Learning CBC Bit-flipping Through Gamification

Jeremy Druin
Learning CBC Bit-flipping Through Gamification

GIAC (GXPN) Gold Certification

Author: Jeremy Druin, jdruin@gmail.com
Advisor: Hamed Khiabani, Ph.D.
Accepted: April 10th, 2018

Abstract

Cryptanalysis concepts like CBC Bit-flipping can be difficult to grasp through study alone. Working through "hands-on" exercises is a common teaching technique intended to assist, but freely available training tools may not be readily available for advanced web application penetration testing practice. To this end, this paper will describe CBC bit-flipping and offer instruction on trying this cryptanalysis technique. Also, a CBC bit-flipping game will be provided within the OWASP Mutillidae II web application. Mutillidae is a large collection of deliberately vulnerable web application challenges designed to teach web security in a stand-alone, local environment.
1. Introduction

Cryptographic systems or cryptosystems may be categorized into three types: asymmetric, symmetric and hash functions (BEHRENS, 2014). Unlike hash functions, the other two allow the data to be easily recovered as long as the recipient has the proper key (Mehmood, 2017). Asymmetric or "public key" systems utilize two encryption keys, one private and one public, to encrypt and decrypt respectively (Microsoft, 2017). In symmetric cryptosystems, the same key is used for encryption and decryption (Figure 1) (TYSON, 2018).

1.1. What is cipher block chaining?

Cipher Block Chaining (CBC) is one mode available to block ciphers. Other modes include Electronic Codebook (ECB), Cipher Feedback (CFB), Output Feedback (OFB) and Counter Mode (CTR) (Kowalczyk, 2017).

Jeremy Druin, jdruin@gmail.com
1.1.1. Modes of Operation

Electronic Codebook mode might be thought of as a null mode. Besides encryption with the symmetric key, no additional transformation occurs (Thijssen, 2010). Each block of plaintext is encrypted independently. Also, the plaintext is not mixed with any additional data. This simplistic mode encrypts each block of plaintext with the key and outputs the ciphertext as-is. In Figure 2, the phrase "sans" is repeated 8 times then encrypted in 128-bit\(^1\) blocks using Advanced Encryption Standard (AES). The two lines of ciphertext output are identical because they were produced by encrypting the same input with the same key.

![Figure 2: Encryption of data with recurring characters using Electronic Codebook (ECB) mode results in a repetitive ciphertext](image)

Other modes incorporate data besides the key to further scramble the plaintext. Cipher Block Chaining (CBC) performs an exclusive-or (XOR) operation on the current plaintext block and the previously encrypted block (Tutorials Point, 2018). This mixing happens before encrypting the current block. The external source of data (from the current block point of view) acts like the salt used in hashing schemes (Morris & Thompson, 1978). The first block of plaintext presents a special case since no previously encrypted block exists. An Initialization Vector (IV) is provided to prime the pre-encryption operation (Barker & Barker, 2016).

Cipher Feedback (CFB) and Output Feedback (OFB) modes are similar to CBC except the characters in the prior block are encrypted rather than the plaintext. The result is then XORed with the current plaintext block to produce the current ciphertext block (Bunzel, 2018). These variants are desirable when it is convenient to have the underlying block cipher encrypt less than a full block of plaintext per encryption cycle (Hudde, 2009).

One mode combines the salting feature of CBC mode while operating on each block separately like ECB mode. Counter Mode (CTR) does not rely on output from previous

---

\(^1\) The character encoding is ASCII-Extended which requires 8-bits (1 byte) per character. Encryption is occurring in blocks of 16 characters.

Jeremy Druin, jdruin@gmail.com
operations. Instead, an unpredictable number-used-once or "nonce" acts as the salt for the first block (Gerard, 2018). The nonce is incremented each encryption cycle before it is used. This "counter" provides uniqueness but allows each block to be encrypted and decrypted autonomously. (Rogaway, 2011).

