Anatomy of a Massive P2P Botnet Takedown: Reversing and Attacking GameOver Zeus

Dennis Andriesse

Vrije Universiteit Amsterdam

Finse Winter School 2018

Acknowledgements

Brett Stone-Gross (Dell SecureWorks)

Tillmann Werner, Christian Dietrich (CrowdStrike)

Christian Rossow (Saarland University)

Frank Ruiz, Michael Sandee (Fox-IT)

Elliott Peterson (FBI)

The ShadowServer Foundation

CERT.PL

Too many others to name here...

Introduction to Botnets

What is a botnet?

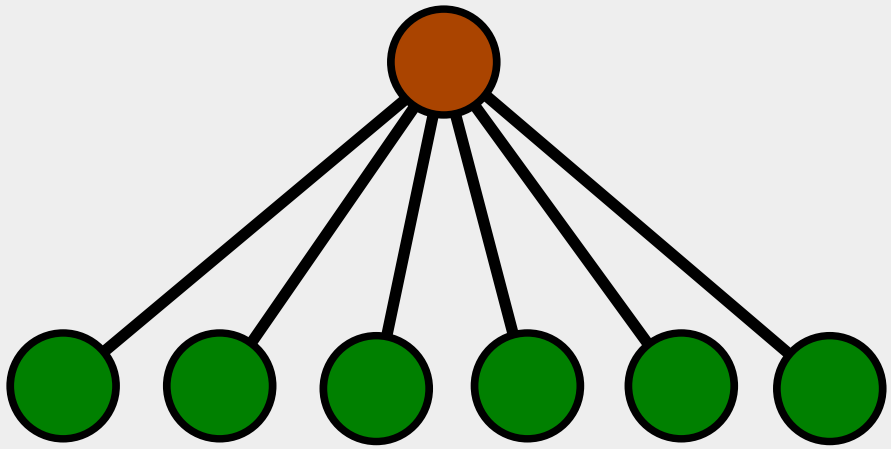
- Network of malware–infected computers (*bots*)
- Controlled by *botmaster* to perform malicious actions
- Typically contains 100.000 - 1.000.000 bots

Evolution of Botnets

Evolution of Botnets

Centralized botnets

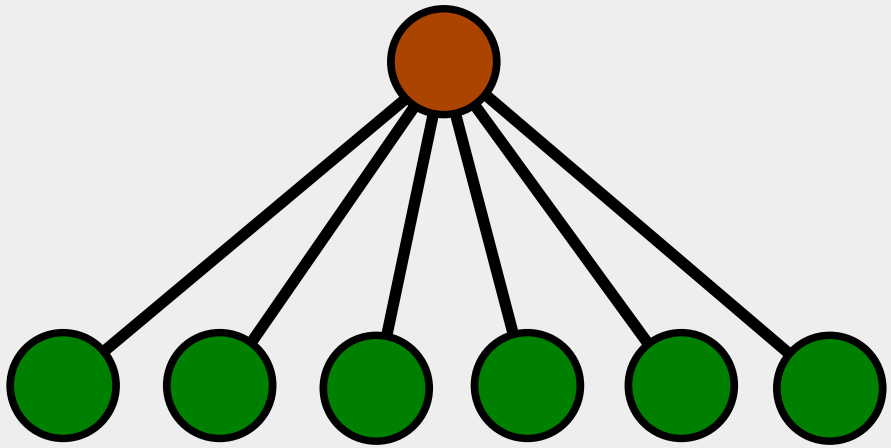
- Original botnets were centralized
- *Command and Control* (*C2*) server spreads commands to bots
- First botnets based on IRC (a chat protocol)
  - Bots enter the "chat room" and listen to commands
- Later botnets used HTTP
  - Bots fetch commands from a "web server"



Evolution of Botnets

Centralized botnets

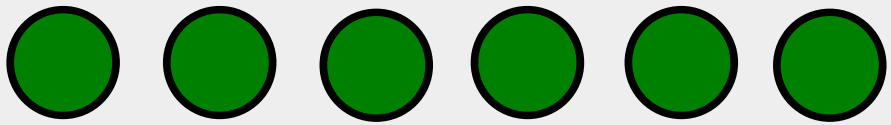
- Simple, easy to maintain for the bad guys
- Easy to disable for the good guys
  - Just take out the C2 server



Evolution of Botnets

Centralized botnets

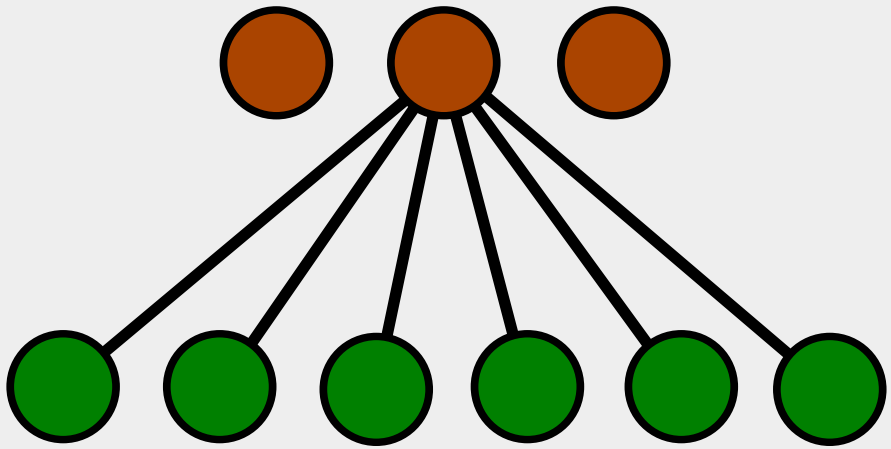
- Simple, easy to maintain for the bad guys
- Easy to disable for the good guys
  - Just take out the C2 server



Evolution of Botnets

Redundant infrastructure

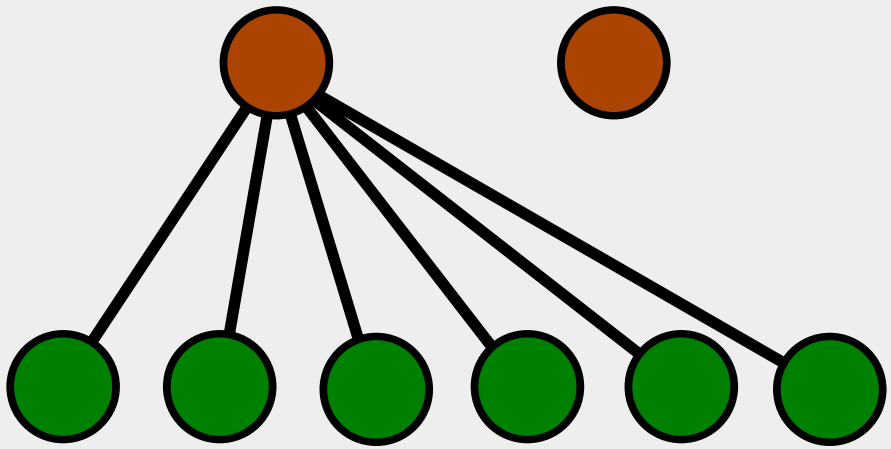
- Early way to strengthen centralized botnets: multiple C2 servers
- If one of the servers is disabled, bots just switch to another



Evolution of Botnets

Redundant infrastructure

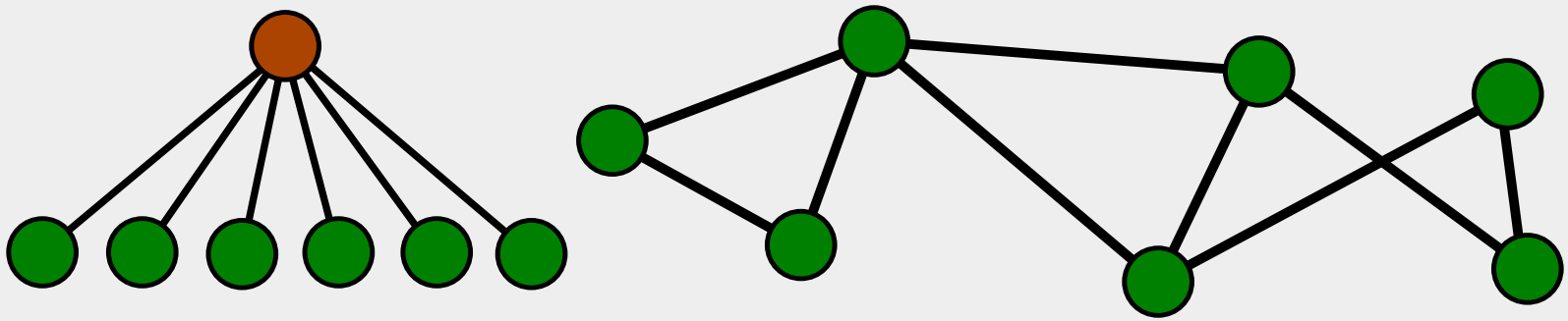
- Early way to strengthen centralized botnets: multiple C2 servers
- If one of the servers is disabled, bots just switch to another



Evolution of Botnets

Peer-to-Peer (P2P) botnets

- Centralized botnets are vulnerable because of their C2 servers
- P2P botnets have no centralized C2 servers
  - Every bot knows some of the other bots
  - Bots use P2P communication to spread commands
  - Much more resilient against takedowns



Functionality of P2P Botnets

P2P Botnets: Terminology

Topologies

- Structured: Peers have addresses, information is routed
- Unstructured: Protocol based on gossiping

Bootstrapping

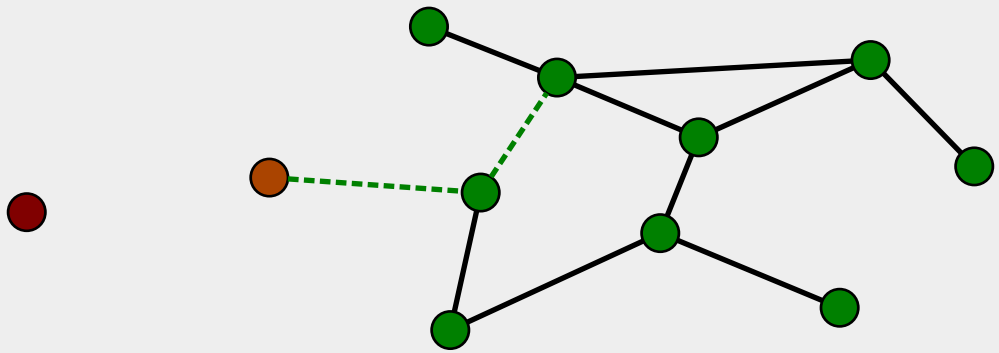
- Process of establishing connectivity with a P2P network
- Finding initial peers
  - Seeding via separate channel
  - Pre-shared peer lists
  - Scanning

Maintenance

- Bots regularly update their *peer list* to account for churn
- Typically some backup channel in case this fails

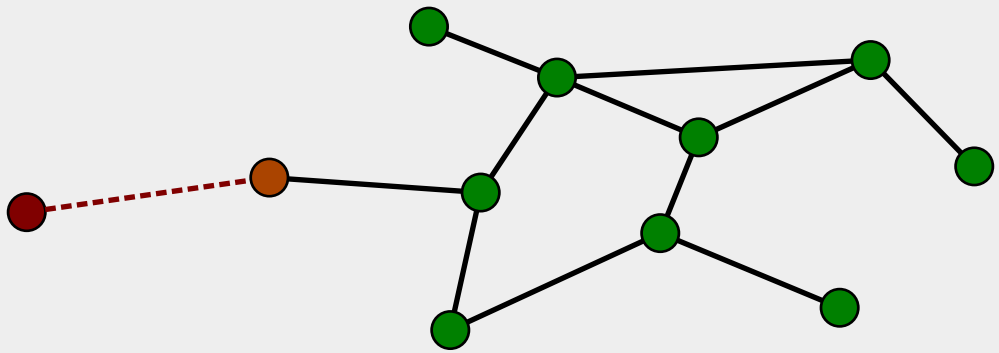
Example: Storm Worm

- Hybrid architecture
- Structured P2P network, nodes have *addresses*
  - Peer-to-Peer network used for C2 server lookups
  - Peers are constantly searching for Time-Dependent Hashes
  - Responses encode C2 IP Address and TCP Port
  - Peers poll announced C2 host for commands



