

Keeping 'em Honest

Testing and Validation of Network Security Devices

Dustin D. Trammell
Security Researcher
BreakingPoint Systems, Inc.



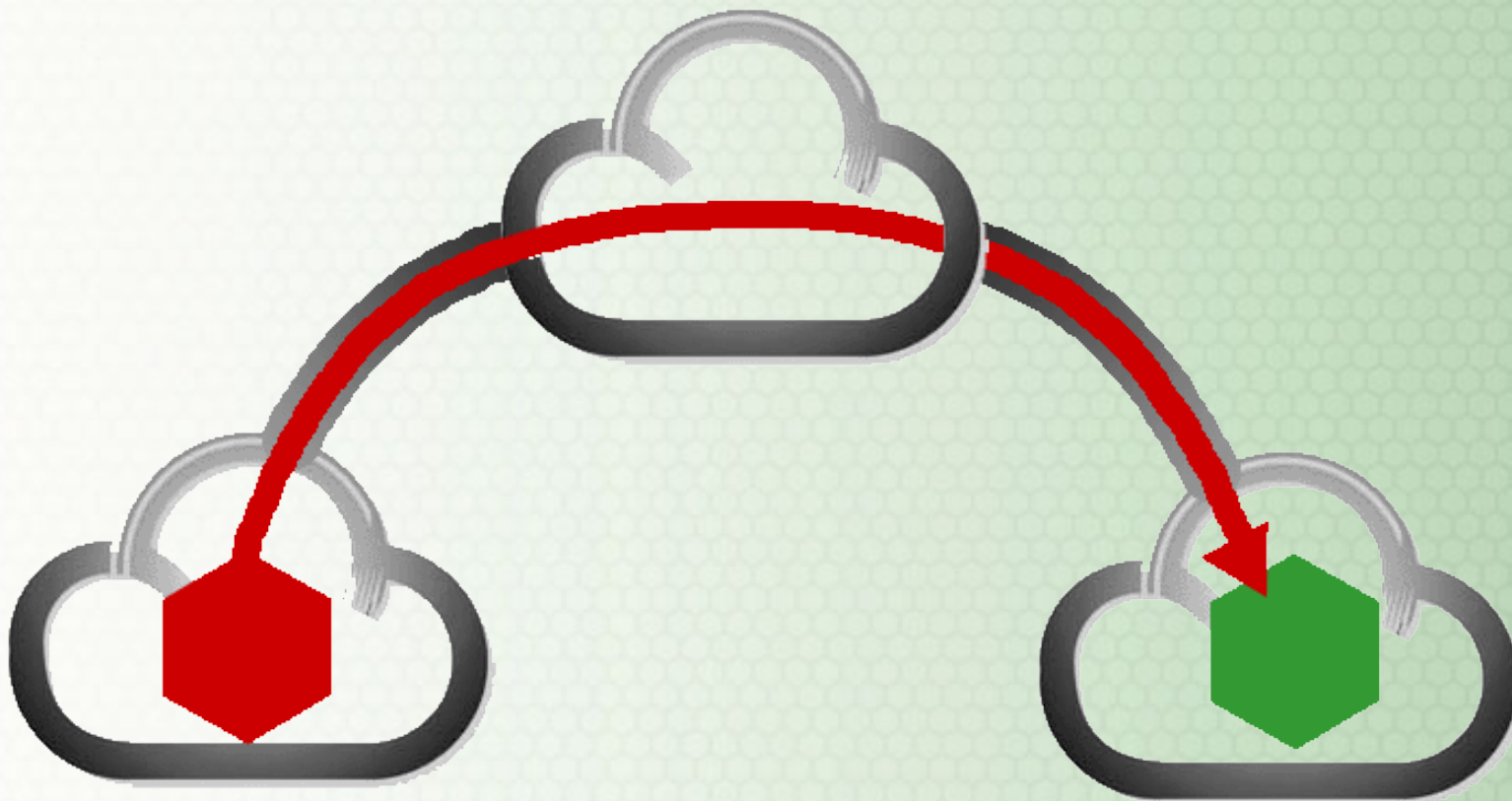
About Me

- ◆ Dustin D. Trammell
 - ◆ <http://www.dustintrammell.com>
- ◆ Founder, Computer Academic Underground (CAU)
- ◆ Co-Founder, Austin Hacker Association (AHA!)
- ◆ Employed by BreakingPoint Systems, Inc.
 - ◆ Security Researcher
 - ◆ Research properties and scope of vulnerabilities
 - ◆ Development of dynamic attacks and exploits

