Embedding Covert Channels into TCP/IP

Steven J. Murdoch and Stephen Lewis

University of Cambridge, Computer Laboratory,
15 JJ Thomson Avenue, Cambridge CB3 0FD, United Kingdom
http://www.cl.cam.ac.uk/users/{sjm217,srl32}/

Abstract. It is commonly believed that steganography within TCP/IP is easily achieved by embedding data in header fields seemingly filled with "random" data, such as the IP identifier, TCP initial sequence number or the least significant bit of the TCP timestamp. We show that this is not the case; these fields naturally exhibit sufficient structure and non-uniformity to be efficiently and reliably differentiated from unmodified ciphertext. Previous work on TCP/IP steganography does not take this into account and, by examining TCP/IP specifications and open source implementations, we have developed tests to detect the use of naïve embedding. Finally, we describe reversible transforms that map block cipher output into TCP ISNs, indistinguishable from those generated by Linux and OpenBSD. The techniques used can be extended to other operating systems. A message can thus be hidden in such a way that an attacker cannot demonstrate its existence without knowledge of a secret key.

1 Introduction

Steganographic covert channels based on modification of network protocol header values are best understood by considering a scenario with three actors; in keeping with the existing literature, we shall call them Alice, Bob and Walter. Each actor has his/her own capabilities and goals. Alice can make arbitrary modifications to network packets originating from a machine within Walter's network. She wants to leak a message to Bob, who can only monitor packets at the egress points of this network. Alice aims to hide the message from Walter, who can see (but not modify) any packet leaving his network. This is analogous to a passive warden within the threat model introduced in [1].

In a practical instantiation of this problem, Alice and Bob may well be the same person. Consider a machine to which an attacker has unrestricted access for only a short amount of time, and which lies within a closely monitored network. The attacker installs a keylogger on the machine, and wishes to leak passwords to himself in such a way that the owner of the network does not observe that anything untoward is happening. An attacker might also want to watermark all transmissions from a particular machine; the steganography described in this paper can be used for this purpose.

Alice can choose which layer of the protocol stack she wishes to hide her message in. Each layer has its own characteristics, which indicate the scenarios

in which it can best be used. In [2], the potential for embedding at all layers of the OSI model is discussed.

At the bottom of the stack, in the Physical and Data-Link layers (e.g. Ethernet), there is some opportunity for embedding data. Physical layer embedding presents problems, however. It requires low-level control of the hardware, which Alice may find difficult to obtain. If she chooses to signal to Bob at this layer, she will find that her messages are stripped out when they reach a device that connects networks at a higher layer (e.g. an IP router). This requires Bob to be on the same LAN. An example of a steganography system that relies on embedding at the Physical layer is described in [3].

Alice might also choose to embed data at the Presentation or Application layers of the network stack (e.g. in Telnet or HTTP/FTP traffic). If, however, she only has brief access to the machine from which she is leaking data, she needs to anticipate which applications are likely to be used on it; she can then modify them to carry her messages in the traffic they generate.

Similarly, the format of files sent over HTTP or FTP (such as JPEG or PDF) may also be viewed as protocols in which steganographic data can be embedded. These provide a high-bandwidth channel to Alice, but only if she is confident of being able to modify these files without arousing suspicion.

The only remaining layers to consider in the OSI model are Network, Transport and Session. TCP and IP (specified in [4] and [5]) fall within these layers, and are common to the vast majority of Internet applications. A message embedded in these protocols has the advantage that it will survive unchanged on its journey out of Walter's network. If Alice designs her embedding correctly, a message transported can thus be sent without suspicion, and received intact.

In this paper we study a number of previously proposed schemes for embedding data within the TCP and IP protocol headers, thus creating a steganographic covert channel. We show how the use of these schemes can easily be detected by a passive warden. The algorithms used in the generation of some TCP/IP header fields are then looked at in detail, and our alternative method for embedding data, *Lathra*, is proposed. We show that a passive warden cannot detect the use of this method without knowledge of a secret key. Our results will also be relevant to the field of operating system fingerprinting.

2 Threat Model

We have thus far assumed that the steganography can only be prevented by detection, not by attempting to remove any hidden information. This is known as the passive warden threat model. An active warden can modify traffic regardless of suspicion. As is shown in [6], an active warden can remove most, if not all, TCP/IP level steganography, and lower layer steganography will have already been removed by routing. He will, however, have difficulty removing steganography at higher layers (e.g. in JPEG images) without damaging the carrier.

