

Evading NIDS, revisited

Sumit Siddharth 2005-12-06

Published 2005-12-02; updated 2005-12-06 citing additional credits to two researchers.

In this article we look at some of the most popular IDS evasion attack techniques.

We start by looking at attacks that are based on fragmentation, which includes attacks based on different fragment reassembly timeouts of different operating systems. Then we go a step further and look at how various operating systems perform fragment reassembly differently and how this can be useful when performing an IDS evasion. These are known as attacks based on overlapping fragments.

The remainder of this article will then look at attacks based on the TTL field. We will turn our focus to widely popular Snort NIDS and describe how Snort deals with all these attacks, as well as what parameters are involved in configuring snort to stop an IDS evasion.

Introduction

The paper titled "*Insertion, Evasion and Denial Of Service: Eluding NIDS*" by Thomas.H. Ptacek and Tim Newsham [[ref 1](#)] triggered research in the field of eluding NIDS. Ever since this paper was published, people have been busy finding new techniques of eluding NIDS. Most intrusion detection systems (IDS) generally have support for TCP-reassembly and the capability to monitor sessions. Some of the DoS attacks focus on overflowing the stream-buffer cache of the IDS so that the stream being monitored gets disrupted. An Insertion Attack sends packets to an end-system (victim) that will reject, but that the IDS will think are valid, thus giving different streams to the IDS and target hosts. In comparison, an evasion attack sends packets which the IDS rejects but the target host accepts, again giving different streams to the IDS and target. In order to achieve these attacks, attackers also use packet fragmentation where the attack stream is broken into smaller ones. We will now describe some of these evasion techniques.

Fragmentation reassembly timeout attacks

Let's start by defining a few terms.

1. **Fragmentation.** If a packet is too large for the underlying link layer, it may be split by any router (unless this behavior is explicitly disabled) into multiple fragments. This is known as fragmentation. Systems need to keep fragments around, wait for future fragments, and then reassemble them. The fragment/packet should have a TTL value more than 1 for it to be passed by a router. As soon as the router receives a packet/fragment with a TTL of 1 it decreases the TTL value by 1. Since now the TTL value is zero, it will discard the packet/fragment and send an ICMP "Time Exceeded In Transit" error message back to the original sender (ICMP type=11, code=0).

2. **The IP Fragment Reassembly Timeout.** This refers to the maximum amount of time a fragment will be held, unassembled, before it expires and is flushed. It differs from one operating system to another - note that this technique is also used in OS fingerprinting. An IDS which does TCP reassembly also has an IP fragment reassembly timer. [ref 2] For instance, Snort by default has a fragment reassembly timeout value of 60 seconds after which the initial fragment will be dropped and the stream will be flushed. This is the 'timeout' parameter for the frag2 (also for frag3) preprocessor and is set in "snort.conf."
3. **ICMP Fragment Reassembly Time Exceeded message.** RFC-792 [ref 3] states that if a host reassembling a fragment datagram cannot complete the reassembly due to missing fragments within its time limit, it discards the datagram and it may send a Time Exceeded Message (ICMP type=11, code=1). If fragment zero is not available then no time exceeded need be sent at all. [ref 4]

We will now consider several different cases where attacks can be made. All these attacks were discovered by different people at various stages during the development of NIDS. The first two cases that we consider here were discovered by Dan Kaminsky, and are the most recent addition to the NIDS Evasion attacks.

Case 1: The IDS fragmentation reassembly timeout is less than fragmentation reassembly timeout of the Victim.

Attack scenario:

Suppose the IDS fragmentation reassembly timeout is 15 seconds and the system is monitoring some Linux hosts which have a default fragmentation reassembly timeout of 30 seconds. As shown below in Figure 1, after sending the first fragment the attacker can send the second fragment with a delay of 15 seconds but still within 30 seconds.

