

Understanding IDS Active Response Mechanisms

Jason Larsen 2002-01-29

Understanding IDS Active Response Mechanisms

by *Jason Larsen*, and *Jed Haile*

last updated January 29, 2002

Introduction

Debates still rage in the developer community over which methods of detecting attackers are best, but IDS customers as a whole are satisfied with the current IDS technology. To get an edge on the competition, many of the IDS vendors are adding active response capabilities to their products.

The concept underlying this tactic is that the IDS will detect an attacker and then move to stop his attack. The problem is that any attacker with a basic knowledge of TCP/IP can easily defeat these mechanisms directly or simply knock the network offline often enough that the Admin is forced to turn off the feature. It is important for Admins to know the limitations of active response mechanisms to avoid being blindsided by them.

Most response mechanisms are one of two types:

- Session Sniping
- Firewall Update

Session Sniping

Session sniping is by far the most popular among IDS vendors. It's popular because it doesn't require drivers for any external device (like firewalls) and is easy to implement. The mechanism is simple. I'm going to drop down into some basic TCP mechanics, so I apologize to everyone who works with this stuff all the time.

Let's take an attacker with a 51-byte exploit on a network that can transfer 20 bytes of data per packet. Let's say the IDS has a signature "system32/cmd.exe" and can do full defragmenting, stream reassembly, and unicode/hexcode/escape/base36 decoding. The IDS is going to catch the exploit (unless the attacker modifies it some way) and generate an alert.

The attacker's IP stack is going to choose an Initial Sequence Number (ISN) at random and break the exploit up into three packets like so:

Data	/scripts/ ..%c0%af../ w			innnt/system32/cmd.exe	e/c+dir+foo
Offset	0	20	40	51	
Sec Num	100	120	140	151	

(Yes I know this won't work without modification. It's just an example.)

Some TCP/IP stacks will send the packets one at a time waiting, for an acknowledgment after each packet. Some stacks will send all three packets at once and then resend some of them if they don't get acknowledged. Still others send one at a time at first and then send more as the session becomes established. The point is the stacks have to be able to deal with any number of packets in flight at the same time.

In the above example, the IDS is going to alert on the third packet because it takes both packets 2 and 3 to complete the signature. An IDS that employs session sniping will forge a TCP RESET packet to both ends of the connection. The stacks will interpret the RESET packet as a request from the other side to stop communications. They will tear down any states and flush any buffers. At this point, the exploit probably still sitting in the target machine's TCP/IP stack's buffer and hasn't been delivered to the application the exploit will be flushed and never delivered.

The RESET packets have to have the correct sequence/acknowledgement numbers or they will be ignored. In this case the acknowledgement number must be 152 (one higher than the last sequence number). If you sent a RESET packet back with an acknowledgement number of 141, it would be ignored by the stack as broken or invalid traffic.

Bypassing Session Sniping

Session sniping can be bypassed in a variety of ways, most of them based on timing.

If the exploit doesn't require an interactive session, you can simply set the PUSH flag on the TCP packets. TCP/IP stacks generally don't deliver each chunk of data to the application as it arrives. Most of the time, it gets too expensive in terms of software interrupts and context switches to push every little bit of data up to the application as it arrives. The stack accumulates data in a buffer until the buffer is full and then pushes the whole buffer up to the application at one time. In the above example, all 51 bytes of the exploit will be delivered to the application at once.

Some applications want the data as fast as possible and are willing to pay the extra overhead to