In many scenarios it may be infeasible for a warden to be active: the kind of filtering necessary to remove TCP/IP steganography can increase network

latency, and might require a filtering router that can store large amounts of state. The warden may also wish to avoid the users being aware that the use of steganography is suspected.

In this paper, we assume that our attacker, Alice, operates in an environment with a passive warden and an unreliable network (permitting packet loss, duplication and reordering) and requires a TCP/IP based covert channel giving

- *indistinguishability*: Walter (a passive warden) should be unable to detect the presence of the data hidden in packets leaving Alice's machine; and
- *reliability*: she desires some indication of whether her messages to Bob have indeed arrived, so she can retransmit them if necessary.

3 Overview of TCP/IP Based Steganography

A common failing of existing proposals is the production of output from a different distribution to that which would be generated by unmodified TCP/IP implementations. In some cases, it is even outside the relevant specifications. For this reason, to design steganographic techniques or to detect their use, it is necessary to be familiar with the applicable standards and the details of their implementation. This section gives an overview of the TCP/IP standards and related work from a steganographic encoding perspective.

The basic TCP/IP protocol suite is specified in [4] and [5]. There are extensions to it (e.g. the TCP Extensions for High Performance [7]) that specify additional header options; these also give some scope for steganographic coding.

IP itself does not aim to provide any reliability guarantees, but rather allows client protocols on a host to transport blocks of data (datagrams) from a source to a destination, both specified by fixed length addresses. One noteworthy feature of IP for our purposes is that it allows fragmentation and reassembly of long datagrams, requiring certain extra header fields.

TCP, on the other hand, does aim to provide a reliable channel to its clients. It has a stream oriented interface, and keeps its reliability properties even within networks exhibiting packet loss, reordering and duplication. Its features for implementing reliability and flow control give scope for steganographic coding.

The TCP/IP header can serve as a carrier for a steganographic covert channel if a header field can take one of a set of values, each of which appears plausible to our passive warden. The warden should not be able to distinguish whether the header was generated by an unmodified TCP/IP stack or by a steganographic encoding mechanism. In this section we examine which header fields have more than one plausible value, and look at the amount of entropy available in each of them for use by a steganographic coding scheme.

TCP/IP steganography exploits the fact that few headers are altered in transit. As mentioned above, IP packets can be fragmented, but (unless we are hiding data in the fragmentation-related headers) no information is lost. The time-to-live field in the IP header is decremented each time the packet passes through a router, but the initial values used by IP stacks are well known, so this field gives little scope for steganography.

Figure 1 illustrates the base TCP/IP headers. The fields shown in italics are those that may be used to embed steganographic data. We now consider each of these fields in turn, assessing their potential for use as steganographic carriers.

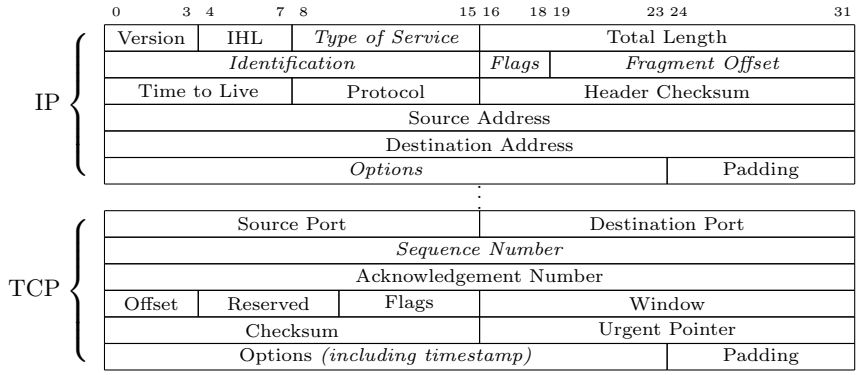

Fig. 1. Basic TCP/IP header structure

3.1 Type of Service

The eight Type of Service (ToS) bits in the IP header are use to indicate quality of service parameters to routers on a packet's path. They are now rarely used with their original semantics (as defined in [5]); they have been reused as, for example, the 'DS field' in [8].

There is potential for using the bits in this field as a steganographic carrier, as described in [2], because many networks never use them. However, this would be easily detected by the warden in our threat model, as the field is set to zero in almost all default operating system configurations.

3.2 IP Identification

As described in [5], the IP Identification field (IP ID) is 'an identifying value assigned by the sender to aid in assembling the fragments of a datagram', and is allocated 16 bits of the IP header. Because the IP ID is used to distinguish fragments making up one packet from fragments making up another, the only constraints on its value are uniqueness over the length of time that fragments of a packet might reasonably remain in a network, and unpredictability.

