build | integrate | secure

The Self Healing Cloud

Protecting Applications and Infrastructure with Automated Virtual Patching

Dan Cornell
CTO, Denim Group
@danielcornell

Bio: Dan Cornell

- Founder and CTO, Denim Group
- Software developer by background (Java, .NET)

- OWASP
  - *San Antonio Chapter Leader*
  - *Open Review Project Leader*
  - *Chair of the Global Membership Committee*

- Speaking
  - *RSA, SOURCE Boston*
  - *OWASP AppSec, Portugal Summit, AppSecEU Dublin*
  - *ROOTS in Norway*

Denim Group Background

- Secure software services and products company
  - *Builds secure software*
  - *Helps organizations assess and mitigate risk of in-house developed and third party software*
  - *Provides classroom training and e-Learning so clients can build software securely*
- Software-centric view of application security
  - *Application security experts are practicing developers*
  - *Development pedigree translates to rapport with development managers*
  - ***Business impact: shorter time-to-fix application vulnerabilities***
- Culture of application security innovation and contribution
  - *Develops open source tools to help clients mature their software security programs*
    - *Remediation Resource Center, ThreadFix, Sprajax*
  - *OWASP national leaders & regular speakers at RSA, SANS, OWASP, ISSA, CSI*
  - *World class alliance partners accelerate innovation to solve client problems*

The Cloud!

An Apology

- Did anyone attend this talk because it had the word “cloud” in the title?
- If so … I’m sorry
  - *Marketing made me do it*
  - *But this really does apply to certain aspects of “the cloud”*
  - *I promise…*
- At least we didn’t mention Advanced Persistent Threats
  - *Yet…*

Who Is Your Worst Enemy?

The Problem

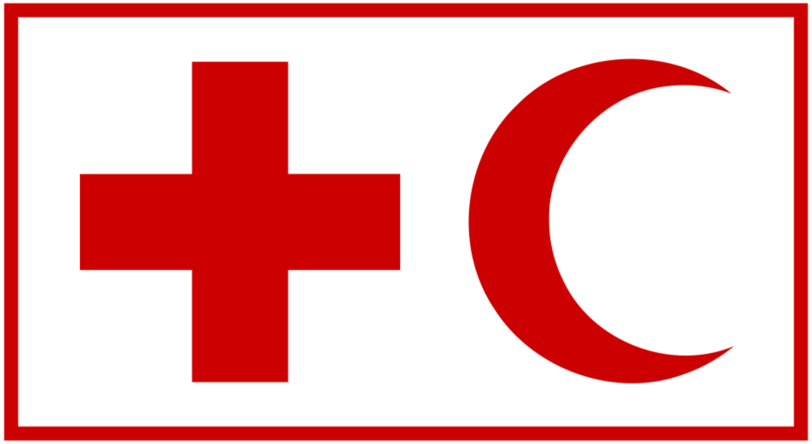
- Code with automatically-identifiable security vulnerabilities gets deployed
- Trolling attackers find vulnerabilities and exploit them
- Profit?

A Proposed Solution

1. Identify newly-deployed code

2. Identify vulnerabilities

3. Block traffic that would exploit those vulnerabilities



Other Potential Solutions

- Run a web application firewall (WAF)
  - *You do not have one*
  - *Code changes too frequently for WAF training*
  - *WAF blocked legitimate transactions and is back in training mode*
- Find the vulnerabilities and fix the code
  - *Prioritization of new features over security fixes*
  - *Code deployments take too long*
- Do not introduce the vulnerabilities in the first place
  - *Very funny…*

Step 1: Identify Newly-Deployed Code

- Wait to be notified about new application deployments by the development teams
- Scan your network space for new servers and applications
- Monitor files and directories

