Intrusion Detection System (IDS) Evasion//

May 10, 2006

An iDefense Security Report

The VeriSign® iDefense® Intelligence Team

INSIDE THIS REPORT

iDEFENSE - A VeriSign Company

21345 Ridgetop Circle, Sterling, VA 20166
Toll Free: 877.516.2974 Main: 703.390.1230 Fax: 703.390.9456
www.idefense.com | di@idefense.com

1 Simple IDS Evasion Techniques

Intrusion detection system (IDS) technology became popular with most system administrators in the mid-to-late 1990s because they allowed administrators to identify if and when a break-in had been attempted. The technology seemed simple enough, a network appliance that monitored network traffic for signs of an attack or abusive behavior and alerted system administrators.

Soon after IDS technology emerged in the corporate environment, hackers introduced several methods for evading detection. At first, evasion techniques were crude, with denial of service (DoS) attacks, false positives and simple pattern-matching techniques among the simplest. Over time, however, more complex methods became available to hackers such as session splicing, fragmentation and polymorphic shellcode.

The introduction of IDS technology and the subsequent response by malicious actors marked the beginning of a race between hackers and IDS developers; developers fixing problems with IDS technology to better detect attack signatures, while hackers busily creating with new ways to elude the improved IDS technology.

This report explores why IDS technology, while useful, should not be considered an all-in-one solution for network security. IDS technology alone should not be relied upon to give an accurate assessment of a networks security status, but should be used in conjunction with other technologies that complement the strengths inherent in this technology.

1.1 Pattern-Matching Weaknesses

Many of the evasion techniques crafted by hackers exploit the pattern-based detection approach employed by most IDS. Pattern-based detection uses pattern matching to identify potential attacks based on known vulnerabilities or commonly used strings within exploit code. This approach is problematic, however, because not all input need be the same to trigger certain vulnerabilities, and even slight changes in input can bypass detection patterns, making it difficult to develop effective patterns. The following example illustrates a pattern that would be processed from an HTTP session, and an obfuscated version that attempts to bypass the pattern.

Pattern:
GET /cgi-bin/phf?

Obfuscation:
GET /cgi-bin/aaaaaaaaaaaaaaaaaaaaaaaa/..%25%2fp%68f?

Both versions result in the same output, yet look very different.

This example is simplistic and may or may not bypass the pattern matching engines of most IDS technology. However, it effectively demonstrates that input can vary and still yield the same output.

Most modern IDSs have greatly improved in this particular area. However, some publicly released signatures can be bypassed due to weaknesses in the signature patterns used.

Copyright 2006 iDefense, A VeriSign Company

1.2 Unicode Evasion Techniques

A similar method of eluding IDS technology involves Unicode. Unicode is a character representation that gives each character a unique identifier for each written language to facilitate uniform the computer representation of each language.

This is problematic for IDS technology because it is possible to have multiple representations of a single character. For example, '\' can be represented as 5C, C19C and E0819C, which makes writing pattern matching signatures very difficult. However, the Unicode standard was recently changed to make multiple representations illegal. It is important to note, however, that some applications may still be using the old standard.

One example of how Unicode affects IDS is present in the Microsoft IIS 4.0/5.0 Directory Traversal vulnerability released in October 2000 by Rain Forrest Puppy. This IIS vulnerability improperly restricts directory listings that were Unicode encoded within the URL request. This allowed remote attackers to view files on the IIS server that they normally would not be permitted to see.

1.3 Denial of Service (DoS) Attacks

Many IDSs employ central logging servers that are used solely to store IDS alert logs. The central server's function is to centralize alert data so it can be viewed as a whole rather than on a system-by-system basis. However, if attackers know the central log server's IP address, they could slow it down or even crash it using a DoS attack. After the server is shut down, attacks could go unnoticed because the alert data is no longer being logged.

Another method attackers might use is to send false positives in an attempt to fill the central log server's disk space. Once the disk space is filled, attacks might go unnoticed because, again, the alert data is no longer being logged.

1.4 False Positive Generation

Another attack similar to the DoS method is to generate a large amount of alert data that must be logged. Attackers can craft packets known to trigger alerts within the IDS, forcing it to generate a large number of false reports. This type of attack is designed to create a great deal of log "noise" in an attempt to blend real attacks with the false. Attackers know all too well that when looking at log data, it can be very difficult to differentiate between legitimate attacks and false positives.

If attackers have knowledge of the IDS system, they can even generate false positives specific to that IDS. Furthermore, most IDSs contain the same, or at least very similar, signatures for several attacks. Both of these factors greatly increase the likelihood of a successful attack because the more signatures that IDSs have in common, and the more attackers know about a given IDS, the more false positives they can generate.

