利用 burp 插件 Hackvertor 绕过 waf 并破解 XOR 加密 - 嘶吼 RoarTalk – 回归最本质的信息安全, 互联网安全新媒体, 4hou.com

我最近在开发一个 burp 插件 Hackvertor，你们可能不知道。它的特点是基于标签转码，比 burp 内置的 decoder 强大得多。基于标签转码的原理是通过标签对它的内容进行转码，然后将转码后的内容传递给外层标签继续转码，这让你可以轻易的进行多重转码。

比如，如果你想对字符串进行 base64 编码，那么你可以使用 base64 标签，如下所示：

```
<@base64_0>test<@/base64_0>
```

当然，你可以进行多重编码，比如，你想先对字符串进行 hex 编码，然后再进行 base64 编码，那么你就可以先用 hex 标签，然后外层再使用 base64 标签将它包起来，如下所示：

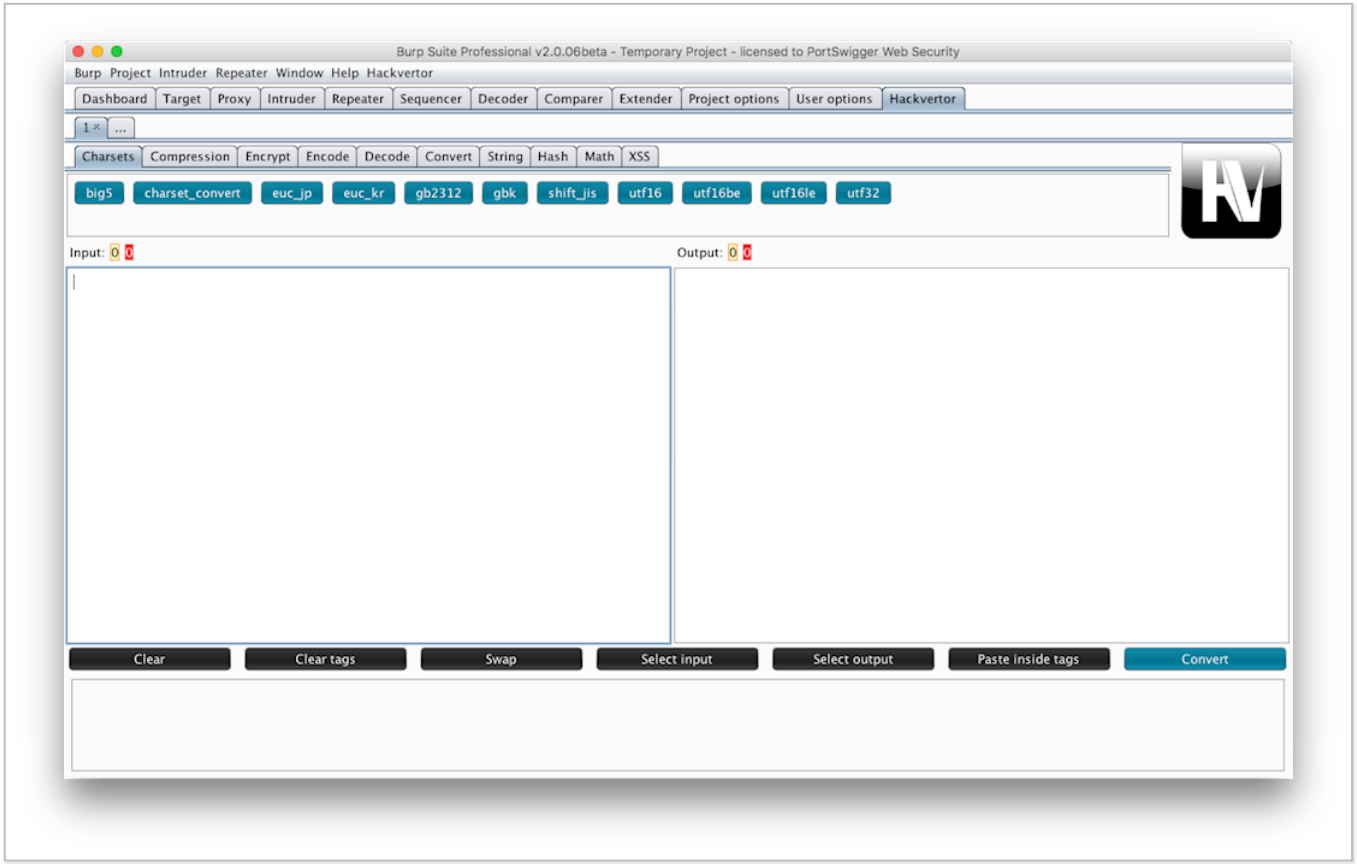
```
<@base64_1><@hex_0(" ")>test<@/hex_0><@/base64_1>
```