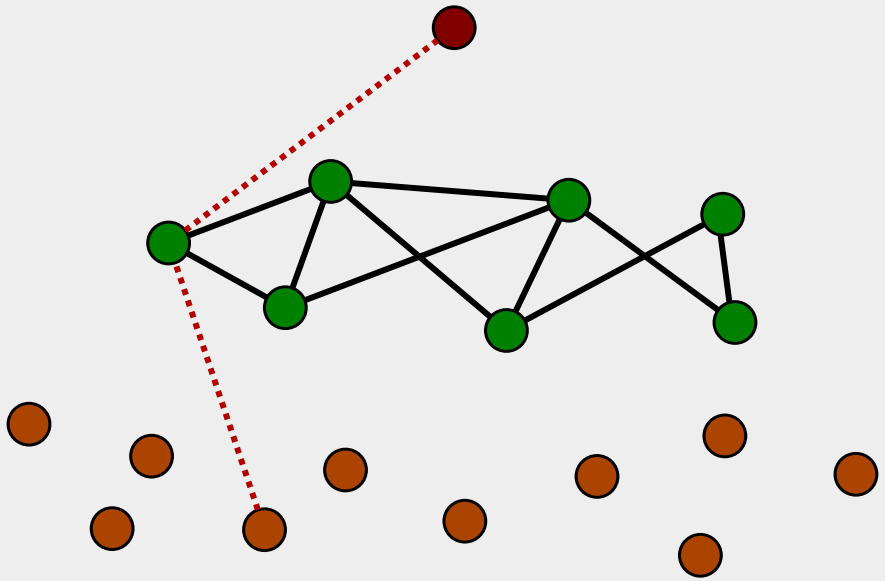
Example: Storm Worm

- Hybrid architecture
- Structured P2P network, nodes have *addresses*
  - Peer-to-Peer network used for C2 server lookups
  - Peers are constantly searching for Time-Dependent Hashes
  - Responses encode C2 IP Address and TCP Port
  - Peers poll announced C2 host for commands



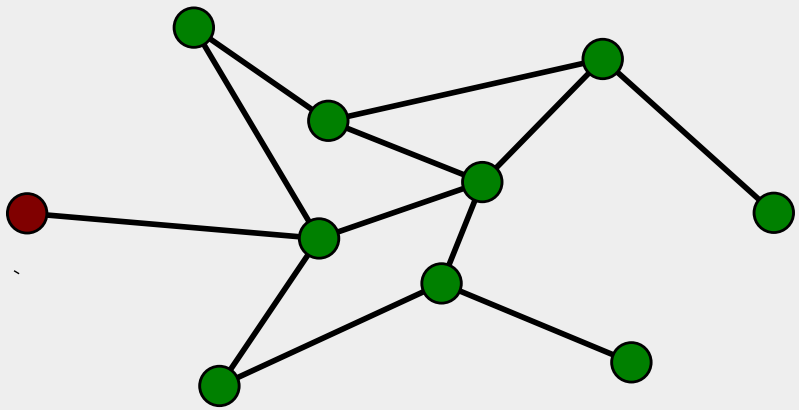
Example: Kelihos

- Strategy: disposable botnets
- P2P layer is part of a multi-tier topology
  - C2 Proxies are announced to all peers
  - Dynamic, self-organizing backbone
  - Router nodes act as intermediate proxies



Example: Sality

- Pure P2P, protocol based on gossiping
- Peers attempt to pull URLs from their neighboring nodes
- Reputation scheme
  - Valid response from p: $Reputation_p := Reputation_p + 1$
  - Invalid response from p: $Reputation_p := Reputation_p - 1$



How to Deal with "Disposable Botnets"?

- When taken down, botnets like Kelihos quickly respawn
- To prevent this, we must take out the droppers (Sality, Zeus, ...)
- Not so easy, especially Sality and Zeus are quite resilient

Attacking P2P Botnets

Attacking P2P Botnets

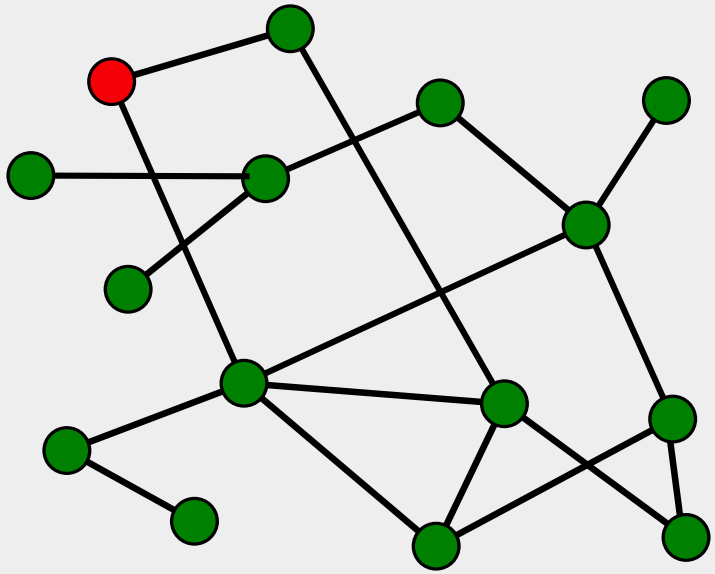
Commanding bots to uninstall

- Usually not possible because of command signing
- Bredolab did not use command encryption
- Team High Tech Crime performed a complete takeover in 2010
- They were rewarded with a Big Brother Award

Attacking P2P Botnets

Reconnaissance

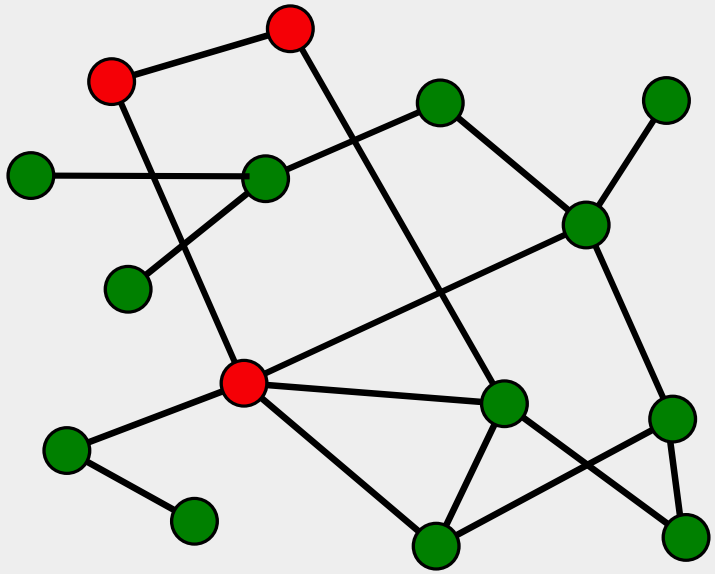
- Reconnaissance attacks try to find all the bots
  - Know how big the botnet is
  - Report bot addresses to Internet providers
- Abuse botnet's maintenance mechanism:
  1. Start with a few known bot addresses
  2. Ask these bots which other bots they know
  3. Repeat for newly found bots



Attacking P2P Botnets

Reconnaissance

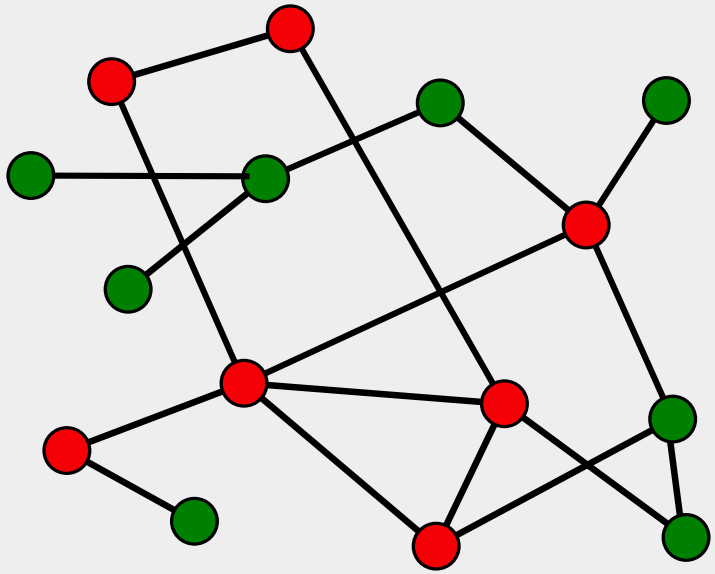
- Reconnaissance attacks try to find all the bots
  - Know how big the botnet is
  - Report bot addresses to Internet providers
- Abuse botnet's maintenance mechanism:
  1. Start with a few known bot addresses
  2. Ask these bots which other bots they know
  3. Repeat for newly found bots



Attacking P2P Botnets

Reconnaissance

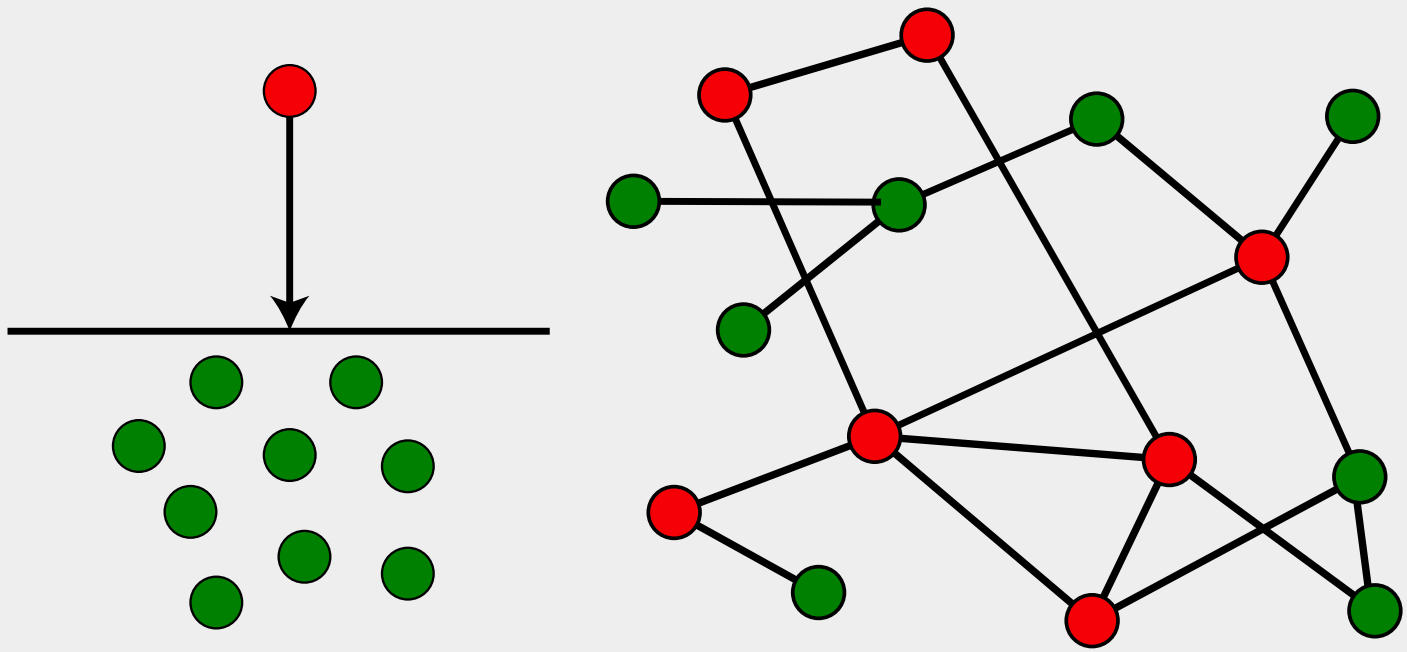
- Reconnaissance attacks try to find all the bots
  - Know how big the botnet is
  - Report bot addresses to Internet providers
- Abuse botnet's maintenance mechanism:
  1. Start with a few known bot addresses
  2. Ask these bots which other bots they know
  3. Repeat for newly found bots



