# *Systems III (420-E31-HR)*

# *Lab 11 – CyberSecurity – Hands-on with WebGoat*

Date assigned: Wednesday, November 15, 2017

Date due: Wednesday, November 15, 2017**, 11:50 a.m.**

**Learning Objectives**

Upon successful completion of this lab exercise, the student will be able to:

* Recognize common cybersecurity vulnerabilities
* Learn how to use basic snooping tools
* Demonstrate cybersecurity vulnerabilities and attacks
* Identify remedies to cybersecurity vulnerabilities demonstrated

To do:

Save this document as a Word document named **YourUserName\_E31\_L11\_CyberGoat.docx** and submit it to moodle.

**Part A – Installation and tools**

1. Log into WebGoat: <http://csdev.cegep-heritage.qc.ca:8080/WebGoat/>
   1. Listed below is your username AND password, they are the same.

|  |  |
| --- | --- |
| Beeftink.Alexander@cegep-heritage.qc.ca | guest01 |
| Ha.Andrew@cegep-heritage.qc.ca | guest02 |
| Stewart.Alexandre@cegep-heritage.qc.ca | guest03 |
| Yadollahibastani.Amirreza@cegep-heritage.qc.ca | guest04 |
| Walker.Brae@cegep-heritage.qc.ca | guest05 |
| Berube.Cody@cegep-heritage.qc.ca | guest06 |
| Mercier.Guillaume@cegep-heritage.qc.ca | guest07 |
| Mills.Isaac@cegep-heritage.qc.ca | guest08 |
| Clarke.Jeremy@cegep-heritage.qc.ca | guest09 |
| Thibodeau.Louis@cegep-heritage.qc.ca | guest10 |
| Lurette.Martin@cegep-heritage.qc.ca | guest11 |
| Haley.Maxwell@cegep-heritage.qc.ca | guest12 |
| Dumaresq.Philip@cegep-heritage.qc.ca | guest13 |
| Phlek.Sereil-Vann@cegep-heritage.qc.ca | guest14 |
| Plouffe.Thomas@cegep-heritage.qc.ca | guest15 |
| McLean.Troy@cegep-heritage.qc.ca | guest16 |
| Zou.WeiHao@cegep-heritage.qc.ca | guest17 |
| Bridge.Zachary@cegep-heritage.qc.ca | guest18 |

1. Install the “Tamper Data” plugin in FireFox. (You will be doing the rest of this lab using FireFox)
   1. Install the Tamper Data Plugin.
   2. You may have to restart FF to enable the plugin
   3. Start up the Tamper Data tool. It should be visible from Firefox’s Menubar ->Tools\_> Tamper Data. It shows all the message exchanges to/from Firefox.
   4. Browse to a couple sites and ensure the Tamper Data shows message exchanges as you pull up new sites.
   5. With only the WebGoat localhost page up, watch the Tamper Data screens. If you do nothing on the browser, then there shouldn’t be new messages generated. If there are periodic messages, then you have extra plugins in FF that need to be disabled as they will generate too much noise for you to do the exercises. (i.e. trying to find out which messages to hack will be easier if there are less messages)
   6. An interesting article about Tamper Data can be found [here](http://netsecurity.about.com/od/hackertools/a/What-Hackers-Do-Not-Want-You-To-Know-About-The-Tamper-Data-Firefox-Add-on.htm)

**Part B – Let the hacking begin**

Log onto WebGoat and perform the following exercises that involve intercept and tampering. Before looking up the solutions or getting help, please attempt each of these on your own.