IP IDs that are unique within a given time window are necessary to ensure that fragments of different packets are not reassembled into one packet on the receiving host. Unpredictability prevents 'idle scanning' [9], whereby an attacker can portscan a host without ever sending a packet directly to it.

A scheme for embedding data in this field is described in [10]. It uses a pseudorandom sequence, generated by a Toral Automorphism System, to ensure

that the modified field is random. However this can be detected since IP ID fields are not random, as shown in Section 5.1.

3.3 IP Flags

IP packets include two flags, *Do Not Fragment* (DF), indicating that the packet should be discarded if it cannot be sent without fragmentation, and *More Fragments* (MF) which is 0 if the packet contains the last fragment, or if a packet has not been fragmented. In [10] the use of the DF bit for steganographic signalling is proposed. If this is used on packets smaller than the maximum segment size the DF flag has no effect on the packets' behaviour. However, the normal state of DF can be predicted from the packet's context, so the warden would detect the use of this technique.

3.4 IP Fragment Offset

When IP packets are fragmented, the individual fragments contain an offset field; this allows the receiving host to reconstruct the fragments in the correct positions in its receive buffers. Information can be transmitted covertly by merely modulating the size of the fragments originated by a host, and thus the fragment offsets. As with the IP identification and ToS fields, this method of steganographic encoding is easily detected. In environments where path MTU discovery [11] is routinely used, fragmented packets are unusual. Fragmentation of a packet on the source host would certainly arouse suspicion.

3.5 IP Options

IP packets very rarely contain 'options', so their steganographic potential is limited. In [2] the use of the IP Timestamp option is described (not to be confused with the TCP Timestamp discussed in Section 3.7), but in addition to being easily detectable, packets with this option present can travel at most 20 hops, so it is of little use in the open Internet.

3.6 TCP Sequence Number

TCP sequence numbers support the reliability features provided by TCP (and to some extent, the flow control features). Each octet of data transmitted over a TCP stream is assigned a sequence number. In TCP, a connection (defined by a pair of sockets) can be reused, and hence the host must be able to detect whether a segment is from a current or previous incarnation of a connection.

When a connection is established, both hosts must choose an *initial sequence number* (ISN). Careful design of the algorithm for generating these initial sequence numbers ensures that overlap in sequence number space between different incarnations of a connection is prevented.

There are other properties required of the algorithm used for initial sequence number generation. For a given connection, the ISNs used must be hard to

guess for those not involved in the connection [12]. To allow a connection in the `TIME_WAIT` state to be restarted, the sequence numbers for a given socket pair should also be monotonically increasing.

A prototype implementation of steganography using TCP ISNs (and also the IP ID), `Covert_TCP`, is described in [13]. It simply replaces the chosen field with the data to be sent, so can be detected either by observing that the field does not meet the required overlap and uniqueness constraints, or by comparing the data observed with statistical patterns of suspected plaintext.

A passive warden using a Support Vector Machine (SVM) is presented in [14]. It is designed to detect the use of `Covert_TCP` within the IP ID and TCP ISN. A SVM is a machine learning technique that is suitable for automatically identifying features which are not well understood. In the case of IP IDs and ISNs, the algorithm for generating them is well understood and precisely described in source code, so it is not necessary to use a machine learning technique. The SVM can only identify simple features, so it cannot detect the complex structure present in these fields and their interdependencies.

The design and implementation of *Nushu*, an improvement to `Covert_TCP` for Linux 2.4, is described in [15]. Nushu uses TCP ISNs for encoding information and encrypts outgoing ISNs to hide the use of steganography, however it still may be detected. Firstly, the output will not exhibit the structure of TCP ISNs expected from Linux. Secondly, a flaw in the use of DES for encryption allows the recovery of statistical information on the plaintext. These techniques will be further discussed in Section 5.3.

3.7 TCP Timestamp

The TCP timestamp option allows a host to accurately measure the round trip time of a path, and also mitigates problems associated with sequence number wrap-around in networks with large bandwidth × delay products. For our purposes, it is only necessary to understand the constraints on the values of TCP timestamps; more details about the features based on them can be found in [7].

The timestamp option consists of two 32 bit fields, TS Value and TS Echo Reply. The TS Value field is set based on the 'timestamp clock' of the sender, and it is into this field that hidden data can be embedded. The only constraints on the timestamp clock are that its tick frequency be between 1 Hz and 1 kHz, and that it be strictly monotonic.

