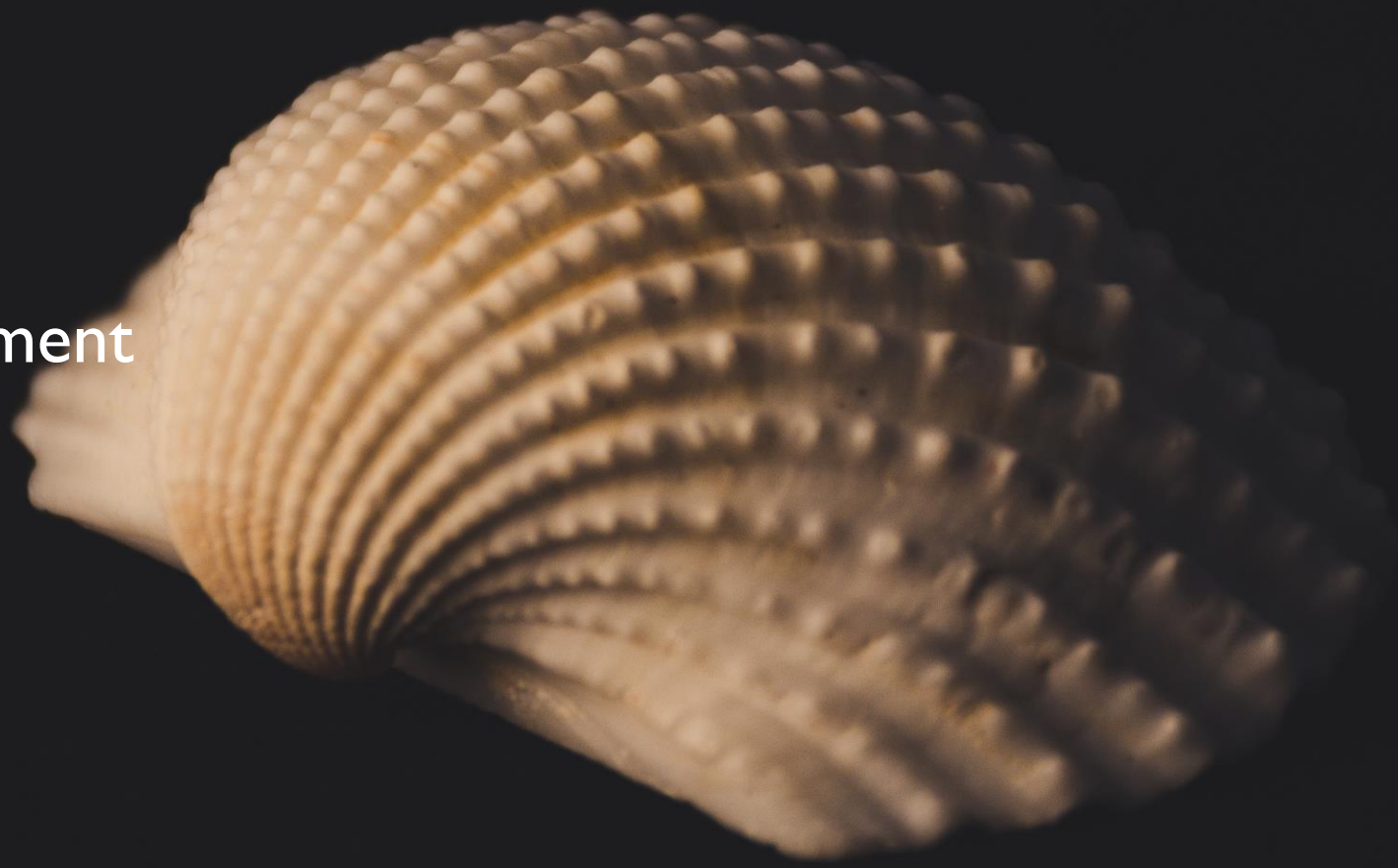
Web Shell 101

Joe Schottman
Triangle OWASP
Feb. 28, 2019

About Me

Senior Security Analyst for BB&T

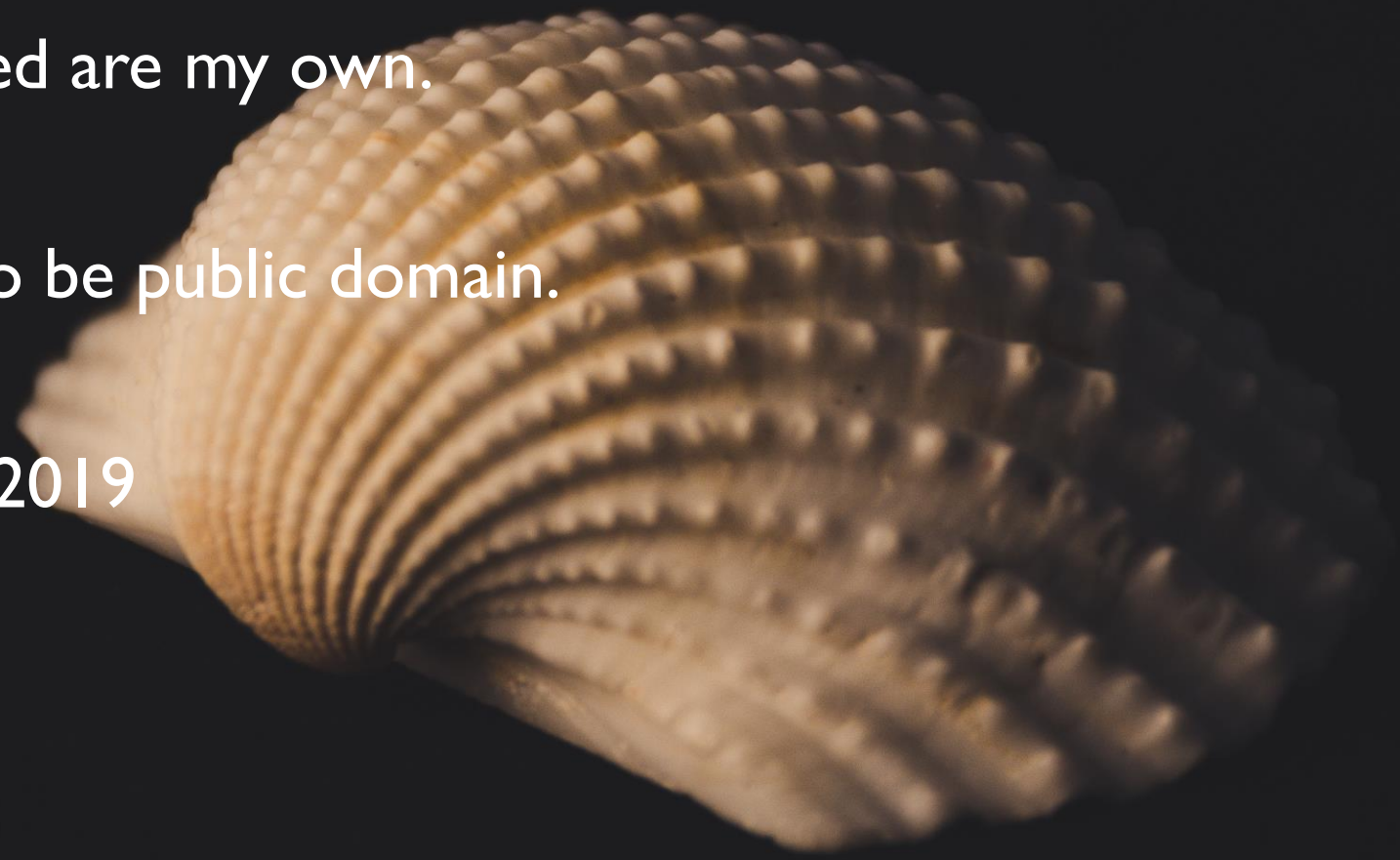
AppSec
Incident Response
Vulnerability Management
Purple Team
Web App Developer
Sysadmin
DevOps



Obligatory Disclaimer

Not speaking on behalf of BB&T or any other entity.
All opinions expressed are my own.

All images believed to be public domain.



How To Reach Me

@JoeSchottman

security@joeschottman.com

Add me on LinkedIn

Find me on local Slacks as @jschottm

Agenda

What is a Web Shell

How do Web Shells work

How can you detect them

Definitions For This Talk

Web Server == Application Server

Web Shell == Proper Noun

Buzzword Bingo

You'll get at least three words from this talk

First, a diversion

Equifax Hack

One of the biggest data breaches in history.