hex 标签有一个分隔符参数，会将每一个 hex 字符分开，这里的分隔符是空格，然后被空格分开的 tes

t 会被传递到标签函数中。

Hackvertor 用法

加载这个插件后，会在 burp 的主界面中新增一个 tab，叫 Hackvertor。在这个 tab 下，有一个输入框和一个输出框。在输入框中输入你要进行转换的内容，选中它，然后选择一个标签，Hackvertor 会自动进行转码并且将结果显示在输出框中。如果你想再次转码的话，旁边还有一个 convert 按钮。标签也有分类，比如 Encode 和 Decode 等，这样你就可以很容易找到你要的标签。Hackvertor 的选项卡也跟 repeater 一样，让你可以多次使用这个工具。Hackvertor 选项卡如图所示：



绕过 Cloudflare WAF

最近，我在 repeater 中发起实际的请求中，使用了 Hackvertor 的标签功能。你可以在 repeater 请求中单击右键，选择 Hackvertor 菜单，在请求中添加标签，在请求发送到服务器之前，它会自动进行转码。当然，你也可以在 Intruder 中使用，不过要先在 repeater 中定义，然后发送到 Intruder 中，这个用法非常强大，因为你可以使用占位符来进行多重编码。你甚至可以在 proxy 选项中使用它，不过这在默认情况下是不开启的。如果你想开启的话，你可以在 Burp 菜单中选中 Hackvertor，然后点击添加到 proxy 选项卡中。

现在，我就来展示一下如何在 repeater 中使用标签来绕过 Cloudflare WAF。将下面的 url 发送到 repeater 中 https://waf.party/xss/xss.php?x=

在等号后面输入下面的代码：

```
<img/src/onerror=alert(1)>
```

在 repeater 中选中 alert(1)，右键单击选中的文本，然后选择 Hackvertor，然后选择 xss，选择 throw_eval。这会在请求中添加标签，添加完之后，选择 go，你就可以在响应的内容中看到包含如下内容：

```
<img/src/onerror=window.onerror=eval;throw'=alert\x281\x29'>
```

如果你想验证它是否确实有效，你只需要在 request 中右击，然后在上下文菜单中选择 Hackvertor 并且复制 url，这会生成一个所有标签都被转码之后的 url。如果你仅仅只是使用 Burp 的 copy URL 命令，它只会复制带有标签的 url。

解码 rotN

研究这个是起源于我女儿的一个问题。当时我穿着 2016 年 BSides Manchester 会议的一件 T 恤，T

恤正面有很多行二进制数据，也就是 01010101 等等，然后，我女儿就问我，“老爸，这些数字都是什么意思啊？”，我告诉她这些都是二进制数据并且问她想不想对这些数据进行解码。然后，我们就开始将这些数据输入到 Hackvertor 中，我发现，这些二进制被解码后，乍一看是一些 base64 编码的字符串，不过，它又看起来像是 rot 编码的字符串，经过测试确实是 rot 编码，于是我们破解了这些 rot 编码的字符串并且得到了信息内容。重点不是这个，而是，这件事引发了我的思考，如果 Hackvertor 能够自动解码 rot 编码的字符串就太好了。

这就需要从随机的杂乱无章的语句中识别英语。为此，我创建了一个 is_like_english 标签，一开始我以为可以使用二元字母模型和三元字母模型来查找常见字母并且识别出英语单词，但是并没有我想的这么简单。于是我就 Google 了一番，发现了一个很不错的 网站 。他们使用的是 4 个字母的四元序列和 4 字母序列在英语中出现的频率。网站上还有一些简单的 Python 脚本，根据对单词和四元字母组合的分析得到一个分数。我借鉴了这些思路，用 Java 重写了代码并且在 Hackvertor 中得以实现。

下一步是完善自动解码器。自动解码器也是一种标签，它可以自动判断字符串是如何编码的并且能够对其进行多次解码。我写了一个简单的正则表达式，用来查找 on 或者更多的 a-z 字符，后面紧跟着空格，逗号或者连字符。然后循环 25 次来暴力破解 rot 编码的字符串，并从每一次的破解中得到一个分数，我计算了平均值，如果最高的得分比平均值还高出 20 的话，那么它就可以自动解码 rot 编码了。

