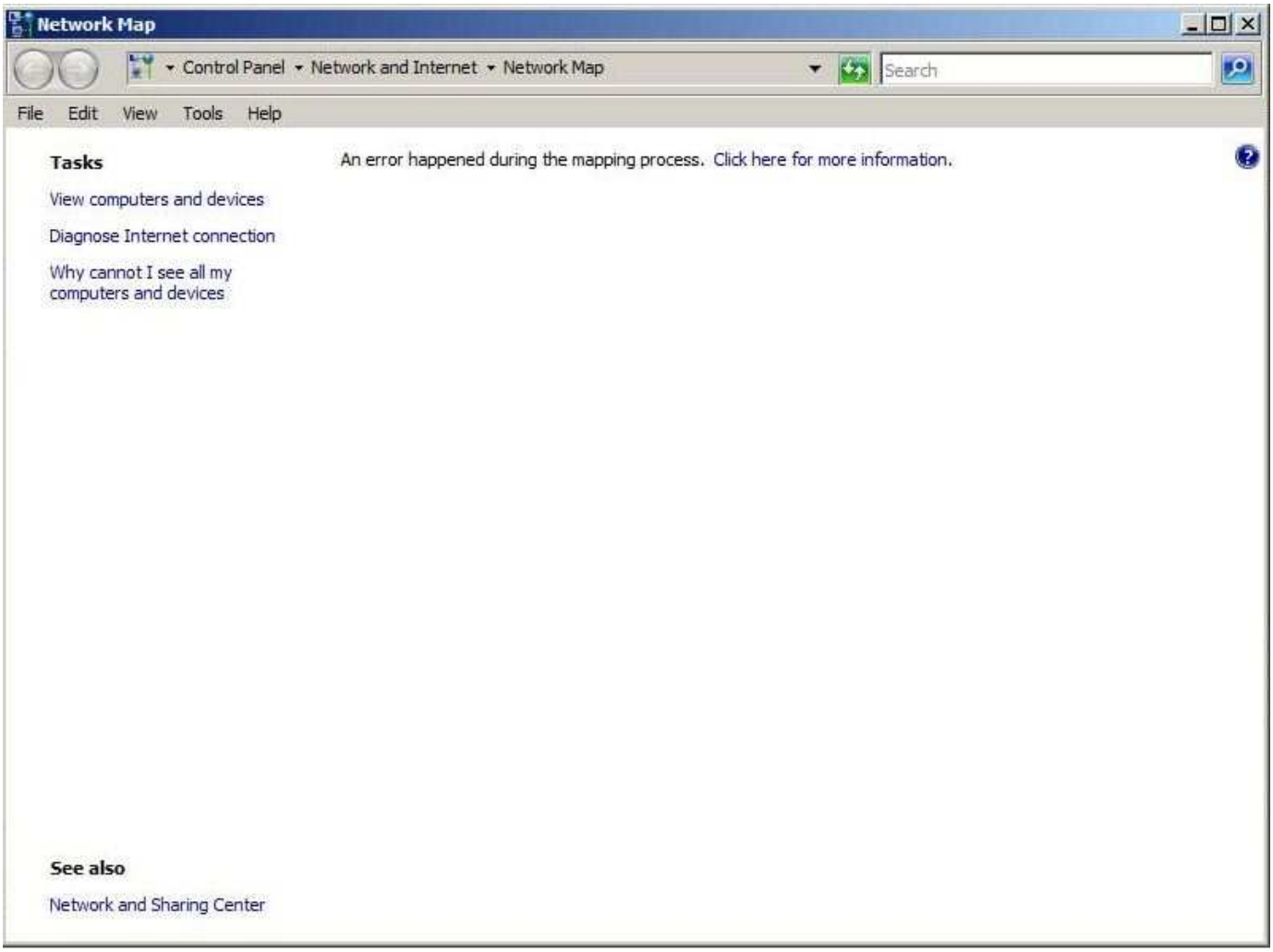

Fig. 27. Mapping denial of service.

A number of SideBySide errors were be present in the Application Event Log on the mapping host. Using the method described by Microsoft to debug SideBySide errors[63], the following debug information was obtained:

```
=================
Begin Activation Context Generation.
Input Parameter:
    Flags = 0
    ProcessorArchitecture = x86
    CultureFallBacks = en-US;en
    ManifestPath = C:\Windows\system32\NetworkMap.dll
    AssemblyDirectory = C:\Windows\system32\
    Application Config File =
-----------------
INFO: Parsing Manifest File C:\Windows\system32\NetworkMap.dll.
    INFO: Manifest Definition Identity is (null).
    ERROR: Line 3: XML Syntax error.
ERROR: Activation Context generation failed.
End Activation Context Generation.
```

As can be seen above, the error occurs due a malformed XML element. The result is that no map is drawn. To sustain this, the attacker must simply have a malicious responder on the segment in question.

O. Quality of Service Component

The Quality of Service (QoS) component of LLTD was briefly researched. It was determined that, as this is a non-broadcast protocol, it poses even less of a threat than the topology component. In addition, the QoS protocol packets only support variable

length payloads in three packets: QoSProbe (sent from the controller (source) to the sink (destination)), QosQueryResp (sent from the sink to the controller), and QoSCounterResult (sent from the sink to the controller).

The only observations that raised concern were those relating to the QoSProbe packet:

- probegap contains an 802.1p (QoS) element to be included in the 802.1q component of the packet. However, no 802.1q/p enabled switches were available during this research to enable further study of the potential impact of this.
- probegap contains a variable size payload, which is replicated on the return path. No maximum size exists for this; as a result, it is only limited to the maximum 802.3 or 802.11 frame size.
- during timed probe the sink (receiver) has to be able to allocate space for 82 probe packets which can contain the variable size payload; as a result, the sink has to dynamically allocate memory to store this.

The implementation was not assessed.

P. Other Attempted Test Cases

Figure 28 documents the test cases which were attempted during the course of the research in to LLTD and the observed results.

Documented Test Cases and Results

Test	LLTD Pkt Type	Result
Set the LLTD Demultiplex type to TOPOLOGY and service to PROBE and have no LLTD Base header.	Demultiplex	No impact
Set the LLTD Base "Real Source" header field to the broadcast MAC address (FF:FF:FF:FF:FF:FF).	Base	Device's MAC address is noted as FF:FF:FF:FF:FF:FF in network map
Set the LLTD Base "Real Source" header field to the broadcast MAC address (FF:FF:FF:FF:FF:FF) in a HELLO packet.	Base	No response from hosts
Send unknown TLV types.	Hello	No impact
Send many TLV types, up to the maximum frame size.	Hello	No impact
Set Machine Name to over 32bytes in length and have the correct length in TLV section of packet.	Hello	Device called Unknown in network map
Send an IP address over 4 bytes in length and have the correct length in TLV section of packet.	Hello	IP address omitted from network map
Send an IP address over 4 bytes in length and have an incorrect length in TLV section of packet.	Hello TLV	Device omitted from network map
Omit the mandatory "Host ID" TLV field.	Hello	Device omitted from network map
Include multiple entries for the same TLV field.	Hello	Only first instance is used in network map
Use a machine name TLV field containing a format string.	Hello	No Impact
Use a machine name TLV field containing XML tags.	Hello	No Impact
Use a machine name TLV field containing 16 < characters (since these will get expanded to "<").	Hello	No Impact
Do not send an ACK to an EMIT.	Emit	This causes network mapping fails with "An error happened during the mapping process".
Set the MORE bit in Query Large Response packets for every response.	Query Large Response	This causes 195 requests to be generated by the mapper to the responder; after 195 the mapper stops sending requests. This has no net impact.
Set the Hardware ID in Query Large Response to over 200 bytes in length.	Query Large Response	No impact
Send Query Response packets with the number of descees set to 255 and the actual contents of one descee.	Query Response	The network mapping fails with "An error happened during the mapping process".
Send Query Response packets with the number of descees set to one more than the actual quantity included.	Query Response	The network mapping fails with "An error happened during the mapping process".
Send Query Response packets with the number of descees set to one less than the actual quantity included.	Query Response	The network mapping succeeds, but takes 114 packets to complete mapping operation between two hosts instead of the normal approximately 50.
Set the Hardware ID in a Query Large Response to contain prohibited characters—ASCII values less than 0x20 and more than 0x80, white space (0x20) and commas.	Query Large Response	No impact
Send Query Large packets to the broadcast address with types 0x00 through 0xFF with an offset of 0xFFFFFF.	Query Large	No impact
Send Query Large packets to the responder address with types 0x00 through 0xFF with an offset of 0xFFFFFF.	Query Large	No impact

Fig. 28. The LLTD test cases we attempted.

APPENDIX IV
XML FORMAT USED BY NETWORK MAP

The XML contained is Figure 29 is used by the Network Map application to produce the diagram contained in Figure 30. This XML is never written to disk but solely used within memory.

```
<map>
<node type="switch" link="root">
    <node type="host" link="direct">
        <prop guid="{00000000-0000-0000-0000-000000000000}" type="mac">00:0c:29:41:3e:3a</prop>
        <prop guid="{00000000-0000-0000-0000-000000000001}" type="mac">00:0c:29:41:3e:3a</prop>
        <prop guid="{58DF2E46-8CCD-4253-B61C-A97D341CCA3D}" type="uint32">6</prop>
        <prop guid="{A083365B-B0DB-4B11-984A-13FD2D95B303}" type="uint32">0</prop>
        <prop guid="{04FB5875-62A0-437B-BE51-864A741C3C0F}" type="uint64">3579545</prop>
        <prop guid="{B8C91D7B-5159-4894-B593-348F452E72A6}" type="uint32">10000000</prop>
        <prop guid="{2AD8768D-0C88-45E4-B174-0ED928254127}" type="ipv4">1921681103</prop>
        <prop guid="{9E459B9C-D773-4CA2-9CB0-1307FA3F41C8}" type="unicode">VISTA01</prop>
        <prop guid="{7CD2CBB4-BC07-438D-992B-5E33D23EB868}" type="uint32">2147483648</prop>
        <prop guid="{94E379F0-E45E-43EA-8CD1-DC1C7AF497F1}" type="ipv6">
fe80:0000:0000:0000:bdad:4b53:7aa1:df2b</prop>
    </node>

    <node type="switch" link="direct">
        <node type="host" link="direct">
            <prop guid="{00000000-0000-0000-0000-000000000000}" type="mac">00:11:25:48:79:00</prop>
            <prop guid="{00000000-0000-0000-0000-000000000001}" type="mac">00:11:25:48:79:00</prop>
            <prop guid="{58DF2E46-8CCD-4253-B61C-A97D341CCA3D}" type="uint32">0</prop>
            <prop guid="{A083365B-B0DB-4B11-984A-13FD2D95B303}" type="uint32">0</prop>
            <prop guid="{9E459B9C-D773-4CA2-9CB0-1307FA3F41C8}" type="unicode">&lt;/&gt;A</prop>
        </node>
    </node>
</node>
</map>
```

Fig. 29. Example of network map XML

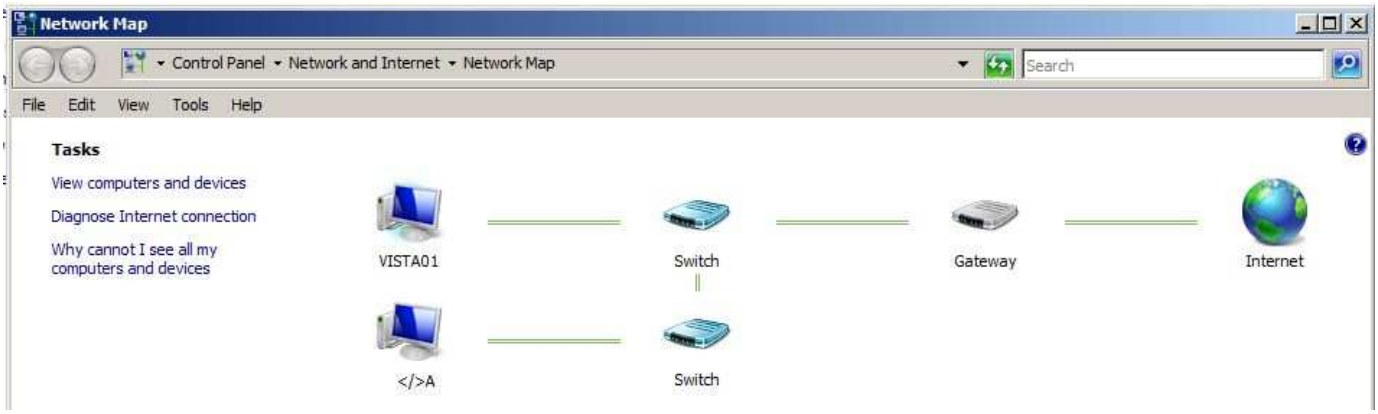

Fig. 30. Network map that resulted from example XML in Figure 29.

APPENDIX V
ARP SPOOFING

We observed that a Vista host accepts solicited responses, but only stores them if they are sent directly to it, as opposed to the Ethernet broadcast address. Vista hosts also overwrite existing ARP table entries with the data contained in unsolicited ARP message. Vista hosts are vulnerable to a denial-of-service attack when they receive a gratuitous ARP for their own address.

We used the arpspoof tool from the dsniff[57] suite for our testing. Testing was performed using a Linux host as the attacker and a Vista machine as the target (the host that gratuitous ARPs were sent to). As a victim (the IP address whose ARP entry is being spoofed) we variously used real (in use) and unused IP addresses on our test network. We used the arp -a command on the target to list the ARP entries (IP addresses, MAC addresses, and entry state). To determine which MAC address Vista would actually use, we ran “ping” on the target with the spoofed IP address, and we monitored network traffic from Linux using Ethereal [61]; if there is no ARP table entry, this is also a way of inducing an ARP request.

We found that no ARP table entry is created for an unsolicited ARP reply and it is apparently not stored in any way. That is true when the ARP message is directed at the victim’s MAC address specifically, and also when it is sent to the Ethernet broadcast address (FF-FF-FF-FF-FF-FF).

For directed replies, an unsolicited reply will overwrite (update) an existing ARP table entry, and an entry will be created for an apparent response to a request (if the victim IP address is in use, this would set up a race to provide the first reply, and to provide an overwriting response). As expected, we found that ARP tables entries are used when needed.

For broadcast replies, an unsolicited reply will overwrite an existing ARP entry. Unlike directed replies, a solicited reply that is sent to the broadcast address, rather than to the requesting address will not create an ARP table entry, but it will get used. This needs to be studied more; there may be a short wait before the broadcast reply is used.

If a Vista machine receives a conflicting directed or broadcast message for the IP address that it is statically configured with, that address becomes unusable and a pop-up message announces the conflict (similar to Windows XP). Attempts to use the network (for example, to run “ping”), result in error code 1231 (ERROR_NETWORK_UNREACHABLE). This condition persists until the network interface is reset.

APPENDIX VI
NEIGHBOR DISCOVERY SPOOFING

We observed that Windows Vista hosts will not process unsolicited Neighbor Advertisements (NAs) unless they update an existing neighbor cache entry. However, it is still possible to perform a redirect attack by sending spoofed NAs in response to actual queries or by blindly sending out NAs periodically. In the event of a conflict, Vista automatically configures a replacement RFC 3041 (privacy) address[47].

We performed our testing with the ndspoof.py script that was constructed specifically for the purpose. Testing was performed using a Linux host as the attacker and a Vista machine as the target (the host that gratuitous NAs were sent to). As a victim (the IP address whose cache entry is being spoofed), we variously used real (in use) and unused IPv6 addresses on our test network. All packets sent out had the solicited bit left unset.

We used the “netsh interface ipv6 show neighbors” command on the target to list the neighbor cache entries (IPv6 addresses, MAC addresses, and entry state). To determine which MAC address Vista would actually use, we ran “ping” on the target with the IPv6 address being spoofed, and we monitored network traffic from Linux using Ethereal; if there is no cache entry, this is also a way of inducing an Neighbor Solicitation.

As with ARP, no neighbor cache entry is created for an unsolicited NA and it is not stored. This is true for both a targeted message and for messages sent out to ff02::1 (local scope all-nodes multicast address). However, existing entries are updated and entries are created for apparently solicited NAs.

For a directed conflict (when the NA is sent to the existing owner of an address), we have seen the following for a link-local RFC 3041 address:

1) the victim sends out four of its own NAs before giving up
2) the victim generates a new RFC 3041 (privacy) address
3) the victim begins Duplicate Address Detection (DAD) for the new address
4) having seen no one else claim that address, it begins using it. MLDv2 and other packets associated with moving to a new IPv6 address are seen.

For conflicts that Vista notices by listening to ff02::1 (again for link-local RFC 3041 addresses), Vista immediately goes to step 2 above (generating a new address). In both cases, no popup is presented to the user as the conflict is handled automatically.

APPENDIX VII
IPV4 ID GENERATION

We monitored the IPv4 ID of packets sent out from a specific Windows Vista machine over the course of a few weeks, gathering 609,482 data points across different IP protocols. We observed that Vista increments the IPv4 ID used for each packet, regardless of the protocol that it is used for and the destination. The range Vista uses is [0,0x7fff]; after sending 0x7fff, it uses 0 for the next packet. This is similar to Windows XP, except that XP uses the full range. This can be used for differentiation; if one sees a (naturally sent) packet from a host and its IP ID is 0x8000 or above, it is not Vista.

Here is some sample output of our ipid.py script from this data collection.

```
# ./ipid.py $acerIP4
0b67    192.168.0.200 -> 192.168.0.102 proto 1
0b68    192.168.0.200 -> 192.168.0.102 proto 1
0b69    192.168.0.200 -> 192.168.0.102 proto 1
0b6a    192.168.0.200 -> 192.168.0.102 proto 1
0b6b    192.168.0.200 -> 192.168.0.102 proto 1
...
1021    192.168.0.200 -> 192.168.0.102 proto 1
1022    192.168.0.200 -> 192.168.0.255 proto 17
1023    192.168.0.200 -> 192.168.0.102 proto 1
1024    192.168.0.200 -> 192.168.0.255 proto 17
1025    192.168.0.200 -> 192.168.0.102 proto 1
1026    192.168.0.200 -> 192.168.0.255 proto 17
1027    192.168.0.200 -> 192.168.0.255 proto 17
...
1ee6    192.168.0.200 -> 192.168.0.102 proto 1
1ee7    192.168.0.200 -> 192.168.0.102 proto 1
1ee8    192.168.0.200 -> 192.168.0.102 proto 6
1ee9    192.168.0.200 -> 192.168.0.102 proto 6
1eea    192.168.0.200 -> 192.168.0.102 proto 6
...
7ffe    192.168.0.200 -> 192.168.0.102 proto 1
7fff    192.168.0.200 -> 192.168.0.102 proto 1
0000    192.168.0.200 -> 192.168.0.102 proto 1
0001    192.168.0.200 -> 192.168.0.102 proto 1
0002    192.168.0.200 -> 192.168.0.102 proto 1
...
```

We needed to filter out some Land attack packets (from testing in Appendix XV) and four other packets in which the IPv4 address was forged, and presenting as from the Vista host being monitored.

To help analyze this, we wrote a script to fold up consecutive IP IDs. This is the full set of IP IDs observed:

```
0b67-0de7 (641 IDs)
0e5a-256c (5907 IDs)
27a8-27da (51 IDs)
281e-28a1 (132 IDs)
2a7f-2d0b (653 IDs)
2d0c-3581 (2166 IDs)
3583-3585 (3 IDs)
3587-358c (6 IDs)
358e-3599 (12 IDs)
359b-35a3 (9 IDs)
35a5-35a7 (3 IDs)
35a9-35ae (6 IDs)
35b0-35bb (12 IDs)
35bd-3b09 (1357 IDs)
3b0b-3b10 (6 IDs)
3b12-3b15 (4 IDs)
3b17-3b1d (7 IDs)
3b1f-3b22 (4 IDs)
3b24-3b4b (40 IDs)
3b4d-3b50 (4 IDs)
3b52-3b53 (2 IDs)
3b55-3b69 (21 IDs)
3b6b-3b71 (7 IDs)
3b73-3b75 (3 IDs)
3b77-3b7c (6 IDs)
3b7e-3b82 (5 IDs)
3b84-3b88 (5 IDs)
3b8a-3b8b (2 IDs)
3b8d-3b8f (3 IDs)
3b91-3b94 (4 IDs)
3b96-3b98 (3 IDs)
3b9a-3b9b (2 IDs)
3b9d-3b9f (3 IDs)
3ba2      (1 ID)
3ba4-3ba8 (5 IDs)
3bab      (1 ID)
3bae-3bb4 (7 IDs)
3bb8-3bc8 (17 IDs)
3bcb-3bcf (5 IDs)
3bd2-3bd4 (3 IDs)
3bd7      (1 ID)
3bd9-3bdc (4 IDs)
3bde-3bdf (2 IDs)
3be1-3be8 (8 IDs)
3bea-3bf3 (10 IDs)
3bf6      (1 ID)
3bf8-3bfa (3 IDs)
3bfd-3c03 (7 IDs)
3c05-3c17 (19 IDs)
3c19-3c1d (5 IDs)
3c1f-3c25 (7 IDs)
3c27      (1 ID)
3c29-3c3a (18 IDs)
3c3c-3c3d (2 IDs)
3c40      (1 ID)
3c42      (1 ID)
3c44      (1 ID)
3c46-3c4a (5 IDs)
3c4c-3c4f (4 IDs)
3c51      (1 ID)
3c53-7fff (17325 IDs)
0000-7fff (32768 IDs)
0000-7fff (32768 IDs)
0000-7fff (32768 IDs)
0000-7fff (32768 IDs)
0000-7fff (32768 IDs)
0000-7fff (32768 IDs)
0000-4d52 (19795 IDs)
4d54-4d55 (2 IDs)
4d57-4d58 (2 IDs)
4d5a-4d5b (2 IDs)
0a70-5836 (19911 IDs)
043f-0af4 (1718 IDs)
0af7-0f2d (1079 IDs)
57d6-7fff (10282 IDs)
0000-7fff (32768 IDs)
0000-7fff (32768 IDs)
0000-7fff (32768 IDs)
```

```
0000-7fff (32768 IDs)          0000-7fff (32768 IDs)          0000-7fff (32768 IDs)
0000-7fff (32768 IDs)          0000-7fff (32768 IDs)          0000-0f06 (3847 IDs)
0000-7fff (32768 IDs)          0000-7fff (32768 IDs)
```

The short breaks are believed to be due to packet loss (the network was occasionally flooded); the large gaps are believed to be due to reboots.

In addition, when Nmap[30] was run in verbose mode for OS detection (see Appendix XX), it reported “IPID Sequence Generation: Incremental”, which supports our conclusion.

APPENDIX VIII
IP FRAGMENT REASSEMBLY

Vista's networking stack behaves differently than earlier versions of the stack (in Windows XP and Windows 2000) when reassembling IP Fragments. We observed that Vista more strictly discarded IP packets with partially overlapping fragments; in many cases, such fragments were discarded. However, the stack never discarded packets with fully overlapping fragments, favoring the old data. We found IPv4 and IPv6 reassembly followed the same pattern.

In many but not all cases when the fragment was not reassembled, a fragmentation timeout message was sent, via ICMPv4 or ICMPv6, as appropriate after a delay of approximately 60 seconds. Thus, a given sequence of fragments can be classified as belonging to one of three groups: will-reassemble, will not reassemble but sends error, and will not reassemble and sends no error.

A. Fragmentation Background

In IPv4, fragmentation is specified in the base IPv4 header[53]. For IPv6, the fragmentation fields were moved to a separate Fragment extension header, which is only supposed to be included if the packet is a fragment[13]. In both cases there is a certain octet range in the reassembled packet that has been fragmented for transmission, which we refer to as the fragmentation space. In IPv4, this is the entire IPv4 payload. In IPv6, it is the space beyond the Fragment extension header; another extension header could precede this and the fragment space could contain the extension header before the IPv6 payload.

For both IPv4 and IPv6, there are four fields pertaining to fragmentation reassembly. An ID field identifies the packet that is being reassembled. An offset field indicates how deeply into the reassembled packet's IP payload the fragment should be placed; this is a multiple of eight octets. From the IP header length field (IPv4 total length and IPv6 payload length), one can infer the length of the fragment and hence, when added to the offset, the result is the ending offset of the fragment. The last field is a more fragments (MF) field; this is a bit which, if set, indicates that this fragment does not contain the end of the fragmentation space. Under normal circumstances, there is a single fragment which has MF unset. The maximum offset of that packet defines the size of the space that is being reassembled and hence the length of the IP payload.

Normally, exactly one fragment should exist that represents any given portion of the fragmentation space. Aside from the possibility of a buggy sending stack, the only legitimate case in which this is not true, is where a fragment becomes duplicated. Even in that case, the fragment data found in fragments should be identical, and the fragments should represent the same part of the fragmentation space.

However, there are illegitimate cases. Neither the IPv4 nor the IPv6 RFCs define how to handle badly overlapped fragments, fragments with mismatched data, and conflicting specifications for the fragment's end. Hence, different stacks reassemble differently, which is a pain point for network-based IDSs/IPSs, which must correctly match the recipient operating systems fragmentation reassembly behavior in order to correctly understand the effect of the reassembled packet[50].

B. Fragmentation Testing Methodology

We developed 64 test cases, each of which consists of a sequence of fragments to be sent in a particular order. Test cases were added in an ad hoc manner to explore which cases produced reassembly or error messages. Every case except #29 and #30 contain at least one piece of data for each part of the fragmentation space. Every test case except #7 and #30 contain conflicting data some part of the fragmentation space or contain fragments that do not overlap exactly.

Although the test cases are shared, we took different approaches to testing the reassembly under IPv4 and IPv6, largely due to protocol differences.

1) IPv4 Methodology: To test reassembly behavior under IPv4, we used a packet that, when reassembled, would be a UDP packet with checksums disabled (using the distinguished value of zero). We avoided sending conflicting UDP headers. However, there could be multiple ways to reassemble the UDP payload. We observed how the Vista stack interpreted it by observing the payload of the reassembled packets that were delivered to the application layer on the target machine.

Specifically, we started our fragorder.py tool on our analysis host with the arguments 1–64, which instruct it to serve up tests 1 through 64 in order. On the Vista machine, we started netcat as "`nc -u 192.168.0.102 999`" and repeatedly hit the Enter key. Fragorder.py monitors UDP packets coming into port 999 and sends the sequence of fragments for a test case in response to each Enter key pressed in netcat. If reassembly succeeds, the Vista stack passes the UDP payload to netcat, which displays it. For IPv4, we always append a newline character to any fragment that includes the last octet of the reassembly space, to ensure proper and unambiguous display through netcat. Any successful reassembly displayed is recorded to a file along with the test number. Since no proper socket had been set up on the analysis machine, we ran "`iptables --protocol icmp -A OUTPUT --icmp-type port-unreachable -j DROP`" to suppress Linux's natural response to the incoming UDP packet.

Prior to the above, we started a network monitoring script, fragorder_watcher.py, on our analysis host. This monitors the network for any packets that correspond to fragorder.py testing; specifically, it looks for the fragmented UDP packet and any fragmentation timeout ICMP messages. When given a signal to shut down, it reads the file in which we were recording successful re-assemblies, and reports on what it saw, organized by test. The following is an example of reported data:

```
Test 46:  (192.168.0.102 -> 192.168.0.200, id=a32e)
  Fragment # 1: [MF] hhhhhhhh1111111111111111
  Fragment # 2: [MF]                          2222222222222222
  Fragment # 3: [MF]                                  44444444
  Fragment # 4:                                                3333333333333333\n
    from 1164810579.6807 to +0.0019
  reassembled as:    hhhhhhhh1111111111111111222222222222222233333333333333333\n
```

It knows to which test number the fragmented packet corresponds, since it is encoded in the lower half of the IP ID. It also organizes the tests into the three result groups and reports those together.

2) IPv6 Methodology: Changes with IPv6 required us to take a different approach, which also allowed for a simpler setup. With IPv6, UDP checksumming is no longer optional. That means that we would need to anticipate the recipients method of reassembly, and use that to compute the checksum correctly; if the checksum was not computed correctly, the recipient stack would silently discard the packet.

We instead developed the approach of intentionally creating a condition in the reassembled packet that would cause the remote stack to respond with an ICMPv6 error message. ICMPv6 error messages are required to include the full content of the original packet (up to 1280 octets) which in this case is really the reassembled packet as perceived by the stack. Thus we can read the reassembly result directly on the analysis machine.

The method of generating an error packet we chose was to have the fragmentation payload consist of a destination options extension header that is encoded with No Next Header as the following header. The options encoded in the extension header began with 9f 04 00 00 00 00, which encodes the option type 0x9f with a four-byte value consisting of zeros. No option 0x9f has been defined, but since the first bit of the option code is set, RFC 2460[13] requires that an ICMPv6 Parameter Problem error be returned to inform the sender of this. Those first 6 octets of the extension header payload are fixed. Including the extension header, next header field and length, this leaves the first 8 octets of the reassembly space as overhead, as in the IPv4 case. The rest of the extension header is tested in the reassembly test; in our testing this did not correspond to valid options, but could have been made to correspond to valid options if required for successful testing.

To do the test, we started our fragorder6.py tool on our analysis host as "`./fragorder6.py $vmLL6%2 $vmMAC 1-64`" which causes the tests 1 though 64 to be sent to vmvista, with short pauses in between. If reassembly succeeds the Vista stack returns the parameter problem; if it fails, it may or may not respond with an fragmentation timeout message. As with the IPv4 case, we also had a network monitoring script (fragorder6_watcher.py) running during the test to capture and collate packets corresponding to fragorder6.py tests. When given a keyboard interrupt, the script reports what it saw, organized by result group. The following is an example:

```
Test 1:  (fe80::020c:29ff:fecd:b316 -> fe80::19e6:47a7:f579:3dfc, id=dca01)
  Fragment # 1: [MF]                 2222222222222222
  Fragment # 2: [MF]                                   5555555555555555
  Fragment # 3:                                                         6666666666666666
  Fragment # 4: [MF]                                4444444444444444
  Fragment # 5: [MF]                                oooooooooooooooo
  Fragment # 6: [MF]                           3333333333333333
  Fragment # 7: [MF] pppppppp1111111111111111
    from 1164060733.6128 to +0.0066
  got frag timeout at: +41.4019
```

Again here, the test number was encoded in the IP ID. It was also represented in the IPv6 flow label, so that it would be present in the reassembled packet.

C. Test Cases and Results

Figure 31 shows the IPv6 test cases in which reassembly succeeded, along with the result. Figure 32 shows the IPv6 test cases in which reassembly did not succeed, but a timeout error was produced, and Figure 33 shows the cases where reassembly failed and no error message was observed.

Our IPv4 testing produced nearly identical results. However, as tested, #16, #18, and #20 produced an error message. We tested without the added newline and produced the same result as IPv4. The addition of the newline to those cases (or anything that made the line other than a multiple of eight) caused the error message to be sent.

For IPv4 we noticed that, in certain cases where reassembly does not take place, an ICMPv4 parameter problem message for "bad ip header" with pointer 0 was sent. This happened whenever a case was similar to the following test case:

```
Test 20:  (192.168.0.102 -> 192.168.0.200, id=8914)
  Fragment # 1:                    22222222222222222222222222222222222222222222222\n
  Fragment # 2: [MF]               33333333333333333333333333333333333333333333333\n
  Fragment # 3: [MF] hhhhhhhh1111111111111111
    from 1164810560.9851 to +0.0090
  got frag timeout at: +41.0352
```

where an MF fragment corresponds in offsets to a previous fragment sent without MF. These fragments are logically contradictory.

```
Test 4:  (fe80::020c:29ff:fecd:b316 -> fe80::19e6:47a7:f579:3dfc, id=dcd04)
  Frag #1:                                                  6666666666666666
  Frag #2: [MF]                              4444444444444444
  Frag #3: [MF]                              ooooooooooooooo
  Frag #4: [MF]                         33333333
  Frag #5: [MF] pppppppp1111111111111111
    from 1164060748.6379 to +0.0589
 reassembled as:pppppppp1111111111111111333333334444444444444446666666666666666
    at +0.0607

Test 6:  (fe80::020c:29ff:fecd:b316 -> fe80::19e6:47a7:f579:3dfc, id=dcf06)
  Frag #1:                         33333333
  Frag #2: [MF] pppppppp1111111111111111
    from 1164060758.7082 to +0.0012
 reassembled as:pppppppp11111111111111133333333
    at +0.0022

Test 7:  (fe80::020c:29ff:fecd:b316 -> fe80::19e6:47a7:f579:3dfc, id=dd007)
  Frag #1: [MF]                                     55555555
  Frag #2:                                                  6666666666666666
  Frag #3: [MF]                              44444444
  Frag #4: [MF]                         33333333
  Frag #5: [MF] pppppppp1111111111111111
    from 1164060763.7135 to +0.0039
 reassembled as:pppppppp11111111111111133333333444444445555555566666666666666666
    at +0.0046

Test 46:  (fe80::020c:29ff:fecd:b316 -> fe80::19e6:47a7:f579:3dfc, id=df72e)
  Frag #1: [MF] pppppppp1111111111111111
  Frag #2: [MF]                         2222222222222222
  Frag #3: [MF]                                   44444444
  Frag #4:                                                  3333333333333333
    from 1164060958.8917 to +0.0042
 reassembled as:pppppppp11111111111111122222222222222223333333333333333
    at +0.0051

Test 47:  (fe80::020c:29ff:fecd:b316 -> fe80::19e6:47a7:f579:3dfc, id=df82f)
  Frag #1: [MF] pppppppp1111111111111111
  Frag #2: [MF]                         2222222222222222
  Frag #3: [MF]                                   44444444
  Frag #4:                                                  3333333333333333
  Frag #5: [MF]                                                    55555555
    from 1164060963.8977 to +0.0045
 reassembled as:pppppppp11111111111111122222222222222223333333333333333
    at +0.0040

Test 48:  (fe80::020c:29ff:fecd:b316 -> fe80::19e6:47a7:f579:3dfc, id=df930)
  Frag #1: [MF] pppppppp1111111111111111
  Frag #2:                         3333333333333333
  Frag #3: [MF]                    22222222
    from 1164060968.9042 to +0.0031
 reassembled as:pppppppp111111111111111333333333333333
    at +0.0022

Test 49:  (fe80::020c:29ff:fecd:b316 -> fe80::19e6:47a7:f579:3dfc, id=dfa31)
  Frag #1: [MF] pppppppp111111111+1
  Frag #2: [MF]                    22222222222222222222222222222222222222
  Frag #3: [MF]                    33333333333333333333333333333333
  Frag #4: [MF]                              444444444444444444444444
  Frag #5: [MF]                                   5555555555555555
  Frag #6: [MF]                                   66666666
  Frag #7:                                                                  77777777
    from 1164060973.9092 to +0.0060
 reassembled as:pppppppp111111111+122222222222222222222222222222222222277777777
    at +0.0064

Test 50:  (fe80::020c:29ff:fecd:b316 -> fe80::19e6:47a7:f579:3dfc, id=dfb32)
  Frag #1: [MF] pppppppp111111111+1
  Frag #2: [MF]                    22222222222222222222222222222222222222
  Frag #3: [MF]                    33333333333333333333333333333333
  Frag #4: [MF]                           444444444444444444444444
  Frag #5: [MF]                                   5555555555555555
  Frag #6: [MF]                           6666666666666666
  Frag #7:                                                                  77777777
    from 1164060978.9162 to +0.0059
 reassembled as:pppppppp111111111+122222222222222222222222222222222222277777777
    at +0.0069

Test 54:  (fe80::020c:29ff:fecd:b316 -> fe80::19e6:47a7:f579:3dfc, id=dff36)
  Frag #1: [MF] pppppppp111111111+1
  Frag #2: [MF]                    22222222222222222222222222222222222222
  Frag #3: [MF]                    33333333333333333333333333333333
  Frag #4: [MF]                                   444444444444444444444444
  Frag #5: [MF]                    55555555555555555555555555555555
  Frag #6: [MF]                           66666666666666666666666666666666
  Frag #7:                                                                  77777777
    from 1164060998.9378 to +0.0055
 reassembled as:pppppppp111111111+122222222222222222222222222222222222277777777
    at +0.0065

Test 55:  (fe80::020c:29ff:fecd:b316 -> fe80::19e6:47a7:f579:3dfc, id=e0037)
  Frag #1: [MF]                         2222222222222222
  Frag #2: [MF] pppppppp1111111111111111
  Frag #3: [MF]                                   44444444
  Frag #4:                                                  3333333333333333
  Frag #5: [MF]                                                    55555555
    from 1164061003.9452 to +0.0087
 reassembled as:pppppppp11111111111111112222222222222223333333333333333
    at +0.0047

Test 58:  (fe80::020c:29ff:fecd:b316 -> fe80::19e6:47a7:f579:3dfc, id=e033a)
  Frag #1: [MF] pppppppp1111111111111111
  Frag #2: [MF]                         2222222222222222
  Frag #3: [MF]                                   55555555
  Frag #4: [MF]                              4444444444444444
  Frag #5:                                                  33333333
    from 1164061018.9663 to +0.0043
 reassembled as:pppppppp11111111111111112222222222222225555555533333333
    at +0.0050

Test 59:  (fe80::020c:29ff:fecd:b316 -> fe80::19e6:47a7:f579:3dfc, id=e043b)
  Frag #1: [MF]                         2222222222222222
  Frag #2: [MF]                                   55555555
  Frag #3: [MF] pppppppp1111111111111111
  Frag #4: [MF]                              4444444444444444
  Frag #5:                                                  33333333
    from 1164061023.9723 to +0.0030
 reassembled as:pppppppp11111111111111112222222222222225555555533333333
    at +0.0039

Test 61:  (fe80::020c:29ff:fecd:b316 -> fe80::19e6:47a7:f579:3dfc, id=e063d)
  Frag #1: [MF] pppppppp1111111111111111
  Frag #2: [MF]                         2222222222222222
  Frag #3: [MF]                                   55555555
  Frag #4: [MF]                                             66666666
  Frag #5: [MF]                              44444444444444444444444
  Frag #6:                                                                33333333
    from 1164061034.0066 to +0.0087
 reassembled as:pppppppp11111111111111112222222222222225555555566666666333333333
    at +0.0092

Test 62:  (fe80::020c:29ff:fecd:b316 -> fe80::19e6:47a7:f579:3dfc, id=e073e)
  Frag #1: [MF] pppppppp1111111111111111
  Frag #2: [MF]                         55555555
  Frag #3: [MF]                                   2222222222222222
  Frag #4: [MF]                                             44444444
  Frag #5:                                                  3333333333333333
    from 1164061039.0185 to +0.0081
 reassembled as:pppppppp11111111111111155555555222222222222222333333333333333
    at +0.0090
```

Fig. 31. Successful IPv6 fragment reassembly test cases and results. On some of the long packets, we shorten the lines by using "+" to represent six additional instances of the previous character. We notice that the last fragment in tests #47 and #55 is extraneous (useless) under any realistic assembly policy, due to the fact that a complete packet was already sent by the time those fragments were sent.

```
Test 1:  (fe80::020c:29ff:fecd:b316 -> fe80::19e6:47a7:f579:3dfc, id=dca01)
  Frag #1: [MF]                    2222222222222222
  Frag #2: [MF]                                   5555555555555555
  Frag #3:                                                  6666666666666666
  Frag #4: [MF]                              4444444444444444
  Frag #5: [MF]                              ooooooooooooooo
  Frag #6: [MF]                         3333333333333333
  Frag #7: [MF] pppppppp1111111111111111
    from 1164060733.6128 to +0.0066
  got frag timeout at: +41.4019

Test 2:  (fe80::020c:29ff:fecd:b316 -> fe80::19e6:47a7:f579:3dfc, id=dcb02)
  Frag #1: [MF]                                   5555555555555555
  Frag #2:                                                  6666666666666666
  Frag #3: [MF]                              4444444444444444
  Frag #4: [MF]                              ooooooooooooooo
  Frag #5: [MF]                         3333333333333333
  Frag #6: [MF] pppppppp1111111111111111
    from 1164060738.6242 to +0.0059
  got frag timeout at: +39.5266

Test 3:  (fe80::020c:29ff:fecd:b316 -> fe80::19e6:47a7:f579:3dfc, id=dcc03)
  Frag #1: [MF]                                   5555555555555555
  Frag #2:                                                  6666666666666666
  Frag #3: [MF]                              4444444444444444
  Frag #4: [MF]                              ooooooooooooooo
  Frag #5: [MF]                         33333333
  Frag #6: [MF] pppppppp1111111111111111
    from 1164060743.6325 to +0.0049
  got frag timeout at: +41.1388

Test 5:  (fe80::020c:29ff:fecd:b316 -> fe80::19e6:47a7:f579:3dfc, id=dce05)
  Frag #1: [MF]                                   4444444466666666
  Frag #2:                                                  6666666666666666
  Frag #3: [MF]                              4444444444444444
  Frag #4: [MF]                              4444444444444444
  Frag #5: [MF]                         33333333
  Frag #6: [MF] pppppppp1111111111111111
    from 1164060753.7000 to +0.0065
  got frag timeout at: +41.0888
```