- 145.5 Million US citizens affected
- Unknown millions of international citizens
- PII including SSNs, drivers license numbers, credit card numbers

Equifax Hack

Initial attack vector believe to have been CVE-2017-5638.

“The Jakarta Multipart parser in Apache Struts 2 2.3.x before 2.3.32 and 2.5.x before 2.5.10.1 has incorrect exception handling and error-message generation during file-upload attempts, which allows remote attackers to execute arbitrary commands via a crafted Content-Type, Content-Disposition, or Content-Length HTTP header, as exploited in the wild in March 2017 with a Content-Type header containing a #cmd= string.”

Equifax Hack

Initial attack vector believe to have been CVE-2017-5638.

"The Jakarta Multipart parser in Apache Struts 2 2.3.x before 2.3.32 and 2.5.x before 2.5.10.1 has incorrect exception handling and error-message generation during file-upload attempts, which allows remote attackers to execute arbitrary commands via a crafted Content-Type, Content-Disposition, or Content-Length HTTP header, as exploited in the wild in March 2017 with a Content-Type header containing a #cmd= string."

Equifax Hack

Initial attack vector believe to have been CVE-2017-5638.

"The Jakarta Multipart parser in Apache Struts 2 2.3.x before 2.3.32 and 2.5.x before 2.5.10.1 has incorrect exception handling and error-message generation during file-upload attempts, which allows remote attackers to execute arbitrary commands via a crafted Content-Type, Content-Disposition, or Content-Length HTTP header, as exploited in the wild in March 2017 with a Content-Type header containing a #cmd= string."

But that's a tale for a different talk.

“Once the hackers found the vulnerability Zheng reported, they installed a simple backdoor known as a web shell. It didn't matter if Equifax fixed the vulnerability after that. The hackers had an invisible portal into the company's network.”

https://www.bloomberg.com/news/features/2017-09-29/the-equifax-hack-has-all-the-hallmarks-of-state-sponsored-pros

"Eventually the intruders installed more than 30 web shells, each on a different web address, so they could continue operating in case some were discovered."

https://www.bloomberg.com/news/features/2017-09-29/the-equifax-hack-has-all-the-hallmarks-of-state-sponsored-pros

What is a Web Shell?

A Subset Of Malware That Runs On Web Servers

Web Shells are someone else's code/application running on your web/application server for the purpose of:

- Plundering the local system
- Backdoors
- Remote Access Tools (RATs)
- Pivoting
- Persistence

Used by APT groups

APT32 (suspected Vietnamese)
APT34 (suspected Iranian)
Deep Panda (suspected Chinese)
Dragonfly (unknown)
Oil Rig (suspected Iranian)

But Also Script Kiddies

Web Shells do not require a high level of skill to install or use at the simplest level.

Many are freely available for download.

Easier to catch a script kiddie than an APT.

Someone Else's Code

Written in the languages that your web or application server supports. Some servers support more than one language.

Someone Else's Code

- PHP
- Perl
- Python
- ASP
- JavaScript
- JSP
- Ruby
- Shell Scripts
- Cold Fusion

Mostly Scripting Languages

Most web shells use scripting languages because they're the easiest to simply drop and execute on servers.

But attackers can use WAR files and the like, it just raises the complexity of the attack.

Designed To Control Your Server Via HTTP

- Remote command execution
- File access and exfiltration
- Database access
- Query system information

Imagine An Evil Web Application

Often uses programming language's built in functionality as much as possible for things such as file access.

When the language doesn't have support for functionality, will use command execution capability.

May install command shell utilities to do tasks.

Executes Just Like Your Web Applications

Runs with the same permissions and limitations as your web server.

Some groups will use escalation tools in conjunction with Web Shells but unless the application server is modified, the Web Shell executes just like your code.

This is useful for hunting for them.

Unless The Attacker Takes Countermeasures…

Files created by Web Shells will be owned by the application server.

File creation/modification time will be accurate.

Access to the Web Shell will appear in the application server logs.

Child processes such as executing cmd.exe or bash will be owned by the application server process.

Multiple Uses

Sometimes used overtly for ongoing command and control.

