



WHITE PAPER

SOURCEFIRE VULNERABILITY RESEARCH TEAM (VRT)



KNOW MORE NETWORK RISKS
NO MORE GUESSING

THE CURRENT IPS LANDSCAPE

Intrusion prevention system (IPS) vendors often promote how many threats they detect and how quickly they release detection capabilities for new threats. Many organizations blindly assume that these claims are accurate, but without evidence to substantiate them, this faith is misplaced.

Unverifiable Protection: If you had a headache, would you purchase a “headache elixir” sold from a roadside stand? Or would you buy Tylenol, Advil, or another FDA-approved headache medication at the drugstore? Most IPS vendors make tenuous protection claims that are untested and unverifiable.

Partial Protection: Would you purchase a car alarm that stopped thieves from breaking into your driver’s side window, but didn’t protect the passenger’s side? Most IPS vendors similarly claim “protection” against vulnerabilities when they only cover a single specific avenue of attack.

Protection After the Threat: If your child had a good chance of catching a dangerous but preventable disease, would you wait for your child’s friends to get the disease before taking action? Or would you vaccinate the child now, before the threat occurred? Most IPS vendors provide protection against exploits only after these attacks are already publicly known and organizations have already been attacked by them.

Unreliable Protection: If the fire alarm system in your building sounded every few days, alerting the fire department and triggering the sprinkler system even though there was no fire, would you keep it in place? What if the system missed a real fire? Most IPS vendors use signatures that do not adequately cover the triggering conditions of a vulnerability – and therefore produce false positives and false negatives.

As attackers become more advanced, IPS vendors need to provide protection that **verifiably** defends against:

- all possible attacks
- before particular methods of attack are known
- without creating false positives or false negatives

Enter the Sourcefire Vulnerability Research Team (VRT), a dedicated research group chartered to deliver this level of protection. The VRT provides the rulesets that power the Sourcefire 3D® System, employing a vulnerability-based methodology to meet the demands necessary for the most advanced IPS deployments.

VERIFIABLE PROTECTION: THE SOURCEFIRE VRT AND THE SNORT® RULES LANGUAGE

Many IPS vendors boast about how quick they are to respond to the release of vulnerabilities - for instance, after the monthly “Microsoft Tuesday,” when Microsoft announces many vulnerabilities and releases patches

to end users. Because most IPS vendors’ signatures are closed, it is impossible for customers to verify what they are actually being protected against. In many cases, IPS vendors claim protection for vulnerabilities that they do not in fact cover at all, or any use (legitimate or illegitimate) of the functionality triggers an alert.

The Sourcefire VRT writes the standard source of Snort rules used by the Sourcefire 3D System. Snort rules are open for anyone to inspect, and can be verified to address the vulnerabilities for which coverage is claimed. The Snort rules format is the industry standard, used by security professionals worldwide.

Snort’s open rules format gives customers the ability to:

- Verify that a rule is providing complete protection against a vulnerability
- Create new rules or modify existing ones to detect issues with custom or unusual services
- Leverage widely available user-contributed rules from a community of hundreds of thousands of Snort users

In order to achieve “open rules,” several closed-signature IPS vendors have actually incorporated Snort into their products! But this is never the default inspection method and is always poorly integrated.

WHY SIGNATURES AND EXPLOIT-BASED DETECTION OFFER LITTLE VALUE

Hackers, computer enthusiasts, and security professionals release hundreds of **exploits** – attacks against security vulnerabilities – every week. These exploits are quickly modified by the community. Some are rewritten to be incorporated into attack tools like Metasploit, CANVAS or CORE IMPACT, and others are modified to use different shellcode or to add additional attack vectors. The result is that numerous exploit variants are being created for each vulnerability on a continual basis.

Many IPS vendors combat this moving target by writing exploit-specific **signatures** for new exploit variants as they emerge. Signatures rely on distinctive marks or characteristics that are present in a particular exploit. Every time a new exploit variant is released, these vendors must find it, write a signature for it, test the signature, and then release the signature to their customers. This approach has the following problems:

1. **Exploit-based protection is partial protection.** For any vulnerability, there is a near-infinite number of potential exploits. At best, an exploit-based system can protect against only a small subset of these exploits. Hackers can modify existing exploits in ways that still successfully exploit a vulnerability but that can go undetected by a signature-based IPS.