Fig. 32. The test cases where IPv6 fragment reassembly did not succeed and an ICMPv6 fragmentation timeout message was produced (part 1). Note that the elapsed times shown are not accurate due to significant clock skew; the error messages were sent approximately 60 seconds after the fragments were sent.

```
Test 11:  (fe80::020c:29ff:fecd:b316 -> fe80::19e6:47a7:f579:3dfc, id=dd40b)
  Frag #1: [MF]                 2222222222222222
  Frag #2:                                      3333333333333333
  Frag #3: [MF] ppppppppp1111111111111111
    from 1164060783.7302 to +0.0022
  got frag timeout at: +39.4492

Test 12:  (fe80::020c:29ff:fecd:b316 -> fe80::19e6:47a7:f579:3dfc, id=dd50c)
  Frag #1:                                      3333333333333333
  Frag #2: [MF]                 2222222222222222
  Frag #3: [MF] ppppppppp1111111111111111
    from 1164060788.7336 to +0.0020
  got frag timeout at: +41.1369

Test 13:  (fe80::020c:29ff:fecd:b316 -> fe80::19e6:47a7:f579:3dfc, id=dd60d)
  Frag #1:                                      3333333333333333
  Frag #2: [MF]                                 22222222
  Frag #3: [MF] ppppppppp1111111111111111
    from 1164060793.7370 to +0.0010
  got frag timeout at: +39.7832

Test 17:  (fe80::020c:29ff:fecd:b316 -> fe80::19e6:47a7:f579:3dfc, id=dda11)
  Frag #1:              2222222222222222
  Frag #2: [MF]         33333333
  Frag #3: [MF] ppppppppp1111111111111111
    from 1164060813.7601 to +0.0019
  got frag timeout at: +40.7206

Test 19:  (fe80::020c:29ff:fecd:b316 -> fe80::19e6:47a7:f579:3dfc, id=ddc13)
  Frag #1: [MF]                                         5555555555555555
  Frag #2:                                                              6666666666666666
  Frag #3: [MF]                                 4444444444444444
  Frag #4: [MF]                                 ooooooooooooooo
  Frag #5: [MF]                                 33333333
  Frag #6: [MF]                                 777777777777777777777777777777777777777777
  Frag #7: [MF] ppppppppp1111111111111111
    from 1164060823.7667 to +0.0085
  got frag timeout at: +41.3263

Test 21:  (fe80::020c:29ff:fecd:b316 -> fe80::19e6:47a7:f579:3dfc, id=dde15)
  Frag #1: [MF]                 22222222222222222222222222222222222222222222
  Frag #2:                                                                    33333333
  Frag #3: [MF] ppppppppp1111111111111111
    from 1164060833.7801 to +0.0019
  got frag timeout at: +41.3841

Test 22:  (fe80::020c:29ff:fecd:b316 -> fe80::19e6:47a7:f579:3dfc, id=ddf16)
  Frag #1:                                                                    33333333
  Frag #2: [MF]                 22222222222222222222222222222222222222222222
  Frag #3: [MF] ppppppppp1111111111111111
    from 1164060838.7831 to +0.0018
  got frag timeout at: +39.9932

Test 23:  (fe80::020c:29ff:fecd:b316 -> fe80::19e6:47a7:f579:3dfc, id=de017)
  Frag #1: [MF]                 22222222222222222222222222222222222
  Frag #2:                                                          33333333
  Frag #3: [MF] ppppppppp1111111111111111
    from 1164060843.7862 to +0.0021
  got frag timeout at: +41.8177

Test 24:  (fe80::020c:29ff:fecd:b316 -> fe80::19e6:47a7:f579:3dfc, id=de118)
  Frag #1:                                      3333333333333333
  Frag #2: [MF]                 22222222222222222222222
  Frag #3: [MF] ppppppppp1111111111111111
    from 1164060848.7897 to +0.0029
  got frag timeout at: +40.9700

Test 25:  (fe80::020c:29ff:fecd:b316 -> fe80::19e6:47a7:f579:3dfc, id=de219)
  Frag #1:                                                  66666666666666666666666
  Frag #2: [MF]                                 4444444444444444
  Frag #3: [MF]                                 oooooooooooooooo
  Frag #4: [MF]                                 33333333
  Frag #5: [MF] ppppppppp1111111111111111
    from 1164060853.7957 to +0.0035
  got frag timeout at: +42.0044

Test 27:  (fe80::020c:29ff:fecd:b316 -> fe80::19e6:47a7:f579:3dfc, id=de41b)
  Frag #1:                                                  66666666666666666666666
  Frag #2: [MF]                                 4444444444444444
  Frag #3: [MF]                                 33333333
  Frag #4: [MF] ppppppppp1111111111111111
    from 1164060863.8048 to +0.0034
  got frag timeout at: +41.4683

Test 28:  (fe80::020c:29ff:fecd:b316 -> fe80::19e6:47a7:f579:3dfc, id=de51c)
  Frag #1:                                      3333333333333333
  Frag #2: [MF]                 2222222222222222
  Frag #3: [MF] ppppppppp1111111111111111
    from 1164060868.8087 to +0.0019
  got frag timeout at: +41.6738

Test 29:  (fe80::020c:29ff:fecd:b316 -> fe80::19e6:47a7:f579:3dfc, id=de61d)
  Frag #1:                                                  66666666666666666666666
  Frag #2: [MF]                                 4444444444444444
  Frag #3: [MF] ppppppppp1111111111111111
    from 1164060873.8115 to +0.0022
  got frag timeout at: +40.5193

Test 30:  (fe80::020c:29ff:fecd:b316 -> fe80::19e6:47a7:f579:3dfc, id=de71e)
  Frag #1:                                                  66666666666666666666666
  Frag #2: [MF]                                 44444444
  Frag #3: [MF] ppppppppp1111111111111111
    from 1164060878.8174 to +0.0035
  got frag timeout at: +41.7998

Test 31:  (fe80::020c:29ff:fecd:b316 -> fe80::19e6:47a7:f579:3dfc, id=de81f)
  Frag #1: [MF]         33333333
  Frag #2:              2222222222222222
  Frag #3: [MF] ppppppppp1111111111111111
    from 1164060883.8271 to +0.0022
  got frag timeout at: +39.6629

Test 32:  (fe80::020c:29ff:fecd:b316 -> fe80::19e6:47a7:f579:3dfc, id=de920)
  Frag #1:              2222222222222222
  Frag #2: [MF]                 33333333
  Frag #3: [MF] ppppppppp1111111111111111
    from 1164060888.8308 to +0.0030
  got frag timeout at: +41.2511

Test 33:  (fe80::020c:29ff:fecd:b316 -> fe80::19e6:47a7:f579:3dfc, id=dea21)
  Frag #1:                      33333333
  Frag #2: [MF]         2222222222222222
  Frag #3: [MF] ppppppppp1111111111111111
    from 1164060893.8338 to +0.0028
  got frag timeout at: +39.9632

Test 34:  (fe80::020c:29ff:fecd:b316 -> fe80::19e6:47a7:f579:3dfc, id=deb22)
  Frag #1: [MF]                                         4444444466666666
  Frag #2:                                                              6666666666666666
  Frag #3: [MF]                                 4444444444444444
  Frag #4: [MF]                         33333333
  Frag #5: [MF] ppppppppp1111111111111111
    from 1164060898.8379 to +0.0044
  got frag timeout at: +41.2675

Test 35:  (fe80::020c:29ff:fecd:b316 -> fe80::19e6:47a7:f579:3dfc, id=dec23)
  Frag #1: [MF]                                         4444444466666666
  Frag #2:                                                              6666666666666666
  Frag #3: [MF]                                 4444444444444444
  Frag #4: [MF]                                 44444444
  Frag #5: [MF]                         33333333
  Frag #6: [MF] ppppppppp1111111111111111
    from 1164060903.8431 to +0.0043
  got frag timeout at: +40.6217

Test 36:  (fe80::020c:29ff:fecd:b316 -> fe80::19e6:47a7:f579:3dfc, id=ded24)
  Frag #1: [MF]                                         5555555555555555
  Frag #2:                                                              6666666666666666
  Frag #3: [MF]                                 4444444444444444
  Frag #4: [MF]                         33333333
  Frag #5: [MF]                                 777777777777777777777777777777
  Frag #6: [MF] ppppppppp1111111111111111
    from 1164060908.8490 to +0.0063
  got frag timeout at: +41.0083

Test 37:  (fe80::020c:29ff:fecd:b316 -> fe80::19e6:47a7:f579:3dfc, id=dee25)
  Frag #1: [MF]                                         5555555555555555
  Frag #2:                                                              6666666666666666
  Frag #3: [MF]                                 4444444444444444
  Frag #4: [MF]                         33333333
  Frag #5: [MF]                                 77777777
  Frag #6: [MF] ppppppppp1111111111111111
    from 1164060913.8559 to +0.0037
  got frag timeout at: +41.1899

Test 38:  (fe80::020c:29ff:fecd:b316 -> fe80::19e6:47a7:f579:3dfc, id=def26)
  Frag #1:                                              5555555555555555
  Frag #2: [MF]                                                 66666666
  Frag #3: [MF]                                 4444444444444444
  Frag #4: [MF]                         33333333
  Frag #5: [MF]                                 77777777
  Frag #6: [MF] ppppppppp1111111111111111
    from 1164060918.8605 to +0.0052
  got frag timeout at: +40.0794

Test 39:  (fe80::020c:29ff:fecd:b316 -> fe80::19e6:47a7:f579:3dfc, id=df027)
  Frag #1: [MF]                 22222222
  Frag #2:                      3333333333333333
  Frag #3: [MF] ppppppppp1111111111111111
    from 1164060923.8664 to +0.0020
  got frag timeout at: +41.2485

Test 40:  (fe80::020c:29ff:fecd:b316 -> fe80::19e6:47a7:f579:3dfc, id=df128)
  Frag #1: [MF]                 22222222
  Frag #2:                      3333333333333333
  Frag #3: [MF] pppppppp11111111
    from 1164060928.8690 to +0.0024
  got frag timeout at: +39.3627

Test 43:  (fe80::020c:29ff:fecd:b316 -> fe80::19e6:47a7:f579:3dfc, id=df42b)
  Frag #1: [MF]         22222222
  Frag #2:              3333333333333333
  Frag #3: [MF] pppppppp
    from 1164060943.8800 to +0.0021
  got frag timeout at: +41.4088

Test 53:  (fe80::020c:29ff:fecd:b316 -> fe80::19e6:47a7:f579:3dfc, id=dfe35)
  Frag #1: [MF]                         2222222222222222
  Frag #2: [MF]                                 44444444
  Frag #3:                                                      3333333333333333
  Frag #4: [MF]                                                         55555555
  Frag #5: [MF] ppppppppp1111111111111111
    from 1164060993.9334 to +0.0023
  got frag timeout at: +39.9880

Test 60:  (fe80::020c:29ff:fecd:b316 -> fe80::19e6:47a7:f579:3dfc, id=e053c)
  Frag #1: [MF]                         2222222222222222
  Frag #2: [MF]                                 55555555
  Frag #3: [MF]                                 4444444444444444
  Frag #4: [MF] ppppppppp1111111111111111
  Frag #5:                                                      33333333
    from 1164061028.9783 to +0.0078
  got frag timeout at: +39.6508
```

Fig. 32. part 2.

```
Test 8:  (fe80::020c:29ff:fecd:b316 -> fe80::19e6:47a7:f579:3dfc, id=dd108)
  Frag #1:                    2222222222222222
  Frag #2: [MF] ppppppppp1111111111111111
    from 1164060768.7189 to +0.0016

Test 9:  (fe80::020c:29ff:fecd:b316 -> fe80::19e6:47a7:f579:3dfc, id=dd209)
  Frag #1:              2222222222222222
  Frag #2: [MF] ppppppppp1111111111111111
    from 1164060773.7225 to +0.0023

Test 10:  (fe80::020c:29ff:fecd:b316 -> fe80::19e6:47a7:f579:3dfc, id=dd30a)
  Frag #1:                    2222222222222222222222222222222222222222222222
  Frag #2: [MF] ppppppppp1111111111111111
    from 1164060778.7276 to +0.0015

Test 14:  (fe80::020c:29ff:fecd:b316 -> fe80::19e6:47a7:f579:3dfc, id=dd70e)
  Frag #1: [MF]         22222222
  Frag #2: [MF]         33333333
  Frag #3:      ppppppppp1111111111111111
    from 1164060798.7407 to +0.0074

Test 15:  (fe80::020c:29ff:fecd:b316 -> fe80::19e6:47a7:f579:3dfc, id=dd80f)
  Frag #1: [MF]         22222222
  Frag #2: [MF]         33333333
  Frag #3: [MF]         44444444
  Frag #4: [MF]         55555555
  Frag #5: [MF]         66666666
  Frag #6:      ppppppppp1111111111111111
    from 1164060803.7503 to +0.0050

Test 16:  (fe80::020c:29ff:fecd:b316 -> fe80::19e6:47a7:f579:3dfc, id=dd910)
  Frag #1:              2222222222222222
  Frag #2: [MF]         3333333333333333
  Frag #3: [MF] ppppppppp1111111111111111
    from 1164060808.7566 to +0.0025

Test 18:  (fe80::020c:29ff:fecd:b316 -> fe80::19e6:47a7:f579:3dfc, id=ddb12)
  Frag #1:                    22222222
  Frag #2: [MF]               33333333
  Frag #3: [MF] ppppppppp1111111111111111
    from 1164060818.7630 to +0.0022

Test 20:  (fe80::020c:29ff:fecd:b316 -> fe80::19e6:47a7:f579:3dfc, id=ddd14)
  Frag #1:                    2222222222222222222222222222222222222222222222
  Frag #2: [MF]               3333333333333333333333333333333333333333333333
  Frag #3: [MF] ppppppppp1111111111111111
    from 1164060828.7772 to +0.0023

Test 26:  (fe80::020c:29ff:fecd:b316 -> fe80::19e6:47a7:f579:3dfc, id=de31a)
  Frag #1:                                                  6666666666666666
  Frag #2: [MF]                           4444444444444444
  Frag #3: [MF]                           ooooooooooooooo
  Frag #4: [MF]               3333333333333333
  Frag #5: [MF] ppppppppp1111111111111111
    from 1164060858.7999 to +0.0040

Test 41:  (fe80::020c:29ff:fecd:b316 -> fe80::19e6:47a7:f579:3dfc, id=df229)
  Frag #1: [MF]         22222222
  Frag #2:              3333333333333333
  Frag #3: [MF] pppppppp11111111
    from 1164060933.8745 to +0.0030

Test 42:  (fe80::020c:29ff:fecd:b316 -> fe80::19e6:47a7:f579:3dfc, id=df32a)
  Frag #1:              2222222222222222
  Frag #2: [MF] pppppppp11111111
    from 1164060938.8779 to +0.0008

Test 44:  (fe80::020c:29ff:fecd:b316 -> fe80::19e6:47a7:f579:3dfc, id=df52c)
  Frag #1: [MF]               2222222222222222
  Frag #2:                                    3333333333333333
  Frag #3: [MF] ppppppppp1111111111111111
    from 1164060948.8834 to +0.0029

Test 45:  (fe80::020c:29ff:fecd:b316 -> fe80::19e6:47a7:f579:3dfc, id=df62d)
  Frag #1: [MF] ppppppppp1111111111111111
  Frag #2: [MF]               2222222222222222
  Frag #3:                                    3333333333333333
    from 1164060953.8880 to +0.0024

Test 51:  (fe80::020c:29ff:fecd:b316 -> fe80::19e6:47a7:f579:3dfc, id=dfc33)
  Frag #1: [MF] pppppppp111111111+1
  Frag #2: [MF]                           22222222222222222222222222222222
  Frag #3: [MF]                  333333333333333333333333333333333
  Frag #4:                                                                77777777
    from 1164060983.9241 to +0.0027

Test 52:  (fe80::020c:29ff:fecd:b316 -> fe80::19e6:47a7:f579:3dfc, id=dfd34)
  Frag #1: [MF] pppppppp111111111+1
  Frag #2: [MF]                                66666666
  Frag #3: [MF]                          4444444444444444444444
  Frag #4: [MF]                 22222222222222222222222222222222222222
  Frag #5:                                                                77777777
    from 1164060988.9278 to +0.0039

Test 56:  (fe80::020c:29ff:fecd:b316 -> fe80::19e6:47a7:f579:3dfc, id=e0138)
  Frag #1: [MF]                       2222222222222222
  Frag #2: [MF]                                44444444
  Frag #3: [MF] ppppppppp1111111111111111
  Frag #4:                                                 3333333333333333
  Frag #5: [MF]                                                     55555555
    from 1164061008.9557 to +0.0043

Test 57:  (fe80::020c:29ff:fecd:b316 -> fe80::19e6:47a7:f579:3dfc, id=e0239)
  Frag #1: [MF]                       2222222222222222
  Frag #2: [MF]                                44444444
  Frag #3:                                                 3333333333333333
  Frag #4: [MF] ppppppppp1111111111111111
  Frag #5: [MF]                                                     55555555
    from 1164061013.9607 to +0.0041

Test 63:  (fe80::020c:29ff:fecd:b316 -> fe80::19e6:47a7:f579:3dfc, id=e083f)
  Frag #1: [MF] ppppppppp1111111111111111
  Frag #2: [MF]                                2222222222222222
  Frag #3: [MF]                                        44444444
  Frag #4: [MF]                       55555555
  Frag #5:                                                          3333333333333333
    from 1164061044.0294 to +0.0138

Test 64:  (fe80::020c:29ff:fecd:b316 -> fe80::19e6:47a7:f579:3dfc, id=e0940)
  Frag #1: [MF] ppppppppp1111111111111111
  Frag #2: [MF]                                22222222
  Frag #3: [MF]                                        55555555
  Frag #4: [MF]                                4444444444444444
  Frag #5: [MF]                       77777777
  Frag #6:                                                          33333333
    from 1164061049.0461 to +0.0051
```

Fig. 33. The test cases where IPv6 fragment reassembly did not succeed and an ICMPv6 fragmentation timeout message was not produced. On some of the long packets, we shorten the lines by using "+" to represent six additional instances of the previous character.

D. Analysis

Vista's reassembly behavior is different that either Windows XP or Red Hat Linux, as demonstrated in Figure 34. Earlier Windows versions, such as Windows XP, used a different reassembly strategy. Overlapping fragments are allowed in some instances, and are dealt with by trimming the left-hand side (from low to high byte offset) from more recently received fragments.

We have developed a working model to predict when fragmentation will take place and when it will not. This model works on all our test cases. We define "leading fragments range" (LFR) as, the extent to which the packet can be completely reassembled, starting from offset 0 using previously received fragments. That is, as far as one can get in the packet without finding a gap. We observed the following:

- Vista follows a favor-old behavior
- If a received fragment's end falls within LFR (i.e., the packet fits inside LFR), it is discarded
- If a received fragment's entire range aligns exactly with a previously received fragment, it is discarded
- If a received fragment partially overlaps with any fragment that is not part of LFR, it becomes impossible to complete packet reassembly[10]; packet reassembly state may have been discarded.
- The received fragment (which does not overlap with previous fragments) becomes part of the reassembled packet, if reassembly succeeds

[10]In some side testing, we were unable to get it to complete assembly, at that point. We hypothesized about different discard scenarios and tried to send fragments to fill in the blanks.

	Linux Red Hat 8	1111111111111111333333333333333300000000555555556666666
Test 2:	Windows XP	not reassembled
	Windows Vista	not reassembled
	Linux Red Hat 8	11111111111111113333333300000000000000005555555566666666
Test 3:	Windows XP	11111111111111113333333344444444555555555555555566666666
	Windows Vista	not reassembled
	Linux Red Hat 8	11111111111111113333333300000000000000006666666666666666
Test 4:	Windows XP	11111111111111113333333344444444444444446666666666666666
	Windows Vista	11111111111111113333333344444444444444446666666666666666
	Linux Red Hat 8	11111111111111113333333344444444444444446666666666666666
Test 5:	Windows XP	11111111111111113333333344444444444444446666666666666666
	Windows Vista	not reassembled

Fig. 34. Selected comparative IPv4 reassembly results between Red Hat Linux, Windows XP, and Windows Vista

- If the packet is not completed after approximately 60 seconds (or perhaps 60 seconds without progress), it is discarded and an ICMP fragmentation timeout message may be sent (depending on the pattern of fragments)

It appears that the reassembly behavior does not occur by design; it may be a byproduct of the algorithms used. Incidental and unplanned behavior can easily change with even minor code updates (e.g., in response to bugs).

We have not looked at the code that handles the reassembly, but it appears that fragments for a packet are kept two data structures: a buffer corresponding to LFR, in which info about the original fragments is not maintained, and a list (probably sorted) of the fragments not in LFR. For fragments that do not overlap with previous fragments, they are either appended to the LFR buffer or added to the list. If they are appended to the LFR buffer, the list is then checked and any fragments that can be appended to the LFR buffer are moved from the list to the LFR buffer. The list does not get consolidated. This seems consistent with the documented use of NET_BUFFERs[42].

For most test cases where reassembly fails, IPv4 or IPv6 sends an ICMP fragmentation timeout message after around 60 seconds, but not in all test cases. Generally, the cases that do not produce an error message look “simpler”, but we have not found good rules to accurately predict the occurrence of either case.

APPENDIX IX
SOURCE ROUTING

"Source routing" occurs when the packet originator predefines a series of "hops" to take on the way to a destination. It is available in both IPv4 and IPv6. Source routing can be exploited by an attacker in various types of attacks, including bypassing access control. A best practice is to block it. The possible encounters of a node with a source-routed packet can be divided into two types[11]. An "en-route" encounter occurs when the packet is encountered on its way to the specified final destination; these packets will either be dropped or forwarded upon receipt. An "at-end" encounter occurs when the node is the final destination; these packets will either be accepted or discarded upon receipt.

According to [34] and [29], XP SP2 and later, and Windows Server 2003 SP1 and later, would neither forward en-route (IPv4) packets nor accept (at-end) packets that were source-routed. However, earlier versions would accept at-end packets. We examined what Vista would do for IPv4 and IPv6 in both cases.

First we used netsh to examine Vista's IPv6 source-routing settings.

```
netsh>interface ipv6 show global
Querying active state...

General Global Parameters
----------------------------------------
Default Hop Limit                   : 128 hops
Neighbor Cache Limit                : 256 entries per interface
Route Cache Limit                   : 128 entries per compartment
Reassembly Limit                    : 4185600 bytes
ICMP Redirects                      : enabled
Source Routing Behavior             : forward
Task Offload                        : enabled
Dhcp Media Sense                    : enabled
Media Sense Logging                 : enabled
MLD Level                           : all
MLD Version                         : version3
Multicast Forwarding                : disabled
Group Forwarded Fragments           : disabled
Randomize Identifiers               : enabled
Address Mask Reply                  : disabled

Current Global Statistics
----------------------------------------
Number of Compartments              : 1
Number of NL clients                : 6
Number of FL providers              : 4

netsh>interface
netsh interface>ipv6
netsh interface ipv6>show interfaces 8

Interface TestNet Parameters
----------------------------------------
IfLuid                              : ethernet_4
IfIndex                             : 8
Compartment Id                      : 1
State                               : connected
Metric                              : 10
Link MTU                            : 1500 bytes
Reachable Time                      : 29000 ms
Base Reachable Time                 : 30000 ms
Retransmission Interval             : 1000 ms
DAD Transmits                       : 1
Site Prefix Length                  : 64
Site Id                             : 1
Forwarding                          : disabled
Advertising                         : disabled
Neighbor Discovery                  : enabled
Neighbor Unreachability Detecion    : enabled
Router Discovery                    : enabled
Managed Address Configuration       : disabled
Other Stateful Configuration        : disabled
Weak Host Sends                     : disabled
Weak Host Receives                  : disabled
Use Automatic Metric                : enabled
Ignore Default routes               : disabled
```

[11] By subdividing "en-route" into "specified hop" versus "in-between hop", it is possible to have a third kind of encounter, which may be a useful division; however our test network does not support having an in-between hop.

In the global settings, Source Routing Behavior is set to “forward”, however the interface for the network had Forwarding set to “disabled”. Thus, it would seem that at least en-route packets would be dropped. We found the same to be the case for Teredo interfaces, as of Vista RC2 (Appendix XIII-M). To test behavior, we developed scripts.

First we sent an empty IPv6 packet that was destined for hpvista, but sent through acervista using type 0 source routing (LSRR). The following shows the packet as shown in Wireshark[61] (output trimmed):

```
Internet Protocol Version 6
    Version: 6
    Traffic class: 0x00
    Flowlabel: 0x00000
    Payload length: 24
    Next header: IPv6 routing (0x2b)
    Hop limit: 128
    Source address: fe80::20c:29ff:fecd:b316 (fe80::20c:29ff:fecd:b316)
    Destination address: fe80::ed59:b7ac:fc61:f865 (fe80::ed59:b7ac:fc61:f865)
Routing Header, Type 0
    Next header: IPv6 no next header (0x3b)
    Length: 2 (24 bytes)
    Type: 0
    Segments left: 1
    address 0: fe80::4d5:1fa6:3777:a480 (fe80::4d5:1fa6:3777:a480)
```

There was no reaction to this packet, thus acervista did not forward this en-route packet.

Next we constructed two at-end packets destined for acervista that appeared to have been forwarded by hpvista, after being sent by our linux analysis machine. One was an empty packet, which would not normally generate a response. This looks like:

```
Internet Protocol Version 6
    Version: 6
    Traffic class: 0x00
    Flowlabel: 0x00000
    Payload length: 24
    Next header: IPv6 routing (0x2b)
    Hop limit: 128
    Source address: fe80::20c:29ff:fecd:b316 (fe80::20c:29ff:fecd:b316)
    Destination address: fe80::ed59:b7ac:fc61:f865 (fe80::ed59:b7ac:fc61:f865)
Routing Header, Type 0
    Next header: IPv6 no next header (0x3b)
    Length: 2 (24 bytes)
    Type: 0
    Segments left: 0
    address 0: fe80::4d5:1fa6:3777:a480 (fe80::4d5:1fa6:3777:a480)
```

The other at-end was a ping packet, which looks like:

```
Internet Protocol Version 6
    Version: 6
    Traffic class: 0x00
    Flowlabel: 0x00000
    Payload length: 36
    Next header: IPv6 routing (0x2b)
    Hop limit: 128
    Source address: fe80::20c:29ff:fecd:b316 (fe80::20c:29ff:fecd:b316)
    Destination address: fe80::ed59:b7ac:fc61:f865 (fe80::ed59:b7ac:fc61:f865)
Routing Header, Type 0
    Next header: ICMPv6 (0x3a)
    Length: 2 (24 bytes)
    Type: 0
    Segments left: 0
    address 0: fe80::4d5:1fa6:3777:a480 (fe80::4d5:1fa6:3777:a480)
Internet Control Message Protocol v6
    Type: 128 (Echo request)
    Code: 0
    Checksum: 0xd231 [correct]
    ID: 0xd82c
    Sequence: 0xc209
    Data (4 bytes)
```

This ping produced an echo reply (we configured a firewall exception on the target to respond to IPv6 pings):

```
Internet Protocol Version 6
    Version: 6
    Traffic class: 0x00
    Flowlabel: 0x00000
    Payload length: 12
    Next header: ICMPv6 (0x3a)
    Hop limit: 128
    Source address: fe80::ed59:b7ac:fc61:f865 (fe80::ed59:b7ac:fc61:f865)
    Destination address: fe80::20c:29ff:fecd:b316 (fe80::20c:29ff:fecd:b316)
Internet Control Message Protocol v6
    Type: 129 (Echo reply)
    Code: 0
    Checksum: 0xd131 [correct]
    ID: 0xd82c
    Sequence: 0xc209
    Data (4 bytes)
```

Note that this does not include a reverse source route, which is not something RFC 2460 requires. Hence for IPv6, Vista by default drops en-route source routing, but will accept at-end source-routed packets.

Next, we examined IPv4 source routing settings in netsh.

```
netsh interface ipv4>show global
Querying active state...

General Global Parameters
----------------------------------------------
Default Hop Limit                   : 128 hops
Neighbor Cache Limit                : 256 entries per interface
Route Cache Limit                   : 128 entries per compartment
Reassembly Limit                    : 4185600 bytes
ICMP Redirects                      : enabled
Source Routing Behavior             : dontforward
Task Offload                        : enabled
Dhcp Media Sense                    : enabled
Media Sense Logging                 : enabled
MLD Level                           : all
MLD Version                         : version3
Multicast Forwarding                : disabled
Group Forwarded Fragments           : disabled
Randomize Identifiers               : enabled
Address Mask Reply                  : disabled

Current Global Statistics
-----------------------------------------
Number of Compartments              : 1
Number of NL clients                : 7
Number of FL providers              : 4

netsh interface ipv4>show interfaces 8

Interface TestNet Parameters
-----------------------------------------
IfLuid                             : ethernet_4
IfIndex                            : 8
Compartment Id                     : 1
State                              : connected
Metric                             : 10
Link MTU                           : 1500 bytes
Reachable Time                     : 39500 ms
Base Reachable Time                : 30000 ms
Retransmission Interval            : 1000 ms
DAD Transmits                      : 3
Site Prefix Length                 : 64
Site Id                            : 1
Forwarding                         : disabled
Advertising                        : disabled
Neighbor Discovery                 : enabled
Neighbor Unreachability Detecion   : enabled
Router Discovery                   : dhcp
Managed Address Configuration      : enabled
Other Stateful Configuration       : enabled
Weak Host Sends                    : disabled
Weak Host Receives                 : disabled
Use Automatic Metric               : enabled
Ignore Default routes              : disabled
```

The global Source Routing Behavior is set to “dontforward” which suggests that it will not pass along en-route IPv4. In addition, Forwarding on the interface is set to “disabled”.

To test this on Vista hosts, we sent an IPv4 ping to hpvista via acervista:

```
Internet Protocol, Src: 192.168.0.102 (192.168.0.102), Dst: 192.168.0.200 (192.168.0.200)
    Version: 4
    Header length: 32 bytes
    Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00)
    Total Length: 44
    Identification: 0x06ba (1722)
    Flags: 0x00
    Fragment offset: 0
    Time to live: 128
    Protocol: ICMP (0x01)
    Header checksum: 0x4509 [correct]
    Source: 192.168.0.102 (192.168.0.102)
    Destination: 192.168.0.200 (192.168.0.200)
    Options: (12 bytes)
        Loose source route (11 bytes)
            Pointer: 4
            192.168.0.200 <- (current)
            192.168.0.201
        NOP
Internet Control Message Protocol
    Type: 8 (Echo (ping) request)
    Code: 0
    Checksum: 0xbf38 [correct]
    Identifier: 0xd82c
    Sequence number: 0xc209
    Data (4 bytes)
```

As in the IPv6 case, acervista did not forward and there were no error messages.

We then sent an IPv4 ping to vmvista that appeared to be sent by our Linux analysis machine and source routed through hpvista.

```
Internet Protocol, Src: 192.168.0.102 (192.168.0.102), Dst: 192.168.0.203 (192.168.0.203)
    Version: 4
    Header length: 32 bytes
    Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00)
    Total Length: 44
    Identification: 0x06ba (1722)
    Flags: 0x00
    Fragment offset: 0
    Time to live: 128
    Protocol: ICMP (0x01)
    Header checksum: 0x3f06 [correct]
    Source: 192.168.0.102 (192.168.0.102)
    Destination: 192.168.0.203 (192.168.0.203)
    Options: (12 bytes)
        Loose source route (11 bytes)
            Pointer: 8
            192.168.0.200
            192.168.0.203 <- (current)
        NOP
Internet Control Message Protocol
    Type: 8 (Echo (ping) request)
    Code: 0
    Checksum: 0xbf38 [correct]
    Identifier: 0xd82c
    Sequence number: 0xc209
    Data (4 bytes)
```

Although vmvista had file sharing enabled, which would normally allow responses to IPv4 pings, it did not produce one in this case. So, it seems Vista actively discards at-end IPv4 source routed packets. Thus, for IPv4 Vista disallows en-route and at-end source routing, which is a continuation of the Windows XP SP2/Windows Server 2003 SP1 behavior.

In conclusion, the only source routing that Vista allows is IPv6 packets that are source-routed.

APPENDIX X
IPV4 PROTOCOL ENUMERATION

We attempted to enumerate the protocols that Vista supports on top of IPv4. With the firewall turned on, as in Windows XP, we received no responses to our probes, so we turned the firewall off in order to do this enumeration. With the firewall turned off, an ICMP protocol unreachable message was sent in response to protocols that were unavailable.

We constructed a tool, proto.py, which enumerates protocols using this method. We took special measures (waiting long enough) to ensure that we did not hit Vista's rate limiting of ICMP errors (section III-D). When run against a Vista machine (firewall turned off) the results were:

```
  linux# ./proto.py $acerIP4
No protocol unreachables for:
    1 icmp ICMP
    2 igmp IGMP
    4 ipencap IP-ENCAP                (IPv4 over IPv4)
    6 tcp TCP
    17 udp UDP
    41 ipv6 IPv6                      (IPv6 over IPv4)
    43 ipv6-route IPv6-Route          (IPv6 Routing extension header)
    44 ipv6-frag IPv6-Frag            (IPv6 Fragment extension header)
    47 gre GRE
    50 esp IPSEC-ESP                  (IPSec ESP)
    51 ah IPSEC-AH                    (IPSec AH)
```

We observed the same results on 32- and 64-bit Vista. The results on an nmap -sO run were also consistent with this. Our inferred meaning of the protocol numbers is based on IANA assignment[24].

The apparent support for protocols 43 and 44 is surprising; in IPv6, these denote IPv6 extension headers, but they have no meaning in IPv4. It would also seem that IPv4 over IPv4 is supported. Testing whether we could use these in any way was out of scope for this project.

APPENDIX XI
IPV6 NEXT HEADER ENUMERATION

We enumerated the supported IPv6 Next Header values on release Vista. The IPv6 Next Header field is similar to the IPv4 protocol field, except that it also encodes extension headers; the same namespace is used for both. We were able to do this enumeration even with the firewall on, since unserviced values yielded ICMPv6 parameter problems for unrecognized Next Header (type 4, code 1) that points to the Next Header field in the IPv6 base header. This mechanism could be used to essentially ping a Vista machine, even if filtering is enabled.

We wrote another tool, proto6.py, to do this test. We had to carefully avoid ICMPv6 error rate limiting. The results were the following:

```
linux# ./proto6.py $acerLL6%2 $acerMAC
No protocol unreachables for:
    0 ip IP                         (IPv6 Hop-by-Hop options extension header)
    4 ipencap IP-ENCAP              (IPv4 over IPv6)
    6 tcp TCP
    17 udp UDP
    41 ipv6 IPv6                    (IPv6 over IPv6)
    43 ipv6-route IPv6-Route        (IPv6 Routing extension header)
    44 ipv6-frag IPv6-Frag          (IPv6 Fragment extension header)
    50 esp IPSEC-ESP                (IPSec ESP extension header)
    51 ah IPSEC-AH                  (IPSec AH extension header)
    58 ipv6-icmp IPv6-ICMP          (ICMPv6)
    59 ipv6-nonxt IPv6-NoNxt        (no next header)
    60 ipv6-opts IPv6-Opts          (IPv6 destination option extension header)
```

From the results, it appears that Vista supports IPv4 over IPv6, which is how IPv4 packets can be carried across an IPv6-only network. These are all standard values[24].

We performed the same test with the firewall off. The same type of errors result for unknown protocols. Unexpectedly, the results of the enumeration were slightly different than with the firewall on. With the firewall off, we saw a parameter problem for Next Header=4, which we were unable to see with the firewall on. Since the firewall was off, this result is not due to dynamic filtering. It appears that IPv4 over IPv6 is supported only when the firewall is on. This could be an intentional policy decision, similar to the policy decision that disallows running of Teredo if there is no IPv6-capable firewall registered[40].

APPENDIX XII
TEREDO INTRODUCTION

As described by Microsoft[36]:

> "Teredo is an IPv6 transition technology that provides address assignment and host-to-host automatic tunneling for unicast IPv6 traffic when IPv6/IPv4 hosts are located behind one or multiple IPv4 network address translators (NATs). To traverse IPv4 NATs, IPv6 packets are sent as IPv4-based User Datagram Protocol (UDP) messages."

Microsoft implements Teredo in Microsoft Windows Vista in order to provide an interim solution to a lack of global IPv6 access for clients. This solution works even if their networks do not allow native IPv6 support. Teredo is available under Windows XP and Windows 2003, but is enabled by default for the first time on Vista[36]. A number of open source implementations of Teredo exist, including a Unix version called Miredo[14].

In [22] (*The Teredo Protocol: Tunneling Past Network Security and Other Security Implications*, November 2006, by Jim Hoagland), Symantec identified a number of security implications associated with the use of Teredo. Some of those are implementation specific and we test some of those for Vista in this paper (Appendix XIII), along with some other security-related analysis of Vista's Teredo implementation.

A. Protocol Overview

Teredo is a mostly well-documented protocol, so we provide only a summary in this report. [22] and [36] provide a good description of Teredo and how it works. RFC 4380 ("Teredo: Tunneling IPv6 over UDP through Network Address Translations (NATs)"[23], February 2006, by Christian Huitema of Microsoft) is the authoritative reference for the protocol.

The need for Teredo arises out of the limitation associated with IPv4 NAT devices (which are widely deployed), namely, that IPv4 NAT devices do not support either native IPv6 packets or IPv6 tunneled directly over IPv4 (ISATAP/6to4). Thus Teredo was developed as an IPv6 provider of last resort[23]. To avoid the NAT problem, Teredo creates UDP tunnel(s) through the NAT, so the NAT has no specific awareness of IPv6 or Teredo; in fact no local network devices need be aware of these protocols.

The Teredo framework consists of three types of components. A *Teredo client* is the node that wants to use Teredo to reach a *peer* on the IPv6 Internet. For example, a node may want to reach an IPv6-only server. Clients are dual-stack (IPv4 and IPv6) nodes that are "trapped" behind one or more IPv4 NATs. Teredo clients always send and receive Teredo IPv6 traffic tunneled in UDP over IPv4 (see Figure 35). The Teredo component on a client prepends the tunnel headers on IPv6 packets that are sent out by an application (encapsulation), and removes the tunnel headers from application-bound incoming traffic (decapsulation), thereby abstracting away the IPv6 connectivity method from the application.

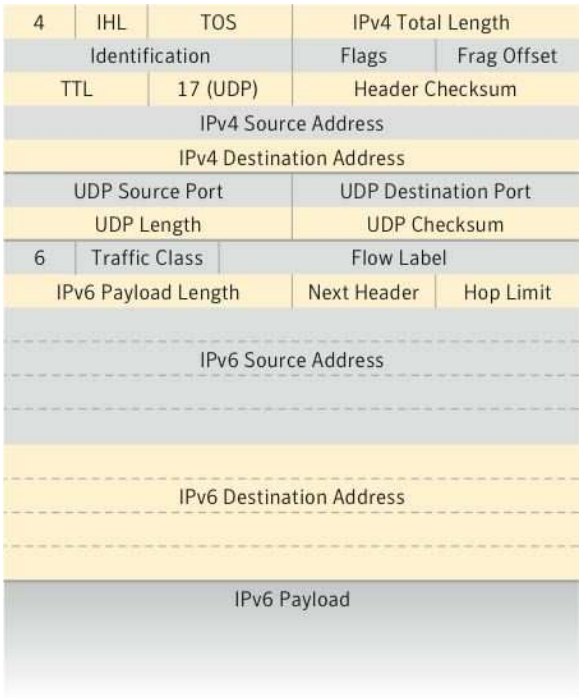

Fig. 35. Teredo encapsulates IPv6 packets in UDP over IPv4 when being routed as IPv4. In certain Teredo packets, one or two chunks of data are inserted between the UDP header and the IPv6 header.

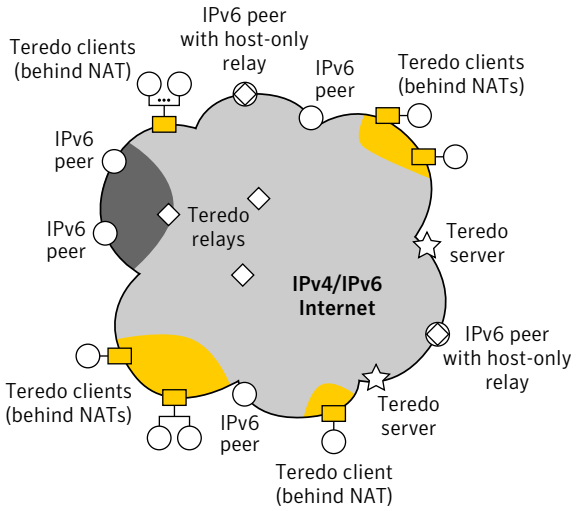

Fig. 36. A Teredo microcosm, including key Teredo components, native IPv6 nodes, and IPv4 NATs. The cloud represents the Internet, where the yellow areas are IPv4 only, the dark gray area is IPv6 only, and the mixed gray area supports both. The interior of the cloud represents Internet routers and infrastructure.