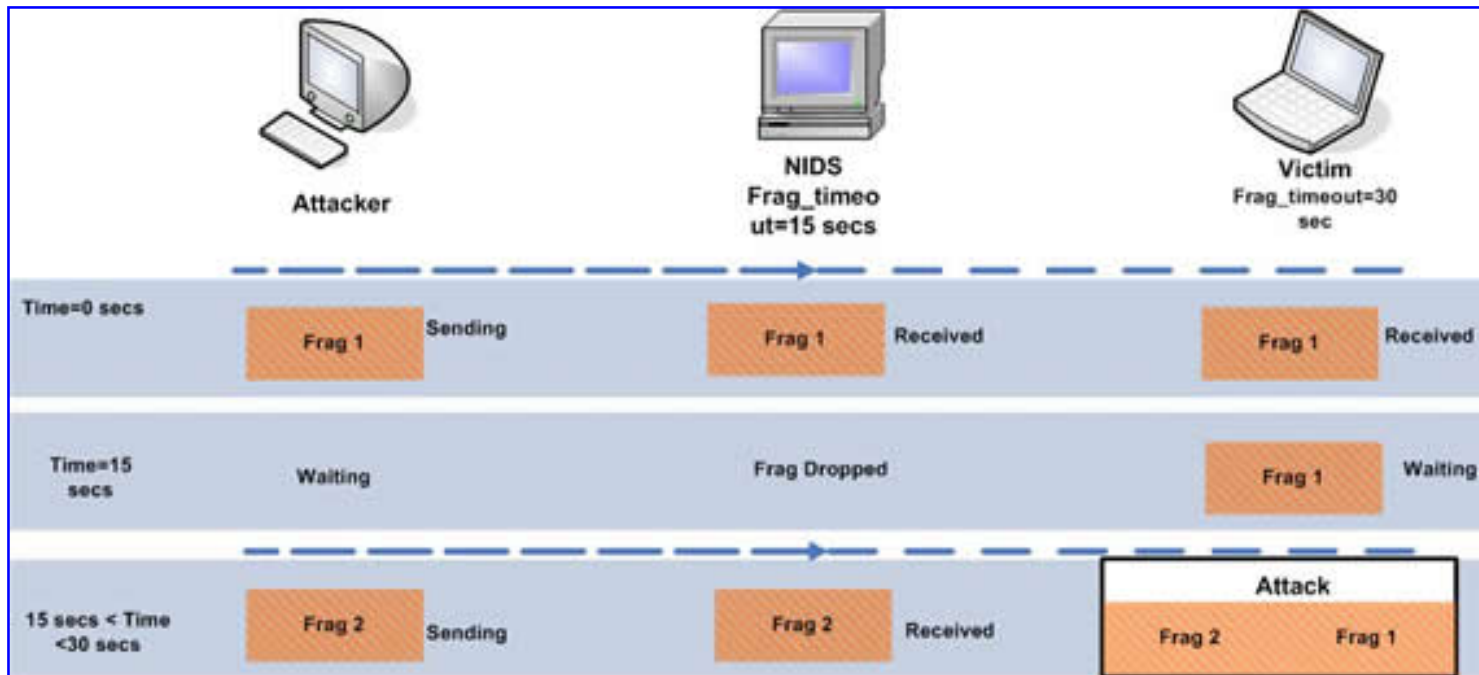


Figure 1. Attack where the NIDS fragmentation re-assembly timeout is less than the Victim's fragmentation reassembly timeout.

Now, the victim reassembles the fragments whereas at the IDS the fragmentation reassembly timeout parameter kicks in and a timeout occurs. Remember, here the second fragment received by the IDS will be dropped as the IDS has already lost the first fragment, due to time out. Thus the victim will reassemble the fragments and will receive the attack whereas the IDS will not make any noise or generate alerts.

Case 2: The IDS fragmentation reassembly timeout is more than the fragmentation reassembly timeout of the operating system.

Attack scenario:

By default, Snort has a fragment reassembly timeout of 60 seconds. Compare that to Linux/FreeBSD where it is 30 seconds. This can be evaded as well. As shown below in Figure 2, consider that the attacker has fragmented the attack packet into four segments: 1,2,3,4.