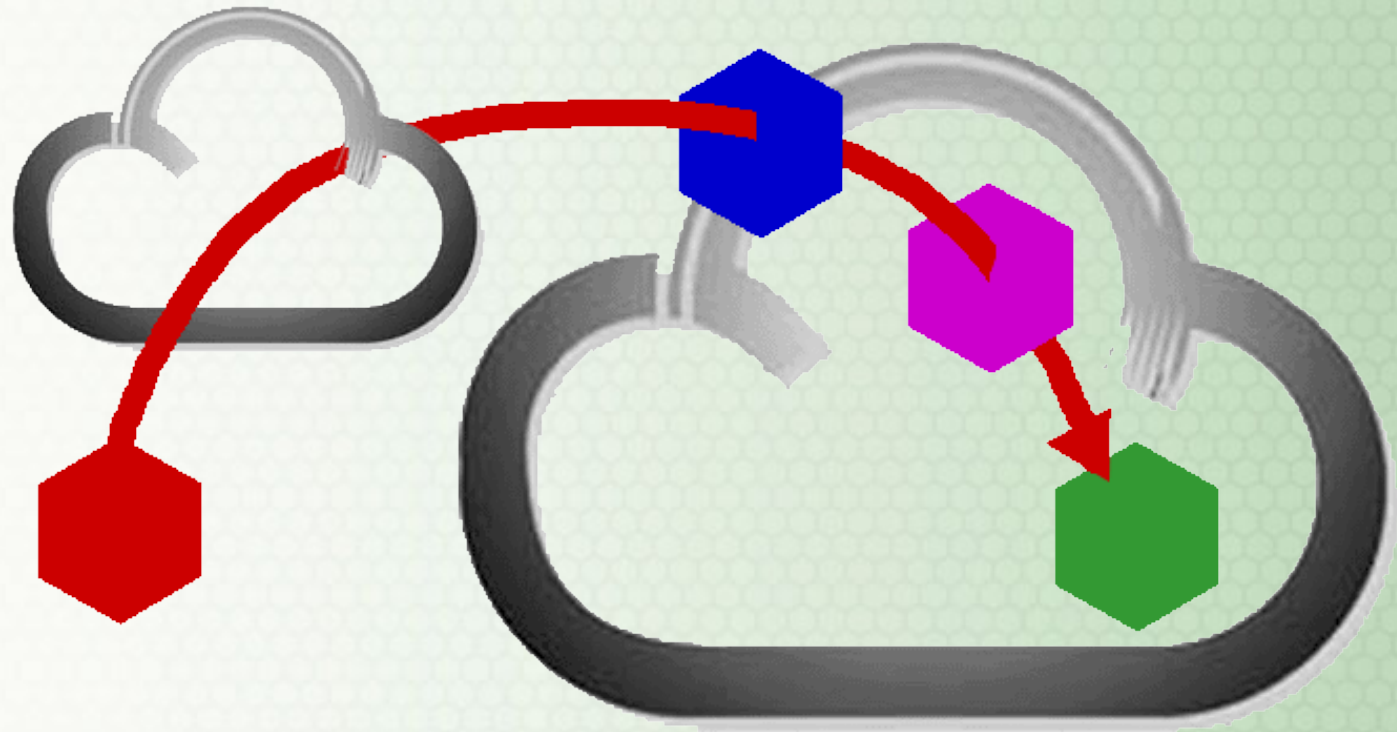
Background & Terminology: Anatomy of an Attack