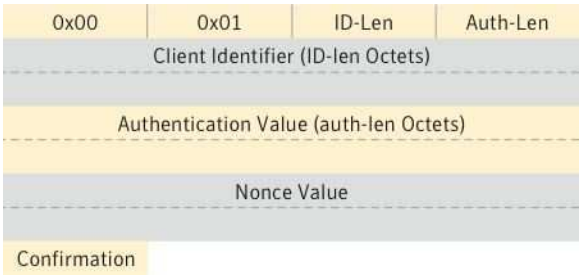

Fig. 37. The general format of the authentication data. In secure qualification, this data is positioned after the UDP header.

Teredo relays serve as a router to bridge the IPv4 and IPv6 Internets for Teredo nodes. IPv6 native packets are encapsulated for transmission over the IPv4 Internet (including the client), and when packets are received from the IPv4 Internet, they are decapsulated into native IPv6 packets for the IPv6 Internet. Thus, the peers need not know that the node they are communicating with is using Teredo. A special case is a host-only relay, which serves as a relay for the local host only. Connections between a client and a peer use the closest relay to the peer.

Teredo servers help a client set up its tunnel to IPv6 nodes. They help clients determine their Teredo address and determine if its NAT is compatible with Teredo. Like relays, Teredo servers sit on both the IPv4 and IPv6 Internets, but they do not serve as a general relay. The Teredo servers do pass along packets to and from the client, but only messages that pertain to the functioning of the Teredo protocol; they do not pass along data packets. The locations of Teredo servers are generally statically configured on the client.

Examples of these components and where they can be situated are shown in Figure 36.

The standard port on which the Teredo servers listen is UDP port 3544. Both clients and relays can use any UDP port for their Teredo service, so their UDP service port could be ephemeral. Due to the IPv4 NAT(s) it is behind, the external port number of a client's Teredo service is, in general, not the same as the local port that is listened on. However, the Teredo protocol tries to keep that external port number stable since this is the port to which the relays need to connect.

During its qualification phase, a Teredo client, with the help of its server, configures a specially formatted, globally addressable IPv6 address called a Teredo address. The IANA assigned prefix for Teredo addresses is 2001:0000::/32, and the rest of the address contains enough information for a relay to reach a client.

Teredo has a provision for "secure qualification". This adds authentication data (referred to in the RFC as authentication encapsulation) to the Teredo encapsulated packet. Without this, the client will not know it is getting a response back from the actual server (instead of, for example, someone randomly sending RAs to the client). The authentication data takes the format shown in Figure 37. The client identifier and authentication value are optional and have their specific length indicated in one octet fields. The nonce value is always present and is always eight octets in length. It is a random number chosen by the client and repeated by the server in the response. This simple measure establishes (with high probability) that if there is any attacker, they are on-path between the client and the server.

A ping test[12] is used in Teredo. In the procedure, the Teredo client sends an ICMPv6 echo request (ping) to a remote native IPv6 peer via the client's server. The server passes the ping directly over the IPv6 Internet to the peer. The peer then replies (assuming that it normally responds to pings). The response is returned to the client via the relay. In creating the ping, the Teredo client sets the ping payload to a large random number, which the RFC suggests should be at least 64 bits in length. That value is checked in the reply as an assurance against spoofing. To spoof a reply, an individual would either need to guess the random number used or be on-path. In addition, the relay that is used is remembered, and subsequent communication from the IPv6 peer is expected to come through that relay, and that relay is used for sending packets to the IPv6 peer.

B. Teredo Security Implications

In [22] we identified that our largest security concern with Teredo was that network-located security controls (including firewalls and IPSs) are bypassed by the Teredo tunnel. Unless they are specifically Teredo-aware, these controls (even if they support IPv6) are not properly applied to the IPv6 content that is located inside the UDP packet. This means that certain restrictions are not applied to the Teredo traffic (those restrictions that do not have an analogue on the host) or, at least, defense in depth is reduced. As Teredo clients are directly addressable from the Internet, organizations may find themselves unexpectedly exposed to the Internet.

Teredo has additional security implications including the following (from [22]):

- The difficulty of finding all Teredo traffic to inspect, due to the lack of fixed client and relay ports.
- Teredo packets are forwarded due to source routing to internal or external hosts after being decapsulated by the client.
- Teredo has provisions for arbitrary IPv4 nodes to poke a hole in a Teredo client's NAT through which they can then send unsolicited traffic to the client. This means that a restricted NAT is essentially turned into an unrestricted one, for each port maintained by a Teredo client.
- Teredo advertises (in the Teredo address) an open port in the client's NAT, and whether the NAT is more or less restrictive.
- Worms benefit from increasing host reachability, and, with certain types of host vulnerabilities, could possibly spread to end hosts with single UDP packets.
- It may be easy to deny Teredo service at either the client or the relay.
- The address space to scan for Teredo IPv6 addresses is much smaller than native IPv6 addresses.
- Teredo supports IPsec, and has some anti-spoofing measures automatically applied.
- The RFC requires that Teredo components make sanity checks on packets, which prevents many potential attacks.

[12]The ping test is referred to by the RFC as the "Direct IPv6 Connectivity Test". That name does not have a basis in the functionality if provides, so we call it the "ping test".

APPENDIX XIII
TEREDO ANALYSIS AND FINDINGS

We conducted a short-scoped project to investigate the implementation of Teredo on Vista, especially security-related items. We document our findings here. This research was conducted using the test network described in Appendix I-C, and was conducted prior to the availability of the release build of Vista. These results were most recently updated with the Release Candidate 2 (RC2) version of Vista (build 5744, October 2006).

The open source Teredo implementation Miredo[14] was used extensively during the project for different purposes, including:

- Reviewing Miredo source code to identify the types of potential implementation flaws that may be present in other Teredo implementations.
- Using Miredo as the base engine to a Symantec Teredo fuzzer.
- Using Miredo in order to better understand the functionality that exists within a Teredo client protocol stack.

We are grateful to Rémi Denis-Courmont for making this available.

A. Teredo Use Under Vista

The circumstances under which a Vista application will use Teredo when connecting to a peer is complicated and not entirely clear. The factors involved include at least:

- Whether the host has native IPv6 access, ISATAP/6to4 access, or neither.
- Whether a Teredo relay is available between the host and the peer or not.
- Whether the peer supports IPv6, IPv4, or both. For access by hostname, this corresponds to whether AAAA (IPv6) or A (IPv4) DNS records are available, or both.
- Whether the peer address is a Teredo address or not.
- Whether the local network supports Teredo (i.e., does it block any important Teredo traffic) or not.
- Which APIs a particular application uses.
- Whether the application is prepared to use IPv4 or IPv6, or both.
- For application that can use both IPv4 and IPv6, whether it has any bias for or against IPv6, or for or against Teredo.

Of particular interest is whether IPv6 using Teredo is preferred to native IPv4, since we know native IPv4 is available if Teredo is being used.

In addition to the circumstances being complicated, the Microsoft documentation on MSDN that should cover this[40] does not fully describe the possibilities and is unclear in parts. One case in which it is unclear is the “Receiving Solicited Traffic Over Teredo” section of [40]. At the time of writing, it says both:

> “Teredo is not utilized if the supplied hostname resolves to IPv4 addresses only. However, if an application calls the WSAConnectByName API and both IPv6 and IPv4 addresses are returned, the Windows Vista stack will resolve the IPv6 address first, allowing the use of the Teredo interface.”

and:

> “Due to current bsence [sic] of Teredo relays on the Internet, connections to native IPv6 addresses are unlikely to succeed over the Teredo interface. If WSAConnectByName is called, Windows Vista will not issue AAAA queries when Teredo is the only IPv6 capable interface available. This ensures that native IPv6 addresses are not obtained as a destination and that connections are attempted over IPv4, which has the highest chance of success. In order to obtain IPv6 addresses when Teredo is the only IPv6 capable interface, an application must explicitly use the DnsQuery API for AAAA records.”

We encourage Microsoft to improve its documentation, since that is in the interest of its users. A flow chart, or an interactive application that answers whether Teredo would be used in certain specified circumstances would be quite helpful.

Our testing was mostly based on using ping. From this, we learned two circumstances under which our NAT-trapped host would use Teredo. One was if “ping -6” was used and the remote peer had both A and AAAA DNS records. The the other was when “ping” was used without the “-6” and the peer had only AAAA records[13]

An important question for security is how often a Teredo address will be qualified (set up and available) in Vista. This is related to the previous question — if an application is using Teredo, a Teredo address will be qualified. Thus the complication and uncertainty associated with that is inherited by this question. Microsoft’s documentation[40] suggests that a Teredo interface will de-qualify itself if it has been inactive for one hour, unless there is an application listening for unsolicited traffic on the interface.

The question as to what fraction of Vista hosts will be using Teredo at any given time is one that may only be answered by history. It will also likely vary over time, as the percent of IPv6 capable servers increases and the percent of NAT trapped clients decreases. The safest assumption that network owners can make is that Teredo will often be used and hence they should plan security with that in mind.

[13]The “-6” option to ping.exe forces it to use IPv6. There is no specific documentation for the IP version choice in the absence of “-4” or “-6”, though the program is clearly capable of using IPv4 or IPv6 in that case.

According to Microsoft's "Teredo Overview" web page[36] (at the time of writing), "In Windows Vista, the Teredo component is enabled but inactive by default. In order to become active, a user must either install an application that needs to use Teredo, or configure advanced Windows Firewall filter settings to allow edge traversal.". The requirements for activation listed in that last sentence do not seem consistent with [40] or our experience with the main test network as described in Appendix XXVIII.

B. Vista Teredo Components

The implementation of Teredo in Vista is broken down into a number of different operating system components, which are detailed in Figure 38.

The role of IPHLPSVC.DLL was verified by listing which process owned the inbound UDP port used to handle incoming Teredo traffic for a connection:

```
[svchost.exe]
  UDP    192.168.0.13:60819     *:*                                    1004
```

As shown above, SVCHOST.EXE, which is responsible for the processing of Teredo traffic, is where IPHLPSVC.DLL exists. This was validated by using the PID to verify that the instance of SVCHOST.EXE had IPHLPSVC.DLL present in its address space.

C. Default Teredo settings

The command used to the configure Teredo under Windows Vista is netsh.exe. This can be used to show the current configuration and to reconfigure the Teredo interface on the host in question. We used netsh to list the default settings:

```
C:\Users\ollie>netsh interface ipv6 show teredo
Teredo Parameters
---------------------------------------------
Type                  : default
Server Name           : teredo.ipv6.microsoft.com.
Client Refresh Interval : 30 seconds
Client Port           : unspecified
```

The client port "unspecified" means that the Teredo client's service port is chosen ephemerally (see Appendix XIII-I). Note that there is no setting for server port; they are always supposed to run on UDP port 3544.

The default server is "teredo.ipv6.microsoft.com". As of early March 2007, this resolves to at least nine IPv4 addresses:

- 65.54.227.120
- 65.54.227.122
- 65.54.227.124
- 65.54.227.126
- 65.54.227.136
- 65.54.227.138
- 65.54.227.140
- 65.54.227.142
- 65.54.227.144

D. Requirements for Elevated Privileges

One interesting observation made during the course of our research was that cmd.exe must be run in elevated mode in order to gain complete information from the host in question. Figures 39 and 40 show the output of the same netsh.exe command only seconds apart: the former shows output from a non-elevated copy of cmd.exe and the latter shows output from an elevated copy. The information contained in Figure 40 is the correct configuration information.

File	Role
TUNMP.SYS	A kernel driver used to create the virtual network interface that Teredo uses. (It is also used for other tunnel interfaces on Microsoft Windows Vista). This component is /GS compiled[31].
TUNNEL.SYS	A kernel driver used to create the virtual network interface that Teredo uses. (It is also used for other tunnel interfaces on Microsoft Windows Vista). This component is /GS compiled.
IPHLPSVC.DLL	A system service which is run by SVCHOST and is the core Teredo tunneling component. This is responsible for tunnel set-up, configuration and transmission between the Teredo client and the Teredo server/relay. This component is /GS compiled.

Fig. 38. Teredo Windows Vista components and their roles.

```
C:\Users\ollie>time
The current time is:  8:16:25.54
Enter the new time:

C:\Users\ollie>netsh interface ipv6 show teredo
Teredo Parameters
---------------------------------------------
Type                    : default
Server Name             : teredo.ipv6.microsoft.com.
Client Refresh Interval : 30 seconds
Client Port             : unspecified
```

Fig. 39. Incorrect Teredo settings in netsh from a non-elevated shell. While these look valid (and in fact match the default settings), they were not in fact accurate for the host being examined. This actual settings at the time this was run are shown in Figure 40.

```
C:\Windows\system32>time
The current time is:  8:16:30.20
Enter the new time:

C:\Windows\system32>netsh interface ipv6 show teredo
Teredo Parameters
---------------------------------------------
Type                    : default
Server Name             : teredo.remlab.net
Client Refresh Interval : 30 seconds
Client Port             : unspecified
State                   : qualified
Client Type             : teredo client
Network                 : managed
NAT                     : restricted
```

Fig. 40. Teredo settings in netsh from an elevated shell.

This finding is interesting for a number of reasons. The finding shows firstly that there is a requirement to run an elevated command shell in order to be able to gain accurate diagnostics information, and secondly, that the first instance shows an incorrect Teredo server name. The result shows that it is possible for a malicious user or program to update the Teredo server settings with a lower likelihood of detection due to the mismatch in available information; however, they would need to make the change as administrator.

E. Disabling Teredo within Vista

Teredo can be disabled on Microsoft Windows Vista in a number of ways:

- Stopping the IPHLPSVC service
- Unbinding IPv6 from the network interface
- Configuring the IPv6 stack parameters

The preferred method for disabling Teredo under Microsoft Windows Vista is the third option outlined above[11]: configuring the IPv6 stack parameters. Under the registry key "HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\-tcpip6\Parameters", create an entry of type DWORD called DisabledComponents, then use this table to disable the appropriate protocols:

Configuration Combination	**DisabledComponents Value**
Disable all tunnel interfaces	0x01
Disable 6to4	0x02
Disable ISATAP	0x04
Disable Teredo	0x08
Disable Teredo and 6to4	0x0A
Disable all LAN and PPP interfaces	0x10
Disable all LAN, PPP, and tunnel interfaces	0x11
Prefer IPv4 over IPv6	0x20
Disable IPv6 over all interfaces and prefer IPv4 to IPv6	0xFF

F. Disabling the Microsoft Windows Firewall Disables Teredo

During the course of the research, we wished to disable Windows Firewall in order to understand the network filtering on a Microsoft Windows Vista host via a Teredo interface. However, it was discovered that by disabling Windows Firewall, the following entry is added to the trace information:

Fig. 41. Example where TracePrintFExW is imported. This is IPHLPSVC.DLL from Microsoft Windows Vista.

```
[1636] 11:33:19: Skipping start as firewall is disabled (0). Handle (0x01263F70)
```

The result of this action is that Microsoft Windows Vista simply reports that it is either unable to locate the host in question or that a general failure has occurred. This result occurs in response to all requests that interact with IPv6 via Teredo using either hostnames or IP addresses. The benefits of this result are obvious: no non-firewalled Microsoft Windows Vista hosts should be available within the Teredo address space. Additionally, this functionality removes any possible race condition in the duration between the Teredo interface opening and a firewall starting at system startup.

This behavior is consistent with the documentation in the “Implementing the Teredo Security Model” section of[40], which indicates “An IPv6-capable host firewall must be registered with Windows Security Center (WSC) on the machine. In the absence of a host-based firewall, or WSC itself, the Teredo interface will not be available for use. This is the only requirement to receive solicited traffic from the Internet over the Teredo interface.”

G. Settings Storage

Microsoft Windows Vista stores all Teredo-related settings, such as the Teredo server used, within the registry. The registry key used to store this information is HKLM\SYSTEM\CurrentControlSet\Services\iphlpsvc\Teredo. These are the configuration options to IPHLPSVC.DLL, whose functionality was described in a previous section.

H. Tracing Code

Microsoft left tracing code in IPHLPSVC.DLL in all examined Vista builds. This code, which can be enabled with a standard API, when combined with the available debugging symbols from Microsoft, provides an invaluable source of information about the implementation of Teredo on Windows Vista (see Appendix XIV-A). The reason Microsoft kept detailed logging available is unknown.

This tracing API is part of RRAS (RTUTILS.DLL). Availability for a given component can be determined by discovering whether it imports TracePrintfExW or similar. If it is available, this can be useful in understanding the states and functionality of the component in question (i.e., Figure 41). That means the application is using the Microsoft tracing API. To turn on tracing functionality simply use a registry editor to go to HKLM\Software\Microsoft\Tracing\[ApplicationName]\ and change the DWORD “EnableFileTracing” to 1. This results in the creation of a file under C:\Windows\Tracing with the same name as the component. This file is an ASCII log file containing the output of all the TracePrintExW statements. For more details please refer to [37].

I. Client Service Port Selection

The way that Microsoft Windows Vista automatically selects a client service port to use was analyzed. This analysis was of the default Teredo configuration, in which the client port is set to “unspecified”. This port is the local UDP port for all Teredo packets. This port, as remapped by the NAT, becomes part of the client’s Teredo address; hence packets from relays and the client’s server arrive on this port.

To accomplish this, a script was developed to force the Teredo client to establish a new connection by changing the Teredo server in use (Figure 42). The script then caused network traffic to be generated; this allowed us to observe the UDP destination port used on inbound traffic.

We determined that the client port was chosen in standard ephemeral manner. From Appendix XVII, we show the Vista ephemeral port range as 49152–65535. That matched our observations: in 1000 connections, the port range was 49596 to 65152.

J. Secure Qualification

Teredo on Microsoft Windows Vista supports only the most basic use of secure qualification. This is demonstrated in figures 43 and 44. Figure 43 shows the authentication data in the Router Solicitation Packet, whereas Figure 44 shows the authentication information in the corresponding Router Advertising Packet. Since client identification length and authentication

```
Dim iCount

Do While iCount < 1000
    rem Wscript.StdOut.write("Setting to RemLabs")
    DoItRem
    Wscript.Sleep(5000)
    rem Wscript.stdout.write "Setting to MS"
    DoItMS
    Wscript.Sleep(5000)
    rem Wscript.stdout.write "Looping..."
    iCount = iCount +1
Loop

Sub DoItRem()
    Dim wshShell

    set WshShell=WScript.CreateObject("Wscript.Shell")

    WshShell.run "netsh interface ipv6 set teredo servername = teredo.remlab.net",1,true

    Wscript.Sleep(20000)

    WshShell.run "ping -6 www.aaisp.co.uk",1,true
    Wscript.Sleep(20000)

    WshShell.run "ping -6 www.aaisp.co.uk",1,true
end Sub

Sub DoItMS()
    Dim wshShell

    set WshShell=WScript.CreateObject("Wscript.Shell")

    WshShell.run "netsh interface ipv6 set teredo servername = teredo.ipv6.microsoft.com",1,true

    Wscript.Sleep(20000)

    WshShell.run "ping -6 www.aaisp.co.uk",1,true
    Wscript.Sleep(20000)

    WshShell.run "ping -6 www.aaisp.co.uk",1,true
end Sub
```

Fig. 42. The UDP port enumeration script. This Windows Script Host (VBS) script was used to force the Teredo client to change its UDP port for firewall purposes. This allowed us to analyze the resulting UDP client port usage. The traffic generated was then captured with Wireshark and the IPv4 UDP destination port on which the ICMP echo replies were seen was analyzed.

```
⊞ Frame 219 (103 bytes on wire, 103 bytes captured)
⊞ Ethernet II, Src: 00:0c:29:82:31:3e (00:0c:29:82:31:3e), Dst: 00:14:6c:a9:bc:a0 (00:14:6c:a9:bc:a0)
⊞ Internet Protocol, Src: 192.168.0.13 (192.168.0.13), Dst: 83.170.6.76 (83.170.6.76)
⊞ User Datagram Protocol, Src Port: 53596 (53596), Dst Port: 3544 (3544)
⊟ Teredo IPv6 over UDP tunneling
  ⊟ Teredo Authentication header
      Client identifier length: 0
      Authentication value length: 0
      Nonce value: 409AF2508AA5AF07
      Confirmation byte: 00
⊟ Internet Protocol Version 6
    Version: 6
    Traffic class: 0x00
    Flowlabel: 0x00000
    Payload length: 8
    Next header: ICMPv6 (0x3a)
    Hop limit: 255
    Source address: fe80::ffff:ffff:fffe
    Destination address: ff02::2
⊟ Internet Control Message Protocol v6
    Type: 133 (Router solicitation)
    Code: 0
    Checksum: 0x7d38 [correct]
```

Fig. 43. Ethereal[61] screenshot of a router solicitation (RS) packet (client to server). Only the nonce is included from the authentication data.

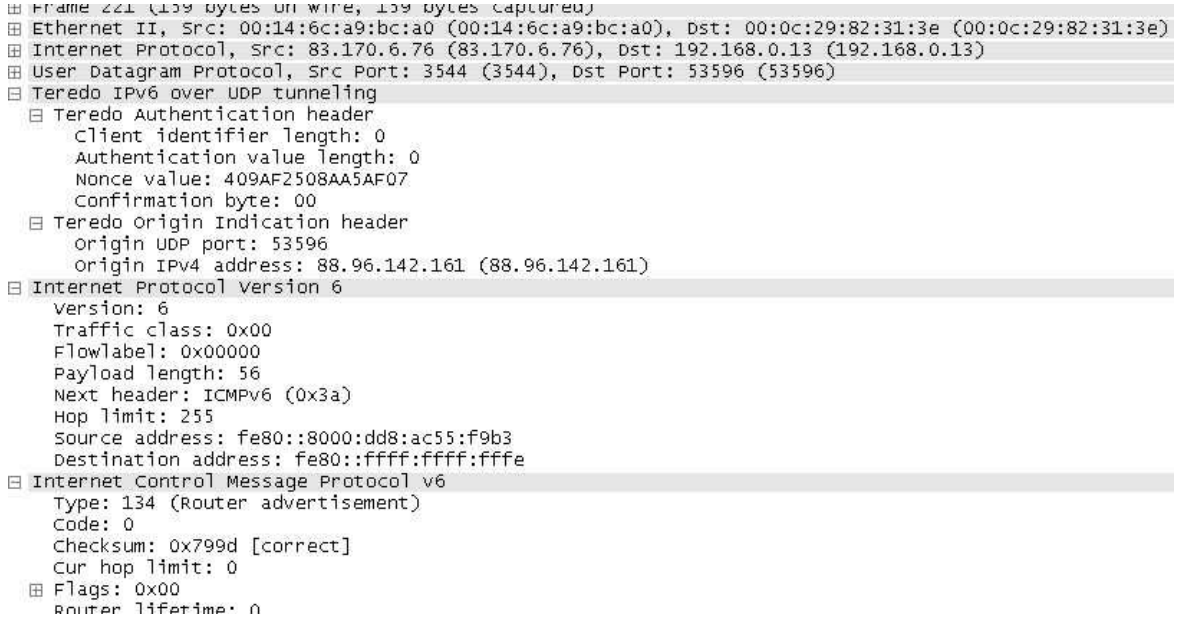

Fig. 44. Ethereal screenshot of a router advertisement (RA) packet (server to client). Only the nonce is included from the authentication data.

value length are both zero, it is apparent from these samples that only nonce authentication is used in the Windows Vista Teredo implementation. Thus, a man-in-the-middle attack becomes increasingly possible.

This finding was validated on both third-party Teredo servers, as well as on the default Vista Teredo servers at teredo.ipv6.microsoft.com.

The nonce field from secure qualification is used partly as an anti-spoofing mechanism and partly for authentication. To assess its effectiveness, it was important to understand how the nonce values are generated. Fortunately, the tracing code in place within the Teredo stack (see section XIII-H above) allowed the monitoring of nonce mismatches. Here are the nonce mismatches observed over a three minute period.

```
[VT:3592] 08:50:01: Nonce mismatch. Received 553648384 Expecting 427116032.
[VT:2880] 08:52:13: Nonce mismatch. Received 620757248 Expecting 1004707164.
[VT:3060] 08:53:16: Nonce mismatch. Received 771752192 Expecting -1769095775.
```

We can see from this small sample that there is an apparent high degree of randomness in the nonce selection. Also, it is evident that Microsoft Windows Vista checks the nonce returned and if not correct, the packet is dropped.

K. Same Nonce Used With Different UDP Ports

We note that a nonce appears to relate to a solicitation session; that is, during the qualification (address configuration) period the same nonce is used to communicate with one or more ports on the same Teredo server and for the life time of that relationship. This marginally increases the chance of spoofing the nonce (which is required to spoof the server) since there is a longer opportunity to try distinct nonces in a brute force attempt, and since there is more opportunity to observe the nonce in transit.

L. Ping Tests

As described earlier (Appendix XII-A), the ping test is used in part to counter spoofing attempts. To accomplish this, section 5.2.9 of RFC 4380[23] says:

> "... the client will pick a random number (a nonce) and format an ICMPv6 Echo Request message whose source is the local Teredo address, whose destination is the address of the IPv6 node, and whose Data field is set to the nonce. (It is recommended to use a random number at least 64 bits long.)"

A selection of different ping tests were collected by performing IPv6 ICMP ECHO REQUESTs to hosts located in geographically diverse areas on the Internet including North America, Asia and Europe. The purpose of this was to cause new relays to open up connections with the test Teredo client.

The sample below shows ping tests to four different destinations on the Internet. The data below shows the ICMP echo ID and sequence number and the data payload of the packet.

Ping ID	Ping Seq	Ping Data
6094	a65e	4275 2fe3
9e05	2971	0000 0000
764f	8e29	0000 0000
8366	f2ea	0000 0000
0ec4	586d	d5ac 308c
dd60	9db4	d5ac 308c
98a4	74e6	0000 0000
9a3b	47d4	0000 0000
30b1	4123	0000 0000

From this limited test there are two noteworthy discoveries. The first is that the ICMP echo data is only 32 bits in length and not the recommended 64 bits. The second is that in some cases the ICMP payload sent is actually all zeros. The consequence of implementing the ping test in this manner way is that the effectiveness of their anti-spoofing measure is reduced significantly.

When this is compared to the Miredo[14] implementation, the difference is striking. In Miredo, teredo_get_pinghash() takes the PID and current timestamp (which is valid for 30 seconds) and creates a hash seed. This, combined with the source and destination address, as well as the timestamp again are passed to teredo_pinghash(), which then passes them to teredo_hash(), which uses MD5 to hash all the elements along with 8 bytes of entropy padded to 64 bytes, which results in a hash with a maximum size of 128 bits.

M. Source Routing

One Teredo concern from [22] was the possibility that source routing could be abused to inject traffic into the local network. We investigated this for Vista hosts.

The command “netsh interface ipv6 show global” was executed as administrator. The output is identical to what is shown in Appendix IX. The Teredo interface was then identified using the ipconfig command in the test environment; this was “Local Area Connection* 7”. Then its the configuration information was extracted:

```
C:\Users\ollie\Desktop>netsh interface ipv6 show interfaces interface="Local Area Connection* 7"

Interface Local Area Connection* 7 Parameters
----------------------------------------------
IfLuid                             : tunnel_3
IfIndex                            : 10
Compartment Id                     : 1
State                              : connected
Metric                             : 10
Link MTU                           : 1280 bytes
Reachable Time                     : 7500 ms
Base Reachable Time                : 15000 ms
Retransmission Interval            : 2000 ms
DAD Transmits                      : 0
Site Prefix Length                 : 64
Site Id                            : 1
Forwarding                         : disabled
Advertising                        : disabled
Neighbor Discovery                 : enabled
Neighbor Unreachability Detecion   : enabled
Router Discovery                   : enabled
Managed Address Configuration      : disabled
Other Stateful Configuration       : disabled
Weak Host Sends                    : disabled
Weak Host Receives                 : disabled
Use Automatic Metric               : enabled
Ignore Default routes              : disabled
```

We can see from the above that the global source routing is set to “forward” but the Teredo interface Forwarding is set to “disabled”.

Although the global parameter sets Source Routing Behavior to forward, the fact that the Teredo network interface is configured with “Forwarding: disabled” means that it will not forward packets that are not destined for itself. Thus source routing redirection after being de-tunneled would not seem to be a concern under Vista.

N. Use of Address Flag Bits

Teredo addresses contain a 16 bit flags field. RFC 4380 only defines one bit of that field, the cone bit. However, according to [36], more bits are being used on Vista:

Test description	Test result
Does Vista (immediately) accept incoming packets from non-Teredo peers, where the IPv4 source address and port do not match the expected address and port.	Packet does not get past Windows Firewall
Does Vista (immediately) accept incoming packets from non-Teredo peers, where the IPv4 source address and port do not match the expected address and port - different source port only.	Packet does not get past Windows Firewall
Does Vista accept incoming packets from Teredo addresses, where the IPv4 source address and port are not the expected address and port expected (e.g. is a regular relay).	Packet does not get past Windows Firewall
Does Vista check the value returned in the ping?	Windows Vista uses the value of the ping in order to map the packet to the corresponding request.
Does mapping keep-alive to server use bubble per [36] or RS per [23].	Per the RFC: RS (Router Solicitation)
Does Vista check that the incoming source IP is not link-local, Unique Local IPv6 Unicast Addresses[21], or multicast?	It was determined that there are specific checks for Link Local, Unicast, Loopback, Teredo, ISATAP and Site Local. This result determined by looking through the IDA disassembly of IPHLPSVC.DLL (see Appendix XIV-B).
Does Vista check that the address is 2001::/32 and not 2001::/16	It was determined that the IN6_IS_ADDR_TERED() function checks that the destination address begins with 2001::/32. This result determined by looking through the IDA disassembly of IPHLPSVC.DLL (see Appendix XIV-B).
Does Vista check that the incoming destination IP is 2001::/32	It was determined that the IN6_IS_ADDR_TERED() function checks that the destination address begins with 2001::/32. This result determined by looking through the IDA disassembly of IPHLPSVC.DLL (see Appendix XIV-B).

Fig. 45. The attempted Teredo test cases

> "For Windows XP-based Teredo clients, the only defined flag is the high order bit known as the Cone flag. The Cone flag is set when Teredo client is behind a cone NAT...
>
> For Windows Vista and Windows Server "Longhorn"-based Teredo clients, unused bits within the Flags field provide a level of protection from address scans by malicious users. The 16 bits within the Flags field for Windows Vista and Windows Server "Longhorn"-based Teredo clients consists of the following: CRAAAAUG AAAAAAAA. The C bit is for the Cone flag. The R bit is reserved for future use. The U bit is for the Universal/Local flag (set to 0). The G bit is Individual/Group flag (set to 0). The A bits are set to a 12-bit randomly generated number. By using a random number for the A bits, a malicious user that has determined the rest of the Teredo address by capturing the initial configuration exchange of packets between the Teredo client and Teredo server will have to try up to 4,096 (2^{12}) different addresses to determine a Teredo client's address during an address scan."

We observed this, though we did not analyze the randomness of the 12 bits. For example, the address 2001:0:4136:e37a:1c1a:1080:f580:ea94 has 0x1c1a as the flags field. The A bits are 01110011010. Thus, we can verify these bits are in use. Assuming that 12 bits are selected at random (of which there is no guarantee), these extra 12 bits of entropy should make it more difficult for the attacker to guess a client's Teredo address (see [22]).

O. Other Attempted Test Cases

Figure 45 describes the test cases included in the Vista Teredo research, and documents the observed results.

P. Vista Teredo Conclusions

Teredo is a fairly simple tunneling protocol. It takes approximately 138KB of C code to implement[14], as a result there is a minimal attack surface to pursue apart from the Teredo protocol itself. We found no issues in Microsoft's Teredo stack implementation that might lead to the compromise of a remote Window Vista host.

We found that some security features in Windows Vista Teredo implementation are implemented minimally. In at least one situation, the implementation is below that recommended by Microsoft[23]. However, the extra 12 bits of randomness should make Vista Teredo addresses 4096 times harder to guess.

APPENDIX XIV
TEREDO IPHLPSVC INVESTIGATION

In the course of our Teredo investigation, we enabled tracing output, studied address checks, and studied function names for IPHLPSVC.DLL.

A. IPHLPSVC.DLL Tracing Output

The following shows an example of the tracing output from IPHLPSVC.DLL on a Microsoft Windows Vista RC1 build.

```
[VT:71240] 14:39:52: TeredoClientTimerCallback: SystemTime 4214681000
[VT:71240] 14:39:52: Get lock invoked at d:\vistarc1\net\netio\iphlpsvc\service\client.c : 896
[VT:71240] 14:39:52: Lock acquired at d:\vistarc1\net\netio\iphlpsvc\service\client.c : 896. Return 0
[VT:71240] 14:39:52: Next callback interval is 1
[VT:71240] 14:39:52: TeredoReferenceClient: ++4 @ d:\vistarc1\net\netio\iphlpsvc\service\client.c:2649
[VT:71240] 14:39:52: Transmitting a Router Solicitation
[VT:71240] 14:39:52: TeredoPrimaryTransmitPacket: 0x02E02608
[VT:71240] 14:39:53: DeviceTransmitComplete: 0x02E02608
[VT:71240] 14:39:53: TeredoDereferenceClient: --5 @ d:\vistarc1\net\netio\iphlpsvc\service\client.c:2080
[VT:71240] 14:39:53: Client State is 5, Router Solicit Count is 5
[VT:71240] 14:39:53: Lock released at d:\vistarc1\net\netio\iphlpsvc\service\client.c : 898. Return 1
[VT:71240] 14:39:53: TeredoClientPrimaryReceive: 0x03012EB8
[VT:71240] 14:39:53: Destination address of IPV6 is link local
[VT:71240] 14:39:53: TeredoClientRouterAdvertisement
[VT:71240] 14:39:53: Get lock invoked at d:\vistarc1\net\netio\iphlpsvc\service\client.c : 3410
[VT:71240] 14:39:53: Lock acquired at d:\vistarc1\net\netio\iphlpsvc\service\client.c : 3410. Return 0
[VT:71240] 14:39:53: Entered: TeredoClientQualified
[VT:71240] 14:39:53: TeredoClientQualified: Mapped-address is 88.96.142.161, source address is 192.168.0.8.
[VT:71240] 14:39:53: TeredoMappedIpAddressToLocation: Location = 88.
[VT:71240] 14:39:53: Lock released at d:\vistarc1\net\netio\iphlpsvc\service\client.c : 3527. Return 1
[VT:71240] 14:39:53: DeviceReceiveComplete: 0x03012EB8
[VT:1028] 14:40:02: SetHelperServiceStatus: Setting state to 3
[VT:1028] 14:40:02: ServiceHandler: Got a SERVICE_CONTROL_STOP control
[VT:67280] 14:40:02: Entered: OnStop
[VT:67280] 14:40:02: OnStop: Synchronizing with startup.
[VT:67280] 14:40:02: OnStop: Done synchronizing with startup, continuing...
[VT:67280] 14:40:02: Entering DeregisterNotificationHandlers
[VT:67280] 14:40:02: DeregisterNotificationHandlers: Handler 1
[VT:67280] 14:40:02: DeregisterNotificationHandlers: Handler 2
[VT:67280] 14:40:02: DeregisterNotificationHandlers: Handler 3
[VT:67280] 14:40:02: DeregisterNotificationHandlers: Handler 4
[VT:67280] 14:40:02: DeregisterNotificationHandlers: Handler 5
[VT:67280] 14:40:02: DeregisterNotificationHandlers: Handler 6
[VT:67280] 14:40:02: DeregisterNotificationHandlers: Handler 7
[VT:67280] 14:40:02: DeregisterNotificationHandlers: Disconnecting...
[VT:67280] 14:40:02: Leaving DeregisterNotificationHandlers
[VT:67280] 14:40:02: Get lock invoked at d:\vistarc1\net\netio\iphlpsvc\service\svcmain.c : 176
[VT:67280] 14:40:02: Lock acquired at d:\vistarc1\net\netio\iphlpsvc\service\svcmain.c : 176. Return 0
[VT:67280] 14:40:02: Entering StopHelperService
[VT:67280] 14:40:02: Entered: TeredoUninitialize
[VT:67280] 14:40:02: TeredoStopServer
[VT:67280] 14:40:02: CloseThreadpoolWait complete
[VT:67280] 14:40:02: CloseThreadpoolWait complete
[VT:67280] 14:40:02: CloseThreadpoolWait complete
[VT:67280] 14:40:02: CloseThreadpoolWait complete
[VT:67280] 14:40:02: CloseThreadpoolWait complete
[VT:67280] 14:40:02: DereferenceService: --5 (TeredoCleanupServer) @
d:\vistarc1\net\netio\iphlpsvc\service\server.c:734
[VT:67280] 14:40:02: Lock released at d:\vistarc1\net\netio\iphlpsvc\service\teredo.c : 1287. Return 1
[VT:67280] 14:40:02: Get lock invoked at d:\vistarc1\net\netio\iphlpsvc\service\teredo.c : 1293
[VT:67280] 14:40:02: Lock acquired at d:\vistarc1\net\netio\iphlpsvc\service\teredo.c : 1293. Return 0
[VT:67280] 14:40:02: Leaving: TeredoUninitialize
[VT:71240] 14:40:02: No timer callbacks pending
[VT:71240] 14:40:02: CloseThreadpoolWait complete
[VT:71240] 14:40:02: DereferenceService: --4 (IsatapTimerCleanup) @
d:\vistarc1\net\netio\iphlpsvc\service\isatap.c:1494
[VT:67280] 14:40:02: IsatapUninitialize: Uninstalling interface isatap.
{9F8B50B5-99A8-47EC-B505-75F0A8846CC4}
[VT:67280] 14:40:02: UpdateLinkAddress: LUID 83000002000000 DlAddressLength 12
[VT:67280] 14:40:02: IsatapUpdateLinkAddress: isatap.{9F8B50B5-99A8-47EC-B505-75F0A8846CC4} - link address
= 0.0.0.0; Succeeded
[VT:67280] 14:40:02: Entered: UninitializePorts
[VT:67280] 14:40:02: Get lock invoked at d:\vistarc1\net\netio\iphlpsvc\service\proxy.c : 1119
[VT:67280] 14:40:02: Lock acquired at d:\vistarc1\net\netio\iphlpsvc\service\proxy.c : 1119. Return 0
[VT:67280] 14:40:02: Lock released at d:\vistarc1\net\netio\iphlpsvc\service\proxy.c : 1129. Return 1
[VT:67280] 14:40:02: Leaving: UninitializePorts
```

```
[VT:67280] 14:40:02: Entering UninitializeRelays
[VT:67280] 14:40:02: Cancelling RT
[VT:67280] 14:40:02: Leaving UninitializeRelays
[VT:67280] 14:40:02: Deleting compartment 1
[VT:67280] 14:40:02: TeredoUninitializeCompartment: 1
[VT:71240] 14:40:02: No timer callbacks pending
[VT:71240] 14:40:02: CloseThreadpoolWait complete
[VT:71240] 14:40:02: TeredoDereferenceCompartment: 0x02E007C8 : --3 @
d:\vistarc1\net\netio\iphlpsvc\service\teredo.c:1055
[VT:71240] 14:40:02: DereferenceService: --3 (TeredoTimerCleanup) @
d:\vistarc1\net\netio\iphlpsvc\service\teredo.c:1056
[VT:67280] 14:40:02: TeredoStopClient 0x02E01488 (compartment 1, state 5)
[VT:67280] 14:40:02: Max previous state entries: 8
[VT:67280] 14:40:02: Unable to open key
System\CurrentControlSet\Services\iphlpsvc\Teredo\PreviousState\00-14-6c-a9-bc-a0. Error 2
[VT:67280] 14:40:02: Open key System\CurrentControlSet\Services\iphlpsvc\Teredo\PreviousState\
00-14-6c-a9-bc-a0 Status = 2
[VT:67280] 14:40:03: Maximum state entries: 0, Oldest state is ??
[VT:67280] 14:40:03: Create key System\CurrentControlSet\Services\iphlpsvc\Teredo\PreviousState\
00-14-6c-a9-bc-a0 Status = 0
[VT:71240] 14:40:03: TeredoClientTunnelReceive: 0x030107C8
[VT:71240] 14:40:03: DeviceReceiveComplete: 0x030107C8
[VT:71240] 14:40:03: TeredoClientTunnelReceive: 0x03010D58
[VT:71240] 14:40:03: DeviceReceiveComplete: 0x03010D58
[VT:71240] 14:40:03: TeredoClientTunnelReceive: 0x03011878
[VT:71240] 14:40:03: DeviceReceiveComplete: 0x03011878
[VT:67280] 14:40:03: TeredoSqmProcessHibernate: Timestamp 68714515
[VT:67280] 14:40:03: TeredoSqmProcessQualified: AddressLifetime = 219 seconds
[VT:67280] 14:40:03: TeredoSqm: set DATAID_TEREDO_SQM_VERSION to 4
[VT:67280] 14:40:03: TeredoSqm: set DATAID_FWT_CONNECTIVITY_TYPE to 5
[VT:67280] 14:40:03: TeredoSqm: set DATAID_AVG_RTT_TO_SERVER (PortPreservingNat) to 0
[VT:67280] 14:40:03: TeredoSqm: set DATAID_TEREDO_PEER_PEER_RTT (UpnpNat) to 0
[VT:67280] 14:40:03: TeredoSqm: set DATAID_TEREDO_CLIENT_TYPE to 0
[VT:67280] 14:40:03: TeredoSqm: set DATAID_TEREDO_NETWORK_TYPE to 0
[VT:67280] 14:40:03: TeredoSqm: Set DATAID_TEREDO_GEOGRAPHIC_LOCATION to 88
[VT:67280] 14:40:03: TeredoSqm: Set DATAID_TEREDO_ADDRESS_ETA to 313
[VT:67280] 14:40:03: TeredoSqm: Set DATAID_TEREDO_ADDRESS_LIFETIME to 219
[VT:67280] 14:40:03: TeredoSqm: Set DATAID_TEREDO_SESSION_DURATION to 0
[VT:67280] 14:40:03: TeredoSqm: Set DATAID_TEREDO_FAILURE_DURATION to 0
[VT:67280] 14:40:03: TeredoSqm: Set DATAID_TEREDO_CLIENT_SERVER_RTT to 251
[VT:67280] 14:40:03: TeredoSqm: Set DATAID_TEREDO_DATA_TRANSFERRED to 1842
[VT:67280] 14:40:03: TeredoSqm: Set DATAID_TEREDO_FAILURE_REASON to 0
[VT:67280] 14:40:03: TeredoSqm: Set DATAID_PORT_MAINTENANCE_TRAFFIC to 1190
[VT:67280] 14:40:03: TeredoSqm: Set DATAID_DATA_TO_OTHER_SERVERS to 0
[VT:67280] 14:40:03: TeredoSqm: Set DATAID_DATA_RECEIVED_FROM_SERVER to 48
[VT:67280] 14:40:03: TeredoSqm: Set DATAID_TOTAL_DATA_SENT_TO_PEERS to 260
[VT:67280] 14:40:03: TeredoSqm: Set DATAID_TOTAL_DATA_RECEIVED_FROM_PEERS to 344
[VT:67280] 14:40:03: TeredoSqm: Set DATAID_MIN_PEER_CONNECTION_TIME to -1
[VT:67280] 14:40:03: TeredoSqm: Set DATAID_MAX_PEER_CONNECTION_TIME to 0
[VT:67280] 14:40:03: TeredoSqm: Set DATAID_SUCCESSFUL_NATIVE_PEER_CONNECTIONS to 1
[VT:67280] 14:40:03: TeredoSqm: Set DATAID_UNSUCCESSFUL_NATIVE_PEER_CONNECTIONS to 0
[VT:67280] 14:40:03: TeredoSqm: Set DATAID_SUCCESSFUL_CONE_PEER_CONNECTIONS to 0
[VT:67280] 14:40:03: TeredoSqm: Set DATAID_UNSUCCESSFUL_CONE_PEER_CONNECTIONS to 0
[VT:67280] 14:40:03: TeredoSqm: Set DATAID_SUCCESSFUL_RESTRICTED_PEER_CONNECTIONS to 0
[VT:67280] 14:40:03: TeredoSqm: Set DATAID_UNSUCCESSFUL_RESTRICTED_PEER_CONNECTIONS to 0
[VT:67280] 14:40:03: TeredoSqm: Set DATAID_TOTAL_CONNECTS_TO_BAD_ADDDRESSES to 0
[VT:67280] 14:40:03: TeredoSqm: Set DATAID_TEREDO_SYSTEM_ERROR_CODE to 0
[VT:67280] 14:40:03: TeredoSqm: Set DATAID_TEREDO_SESSION_END_REASON to 0
[VT:67280] 14:40:03: TeredoSqm: Set DATAID_TEREDO_LIST_APP_TRIGGER_COUNT to 0
[VT:67280] 14:40:03: TeredoSqm: Set DATAID_TEREDO_OUT_PKT_TRIGGER_COUNT to 1
[VT:67280] 14:40:03: TeredoSqm: Set DATAID_TEREDO_NSI_TRIGGER_COUNT (Total dormancy exits) to 1
[VT:67280] 14:40:03: TeredoSqm: Set DATAID_TEREDO_DORMANCY_RELATED_ADDR_CHANGES to 1
[VT:67280] 14:40:03: TeredoSqm: Set DATAID_TEREDO_STARTUP_TIME to 360
[VT:67280] 14:40:03: TeredoSqm: Set DATAID_TEREDO_PS_RESPONSE_TIME to 78
[VT:67280] 14:40:03: TeredoSqm: Set DATAID_TEREDO_CRYPTO_HASH to 1
[VT:67280] 14:40:03: TeredoSqm: Set DATAID_TEREDO_ACTIVE_LIFETIME to 219
[VT:67280] 14:40:03: TeredoSqmStopClient: Timestamp 68714515.
[VT:67280] 14:40:03: TeredoSqmStopClient: SessionDuration = 68658 seconds
[VT:67280] 14:40:03: TeredoSqm: set DATAID_TEREDO_SQM_VERSION to 4
[VT:67280] 14:40:03: TeredoSqm: set DATAID_FWT_CONNECTIVITY_TYPE to 2
[VT:67280] 14:40:03: TeredoSqm: set DATAID_AVG_RTT_TO_SERVER (PortPreservingNat) to 1
[VT:67280] 14:40:03: TeredoSqm: set DATAID_TEREDO_PEER_PEER_RTT (UpnpNat) to 0
[VT:67280] 14:40:03: TeredoSqm: set DATAID_TEREDO_CLIENT_TYPE to 2
```