Attacking P2P Botnets

Reconnaissance

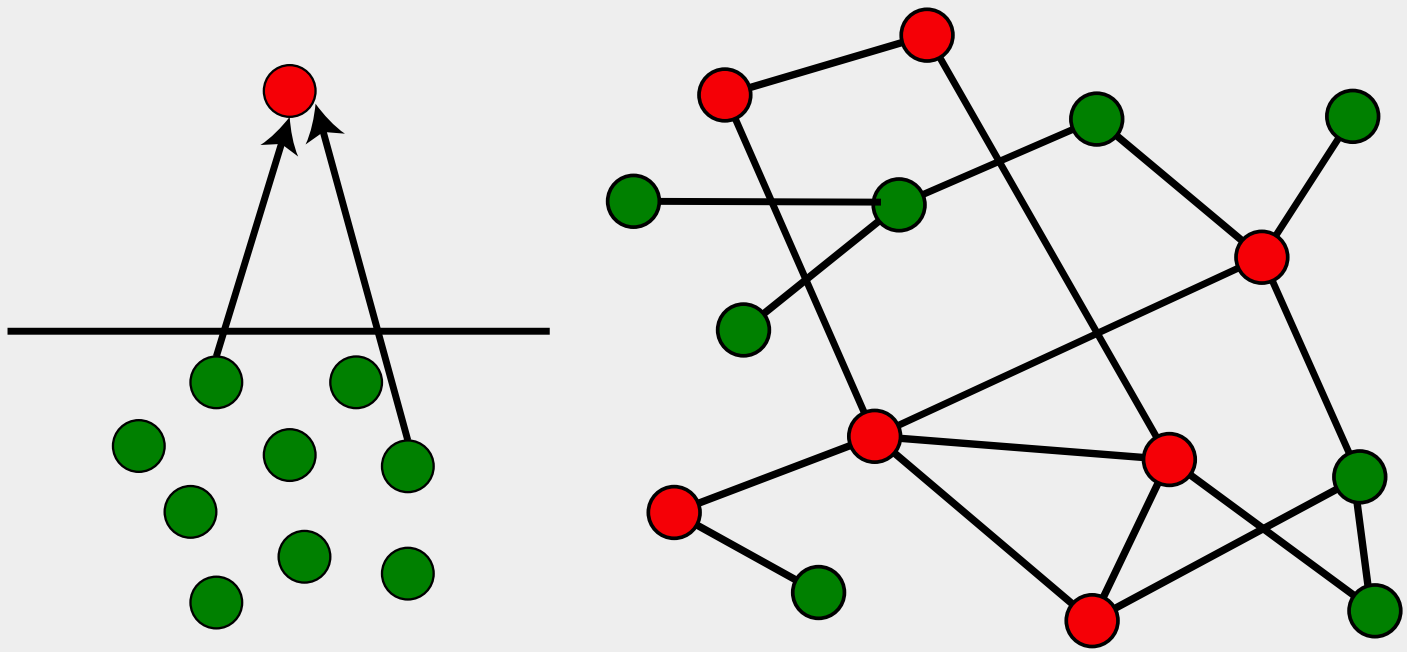
- Cannot find NATed nodes this way
- 60% – 87% of nodes is NATed!
- Infiltrate the botnet and get them to connect to *you*



Attacking P2P Botnets

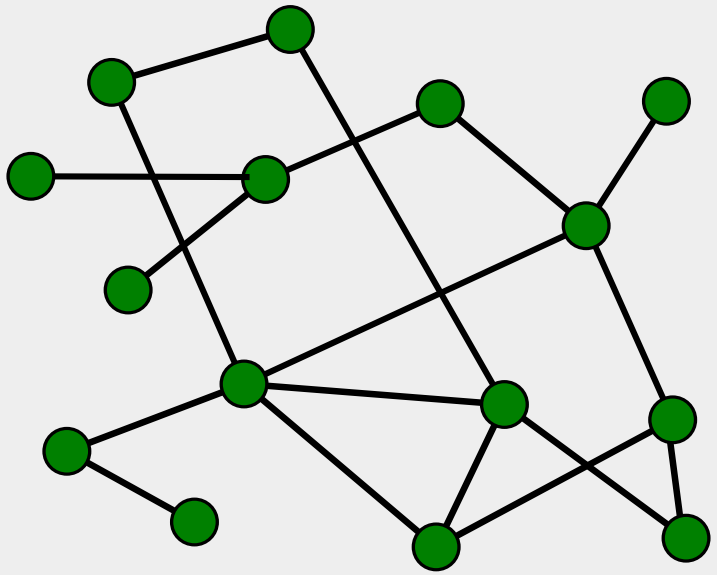
Reconnaissance

- Cannot find NATed nodes this way
- 60% – 87% of nodes is NATed!
- Infiltrate the botnet and get them to connect to *you*



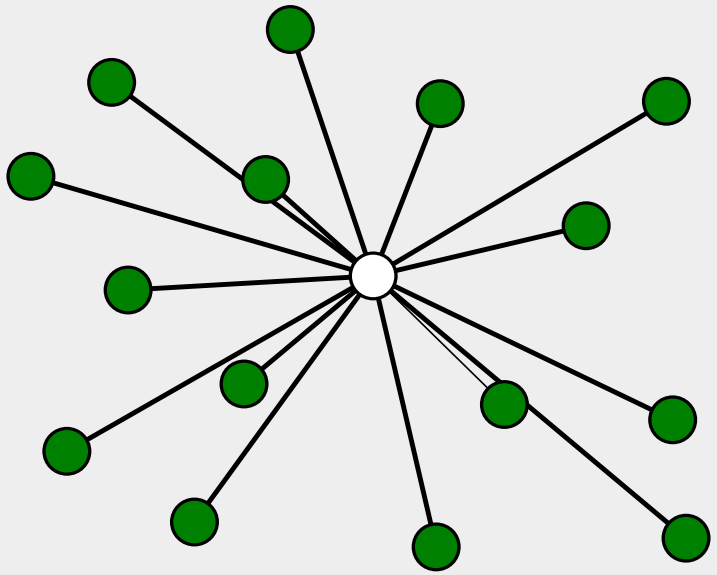
Sinkholing

- Sinkholing attacks try to disconnect bots from each other
- Requires a way to modify bots' *peer lists*
- Try to redirect all bots to a benign *sinkhole* server



Sinkholing

- Sinkholing attacks try to disconnect bots from each other
- Requires a way to modify bots' *peer lists*
- Try to redirect all bots to a benign *sinkhole* server



P2P Zeus

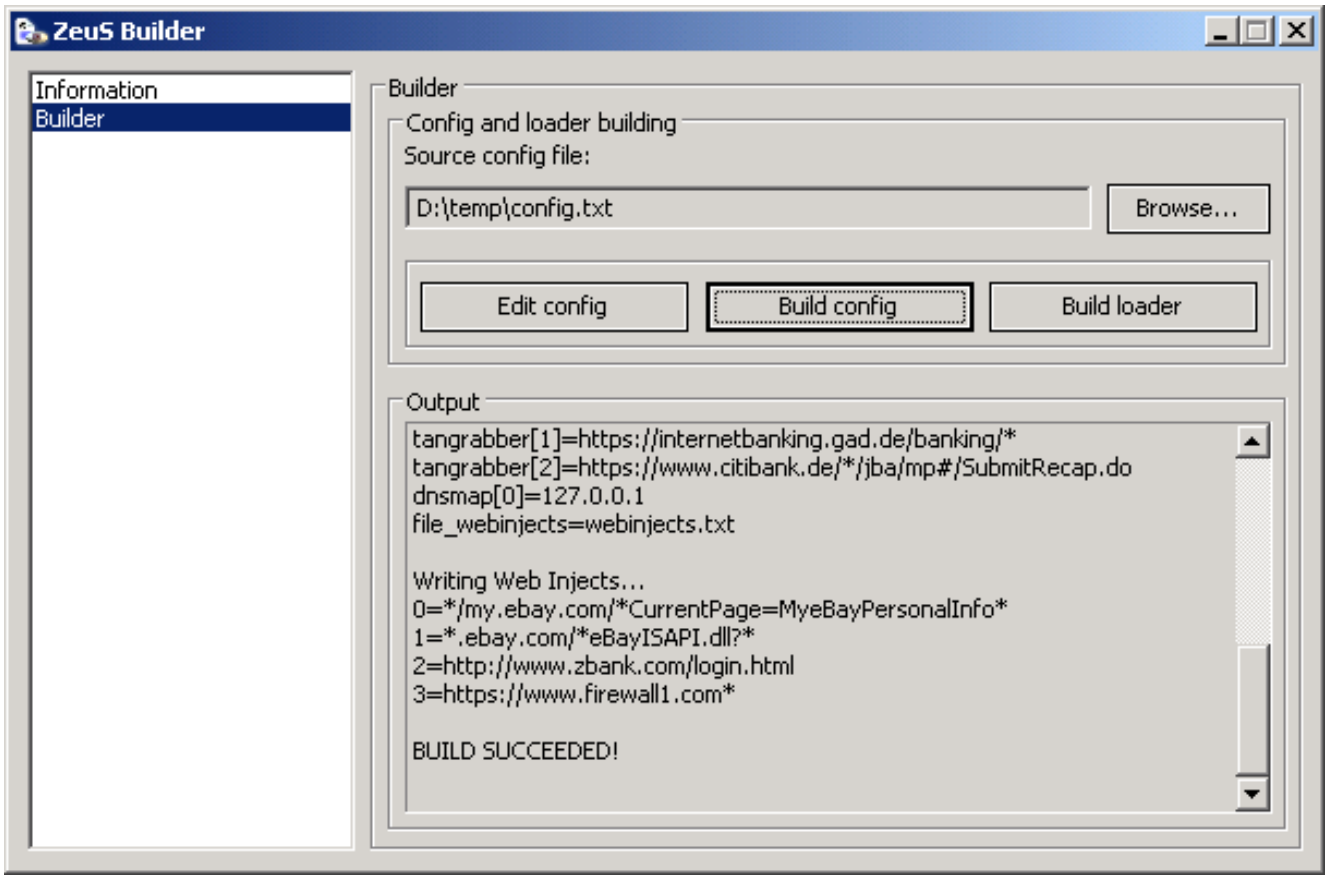
Introduction to P2P Zeus

The Zeus Bot

- Banking trojan, information stealer
- Centralized version around since 2005
- Sold as DIY toolkit for $4000
- FBI tracked a group in 2010 which stole over $70m with it