1.5 Session Splicing

Session splicing is an IDS evasion technique that exploits how some IDSs do not reconstruct sessions before performing pattern matching on the data. In addition, some IDSs only reassemble parts of a session because reassembly is a very processor-intensive task.

Copyright 2006 iDefense, A VeriSign Company

The idea behind session splicing is to split data between several packets, making sure that no single packet matches any patterns within an IDS signature. Furthermore, if attackers know what IDS system is in use, they could add delays between packets to bypass reassembly checking. Many IDSs reassemble communication streams, so if a packet is not received within a reasonable amount of time, many IDSs stop reassembling and handling that stream. If the application under attack keeps a session active longer than an IDS will spend on reassembling it, the IDS will stop. As a result, anything after the IDS stops reassembling the session will be susceptible to malicious data sent by the attacker, which would go unnoticed.

Modern IDSs have found ways to handle session splicing. For example, reassembling the full packet stream is now common practice, which essentially puts an end to session splicing attacks.

Copyright 2006 iDefense, A VeriSign Company

2 Complex IDS Evasion Techniques

2.1 Fragmentation

Fragmentation attacks are similar to session splicing attacks in that attackers send packets in blocks that do not trigger an IDS signature or cause an alert. Fragmentation attacks are generally more powerful than session splicing attacks, but attackers can be more creative in evasion. There are two fragmentation methods commonly used to elude IDSs. One method overwrites a section of a previous fragment, while the second method overwrites a complete fragment. This can be useful for attackers because it enables them to write an entire packet of garbage information and craft their attack to blend in with standard protocols. The following are two examples of fragmentation attacks:

Attack 1: Overlap Method

Packet 1: GET /cgi-bin/
Packet 2: aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa/../phxx
Packet 3: f?

This example of fragmentation can overwrite the 'xx' portion of Packet 2 with the data in Packet 3, making the information resemble the following:
GET /cgi-bin/ aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa/../phf?

Attack 2: Overwrite Method

Packet 1: GET /cgi-bin/
Packet 2: some_normal_filename.cgi
Packet 3: /aaa/../aaa/../aaa/../phxx
Packet 4: f?

This example is similar to the first, but in this example, the 'xx' portion is overwritten and the some_normal_filename.cgi packet is completely overwritten with the last two packets. This leaves GET /cgi-bin/phf? as the end result.

It is important to note, however, that IDSs such as Snort have found ways to handle these types of attacks through reassembly.

2.2 Time-To-Live Attacks

Time-to-live attacks are yet another way attackers can bypass IDS technology. For this technique to work properly, attackers must have some knowledge of the internal network topology. If attackers know the distance to the end host and whether an IDS is placed in front of the end host, they can bypass detection. By using a small TTL flag in a TCP packet, attackers can use their knowledge of the network topology and send packets that will only reach the IDS. The IDS, in turn, will think the packet addressed to the end host will make it there, which allows attackers to inject garbage packets into the IDS stream processing. By sending some packets with a large TTL flag, attackers guarantee the packet will reach the end host.

Copyright 2006 iDefense, A VeriSign Company

Using a small TTL flag allows attackers to send packets addressed to the end host to the IDS without the packet ever getting to the end host. The following is an example of a time-to-live attack:

Packet 1: GET /cgi-bin/p TTL 10
Packet 2: some_file.cgi?= TTL 5
Packet 3: hf? TTL 10

This example assumes that the end host is beyond the 10 TTL limit and will receive the data. It also assumes that the IDS is within the 5 TTL limit, and any data lower than that will not reach the destination host. The means that the end host will receive "GET /cgi-bin/phf?" while the IDS receives "GET /cgi-bin/psome_file.cgi?=hf?."

2.3 Invalid RST Packets

The TCP protocol uses checksums to ensure that communication is reliable. A checksum is added to every transmitted segment and it is checked at the receiving end. When a checksum differs from the checksum expected by the receiving host, the packet is dropped at the receiver's end.

The TCP protocol also uses an RST packet to end two-way communications. Attackers can use this feature to elude detection by sending RST packets with an invalid checksum, which causes the IDS to stop processing the stream because the IDS thinks the communication session has ended. However, the end host sees this packet and verifies the checksum value, then drops the packet if it is invalid.

Some IDS systems might interpret this packet as an actual termination of the communication and stop reassembling the communication. Such instances allow attackers to continue to communicate with the end host while confusing the IDS because the end host accepts the packets that follow the RST packet with an invalid checksum value.

2.4 Urgency Flag

The urgency flag is used within the TCP protocol to mark data as urgent. TCP uses an urgency pointer that points to the beginning of urgent data within a packet. When the urgency flag is set, all data before the urgency pointer is ignored, and the data to which the urgency pointer points is processed.