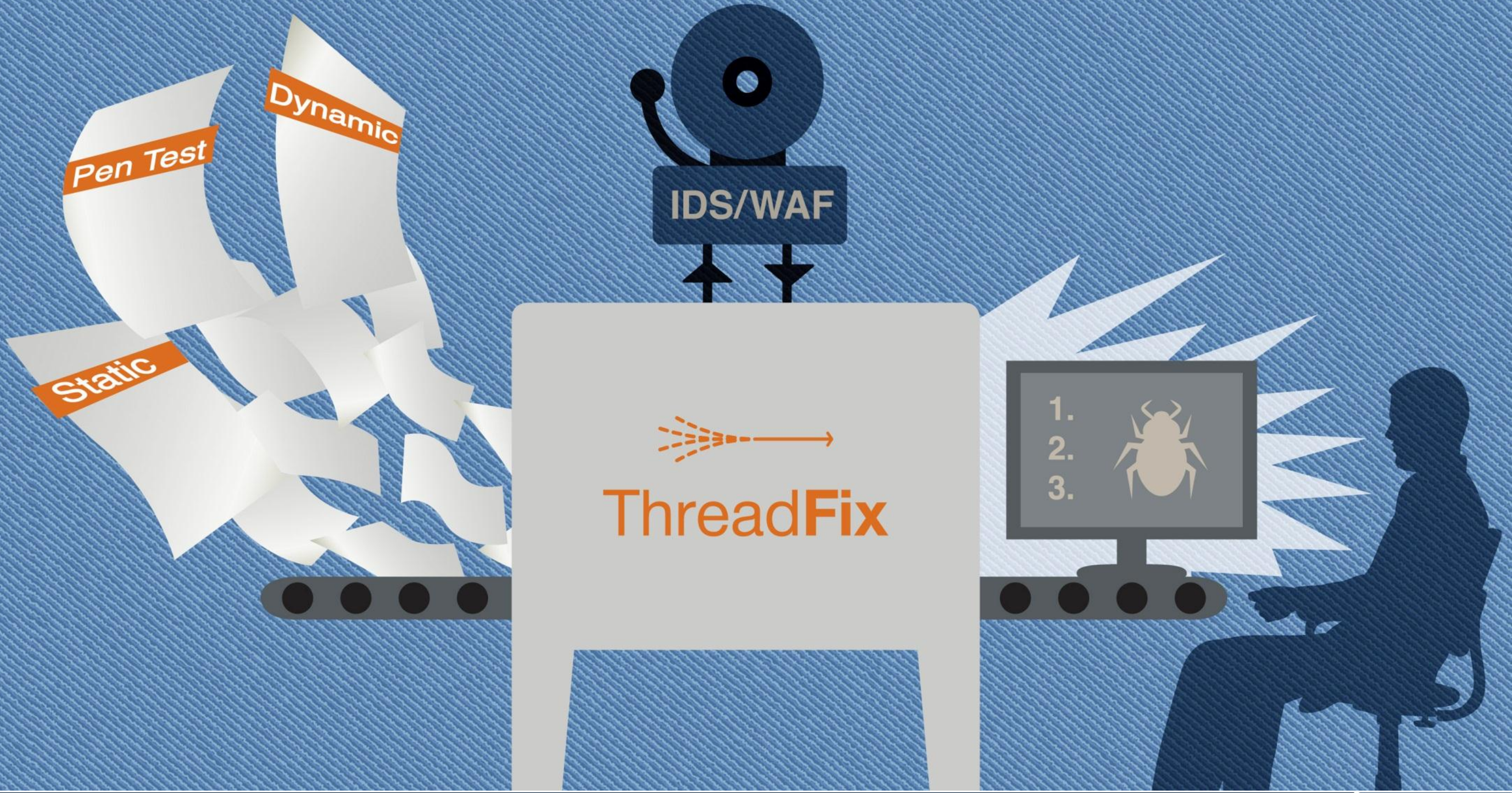
Step 2: Identify Vulnerabilities

- Manual testing
- Automated scanning
- Manual-assisted scanning

Step 3: Block Traffic That Would Exploit Vulnerabilities

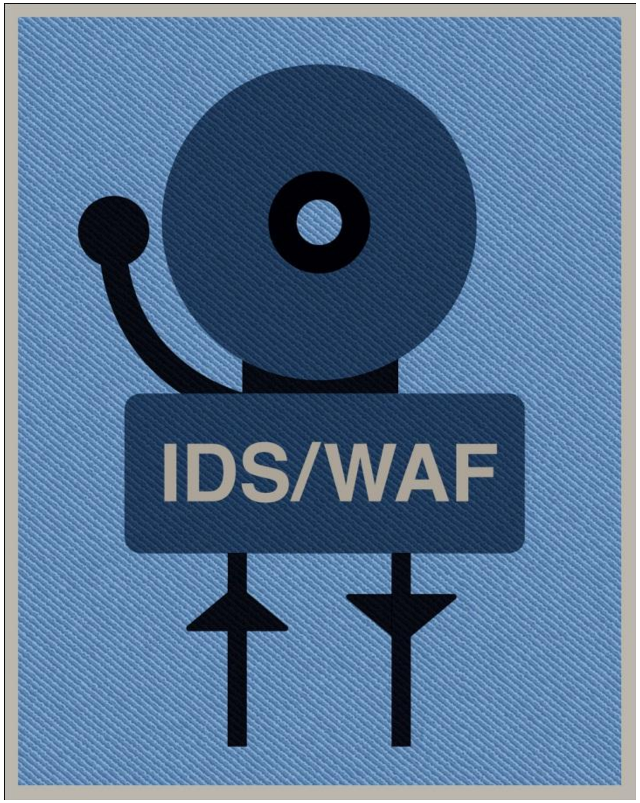
- Generate virtual patches to block traffic to identified vulnerabilities

ThreadFix – Consolidating vulnerability data so managers can speak intelligently about the status and trends of security within their organization



Virtual Patching

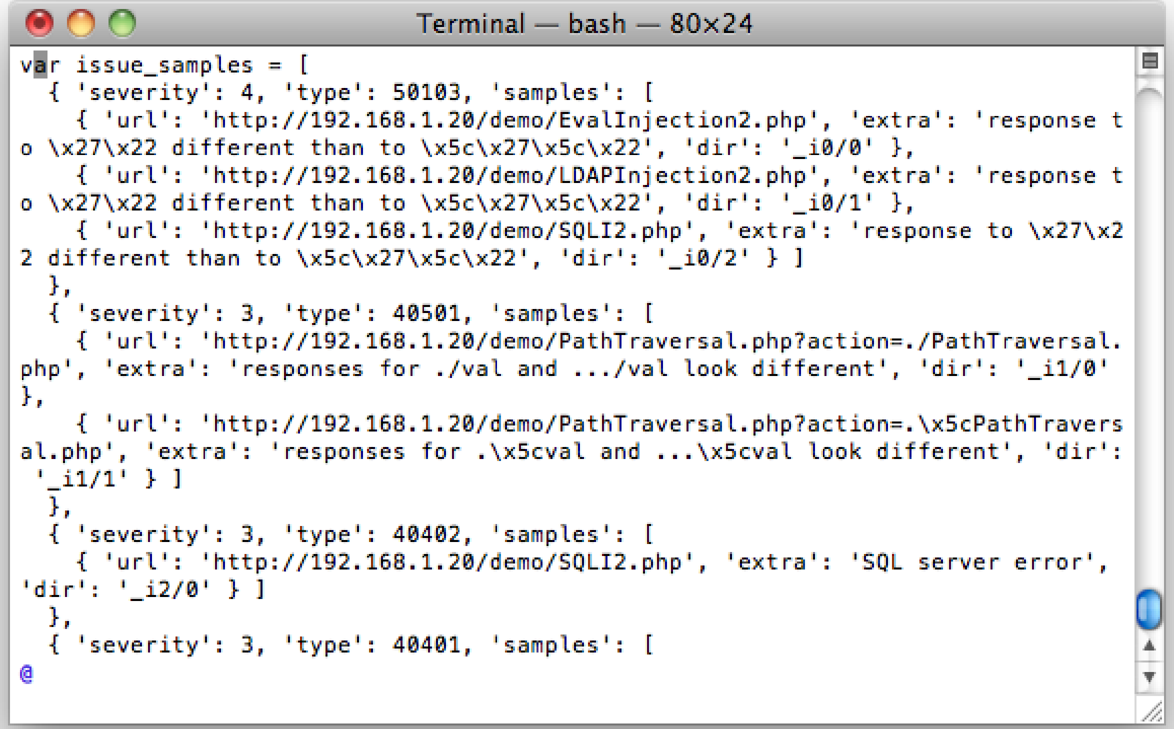
- Connect vulnerability scanners to IDS/IPS/WAF systems
- Map data from sensors back to data about vulnerabilities



Solution Specifics

- Code Change Detection: Watch for filesystem changes
  - *Could wire up to diffs of nmap scans but this was easier given test environment*
- Vulnerability Detection: Automated skipfish and w3af scans
  - *Open source technologies: anyone can replicate*
  - *Ability to run unattended*
- Blocking Traffic: Rules for snort and mod_security
  - *Open source technologies: anyone can replicate*
  - *Rule compatibility*