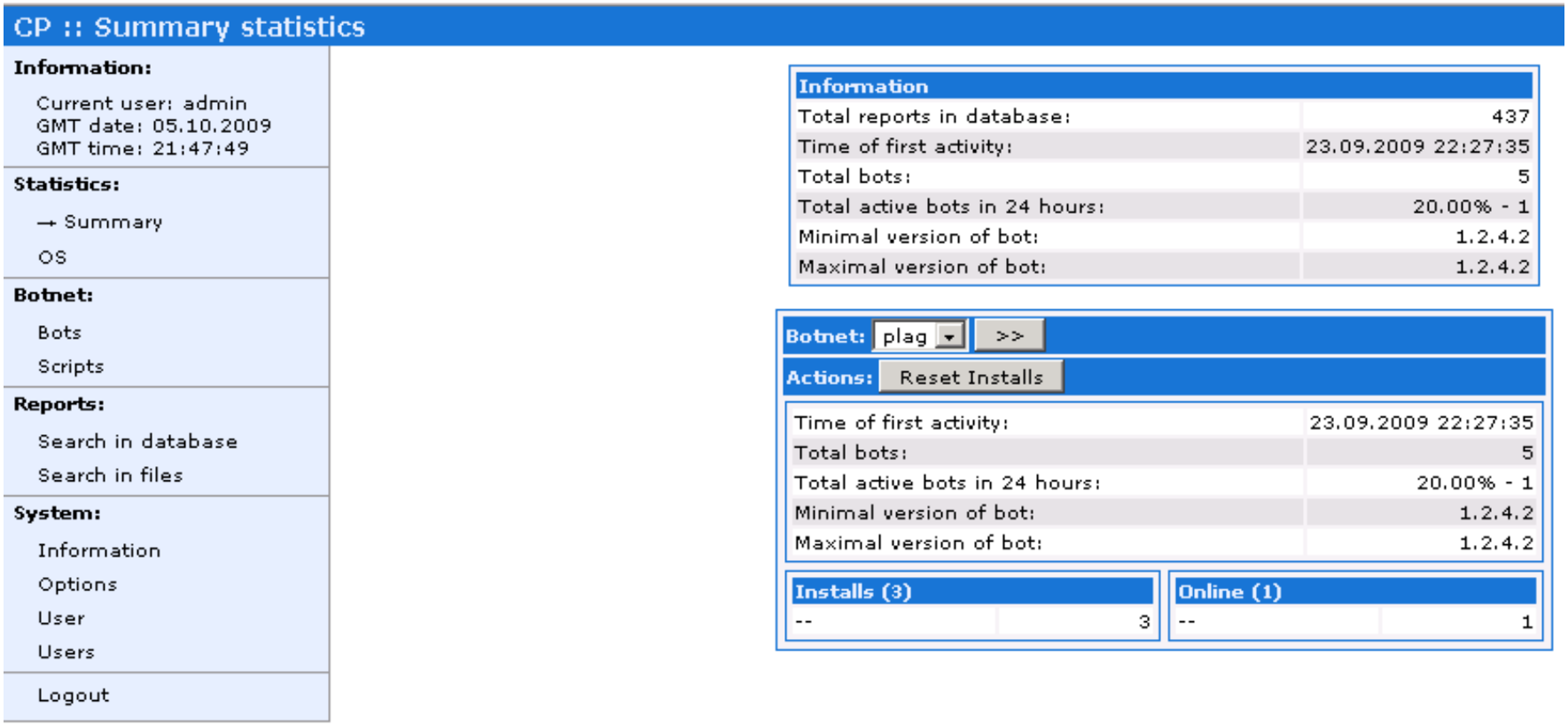
Zeus v1/v2

- Configuring your own Zeus is as easy as running a wizard program
- Zeus toolkits even include anti–piracy mechanisms



Zeus v1/v2

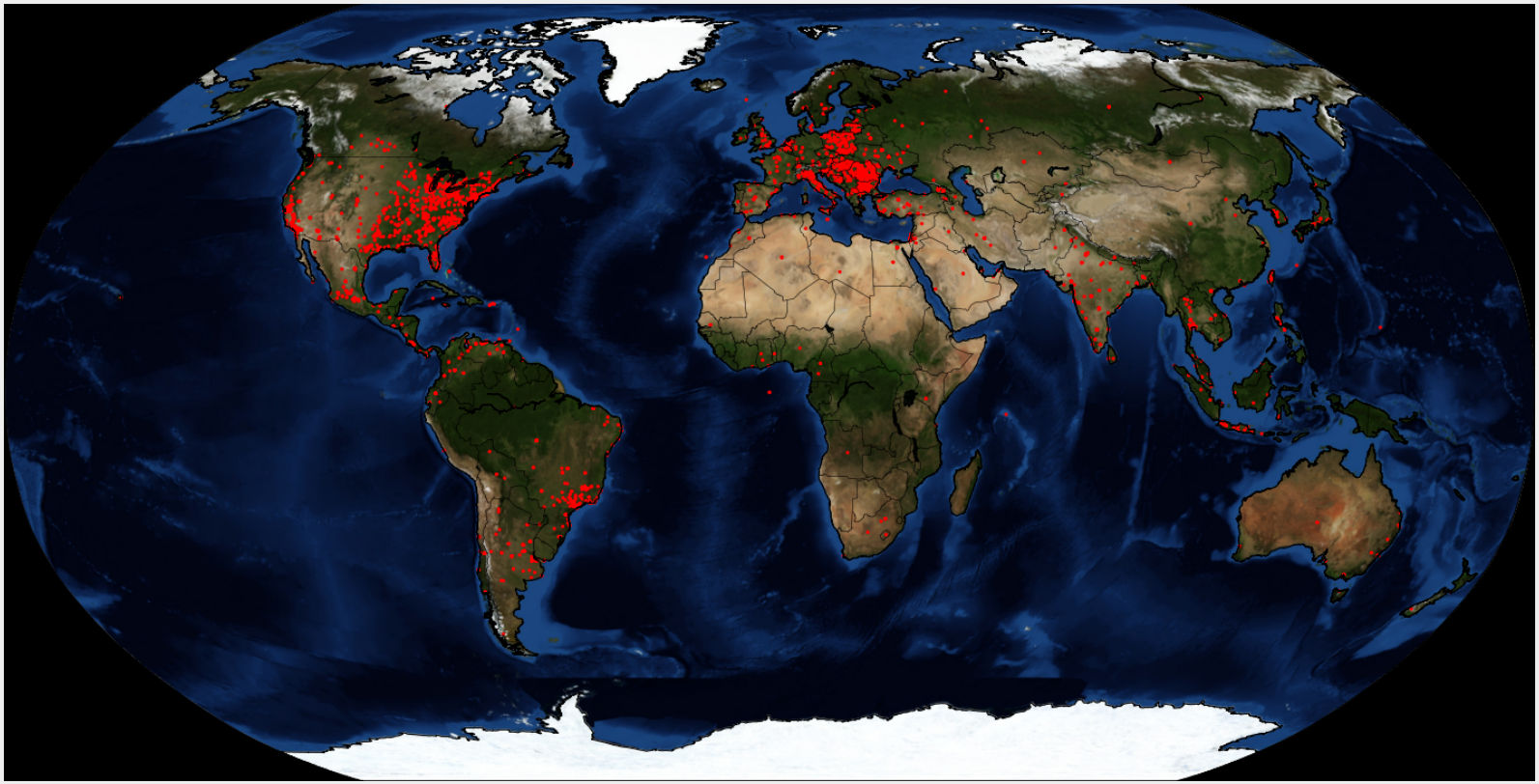
- Your Zeus can be controlled using a handy web interface



Introduction to P2P Zeus

P2P Zeus/Gameover

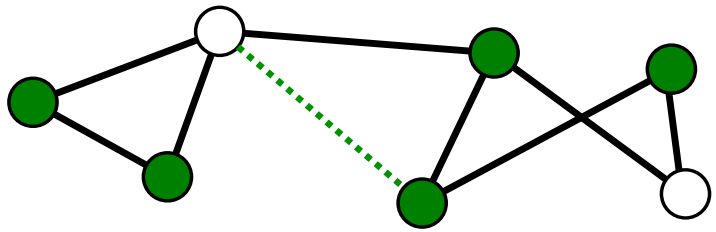
- Zeus evolved into a P2P variant around October 2011
- The P2P network contained 200.000 bots
- Taken down in June 2014 (Operation Tovar)! Stay tuned



Botnet Topology

P2P Layer

- Daily configuration updates
- Weekly binary updates



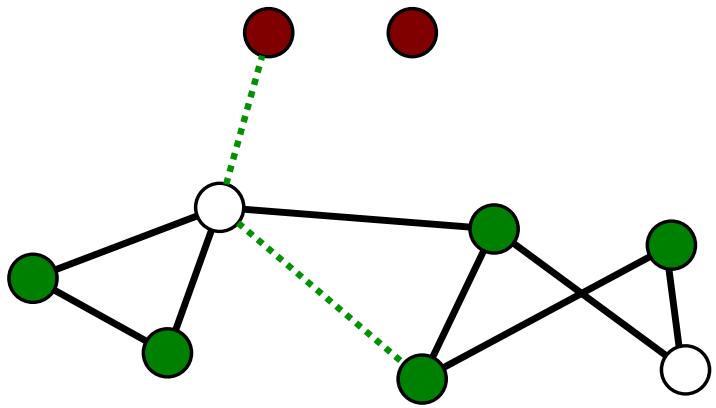
Botnet Topology

P2P Layer

- Daily configuration updates
- Weekly binary updates

Proxy Nodes

- Announced by special messages
- Route C2 communication
  - Stolen data
  - Commands



Botnet Topology

P2P Layer

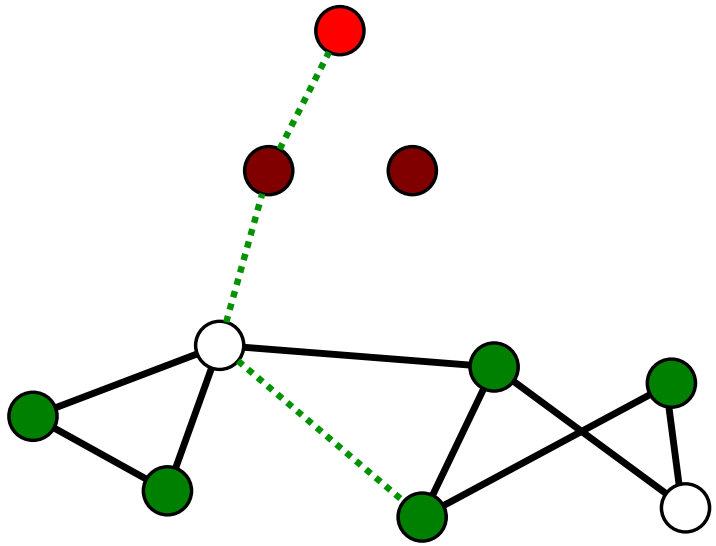
- Daily configuration updates
- Weekly binary updates