A covert channel based on modulating the least significant bit of the TCP timestamps transmitted by a host, `devcc`, is described in [16]. The scheme works by incrementing the timestamp associated with a packet (and delaying it accordingly) in order to transmit a '1' bit of ciphertext. The use of TCP timestamps is not universal, but it is deployed as standard on newer versions of Linux and other Unix-like operating systems, so the observation of timestamps from an operating system which does not support them would be suspicious. As described in Section 5.3, the distribution of values in the timestamp field is modified from the expected one in a detectable manner by the use of this covert channel technique.

3.8 Packet Order

In addition to the content of the packet, the ordering of packets can be used to carry information, as is described in [10]. This relies on being used on an IPSec network to recover the original order, limiting its applicability. Since packets are seldom reordered by the transmitting host, a warden who is close to Alice will undoubtedly notice the unusually large amount of re-ordering.

4 IP ID and TCP ISN Implementations

The passive warden considered in this paper has knowledge of both the TCP/IP standards and particular implementations. He can check whether the values he observes could have been generated by an unmodified operating system, or even by the operating system he knows to be installed on the originating host.

Two fields which are commonly used to embed steganographic data are the IP ID and TCP ISN. A sufficiently precise description of their generation cannot be found within the public literature, so details of the implementation are included here. Due to their construction, these fields contain some structure, but as mentioned in Section 3.2 and 3.6, they must also be partially unpredictable. This is achieved by having randomly generated, per-host, secrets and by the use of cryptographic functions. We assume that the warden is aware of the implementation, but does not have access to these secrets and is not able to exploit vulnerabilities in the cryptographic primitives.

4.1 Linux

The Linux 2.0 ISN generator (shown in Figure 2) is based on RFC1948 [17]. It uses SHA-1 to hash a block of 16 32-bit words, with words 9–11 set to the source and destination IP address and port, and the remaining 13 words filled with a cryptographically secure, random secret, initialised on boot. Rather than using the values defined in the SHA-1 standard for the initial state, the first 5 words of the block are used. To obtain the ISN, the second word of the hash is selected and the current time (in microseconds) added. This achieves the goals of RFC1948, but calculation of a SHA-1 hash is slow, and hence this algorithm causes a significant delay in the TCP connection establishment process.

The algorithm used in Linux 2.2 (shown on the left in Figure 3) was modified to reduce the time needed to calculate each ISN. Rather than using SHA-1, a reduced block-size variant of MD4 was used, which reads 8 32-bit blocks per iteration, rather than the 16 in the original, and so it also reduces the steps per round from 16 to 8. This is used in a similar way to SHA-1 in Linux 2.0, except the reuse of random data is avoided. Since even the full size MD4 algorithm is known to be insecure, the random data is rekeyed every 300 seconds (5 minutes) to limit the impact of secret compromise. To avoid this resulting in repeated ISNs, after the hash is calculated, the most significant byte is replaced with a counter incremented on rekeying and initialised to the current time divided by 300. Finally, as with Linux 2.0, the time in microseconds is added.

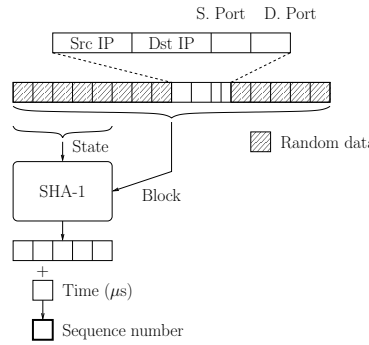

Fig. 2. Linux 2.0 ISN generator

Early versions of Linux 2.4 contained the same ISN generator as Linux 2.2. It was also used (up to the hashing step) with a different secret to initialise the per-destination counters for IP IDs. A global counter previously had been used but this was vulnerable to idle scanning. In later versions of Linux 2.4 and in Linux 2.6 the algorithm was changed slightly, as shown on the right of Figure 3, mainly to improve performance on multiprocessor systems. The difference from a detection perspective is that the rekey counter is initialised to zero on boot. The use of MD4 is changed, and the same secret is used for both ISN and IP ID generation but using this for detection would require exploiting a vulnerability in MD4.

4.2 OpenBSD

The algorithm used for ISN generation in OpenBSD was introduced in December 2000; Figure 4 shows its operation. It is initialised by keying a block cipher with 1024 bits of random data and setting the most significant bit of the generated ISNs to be zero. It is rekeyed every 2 hours, or every time 30,000 connections, whichever is sooner. On rekeying, the MSB of the generated ISNs is toggled: this prevents collisions between ISNs generated in adjacent rekey intervals.