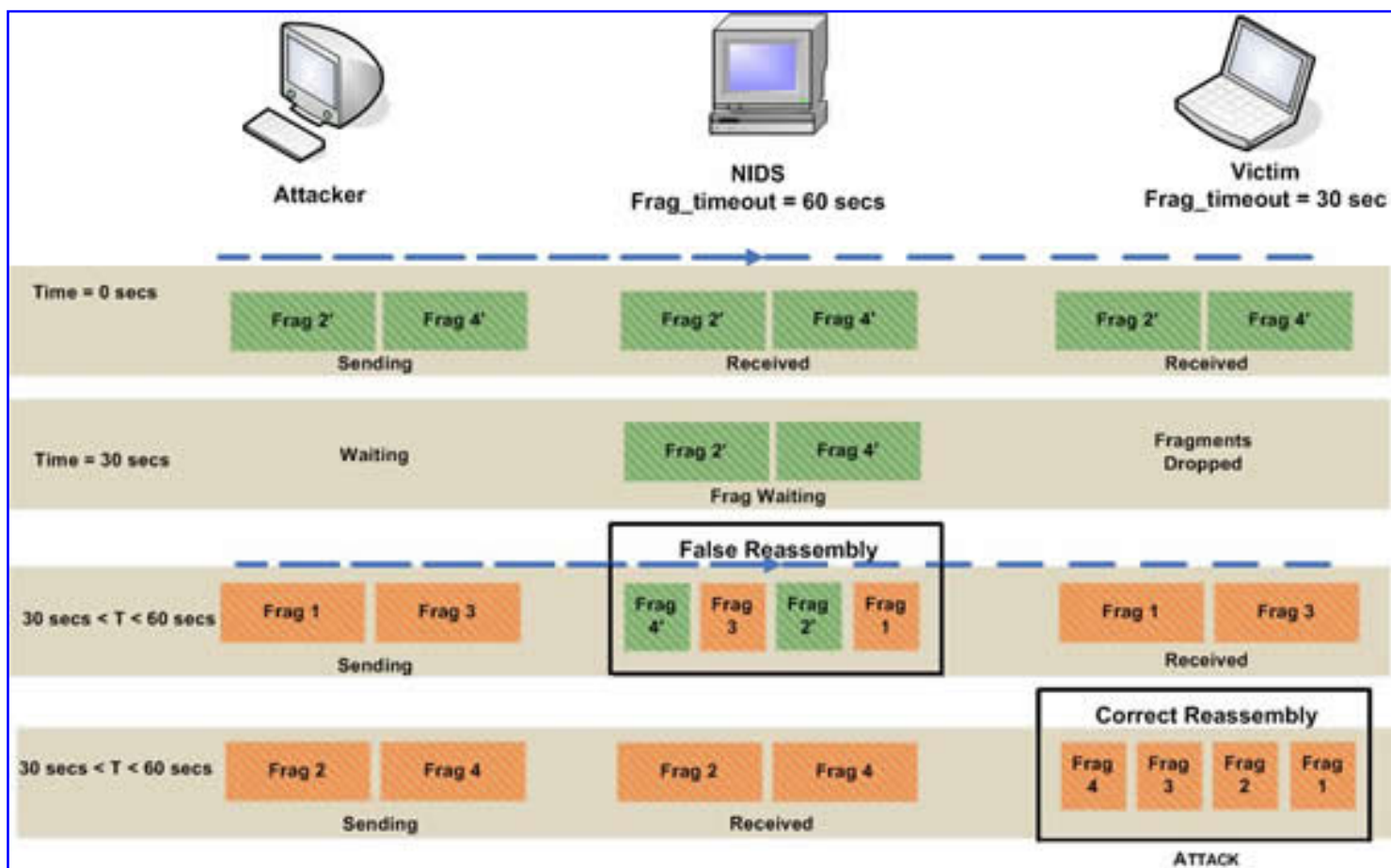


Figure 2. Attack where the NIDS' fragmentation re-assembly timeout is greater than the victim's fragmentation reassembly timeout.