Proxy Nodes

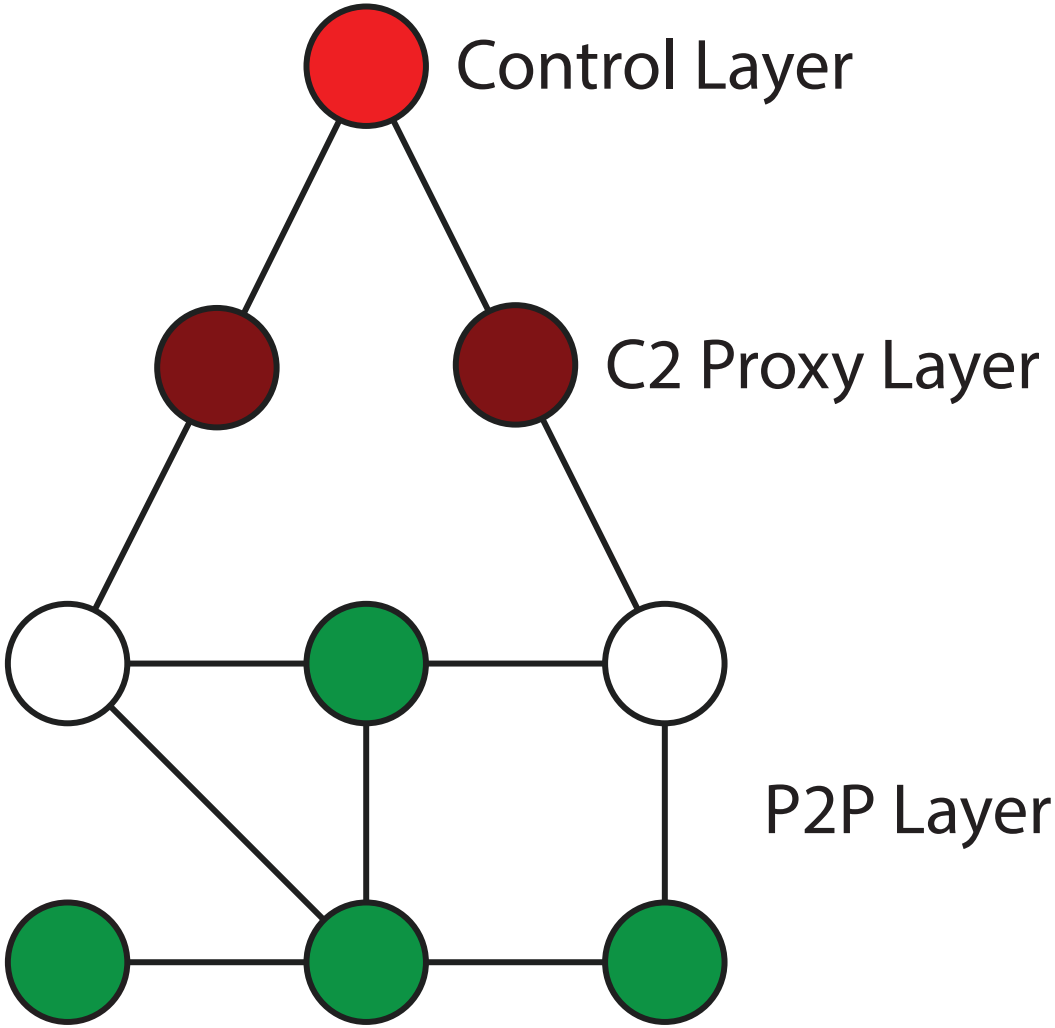
- Announced by special messages
- Route C2 communication
  - Stolen data
  - Commands

C2 Proxies

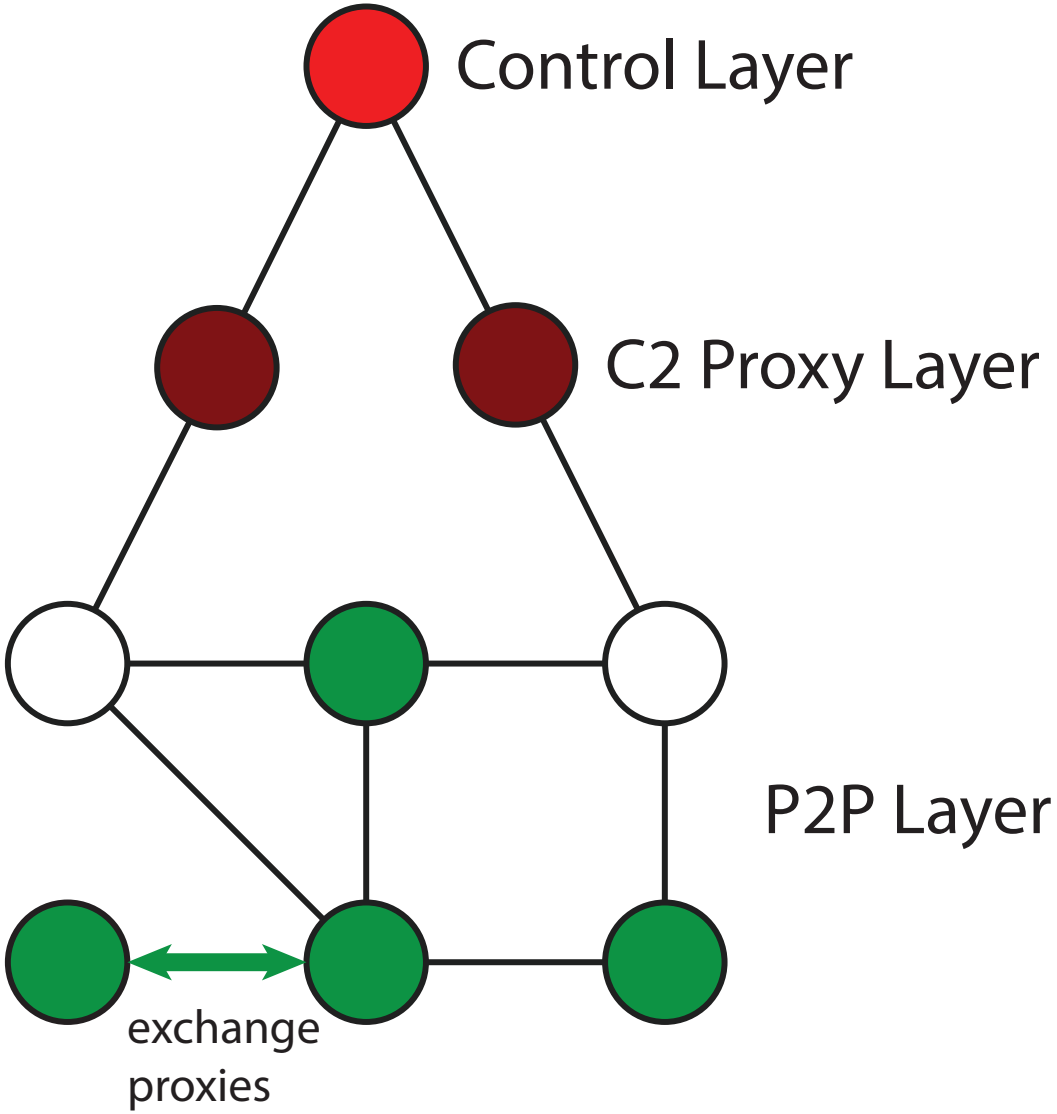
- Plain HTTP proxies
- Additional layer between botnet and backend



