CobaltStrike Stager Utilizing Floating Point Math

medium.com/walmartglobaltech/cobaltstrike-stager-utilizing-floating-point-math-9bc13f9b9718

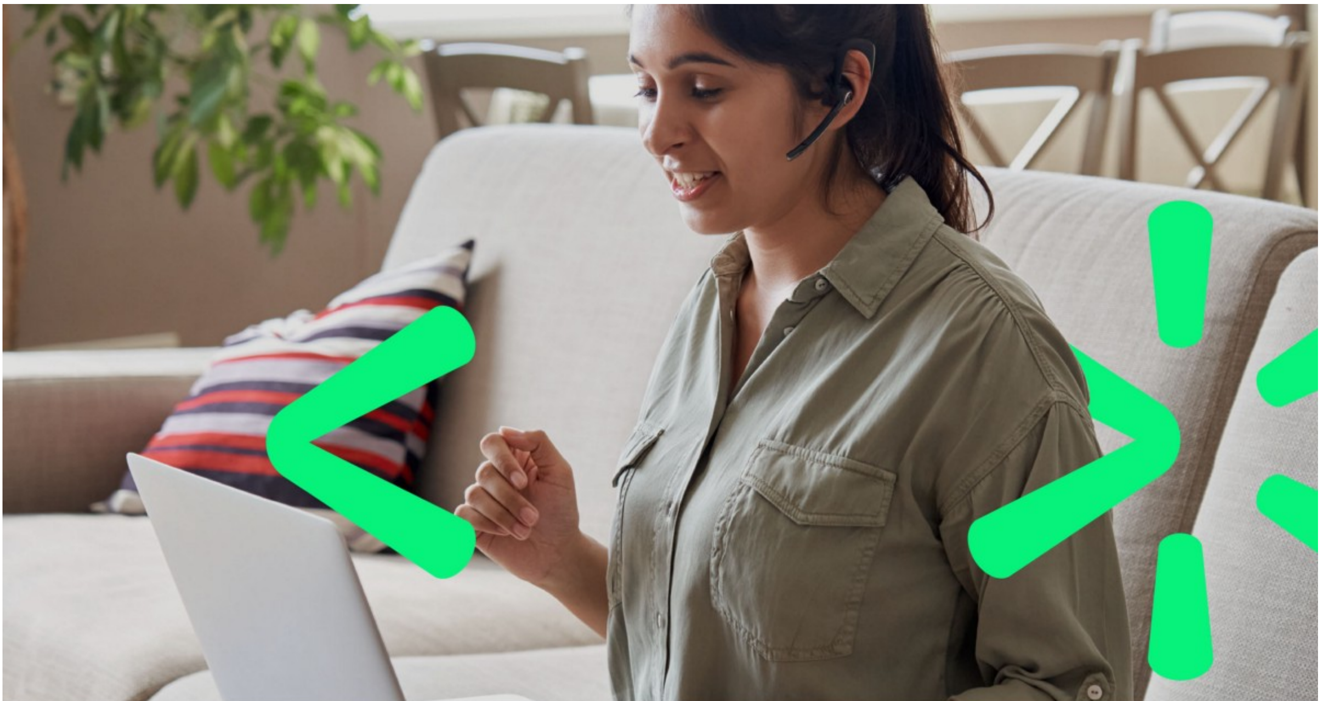
Jason Reaves April 20, 2021

Jason Reaves

Apr 20, 2021

.