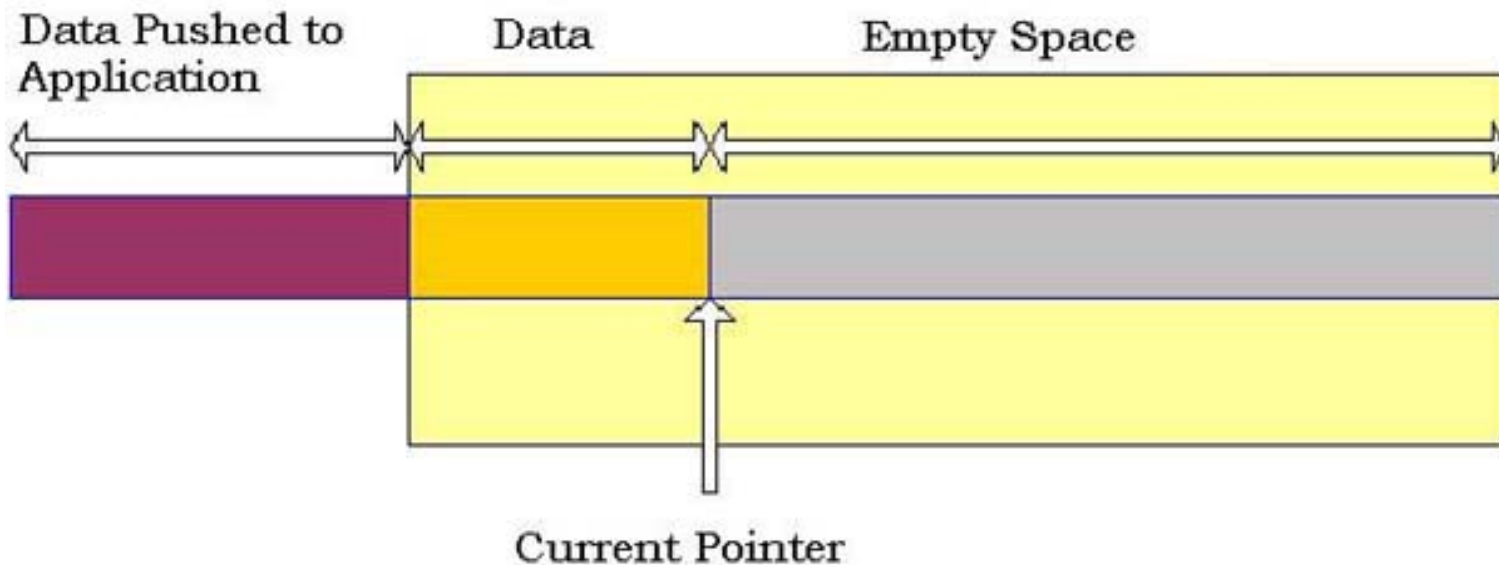
get it as soon as possible. The PUSH flag instructs the stack to deliver the data to the application as soon as it arrives. If you wanted to get a directory listing, this wouldn't help you because the session would be torn down before the response was delivered. But if you wanted to copy the password file to a directory that is accessible to the Web server, you wouldn't care if the session is torn down after the exploit is processed. Simply setting the PUSH flag would complete the exploit.

If you need to keep the session open, there are a couple of techniques you can use. One is just a variant of the other. The trick is to get the target machine to ignore the RESET packet. The IDS will think it has torn down the session and life is good for the attacker.

The first just takes advantage of the lag time of the IDS. It takes time for an IDS to pull a packet off the wire, detect the exploit, generate the RESET packets, and drop the RESET packets onto the wire. Many IDS's use libpcap to pull packets off the wire. Many are also run on top of *BSD. BSD variants use Berkeley Packet Filters, which have huge buffers by default. On a typical network, it may be half a second before the RESET packet is ready to hit the wire. Linux and Solaris perform better, but still have lag.

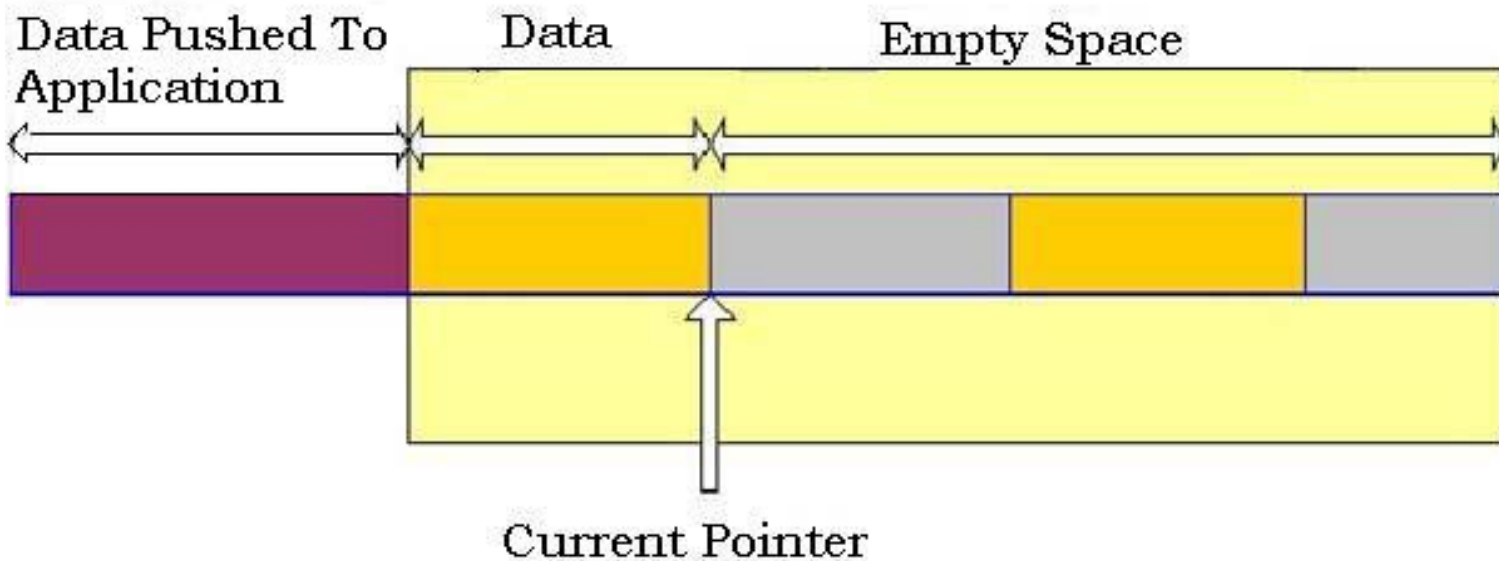
To defeat the IDS, the next packet in the TCP session has to arrive before the RESET packet. I'm going to drop back into some TCP mechanics again, so I apologize to anyone who doesn't need a refresher.