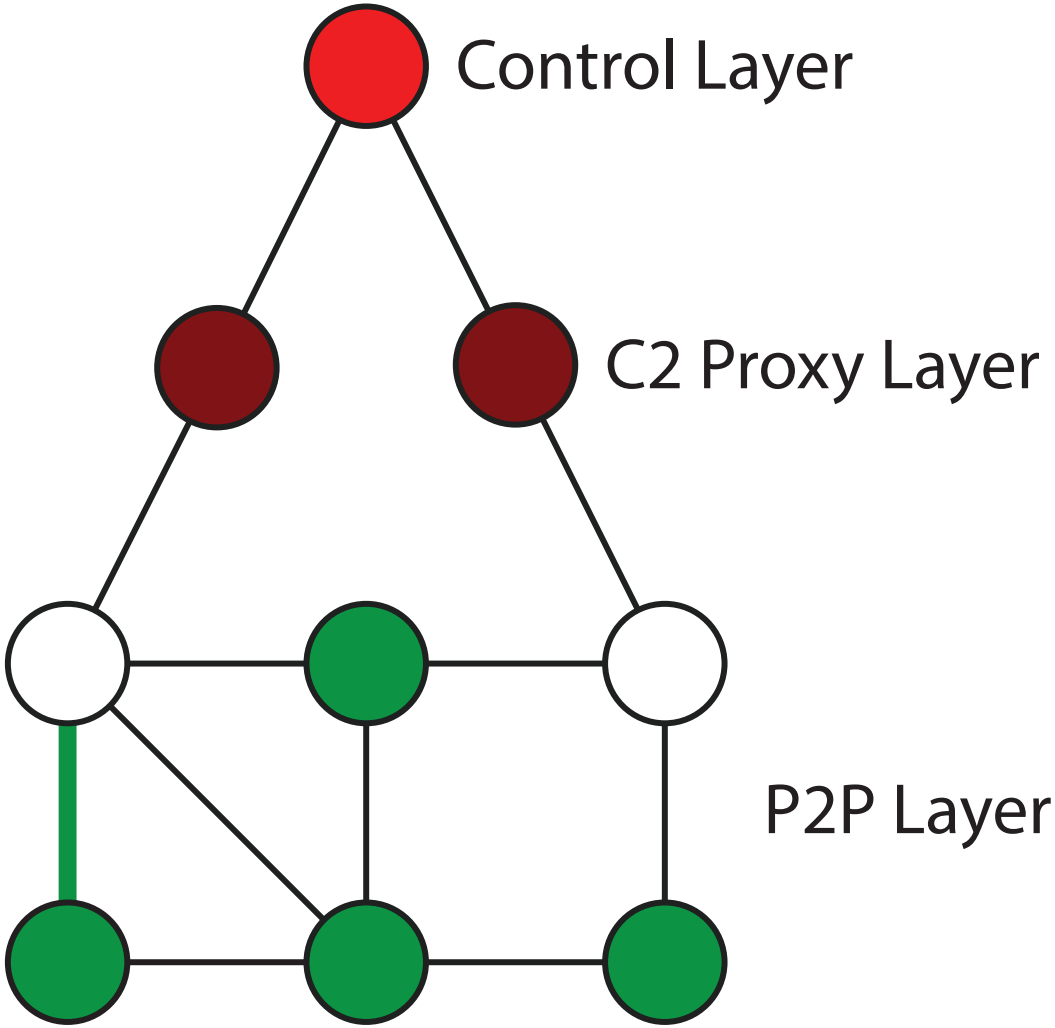
C2 Communication



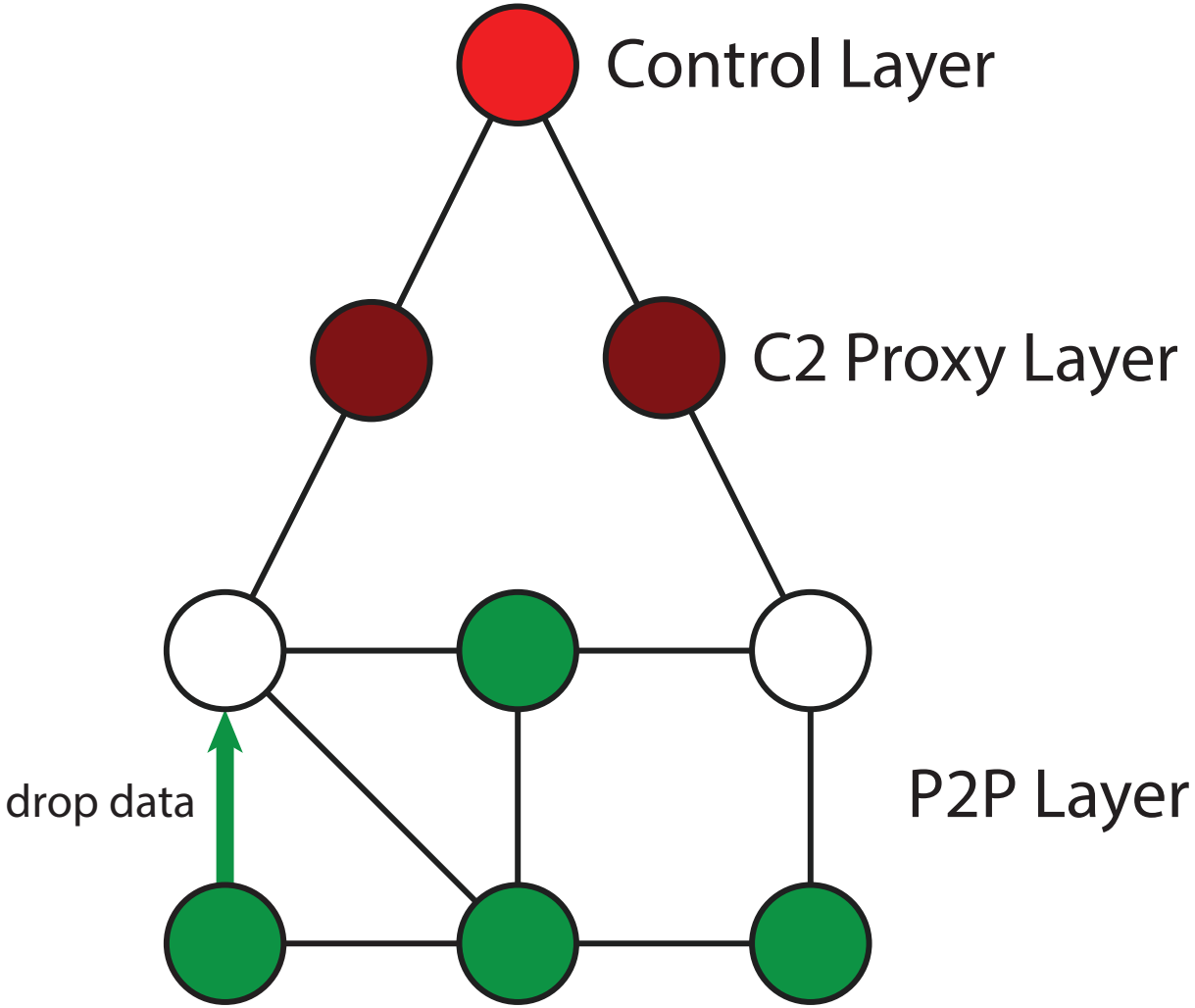
C2 Communication



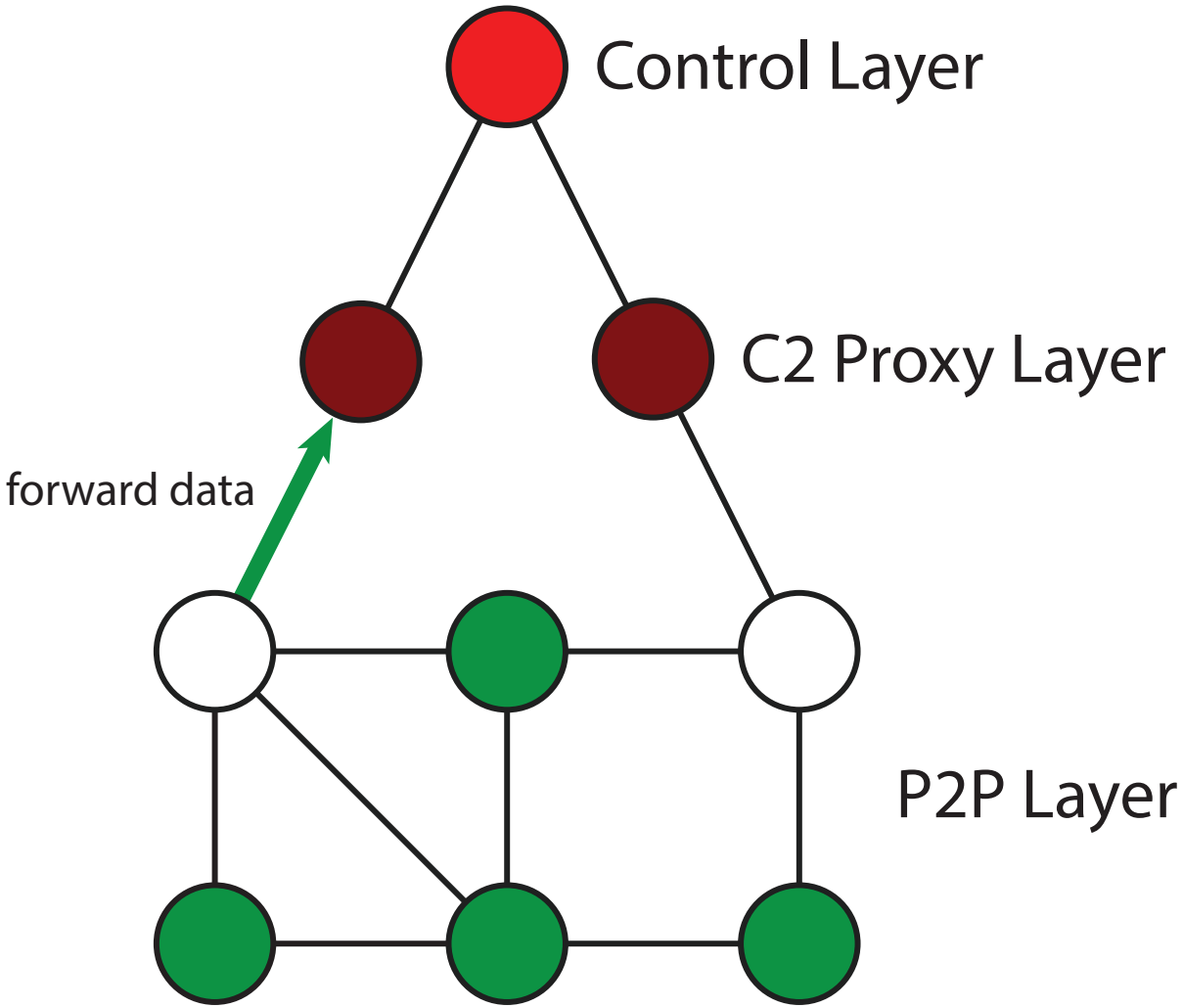
C2 Communication



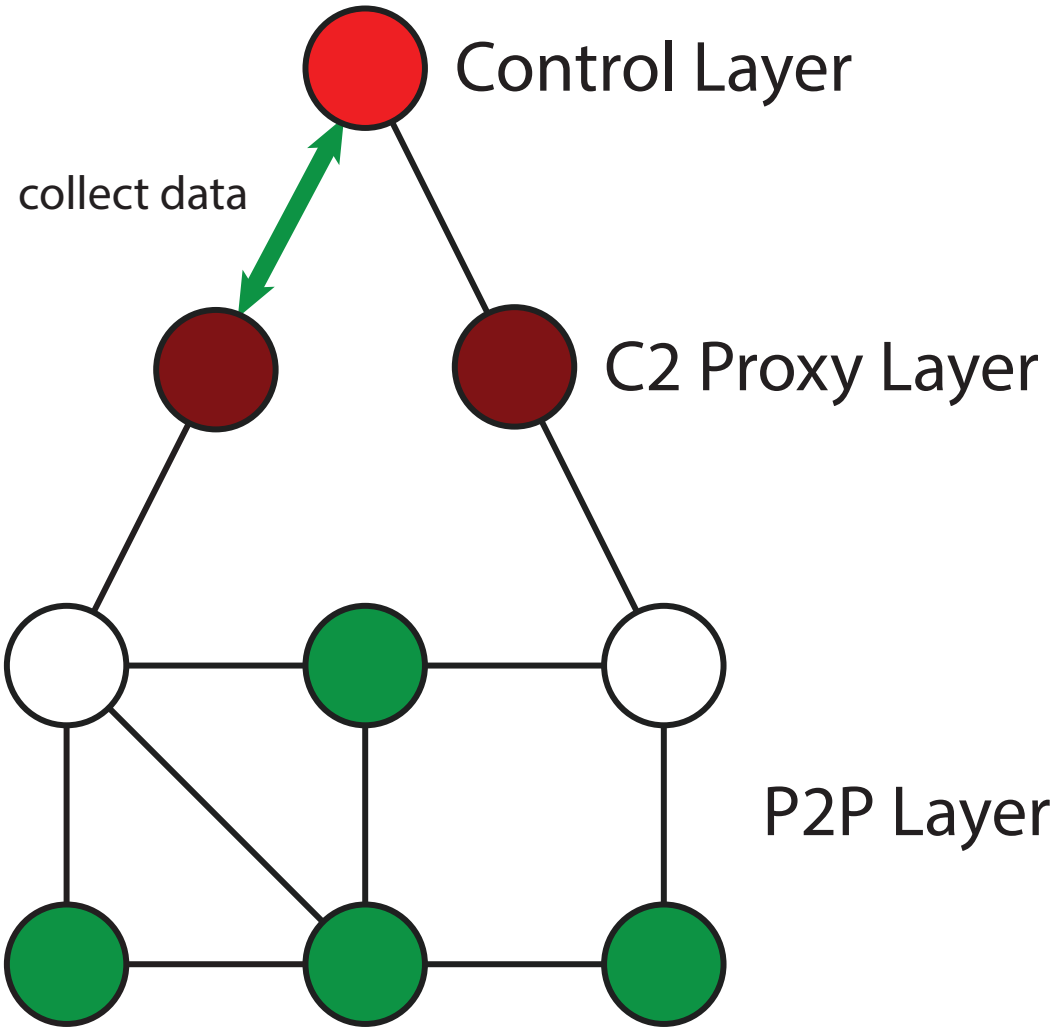
C2 Communication



C2 Communication



C2 Communication



Extracting a Zeus binary

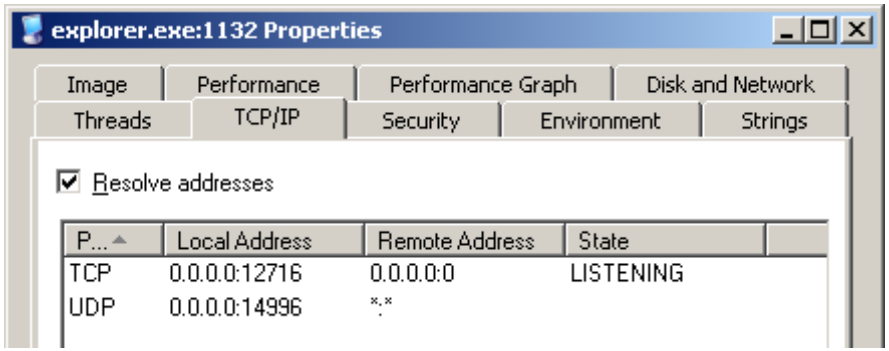
- In an ideal world, we could just get a Zeus sample and disassemble it
- Unfortunately, the Zeus authors don't like our lives to be simple
  - Zeus samples have several layers of encryption and obfuscation, and only unpack themselves at runtime
  - Simply disassembling a sample will only reveal encrypted code and obfuscated decryption code

Extracting a Zeus binary

- We run a Zeus sample in a VM and observe its behaviour

Extracting a Zeus binary

- We run a Zeus sample in a VM and observe its behaviour
- As soon as we run Zeus, Windows Explorer is infected and opens TCP and UDP sockets



- If we dump the VM's memory, we may be able to extract the injected Zeus code

Extracting a Zeus binary

- We use a memory forensics platform called Volatility
- Look for memory ranges marked executable but not listed in the Windows Process Environment Block

Extracting a Zeus binary

- We use a memory forensics platform called Volatility
- Look for memory ranges marked executable but not listed in the Windows Process Environment Block
- We find an MZ header, marking the start of an injected Windows binary → Zeus!

