

**The Science of Vulnerability Filters:
A Virtual Software Patch**

ABSTRACT.....	3
INTRODUCTION.....	2
AWARD WINNING ACCURACY AND PERFORMANCE	2
FILTERS FOR A BLOCKING DEVICE.....	3
FILTERS AND SIGNATURES	3
RULES OF IPS FILTER WRITING.....	4
DERIVING A ZERO FALSE NEGATIVE FILTER	4
DERIVING A ZERO FALSE POSITIVE FILTER	5
DIFFERENT TYPES OF FILTERS EXPLAINED BY EXAMPLE.....	5
THE BLASTER/NACHI RPC DCOM BUFFER OVERFLOW	6
EXPLOIT-SPECIFIC FILTERS.....	7
VULNERABILITY FILTERS	7
POLICY FILTERS	8
PROTOCOL ANOMALY FILTERS	9
FILTER SUMMARY.....	11
TIPPINGPOINT'S VIRTUAL SOFTWARE PATCH.....	11
TCP REASSEMBLY AND ATTACK DETECTION	11
OUT OF ORDER PACKETS	12
EVASION RESISTANCE	12
ATTACK RECOGNITION UNDER LOAD	13
DIGITAL VACCINE SERVICE.....	13
CONFIGURATION AND TUNING.....	13
CONCLUSION	14
REFERENCES.....	15

Abstract

In this paper, we explore the technical basis behind a *virtual patch*, also known as a *virtual software patch*, a function provided by high-performance Intrusion Prevention Systems (IPS). A virtual software patch is a powerful frontline defense against compromise that can be deployed in minutes to protect an entire organization. The virtual patch capability relies on a special type of IPS filter called a “vulnerability” or “vulnerability-based” filter. Vulnerability filters *precisely* block attempts to exploit a particular software flaw, while allowing non-attack traffic to pass unhindered.

We will consider a specific attack example - the Microsoft RPC DCOM buffer overflow exploited by the Blaster and Nachi worms. The context of the example helps quantify the significant research responsibilities borne by the filter developer, and shows how engine capabilities play a critical role in the quality of an IPS's security protection. In the RPC DCOM example, as with any vulnerability, the implementation challenge is threefold. First, the filters must be designed to avoid false positives, since blocking legitimate traffic will cause a denial of service. Second, the filters must be resistant to evasion techniques, since a missed attack may lead to network compromise. Third, and most importantly, the detection engine must be powerful enough to support the necessary test criteria, while operating in-line at practical network speeds and latencies.

Weak engines are often forced to forego precise vulnerability filters in favor of simpler exploit-specific and policy-type filters. Simpler filters place a minimal performance burden on the engine, but also give rise to false positives and false negatives. In general, the weaker the engine, the more often IPS filter development reduces to a choice between either (a) implementing simple filters and having acceptable performance, or (b) implementing precision filters and having unacceptable performance, assuming the filter logic can be supported at all. Some vendors try to avoid the choice altogether by implementing filter logic that pushes the computational limits of the engine, and then handling individual customer performance problems through site-specific IPS configuration and tuning.

In order for an IPS to provide widely applicable real-world value, however, the device must perform well on all fronts simultaneously. That is, the IPS must implement high precision vulnerability filters, must be able to handle a heavily loaded gigabit network with a full filter suite enabled (no dropped packets), and must be able to perform a useful virtual patching function without requiring the administrator to perform tedious tuning and configuration tasks.

Introduction

It has become widely accepted that firewalls and intrusion detection systems (IDS) are not sufficient to protect networks from compromise. Firewalls do not adequately analyze application-layer protocol data for signs of attack, and intrusion detection systems do not take any action to stop an attack that is detected. The solution has traditionally been to patch servers and workstations host-by-host, a time consuming process. For some organizations, the challenge in patching lies in testing each patch to understand how it will affect the performance of critical systems. Others find the biggest challenge in reaching each individual in a diverse user base and persuading them to install the patch. In all cases, the patching process takes time, and there is a significant window of opportunity where hosts are unprotected and the vulnerability can be exploited. Organizations need a new security solution.

Intrusion Prevention Systems (IPS's) fill this gap and can revolutionize an administrator's approach to network defense. An IPS operates **in-line** in a network, **blocking malicious traffic**; not just passively detecting. The system analyzes active connections and intercepts attacks in transit, so malicious attack traffic never reaches the target. The device typically has no MAC addresses or IP addresses, and is thus a "bump in the wire." Legitimate traffic passes unhindered through the system at full network speed and with microsecond latency.

Out of the box, the TippingPoint IPS automatically sets hundreds of filters to block in what is called the "Recommended" configuration. These filters recognize traffic that is known to be malicious at all times, under all conditions, in all network environments. The administrator only needs to turn the system on, configure a username and password for the administrative interface, and plug in the data cables. At this point the system is up and running and blocking attacks, shielding vulnerable systems from compromise. There is no need to configure, correlate, integrate or tune anything before the TippingPoint is providing value.

