UTF-8 Overlong Encoding导致的安全问题

原创 phith0n 代码审计 2024-02-23 20:44 新加坡

「代码审计」知识星球中@1ue 发表了一篇有趣的文章《**探索Java反序列化绕WAF新姿势**》，深入研究了一下其中的原理，我发现这是一个对我来说很“新”，但实际上年纪已经很大的Trick。

0x01 UTF-8编码原理

UTF-8是现在最流行的编码方式，它可以将unicode码表里的所有字符，用某种计算方式转换成长度是1到4位字节的字符。

参考这个表格，我们就可以很轻松地将unicode码转换成UTF-8编码：

First code point	Last code point	Byte 1	Byte 2	Byte 3	Byte 4
U+0000	U+007F	0xxxxxxx			
U+0080	U+07FF	110xxxxx	10xxxxxx		
U+0800	U+FFFF	1110xxxx	10xxxxxx	10xxxxxx	
U+10000	U+10FFFF	11110xxx	10xxxxxx	10xxxxxx	10xxxxxx

举个例子，欧元符号€的unicode编码是U+20AC，按照如下方法将其转换成UTF-8编码：

- 首先，因为U+20AC位于U+0800和U+FFFF之间，所以按照上表可知其UTF-8编码长度是3
- 0x20AC的二进制是 `10 0000 1010 1100` ，将所有位数从左至右按照4、6、6分成三组，第一组长度不满4前面补0： `0010` ， `000010` ， `101100`
- 分别给这三组增加前缀 `1110` 、 `10` 和 `10` ，结果是 `11100010` 、 `10000010` 、 `10101100` ，对应的就是 `\xE2\x82\xAC`
- `\xE2\x82\xAC` 即为欧元符号€的UTF-8编码

```
In [36]: hex(0b11100010)
Out[36]: '0xe2'

In [37]: hex(0b10000010)
Out[37]: '0x82'

In [38]: hex(0b10101100)
Out[38]: '0xac'

In [39]: b'\xE2\x82\xAC'.decode()
Out[39]: '€'
```

0x02 Overlong Encoding是什么问题?

那么，了解了UTF-8的编码过程，我们就可以很容易理解Overlong Encoding是什么问题了。

Overlong Encoding就是将1个字节的字符，按照UTF-8编码方式强行编码成2位以上UTF-8字符的方法。

仍然举例说明，比如点号 `.` ，其unicode编码和ascii编码一致，均为 `0x2E` 。按照上表，它只能被编码成单字节的UTF-8字符，但我按照下面的方法进行转换：

- `0x2E` 的二进制是 `10 1110` ，我给其前面补5个0，变成 `00000101110`
- 将其分成5位、6位两组： `00000` ， `101110`
- 分别给这两组增加前缀 `110` ， `10` ，结果是 `11000000` ， `10101110` ，对应的是 `\xC0AE`

`0xC0AE` 并不是一个合法的UTF-8字符，但我们确实是按照UTF-8编码方式将其转换出来的，这就是UTF-8设计中的一个缺陷。

按照UTF-8的规范来说，我们应该使用字符可以对应的最小字节数来表示这个字符。那么对于点号来说，就应该是0x2e。但UTF-8编码转换的过程中，并没有限制往前补0，导致转换出了非法的UTF-8字符。

这种攻击方式就叫“Overlong Encoding”。

Overlong Encoding实际上很早就被提出了，早到那时候我还没开始学安全。很多语言在实现UTF-8的转换时，会对这个攻击方式做一定检查。比如，Python中如果你想将 `0xC0AE` 转换成点号，就会抛出异常：

```
b'\xC0\xAE'.decode()
```

