Name: Jun Wang 000791814

Lab 6: Attacking Data Stores and Back-End Components

Web Application Security

Mark:

Bonus:

Contents

[Lab Outcome 3](#_Toc23786622)

[Background Reading 3](#_Toc23786623)

[Architecture Diagram 3](#_Toc23786624)

[Required Hardware/Software 3](#_Toc23786625)

[Introduction 4](#_Toc23786626)

[1.0 Numeric SQL Injection 4](#_Toc23786627)

[2.0 String SQL Injection 5](#_Toc23786628)

[3.0 SQL Injection 6](#_Toc23786629)

[4.0 Blind Numeric SQL Injection 10](#_Toc23786630)

[5.0 Blind String SQL Injection 11](#_Toc23786631)

[6.0 Log Spoofing 13](#_Toc23786632)

[7.0 Command Injection 13](#_Toc23786633)

[8.0 Create a SOAP Request 14](#_Toc23786634)

[9.0 WSDL Scanning 17](#_Toc23786635)

[10.0 Sign-Off – Lab 6: Attacking Data Stores and Back-End Components 18](#_Toc23786636)

Web Application Security

Lab 6: Attacking Data Stores and Back-End Components

# Lab Outcome

Exploit the web application data stores and back-end components using various injection techniques.

Background Reading

Read the textbook sections listed in the Course Schedule.

Architecture Diagram

|  |
| --- |
|  |

Required Hardware/Software

* VM Ubuntu 18.04 – 2 CPU, 4GB Ram, 20 GB hard disk
  + Docker
    - WebGoat v7.1
    - DVWA (Dawn Vulnerable Web App)
* VM Kali
  + Burp or other Web proxy (scanner)

# Introduction

By attacking the data store, you can access web application data, including user accounts, permissions, configuration settings and everything else stored on a system.

In addition, web applications are often the access control to back-end components, such as filesystems and the operating system, as well as any networked resources like web servers and mail servers. Attacking the back-end can provide access to sensitive data and functionality.

# 1.0 Numeric SQL Injection

In WebGoat, complete the **Injection Flaws > Numeric SQL Injection** lesson.

Step1> set up proxy port 8084 on ZAP and Firefox

Step2> select Columbia on Firefox and click on Go!

Step3> intercept on ZAP, find the post request.

Step4> right click the post request, click on Open/Resend with Request Editor

Modify station=101 to station=101 OR 1=1 then resend the request

Step 5 > refresh the Firefox.

|  |
| --- |
| Selection_002 |
| Selection_003 |
| Selection_008 |
| Selection_004 |

# 2.0 String SQL Injection

In WebGoat, complete the **Injection Flaws > String SQL Injection** lesson.

Step1> set up proxy port 8084 on ZAP and Firefox

Step2> select Columbia on Firefox and click on Go!

Step3> intercept on ZAP, find the post request.

Step4> right click the post request, click on Open/Resend with Request Editor

Modify account\_name=Jun to account\_name=Jun’ OR ‘A’=’A then resend the request.

Step 5 > refresh the Firefox.

|  |
| --- |
| Selection_010 |
| Selection_011 |
| Selection_012 |
| Selection_016 |
| Selection_017 |

# 3.0 SQL Injection

1. In WebGoat, complete the **Injection Flaws > SQL Injection Stage 1** lesson.

Step1> choose username Nevillie

Step2> inspect the elements of password, delete maxlength=”8”

Step3> input the password as: tom’ OR ‘1’=’1

|  |
| --- |
| Selection_031 |
| Selection_029 |
| Selection_030 |
|  |

1. In WebGoat, complete the **Injection Flaws > SQL Injection Stage 2** lesson.

|  |
| --- |
| Source Code changes optional, 3 points for valid evidence |
|  |
|  |
|  |

1. In WebGoat, complete the **Injection Flaws > SQL Injection Stage 3** lesson.

Step1> login as Larry passwd: larry

Step2> inspect elements employee list, modify the value=’101’ to value=’101 or 1=1 order by salary desc’

Step3> click on viewprofile

|  |
| --- |
| Selection_033 |
| Selection_034 |
| Selection_038 |
| Selection_039 |
| Selection_040 |

1. In WebGoat, complete the **Injection Flaws > SQL Injection Stage 4** lesson.

|  |
| --- |
| Source Code changes optional, 3 points for valid evidence. |
|  |
|  |
|  |

# 4.0 Blind Numeric SQL Injection

In WebGoat, complete the **Injection Flaws > Blind Numeric SQL Injection** lesson.

