# Going on the Defensive: Intrusion-Detection Systems

An *intrusion-detection system* (IDS) attempts to automate the recognition of hacker attack and penetration. Although the technology has not yet shown that it can produce a statistically significant return on investment, IDSs continue to grow in popularity. However, as the use of IDSs grow, attacks against them grow as well.

In this article, we review existing IDS technologies and will show you how to attack them. The purpose is to help you understand inherent weaknesses of IDSs so that you can implement the technology more appropriately in your layered security policy. In addition, we show how threats against IDSs are forcing their evolution into host-based, strict anomaly detectors.

## Types of IDSs

IDS systems come in multiple flavors, ranging in complex enterprise size systems to smaller single host sizes. Regardless of the level of complexity, IDSs all share the same general operational characteristics. This section breaks down the different types of IDSs available, and their methods of operation.

### Log File Monitors

The simplest form of an IDS is a *log file monitors,* which attempt to detect intrusions by parsing system event logs. For example, a basic log file monitor might grep (search) an Apache access.log file for characteristic /cgi-bin/ requests. This technology is limited, in that it detects only logged events, which attackers can easily alter. In addition, such a system will miss low-level system events because event logging is a relatively high-level operation. Example 1 illustrates the log of a real CGI scan against a web server.

***Example 1:*** *Sample Log Generated by cgi-scan Attack Script*

```
127.0.0.1 - - [14/Aug/2002:09:48:14 -0400] [ccc] "HEAD /cgi-bin/test-cgi HTTP/1.0" 404 0
127.0.0.1 - - [14/Aug/2002:09:48:14 -0400] [ccc] "HEAD /cgi-bin/nph-test-cgi HTTP/1.0" 404 0
127.0.0.1 - - [14/Aug/2002:09:48:14 -0400] [ccc] "HEAD /cgi-bin/phf HTTP/1.0" 404 0
127.0.0.1 - - [14/Aug/2002:09:48:14 -0400] [ccc] "HEAD /cgi-bin/phf.pp HTTP/1.0" 404 0
127.0.0.1 - - [14/Aug/2002:09:48:14 -0400] [ccc] "HEAD /cgi-bin/phf.cgi HTTP/1.0" 404 0
127.0.0.1 - - [14/Aug/2002:09:48:14 -0400] [ccc] "HEAD /cgi-bin/websendmail HTTP/1.0" 404 0
```

Log file monitors are a prime example of *host-based* IDSs because they primarily lend themselves to monitoring a computer system at the host level. In other words, a host-based IDS detects changes to file systems, log files, and applications on each computer.

Typically, a central IDS server then receives updates from each host as changes occur (see Figure 1).

In contrast, *network-based* IDSs typically scan the network at the packet level, much like a sniffer. This allows the IDS to monitor all the data passing over a network, and it can detect hacker activity by comparing the data to a set of rules. By doing this, the IDS does not need to interact with every computer on the network[md]only those that are sniffing data. In addition, a network-based IDS can have multiple logging systems spread out over the entire network to comprehensively monitor all legs.