B. Address Checks in IPHLPSVC.DLL

It was discovered the Teredo stack specifically checks that incoming packets are not from link local addresses:

```
.text:25254225 ; __stdcall IN6_IS_ADDR_LINKLOCAL(x)
.text:25254225 _IN6_IS_ADDR_LINKLOCAL@4 proc near    ; CODE XREF: Ipv6UnicastAddressScope(x)+20 p
.text:25254225                                       ; TeredoClientProcessPeerMulticast(x)+2D p ...
.text:25254225
.text:25254225 arg_0           = dword ptr  8
.text:25254225
.text:25254225 ; FUNCTION CHUNK AT .text:25254213 SIZE 0000000D BYTES
.text:25254225
.text:25254225                 mov     edi, edi
.text:25254227                 push    ebp
.text:25254228                 mov     ebp, esp
.text:2525422A                 mov     eax, [ebp+arg_0]
.text:2525422D                 cmp     byte ptr [eax], 0FEh
.text:25254230                 jz      short loc_25254213
.text:25254232
.text:25254232 loc_25254232:                         ; CODE XREF: IN6_IS_ADDR_LINKLOCAL(x)-B j
.text:25254232                 xor     al, al
.text:25254234
.text:25254234 loc_25254234:                         ; CODE XREF: IN6_IS_ADDR_LINKLOCAL(x)-7 j
.text:25254234                 pop     ebp
.text:25254235                 retn    4
.text:25254235 _IN6_IS_ADDR_LINKLOCAL@4 endp
```

We can see in the above disassembly at 2525422D that it specifically compares the least significant octet of the address with FEh, which is an indicator of a link local address. After the JZ above is taken there is a further comparison to ensure that the address is within fe80::/9, which is the link local address space for IPv6.

```
.text:25254213 loc_25254213:                         ; CODE XREF: IN6_IS_ADDR_LINKLOCAL(x)+B j
.text:25254213                 mov     al, [eax+1]
.text:25254216                 and     al, 0C0h
.text:25254218                 cmp     al, 80h
.text:2525421A                 jnz     short loc_25254232
.text:2525421C                 add     al, 81h
.text:2525421E                 jmp     short loc_25254234
.text:2525421E ; END OF FUNCTION CHUNK FOR _IN6_IS_ADDR_LINKLOCAL@4
```

It was also discovered that the checks performed to determine if an address is a Teredo address or not, involve checking the first four octets as expected. At 25262DEE below we see as comparison between the value of the address and _in6addr_teredoprefix which is 0120h (2001) then below that at 25262DFB we see a comparison between that AX and word_2525FBE2 which is 00.

```
_IN6_IS_ADDR_TEREDO@4 proc near      ; CODE XREF: TeredoValidAdvertisedPrefix(x,x,x)+14 p
.text:25262DE3                       ; TeredoSqmProcessPeerConnection(x,x)+6F p ...
.text:25262DE3
.text:25262DE3 arg_0           = dword ptr  8
.text:25262DE3
.text:25262DE3                 mov     edi, edi
.text:25262DE5                 push    ebp
.text:25262DE6                 mov     ebp, esp
.text:25262DE8                 mov     eax, [ebp+arg_0]
.text:25262DEB                 mov     cx, [eax]
.text:25262DEE                 cmp     cx, ds:_in6addr_teredoprefix
.text:25262DF5                 jnz     short loc_25262E08
.text:25262DF7                 mov     ax, [eax+2]
.text:25262DFB                 cmp     ax, ds:word_2525FBE2
.text:25262E02                 jnz     short loc_25262E08
.text:25262E04                 mov     al, 1
.text:25262E06                 jmp     short loc_25262E0A
.text:25262E08 ; ---------------------------------------------------------------------------
.text:25262E08
.text:25262E08 loc_25262E08:                         ; CODE XREF: IN6_IS_ADDR_TEREDO(x)+12 j
.text:25262E08                                       ; IN6_IS_ADDR_TEREDO(x)+1F j
.text:25262E08                 xor     al, al
.text:25262E0A
.text:25262E0A loc_25262E0A:                         ; CODE XREF: IN6_IS_ADDR_TEREDO(x)+23 j
.text:25262E0A                 pop     ebp
.text:25262E0B                 retn    4
.text:25262E0B _IN6_IS_ADDR_TEREDO@4 endp
```

C. Teredo Functions from IPHLPSVC.DLL

The following lists the 244 functions exported from IPHLPSVC.DL that contain the word “Teredo” as part of the name:

```
_TeredoAddressArrival@4
_TeredoAddressDeletion@4
_TeredoAlterForwarding@8
_TeredoAlterWeakHostReceive@8
_TeredoAlterWeakHostSend@8
_TeredoChangeOrigin@8
_TeredoChecksumDatagram@20
_TeredoCleanup@0
_TeredoCleanupClient@4
_TeredoCleanupDevice@4
_TeredoCleanupIo@4
_TeredoCleanupServer@0
_TeredoClientCalculateInterval@4
_TeredoClientCompletePacket@4
_TeredoClientCompleteRsPacket@4
_TeredoClientConstructEchoRequest@8
_TeredoClientConstructIndirectBubble@8
_TeredoClientConstructNeighborAdvertisement@8
_TeredoClientConstructNeighborSolicitation@8
_TeredoClientFreeQueuedPackets@4
_TeredoClientGenerateTeredoAddress@4
_TeredoClientIpv4AddressDeletionNotification@8
_TeredoClientLoadWscLibrary@0
_TeredoClientLocalReceiveIpv6@4
_TeredoClientLocalReceiveNeighborDiscovery@4
_TeredoClientLocalReceiveRouterSolicitation@4
_TeredoClientPrimaryReceive@8
_TeredoClientProbeRestrictedSecondary@8
_TeredoClientProcessPeerMulticast@4
_TeredoClientProcessPeerPacket@4
_TeredoClientProcessQueuedPackets@4
_TeredoClientProcessQueuedPacketsEx@8
_TeredoClientProcessServerPacket@8
_TeredoClientProcessWSCNotification@4
_TeredoClientQualified@8
_TeredoClientRefreshIntervalChangeNotification@4
_TeredoClientRegisterWSCNotifications@4
_TeredoClientSecondaryReceive@8
_TeredoClientStartTypeSpecificBehavior@4
_TeredoClientStopIoComplete@4
_TeredoClientStopTypeSpecificBehavior@4
_TeredoClientTimerCallback@12
_TeredoClientTimerCallbackUnderLock@4
_TeredoClientTimerCleanup@16
_TeredoClientTunnelReceive@8
_TeredoClientTunnelReceiveHelper@8
_TeredoClientUpdateStateInNsi@4
_TeredoCompartmentAddAdapterNotification@8
_TeredoCompartmentArrival@8
_TeredoCompartmentChangeNotification@8
_TeredoCompartmentConfigurationChangeNotification@8
_TeredoCompartmentDeleteAdapterNotification@8
_TeredoCompartmentDeletion@8
_TeredoCompartmentNetworkChangeNotification@8
_TeredoCompartmentQueryGlobals@8
_TeredoCompartmentRequirementChangeNotification@8
_TeredoCompartmentRouteChangeNotification@8
_TeredoCompartmentTeredoChangeNotification@8
_TeredoCompartmentTunnelChangeNotification@8
_TeredoConfigurationChangeNotification@4
_TeredoConstructRouterAdvertisement@28
_TeredoCreateAndSendDirectBubble@8
_TeredoCreateAuthContext@16
_TeredoCreateGatewayKey@8
_TeredoCreatePacket@16
_TeredoCreatePeer@8
_TeredoCreatePrimarySocket@4
_TeredoCreateRegKey@8
_TeredoCreateSecondarySocket@4
_TeredoCreateTunnel@4
_TeredoDeleteAndRecreatePreviousState@4
_TeredoDeleteGatewayKey@4
_TeredoDeletePeer@4
_TeredoDereferenceClientEx@12
_TeredoDereferenceIoEx@12
_TeredoDereferencePeer@4
_TeredoDereferenceServer@4
_TeredoDeregisterFirewallExceptions@8
_TeredoDeregisterPortMapping@0
_TeredoDeregisterWmiEventNotification@0
_TeredoDestroyPacket@4
_TeredoDestroyPeer@4
_TeredoDestroyPrimarySocket@4
_TeredoDestroySecondarySocket@4
_TeredoDestroyTunnel@4
_TeredoDisableForwardingBehavior@4
_TeredoDisableWeakHostBehavior@4
_TeredoEnableForwardingBehavior@4
_TeredoEnableWeakHostBehavior@4
_TeredoEnableWmiEvent@8
_TeredoEnumerateInterfaces@12
_TeredoExtractAdvertisedAddresses@12
_TeredoFillInAddressTrailer@12
_TeredoFillInAuthInfo@20
_TeredoFillInNonceTrailer@12
_TeredoFindOrCreatePeer@8
_TeredoFindPeer@8
_TeredoGetAuthInfoSize@4
_TeredoGetMaximumAddressLifetime@0
_TeredoGetMaximumPreviousState@0
_TeredoGetPreferredSource@12
_TeredoGetPreviousAddressState@8
_TeredoGetTime@0
_TeredoHash@4
_TeredoHibernateClient@4
_TeredoInitialize@0
_TeredoInitializeAddressComponents@4
_TeredoInitializeAuthProvider@0
_TeredoInitializeClient@4
_TeredoInitializeClientPortMappings@4
_TeredoInitializeDevice@8
_TeredoInitializeIo@24
_TeredoInitializePacket@16
_TeredoInitializePacketContext@4
_TeredoInitializePeer@8
_TeredoInitializeRsPacket@4
_TeredoInitializeServer@0
_TeredoInitializeSqmState@4
_TeredoInitializeTimer@4
_TeredoInstallDeviceInCompartment@4
_TeredoInterface@8
_TeredoInterfaceChange@4
_TeredoInterfaceDeletion@4
_TeredoIpv4GlobalAddress@4
_TeredoIpv4ValidAddress@4
_TeredoIpv6GlobalAddress@4
_TeredoIsMappingEqual@8
_TeredoManagedNetwork@4
_TeredoMappedIpAddressToLocation@8
_TeredoMatchServerAddresses@8
_TeredoNetworkChangeNotificationWorker@8
_TeredoOpenRegKey@8
_TeredoParseAddress@8
_TeredoParseIpv6Headers@8
_TeredoParseOrigin@8
_TeredoPeerCompleteDynamicPacket@4
_TeredoPeerCompletePacket@4
_TeredoPeerRecentlyRefreshed@4
_TeredoPrimaryIoComplete@24
_TeredoPrimaryPostReceive@8
_TeredoPrimaryReceiveNotification@16
_TeredoPrimaryTransmitPacket@8
_TeredoProcessBubbleTrailer@16
_TeredoProcessRouterAdvertisement@8
```

```
_TeredoQueryGlobals@8
_TeredoQueuePacketAndTriggerClient@12
_TeredoReferenceClientEx@12
_TeredoReferenceIoEx@12
_TeredoRefreshIo@4
_TeredoRefreshPortMapping@4
_TeredoRegisterFirewallExceptions@8
_TeredoRegisterPortMapping@4
_TeredoRegisterWmiEventNotification@0
_TeredoRelayConstructNeighborAdvertisement@36
_TeredoRelayProcessReadPacket@4
_TeredoRelayProcessReceive@4
_TeredoResolveDefaultGateway@8
_TeredoResolveInterval@8
_TeredoResolveServer@4
_TeredoResolveServerAndGetPreferredSource@4
_TeredoRestartClient@4
_TeredoReusablePeer@4
_TeredoReuseOrCreatePeer@12
_TeredoReusePeer@8
_TeredoRouteChangeNotification@0
_TeredoSecondaryIoComplete@24
_TeredoSecondaryPostReceive@8
_TeredoSecondaryTransmitPacket@8
_TeredoServerIpv4AddressDeletionNotification@4
_TeredoServerPrimaryReceive@8
_TeredoServerProcessBubble@4
_TeredoServerProcessEchoRequest@4
_TeredoServerProcessReceive@8
_TeredoServerProcessRouterSolicitation@12
_TeredoServerSecondaryReceive@8
_TeredoServerStopIoComplete@4
_TeredoServerTunnelReceive@8
_TeredoServerUnreachabilityError@4
_TeredoSetPreviousAddressState@4
_TeredoSetPreviousTeredoAddress@8
_TeredoSqmBackupFailure@4
_TeredoSqmPeriodicCallback@4
_TeredoSqmPrintDatapoints@4
_TeredoSqmProcessDataPacketReceive@8
_TeredoSqmProcessDataPacketSend@8
_TeredoSqmProcessExitDormant@4
_TeredoSqmProcessExitOffline@4
_TeredoSqmProcessFailure@4
_TeredoSqmProcessHibernate@4
_TeredoSqmProcessIndirectBubbleReceive@8
_TeredoSqmProcessIndirectBubbleSend@12
_TeredoSqmProcessOutgoingPktTrigger@4
_TeredoSqmProcessPeerConnection@8
_TeredoSqmProcessPeerReuse@8
_TeredoSqmProcessQualified@4
_TeredoSqmProcessRouterAdvertisementReceive@8
_TeredoSqmProcessRouterSolicitationSend@8
_TeredoSqmProcessWakeup@4
_TeredoSqmResetData@4
_TeredoStartClient@4
_TeredoStartCompartment@8
_TeredoStartDevice@4
_TeredoStartIo@4
_TeredoStartOrRefreshClient@12
_TeredoStartOrRefreshServer@4
_TeredoStartServer@0
_TeredoStopClient@4
_TeredoStopCompartment@8
_TeredoStopDevice@4
_TeredoStopDeviceComplete@4
_TeredoStopEventCallback@16
_TeredoStopIo@4
_TeredoStopServer@0
_TeredoTimerCallback@12
_TeredoTimerCleanup@16
_TeredoTraceAddress@16
_TeredoTransmitDirectBubble@12
_TeredoTransmitIndirectBubble@12
_TeredoTransmitMulticastBubble@8
_TeredoTransmitPacket@8
_TeredoTransmitRouterSolicitation@8
_TeredoTunnelIoComplete@24
_TeredoTunnelPostReceive@8
_TeredoTunnelTransmitPacket@8
_TeredoTypeClient@4
_TeredoTypeServer@4
_TeredoUninitialize@4
_TeredoUninitializeAuthProvider@0
_TeredoUninitializeClient@4
_TeredoUninitializePeerSet@4
_TeredoUninitializeServer@0
_TeredoUninitializeSqmState@4
_TeredoUninitializeTimer@4
_TeredoUninstallDeviceInCompartment@4
_TeredoUpdatePeer@12
_TeredoUpdatePendingSqmDatapoints@4
_TeredoUpnpSymmetricNatEnvironment@4
_TeredoValidAddress@4
_TeredoValidAdvertisedPrefix@12
_TeredoValidPreviousStateAddress@8
_TeredoVerifyAndAssignServerAddress@8
_TeredoVerifyAuthInfo@12
_TeredoVerifyPreviousStateAddress@16
_TeredoWakeupClient@4
_TeredoWmiEventNotification@8
```

APPENDIX XV
HISTORIC ATTACKS

We tested the Vista networking stack using a suite of historical network stack attack tools. These were all sent across a 100Mbps Ethernet link. While previous Vista beta builds had been subject to some of these[49], we observed no abnormal effect from these on the release build of Vista.

The only attacks that had noticeable effect were opentear, udp, and udp2. Udp and udp2, which are UDP flooders, flooded the network and caused a reverse ping to begin failing.

Opentear sends a flood of improperly fragmented UDP packets, each from a different forged source address. A single instance could generate 12,000 packets per second. When hpvista was the target, the Windows GUI became very sluggish. When hpvista was running a reverse ping to the attacking machine, the pings were still generated promptly at one-second intervals and the GUI was sluggish, but we could only achieve 4000 pps in that situation. When acervista was targeted, a flood of 10–11,000 pps from a single instance of opentear caused the target to become unresponsive and reverse pings to cease. However, if the user was moving the mouse pointer at the attack onset, it could continue to be moved until the user paused. Nevertheless, it survived several copies of opentear running against it at the same time; in all cases, responsiveness resumed after that attack was stopped. The effects would be less pronounced on a slower link than 100Mbps.

While these attacks succeeded to an extent, it is apparently only due to the sheer packet load going across a fast Ethernet connection. We do not believe that a client workstation OS and its associated hardware should be expected to handle that packet volume.

We also tried blat, boink, bonk, land, naptha, neptune, pingexpoit, syndrop, synk4, and teardrop but noticed no effect. When there is no apparent impact, it is difficult to know with certainly that the exploit is functioning properly. The original exploits were used, except that SOCK_RAW was replaced with a modified pcap (containing a pcap_write() function). The changes were necessary because the Linux kernel does not allow control of the fragment offset field of the IP header when using SOCK_RAW.

APPENDIX XVI
IPV6 OPTIONS

We explored Vista's resilience to malformed IPv6 destination options by developing two scripts to send an IPv6 packet with a Destination Options extension header[13] with malformed options encoded in it. Recall that the options in the Destination Options extension header are similar to those in IPv4 in that they typically follow a type-length-value structure and are packed together without alignment considerations; however in IPv6 the length is the length of value and not the length of the entire option as it is in IPv4.

In this test, the Vista hosts were in their initial configuration, except that file sharing was enabled on vmvista2 and the "File and Printer Sharing (Echo Request - ICMPv6-In)" firewall exception was enabled on the other three hosts (see Appendix XXI).

A. Random Option Sending

The first script, randip6opts.py, choses a destination options extension header length[14] at random and fills the data portion of that header with random octets (which makes the options field quite likely not well formed or not sensical). This is sent to a specified target with No Next Header encoded as the next header. To aid in repeatability, the seed to the random number generator is a command line argument (-s) and it is possible to skip the actual sending of a specified number of packets.

A -F option is available to cause a random-sized region of the destination options to precede the totally random octets. That region contains options that are well-formed in that their encoded length matches their actual length. The option type is a randomly chosen value n which the highest two bits are unset; per RFC 2460[13], this means that the option is ignorable (can be skipped) if the option type is not understood. The option type may not be understood by Vista (or even defined) and the options data length and contents are probably abnormal.

We used this script with different options to send random options to each of our four Vista hosts in parallel:

- `./randip6opts.py $acerLL6%2 $acerMAC -s1 -w0`
- `./randip6opts.py $hpLL6%2 $hpMAC -s2 -F -w0`
- `./randip6opts.py $vmLL6%2 $vmMAC -s1 -l2100 -w0`
- `./randip6opts.py $vm2LL6%2 $vm2MAC -s2 -F -l2100 -w0`

The -l option changes the maximum packet length from the default 1500; to send packets over this 1500 octets, we split the packet into multiple fragments and send those separately. -w 0 means that there should be no pause between sending packets. The script can also send Hop-by-hop options, ping packets containing options, and include a region of well-formed options with random option types, but we did not explore these on release Vista.

After sending 210 million packets, we have not noticed any persistent side-effects on the Vista systems. Our monitoring of the Vista systems consists of periodic checks to ensure that the Windows GUI is still available and usable, and that a ping from the host to our analysis machine succeeds. Thus, we are unlikely to notice any temporary effects.

B. Ordered Option Sending

Our second script, seqip6opts.py, takes a more orderly approach to finding Vista stack deficiencies from IPv6 options processing. It always sends a single IPv6 destination option (other than possible no-ops), but varies three parameters: option type, encoded option value length, and actual option value length. Option type and encoded option length form the first two octets of the option. The actual option length varies between zero and the encoded option value length, thus there is a degree of truncation. Due to the increasing search space as encoded length increases, we vary that parameter most slowly. Within an given encoded length, we vary option number more slowly than the actual length. The octets in the option are filled in randomly. Since the Destination Options extension header is required to be a multiple of eight octets in length, we prefix our test destination option with no-ops as needed to start our test option so that it can be truncated to the desired length. For a required padding of one, we send option 0 (one octet pad). For two to seven octet pads, we send option 1 (multi-octet pad).

To enhance repeatability, the seed to the random number generator is a command line argument (-s), and the -k option specifies how many actual packet sends to skip. The -p option specifies that a short (4 octet) ping be included past the Destination Options extension header (as opposed to the default, which is to include nothing (No Next Header) past that header. -w specifies how long to pause after sending a packet.

We used -s1, -p, and one second pauses to test Vista. We divided up the sequence space across the four Vista hosts though the use of the -k option. We also collected a traffic capture while doing this. We pinged slowly so that, in the future, we may be able to use this traffic capture data to enumerate Vista's supported IPv6 options and lengths, through inspection of ping responses and any ICMP errors produced by the stack.

After sending the 8,421,376 packets in the defined sequence, we have not noticed any persistent effects on Vista. We are probably more likely to successfully find a defect without the -p option, since it will be more common for options to span past the end of the header.

[14]Extension headers are always multiples of eight octets in length. A one-octet length field indicates the options extension header length; to get the actual length, add one to the encoded length and multiply by eight. Thus the maximum extension header length is 2048 octets. The first two octets of the extension header are used to specify the next header and extension header length.

APPENDIX XVII
EPHEMERAL PORTS

We studied how ephemeral ports are used in Vista. First we used netsh to examine the default Vista ephemeral port setting:

```
netsh>interface ipv6 show dynamicport tcp

Protocol tcp Dynamic Port Range
---------------------------------
Start Port      : 49152
Number of Ports : 16384

netsh>interface ipv6 show dynamicport udp

Protocol udp Dynamic Port Range
---------------------------------
Start Port      : 49152
Number of Ports : 16384

netsh>interface ipv4 show dynamicport tcp

Protocol tcp Dynamic Port Range
---------------------------------
Start Port      : 49152
Number of Ports : 16384

netsh>interface ipv4 show dynamicport udp

Protocol udp Dynamic Port Range
---------------------------------
Start Port      : 49152
Number of Ports : 16384
```

As shown, the default range for TCP and UDP for IPv4 and IPv6 is 49152–65535. This corresponds to what IANA refers to as the “Dynamic and/or Private” range [25], and is in stark contrast to the Windows XP ephemeral port range of 1024–5000.

We experimented to see how ephemeral port settings are tied together:

```
netsh interface ipv6>set dynamicport protocol=tcp startport=49152 numberofports=16161
Ok.

netsh interface ipv6>show dynamicport tcp

Protocol tcp Dynamic Port Range
---------------------------------
Start Port      : 49152
Number of Ports : 16161

netsh interface ipv6>..

netsh interface>ipv4
netsh interface ipv4>show dynamicport tcp

Protocol tcp Dynamic Port Range
---------------------------------
Start Port      : 49152
Number of Ports : 16161

netsh interface ipv4>show dynamicport udp

Protocol udp Dynamic Port Range
---------------------------------
Start Port      : 49152
Number of Ports : 16384

netsh interface ipv4>set dynamicport protocol=tcp startport=49152 numberofports=16384
Ok.
```

The results show that there is a single setting for TCP which is applied to both IPv4 and IPv6, but which does not apply to UDP.

```
netsh interface ipv4>set dynamicport protocol=udp startport=49152 numberofports=16161
Ok.

netsh interface ipv4>show dynamicport udp

Protocol udp Dynamic Port Range
---------------------------------
Start Port      : 49152
Number of Ports : 16161

netsh interface ipv4>..

netsh interface>ipv6
netsh interface ipv6>show dynamicport udp

Protocol udp Dynamic Port Range
---------------------------------
Start Port      : 49152
Number of Ports : 16161

netsh interface ipv6>set dynamicport protocol=udp startport=49152 numberofports=16384
Ok.
```

The results show that the IPv4 and IPv6 UDP ephemeral port settings are similarly shared.

In our Vista testing, we have seen low-numbered ports in this ephemeral range used for both TCP and UDP, and Vista seems to increment through the range. It also appears that, while the same port number can be used for both IPv4 and IPv6 (as is often seen for TCP—see Appendix XXII), this happens only when the same process is behind both. For UDP, we see different port numbers being used for IPv4 and IPv6. This fact, and the way IPv4 and IPv6 are joined together for ephemeral port range setting, suggests that the allocation of used ephemeral ports is associated with TCP or UDP, and not the underlying transport protocol. However, TCP and UDP use different number-spaces.

APPENDIX XVIII
TCP INITIAL SEQUENCE NUMBER GENERATION

We observed the initial sequence number (ISN) generation of the Windows Vista stack for the release build, by sending SYN packets to an open port and observing the sequence number in the returned SYN+ACK packet. A custom utility, isn.py, was used for this testing. The utility sends SYN packets to port 5357 of the target, either over IPv4 or IPv6.

The source packet of each request was chosen sequentially, with either one or 100 repeated requests from the same port. When sending a single request from each source port the sequence numbers appear to be evenly distributed across the entire space:

```
linux# isn.py -c 1
src port 3340 1bd39480  (delta 466850944)
src port 3341 b19a5cfb  (delta 2512832635)
src port 3342 579029a0  (delta 2784349349)
src port 3343 9d74b6e2  (delta 1172606274)
src port 3344 7dcfabcf  (delta 3764057325)
src port 3345 db157fb9  (delta 1564857322)
src port 3346 1e02e871  (delta 1122855096)
src port 3347 2947f505  (delta 189074580)
src port 3348 6deb3fa3  (delta 1151552158)
src port 3349 f81f8add  (delta 2318682938)
src port 3350 2f80d0f7  (delta 929121818)
src port 3351 c7037372  (delta 2541920891)
src port 3352 813cb224  (delta 3124313778)
src port 3353 03848cd5  (delta 2185747121)
src port 3354 5cbd160f  (delta 1496877370)
src port 3355 aeace30c  (delta 1374670077)
src port 3356 24b4ac77  (delta 1980221803)
...
```

However, when sending multiple requests using the same source port (which causes the TCP connection identifier to remain unchanged across requests), we observe that the ISN is randomly incremented, based on an internal timer:

```
linux# isn.py -c 100
1c2a9fb2  (delta 472555442)
1c2b0339  (delta 25479)
1c2b255e  (delta 8741)
1c2b50a0  (delta 11074)
1c2b50a0  (delta 0)
1c2b6fa0  (delta 7936)
1c2bab19  (delta 15225)
1c2c0033  (delta 21786)
1c2c3018  (delta 12261)
1c2c3018  (delta 0)
1c2c3018  (delta 0)
1c2c3018  (delta 0)
1c2c3018  (delta 0)
1c2c3018  (delta 0)
...
avg dseq 00001513

src port 3341
b1ef9ee0  (delta 2512181399)
b1efd3a6  (delta 13510)
no reply
no reply
b1eff5c9  (delta 8739)
b1f0327e  (delta 15541)
b1f0639f  (delta 12577)
no reply
b1f0ba72  (delta 22227)
b1f0f771  (delta 15615)
no reply
b1f12258  (delta 10983)
b1f15461  (delta 12809)
b1f16361  (delta 3840)
b1f19d97  (delta 14902)
b1f1f614  (delta 22653)
b1f21660  (delta 8268)
...
avg dseq 00001661
```

Repeating these results with an additional “6” argument causes the tests to be performed with IPv6 instead of IPv4. The results are substantially the same as the results above, and are omitted.

The Vista stack appears to be utilizing random increments in its ISN generation while using the technique described in

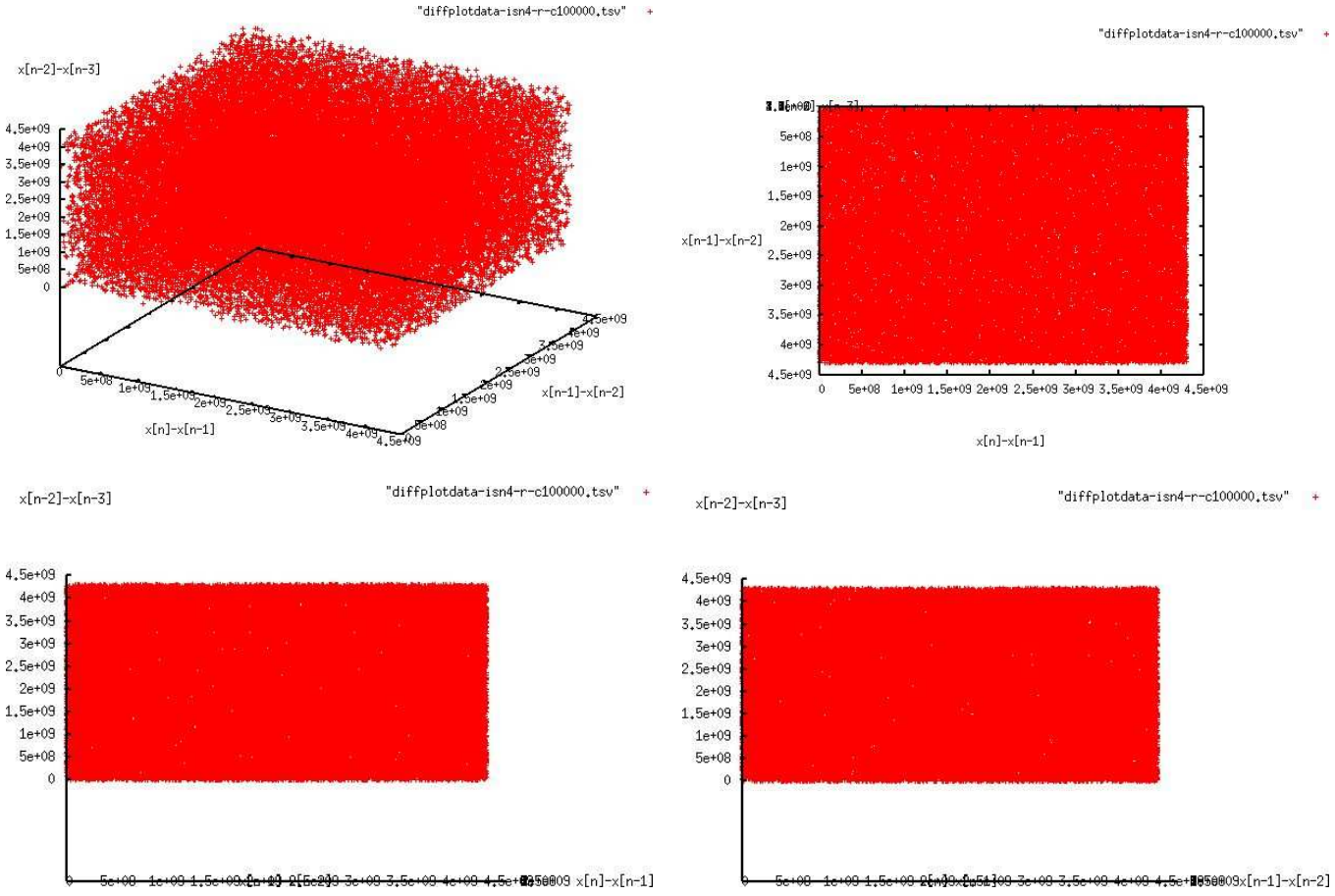

Fig. 46. Plot of ⟨x[n] - x[n-1], x[n-1] - x[n-2], x[n-2] - x[n-3]⟩ for each of 100,000 ISNs in response to IPv4 TCP SYNs with randomly generated source ports. The view at an angle and head-on from three different non-parallel sides.

RFC 1948[3] to maintain a separation between the ISN generation of connections with different connection identifiers. This is typically done by adding the value of a secret hash of the connection identifier to a global ISN counter. This is the same behavior seen in the Windows XP stack, except that Windows XP seems to increment the ISN counter more often, or by larger increments.

In order to collect a large number of data points for analysis, we ran isn.py so that it would randomly chose a source port for each of 100,000 probes; choosing the source ports at random means that ideally there should be no predictable pattern in the ISN. We collected data this way for both IPv4 and IPv6.

A rough measure of the strength of the ISN generation can be taken by making a state-space plot of the ISN deltas[62]. The values of ⟨x[n] - x[n-1], x[n-1] - x[n-2], x[n-2] - x[n-3]⟩ from a sequence, x, of ISN values are plotted in three dimensions. Patterns in this plot are often apparent for weaker generation schemes. We used gnuplot to examine these. Figure 46 shows four perspectives on the IPv4 plot. As in Windows XP, the plot appears uniformly distributed across the available space, indicating the strength of the generator. The IPv6 plot is substantially the same and is not shown.

We also used a different method of creating plot points from the ISN sequence, which looks for patterns that may skip an ISN or two. In this method, ⟨x[n]-x[n-1], x[n]-x[n-2], x[n]-x[n-3]⟩ form the data points. Figure 47 shows four perspectives on the IPv6 plot. This too appears uniform; the IPv4 plot is substantially the same.

Supporting our conclusion is the Nmap[18] stack fingerprinting output from verbose mode (see Appendix XX). In that output, Nmap reports “TCP Sequence Prediction: Difficulty=261 (Good luck!)”.

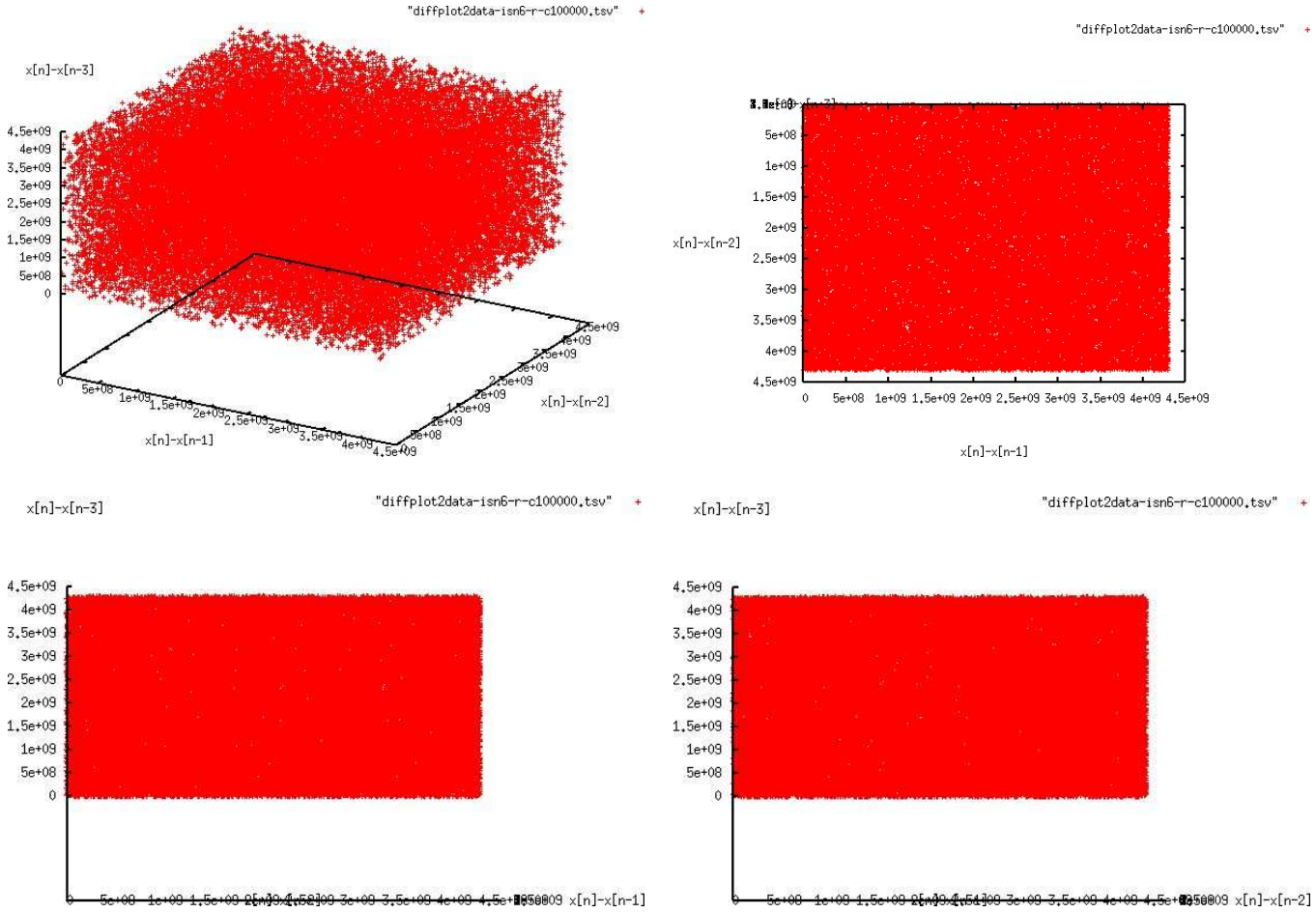

Fig. 47. Plot of ⟨x[n]-x[n-1], x[n]-x[n-2], x[n]-x[n-3]⟩ for each of 100,000 ISNs in response to IPv6 TCP SYNs with randomly generated source ports. The view at an angle and head-on from three different non-parallel sides.

