Automatic string formatting deobfuscation

mss**** / 2019-01-22 09:17:00 / 浏览数 3741

原文：https://maxkersten.nl/binary-analysis-course/malware-analysis/automatic-string-formatting-deobfuscation/

对于未经编译的恶意软件来说，一旦落入安全分析人员的手里，其源代码就会一览无余了。攻击者当然知道这一点，因此，通常会对源代码进行混淆处理，从而提高样本分析

```
MD5: 907dbc3048f75bb577ff9c064f860fc5

SHA-1: 667b8fa87c31f7afa9a7b32c3d88f365a2eeab9c

SSDeep: 6144:F7EhX4jlKpvFnMt8NKKfoIEFtUVETlqds6YGTC9HIN5Tao0jCGIop1Y6aiiNelyb:pQ39oIpyK+HI+3Npi6aiiNeewudtv

File size: 368.42 KB

File length: 2763 lines
```

在本例中，使用了一种特定类型的混淆处理方法：字符串格式化混淆技术。首先，我们将简单介绍一下混淆的概念，并快速分析该脚本的其他混淆部分。之后，将逐步创建一

样本使用的混淆技术

字符串格式化

一个字符串可以由多个组件组成。通过字面量字符串中的标志，能够让我们在读写格式化设置的时候更加轻松。在C语言中，标志%d用于十进制。下面是一个用C语言编写的

```
int age = 100;
printf("My age equals %d\n", age);
```

上述代码的输出结果如下所示：

```
My age equals 100
```

在C#语言中，类型是从引用变量派生的，具体如下例所示。注意，我们仍然可以使用第二个示例中给出的类型。

```
//Types are derived from the variable
string s = String.Format("{0}'s age is {1}", "Libra", 100);
Console.WriteLine(s);

//Types are specified within the code
string s = String.Format("{0}'s age is {1:f}", "Libra", 100.5);
Console.WriteLine(s);
```

要使用这种方式对一段文本进行混淆处理，只需将其拆分为字符串，并对各个字符串分别进行相应的处理即可。这些字符串会在运行时以正确的顺序进行组装，从而允许程序

```
string something = String.Format("{4}{2}{3}{1}{0}", "ion", "at", "fu", "sc", "ob");
```

其中，something存放的是混淆后的字符串。

字符串串联

一个字符串可以由多个字符或多个字符串组成。通过将其相加，就可以混淆变量的名称。下面给出一个例子。

```
string x = "ab" + "cd";
string y = x + "ef";
```

在这个示例中，x等于abcd，y等于abcdef。如果变量的名称是以这种方式创建的，则该方法还能够阻止代码重构。在普通代码中，这是不可行的，但是，当通过反射来调用

反引号

在PowerShell中，可以在变量名中使用反引号(`)。这些都会被忽略，但是，也会妨碍重构。

正则表达式

要在文本（以及代码）中寻找模式的匹配，可以使用正则表达式。正则表达式的应用非常广泛，并且，目前有许多在线工具可以用来帮助我们编写正则表达式。在该示例中，
expressions）也称为regexes。

自动去混淆

为了实现去混淆过程的自动化，我们可以借助于Python3。首先，分析需要匹配的模式，然后，构造正则表达式。最后，将匹配的数据替换为去混淆过的数据，以便生成具有

查找模式

在检查样本时，发现了上述三种类型的字符串混淆方法。下面，我们看看基于反引号的混淆方法。

```
function iN`VokE`-r`F`BuxmE`HAEmZbhI
```

正如上面所看到的，字符的顺序是正确的，但是，如果不删除或忽略反引号，就无法进行重命名。

样本中的第二种混淆方法，是一种基于字符串格式化的混淆处理，具体如下所示。

```
$I7KUHX  =[tYpe]("{7}{2}{5}{10}{13}{1}{4}{9}{14}{12}{0}{3}{11}{6}{8}"-f 'mAR','opse','yst','sh','R','eM.RuNti','T','S','E','v'
```