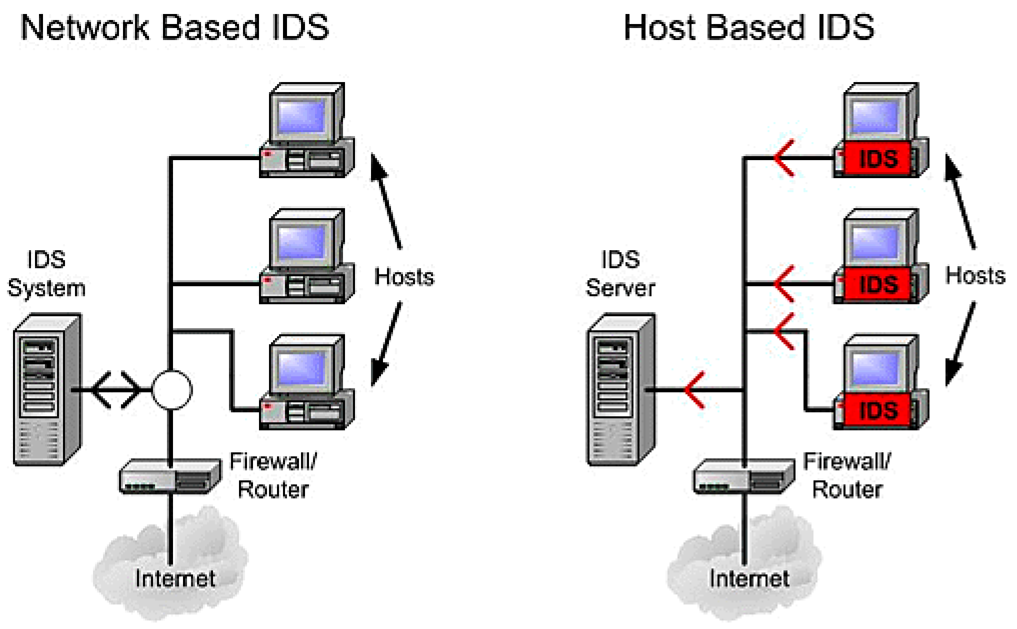

**Figure 1**

*Network-based IDS vs. host-based IDS.*

One well-known log file monitor is [Swatch](), short for Simple WATCHer. Whereas most log analysis software scans the logs periodically (at best), Swatch can actively scan log entries in real time, which also results in real time alerts.

To install Swatch, first download the latest version. Then run the following:

```
perl Makefile.PL
make
make test
make install
make realclean
```

After Swatch is installed, you might also have to download and install Perl modules that are required for it.

Swatch uses regular expressions to find lines of interest. When Swatch finds a line that matches a pattern, it prints it to the screen, e-mails an alert, or takes a user-defined action.

The following is an excerpt from a sample Swatch configuration script:

```
watchfor   /[dD]enied|/DEN.*ED/
echo bold
bell 3
mail
exec "/etc/call_pager 5551234 08"
```

In this example, Swatch looks for a line that contains the words denied or Denied, or anything that starts with DEN and ends with ED. When it finds a line that contains one of the three search strings, it echoes the line in bold on to the terminal and makes a bell sound (^G) three times. Then Swatch e-mails the user that is running Swatch (usually root) with the alert and executes the /etc/call_pager program with the given options.

**Integrity Monitors**

An *integrity monitor* watches key system structures for change. For example, a basic integrity monitor uses system files or registry keys as "bait" to track changes by an intruder. Although they have limited functionality, integrity monitors can add an additional layer of protection to other forms of intrusion detection.

The most popular integrity monitor is Tripwire, available for both Windows and UNIX. It can monitor a number of attributes, including the following:

- File additions, deletions, or modifications
- File flags (hidden, read-only, archive, and so on)
- Last access time
- Last write time
- Create time
- File size
- Hash checking

Tripwire can be customized to your network's individual characteristics. In fact, you can use Tripwire to monitor *any* change to your system. Thus, it can be a powerful tool in your IDS arsenal.

**Signature Scanners**

Like traditional hex-signature virus scanners, the majority of IDSs attempt to detect attacks based on a database of known attack signatures. When a hacker attempts a known exploit, the IDS attempts to match the exploit against its database. For example, Snort is a freeware signature-based IDS that runs on both UNIX and Windows.

Because it is open source, Snort has the potential to grow its signature database faster than any proprietary tool. Snort consists of a packet decoder, a detection engine, and a logging and alerting subsystem. Snort is a *stateful* IDS, which means that it can reassemble and track fragmented TCP attacks.