The TCP stack works on a window. Some of the data received has already been pushed up to the application, and some of it is waiting in the buffer to be pushed up. There is also some empty space waiting for new data to arrive. Collectively the data in the buffer and the empty space is known as the window. Only data in the window can be sent/received/reset/etc. Data before the window has already been pushed up to the application. Data after the window hasn't been requested yet and will be ignored if received.



The TCP stack also maintains a current pointer (CP). This is the pointer to the next piece of data the stack expects to receive. The CP is basically the acknowledgement number. If the stack has received 76 bytes, the acknowledgement number will be 77. That's basically the stack's way of saying, "Hey, I've got everything up to byte 76, so send me the next chunk starting with byte 77." When the next chunk arrives, the current pointer will be moved to the end of that chunk.

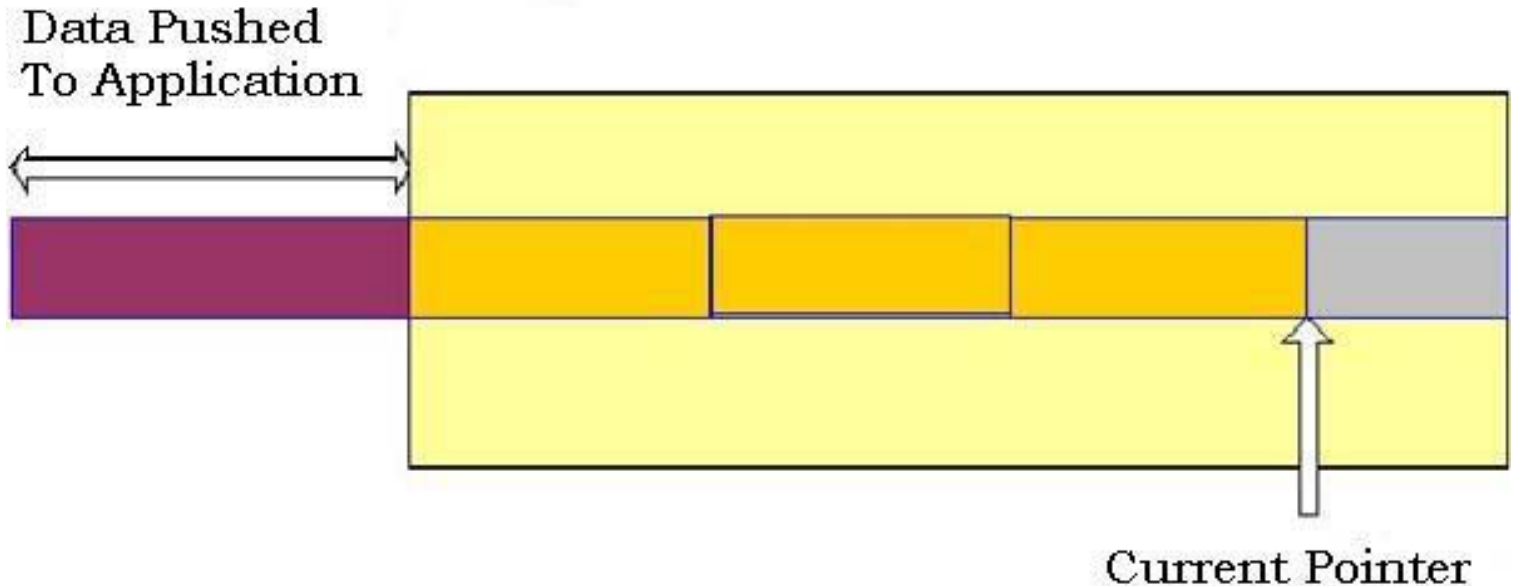
Out-Of-Order Data



Chunks don't necessarily have to arrive in order. A chunk starting at byte 90 can arrive before the chunk starting at byte 77. The out-of-order chunk will be copied into the buffer, but the CP will

remain at 77 until the chunk starting at 77 arrives. At that point, the CP will be moved all the way to the end of the received bytes in one move.

Missing Piece Archive



In most stacks, the RESET has to coincide with the CP or the packet is ignored. Knowing that, an attacker can construct a follow-on packet that defeats the IDS RESET. In our example above, that attack required three packets. The attacker constructs a fourth packet containing nothing but a space or another harmless packet. The attacker then drops the third and fourth packets on the wire in rapid succession. The IDS sees packet 3 and generates a RESET based on that packet. While it generates the RESET, packet 4 arrives at the target machine's stack and moves the current pointer. The RESET packet arrives an instant later and is ignored by the target's stack.

There is also a variant of the method that makes it impossible for the IDS to tear down the session, but takes a little more coding on the part of the attacker. In this method, the attacker simply sends packet 4 before packet 3. Packet 4 is copied into the buffer when it arrives, but the CP isn't moved. As soon as packet 3 arrives, the CP is moved all the way to the end of packet 4. The RESET will be generated off of packet 3, but no matter how fast it gets there, it will be ignored because the CP has moved.

Detecting Session Sniping

It should become obvious to anyone running TCPDump that a particular network is running an IDS

that uses session sniping. The attacker can generate packets that contain data likely to trigger an alert and send them into the network. The particular IDS can be determined by the artifacts in the RESET packet. For a more detailed explanation how this is accomplished do a Google search for pOf.