3 min read



By: Jason Reaves and Joshua Platt

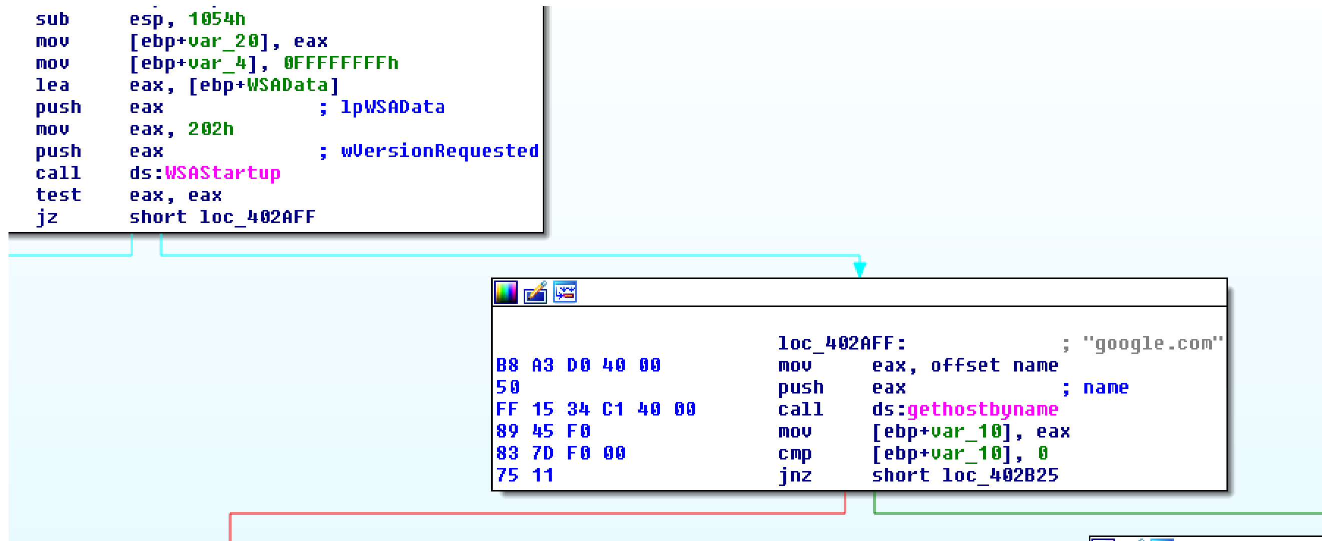
Executive summary

1. New CobaltStrike stagers utilizing floating point mnemonics[1] to decode out stager shellcode.
2. Using raw sockets and date value from Google headers to check overwritten sleep values such as in some sandbox detonations.

Date checking

The stager employs an interesting technique to check for being detonated in controlled environments such as sandboxes that might overwrite sleep values, at the same time it also checks for network connectivity.

The stager utilizes raw sockets to connect to ‘google.com’ over port 80 and send a GET request.



Raw socket to google.com

The request is not parsed as an HTTP request in most utilities including Wireshark[2] and Suricata[3] because it is incomplete with just a newline and no carriage return.

```
mov     eax, offset aGetDrv ; "GET drv\n"
push    eax
lea     eax, [ebp+WSAData]
push    eax
call    sub_4030C0
add     esp, 8
push    0                ; flags
lea     eax, [ebp+WSAData]
call    sub_4030F0
push    eax              ; len
lea     eax, [ebp+WSAData]
push    eax              ; buf
mov     eax, [ebp+s]
push    eax              ; s
call    ds:send
```

Incomplete request

The request is enough to retrieve the 404 response from the webserver and then the malware begins parsing the values out of the date, specifically it parses out the day, year and time values.

```
add     esp, 8
mov     edx, offset aDate ; "Date: "
lea     eax, [ebp+WSAData]
call    loc_403110
mov     [ebp+var_C], eax
mov     edx, ','
mov     eax, [ebp+var_C]
call    FindChar_4031C0
mov     [ebp+var_C], eax
mov     eax, [ebp+var_C]
inc     [ebp+var_C]
mov     eax, [ebp+var_C]
inc     [ebp+var_C]
mov     edx, offset aGmt ; "GMT"
mov     eax, [ebp+var_C]
call    loc_403110
```

Parse values from response

After parsing out the values it converts it to seconds but without accounting for the month.

```
83 C4 1C            add     esp, 1Ch
8D 45 AC            lea     eax, [ebp+var_54]
E8 AB 04 00 00      call    ConvertDate_403230
```

Convert values to seconds

```
sub     esp, 1Ch
mov     [ebp+var_1C], eax
mov     [ebp+var_18], edx
xor     eax, eax
call    GetTimeValueFromGoogle_402AB0
mov     [ebp+var_8], eax
cmp     [ebp+var_8], 0
jge     short loc_401114
```

```
                        loc_401114:             ; dwMilliseconds
68 30 75 00 00          push    7530h
FF 15 C4 C1 40 00       call    ds:Sleep
31 C0                   xor     eax, eax
E8 8A 19 00 00          call    GetTimeValueFromGoogle_402AB0
89 45 FC                mov     [ebp+var_4], eax
83 7D FC 00             cmp     [ebp+var_4], 0
7D 0E                   jge     short loc_40113D
```

```
                        loc_40113D:
8B 45 FC                mov     eax, [ebp+var_4]
2B 45 F8                sub     eax, [ebp+var_8]
83 F8 1C                cmp     eax, 1Ch
7D 0E                   jge     short loc_401156
```

