

Dissecting CoreBot, From Loader to Config

Jason Reaves

December 10, 2015

Abstract

While analyzing the CoreBot malware I hope to show my work reverse engineering a number of the routines used by Corebot as well as break down most of the encryption used to conceal various portions of data used by the bot. By doing so we can hope to not only help security researchers gain insight into the workings of this bot but also help incident response personnel with being able to understand the bot in hopes that they can better detect or prevent infections in their respective environments.

Keywords - Reverse Engineering, Malware Analysis, CoreBot, Banking Trojan

```

00401000 8BF9          MOV     EDI,ECX
00401001 53 00F00000  CALL  @FS:00F00000             JMP to MSUCRT.mnset
00401002 0FB646 0F     MOVZX  EAX, BYTE PTR DS:[ESI+F]
00401003 50          PUSH   EAX
00401004 0FB646 0E     MOVZX  EAX, BYTE PTR DS:[ESI+E]
00401005 50          PUSH   EAX
00401006 0FB646 0D     MOVZX  EAX, BYTE PTR DS:[ESI+D]
00401007 50          PUSH   EAX
00401008 0FB646 0C     MOVZX  EAX, BYTE PTR DS:[ESI+C]
00401009 50          PUSH   EAX
0040100A 0FB646 0B     MOVZX  EAX, BYTE PTR DS:[ESI+B]
0040100B 50          PUSH   EAX
0040100C 0FB646 0A     MOVZX  EAX, BYTE PTR DS:[ESI+A]
0040100D 50          PUSH   EAX
0040100E 0FB646 09     MOVZX  EAX, BYTE PTR DS:[ESI+9]
0040100F 50          PUSH   EAX
00401010 0FB646 08     MOVZX  EAX, BYTE PTR DS:[ESI+8]
00401011 50          PUSH   EAX
00401012 0FB746 06     MOVZX  EAX, WORD PTR DS:[ESI+6]
00401013 50          PUSH   EAX
00401014 0FB746 04     MOVZX  EAX, WORD PTR DS:[ESI+4]
00401015 50          PUSH   EAX
00401016 FF36       PUSH   DWORD PTR DS:[ESI]
00401017 3045 88     LEA   EAX, DWORD PTR SS:[EBP-78]
00401018 68 78413E01  PUSH  @FS:78413E01             ASCII ""08x-004x-004x-002x02x-002x02x02x02x02x
00401019 6A 78     PUSH  78
0040101A 50          PUSH   EAX
0040101B FF15 B8E33D01  CALL  DWORD PTR DS:[13DE3B8]   MSUCRT._sprintf
0040101C 83C4 40     ADD   ESP,40

```

Figure 1: FolderName Generate

```

: 05 06 07 08 09 0A 0B 0C 0D 0E 0F
: C4 19 DC 61 82 0C 68 32 BE 7A 56 :Š*æ.Ä.Üa,.h2%zV
: 1D 5F 92 2B EA 62 53 A4 42 09 DD "Ò}XE._'+ébSxB.Ý
: FE A6 81 E4 92 30 68 9A B9 36 19 8Tü)mp|.ä'Ohs'6.
: 39 FD 11 58 30 85 9F B3 F8 2F 9C 1øÀZ\9ý.XO...Ý²ø/œ
: 7A 27 E3 82 57 ED E0 59 79 EF DA <þ8€.z'ã,WiàYyiÚ
: 7F F7 B7 3B B3 3E EF 73 21 03 B6 i6i'Ì.÷ ;;>is!.¶
: 25 DD C4 59 2A 9F 35 D7 56 1C D5 o±Ýã%ÝÄY*Ý5×V.Ö
: E7 45 C7 2A C8 1F E1 38 50 AA A7 ÷Æ.BQçEQ*È.á8P²§
: 0C 1D 1B 30 C5 B1 E3 8D 4E CB 44 j=ò,, "...OÄ±ã.NED
: 52 CD 9F 3E 02 6F 19 B0 91 9F 89 ÊâKñ.RÍÿ>.o.°`ÿ%
: B9 45 77 11 26 5E 11 E9 2B D9 2A XZ²Äf²Ew.ε^.é+Û*
: 17 BA AB 2F 01 5A DC 26 B7 AB D1 01(AN 0...V7itc...ÿ

