35C3CTF 高质量逆向题Juggle 详解

dotsu / 2019-01-07 09:30:00 / 浏览数 9932 安全技术 CTF 顶(0) 踩(0)

前言

Juggle是35C3
CTF中的一道中等难度的逆向题目。虽然由于多个非预期解的存在（也可能是作者故意为之），使得最后题目的动态分数只有90分，但题目整体的考察点比较全面，包括XSL
这里我将自己的解法和几种非预期解整理了一下，供大家参考。

初识XSLT

题目给出了一个XSLT文件和一个Dockerfile，Dockerfile用来配置XSLT环境，主要逻辑都在XSLT文件中。

XSLT是一种语言，用来处理XML格式的数据，将其转换为其他格式，例如HTML等。更多详细信息可以参考w3school。

题目给出的docker中使用了xalan对XSLT进行解释。本地测试时也可以使用Visual Studio（部分版本）进行调试，可以设置断点和观察变量等，十分方便。

非预期解1 —— XXE

首先打开XSLT文件，大致浏览一下，可以找到读取flag的位置：

```
<xsl:when test="count($c/■■■) = 1">
   <xsl:if test="count($chef-drinks) = 0">
       <xsl:copy-of select="document('/flag')"/>
   </xsl:if>
   <xsl:call-template name="consume-meal"> <!-- ■■■■consume-meal■■ -->
       <xsl:with-param name="chef-drinks" select="$chef-drinks"/>
       <xsl:with-param name="food-eaten" select="$food-eaten + 1"/>
       <xsl:with-param name="course" select="$r"/>
       <xsl:with-param name="drinks" select="$drinks"/>
   </xsl:call-template>
</xsl:when>
```

可以看到当满足count($c/■■■) = 1和count($chef-drinks) = 0两个条件时，flag的内容就会被读取到转换后的文件并输出给我们。
这里便存在第一个非预期解——XXE，即XML外部实体注入。这里引用一下OpenToAll的wp：

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE foo [
<!ELEMENT foo ANY >
<!ENTITY xxe SYSTEM "file:///flag" >]><foo>&xxe;</foo>
```

即可成功读取到flag。网上有关XXE的资料很多，这里不过多介绍。

梳理程序逻辑

我们继续来看读flag的条件，其中的$c可以在前面找到：

```
<xsl:template name="consume-meal"> <!-- ■■consume-meal■■ -->
   <xsl:param name="chef-drinks"/> <!-- ■■■■1 -->
   <xsl:param name="food-eaten"/> <!-- ■■■■2 -->
   <xsl:param name="course"/> <!-- ■■■■3 -->
   <xsl:param name="drinks"/> <!-- ■■■■4 -->
   <xsl:if test="$food-eaten > 30000">
       <xsl:message terminate="yes">You ate too much and died</xsl:message>
   </xsl:if>
   <xsl:if test="count($drinks) > 200">
       <xsl:message terminate="yes">You cannot drink that much</xsl:message>
   </xsl:if>
   <xsl:if test="count($course) > 0">
       <xsl:variable name="c" select="$course[1]" /> <!-- ■course■■■■■ -->
       <xsl:variable name="r" select="$course[position()>1]" /> <!-- ■■■■■■■■■ -->
           <xsl:choose> <!-- ■■■■c■■■■■■■■■■■■■■■■ -->
               <xsl:when test="count($c/■■■■) = 1">
               ...
               </xsl:when>
               <xsl:when test="count($c/paella) = 1">
```

```
            ...
            </xsl:when>
            ...
        </xsl:choose>
    </xsl:if>
</xsl:template>
```

这里的xsl:template可以理解为函数，即定义了一个名为consume-meal的函数，其中的几个xsl:param为函数参数。

之前的c变量为course[1]，即course数组中第一个元素(XSLT中数组下标从1开始)。而course数组剩下的元素被放到了r变量中。

在每个xsl:when分支判断中，通过count($c/...)是否为1判断了c的元素类型，并进行不同的处理，最后将r作为course参数，递归调用了consume-meal函数。

所以整个consume-meal相当于对course数组中的每个元素依次进行了一个很大的case分支判断。

那么这些case分支到底在干什么呢？我们可以再观察一下其他几个变量：

- food-eaten在每次递归调用会加一，到30000后会报错退出，相当于限制了输入参数courses的长度。
- drinks参数比较复杂，在许多分支中都会进行修改和判断，其中比较明显的有几个加减乘除的操作，与基于栈的四则运算很像。
- chef-drinks只有在■■■■分支中会进行修改，在■■■■分支中会与drinks进行比较，最终在■■■分支中判断为空便可以读取flag。

首先看■■■■分支：