The primary value of an IPS is that it can provide a "virtual patch" functionality that protects vulnerable systems from compromise when host-by-host patches have not been applied and the network is running the protocol at risk. It is possible for an IPS filter, or a suite of filters, to effectively provide a barrier to all attempts to exploit a particular vulnerability. This means different attackers can come and go, the exploit codes can all be different, attackers can use their polymorphic shellcode generators and other evasion techniques, and the filter will reliably block all the attacks, while allowing legitimate traffic through. A filter that operates in this manner is called a "vulnerability filter," and such filters make unpatched systems appear patched from an external attacker's point of view. A virtual patch is a powerful, scalable protection against compromise.

Award Winning Accuracy and Performance

TippingPoint specializes in virtual software patch creation for high-speed networks, providing a solution that operates intelligently out of the box. TippingPoint's IPS recently received the prestigious "NSS Gold Award" from the internationally recognized NSS Group Network Testing Labs. The NSS Gold Award is the highest achievement level recognized by NSS, and the award has only been granted three other times in the NSS Group's six-year history. Further, no other IPS or IDS product has ever won the award, despite the fact that NSS has performed five rounds of IDS group product testing.

NSS tested TippingPoint along with competing products from ISS, NetScreen, NAI and TopLayer. All products were evaluated based on performance, attack recognition and usability,

via an extensive test suite containing over 750 individual tests. Note that the IPS attack recognition test was an updated and more challenging version of that used in recent IDS group evaluations. A few quotes from Bob Walder, president of NSS [1,2]:

“The NSS Gold Award is a standard that we carefully reserve for products that we deem near perfect at the time of testing. In our testing, which covered all aspects of performance, security accuracy and usability, TippingPoint demonstrated outstanding levels of performance. In fact, TippingPoint’s product is the only gigabit IPS we tested that accurately blocked 100% of our attack suite without blocking any legitimate traffic at full wire speed.”

“TippingPoint achieved the highest “out of box” score in all of our signature recognition, false negative and false positive tests, and was one of only two products to achieve a clean 100 per cent detection following a signature update.”

“The concept of Recommended Settings is a very simple one and yet the effect on the usability of the system is impressive, making this product extremely straightforward to deploy in blocking mode from day one. Resistance to false positives was excellent, and we would be quite happy to deploy this device in a live network with the Recommended Settings enabled. Most users would choose the Recommended Settings from day one and might never alter them.”

In the remainder of this paper, we explore the science behind virtual patch creation for high-speed, in-line intrusion prevention systems.

Filters for a Blocking Device

The IPS filter writer works to design filters that have no false positives, cause no performance degradation, and are hardened against attackers who are specifically attempting to evade detection. This task generally requires a change in philosophy from that traditionally accepted in the IDS community. Further, the task requires unprecedented power and flexibility in both the detection engine and the filtering language.

Filters and Signatures

First, some definitions are in order. The word **signature** is used to describe the detection logic; that is, the collection of test criteria used to discriminate attack traffic from benign traffic. In general, *signature* is a rather abstract term that describes the classification algorithm, but says nothing about what action is taken in response to the positive identification of an attack. Since IDS systems only identify attacks, but take no responsive action, the word *signature* is always used when discussing IDS.

On the other hand, the word **filter** is used when referring to the detection logic in combination with an assumed blocking action. Filtering implies *pruning*, or the removal of something from the traffic stream. The term *filtering* is generally used when discussing firewalls, since firewalls actively remove certain types of packets from the overall flow. However, the “intelligence” behind firewall filtering has traditionally been rudimentary, based mostly on port numbers and protocol types. Hence, there is little worth in discussing the “detection logic” or “signature” driving a particular firewall filter.

The entry of intrusion prevention systems onto the scene has muddied the terminology waters, causing a great deal of confusion for anyone involved in network security. Neither of the two legacy terms - *signature* or *filter*, adequately describe the capabilities of an IPS when interpreted

with a traditional IDS or firewall mindset. An IPS possesses intelligence similar to an intrusion detection system, in that the device is able to discriminate very precisely between attack traffic and benign traffic. The IPS test criteria used to detect each attack are certainly worthy of the title “signature” in terms of flexibility and classification power. Further, the IPS actively prunes malicious traffic from the packet flow, making the word “filter” immediately applicable. However, marketers are left in a quandary. If an IPS is described using the word *signature*, this does not adequately conjure to mind the blocking power of the device. On the other hand, if the IPS is described using the word *filter*, people may assume that the “intelligence” of the device is limited to simplistic firewall-style rule matching. Further, there is still one more factor that makes the confusion even worse.

Over the years, IDS products have suffered from rather extreme false positive problems, to the point where IDS and false positives are thought of as two sides of the same coin. In addition, IDS is historically and inextricably linked with the *signature* terminology. Hence, whenever the word *signature* is used in an IPS context, people immediately think of false detections, and immediately worry about blocking legitimate traffic. For this specific reason, most IPS vendors have gravitated towards using the word *filter* rather than *signature* when describing their products. Some security professionals exacerbate the confusion by pretending there is some false positive-related stigma associated with the word *signature* when used in reference to an IPS.

