

IDS Evasion with Unicode

Eric Hacker 2001-01-03

IDS Evasion with Unicode

by *Eric Hacker*

last updated Jan. 3, 2001

Recently, there has been much discussion of the Unicode problem with regard to intrusion detection. Some pundits, such as [Stuart McClure and Joel Scambray](#), have gone so far as to claim that Unicode will contribute to the demise of Intrusion Detection Systems (IDS). This article will explain what Unicode is, how it complicates IDS and provides opportunities for IDS evasion, and what can be done about it. This discussion will particularly focus on the role of UTF-8, a means by which Unicode code points are encoded, in circumventing IDSs. The Unicode threat to IDSs is real and complicated - I will attempt to separate hype from the truth and provide readers with the knowledge to understand the risk this threat presents to them.

IDS Evasion

A thief obtains his prize by bypassing alarms and security systems that are in place. IDS evasion is no different. An attacker knows that the IDS will alarm on certain attack signatures and, therefore, will try to evade the IDS by disguising the attack. Evasion techniques are available at many different OSI layers, but network-based IDSs (NIDS) are getting better at detecting lower layer evasion techniques. For instance, evasion attempts using small fragments will set off alarms on all good NIDS products.

However, evasion at the application layer is a complex NIDS problem. The NIDS must completely mimic the application protocol interpretation. An attacker can use the differences between the application and the IDS as an unlocked, unwatched window to the prize. Signature-based NIDSs can have trouble dealing with the complexities of application interactions. The potential for evasion at the application layer is increasing because new protocols are becoming more complex with support for features like Unicode.

What is Unicode?

Unicode provides a unique identifier for every character in every language to facilitate uniform computer representation of the world's languages. Unicode is managed by the [Unicode Consortium](#) and has been adopted by most information technology industry leaders. Unicode is required by many modern standards including Java, LDAP, and XML. Unicode support is available in many operating systems and applications. Unicode characters are called code points and can be represented by U

+xxxx where x is a hexadecimal digit.

What is UTF-8?

As [defined](#) by the Unicode Consortium in its "Corrigendum to Unicode 3.0.1", section D36, "UTF-8 is the Unicode Transformation Format that serializes a Unicode code point as a sequence of one to four bytes." UTF-8 provides a way to encode Unicode code points and still be compatible with ASCII, the common representation of text on the Internet.

UTF-8 does this by representing the standard seven bit ASCII (U+0000 through U+007F) unchanged as a single byte. Unicode characters U+0080 through U+07FF are represented with two byte sequences. U+0800 through U+FFFF become three byte sequences and four byte sequences are used to encode Unicode code points beyond that. The UTF-8 to Unicode transformation is algorithmic so that it can take place in code without resorting to lookup tables.

UTF-8 URLs are supported by Microsoft Internet Explorer and Office 2000. IIS can interpret UTF-8 up through three-byte sequences by default. Apache can be configured to support UTF-8, but is not by default.

Security Problems With UTF-8 and Unicode

Bruce Schneier was one of the first to raise the security issues with Unicode in the July 15, 2000 issue of [Crypto-Gram](#) newsletter. He pointed out that with the Unicode character set, it is possible that there could be multiple representations of a single character. There are also code points that may be used to modify the previous code point. In a security context, one must absolutely determine the meaning of a character. This is made more difficult by multiple representations and applications that may not enforce appropriate security controls due to Unicode's complexity.

The Unicode Consortium recognized multiple representations to be a problem and has revised the Unicode Standard to make multiple representations of the same code point with UTF-8 illegal. The [UTF-8 Corrigendum](#) lists the newly restricted UTF-8 range. This change came about only recently, however, and current applications may not have been revised to follow this rule. IIS is one of these applications.

How Many Ways to Say That?

The problem with the original UTF-8 transformation formula is that it re-mapped the complete set of previous code points every time the representation was extended a byte. Thus when UTF-8 started

two-byte representations, it repeated the Unicode code points from the one byte representations. When UTF-8 started three byte representations, it repeated the code points for the one and two-byte representations.