Time Check

Above you can see a sleep call is sandwiched by two of these calls to the function responsible for retrieving the converted value from a google request, the sleep is 30 seconds and then it checks if the values differ less than 28. It is checking if the process took less than 28 seconds or not.

```
E8 63 FF FF FF          call    DisplayDirectXError_4010B0
C7 45 EC 00 00 00 00    mov     [ebp+var_14], 0
EB 4B                   jmp     short loc_4011A1
```

```
                        loc_401156:                 ; flProtect
6A 40                   push    40h
68 00 30 00 00          push    3000h               ; flAllocat:
68 1F 03 00 00          push    31Fh                ; dwSize
6A 00                   push    0                   ; lpAddress
FF 15 CC C1 40 00       call    ds:VirtualAlloc
89 45 F0                mov     [ebp+var_10], eax
BA 1F 03 00 00          mov     edx, 31Fh
B8 AC 11 40 00          mov     eax, offset dword_4011AC
E8 94 FE FF FF          call    DecodeStagerSC_401010
89 45 F4                mov     [ebp+var_C], eax
BB 1F 03 00 00          mov     ebx, 31Fh
8B 55 F4                mov     edx, [ebp+var_C]
8B 45 F0                mov     eax, [ebp+var_10]
```

Error or decode logic

If the check fails then a fake DirectX error message is displayed, otherwise the process for decoding the stager shellcode begins.

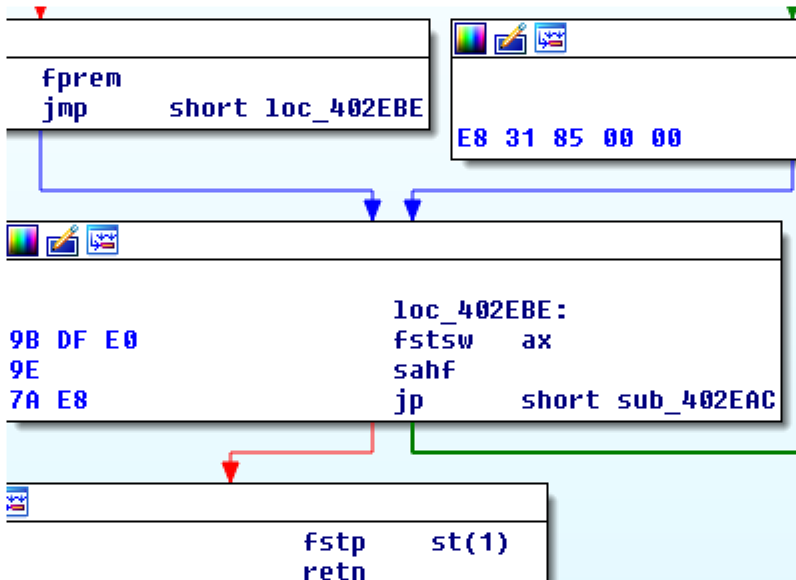
Shellcode decode

The shellcode is decoded by utilizing floating point mnemonics, judging by some of the actors testing this appears to be pretty good at bypassing static detection engines.

```
push    ds:dword_40D194
push    ds:dword_40D190 ; double
mov     eax, [ebp+var_C]
shl     eax, 3
add     eax, [ebp+var_18]
push    dword ptr [eax+4]
push    dword ptr [eax] ; double
call    fpmod_402EC7
fstp    [ebp+var_20]
fld     [ebp+var_20]
call    fp_rndint
fistp   [ebp+var_10]
fild    [ebp+var_10]
fcomp   [ebp+var_20]
fnstsw  ax
sahf
jnb     short loc_401080
```

Decode loop

The process involved begins with floating point modulus against a table of data using a key value that is hardcoded.



fpmod

After the modulus the value is rounded to an int value. Example python code for decoding the data can be seen below:

```
def fpmod_decode(key, data, l): out = "" for i in range(l):  temp =
struct.unpack_from('<d', data[i*8:])[0]  if temp > int(temp%key):   out +=
chr((ord(struct.pack('<Q', int(temp%key))[0])+1)&0xff)  else:   out +=
chr((ord(struct.pack('<Q', int(temp%key))[0]))&0xff) return out
```

Using our decode code we can quickly enumerate samples for decoding out the shellcode and harvesting IOCs.

Indicators of compromise

```
cda7edc9414814ef57c31e473ce87e489bcd6f1ed8d81a504e960e184fce1609abaf70728e6f940195e35e
 tcp $HOME_NET any -> $EXTERNAL_NET 80 (msg:"CS stager time check 1"; dsize:8;
content:"GET drv|0a|"; offset:0; classtype:trojan-activity; sid:9000009; rev:1;
metadata:author Jason Reaves;)alert tcp $HOME_NET any -> $EXTERNAL_NET 80 (msg:"CS
stager time check 2"; dsize:11; content:"GET driver|0a|"; offset:0; classtype:trojan-
activity; sid:9000010; rev:1; metadata:author Jason Reaves;)
```

References