As this paper will attempt to explain, whether or not a system has false positives has nothing to do with whether the system is an IDS or an IPS, or whether the system uses *signatures* or *filters*. Most false positives arise due to either (a) weaknesses in the engine or signature language that prevent the detection logic from being implemented with adequate precision, or (b) signatures being written without regard for false positives (a philosophical, or in some cases, a sloppiness problem). Therefore, for lack of better terminology, we will use the terms *signature* and *filter* interchangeably throughout this document.

Rules of IPS Filter Writing

When writing blocking filters, two rules must be obeyed:

1. **No false positives.** Do not ever block legitimate traffic under any circumstances. This is always the top priority.
2. **No false negatives.** Do not miss attacks, even when the attacker intentionally tries to evade detection. This is a high priority.

The filter writer must keep these priorities, and their relative ordering, in mind at all times. A key distinction between IDS and IPS signature-writing philosophy lies in the relative ranking of these two goals. The art of writing filters for a blocking device lies in generalizing the detection logic as much as possible to rigorously meet rule #2, without ever violating rule #1.

Deriving a Zero False Negative Filter

When performing technical vulnerability research, the engineer must first search for all of the *necessary conditions* for an attack to succeed. The researcher starts by creating a program that triggers the vulnerability remotely. The program is used to vary all the “interesting-looking” parts of the attack. Changes are made one at a time, in steps, keeping careful notes. (Strings, flags, length values, banners, version numbers, character encoding, white space... the list goes on. All are good things to vary.) If the attack succeeds even when a particular variable is set to a random value, that variable is not important for the signature. Eventually the researcher can identify the complete set of variables that *are* important to the attack’s success, and arrive at a

set of criteria that must be collectively satisfied for any attack to succeed. If there are multiple distinct attack vectors, the researcher must perform this analysis on each one separately.

Given a set of criteria that must be satisfied for an attack to succeed, it is possible to describe filter logic that has zero false negatives. That is, an attack simply cannot succeed unless the associated network traffic has exactly the characteristics that the filter is looking for.

Deriving a Zero False Positive Filter

Given a zero false negative filter as previously described, the researcher must also evaluate the accuracy of the signature in terms of false positives. At this stage, the researcher attempts to identify at least one characteristic that would *never* occur in normal traffic. If a characteristic exists that is both anomalous compared to normal traffic and critical to the attack's success, then the zero false negative signature is also a zero false positive signature.

Many attack types lend themselves well to creating these types of filters. For example:

- SQL Injection Attacks: special characters such as ' and %27 are provided in a particular value in particular Web request.
- PHP Remote File Include Attacks: a remote URL is provided in a particular value in a particular Web request.
- Buffer Overflows: too much of a certain kind of data is provided to a specific variable in a particular message exchanged in a particular type of conversation.
- Integer Overflows and Signedness Problems: an extremely large or negative number is provided for a specific value that should only ever be a (relatively) small positive number.
- Format String Attacks: format string characters are provided in a specific value in a certain message sent during a particular protocol exchange.

As the examples above imply, the bulk of the work of the detection logic often lies in gaining the contextual positioning necessary to ask a key question, such as whether a value is too large or too small, or whether a value contains special characters or strings. Whether or not an IPS device is able to take advantage of the logical perfection of a theoretical signature becomes a question of (a) whether the filtering language is flexible enough to express the necessary logic, and (b) whether the engine is powerful enough to correctly apply the various tests to network traffic, running in-line at full speed.

Different Types of Filters Explained by Example

In this section, we explore the art of IPS filter writing, and discuss four different types of filters in the context of a specific example. The four filter types are:

- Exploit-specific
- Vulnerability-based
- Policy
- Protocol anomaly

The example we will consider is the Microsoft RPC DCOM buffer overflow exploited by the Blaster and Nachi worms, and disclosed in security bulletin MS03-026. This vulnerability was a huge problem for many organizations. There was only a month between the time when the vulnerability was announced and when a worm was released. Many enterprises had tens of thousands of vulnerable machines exposed to the Internet and no way to patch them in time.

The Blaster/Nachi RPC DCOM Buffer Overflow

This vulnerability lies in Microsoft's proprietary implementation of the DCE/RPC network protocol. The mechanics of the protocol are well understood, but the different network function calls provided by the various Microsoft RPC interfaces and their corresponding arguments are not publicly documented.