```

Figure 2: Encrypted File

1 Introduction

Corebot is a relatively new bot commonly found being delivered by a newer Kovter variant at the time of this writing. The goal of Corebot appears to be in data and credential theft, coming with modules commonly found in most of the bigger banking trojans such as a keylogger module, a stealer(pony) module and a module for performing mitm with the browser in order to facilitate credential theft for targeted institutions found in a zeus-like config.

2 Loader

Most of the relevant data stored on disk was found in APPDATA\local\Microsoft in a series of sub folders. The subfolder names are generated by XORing the computers volume serial number against a few sets of 16 byte hex strings and

```

from Crypto.Cipher import AES
import binascii
import sys
#AES key for main corebot component
#11bff5d5cfad69e638e73f00e35a2b3c77ab5710692c42dfe4e5e7aa6e54a75e
#AES key for corebot modules
#1473c09a3ce86f1f5d2bf8e69e5ecc2598e380d740cc68a7c58019b9c5799388

data = open(sys.argv[1], 'rb').read()

key = binascii.unhexlify('11bff5d5cfad69e638e73f00e35a2b3c77ab5710692c42dfe4e5e7aa6e54a75e')
#Modules
#key = binascii.unhexlify('1473c09a3ce86f1f5d2bf8e69e5ecc2598e380d740cc68a7c58019b9c5799388')

iv = binascii.unhexlify('00'*16)

aes = AES.new(key, AES.MODE_CBC, iv)

decrypted = aes.decrypt(data)

open(sys.argv[1]+'.decr', 'wb').write(decrypted)

```

Figure 3: CoreBot component decryption

then sprintf the hex data into the format string `%08x-%04x-%04x-%02x%02x-%02x%02x%02x%02x%02x%02x` (Figure 1). The loader finds the encrypted corebot module on disk by recreating the pathnames, at this point it decrypts the module using an hardcoded AES key and a zeroed 16 byte IV(Figure 2). A POC for decrypting both the main CoreBot component and its modules can be seen in Figure 3. This main module is spun up into a hollowed out svchost.exe.

3 Main Module

The main module of CoreBot uses a similar routine for path names, registry keys and encryption keys. I've pulled out some of the hardcoded hex data used to generate this data and written a small proof of concept for some of the items in Figure 4, in the script you can see the hardcoded hex data used to produce multiple folder names, file names, the registry persistence key and the RC4 key used to encrypt the CoreBot settings file. The settings file is constantly updated during CoreBots activity, it's not only used by the loader, the main CoreBot module but also used by the downloaded modules themselves.

4 Watchdog

The svchost.exe spins up a dllhost.exe and connects to a named pipe 'core_ps', the process writes it's processid to the named pipe while the watchdog process also communicates with a heartbeat pipe(Figure 5). Should the watchdog not receive a proper response in the proper time frame then it will interrupt the main process tearing it down and then spinning up the loader again. Using WinDBGs child process ability however we can attach in during the loader process, finding relevant code sections that look interesting for exploration in IDA we can set our breakpoints and then pause the watchdog to easily circumvent this system.

5 Modules

The decrypted modules can be seen in Figure 6. All modules communicate with the main CoreBot component over the core_ps pipe, the stealer component works similar to Pony malware except that it packages up all the data and sends it to the main component for delivery to the C2. The keylogger will log keystrokes to a text file which gets GZIP compressed and also sent back for delivery to the C2. The mitm module is a bit more complex in that it has references in the settings to it's own config and an interval on when it should be updated.