1. On WebGoat, do the exercise: General->HTTP Basics
   1. Using Tamper Data, find the message where you send your name to from the web page to the server.
   2. Provide answers below:

|  |  |
| --- | --- |
| HTTP Message Method (Get/Post) | POST  9:11:17.554[91ms][total 91ms] Status: 200[OK]  POST http://csdev.cegep-heritage.qc.ca:8080/WebGoat/attack?Screen=93&menu=100 Load Flags[LOAD\_BACKGROUND LOAD\_BYPASS\_LOCAL\_CACHE ] Content Size[1036] Mime Type[text/html]  Request Headers:  Host[csdev.cegep-heritage.qc.ca:8080]  User-Agent[Mozilla/5.0 (Windows NT 6.1; Win64; x64; rv:56.0) Gecko/20100101 Firefox/56.0]  Accept[\*/\*]  Accept-Language[en-US,en;q=0.5]  Accept-Encoding[gzip, deflate]  Content-Type[application/x-www-form-urlencoded; charset=UTF-8]  X-Requested-With[XMLHttpRequest]  Referer[http://csdev.cegep-heritage.qc.ca:8080/WebGoat/start.mvc]  Content-Length[24]  Cookie[JSESSIONID=1B03EE642720D5D43EFC44C98780528F; \_\_utma=165456813.1442289471.1504018848.1510671017.1510678937.9; \_\_utmz=165456813.1504018848.1.1.utmcsr=(direct)|utmccn=(direct)|utmcmd=(none)]  Connection[keep-alive]  Post Data:  person[Philip]  SUBMIT[Go!]  Response Headers:  Server[Apache-Coyote/1.1]  Content-Type[text/html;charset=ISO-8859-1]  Content-Length[1036]  Date[Wed, 15 Nov 2017 14:11:17 GMT] |
| Field containing your name | person |
| POSTDATA value | POSTDATA=person=Philip&SUBMIT=Go! |

Congratulations, you’ve demonstrated packet sniffing.

Anything you send from the web client to a web server can be visible

1. On WebGoat, do the exercise: Insecure Communication -> Insecure Login
   1. Provide answers below:

|  |  |
| --- | --- |
| HTTP Message Method (Get/Post) | 9:12:42.283[89ms][total 89ms] Status: 200[OK]  POST http://csdev.cegep-heritage.qc.ca:8080/WebGoat/attack?Screen=145&menu=1300 Load Flags[LOAD\_BACKGROUND LOAD\_BYPASS\_LOCAL\_CACHE ] Content Size[2770] Mime Type[text/html]  Request Headers:  Host[csdev.cegep-heritage.qc.ca:8080]  User-Agent[Mozilla/5.0 (Windows NT 6.1; Win64; x64; rv:56.0) Gecko/20100101 Firefox/56.0]  Accept[\*/\*]  Accept-Language[en-US,en;q=0.5]  Accept-Encoding[gzip, deflate]  Content-Type[application/x-www-form-urlencoded; charset=UTF-8]  X-Requested-With[XMLHttpRequest]  Referer[http://csdev.cegep-heritage.qc.ca:8080/WebGoat/start.mvc]  Content-Length[47]  Cookie[JSESSIONID=1B03EE642720D5D43EFC44C98780528F; \_\_utma=165456813.1442289471.1504018848.1510671017.1510678937.9; \_\_utmz=165456813.1504018848.1.1.utmcsr=(direct)|utmccn=(direct)|utmcmd=(none)]  Connection[keep-alive]  Post Data:  clear\_user[Jack]  clear\_pass[sniffy]  Submit[Submit]  Response Headers:  Server[Apache-Coyote/1.1]  Content-Type[text/html;charset=ISO-8859-1]  Content-Length[2770]  Date[Wed, 15 Nov 2017 14:12:42 GMT] |
| Password | Sniffy |

Note: once you get to the password, you may stop this exercise. We don’t have the setup to use HTTPS

You just intercepted a user’s password. Users typically use the same usernames and passwords. Once a username/password set is known, it can be applied to goggle, apple itunes/icloud, bank accounts, credit card accounts, or anywhere that you are a registered online user.

Explain below, what is the vulnerability and how to solve it?

It is a confidentiality vulnerability. Only send the data to the server once it’s already been encrypted. Don’t send the raw text.