2. **Exploit-based detection is always behind the current threat.**
Until an exploit is available to an exploit-based IPS vendor, the vendor cannot write detection for it. This exploit may be wreaking havoc on corporate networks while it is still in the process of being identified. Until the exploit is identified and a signature is written and tested, no protection is available from an exploit-based IPS vendor.
3. **Signatures are unreliable, producing false positives and false negatives.**
Because signatures do not model protocols and identify exact triggering conditions, they are much more susceptible to false positives and false negatives. Signature strings are sometimes present in legitimate traffic and are often missing in malicious traffic, sharply reducing confidence in the IPS events that are generated.
4. **Exploit-based detection requires constant updates.**
Every time an exploit-based IPS vendor releases a new signature, customers must download it. Missing a daily download can mean being completely unprotected against current, propagating threats. The signature set requires significant maintenance, quickly becomes unwieldy, and limits performance.

WHY RULES AND VULNERABILITY-BASED PROTECTION PROVIDE ACTUAL VALUE

1. **Rules offer protection against any possible exploitation of a vulnerability.**
If a protocol model is defined properly, and the vulnerability's triggering conditions are programmed correctly, then the corresponding rule will protect against any possible exploit of that vulnerability. A single rule in a vulnerability-based IPS can cover hundreds of known exploits – plus an infinite number of potential ones that might be attempted. An exploit-based IPS needs hundreds of signatures and constant updates to achieve only part of this protection.
2. **Rules protect customers before exploits are released.**
Customers that deploy rule-based protection are shielding their systems on the network in advance of exploits occurring. With vulnerability-based protection, there is no “window of exposure” between the announcement of a particular exploit and an update to the IPS. When new exploits emerge for an existing vulnerability, systems are already protected.
3. **Rules trigger reliably, without false positives and false negatives.**
Modeling the protocol ensures that only relevant fields and communication states are monitored for attacks. Identifying the necessary set of triggering conditions for a vulnerability ensures that only malicious traffic is flagged.
4. **Vulnerability-based protection allows for a manageable number of updates and rules.**
One vulnerability-based rule can cover hundreds of known (and unknown) exploits. Far fewer updates are needed, and far fewer rules cover a much larger number of exploits.

The Sourcefire 3D System and the Sourcefire VRT use fully vulnerability-based rules, for protection ahead of emerging threats.

THE SOURCEFIRE VRT RULE METHODOLOGY

The Sourcefire VRT rule creation methodology has four main components. They are:

- Researching the vulnerability
- Modeling the protocol
- Identifying the triggering conditions
- Testing and verifying the assumptions

The result is a set of rules that is:

- Able to detect all potential exploits of a vulnerability – before a threat emerges
- Optimized for maximum performance
- Free of false negatives and as free as possible of false positives

RESEARCHING THE VULNERABILITY

Sourcefire makes a major investment in vulnerability research. Sourcefire's VRT employs some of the most renowned security professionals in the industry, including the authors of several standard security reference books.

More than any other organization, the Sourcefire VRT focuses on identifying all of the possible **triggering conditions** for vulnerabilities and building up **protocol models** for protocols that are potentially subject to attack. Doing so requires substantial security programming expertise, a thorough understanding of networking protocols and potential attack paths, and the ability to reverse-engineer both uncommon protocols and closed-source applications.

As the creators of Snort, Sourcefire also has the unique advantage of the Snort community. With more than 3,000,000 downloads and over 225,000 active users, Snort is the most widely used intrusion prevention product. Its user community provides early warnings of new threats and vulnerabilities, contributes to expanded “community” rulesets, and provides immediate feedback as an additional guard against false positives and a layer of quality control on new content. The Snort community also helps to provide access to applications and systems that are not present in-house.

The Sourcefire VRT also leverages a variety of paid research and public and private security feeds for information on newly-released vulnerabilities.

MODELING THE PROTOCOL

Several years ago, the only way to buy goods at a retail store was at a checkout counter from a human being – the cashier. The cashier would scan each product's bar code – and then double-check a screen.