The attacker sends frag2 and frag4 with a false payload (referred as 2', 4'), which are received by both the victim and the IDS. She waits until the fragments' reassembly timeout occurs at the victim's end and it drops the initial fragments (30 seconds in this case).

The beauty of the attack is that the victim still has not received fragment 1 so it will quietly drop the fragments and no ICMP error message will be thrown by the victim. The attacker then sends packet (1, 3) with a legitimate payload. At this stage, the victim has only fragments (1, 3) whereas the IDS has fragments (1,2',3,4'). Remember that the 2,4 fragments sent by attacker have a false payload.

Since the IDS has all the 4 fragments it will do a TCP reassembly. Also, since fragments 2 and 4 have false payloads the net checksum computed will be invalid. So, the IDS will drop the packet. However, now the victim has only fragments 1,3. If the attacker now sends fragments 2, 4 again with valid payload, the IDS will have only these two fragments (2,4 with a valid payload as the previous fragments have been reassembled and dropped) whereas the victim will have all (1,3,2,4) fragments all with a valid payload, and it will do a reassembly and read the packet as an attack.

TTL based attacks

These attacks require the attacker to have a prior knowledge of the topology of the victim's network. This information can be obtained by using tools such as traceroute [ref 5] which give the information on the number of routers between the attacker and the victim. A TTL based attack is shown below in Figure 3.