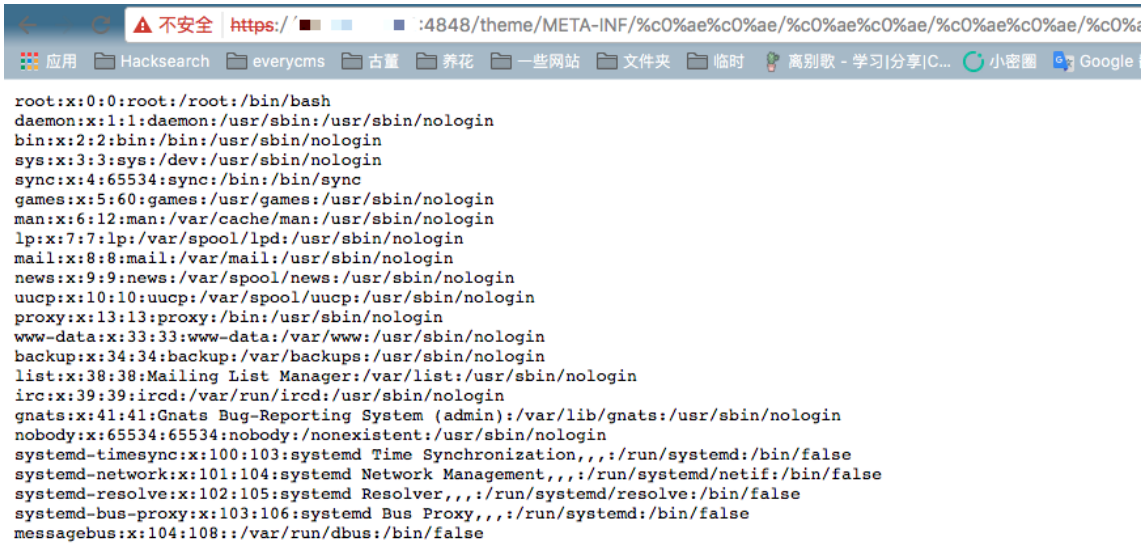
```
In [59]: b'\xC0\xAE'.decode()
---------------------------------------------------------------------------
UnicodeDecodeError                        Traceback (most recent call last)
Input In [59], in <cell line: 1>()
----> 1 b'\xC0\xAE'.decode()

UnicodeDecodeError: 'utf-8' codec can't decode byte 0xc0 in position 0: invalid start byte
```

但我们质朴刚健的Java生态，在很多地方是没有对其进行防御的，这就导致了一些安全问题。

0x03 GlassFish 任意文件读取漏洞

如果对安全熟悉的读者，看到前面的 `0xC0AE` ，其实应该很快想起来一个经典漏洞——**GlassFish任意文件读取漏洞**。

这个漏洞就是在URL中使用 `%C0%AE` 来代替点号 `.` ，绕过目录穿越的限制，导致任意文件读取漏洞：



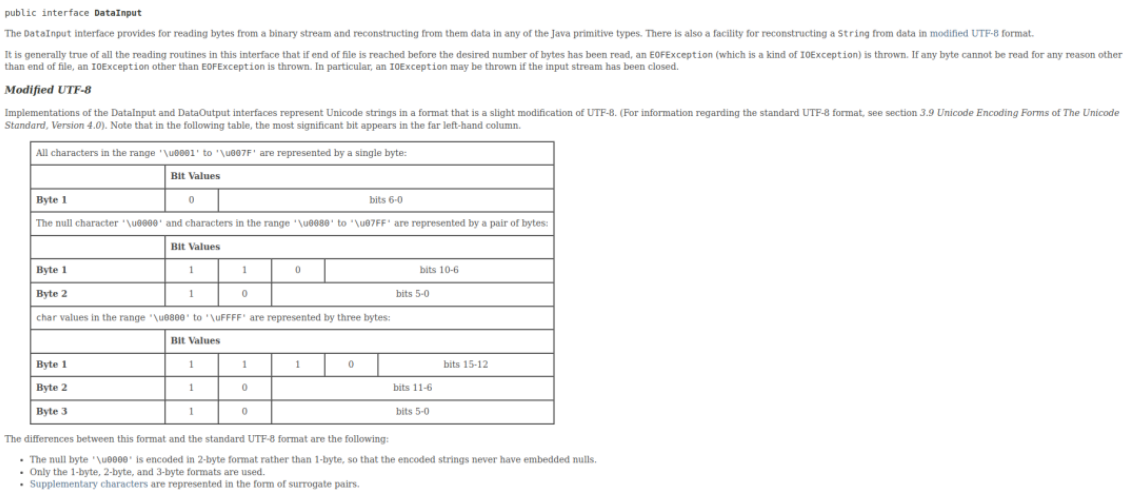
其原理就是GlassFish在路径解码时使用UTF-8编码，很典型的Overlong Encoding利用。