Skipfish Vulnerability Data




```
var issue_samples = [
  { 'severity': 4, 'type': 50103, 'samples': [
    { 'url': 'http://192.168.1.20/demo/EvalInjection2.php', 'extra': 'response t
o \x27\x22 different than to \x5c\x27\x5c\x22', 'dir': '_i0/0' },
    { 'url': 'http://192.168.1.20/demo/LDAPInjection2.php', 'extra': 'response t
o \x27\x22 different than to \x5c\x27\x5c\x22', 'dir': '_i0/1' },
    { 'url': 'http://192.168.1.20/demo/SQLI2.php', 'extra': 'response to \x27\x2
2 different than to \x5c\x27\x5c\x22', 'dir': '_i0/2' } ]
  },
  { 'severity': 3, 'type': 40501, 'samples': [
    { 'url': 'http://192.168.1.20/demo/PathTraversal.php?action=./PathTraversal.
php', 'extra': 'responses for ./val and .../val look different', 'dir': '_i1/0'
},
    { 'url': 'http://192.168.1.20/demo/PathTraversal.php?action=.\x5cPathTravers
al.php', 'extra': 'responses for .\x5cval and ...\x5cval look different', 'dir':
 '_i1/1' } ]
  },
  { 'severity': 3, 'type': 40402, 'samples': [
    { 'url': 'http://192.168.1.20/demo/SQLI2.php', 'extra': 'SQL server error',
'dir': '_i2/0' } ]
  },
  { 'severity': 3, 'type': 40401, 'samples': [
@
```

w3af Vulnerability Data

Terminal — vim — 80×24


```
    <vulnerability id="[4268]" method="POST" name="OS commanding vulnerability"
plugin="osCommanding" severity="High" url="http://192.168.1.20/demo/OSCommandInj
ection2.php" var="fileName">
        OS Commanding was found at: &quot;http://192.168.1.20/demo/OSCommandInje
ction2.php&quot;, using HTTP method POST. The sent post-data was: &quot;fileName
=%7Cping+-n+3+localhost&quot;. This vulnerability was found in the request with
id 4268.
    </vulnerability>
    <vulnerability id="[4518, 4519]" method="POST" name="Blind SQL injection vul
nerability" plugin="" severity="High" url="http://192.168.1.20/demo/SQLI2.php" v
ar="username">
        Blind SQL injection was found at: &quot;http://192.168.1.20/demo/SQLI2.p
hp&quot;, using HTTP method POST. The injectable parameter is: &quot;username&qu
ot;. This vulnerability was found in the requests with ids 4518 to 4519.
    </vulnerability>
    <vulnerability id="[4803]" method="POST" name="SQL injection vulnerability"
plugin="sqli" severity="High" url="http://192.168.1.20/demo/SQLI2.php" var="user
name">
        SQL injection in a MySQL database was found at: &quot;http://192.168.1.2
0/demo/SQLI2.php&quot;, using HTTP method POST. The sent post-data was: &quot;us
ername=d'z&quot;0&quot;. This vulnerability was found in the request with id 480
3.
    </vulnerability>
```

IBM Rational AppScan Vulnerability Data

Terminal — vim — 80×24

```
  <IssueTypes>^M
    <Total>18</Total>^M
    <IssueType ID="attBlindSqlInjectionStrings" Count="2">^M
      <RemediationID>fix_52000</RemediationID>^M
      <advisory>^M
        <name>Blind SQL Injection</name>^M
        <testDescription>Application-level test</testDescription>^M
        <threatClassification>^M
          <name>Command Execution: SQL Injection</name>^M
          <reference>http://www.webappsec.org/projects/threat/classes/sql_inje
ction.shtml</reference>^M
        </threatClassification>^M
        <testTechnicalDescription>^M
          <text>Web applications often use databases at the backend to interac
t with the enterprise data warehouse. The de-facto standard language for queryin
g databases is SQL (each major database vendor has its own dialect). Web applica
tions often take user input (taken out of the HTTP request) and incorporate it i
n an SQL query, which is then sent to the backend database. The query results ar
e then processed by the application and sometimes displayed to the user.</text>^
M
          <br />^M
          <br />^M
@
```

Vulnerability Data

- Normalize what is provided by the scanners
- De-duplicate the results
  - *Allows for use of multiple scanning technologies*

- (vulnerability_type, vulnerable_url, injection_point)
  - *Typically needed for injection-type vulnerabilities: SQL injection, XSS*

- (vulnerability_type, vulnerable_url)
  - *Sufficient for some vulnerabilities: Predictable resource location, directory listing*

Vulnerability Data – What Else Do We Need?