Some IDSs do not take into account the TCP protocol's urgency feature, which could allow attackers to evade IDS, as seen in other evasion techniques. Attackers can place garbage data before the urgency pointer, and the IDS reads that data without consideration for the end host's urgency flag handling. This means the IDS has more data than the end host actually processed. The following is an example of an urgency flag attack:

"1 Byte data, next to Urgent data, will be lost, when Urgent data and normal data are combined."

Packet 1: ABC
Packet 2: DEF Urgency Pointer: 3
Packet 3: GHI
End result: ABCDEFHI

This example illustrates how the urgency flag works in conjunction with the urgency pointer. According to the 1122 RFC, the urgency pointer causes one byte of data next to the urgent data to be lost when urgent data is combined with normal data.

Copyright 2006 iDefense, A VeriSign Company

2.5 Polymorphic Shellcode

Most IDSs contain signatures for commonly used strings within shellcode. This is easily bypassed by using encoded shellcode containing a stub that decodes the shellcode that follows. This means that shellcode can be completely different each time it is sent.

Polymorphic shellcode allows attackers to hide their shellcode by encrypting it in a simplistic form. It is difficult for IDSs to identify this data as shellcode. This method also hides the commonly used strings within shellcode, making shellcode signatures useless.

2.6 ASCII Shellcode

Similar to polymorphic shellcode, ASCII shellcode contains only characters contained within the ASCII standard. This form of shellcode is useful to attackers because it allows them to bypass commonly enforced character restrictions within string input code. It also helps attackers bypass IDS pattern matching signatures because strings are hidden within the shellcode in a similar fashion to polymorphic shellcode.

Using ASCII for shellcode is very restrictive in that it limits what the shellcode can do under some circumstances because not all assembly instructions convert directly to ASCII values. This restriction can be bypassed using other instructions or a combination of instructions that convert to ASCII character representation, which serves the same purpose of the instructions that improperly convert. The following is an ASCII shellcode example:

```
char shellcode[] =
    "LLLLYhb0pLX5b0pLHSSPPWQPPaPWSUTBRDJfh5tDS"
    "RajYX0Dka0TkafhN9fYf1Lkb0TkdjfY0Lkf0Tkgfh"
    "6rfYf1Lki0tkkh95h8Y1LkmjpY0Lkq0tkrh2wnuX1"
    "Dks0tkwjfX0Dkx0tkx0tkyCjnY0LkzC0TkzCCjtX0"
    "DkzC0tkzCj3X0Dkz0TkzC0tkzChjG3IY1LkzCCCC0"
    "tkzChpfcMX1DkzCCCC0tkzCh4pCnY1Lkz1TkzCCCC"
    "fhJGfXf1Dkzf1tkzCCjHX0DkzCCCCjvY0LkzCCCjd"
    "X0DkzC0TkzCjWX0Dkz0TkzCjdX0DkzCjXY0Lkz0tk"
    "zMdgvvn9F1r8F55h8pG9wnuvjrNfrVx2LGkG3IDpf"
    "cM2KgmnJGgbinYshdvD9d";
```

When executed, the shellcode above executes a "/bin/sh" shell. Looking closely, readers will see that 'bin' and 'sh' are contained in the last few bytes of the shellcode.

2.7 Application-Layer Attacks

Application layer attacks enable many different forms of evasion. Many applications that deal with media such as images, video and audio employ some form of compression that allows the media to be sent in a form much smaller than the original, uncompressed form, which increases data transfer speeds. When a flaw is found in these applications, the entire attack can occur within compressed data, and the IDS will have no way to check the compressed file format for signatures.

Many IDSs look for specific conditions that allow for an attack. However, there are times when the attack can take many different forms. For example, integer overflow vulnerabilities could be exploited using

Copyright 2006 iDefense, A VeriSign Company

several different integer values. This fact combined with compressed data makes signature detection extremely difficult.

A recent example of an application layer attack was possible within the Windows Media Player BMP flaw MS06-005. By making shellcode that contained several different forms of "nops" (i.e., no operation opcode), it was possible to create a seemingly legitimate bitmap file. Combined with polymorphic shellcode at the end of the nopsled, this could easily evade any IDS signatures developed for this flaw. In addition, if the IDS does not look for the cause of the flaw (i.e., the size field within the BMP format's header), a different size could be used nearly every time with almost no side effects on exploitation.

Copyright 2006 iDefense, A VeriSign Company

3 Solving The Evasion Problem

So, what can IDS developers and IDS users do to resolve the evasion problem? Surprisingly, the answer for users is very little. The best approach to mitigating the threat of IDS evasion is to maintain security vulnerability awareness, patch vulnerabilities as soon as possible and wisely choose the IDS based on the network topology and network traffic received.