A classic example of a signature that IDSs detect involves CGI scripts. A hacker's exploit-scanning tools usually include a CGI scanner that probes the target Web server for known CGI bugs. For example, the well-known phf exploit allowed an attacker to return any file instead of the proper HTML. To detect a phf attack, a network IDS scanner would search packets for part of the following string:

```
GET /cgi-bin/phf?
```

When this string is detected, the IDS would react according to its preprogrammed instructions. Because this is a basic CGI scanner query, the danger level is medium, at most. However, if the IDS detected the following string:

```
cgi-bin/phf?Qalias=x%0a/bin/cat%20/etc/passwd
```

[el]and it resulted in a log entry as follows:

```
GET /cgi-bin/phf?Qalias x%0a/bin/cat%20/etc/passwd HTTP/1.0" 200 267
```

[el]then the problem just got more serious. Now, instead of an attack from a nosy script kiddie, you have a Web server that is vulnerable to the phf exploit. By looking at the log file, you can see that the phf query returned a successful 200, which means that the passwd file was just written to the hacker's screen.

**Anomaly Detectors**

Anomaly detection involves establishing a baseline of "normal" system or network activity and then sounding an alert when a deviation occurs. Because network traffic is constantly changing, such a design lends itself more to host-based IDS than network IDS. Anomaly detection provides high sensitivity but low specificity. Thus, it is most useful on absolutely critical (yet fairly static) machines in your network, such as a Web server.

In such a situation, when the anomaly detector reports any change from baseline activity, it requires human investigation. Thus, this high sensitivity comes with an accordingly high price in terms of human input.

## Attacking IDSs

To help you plan your security strategy, this section shows you how hackers exploit vulnerabilities in IDSs. This will help you to better understand inherent weaknesses in the technology so that you can plan your layered security strategy more effectively.

### Fragmentation

Fragmentation is the most common attack against IDSs. By splitting packets into smaller pieces, hackers can often fool the IDS. A *stateful* IDS can reassemble fragmented packets for analysis, but as throughput increases, this consumes more resources and becomes less accurate. For example, one of the most popular network-scanning tools available, known as NMAP, has a fragmentation engine built right into it. If a hacker is using the GUI version of this tool, she only has to check a box to fragment her probe. By doing this, an ICMP probe packet is down into small chunks that are reassembled at the target computer. Using this technique, firewalls and IDSs often see only the partial packet, which does not raise any warning flags.

### Spoofing

In addition to fragmenting data, it is possible to spoof the TCP sequence number that the IDS sees. For example, by sending a post-connection SYN packet with a forged sequence number, the IDS is desynchronized from the host. That is because the host drops the unexpected and inappropriate SYN, whereas the IDS might reset itself to the new sequence number. Thus, the IDS ignores the true data stream because it is waiting for a new sequence number that does not exist. Sending an RST packet with a forged address that corresponds to the forged SYN can also close this new connection to the IDS.

### HTTP Mutation

Whisker is a software tool designed to hack Web servers by sneaking carefully deformed HTTP requests past the IDS. For example, a typical cgi-bin request has the following standard HTTP format:

```
GET /cgi-bin/script.cgi HTTP/1.0
```

Obfuscated HTTP requests can often fool IDSs that parse Web traffic. For example, if an IDS scans for the classic phf exploit that follows, we can often fool it by adding extra data to our request:

```
/cgi-bin/phf
```

For example, we can issue this request:

```
GET /cgi-bin/subdirectory/../phf HTTP/1.0
```

In this case, we request a subdirectory and then use the /../ command to move back up to the parent directory and execute the target script. This technique of sneaking in the back door is referred to as directory traversal, and it is currently one of the most commonly used forms of exploit.

Whisker automates a variety of such anti-IDS attacks. Because of this, Whisker is known as an *anti-IDS (AIDS)*. Whisker has split into two projects[md]whisker (the scanner) and libwhisker (Perl module used by Whisker)[md]and has been updated regularly.

## The Future of IDSs

The future of IDSs is unknown. While we have shown the benefit of such systems, they are limited due to the fact that most a reactive. This section outlines the issues, and possible solutions to dealing with outdated IDSs.

### The Changing Landscape

As shown above, the technology to defeat IDSs continues to evolve. In addition, the milieu changes; IDSs must attempt to keep pace. Table 1 lists future trends that pose threats to IDSs and also lists potential solutions.