1.1.2. Failings of Electronic Codebook

In Electronic Codebook (ECB) mode, the plaintext block is encrypted to produce the ciphertext block independent of any other block (IBM, 2018). If a plaintext block contains the same plaintext as another, an identical ciphertext block will be created (Figure 2). This redundancy provides opportunities for attacks.

![Figure 3: Symmetric Encryption - Electronic Codebook (ECB) Mode](image)

For example, plaintext blocks can be substituted or replayed without knowing the contents (Sharma, 2007). Also, any previously cracked block betrays all identical ciphertext blocks (Crowley, 2013). In some cases, such as encrypting images, the image pattern can leak into the ciphertext (abpolym, 2015).

Jeremy Druin, jdruin@gmail.com
Identical plaintext is common in network communications. Many messages have well-known headers opening ECB-mode encrypted ciphertext to chosen plaintext attack (Chaudhary, 2014). Networking protocols such as Address Resolution Protocol and Hypertext Transport Protocol contain predictable headers found at the start of each network packet (Plummer, 1982) (Fielding & Reschke, 2014). These packets and their associated headers are transmitted many times until the entire message is sent.

1.1.3. Improvement over ECB

Cipher block chaining (CBC) offers advantages of Electronic Codebook mode. Each block of plaintext is mixed with the previous block of ciphertext (Figure 4). The pre-encryption XOR operation alleviates identical ciphertext blocks; even when the same plaintext is encrypted with the same key2.

Figure 4: Symmetric Encryption - Cipher Block Chaining (CBC) Mode

---

2 This assumes best-practice is followed. An initialization vector must never be reused with a given key (Gordon, 2015).

Jeremy Druin, jdruin@gmail.com
Since the first ciphertext block has no predecessor, an initial random value stands in for the first ciphertext block. This "initialization vector" (IV) provides the randomness needed to ensure identical plaintexts do not encrypt to repetitive ciphertexts. While it is not necessary to keep the IV secret since its only function is to introduce distinctiveness into the ciphertext, it is critical the IV be as unpredictable as possible (Diana-Lynn Contesti, 2007). The IV is provided to the recipient along with the ciphertext so the first block can be decrypted (D.I. Management Services Pty Ltd, 2001).

1.2. What is CBC bit-flipping?

CBC improves the integrity of the ciphertext by doing away with the repeated patterns in the plaintext. Also, exchanging, replicating or deleting ciphertext will corrupt at least one block of the message (Focardi, 2014). However, the exclusive-or (XOR)\(^3\) operation is linear. Changing one bit of ciphertext in a previous block has a predictable consequence on the respective bit in the current block of plaintext (Regalado, 2013).

1.2.1. Exclusive-Or

As a logical operation, XOR follows the rules expressed in Table 1. If the two inputs are different, the result is True but otherwise False (Electronics Tutorials, 2018). Assuming False is encoded as 0 and True as 1, the operation outputs bits as prescribed in Table 2.

\[ A \oplus B \]

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>A \oplus B</th>
</tr>
</thead>
<tbody>
<tr>
<td>False</td>
<td>False</td>
<td>False</td>
</tr>
<tr>
<td>False</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>True</td>
<td>False</td>
<td>True</td>
</tr>
<tr>
<td>True</td>
<td>True</td>
<td>False</td>
</tr>
</tbody>
</table>

\(^3\) The exclusive-or (XOR) operation is represented by the symbol \( \oplus \) (Addition symbol "plus" embedded in a circle)

Jeremy Druin, jdruin@gmail.com
Table 2: Result of Exclusive-Or operation on bits

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
<td>A (\oplus) B</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Exclusive-Or can be implemented as addition modulo 2 (Germundsson, 2002). For the first 3 rows of Table 2, addition works regardless of modulo (i.e. \(0 + 0 = 0, 0 + 1 = 1\ldots\)). In row 4, \((1 + 1) \mod 2 = 2 \mod 2 = 0\). Alternatively, one of the two input bits can be thought of as a switch that decides whether the second bit should change. For \(0 \oplus 1\), the first bit has a value of zero which indicates we should leave the second bit alone. "1" is output. For \(1 \oplus 1\), the first bit has a value of one telling us to change the second bit. The answer is "0".

