

## Running Snort Part 2

Mark Burnett 2001-03-07

### Running Snort on IIS Web Servers Part 2: Advanced Techniques

by Mark Burnett

last updated March 7, 2001

---

## Introduction

Intrusion detection is the process of monitoring a network to identify, and thereby prevent, malicious network-based attacks. This process can be automated by a software application or hardware device known as an Intrusion Detection System or IDS. An IDS provides a wide range of monitoring techniques including packet sniffing, file integrity monitoring, and even artificial intelligence algorithms that detect anomalies in network traffic.

**Snort**, a public domain intrusion detection system, monitors traffic by analyzing every packet on a network, looking for malevolent content. It does this by putting the network adaptor in promiscuous mode so that it can see all network traffic on the wire, a process referred to as packet sniffing. Snort is a rule-based IDS, which means that it applies a set of rules to each packet based on known attack signatures. When it detects an attack signature, it performs the action designated in the rule.

As a rule-based IDS, Snort is highly dependent upon its ruleset. A system administrator can establish that a network is being attacked by identifying network packets that contain signatures that are known to be malicious. For example, a request in the web logs for `"/global.asa+.htr"` (see [MS00-006](#)) is a good indication of an attack. Why? First of all, it is unlikely that anyone would have a legitimate reason for requesting `global.asa` from a web server. Second, appending the invalid URL syntax `+.htr` is likely to be deliberate rather than a typo. A sysadmin can easily create an IDS rule to watch for this request and issue a warning when it is spotted. By building enough rules like this, the administrator can catch most intruders.

One significant flaw with rule-based IDS is that it a sysadmin must know about an exploit in order to be able to create a rule to protect against it. But what about newly discovered or non-public exploits? When Microsoft releases a security bulletin, the exploit it refers to is by no means new. Months may have passed since the vulnerability was first discovered. Time must be spent assessing the problem, correcting the code, then testing and deploying a hotfix before a bulletin is released. In that time, many people may have been made aware the problem. Sometimes, by the time exploits are made public, they are already well known in hacker circles. Even if it is a new exploit that no one has heard of, if a hacker reads the security bulletin before you patch your server, you may have a security breach on your hands.

Any of these scenarios offers a window of opportunity for a "new" vulnerability to be exploited. Moreover, a number of techniques have been developed over the years to evade intrusion detection systems. Eventually most IDS software may be updated to render those techniques ineffective, but it is always a game of being one step behind the attackers. In the meantime, however, the IDS is ineffective against potentially malicious intrusion.

It is evident from these examples that systems administrators cannot rely on rulesets alone to provide optimal IDS protection. To take intrusion detection one step further, security personnel can implement a number of strategies to gain the edge on attackers. The remainder of this article will examine some of these strategies, how they can be implemented and why they work. Although these techniques are discussed as they are implemented with Snort, the concepts can be applied to just about any rules-based IDS.

## Strategy One: Watch the Exits

The first strategy is to watch what is leaving the network. Although attackers can modify incoming packets to evade detection, they generally do not have any control over how the computers on your network respond to incoming requests. Watching a response to a remote request is frequently more effective than watching the request itself. Generally speaking, administrators are not as concerned about what people are sending them as about what their network sends back in response. Clearly, the sysadmin knows what data or information he does not want anyone else to see. If the administrator can create rules to detect restricted data crossing the router, no matter how new or well-known an exploit used to access it may be, the administrator will be notified.

For example, most system administrators do not want directory listings of their site to be viewed without explicit permissions. To catch directory listings leaving the web server, the sysadmin can create an empty subdirectory in the web root with a unique name such as

```
IDS_DIRECTORY_TAG.
```

Next, a rule must be created that watches for that string:

```
alert tcp $HTTP_SERVERS any -> $EXTERNAL_NET any (msg:"Outgoing directory listing";  
content: "IDS_DIRECTORY_TAG"; nocase;)
```

This rule will watch every TCP packet coming from the web servers and will create an alert when a directory listing leaves the network. Taking this strategy one step further, an admin could create similarly tagged subdirectories in sensitive areas such as WINNT, System2, and Program Files.

Another thing that most systems administrators want to protect is source code and scripts on their web site. Historically, there have been a number of IIS exploits that allow a remote attacker to view scripts or other sensitive files on the server. Depending on the web application, having source code exposed to the world could be a very serious problem.

To prevent such exploits, a comment tag such as this can simply be embedded in the server-side code:

```
<!--IDS_SOURCE_CODE_TAG-->
```

Then, as with the previous example, a Snort rule can be created to watch for that string:

```
alert tcp $HTTP_SERVERS any -> $EXTERNAL_NET any (msg:"Outgoing source code";
```

```
content: "IDS_SOURCE_CODE_TAG"; nocase;)
```

With that rule in place, the administrator will know when sensitive source code has been revealed to outsiders.

For other sensitive files such as .INI files, a comment can be embedded by prefacing the line with a special character. For example, the following line could be added to the win.ini and system.ini files:

```
; IDS_INIFILE_TAG
```

Again, a rule to watch for that tag can be created so that the administrator will be informed of outside attacks.

In addition to tagging, there are a number of other common strings that can be used to spot outgoing information. Here are some that administrators may want to consider adding to the ruleset:

```
Directory listing of
Index of
<%
C:\inetpub\wwwroot
C:\>
Command completed successfully
1 file(s) copied
Bad command or filename
```

Another effective way to scrutinize outgoing data is to look for error codes coming from an application. Often, VBScript or ADO errors reveal sensitive information about the server. Administrators should look at what they have to protect, then watch for common strings that they do not want leaving the network.