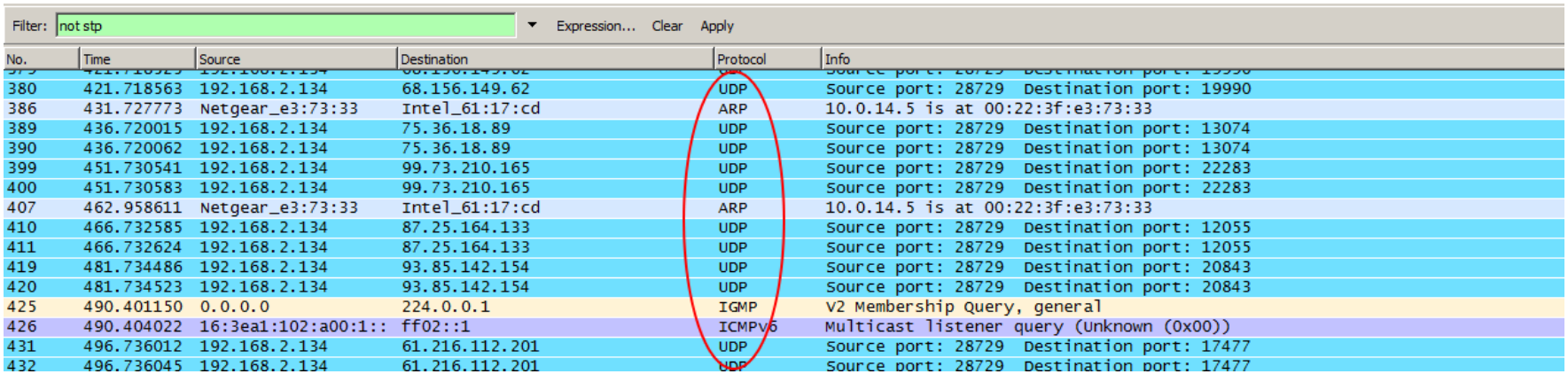
```
$ volatility --profile=WinXPSP3x86 malfind -f zeus.raw -D hidden_dumps/
Volatile Systems Volatility Framework 2.0
Name                 Pid    Start      End        Tag      Hits   Protect
explorer.exe         1280   0x00fb0000 0xfecfff00 VadS     0      PAGE_EXECUTE_RW
Dumped to: hidden_dumps/explorer.exe.2e68960.00fb0000-00fecfff.dmp
0x00fb0000   4d 5a 00 00 00 00 00 00 00 00 00 00 00 00 00 00    MZ..............
0x00fb0010   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ................
0x00fb0020   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ................
0x00fb0030   00 00 00 00 00 00 00 00 00 00 00 00 08 01 00 00    ................
0x00fb0040   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ................
0x00fb0050   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ................
0x00fb0060   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ................
0x00fb0070   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ................
```

Reversing the Zeus P2P communication protocol

- Now we have a Zeus sample we can disassemble... into 268.800 lines of assembly :-(
- The Zeus authors have stripped all symbolic information
  - No variable names
  - No function names
  - No struct definitions
- Where do we begin?

Reversing the Zeus P2P communication protocol

- Since we want to infiltrate the botnet, we're most interested in the communication protocol and encryption used
- Wireshark tells us that Zeus mostly relies on UDP



No.	Time	Source	Destination	Protocol	Info
380	421.718563	192.168.2.134	68.156.149.62	UDP	Source port: 28729 Destination port: 19990
386	431.727773	Netgear_e3:73:33	Intel_61:17:cd	ARP	10.0.14.5 is at 00:22:3f:e3:73:33
389	436.720015	192.168.2.134	75.36.18.89	UDP	Source port: 28729 Destination port: 13074
390	436.720062	192.168.2.134	75.36.18.89	UDP	Source port: 28729 Destination port: 13074
399	451.730541	192.168.2.134	99.73.210.165	UDP	Source port: 28729 Destination port: 22283
400	451.730583	192.168.2.134	99.73.210.165	UDP	Source port: 28729 Destination port: 22283
407	462.958611	Netgear_e3:73:33	Intel_61:17:cd	ARP	10.0.14.5 is at 00:22:3f:e3:73:33
410	466.732585	192.168.2.134	87.25.164.133	UDP	Source port: 28729 Destination port: 12055
411	466.732624	192.168.2.134	87.25.164.133	UDP	Source port: 28729 Destination port: 12055
419	481.734486	192.168.2.134	93.85.142.154	UDP	Source port: 28729 Destination port: 20843
420	481.734523	192.168.2.134	93.85.142.154	UDP	Source port: 28729 Destination port: 20843
425	490.401150	0.0.0.0	224.0.0.1	IGMP	V2 Membership Query, general
426	490.404022	16:3ea1:102:a00:1::	ff02::1	ICMPv6	Multicast listener query (Unknown (0x00))
431	496.736012	192.168.2.134	61.216.112.201	UDP	Source port: 28729 Destination port: 17477
432	496.736045	192.168.2.134	61.216.112.201	UDP	Source port: 28729 Destination port: 17477

- Let's start by looking for `sendto()` and `recvfrom()` calls

Encryption

Zeus runs this interesting-looking code before `sendto`

```
1   ; sub_4287B3(int arg0<eax>, int arg1<edx>, int arg2)
2   cmp     [esp + arg2], edx
3   jz      short loc_1
4   push    eax
5   push    [esp + arg2]
6   push    edx
7   call    sub_409B62
8   jmp     short loc_1
9
10  loc_0:
11  mov     cl, [eax + edx - 1]
12  xor     [eax + edx], cl
13
14  loc_1:
15  dec     eax
16  jnz     short loc_0
17
18  retn    4
```

Encryption

Zeus runs this interesting-looking code before `sendto`

```
1  ; xor_encrypt(int len<eax>, void *dest<edx>, const void *src)
2  cmp     [esp + src], edx      ; are src and dest arrays the same?
3  jz      short loop_preamble   ; if so, go straight to the loop
4  push    eax                   ; else push arguments...
5  push    [esp + src]
6  push    edx
7  call    memcpy                ; ...and call memcpy
8  jmp     short loop_preamble
9
10 loop_main:
11 mov     cl, [eax + edx - 1]   ; load previous byte
12 xor     [eax + edx], cl       ; and xor it with current byte
13
14 loop_preamble:
15 dec     eax
16 jnz     short loop_main
17
18 retn    4
```

Protocol Details

- Let's see if we can make sense of decrypted messages
- Zeus typically splits messages at the 44 byte boundary → header
- Zeus headers show obvious regularities

```
9f017302d522b7743af88d1470ef69425922c9e2864baba50da559736673051b51837d303c97c39b132cb288
07010503d522b7743af88d1470ef69425922c9e2864baba595ac9c8b5befecb554f7de1a97831333b85be023
b3019400ba41939defcbac367d95a805f3a61f28ce02efa90da559736673051b51837d303c97c39b132cb288
da019201ba41939defcbac367d95a805f3a61f28ce02efa995ac9c8b5befecb554f7de1a97831333b85be023
71011600569419ed8075e158922460a5e1cf6c8663f34bb20da559736673051b51837d303c97c39b132cb288
35016801569419ed8075e158922460a5e1cf6c8663f34bb295ac9c8b5befecb554f7de1a97831333b85be023
```

Protocol Details

- The fourth byte exhibits the pattern 2, 3, 0, 1, 0, 1, ...

```
9f017302d522b7743af88d1470ef69425922c9e2864baba50da559736673051b51837d303c97c39b132cb288
07010503d522b7743af88d1470ef69425922c9e2864baba595ac9c8b5befecb554f7de1a97831333b85be023
b3019400ba41939defcbac367d95a805f3a61f28ce02efa90da559736673051b51837d303c97c39b132cb288
da019201ba41939defcbac367d95a805f3a61f28ce02efa995ac9c8b5befecb554f7de1a97831333b85be023
71011600569419ed8075e158922460a5e1cf6c8663f34bb20da559736673051b51837d303c97c39b132cb288
35016801569419ed8075e158922460a5e1cf6c8663f34bb295ac9c8b5befecb554f7de1a97831333b85be023
```

Protocol Details

- The fourth byte exhibits the pattern 2, 3, 0, 1, 0, 1, ...

```
9f017302d522b7743af88d1470ef69425922c9e2864baba50da559736673051b51837d303c97c39b132cb288
07010503d522b7743af88d1470ef69425922c9e2864baba595ac9c8b5befecb554f7de1a97831333b85be023
b3019400ba41939defcbac367d95a805f3a61f28ce02efa90da559736673051b51837d303c97c39b132cb288
da019201ba41939defcbac367d95a805f3a61f28ce02efa995ac9c8b5befecb554f7de1a97831333b85be023
71011600569419ed8075e158922460a5e1cf6c8663f34bb20da559736673051b51837d303c97c39b132cb288
35016801569419ed8075e158922460a5e1cf6c8663f34bb295ac9c8b5befecb554f7de1a97831333b85be023
      type?
```

Protocol Details

- Bytes 5-24 are the same in every two consecutive messages

```
9f017302d522b7743af88d1470ef69425922c9e2864baba50da559736673051b51837d303c97c39b132cb288
07010503d522b7743af88d1470ef69425922c9e2864baba595ac9c8b5befecb554f7de1a97831333b85be023
b3019400ba41939defcbac367d95a805f3a61f28ce02efa90da559736673051b51837d303c97c39b132cb288
da019201ba41939defcbac367d95a805f3a61f28ce02efa995ac9c8b5befecb554f7de1a97831333b85be023
71011600569419ed8075e158922460a5e1cf6c8663f34bb20da559736673051b51837d303c97c39b132cb288
35016801569419ed8075e158922460a5e1cf6c8663f34bb295ac9c8b5befecb554f7de1a97831333b85be023
      type?
```

Protocol Details

- Bytes 5-24 are the same in every two consecutive messages

```
9f017302d522b7743af88d1470ef69425922c9e2864baba50da559736673051b51837d303c97c39b132cb288
07010503d522b7743af88d1470ef69425922c9e2864baba595ac9c8b5befecb554f7de1a97831333b85be023
b3019400ba41939defcbac367d95a805f3a61f28ce02efa90da559736673051b51837d303c97c39b132cb288
da019201ba41939defcbac367d95a805f3a61f28ce02efa995ac9c8b5befecb554f7de1a97831333b85be023
71011600569419ed8075e158922460a5e1cf6c8663f34bb20da559736673051b51837d303c97c39b132cb288
35016801569419ed8075e158922460a5e1cf6c8663f34bb295ac9c8b5befecb554f7de1a97831333b85be023
      type?                    request/reply id?
```

Protocol Details

- Bytes 25-44 are constant per message direction (request/reply)

```
9f017302d522b7743af88d1470ef69425922c9e2864baba50da559736673051b51837d303c97c39b132cb288
07010503d522b7743af88d1470ef69425922c9e2864baba595ac9c8b5befecb554f7de1a97831333b85be023
b3019400ba41939defcbac367d95a805f3a61f28ce02efa90da559736673051b51837d303c97c39b132cb288
da019201ba41939defcbac367d95a805f3a61f28ce02efa995ac9c8b5befecb554f7de1a97831333b85be023
71011600569419ed8075e158922460a5e1cf6c8663f34bb20da559736673051b51837d303c97c39b132cb288
35016801569419ed8075e158922460a5e1cf6c8663f34bb295ac9c8b5befecb554f7de1a97831333b85be023
      type?                    request/reply id?
```