Firewall Updates

A second method that IDSs are starting to use to actively respond to attackers uses firewall rules manipulation. In this scenario, the IDS instructs the firewall to drop all traffic coming from the source IP of the attacker and some remove the ban after a period of time has expired. Some IDSs require multiple detects to ban an IP address. The thought behind this kind of response is to stop the attacker from further using the foothold that the exploit gave him.

The conditions the IDS will ban an IP on are easy to deduce. The attacker simply sends in packets likely to generate alerts while doing a ping or equivalent. When the ping stops, the attacker knows he's being banned.

Bypassing Firewall Updates

Attackers can adopt a hit-and-run tactic to evade most of the effects. Most IDSs have a wealth of signatures for exploits, but are not good at detecting the numerous backdoors that are available on the Internet. Updating firewall rules usually takes a second or two so there is more an enough time to break in and run a script that installs a backdoor. The attacker can then switch IPs and remotely admin the machine at his leisure.

If the attacker wants, he can bounce off of bouncable FTP servers (see phrack 51) for the initial attack. Many HP printers are useful for bouncing and are easy to find.

Most attackers never go to the trouble though. They simply start disabling the network by spoofing attacks from common IPs. The IDS will happily ban cnn.com, ebay.com, aol.com, etc. like any other attacker. Soon the Admin's phone is flooded with calls demanding the network be restored to usability. Soon after, the firewall update mechanism is usually disabled.

If the attacker spoofs attacks from the network's upstream router or it's DNS servers, the entire network can be disabled.

Most such IDSs can be made to block an IP simply by portscanning. The configuration `nmap -sS -P0 -S <host to ban> -T5` will take out most networks.

Conclusion

Common active response measures can be more effective than no response, but they are by no means a sure way of securing a network. Anyone with a basic knowledge of TCP/IP can defeat the mechanisms, bypassing them or turning them back on the network.

Firewall updating software should be approached with extreme caution with rules to prevent it from banning common Internet services, trading partners, and internal resources. Allowing the network to be shut down repeatedly can be the quickest path to an insecure network.

In the future, IDS vendors should adopt more sane measures such as dropping the packets with inline filters or integrating the IDS with firewall to only filter specific attack. No amount of automation is going to replace vigilance on the part of the security personnel.

[Privacy Statement](#)

Copyright 2006, SecurityFocus