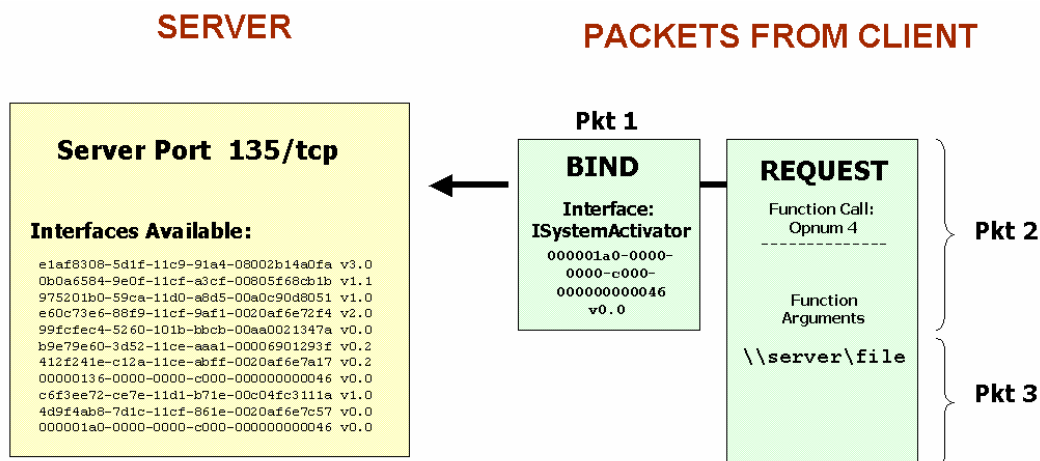


Figure 1:

This diagram illustrates how the RPC network protocol works, and illustrates the attack vector that was most widely used to exploit the MS03-026 vulnerability.

The way that the RPC protocol works is that an RPC server makes available various interfaces, where each interface is uniquely identified by a 16-byte number called the IID (See Figure 1). Each interface offers a suite of functions that are uniquely identified by another number, generally a small integer, called the opnum. Each different interface/opnum combination accepts a different set of arguments. In order for a client to issue a function call, the client first connects to the RPC server port, then sends a BIND message to attach to the desired interface; and then sends a REQUEST message to call the specific opnum against the bound interface. The REQUEST payload carries the data that is interpreted by the server as the arguments supplied to the function call.

The MS03-026 buffer overflow vulnerability is exploitable via two different RPC interface/opnum function call combinations, the first of which is illustrated in figure 1. Specifically, the flaw can be exploited via function call (opnum) 4 issued against the ISystemActivator interface (IID = 000001a0-0000-0000-c000-000000000046), and via function call (opnum) 0 issued against the IRemoteActivation interface (IID = 4d9f4ab8-7d1c-11cf-861e-0020af6e7c57). Further, both of these interfaces are reachable via multiple network ports, including 135/tcp, 139/tcp, 445/tcp, 593/tcp and 135/udp. The wide variety of attack vectors results from the fact that the underlying vulnerability is located in a shared system library. For this discussion we will only consider exploitation using a single attack vector - the opnum 4 function call issued against the ISystemActivator interface on port 135/tcp, which is the approach taken by various public exploits and worms. However, note that an IPS system must recognize *all* possible attack vectors (different interfaces, function calls, server ports) in order to provide a complete virtual software patch solution.

The opnum 4 function call issued against the ISystemActivator interface requests the server to instantiate a DCOM object from a file. This function call requires several arguments, and the

file's name is provided in one of them. The filename has the form `\\server\file`, and the vulnerability arises because the Windows system expects "server" to be a NetBIOS name, and therefore never larger than 32 bytes. An attacker can overflow a stack buffer on the target by providing a server name that is larger than the expected size, and that meets certain alignment conditions ($\text{len}(\text{server}) \bmod 16 = 12$).

In the published exploits, the `\\server\file` string becomes `\\...long_string_with_shellcode...\filename`.

Exploit-Specific Filters

An easy way to write a filter is to use a distinctive string from an exploit's shellcode as a pattern match. For example, the following hex string was found in HD Moore's exploit and the Blaster worm, which can be used as a signature to detect both of those attacks. The string contains machine instructions that are passed directly to the victim processor once the overflow is successful.

```
EB 19 5E 31 C9 81 E9 89 FF FF FF 81 36 80 BF 32 94 81 EE FC FF FF FF
E2 F2 EB 05 E8 E2 FF FF FF 03 53 06 1F 74 57 75 95 80 BF BB 92 7F 89
5A 1A CE B1 DE 7C E1 BE 32
```

The advantages of such a filter are that it (a) is easy to create quickly, (b) places a light load on most detection engines, and (c) will not false positive on non-attack traffic. The disadvantage is that the filter is specific to a particular exploit, or handful of exploits, that use the same shellcode. Hence, the filter has a terrible false negative problem. If a different piece of exploit code is used for the same vulnerability, the filter will be blind to the attack. In fact, the hex string above would *not* match on the Nachi worm. This type of signature is called **exploit-specific**.

Vulnerability Filters

Clearly, a much better signature would be one that works more like a human, who would do the following:

1. Watch the TCP session setup
2. Watch for a BIND to a vulnerable RPC interface
3. Look for a REQUEST for the appropriate function call (opnum)
4. Navigate to the vulnerable parameter in the argument list
5. Notice that an overlong server name has been provided

These events represent the "necessary conditions" that must be met for any attack to succeed. By checking for this specific sequence of events, it is possible to implement a zero false negative signature. Further, the fifth item in the list of criteria (is the server name longer than 32 bytes?) guarantees that the filter is free of false positives. This is true because a server name longer than 32 bytes will never be used in the course of normal communications. If the IPS can detect condition number five very precisely, just as outlined above, the device will have no false positives and no false negatives. A filter that implements this type of logic is called a **vulnerability filter**.