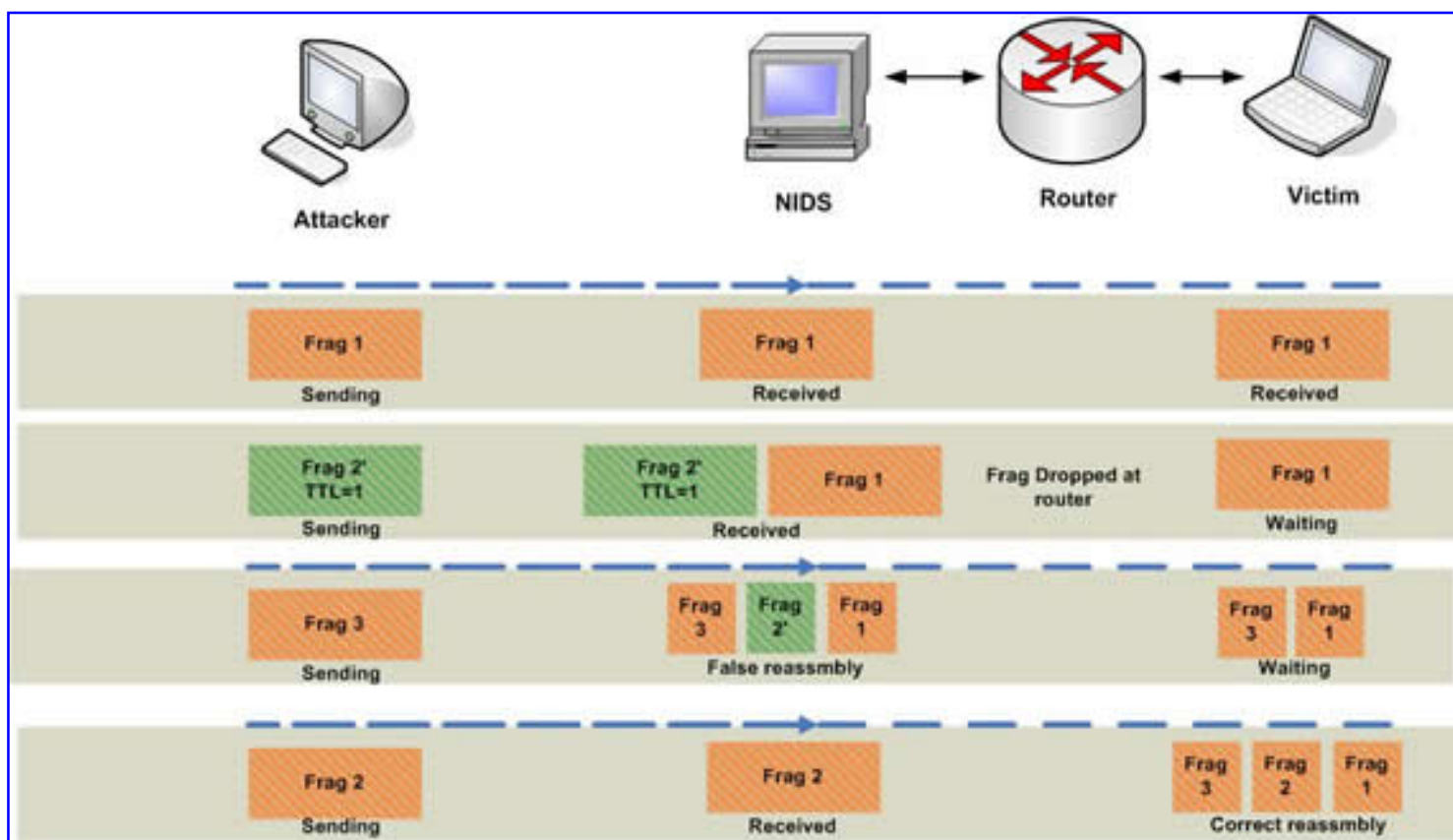


Figure 3. TTL-based evasion attack.

A router is present between the IDS and a victim - and the attacker is assumed to have this prior information. The attacker carries out the attack by breaking it into three fragments. She sends fragment 1 with a large TTL value and this is received by both the IDS and the

victim. However, the second fragment (frag2') sent by the attacker has a TTL value of 1 and also has a false payload. This fragment is received by the IDS whereas the router (which is situated between the IDS and the victim) discards it as the TTL value is now reduced to zero.

The attacker then sends fragment 3 with a valid TTL. This makes the IDS perform a TCP-reassembly on fragments (1,2',3), whereas the victim still waits for the third fragment. The attacker finally sends the third fragment with a valid payload and the victim performs a reassembly on fragments (1, 2, 3) and gets the attack. At this stage, the IDS has only fragment 3 as it has already performed a reassembly and the stream has been flushed.

Overlapping fragments

In addition to these attacks, which are based on the fragmentation reassembly timeout, there is another class of attacks based on overlapping fragments. Paxson and Shankar pointed out in their paper titled "*Active mapping: resisting NIDS Evasion without altering traffic*" [ref 6] that different operating systems perform fragmentation reassembly differently. Based on this model they deduced that there are five different re-assembly policies.

In this paper, we will give a brief description of two of these fragmentation reassembly policies, in particular, the ones named first and last.

First. This is where the operating System favors the original fragments with a given offset. For example, Windows 95/98/NT4/ME/W2K/XP/2003. **Last.** This is where the operating System favors the subsequent fragments with a given offset. For example, Cisco IOS.

The first and the last fragmentation policy are explained below in Figure 4.

