TATA CONSULTANCY SERVICES
Experience certainty.

ORACLE PADDING

February 2019
Author : Binayak Banerjee

Contents

List of Tables

ORACLE PADDING

In this article padding oracle attack has been described in following three sections.

1. Understanding the Padding Methodology in Use
2. Encryption and Decryption Logic for Padding Encryption
3. Attacking Methodology in a Real Life Scenario

1. Understanding Padding Methodology in Use

The core to understand padding oracle attack is understanding the method that is in use. The method includes cryptographic padding using certain block ciphers. Usage of cryptographic block ciphers however leads to the conclusion that 'a text which has undergone a cryptographic block cipher encoding, must be a multiple of the block size that was decided upon, before enforcing the encoding algorithm'. This idea can be better explained with the following words being padded as an example.

Table 1: Padding Methodology

Words	Block 1								Block 2							
	1	2	3	4	5	6	7	8								
TOY	T	O	Y	0x5	0x5	0x5	0x5	0x5								
	1	2	3	4	5	6	7	8								
APPLE	A	P	P	L	E	0x3	0x3	0x3								
	1	2	3	4	5	6	7	8								
BOTTLES	B	O	T	T	L	E	S	0x1								
	1	2	3	4	5	6	7	8	1	2	3	4	5	6	7	8
UMBRELLA	U	M	B	R	E	L	L	A	0x8	0x8	0x8	0x8	0x8	0x8	0x8	0x8
	1	2	3	4	5	6	7	8	1	2	3	4	5	6	7	8
DRAGONFLY	D	R	A	G	O	N	F	L	Y	0x7	0x7	0x7	0x7	0x7	0x7	0x7

Points to be noted

From the above padded values in blocks, following rules regarding padding can be noted.

- There must be at least one padding value associated with the actual text.
- In case the actual text consumes one whole block, the next block is completely used for padding (UMBRELLA).
- Since a block cipher is being used (block size: 8 in the example), the resultant data set size of data must be a multiple of 8.

Cipher Text																Block Size
T	O	Y	0x5	0x5	0x5	0x5	0x5									8
A	P	P	L	E	0x3	0x3	0x3									8
B	O	T	T	L	E	S	0x1									8
U	M	B	R	E	L	L	A	0x8	0x8	0x8	0x8	0x8	0x8	0x8	0x8	16
D	R	A	G	O	N	F	L	Y	0x7	0x7	0x7	0x7	0x7	0x7	0x7	16

- The value of the padding byte also indicates the remaining bytes to fill the block (highlighted in the block table)
- Therefore, it can be confirmed from the above table that for a block size of 8, the possible padding values are: 0x1, 0x2, 0x3, 0x4, 0x5, 0x6, 0x7, 0x8

Let's consider an implementation scenario of a real-life implementation of padding and the ways to bypass it.

Real Life Scenario

Scenario: An application uses a string parameter to pass encrypted username, userid, roleid. The regular text is encoded using CBC (using an Initialization Vector - IV). This IV is pre-pended to the resulting cipher text.

When the application receives the encrypted value, it has one of the following three responses.

- Valid cipher text received (properly padded with valid data) – respond with 200 OK
- The cipher text is invalid (decryption does not result in valid padding value as discussed in padding rules) – respond with 500 internal server error
- Valid cipher but invalid/unexpected data when decrypted (data received is not recognized by application logic) – respond with 200 OK and a custom error message

This is a very usual scenario of oracle padding, where the application response may be used to tinker with the cipher text, in order to predict the actual text that was sent in to the application. The following URL contains the encrypted value for actual text (*BRIAN;12;1;*) in the format *username;userid;roleid;* in an encoded ASCII HEX representation.

http://demoapp/home.jsp?UID=7B216A634951170FF851D6CC68FC9537858795A28ED4AAC6

Decoding the ASCII HEX: *{!jcIQ##øQÖÌhü•7…‡•¢Ž ÔªÆ*

As the actual text is padded, it is impossible retrieve the decoded value from the same (decoded ASCII HEX mentioned in preceding line). This is where oracle padding attack comes into play. The actual text can be deduced if the encoding and decoding mechanisms can be traced back.

The UID (ciphertext) in the URL seems to be of 24bytes. This number is divisible by 8 and not 16. This confirms the algorithm must have used a block size of 8bytes while encoding. This idea combined with what learnings regarding the IV being prepended to the cipher text can help us break the whole word into following.