Although condition five is quite simple, it is important to note that there is a lot of work required in steps one through four to gain enough contextual positioning to apply the test effectively. Detecting the attack precisely requires applying a specific test to a particular argument supplied to a particular function call issued against a particular interface. If the filter fails to verify that the

correct interface has been bound (step two), the filter may trigger erroneously on function calls issued against irrelevant interfaces. Similarly, if the filter fails to verify that the appropriate opnum has been requested (step three), the filter may match incorrectly on irrelevant function calls. Legitimate connections sometimes carry data that when viewed without regard for the structure of the request payload, look like two Unicode-encoded backslash characters (“\\”) sitting next to each other, followed by a large chunk of data. Thus, if the filter fails to navigate to the appropriate location in the stream (step four) before applying the overlong server name test, the filter can false positive on legitimate ISystemActivator opnum 4 requests.

Experience shows that it is crucial to perform all steps (1-5) to avoid false positives. Clearly, applying this level of logical analysis at gigabit speeds requires a powerful filtering engine.

Policy Filters

Weaker signature engines are generally unable to implement signature tests with the precision necessary to avoid false positives, that is, with the precision necessary to perform blocking. An IPS or IDS that cannot precisely implement the complete detection logic will be forced to implement a simpler signature.

As an example, consider Snort’s signature for the RPC vulnerability we have been discussing. This signature is reproduced below from <http://www.snort.org/snort-db/sid.html?sid=2192>.

```
alert tcp $EXTERNAL_NET any -> $HOME_NET 135 (msg:"NETBIOS DCERPC
ISystemActivator bind attempt"; flow:to_server,established;
content:"|05|"; distance:0; within:1; content:"|0b|"; distance:1;
within:1; byte_test:1,&,1,0,relative; content:"|A0 01 00 00 00 00 00
00 C0 00 00 00 00 00 46|"; distance:29; within:16;
reference:cve,CAN-2003-0352; classtype:attempted-admin; sid:2192;
rev:1;)
```

The Snort signature looks for a TCP handshake followed by a BIND to the vulnerable RPC interface. In order to detect the BIND, the signature performs a simple string match for the interface’s IID at a particular offset in the TCP packet payload. Snort’s description of the signature is as follows: “This event is generated when an attempt is made to exploit a known vulnerability in Microsoft RPC DCOM. Impact: Execution of arbitrary code. Classtype: attempted-admin. False positives: None known. False negatives: None known.” As will be explained, there is a problem with coupling this description to Snort’s RPC signature above.

Referring back to Figure 1 and its associated description, notice that an RPC client must always send a BIND message in order to initiate communications with a particular RPC interface. Thus, an ISystemActivator BIND request is always sent during any legitimate transactions with the DCOM interface, as well as during any attacks. Detecting the BIND alone (rather than using the five steps described earlier) works fine to detect attacks in networks that are not using RPC DCOM. However, such a signature will false positive like mad in an environment that is actually using the protocol in the course of normal business communication - ironically, the very type of environment that most needs protection.

Although Snort’s description indicates that the signature will detect exploitation of the MS03-026 (CAN-2003-0352) buffer overflow vulnerability with no false positives and no false negatives, this is no vulnerability filter. If the Snort signature is set to block, *all* RPC communications involving the vulnerable interface will be disabled. Shutting down the interface entirely is a much broader action than precisely blocking exploitation of the vulnerability. Before the Snort

signature can be set to block, the administrator must make a “policy decision”. Specifically, the administrator must decide whether it makes sense to disable all DCOM communications throughout the network in order to provide security protection for a specific vulnerability. Filters that operate in this manner are most accurately called **policy filters**. When a policy filter is used to do the job of a vulnerability filter, false positives necessarily result.

Finally, it is worthwhile to mention that the Snort signature above also has a false negative problem. It turns out that an RPC BIND message is designed to carry not one, but a list of interfaces that the client wishes to bind on the server. By specifying the exact byte position in the stream where the 16-byte interface IID is looked for (notice the `distance:29, within:16` keywords), the Snort signature implicitly forces the IID of interest to be the first one in the list. Thus, an attacker can easily evade this signature by simply providing the IID of the vulnerable interface further down in the list. The most likely reason for this particular error is that the signature-writer was not diligent enough in discovering the methods by which the filter could be evaded. Through this false negative example, we see how the skill and thoroughness of the vulnerability researcher also plays a critical part.

Protocol Anomaly Filters

Some vendors emphasize enforcing RFC protocol conformance as a way to protect systems from zero-day attacks. The associated filter type, which detects out-of-spec network traffic, is called a **protocol anomaly filter**. RFC enforcement can be quite useful for some protocols, but the technique is unlikely to provide protection against attacks such as the Blaster worm. It is important to understand the circumstances where protocol anomaly detection makes sense, and where it does not. Let's start by considering a case where protocol anomaly filtering works well: the Simple Mail Transfer Protocol or SMTP.