6 Settings

The settings file stored on disk is in the core.work_dir which is the same directory where the encrypted main module is kept. To generate the RC4 key needed to decrypt this file you take the hardcoded hex data stream and the volume serial number(VSN). The VSN will be XORd with the hardcoded dword 0x42ab3122 and then this value will be XORd against every dword in the hardcoded hex stream while rolling the key to the right 2 bytes every iteration. A proof of concept script for decrypting a settings file can be found in Figure 7. In this decrypted settings file you can find the RC4 key used to decrypt C2 traffic which is called core.server_key.

7 C2 Traffic

C2 traffic is encrypted with RC4 using the core.server_key from the settings file(Figure 8). Aside from checkins and messages there are a few big areas that most people will be interested in when it comes to banking trojans, the modules and the config(Figure 9). In both cases the data is RC4 encrypted on top of custom structured data, in the case of the modules there is extra header data on top describing the module(Figure 10). In both cases however it boils down to the same structure of a single byte, dword and blob of zlib compressed data(Figure 11).

Going through getting the config from the C2 traffic, we start with the encrypted

data(Figure 12) from there we simply RC4 decrypt the data and then pull out the length of the compressed blob that follows as seen in Figure 13. A quick decompress and we can see the beginnings of an unparsed config(Figure 14).

8 Conclusions

Sample SHA256: 781c6743230918c591b20121ab34ff639968ac14c958f6207effbba465b71ee0

8.1 Signatures

ET: ET TROJAN Corebot Checkin

Yara:

```
rule CoreBot_Scan_Mem {
strings:
    $s0 = "core.dga"
    $s1 = "core.server_key"
    $s2 = "core_ps"
    $m0 = "keylogger.dll"
    $m1 = "stealer.dll"
    $m2 = "m3.dll"
    $m3 = "mitm"
condition:
    any of ($s*) and any of ($m*)
}
```

References

- [1] Hex-Rays Decompiler, <http://www.hex-rays.com/products/decompiler/index.shtml>.
- [2] Python, <https://www.python.org/>
- [3] Python and Bitwise Rotation, <http://www.falatic.com/index.php/108/python-and-bitwise-rotation>
- [4] Emerging Threats, <http://www.emergingthreats.net/>

```

import ctypes
import binascii
import struct

filekey1 = binascii.unhexlify('74c16ccc7a2928459911b21a2fd4d8be')
regkey1 = binascii.unhexlify('3aa7af21467cf0429694560492b893a0')
folderkey1 = binascii.unhexlify('40cd38f54df3ec43854bccb8b9931c48')
folderkey2 = binascii.unhexlify('b94afd7ee333414399d916b60956beaa')
rc4key = binascii.unhexlify('64306743d52625f694d4e2d3654d63d81beac3a4a4bd4675')

def gen_name(vsn, fkey):
    (f1, f2, f3, f4) = struct.unpack('<IIII', fkey)
    new = struct.pack('<IIII', f1^vsn, f2^rol(vsn, 2), f3^rol(vsn,4), f4^rol(vsn,8))
    print "%08x-%04x-%04x-%02x%02x-%02x%02x%02X%02X%02x%02x" % struct.unpack('<IIII', new)

def gen_key(vsn, hkey):
    temp = vsn ^ int('42ab3122',16)
    (f1, f2, f3, f4, f5, f6) = struct.unpack('<IIIII', hkey)
    new = struct.pack('<IIIII', f1^temp, f2^ror(temp, 2), f3^ror(temp,4), f4^ror(temp,8), f5^ror(temp,12), f6^ror(temp,16))
    print(binascii.hexlify(new))

rol = lambda val, r_bits, max_bits=32: \
    (val << r_bits%max_bits) & (2 ** max_bits-1) | \
    ((val & (2**max_bits-1)) >> (max_bits-(r_bits%max_bits)))

ror = lambda val, r_bits, max_bits=32: \
    ((val & (2**max_bits-1)) >> r_bits%max_bits) | \
    (val << (max_bits-(r_bits%max_bits)) & (2**max_bits-1))

kernel32 = ctypes.windll.kernel32

volumesn = ctypes.c_ulong()
kernel32.GetVolumeInformationW(ctypes.c_wchar_p("C:\\"), None, 0, ctypes.byref(volumesn), None, None, None, None)

vsn = volumesn.value

gen_name(vsn, folderkey1)
gen_name(vsn, filekey1)
gen_key(vsn, rc4key)
gen_name(vsn, folderkey2)
gen_name(vsn, regkey1)