APPENDIX XIX
TCP SEGMENT REASSEMBLY

Vista's networking stack behaves differently to earlier versions in Windows XP or Windows 2000, when reassembling TCP segments. Vista uses a policy that prefers previously received data to more recently received data. This preference is enforced on a byte-by-byte basis and not across entire segments.

We performed our testing by sending out several out-of-order segments in a TCP stream that contained conflicting data. We then observed the stream data that were delivered to the application layer on the target machine. Tests were performed using a tool that we wrote for the test, which listens for SYN packets sent to port 998 on the test machine. No proper socket is opened on 998 though, so we used "`iptables --protocol tcp -A OUTPUT --tcp-flags RST RST -j DROP`" to suppress Linux's natural response to the SYN. After the test script receives a SYN, it complete the three-way handshake to establish a connection with the sender. At that point, the conflicting segments are sent, and then finally, an RST packet. To test a host, netcat is used and the results received by netcat are recorded (i.e. "nc 192.168.0.102 999").

A. Test Data

Seven segments containing ambiguous data were sent out with 4-bytes of data each. Each segment overlapped at least one other segment by two bytes. The following indicates how the segments data overlapped:

```
Segment #1            2222
Segment #2                  5555
Segment #3                    6666
Segment #4                4444
Segment #5                oooo
Segment #6              3333
Segment #7          1111
```

The following shows the resulting TCP stream after reassembly:

```
Linux Red Hat 8     11112233445566
Windows 2000        11112244445566
Windows XP          11112244445566
Windows Vista       11222244555566
```

Note that it was important that the last segment is the only one from the start of the range on which the testing is performed. Otherwise, part of the segment space would be passed along to the socket listener before the stack receives all the segments. This is a stronger requirement than for IP fragmentation testing, which only requires that the last fragment be a missing piece of the fragment space.

B. Analysis

Windows Vista resolved all conflicts by preferring bytes from segments that were received earliest. This behavior differs from the behavior of earlier versions (for example, Windows 2000 and Windows XP) which seem to process segments in the order they are received by first trimming any excess from the left, and then using the rest of the segment in its entirety, overwriting any existing data.

APPENDIX XX
STACK FINGERPRINT

We ran the TCP-based Nmap with the -O option to gather a stack fingerprint for Vista. With the firewall on (the default) and in the private profile, we obtained the following results:

```
linux# nmap -O -p 5357,999 $vm2IP4

Starting Nmap 4.10 ( http://www.insecure.org/nmap/ ) at 2006-12-04 09:16 PST
Warning:  OS detection will be MUCH less reliable because we did not find at least 1 open
and 1 closed TCP port
Interesting ports on 192.168.0.204:
PORT     STATE    SERVICE
999/tcp  filtered garcon
5357/tcp open     unknown
MAC Address: 00:0C:29:1B:50:AA (VMware)
Aggressive OS guesses: Compaq Inside Management Board (91%), Phillips ReplayTV 5000 DVR
 (91%), Microsoft Windows XP Home Edition (German) SP2 (90%), Microsoft Windows XP Pro SP2
 (90%), NetScreen NS-204 Firewall (90%), Symantec Enterprise Firewall 7.0 running on
 Windows 2000 SP2 (90%), Enterasys XSR-1805 Security Route (90%), FreeBSD 4.6 (90%),
 Microsoft Windows 2003 Server or XP SP2 (90%), Apple Mac OS X 10.3.6 or 10.3.7 (88%)
No exact OS matches for host (test conditions non-ideal).

Nmap finished: 1 IP address (1 host up) scanned in 36.559 seconds
```

Nmap did not yield much information, since there were no closed ports.

With the firewall off, additional results were obtained. Nmap reported the following:

```
linux# nmap -O -p 5357,999 $vm2IP4

Starting Nmap 4.10 ( http://www.insecure.org/nmap/ ) at 2006-12-04 09:17 PST
Interesting ports on 192.168.0.204:
PORT     STATE  SERVICE
999/tcp  closed garcon
5357/tcp open   unknown
MAC Address: 00:0C:29:1B:50:AA (VMware)
No exact OS matches for host (If you know what OS is running on it, see
http://www.insecure.org/cgi-bin/nmap-submit.cgi).
TCP/IP fingerprint:
SInfo(V=4.10%P=i686-pc-linux-gnu%D=12/4%Tm=45745856%O=5357%C=999%M=000C29)
TSeq(Class=TR%IPID=I)
T1(Resp=Y%DF=N%W=2000%ACK=S++%Flags=AS%Ops=MNWNNT)
T2(Resp=Y%DF=Y%W=0%ACK=S%Flags=AR%Ops=)
T3(Resp=Y%DF=Y%W=0%ACK=O%Flags=AR%Ops=)
T4(Resp=Y%DF=Y%W=0%ACK=O%Flags=R%Ops=)
T5(Resp=Y%DF=Y%W=0%ACK=S++%Flags=AR%Ops=)
T6(Resp=Y%DF=Y%W=0%ACK=O%Flags=R%Ops=)
T7(Resp=Y%DF=Y%W=0%ACK=S++%Flags=AR%Ops=)
PU(Resp=Y%DF=N%TOS=0%IPLEN=164%RIPTL=148%RID=E%RIPCK=E%UCK=E%ULEN=134%DAT=E)



Nmap finished: 1 IP address (1 host up) scanned in 20.213 seconds
```

Even without fully understanding how to read Nmap signatures, this result is noticeably different from that obtained with XP SP2. The result obtained from XP SP2 has the following signature:

```
Fingerprint Microsoft Windows 2003 Server or XP SP2
Class Microsoft | Windows | 2003/.NET | general purpose
Class Microsoft | Windows | NT/2K/XP | general purpose
TSeq(Class=TR%gcd=<6%IPID=I)
T1(DF=Y%W=402E|FB8B%ACK=S++%Flags=AS%Ops=MNWNNT)
T2(Resp=Y%DF=N%W=0%ACK=S%Flags=AR%Ops=)
T3(Resp=Y%DF=Y%W=402E|FB8B%ACK=S++%Flags=AS%Ops=MNWNNT)
T4(DF=N%W=0%ACK=O%Flags=R%Ops=)
T5(DF=N%W=0%ACK=S++%Flags=AR%Ops=)
T6(DF=N%W=0%ACK=O%Flags=R%Ops=)
T7(DF=N%W=0%ACK=S++%Flags=AR%Ops=)
PU(DF=N%TOS=0%IPLEN=B0%RIPTL=148%RID=E%RIPCK=E|F%UCK=E|F%ULEN=134%DAT=E)
```

There is little difference between the release Vista signature and the Vista 5384 signature that we reported in [49]. The only difference is that 5384 build had Ops=MWNNNT instead of Ops=MNWNNT for T1. Thus Microsoft switched one of the null

pads (‘N’) to the other side of the Window scale factor option (‘W’), which makes it match that part of the XP SP2 signature. This option ordering gives more convenient word alignment to the scale option, avoiding the need to do an eight-bit shift right to access the field if the options are read into 32 or 64 bit words, since the MSS option (‘M’) is 4 octets long.

Using the newly released Nmap version 4.20, which incorporates a second generation OS fingerprinting engine, yielded two different results with the firewall off. Run non-verbose yielded the following results:

```
linux# nmap -O -p 5357,999 $vmIP4

Starting Nmap 4.20 ( http://insecure.org ) at 2006-12-08 13:50 PST
Interesting ports on 192.168.0.203:
PORT     STATE  SERVICE
999/tcp  closed garcon
5357/tcp open   unknown
MAC Address: 00:0C:29:72:E4:82 (VMware)
Device type: general purpose
Running: Microsoft Windows Vista
OS details: Microsoft Windows Vista Beta 2 (Build 5472)
Uptime: 0.889 days (since Thu Dec  7 16:30:16 2006)
Network Distance: 1 hop

OS detection performed. Please report any incorrect results at
http://insecure.org/nmap/submit/ .
Nmap finished: 1 IP address (1 host up) scanned in 15.098 seconds
```

When run non-verbose, Nmap did not venture a guess at the OS, but did print its signature. The output:

```
linux# nmap -O -p 5357,999 $vmIP4

Starting Nmap 4.20 ( http://insecure.org ) at 2006-12-08 13:53 PST
Initiating ARP Ping Scan at 13:53
Scanning 192.168.0.203 [1 port]
Completed ARP Ping Scan at 13:53, 0.01s elapsed (1 total hosts)
Initiating Parallel DNS resolution of 1 host. at 13:53
Completed Parallel DNS resolution of 1 host. at 13:53, 13.00s elapsed
Initiating SYN Stealth Scan at 13:53
Scanning 192.168.0.203 [2 ports]
Discovered open port 5357/tcp on 192.168.0.203
Completed SYN Stealth Scan at 13:53, 1.11s elapsed (2 total ports)
Initiating OS detection (try #1) against 192.168.0.203
Retrying OS detection (try #2) against 192.168.0.203
Retrying OS detection (try #3) against 192.168.0.203
Retrying OS detection (try #4) against 192.168.0.203
Retrying OS detection (try #5) against 192.168.0.203
Host 192.168.0.203 appears to be up ... good.
Interesting ports on 192.168.0.203:
PORT     STATE  SERVICE
999/tcp  closed garcon
5357/tcp open   unknown
MAC Address: 00:0C:29:72:E4:82 (VMware)
No exact OS matches for host (If you know what OS is running on it, see
http://insecure.org/nmap/submit/ ).
TCP/IP fingerprint:
OS:SCAN(V=4.20%D=12/8%OT=5357%CT=999%CU=35902%PV=Y%DS=1%G=Y%M=000C29%TM=457
OS:9DEEA%P=i686-pc-linux-gnu)SEQ(SP=105%GCD=1%ISR=FF%TI=I%II=I%SS=S%TS=7)SE
OS:Q(SP=104%GCD=1%ISR=FF%TI=I%II=I%SS=S%TS=7)SEQ(SP=105%GCD=1%ISR=FF%TI=I%I
OS:I=I%SS=S%TS=7)OPS(O1=M5B4NW8ST11%O2=M5B4NW8ST11%O3=M5B4NW8NNT11%O4=M5B4N
OS:W8ST11%O5=M5B4NW8ST11%O6=M5B4ST11)WIN(W1=2000%W2=2000%W3=2000%W4=2000%W5
OS:=2000%W6=2000)ECN(R=Y%DF=Y%T=80%W=2000%O=M5B4NW8NNS%CC=N%Q=)T1(R=Y%DF=Y%
OS:T=80%S=O%A=S+%F=AS%RD=0%Q=)T2(R=Y%DF=Y%T=80%W=0%S=Z%A=S%F=AR%O=%RD=0%Q=)
OS:T3(R=Y%DF=Y%T=80%W=0%S=Z%A=O%F=AR%O=%RD=0%Q=)T4(R=Y%DF=Y%T=80%W=0%S=A%A=
OS:O%F=R%O=%RD=0%Q=)T5(R=Y%DF=Y%T=80%W=0%S=Z%A=S+%F=AR%O=%RD=0%Q=)T6(R=Y%DF
OS:=Y%T=80%W=0%S=A%A=O%F=R%O=%RD=0%Q=)T7(R=Y%DF=Y%T=80%W=0%S=Z%A=S+%F=AR%O=
OS:%RD=0%Q=)U1(R=Y%DF=N%T=80%TOS=0%IPL=164%UN=0%RIPL=G%RID=G%RIPCK=G%RUCK=G
OS:%RUL=G%RUD=G)IE(R=Y%DFI=N%T=80%TOSI=Z%CD=Z%SI=S%DLI=S)


Uptime: 0.558 days (since Fri Dec  8 00:30:56 2006)
Network Distance: 1 hop
TCP Sequence Prediction: Difficulty=261 (Good luck!)
IPID Sequence Generation: Incremental
```

```
OS detection performed. Please report any incorrect results at
http://insecure.org/nmap/submit/ .
Nmap finished: 1 IP address (1 host up) scanned in 23.820 seconds
               Raw packets sent: 84 (7264B) | Rcvd: 84 (7010B)
```

APPENDIX XXI
WINDOWS FIREWALL CONFIGURATION

We measured Windows Firewall configuration on a clean Vista machine, and we noted the changes that occurred when configuration changes were made to the Windows Vista system. We read the firewall settings from the inbound rules in an application called Windows Firewall with Advanced Security. The table represents the live state of the firewall configuration (as of the last refresh request), and the program can export this as text.

During testing, we went through a series of states. At each state, we refreshed the table and exported that state. We also ran `netstat -a -b -n -o` as Administrator to get a full list of network sockets and their owners. In this appendix, we report on the initial state (Appendix XXI-B) and on changes in the data in between states (Appendix XXI-C and XXI-D).

When a user makes a configuration change that alters Windows Firewall configuration, Windows Vista typically asks for the users permission before making the change using the consent mechanism. Opening up Windows Firewall with Advanced Security also requires the consent prompt to be accepted.

A. Firewall ruleset

The initial firewall ruleset consists of 166 entries. Each entry consists of a number of parameters:

- Name
- Group
- Profile
- Enabled
- Action
- Override
- Program
- Local Address
- Remote Address
- Protocol
- Local Port
- Remote Port

There are three profiles to which firewall rules can apply: private, domain, and public. A given network connection is in one of those profiles at a time, representing the level of trust in the network, as described in [12]. The firewall ruleset could be regarded as three different rulesets. Some rules are listed as applying to multiple profiles, but when comparing rulesets, our analysis script considers those in a split-out manner.

Since we only saw Action=Allow, Enabled=Yes means that an exception is created for the combination (union) of program, protocol, addresses, and ports in the rule, and Enabled=No means the exception is not in place. We observed that the automatic enabling or disabling of rules took place for an entire group and profile together (for convenience, the logical combination of the rules in a profile within a group is termed a “group-profile”). The group name is also displayed on the Windows Firewall control panel.

B. Initial State

The following table lists the initial inbound firewall state, as exported from Windows Firewall with Advanced Security. In the table, certain columns are omitted since they have a uniform value (Action=Allow, Override=No, Local Address=Any). Remote Port was set to Any except for a Core Networking - Dynamic Host Configuration Protocol (DHCP-In), which is bound to remote port 67. The name of the rule always begins with the name of the group it is in, so we omit group name; the group name is the part before the first dash or open parenthesis. Originally the ports were all separated by commas, but we condensed the Local Port specification for a couple rules into a range.

TABLE I: The initial firewall ruleset state

Name	Profile	Enabled	Program	Remote Address	Protocol	Local Port
BITS Peercaching (Content-In)	Domain, Private, Public	No	System	Local subnet	TCP	2178
BITS Peercaching (RPC-EPMAP)	Domain, Private, Public	No	%SystemRoot%\system32\svchost.exe	Local subnet	TCP	RPC Endpoint Mapper
BITS Peercaching (RPC)	Domain, Private, Public	No	%SystemRoot%\system32\svchost.exe	Local subnet	TCP	Dynamic RPC
BITS Peercaching (WSD-In)	Domain, Private, Public	No	%SystemRoot%\system32\svchost.exe	Local subnet	UDP	3702
Connect to a Network Projector (TCP-In)	Domain	No	%SystemRoot%\system32\netproj.exe	Any	TCP	Any
Connect to a Network Projector (TCP-In)	Private, Public	No	%SystemRoot%\system32\netproj.exe	Local subnet	TCP	Any
Connect to a Network Projector (WSD Events-In)	Domain	No	System	Any	TCP	5357

(Continued on next page)

Name	Profile	Enabled	Program	Remote Address	Protocol	Local Port
Connect to a Network Projector (WSD Events-In)	Private, Public	No	System	Local subnet	TCP	5357
Connect to a Network Projector (WSD EventsSecure-In)	Domain	No	System	Any	TCP	5358
Connect to a Network Projector (WSD EventsSecure-In)	Private, Public	No	System	Local subnet	TCP	5358
Connect to a Network Projector (WSD-In)	Domain, Private, Public	No	%SystemRoot%\system32\netproj.exe	Local subnet	UDP	3702
Core Networking - Destination Unreachable (ICMPv6-In)	Domain, Private, Public	Yes	System	Any	ICMPv6	Any
Core Networking - Destination Unreachable Fragmentation Needed (ICMPv4-In)	Domain, Private, Public	Yes	System	Any	ICMPv4	Any
Core Networking - Dynamic Host Configuration Protocol (DHCP-In)	Domain, Private, Public	Yes	%SystemRoot%\system32\svchost.exe	Any	UDP	68
Core Networking - Internet Group Management Protocol (IGMP-In)	Domain, Private, Public	Yes	System	Any	IGMP	Any
Core Networking - IPv6 (IPv6-In)	Domain, Private, Public	Yes	System	Any	IPv6	Any
Core Networking - Multicast Listener Done (ICMPv6-In)	Domain, Private, Public	Yes	System	Any	ICMPv6	Any
Core Networking - Multicast Listener Query (ICMPv6-In)	Domain, Private, Public	Yes	System	Any	ICMPv6	Any
Core Networking - Multicast Listener Report (ICMPv6-In)	Domain, Private, Public	Yes	System	Any	ICMPv6	Any
Core Networking - Multicast Listener Report v2 (ICMPv6-In)	Domain, Private, Public	Yes	System	Any	ICMPv6	Any
Core Networking - Neighbor Discovery Advertisement (ICMPv6-In)	Domain, Private, Public	Yes	System	Any	ICMPv6	Any
Core Networking - Neighbor Discovery Solicitation (ICMPv6-In)	Domain, Private, Public	Yes	System	Any	ICMPv6	Any
Core Networking - Packet Too Big (ICMPv6-In)	Domain, Private, Public	Yes	System	Any	ICMPv6	Any
Core Networking - Parameter Problem (ICMPv6-In)	Domain, Private, Public	Yes	System	Any	ICMPv6	Any
Core Networking - Router Advertisement (ICMPv6-In)	Domain, Private, Public	Yes	System	Any	ICMPv6	Any
Core Networking - Teredo (UDP-In)	Domain, Private, Public	Yes	%SystemRoot%\system32\svchost.exe	Any	UDP	Edge Traversal
Core Networking - Time Exceeded (ICMPv6-In)	Domain, Private, Public	Yes	System	Any	ICMPv6	Any
Distributed Transaction Coordinator (RPC-EPMAP)	Domain	No	%SystemRoot%\system32\svchost.exe	Any	TCP	RPC Endpoint Mapper
Distributed Transaction Coordinator (RPC-EPMAP)	Private, Public	No	%SystemRoot%\system32\svchost.exe	Local subnet	TCP	RPC Endpoint Mapper
Distributed Transaction Coordinator (RPC)	Domain	No	%SystemRoot%\system32\svchost.exe	Any	TCP	Dynamic RPC
Distributed Transaction Coordinator (RPC)	Private, Public	No	%SystemRoot%\system32\svchost.exe	Local subnet	TCP	Dynamic RPC
Distributed Transaction Coordinator (TCP-In)	Domain	No	%SystemRoot%\system32\msdtc.exe	Any	TCP	Any
Distributed Transaction Coordinator (TCP-In)	Private, Public	No	%SystemRoot%\system32\msdtc.exe	Local subnet	TCP	Any
File and Printer Sharing (Echo Request - ICMPv4-In)	Domain, Private, Public	No	Any	Any	ICMPv4	Any
File and Printer Sharing (Echo Request - ICMPv6-In)	Domain, Private, Public	No	Any	Any	ICMPv6	Any
File and Printer Sharing (NB-Datagram-In)	Domain	No	System	Any	UDP	138
File and Printer Sharing (NB-Datagram-In)	Private, Public	No	System	Local subnet	UDP	138
File and Printer Sharing (NB-Name-In)	Domain	No	System	Any	UDP	137
File and Printer Sharing (NB-Name-In)	Private, Public	No	System	Local subnet	UDP	137
File and Printer Sharing (NB-Session-In)	Domain	No	System	Any	TCP	139
File and Printer Sharing (NB-Session-In)	Private, Public	No	System	Local subnet	TCP	139
File and Printer Sharing (SMB-In)	Domain	No	System	Any	TCP	445
File and Printer Sharing (SMB-In)	Private, Public	No	System	Local subnet	TCP	445
File and Printer Sharing (Spooler Service - RPC-EPMAP)	Domain	No	Any	Any	TCP	RPC Endpoint Mapper
File and Printer Sharing (Spooler Service - RPC-EPMAP)	Private, Public	No	Any	Local subnet	TCP	RPC Endpoint Mapper
File and Printer Sharing (Spooler Service - RPC)	Domain	No	%SystemRoot%\system32\spoolsv.exe	Any	TCP	Dynamic RPC
File and Printer Sharing (Spooler Service - RPC)	Private, Public	No	%SystemRoot%\system32\spoolsv.exe	Local subnet	TCP	Dynamic RPC
iSCSI Service (TCP-In)	Domain	No	%SystemRoot%\system32\svchost.exe	Any	TCP	Any
iSCSI Service (TCP-In)	Private, Public	No	%SystemRoot%\system32\svchost.exe	Local subnet	TCP	Any
Media Center Extenders (HTTP-Streaming-In)	Domain, Private, Public	No	System	Local subnet	TCP	10244

(Continued on next page)

Name	Profile	Enabled	Program	Remote Address	Protocol	Local Port
Media Center Extenders (qWave-TCP-In)	Domain, Private, Public	No	%SystemRoot%\system32\svchost.exe	Local subnet	TCP	2177
Media Center Extenders (qWave-UDP-In)	Domain, Private, Public	No	%SystemRoot%\system32\svchost.exe	Local subnet	UDP	2177
Media Center Extenders (RDP-In)	Domain, Private, Public	No	System	Local subnet	TCP	3390
Media Center Extenders (RTSP-In)	Domain, Private, Public	No	%SystemRoot%\ehome\ehshell.exe	Local subnet	TCP	554, 8554–8558
Media Center Extenders (SSDP-In)	Domain, Private, Public	No	%SystemRoot%\system32\svchost.exe	Local subnet	UDP	1900
Media Center Extenders (WMDRM-ND/RTP/RTCP-In)	Domain, Private, Public	No	%SystemRoot%\ehome\ehshell.exe	Local subnet	UDP	7777–7780, 7781, 5004, 5005, 50004–50013
Network Discovery (LLMNR-UDP-In)	Domain	No	%SystemRoot%\system32\svchost.exe	Any	UDP	5355
Network Discovery (LLMNR-UDP-In)	Private	Yes	%SystemRoot%\system32\svchost.exe	Local subnet	UDP	5355
Network Discovery (LLMNR-UDP-In)	Public	No	%SystemRoot%\system32\svchost.exe	Local subnet	UDP	5355
Network Discovery (NB-Datagram-In)	Domain	No	System	Any	UDP	138
Network Discovery (NB-Datagram-In)	Private	Yes	System	Local subnet	UDP	138
Network Discovery (NB-Datagram-In)	Public	No	System	Local subnet	UDP	138
Network Discovery (NB-Name-In)	Domain	No	System	Any	UDP	137
Network Discovery (NB-Name-In)	Private	Yes	System	Local subnet	UDP	137
Network Discovery (NB-Name-In)	Public	No	System	Local subnet	UDP	137
Network Discovery (Pub-WSD-In)	Domain, Public	No	%SystemRoot%\system32\svchost.exe	Local subnet	UDP	3702
Network Discovery (Pub-WSD-In)	Private	Yes	%SystemRoot%\system32\svchost.exe	Local subnet	UDP	3702
Network Discovery (SSDP-In)	Domain, Public	No	%SystemRoot%\system32\svchost.exe	Local subnet	UDP	1900
Network Discovery (SSDP-In)	Private	Yes	%SystemRoot%\system32\svchost.exe	Local subnet	UDP	1900
Network Discovery (UPnP-In)	Domain	No	System	Any	TCP	2869
Network Discovery (UPnP-In)	Private	Yes	System	Local subnet	TCP	2869
Network Discovery (UPnP-In)	Public	No	System	Local subnet	TCP	2869
Network Discovery (WSD Events-In)	Domain	No	System	Any	TCP	5357
Network Discovery (WSD Events-In)	Private	Yes	System	Local subnet	TCP	5357
Network Discovery (WSD Events-In)	Public	No	System	Local subnet	TCP	5357
Network Discovery (WSD EventsSecure-In)	Domain	No	System	Any	TCP	5358
Network Discovery (WSD EventsSecure-In)	Private	Yes	System	Local subnet	TCP	5358
Network Discovery (WSD EventsSecure-In)	Public	No	System	Local subnet	TCP	5358
Network Discovery (WSD-In)	Domain	No	%SystemRoot%\system32\svchost.exe	Any	UDP	3702
Network Discovery (WSD-In)	Private	Yes	%SystemRoot%\system32\svchost.exe	Local subnet	UDP	3702
Network Discovery (WSD-In)	Public	No	%SystemRoot%\system32\svchost.exe	Local subnet	UDP	3702
Performance Logs and Alerts (DCOM-In)	Domain	No	%systemroot%\system32\svchost.exe	Any	TCP	135
Performance Logs and Alerts (DCOM-In)	Private, Public	No	%systemroot%\system32\svchost.exe	Local subnet	TCP	135
Performance Logs and Alerts (TCP-In)	Domain	No	%systemroot%\system32\plasrv.exe	Any	TCP	Any
Performance Logs and Alerts (TCP-In)	Private, Public	No	%systemroot%\system32\plasrv.exe	Local subnet	TCP	Any
Remote Administration (NP-In)	Domain	No	System	Any	TCP	445
Remote Administration (NP-In)	Private, Public	No	System	Local subnet	TCP	445
Remote Administration (RPC-EPMAP)	Domain	No	%SystemRoot%\system32\svchost.exe	Any	TCP	RPC Endpoint Mapper
Remote Administration (RPC-EPMAP)	Private, Public	No	%SystemRoot%\system32\svchost.exe	Local subnet	TCP	RPC Endpoint Mapper
Remote Administration (RPC)	Domain	No	%SystemRoot%\system32\svchost.exe	Any	TCP	Dynamic RPC
Remote Administration (RPC)	Private, Public	No	%SystemRoot%\system32\svchost.exe	Local subnet	TCP	Dynamic RPC
Remote Assistance (DCOM-In)	Domain	No	%SystemRoot%\system32\svchost.exe	Any	TCP	RPC Endpoint Mapper
Remote Assistance (RA Server TCP-In)	Domain	No	%SystemRoot%\system32\-raserver.exe	Any	TCP	Any
Remote Assistance (SSDP-In)	Domain	No	%SystemRoot%\system32\svchost.exe	Local subnet	UDP	1900
Remote Assistance (SSDP-In)	Private	Yes	%SystemRoot%\system32\svchost.exe	Local subnet	UDP	1900
Remote Assistance (TCP-In)	Domain, Public	No	%SystemRoot%\system32\msra.exe	Any	TCP	Any
Remote Assistance (TCP-In)	Private	Yes	%SystemRoot%\system32\msra.exe	Any	TCP	Any
Remote Assistance (UPnP-In)	Domain	No	System	Local subnet	TCP	2869
Remote Assistance (UPnP-In)	Private	Yes	System	Local subnet	TCP	2869
Remote Desktop (TCP-In)	Domain, Private, Public	No	System	Any	TCP	3389
Remote Event Log Management (NP-In)	Domain	No	System	Any	TCP	445
Remote Event Log Management (NP-In)	Private, Public	No	System	Local subnet	TCP	445
Remote Event Log Management (RPC-EPMAP)	Domain	No	%SystemRoot%\system32\svchost.exe	Any	TCP	RPC Endpoint Mapper
Remote Event Log Management (RPC-EPMAP)	Private, Public	No	%SystemRoot%\system32\svchost.exe	Local subnet	TCP	RPC Endpoint Mapper
Remote Event Log Management (RPC)	Domain	No	%SystemRoot%\system32\svchost.exe	Any	TCP	Dynamic RPC
Remote Event Log Management (RPC)	Private, Public	No	%SystemRoot%\system32\svchost.exe	Local subnet	TCP	Dynamic RPC

(Continued on next page)

Name	Profile	Enabled	Program	Remote Address	Protocol	Local Port
Remote Scheduled Tasks Management (RPC-EPMAP)	Domain	No	%SystemRoot%\system32\svchost.exe	Any	TCP	RPC Endpoint Mapper
Remote Scheduled Tasks Management (RPC-EPMAP)	Private, Public	No	%SystemRoot%\system32\svchost.exe	Local subnet	TCP	RPC Endpoint Mapper
Remote Scheduled Tasks Management (RPC)	Domain	No	%SystemRoot%\system32\svchost.exe	Any	TCP	Dynamic RPC
Remote Scheduled Tasks Management (RPC)	Private, Public	No	%SystemRoot%\system32\svchost.exe	Local subnet	TCP	Dynamic RPC
Remote Service Management (NP-In)	Domain	No	System	Any	TCP	445
Remote Service Management (NP-In)	Private, Public	No	System	Local subnet	TCP	445
Remote Service Management (RPC-EPMAP)	Domain	No	%SystemRoot%\system32\svchost.exe	Any	TCP	RPC Endpoint Mapper
Remote Service Management (RPC-EPMAP)	Private, Public	No	%SystemRoot%\system32\svchost.exe	Local subnet	TCP	RPC Endpoint Mapper
Remote Service Management (RPC)	Domain	No	%SystemRoot%\system32\-services.exe	Any	TCP	Dynamic RPC
Remote Service Management (RPC)	Private, Public	No	%SystemRoot%\system32\-services.exe	Local subnet	TCP	Dynamic RPC
Remote Volume Management - Virtual Disk Service (RPC)	Domain	No	%SystemRoot%\system32\vds.exe	Any	TCP	Dynamic RPC
Remote Volume Management - Virtual Disk Service (RPC)	Private, Public	No	%SystemRoot%\system32\vds.exe	Local subnet	TCP	Dynamic RPC
Remote Volume Management - Virtual Disk Service Loader (RPC)	Domain	No	%SystemRoot%\system32\vdsldr.exe	Any	TCP	Dynamic RPC
Remote Volume Management - Virtual Disk Service Loader (RPC)	Private, Public	No	%SystemRoot%\system32\vdsldr.exe	Local subnet	TCP	Dynamic RPC
Remote Volume Management (RPC-EPMAP)	Domain	No	%SystemRoot%\system32\svchost.exe	Any	TCP	RPC Endpoint Mapper
Remote Volume Management (RPC-EPMAP)	Private, Public	No	%SystemRoot%\system32\svchost.exe	Local subnet	TCP	RPC Endpoint Mapper
Routing and Remote Access (L2TP-In)	Domain, Private, Public	No	System	Any	UDP	1701
Routing and Remote Access (PPTP-In)	Domain, Private, Public	No	System	Any	TCP	1723
SNMP Trap Service (UDP In)	Domain	No	%SystemRoot%\system32\-snmptrap.exe	Any	UDP	162
SNMP Trap Service (UDP In)	Domain, Private, Public	No	%SystemRoot%\system32\-snmptrap.exe	Any	UDP	162
Windows Collaboration Computer Name Registration Service (PNRP-In)	Domain, Private, Public	No	%SystemRoot%\system32\svchost.exe	Any	UDP	3540
Windows Collaboration Computer Name Registration Service (SSDP-In)	Domain, Private, Public	No	%SystemRoot%\system32\svchost.exe	Local subnet	UDP	1900
Windows Firewall Remote Management (RPC-EPMAP)	Domain	No	%SystemRoot%\system32\svchost.exe	Any	TCP	RPC Endpoint Mapper
Windows Firewall Remote Management (RPC-EPMAP)	Private, Public	No	%SystemRoot%\system32\svchost.exe	Local subnet	TCP	RPC Endpoint Mapper
Windows Firewall Remote Management (RPC)	Domain	No	%SystemRoot%\system32\svchost.exe	Any	TCP	Dynamic RPC
Windows Firewall Remote Management (RPC)	Private, Public	No	%SystemRoot%\system32\svchost.exe	Local subnet	TCP	Dynamic RPC
Windows Management Instrumentation (ASync-In)	Domain	No	%systemroot%\system32\wbem\-unsecapp.exe	Any	TCP	Any
Windows Management Instrumentation (ASync-In)	Private, Public	No	%systemroot%\system32\wbem\-unsecapp.exe	Local subnet	TCP	Any
Windows Management Instrumentation (DCOM-In)	Domain	No	%SystemRoot%\system32\svchost.exe	Any	TCP	135
Windows Management Instrumentation (DCOM-In)	Private, Public	No	%SystemRoot%\system32\svchost.exe	Local subnet	TCP	135
Windows Management Instrumentation (WMI-In)	Domain	No	%SystemRoot%\system32\svchost.exe	Any	TCP	Any
Windows Management Instrumentation (WMI-In)	Private, Public	No	%SystemRoot%\system32\svchost.exe	Local subnet	TCP	Any
Windows Media Player (UDP-In)	Domain, Private, Public	No	%ProgramFiles%\Windows Media Player\wmplayer.exe	Any	UDP	Any
Windows Media Player Network Sharing Service (HTTP-Streaming-In)	Domain	No	System	Any	TCP	10243
Windows Media Player Network Sharing Service (HTTP-Streaming-In)	Private, Public	No	System	Local subnet	TCP	10243
Windows Media Player Network Sharing Service (qWave-TCP-In)	Domain	No	%SystemRoot%\system32\svchost.exe	Any	TCP	2177
Windows Media Player Network Sharing Service (qWave-TCP-In)	Private, Public	No	%SystemRoot%\system32\svchost.exe	Local subnet	TCP	2177
Windows Media Player Network Sharing Service (qWave-UDP-In)	Domain	No	%SystemRoot%\system32\svchost.exe	Any	UDP	2177

(Continued on next page)

Name	Profile	Enabled	Program	Remote Address	Protocol	Local Port
Windows Media Player Network Sharing Service (qWave-UDP-In)	Private, Public	No	%SystemRoot%\system32\svchost.exe	Local subnet	UDP	2177
Windows Media Player Network Sharing Service (SSDP-In)	Domain, Private, Public	No	%SystemRoot%\system32\svchost.exe	Local subnet	UDP	1900
Windows Media Player Network Sharing Service (Streaming-UDP-In)	Domain	No	%ProgramFiles%\Windows Media Player\wmplayer.exe	Any	UDP	Any
Windows Media Player Network Sharing Service (Streaming-UDP-In)	Private, Public	No	%ProgramFiles%\Windows Media Player\wmplayer.exe	Local subnet	UDP	Any
Windows Media Player Network Sharing Service (TCP-In)	Domain	No	%ProgramFiles%\Windows Media Player\wmpnetwk.exe	Any	TCP	Any
Windows Media Player Network Sharing Service (TCP-In)	Private, Public	No	%ProgramFiles%\Windows Media Player\wmpnetwk.exe	Local subnet	TCP	Any
Windows Media Player Network Sharing Service (UDP-In)	Domain	No	%ProgramFiles%\Windows Media Player\wmpnetwk.exe	Any	UDP	Any
Windows Media Player Network Sharing Service (UDP-In)	Private, Public	No	%ProgramFiles%\Windows Media Player\wmpnetwk.exe	Local subnet	UDP	Any
Windows Media Player Network Sharing Service (UPnP-In)	Domain, Private, Public	No	System	Local subnet	TCP	2869
Windows Meeting Space (DFSR-In)	Domain, Private, Public	No	%SystemRoot%\system32\dfsr.exe	Any	TCP	5722
Windows Meeting Space (P2P-In)	Domain, Private, Public	No	%SystemRoot%\system32\svchost.exe	Any	TCP	3587
Windows Meeting Space (TCP-In)	Domain, Private, Public	No	%ProgramFiles%\Windows Collaboration\WinCollab.exe	Any	TCP	Any
Windows Meeting Space (UDP-In)	Domain, Private, Public	No	%ProgramFiles%\Windows Collaboration\WinCollab.exe	Any	UDP	Any
Windows Peer to Peer Collaboration Foundation (PNRP-In)	Domain, Private, Public	No	%SystemRoot%\system32\svchost.exe	Any	UDP	3540
Windows Peer to Peer Collaboration Foundation (SSDP-In)	Domain, Private, Public	No	%SystemRoot%\system32\svchost.exe	Local subnet	UDP	1900
Windows Peer to Peer Collaboration Foundation (TCP-In)	Domain, Private, Public	No	%SystemRoot%\system32\-p2phost.exe	Any	TCP	Any
Windows Peer to Peer Collaboration Foundation (WSD-In)	Domain, Private, Public	No	%SystemRoot%\system32\-p2phost.exe	Local subnet	UDP	3702
Windows Remote Management (HTTP-In)	Domain	No	System	Any	TCP	80
Windows Remote Management (HTTP-In)	Private, Public	No	System	Local subnet	TCP	80
Wireless Portable Devices (SSDP-In)	Domain, Private, Public	No	%SystemRoot%\system32\svchost.exe	Local subnet	UDP	1900
Wireless Portable Devices (UPnP-In)	Domain, Private, Public	No	System	Local subnet	TCP	2869

The group-profiles that are initially enabled are the following: Core Networking group (all profiles), Network Discovery group (private profile only), and Remote Assistance (private profile only). Core networking includes Teredo (allowed for local port=Edge Traversal), so it is enabled by default.

We observe that all TCP and UDP firewall rules that have the remote port Any have a specific program to which they are bound, thus apparently limiting exposure. The other listed protocols (ICMPv4, ICMPv6, IPv6, and IGMP) rules have Program=System except for two ping-related rules, which have Program=Any. Many—but not all—rules with a given name are represented in all three profiles. Specific Remote Assistance rules are only present for domain or for domain and private. There is one group-profile that is duplicated in the ruleset: SNMP Trap Service for private; this seems to be a bug.

There are several named rules that are bound to Remote Address=Local subnet for private and public, but which are bound to Remote Address=Any for domain. Thus in a domain, for these cases, the host is exposed to packets from outside the subnet (potentially including the Internet).

The initial netstat output is shown in Figure 48. Except for IP addresses and PIDs, this was the same as a netstat list on a different Vista that was taken first thing after an installation. That output remained unchanged when we checked back later, after we turned off the firewall, and after we turned the firewall back on.

C. Firewall Changes with Configuration Changes

We studied how the firewall inbound ruleset changed when we made different changes to the system. We were in the private profile when doing these tests. Our results are summarized in Figure 3 on page 12.