## Strategy Two: Watching for Exploit Commands

Generally when a server is attacked, the attacker tries to run certain commands or access certain files. One command an attacker may want to run is cmd.exe. Another file that may be of interest to an attacker on IIS is global.asa, which often contains passwords or other sensitive information. By watching for commands that an attacker would want to run or files that an attacker would want to see, one can increase the chances of spotting an intruder, even if they are using a new or non-public exploit.

Building rules of this type requires a thorough understanding of why someone would attack your server and what they would want to accomplish. Every attack has a purpose; it is up to the security administrator to know how that purpose will be accomplished and build rules accordingly.

Regardless of the method they use, an attacker will eventually need to do something on the target server that could potentially give them away. Historically, many exploits on IIS have involved traversing directories, viewing server-side scripts, or running a remote command. While each exploit is executed differently, they all are going after the same target. Instead of (or in addition to) alerting on the exploit itself, system administrators can build rules that match the commands or file requests made through the exploit. When an attacker runs a command that is used for malicious purposes, the IDS ruleset detects the prohibited command and issues an alert.

Let us take a look at an example of this command-based rule strategy. We have already established that `global.asa` contains sensitive information and that there is no legitimate reason for a web visitor to request that file. With that in mind, we should build a rule that alerts whenever a request is made that includes the string "global.asa." Such a rule would be written as follows:

```
alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS 80 (msg:"WEB-IIS: global.asa access";
flags: A+; content:"global.asa"; nocase;)
```

Another obviously malicious web request would be one that contains the string "cmd.exe." Since your web site should never contain a command interpreter, this is another request we can assume is an attempt to run a remote command. A rule to detect such an attempt can be written as follows (note there is a similar rule in the standard Snort rules):

```
alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS 80 (msg:"WEB-MISC: Attempt to execute
cmd.exe";
flags: A+; content:"cmd.exe"; nocase;)
```

In addition to `cmd.exe`, there are a number of other built-in and third party applications that could prove to be dangerous if used improperly. Some of these include:

- `ftp.exe`;
- `tftp.exe`;
- `net.exe`;
- `telnet.exe`;
- `rcmd.exe`;
- `wsh.exe`;
- `csn.exe`;
- `perl.exe`;
- `nc.exe`;
- `lsadump2.exe`; and
- `pwdump.exe`.

In the hands of an attacker, even `snort.exe` could be a dangerous tool!

An attempt to send SQL commands through a web form may also constitute an exploit. If not properly filtered, form data can contain malicious SQL commands that may execute code an attacker wishes to run. Administrators should never pass an SQL statement on a URL themselves, so here are some generic SQL strings that they may be able to write rules for:

```
SELECT * FROM
INNER JOIN
LEFT JOIN
```

If using Microsoft SQL Server, the administrator may also want to watch for dangerous stored procedures. Here are some examples of built-in stored procedures that may threaten security:

```
xp_cmdshell  
xp_availablemedia  
xp_fileexist  
xp_enumdsn  
xp_regread
```

## Strategy Three: Watching for Traffic from Scanning Sites

Before being able to attack services on a server, an attacker must first know what ports are open to exploit. Once they have found an open service, they often run another scanner (such as a CGI scanner) to find which vulnerabilities exist.

To make this all easier and more anonymous, many web sites have made scanning tools available for public use. This allows an attacker to scan any IP address from a web page. While they are often used by attackers, many of these scanning tools have both legitimate and illegitimate uses: they are often used by system administrators to diagnose network problems or to gather statistical information about the Internet. Regardless of the reason, it is nice to know when your network is being scanned, particularly as this information may support other evidence of an attack. As such, the third strategy is to monitor traffic from online scanning sites.

By watching for traffic from scanning sites, an administrator may be forewarned of an impending attack. The following is only a brief sample of some of the sites that have online scanning tools:

<http://www.cotse.com>

<http://www.all-nettools.com>

<http://www.sampade.org>

<http://www.subdimension.com>

<http://mixter.void.ru/evil.html>

<http://www.netcraft.com>

It is important not to react too quickly to traffic from these sites as the traffic may not be malicious. For instance, some of those sites have CGI scripts that allow a user to browse a web site anonymously. Traffic from one of these sites may simply indicate an anonymous visitor rather than a potential attacker.

Administrators can build IDS rules to detect traffic from the more common scanners out there. Most scanners leave very clear fingerprints and a quick analysis of each scanner could give enough information to build some very effective rules. Imagine how nice it would be to not only know you are being scanned, but also knowing which scanner they are using.

## Conclusion

When building an IDS ruleset, it is easy to get stuck in the mode of building rules based on known exploits. However, it is also important to protect against exploits that may not be well-known. But if administrators can step back and build rules that will cover as many current and future exploits as possible, the security offered by intrusion detection systems will be vastly increased. By watching for outgoing directory listings or source code,

requests for cmd.exe, and requests for global.asa administrators can stay one step ahead of the attacker. With a little creativity and a few more rules, very little will go by without the sysadmin knowing. And in this game, knowing is winning.

#### Relevant Links

[Running Snort on IIS Web Servers: Part I](#)

*Mark Burnett, SecurityFocus.com*

[Snort](#)

[Writing Snort Rules](#)

*Snort*

[Xato's snort.panel tool](#)

*Xato*

[Privacy Statement](#)

Copyright 2006, SecurityFocus