1. On WebGoat, do the exercise: parameter tampering -> exploit hidden fields
   1. Using Tamper Data, purchase a tv for $1 instead of $2999.99. You will have to use the “Start Tamper” function in Tamper Data to do this
   2. Provide answers below:

|  |  |
| --- | --- |
| HTTP Message Method (Get/Post) | 9:17:59.227[42045ms][total 42068ms] Status: 200[OK]  POST http://csdev.cegep-heritage.qc.ca:8080/WebGoat/attack?Screen=111&menu=1700 Load Flags[LOAD\_BACKGROUND LOAD\_BYPASS\_LOCAL\_CACHE ] Content Size[1482] Mime Type[text/html]  Request Headers:  Host[csdev.cegep-heritage.qc.ca:8080]  User-Agent[Mozilla/5.0 (Windows NT 6.1; Win64; x64; rv:56.0) Gecko/20100101 Firefox/56.0]  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]  Content-Type[application/x-www-form-urlencoded]  X-Requested-With[XMLHttpRequest]  Referer[http://csdev.cegep-heritage.qc.ca:8080/WebGoat/start.mvc]  Content-Length[19]  Cookie[JSESSIONID=1B03EE642720D5D43EFC44C98780528F; \_\_utma=165456813.1442289471.1504018848.1510671017.1510678937.9; \_\_utmz=165456813.1504018848.1.1.utmcsr=(direct)|utmccn=(direct)|utmcmd=(none)]  Connection[keep-alive]  Upgrade-Insecure-Requests[1]  Post Data:  POST\_DATA[QTY=1&SUBMIT=Purchase&Price=1]  Response Headers:  Server[Apache-Coyote/1.1]  Content-Type[text/html;charset=ISO-8859-1]  Content-Length[1482]  Date[Wed, 15 Nov 2017 14:18:40 GMT] |
| POSTDATA value | POSTDATA =QTY=1&SUBMIT=Purchase&Price=1 |

You’ve just paid $1 for a product worth $2999.99

Explain below, what is the vulnerability and how to solve it?

The vulnerability below is a parameter manipulation problem. You can solve this by saving the price on the server side and not deciding the price by pulling from the front end.

1. Research and answer
   1. You’ve been using Tamper Data to intercept and modify messages between a web client and server. As it is built right into FireFox, it is able to view across HTTP and HTTPS connections. Investigate WebScarab and how web proxys work
   2. Answer:
      1. What steps would be needed to change a browser, like firefox to use a proxy?

You would enable the Firefox proxy settings by going to Tools > Options > Proxy Settings > and set firefox to use a proxy

* + 1. Would you need a proxy to intercept and sniff messages?

No, you would not need a proxy to just intercept and sniff messages, you can do that with all sorts of software just by looking at the network traffic.

* + 1. Would you need a proxy to modify messages?

Yes, you would need a proxy for this, since you need to have yourself sitting between the web client and the server. This means you can sit anywhere on your client, anywhere in the networks, or on your server before the information gets received.

* + 1. Would proxies work across HTTPS connections?

Yes, they can work across HTTPS, WebScarab does.

* + 1. How is Tamper Data able to decode HTTPS?

**Part C – Injection Flaws**

Log onto WebGoat and perform the following exercises that involve injection flaws.

1. On WebGoat, do the exercise: Injection Flaws->Modify Data with SQL injection
   1. Increase the salary of jsmith from $10000 to $30000. Hint to do this, you have to inject the following SQL: UPDATE salaries SET salary=30000 WHERE userid=’jsmith’
   2. Provide answers below:

|  |  |
| --- | --- |
| Show the SQL from a typical, well-behaved user | SELECT \* FROM SALARIES WHERE userid=’jsmith’; |
| Show the resultant SQL from your hack | SELECT \* FROM SALARIES WHERE userid=’jsmith’; UPDATE salaries SET salary=30000 WHERE userid=’jsmith’; --‘; |

Explain below, what is the vulnerability and how to solve it?

This is a SQL injection attack. It can be solved by escaping the data that’s passed into the input field. For example, something simple would be putting backslashes in front of any special SQL characters so they’ll be interpreted as a literal string. Although this is easy to get around.