- Standardized access to payload information would be nice

- Current rules have potential for false blocks
  - *SQL injection: Is the problem based on the code mis-handling ‘ or “*

Virtual Patches - Snort

Terminal — vim — 80×24

```
drop tcp $EXTERNAL_NET any -> $HTTP_SERVERS $HTTP_PORTS (uricontent:"/demo/EvalI
njection2.php"; msg:"SQL Injection attempt"; flow: to_server,established; pcre:"
/(\n|^|\?|\&)(command=[^\&|\n]*('|\%27|\"|\%22|--|\%2D\%2D))/i"; classtype:Web-a
pplication-attack; sid:100000;)^M

drop tcp $EXTERNAL_NET any -> $HTTP_SERVERS $HTTP_PORTS (uricontent:"/demo/LDAPI
njection2.php"; msg:"SQL Injection attempt"; flow: to_server,established; pcre:"
/(\n|^|\?|\&)(username=[^\&|\n]*('|\%27|\"|\%22|--|\%2D\%2D))/i"; classtype:Web-
application-attack; sid:100001;)^M

drop tcp $EXTERNAL_NET any -> $HTTP_SERVERS $HTTP_PORTS (uricontent:"/demo/SQLI2
.php"; msg:"SQL Injection attempt"; flow: to_server,established; pcre:"/(\n|^|\?
|\&)(username=[^\&|\n]*('|\%27|\"|\%22|--|\%2D\%2D))/i"; classtype:Web-applicati
on-attack; sid:100002;)^M

drop tcp $EXTERNAL_NET any -> $HTTP_SERVERS $HTTP_PORTS (uricontent:"/demo/PathT
raversal.php"; msg:"Path Traversal attempt"; flow: to_server,established; pcre:"
/(\n|^|\?|\&)(action=[^\&|\n]*(.|\%2e))/i"; classtype:Web-application-attack; si
d:100003;)^M

@
@
@
```

Virtual Patches – mod_security

Terminal — vim — 80×24

```
SecRule REQUEST_URI "^\/demo\/XPathInjection2\.php""phase:2,chain,deny,msg:'Cros
s-site Scripting attempt: /demo/XPathInjection2.php [password]',id:'100000',seve
rity:'2'"
SecRule ARGS:password "<|\%3C|>|\%3E"
^M
SecRule REQUEST_URI "^\/demo\/EvalInjection2\.php""phase:2,chain,deny,msg:'Cross
-site Scripting attempt: /demo/EvalInjection2.php [command]',id:'100001',severit
y:'2'"
SecRule ARGS:command "<|\%3C|>|\%3E"
^M
SecRule REQUEST_URI "^\/demo\/XSS-reflected2\.php""phase:2,chain,deny,msg:'Cross
-site Scripting attempt: /demo/XSS-reflected2.php [username]',id:'100002',severi
ty:'2'"
SecRule ARGS:username "<|\%3C|>|\%3E"
^M
SecRule REQUEST_URI "^\/demo\/XPathInjection2\.php""phase:2,chain,deny,msg:'Cros
s-site Scripting attempt: /demo/XPathInjection2.php [username]',id:'100003',seve
rity:'2'"
SecRule ARGS:username "<|\%3C|>|\%3E"
^M
SecRule REQUEST_URI "^\/demo\/XPathInjection2\.php[^?]*(<|\%3C|>|\%3E)""phase:2,
deny,msg:'Cross-site Scripting attempt: /demo/XPathInjection2.php',id:'100004',s
everity:'2'"
```

Virtual Patches - Formats

- Two approaches
  1. *(vulnerability_type, vulnerability_location)*
  2. *(vulnerability_signature , vulnerability_location)*

(1) “There is a reflected XSS vulnerability in login.php for the username parameter”

versus

(2) “Watch out for HTML-ish characters in login.php for the username parameter”