Sometimes used to kick off a completely different channel.

Sometimes not used at all - kept as a sleeper agent.

Hidden In Different Ways

Sometimes masquerade as legitimate file.

Sometimes appear as a random executable file.

Sometimes inserted into legitimate files.

Sometimes hidden in image files.

Very common to present itself to web users as harmless content (or even a 404 error) unless provided with a password.

Attackers may modify server to execute non-standard files as application (.htacess, apache config).

Hidden Authentication

- Subtle GET or POST variables
- User Agent
- Referrer
- IP Address
- Extra HTTP header

How Do They Get On Systems?

- File upload attacks
- Remote inclusion attacks
- SQL Injection
- Insecure admin interfaces
- Added post-exploitation of a web app vulnerability
- Dropped on a server after internal exploitation

Why Use Web Shells?

In the good old days, networks were pretty open and you could just smash and grab directly.

Now, perimeters are pretty good and there's three main ways into networks:

- Phishing
- Guessable passwords with external logins
- Application servers

How Are They Used?

Preconfigured commands
Execute code in the Application Server's Language
Execute shell script

Sometimes Surprisingly Sophisticated

China Chopper

- Windows GUI Frontend
- ASP, JSP, and Cold Fusion Applications
- Multiple capabilities: vulnerability scanning, file management, database management, virtual terminal

Questions?

Web Shells Are Never The Initial Attack

If You Find A Web Shell, You Have At Least Two Problems

Why At Least Two Problems?

- Initial vector
- The Web Shell you found
- There may be more Web Shells

As with the Equifax hack, it is very common to install multiple Web Shells to attempt to avoid evasion. If the exploited server supports more than one language, they may leverage that to avoid detection.

Let's Consider Where Web Shells Are Used

Two common frameworks for understanding the process followed by attackers:

- Cyber Kill Chain ® - created by Lockheed Martin
- ATT&CK - created by MITRE

Cyber Kill Chain

Reconnaissance
Weaponization
Delivery
Exploitation
Installation
Command and Control
Action on Objectives

Cyber Kill Chain

Reconnaissance
Weaponization
Delivery
Exploitation
Installation
Command and Control
Action on Objectives

ATT&CK

Initial access
Execution
Persistence
Privilege escalation
Defense evasion
Credential access
Discovery
Lateral movement
Collection
Exfiltration
Command and control

ATT&CK

Initial access
Execution
Persistence
Privilege escalation
Defense evasion
Credential access
Discovery
Lateral movement
Collection
Exfiltration
Command and control

ATT&CK

ATT&CK Matrix for Enterprise

The full ATT&CK Matrix below includes techniques spanning Windows, Mac, and Linux platforms and can be used to navigate through the knowledge base.