1.2.2. Bit-flipping

When considering CBC, it may be more convenient to think in terms of the switch analogy. In the first two rows of Table 2, the first bit "A" is 0 so the second bit "B" is output as-is. In the third and fourth rows of Table 2, the first bit "A" is 1 so the opposite of bit "B" is output. Regardless, note that changing bit "A" no matter the value of bit "B" always causes the output to alternate or flip\(^4\) with respect to "B". Viewing the first bit as a switch dovetails nicely with the attack. The attacker decides which bit(s) in the previous block to change. These "flips" consistently cause the corresponding bits in the next block to change their value.

In CBC, the mixing of the plaintext and previous ciphertext block (or initialization vector) occurs prior to encryption (Figure 4). Exclusive-Or respects bit order when operating on blocks. The first bit of prior ciphertext is XORed with the first bit of plaintext, etc. If bit 3 of the previous ciphertext (or IV) is switched, bit 3 of the plaintext will be flipped prior to encryption (Figure 5).

\(^4\) The term "flip" may refer to turning over a coin so head becomes tails or vice versa

Jeremy Druin, jdruin@gmail.com
The consequence of bit-flipping on an actual application is highly contextual. It depends on how the application decodes and interprets the affected bits, but the consequences can be significant (GOODIN, 2016) (Marclass, 2015).

1.3. How might we learn about CBC bit-flipping?

Complex information security concepts may be easier to learn through experience (Griffiths & Guile, 2004). "Gamification" is an interactive simulation in the form of a goal-oriented challenge. Gamification is shown to increase awareness and interest in difficult notions (Gjertsen, Gjære, Bartnes, & Flores, 2017). OWASP Mutillidae II is a deliberately vulnerable web-application that uses gamification to teach web application security attacks and defenses (Druin, OWASP Mutillidae II, 2018). A CBC bit-flipping challenge is included (Druin, Listing of Vulnerabilities, 2018).

Figure 5: CBC Bit-flipping

Jeremy Druin, j druin@gmail.com
2. The Game

2.1. OWASP Mutillidae II Web Application

OWASP Mutillidae II is a free, open source web application that implements over 40 web application vulnerabilities including multiple defects from each of the OWASP Top Ten list (Druin, 2018). It is a training system that guides the user with hints that pop-up when the user hovers over a vulnerable input parameter and context-sensitive tutorials built into each page. Relevant instructional videos are linked from the bottom of each tutorial.

There are two learning modes: vulnerable ("Level 0"/"Level 1") and secure ("Level 5"). "Level 0" and "Level 1" implement identical vulnerable PHP source code which can be hacked. "Level 5" implements a different set of PHP source code that has been patched against the respective vulnerability. The system is in "Level 0" by default (Figure 6).

![Figure 6: Security Level](image)

The following demos use "Level 0" exclusively. If needed, the level can be changed by clicking "Toggle Security" (Figure 7).

![Figure 7: Toggle Security button](image)

2.2. Installation

A version of Mutillidae is pre-installed on the Samurai Web Testing Framework virtual machine. Mutillidae can be updated to the latest version by copying the latest scripts over the

---

5 "Level 1" layers on several client-side "security controls"
6 [https://sourceforge.net/projects/samurai/files/SamuraiWTF%203.0%20Branch/](https://sourceforge.net/projects/samurai/files/SamuraiWTF%203.0%20Branch/)

Jeremy Druin, jdruin@gmail.com
existing. Also, the source is available on SourceForge. The project may be installed on Linux Ubuntu using LAMP or Windows on XAMPP.

2.3. Solving the CBC bit-flipping Challenge

2.3.1. Understanding the goal

The challenge is implemented within the "View User Privilege Level" page. On the left menu, click "OWASP 2017" → "A2 - Broken Authentication and Session Management" → "Privilege Escalation" → "Via CBC bit-flipping" (Figure 8). The page will load (Figure 9).