IDS developers have created methods of avoiding many of the evasion techniques discussed above. Methods such as stream normalization, support to properly reassemble fragmented streams and Unicode and UTF8 decoding are just a few of the features to look for when deciding on which IDS to deploy.

3.1 Normalization

Normalization takes obfuscated input and attempts to translate it into what the end host will eventually see. This usually entails encoding in formats such as Unicode and UTF8. The normalization process allows for encoding, translation and the application of pattern matching to the normalized data, which prevents attackers from obfuscating the attack strings using Unicode or UTF8 strings. However, there are other obfuscation methods that could circumvent this such as polymorphic shellcode, ASCII shellcode and other encodings supported by the application being exploited. Some IDSs are attempting to apply normalization to polymorphic shellcode, but it is difficult to spend the time required to decode polymorphic shellcode while trying to effectively monitor the remaining network traffic.

Normalization also applies to network data. Some IDSs normalize fragmented packets and allow those packets to be reassembled in the proper order, which enables the IDS to look at the information just as the end host will see it. In addition, some IDSs change the time-to-live field to a large number, which ensures that it reaches the end host.

Copyright 2006 iDefense, A VeriSign Company

3.2 Packet Interpretation Based on Target Host

When comparing the strengths of evasion techniques to the methods used to prevent them, one can see that IDS is at a disadvantage because these systems attempt to recreate what the end host will see and handle. This is problematic because there are so many disparate methods of communicating data over a network. In order to properly complete this task, the end host's TCP/IP stack should be used rather than trying to recreate the stream in a way that the stream may be handled. In using the host to do the work, the guessing portion of the task is eliminated. This type of system must still handle various forms of encodings and would still be flawed in some ways, but the impact of the majority of network-based evasions would be greatly mitigated.

Another option that may be as effective as host-based IDSs is using modular TCP/IP stacks within an IDS, and using the stacks based on the targeted host's operating system. Before it would be practical, however, this method must be thoroughly researched. For example, the manner in which each operating system handles anomalous traffic must be thoroughly reviewed. Ultimately, this method may prove very effective in mitigating network-level evasions such as fragmentation, RST packet handling and Urgency Flags.

3.3 Time-To-Live Problem

The TTL problem presents some interesting solutions. There are several options that could solve this problem, some of which may even help identify and eliminate future problems.

The first method of dealing with this issue is to automatically change the TTL field to a large value, which ensures that the end host always receives the packets. In such cases attackers cannot slip information to the IDS. As a result, that data never reaches the end host, leaving the end host with the malicious payload. The data slipped to the IDS is intended to only reach the IDS, but if it reaches the end host, the attack is successfully mitigated.

The second method is more complex and provides some other possibilities for the data. The first thing that must be done is the IDS must collect all the MAC addresses within the network it is monitoring. It must then get TTL information for each host and map that information to each host. As a result, the IDS will know the distance to the host for which the data is destined and that host's MAC address. This could lend itself to other forms of detection that are purely signature based. However, if hosts regularly move within the network, this data must be updated.

3.4 Dealing With The Shellcode Problem

One method of detecting polymorphic shellcode deals with looking for nop opcode other than 0x90. There are several op codes that do not touch memory, but alter register values. These opcodes are commonly used within polymorphic shellcode to mask it from IDSs. The current method entails referencing a number of nop opcodes within a threshold and creating an alert if that threshold is reached. This method is fairly accurate, but has been known to yield false positives. In addition, this method does not limit the impact of shellcode and the creative things that attackers can do with it.

Copyright 2006 iDefense, A VeriSign Company

4 Conclusion

While there are many ways to evade IDSs, there equally as many ways to detect attacks that use evasion techniques. For IDS technology to be truly effective, they must attempt to detect all attacks and mitigate evasion. Clearly, this gives attackers the upper hand.

This paper is intended to inform organizations about the strengths and weaknesses inherent in IDSs. IDSs are useful tools, but they have limitations. By using IDS technology, it is easier to track what happens within a network. However, information provided solely by IDSs may not be sufficient enough to thoroughly protect the network. Sometimes the wire will be quiet during an attack, other times it will be noisy. So how can system administrators truly know when an attack is taking place? The answer is to reinforce IDSs with other technologies that complement their inherent strengths.

The soundest approach to security in any network environment is to limit the avenues of attack. This means applying the latest software fixes and patches as soon as possible. Sound security practices do not alleviate the need for IDS technology, but do limit the dependency on such systems.

When deploying an IDS, organization should have a thorough understanding of the network on which it is deployed. It is advisable to identify network entry points, the number of hosts that will be monitored and network traffic when deploying any IDS. Having a thorough understanding of the network topology will help mitigate the evasion techniques discussed in this report and improve the efficacy of IDS technology.

Copyright 2006 iDefense, A VeriSign Company