Initial Access	Execution	Persistence	Privilege Escalation	Defense Evasion	Credential Access	Discovery	Lateral Movement	Collection	Exfiltration	Command and Control
Drive-by Compromise	AppleScript	.bash_profile and .bashrc	Access Token Manipulation	Access Token Manipulation	Account Manipulation	Account Discovery	AppleScript	Audio Capture	Automated Exfiltration	Commonly Used Port
Exploit Public-Facing Application	CMSTP	Accessibility Features	Accessibility Features	BITS Jobs	Bash History	Application Window Discovery	Application Deployment Software	Automated Collection	Data Compressed	Communication Through Removable Media
Hardware Additions	Command-Line Interface	AppCert DLLs	AppCert DLLs	Binary Padding	Brute Force	Browser Bookmark Discovery	Distributed Component Object Model	Clipboard Data	Data Encrypted	Connection Proxy
Replication Through Removable Media	Control Panel Items	AppInit DLLs	AppInit DLLs	Bypass User Account Control	Credential Dumping	File and Directory Discovery	Exploitation of Remote Services	Data Staged	Data Transfer Size Limits	Custom Command and Control Protocol
Spearphishing Attachment	Dynamic Data Exchange	Application Shimming	Application Shimming	CMSTP	Credentials in Files	Network Service Scanning	Logon Scripts	Data from Information Repositories	Exfiltration Over Alternative Protocol	Custom Cryptographic Protocol
Spearphishing Link	Execution through API	Authentication Package	Bypass User Account Control	Clear Command History	Credentials in Registry	Network Share Discovery	Pass the Hash	Data from Local System	Exfiltration Over Command and Control Channel	Data Encoding
Spearphishing via Service	Execution through Module Load	BITS Jobs	DLL Search Order Hijacking	Code Signing	Exploitation for Credential Access	Password Policy Discovery	Pass the Ticket	Data from Network Shared Drive	Exfiltration Over Other Network Medium	Data Obfuscation
Supply Chain Compromise	Exploitation for Client Execution	Bootkit	Dylib Hijacking	Component Firmware	Forced Authentication	Peripheral Device Discovery	Remote Desktop Protocol	Data from Removable Media	Exfiltration Over Physical Medium	Domain Fronting
Trusted Relationship	Graphical User Interface	Browser Extensions	Exploitation for Privilege Escalation	Component Object Model Hijacking	Hooking	Permission Groups Discovery	Remote File Copy	Email Collection	Scheduled Transfer	Fallback Channels
Valid Accounts	InstallUtil	Change Default File Association	Extra Window Memory Injection	Control Panel Items	Input Capture	Process Discovery	Remote Services	Input Capture		Multi-Stage Channels
	LSASS Driver	Component Firmware	File System Permissions Weakness	DCShadow	Input Prompt	Query Registry	Replication Through Removable Media	Man in the Browser		Multi-hop Proxy
	Launchctl	Component Object Model Hijacking	Hooking	DLL Search Order Hijacking	Kerberoasting	Remote System Discovery	SSH Hijacking	Screen Capture		Multiband Communication
	Local Job Scheduling	Create Account	Image File Execution Options Injection	DLL Side-Loading	Keychain	Security Software Discovery	Shared Webroot	Video Capture		Multilayer Encryption
	Mshta	DLL Search Order Hijacking	Launch Daemon	Deobfuscate/Decode Files or Information	LLMNR/NBT-NS Poisoning	System Information Discovery	Taint Shared Content			Port Knocking
	PowerShell	Dylib Hijacking	New Service	Disabling Security Tools	Network Sniffing	System Network Configuration Discovery	Third-party Software			Remote Access Tools
	Regsvcs/Regasm	External Remote Services	Path Interception	Exploitation for Defense Evasion	Password Filter DLL	System Network Connections Discovery	Windows Admin Shares			Remote File Copy
	Regsvr32	File System Permissions Weakness	Plist Modification	Extra Window Memory Injection	Private Keys	System Owner/User Discovery	Windows Remote Management			Standard Application Layer Protocol
	Rundll32	Hidden Files and Directories	Port Monitors	File Deletion	Securityd Memory	System Service Discovery				Standard Cryptographic Protocol
	Scheduled Task	Hooking	Process Injection	File System Logical Offsets	Two-Factor Authentication Interception	System Time Discovery				Standard Non-Application Layer Protocol
	Scripting	Hypervisor	SID-History Injection	Gatekeeper Bypass						Uncommonly Used Port
	Service Execution	Image File Execution Options Injection	Scheduled Task	HISTCONTROL						Web Service

You’re Not Dealing With An Attack, You’re Dealing With An Incident

Time To Engage Incident Reponse

If you find a Web Shell and your company has an IR team and/or policy, you need to engage them.

- It may be necessary to take steps to preserve forensic evidence
- Sleuthing may tip off the attackers
- There may be legal requirements to do so
- Cleanup requires finding the initial attack vector and closing it and finding and removal all web shells in your network

A Funny Aside

"This module exploits a lack of authentication in the shell developed by v0pCr3w and is widely reused in automated RFI payloads. This module takes advantage of the shell's various methods to execute commands."

-Metasploit v0pcr3w_exec module

Metasploit

Modular exploit framework

Combines different exploits, payloads, and tools

apache_activemq_upload_jsp module
Meterpreter via web_delivery

Questions?

Detecting Web Shells

Strategies

Antivirus/Antimalware
File Integrity Monitoring
File System
Log Files
Network Traffic
Endpoint Anomaly Detection

Antivirus/Antimalware

AV can find some Web Shells.

Very often simple hash matching or looking for a few specific lines of an interpreted language.

Easy to fool by changing the code to get a different hash or change those key lines.

It's still better than nothing - if you find new examples of Web Shells and your AV doesn't flag them, submit samples to your provider.