![Figure 8: Menu to View User Privilege Level page](image)

7 https://www.youtube.com/watch?v=uQ2lr0TlqE
8 https://sourceforge.net/projects/mutillidae/files/mutillidae-project/
9 https://sourceforge.net/projects/mutillidae/files/documentation/mutillidae-installation-on-ubuntu.txt/download

Jeremy Druin, jdruin@gmail.com
By default, the "User ID" and "Group ID" displayed will be "100" (Figure 9). The challenge is to set both the "User ID" and "Group ID" to "000" using a CBC bit-flipping attack. Both fields are close to the target value. Only the leading "1" needs to be changed to a "0".

If the player would like to attempt a more difficult scenario, security "Level 1" implements "User ID" of "174" and "Group ID" of "235".

Jeremy Druin, jdruin@gmail.com
2.3.2. How to locate the vulnerable bytes

To attempt a CBC bit-flip attack on a given block, except for the first block, any previous block of ciphertext can be targeted (Figure 4). To manipulate the first block, the initialization vector has to be targeted. Each input on the page needs to be inspected to determine if the parameter provides an opportunity to influence some block.

When first approaching the page, it is not clear if an attack vector is exposed to the client although one parameter is suspicious. Inspection of the URL shows a query parameter named "iv" with a value "6bc24fc1ab650b25b4114e93a98f1eba" (Figure 11). The same parameter can be seen in Burp-Suite12 13 (Figure 12).

There does not appear to be any other input parameter that could have an effect on either the IV or a block of ciphertext. Also, the parameter name is dubious. "Pareto tests"15, trying only the input most likely to confirm the correct variable, are warranted to verify "iv". If these quick

---

12 Burp-Suite Community Edition 1.7.30 on Ubuntu 16.04.3 LTS was used for demonstration
13 Please refer to Appendix A for video tutorials on Burp-Suite
14 Proxy → HTTP History, click the request, select Request tab → Params tab
15 Vilfredo Pareto noted in the context of economics that about 20% of the input leads to 80% of results (Investopedia, 2018). The principle also applies to results relative to effort.

Jeremy Druin, jdruin@gmail.com
tests are inconclusive, methodical testing can be done. This test can be done formally or informally using manual methods or automated tools.

To test informally using a manual method, the first character or two of the "iv" field can be changed. Because the test is informal, the first two characters can be set to an arbitrary value and resubmitted. "00" is used in this example. Any change in output may indicate the "iv" parameter is a viable target. Indeed, the value of Application ID changes from "A1B2" to "*1B2" (Figure 13). Since there are only 32 characters in the "iv" parameter, manual analysis works well even if all positions need to be tested.

![User Privilege Level](image)

Figure 13: Application ID changes when "iv" parameter manipulated

The remaining bytes are mapped by changing pairs of characters in the IV to "00", observing the page output, and recording the results.

Formal testing can be done with Burp-Suite Community Edition\(^\text{16}\). The request is intercepted in Burp-Suite Proxy tool (Figure 14) then sent to the Burp-Suite Repeater tool where it is replayed (Figure 15). To ensure that replaying the HTTP request did not result in an unexpected response\(^\text{18}\), the response from the intercepted request is compared against the response from the repeated request using the Burp-Suite Comparer tool. No significant

\(^{16}\) https://portswigger.net/burp/communitydownload
\(^{17}\) Please see Appendix A for links to video tutorials prepared for each of the Burp-Suite features referenced
\(^{18}\) For example, if the site employed cross-site request forgery (CSRF) tokens valid for only one request

Jeremy Druin, jdruin@gmail.com
difference is found (Figure 16). This indicates requests can be repeated with minimal risk of invalid responses.