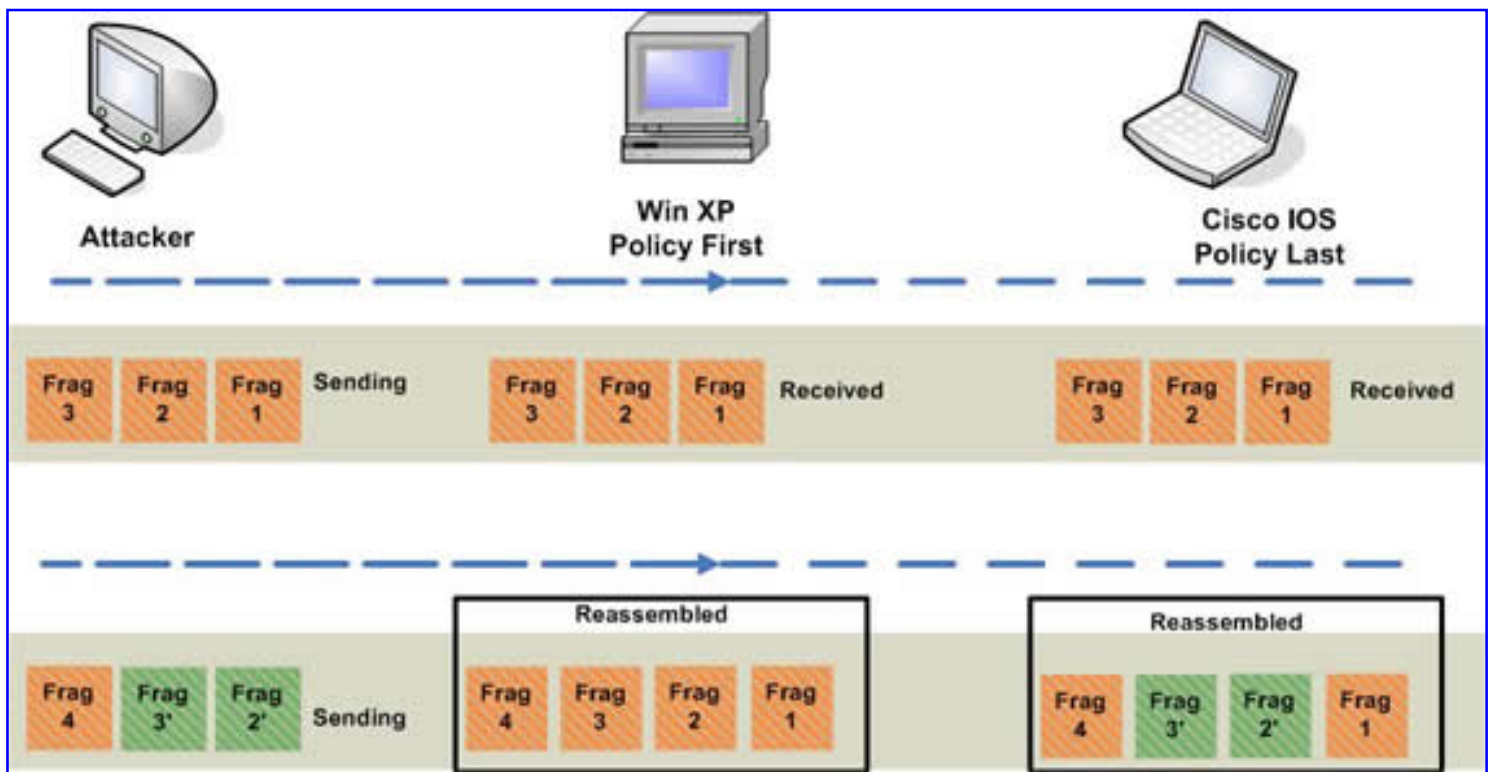


Figure 4. Windows -XP favors original fragments whereas Cisco IOS favors subsequent fragments.

The attacker carries out the attack by breaking it in 4 fragments. She sends fragment 1, fragment 2 and fragment 3 first, which both operating systems accept. Now the attacker sends fragment 2', fragment 3' and fragment 4. Here the payload of fragment 2' and fragment 3' is different from fragment 2 and fragment 3, respectively, but the fragment offset and the length of the fragment along with other fields in the IP-header remain the same.

In such a scenario, an operating system doing a fragment reassembly based on policy First will do a reassembly on fragments 1,2,3,4 whereas the operating system doing a fragments reassembly based on policy Last will reassemble fragments 1,2',3',4.

The mapping of different operating systems, along with their respective fragmentation reassembly policy, is listed below in Table 1.

Fragmentation Policy	Platforms
BSD-right	HP JetDirect
BSD	AIX 2, 4.3, 8.9.3, FreeBSD, HP-UX B.10.20, IRIX 4.0.5F, 6.2, 6.3, 6.4, NCD Thin Clients, OpenBSD, OpenVMS, OS/2, OSF1, SunOS 4.1.4, Tru64 Unix V5.0A,V5.1, Vax/VMS
Linux	Linux 2.x

First	HP-UX 11.00, MacOS (version unknown), SunOS 5.5.1,5.6,5.7,5.8, Windows (95/98/NT4/ME/W2K/XP/2003)
Last	Cisco IOS

Table 1. Fragmentation policies and associated platforms.