Did that \$200 PlayStation ring up as a \$1.25 Diet Coke? Since a bar-code scanner didn't know anything about the product the bar code was attached to, the store could make costly errors or miss malicious behavior if the cashier didn't double-check the transaction. It wouldn't make sense to entrust the decision to the scanner and leave the cashier out of the picture.

Now, unmanned automatic checkout systems are available that both weigh a product and scan its bar code. Most items weigh a specific amount and have a specific bar code. In the future, checkout systems may be able to visually recognize the forms of products being scanned. As the protocol model gets better and more information can be determined about a product, more automatic decision-making capability can be entrusted to the machine. Protocol modeling is like defining the visual form, weight, and bar code of a product so it can be recognized later: identifying exactly how a vulnerable protocol looks and functions so vulnerable areas can be pinpointed.

The information in a protocol model can be divided into three components:

- Identifiers for the protocol and protocol version
- Identifiers for the state of communication
- The structure of the packet(s) and message(s)

The protocol model, once built, is used by the Sourcefire VRT:

- to limit detection to the protocol that is vulnerable
- to limit detection to the communications state in which the vulnerability could be exploited
- to limit detection to the field(s) of a packet and field(s) of a message that are potentially vulnerable

The Snort rules language allows modeling any protocol within a rule. If a vulnerability emerges on a new protocol that was previously thought secure, the VRT can quickly generate protection for it. Other vendors claim many pre-built protocol decoders, but do not provide the capability to model protocols at the signature level, meaning that detection for new protocols comes slowly.

PROTOCOL IDENTIFIERS

SSL is one of the most common protocols used on the Internet today, as it is used to encrypt web-based communications. SSL traffic typically occurs on port 443 (the default port for HTTPS) – but different variants of SSL and other encryption protocols can communicate on this port. SSL v2, SSL v3, TLS, and PCT are all fairly common possibilities. If a vulnerability can only be exploited in SSL v2, then looking for exploits in SSL v3, TLS, or PCT traffic will potentially result in false positives and hinder performance.

Therefore, part of protocol modeling is identifying the particular protocol and protocol version, so that alternatives can be eliminated from inspection.

COMMUNICATION STATES

SSL communication sessions establish themselves as follows:

1. A TCP 3-way handshake occurs.
2. The client system sends a `Client-HELLO` message.
3. The server system responds with a `Server-HELLO` message.
4. Both sides exchange key information to determine what type of encryption to use.
5. Both sides begin encrypted communication.

A vulnerability is likely to be relevant to only one of these steps, or communications "states." To avoid false positives and limit performance impact, only the relevant state(s) should be inspected for an exploitation of a particular vulnerability.

So if there is a vulnerability in the `Client-HELLO` message on certain servers, inspection for exploits of this vulnerability should only be performed when the connection is in the "Client-HELLO" state. During the `Server-HELLO`, key exchange, or encrypted communication, inspection should not be performed for this vulnerability.

PACKET STRUCTURE AND FIELDS

Assume that the `Client-HELLO` message (State 2) is vulnerable.

The `Client-HELLO` message in SSL v2 contains the following information:

```
char MSG-CLIENT-HELLO
char CLIENT-VERSION-MSB
char CLIENT-VERSION-LSB
char CIPHER-SPECS-LENGTH-MSB
char CIPHER-SPECS-LENGTH-LSB
char SESSION-ID-LENGTH-MSB
char SESSION-ID-LENGTH-LSB
char CHALLENGE-LENGTH-MSB
char CHALLENGE-LENGTH-LSB
char CIPHER-SPECS-DATA [ (MSB<<8) | LSB ]
char SESSION-ID-DATA [ (MSB<<8) | LSB ]
char CHALLENGE-DATA [ (MSB<<8) | LSB ]
```

Perhaps only one of these fields is potentially vulnerable on a server – let's say hypothetically that a program parses the `CIPHER-SPECS-DATA` field incorrectly. For the purposes of detecting exploits of this vulnerability, all the other fields should be ignored.