Antivirus/Antimalware

AV will also complicate Web Shell research by deleting your samples. This includes some cloud providers' storage.

If you're doing research on a corporate machine, make sure you have permission to do so, in case you light up the A/V dashboard.

You Do Get Permission Before Doing Research, Right?

VirusTotal

Uploading a suspicious file to VirusTotal may tell you if multiple A/V vendors identify it as a Web Shell, but I don't necessarily recommend it.

Once you identify a suspicious file, it should be pretty easy to determine if it's malicious or not.

Attackers monitor VirusTotal to see if their tools have been found.

Uploading your company's IP to a third party is … non-ideal.

File Integrity Monitoring

If you have a robust DevOps toolkit, it may be capable of detecting new or modified files it doesn't know about.

You may have a specific FIM tool installed that can do the same.

You can generate hashes of all files from a known clean install and look for changes.

In An Ideal World...

Your application would not allow uploads.

If you must allow them:

- restrict file types
- scan with A/V tool
- store uploaded files outside of the web root

Also In An Ideal World...

Your web root would be read-only.

It's hard to add files to something that's immutable.

Many Container strategies help with this.

File Integrity Monitoring Testing

Conduct tests - put new files on your web servers and see how long detection takes.

1/10/60

- One minute to detect
- Ten minutes to investigate
- Sixty minutes to remediate

File System Techniques

Search for hashes of known Web Shells.

Search for unusual file modification times.

Search for unusual file locations.

Utilize endpoint security products, threat hunting tools, or shell scripts.

Limited use if the attacker makes each Web Shell unique and obfuscated.

File System Forensics

If you locate a Web Shell through a different means, you will need to conduct a forensic analysis for deleted artifacts.

Dirty Word List

Dirty word list is a term from file IR.

Compile a list of suspicious words to look for in files such as exec() or shell_exec() in PHP.

Ideally be able to tell when a file has multiple suspicious words.

As you find new Web Shells, add distinctive things from them to dirty word list.

Let's Look At Some Examples

Laudanum - open source collection of web shells created by Secure Ideas.

https://sourceforge.net/projects/laudanum/

shell.php

```
19 *** This file provides shell access to the system. It is built based on the 2.1
20 *** version of PHPShell which is Copyright (C) 2000-2005 Martin Geisler

81 <title>Laudanum PHP Shell Access</title>
```

shell.php

```
246 $p = proc_open($command,
247 array(1 => array('pipe', 'w'),
248 2 => array('pipe', 'w')),
249 $io);
```

shell.aspx

```
80 // create the ProcessStartInfo using "cmd" as the program to be run, and "/c "
as the parameters.
81 // "/c" tells cmd that we want it to execute the command that follows, and
exit.
82 System.Diagnostics.ProcessStartInfo procStartInfo = new
System.Diagnostics.ProcessStartInfo("cmd", "/c " + Request.Form["c"]);
```

Look For Encrypted Or Encoded Content

Attackers often encode or encrypt the majority of the code to hide it from text string searches.

Look for unusual amounts of Base64 and other encodings in executable files.

Look for executable files with high amounts of entropy.

Base64 Example

```
$p = proc_open($command,
                                array(1 =>
array('pipe', 'w'),
                                        2 =>
array('pipe', 'w')),
                                $io);
```

```
JHAgPSBwcm9jX29wZW4oJGNvbW1hb
mQsCiAgICAgICAgICAgICAgICAgIC
AgICAgICAgIGFycmF5KDEgPT4gYXJ
yYXkoJ3BpcGUnLCAndycpLAogICAg
ICAgICAgICAgICAgICAgICAgICAgI
CAgICAgICAyID0+IGFycmF5KCdwaX
BlJywgJ3cnKSksCiAgICAgICAgICA
gICAgICAgICAgICAgICRpbyk7
```