Every single SMTP command defined in RFC 821 that accepts arguments and is used in the course of a MAIL transaction has been the subject of a buffer overflow vulnerability in at least one SMTP server product. These commands are MAIL, RCPT, HELO, VRFY, EXPN and HELP; and the vulnerability can be exploited by providing an overlong argument to the command. An IPS can “enforce protocol conformance” for SMTP by looking for inappropriate values to be provided as arguments to any of these well-defined commands; thereby providing protection against all the old buffer overflow attacks, as well as any new ones that may emerge in the future (i.e., zero-day protection).

Similarly, an IPS can block a number of HTTP-based attacks by checking for inappropriate values in well-defined HTTP header entities, such as the Content-Type: and Host: fields. For example, both Microsoft IIS [3] and Apple QuickTime Player [4] have had vulnerabilities in handling overlong *Content-Type:* values; while Lotus Notes/Domino [5], Macromedia JRUN [6] and Microsoft IIS [7] have all had problems handling overlong *Host:* header values. Protocol anomaly filters which block all HTTP requests/responses containing overlong *Content-Type:* and *Host:* values provide protection for these known vulnerabilities as well as protection against new attacks.

In each of the examples considered so far, the anomaly filters detect conditions that are both necessary to the attack's success and guaranteed never to occur in normal traffic. Thus, the filters can detect multiple attacks without false negatives and without false positives. Under these ideal conditions, a protocol anomaly filter is an effective generalization of a vulnerability filter. However, the usefulness of an anomaly filter is greatly diminished if the detected condition ever occurs in legitimate traffic. Legitimate applications that produce “out-of-spec” traffic can

render anomaly filters useless by causing large numbers of false positives. Further, some protocol specifications are only loosely enforced (meaning many products produce out-of-spec traffic), or do not exist in the public domain at all (meaning it is nearly impossible to determine how traffic on the wire compares to what “good” traffic should look like). In the following, we explore a few of these more difficult cases.

HTTP URLs are protocol entities that often provide the vehicle for web-based attacks, but are difficult to filter using an anomaly-based approach. An attacker can easily provide a malicious value in a web request to a CGI program that will compromise a web server, yet never violate any HTTP protocol specification. For example, even the famous “phf” attack does not violate the Uniform Resource Locator (URL) RFC 1738. A system trying to block the attack via anomaly detection would have to specifically know how the phf script interprets data provided to the Qalias variable. (A phf attack example is given below. Note that the vulnerability allows for execution of arbitrary shell commands, not just “/bin/cat /etc/passwd” as used in the example.)

<http://<target>/cgi-bin/phf?Qalias=x%0a/bin/cat%20/etc/passwd>

To operate at this level, a true protocol enforcement system would have to understand how all Web-based applications running on the Internet handle their inputs. Such understanding would require an individual source code review or black box fuzzing test of each version of each Web application. Further, attempts to enforce RFC 1738 across the board would lead to blocking normal traffic. For example, the following URL violates the specification by including more than one literal question mark, but is a real world URL in use today:

<http://www.glic.com/frameset.html?http://www.geoaccess.com/directoriesonline/theguardiantent al/chooseproduct.asp?prodpasscode=p>

As another example, consider the following Web request. This string is valid and will perform a Google search:

<http://www.google.com/search?q=%27+having+1%3D1-->

While this similar request carries an SQL injection attack [8]:

<http://<target>/productcart/pc/Custvb.asp?Email=%27+having+1%3D1-->

Notice the suspicious-looking arguments that follow the equals sign in both cases. Both URLs are valid according to the RFC. How can the IPS distinguish an attack from a benign request without some knowledge of how the receiving application handles user-supplied data? Indeed, the wide variability in URLs leads to a significant amount of “tuning” on the part of the administrator to control false positives when an anomaly-based technique is used.

Protocol anomaly detection is also less useful with respect to network protocols such as Microsoft DCE/RPC/DCOM. The source code of the implementation is not available, and the network protocol has no RFC, nor is it publicly documented. A limited number of anomaly signatures can be written to look for unusual values in the standard DCE/RPC headers, but even attacks against the MS03-026 vulnerability do not violate any aspect of the protocol at that level. The buffer overflow attack can only be identified as a *protocol anomaly* (or more accurately, an *application anomaly*) if the IPS knows precisely how all the different arguments supplied to all the different interfaces/function calls are interpreted by the receiver. Under such circumstances, a vulnerability-based filtering approach is much more practical and precise.

Filter Summary

Through illustration of the well-known RPC DCOM system vulnerability MS03-026, we have attempted to shed light on some of the more subtle aspects of filter development.

First, we see that the skill and diligence of the vulnerability researcher is critically important. The researcher must understand a vulnerability quite well in order to predict attack vectors, and in order to harden filters against false positives and false negatives. Second, we see that protocol anomaly detection is often very useful, but certainly no silver bullet. Third and perhaps most importantly, we see why both false positives and false negatives will plague IPS/IDS products that are not capable of implementing precise detection logic.