MODELING THE PROTOCOL: SUMMARY

By limiting detection to the specific field(s) and state(s) of communication that are potentially vulnerable, good protocol modeling vastly reduces the chances of false positives and increases performance.

IDENTIFYING THE TRIGGERING CONDITIONS

Triggering conditions are the set of conditions that must be met in order for an attacker to take advantage of a vulnerability. At its most simplistic, a triggering condition compares the length or contents of a relevant field (identified in the protocol model) to a specific value or set of values. If the comparison “succeeds,” then the triggering condition is successful. Many different categories of triggering conditions exist – for example:

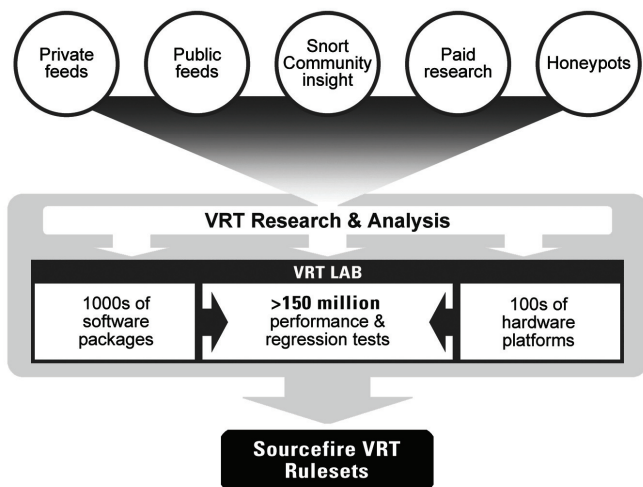
- Range and type errors - buffer overflows, off-by-one errors, etc. These are common, simple programming errors that account for a large number of vulnerabilities today.
- Synchronization and time errors - vulnerabilities that occur when events happen outside of an expected sequence. Many DoS attacks fall into this category.
- Protocol errors - static content, static passwords, etc. A default password that is shipped with a product and commonly unchanged may fall into this category.

If you build a house, building steps must follow a specific order: a foundation must be laid before framing goes up; framing must be up before plumbing and wiring can go in; plumbing and wiring must be in place before the drywall can go up, and so forth.

Much like building a house, if there are multiple triggering conditions, they must also be checked in a **specific order**. In order to check a triggering condition, an IPS must read a field in a packet. Often, because fields are variable-sized, only by reading that field will the IPS know where to begin reading the next field.

TESTING AND VERIFYING THE ASSUMPTIONS

Once a set of triggering conditions have been established, the rules undergo a period of testing to ensure that real-world execution of the rules does not introduce any performance issues, false positives, or false negatives.



The Sourcefire VRT runs an automated test suite of over 16,000,000 checks against the ruleset. Automated test cases are written and added to the test suite as new vulnerabilities are found and tested. These test cases verify that the rule triggers if and only if the set of necessary triggering conditions is present in network traffic.

Extensive tests are also run against real-world traffic, with a variety of application and OS variants. Sourcefire leverages both security partners and the hundreds of thousands of users in the Snort community for further verifications to help ensure that Sourcefire VRT rules offer reliable and accurate protection. Sourcefire uses high-end network performance testing equipment to verify that rule changes do not introduce performance problems.

SOURCEFIRE VRT RULE METHODOLOGY: PUTTING IT ALL TOGETHER – A SIMPLE EXAMPLE

In 2001, a vulnerability was reported in RPC communications to the ToolTalk database server, in operating systems from HP, IBM, SCO Group, SGI, and Sun Microsystems (Bugtraq ID 122, CVE-1999-0003). The vulnerability allowed a remote attacker to run malicious code with superuser privileges.

Exploit code was made available, and several IPS vendors created signatures that detected the use of this specific exploit code – for instance, the string “\x80\x1c\x40\x11”, which was present in a particular exploit.

To address this vulnerability, the Sourcefire VRT followed its established methodology. It leveraged its model of RPC, identified the vulnerable field and communication state, identified the triggering condition of the vulnerability, created a rule, and verified the assumptions behind that rule. The process is outlined below.