```
<xsl:when test="count($c/■■■■) = 1">
    <xsl:variable name="arg0">
        <value>
            <xsl:value-of select="$drinks[1] + 0"/>
        </value>
    </xsl:variable>
    <xsl:call-template name="consume-meal">
        <xsl:with-param name="chef-drinks" select="$chef-drinks[position() > 1 or $chef-drinks[1] != $arg0]"/>
        <xsl:with-param name="food-eaten" select="$food-eaten + 1"/>
        <xsl:with-param name="course" select="$r"/>
        <xsl:with-param name="drinks" select="$drinks[position() > 1]"/>
    </xsl:call-template>
</xsl:when>
```

可以看出，当drinks[1] != chef-drinks[1]时，传入下一步的chef-drinks值为$chef-drinks[position() > 1 or 1]，即原来的chef-drinks会被完整传入。
而drinks[1] == chef-drinks[1]时，传入的是$chef-drinks[position() > 1 or 0]，即删掉了chef-drinks的第一个元素。

所以■■■■分支判断了drinks[1]与chef-drinks[1]是否相等，相等则删掉chef-drinks[1]。

其实从这里我们就可以看出一点端倪了。读取flag的条件是count($chef-drinks) = 0，而这里又有一个类似于猜数的功能，猜对所有chef-drinks的值就可以将其删光，以满足flag读取的条件。

然后我们来看■■■■分支：

```
<xsl:when test="count($c/■■■■) = 1">
    <xsl:variable name="arg0">
        <value>
            <xsl:value-of select="$drinks[1] + 0"/>
        </value>
    </xsl:variable>
    <xsl:variable name="chefvalue">
        <value>
            <xsl:value-of select="$chef-drinks[1] + 0"/>
        </value>
    </xsl:variable>
    <xsl:variable name="newdrinks">
        <value>
            <xsl:value-of select="1 * ($arg0 > $chefvalue)"/>
        </value>
        <xsl:copy-of select="$drinks[position() > 1]"/>
    </xsl:variable>
    <xsl:call-template name="consume-meal">
        <xsl:with-param name="chef-drinks" select="$chef-drinks"/>
        <xsl:with-param name="food-eaten" select="$food-eaten + 1"/>
        <xsl:with-param name="course" select="$r"/>
        <xsl:with-param name="drinks" select="exsl:node-set($newdrinks)//value"/>
    </xsl:call-template>
```

```
</xsl:when>
```

可以看出，这里会将`drinks[1] > chef-drinks[1]`的结果与原来的`drinks`拼在一起，作为新的`drinks`参数。
到这里就基本一目了然了，这其实就相当于汇编语言中的`cmp drinks[1],chef-drinks[1]`指令，将比较的结果放到了`drinks`的第一位，所以这个大的分支判断其实就是一个基于栈的VM的解释器。

其中分支中判断的菜名其实就是指令名，`drinks`就是栈，栈的初始值使我们可以控制的，我们需要通过VM中提供的指令猜出`chef-drinks`中所有的值，就可以成功拿到fl

通过分析代码，我们可以把所有指令的作用还原：

- 宫保鸡丁
  - 打印出drinks和chef-drinks的值，相当于打log
- paella
  - push(const)
- □□□
  - push(drinks[pop(drinks) + 2])
- Борщ
  - if drinks[1] == chef-drinks[1], pop(chef-drinks)
- □□□
  - if chef-drinks.len == 0, print_flag
- ラーメン
  - push(pop(drinks) > chef-drinks[1])
- stroopwafels
  - push(pop(drinks) < pop(drinks))
- köttbullar
  - insert_val_into_pos(pop(drinks), pop(drinks))
- γύρος
  - remove(pop(drinks) + 2)
- rösti
  - push(pop(drinks) + pop(drinks))
- □□□□□□□
  - push(pop(drinks) - pop(drinks))
- poutine
  - push(pop(drinks) + pop(drinks))
- □□□□□□
  - push(pop(drinks) + pop(drinks))
- æblegrød
  - if pop(drinks), goto block[pop(drinks) + 1]

注意`course`相当于`block`，通过`æblegrød`指令可以进行分支跳转，跳转到不同的block。

非预期解2——预测随机数

现在题目的目标已经明确了，我们再回头看一下函数调用的初始值：