Cipher text: *7B216A634951170FF851D6CC68FC9537858795A28ED4AAC6*

Table 2: Cipher Text

	Initialization Vector (IV)								BLOCK 1								BLOCK 2							
Plain-Text	-	-	-	-	-	-	-	-	B	R	I	A	N	;	1	2	;	1	;	-	-	-	-	-
Plain-Text(Padded)	-	-	-	-	-	-	-	-	B	R	I	A	N	;	1	2	;	1	;	0x 05	0x 05	0x 05	0x 05	0x 05
Encrypted Value (Hex)	0x 7B	0x 21	0x 6A	0x 63	0x 49	0x 51	0x 17	0x 0F	0x F8	0x 51	0x D6	0x CC	0x 68	0x FC	0x 95	0x 37	0x 85	0x 37	0x 95	0x A2	0x 8E	0x 04	0x AA	0x C6

2. Encryption and Decryption Logic for Padding Encryption

- **Encryption (Regular Text to Padded Encrypted Text)**

Table 3: Encryption (Regular Text to Padded Encrypted Text)

	Block 1									Block 2							
	1	2	3	4	5	6	7	8		1	2	3	4	5	6	7	8
IV	0x7B	0x21	0x6A	0x63	0x49	0x51	0x17	0x0F		0x39	0x78	0x23	0x22	0x07	0x6A	0x26	0x3D
	XOR									XOR							
PT	B	R	I	A	N	;	1	2		;	1	;	0x05	0x05	0x05	0x05	0x05
	Equals									Equals							
IN V	0x39	0x73	0x23	0x22	0x07	0x6A	0x26	0x3D		0xC3	0x60	0xED	0xC9	0x6D	0xF9	0x90	0x32
	Triple DES									Triple DES							
EO	0xF8	0x51	0xD6	0xCC	0x68	0xFC	0x95	0x37		0x85	0x87	0x95	0xA2	0x8E	0xD4	0xAA	0xC6

IV: Initialization Vector PT: Plain-Text IN V: Intermediary Value (HEX)

EO: Encrypted Output

Cipher text: *7B216A634951170FF851D6CC68FC9537858795A28ED4AAC6*

From the Table 3, it is observed that the Initialization Vector (IV) value in Block 1 is initial 8 bytes of the cipher text.

Initialization Vector: *7B216A634951170F* (1st Part of Cipher text)

This acts as a salt in a password in this scenario. This IV is XORed with Plain-Text and on the result Triple DES symmetric encryption is implemented. This becomes the second half of the cipher text.

1st Block Cipher text: *F851D6CC68FC9537* (2nd Part of Cipher text)

Intermediary Value (HEX): *39782322076A263D*

The intermediary value from Block 1 is used as IV for Block 2. Same steps are followed again where, IV is XORed with plain-text + padded value = result. And the resultant is encrypted with Triple DES.

2nd Block Cipher text: *858795A28ED4AAC6* (3rd Part of Cipher text)

- **Decryption (Padded Encrypted Text to Regular Text)**

Table 4: Decryption (Padded Encrypted Text to Regular Text)

	Block 1									Block 2							
	1	2	3	4	5	6	7	8		1	2	3	4	5	6	7	8
EI	0xF8	0x51	0xD6	0xCC	0x68	0xFC	0x95	0x37		0x85	0x87	0x95	0xA2	0x8E	0xD4	0xAA	0xC6
	Triple DES									Triple DES							
IN V	0x39	0x73	0x23	0x22	0x07	0x6A	0x26	0x3D		0xC3	0x60	0xED	0xC9	0x6D	0xF9	0x90	0x32

V

	XOR									Equals							
IV	0x7B	0x21	0x6A	0x63	0x49	0x51	0x17	0x0F		0xF8	0x51	0xD6	0xCC	0x68	0xFC	0x95	0x37
	Equals									Triple DES							
PT	B	R	I	A	N	;	1	2		;	1	;	0x05	0x05	0x05	0x05	0x05

EI: Encrypted Output Vector IN V: Intermediary Value (HEX) IV: Initialization

PT: Plain-Text

Encrypted Input: *7B216A634951170F F851D6CC68FC9537 858795A28ED4AAC6*

Part 1 *Part 2* *Part 3*

From the above, the encrypted text can be divided into three distinct parts -

- Initialization Vector (IV),
- Block 1,
- Block 2

Triple DES is applied on both the blocks and Intermediate Hex value (IN V) is obtained.
Further,

Block 1 IN V is XORed with the IV (the first part of the encrypted input) to get Regular Text.
Block 2 IN V is XORed with the EI of Block 1 (the second part of the encrypted input) to get Regular Text.

3. Attacking Methodology in a Real Life Scenario

Padding oracle attack operates in a way where a single encrypted block is cracked at a time. Isolate the first block and start sending the request with null values attached as the initialization vector.

- **Decrypting Block 1**

Request: *http://demoapp/home.jsp?UID=0000000000000000F851D6CC68FC9537*

Response: 500 Internal Server Error

This response must be due to invalid padding. This was confirmed by the application behavior previously. While trying to simulate the application behavior, the following must have happened at backend while decrypting the value.