When a new TCP connection is made, the ISN is generated as follows:

- The MSB set to either '1' or '0', depending on whether the operating system is in an 'odd' or 'even' rekey interval.
- The next 15 bits are set to the output of a custom block cipher run in counter mode; the counter is updated each time an ISN is generated.
- The next bit is *always* zero.
- The final 15 bits are generated by an RC4 based pseudorandom number generator (PRNG).

The result of running the block cipher in counter mode is that a different pseudorandom sequence is defined in each rekey interval. The 15-bit values in

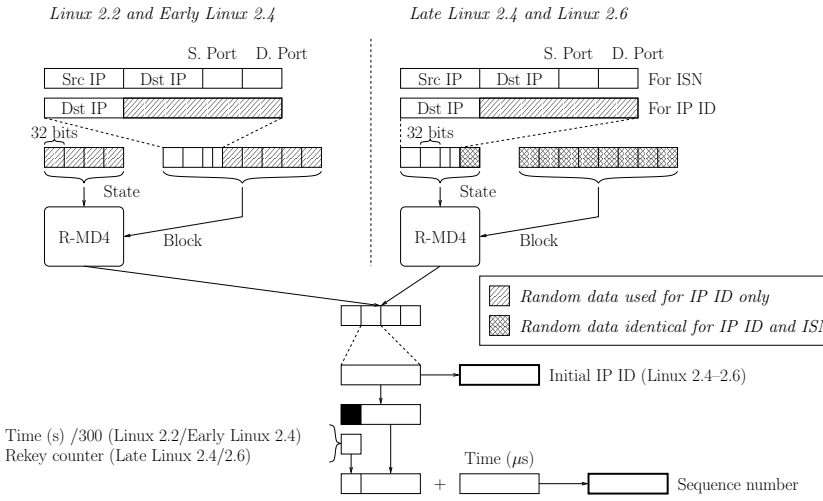

Fig. 3. Linux 2.2–2.6 ISN generator and Linux 2.4–2.6 IP ID generator

this sequence are then inserted into the ISNs, followed by a zero bit: this ensures that no two ISNs within a given rekey interval are closer together than 2^{15} octets. The scheme thus satisfies all of the constraints described in Section 3.6 apart from per socket pair monotonicity.

The IP ID algorithm in OpenBSD uses a linear congruential generator, described in [18], rekeyed every 3 minutes (or after 30,000 IDs have been generated, whichever is sooner). It uses the same MSB-toggling mechanism as the sequence number generator to prevent collisions between rekey intervals.

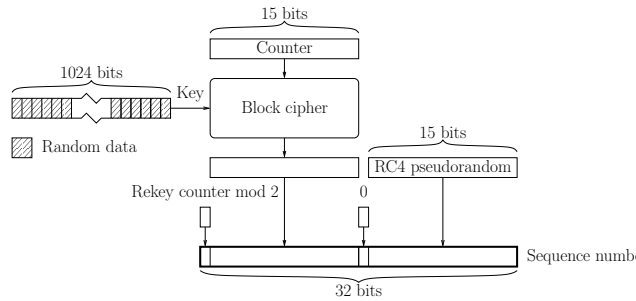

Fig. 4. OpenBSD ISN generator

5 Detection of TCP/IP Steganography

As described above, each operating system exhibits well defined characteristics in generated TCP/IP fields. These can be used to identify any anomalies that may indicate the use of steganography. We have therefore defined a suite of tests which may be applied to network traces and used to identify whether the results are consistent with known operating systems (and in particular with the operating system believed to be installed on the source host).

5.1 IP ID Characteristics

1. *Sequential Global IP ID.* Some operating systems, particularly older ones (e.g. Linux <2.4), use a global counter for the IP ID. If connections to different hosts have sequentially increasing IP IDs then it is likely that this strategy is in use.
2. *Sequential Per-host IP ID.* Others (e.g. Linux ≥2.4) use a per-host counter. The warden can test whether connections to different hosts use apparently unrelated IP IDs, but connections to the same host have a sequentially increasing IP ID.
3. *IP ID MSB Toggle.* OpenBSD toggles the most significant bit of the IP ID every rekey interval (3 minutes or 30,000 IP IDs), so the MSB is be examined to check if it meets this pattern.
4. *IP ID Permutation.* Within a rekey interval, the OpenBSD IP ID is non-repeating; the presence of any duplicates eliminates the possibility that this strategy is in use.