```
<xsl:template match="/meal">
   <all>
       <xsl:if test="count(//plate) > 300">
           <xsl:message terminate="yes">You do not have enough money to buy that much food</xsl:message>
       </xsl:if>
       <xsl:variable name="chef-drinks">
           <value>
               <xsl:value-of select="round(math:random() * 4294967296)"/>
           </value>
           <value>
               <xsl:value-of select="round(math:random() * 4294967296)"/>
           </value>
           <value>
               <xsl:value-of select="round(math:random() * 4294967296)"/>
           </value>
           <value>
               <xsl:value-of select="round(math:random() * 4294967296)"/>
           </value>
           <value>
               <xsl:value-of select="round(math:random() * 4294967296)"/>
```

```
            </value>
        </xsl:variable>
        <xsl:call-template name="consume-meal">
            <xsl:with-param name="chef-drinks" select="exsl:node-set($chef-drinks)//value"/>
            <xsl:with-param name="food-eaten" select="1"/>
            <xsl:with-param name="course" select="course[position() = 1]/plate"/>
            <xsl:with-param name="drinks" select="state/drinks"/>
        </xsl:call-template>
    </all>
</xsl:template>
```

可以看到chef-drinks中共有5个值，均由math:random()生成。这里便产生了第二个非预期解——预测随机数。

通过研究xalan对于random函数的实现，或者直接连续运行两次程序比较随机数都可以发现，生成随机数的种子其实就是当前的时间。而且实际上xalan就是使用的c中标准

所以这里就有两种做法，一种是直接调用c中的srand(time(NULL))和rand()。另一种是使用VM中的log指令得到当前时间生成的随机数，然后马上用得到的这些数生成XSLT

两种方法的实现可以参考这两份WP：
https://ctftime.org/writeup/12780
https://teamrocketist.github.io/2018/12/30/Reverse-35C3-juggle/

预期解——二分查找

基于之前的分析，在VM中，我们不能直接获取chief-drinks或对齐进行运算，只能将其与一个值进行比较得到大小关系。到这里，标准解法就已经很明显了，那就是使用

```
public int guessNumber(int n) {
    int left = 1;
    int right = n;
    int mid = 0;
    while(left<=right){
        mid = left + (right-left)/2;
        if(guess(mid)==0) return mid;
        else if(guess(mid)==1) left = mid+1;
        else if(guess(mid)==-1) right = mid-1;
    }
    return mid;
}
```

所以我们剩下的工作，就是把二分查找的代码翻译成题目VM对应的代码。这部分其实是题目最麻烦的一步，但是难度并不大，需要我们写汇编代码和进行debug。
当然也可以考虑使用一些基于栈的语言（例如python）编译后再转换过去。这里我是自己从头写的，大家也可以尝试不同的做法。

下面是我的最终实现代码，其中有注释代码对应的功能，这里我就不再详述代码细节。注意调试时可以多使用log指令（宫保鸡丁），对于debug非常有帮助。另外Visual Studio调试时会发生栈溢出，可以直接用xalan跑。

最终解：

```
<?xml version="1.0" encoding="UTF-8"?>
<meal>
 <course>
   <!-- stack [st, ed, 0, MAX] -->
   <!-- push s[0] (t = st) -->
   <plate>
     <■■■></■■■>
   </plate>
   <plate>
     <■■■■></■■■■>
   </plate>
   <plate>
     <paella>0</paella>
   </plate>
   <plate>
     <■■■></■■■>
   </plate>

   <!-- s[0] += 2 (t += 2)-->
   <plate>
     <paella>2</paella>
   </plate>
   <plate>
     <rösti></rösti>
```

```
</plate>

<!-- push s[2] (t2 = ed)-->
<plate>
  <paella>2</paella>
</plate>
<plate>
  <[redacted]></[redacted]>
</plate>

<!-- push s[1] < s[2] (cmp t2, t)-->
<plate>
  <stroopwafels></stroopwafels>
</plate>

<!-- if s[1], jmp course2 (jl course2) -->
<plate>
  <paella>1</paella>
</plate>
<plate>
  <paella>2</paella>
</plate>
<plate>
  <köttbullar></köttbullar>
</plate>
<plate>
  <æblegrød></æblegrød>
</plate>