PROTOCOL MODEL

RPC packets are structured as follows:

0x0000 :	A	B	C	D	
	00 00 0F 9C	36 51 D5 2B	00 00 00 00	00 00 00 02	
0x0010 :	E	F	G	H	
	00 01 86 F3	00 00 00 01	00 00 00 07	00 00 00 01	
0x0020 :	I	J			
	00 00 00 20	37 5E D1 6A	00 00 00 09	6C 6F 63 61	
0x0030 :		J continued			
	6C 68 6F 73	74 00 00 00	00 00 00 00	00 00 00 00	
0x0040 :	J continued	K	L	M	
	00 00 00 00	00 00 00 00	00 00 00 00	00 00 0F FF	
0x0050 :		N			
	41 41 41 41	41 41 41 41	41 41 41 41	41 41 41 41	
0x0060 :		N continued			
	41 41 41 41	41 41 41 41	41 41 41 41	41 41 41 41	

A: Fragmentation data
B: RPC ID number
C: RPC call
D: RPC version
E: RPC program number
F: RPC response
G: RPC procedure number
H: Authorization type
I: Authorization byte size
J: Authorization information
K: Verification type
L: Verification byte size
M: Number of data bytes
N: Data

PROTOCOL IDENTIFICATION

- Bytes 8 through 11 indicate an RPC call, so we have a content match for "00 00 00 00" for those bytes. We can move straight to that location and look for that content match in those four bytes like this:

```
-- content:"|00 00 00 00|"; offset:8; depth:4;
```

- Bytes 12 through 15 indicate that we are seeing RPC version 2

```
-- content:"|00 00 00 02|"; offset:12; depth:4;
```

- Bytes 16 through 19 indicate the RPC program number. The one we are concerned with in this case is "00 01 86 F3", so we can jump to that position and look for that particular program number to occur, like this:

```
-- content:"|00 01 86 F3|"; offset:16; depth:4;
```

- We know that four more bytes further into the packet from this point is a four byte field that indicates the RPC procedure number. The one we are concerned with is "00 00 00 07", so we move forward four bytes and look for that procedure number like this:

```
-- content:"|00 00 00 07|"; distance:4; within:4;
```

STATE OF COMMUNICATION

- We are only concerned about sessions that have already been established. We verify this as follows:

```
-- flow:to_server,established;
```

RELEVANT FIELDS

- First, we need to look at how much data is being sent as authorization data and then jump past this data. This length is stored at a position four bytes further into the packet and is a four-byte field, so we do that like this:

```
-- byte_jump:4,4,relative,align;
```

This means we can look at the length of data being sent and move forward that length to look for our next important piece of data.

- The next piece of data we are concerned with is the amount of data being sent as verification. We know this is a further four bytes into the packet data and that the length is contained in a four-byte field, so we can do that like this:

```
-- byte_jump:4,4,relative,align;
```

This means we can look at the length of data being sent and once again move forward to go past that data.

TRIGGERING CONDITIONS

- Now we are at a point in the packet data where we can look at how much data is being sent to the RPC program. We know that the information is contained in a four byte field immediately following the verification data, and since we just jumped past that data, we can now read in the field value and compare it against a value that might indicate that too much data is being sent. We can do that like this:

```
-- byte_test:4,>,128,0,relative;
```

This means we are reading in the four bytes that indicate how much data is being sent and we are comparing it to a value of 128 decimal to see if it is larger than that value. If all these conditions are met we have a successful match and a possible overflow attempt on our hands.

The rule itself when complete looks like this:

```
alert tcp $EXTERNAL_NET any -> $HOME_NET any \  
(msg:"RPC tooltalk TCP overflow attempt"; \  
flow:to_server,established; \  
content:"|00 00 00 02|"; depth:4; offset:12; \  
content:"|00 01 86 F3|"; depth:4; offset:16; \  
content:"|00 00 00 07|"; within:4; distance:4; \  
byte_jump:4,4,relative,align; \  
byte_jump:4,4,relative,align; \  
byte_test:4,>,128,0,relative; \  
content:"|00 00 00 00|"; depth:4; offset:8; \  
reference:bugtraq,122; \  
reference:cve,1999-0003; \  
classtype:misc-attack; sid:1965; rev:8;)
```

This rule will now generate events for *any* attempts to exploit the ToolTalk RPC vulnerability, as opposed to just the published exploit code.