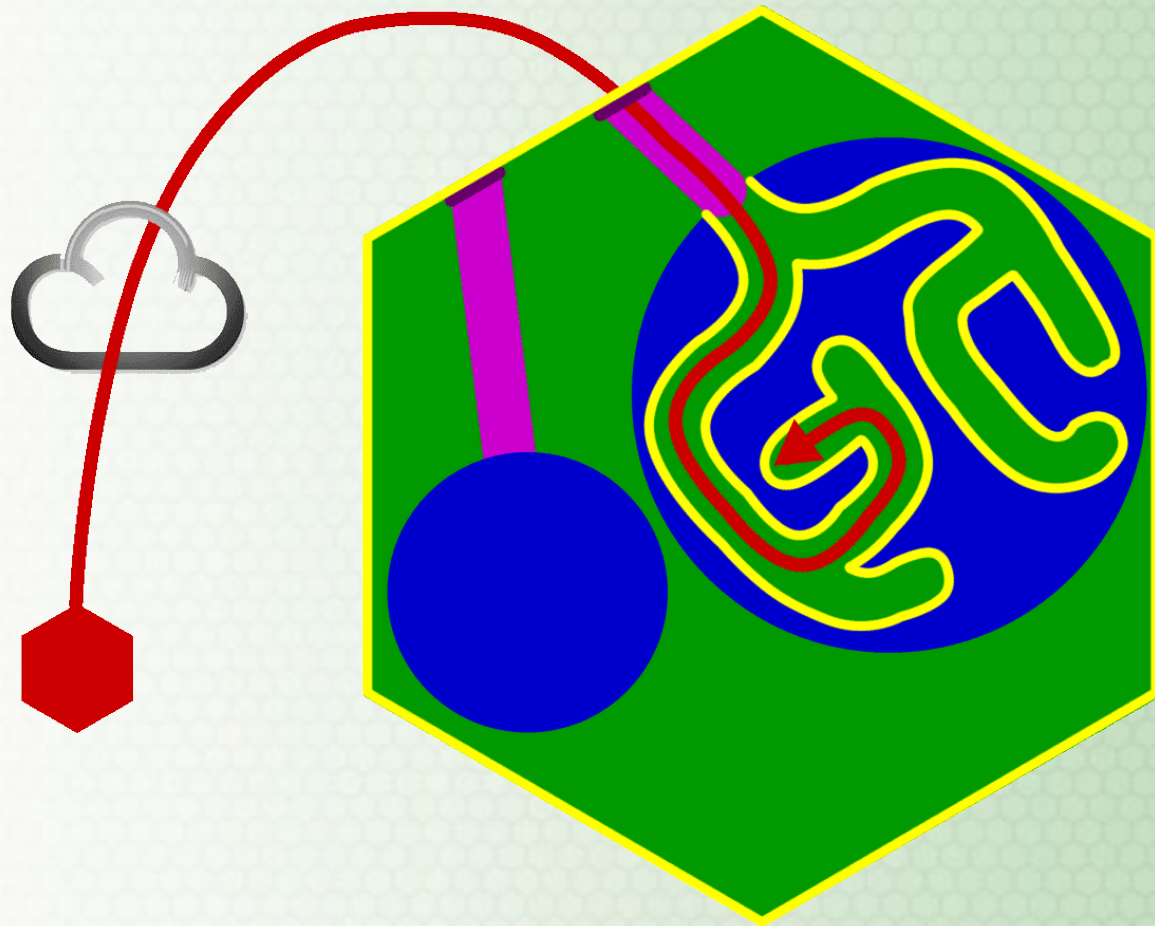
Anatomy of an Attack: Overview



Anatomy of an Attack: Network



Anatomy of an Attack: Target



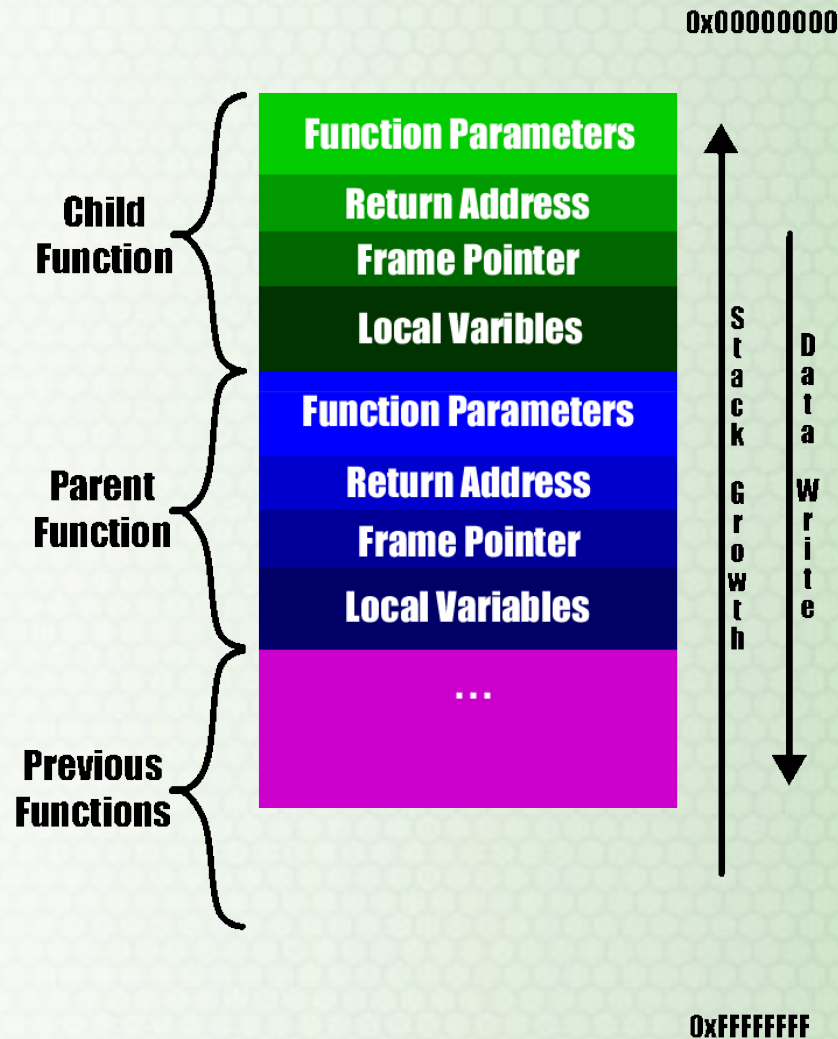
Anatomy of an Exploit: Vulnerable Code

```
int main() {  
    parent();  
    return 0;  
}
```