<!-- push 2 (pos = 2) -->
<plate>
  <paella>1</paella>
</plate>

<!-- push 2 (t = 2) -->
<plate>
  <paella>2</paella>
</plate>

<!-- push s[3] (t2 = ed) -->
<plate>
  <paella>3</paella>
</plate>
<plate>
  <[redacted]></[redacted]>
</plate>

<!-- push s[3] (t3 = st) -->
<plate>
  <paella>3</paella>
</plate>
<plate>
  <[redacted]></[redacted]>
</plate>

<!-- push s[1] + s[2] (t2 += t3)-->
<plate>
  <rösti></rösti>
</plate>

<!-- push s[1] / s[2] (t2 /= t)-->
<plate>
  <[redacted]></[redacted]>
</plate>

<!-- push s[0] (mid = t2) -->
<plate>
  <paella>0</paella>
```

```
  </plate>
  <plate>
    <■■■></■■■>
  </plate>


  <!-- push s[1] > cd[1] (cmp t2, cd[1] )-->
  <plate>
    <■■■■></■■■■>
  </plate>

  <!-- if s[1], jmp course1 (jg course1) , mid > cd[1] -->
  <plate>
    <paella>1</paella>
  </plate>
  <plate>
    <paella>1</paella>
  </plate>
  <plate>
    <köttbullar></köttbullar>
  </plate>
  <plate>
    <æblegrød></æblegrød>
  </plate>
  <!-- mid <= cd[1], s = [pos, mid, st, ed, 0, MAX] -->
  <!-- insert(pos, mid) -->
  <plate>
    <köttbullar></köttbullar>
  </plate>
  <!-- push 0 (pos = 0)-->
  <plate>
    <paella>0</paella>
  </plate>

  <!-- remove(pos+1) -->
  <plate>
    <γ■ρος></γ■ρος>
  </plate>

  <!-- jmp course0 -->
  <plate>
    <paella>0</paella>
  </plate>
  <plate>
    <paella>1</paella>
  </plate>
  <plate>
    <æblegrød></æblegrød>
  </plate>
</course>

<course>
  <!-- mid > cd[1], s = [pos, mid, st, ed, 0, MAX] -->
  <!-- insert(pos, mid) -->
  <plate>
    <köttbullar></köttbullar>
  </plate>
  <!-- push 2 (pos = 2)-->
  <plate>
    <paella>2</paella>
  </plate>

  <!-- remove(pos+1) -->
  <plate>
    <γ■ρος></γ■ρος>
  </plate>

  <!-- jmp course0 -->
  <plate>
```

```
    <paella>0</paella>
   </plate>
   <plate>
    <paella>1</paella>
   </plate>
   <plate>
    <æblegrød></æblegrød>
   </plate>
  </course>

  <course>
   <!-- st = cd[1], stack [st, ed, 0, MAX] -->
   <!-- pop(cd[1] ) -->
   <plate>
    <■■■■></■■■■>
   </plate>

   <!-- remove(1) -->
   <plate>
    <paella>0</paella>
   </plate>
   <plate>
    <γ■ρος></γ■ρος>
   </plate>


   <!-- push s[0] (ed = MAX) -->
   <plate>
    <paella>1</paella>
   </plate>
   <plate>
    <■■■></■■■>
   </plate>

   <!-- push s[0] (st = 0) -->
   <plate>
    <paella>1</paella>
   </plate>
   <plate>
    <■■■></■■■>
   </plate>
   <!-- jmp course0 -->
   <plate>
    <paella>0</paella>
   </plate>
   <plate>
    <paella>1</paella>
   </plate>
   <plate>
    <æblegrød></æblegrød>
   </plate>
  </course>

  <state>
   <!-- stack init -->
   <drinks>
    <value>0</value>
   </drinks>
   <drinks>
    <value>4294967296</value>
   </drinks>
   <drinks>
    <value>0</value>
   </drinks>
   <drinks>
    <value>4294967296</value>
   </drinks>
  </state>
 </meal>
```

也可以参考CTFTIME上其他两个类似解法的WP：
https://tcode2k16.github.io/blog/posts/2018/35c3ctf-writeup/#juggle
https://ctftime.org/writeup/12803

上一篇：某CMS v4.2.3 测试笔记（... 下一篇：CVE-2017-8570复现及编...

1. 0 条回复
   - 动动手指，沙发就是你的了！