```

Figure 4: CoreBot Folder/FileName and Key Generation POC

File	\Device\NamedPipe\core_ps	0x1a4
File	\Device\NamedPipe\core_ps	0x1b4

Figure 5: Main named pipe

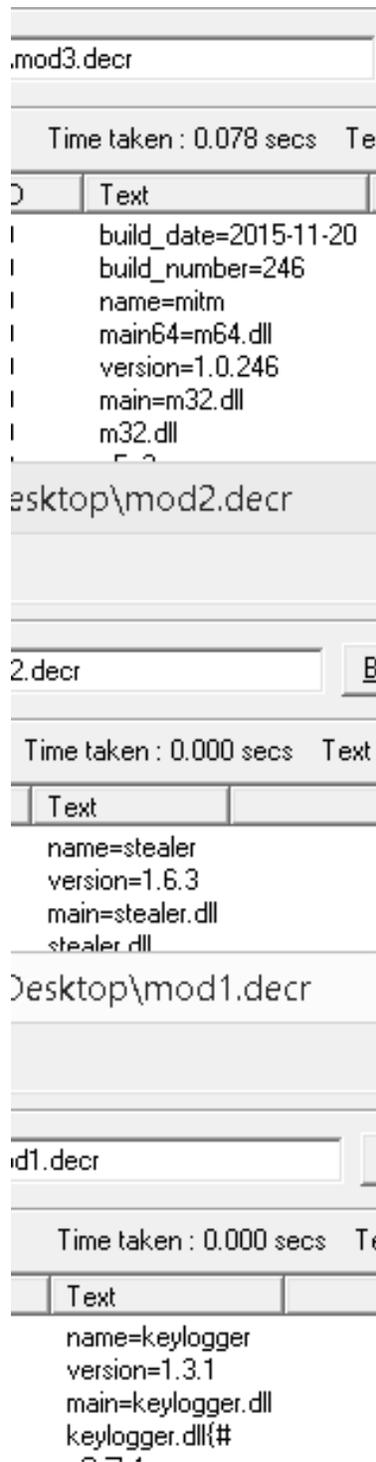


Figure 6: CoreBot Modules

```

import ctypes
from Crypto.Cipher import ARC4
import binascii
import struct
import sys

rc4key = binascii.unhexlify('64306743d52625f694d4e2d3654d63d81beac3a4a4bd4675')

def gen_key(vsn, hkey):
    temp = vsn ^ int('42ab3122',16)
    (f1, f2, f3, f4, f5, f6) = struct.unpack('<IIIII', hkey)
    new = struct.pack('<IIIII', f1^temp, f2^ror(temp, 2), f3^ror(temp,4), f4^ror(temp,6), f5^ror(temp,8), f6^ror(temp,10))
    return(new)

ror = lambda val, r_bits, max_bits=32: \
((val & (2**max_bits-1)) >> r_bits%max_bits) | \
(val << (max_bits-(r_bits%max_bits)) & (2**max_bits-1))

kernel32 = ctypes.windll.kernel32

volumesn = ctypes.c_ulong()
kernel32.GetVolumeInformationW(ctypes.c_wchar_p("C:\\"), None, 0, ctypes.byref(volumesn))

vsn = volumesn.value

key = gen_key(vsn, rc4key)

rc4 = ARC4.new(key)
data = open(sys.argv[1], 'rb').read()
open(sys.argv[1]+'_decr', 'wb').write(rc4.decrypt(data))