其中，字符串本身为System.Runtime.InteropServices.MarshalasAttribute。需要注意的是，由于这里使用了[type]，所以，它会查找给定名称下的类型。[type]返回的值等

上面样本中的第三种也是最后一种混淆类型是字符串串联。通过将字符串拆分为不同的子串，可以把同一个名称变成不同的形式，如下所示。

```
("VaRIA"+"BLE"+":Rb"+"h0")
("Va"+"RI"+"ABL"+"E:Rbh"+"0")
```

清除反引号

由于反勾号在脚本中使用不当，因此，可以使用Python3中的字符串替换函数来删除所有反引号。

```
powershellPath = 'powershellSample.txt'
powershellFile = open(powershellPath,'r')
powershellContent = powershellFile.readlines()
for line in powershellContent:
   line = line.replace("`", "")
```

清除字符串格式化符

使用同一个模式，就能找到字符串字面量（第一部分）和原始字符串的各个部分（添加的变量），下面举几个例子。

```
&("{0}{2}{1}"-f'sE','-iTeM','T')
$7eq=  [tyPe]("{1}{0}" -f '32','INT')  ;
&("{1}{0}{2}"-f 'eT-','S','itEM')
$j1v  =[tyPe]("{0}{1}"-F 'Co','NveRt') ;
```

通过这个例子，可以发现：

- 每个格式选项都以左括号和引号开头，即（和“
- 花括号用于决定字符串格式化的顺序。
- 会用到格式标志-f及其大写字母版本-F
- 有时格式标志前面会有空格。
- 有时格式标志后面会有空格。

基于上面的观察，正则表达式以引号开头。然后，由于长度未知，所有花括号之间的数字需要匹配n次。并且，它还应以引号结尾，具体见下面的正则表达式：

```
"((?:\{\d+\})+)"
```

其中，“(?:”和“)”表示不应捕获的组；“\”用于对花括号等字符进行转义；d用来匹配一个数字。如果要匹配多个数字，请使用“+”。在上面的示例中，右括号后的“+”表示匹配一

上面的表达式中，外面用括号将其括起来，这表明这一组应该被捕获。索引仍然被括在一对花括号之间，是表达式的第一个捕获组（位于索引0处）。在后面的步骤中，将从

在此之后，是格式标志，需要注意的是，在格式标志的前后可能会遇到空白符。当然，空白符的长度是未知的。正则表达式中的“”与“+”的功能类似，因为它也是用于重复前

要检测单个空白符、格式标志（包括大小写形式）与更多空白符，需要使用另一个正则表达式，具体如下所示。

```
\s*-[fF]\s*
```

其中，“%s”语句与空白符匹配。结合“*”，空白符的长度可以是任意的，甚至是零。而破折号（-）则与“[”和“]”之间的任意字符匹配。在这个例子中，字符f或F是匹配的。

正则表达式的最后一部分匹配放在单引号之间的字符。引号（前引号和后引号）可以使用其字面量字符进行匹配，中间的字符可以使用点（.）号进行匹配——匹配除行尾外

```
((?:'.*?',?)+)
```

在这里，我们并没有捕获单个字符串（使用“(?:”和“)”），而是捕获完整的结果集，因为"+"是用于匹配一个或多个部分的。“.*?”表示对任意字符匹配零次或多次，如有可能，

完整的正则表达式如下所示。

```
"((?:\{\d+\})+)"\s*-[fF]\s*((?:'.*?',?)+)
```

最后，需要从上述正则表达式的匹配中提取索引和字符串。两个捕获组的输出如下所示：

```
'{7}{2}{5}{10}{13}{1}{4}{9}{14}{12}{0}{3}{11}{6}{8}'
'mAR','opse','yst','sh','R','eM.RuNti','T','S','E','v','ME.','ALAsATtrIbu','ces.','intEr','I'
```