One surprising result, noted in many cases, is that firewall rules are not disabled upon turning off the Vista function that caused them to be enabled. The exceptions even persist across a system restart; thus, until they are manually disabled, a legacy of firewall exceptions accumulates on a system. The following describes the possible negative effects:

- A malicious application could communicate through the exception without a consent prompt

Proto	Local Address	Local Port	Foreign Address	For. Port	State	PID	Components	Owning Process
TCP	0.0.0.0	135	0.0.0.0	0	LISTENING	796	RpcSs	[svchost.exe]
TCP	0.0.0.0	49152	0.0.0.0	0	LISTENING	476		[wininit.exe]
TCP	0.0.0.0	49153	0.0.0.0	0	LISTENING	952	Eventlog	[svchost.exe]
TCP	0.0.0.0	49154	0.0.0.0	0	LISTENING	1192	nsi	[svchost.exe]
TCP	0.0.0.0	49155	0.0.0.0	0	LISTENING	1008	Schedule	[svchost.exe]
TCP	0.0.0.0	49156	0.0.0.0	0	LISTENING	564		[lsass.exe]
TCP	0.0.0.0	49157	0.0.0.0	0	LISTENING	544		[services.exe]
TCP	192.168.0.204	139	0.0.0.0	0	LISTENING	4		Can not obtain ownership information
TCP	[::]	135	[::]	0	LISTENING	796	RpcSs	[svchost.exe]
TCP	[::]	445	[::]	0	LISTENING	4		Can not obtain ownership information
TCP	[::]	5357	[::]	0	LISTENING	4		Can not obtain ownership information
TCP	[::]	49152	[::]	0	LISTENING	476		[wininit.exe]
TCP	[::]	49153	[::]	0	LISTENING	952	Eventlog	[svchost.exe]
TCP	[::]	49154	[::]	0	LISTENING	1192	nsi	[svchost.exe]
TCP	[::]	49155	[::]	0	LISTENING	1008	Schedule	[svchost.exe]
TCP	[::]	49156	[::]	0	LISTENING	564		[lsass.exe]
TCP	[::]	49157	[::]	0	LISTENING	544		[services.exe]
UDP	0.0.0.0	123	*	*		1192	W32Time	[svchost.exe]
UDP	0.0.0.0	500	*	*		1008	IKEEXT	[svchost.exe]
UDP	0.0.0.0	3702	*	*		1192	FDResPub	[svchost.exe]
UDP	0.0.0.0	3702	*	*		1192	EventSystem	[svchost.exe]
UDP	0.0.0.0	3702	*	*		1192	FDResPub	[svchost.exe]
UDP	0.0.0.0	3702	*	*		1192	EventSystem	[svchost.exe]
UDP	0.0.0.0	4500	*	*		1008	IKEEXT	[svchost.exe]
UDP	0.0.0.0	5355	*	*		1292	Dnscache	[svchost.exe]
UDP	0.0.0.0	49181	*	*		1192	FDResPub	[svchost.exe]
UDP	0.0.0.0	49187	*	*		1192	EventSystem	[svchost.exe]
UDP	127.0.0.1	1900	*	*		1192	SSDPSRV	[svchost.exe]
UDP	127.0.0.1	49180	*	*		1192	SSDPSRV	[svchost.exe]
UDP	192.168.0.204	137	*	*		4		Can not obtain ownership information
UDP	192.168.0.204	138	*	*		4		Can not obtain ownership information
UDP	192.168.0.204	1900	*	*		1192	SSDPSRV	[svchost.exe]
UDP	192.168.0.204	49179	*	*		1192	SSDPSRV	[svchost.exe]
UDP	[::]	123	*	*		1192	W32Time	[svchost.exe]
UDP	[::]	500	*	*		1008	IKEEXT	[svchost.exe]
UDP	[::]	3702	*	*		1192	EventSystem	[svchost.exe]
UDP	[::]	3702	*	*		1192	FDResPub	[svchost.exe]
UDP	[::]	3702	*	*		1192	EventSystem	[svchost.exe]
UDP	[::]	3702	*	*		1192	FDResPub	[svchost.exe]
UDP	[::]	5355	*	*		1292	Dnscache	[svchost.exe]
UDP	[::]	49182	*	*		1192	FDResPub	[svchost.exe]
UDP	[::]	49188	*	*		1192	EventSystem	[svchost.exe]
UDP	[::1]	1900	*	*		1192	SSDPSRV	[svchost.exe]
UDP	[::1]	49177	*	*		1192	SSDPSRV	[svchost.exe]
UDP	[fe80::100:7f:fffe%9]	1900	*	*		1192	SSDPSRV	[svchost.exe]
UDP	[fe80::100:7f:fffe%9]	49178	*	*		1192	SSDPSRV	[svchost.exe]
UDP	[fe80::f426:13fe:e8e7:720c%8]	1900	*	*		1192	SSDPSRV	[svchost.exe]
UDP	[fe80::f426:13fe:e8e7:720c%8]	49176	*	*		1192	SSDPSRV	[svchost.exe]

Fig. 48. Initial netstat -a -b -n -o output, converted into a table. IPv6 addresses are listed inside square brackets. "[::]" represents the IPv6 unassigned address, analogous to 0.0.0.0 in IPv4 and "[::1]" is the IPv6 loopback address. 192.168.0.204 and ffe80::f426:13fe:e8e7:720c are addresses assigned to the host the testing was done on (vmvista2). Although netstat did not report it, PID 4 is System.

- If a legitimate program that was using the exception remained active (or if another legitimate program that could use the same exception was running), then that program could be reached
- If the stack itself were providing the service for the exception, then the stack could be reached (e.g. echo replies could continue to be sent)

We refer to this phenomenon as "sticky" rules.

1) Sharing and Discovery Controls: We explored making changes to settings in the "Sharing and Discovery" section of the "Network and Sharing Center" control panel. This control panel is shown in Figure 15 on page 25.

We found that turning on File Sharing or Public Folder Sharing caused all private profile in the File and Printer Sharing group to become enabled. The exceptions associated with this are the following:

+	File and Printer Sharing (Spooler Service - RPC-EPMAP)	TCP	RPC Endpoint Mapper	
	File and Printer Sharing (Spooler Service - RPC)	TCP	Dynamic RPC	spoolsv.exe
	File and Printer Sharing (SMB-In)	TCP	445	
	File and Printer Sharing (NB-Session-In)	TCP	139	
	File and Printer Sharing (NB-Name-In)	UDP	137	
	File and Printer Sharing (NB-Datagram-In)	UDP	138	
	File and Printer Sharing (Echo Request - ICMPv6-In)	ICMPv6		
	File and Printer Sharing (Echo Request - ICMPv4-In)	ICMPv4		

These were turned off when file sharing was turned off. However, if media sharing was also active, they were turned off when media sharing was turned off.

Activating Media Sharing with File Sharing already active caused domain and private rules for the Windows Media Player Network Sharing Service and Windows Media Player groups to become active. Those are the following:

+	Windows Media Player Network Sharing Service (UPnP-In)	TCP	2869	
	Windows Media Player Network Sharing Service (UDP-In)	UDP	Any	wmpnetwk.exe
	Windows Media Player Network Sharing Service (TCP-In)	TCP	Any	wmpnetwk.exe
	Windows Media Player Network Sharing Service (Streaming-UDP-In)	UDP	Any	wmplayer.exe
	Windows Media Player Network Sharing Service (SSDP-In)	UDP	1900	
	Windows Media Player Network Sharing Service (qWave-UDP-In)	UDP	2177	
	Windows Media Player Network Sharing Service (qWave-TCP-In)	TCP	2177	
	Windows Media Player Network Sharing Service (HTTP-Streaming-In)	TCP	10243	
	Windows Media Player (UDP-In)	UDP	Any	wmplayer.exe

Presumably Media Sharing would also turn on the File and Printer Sharing group as well. The group Windows Media Player Network Sharing Service was turned off when Media Sharing was disabled, however we never saw the Windows Media Player group become disabled (i.e. it was sticky).

The private rules for Network Discovery group are enabled out-of-the-box. We found they could be turned off by disabling Network Discovery in this control panel:

−	Network Discovery (WSD-In)	UDP	3702	svchost.exe
	Network Discovery (WSD EventsSecure-In)	TCP	5358	System
	Network Discovery (WSD Events-In)	TCP	5357	System
	Network Discovery (UPnP-In)	TCP	2869	System
	Network Discovery (SSDP-In)	UDP	1900	svchost.exe
	Network Discovery (Pub-WSD-In)	UDP	3702	svchost.exe
	Network Discovery (NB-Name-In)	UDP	137	System
	Network Discovery (NB-Datagram-In)	UDP	138	System
	Network Discovery (LLMNR-UDP-In)	UDP	5355	svchost.exe

2) People Near Me: Setting up a People Near Me (PNM) profile (e.g., establishing a username and preferences) did not cause any firewall changes. However, signing in caused the Windows Peer to Peer Collaboration Foundation group to become enabled for all profiles:

+	Windows Peer to Peer Collaboration Foundation (WSD-In)	UDP	3702	p2phost.exe
	Windows Peer to Peer Collaboration Foundation (TCP-In)	TCP	Any	p2phost.exe
	Windows Peer to Peer Collaboration Foundation (SSDP-In)	UDP	1900	svchost.exe
	Windows Peer to Peer Collaboration Foundation (PNRP-In)	UDP	3540	svchost.exe

We never saw this group become disabled, even after signing out of PNM and closing the window.

3) Windows Meeting Space: Windows Meeting Space (WMS), referred to as Windows Collaboration in beta builds, requires PNM to be active so the above firewall changes apply to WMS as well. In addition, setting up WMS caused the Windows Meeting Space and Connect to a Network Projector groups to become enabled for all profiles:

+	Windows Meeting Space (UDP-In)	UDP	Any	WinCollab.exe
	Windows Meeting Space (TCP-In)	TCP	Any	WinCollab.exe
	Windows Meeting Space (P2P-In)	TCP	3587	svchost.exe
	Windows Meeting Space (DFSR-In)	TCP	5722	dfsr.exe
	Connect to a Network Projector (WSD-In)	UDP	3702	netproj.exe
	Connect to a Network Projector (WSD EventsSecure-In)	TCP	5358	
	Connect to a Network Projector (WSD Events-In)	TCP	5357	
	Connect to a Network Projector (TCP-In)	TCP	Any	netproj.exe

We never saw these groups become disabled, even after ending WMS, signing out of PNM, and rebooting. Creating and later leaving a WMS meeting had no effect on the firewall.

D. Active Socket Changes with Configuration Changes

We examined the effect on active sockets (specifically, the netstat report on active sockets) as a result of various configuration changes. We ignored differences in netstat's ordering of the output, but considered any differences between compared lines significant. Since the observations are made at a single point in time (usually soon after making the configuration change), we sometimes catch transient connections. We include those here, but there are likely others we did not see; this testing was focused on persistent changes.

The sequences of changes described in each of the subsections below have essentially the same initial state. This was achieved through the use of VMware snapshots—each sequence began by reverting back to a snapshot representing the initial state. The netstat output has been reformatted for readability.

UDP port 3702 is seen frequently; this corresponds to UPnPv2 and web services discovery (see [9]).

1) File Sharing: We turned on File Sharing. Since most of the associated processes are running already (see PID 4 in Figure 48), the only netstat addition is as follows:

+	TCP	192.168.0.204:49164	→ 192.168.0.203:139	SYN_SENT	PID=4 [System]

This was later replaced with the following:

⋈	TCP	192.168.0.204:49165	→ 192.168.0.203:139	SYN_SENT	PID=4 [System]

Hence this host is attempting to establish a NetBIOS session connection to another Vista host on the network (vmvista=192.168.0.203).

The entry disappeared after turning off file sharing. Thus turning on file sharing had no side-effects.

2) Sharing and Discovery Controls: For this sequence, we further explored changing the settings in the Sharing and Discovery section of the Network and Sharing Center control panel (see Figure 15, page 25).

After turning on Public Folder Sharing with public write access, the following new entries were observed in netstat:

+	TCP	192.168.0.204:5357	→ 192.168.0.203:49176	ESTABLISHED	PID=4 [System]
	TCP	192.168.0.204:49163	→ 192.168.0.203:5357	ESTABLISHED	PID=1192 [svchost.exe:EventSystem]
	UDP	0.0.0.0:49251	→ *:*		PID=1192 [svchost.exe:EventSystem]
	UDP	[::]:49252	→ *:*		PID=1192 [svchost.exe:EventSystem]

After turning on media sharing, two of the above were gone:

−	TCP	192.168.0.204:5357	→ 192.168.0.203:49176	ESTABLISHED	PID=4 [System]
	TCP	192.168.0.204:49163	→ 192.168.0.203:5357	ESTABLISHED	PID=1192 [svchost.exe:EventSystem]

We believe these were transient connections that had completed by that point in time. Several new additions appeared from media sharing:

+	TCP	0.0.0.0:554	→ 0.0.0.0:0	LISTENING	PID=2824 [wmpnetwk.exe]
	TCP	[::]:554	→ [::]:0	LISTENING	PID=2824 [wmpnetwk.exe]
	UDP	0.0.0.0:5004	→ *:*		PID=2824 [wmpnetwk.exe]
	UDP	0.0.0.0:5005	→ *:*		PID=2824 [wmpnetwk.exe]
	UDP	127.0.0.1:49268	→ *:*		PID=2824 [wmpnetwk.exe]
	UDP	[::]:5004	→ *:*		PID=2824 [wmpnetwk.exe]
	UDP	[::]:5005	→ *:*		PID=2824 [wmpnetwk.exe]
	TCP	127.0.0.1:49168	→ 127.0.0.1:2869	ESTABLISHED	PID=1192 [svchost.exe:EventSystem]
	TCP	127.0.0.1:2869	→ 127.0.0.1:49168	ESTABLISHED	PID=4 [System]
	TCP	192.168.0.204:2869	→ 192.168.0.203:49178	ESTABLISHED	PID=4 [System]
	TCP	[::]:2869	→ [::]:0	LISTENING	PID=4 [System]
	TCP	[::]:10243	→ [::]:0	LISTENING	PID=4 [System]

File Sharing had become enabled as a result of one of the above. We turned that off. The only changes seen from this are apparently coincidental removals:

−	TCP	127.0.0.1:2869	→ 127.0.0.1:49168	ESTABLISHED	PID=4 [System]
	TCP	192.168.0.204:2869	→ 192.168.0.203:49178	ESTABLISHED	PID=4 [System]
	TCP	127.0.0.1:49168	→ 127.0.0.1:2869	ESTABLISHED	PID=1192 [svchost.exe:EventSystem]

We then turned off media sharing and many of the additions caused by turning it on disappeared:

	TCP	0.0.0.0:554	→ 0.0.0.0:0	LISTENING	PID=2824 [wmpnetwk.exe]
	TCP	[::]:554	→ [::]:0	LISTENING	PID=2824 [wmpnetwk.exe]
	UDP	0.0.0.0:5004	→ *:*		PID=2824 [wmpnetwk.exe]
	UDP	0.0.0.0:5005	→ *:*		PID=2824 [wmpnetwk.exe]
−	UDP	127.0.0.1:49268	→ *:*		PID=2824 [wmpnetwk.exe]
	UDP	[::]:5004	→ *:*		PID=2824 [wmpnetwk.exe]
	UDP	[::]:5005	→ *:*		PID=2824 [wmpnetwk.exe]
	TCP	[::]:2869	→ [::]:0	LISTENING	PID=4 [System]
	TCP	[::]:10243	→ [::]:0	LISTENING	PID=4 [System]

Only two entries remained that were not in the initial state. These entries came as a result of enabling public folder sharing.

Δ	UDP	0.0.0.0:49251	→ *:*	PID=1192 [svchost.exe:EventSystem]
	UDP	[::]:49252	→ *:*	PID=1192 [svchost.exe:EventSystem]

Turning off password-protected sharing and network discovery from the control panel had no effect on the netstat listing. However, from Appendix III-B we know that disabling network discovery causes LLTD to be disabled.

3) People Near Me: Opening up the People Near Me (PNM) application and setting up a username, etc, did not have any effect on the netstat listing. However, signing into it did, adding the following IPv6 entries:

	TCP	[::]:49163	→ [::]:0	LISTENING	PID=3564 [p2phost.exe]
	UDP	[::]:3702	→ *:*		PID=3564 [p2phost.exe]
	UDP	[::]:3702	→ *:*		PID=3564 [p2phost.exe]
+	UDP	[::]:3702	→ *:*		PID=3564 [p2phost.exe]
	UDP	[::]:3702	→ *:*		PID=3564 [p2phost.exe]
	UDP	[::]:49251	→ *:*		PID=3564 [p2phost.exe]
	UDP	[::]:49252	→ *:*		PID=3564 [p2phost.exe]

Signing out of PNM caused those entries to disappear though (even though the firewall rules remained enabled).

4) Windows Meeting Space: Opening and running Windows Meeting Space (WMS) requires PNM to be set up and signed in to. As a result of turning on WMS and PNM, the following additions were seen in netstat:

	TCP	127.0.0.1:49164	→ 127.0.0.1:*		PID=2616 [WinCollab.exe]
	UDP	[::]:3702	→ *:*		PID=2616 [WinCollab.exe]
	UDP	[::]:3702	→ *:*		PID=2616 [WinCollab.exe]
	TCP	[::]:49163	→ [::]:0	LISTENING	PID=3520 [p2phost.exe]
+	UDP	[::]:3702	→ *:*		PID=3520 [p2phost.exe]
	UDP	[::]:3702	→ *:*		PID=3520 [p2phost.exe]
	UDP	[::]:3702	→ *:*		PID=3520 [p2phost.exe]
	UDP	[::]:3702	→ *:*		PID=3520 [p2phost.exe]
	UDP	[::]:49251	→ *:*		PID=3520 [p2phost.exe]
	UDP	[::]:49252	→ *:*		PID=3520 [p2phost.exe]

Aside from a difference in the process ID, the p2phost.exe entries exactly match the result from the above sequence, in which PNM was enabled by itself. The WinCollab.exe entries correspond to Windows Meeting Space.

Creating a meeting in WMS caused a variety of additions to the active ports:

	TCP	127.0.0.1:49165	→ 127.0.0.1:49156	ESTABLISHED	PID=2616 [WinCollab.exe]
	TCP	127.0.0.1:49156	→ 127.0.0.1:49165	ESTABLISHED	PID=564 [lsass.exe]
	TCP	127.0.0.1:49164	→ 127.0.0.1:135	ESTABLISHED	PID=2616 [WinCollab.exe]
	TCP	127.0.0.1:135	→ 127.0.0.1:49164	ESTABLISHED	PID=796 [svchost.exe:RpcSs]
	UDP	[::]:3702	→ *:*		PID=2616 [WinCollab.exe]
+	UDP	[::]:3702	→ *:*		PID=2616 [WinCollab.exe]
	UDP	[::]:49254	→ *:*		PID=2616 [WinCollab.exe]
	TCP	[::]:3587	→ [::]:0	LISTENING	PID=952 [svchost.exe]
	UDP	[::]:3540	→ *:*		PID=952 [svchost.exe]
	TCP	0.0.0.0:5722	→ 0.0.0.0:0	LISTENING	PID=2840 [DFSR.exe]
	TCP	[::]:5722	→ [::]:0	LISTENING	PID=2840 [DFSR.exe]

The top four entries correspond to two connections from WMS to services on the same host. The DFSR.exe entries correspond to Distributed File System Replication (keeping files and folders synced among different hosts) for IPv4 and IPv6.

On departing that one-participant meeting, several entries were removed. These include the four localhost entries, though they were represented in the TIME_WAIT state by two entries:

⋈

TCP	127.0.0.1:49164 → 127.0.0.1:135		TIME_WAIT	PID=0
TCP	127.0.0.1:49165 → 127.0.0.1:49156		TIME_WAIT	PID=0

The following entries were also removed:

−

UDP	[::]:3702 → *:*		PID=2616 [WinCollab.exe]
UDP	[::]:3702 → *:*		PID=2616 [WinCollab.exe]
UDP	[::]:49254 → *:*		PID=2616 [WinCollab.exe]
TCP	[::]:3587 → [::]:0	LISTENING	PID=952 [svchost.exe]

We closed WMS and ran netstat again. The TIME_WAIT entries were gone by that point, as were the remaining WinCollab entries:

−

UDP	[::]:3702 → *:*	PID=2616 [WinCollab.exe]
UDP	[::]:3702 → *:*	PID=2616 [WinCollab.exe]
UDP	[::]:49253 → *:*	PID=2616 [WinCollab.exe]

Although PNM was still running at this point, we expected all the sockets associated with WMS to be closed. However, the following shows the difference from the initial state:

Δ

UDP	[::]:3540 → *:*			PID=952 [svchost.exe]
TCP	0.0.0.0:5722 → 0.0.0.0:0		LISTENING	PID=2840 [DFSR.exe]
TCP	[::]:5722 → [::]:0		LISTENING	PID=2840 [DFSR.exe]
TCP	[::]:49163 → [::]:0		LISTENING	PID=3520 [p2phost.exe]
UDP	[::]:3702 → *:*			PID=3520 [p2phost.exe]
UDP	[::]:3702 → *:*			PID=3520 [p2phost.exe]
UDP	[::]:3702 → *:*			PID=3520 [p2phost.exe]
UDP	[::]:3702 → *:*			PID=3520 [p2phost.exe]
UDP	[::]:49251 → *:*			PID=3520 [p2phost.exe]
UDP	[::]:49252 → *:*			PID=3520 [p2phost.exe]

The first three entries appear to be leftovers. However, these three sockets were not restarted after a reboot. The continued external exposure of the three depends on the firewall state. The firewall entries for UDP port 3540 (PNRP-In for either Windows Peer to Peer Collaboration Foundation or Windows Collaboration Computer Name Registration Service) were not enabled, so this port is not exposed. On the other hand, the firewall entry for DFSR (“Windows Meeting Space (DFSR-In)”) was sticky (remained enabled), so the process was still remotely accessible over IPv4 and IPv6. However, this extra TCP port 5722 exposure was short lived. The sockets went away in a couple minutes. The socket for UDP 3540 also went away by itself, though it took longer.

Signing out of PNM, disabling auto-login, and closing the window caused all the p2phost.exe entries to disappear, which matched our result from the previous sequence. With this, all the netstat additions that were due to this sequence have been removed.

APPENDIX XXII
EXPOSED TCP SERVICES

We used traditional port scanning techniques to identify exposed TCP services running on a default Windows Vista installation and with the network set to the private profile, both over IPv4 and over IPv6. This was done from the same subnet as the scanned host.

For IPv4, we employed Nmap[18] to scan Vista with the firewall enabled (the default) and with the network set as a private network:

```
linux# nmap -P0 -r -sS -p0-65535 $acerIP4

Starting Nmap 4.10 ( http://www.insecure.org/nmap/ ) at 2006-11-10 14:07 PST
Interesting ports on 192.168.0.200:
Not shown: 65535 filtered ports
PORT     STATE SERVICE
5357/tcp open  unknown
MAC Address: 00:C0:9F:D2:0C:F8 (Quanta Computer)

Nmap finished: 1 IP address (1 host up) scanned in 382.610 seconds
```

Thus almost all ports are filtered (produce no response), but one port is open (5357).

Port 5357 corresponds to Web Services on Devices (WSD):

```
wsdapi         5357/tcp   Web Services for Devices
wsdapi         5357/udp   Web Services for Devices
```

WSD[39] is a solution from Microsoft which forms part of the Windows Rally set of technologies, which includes LLTD (Appendix II). The open port is owned by the kernel driver HTTP.SYS.

To avoid filtering, we ran Nmap again on a host which had the firewall disabled.

```
linux# nmap -P0 -r -sS -p0-65535 $hpIP4

Starting Nmap 4.10 ( http://www.insecure.org/nmap/ ) at 2006-11-10 14:34 PST
Interesting ports on 192.168.0.201:
Not shown: 65526 closed ports
PORT      STATE SERVICE
135/tcp   open  msrpc
139/tcp   open  netbios-ssn
445/tcp   open  microsoft-ds
5357/tcp  open  unknown
49152/tcp open  unknown
49153/tcp open  unknown
49154/tcp open  unknown
49155/tcp open  unknown
49156/tcp open  unknown
49157/tcp open  unknown
MAC Address: 00:14:C2:D5:7E:96 (Hewlett Packard)

Nmap finished: 1 IP address (1 host up) scanned in 91.708 seconds
```

In this case, a RST is produced for closed ports. From Appendix XXIV we know that port 135 and the six ephemeral ports are used by RPC; that is also what nmap -sV reports. This is consistent with netstat reports on a similarly unmodified Vista host (Figure 48), except that port 5357 is not listed there for IPv4.

Similar results are observed when using IPv6. This test was performed using a custom-written tcpscan utility, which works with both IPv4 and IPv6 addresses. (Using it for IPv4 produced results that match the above.) The following results are observed with the firewall on:

```
linux# ../tcpscan -p 0-65535 $acerLL6%2
0 Connection timed out
1 Connection timed out
...
5356 Connection timed out
5357 open
5358 Connection timed out
...
65535 Connection timed out
```

The results show that most ports are filtered (i.e. the connection timed out) and port 5357 is the only open port.

The following results are observed on a host with the firewall off (excluding timeouts):

```
linux# ../tcpscan -p 0-65535 $hpLL6%2
135 open
445 open
5357 open
49152 open
49153 open
49154 open
49155 open
49156 open
49157 open
```

The same results are obtained as with IPv4, but port 139 is absent. These results match the netstat results for TCP over IPv6 (Figure 48).

Figure 4 (page 13) summarizes the findings.

APPENDIX XXIII
EXPOSED UDP SERVICES

We used traditional port scanning techniques to identify exposed UDP services running on a default Windows Vista installation (with the network profile set to private), using both IPv4 and IPv6 as transport. This was done from the same subnet as the scanned host.

We used a custom script to scan UDP ports when the firewall was on. Using this for both IPv4 and IPv6, the script reported "open or filtered" for all ports. Clearly, the firewall is filtering all closed (unused) ports rather than allowing an ICMP port unreachable message to be sent out. Access to ports that are active on the system may also be filtered by the firewall. Thus, the effect of the firewall is disallow mapping of UDP ports, at least in a protocol-independent manner.

However we note that in order to have a port open through the firewall, there must be both an active socket on that port and a firewall exception covering the port. Using the initial netstat output and initial firewall rule settings in Appendix XXI-B, it would seem that the only ports that could be opened through the firewall are:

- 137 (NetBIOS name service, IPv4 only)
- 138 (NetBIOS datagram)
- 3702 (Web Services Discovery)
- 5355 (LLMNR)

Significant caveats with this conclusion are the assumptions that:

- Vista works as it apparently should.
- The data that is being reported by netstat is representative, accurate, and complete.
- The data reported for the Windows Firewall settings is accurate and complete.

Based on firewall exceptions that are initially in place, if a process for DHCP, SSDP, or Teredo were to open its corresponding port (68, 1900, and ephemeral, respectively), these ports could be open through the firewall, as well.

The following results were obtained for IPv4, with the firewall off:

```
linux# ../udpscan -p 0-65535 $hpIP4
123 opened or filtered
137 opened or filtered
138 opened or filtered
500 opened or filtered
1900 opened or filtered
3702 opened or filtered
4500 opened or filtered
5355 opened or filtered
49191 opened or filtered
49193 opened or filtered
49195 opened or filtered
49199 opened or filtered
```

Thus, the majority of ports produced an ICMP port unreachable message. The above listed ports produced no port unreachable message; since there is no firewall to do filtering, these are likely open. These ports represent both client and server use of ports. We noticed that the ephemeral ports varied in quantities and location between runs; these likely represent clients. The following results were obtained for IPv6, with the firewall off:

```
linux# ../udpscan -p 0-65535 $hpLL6
123 opened or filtered
500 opened or filtered
1900 opened or filtered
3702 opened or filtered
5355 opened or filtered
49189 opened or filtered
49194 opened or filtered
49196 opened or filtered
49200 opened or filtered
```

Many of the port numbers are the same, but the well-known ports are a subset of those from IPv4. As with TCP, the results show corresponding IPv4 and IPv6 use of a port, although the same ephemeral port number is not used for both. (The above scans were of the same host and relatively close together in time.)

Aside from the specific ephemeral ports used, these results correspond exactly to the initial netstat output shown in Figure 48 (page 84).

Figure 5 (page 13) summarizes the findings.

APPENDIX XXIV
RPC ENDPOINT MAPPER ENUMERATION

The RPC endpoint mapper service is not available by default in Vista installations. However, it is available if file sharing is turned on. We walked through the information made available by endpoint mapper, using a script. We obtained the following results:

```
linux$ epdump.py -T -p 135 $vmvista
06bba54a-be05-49f9-b0a0-30f790261023 1.0: Security Center
  LPC: AudioClientRpc
  LPC: Audiosrv
  LPC: OLE24E5DF29D091443BA08029561147
  LPC: dhcpcsvc
  LPC: dhcpcsvc6
  LPC: eventlog
  path: \pipe\eventlog
  tcp 49153

0767a036-0d22-48aa-ba69-b619480f38cb 1.0: PcaSvc
  LPC: LRPC-8c22bd6a4265c55669
  LPC: OLEBBDBE02F4D97481B8CC9D12DD894

0a74ef1c-41a4-4e06-83ae-dc74fb1cdd53 1.0:
  LPC: IUserProfile2
  LPC: OLE07A55C606630488F9A1CE31ACF85
  LPC: senssvc

0b6edbfa-4a24-4fc6-8a23-942b1eca65d1 1.0: Spooler function endpoint
  LPC: spoolss

12345678-1234-abcd-ef00-0123456789ab 1.0: IPSec Policy agent endpoint
  LPC: LRPC-d509414700694d948e

12345778-1234-abcd-ef00-0123456789ac 1.0:
  LPC: LRPC-44cba3e606f899328a
  LPC: audit
  LPC: protected_storage
  LPC: samss lpc
  LPC: securityevent
  path: \PIPE\protected_storage
  path: \pipe\lsass
  tcp 49154

12e65dd8-887f-41ef-91bf-8d816c42c2e7 1.0: Secure Desktop LRPC interface
  LPC: WMsgKRpc090901

1ff70682-0a51-30e8-076d-740be8cee98b 1.0:
  LPC: IUserProfile2
  LPC: OLE07A55C606630488F9A1CE31ACF85
  LPC: senssvc
  path: \PIPE\atsvc

201ef99a-7fa0-444c-9399-19ba84f12a1a 1.0: AppInfo
  LPC: IUserProfile2
  LPC: OLE07A55C606630488F9A1CE31ACF85
  LPC: RasmanRpc
  LPC: SECLOGON
  LPC: senssvc
  path: \PIPE\ROUTER
  path: \PIPE\atsvc
  path: \PIPE\srvsvc
  tcp 49156

2eb08e3e-639f-4fba-97b1-14f878961076 1.0:
  LPC: IUserProfile2

2f5f6521-cb55-1059-b446-00df0bce31db 1.0: Unimodem LRPC Endpoint
  LPC: DNSResolver
  LPC: OLEADBE270CE7AB49E78322DD993A07
  LPC: keysvc
```

```
  LPC: keysvc2
  LPC: nlaapi
  LPC: nlaplg
  LPC: tapsrvlpc
  LPC: unimdmsvc
  path: \pipe\keysvc
  path: \pipe\tapsrv

2fb92682-6599-42dc-ae13-bd2ca89bd11c 1.0: Fw APIs
  LPC: LRPC-a7d7a7c48531b65290

3473dd4d-2e88-4006-9cba-22570909dd10 5.0: WinHttp Auto-Proxy Service
  LPC: LRPC-ba6a873927fc322d90
  LPC: OLE2C02BDEDDF2B45CFBB2DCAA76F31
  LPC: W32TIME_ALT
  path: \PIPE\DAV RPC SERVICE
  path: \PIPE\W32TIME_ALT
  path: \PIPE\wkssvc
  tcp 49155

367abb81-9844-35f1-ad32-98f038001003 2.0:
  tcp 49157

378e52b0-c0a9-11cf-822d-00aa0051e40f 1.0:
  LPC: IUserProfile2
  LPC: OLE07A55C606630488F9A1CE31ACF85
  LPC: senssvc
  path: \PIPE\atsvc

3c4728c5-f0ab-448b-bda1-6ce01eb0a6d5 1.0: DHCP Client LRPC Endpoint
  LPC: AudioClientRpc
  LPC: Audiosrv
  LPC: OLE24E5DF29D091443BA08029561147
  LPC: dhcpcsvc
  LPC: dhcpcsvc6
  LPC: eventlog
  path: \pipe\eventlog
  tcp 49153

3c4728c5-f0ab-448b-bda1-6ce01eb0a6d6 1.0: DHCPv6 Client LRPC Endpoint
  LPC: AudioClientRpc
  LPC: Audiosrv
  LPC: OLE24E5DF29D091443BA08029561147
  LPC: dhcpcsvc6
  LPC: eventlog
  path: \pipe\eventlog
  tcp 49153

4a452661-8290-4b36-8fbe-7f4093a94978 1.0: Spooler function endpoint
  LPC: spoolss

4b112204-0e19-11d3-b42b-0000f81feb9f 1.0:
  LPC: LRPC-ba6a873927fc322d90
  LPC: OLE2C02BDEDDF2B45CFBB2DCAA76F31
  path: \PIPE\DAV RPC SERVICE
  path: \PIPE\wkssvc
  tcp 49155

5f54ce7d-5b79-4175-8584-cb65313a0e98 1.0: AppInfo
  LPC: IUserProfile2
  LPC: OLE07A55C606630488F9A1CE31ACF85
  LPC: RasmanRpc
  LPC: SECLOGON
  LPC: senssvc
  path: \PIPE\ROUTER
  path: \PIPE\atsvc
  path: \PIPE\srvsvc
  tcp 49156
```

```
654976df-1498-4056-a15e-cb4e87584bd8 1.0:
  LPC: LRPC-8c22bd6a4265c55669
  LPC: OLEBBDBE02F4D97481B8CC9D12DD894
  LPC: trkwks
  path: \pipe\trkwks

76f226c3-ec14-4325-8a99-6a46348418af 1.0:
  LPC: WMsgKRpc090170
  LPC: WMsgKRpc090901
  LPC: WindowsShutdown
  path: \PIPE\InitShutdown

7ea70bcf-48af-4f6a-8968-6a440754d5fa 1.0: NSI server endpoint
  LPC: LRPC-ba6a873927fc322d90
  LPC: OLE2C02BDEDDF2B45CFBB2DCAA76F31
  tcp 49155

7f9d11bf-7fb9-436b-a812-b2d50c5d4c03 1.0: Fw APIs
  LPC: LRPC-a7d7a7c48531b65290

86d35949-83c9-4044-b424-db363231fd0c 1.0:
  LPC: IUserProfile2
  LPC: OLE07A55C606630488F9A1CE31ACF85
  LPC: senssvc
  path: \PIPE\atsvc
  tcp 49156

a398e520-d59a-4bdd-aa7a-3c1e0303a511 1.0: IKE/Authip API
  LPC: IUserProfile2
  LPC: OLE07A55C606630488F9A1CE31ACF85
  LPC: senssvc
  path: \PIPE\atsvc
  path: \PIPE\srvsvc
  tcp 49156

ae33069b-a2a8-46ee-a235-ddfd339be281 1.0: Spooler base remote object endpoint
  LPC: spoolss

b58aa02e-2884-4e97-8176-4ee06d794184 1.0:
  LPC: LRPC-8c22bd6a4265c55669
  LPC: OLEBBDBE02F4D97481B8CC9D12DD894
  LPC: trkwks
  path: \pipe\trkwks

c9ac6db5-82b7-4e55-ae8a-e464ed7b4277 1.0: Impl friendly name
  LPC: IUserProfile2
  LPC: IUserProfile2
  LPC: IUserProfile2
  LPC: LRPC-00ee2fc62c1c1f0e4b
  LPC: LRPC-44cba3e606f899328a
  LPC: audit
  LPC: protected_storage
  LPC: samss lpc
  LPC: securityevent
  LPC: senssvc
  path: \PIPE\protected_storage
  path: \pipe\lsass

d95afe70-a6d5-4259-822e-2c84da1ddb0d 1.0:
  LPC: WMsgKRpc090170
  LPC: WindowsShutdown
  path: \PIPE\InitShutdown
  tcp 49152

dd490425-5325-4565-b774-7e27d6c09c24 1.0: Base Firewall Engine API
  LPC: LRPC-a7d7a7c48531b65290

f6beaff7-1e19-4fbb-9f8f-b89e2018337c 1.0: Event log TCPIP
  LPC: eventlog
```

```
  path: \pipe\eventlog
  tcp 49153

fd7a0523-dc70-43dd-9b2e-9c5ed48225b1 1.0: AppInfo
  LPC: IUserProfile2
  LPC: OLE07A55C606630488F9A1CE31ACF85
  LPC: RasmanRpc
  LPC: SECLOGON
  LPC: senssvc
  path: \PIPE\ROUTER
  path: \PIPE\atsvc
  path: \PIPE\srvsvc
  tcp 49156
```

In the above, “path” refers to named pipes.

When several RPC services share the same process, they also share the same endpoints[30]. As a result there may be interfaces bound to a network port that are not registered with the endpoint mapper.

APPENDIX XXV
ANONYMOUS AND AUTHENTICATED ACCESS TO NAMED PIPES

We determined which named pipes are remotely accessible, with and without authentication. To do this, we needed to turn on file sharing on the target machine. We conducted this test from both a Windows XP client and from a Windows Vista client. XP was observed to use SMB for access whereas Vista used SMB2 following a brief protocol negotiation; both use TCP port 445. We show the accessible named pipes in this appendix, and also show the results of RPC enumeration on those in the following appendix.

To identify the pipes that might be accessible remotely, we first locally enumerated all of the named pipes using the pipelist.exe tool[15]. The following is the result:

```
vista> pipelist > pipes.txt
vista> type pipes.txt

PipeList v1.01
by Mark Russinovich
http://www.sysinternals.com

Pipe Name                                  Instances       Max Instances
---------                                  ---------       -------------
InitShutdown                                   3              -1
lsass                                          4              -1
protected_storage                              3              -1
ntsvcs                                         3              -1
scerpc                                         3              -1
net\NtControlPipe1                             1               1
plugplay                                       3              -1
net\NtControlPipe2                             1               1
Winsock2\CatalogChangeListener-32c-0           1               1
net\NtControlPipe3                             1               1
epmapper                                       3              -1
Winsock2\CatalogChangeListener-1ec-0           1               1
LSM_API_service                                3              -1
net\NtControlPipe4                             1               1
eventlog                                       3              -1
net\NtControlPipe5                             1               1
Winsock2\CatalogChangeListener-3cc-0           1               1
net\NtControlPipe6                             1               1
net\NtControlPipe7                             1               1
net\NtControlPipe8                             1               1
Winsock2\CatalogChangeListener-24c-0           1               1
net\NtControlPipe0                             1               1
net\NtControlPipe9                             1               1
Winsock2\CatalogChangeListener-4dc-0           1               1
atsvc                                          3              -1
Winsock2\CatalogChangeListener-404-0           1               1
net\NtControlPipe10                            1               1
net\NtControlPipe11                            1               1
DAV RPC SERVICE                                3              -1
srvsvc                                         4              -1
wkssvc                                         4              -1
net\NtControlPipe12                            1               1
keysvc                                         3              -1
net\NtControlPipe13                            1               1
trkwks                                         3              -1
net\NtControlPipe14                            1               1
net\NtControlPipe15                            1               1
W32TIME_ALT                                    3              -1
Winsock2\CatalogChangeListener-238-0           1               1
PIPE_EVENTROOT\CIMV2SCM EVENT PROVIDER         2              -1
MsFteWds                                       2              -1
tapsrv                                         3              -1
ROUTER                                         3              -1
browser                                        3              -1
```

As additional information, we also enumerated the pipes in the HKEY_LOCAL_MACHINE\System\CurrentControlSet-\Services\LanmanServer\Parameters\NullSessionPipes registry key using regedit:

[15]Pipelist.exe was previously released online at www.sysinternals.com, but appears to no longer be available

- netlogon
- lsarpc
- samr
- browser

Items on this list are defined to be accessible via a null (anonymous) session.

We also located the HKEY_LOCAL_MACHINE\System\CurrentControlSet\Services\Npfs\Aliases\lsass key, which indicates that the following pipes are aliases of lsass:

- netlogon
- lsarpc
- samr
- protected_storage

A. Null Session Access to Named Pipes

To establish an anonymous connection to access the IPC$ share of the target machine, we used the following command:

```
xp> net use \\192.168.0.203\ipc$ /u:"" ""
```

We then ran a script, which was developed to establish connections to each of these named pipes, to determine which ones could be opened for read and write access. When run from a Windows XP machine we observed the following results:

```
xp> c:\python24\python trypipes.py -m 192.168.0.203 pipes.txt
\\192.168.0.203\PIPE\netlogon
\\192.168.0.203\PIPE\lsarpc
\\192.168.0.203\PIPE\samr
```

These results are consistent with the values in the NullSessionPipes registry key. However, the browser pipe was not accessible; perhaps different circumstances such as a local connection are required for a null session to be allowed to do that. These are all aliased to the lsass pipe.

We performed the same tests on a Windows Vista machine and achieved identical results.

B. Authenticated Session Access to Named Pipes

We repeated this test with an authenticated session, using the first account created during Vista install. We set that up using

```
xp> net use \\192.168.0.203\ipc$ /u:"jim"
```

and typing the password. We did not need to do this from the Vista client since the current user and associated password are identical to the target account.

From XP, trypipes.py yielded the following[16]:

```
xp> c:\python24\python trypipes.py -m 192.168.0.203 pipes.txt
\\192.168.0.203\PIPE\InitShutdown
\\192.168.0.203\PIPE\lsass
\\192.168.0.203\PIPE\protected_storage
\\192.168.0.203\PIPE\ntsvcs
\\192.168.0.203\PIPE\scerpc
\\192.168.0.203\PIPE\plugplay
\\192.168.0.203\PIPE\epmapper
\\192.168.0.203\PIPE\LSM_API_service
\\192.168.0.203\PIPE\eventlog
\\192.168.0.203\PIPE\atsvc
\\192.168.0.203\PIPE\DAV RPC SERVICE
\\192.168.0.203\PIPE\srvsvc
\\192.168.0.203\PIPE\wkssvc
\\192.168.0.203\PIPE\keysvc
\\192.168.0.203\PIPE\trkwks
\\192.168.0.203\PIPE\W32TIME_ALT
\\192.168.0.203\PIPE\MsFteWds
\\192.168.0.203\PIPE\tapsrv
\\192.168.0.203\PIPE\ROUTER
\\192.168.0.203\PIPE\browser
\\192.168.0.203\PIPE\netlogon
\\192.168.0.203\PIPE\lsarpc
\\192.168.0.203\PIPE\samr
```

[16]Initially “DAV RPC SERVICE” was omitted from this list due to a limit in trypipes.py that was caught on spaces in the name of the pipe. While we now include it here, it has not been included in the results elsewhere in this report.

These results include the three pipes that were available to a null session, and 20 additional pipes. We could find no documentation for, or public mention of, the LSM_API_service or MsFteWds pipes.