5.2 TCP ISN Characteristics

5. *Rekey Timer.* In Linux 2.2 (and early 2.4) the most significant byte of the ISN is initialised to the current time since the epoch, divided by 300. The system time in microseconds is then added. The rekey timer can be recovered by subtracting the host time, in microseconds, from each ISN and verifying that the top byte increases by one every 5 minutes. This requires a clock synchronised to 8 seconds accuracy (2^{23}/1,000,000). This seems a reasonable assumption as many systems use NTP synchronisation. The host time can even be queried directly, for example by using the daytime service, or indirectly, by observing patterns in the ISNs.
6. *Rekey Counter.* In Linux 2.6 (and late 2.4) the MSB of the ISN is set to the time since system startup (in seconds) divided by 300. The system time in microseconds is added, as before, and hence the rekey counter can be recovered using the same method as in Test 5.
7. *ISN MSB Toggle.* As with the IP ID, OpenBSD toggles the MSB of the generated ISN every rekey interval (2 hours or 30,000 IP IDs).
8. *ISN Permutation.* Bits 16 to 30 within OpenBSD ISNs are non-repeating within a rekey interval.
9. *Zero bit 15.* All ISNs generated by OpenBSD will have bit 15 cleared.

10. *Full TCP Collisions.* In Linux 2.0–2.6, and other RFC1948 inspired systems, the hash used for ISN generation is based on the socket pair, and it is possible that collisions will be encountered. For Linux 2.0 there is no rekeying, so all 32 bits will be identical after subtracting the time.
11. *Partial TCP Collisions.* For Linux 2.2–2.6 it would be expected that collisions within a rekey period will have the same least significant 24 bits, after subtracting the time.

5.3 Explicit Steganography Detection

12. *Nushu Cryptography* As covered in Section 3.6, Nushu encrypts data before including it in the ISN field. This will result in a distribution unlike that normally generated by Linux and so will be detected by the other TCP tests. However due to a flaw in the way that encryption is used, Nushu also exhibits characteristics of its own which may be exploited. The encryption operates by DES encrypting the IV (source port $\oplus$ destination port, source IP address $\oplus$ destination IP address) with a shared key, then xoring the first 32 bits of the resulting keystream with the hidden data. When IV collisions occur, the ISNs can be xored to remove the key-stream; the result is the xor of two plaintexts. If these plaintexts are the same, as is the case when data is not being sent, the result would be zero, and in other cases redundancy in encoding would be apparent.
13. *TCP Timestamp* The scheme used in `devcc`, described in [16], can be detected using the methods outlined in [19]. If a low bandwidth TCP connection is being used to leak information, a randomness test can be applied to the least significant bits of the timestamps in the TCP packets. (For examples of such tests, see [20].) If 'too much' randomness is detected in the LSBs, it can be deduced that a steganographic covert channel is in use.
    For a high bandwidth TCP connection (where segment transmission rate $\gg$ timestamp update rate), a warden can merely calculate the ratio of the number of distinct timestamp values seen to the difference between the start and end timestamp values. If the covert channel described in [16] is in use, this ratio will be close to 0.75; if not, it will be very close to 1.
14. *Other Anomalies* Features which would indicate the use of steganography include: unusual flags (e.g. DF when not expected, ToS set), excessive fragmentation, use of IP options, non-zero padding, unexpected TCP options (e.g. timestamps from operating systems which do not generate them), excessive re-ordering.

5.4 Accuracy

Table 1 shows which tests detect which operating systems. All of these tests (except Test 13) are defined based directly on the implementation and make no assumptions about the probabilistic effects. Hence, they will not suffer from false negatives. False positives are possible, so in this section we consider the number of packets required to avoid these.

Table 1. Expected results of tests on unmodified operating systems and TCP/IP steganography systems. A matching test is indicated by "•". The last three columns are tests for the presence of steganography, the others test for the absence. Nushu and `devcc` were written for early Linux 2.4 and are assumed to share the characteristics of all fields which are not explicitly modified. `Covert_TCP` creates all fields itself. Our improved TCP/IP steganography scheme, Lathra, is described in Section 6

Software	Tests													
	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Linux 2.0	•									•				
Linux 2.2	•				•						•			
Early Linux 2.4		•			•						•			
Late Linux 2.4/2.6		•				•					•			
OpenBSD			•	•			•	•	•					
`Covert_TCP`														
Nushu		•										•		
`devcc`		•			•						•		•	•
Lathra/Linux		•				•					•			
Lathra/OpenBSD			•	•			•	•	•					

IP ID. Test 1 will reach an error probability of $1/2^{16}$ after only 2 packets and Test 2 will also for 2 packets directed to the same host within a rekey interval. The probability of error in Test 3 halves with every packet. From the "birthday paradox", after around 181 packets a collision would be expected which would match Test 4.