要查找所有不带括号的索引，必须在花括号之间匹配一个或多个数字，具体如下面的正则表达式所示。

```
{(\d+)}
```

这时，它会捕获一个或多个数字，同时忽略花括号。需要注意的是，由Python3的正则表达式包返回的索引列表仍被视为字符串列表。要将索引转换为整数，可以使用map函

```
for line in powershellContent:
   matchedLine = re.findall("""((?:\{\d+\})+)"\s*-[fF]\s*((?:'.*?',?)+)""", line)
   if len(matchedLine) > 0:
       for match in matchedLine:
           indices = list(map(int, re.findall("{(\d+)}", match[0])))
           strings = re.findall("'([^']+?)'", match[1])
           result = "".join([strings[i] for i in indices])
           line = line.replace(match[0], result, 1)
           line = line.replace(match[1], "", 1)
```

直接替换每一行中的-f和-F字符是可能的，但是这样做会破坏代码中的某些名称，具体如下所示。

```
[IntPtr]${nE`WTH`Unkr`ef} = &("{0}{6}{1}{4}{5}{3}{2}" -f 'G','Remo','s','Addres','te','Proc','et-') -RemoteProcHandle ${R`eMOt
```

"其中，-FunctionNamePtr也以-F开头。为了避免这种情况，还可以使用正则表达式。我们应删除格式标志，并且，仅当其后面没有字符时才应删除。在这里，空格或括号并

为了匹配格式标志，需要使用之前创建的正则表达式(-[fF])。然后，使用正则表达式中的look
ahead语句检查匹配后面的字符。这是一个新的捕获组，以"?="开头。除了字符"a"到"z"、"A"到"Z"、"0"到"9"和"_"之外的所有字符都可以使用"[^\w]"进行匹配。其中，"^

```
(-[fF])(?=[^\w])
```

若要从示例中删除格式标志，请使用空字符串替换此正则表达式的匹配项。

```
formatFlag = re.findall("""(-[fF])(?=[^\w])""", line)
           if len(formatFlag) > 0:
               for formatFlagMatch in formatFlag:
                   line = line.replace(formatFlagMatch, "")
```

清除串联字符串

通过添加多个字符串而生成的字符串总是具有相同的布局：

- 它以一个左括号开头
- 它以右括号结束。
- 每个字符串都用引号括起来。
- 将两个字符串相互串联的运算符是加号（"+"）。

首先，可以检查引号前的字符是否是左括号。为此，可以使用look behind语句。

```
(?<=\()\"
```

请注意，引号和右括号会被转义，因此，要使用反斜杠。

我们已经知道，可以用look ahead语句检查引号后面的字符。不过，这些字符不应与右括号匹配，而应与加号匹配。

```
(?=[^\)]+\+[^\)]+\))
```

然后，任何不是花括号、连字符或右括号的内容都应匹配，匹配次数应尽可能少，但是至少匹配一次。这些就是我们需要的数据，但并没有放在单独的捕获组中（这里使用了

```
(?:[^\{\}\-\)])+
```

字符串后的引号后面应跟一个右括号，以表示变量结束。

```
\"(?=\))
```

完整的正则表达式如下所示：

```
(?<=\()\"(?=[^\)]+\+[^\)]+\))(?:[^\{\}\-\)])+\"(?=\))
```

完全匹配的内容如下所示：

```
"V"+"Ari"+"Ab"+"LE:cF84"
```

为了用串联的字符串正确替换相应的字符串，应删除所有引号、加号和空格。之后，应该在变量的前面和末尾放一个单引号。通过搜索和替换给定的值，可以对其进行重构。

```
varDeclaration = re.findall("""(?<=\()\"(?=[^\)]+\+[^\)]+\))(?:[^\{\}\-\)])+\"(?=\))""", line)
            variable = ''
            if len(varDeclaration) > 0:
                for string in varDeclaration:
                    variable = string.replace("\"", "")
                    variable = variable.replace("+", "")
                    variable = variable.replace(" ", "")
                    variable = "\"" + variable + "\""
                line = line.replace(varDeclaration[0], variable)
```

最后，将所有行都保存在一个新文件中。

```
with open('deobfuscatedSample.txt', 'w') as f:
   f.write(output)
```