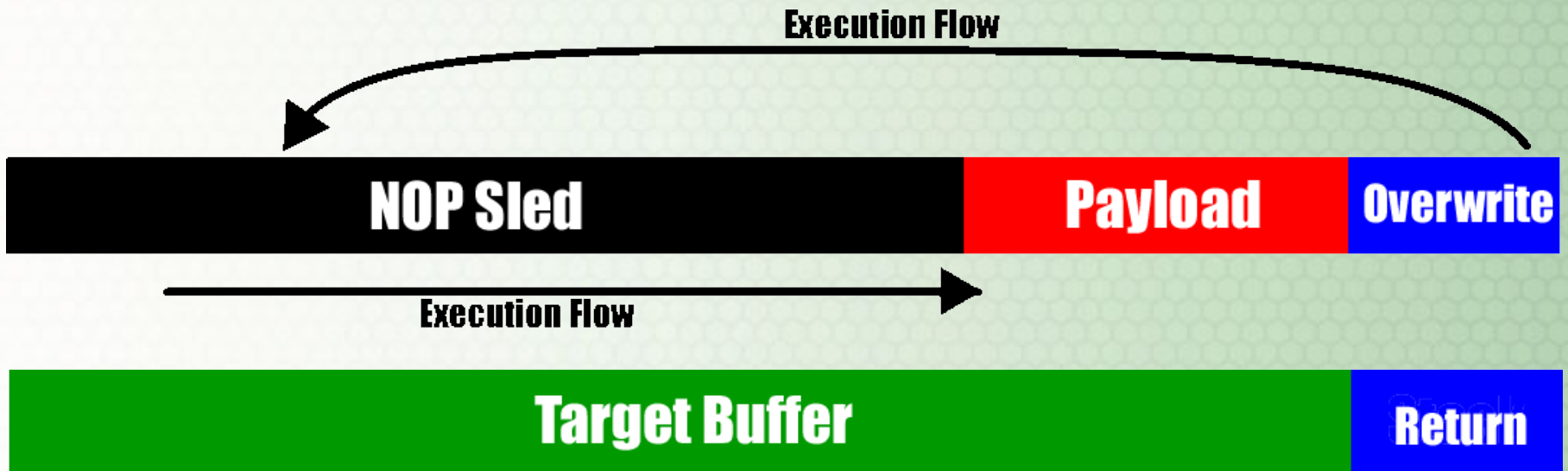
```
void parent() {  
    char string[13] = "Hello, World!";  
    child( string );  
}
```

```
void child( char string[9] ) {  
    printf( "%s\n", string );  
}
```

Anatomy of an Exploit: Call Stack



Anatomy of an Exploit: Buffer Overflow



Network Security Devices



Scope of Discussion

◆ Firewalls

- ◆ Traditional
- ◆ Application Aware
- ◆ Web Application
- ◆ Session Border Controllers (SBC) (VoIP)
- ◆ Client-side / Host-based / “Personal”

◆ Intrusion Monitoring

- ◆ Intrusion Detection Systems (IDS)
- ◆ Intrusion Prevention Systems (IPS)
- ◆ Host-based IPS

Firewalls



Firewall Business Model

- ◆ One-time sale of appliance or software
- ◆ Sale of independent “Application Modules” for content-awareness
- ◆ Service contract for:
 - ◆ Software updates
 - ◆ Support