Once again, Vista yielded identical results (though in Appendix XXVI-D we show the RPC access available is not identical).

APPENDIX XXVI
RPC PROCEDURE ACCESS

In this appendix we show which RPC interfaces and procedures are remotely available with file sharing turned on. (There does not appear to be any RPC interfaces available on a freshly installed Vista host.) We tested using direct TCP-based access and via named pipes (for which we tried both anonymous and authenticated access, and from XP and Vista).

A. Tools

We wrote two scripts to help us with the testing. Identify2.py attempts to brute force the interfaces and procedures actually available over a network port or over a named pipe. The tool attempts to call each procedure between 0 and 99 to each interface in an large list of known UUIDs. By using this tool, we are able to enumerate most RPC services. If it receives an UNKNOWN_INTERFACE (0x1c010003), no output is produced. Otherwise the result is reported. From this, whether or not the interface is truly accessible can be inferred. “Success” indicates that the call reportedly succeeded. The call is made without knowledge of the correct arguments, so an ERR_BAD_STUB_DATA (1783) strongly suggests it could succeed if given the correct arguments. RANGE_ERROR (0x1c010002) is the result that is expected when calling a procedure that does not exist.

Not all interfaces available on a network port are usable; RPC mechanisms exist for blocking requests arriving over the network ([30], [28]). This is useful for services that do not wish to be available over the network, but share a process with another service that uses a network transport. This could explain some of the ACCESS_DENIED (5) errors received when calling certain procedures.

The second tool, identify.py, is similar to identify2.py. However, it only calls procedure 9999: a procedure number presumed to be higher than is actually accessible. The results from this tool are identical to the results from procedure 99, under the same circumstances, and in almost all cases; so we do not report these results separately.

The long list of UUIDs that we cycle over consists of the UUID, the version number, and human-readable informational text (to the extent we happened to know what the UUID is about). Some of the UUIDs on the list are included, based on their association with the release build of Vista. The interfaces we saw from the endpoint mapper (Appendix XXIV) are included. In addition, we attempted to extract client and server uses of UUIDs though static analysis of system executables found in a Vista RTM install. From rpcdecp.h, servers have the interface structure shown in Figure 49. Clients have an almost identical interface structure. In our observation, servers consistently have a InterpreterInfo value, while clients do not seem to have this value. From this, we can identify client instances versus server instances. Including all the unique UUIDs on the long list resulted in 64 additions, though some of those were only seen for clients (and hence would not be callable). We have 214 UUIDs on our list, but there could be other unknown ones.

B. Direct TCP Access

The endpoint mapper results we listed in Appendix XXIV referred to TCP port numbers 49152 to 49157. These, plus port 135 (the well-known endpoint mapper port) are potential ways to access RPC directly over TCP. However, we found that, per Nmap, only port 135 was open for remote access; the others are filtered by the firewall. This following is what Nmap reported:

```
linux# nmap -p 135,49152-49157 -sS $vmIP4

Starting Nmap 4.20 ( http://insecure.org ) at 2006-12-25 01:53 PST
Interesting ports on 192.168.0.203:
PORT      STATE    SERVICE
135/tcp   open     msrpc
49152/tcp filtered unknown
49153/tcp filtered unknown
49154/tcp filtered unknown
49155/tcp filtered unknown
49156/tcp filtered unknown
49157/tcp filtered unknown
MAC Address: 00:0C:29:72:E4:82 (VMware)

Nmap finished: 1 IP address (1 host up) scanned in 14.498 seconds
```

We ran identify2.py against port 135 to see the results of an attempt to call the first 100 procedures in each of the long list of UUIDs. The following is an excerpt:

```
vista> identify2.py -p 135 -T 192.168.0.203
00000136-0000-0000-c000-000000000046[v0.0] (ISCMLocalActivator), proc0: ACCESS_DENIED (5)
00000136-0000-0000-c000-000000000046[v0.0] (ISCMLocalActivator), proc1: ACCESS_DENIED (5)
00000136-0000-0000-c000-000000000046[v0.0] (ISCMLocalActivator), proc2: ACCESS_DENIED (5)
...
00000136-0000-0000-c000-000000000046[v0.0] (ISCMLocalActivator), proc99: ACCESS_DENIED (5)
000001a0-0000-0000-c000-000000000046[v0.0] (ISystemActivator), proc0: ACCESS_DENIED (5)
```

```
struct _RPC_SERVER_INTERFACE
{
    unsigned int Length;
    RPC_SYNTAX_IDENTIFIER InterfaceId;
    RPC_SYNTAX_IDENTIFIER TransferSyntax;
    PRPC_DISPATCH_TABLE DispatchTable;
    unsigned int RpcProtseqEndpointCount;
    PRPC_PROTSEQ_ENDPOINT RpcProtseqEndpoint;
    RPC_MGR_EPV __RPC_FAR *DefaultManagerEpv;
    void const __RPC_FAR *InterpreterInfo;
    unsigned int Flags;
}
```

Fig. 49. The RPC server interface structure

```
000001a0-0000-0000-c000-000000000046[v0.0] (ISystemActivator), proc1: ACCESS_DENIED (5)
...
64fe0b7f-9ef5-4553-a7db-9a1975777554[v1.0] (???), proc98: ACCESS_DENIED (5)
64fe0b7f-9ef5-4553-a7db-9a1975777554[v1.0] (???), proc99: ACCESS_DENIED (5)
99fcfec4-5260-101b-bbcb-00aa0021347a[v0.0] (IOXIDResolver), proc0: BAD_STUB_DATA (1783)
99fcfec4-5260-101b-bbcb-00aa0021347a[v0.0] (IOXIDResolver), proc1: BAD_STUB_DATA (1783)
99fcfec4-5260-101b-bbcb-00aa0021347a[v0.0] (IOXIDResolver), proc2: BAD_STUB_DATA (1783)
99fcfec4-5260-101b-bbcb-00aa0021347a[v0.0] (IOXIDResolver), proc3: success
99fcfec4-5260-101b-bbcb-00aa0021347a[v0.0] (IOXIDResolver), proc4: BAD_STUB_DATA (1783)
99fcfec4-5260-101b-bbcb-00aa0021347a[v0.0] (IOXIDResolver), proc5: success
99fcfec4-5260-101b-bbcb-00aa0021347a[v0.0] (IOXIDResolver), proc6: RANGE_ERROR (469827586)
99fcfec4-5260-101b-bbcb-00aa0021347a[v0.0] (IOXIDResolver), proc7: RANGE_ERROR (469827586)
99fcfec4-5260-101b-bbcb-00aa0021347a[v0.0] (IOXIDResolver), proc8: RANGE_ERROR (469827586)
...
99fcfec4-5260-101b-bbcb-00aa0021347a[v0.0] (IOXIDResolver), proc99: RANGE_ERROR (469827586)
afa8bd80-7d8a-11c9-bef4-08002b102989[v1.0] (rpcmgmt (ifids)), proc0: ACCESS_DENIED (5)
afa8bd80-7d8a-11c9-bef4-08002b102989[v1.0] (rpcmgmt (ifids)), proc1: ACCESS_DENIED (5)
afa8bd80-7d8a-11c9-bef4-08002b102989[v1.0] (rpcmgmt (ifids)), proc2: ACCESS_DENIED (5)
afa8bd80-7d8a-11c9-bef4-08002b102989[v1.0] (rpcmgmt (ifids)), proc3: ACCESS_DENIED (5)
afa8bd80-7d8a-11c9-bef4-08002b102989[v1.0] (rpcmgmt (ifids)), proc4: BAD_STUB_DATA (1783)
afa8bd80-7d8a-11c9-bef4-08002b102989[v1.0] (rpcmgmt (ifids)), proc5: ACCESS_DENIED (5)
afa8bd80-7d8a-11c9-bef4-08002b102989[v1.0] (rpcmgmt (ifids)), proc6: ACCESS_DENIED (5)
...
afa8bd80-7d8a-11c9-bef4-08002b102989[v1.0] (rpcmgmt (ifids)), proc98: ACCESS_DENIED (5)
afa8bd80-7d8a-11c9-bef4-08002b102989[v1.0] (rpcmgmt (ifids)), proc99: ACCESS_DENIED (5)
```

We noticed that sometimes we would encounter a -1 result in unexpected locations (where they contradict the apparent pattern). Based on the results of multiple runs, we concluded that there is a limitation in the script, that sometimes yields error -1 as the result; perhaps it is making requests too fast. The -1 result could appear in place of UNKNOWN_INTERFACE, ACCESS_DENIED, or BAD_STUB_DATA. By combining the results of multiple runs, we were able to get a result other than -1 for all procedure calls for all interfaces.

The results can be summarized as follows:

```
00000136-0000-0000-c000-000000000046[v0.0] (ISCMLocalActivator), proc0-99: ACCESS_DENIED (5)

000001a0-0000-0000-c000-000000000046[v0.0] (ISystemActivator), proc0-99: ACCESS_DENIED (5)

0b0a6584-9e0f-11cf-a3cf-00805f68cb1b[v1.1] (localpmp), proc0-99: ACCESS_DENIED (5)

1d55b526-c137-46c5-ab79-638f2a68e869[v1.0] (???), proc0-99: ACCESS_DENIED (5)

412f241e-c12a-11ce-abff-0020af6e7a17[v0.2] (ISCM), proc0-99: ACCESS_DENIED (5)

4d9f4ab8-7d1c-11cf-861e-0020af6e7c57[v0.0] (from rpcss.dll), proc0-99: ACCESS_DENIED (5)

64fe0b7f-9ef5-4553-a7db-9a1975777554[v1.0] (???), proc0-99: ACCESS_DENIED (5)

99fcfec4-5260-101b-bbcb-00aa0021347a[v0.0] (IOXIDResolver), proc0-2: BAD_STUB_DATA (1783)
99fcfec4-5260-101b-bbcb-00aa0021347a[v0.0] (IOXIDResolver), proc3: success
99fcfec4-5260-101b-bbcb-00aa0021347a[v0.0] (IOXIDResolver), proc4: BAD_STUB_DATA (1783)
99fcfec4-5260-101b-bbcb-00aa0021347a[v0.0] (IOXIDResolver), proc5: success
99fcfec4-5260-101b-bbcb-00aa0021347a[v0.0] (IOXIDResolver), proc6-99: RANGE_ERROR (469827586)

afa8bd80-7d8a-11c9-bef4-08002b102989[v1.0] (rpcmgmt (ifids)), proc0-3: ACCESS_DENIED (5)
afa8bd80-7d8a-11c9-bef4-08002b102989[v1.0] (rpcmgmt (ifids)), proc4: BAD_STUB_DATA (1783)
afa8bd80-7d8a-11c9-bef4-08002b102989[v1.0] (rpcmgmt (ifids)), proc5-99: ACCESS_DENIED (5)
```

Two interfaces existed that allowed us to successfully call some procedures—at least, this would be true with the correct arguments. In the IOXIDResolver interface, the first five procedures (ResolveOxid, SimplePing, ComplexPing, ServerAlive, ResolveOxid2, ServerAlive2) were accessible; apparently, only five procedures exist. In the RPC remote management interface, only the fourth procedure (rpc_mgmt_inq_princ_name) was callable.

We see ACCESS_DENIED appearing at the procedure level, suggesting access control is employed. We also see ACCESS_DENIED for all procedures from several other interfaces that we could detect (UNKNOWN_INTERFACE was not returned). The exposure of shared-process servers gives some insight into what other services are running on the machine. This information can be used for fingerprinting purposes[56]. It also seems within the realm of possibility that, even though ACCESS_DENIED is the normal result, there could be a way to attack the procedure call.

We present this, along with the additional results from this appendix, in Table II. The table contains the results of our attempts to call the first 100 procedures in a long list of UUIDs in each of five configurations: TCP 135, using named pipes over a null session from XP (Section XXVI-C), using named pipes over a null session from Vista (Section XXVI-C), using named pipes over a authenticated session from XP (Section XXVI-D), and using named pipes over a authenticated session from Vista (Section XXVI-D). Results from named pipes that produced the same error for all interfaces and procedures are omitted. Empty spaces indicate a UNKNOWN_INTERFACE; UUIDs that would be entirely blank (i.e. that were not seen anywhere) are omitted. “succ” represents a successful connection; “bad-stub” represents a “BAD_STUB_DATA (1783)” error; “range” represents a “RANGE_ERROR (469827586)” error; “cant-perf” represents a “CANT_PERFORM (1752)” error, and “denied” represents a “ACCESS_DENIED (5)” error.

TABLE II: The results of calling the first 100 procedures

Interface	Procedure #’s	TCP 135	XP null	Vista null	XP auth	Vista auth
`afa8bd80-7d8a-11c9-bef4-08002b102989[v1.0]`	0	denied	succ	succ	succ	succ
(rpcmgmt (ifids))	1	denied	bad-stub	bad-stub	bad-stub	bad-stub
	2–3	denied	succ	succ	succ	succ
	4	bad-stub	bad-stub	bad-stub	bad-stub	bad-stub
	5–99	denied	range	range	range	range
`12345778-1234-abcd-ef00-0123456789ab[v0.0]`	0		bad-stub	bad-stub	bad-stub	bad-stub
(LSA access (lsarpc))	1		denied	denied	bad-stub	bad-stub
	2–3		bad-stub	bad-stub	bad-stub	bad-stub
	4		denied	denied	bad-stub	bad-stub
	5–7		bad-stub	bad-stub	bad-stub	bad-stub
	8		denied	denied	bad-stub	bad-stub
	9		bad-stub	bad-stub	bad-stub	bad-stub
	10		denied	denied	bad-stub	bad-stub
	11		bad-stub	bad-stub	bad-stub	bad-stub
	12		denied	denied	bad-stub	bad-stub
	13–15		bad-stub	bad-stub	bad-stub	bad-stub
	16		denied	denied	bad-stub	bad-stub
	17–18		bad-stub	bad-stub	bad-stub	bad-stub
	19–20		denied	denied	bad-stub	bad-stub
	21		bad-stub	bad-stub	bad-stub	bad-stub
	22		denied	denied	bad-stub	bad-stub
	23		bad-stub	bad-stub	bad-stub	bad-stub
	24		denied	denied	bad-stub	bad-stub
	25–26		bad-stub	bad-stub	bad-stub	bad-stub
	27–30		denied	denied	bad-stub	bad-stub
	31–33		bad-stub	bad-stub	bad-stub	bad-stub
	34		denied	denied	bad-stub	bad-stub
	35–36		bad-stub	bad-stub	bad-stub	bad-stub
	37–38		denied	denied	bad-stub	bad-stub
	39		bad-stub	bad-stub	bad-stub	bad-stub
	40–43		denied	denied	bad-stub	bad-stub
	44–46		bad-stub	bad-stub	bad-stub	bad-stub
	47		denied	denied	bad-stub	bad-stub
	48		bad-stub	bad-stub	bad-stub	bad-stub
	49		denied	denied	bad-stub	bad-stub
	50		bad-stub	bad-stub	bad-stub	bad-stub
	51		denied	denied	bad-stub	bad-stub
	52–53		bad-stub	bad-stub	bad-stub	bad-stub
	54		denied	denied	bad-stub	bad-stub
	55		bad-stub	bad-stub	bad-stub	bad-stub
	56		denied	denied	bad-stub	bad-stub
	57–58		bad-stub	bad-stub	bad-stub	bad-stub
	59		denied	denied	bad-stub	bad-stub

(Continued on next page)

Interface	Procedure #'s	TCP 135	XP null	Vista null	XP auth	Vista auth
	60–67		denied	denied	bad-stub	denied
	68		bad-stub	bad-stub	bad-stub	bad-stub
	69–72		denied	denied	bad-stub	denied
	73		bad-stub	bad-stub	bad-stub	bad-stub
	74		denied	denied	bad-stub	bad-stub
	75–78		denied	denied	bad-stub	denied
	79–99		denied	denied	range	denied
12345778-1234-abcd-ef00-0123456789ac[v1.0]	0–1		bad-stub	bad-stub	bad-stub	bad-stub
(samsrv)	2		denied	denied	bad-stub	bad-stub
	3		bad-stub	bad-stub	bad-stub	bad-stub
	4		denied	denied	bad-stub	bad-stub
	5–8		bad-stub	bad-stub	bad-stub	bad-stub
	9–10		denied	denied	bad-stub	bad-stub
	11		bad-stub	bad-stub	bad-stub	bad-stub
	12		denied	denied	bad-stub	bad-stub
	13		bad-stub	bad-stub	bad-stub	bad-stub
	14		denied	denied	bad-stub	bad-stub
	15–20		bad-stub	bad-stub	bad-stub	bad-stub
	21–24		denied	denied	bad-stub	bad-stub
	25		bad-stub	bad-stub	bad-stub	bad-stub
	26		denied	denied	bad-stub	bad-stub
	27–28		bad-stub	bad-stub	bad-stub	bad-stub
	29–32		denied	denied	bad-stub	bad-stub
	33–34		bad-stub	bad-stub	bad-stub	bad-stub
	35		denied	denied	bad-stub	bad-stub
	36		bad-stub	bad-stub	bad-stub	bad-stub
	37		denied	denied	bad-stub	bad-stub
	38–41		bad-stub	bad-stub	bad-stub	bad-stub
	42–43		denied	denied	bad-stub	bad-stub
	44		bad-stub	bad-stub	bad-stub	bad-stub
	45		denied	denied	bad-stub	bad-stub
	46–49		bad-stub	bad-stub	bad-stub	bad-stub
	50		denied	denied	bad-stub	bad-stub
	51		bad-stub	bad-stub	bad-stub	bad-stub
	52–53		denied	denied	bad-stub	bad-stub
	54–57		bad-stub	bad-stub	bad-stub	bad-stub
	58		denied	denied	bad-stub	bad-stub
	59		denied	denied	bad-stub	denied
	60		denied	denied	bad-stub	bad-stub
	61–65		bad-stub	bad-stub	bad-stub	bad-stub
	66		denied	denied	bad-stub	bad-stub
	67–99		denied	denied	range	denied
3919286a-b10c-11d0-9ba8-00c04fd92ef5[v0.0]	0		bad-stub	bad-stub	bad-stub	bad-stub
(LSA DS access (lsarpc))	1–99		range	range	range	range
c681d488-d850-11d0-8c52-00c04fd90f7e[v1.0]	0–13		bad-stub	bad-stub	bad-stub	bad-stub
(efsrpc)	14–16		bad-stub	bad-stub	range	bad-stub
	17		succ	succ	range	succ
	18–19		bad-stub	bad-stub	range	bad-stub
	20		succ	succ	range	succ
	21–99		range	range	range	range
0b0a6584-9e0f-11cf-a3cf-00805f68cb1b[v1.1]	0	denied			succ	denied
(localpmp)	1–4	denied			bad-stub	denied
	5	denied			succ	denied
	6–99	denied			range	denied
11220835-5b26-4d94-ae86-c3e475a809de[v1.0]						
(ICryptProtect)	0–99		denied	denied	denied	denied
1ff70682-0a51-30e8-076d-740be8cee98b[v1.0]	0–3				bad-stub	bad-stub
(atsvc)	4–99				range	range
300f3532-38cc-11d0-a3f0-0020af6b0add[v1.2]	0–3				bad-stub	bad-stub
(trkwks)	4				succ	succ
	5–7				bad-stub	bad-stub
	8				succ	succ
	9–12				bad-stub	bad-stub
	13–99				range	range

(Continued on next page)

Interface	Procedure #'s	TCP 135	XP null	Vista null	XP auth	Vista auth
367abb81-9844-35f1-ad32-98f038001003[v2.0] (Services Control Manager (SCM))	0–43				bad-stub	bad-stub
	44–55				range	bad-stub
	56–99				range	range
4b324fc8-1670-01d3-1278-5a47bf6ee188[v3.0] (from srvsvc.dll, Netr*)	0–53				bad-stub	bad-stub
	54–57				range	bad-stub
	58–99				range	range
5cbe92cb-f4be-45c9-9fc9-33e73e557b20[v1.0] (from lsasrv.dll)	0–99		denied	denied	denied	denied
6bffd098-a112-3610--9833-012892020162[v0.0] (from browser.dll, I_Browserr*, NetrBrowser*)	0–11				bad-stub	bad-stub
	12–99				range	range
6bffd098-a112-3610--9833-46c3f87e345a[v1.0] (wkssvc)	0–30				bad-stub	bad-stub
	31				bad-stub	range
	32–99				range	range
82273fdc-e32a-18c3-3f78-827929dc23ea[v0.0] (eventlog, from wevtsvc.dll)	0–23				bad-stub	bad-stub
	24				range	bad-stub
	25–99				range	range
99fcfec4-5260--101b-bbcb-00aa0021347a[v0.0] (IOXIDResolver)	0–2	bad-stub			denied	denied
	3	succ			denied	denied
	4	bad-stub			denied	denied
	5	succ			denied	denied
	6–99	range			denied	denied
e1af8308-5d1f-11c9-91a4-08002b14a0fa[v3.0] (epmapper)	0–1				bad-stub	cant-perf
	2–4				bad-stub	bad-stub
	5–6				bad-stub	cant-perf
	7–8				range	bad-stub
	9–99				range	range
00000136-0000-0000-c000-000000000046[v0.0] (ISCMLocalActivator)	0–99	denied			denied	denied
000001a0-0000-0000-c000-000000000046[v0.0] (ISystemActivator)	0–99	denied			denied	denied
1d55b526-c137-46c5-ab79-638f2a68e869[v1.0] (???)	0–99	denied			denied	denied
412f241e-c12a-11ce-abff-0020af6e7a17[v0.2] (ISCM)	0–99	denied			denied	denied
4d9f4ab8-7d1c-11cf-861e-0020af6e7c57[v0.0] (from rpcss.dll)	0–99	denied			denied	denied
894de0c0-0d55-11d3-a322-00c04fa321a1[v1.0] (InitShutdown)	0–2				bad-stub	denied
	3–99				range	denied
8d9f4e40-a03d-11ce-8f69-08003e30051b[v1.0] (umpnpmgr)	0–1				bad-stub	denied
	2				succ	denied
	3				bad-stub	denied
	4				succ	denied
	5–38				bad-stub	denied
	39				succ	denied
	40–64				bad-stub	denied
	65–99				range	denied
8fb6d884-2388-11d0-8c35-00c04fda2795[v4.1] (w32time)	0				bad-stub	denied
	1				succ	denied
	2				bad-stub	denied
	3–99				range	denied
93149ca2-973b-11d1-8c39-00c04fb984f9[v0.0] (scesrv)	0–19				bad-stub	denied
	20				succ	denied
	21–33				bad-stub	denied
	34–99				range	denied
c9ac6db5-82b7-4e55-ae8a-e464ed7b4277[v1.0] (sysntfy)	0–99		denied	denied		denied
00000131-0000-0000-c000-000000000046[v0.0] (from ole32.dll)	0–99				denied	denied
00000132-0000-0000-c000-000000000046[v0.0]						

(Continued on next page)

Interface	Procedure #'s	TCP 135	XP null	Vista null	XP auth	Vista auth
((used by rpcss.dll))	0–99				denied	denied
00000134-0000-0000-c000-000000000046[v0.0] ((used by rpcss.dll))	0–99				denied	denied
00000143-0000-0000-c000-000000000046[v0.0] (from ole32.dll)	0–99				denied	denied
06bba54a-be05-49f9-b0a0-30f790261023[v1.0] (wscsvc)	0–99				denied	denied
0a74ef1c-41a4-4e06-83ae-dc74fb1cdd53[v1.0] (taskeng (idletask))	0–99				denied	denied
0d72a7d4-6148-11d1-b4aa-00c04fb66ea0[v1.0] (ICertProtect)	0–99				denied	denied
12b81e99-f207-4a4c-85d3-77b42f76fd14[v1.0] (seclogon (ISeclogon))	0–99				denied	denied
18f70770-8e64-11cf-9af1-0020af6e72f4[v0.0] (ole32 (IOrCallback))	0–99				denied	denied
20610036-fa22-11cf-9823-00a0c911e5df[v1.0] (rasmans)	0–99				denied	denied
2f5f6520-ca46-1067-b319-00dd010662da[v1.0] (tapisrv)	0–99				denied	denied
2f5f6521-cb55-1059-b446-00df0bce31db[v1.0] (unimdm)	0–99				denied	denied
326731e3-c1c0-4a69-ae20-7d9044a4ea5c[v1.0] (profsvc (IUserProfile))	0–99				denied	denied
378e52b0-c0a9-11cf-822d-00aa0051e40f[v1.0] (sasec)	0–99				denied	denied
3c4728c5-f0ab-448b-bda1-6ce01eb0a6d5[v1.0] (dhcpcsvc (RpcSrvDHCPC))	0–99				denied	denied
3dde7c30-165d-11d1-ab8f-00805f14db40[v1.0] (BackupKey)	0 1–99				bad-stub range	
3faf4738-3a21-4307-b46c-fdda9bb8c0d5[v1.1] (AudioSrv)	0–99				denied	denied
63fbe424-2029-11d1-8db8-00aa004abd5e[v1.0] (SensApi)	0–99				denied	denied
64fe0b7f-9ef5-4553-a7db-9a1975777554[v1.0] (???)	0–99	denied				denied
b9e79e60-3d52-11ce-aaa1-00006901293f[v0.2] (IROT)	0–99				denied	denied
c6f3ee72-ce7e-11d1-b71e-00c04fc3111a[v1.0] (IMachineActivatorControl)	0–99				denied	denied
e60c73e6-88f9-11cf-9af1-0020af6e72f4[v2.0] (ILocalObjectExporter)	0–99				denied	denied
f50aac00-c7f3-428e-a022-a6b71bfb9d43[v1.0] (ICatDBSvc)	0–99				denied	denied
0767a036-0d22-48aa-ba69-b619480f38cb[v1.0] (?PcaSvc)	0–99					denied
11899a43-2b68-4a76-92e3-a3d6ad8c26ce[v1.0] (???)	0–99					denied
11f25515-c879-400a-989e-b074d5f092fe[v1.0] (???)	0–99					denied
12345678-1234-abcd-ef00-0123456789ab[v1.0] (IPSECSVC (winipsec))	0–99				denied	
1e665584-40fe-4450-8f6e-802362399694[v1.0] (???)	0–99					denied
201ef99a-7fa0-444c-9399-19ba84f12a1a[v1.0] (?AppInfo)	0–99					denied
2eb08e3e-639f-4fba-97b1-14f878961076[v1.0]						

(Continued on next page)

Interface	Procedure #'s	TCP 135	XP null	Vista null	XP auth	Vista auth
(???)	0–99					denied
3c4728c5-f0ab-448b-bda1-6ce01eb0a6d6[v1.0] (dhcpcsvc6)	0–99					denied
45776b01-5956-4485-9f80-f428f7d60129[v2.0] (from dnsrslvr.dll)	0–99					denied
484809d6-4239--471b-b5bc-61df8c23ac48[v1.0] (from lsm.exe)	0–99					denied
4b112204-0e19-11d3-b42b-0000f81feb9f[v1.0] (????)	0–99					denied
5f54ce7d-5b79-4175-8584-cb65313a0e98[v1.0] (?AppInfo??)	0–99					denied
621dff68-3c39-4c6c-aae3-e68e2c6503ad[v1.0] (wzcsvc (winwzc))	0–99				denied	
629b9f66-556c-11d1-8dd2-00aa004abd5e[v3.0] (SENSNotify)	0–99				denied	
654976df-1498-4056-a15e-cb4e87584bd8[v1.0] (????)	0–99					denied
68b58241-c259-4f03-a2e5-a2651dcbc930[v1.0] (???)	0–99					denied
76f226c3-ec14-4325-8a99-6a46348418ae[v1.0] (???)	0–99					denied
76f226c3-ec14-4325-8a99-6a46348418af[v1.0] (???)	0–99					denied
7ea70bcf-48af-4f6a-8968-6a440754d5fa[v1.0] (nsisvc)	0–99					denied
86d35949-83c9-4044-b424-db363231fd0c[v1.0] (???)	0–99					denied
88143fd0-c28d-4b2b-8fef-8d882f6a9390[v1.0] (???)	0–99					denied
9b8699ae-0e44-47b1-8e7f-86a461d7ecdc[v0.0] (???)	0–99					denied
a002b3a0-c9b7-11d1-ae88-0080c75e4ec1[v1.0] ((used by MigAutoPlay.exe and wuaueng.dll))	0–99				denied	
a0bc4698-b8d7-4330-a28f-7709e18b6108[v4.0] (from Sens.dll)	0–99					denied
a398e520-d59a-4bdd-aa7a-3c1e0303a511[v1.0] (IKEEXT)	0–99					denied
aa411582-9bdf-48fb-b42b-faa1eee33949[v1.0] (???)	0–99					denied
b58aa02e-2884-4e97-8176-4ee06d794184[v1.0] (sysmain)	0–99					denied
c13d3372-cc20-4449-9b23-8cc8271b3885[v1.0] (???)	0–99					denied
c33b9f46-2088-4dbc-97e3-6125f127661c[v1.0] (???)	0–99					denied
c386ca3e-9061-4a72-821e-498d83be188f[v1.1] (???)	0–99					denied
c8cb7687-e6d3-11d2-a958-00c04f682e16[v1.0] (WebClnt (davclntrpc))	0–99					denied
c9378ff1-16f7-11d0-a0b2-00aa0061426a[v1.0] (pstorsvc (IPStoreProv))	0–99				denied	
d95afe70-a6d5-4259--822e-2c84da1ddb0d[v1.0] (???)	0–99					denied
f6beaff7-1e19-4fbb-9f8f-b89e2018337c[v1.0] (???)	0–99					denied
fd7a0523-dc70-43dd-9b2e-9c5ed48225b1[v1.0]						

(Continued on next page)

Interface	Procedure #'s	TCP 135	XP null	Vista null	XP auth	Vista auth
(?AppInfo)	0–99					denied

C. Null Session Named Pipe Access

Three named pipes are available over a null session (in Appendix XXV-A): netlogon, lsarpc, and samr. We focused on those three to find the set of procedures that are accessible anonymously over named pipes.

With the null session set up, we ran identify2.py on netlogon from an XP client:

```
xp> c:\python24\python identify2.py -P -f netlogon 192.168.0.203
```

The output of this is summarized as follows:

```
11220835-5b26-4d94-ae86-c3e475a809de[v1.0] (ICryptProtect), proc0-99: ACCESS_DENIED (5)

12345778-1234-abcd-ef00-0123456789ab[v0.0] (LSA access (lsarpc)), proc0: BAD_STUB_DATA (1783)
12345778-1234-abcd-ef00-0123456789ab[v0.0] (LSA access (lsarpc)), proc1: ACCESS_DENIED (5)
12345778-1234-abcd-ef00-0123456789ab[v0.0] (LSA access (lsarpc)), proc2-3: BAD_STUB_DATA (1783)
12345778-1234-abcd-ef00-0123456789ab[v0.0] (LSA access (lsarpc)), proc4: ACCESS_DENIED (5)
12345778-1234-abcd-ef00-0123456789ab[v0.0] (LSA access (lsarpc)), proc5-7: BAD_STUB_DATA (1783)
12345778-1234-abcd-ef00-0123456789ab[v0.0] (LSA access (lsarpc)), proc8: ACCESS_DENIED (5)
12345778-1234-abcd-ef00-0123456789ab[v0.0] (LSA access (lsarpc)), proc9: BAD_STUB_DATA (1783)
12345778-1234-abcd-ef00-0123456789ab[v0.0] (LSA access (lsarpc)), proc10: ACCESS_DENIED (5)
12345778-1234-abcd-ef00-0123456789ab[v0.0] (LSA access (lsarpc)), proc11: BAD_STUB_DATA (1783)
12345778-1234-abcd-ef00-0123456789ab[v0.0] (LSA access (lsarpc)), proc12: ACCESS_DENIED (5)
12345778-1234-abcd-ef00-0123456789ab[v0.0] (LSA access (lsarpc)), proc13-15: BAD_STUB_DATA (1783)
12345778-1234-abcd-ef00-0123456789ab[v0.0] (LSA access (lsarpc)), proc16: ACCESS_DENIED (5)
12345778-1234-abcd-ef00-0123456789ab[v0.0] (LSA access (lsarpc)), proc17-18: BAD_STUB_DATA (1783)
12345778-1234-abcd-ef00-0123456789ab[v0.0] (LSA access (lsarpc)), proc19-20: ACCESS_DENIED (5)
12345778-1234-abcd-ef00-0123456789ab[v0.0] (LSA access (lsarpc)), proc21: BAD_STUB_DATA (1783)
12345778-1234-abcd-ef00-0123456789ab[v0.0] (LSA access (lsarpc)), proc22: ACCESS_DENIED (5)
12345778-1234-abcd-ef00-0123456789ab[v0.0] (LSA access (lsarpc)), proc23: BAD_STUB_DATA (1783)
12345778-1234-abcd-ef00-0123456789ab[v0.0] (LSA access (lsarpc)), proc24: ACCESS_DENIED (5)
12345778-1234-abcd-ef00-0123456789ab[v0.0] (LSA access (lsarpc)), proc25-26: BAD_STUB_DATA (1783)
12345778-1234-abcd-ef00-0123456789ab[v0.0] (LSA access (lsarpc)), proc27-30: ACCESS_DENIED (5)
12345778-1234-abcd-ef00-0123456789ab[v0.0] (LSA access (lsarpc)), proc31-33: BAD_STUB_DATA (1783)
12345778-1234-abcd-ef00-0123456789ab[v0.0] (LSA access (lsarpc)), proc34: ACCESS_DENIED (5)
12345778-1234-abcd-ef00-0123456789ab[v0.0] (LSA access (lsarpc)), proc35-36: BAD_STUB_DATA (1783)
12345778-1234-abcd-ef00-0123456789ab[v0.0] (LSA access (lsarpc)), proc37-38: ACCESS_DENIED (5)
12345778-1234-abcd-ef00-0123456789ab[v0.0] (LSA access (lsarpc)), proc39: BAD_STUB_DATA (1783)
12345778-1234-abcd-ef00-0123456789ab[v0.0] (LSA access (lsarpc)), proc40-43: ACCESS_DENIED (5)
12345778-1234-abcd-ef00-0123456789ab[v0.0] (LSA access (lsarpc)), proc44-46: BAD_STUB_DATA (1783)
12345778-1234-abcd-ef00-0123456789ab[v0.0] (LSA access (lsarpc)), proc47: ACCESS_DENIED (5)
12345778-1234-abcd-ef00-0123456789ab[v0.0] (LSA access (lsarpc)), proc48: BAD_STUB_DATA (1783)
12345778-1234-abcd-ef00-0123456789ab[v0.0] (LSA access (lsarpc)), proc49: ACCESS_DENIED (5)
12345778-1234-abcd-ef00-0123456789ab[v0.0] (LSA access (lsarpc)), proc50: BAD_STUB_DATA (1783)
12345778-1234-abcd-ef00-0123456789ab[v0.0] (LSA access (lsarpc)), proc51: ACCESS_DENIED (5)
12345778-1234-abcd-ef00-0123456789ab[v0.0] (LSA access (lsarpc)), proc52-53: BAD_STUB_DATA (1783)
12345778-1234-abcd-ef00-0123456789ab[v0.0] (LSA access (lsarpc)), proc54: ACCESS_DENIED (5)
12345778-1234-abcd-ef00-0123456789ab[v0.0] (LSA access (lsarpc)), proc55: BAD_STUB_DATA (1783)
12345778-1234-abcd-ef00-0123456789ab[v0.0] (LSA access (lsarpc)), proc56: ACCESS_DENIED (5)
12345778-1234-abcd-ef00-0123456789ab[v0.0] (LSA access (lsarpc)), proc57-58: BAD_STUB_DATA (1783)
12345778-1234-abcd-ef00-0123456789ab[v0.0] (LSA access (lsarpc)), proc59-67: ACCESS_DENIED (5)
12345778-1234-abcd-ef00-0123456789ab[v0.0] (LSA access (lsarpc)), proc68: BAD_STUB_DATA (1783)
12345778-1234-abcd-ef00-0123456789ab[v0.0] (LSA access (lsarpc)), proc69-72: ACCESS_DENIED (5)
12345778-1234-abcd-ef00-0123456789ab[v0.0] (LSA access (lsarpc)), proc73: BAD_STUB_DATA (1783)
12345778-1234-abcd-ef00-0123456789ab[v0.0] (LSA access (lsarpc)), proc74-99: ACCESS_DENIED (5)

12345778-1234-abcd-ef00-0123456789ac[v1.0] (samsrv), proc0-1: BAD_STUB_DATA (1783)
12345778-1234-abcd-ef00-0123456789ac[v1.0] (samsrv), proc2: ACCESS_DENIED (5)
12345778-1234-abcd-ef00-0123456789ac[v1.0] (samsrv), proc3: BAD_STUB_DATA (1783)
12345778-1234-abcd-ef00-0123456789ac[v1.0] (samsrv), proc4: ACCESS_DENIED (5)
12345778-1234-abcd-ef00-0123456789ac[v1.0] (samsrv), proc5-8: BAD_STUB_DATA (1783)
12345778-1234-abcd-ef00-0123456789ac[v1.0] (samsrv), proc9-10: ACCESS_DENIED (5)
12345778-1234-abcd-ef00-0123456789ac[v1.0] (samsrv), proc11: BAD_STUB_DATA (1783)
12345778-1234-abcd-ef00-0123456789ac[v1.0] (samsrv), proc12: ACCESS_DENIED (5)
12345778-1234-abcd-ef00-0123456789ac[v1.0] (samsrv), proc13: BAD_STUB_DATA (1783)
12345778-1234-abcd-ef00-0123456789ac[v1.0] (samsrv), proc14: ACCESS_DENIED (5)
12345778-1234-abcd-ef00-0123456789ac[v1.0] (samsrv), proc15-20: BAD_STUB_DATA (1783)
12345778-1234-abcd-ef00-0123456789ac[v1.0] (samsrv), proc21-24: ACCESS_DENIED (5)
12345778-1234-abcd-ef00-0123456789ac[v1.0] (samsrv), proc25: BAD_STUB_DATA (1783)
12345778-1234-abcd-ef00-0123456789ac[v1.0] (samsrv), proc26: ACCESS_DENIED (5)
12345778-1234-abcd-ef00-0123456789ac[v1.0] (samsrv), proc27-28: BAD_STUB_DATA (1783)
12345778-1234-abcd-ef00-0123456789ac[v1.0] (samsrv), proc29-32: ACCESS_DENIED (5)
12345778-1234-abcd-ef00-0123456789ac[v1.0] (samsrv), proc33-34: BAD_STUB_DATA (1783)
```

```
12345778-1234-abcd-ef00-0123456789ac[v1.0] (samsrv), proc35: ACCESS_DENIED (5)
12345778-1234-abcd-ef00-0123456789ac[v1.0] (samsrv), proc36: BAD_STUB_DATA (1783)
12345778-1234-abcd-ef00-0123456789ac[v1.0] (samsrv), proc37: ACCESS_DENIED (5)
12345778-1234-abcd-ef00-0123456789ac[v1.0] (samsrv), proc38-41: BAD_STUB_DATA (1783)
12345778-1234-abcd-ef00-0123456789ac[v1.0] (samsrv), proc42-43: ACCESS_DENIED (5)
12345778-1234-abcd-ef00-0123456789ac[v1.0] (samsrv), proc44: BAD_STUB_DATA (1783)
12345778-1234-abcd-ef00-0123456789ac[v1.0] (samsrv), proc45: ACCESS_DENIED (5)
12345778-1234-abcd-ef00-0123456789ac[v1.0] (samsrv), proc46-49: BAD_STUB_DATA (1783)
12345778-1234-abcd-ef00-0123456789ac[v1.0] (samsrv), proc50: ACCESS_DENIED (5)
12345778-1234-abcd-ef00-0123456789ac[v1.0] (samsrv), proc51: BAD_STUB_DATA (1783)
12345778-1234-abcd-ef00-0123456789ac[v1.0] (samsrv), proc52-53: ACCESS_DENIED (5)
12345778-1234-abcd-ef00-0123456789ac[v1.0] (samsrv), proc54-57: BAD_STUB_DATA (1783)
12345778-1234-abcd-ef00-0123456789ac[v1.0] (samsrv), proc58-60: ACCESS_DENIED (5)
12345778-1234-abcd-ef00-0123456789ac[v1.0] (samsrv), proc61-65: BAD_STUB_DATA (1783)
12345778-1234-abcd-ef00-0123456789ac[v1.0] (samsrv), proc66-99: ACCESS_DENIED (5)

3919286a-b10c-11d0-9ba8-00c04fd92ef5[v0.0] (LSA DS access (lsarpc)), proc0: BAD_STUB_DATA (1783)
3919286a-b10c-11d0-9ba8-00c04fd92ef5[v0.0] (LSA DS access (lsarpc)), proc1-99: RANGE_ERROR (469827586)

5cbe92cb-f4be-45c9-9fc9-33e73e557b20[v1.0] (from lsasrv.dll), proc0-99: ACCESS_DENIED (5)

afa8bd80-7d8a-11c9-bef4-08002b102989[v1.0] (rpcmgmt (ifids)), proc0: success
afa8bd80-7d8a-11c9-bef4-08002b102989[v1.0] (rpcmgmt (ifids)), proc1: BAD_STUB_DATA (1783)
afa8bd80-7d8a-11c9-bef4-08002b102989[v1.0] (rpcmgmt (ifids)), proc2-3: success
afa8bd80-7d8a-11c9-bef4-08002b102989[v1.0] (rpcmgmt (ifids)), proc4: BAD_STUB_DATA (1783)
afa8bd80-7d8a-11c9-bef4-08002b102989[v1.0] (rpcmgmt (ifids)), proc5-99: RANGE_ERROR (469827586)

c681d488-d850-11d0-8c52-00c04fd90f7e[v1.0] (efsrpc), proc0-16: BAD_STUB_DATA (1783)
c681d488-d850-11d0-8c52-00c04fd90f7e[v1.0] (efsrpc), proc17: success
c681d488-d850-11d0-8c52-00c04fd90f7e[v1.0] (efsrpc), proc18-19: BAD_STUB_DATA (1783)
c681d488-d850-11d0-8c52-00c04fd90f7e[v1.0] (efsrpc), proc20: success
c681d488-d850-11d0-8c52-00c04fd90f7e[v1.0] (efsrpc), proc21-99: RANGE_ERROR (469827586)

c9ac6db5-82b7-4e55-ae8a-e464ed7b4277[v1.0] (sysntfy), proc0-99: ACCESS_DENIED (5)
```

Specific procedures in five interfaces are callable, but none in the three others were. Identical results were obtained from lsarpc and samr, which is consistent with the three being aliased to the same pipe. We also obtained the same result using Vista as the client. `12345778-1234-abcd-ef00-0123456789ab` has 102 procedures thus we did not test the last two. Figure 7 (page 15) shows the the names of the procedures that were successfully called via a null session.