Figure 14: HTTP request for the page is intercepted in Burp-Suite Proxy tool

Figure 15: HTTP request for the page is sent to Burp-Suite Repeater tool

Jeremy Druin, jdruin@gmail.com
Figure 16: HTTP responses are compared to ensure requests are repeatable

The Character Frobber attack found in the Burp-Suite Intruder tool is designed to swiftly check if characters in a lengthy string impact how an application handles input (Portswigger, 2018). During the test, Burp-Suite sends one HTTP request for each character in the value of the "iv" parameter. The respective character is "incremented" before sending each request so that 6 becomes a 7, "b" becomes "c" and so on (PortSwigger, 2018). There are 32 characters in the value of "iv" so 32 unique requests are sent. The "Grep - Extract" feature can display the Application ID for each request to make changes in the Application ID easier to spot (Figure 17).

Jeremy Druin, jdruin@gmail.com
Changes in the "iv" parameter are reflected in the Application ID

In the first request, the value of the Application ID is "A1B2"; the default value. In the second request, Burp changed the first character of the IV from 6 to 7. The Application ID changed to "Q1B2". This implies the "iv" parameter is the initialization vector sought. Requests 2 - 7 provide additional evidence.

The test also suggests the IV is encoded as hexadecimal digits. The effects of character fiddling wear off once Burp moves on to characters 9 - 32 (Figure 17). The Application ID is only influenced if the first 8 characters are altered and the IV is made of 0-9/A-F. Assuming each pair of letters in the IV is a byte, the first letter of the Application ID corresponds to the first byte of the IV, the second letter to the second byte, etc. The information gathered so far is summarized in Table 3.

<table>
<thead>
<tr>
<th>Position</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>6b</td>
<td>c2</td>
<td>4f</td>
<td>c1</td>
<td>ab</td>
<td>65</td>
<td>0b</td>
<td>25</td>
<td>b4</td>
<td>11</td>
<td>4e</td>
<td>93</td>
<td>a9</td>
<td>8f</td>
<td>le</td>
<td>ba</td>
</tr>
</tbody>
</table>

Jeremy Druin, jdruin@gmail.com
Manipulating characters 9 - 32 did not impact the Application ID but may have affected other output. The boundary of character 8 and 9 can be tested with the Comparer tool\textsuperscript{19} or by looking at each HTTP response in a new window\textsuperscript{20}. The comparison shows IV-character 8 affects the end of the Application ID and the IV-character 9 does the same for the first number in the User ID (Figure 18).

Opening each response for manual comparison is tedious. The User ID and Group ID fields can be added to Grep - Extract for easy comparison. The output shows IV-characters 9 and 10 (IV byte 5) map to the first digit in the User ID (Figure 19). Similarly, IV-characters 15 and 16 (IV byte 8) map to the first digit in the Group ID (Figure 20). Recall these are the two digits need to be changed to "0" to win the game (Figure 10). Table 4 updates how each byte of the IV

\textsuperscript{19} Highlight rows 8 and 9 in the Intruder Attack window (Figure 17), right-click and select "Send to Comparer (Responses)".

\textsuperscript{20} Double-click on row 8 and 9 respectively in the Intruder Attack window (Figure 17), click the Response tab, search for "Application ID" (Figure 18)

Jeremy Druin, jdruin@gmail.com
maps to the digits of the Application ID, User ID, and Group ID. The targeted IV bytes 5 and 8 are highlighted.

Figure 19: IV-characters 9 and 10 influence the first digit of the User ID

Figure 20: IV-characters 15 and 16 influence the first digit of the Group ID

Table 4: Bytes of Initialization Vector by Position and Effect

<table>
<thead>
<tr>
<th>Position</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>6b</td>
<td>c2</td>
<td>4f</td>
<td>cl</td>
<td>ab</td>
<td>65</td>
<td>0b</td>
<td>25</td>
<td>b4</td>
<td>11</td>
<td>4e</td>
<td>93</td>
<td>a9</td>
<td>8f</td>
<td>1e</td>
<td>ba</td>
</tr>
<tr>
<td>Effect</td>
<td>A</td>
<td>1</td>
<td>B</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>?</td>
<td>?</td>
<td>?</td>
<td>?</td>
<td>?</td>
<td>?</td>
</tr>
</tbody>
</table>

Jeremy Druin, jdruin@gmail.com
2.3.3. How to modify the vulnerable bytes to win

The two IV bytes identified (Table 4) can be altered manually until the User ID and Group ID read "000". The bytes are encoded as ASCII hex so there are 256 values for each byte ranging from 00 to FF. However, switching the value and resubmitting the URL up to 512 times could be tiresome. Burp-Suite can try all possible values of each byte (Figure 21); however, the Community Edition throttles the Intruder (PortSwigger, 2016). Without the "Pro" license, the Intruder operates too slowly when fuzzing with more than about 25 values.