下面给出完整的Python3脚本及相关的注释。

```
#Created by Max 'Libra' Kersten (@LibraAnalysis)

import re

#Define information regarding the original script's location
powershellPath = 'powershellSample.txt'
powershellFile = open(powershellPath,'r')
#Read all lines of the original script
powershellContent = powershellFile.readlines()

#The variable which contains all deobfuscated lines
output = ''
#The variable which keeps track of the amount of string formats that have been replaced
formatCount = 0
#The variable which keeps track of the amount of variables that have been replaced
variableCount = 0
#The variable which keeps track of the amount of removed back ticks
backtickCount = 0

#Loop through the file, line by line
for line in powershellContent:
   backtickCount += line.count("`")
   #Replace the back tick with nothing to remove the needless back ticks
   line = line.replace("`", "")
   #Match the string formatting
   matchedLine = re.findall("""((?:\{\d+\})+)"\s*-[fF]\s*((?:'.*?',?)+)""", line)
   #If one or more matches have been found, continue. Otherwise skip the replacement part
   if len(matchedLine) > 0:
       #Each match in each line is broken down into two parts: the indices part ("{0}{2}{1}") and the strings ("var", "ble", "
       for match in matchedLine:
           #Convert all indices to integers within a list
           indices = list(map(int, re.findall("{(\d+)}", match[0])))
           #All strings are saved in an array
           strings = re.findall("'([^']+?)'", match[1])
           #The result is the correctly formatted string
           result = "".join([strings[i] for i in indices])
           #The current line is altered based on the found match, with which it is replaced
           line = line.replace(match[0], result, 1)
           line = line.replace(match[1], "", 1)
           #Regex the "-f" and "-F" so that "-f[something]" is not replaced
           formatFlag = re.findall("""(-[fF])(?=[^\w])""", line)
           if len(formatFlag) > 0:
               for formatFlagMatch in formatFlag:
                   line = line.replace(formatFlagMatch, "")
           #Find all strings between quotation marks.
           varDeclaration = re.findall("""(?<=\()\"(?=[^\)]+\+[^\)]+\))(?:[^\{\}\-\)])+\"(?=\))""", line)
           #The concatenated variable
           variable = ''
           #For each string in the list, the items are concatenated
           if len(varDeclaration) > 0:
               for string in varDeclaration:
```

```
                    variable = string.replace("\"", "")
                    variable = variable.replace("+", "")
                    variable = variable.replace(" ", "")
                    variable = "\"" + variable + "\""
                    variableCount += 1
            #Replace the variable with the concatenated one
                line = line.replace(varDeclaration[0], variable)
            formatCount += 1
   #When all matches are done, add the altered line to the output
   output += line
#When all lines are checked, write the output variable to a file
with open('deobfuscatedSample.txt', 'w') as f:
   f.write(output)
print("Amount of removed back ticks:")
print(backtickCount)
print("Amount of formatted strings that have been deobfuscated and concatenated:")
print(formatCount)
print("Amount of variables that have been concatenated:")
print(variableCount)
print("Total amount of modifications:")
print((backtickCount + formatCount + variableCount))
```

要想了解这里修改了多少个值，可以使用count变量。这个脚本的输出如下所示。

```
Amount of removed back ticks:
8634
Amount of formatted strings that have been deobfuscated and concatenated:
1963
Amount of variables that have been concatenated:
51
Total amount of modifications:
10648
```