Firewalls: Filtering Approach

- ◆ Layer 3
 - ◆ Source and Destination Network Addresses
- ◆ Layer 4
 - ◆ Transport Protocol (TCP/UDP)
 - ◆ Transport Protocol Ports
- ◆ Layer 5 - 7
 - ◆ Session Properties
 - ◆ Application Identification
 - ◆ Application Content

Firewall Vendor Claims



“We block Application X”

- ◆ What this may really mean:
 - ◆ “We block TCP ports xxx-yyy which App X uses”
 - ◆ “We detect App X’s setup sequence and block it”
 - ◆ “We actually do detect Application X and track it”
 - ◆ “...but only if it’s unencrypted.”
- ◆ Tests:
 - ◆ Try App X “raw” on different ports
 - ◆ Try detecting an App X session already in progress
 - ◆ Try App X over any legitimate transport protocol
 - ◆ Tunnel App X inside a different protocol
 - ◆ Wrap App X’s traffic in encryption

Intrusion Detection and Prevention Systems



IDS/IPS Business Model

- ◆ One-time sale of appliance or software
- ◆ Content service for signature / filter updates
 - ◆ Content packs add and remove filters
 - ◆ Incremental signature / filter solutions for new vulnerabilities
- ◆ Service contract for:
 - ◆ Software updates
 - ◆ Support

IDS/IPS Filter Approach

- ◆ Network Layer Information:
 - ◆ Source and Destination Network Addresses
- ◆ Transport Layer Information:
 - ◆ Transport Protocol (TCP/UDP)
 - ◆ Transport Protocol Ports
- ◆ Application Layer Information:
 - ◆ Static data matching
 - ◆ Regular Expression matching
 - ◆ Behavioral tracking / Anomaly Detection

IDS/IPS Filter Classification

- ◆ Vulnerability Filters (the holy grail)
 - ◆ Usually a combination of:
 - ◆ Attack vector
 - ◆ Something vulnerability specific
- ◆ Policy Filters
 - ◆ Transports, Protocols, file types, functions, etc.
 - ◆ Think firewall & attack vectors
- ◆ Exploit Filters
 - ◆ Generally one-offs, found in the wild
 - ◆ Think malware/anti-virus approach
- ◆ Payload Filters
 - ◆ `exec /bin/sh`, `mail /etc/passwd`, etc.
- ◆ Evasion Filters
 - ◆ Data encodings / padding techniques
 - ◆ Encoded payload decoder stubs

Retaining IDS/IPS Performance

◆ Vendor

- ◆ Ship product with a small default filter set
- ◆ Provide a larger “recommended” filter set
- ◆ Disable old default filters in favor of new
- ◆ Encourage customers to customize filter set

◆ Customer

- ◆ Define your own filter set
- ◆ Test content updates prior to production

IDS/IPS Vendor Claims



“We detect/block vulnerability X”

◆ What this may really mean:

- ◆ “We detect traffic that looks like exploit A, B, etc. for vulnerability X”
- ◆ “We detect traffic targeting TCP/UDP ports xxxx-yyyy”
- ◆ “We block legitimate functionality which is used as an attack vector for vulnerability X”
- ◆ “We actually do block vulnerability X”

◆ Tests:

- ◆ Trigger the vulnerability via different attack vectors
- ◆ Randomize data in the exploit and payload
- ◆ Use exploit and payload encoders
- ◆ Send legitimate traffic that uses the functionality (false positive)

Designing Test Cases for Vulnerability Coverage Testing



Common Failures

- ◆ Filter is specific to a particular attack vector
 - ◆ Vectors: HTTP, FTP, SMTP, etc.
 - ◆ Reason: Most devices are architected to target narrow scopes (performance)
- ◆ Filter is specific to a particular exploit
 - ◆ Many exploits are caught in the wild prior to the vulnerability being completely understood
 - ◆ Reason: Filters are developed once and not updated
 - ◆ Reason: Filters aren't aware of appropriate locations for permutation (too static)

Who Should Test?