![Burp-Suite Intruder Brute-Force payload configured to try ASCII hex values 00 - FF](image)

Figure 21: Burp-Suite Intruder Brute-Force payload configured to try ASCII hex values 00 - FF

The OWASP Zed Attack Proxy (ZAP)\(^{21}\) is a free interception proxy that contains a versatile fuzzing tool (McRee, 2011)\(^{22}\). A new ZAP Fuzzer is opened. IV byte 5 which has a default value of "ab" for is prepared for fuzzing. To send all possible values, the digits "a" and "b" are selected independently and configured so that values 0-9 and a-f will be injected (Figure 22 - 1). Sixteen values are inserted into each digit for a total of 256 combinations (Figure 22 - 2).


\(^{22}\) Please refer to Appendix B for video tutorials on notable OWASP ZAP features

Jeremy Druin, jdruin@gmail.com
The fuzzing results appear in the ZAP Fuzzer output window (Figure 23). Somewhere within is an HTTP response with a value of "000" for the User ID. The ZAP Search feature can locate strings inside the fuzz results. Inspection of the User ID output shows the value is within an HTML table data element `<td style="text-align: left;">100 ( Hint: 0X31 0X30 0X30 )</td>`. A search pattern such as "<td style="text-align: left;">000" or just ">000" is enough to locate the corresponding HTTP response (Figure 24).

Figure 23: Results from OWASP ZAP Fuzzer injecting all combinations of two hex digits into IV byte 5

23 The View Source feature of the browser can show the HTML or the HTTP response can be viewed in Burp-Suite or OWASP ZAP

Jeremy Druin, jdruin@gmail.com
Learning CBC Bit-flipping Through Gamification

The digits needed in IV byte 8 to cause the Group ID to equal "000" is determined using the same method. Fuzzing will show the default value of 0x25 must be changed to 0x24. The "iv" parameter in the URL is updated with 6bc24fc1aa650b24b4114e93a98f1eba, then the page is submitted to the server. The User ID and Group ID fields match "000", so the game is won (Figure 25).

Figure 24: (1)(2) Using the "Search" feature of OWASP ZAP to locate fuzzing result with a User ID of "000". (3) The corresponding HTTP request shows the IV that produced the User ID. (4) The injected byte was "aa" (0XAA).

Figure 25: Bytes 5 and 8 are updated to cause the User ID and Group ID to equal "000". The web page outputs "User is Root!" indicating the game is won.

Jeremy Druin, jdruin@gmail.com
3. Conclusion

Training environments like Mutillidae II and Damn Vulnerable Web Application (DVWA)²⁴ provide safe environments to practice complex penetration testing tasks. Particularly abstract concepts like CBC bit-flipping can be easier to learn by experience. The game provided allows the user to work through two bit-flipping challenges: one for novices and another requiring intermediate skill. Also, the game in Mutillidae lets the user practice remediating the vulnerability. Mutillidae also shows the user how to fix the issue. In "Security Level 5", Mutillidae does not accept an initialization vector from the user. The IV is stored server-side and any input from the user is ignored.

Mutillidae chooses to remediate the bit-flipping vulnerability by preventing the user from changing the initialization vector. Because the IV can be private to the server in this use-case, keeping the IV from the user works well to fix the issue. In many cases, the IV must be changed frequently and/or passed between systems. In systems where the IV is only required to be unique and unpredictable, the integrity must be protected (Dworkin, 2001). The sender can provide a Message Authentication Code (MAC) for the encrypted message (Encrypt-then-MAC) or use encryption that includes authentication such as AES-GCM (Ducklin, 2013).