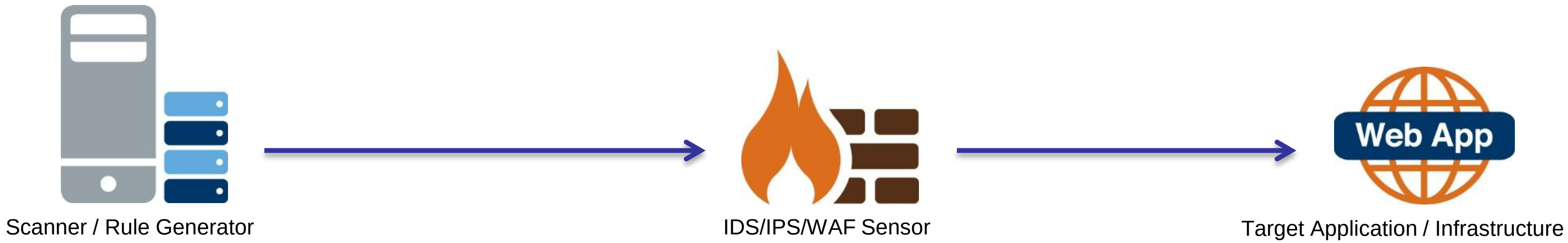
- The snort and mod_security rules follow approach (2)
- Integration with commercial solutions may use approach (1)

Standard for Virtual Patch Success

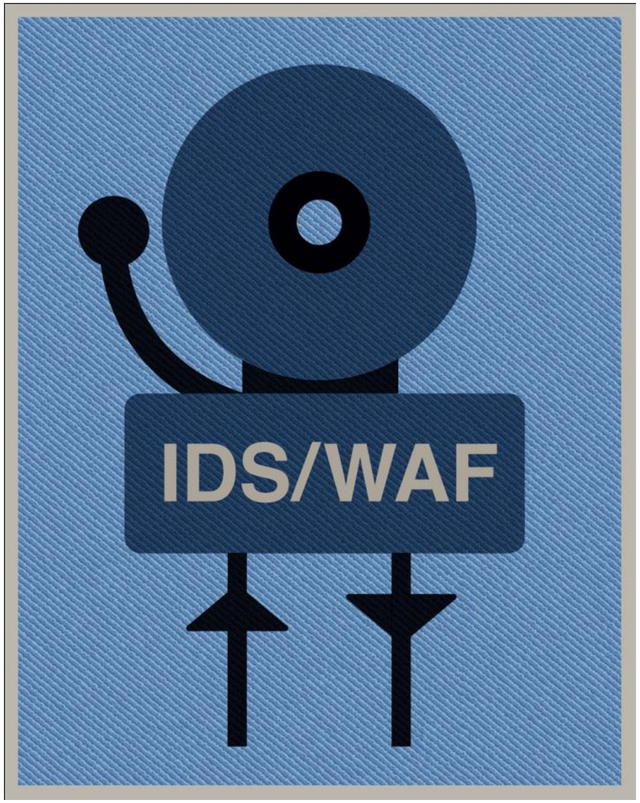
- If the scanner shuts up the vulnerability is considered “fixed”
- Tweak the detection payloads until this is the case for all scanners
- Watch out for overly-aggressive signatures

- But that won’t stop Advanced Persistent Threats!
  - *True*
  - *But that wasn’t really the goal at the current time*

Test Environment



Demo!



Results

- Snort

- mod_security
  - *No rules*
  - *Compared to Core Ruleset (CRS)*

- Why compare to the Core Ruleset?