AES Example

72	95	33	10	6b	df	a8	0b	2d	a9	b1	3c	b7	e7	4f	eb
f7	c4	12	a3	e5	6c	f5	5a	67	1f	52	0e	bc	b2	1c	0b
25	4a	76	24	63	4c	96	14	74	c0	93	56	66	11	c1	b4
71	e9	09	81	39	f7	20	83	7c	f7	de	bb	9a	d8	c4	cf
a9	99	01	63	96	0f	78	80	13	eb	a1	bf	c5	00	12	83
d1	8f	65	ca	ed	b9	62	03	f9	31	4e	d0	20	28	9c	36
25	4a	76	24	63	4c	96	14	74	c0	93	56	66	11	c1	b4
21	86	22	b5	00	b5	0f	71	34	bf	5d	3c	01	66	ce	d7
0d	88	fe	1d	94	14	cb	11	5f	33	d1	a8	e1	66	e0	af
f7	7b	22	e3	3d	eb	24	38	ba	cb	4e	5b	ef	f8	81	4c
b3	8e	6e	90	3d	a5	bf	81	fc	e7	ec	69	e1	05	9e	27
e9	61	6e	75	5d	a9	5c	11	4a	b5	52	20	3a	3b	a2	7d

Obfuscated Code Still Has To Execute

eval() or assert() commands needed to execute hidden code so look for them.

Deserialization libraries may also be used.

YARA

Malware research and detection tool.

Searches based on text and binary patterns.

Rules for Web Shell detection are available - don't reinvent the wheel.

I'll leave it to you to figure out what YARA stands for.

OpenIOC is similar.

YARA Example

```
rule Weevely_Webshell {
        meta:
                description = "Weevely Webshell - Generic Rule - heavily scrambled tiny web
shell"
                author = "Florian Roth"
                reference = "http://www.ehacking.net/2014/12/weevely-php-stealth-web-backdoor-
kali.html"
                date = "2014/12/14"
                score = 60
        strings:
                $php = "<?php" ascii
                $s0 = /\$[a-z]{4} = \$[a-z]{4}\("[a-z][a-z]?",[\s]?"",[\s]?"/ ascii
                $s1 = /\$[a-z]{4} = str_replace\("[a-z][a-z]?","","/ ascii
                $s2 = /\$[a-z]{4}\.\$[a-z]{4}\.\$[a-z]{4}\.\$[a-z]{4}\)\)\); \$[a-z]{4}\(\);/
ascii
                $s4 = /\$[a-z]{4}="[a-zA-Z0-9]{70}/ ascii
        condition:
                $php at 0 and all of ($s*) and filesize > 570 and filesize < 800
}
```

Don’t Forget The Database

Some Web Shells use the database to store large amounts of data or to hide executable code.

Look for new and unexpected databases or tables.

If your tooling supports it, look for unusual types of data within tables (which is also good for finding application bugs).

Knowing the language is key to understanding maliciousness.

Log Files

Look for the appearance of a new URI that you've never seen before.

This may duplicate file system monitoring but is helpful if you don't have access to the file system and also is important with some web frameworks.

New GET variables can also be an indicator but are much harder to create rules for.

Log Files

URIs that only a single (or handful) of IPs access.

Especially if they're international, VPN/proxy, or Tor IPs!

Log Files

New 400 or 500 errors may be indicative of an uploaded Web Shell failing to work.

A 400 indicates they called it incorrectly.

A 500 indicates something went wrong on the server (e.g. the Web Shell uses a PHP function that you banned in php.ini).

Log File Smells

All of these may be completely normal - you have to know your environment.

- Base64 encoded GET variables
- Repeated visits to the same URI with different request and response lengths
- IP addresses that never load JavaScript, CSS, images, fonts, etc.
- Obvious command shell activity
- POSTs to file types that should not get them such as images
- Unusual spikes of activity
- Unusual user agents
- Lack of referrer header

Network Analysis

A web server begins connecting to an unexpected external site.

A web server begins connecting to an unexpected *internal* site.

Web Application Firewalls (WAF) or IDS/IPS tools with network visibility will detect some Web Shells.

Endpoint Anomaly Detection

Unusual child process for web server.

Very high CPU utilization for web server process or unusual run time.

Sudden increase in disk utilization.

Sudden increase in temp folder directories.

Log file tampering.

Defensive Posture

- Follow best practices – avoid functions that can be used to execute code, write files outside of Web Root, don't use remote file inclusion, etc.
- Know normal – when you must go outside of best practices, document it so you know where to expect it.
- Make the Web Root immutable if possible.

Thanks! Questions?

You can find me at:
@JoeSchottman
security@joeschottman.com