	Block 1							
	1	2	3	4	5	6	7	8
EI	0xF8	0x51	0xD6	0xCC	0x68	0xFC	0x95	0x37
	Triple DES							
IN V	Server can decrypt this value							
	XOR							

IV	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00
	Equals							
Decrypted Value	0x39	0x73	0x23	0x22	0x07	0x6a	0x26	0x3D

From this decryption mechanism, it is found that the application did not get a valid padding value in the decrypted result. This contradicts, the basic padding rule which says there must be at least one padding value in the decrypted result. Hence, the IV value is incremented by one and make the request again.

Request: *http://demoapp/home.jsp?UID=0000000000000001F851D6CC68FC9537*

Response: 500 Internal Server Error

	Block 1							
	1	2	3	4	5	6	7	8
EI	0xF8	0x51	0xD6	0xCC	0x68	0xFC	0x95	0x37
	Triple DES							
IN V	Server can decrypt this value							
	XOR							
IV	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x01
	Equals							
Decrypted Value	0x39	0x73	0x23	0x22	0x07	0x6a	0x26	0x3C

It is observed that by incrementing the value by one, the decrypted value decreases. Keep changing the values until a valid padding value for this situation is obtained(which would be 0x01).

Request: *http://demoapp/home.jsp?UID=000000000000003CF851D6CC68FC9537*

Response: 200 Custom Error Page

	Block 1							
	1	2	3	4	5	6	7	8
EI	0xF8	0x51	0xD6	0xCC	0x68	0xFC	0x95	0x37
	Triple DES							
IN V								0x3D
	XOR							
IV	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x3C
	Equals							
Decrypted Value	0x39	0x73	0x23	0x22	0x07	0x6a	0x26	0x01

This time, a 200 OK response with custom error is received. This suggests the decrypted value has proper padding. From this, a XOR property may be used to guess the IN V.
In this case :

IN V XORed 0x3C = 0x01
IN V = 0x01 XORed 0x3C
IN V = 0x3D

Hence, the IN V value is deduced by following the above method. Similarly the other values for IV V can be deduced by following the same method.

Request: *http://demoapp/home.jsp?UID=000000000000243FF851D6CC68FC9537*

Response: 200 Custom Error Page

	Block 1							
	1	2	3	4	5	6	7	8
EI	0xF8	0x51	0xD6	0xCC	0x68	0xFC	0x95	0x37
	Triple DES							
IN V							0x26	0x3D
	XOR							
IV	0x00	0x00	0x00	0x00	0x00	0x00	0x24	0x3F
	Equals							
Decrypted Value	0x39	0x73	0x23	0x22	0x07	0x6a	0x02	0x02

As shown in this table, this is how the IV can be updated in the request in an attempt to get valid padding values. This way, the IN V values for the whole of Block 1 can be obtained.

Request: *http://demoapp/home.jsp?UID=317B2B2A0F622E35F851D6CC68FC9537*

Response: 200 Custom Error Page

	Block 1							
	1	2	3	4	5	6	7	8
EI	0xF8	0x51	0xD6	0xCC	0x68	0xFC	0x95	0x37
	Triple DES							
IN V	0x39	0x73	0x23	0x22	0x07	0x6A	0x26	0x3D
	XOR							
IV	0x31	0x7B	0x2B	0x2A	0x0F	0x62	0x2E	0x35
	Equals							
PT	0x08	0x08	0x08	0x08	0x08	0x08	0x08	0x08

Thus the IN V values are retrieved successfully. Hence IN V and IV can be XORed(where IV is the 1st part of the encrypted string) to get the Plain-Text.
This way, IN V can be obtained even without knowing the decryption mechanism from EI → IN V.
Further IN V and IV (7B216A634951170F) are XORed to get the Plain-Text result.

	Block 1							
	1	2	3	4	5	6	7	8
EI	0xF8	0x51	0xD6	0xCC	0x68	0xFC	0x95	0x37
	Triple DES							
IN V	0x39	0x73	0x23	0x22	0x07	0x6A	0x26	0x3D
	XOR							
IV	0x7B	0x21	0x6A	0x63	0x49	0x51	0x17	0x0F
	Equals							
PT	B	R	I	A	N	;	1	2

- **Decrypting Block 2**

Similarly, IN V for Block 2 is guessed in such a way that the padding value changes to 0x1, 0x2, 0x3 ...and so on. This way, the IN V for Block 2 is obtained by following the same method as in Block 1. Once the IN V is deduced, it can be XORed with the 2nd part of the encrypted value (F851D6CC68FC9537). This will result in the value for second part of the Plain-Text, thus cracking the padded encryption.

To automate the above process of cracking encrypted padded values, tools like pad buster can be used. This will do the heavy lifting job by automating the trial and error process. It analyzes the response and produces the cracked values.

Example usage:
padBuster.pl
http://demoapp/home.jsp?UID=7B216A634951170FF851D6CC68FC9537858795A28ED4AAC6
7B216A634951170FF851D6CC68FC9537858795A28ED4AAC6 8 -encoding 2

The three mandatory formats to decode the padded values are -

- The URL
- Encrypted Sample
- Block Size
- Encoding type (Optional: if the input data is encoded)