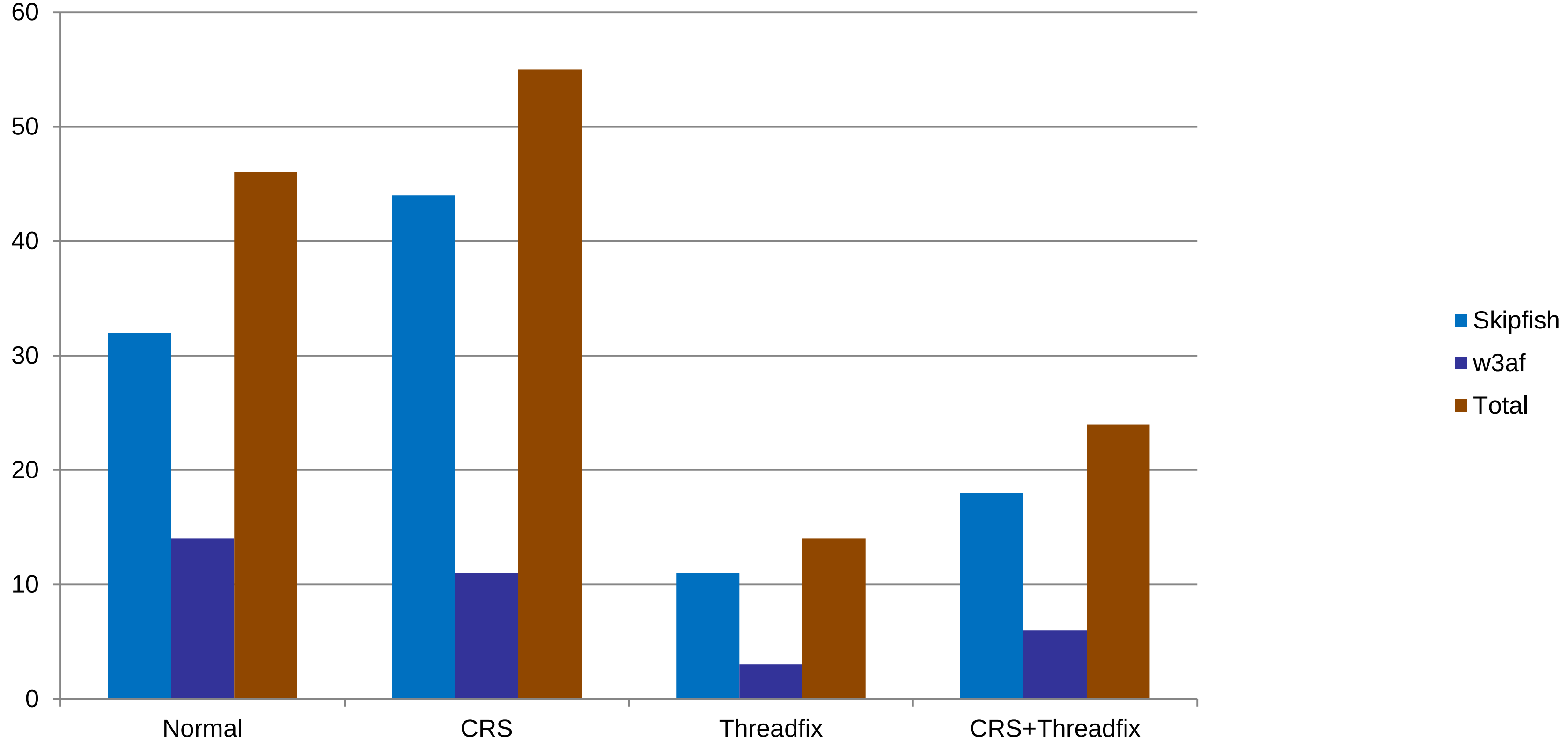
Snort Results

Snort v. 2.9.0.5			
All Vulns			
	Skipfish	**w3af**	**Total**
Normal	20	10	30
Threadfix	10	?	10