◆ Vendor

- ◆ Verify their filters are as robust as possible
- ◆ Verify their filters do what they claim they do

◆ Customer

- ◆ Verify the vendor's claims are valid
- ◆ Peace of mind

◆ 3rd Party

- ◆ Customer Audit / Compliance
- ◆ Vendor Product Certification

Testing Methodology Overview

- ◆ Run as many unique test cases for a single vulnerability as you possibly can
- ◆ Accurately reproducing a single or a set of test cases is extremely important
- ◆ Use randomization similar to Fuzzing methodologies

Know Your Target

◆ Techniques:

- ◆ Test as many ways to trigger the vulnerability as possible
- ◆ Test legitimate use cases of the target vulnerable functionality

◆ Goals:

- ◆ Verify that vulnerability filters are actually triggering on the vulnerability itself
- ◆ Verify that vulnerability filters are not triggering solely on the target functionality

Know Your Attack Vectors

◆ Techniques:

- ◆ Use as many network transports as possible
- ◆ Use as many protocols as possible
- ◆ Use as many code paths as possible
- ◆ Test with both enabled and disabled Policy Filters

◆ Goals:

- ◆ Verify that vulnerability filters are not triggering solely on the attack vector
- ◆ Verify that vulnerability filters are not reliant upon a particular attack vector
- ◆ Verify that policy filters trigger on appropriate attack vectors

Know Your Exploit

◆ Techniques:

- ◆ Randomize values as much as possible
- ◆ Randomize locations as much as possible
- ◆ Randomize sizes as much as possible
- ◆ Use as many data encodings as possible

◆ Goals:

- ◆ Verify that vulnerability filters aren't reliant upon specific versions of exploits
- ◆ Verify that filters aren't reliant upon a particular data encoding
- ◆ Verify that exploit filters catch exploit permutations
- ◆ Verify that evasion filters catch encodings, padding, etc.

Know Your Payload

◆ Techniques:

- ◆ Use randomized data as payload
- ◆ Use payload encoders

◆ Goals:

- ◆ Verify that vulnerability filters are not reliant upon specific payloads
- ◆ Verify that exploit filters are not reliant upon specific payloads
- ◆ Verify that evasion filters are triggering on appropriate payload encoders

Vulnerability Filter Case Studies



MS07-055

- ◆ Kodak Image Viewer Remote Code Execution
- ◆ Microsoft Rating: Critical
- ◆ Default image viewer on Windows 2000 and systems upgraded from Windows 2000
- ◆ Vulnerability in how the viewer handles maliciously-crafted TIFF files
- ◆ Bug in parsing the BitsPerSample IFD entry
 - ◆ Set this structure's Offset field to arbitrary file data

MS07-055 Properties

- ◆ TIFF internal values can be encoded as either little-endian or big-endian format
- ◆ TIFF is an object-oriented file format
- ◆ Objects can appear in any order
- ◆ Vulnerability is in the parsing of a particular IFD entry object
- ◆ IFD entry objects can be in any order
- ◆ Multiple IFD objects can exist in the file

MS07-055 Filter Deficiencies

- ◆ One IPS would only detect malicious big-endian TIFF files
- ◆ One IPS would only detect if the malicious object was within the first 310 bytes
- ◆ Multiple IPSs would only detect the malicious file when transferred via HTTP

MS07-061

- ◆ Windows URI Handling Remote Code Execution
- ◆ Microsoft Rating: Critical
- ◆ All versions of Windows XP and Server 2003
- ◆ Vulnerability in how the Windows shell handles maliciously-crafted URIs
- ◆ Vector Requirement: Needs a method for resolving a URI without user intervention

MS07-061 Properties

- ◆ Bug in Adobe PDF Viewer allows “mailto:” URI resolution without user intervention!
- ◆ PDF is an object-oriented file format
- ◆ PDF objects can appear in any order
- ◆ PDF action object triggers URI resolution via the OSs ShellExecute()
- ◆ ShellExecute() calls IE7 as default URI handler for “mailto:”
- ◆ IE7 bails back to the OS because the URI is malformed
- ◆ ShellExecute() attempts to fix and resolve the URI again
- ◆ ShellExecute() inadvertently executes arbitrary commands

MS07-061 Filter Deficiencies

- ◆ One IPS would only detect a particular version of the malicious URI
- ◆ Multiple IPSs would only detect the malicious file when transferred via HTTP or SMTP
- ◆ One IPS would only detect the malicious file when encoded within SMTP as base64
- ◆ Multiple IPSs tested expected PDF as the attack vector for the malicious URI