```

Figure 7: CoreBot Settings File Decryption POC

```

mov     [esp+11Ch+var_F4], edx
mov     ecx, offset aCore_server_ke ; "core.server_key"
mov     [esp+11Ch+var_108], bl
call    FindValInSettings_418CFC
test    al, al
jz      loc_4149F9

mov     edx, [esp+11Ch+var_F8]
lea     eax, [esi+4]
mov     ecx, [esp+11Ch+var_F4]
sub     edx, ecx
push    edi
push    eax
call    RC4_40E6D9
lea     eax, [esp+124h+var_EC]
mov     [esp+124h+var_EC], bl
push    eax

```

Figure 8: CoreBot C2 Data Process

haloadoxy.com /gate/	61,359	r
haloadoxy.com /gate/	0	
haloadoxy.com /gate/	0	
haloadoxy.com /gate/	166	r
haloadoxy.com /gate/	5	r
haloadoxy.com /gate/	9,184	r
haloadoxy.com /gate/	0	
haloadoxy.com /gate/	0	
haloadoxy.com /gate/	110	r
haloadoxy.com /gate/	5	r
haloadoxy.com /gate/	5	r
haloadoxy.com /gate/	5	r
haloadoxy.com /gate/	564,533	r
haloadoxy.com /gate/	5	r
haloadoxy.com /gate/	5	r
haloadoxy.com /gate/	14	r
haloadoxy.com /gate/	5	r
haloadoxy.com /gate/	377,322	r

Figure 9: CoreBot Traffic

```

027c8020 01 5c 00 00 00 62 75 69 6c 64 5f 64 61 74 65 3d 32 30 31  \...\build_date=201
027c8033 35 2d 31 31 2d 32 30 0a 62 75 69 6c 64 5f 6e 75 6d 62 65  5-11-20.build_numbe
027c8046 72 3d 32 34 36 0a 6e 61 6d 65 3d 6d 69 74 6d 0a 6d 61 69  r=246.name=mitm.mai
027c8059 6e 36 34 3d 6d 36 34 2e 64 6c 6c 0a 76 65 72 73 69 6f 6e  n64=m64.dll.version
027c806c 3d 31 2e 30 2e 32 34 36 0a 6d 61 69 6e 3d 6d 33 32 2e 64  =1.0.246.main=m32.d
027c807f 6c 6c 03 07 00 00 00 6d 33 32 2e 64 6c 6c 01 0c 03 00 78  11.....m32.dll....x
027c8092 9c ec bd 0d 78 1b d5 95 30 3c 92 c6 f6 d8 56 a2 49 22 83  ....x...0<...V.I".
027c80a5 00 93 18 50 82 17 a5 60 50 80 18 25 20 ff c8 36 21 32 52  ...P...P...%...612R
027c80b8 8c e5 b8 21 4e ba 4d 5c 55 d0 36 38 33 76 b2 8d 83 c3 d8  ...IN.MNU.683v....
027c80cb e0 f1 45 bb 74 4b bb b4 e5 ed c2 47 f7 5d b6 65 77 b3 0b  .EtK...G.]ew...
027c80de 4d 42 49 41 b2 53 ff 40 4a e2 84 06 87 78 a9 a1 86 8e 91  MBIA.S@J...x....
027c80f1 db 3a e0 c6 0a 51 3d df 39 77 46 b2 9c 3f da b4 ef f3 7e  ....Q=.9wF...?....~
027c8104 cf f7 34 4f ac b9 73 7f ce 3d f7 dc f3 77 7f c7 fb f9 27  ...40...s...w....
027c8117 18 13 c3 30 2c fc a9 2a c3 ec 63 b4 7f 6e e6 8f f8 67 60  ...0...*...c...n...g'
027c812a 98 b9 8b 7e 32 97 79 29 fb e7 d7 ec 33 ac fe f9 35 f7 05  ...~2.y)....3...5..
027c813d bf bc b5 60 4b d3 d7 be d4 f4 85 af 14 7c f1 0b 5f fd ea  ...K.....-.....
027c8150 d7 84 82 bf dd 5c d0 24 7e b5 e0 cb 5f 2d 28 bf b7 a6 e0  ....\s~...|.....
027c8163 2b 5f db b4 f9 c6 39 73 72 ec 3a 88 1b 2f af 13 43 cd c5  +.....9sr.../...C.
027c8176 4b 92 7f 5f b2 0f 2d c9 80 e7 f3 df 9d 5a f2 20 3c 33 98  K.....-.....Z...<3.
027c8189 7f 59 12 84 e7 17 e7 be b0 e4 0b f4 f9 d3 25 7f 47 9f 3f  .Y.....%...G.?
027c819c 59 f2 1f 7d fe 70 c9 43 b4 ec f2 25 77 b4 14 2f f1 ee ae  Yr.}p.C...%w.../...
027c81af 5c f2 00 85 33 bc e4 46 fa fc 40 7f ff 35 7d ae f9 f2 17  \...3...F...@...5)....