***Table 1:*** *Potential Solutions to Future Difficulties in IDS*

| **Problem** | **Solution** |
|---|---|
| Encrypted traffic (IPSec) | Embed IDS throughout host stack |
| Increasing speed and complexity of attacks | Use strict anomaly detection |
| Switched networks | Monitor each host individually |
| Increasing burden of | Use statistically based implementation data to interpret |

The following sections examine each of these growing problems, along with a potential solution.

### Embedded IDSs

*IPSec* is becoming a popular standard for securing data over a network. IPSec (short for *IP Security*) is a set of security standards designed by the Internet Engineering Task Force (IETF) to provide end-to-end protection of private data. Implementing this standard allows an enterprise to transport data across an untrustworthy network such as the Internet, while preventing hackers from corrupting, stealing, or spoofing private communication.

By securing packets at the network layer, IPSec provides application-transparent encryption services for IP network traffic, as well as other access protections for secure networking. For example, IPSec can provide for end-to-end security from client-to-server, server-to-server, and client-to-client configurations.

Unfortunately for IDSs, IPSec becomes a dual-edged sword. On one hand, IPSec allows users to securely log into their corporate network from home using a VPN. On the other hand, IPSec encrypts traffic, thus rendering promiscuous-mode IDSs useless. Therefore, if a hacker compromises a remote user's machine, he will have a secure tunnel through which to hack the corporate network.

To correct for IPSec, future IDSs might need to be embedded throughout each level of a host's TCP/IP stack. This will allow the IDS to watch data as it is unencapsulated and processed through each layer of the stack, and to analyze the decrypted payload at higher levels.

## Strict Anomaly Detection

Another growing problem is that as both the speed and complexity of attacks continue to increase, IDSs are becoming less able to keep pace. One answer to this dilemma might be the growing use of *strict anomaly detection*. This means that every abnormality, no matter how minor, is considered a true positive alarm.

Again, such a method would require that the IDS move onto individual hosts rather than the network as a whole. An individual host should have a more predictable traffic pattern, as opposed to the entire network. Each critical host would have an IDS that detects every anomaly. Then the administrator can make *rules* (exceptions) for acceptable variations in behavior. In this way, IDSs monitor behavior in much the same way that firewalls monitor traffic.

How would we design an IDS that performs host-based, strict anomaly detection? In this case, we are dealing with individual hosts that are somewhat isolated by firewalls and routers, so we can customize our IDS for each unique host. Because we are dealing with the host only, we know that any packets received are destined for that specific host. We can then set our sensitivity very high to look for any abnormality.

For example, at the packet level, our host-based anomaly detector would scan packets as they are processed up the stack. We could ask the IDS to monitor any of the following:

- Unexpected signatures

- TCP/IP violations
- Packets of unusual size
- Low TTL
- Invalid checksums

Similarly, at the application level, we can ask our anomaly detector to scan for unusual fluctuations in the following system characteristics:

- CPU utilization
- Disk activity
- User logins
- File activity
- Number of running services
- Number of running applications
- Number of open ports
- Log file size

When any abnormality is detected, an alert is sent to the centralized console. This method has a high sensitivity but, unfortunately, generates a great deal of data.

**Host- vs. Network-Based IDS**

The increasing use of switched networks hinders IDSs that monitor the network using promiscuous-mode passive protocol analysis. It is therefore becoming more difficult to monitor multiple hosts simultaneously. There have been attempts to rectify this using spanning (spy) ports to monitor multiple ports on a switch, but to date such solutions have been ineffective. In addition, the growing use of encrypted traffic foils passive analysis off the wire. Thus, IDSs might be moving more toward host-based monitoring.

## Summary

The use of IDSs continues to grow in popularity. However, as the use of IDSs grow, attacks against them grow as well. In this article, we reviewed existing IDS technologies and showed you how to attack them. By understanding the weaknesses of IDSs, you can more appropriately implement them in your layered security policy. In addition, we showed how threats against IDSs and the changing network milieu of IPSec are combining to force the evolution of IDSs into host-based, strict anomaly detectors.