

Network Intrusion Detection Signatures, Part Four

Karen Kent Frederick 2002-03-05

Network Intrusion Detection Signatures, Part Four

by *Karen Kent Frederick*

last updated March 5, 2002

This is the fourth in a series of articles on understanding and developing signatures for network intrusion detection systems. In [part one](#) we discussed the basics of network IDS signatures and then took a closer look at signatures that focus on IP, TCP, UDP and ICMP header values. In the [second installment](#) we looked at some signature examples. In [the previous article](#), we began to examine the topic of protocol analysis, which means that the intrusion detection system actually understands how various protocols, such as FTP, are supposed to work. In this article, we will continue to look at protocol analysis and how it can overcome attempts by attackers to obfuscate their exploits so that they cannot be detected by simple intrusion detection signature methods.

Protocol Analysis Review

Let's quickly review what protocol analysis means. In protocol analysis, the network intrusion detection system (IDS) sensors examine TCP and UDP payloads, which contain other protocols such as DNS, FTP, HTTP and SMTP. The sensors understand how these protocols are supposed to work, based on RFCs and on real-world implementations of the protocols, and they can fully decode them. This allows a much larger range of signatures to be created than would be possible through simpler signature techniques. Some IDS sensors can only utilize "packet grepping" signatures, which do character-by-character or byte-by-byte matches within a TCP or UDP payload. Although packet grepping signatures are useful for identifying certain types of attacks, they lack the flexibility to identify many types of attacks. In particular, packet grepping signatures are typically unable to handle the obfuscation techniques that attackers often use to attempt to evade detection by IDS sensors.

In the previous article, we reviewed a simple example of how an attacker can attempt to obfuscate an attack through the use of extra white space. If an intrusion detection system uses a packet grepping signature that looks for the FTP "SITE EXEC" request, it will be looking for that exact match in the TCP payloads. However, most FTP servers ignore extra white space, so an attacker could place an extra space between "SITE" and "EXEC", and the packet grepping

signature would fail to make a match because of the space. Protocol analysis signatures would break the whole "SITE EXEC" command into its components, the command name "SITE" and the argument "EXEC", making the extra white space irrelevant. This is probably the simplest way by which attackers obfuscate attacks; now let's dig in and look at some more interesting examples of attempting to evade detection.

Path Obfuscation

Another simple IDS evasion method that many attackers use is path obfuscation. The idea of this technique is to alter the path so it has a different appearance but the same meaning. This technique is most frequently used within URLs to hide HTTP-based attacks. Here are three of the ways that attackers commonly use to obfuscate paths. In these examples, we'll assume that we want to alert whenever we see `scripts/iisadmin` as part of the path in the URL.

Backslashes are substituted for regular slashes. Most Web servers don't care whether backslashes or regular slashes are used to separate directories. So the Web server will treat the URL excerpts `scripts/iisadmin` and `scripts\iisadmin` the same way. Unfortunately, IDS signatures that just do simple text matching will see these two strings as being different and will not generate an alert if `scripts\iisadmin` is used.

Single-dot sequences, like `./`, may be added to the path. When a single period is listed as a directory, it refers to the current directory. When it's used within a path, it is completely ignored by the Web server. So `scripts/./iisadmin` has exactly the same meaning as `scripts/iisadmin`. Again, a simple packet grepping signature won't identify `scripts/./iisadmin` as an attack.

A slightly more complex technique utilizes double-dot sequences, such as `../`, to further obfuscate paths. A double-dot directory means that the parent directory should be used. How does this help an attacker? Well, the attacker can list an irrelevant directory immediately before a `../` sequence; the `../` will wipe out that directory. So if the attacker uses `scripts/sample/../iisadmin`, the `../` will wipe out `sample`, leaving `scripts/iisadmin` again.

Of course, an attacker may mix and match these methods. She can use multiple instances of any of these methods within a URL, as well as using multiple methods in a single URL. For example, the attack could be hidden as `scripts/../.././iisadmin`, or as `scripts\./iisadmin`. Obviously, there are countless obfuscation variations for this URL, so packet grepping

signatures can't possibly catch all the attempts. So how can we create a network IDS solution that overcomes these obfuscation techniques?

Protocol analysis is able to handle these techniques because it performs much of the same processing on URLs that a Web server or operating system does. When HTTP traffic is being monitored, the IDS sensor extracts the path from the URL and analyzes it. It looks for backslashes and single-dot and double-dot directories, and it handles them appropriately. After it has "standardized" the URL, it can then search it for likely suspicious directory content, such as `scripts/iisadmin`. Now let's look at a stronger example of how protocol analysis performs the same processing that a server would do: handling hex encoding within a URL.

Hex Encoding

Hex encodings can be used to represent characters in URLs. You may have seen URLs that contain `%20`. This is a hex encoding, which is the equivalent of a single space. Because a URL cannot contain an actual space, it will use `%20` instead if a directory name or filename contains a space. All alphanumeric characters have hex encoding equivalents that can also be used within URLs. Unfortunately, although Web servers understand hex encoding, many intrusion detection signatures do not. This provides a golden opportunity for attackers to utilize hex encoding to evade IDS detection. Fortunately, we can de-obfuscate such attacks through the use of protocol analysis-based signatures.

Let's use our `scripts/iisadmin` example again. An attacker wishes to try to use this directory in a URL but does not want an IDS sensor to detect its usage. So he uses a hex encoding in place of one of the characters. `%73` is the hex encoded equivalent of `s`, so he alters his URL to use `script%73/iisadmin`. Hex encoding gives attackers countless ways to obfuscate each URL that they use; the best way to identify the underlying attack is to rely on protocol analysis. Signatures that perform HTTP protocol analysis perform the same hex decoding that a Web server would do, so they first convert the `%73` to `s` before evaluating the URL for potential attacks. Actually, as you're about to see, Microsoft IIS Web servers perform two rounds of hex decodings, which gives attackers yet another way to obfuscate their attacks.