Examining these results in Table II (second and third results columns) and comparing these results to the direct TCP 135 results, shows that there is one interface in common, the RPC management interface. However, we have more access to the procedures in that interface using named pipes and a null session, than using a direct connection to port 135, since we can call all of the first five procedures. With the range error now received, there appears to be exactly five procedures.

When we tried to call procedures in other named pipes including browser, lsass, InitShutdown, and an arbitrary non-existent pipe, we received a uniform result of -1, since the attempt to open the pipe had yielded STATUS_ACCESS_DENIED (0xc0000022).

D. Authenticated Session Named Pipe Access

From Appendix XXV-B, we have a list of 22 named pipes[17] that can be opened from Vista or XP across an authenticated SMB/SMB2 session. For each of the named pipes, we ran identify2.py in an authenticated session from both XP and Vista, in order to determine which interfaces and procedures could be accessed.

From XP, we found that we could not get useful results from five of the named pipes (LSM_API_service, MsFteWds, W32TIME_ALT, plugplay, and tapsrv) since all attempts to call procedures yielded a -1 result. From Vista, scerpc and srvsvc always returned -1, and MsFteWds calls were aborted due to a shorter RPC header than expected being returned by the server. We excluded those pipes from the analysis of accessibility.

In our testing from XP, we got certain erroneous -1 results as a result for pipes what were mainly usable. This is similar to the results we saw from direct access to TCP port 135 (Appendix XXVI-B) and we corrected it similarly, by merging in the results of a second run.

We got varied results from the different pipes, as depicted in results columns 4 and 5 in Table II. However, interfaces that appear in multiple pipes yield the same result within a configuration (authenticated XP or authenticated Vista). That result is also true for the null session named pipe testing, so the specific named pipe does not matter as long as that pipe has access to the relevant interface for the scenario

Table III depicts the observed UUIDs and pipes, and the circumstances under which they resulted in an error other than UNKNOWN_INTERFACE. X-A means the interface could only be accessed from Windows XP (SMB) with an authenticated session; V-A means the interface could only be accessed from Windows Vista (SMB2) with an authenticated session; A means

[17] As noted in that appendix, the named pipe “DAV RPC SERVICE” should have also been included in this list.

it could be accessed from either source of authenticated session; VorN means it could be accessed from either a XP or Vista null session or from an authenticated Vista session, and any means that it could be accessed from both OSs and session types. Note that some columns represent multiple pipes that behaved identically for UUID access. Also described are the circumstances under which pipes could be opened and usefully used. The faint dotted grid lines on every fifth row and column are depicted only to facilitate reading the rows and columns; no grouping is implied.

TABLE III: When UUIDs and pipes result in an error other than UNKNOWN_INTERFACE

Interface	lsarpc,samr,netlogon	lsass,protected_storage	LSM_API_service	W32TIME_ALT	wkssvc	atsvc,ROUTER,browser	srvsvc	trkwks	keysvc	tapsrv	InitShutdown	eventlog	ntsvcs	scerpc	plugplay	epmapper	MsFteWds
can be opened	any	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A
can be opened and used to call interfaces	any	A	V-A	V-A	A	A	X-A	A	A	V-A	A	A	A	X-A	V-A	A	
`11220835-5b26-4d94-ae86-c3e475a809de[v1.0]` (rpcmgmt (ifids))	any	A	V-A	V-A	A	A	X-A	A	A	V-A	A	A	A	X-A	V-A	A	
`11220835-5b26-4d94-ae86-c3e475a809de[v1.0]` (ICryptProtect)	any	A															
`12345778-1234-abcd-ef00-0123456789ab[v0.0]` (LSA access (lsarpc))	any	A															
`12345778-1234-abcd-ef00-0123456789ac[v1.0]` (samsrv)	any	A															
`3919286a-b10c-11d0-9ba8-00c04fd92ef5[v0.0]` (LSA DS access (lsarpc))	any	A															
`5cbe92cb-f4be-45c9-9fc9-33e73e557b20[v1.0]` (from lsasrv.dll)	any	A															
`c681d488-d850-11d0-8c52-00c04fd90f7e[v1.0]` (efsrpc)	any	A															
`12345678-1234-abcd-ef00-0123456789ab[v1.0]` (IPSECSVC (winipsec))	X-A	X-A															
`c9378ff1-16f7-11d0-a0b2-00aa0061426a[v1.0]` (pstorsvc (IPStoreProv))	X-A	X-A															
`c9ac6db5-82b7-4e55-ae8a-e464ed7b4277[v1.0]` (sysntfy)	VorN	V-A	V-A			V-A											
`11899a43-2b68-4a76-92e3-a3d6ad8c26ce[v1.0]` (???)			V-A														
`11f25515-c879-400a-989e-b074d5f092fe[v1.0]` (???)			V-A														
`1e665584-40fe-4450-8f6e-802362399694[v1.0]` (???)			V-A														
`484809d6-4239-471b-b5bc-61df8c23ac48[v1.0]` (from lsm.exe)			V-A														
`88143fd0-c28d-4b2b-8fef-8d882f6a9390[v1.0]` (???)			V-A														
`300f3532-38cc-11d0-a3f0-0020af6b0add[v1.2]` (trkwks)					X-A	X-A	X-A	A	X-A			X-A	X-A	X-A			
`6bffd098-a112-3610-9833-46c3f87e345a[v1.0]` (wkssvc)				V-A	A	X-A	X-A	X-A	X-A								
`8fb6d884-2388-11d0-8c35-00c04fda2795[v4.1]` (w32time)				V-A	A	X-A	X-A	X-A	X-A								
`000001a0-0000-0000-c000-000000000046[v0.0]` (ISystemActivator)				V-A	A	A	X-A	A	X-A							A	
`00000132-0000-0000-c000-000000000046[v0.0]` ((used by rpcss.dll))				V-A	A	A	X-A	A	A	V-A							
`00000134-0000-0000-c000-000000000046[v0.0]` ((used by rpcss.dll))				V-A	A	A	X-A	A	A	V-A	X-A	V-A					
`00000131-0000-0000-c000-000000000046[v0.0]` (from ole32.dll)				V-A	A	A	X-A	A	A	V-A	X-A	V-A					
`00000143-0000-0000-c000-000000000046[v0.0]` (from ole32.dll)				V-A	A	A	X-A	A	A	V-A	X-A	V-A					
`18f70770-8e64-11cf-9af1-0020af6e72f4[v0.0]` (ole32 (IOrCallback))				V-A	A	A	X-A	A	A	V-A	X-A	V-A					
`0d72a7d4-6148-11d1-b4aa-00c04fb66ea0[v1.0]` (ICertProtect)					X-A	X-A	X-A	X-A	A	V-A							
`2f5f6520-ca46-1067-b319-00dd010662da[v1.0]` (tapisrv)					X-A	X-A	X-A	X-A	A	V-A							
`2f5f6521-cb55-1059-b446-00df0bce31db[v1.0]` (unimdm)					X-A	X-A	X-A	X-A	A	V-A							
`f50aac00-c7f3-428e-a022-a6b71bfb9d43[v1.0]` (ICatDBSvc)					X-A	X-A	X-A	X-A	A	V-A							
`06bba54a-be05-49f9-b0a0-30f790261023[v1.0]` (wscsvc)					X-A	X-A	X-A	X-A	X-A			V-A					
`3c4728c5-f0ab-448b-bda1-6ce01eb0a6d5[v1.0]` (dhcpcsvc (RpcSrvDHCPC))					X-A	X-A	X-A	X-A	X-A			V-A					
`3faf4738-3a21-4307-b46c-fdda9bb8c0d5[v1.1]` (AudioSrv)					X-A	X-A	X-A	X-A	X-A			V-A					
`0a74ef1c-41a4-4e06-83ae-dc74fb1cdd53[v1.0]` (taskeng (idletask))					X-A	A	X-A	X-A	X-A								
`12b81e99-f207-4a4c-85d3-77b42f76fd14[v1.0]` (seclogon (ISeclogon))					X-A	A	X-A	X-A	X-A								
`1ff70682-0a51-30e8-076d-740be8cee98b[v1.0]` (atsvc)					X-A	A	X-A	X-A	X-A								
`20610036-fa22-11cf-9823-00a0c911e5df[v1.0]` (rasmans)					X-A	A	X-A	X-A	X-A								
`378e52b0-c0a9-11cf-822d-00aa0051e40f[v1.0]` (sasec)					X-A	A	X-A	X-A	X-A								
`4b324fc8-1670-01d3-1278-5a47bf6ee188[v3.0]` (from srvsvc.dll, Netr*)					X-A	A	X-A	X-A	X-A								
`63fbe424-2029-11d1-8db8-00aa004abd5e[v1.0]` (SensApi)					X-A	A	X-A	X-A	X-A								
`6bffd098-a112-3610-9833-012892020162[v0.0]` (from browser.dll, I_Browserr*, NetrBrowser*)					X-A	A	X-A	X-A	X-A								
`621dff68-3c39-4c6c-aae3-e68e2c6503ad[v1.0]` (wzcsvc (winwzc))					X-A	X-A	X-A	X-A	X-A								

(Continued on next page)

Interface	lsarpc,samr,netlogon	lsass,protected_storage	LSM_API_service	W32TIME_ALT	wkssvc	atsvc,ROUTER,browser	srvsvc	trkwks	keysvc	tapsrv	InitShutdown	eventlog	ntsvcs	scerpc	plugplay	epmapper	MsFteWds
629b9f66-556c-11d1-8dd2-00aa004abd5e[v3.0] (SENSNotify)					X-A	X-A	X-A	X-A	X-A								
45776b01-5956-4485-9f80-f428f7d60129[v2.0] (from dnsrslvr.dll)									V-A	V-A							
68b58241-c259-4f03-a2e5-a2651dcbc930[v1.0] (???)									V-A	V-A							
aa411582-9bdf-48fb-b42b-faa1eee33949[v1.0] (???)									V-A	V-A							
c33b9f46-2088-4dbc-97e3-6125f127661c[v1.0] (???)									V-A	V-A							
4b112204-0e19-11d3-b42b-0000f81feb9f[v1.0] (????)				V-A	V-A												
7ea70bcf-48af-4f6a-8968-6a440754d5fa[v1.0] (nsisvc)				V-A	V-A												
c8cb7687-e6d3-11d2-a958-00c04f682e16[v1.0] (WebClnt (davclntrpc))				V-A	V-A												
3c4728c5-f0ab-448b-bda1-6ce01eb0a6d6[v1.0] (dhcpcsvc6)												V-A					
c386ca3e-9061-4a72-821e-498d83be188f[v1.1] (???)												V-A					
f6beaff7-1e19-4fbb-9f8f-b89e2018337c[v1.0] (???)												V-A					
82273fdc-e32a-18c3-3f78-827929dc23ea[v0.0] (eventlog, from wevtsvc.dll)												A	X-A	X-A			
367abb81-9844-35f1-ad32-98f038001003[v2.0] (Services Control Manager (SCM))												X-A	A	X-A			
93149ca2-973b-11d1-8c39-00c04fb984f9[v0.0] (scesrv)												X-A	A	X-A			
3dde7c30-165d-11d1-ab8f-00805f14db40[v1.0] (BackupKey)												X-A	X-A	X-A			
8d9f4e40-a03d-11ce-8f69-08003e30051b[v1.0] (umpnpmgr)												X-A	X-A	X-A	V-A		
9b8699ae-0e44-47b1-8e7f-86a461d7ecdc[v0.0] (???)															V-A		
c13d3372-cc20-4449-9b23-8cc8271b3885[v1.0] (???)											V-A				V-A		
76f226c3-ec14-4325-8a99-6a46348418ae[v1.0] (???)											V-A						
76f226c3-ec14-4325-8a99-6a46348418af[v1.0] (???)											V-A						
d95afe70-a6d5-4259-822e-2c84da1ddb0d[v1.0] (???)											V-A						
894de0c0-0d55-11d3-a322-00c04fa321a1[v1.0] (InitShutdown)											A						
a002b3a0-c9b7-11d1-ae88-0080c75e4ec1[v1.0] ((used by MigAutoPlay.exe and wuaueng.dll))											X-A						
326731e3-c1c0-4a69-ae20-7d9044a4ea5c[v1.0] (profsvc (IUserProfile))						V-A					X-A						
201ef99a-7fa0-444c-9399-19ba84f12a1a[v1.0] (?AppInfo)						V-A											
2eb08e3e-639f-4fba-97b1-14f878961076[v1.0] (???)						V-A											
5f54ce7d-5b79-4175-8584-cb65313a0e98[v1.0] (?AppInfo??)						V-A											
86d35949-83c9-4044-b424-db363231fd0c[v1.0] (???)						V-A											
a0bc4698-b8d7-4330-a28f-7709e18b6108[v4.0] (from Sens.dll)						V-A											
a398e520-d59a-4bdd-aa7a-3c1e0303a511[v1.0] (IKEEXT)						V-A											
fd7a0523-dc70-43dd-9b2e-9c5ed48225b1[v1.0] (?AppInfo)						V-A											
00000136-0000-0000-c000-000000000046[v0.0] (ISCMLocalActivator)																A	
0b0a6584-9e0f-11cf-a3cf-00805f68cb1b[v1.1] (localpmp)																A	
1d55b526-c137-46c5-ab79-638f2a68e869[v1.0] (???)																A	
412f241e-c12a-11ce-abff-0020af6e7a17[v0.2] (ISCM)																A	
4d9f4ab8-7d1c-11cf-861e-0020af6e7c57[v0.0] (from rpcss.dll)																A	
99fcfec4-5260-101b-bbcb-00aa0021347a[v0.0] (IOXIDResolver)																A	
b9e79e60-3d52-11ce-aaa1-00006901293f[v0.2] (IROT)																A	
c6f3ee72-ce7e-11d1-b71e-00c04fc3111a[v1.0] (IMachineActivatorControl)																A	
e1af8308-5d1f-11c9-91a4-08002b14a0fa[v3.0] (epmapper)																A	
e60c73e6-88f9-11cf-9af1-0020af6e72f4[v2.0] (ILocalObjectExporter)																A	
64fe0b7f-9ef5-4553-a7db-9a1975777554[v1.0] (???)																V-A	
0767a036-0d22-48aa-ba69-b619480f38cb[v1.0] (?PcaSvc)								V-A									
654976df-1498-4056-a15e-cb4e87584bd8[v1.0] (????)								V-A									
b58aa02e-2884-4e97-8176-4ee06d794184[v1.0] (sysmain)								V-A									

Comparing our authenticated procedure access results to the results from the null session (Table II), we find that, partly due to more pipes being accessible, we can get to many more interfaces and procedures. Using an authenticated session, we could access all the interfaces that we could access from the null session, with one exception.

The following lists the additional procedures that were successfully called over an authenticated session, compared to those in Figure 7. The dagger symbol (†) denotes calls that could only succeed from XP and the double dagger (‡) denotes calls that could only succeed from Vista.

`0b0a6584-9e0f-11cf-a3cf-00805f68cb1b[v1.1]` (localpmp):

- OpenEndpointMapper†
- AllocateReservedIPPort†
- ept_insert_ex†
- ept_delete_ex†
- SetRestrictRemoteClients†
- ResetWithNoAuthException†

`12345778-1234-abcd-ef00-0123456789ab[v0.0]` (LSA access (lsarpc)):

- LsarDelete
- LsarSetSecurityObject
- LsarSetInformationPolicy
- LsarCreateAccount
- LsarCreateTrustedDomain
- LsarCreateSecret
- LsarAddPrivilegesToAccount
- LsarRemovePrivileges-FromAccount
- EfsSsoOnReconnect_WL
- LsarSetSystemAccessAccount
- LsarSetInformation-TrustedDomain
- LsarOpenSecret
- LsarSetSecret
- LsarQuerySecret
- LsarDeleteObject
- LsarAddAccountRights
- LsarRemoveAccountRights
- LsarSetTrustedDomainInfo
- LsarDeleteTrustedDomain
- LsarStorePrivateData
- LsarRetrievePrivateData
- LsarSetInformationPolicy2
- LsarSetTrustedDomain-InfoByName
- LsarCreateTrustedDomainEx
- LsarSetDomainInformation-Policy
- LsaITestCall
- LsarCreateTrustedDomainEx2
- CredrWrite†
- CredrRead†
- CredrEnumerate†
- CredrWriteDomain-Credentials†
- CredrReadDomain-Credentials†
- CredrDelete†
- CredrGetTargetInfo†
- CredrProfileLoaded†
- CredrGetSessionTypes†
- LsarRegisterAuditEvent†
- LsarGenAuditEvent†
- LsarUnregisterAuditEvent†
- LsarSetForestTrustInformation
- CredrRename†
- LsarLookupSids3†
- LsarLookupNames4†
- LsarOpenPolicySce†

`1ff70682-0a51-30e8-076d-740be8cee98b[v1.0]` (atsvc):

- NetrJobAdd
- NetrJobDel
- NetrJobEnum
- NetrJobGetInfo

`300f3532-38cc-11d0-a3f0-0020af6b0add[v1.2]` (trkwks):

- Stubold_LnkMendLink
- Stubold_LnkSearchMachine
- StubLnkCallSvrMessage
- StubLnkSetVolumeId
- StubLnkRestartDc-Synchronization
- StubGetVolumeTracking-Information
- StubGetFileTracking-Information
- StubTriggerVolumeClaims
- StubLnkOnRestore
- StubLnkMendLink
- Stubold2_LnkSearchMachine
- StubLnkCallSvrMessage
- StubLnkSearchMachine

`367abb81-9844-35f1-ad32-98f038001003[v2.0]` (Services Control Manager (SCM)):

- RCloseServiceHandle
- RControlService
- RDeleteService
- RLockServiceDatabase
- RQueryServiceObjectSecurity
- RSetServiceObjectSecurity
- RQueryServiceStatus
- RSetServiceStatus
- RUnlockServiceDatabase
- RNotifyBootConfigStatus
- RI_ScSetServiceBitsW
- RChangeServiceConfigW
- RCreateServiceW
- REnumDependentServicesW
- REnumServicesStatusW
- ROpenSCManagerW
- ROpenServiceW
- RQueryServiceConfigW
- RQueryServiceLockStatusW
- RStartServiceW
- RGetServiceDisplayNameW
- RGetServiceKeyNameW
- RI_ScSetServiceBitsA
- RChangeServiceConfigA
- RCreateServiceA
- REnumDependentServicesA
- REnumServicesStatusA
- ROpenSCManagerA
- ROpenServiceA
- RQueryServiceConfigA
- RQueryServiceLockStatusA
- RStartServiceA
- RGetServiceDisplayNameA
- RGetServiceKeyNameA
- RI_ScGetCurrentGroupStateW
- REnumServiceGroupW
- RChangeServiceConfig2A
- RChangeServiceConfig2W
- RQueryServiceConfig2A
- RQueryServiceConfig2W
- RQueryServiceStatusEx
- REnumServicesStatusExA
- REnumServicesStatusExW
- RI_ScSendTSMessage
- RCreateServiceWOW64A‡
- RCreateServiceWOW64W‡
- RI_ScQueryServiceTagInfo‡
- RNotifyServiceStatusChange‡
- RGetNotifyResults‡
- RCloseNotifyHandle‡
- RControlServiceExA‡
- RControlServiceExW‡
- RI_ScSendPnPMessage‡
- RI_ScValidatePnPService‡
- RI_ScOpenServiceStatus-Handle‡
- RI_ScQueryServiceConfig‡

`3dde7c30-165d-11d1-ab8f-00805f14db40[v1.0]` (BackupKey):

- s_BackuprKey†

`4b324fc8-1670-01d3-1278-5a47bf6ee188[v3.0]` (from srvsvc.dll, Netr*):

- NetrCharDevQGetInfo
- NetrCharDevGetInfo
- NetrCharDevControl
- NetrCharDevQEnum
- NetrCharDevQGetInfo
- NetrCharDevQPurge
- NetrCharDevControl
- NetrConnectionEnum
- NetrFileEnum
- NetrFileGetInfo
- NetrFileClose
- NetrSessionEnum
- NetrSessionDel
- NetrShareAdd
- NetrShareEnum
- NetrShareGetInfo
- NetrShareSetInfo
- NetrShareDel
- NetrShareDelSticky
- NetrShareCheck
- NetrServerGetInfo
- NetrServerSetInfo
- NetrServerDiskEnum
- NetrServerStatisticsGet
- NetrServerTransportAdd
- NetrServerTransportEnum
- NetrServerTransportDel
- NetrRemoteTOD
- I_NetrServerSetServiceBits
- NetprPathType
- NetprPathCanonicalize
- NetprPathCompare
- NetprNameValidate
- NetprNameCanonicalize
- NetprNameCompare
- NetrShareEnumSticky
- NetrShareDelStart
- NetrShareDelCommit
- NetrpGetFileSecurity
- NetrpSetFileSecurity
- NetrServerTransportAddEx
- I_NetrServerSetServiceBitsEx
- NetrDfsGetVersion
- NetrDfsCreateLocalPartition
- NetrDfsDeleteLocalPartition
- NetrDfsSetLocalVolumeState
- NetrDfsSetServerInfo
- NetrDfsCreateExitPoint
- NetrDfsDeleteExitPoint
- NetrDfsModifyPrefix
- NetrDfsFixLocalVolume
- NetrDfsManagerReport-SiteInfo
- NetrServerTransportDelEx
- NetrServerAliasAdd‡
- NetrServerAliasEnum‡
- NetrServerAliasDel‡
- NetrShareDelEx‡

`6bffd098-a112-3610-9833-012892020162[v0.0]` (from browser.dll, I_Browserr*, NetrBrowser*):

- I_BrowserrServerEnum
- I_BrowserrDebugCall
- I_BrowserrQueryOther-Domains
- I_BrowserrResetNetlogonState
- I_BrowserrDebugTrace
- I_BrowserrQueryStatistics
- I_BrowserrResetStatistics
- NetrBrowserStatisticsClear
- NetrBrowserStatisticsGet
- I_BrowserrSetNetlogonState
- I_BrowserrQueryEmulated-Domains
- I_BrowserrServerEnumEx

`6bffd098-a112-3610-9833-46c3f87e345a[v1.0]` (wkssvc):

- NetrWkstaGetInfo
- NetrWkstaSetInfo
- NetrWkstaUserEnum
- NetrWkstaUserGetInfo
- NetrWkstaUserSetInfo
- NetrWkstaTransportEnum
- NetrWkstaTransportAdd
- NetrWkstaTransportDel
- NetrUseAdd
- NetrUseGetInfo
- NetrUseDel
- NetrUseEnum
- NetrValidateName
- NetrWorkstationStatisticsGet
- I_NetrLogonDomainNameAdd
- NetrGetJoinableOUs
- NetrUnjoinDomain
- NetrValidateName
- NetrGetJoinInformation
- NetrGetJoinableOUs
- NetrJoinDomain2
- NetrUnjoinDomain2
- NetrRenameMachineIn-Domain2
- NetrValidateName2
- NetrGetJoinableOUs2
- NetrAddAlternateComputer-Name
- NetrRemoveAlternate-ComputerName
- NetrSetPrimaryComputer-Name
- NetrEnumerateComputer-Names
- (proc31, name not known)†

`82273fdc-e32a-18c3-3f78-827929dc23ea[v0.0]` (eventlog, from wevtsvc.dll):

- ElfrClearELFW
- ElfrBackupELFW
- ElfrCloseEL
- ElfrDeregisterEventSource
- ElfrNumberOfRecords
- ElfrOldestRecord
- ElfrChangeNotify
- ElfrOpenELW
- ElfrRegisterEventSourceW
- ElfrOpenBELW
- ElfrReadELW
- ElfrReportEventW
- ElfrClearELFA
- ElfrBackupELFA
- ElfrOpenELA
- ElfrRegisterEventSourceA
- ElfrOpenBELA
- ElfrReadELA
- ElfrReportEventA
- ElfrRegisterClusterSvc
- ElfrDeregisterClusterSvc
- ElfrWriteClusterEvents
- ElfrGetLogInformation
- ElfrFlushEL
- ElfrReportEventAndSourceW‡

`894de0c0-0d55-11d3-a322-00c04fa321a1[v1.0]` (InitShutdown):

- s_BaseInitiateShutdown†
- s_BaseAbortShutdown†
- s_BaseInitiateShutdownEx†

8d9f4e40-a03d-11ce-8f69-08003e30051b[v1.0] (umpnpmgr):

- PNP_Disconnect†
- PNP_GetVersion†
- PNP_GetGlobalState†
- PNP_InitDetection†
- PNP_ReportLogOn†
- PNP_ValidateDeviceInstance†
- PNP_GetRootDeviceInstance†
- PNP_GetRelatedDeviceInstance†
- PNP_EnumerateSubKeys†
- PNP_GetDeviceList†
- PNP_GetDeviceListSize†
- PNP_GetDepth†
- PNP_GetDeviceRegProp†
- PNP_SetDeviceRegProp†
- PNP_GetClassInstance†
- PNP_CreateKey†
- PNP_DeleteRegistryKey†
- PNP_GetClassCount†
- PNP_GetClassName†
- PNP_DeleteClassKey†
- PNP_GetInterfaceDeviceAlias†
- PNP_GetInterfaceDeviceList†
- PNP_GetInterfaceDeviceListSize†
- PNP_RegisterDeviceClassAssociation†
- PNP_UnregisterDeviceClassAssociation†
- PNP_GetClassRegProp†
- PNP_SetClassRegProp†
- PNP_CreateDevInst†
- PNP_DeviceInstanceAction†
- PNP_GetDeviceStatus†
- PNP_SetDeviceProblem†
- PNP_DisableDevInst†
- PNP_UninstallDevInst†
- PNP_AddID†
- PNP_RegisterDriver†
- PNP_QueryRemove†
- PNP_RequestDeviceEject†
- PNP_IsDockStationPresent†
- PNP_RequestEjectPC†
- PNP_HwProfFlags†
- PNP_GetHwProfInfo†
- PNP_AddEmptyLogConf†
- PNP_FreeLogConf†
- PNP_GetFirstLogConf†
- PNP_GetNextLogConf†
- PNP_GetLogConfPriority†
- PNP_AddResDes†
- PNP_FreeResDes†
- PNP_GetNextResDes†
- PNP_GetResDesData†
- PNP_GetResDesDataSize†
- PNP_ModifyResDes†
- PNP_DetectResourceConflict†
- PNP_QueryResConfList†
- PNP_SetHwProf†
- PNP_QueryArbitratorFreeData†
- PNP_QueryArbitratorFreeSize†
- PNP_RunDetection†
- PNP_RegisterNotification†
- PNP_UnregisterNotification†
- PNP_GetCustomDevProp†
- PNP_GetVersionInternal†
- PNP_GetBlockedDriverInfo†
- PNP_GetServerSideDeviceInstallFlags†

8fb6d884-2388-11d0-8c35-00c04fda2795[v4.1] (w32time):

- s_W32TimeSync†
- s_W32TimeGetNetlogonServiceBits†
- s_W32TimeQueryProviderStatus†

93149ca2-973b-11d1-8c39-00c04fb984f9[v0.0] (scesrv):

- SceSvcRpcQueryInfo†
- SceSvcRpcSetInfo†
- SceRpcSetupUpdateObject†
- SceRpcSetupMoveFile†
- SceRpcGenerateTemplate†
- SceRpcConfigureSystem†
- SceRpcGetDatabaseInfo†
- SceRpcGetObjectChildren†
- SceRpcOpenDatabase†
- SceRpcCloseDatabase†
- SceRpcGetDatabaseDescription†
- SceRpcGetDBTimeStamp†
- SceRpcGetObjectSecurity†
- SceRpcGetAnalysisSummary†
- SceRpcAnalyzeSystem†
- SceRpcUpdateDatabaseInfo†
- SceRpcUpdateObjectInfo†
- SceRpcStartTransaction†
- SceRpcCommitTransaction†
- SceRpcRollbackTransaction†
- SceRpcGetServerProductType†
- SceSvcRpcUpdateInfo†
- SceRpcCopyObjects†
- SceRpcSetupResetLocalPolicy†
- SceRpcNotifySaveChangesInGP†
- SceRpcControlNotificationQProcess†
- SceRpcBrowseDatabaseTable†
- SceRpcGetSystemSecurity†
- SceRpcSetSystemSecurityFromHandle†
- SceRpcSetDatabaseSetting†
- SceRpcGetDatabaseSetting†
- SceRpcConfigureConvertedFileSecurityImmediately†

e1af8308-5d1f-11c9-91a4-08002b14a0fa[v3.0] (epmapper):

- ept_mgmt_delete†
- ept_lookup
- ept_map
- ept_lookup_handle_free
- ept_mgmt_delete†
- ept_map_auth‡
- ept_map_auth_async‡

Only one procedure exists that we do not know the name of; from Vista, 6bffd098-a112-3610-9833-46c3f87e345a (wkssvc) has procedures 0–31 callable, but only 0–30 are callable from XP (#31 yields a range error). In addition, wkssvc.dll, the binary that contains the procedures, only has 0–30 listed—at least in the usual way; it remains unclear how there could be a response to procedure #31, but multiple pipes obtained the same result from XP (SMB).

Authenticated named pipe sessions have access to a superset of the interfaces visible through TCP 135. Among interfaces visible in both, the RPC management interface and localpmp had more successful procedures in authenticated pipe sessions. However IOXIDResolver had been successfully called via port 135, but we only get ACCESS_DENIED over authenticated pipes. From this we can deduce that no strict ordering exists based on privilege.

For calls to interfaces that were accessible in both session types, we did not gain any access denied cases, and in several cases, authentication eliminated access denied. For c681d488-d850-11d0-8c52-00c04fd90f7e though, XP over an authenticated session produced a range error for procedure 14 and higher, whereas in all other named pipe cases, there was consistently either success or BAD_STUB_DATA for procedures 14–20 (EfsUsePinForEncryptedFiles, EfsRpcAddUsersToFileEx, EfsRpcFileKeyInfoEx, EfsRpcGenerateEfsStream, EfsRpcGetEncryptedFileMetadata, EfsRpcSetEncryptedFileMetadata, EfsRpcFlushEfsCache) and range errors only above procedure 20; perhaps these are not relevant to XP over authenticated sessions. There were more cases where authenticated XP (but not Vista) gained access to a call that was denied from null sessions, perhaps due to applicability or stronger access requirements. On the epmapper UUID (e1af8308-5d1f-11c9-91a4-08002b14a0fa) from Vista, we got the only CANT_PERFORM (1752) results that we have seen; these are on procedures 0, 1, 5, and 6, which are all named ept_mgmt_delete. 1752 means, “the server endpoint cannot perform the operation”, but there is still uncertainty as to the reason. Under XP, these received BAD_STUB_DATA.

The majority of the interfaces that we list in tables II and III uniformly return ACCESS_DENIED for all procedures, in all cases where they are available. Some of these results stem from the fact that we only saw client use of the UUID on Vista (for example, 00000132-0000-0000-c000-000000000046, 00000132-0000-0000-c000-000000000046, and a002b3a0-c9b7-11d1-ae88-0080c75e4ec1). Others may require a higher level of privilege, or may require additional services to be active on the Vista server.

Figure 6 (page 14) is like Table III, but results show up only if they produced either success or BAD_STUB_DATA. This makes the table simpler, reducing the list of UUIDs to 19. That list does not include IOXIDResolver, which only succeeded across TCP 135.

APPENDIX XXVII
TRANSITION TRAFFIC

We observed the traffic from Vista that occurred when we started up, shut down, and changed the static IPv4 address of a clean Vista install. We summarize the traffic observed here. There was no traffic from logging in.

A. Vista Starting Up

These types of messages were observed from a Vista host starting up. Key factors suspected of affecting this traffic are the fact that the host is on an isolated network and that the host has a statically configured IPv4 address. We observe that IPv4 and IPv6 traffic seems tightly coupled; for example, corresponding MLDv2 and IGMPv3 subscribes or unsubscribes will appear at around the same time. Multicast Listener Discovery version 2 (MLDv2) is the IPv6 equivalent of IGMPv3 and sits on top of ICMPv6. For both IGMP and MLD, we see the interesting pattern of unsubscribing from a multicast address immediately prior to subscribing to it.

ARP:

- broadcast query for ⟨statically configured IPv4 address⟩, tell 0.0.0.0

NDP:

- Neighbor Solicitation for ⟨last used IPv6 link local address⟩
- Router Solicitation

MLDv2:

- subscribe ⟨the solicited nodes multicast address for the last used IPv6 link local address⟩
- subscribe/unsubscribe ff02::1:3 (IPv6 multicast address used by LLMNR)
- subscribe ff02::c (IPv6 multicast address used by SSDP)

IGMPv3:

- subscribe/unsubscribe 224.0.0.252 (IPv4 multicast address used by LLMNR)
- subscribe 239.255.255.250 (IPv4 multicast address used by UPnP)

LLMNR: (to both ff02::1:3 and 224.0.0.252 port 5355)

- type A and AAAA queries for ⟨defined hostname⟩
- type A and AAAA queries for “isatap”
- type A and AAAA queries for “wpad”

NBNS:

- register ⟨defined hostname⟩ <00>
- register “workgroup<00>”
- query “isatap<00>”
- query “wpad<00>”

B. Vista Shutting Down

From our observations, shutting down a Vista host yields the following message types:

MLDv2:

- unsubscribe ff02::c
- subscribe/unsubscribe ff02::1:3

IGMPv3:

- unsubscribe 239.255.255.250
- subscribe/unsubscribe 224.0.0.252

NBNS:

- release “workgroup<00>”
- release ⟨defined hostname⟩ <00>

BROWSER:

- host announcement for ⟨defined hostname⟩

LLMNR: (to both ff02::1:3 and 224.0.0.252 port 5355)

- type A and AAAA queries for ⟨defined hostname⟩

UPnP:

- bye messages

C. Vista Changing Static IPv4 Addresses

Changing the statically configured IPv4 address of a Vista host produces a variety of messages. These were seen when the Vista host was still using the old IPv4 address:

ARP:
- broadcast query for ⟨new IPv4 address⟩, tell 0.0.0.0

NDP:
- Neighbor Solicitation for ⟨IPv6 address⟩ by peer Vista host
- Neighbor Advertisement for ⟨IPv6 address⟩

MLDv2:
- subscribe/unsubscribe ff02::1:3

LLMNR: (to ff02::1:3 port 5355)
- type A query for “isatap”
- type A and AAAA query for ⟨defined hostname⟩

WSD: (TCP port 5357)
- connect by peer Vista host (connection established)
- post of data by peer Vista host
- response with data to peer Vista host

UPnP:
- hello message to ff02::c
- resolve message to ff02::c
- resolve matches message IPv6 from peer Vista host
- resolve message to 239.255.255.250 and ff02::c by peer Vista host
- resolve matches message IPv6 to peer Vista host

SSDP: (to ff02::c)
- discover WFADevice, UPnP rootdevice, nhed:presence, MediaCenterExtender

The following were seen after the Vista host started to use the new address:

ARP:
- broadcast query for ⟨new IPv4 address⟩, tell ⟨peer Vista host⟩
- reply ⟨new IPv4 address⟩ is ⟨MAC address⟩

NDP:
- Neighbor Solicitation for ⟨IPv6 address⟩ by peer Vista host
- Neighbor Solicitation for ⟨peer Vista host⟩
- Neighbor Advertisement for ⟨IPv6 address⟩
- Neighbor Advertisement for ⟨peer Vista host⟩

MLDv2:
- subscribe/unsubscribe ff02::1:3 (IPv6 multicast address used by LLMNR)

IGMPv3:
- subscribe/unsubscribe 224.0.0.252 (IPv4 multicast address used by LLMNR)

NBNS:
- register ⟨defined hostname⟩ <00>
- query “isatap<00>”
- register workgroup<00>

LLMNR: (to both ff02::1:3 and 224.0.0.252 port 5355)
- type A query for “isatap”
- type A and AAAA query for ⟨defined hostname⟩

WSD: (TCP port 5357)
- post of data by peer Vista host
- response with data to peer Vista host
- close down connection

UPnP: (sent to 239.255.255.250 and ff02::c)
- hello message

SSDP:

- discover InternetGatewayDevice (to 239.255.255.250)
- discover WFADevice, UPnP rootdevice, nhed:presence, MediaCenterExtender (to ff02::c and 239.255.255.250)

APPENDIX XXVIII
UNSOLICITED TRAFFIC

We observed the network traffic of a clean installation of Windows Vista over a couple weeks. The following is the "unsolicited" traffic that we saw. That is, traffic that was not a result of an explicit user request, to the best of our knowledge. Traffic was captured from the Linux host (see Appendix I-A). The following is a summary of all traffic observed:

LLTD (may have been solicited)

ARP:

- request to broadcast addr and to a previously used MAC
- reply

NDP:

- NS for another Vista neighbor IPv6 address
- NS for the IPv6 address we want
- NA of our IPv6 address (reply to other Vista)
- RS

MLDv2: (includes a hop-by-hop header with router alert for MLD)

- subscribe/unsubscribe ff02::1:3 (LLMNR)
- subscribe/unsubscribe ff02::c (SSDP)
- subscribe to solicited nodes addresses for our IPv6 address

IGMPv3:

- subscribe/unsubscribe 224.0.0.252 (LLMNR)
- subscribe/unsubscribe 239.255.255.250 (UPnP)

NBNS: (over IPv4)

- query for isatap<00>
- query for wpad<00>
- query ⟨neighbor hostname⟩ <20> plus response
- query WORKGROUP<1e>
- register ⟨our hostname⟩ <00>
- register workgroup<*>
- register <01><02>__MSBROWSE__<02><01>
- release ⟨our hostname⟩ <00>
- release workgroup<00>

BROWSER:

- announce hostname as workstation, server, NT workstation, potential browser
- get backup list request
- browser election request
- request announcement of self

LLMNR:

- type A query for own hostname (over IPv4,IPv6)
- type AAAA query for own hostname (over IPv4,IPv6)
- type A query for isatap (over IPv4,IPv6)
- type A query for wpad (over IPv4,IPv6)

WSD: (TCP port 5357)

- connect to local peer, pull info/reply with info

UPnP:

- bye (IPv4,IPv6)
- hello (IPv4,IPv6)
- resolve (IPv4,IPv6)
  - plus resolve matches (reply to another Vista)
- Probe (IPv4,IPv6)
  - plus probe matches (reply to another Vista)

SSDP:

- requests:
  - InternetGatewayDevice (over IPv4)

- MediaCenterExtender (over IPv4,IPv6)
- schemas-microsoft-com:nhed:presence:1 (over IPv4,IPv6)
- UPnP rootdevice (over IPv4,IPv6)
- WFADevice (over IPv4,IPv6)

Most of the time, the Vista hosts were on an isolated network. However, they were individually connected to an Internet-connected network in order to complete Windows Activation on the system and complete its licensing. In at least one case, we observed that a Teredo address was configured by the time activation was completed. On a separate occasion, when Vista was being installed, the host was accidentally connected to an Internet-connected network; before we realized it was on the Internet instead of the isolated network, a Teredo address was established. As we had made no configuration changes and no installations, this action seems to contradict Microsoft's own statements on their web site[36], at the time of writing. Microsoft states that "In Windows Vista, the Teredo component is enabled but inactive by default. In order to become active, a user must either install an application that needs to use Teredo, or configure advanced Windows Firewall filter settings to allow edge traversal." In fact, we had not made any Internet requests.

We expect that we would see DHCP traffic if we had opted for DHCP-assigned addresses. Similarly, there may have been other protocols, if the hosts were not on an isolated network.