A more detailed description of the different fragments reassembly policy can be found in the Snort documentation area. A relevant part of that affects our discussion will be quoted in the next section.

Snort evasion countermeasures

Snort is undoubtedly the most popular NIDS. It provides the solution for dealing with these evasion attacks. The frag3 preprocessor is a target-based IP defragmentation module for Snort. Quoting from the official Snort site, "Frag3 is intended as a replacement for the frag2 defragmentation module and was designed with the following goals:

1. Faster execution than frag2 with less complex data management.
2. Target-based host modeling anti-evasion techniques." [ref 7]

The Frag3 preprocessor of Snort [ref 6] implements target-based fragmentation policies by allowing a user to identify the fragmentation reassembly method and the corresponding fragment timeout value that is applied to a particular destination IP address or subnet. This makes the IDS perform a fragmentation reassembly in the same way the different IP addresses on a home network would do it. For example, assume that you have an entire subnet (192.168.1.x) that consists only of OpenBSD hosts and you want to configure Snort to use the appropriate reassembly policy for these hosts. You only need to enable the frag3 preprocessor in the configuration file and designate the appropriate fragmentation policy in a frag3 engine statement, as follows:

```
preprocessor frag3_global:
preprocessor frag3_engine: policy bsd\
    bind_to 192.168.1.0/24 \
    timeout 30 \
    min_ttl 2
preprocessor frag3_engine: policy first\
    bind_to[10.1.47.0/24, 172.16.8.0/24]
```

Now, any overlapping fragments that Snort sees [ref 8] destined for subnet 192.168.1.x are reassembled using the "bsd" fragmentation policy, and the fragment timeout field is set according to the BSD operating system. Similarly, Snort will reassemble all the fragments destined to subnet 10.1.47.x using the fragmentation policy "First". Snort frag3 preprocessor also provides the min_ttl field in which one can specify the minimum

acceptable TTL value for a fragment. If there exists a router between the IDS and the subnet of 192.168.1.x, then by specifying the minimum acceptable TTL for fragments destined to this subnet as 2 we can prevent TTL based evasion attacks.

Conclusion

In this paper we've looked at a few IDS evasion attacks and the different methodologies involved with various attack. We have shown that it is important to know how different operating systems perform fragmentation reassembly. We finally conclude that these evasion attacks can be prevented by setting the frag3 preprocessor in Snort properly.

The different attack scenarios presented in this article are a result of an analysis of a compilation of different IDS evasion attacks. It showcases the varying fragment reassembly mechanisms based on the overlapping of fragments and the reassembly timeout for different operating systems

References

- [ref 1] Insertion, Evasion and Denial Of Service: -Eluding Network Intrusion detection System -Thomas H. Ptacek, Timothy N. Newsham. <http://www.snort.org/docs/idspaper/>
- [ref 2] http://www.doxpara.com/slides/Black%20Ops%20of%20TCP2005_Japan.ppt
- [ref 3] RFC-792 <http://www.faqs.org/rfcs/rfc792.html>
- [ref 4] Snort - Sid: 410 <http://www.snort.org/pub-bin/sigs.cgi?sid=410>
- [ref 5] Traceroute: www.traceroute.org
- [ref 6] Active Mapping: Resisting NIDS Evasion Without Altering Traffic <http://www.icir.org/vern/papers/activemap-oak03.pdf>
- [ref 7] Snort Frag3 development paper <http://www.snort.org/docs>
- [ref 8] Snort Frag3 README file <http://cvs.snort.org/viewcvs.cgi/snort/doc/README.frag3>

Other useful links

Fragroute: <http://www.monkey.org/~dugsong/fragroute/>
TCP/IP Lean by Jeremy Bentham <http://www.iosoft.co.uk/tcplean.php>

About the author

[Sumit Siddharth](#) is an Information Security Analyst working for [Network Intelligence India \(NII\)](#). His areas of interest include Intrusion Detection & Analysis, Penetration Testing and Network Forensics. Sumit is a graduate from IIT Kanpur.

[Privacy Statement](#)

Copyright 2006, SecurityFocus