Other games are available to help learn other security penetration testing and secure application development concepts. Mutillidae and DVWA have many other web application security challenges. VulnHub provides vulnerable virtual machines with "boot2root" games in which a participant uses operating system and network service vulnerabilities to practice penetration testing (g0tmi1k, 2018). Several sites list practice systems and sites are available²⁵ ²⁶ ²⁷ giving students of information security many scenarios and contexts to hone skills (Vonnegut, 2015) (Edwards, 2017) (Shukla, 2014). These systems help learn concepts that can be challenging to learn through academic study alone. They also provide "hands-on" exercises in a safe but realistic environment giving students an invaluable resource that might otherwise not be offered.

²⁴ http://www.dvwa.co.uk
Jeremy Druin, jdruin@gmail.com
Appendix A: Video Tutorials for Notable Burp-Suite Features

<table>
<thead>
<tr>
<th>Title</th>
<th>URL</th>
</tr>
</thead>
<tbody>
<tr>
<td>How to Install Burp Suite on Linux</td>
<td><a href="https://www.youtube.com/watch?v=OsSPwe-DUOU">https://www.youtube.com/watch?v=OsSPwe-DUOU</a></td>
</tr>
<tr>
<td>Introduction to using the Burp-Suite Targets Tab</td>
<td><a href="https://www.youtube.com/watch?v=q3iG7YMjcmE">https://www.youtube.com/watch?v=q3iG7YMjcmE</a></td>
</tr>
<tr>
<td>Introduction to Web Request and Response Interception with Burp-Suite</td>
<td><a href="https://www.youtube.com/watch?v=qsE04AhJJrc">https://www.youtube.com/watch?v=qsE04AhJJrc</a></td>
</tr>
<tr>
<td>Introduction to Burp-Suite's Repeater Tool</td>
<td><a href="https://www.youtube.com/watch?v=IstpobN5azo">https://www.youtube.com/watch?v=IstpobN5azo</a></td>
</tr>
<tr>
<td>Introduction to Burp-Suite Intruder's Character Frobrber Payload</td>
<td><a href="https://www.youtube.com/watch?v=7vWTEbOfa-8">https://www.youtube.com/watch?v=7vWTEbOfa-8</a></td>
</tr>
<tr>
<td>Introduction to Burp-Suite Intruder's &quot;Grep Extract&quot; Feature</td>
<td><a href="https://www.youtube.com/watch?v=t0uMReqs8Ng">https://www.youtube.com/watch?v=t0uMReqs8Ng</a></td>
</tr>
<tr>
<td>Introduction to Burp-Suite Comparer Tool</td>
<td><a href="https://www.youtube.com/watch?v=KxqY_bp13gc">https://www.youtube.com/watch?v=KxqY_bp13gc</a></td>
</tr>
</tbody>
</table>

Appendix B: Video Tutorials for Notable OWASP ZAP Features

<table>
<thead>
<tr>
<th>Title</th>
<th>URL</th>
</tr>
</thead>
<tbody>
<tr>
<td>How to Install OWASP Zap on Linux</td>
<td><a href="https://www.youtube.com/watch?v=MpuFW_mkJ3M">https://www.youtube.com/watch?v=MpuFW_mkJ3M</a></td>
</tr>
<tr>
<td>How to Proxy Web Traffic through OWASP ZAP</td>
<td><a href="https://www.youtube.com/watch?v=ICPqz1A19fK">https://www.youtube.com/watch?v=ICPqz1A19fK</a></td>
</tr>
<tr>
<td>How to Spider a Web Site with OWASP ZAP</td>
<td><a href="https://www.youtube.com/watch?v=pGCBivHNRn8">https://www.youtube.com/watch?v=pGCBivHNRn8</a></td>
</tr>
<tr>
<td>How to Intercept HTTP Requests with OWASP ZAP</td>
<td><a href="https://www.youtube.com/watch?v=fa5LAFXmwoo">https://www.youtube.com/watch?v=fa5LAFXmwoo</a></td>
</tr>
<tr>
<td>How to Fuzz Web Applications with OWASP ZAP (Part 1)</td>
<td><a href="https://www.youtube.com/watch?v=uSfGeyJKIVA">https://www.youtube.com/watch?v=uSfGeyJKIVA</a></td>
</tr>
<tr>
<td>How to Fuzz Web Applications with OWASP ZAP (Part 2)</td>
<td><a href="https://www.youtube.com/watch?v=tBXX_GAK7BU">https://www.youtube.com/watch?v=tBXX_GAK7BU</a></td>
</tr>
</tbody>
</table>