TCP ISN. Test 5 needs one packet to achieve a $1/2^8$ error probability and Test 6 needs 2 packets to get the same. Test 7 halves the error probability with every SYN packet. As with the equivalent IP ID check, Test 8 needs around 181 SYN packets within a rekey interval. Tests 10–12 depend on the randomness of the source port selection, but on a heavily loaded machine, our experiments show these collisions occur approximately every 1,000 SYN packets for a fixed destination port.

Steganography. Test 12 also depends on port selection randomness but our experiments show collisions every 1,000 SYN packets, even with random port selection. Test 13 relies on observation of consecutive TCP timestamps: if n timestamps are seen without discontinuity, the probability that the `devcc` scheme is in use is $1/2^{n-1}$. The accuracy of Test 14 depends on the steganography being used, but for naïve implementations only one packet is needed.

6 Detection-Resistant TCP Steganography Schemes

Our aim in this section is to develop a robust scheme, Lathra, for using the TCP ISNs generated by OpenBSD and Linux as a steganographic carrier. We achieve

this by following the implementation of the real ISN generator, so OpenBSD and Linux are discussed as separate cases.

6.1 OpenBSD

The MSB of our output must exactly mimic the output of the real OpenBSD TCP stack: it must toggle every 2 hours (or 30,000 connections). The next 15 bits, when extracted in turn from each ISN generated within a rekey interval, must resemble a pseudorandom sequence. We therefore need a scheme that codes arbitrary data into a pseudorandom sequence of integers between 0 and 2^{15}. The data must still be recoverable even if some elements of this sequence are never seen by the recipient, Bob.

The functions in Figure 5 encode (and decode) the integer n as a permutation of the sequence $(0, 1, 2, \ldots, m)$, with x-times redundancy (i.e. Bob only needs to receive one in x ISNs transmitted by Alice). In order to remove patterns in our permutation, Alice must choose a key, K, for each rekey interval, and transmit it to Bob in, for example, the least significant 15 bits of the first ISN.

These functions allow us to code the 16 most significant bits of the ISN. The 17th bit must be a zero, and we can code data into the remaining 15 bits using a block cipher with a preshared key. The output of the block cipher is analogous to the RC4 based PRNG in the genuine ISN generator. In order to ensure freshness, a hash of the other TCP and IP header fields is xored with the data to be encoded before the encryption step.

6.2 Linux

Simply encoding the data within the least significant 24 bits of the ISN could be detected by the warden, since after subtracting the time he would be left with a non-uniform distribution. Instead we mirror the actual ISN generation process as closely as possible. If we assume that Alice and Bob have clocks synchronised to 0.1 seconds, Alice embeds the 6-bit encrypted message in bits 17–23 of the hash and pads the rest with random data. Then Alice adds the rekey counter and time as normal. Bob simply needs to subtract the time to recover the message.

To ensure reliability, Alice can check if she receives an ACK for each ISN and resend if not, then use a standard reliability protocol, for example that used in Nushu, to reassemble the data. So that the data cannot be differentiated from random numbers, it must be encrypted. To achieve freshness, while allowing each packet to be decrypted independently, the plaintext is xored with a hash of the TCP/IP header, excluding the ISN and other fields which may change during transport, then encrypted with a variable length block cipher. Also, due the the RFC1948 based design, if Alice encounters a packet with the same source and destination IP address and port as one already used, within a rekey interval, this must be skipped.

```
PERMUTATION-CODE(m, n, x)
 1  base ← m
 2  output_symbols ← (0, 1, 2, ..., m)
 3  while n ≠ 0
 4  do index ← n mod base
 5     n ← ⌊n/base⌋
 6     for i ← 0 to x − 1
 7     do output ENCIPHER(output_symbols[index] + i × m, K)
 8     output_symbols ← output_symbols \ output_symbols[index]
 9     base ← base − 1
```

```
PERMUTATION-DECODE(m, x)
 1  base ← m
 2  multiplicand = 1
 3  input_symbols ← (0, 1, 2, ..., m)
 4  n ← 0
 5  while input symbol
 6  do symbol ← DECIPHER(symbol, K)
 7     symbol ← symbol mod m
 8     if seen symbol
 9       then skip
10     n ← n + INDEX-OF(symbol in input_symbols) × multiplicand
11     input_symbols ← input_symbols \ symbol
12     multiplicand ← multiplicand × base
13     base ← base − 1
14  return n
```

Fig. 5. OpenBSD permutation coding and decoding functions

7 Conclusion

In this paper, we have provided an overview of the opportunities for using TCP/IP header fields as a carrier for a steganographic covert channel. A detailed description of the ISN and IP ID generation schemes in Linux and OpenBSD was presented, and a number of previously proposed schemes for TCP/IP-based steganography were described.