Device Performance Testing



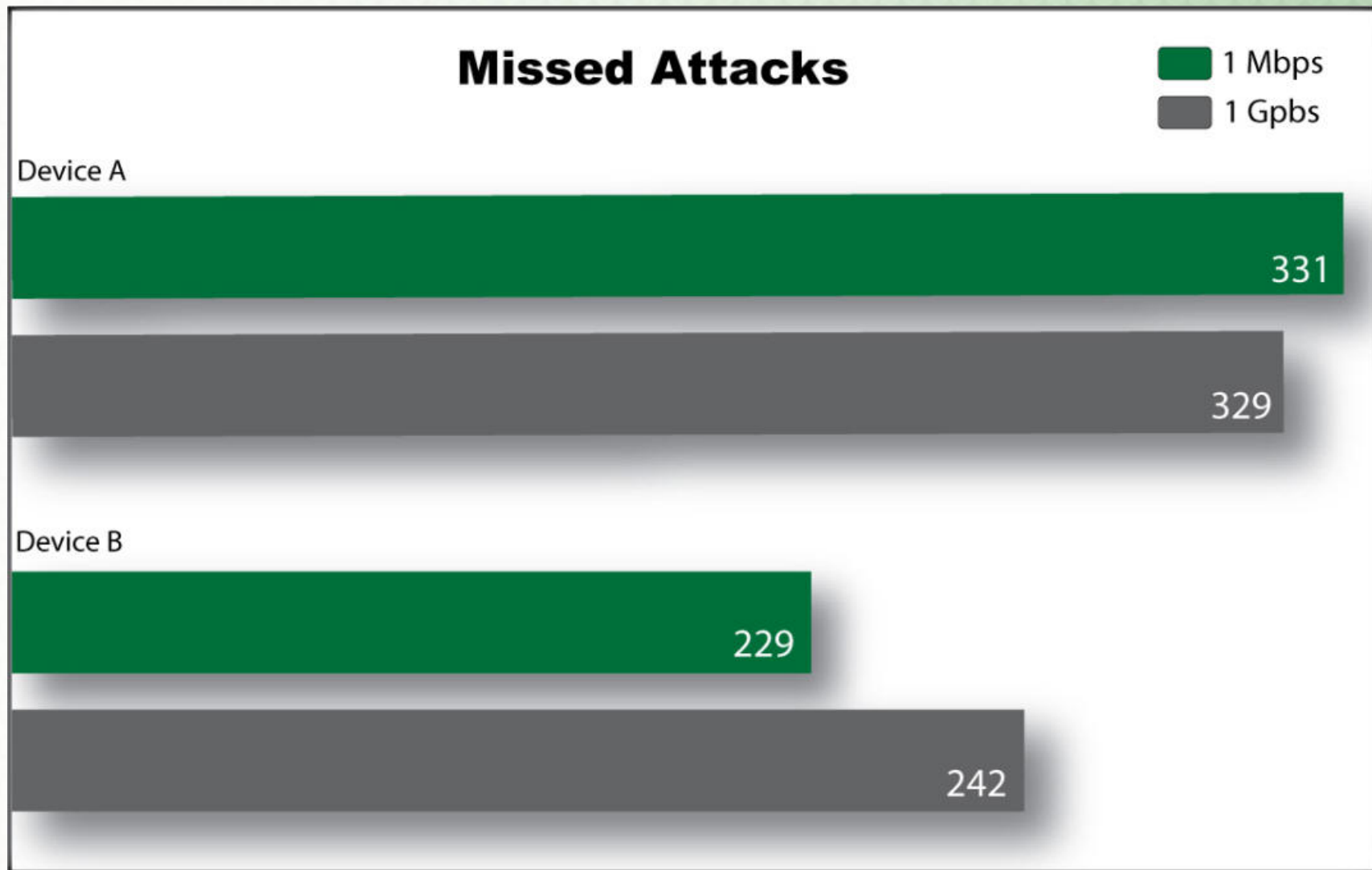
Performance Testing

- ◆ Device's ability to provide security coverage while under load
- ◆ Most security vendors don't test this:
 - ◆ Filter set tests are usually performed in a vacuum
 - ◆ Filter sets are tested with a single test case or small samplings of test data
- ◆ Approaches:
 - ◆ Test on a live network (not recommended)
 - ◆ Use application simulation appliance or software
 - ◆ Re-use vulnerability test cases from previous tests
 - ◆ Run test cases and application simulation simultaneously