An example is the backslash character "\", U+005C. Under the original UTF-8, it could be represented by a hex 5C, C19C and E0819C. All these represent the exact same Unicode code point: when one applies the transformation algorithm, one gets the same value. Many older applications that support UTF-8 will accept the three values and perform the transformation to the backslash.

Applications Add Further Complexity

Another way that Unicode can cause problems is that the application or operation system can assign the same interpretation to different code points. Thus, even though the Unicode specification dictates that the code points should be treated differently, the application actually treats them the same.

I tested IIS on Windows 2000 Advanced Server (English) and found that it was very good at exhibiting this behavior. For example, here is a list of the various code points that resolved to the capital letter "A": U+0041, U+0100, U+0102, U+0104, U+01CD, U+01DE, U+8721. Remember that many of these code points have multiple representations themselves. Since IIS is not case-sensitive, this leads to 30 different representations for the letter "A". There are 34 for "E", 36 for "I", 39 for "O", and 58 for "U". The string "AEIOU" can be expressed 83,060,640 different ways.

What Kind of Problems Does This Cause?

A clear example of the kind of problems that arise out of multiple representations is the [Microsoft IIS 4.0 / 5.0 Extended UNICODE Directory Traversal Vulnerability](#). Rain Forest Puppy was the first to publicly [discuss this bug](#). He described the problem as simple: IIS checked for directory traversal before it decoded UTF-8 and thus it missed UTF-8 encoded directory traversals. The operating system decoded the UTF-8 and proceeded accordingly.

The attack to exploit this vulnerability was simple. If an attacker submits a URL like `http://victim/../../../../winnt/system32/cmd.exe` in an attempt escape out of web root, IIS correctly strips off the extra `../../../../` and generates an error. However, if the attacker encoded the `../../../../` portion of the attack with UTF-8 so that `..%C1%9C..` was sent instead, the directory traversal would not get stripped. The attacker could then run other programs and access other files if default permissions were not changed.

What's this got to do with IDS?

Naturally, once this exploit was made public, IDS vendors rushed out new signature databases for it. (Well, rush is a relative term for some vendors, but most made an attempt at the basics.) Only one vendor, NetworkICE, was prepared to attempt to tackle the problem correctly. First we will look at what does not work.

Here is what Snort's latest rule set contains:

```
alert tcp !$HOME_NET any -> $HOME_NET 80 (msg: "IDS434 - WEB IIS - Unicode traversal backslash"; flags: AP; content: "..|25|c1|25|9c"; nocase;)
alert tcp !$HOME_NET any -> $HOME_NET 80 (msg: "IDS433 - WEB-IIS - Unicode traversal optyx"; flags: AP; content: "..|25|c0|25|af"; nocase;)
alert tcp !$HOME_NET any -> $HOME_NET 80 (msg: "IDS432 - WEB IIS - Unicode traversal"; flags: AP; content: "..|25|c1|25|1c"; nocase;)
```

All these rules are simply two periods followed by one of the publicly discussed UTF-8 representations for a slash or backslash.

ISS RealSecure has implemented something a little stronger. Their match string found in their description (<http://xforce.iss.net/alerts/advise68.php>) of the attack, `\\.\\(\xc0|\xc1|\xe0|\xf0|\xf8|\xfc)`, would actually catch many variations of UTF-8 slashes that might follow two periods.

But both the Snort and RealSecure signatures only catch the publicly discussed methods. It turns out there are many other ways around this. For instance, an attacker could use `%C0%AE` to substitute for the periods instead of the slashes. IIS may still be vulnerable, but the IDS misses the attack.

Another little known fact, at least to some IDS vendors, is that IIS will accept un-escaped UTF-8 bytes if they are sent directly. Web browsers do not support sending these bytes, but it is a trivial matter to open a port in some other program and send them. Thus, if one sends the bytes `0xC0AE` to IIS in a URI, IIS will happily interpret it as a UTF-8 period, but many IDS will not see any problem and will have been successfully evaded.