Double Hex Encoding

In September 2001, the Nimda worm spread throughout the Internet. It took advantage of a vulnerability that was named the Escaped Character Decoding Vulnerability, which involves

double hex encoding. An attacker could craft a URL so that it contained special hex-encoded sequences. When a vulnerable Microsoft IIS server received a URL, it performed one round of hex decoding on the path in the URL. It performed a security check on the URL at that point, but afterwards performed a second round of hex decoding on it. Let's look at an example from the Nimda worm to illustrate how this worked.

The Nimda worm used 16 different URLs to probe Microsoft IIS servers for known vulnerabilities, including the double hex encoding one. One example of the relevant part of the URL is `scripts/..%255c../winnt.%25` is the hex encoding equivalent of the `%` character. So when the URL is hex decoded, that text becomes `scripts/..%5c../winnt.%5c` is the hex encoding equivalent of a backslash. So when a second round of hex decoding is done, `%5c` is hex decoded and the URL will contain `scripts/..\../winnt`. As we learned earlier, this is a path obfuscation of `scripts/../../winnt`, which is a type of attack known as a root traversal. Thorough HTTP protocol analysis will perform two rounds of hex decoding on the entire URL path in order to identify such obfuscation attempts.

Unicode (UTF-8)

The final obfuscation method that we will look at in this article is Unicode (or, more correctly, UTF-8). This is another alternate way of representing characters and is most infamously known for being used for various HTTP-based attacks. When you first look at Unicode, you might think that you are really looking at hex encoding. For example, `%c0%af` is a valid Unicode sequence, not two consecutive hex encodings. Hex encodings correspond to ASCII characters up to `%7f`; these values are higher, so we can recognize them as Unicode and not hex encodings.

An example of a URL path excerpt containing Unicode is `scripts/..%c0%af../winnt.%c0%af` is the Unicode equivalent of a slash, so this path actually means the same as `scripts/../../winnt`, the same root traversal attempt we just looked at in the Double Hex Encoding section. By including Unicode handling in the HTTP protocol analysis signature capabilities, we can convert Unicode sequences to their regular ASCII equivalents and identify the underlying attacks that were obfuscated.

Signature Example

Now that we understand why handling obfuscation is so important, let's look at an example of how we would make a signature for a particular attack. This signature will be composed not

only of the de-obfuscation methods we have already reviewed in this article, but also on the other signature principles we studied in earlier articles in this series. Let's go through the steps that the IDS sensor and signatures would perform to identify URLs that include `/hidden/admin/` in their paths:

1. Decode the IP packet header to determine what protocol the IP payload contains. In this example, we are looking for IP protocol number 6, which corresponds to TCP.
2. Decode the TCP header to find the TCP destination port number. Assuming that our web server listens on port 80, we would look for that value as the destination port. This tends to indicate that the user is sending an HTTP request to the server.
3. Rely on HTTP protocol analysis to parse the HTTP request into all of its many components, including the URL's path.
4. Process the URL path by handling path obfuscation, hex encoding, double hex encoding, and Unicode.
5. Review the de-obfuscated path for a match with `"/hidden/admin/"`, and generate an alert if a match is found.

This is a great example of how intrusion detection, and protocol analysis in particular, is really done. What sounds like a simple signature – looking for `/hidden/admin/` – is a much more complex process than you might initially think. But that level of complexity is necessary because attackers can literally use an infinite number of attack variations. The only way of identifying all such attacks with network intrusion detection is by performing protocol analysis.

Conclusion

In this article, we have continued to examine the topic of protocol analysis, where network IDS sensors understand how protocols such as FTP and HTTP are supposed to work, and decode and analyze them accordingly. Protocol analysis-based signatures provide a superior intrusion detection solution, compared to other signature methods, because they can detect a much wider range of attacks, including known and unknown attacks. As we have examined in detail, protocol analysis-based signatures are also far more resistant to attackers' obfuscation attempts than other signature techniques.

The next article in this series will continue to examine protocol analysis, focusing on the concept of stateful protocol analysis. Stateful protocol analysis is when protocol analysis is performed on an entire protocol session, and key attributes are tracked throughout the session.

For example, the IDS sensor can pair a command with its corresponding response, and it can verify that commands are issued in the correct sequence. Stateful protocol analysis is an incredibly powerful signature technique that we will examine in depth in the next article.

[Karen Kent Frederick](#) is a senior security engineer for the Rapid Response Team at NFR Security. She is a graduate of the University of Wisconsin-Parkside and is currently completing her master's in computer science, focusing in network security, through the University of Idaho's Engineering Outreach program. Karen has over 10 years of experience in technical support, system administration and information security. She holds several certifications, including SANS GIAC Certified Intrusion Analyst, GIAC Certified Unix Security Administrator, and GIAC Certified Incident Handler. She is one of the authors of "Intrusion Signatures and Analysis", and she is a contributing author to the "Handbook of Computer Crime Investigation".

Relevant Links

[Unicode Home Page](#)

Chris Weber, SecurityFocus

[Network Intrusion Detection Signatures, Part One](#)

Karen Kent Frederick, SecurityFocus

[Network Intrusion Detection Signatures, Part Two](#)

Karen Kent Frederick, SecurityFocus

[Network Intrusion Detection Signatures, Part Three](#)

Karen Kent Frederick, SecurityFocus

[Privacy Statement](#)

Copyright 2006, SecurityFocus