1. On WebGoat, do the exercise: Cross Site Scripting -> LAB: Cross Site Scripting (stop before doing Stage 2).
   1. Log in as Tom and inject javascript into the street address: <script language=”javascript” type=”text/javascript”>alert(“The Heritage Hedgehog strikes again!”);</script>
   2. Login in as Jerry and view Tom’s profile. Answer below:

What happens when Jerry views Tom’s profile?

An alert box is comes up saying “The Heritage Hedgehog strikes again”

What is the vulnerability and how to solve it?

This can be solved in a similar way to the SQL injection attack. You can solve it by replacing all instances of a < or > character with the equivalent Unicode escaped character.

**Part D – Concurrency flaw**

Log onto WebGoat and perform the following exercises that involve a concurrency flaw

1. On WebGoat, do the exercise: Concurrency->Shopping Cart Concurrency Flaw
   1. Buy the Sony Vaio for the price of the hard drive
   2. Provide answers below:

Explain below, what is the vulnerability and how to solve it?

This is a concurrency issue, because you’re overriding the object storing your purchase, but the price is still pulled from the front end, you can manipulate how much you’re paying for everything.

**Part E – AJAX**

Log onto WebGoat and perform the following exercises that involve an AJAX security flaw

1. On WebGoat, do the exercise: AJAX Security->Dangerous Use of Eval
   1. The 3 digit access code is eval’ed. Figure out how to display the document cookie. (document.cookie)

Explain below, what is the vulnerability and how to solve it?

This is an eval bug. In the eval function, any code that is entered into it is going to be rendered as JavaScript code. So if I am to alert the document.cookie, then it would output that to me. A simpler way of doing this though is just typing it directly into the JavaScript console and getting the answer from there.

**Part F: Free Hunt**

1. Help me build up more labs. Find another exercise on WebGoat that you can solve that hasn’t been covered in the lab yet. What exercise did you do?

Code quality > Discover clues in the HTML

Concurrency > Thread safety problems

Injection flaws > Numeric SQL inject

Injection flaws > String SQL injection

Injection flaws > Add data with SQL injection

Injection flaws > Database backdoors

1. Explain below, what is the vulnerability and how to solve it?

This is a privacy vulnerability – the admin username and password are stored in a comment in the HTML.

This was solved by having latency in the submission of the field, so while it’s working on retrieving that data for one user, you can retrieve it for another.

I didn’t solve this one the way it wanted me to, I modified the value for the selected item in the drop down’s HTML to include OR 1=1

I did this one in the same way I did the last one, I entered smith and then just terminated the string, added OR 1 = 1; and commented out the rest of the command.

Terminated the current statement and added and insert command, then commented out the rest of the existing SELECT statement.

Just followed the instructions

**Part G: Feedback**

1. What did you like about the WebGoat lab?

I find system security interesting, so this lab it was fun to get to learn a bit of that.

1. What would you like to see changed?

Well the whole thing is one page that dynamically re-loads content, so refreshing the page always brings you out of your current lesson which is annoying.

1. Rate the Duration of the lab (1-10); 1- too short; 10 – too long

7

1. Rate the Difficulty of the lab (1-10); 1 – too short; 10 – too long

6

1. Keep in the curriculum (Y/N, why?)

**Yes, offensive security testing is important to learn, plus it’s interesting material and gives us things to look forward to in a course stream that otherwise has no code.**

**Marking Scheme**

|  |  |
| --- | --- |
|  | Marks |
| Part B |  |
| B.1 HTTP basics | 5 |
| B.2 Insecure Login | 5 |
| B.3 parameter Tampering | 5 |
| B.4 Research | 10 |
| C – injection flaws - SQL | 5 |
| C – injection flaws - XSS | 5 |
| D – concurrency | 5 |
| E – AJAX eval | 5 |
| F – Free Hunt | 5 |
| Organization/English | 5 |
| Total | 55 |

**To submit**

When you have completed the assignment, upload the following documents to Moodle.

* **YourUserName\_E31\_L11\_CyberGoat.docx**