Jeremy Druin, jdruin@gmail.com
References


Jeremy Druin, jdruin@gmail.com

Edwards, B. (2017, 1 2). *40+ INTENTIONALLY VULNERABLE WEBSITES TO (LEGALY) PRACTICE YOUR HACKING SKILLS.* Retrieved from Bonkers About Tech:

https://www.electronics-tutorials.ws/logic/logic_7.html

http://blog.centrify.com/sfa-mfa-difference/


Focardi, R. (2014). *Block cipher modes.* Retrieved from SecGroup Unive:
https://secgroup.dais.unive.it/teaching/cryptography/block-cipher-modes/

General. (2017, 4 1). *Block cipher mode of operation.* Retrieved 4 1, 2017, from Wikipedia, the free encyclopedia:


http://mathworld.wolfram.com/XOR.html

https://brage.bibsys.no/xmlui/bitstream/handle/11250/2462736/ICISSP_2017-gamification.pdf?sequence=2


https://www.emsec.rub.de/media/crypto/attachments/files/2011/03/hudde.pdf


Jeremy Druin, jdruin@gmail.com


Jeremy Druin, jdruin@gmail.com


http://www.cs.man.ac.uk/~banach/COMP61411.Info/CourseSlides/Wk2.1.DES.pdf


Titcomb, J. (2016, 3 22). *Do you have one of the most common passwords? They’re ridiculously easy to guess*. Retrieved 4 1, 2017, from The Telegraph:
http://www.telegraph.co.uk/technology/2016/01/26/most-common-passwords-revealed---and-theyre-ridiculously-easy-to/


https://computer.howstuffworks.com/encryption2.htm


Jeremy Druin, jdruin@gmail.com
<table>
<thead>
<tr>
<th>Event Name</th>
<th>Location</th>
<th>Dates</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>SANS Chicago Spring 2020</td>
<td>Chicago, ILUS</td>
<td>Jun 01, 2020 - Jun 06, 2020</td>
<td>Live Event</td>
</tr>
<tr>
<td>SANS ICS Europe Summit &amp; Training 2020</td>
<td>Munich, DE</td>
<td>Jun 08, 2020 - Jun 13, 2020</td>
<td>Live Event</td>
</tr>
<tr>
<td>SANS Budapest June 2020</td>
<td>Budapest, HU</td>
<td>Jun 08, 2020 - Jun 13, 2020</td>
<td>Live Event</td>
</tr>
<tr>
<td>SANS Las Vegas Summer 2020</td>
<td>Las Vegas, NVUS</td>
<td>Jun 08, 2020 - Jun 13, 2020</td>
<td>Live Event</td>
</tr>
<tr>
<td>SANS Zurich June 2020</td>
<td>Zurich, CH</td>
<td>Jun 15, 2020 - Jun 20, 2020</td>
<td>Live Event</td>
</tr>
<tr>
<td>SANS London June 2020</td>
<td>OnlineGB</td>
<td>Jun 01, 2020 - Jun 06, 2020</td>
<td>Live Event</td>
</tr>
<tr>
<td>SANS OnDemand</td>
<td>Books &amp; MP3s OnlyUS</td>
<td>Anytime</td>
<td>Self Paced</td>
</tr>
</tbody>
</table>