Step1> account number 101 and Go! , server returns the “ account number is valid”

It means account\_number value=101 is true.

Step2> 101 AND ((SELECT pin FROM pins WHERE cc\_number=’11112222333344445555’)>3000);

Search from table pins to find the pin value on the row cc\_number=’1111...’

We got response: invalid account number. It means the pin>3000 is FALSE. So the pin value is less than or equals to 3000.

Step3> 101 AND ((SELECT pin FROM pins WHERE cc\_number=’11112222333344445555’)>2000);

Search from table pins to find the pin value on the row cc\_number=’1111...’

We got response: account number is valid. It means the pin>2000 is TRUE. So the pin value in the range of 2000-3000.

Keep changing the number to narrow down the range of pin.

|  |
| --- |
| **Selection_042** |
| Selection_047 |
| Selection_044 |
| Selection_048 |
| Selection_049 |

# 5.0 Blind String SQL Injection

In WebGoat, complete the **Injection Flaws > Blind String SQL Injection** lesson.

Step1> inspect elements of the account number, Since we already know that the account\_number value=101 is True. We need find the first letter of name field. So, I use the following code :

101 AND (SUBSTRING((SELECT name FROM pins WHERE cc\_number=’4321432143214321’),1 ,1)>’M’);

To test if the first letter is greater than M(13th in alphabets). I got the response as the Invalid account number, which means the first letter is less than ‘M’.

Step2> Test the first letter of name is > ‘G’, the response is : account number is valid. It means the expression(SUBSTRING((SELECT name FROM pins WHERE cc\_number=’4321432143214321’),1 ,1)>’G’) is TRUE.

Follow this way and keep changing the range, I found the first letter is ‘J’

Step3> To test the 2nd letter of the name value, use the same method and change the code to

101 AND (SUBSTRING((SELECT name FROM pins WHERE cc\_number=’4321432143214321’),2 ,1)>’m’);

Step4> Repeat the same routine, I found the account name is ‘Jill’

|  |
| --- |
| Insert evidence here |
|  |
| Selection_052 |
|  |
| Selection_054 |
| Selection_056 |
|  |

# 6.0 Log Spoofing

In WebGoat, complete the **Injection Flaws > Log Spoofing** lesson.

admin%0aLogin succeeded for username: admin

|  |
| --- |
| Selection_062 |
| Selection_061 |
|  |
|  |

# 7.0 Command Injection

In WebGoat, complete the **Injection Flaws > Command Injection** lesson.

|  |
| --- |
| Insert evidence here |
|  |

# 8.0 Create a SOAP Request

In WebGoat, complete the **Web Services > Create a SOAP Request** lesson.

|  |
| --- |
| Selection_065 |
| Selection_066 |
| Selection_067 |
| Selection_068Selection_068 |
| Selection_069 |
| Selection_070 |
| Selection_032 |

# 9.0 WSDL Scanning

In WebGoat, complete the **Web Services > WSDL Scanning** lesson.

We can see from reading the WSDL File, 4 options are defined: getLastName , getFirstName, getLoginCount and getCreditCard. When I choose option Last Name, it calls the option of getLastName.

Step1> inspect elements of the form.

Step2> right click on one option, choose duplicate the node

Step3> modify the node into:

<option value=”getCreditCard”> Credit Card</option>

Step4> choose option Credit Card and Submit

|  |
| --- |
| Selection_075 |
| Selection_076 |
|  |
| Selection_073 |
| Selection_074 |

# 10.0 Sign-Off – Lab 6: Attacking Data Stores and Back-End Components

Detach this page and submit it to your instructor to indicate you have completed all sections.

Name:

Student ID:

|  |  |
| --- | --- |
| **Section** | **Instructor Initials** |
| 1.0 Numeric SQL Injection |  |
| 2.0 String SQL Injection |  |
| 3.0 SQL Injection – 1 |  |
| 3.0 SQL Injection – 2 |  |
| 3.0 SQL Injection – 3 |  |
| 3.0 SQL Injection – 4 |  |
| 4.0 Blind Numeric SQL Injection |  |
| 5.0 Blind String SQL Injection |  |
| 6.0 Log Spoofing |  |
| 7.0 Command Injection |  |
| 8.0 Create a SOAP Request |  |
| 9.0 WSDL Scanning |  |