Furthermore, there is no reason why UTF-8 can't be used to encode other attacks. It will be only a matter of time before tools like [whisker](#) and [fragrouter](#) will have UTF-8 capabilities. The way it stands right now, any NIDS that is supposed to be watching over IIS servers can be evaded successfully for all URL signatures.

The only possible way for NIDS to catch these attacks is through UTF-8 processing, just as IIS does. This concept is not new to NIDS vendors. Many already support telnet session reconstruction. Many also support http session conversion to catch basic single byte encoding. Some will call this protocol

analysis, others will call it pattern matching with pre-canonicalization of the data. Whatever one wants to call it, it is a very tough problem.

NetworkICE Almost Gets It Right.

The only vendor to even attempt UTF-8 analysis was NetworkICE. According to a post on the Focus-IDS mailing list, NetworkICE had been working on this problem from the time the issue was raised by Schneier. As a result, they were prepared for a more comprehensive solution. I have had the opportunity to test NetworkICE on its UTF-8 capabilities and it does just what one would expect it to.

NetworkICE successfully canonicalizes and detects attacks using the UTF-8 standard encoding. When presented with one, two, or three byte UTF-8 representations of a single Unicode code point, BlackICE Agent will successfully identify the attack. Whether the attack is the UTF-8 backticking attack or some other web based attack, BlackICE seems to detect it in my testing (provided one has used the Unicode standard UTF-8 encoding). The problem is those extra "AEIOU"s that IIS also interprets.

Remember how UTF-8 is supposed to be an algorithmic translation to Unicode? It is likely that that is what NetworkICE implemented in their UTF-8 parsing engine. Remember though, IIS does not seem to operate that way. Thus attacks UTF-8 encoded with non-standard characters can evade even BlackICE.

Other Applications Could Make It Worse

It is a very good possibility that UTF-8 with Apache on Solaris would be interpreted differently. That means one rule does not fit all. NIDS would have to not only have UTF-8 tables for each application, but also keep track of which IP addresses have which applications so that the proper translation can occur. This is a problem some NIDS vendors are working on, but it certainly will not make NIDS any easier.

So What Can Be Done?

The biggest part of the problem at the application layer is that it is very difficult to synchronize the parsing done by the NIDS with the parsing done by the application. Host-based IDS that has access to the application logs is much more effective at detecting application layer attacks. The logs should reflect how the application parsed the request and, thus, will give an accurate portrayal of what occurred. Host-based IDS is simply the better technology to apply to application layer intrusion detection.

There are some other things that could be done to help detect and deter UTF-8 evasion attempts. NIDS that do some protocol analysis, such that they can pull the URI out of the port 80 traffic, should be able to identify any UTF-8 encoding within the URI. If a web site does not use UTF-8, then an alert on its use could be valuable. This, of course, will not help the foreign language servers that actually use UTF-8, but may provide some detection capabilities for others.

It would also be nice if IIS allowed one to turn off multi-byte UTF-8 processing. Unicode is a security issue that is likely to rise again and for many sites it is completely unnecessary. Certainly, the current IIS/Windows implementation is broken, as it does not conform to the recently revised standard; but is that a "bug" that Microsoft will fix anytime soon? Unicode support in applications is a laudable goal, but until the complete security ramifications of the standard are understood, applications should allow the option of turning it off.

Conclusion

Unicode support in Internet applications can cause security problems, especially for those trying to watch and protect their servers. Today it is possible to use UTF-8 encoding to attack an IIS server and evade detection from all vendor's NIDS. Application layer protection from NIDS will be a difficult problem to solve and one should not expect any miracles, despite the future claims of the vendors' marketing departments.

Finally, I have compiled a list of codes that I believe create an IDS evasion a problem for IIS. Some are well known and are merely included for completeness. My list, however, is not to be taken as complete. I merely brute-forced through the entire UTF-8 character space and parsed the logs for single byte interpretations. I did not try any multi-code point Unicode characters. I very likely made a mistake or two.

The list is available directly from the [author](#).

Eric Hacker, CISSP, GCIA is a Network Security Consultant with [Lucent Worldwide Services](#).

[Privacy Statement](#)

Copyright 2006, SecurityFocus