We have shown that a passive warden can easily detect use of these schemes because the modified headers that they produce can easily be distinguished from those generated by a genuine TCP/IP stack.

Finally, we have outlined two schemes for encoding data with ISNs generated by OpenBSD and Linux. Both schemes generate ISNs that are indistinguishable from those generated by a genuine TCP stack, except by those with knowledge of a shared secret key.

Acknowledgments: Thanks are due to Joanna Rutkowska, George Danezis and Richard Clayton for their helpful contributions.

References

1. Simmons, G.J.: The prisoners' problem and the subliminal channel. In Chaum, D., ed.: Crypto '83. Advances in Cryptography, Plenum Press (1983) 51–67
2. Handel, T., Sandford, M.: Hiding data in the OSI network model. In Anderson, R., ed.: Information Hiding. Volume 1174 of Lecture Notes in Computer Science., Springer-Verlag (1996) 23–38
3. Szczypiorski, K.: HICCUPS: Hidden communication system for corrupted networks. In: International Multi-Conference on Advanced Computer Systems. (2003) 31–40 `http://krzysiek.tele.pw.edu.pl/pdf/acs2003-hiccups.pdf`.
4. Postel, J.: STD7: Transmission control protocol. IETF (1981)
5. Postel, J.: STD5: Internet protocol. IETF (1981)
6. Fisk, G., Fisk, M., Papadopoulos, C., Neil, J.: Eliminating steganography in Internet traffic with active wardens. In Petitcolas, F., ed.: Information Hiding. Volume 2578 of Lecture Notes in Computer Science., Springer-Verlag (2002) 18–35
7. Jacobson, V., Braden, R., Borman, D.: RFC1323: TCP extensions for high performance. IETF (1992)
8. Nichols, K., Blake, S., Baker, F., Black, D.: RFC2474: Definition of the differentiated services field (DS field) in the IPv4 and IPv6 headers. IETF (1998)
9. Fyodor: Nmap: free security scanner (1998) `http://www.insecure.org/nmap/`.
10. Ahsan, K., Kundur, D.: Practical data hiding in TCP/IP. In: ACM Workshop on Multimedia and Security. (2002) `http://ee.tamu.edu/~deepa/pdf/acm02.pdf`.
11. Mogul, J., Deering, S.: RFC1191: Path MTU discovery. IETF (1990)
12. Bellovin, S.M.: Security problems in the TCP/IP protocol suite. Computer Communication Review **19** (1989) 32–48
13. Rowland, C.H.: Covert channels in the TCP/IP protocol suite. First Monday **2** (1997) `http://www.firstmonday.org/issues/issue2_5/rowland/`.
14. Sohn, T., Seo, J., Moon, J.: A study on the covert channel detection of TCP/IP header using support vector machine. In Perner, P., Qing, S., Gollmann, D., Zhou, J., eds.: Information and Communications Security. Volume 2836 of Lecture Notes in Computer Science., Springer-Verlag (2003) 313–324
15. Rutkowska, J.: The implementation of passive covert channels in the Linux kernel. In: Chaos Communication Congress, Chaos Computer Club e.V. (2004) `http://www.ccc.de/congress/2004/fahrplan/event/176.en.html`.
16. Giffin, J., Greenstadt, R., Litwack, P., Tibbetts, R.: Covert messaging in TCP. In Dingledine, R., Syverson, P., eds.: Privacy Enhancing Technologies. Volume 2482 of Lecture Notes in Computer Science., Springer-Verlag (2002) 194–208
17. Bellovin, S.: RFC1948: Defending against sequence number attacks. IETF (1996)
18. de Raadt, T., Hallqvist, N., Grabowski, A., D. Keromytis, A., Provos, N.: Cryptography in OpenBSD: An overview. In: USENIX Annual Technical Conference (FREENIX Track). (1999) 93–102
19. Hintz, A.: Covert channels in TCP and IP headers. Presentation at DEFCON 10 (2002) `http://guh.nu/projects/cc/`.
20. Rukhin, A., Soto, J., Nechvatal, J., Smid, M., Barker, E., Leigh, S., Levenson, M., Vangel, M., Banks, D., Heckert, A., Dray, J., Vo, S.: A statistical test suite for the validation of random number generators and pseudo random number generators for cryptographic applications. Technical Report 800-22, NIST (2001)