0x04 利用Overlong Encoding绕过WAF

回到本文开头的文章，其实@1ue 是完全在分析反序列化代码的时候发现了这个问题，换句话说，就等于把Overlong Encoding攻击重新发现了一遍，还是挺厉害的。

Java在反序列化时使用 `ObjectInputStream` 类，这个类实现了 `DataInput` 接口，这个接口定义了读取字符串的方法 `readUTF` 。在解码中，Java实际实现的是一个魔改过的UTF-8编码，名为“Modified UTF-8”。

参考其文档可以发现，“Modified UTF-8”类似于MySQL中的UTF8，只使用三个字节来表示：



public interface **DataInput**

The DataInput interface provides for reading bytes from a binary stream and reconstructing from them data in any of the Java primitive types. There is also a facility for reconstructing a String from data in modified UTF-8 format.

It is generally true of all the reading routines in this interface that if end of file is reached before the desired number of bytes has been read, an EOFException (which is a kind of IOException) is thrown. If any byte cannot be read for any reason other than end of file, an IOException other than EOFException is thrown. In particular, an IOException may be thrown if the input stream has been closed.

Modified UTF-8

Implementations of the DataInput and DataOutput interfaces represent Unicode strings in a format that is a slight modification of UTF-8. (For information regarding the standard UTF-8 format, see section *3.9 Unicode Encoding Forms* of *The Unicode Standard, Version 4.0*). Note that in the following table, the most significant bit appears in the far left-hand column.

All characters in the range '\u0001' to '\u007F' are represented by a single byte:					
	Bit Values				
Byte 1	0	bits 6-0			
The null character '\u0000' and characters in the range '\u0080' to '\u07FF' are represented by a pair of bytes:					
	Bit Values				
Byte 1	1	1	0	bits 10-6	
Byte 2	1	0	bits 5-0		
char values in the range '\u0800' to '\uFFFF' are represented by three bytes:					
	Bit Values				
Byte 1	1	1	1	0	bits 15-12
Byte 2	1	0	bits 11-6		
Byte 3	1	0	bits 5-0		

The differences between this format and the standard UTF-8 format are the following:

- The null byte '\u0000' is encoded in 2-byte format rather than 1-byte, so that the encoded strings never have embedded nulls.
- Only the 1-byte, 2-byte, and 3-byte formats are used.
- Supplementary characters are represented in the form of surrogate pairs.

但其三字节以内的转换过程是和UTF-8相同的，所以仍然继承了“Overlong Encoding”缺陷。

攻击者可以将反序列化字节流里一些字符按照“Overlong Encoding”的方法转换成非法UTF-8字符，用来绕过一些基于流量的防御方法。

我写了一个简单的Python函数，用于将一个ASCII字符串转换成Overlong Encoding的UTF-8编码：

```
def convert_int(i: int) -> bytes:
    b1 = ((i >> 6) & 0b11111) | 0b11000000
    b2 = (i & 0b1111111) | 0b10000000
    return bytes([b1, b2])


def convert_str(s: str) -> bytes:
    bs = b''
    for ch in s.encode():
        bs += convert_int(ch)

    return bs


if __name__ == '__main__':
    print(convert_str('.')) # b'\xc0\xae'
    print(convert_str('org.example.Evil')) # b'\xc1\xef\xc1\xf2\xc1\xe7\xc0\xae\xc1\xe5\xc1\xf8\x
```

参考链接：

- https://t.zsxq.com/17LkqCzk8
- https://capec.mitre.org/data/definitions/80.html
- https://en.wikipedia.org/wiki/UTF-8#Overlong_encodings
- https://utf8-chartable.de/unicode-utf8-table.pl
- https://github.com/vulhub/vulhub/tree/master/glassfish/4.1.0
- https://docs.oracle.com/javase%2F8%2Fdocs%2Fapi%2F%2F/java/io/DataInput.html

喜欢这篇文章，点个在看再走吧~

加入「代码审计」，学习更多安全知识。一次付费，终身学习免续费。

知识星球

「代码审计」星球，一次付费，终身学习

小程序

知识星球 1 Java安全 1 UTF-8 1

阅读原文 文章已于2024-02-23修改