之前，样本的第一部分如下所示：

```
$I7KUHX  =[tYpe]("{7}{2}{5}{10}{13}{1}{4}{9}{14}{12}{0}{3}{11}{6}{8}"-f 'mAR','opse','yst','sh','R','eM.RuNti','T','S','E','v'
&("{0}{2}{1}"-f'sE','-iTeM','T')
("V"+"Ari"+"Ab"+"LE:cF84") ([TYPe]("{2}{0}{1}{7}{9}{6}{10}{3}{4}{5}{8}" -F'yste','m.RU','s','es','.un','ManaGeDty','rV','nTiME
$7eq=  [tyPe]("{1}{0}" -f '32','INT')  ;
&("{0}{1}" -f's','ET') tIAfhC  ([tyPE]("{0}{1}" -F'bO','ol')  )  ;
&("{0}{1}{2}"-f 's','ET','-VARIaBLE')
kM5l ( [tYPE]("{0}{1}{2}"-F 'U','I','Nt32')  )  ;
$XD1h =[TYpE]("{1}{0}{2}"-f'NVE','BItco','rtEr');
&("{2}{1}{0}"-f'tem','ET-I','s')
("VaRIA"+"BLE"+":Rb"+"h0")  ( [tYPE]("{1}{8}{6}{4}{2}{5}{9}{11}{10}{7}{12}{0}{3}" -F 'S','S','r','s','EM.','EFLecT','t','DEraC
$eGj7  =  [tyPe]("{0}{1}{2}" -F 'aPPDOma','i','N');
&("{1}{0}{2}"-f 'eT-','S','itEM')
VAriablE:tg58U ( [TYpE]("{8}{5}{4}{7}{3}{0}{2}{6}{1}" -F'n','gcOnvENtIoNS','.c','o','.REFLEC','sTeM','ALLin','ti','sy') );
&("{0}{1}" -f 'S','eT-iTEm')
variablE:urYi12 ( [tYPE]("{2}{3}{0}{1}" -F 'I','RONmENt','eN','V')) ;
$9hRwNy  =  [tYpE]("{1}{0}"-f'R','uIntpt') ;
&("{0}{1}{2}" -f'SeT-i','te','m')
("VARI"+"ABLe:6"+"3"+"Y")  ( [tyPe]("{1}{0}" -f'h','MAT') ) ;
$MlHiT=[typE]("{5}{6}{4}{1}{2}{3}{0}"-F 'HAl','OpSe','R','vIcEs.mArs','R','syStEm.RunT','Ime.iNte');
&("{1}{0}" -f 'T','SE')
T2NGf  ( [type]("{0}{2}{1}" -F 'IN','PTR','t')) ;
$j1v  =[tyPe]("{0}{1}"-F 'Co','NveRt') ;
function iN`VokE`-r`F`BuxmE`HAEmZbhI
```

在自动去混淆之后，它就变成具有可读性的代码了：

```
$I7KUHX  =[tYpe]("SysteM.RuNtiME.intEropseRvIces.mARshALAsATtrIbuTE" ) ;
&("sET-iTeM")
("VARIABLe:63Y") ([TYPe]("system.RUnTiME.inTEroPserViCes.unManaGeDtype" )  );
$7eq=  [tyPe]("INT32"  )  ;
&("sET" )
tIAfhC  ([tyPE]("bOol" )  )  ;
&("sET-VARIaBLE" )
kM5l ( [tYPE]("UINt32" )  )  ;
$XD1h =[TYpE]("BItcoNVErtEr");
```

```
&("sET-Item")
("VARIABLe:63Y")  ( [tYPE]("SYstEM.rEFLecTiOn.emIt.ASSEmBlYbUIlDEraCCeSs"  ))  ;
$eGj7  =  [tyPe]("aPPDOmaiN"  );
&("SeT-itEM" )
VAriablE:tg58U ( [TYpE]("sysTeM.REFLECtion.cALLingcOnvENtIoNS" ) );
&("SeT-iTEm"  )
variablE:urYi12 ( [tYPE]("eNVIRONmENt"  )) ;
$9hRwNy  =  [tYpE]("uIntptR") ;
&("SeT-item" )
("VARIABLe:63Y")  ( [tyPe]("MATh" ) ) ;
$MlHiT=[typE]("syStEm.RunTIme.iNteROpSeRvIcEs.mArsHAl" );
&("SET"  )
T2NGf  ( [type]("INtPTR"  )) ;
$j1v  =[tyPe]("CoNveRt" ) ;
function iNVokE-rFBuxmEHAEmZbhI
```

这样一来，安全分析人员就可以轻松地分析和重构脚本，而不用为去混淆而劳心劳力了。

上一篇：Hitcon-Training l... 下一篇：攻击Epic Games接管堡垒之夜账户

1. 0 条回复
   - 动动手指，沙发就是你的了！