下面这张图片就是我当时穿的 T 恤，这里我就不告诉大家这些二进制解码后的字符串了，有兴趣的话可以自己去解码：

<@auto_decode_0>01010111 01101101 00110101 01101000 01100011 01001000 01010110 01111001 01011010 01101101 01100100 01111001 01011010 01010011 01000010 01000111 01100101 01101101 00110101 00110101 01100101 01010011 00110001 01000111 01100011 01000111 00110101 00110101 01100011 01101001 01000010 01010011 01100001 00110010 01001110 01111001 01011010 01011000 01011010 00110110 01100011 01101101 01000110 01101110 01100010 01101110 01101011 01100111 01010111 01101101 00110101 01110111 01100100 01011000 01011010 01101000 01100011 01100111 00111101 00111101<@/auto_decode_0>

James 也有一件这样的 T 恤，不过上面的二进制不太一样，所以我把他 T 恤上的二进制数据输入到了 Hackvertor 中，看看是否能够自动解码出消息，最终成功的解码出来了。二进制数据如下，你可以自己粘贴到输入框中试一试

<@auto_decode_10>01011010 01101110 01100001 01110000 01110101 01110010 01100110 01100111 01110010 01100101 00101100 00100000 01100110 01100010 00100000 01111101

```
0 01101000 01110000 01110101 00100000 01100111 01100010 00100000 01101110 0110000
1 01100110 01101010 01110010 01100101 00100000 01110011 01100010 01100101<@/auto_
decode_10>
```

解密 XOR 重复密钥加密

本来我准备这篇博客就写到这里了，不过 James 又提出了挑战，问我能不能够破解重复 XOR 加密。于是，我在一个非常优秀的网站 practical cryptography 上学习了所有关于 XOR 和频率分析的知识。第一步就是判断密钥的长度，你可以使用频率分析方法来对每一个 key 进行判断。这里我使用 30 作为最大的密钥长度来进行猜测。我将每一个字符存储在一个频率表中，并且每次当他们出现在密文中时对它们进行递增。当你得到了所有字符的频率时，你就可以计算出每列和频率的重合指数 (或者叫汉明距离)。当我获得了每个 key 的重合指数 (IC)，我就得到了 top7 的 key，并且通过除以选定的密钥长度来将 IC 规范化，然后使用 IC 对前 top7 进行排序，并将 top1 作为猜测密钥返回。

我花了大量时间来尝试提高密钥猜测的准确性，并且多次重写代码。trustedsignal 网站上有一篇 博客 写到，你可以使用第五和第六位的最大公分母来提高检测密钥长度的准确性，不过在我的测试中，却无法提高准确性，不知道为什么。不管怎样，只要你得到了密钥长度，只需要循环遍历密文和每个字符并

对其进行 XOR，并根据每个字符的结果来得到一个分数。这里我是用的是 Alexey Hellman 的 Python 脚本 来对我的代码进行 XOR 的。

最后我重用了 is_like_english 函数来判断文本的分数来确定是否转换成功。如果文本内容不多，确实可以成功，但是如果内容太多的话就失败了，因为文本越长，ngram（多元模型）的得分就越低，所以我将固定值更改为平均值之间的差异百分比，这样无论密文多长都能成功。不过，对于太短的密文，XOR 解密还是会失败，我想可能是因为密文长度不够，所以无法执行频率分析来判断密钥长度和每个解密字符的分数。

为了演示自动解码功能，我已经使用密钥对一段文本进行了 XOR 操作并且进行了 hex 编码，当你在输入框中输入这段密文时，Hackvertor 会自动进行 hex 解码，猜测密钥长度然后自动破解 XOR 加密，甚至还会提供再次编码的正确的标签，如下所示：

```
<@auto_decode_8>1C090C1E05041C101C523D296324212F000D020C04061D001C216F363836
6823161906452101060637 6F3724732E080D0F561617171A003B3B3A6B3630110C18031717074
F1037292C39366808174C0545061B00523E2E372E7D68231A4B03161B1A0852313A373F3A2606
4E0E120217541C1133212D223D2F41170E150D1C1B031D35366F6B2A27144308170B521D0B17
3C3B2A2D2A68150B0E5613170616523E2E372E203C41151E1A0B17060E103B232A3F3A2D124D
4B391000541D17212A223930200411185603001 11E07372137272A68140D08191317064F10202
E2D2F732604144B00101E1A0A00332D2A273A3C1843081A0401070A01723B2B2A27682316190
6451B074F063A2A632D3A3A12174B020A52060A023D3D3765<@/auto_decode_8>
```

现在人们经常使用重复异或加密，希望这个功能能够防止某些应用程序使用这个看起来合法的加密方式。

谢谢大家！

本文翻译自：https://portswigger.net/blog/bypassing-wafs-and-cracking-xor-with-hackvertor 如若转载，请注明原文地址： https://www.4hou.com/tools/14353.html

全文完