```

Figure 10: CoreBot Decrypted Module

0131c10500789ce4bd87

Figure 11: CoreBot Data Header

```

s>>> conf[:100]
"\xd9\xfa \x8e\xb9\x02\xa6\x87\x90r:\x9br\x87\x9b0,\xc8q~\xec7\xe5h\x1bm+\|\n6y\
x9F\x17\xd6\x97 \x85j\x0e\x94:\xb9\x95\xea6>j\x9a\xad\x8a@u\xd6nt\x1a\xa9\xf8\xce
a\x8f\x8f\xfd1j\x98\x99\xdf~\xaaC\xefF\xF7\xcf\xe7i\xd4;* \xb2{F\x8aa\xb0\xa6\xfc
j\xa3q@\xadj\x9c0\x84\x93=0\xbF"

```

Figure 12: CoreBot Encrypted Config

```

>>> key
'e3f33a48fad320f43ca6130294cfb191'
>>> rc4 = ARC4.new(key)
>>> c = rc4.decrypt(conf)
>>> c[:100]
'\x011\xc1\x05\x00\x09c\xe4\xbd\x87r\xdb\xc8\xba0\x\xff\xad\xda\xaa\xf5\xd4V\xed+
p4u\xc6\xd2P\x122H\xd8#01\x13\xcc9\xcd\xf8\xaa\x90\t\x12\x89\x08\x0c\x18\xf9>\xc
3>\xc0>\xd6>\xd0v\x03 \x08\x06\xc9\xb4\xcf\x9cs\xcf\xb9\x0b[\x12\xd0\xf8\xba\xfb
\xcb\xdf\xd7\x01@xe1? \xfe\xe3?\xba\xff\x11\x1c\xff+\xfc\xf3\x7f\x81\x9f'
>>> struct.unpack_from('<BI', c)
(1, 377137)
>>> len(c[5:])
377137
>>>

```

Figure 13: CoreBot Decrypted Config Data Structure

```

>>> zo = zlib.decompressobj()
>>> zo.decompress(c[5:])[200]
'\c\x00\x00\x00q\x00\x00\x00\x00\x00\x01\x00\x00\x00\x00\x00\x0f\x00\x00\
\x00*/tkn_api_v1/*\x00\x01\x00\x00\x00.\x00\x00\x00\x00\x0c&\x00\x00\x00http://18
5.14.29.123:18000/tkn/api.phpT\x00\x00\x00\x00\x00\x00\x00\x01\x00\x00\x00\x0
00\x0f\x00\x00\x00*/tkn_assets/*\x00\x01\x00\x00\x00/\x00\x00\x00\x00\x0c)\x00\x
00\x00http://185.14.29.123:18000/tkn/assets.php\x9c\n\x00\x00\x00\x00\x01\x0
0\x00\x00\x00\x00w\n\x00\x00*/onl'
>>>

```

Figure 14: CoreBot Decompressed Config