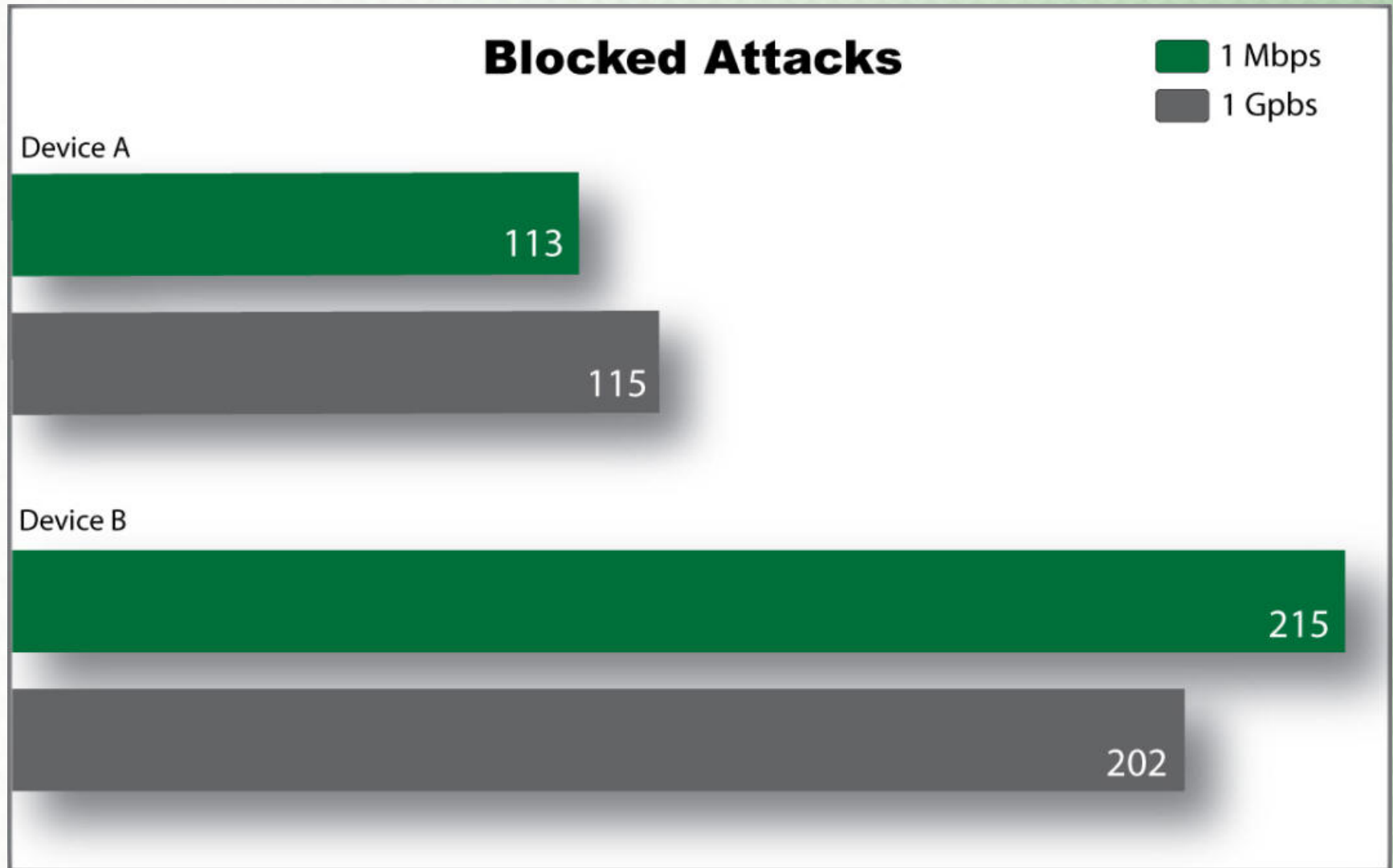
Performance Test Case

- ◆ Two test scenarios:
 - ◆ Set of attacks with 1 Mb/s background traffic
 - ◆ Set of attacks with 1 Gb/s background traffic
- ◆ Two industry leading devices tested:
 - ◆ Device A
 - ◆ Industry Leader
 - ◆ Multi-gigabit IPS
 - ◆ Device B
 - ◆ Industry Leader's close competitor
 - ◆ Multi-gigabit IPS

Missed Attacks Under Load



Blocked Attacks Under Load



Filter Set Performance Test

- ◆ Determining how a content update and filter-set size impacts latency
- ◆ Three tests
 - ◆ Baseline: All Filters Disabled
 - ◆ Pre-Update: Vendor Default Filter Set
 - ◆ Post-Update: Updated Filters Enabled

Maximum Latency Observed

Maximum Latency

Post-Update Device Latency 69.74 ms



Pre-Update Device Latency 0.14 ms

1.2x

Baseline Latency 0.11 ms



Average Latency Observed

Average Latency

Post-Update Average Latency 0.30 ms

10x

Pre-Update Average Latency 0.03 ms

Baseline Average Latency 0.03 ms

Conclusions