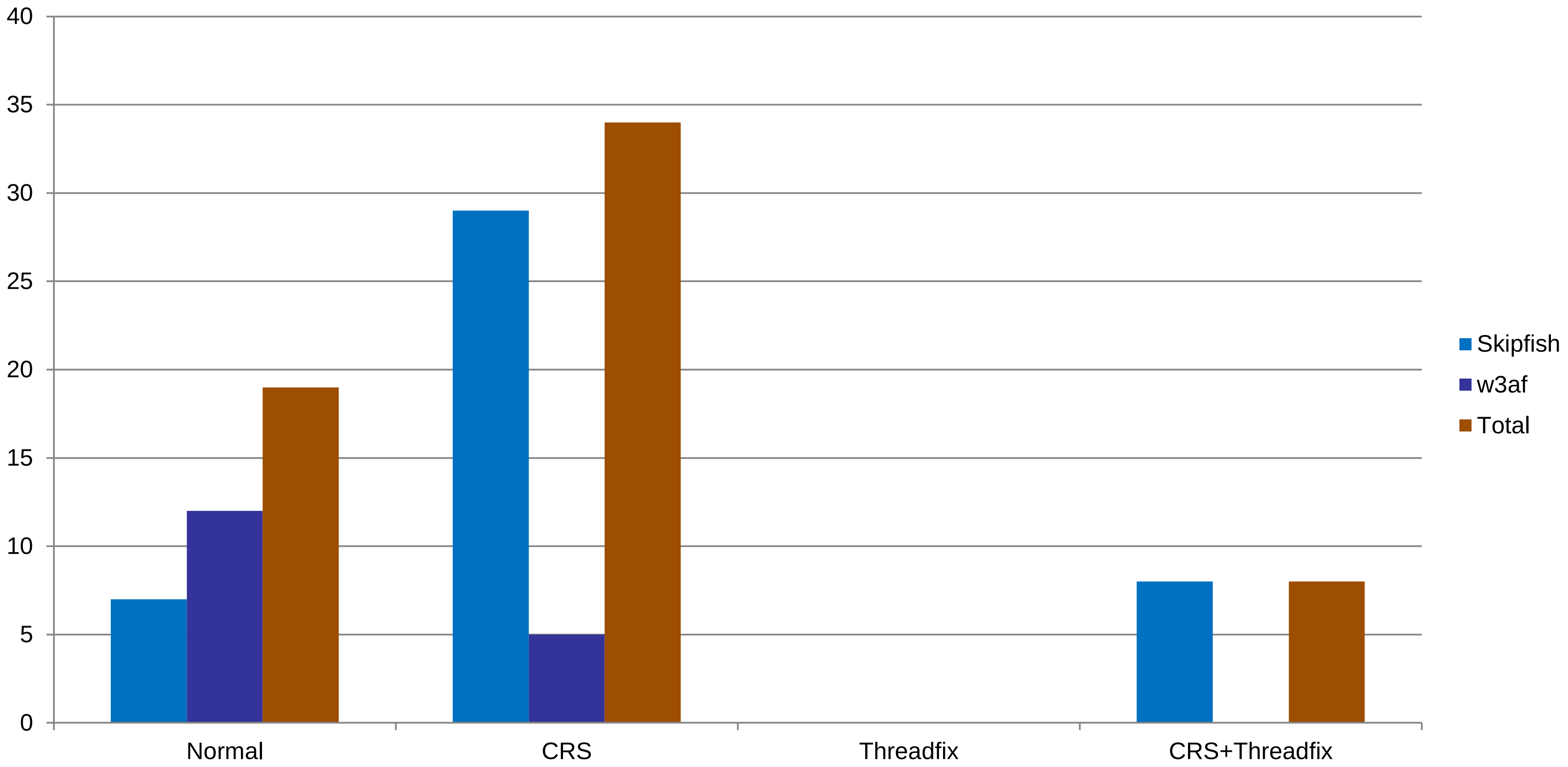
mod_security Results – Raw

Raw Total			
	Skipfish	**w3af**	**Total**
Normal	32	14	46
CRS	44	10	54
Threadfix	11	2	13
CRS+Threadfix	18	6	24

mod_security Results – All Vulnerability Types



mod_security Results – Focus on Injection



Trivia and Analysis

- IDS/IPS/WAF has an impact on the scanning process
  - *Snort breaks w3af scanning*
  - *mod_security CRS introduces some false positives into skipfish scanning*

- mod_security CRS is quite good
  - *And getting better all the time: SQL Injection Challenge*
  - *http://blog.spiderlabs.com/2011/06/announcing-the-modsecurity-sql-injection-challenge.html*

- Virtual patching appears to win for injection flaws

Where Is This Useful?

- Environments where you have little or no control over deployed code
  - *XaaS – PaaS, IaaS*
  - *99% of all corporate data centers*

- Environments where you have a large “application security debt”
  - *Actual code fixes: take time and can be hard to get on the schedule*

What Are The Problems?

- Current vulnerability data formats only allow for coarse-grained virtual patches
  - *Can lead to false blocks*

- Virtual patches likely will not stop well-informed, determined attackers
  - *See the results of the mod_security SQL Injection Challenge*

Next Steps

- MOAR DATA!!!
  - *Target applications*
  - *Live traffic*

- Develop import support for more scanner technologies

- Create virtual patch signatures for new vulnerability classes
  - *“Borrow” emerging CSRF protection from mod_security CRS?*
  - *There are limitations on what can be done but we are not there yet*

- Create virtual patches for new IDS/IPS/WAF technologies

Questions

Dan Cornell
dan@denimgroup.com
Twitter: @danielcornell

www.denimgroup.com
(210) 572-4400