False positives commonly arise from policy-type signatures, such as the Snort example, while false negatives occur with exploit-specific signatures, such as the Blaster shellcode example. Because exploit-specific and policy signatures are the easiest to implement, they are frequently used by systems with weak engines. It takes a powerful engine to support effective vulnerability filters that can operate in high-speed environments that are actively running the protocols at risk.

TippingPoint's Virtual Software Patch

TippingPoint implements several vulnerability-based filters to provide protection against exploitation of the MS03-026 buffer overflow. Multiple filters are required to cover the many various attack vectors (different ports, interfaces, protocols). Over time, attackers have produced various exploits and even released multiple worms, and all have been blocked with the same set of vulnerability filters. Note that many customers with TippingPoint's IPS in their network actively use the Microsoft RPC DCOM protocol. All organizations were able to use the same filters to block attacks without false positives and without tuning.

A quote from Doug Brown, Manager of Security Resources for the University of North Carolina:
"TippingPoint allowed us to run a demo of their IPS appliance and it was in place when the Blaster/Nachi worms hit. The part of the network protected by the unit suffered no infections. We had been planning on returning the unit due to cost constraints, but after comparing the night and day difference between the protected and unprotected parts of our network, the clear results led us to find the funding for several units. The TippingPoint units are really worth the money."

The TippingPoint Intrusion Prevention System provides protection against exploitation of hundreds of other vulnerabilities in a similar manner.

TCP Reassembly and Attack Detection

In order to understand how a virtual software patch looks to the would-be intruder, it is important to understand how TippingPoint's TCP stream reassembly works. This concept can be explained in the context of the same MS03-026 buffer overflow example previously discussed.

Most published exploits for the MS03-026 buffer overflow use three data-carrying TCP packets to complete the attack (please refer back to figure 1). The first data packet, that is, the first packet sent by the client following the three-way handshake, is the BIND message. The second and third data packets carry the REQUEST message. The second packet carries the REQUEST header, which specifies the opnum, and a large portion of the argument data. All of the argument data does not fit into the second packet, so the third packet carries the rest of the

data. In most exploits, the beginning of the long server name is found at the end of the second packet, and the rest of the server name is found in the third packet. Thus, the target needs to receive all three packets for an attack to complete successfully. Similarly, TippingPoint needs to see all three packets to recognize an attack.

Typically, the packets arrive in order. That is, first the BIND arrives. The IPS matches the packet data against the filter and fails to get a hit. The BIND is sent on its way, but state data extracted from the packet is stored to memory. Next, the REQUEST “part one” packet arrives. This data is combined with the saved data and the result is matched against the filter. At this point, there is still no filter match. The stored state data is updated and the packet is sent on its way.

Finally, the REQUEST “part two” packet arrives. When this data is combined with the saved data, it triggers a match on the vulnerability filter. The filter match causes the packet to be dropped, and the corresponding TCP flow is marked as “blocked” in the engine. From that point forward the engine hardware automatically drops any packets received in either direction of that connection. The attack never completes on the victim and the intrusion is prevented.

Out of Order Packets

If packets arrive out of order, TippingPoint’s TCP reassembly engine takes care of the details. The first packet that is blocked is always the one that provides the final piece of the detection puzzle. For example, in attacks against the RPC DCOM vulnerability, often the two pieces of the REQUEST arrive out of order, so that “part two” arrives before “part one”. This out of order data does not present a problem for the TippingPoint IPS. The system simply stores state data based on “part two” and passes the packet on, and then blocks “part one” when it arrives and all subsequent packets.

Evasion Resistance

If the data had been sent in a stream of randomly ordered small packets, this would not have impacted the attack detection. The TCP stream reassembly engine handles re-ordering arbitrary TCP segments according to sequence number. Similarly, if any of the packets had been fragmented at the IP layer, the IP fragment reassembly engine puts the fragments back together according to offset before passing the data through the signature engine. In either case, the in-line IPS has an advantage in that it can dictate what the target system ultimately receives by controlling what packets are put onto the wire for transmission. This means any ambiguities in fragmented/segmented data can be resolved by the IPS.

In the NSS testing, TippingPoint was subjected to attacks using a variety of network-, transport- and application-layer evasive techniques. The tests included attacks obfuscated with tools such as Fragroute, Whisker, and ADMutate. Further, several exploits were modified to use custom evasion techniques. The TippingPoint system received a perfect score out of the box on every one of the NSS false negative and evasion tests. In fact, TippingPoint was the *only* product to achieve this score. Evasion / false negative methods tested by NSS include:

- **Packet Fragmentation and Segmentation Evasions:** Includes fragmentation with various packet sizes at both the IP and TCP layers, out of order data, duplicate data, bad checksums. (Fragroute)
- **Evasions for HTTP Attacks:** URL encoding, /./ Directory insertion, Premature URL ending, Long URL, Fake parameter, TAB separation, Case sensitivity, Windows \ delimiter, Session splicing. (Whisker)

- **Miscellaneous Evasions:** Altering default ports, Inserting spaces in FTP command lines, Inserting non-text telnet opcodes in FTP data stream, Altering RPC protocol and procedure numbers, RPC record fragging, Polymorphic shellcode mutation (ADMutate), custom exploit modification.