IMPACT AND CONTEXT: SOURCEFIRE RNA® (REAL-TIME NETWORK AWARENESS)

Sourcefire RNA provides security analysts an additional level of context that is not present in any competing IPS. Sourcefire RNA provides the following benefits:

1. **Endpoint intelligence.** Sourcefire RNA Sensors discover the assets and traffic patterns on networks in real time. They listen on networks to identify new and existing hosts, fingerprint operating systems and services, and discover potential vulnerabilities. RNA Sensors use pervasive, passive listening, unlike competing IPS solutions that rely on infrequently updated, bandwidth-leeching active scans. RNA Sensors can perform targeted active scanning to obtain more information about specific hosts.
2. **Network intelligence.** Sourcefire RNA Sensors can detect abnormal traffic patterns, DDoS attacks, and internal worm propagation using Network Behavior Analysis (NBA) functionality. Sourcefire RNA profiles network traffic and watches for deviations from that baseline using statistical thresholds.

Sourcefire Defense Center combines the information from multiple Sourcefire 3D Sensors running Sourcefire IPS and RNA to determine the most critical security events on networks. Using Sourcefire RNA information, Sourcefire Defense Center prioritizes attacks on potentially vulnerable machines and deprioritize others, allowing immediate response to high-criticality events and sharply reducing or even removing the need to tune sensors.

| Vulnerability-based Protection Ahead of the Threat - Real World Example: Conficker

The Conficker worm first emerged in November 2008, nearly one month after Microsoft issued an out-of-band Security Bulletin and corresponding patch. By January 2009, Conficker had infected an estimated 10 million Microsoft Windows hosts resulting in one of the largest botnets in history. Fortunately, Sourcefire 3D customers and open source Snort users were afforded zero-day protection more than two years in advance.

- **August 7, 2006:** Microsoft issues Security Bulletin MS06-040 corresponding to a remote code execution vulnerability discovered in the Microsoft Windows Server Service.
- **August 9, 2006:** Sourcefire VRT issues rules that protect against all potential exploits of the MS06-040 vulnerability.
- **October 23, 2008:** Microsoft issues Security Bulletin MS08-067 corresponding to a remote code execution vulnerability (similar to MS06-040) in the Microsoft Windows Server Service.
- **October 23, 2008:** Sourcefire VRT issues rules that protect against all potential exploits of the MS08-067 vulnerability.
- **November 21, 2008:** The Conficker.A worm is identified. Sourcefire VRT rules published on both August 9, 2006 and October 23, 2008 are triggered by Conficker.A due to the similarity of the corresponding Windows vulnerabilities.
- **December 29, 2008:** The Conficker.B worm is identified. This variant is covered by the VRT's existing rules.
- **March 4, 2009:** The Conficker.C worm is identified. This variant is covered by the VRT's existing rules.

As the Sourcefire VRT consistently publishes vulnerability-based rules rather than exploit-based signatures, Sourcefire 3D customers and open source Snort users benefited from zero-day protection more than two years in advance of the first Conficker outbreak.

SUMMARY

The research provided by the Sourcefire VRT is a critical component of the Sourcefire 3D System. Validated by a proven track record, the Sourcefire VRT has protected customers in advance of every significant outbreak of malware, including Nachi, Blaster, Sasser, Zotob, and many more, without the need for further updates to cover new variants.

The Sourcefire 3D System uses the following capabilities to provide protection before threats emerge:

- Vulnerability-based rules
- A powerful, open rules language
- The VRT methodology of:
 - » Researching the vulnerability
 - » Modeling the protocol
 - » Writing triggering conditions
 - » Testing the assumptions
- Sourcefire RNA, which:
 - » Identifies the set of threats that is potentially dangerous on customer networks
 - » Reduces or eliminates the need for tuning
 - » Provides network and endpoint intelligence

The Sourcefire 3D System – based on the 3D Approach (Discover, Determine, Defend) to securing real networks in real time – unifies intrusion and vulnerability management technologies to provide customers with the most effective network security solution, against all threats, from all vectors, all the time.