Attack Recognition Under Load

The NSS test also stressed an IPS' ability to correctly recognize and block attacks under a heavy load of background traffic consisting of HTTP connections, UDP packets of various sizes, mixed protocol simulated traffic, and real world traffic. In order to achieve a perfect score, an IPS device must detect and block all attacks, must not drop any packets due to congestion, must not block any legitimate traffic, and must support a level of connections per second commensurate with the maximum rated throughput of the device (e.g. 1 million connections per second in the case of 1 Gbps throughput). The TippingPoint device was the *only* IPS product participating in the NSS test to achieve a perfect score under all load conditions while running at a full 1 Gbps. A quote from Bob Walder, president of The NSS Group, is below.

"TippingPoint was tested up to 1 Gbps, and performance at all levels of our load tests was impeccable, with 100% of all attacks being detected and blocked under all load conditions. We would have no hesitation in rating the TippingPoint IPS as a true 1 Gbps device."

Digital Vaccine Service

In order to provide such high accuracy, high performance and high evasion resistance, all TippingPoint filters are designed and developed in-house at TippingPoint, and are subjected to a rigorous QA test cycle before being released to the public. Customers are updated with new filters to protect against exploitation of the most critical vulnerabilities on an ongoing basis, and filters are automatically installed and set to block according to customer policy when a new Digital Vaccine is delivered.

Configuration and Tuning

With TippingPoint, the administrator does not need to be concerned with configuring the device to understand which interfaces are trusted and untrusted, internal or external. The administrator does not need to limit the number of filters applied to each segment, or configure the filters so that they are only applied to certain types of traffic flows. The administrator does not need to provide the device with the results of a vulnerability scanner.

Vendors who are emphasizing these aspects of configuration, tuning and correlation in IPS offerings are often trying to cover up either a performance problem or a false positive problem. A false positive problem can be minimized by applying fewer signatures to a smaller amount of traffic. Similarly, a performance problem can be eased by applying fewer signatures to less traffic, translating to a lighter computational load. However, if the IPS has no problems with performance or false positives, why not turn on the full filter suite across the board and let the IPS figure things out? Such a scheme places much less burden on the administrator and allows less room for human error. The administrator does not need to be concerned with reconfiguring the IPS system each time the network starts using a new protocol.

Also, note that if a traffic stream can be identified as malicious with certainty, then it becomes irrelevant whether or not a vulnerability scanner reports that the target would have been vulnerable to the attack. The malicious connection should be dropped regardless.

Conclusion

A high-performance intrusion prevention system can function as a virtual software patch, protecting vulnerable computers from compromise in networks where host-by-host patches have not been applied. The virtual patching capability arises directly from the ability to identify and **block** malicious traffic in transit, *before* attacks reach their intended targets. The enabling technology behind virtual patching lies in high precision vulnerability filters. These filters are professionally designed to provide optimum attack space coverage and maximum resiliency to evasions.

In order for an IPS to provide a practical virtual software patch solution, the device must perform well on multiple fronts simultaneously. Specifically, the IPS must implement high precision vulnerability filters, handle a heavily-loaded gigabit network with a full filter set enabled (no dropped packets), and perform a useful virtual patching function without requiring the administrator to perform tedious tuning and configuration tasks. To date, TippingPoint is the **only** IPS shown to be capable of meeting these fundamental requirements [1,2].

References

- [1] NSS Group IPS Test Results, published January 2004
http://www.nss.co.uk/acatalog/Intrusion_Prevention_Systems_IPS_.html
- [2] TippingPoint Press Release, January 2004
http://www.tippingpoint.com/news_events/pdf/NSSGold_011904.pdf
- [3] Content-Type overflow in Microsoft IIS
<http://www.securityfocus.com/bid/6214>
<http://www.foundstone.com/knowledge/randd-advisories-display.html?id=337>
- [4] Content-Type overflow in Apple QuickTime Player
<http://www.securityfocus.com/bid/4064>
<http://www.securiteam.com/windowsntfocus/5HP022K6AE.html>
- [5] Host field overflow in Lotus Notes and Domino
<http://www.securityfocus.com/bid/6870/credit/>
<http://archives.neohapsis.com/archives/bugtraq/2003-02/0189.html>
- [6] Host field overflow in Macromedia JRUN
<http://www.securityfocus.com/bid/4873>
<http://www.securiteam.com/windowsntfocus/5XP0M2K75I.html>
- [7] Host field overflow in Microsoft IIS
<http://www.securityfocus.com/bid/2674>
<http://www.eeye.com/html/Research/Advisories/AD20010501.html>
- [8] Bugtraq posting: SQL injection attack
<http://archives.neohapsis.com/archives/bugtraq/2003-07/0057.html>