Protocol Details

- Bytes 25-44 are constant per message direction (request/reply)

```
9f017302d522b7743af88d1470ef69425922c9e2864baba50da559736673051b51837d303c97c39b132cb288
07010503d522b7743af88d1470ef69425922c9e2864baba595ac9c8b5befecb554f7de1a97831333b85be023
b3019400ba41939defcbac367d95a805f3a61f28ce02efa90da559736673051b51837d303c97c39b132cb288
da019201ba41939defcbac367d95a805f3a61f28ce02efa995ac9c8b5befecb554f7de1a97831333b85be023
71011600569419ed8075e158922460a5e1cf6c8663f34bb20da559736673051b51837d303c97c39b132cb288
35016801569419ed8075e158922460a5e1cf6c8663f34bb295ac9c8b5befecb554f7de1a97831333b85be023
      type?                    request/reply id?                                  sender id?
```

Protocol Details

- Second byte is not so obvious – it is always 1

```
9f017302d522b7743af88d1470ef69425922c9e2864baba50da559736673051b51837d303c97c39b132cb288
07010503d522b7743af88d1470ef69425922c9e2864baba595ac9c8b5befecb554f7de1a97831333b85be023
b3019400ba41939defcbac367d95a805f3a61f28ce02efa90da559736673051b51837d303c97c39b132cb288
da019201ba41939defcbac367d95a805f3a61f28ce02efa995ac9c8b5befecb554f7de1a97831333b85be023
71011600569419ed8075e158922460a5e1cf6c8663f34bb20da559736673051b51837d303c97c39b132cb288
35016801569419ed8075e158922460a5e1cf6c8663f34bb295ac9c8b5befecb554f7de1a97831333b85be023
 ??   type?                  request/reply id?                              sender id?
```

Protocol Details

- First and third bytes seem totally random

```
9f017302d522b7743af88d1470ef69425922c9e2864baba50da559736673051b51837d303c97c39b132cb288
07010503d522b7743af88d1470ef69425922c9e2864baba595ac9c8b5befecb554f7de1a97831333b85be023
b3019400ba41939defcbac367d95a805f3a61f28ce02efa90da559736673051b51837d303c97c39b132cb288
da019201ba41939defcbac367d95a805f3a61f28ce02efa995ac9c8b5befecb554f7de1a97831333b85be023
71011600569419ed8075e158922460a5e1cf6c8663f34bb20da559736673051b51837d303c97c39b132cb288
35016801569419ed8075e158922460a5e1cf6c8663f34bb295ac9c8b5befecb554f7de1a97831333b85be023
 ??   type?                  request/reply id?                              sender id?
```

Protocol Details

- First byte seems random because it is → no xor encryption
- Second byte is TTL → only used in some message types
- Third byte is amount of random padding → confuse network IDS

```
9f017302d522b7743af88d1470ef69425922c9e2864baba50da559736673051b51837d303c97c39b132cb288
07010503d522b7743af88d1470ef69425922c9e2864baba595ac9c8b5befecb554f7de1a97831333b85be023
b3019400ba41939defcbac367d95a805f3a61f28ce02efa90da559736673051b51837d303c97c39b132cb288
da019201ba41939defcbac367d95a805f3a61f28ce02efa995ac9c8b5befecb554f7de1a97831333b85be023
71011600569419ed8075e158922460a5e1cf6c8663f34bb20da559736673051b51837d303c97c39b132cb288
35016801569419ed8075e158922460a5e1cf6c8663f34bb295ac9c8b5befecb554f7de1a97831333b85be023
ttl pad type                  request/reply id?                                sender id?
```

Protocol Details

P2P Message Example: Peer List Request

- Type: 02
- Padding Length: 50
- Session ID
- Bot ID

```
00000000  6f 01 02 50 c5 77 aa  be 9d 03 a4 99 60 1d 2d f4
00000010  13 9e 9c 81 6b ef 8c  e7 4b ce 83 d7 14 21 67 29
00000020  aa c4 b7 b2 38 f1 4d  89 cf 55 eb 53 4d 31 9b 94
00000030  5c f5 53 57 24 87 7a  6b bd 3a 24 0a 3b d2 f6 9a
00000040  01 a6 b5 e0 ab 4e a6  35 86 ca 4c 9e b3 d8 a1 4a
00000050  f0 ee c9 b6 72 c2 4b  9a c6 52 e4 12 58 ed fd 45
00000060  12 da 17 dc 98 b8 17  59 ab 1e 0a 4f 6c 7d 8e f7
00000070  b3 a2 a9 37 86 36 3a  f7 2e 26 25 64 b1 44 cf fe
00000080  2e d7 46 97 3c 35 de  ff e2 b4 8d 14 53 3b 35 8a
00000090  ca 88 38 f7 4a 14 74  cb 29 af 99 a7 ba 10 e6 73
000000a0  8d 9f 29 24 72 7b 65  ad 1b ef ef b7 a2 ae 2b 97
000000b0  df ea 28 8a 2f 4a 06  2a ed 5b aa da 51 a7 a5 06
000000c0  76 be 4a 07 35 3a 56  25 bf 09 9d 67 b3 c6 01 5f
000000d0  d6 48 7e b8 65 d1 58  41 65 4f 01
```

Protocol Details

Message Types

No.	Meaning	Comment
00	Version Request	Ask for binary/config version
01	Version	Report version information
02	Peer List Request	Ask peer for some neighbors
03	Peer List	Report up to 10 neighbor peers
04	Data Request	Ask for binary or config
05	Data	Current binary or config
06	Proxy List	Contains list of proxy nodes
50	Proxy Announcement	Used to propagate a proxy node
204	C2 Message	Sent to proxy nodes, wraps C2 data

A Backup Channel

Domain Name Generation

- Bots that cannot connect to the botnet launch a DGA
- Generates 1000 domain names per week
  - Starts trying from random initial domain
  - Downloads new seed peer list

```
$ ./zeus_dga.py -d 23.01.2013 -o zeus-dga-domains.txt

Generated 1000 domain names:
   .biz: 166
   .com: 266
  .info: 133
   .net: 134
   .org: 134
    .ru: 167
```

zxqcmbamypfmtuwqoibuoy.ru
xthzltayhiusmbdiblrrgukvts.com
fqgyssobrgtopmftxslbqeqy.net
nvqmjsfzdcmxsmdsgofeil.org
...

Weaknesses in the P2P Protocol

Peer list poisoning

- Peer list responses contain peers close to a query ID
- Possible to push a peer list update to a node
  - Bots store sender if $|peerlist| < 50$
  - If remote peer known but different IP address: Update entry
- Can "poison" a bot by claiming to be a known bot with new IP

186.88.196.115
142.163.184.154
208.41.173.138
95.104.110.191

Peer List Request
ID: 4; IP: 192.168.0.1

Weaknesses in the P2P Protocol

Peer list poisoning

- Peer list responses contain peers close to a query ID
- Possible to push a peer list update to a node
  - Bots store sender if $|peerlist| < 50$
  - If remote peer known but different IP address: Update entry
- Can "poison" a bot by claiming to be a known bot with new IP

186.88.196.115
142.163.184.154
208.41.173.138
192.168.0.1

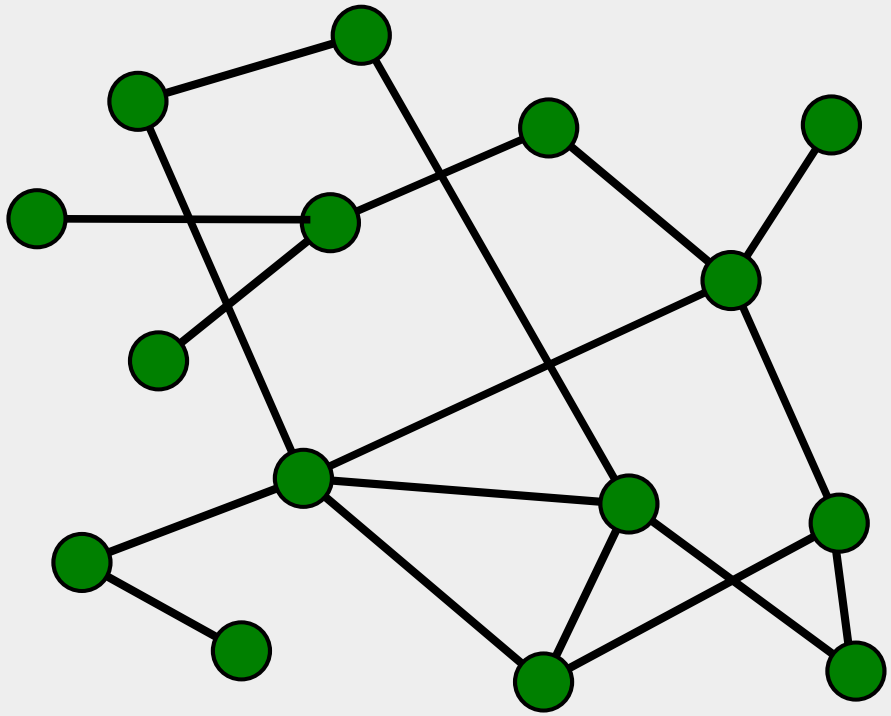
Peer List Response

First Sinkholing Attack

First Sinkholing Attack

Attack Plan

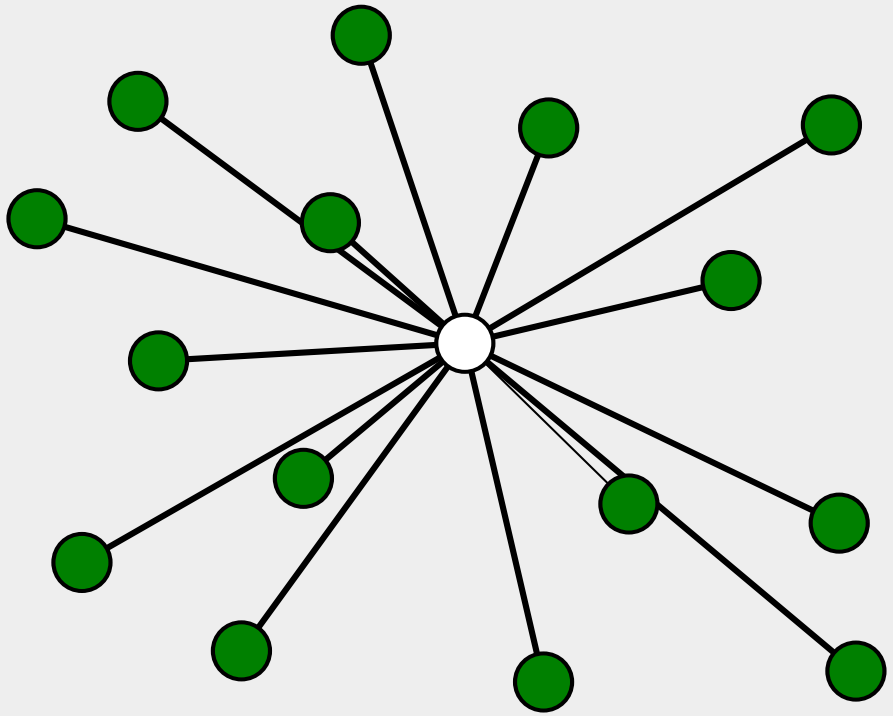
- Goal: Redirect bots to sinkholes, prevent normal operation
- Method: Replace peer list entries with our sinkholes
- Bonus: Log bots as they check in at the sinkholes, report them



First Sinkholing Attack

Attack Plan

- Goal: Redirect bots to sinkholes, prevent normal operation
- Method: Replace peer list entries with our sinkholes
- Bonus: Log bots as they check in at the sinkholes, report them



First Sinkholing Attack

Challenges

- New peer list entries are verified before acceptance
  - Sinkholes must act like valid bots
- Bots do not allow duplicate IP addresses
  - To overwrite 50 peer list entries, we need 50 IPs
- Cannot actively poison bots that are not externally reachable
  - But already poisoned peers will propagate the sinkholes to these
  - Still not possible to replace entries
  - Fully poisoned NATed nodes remain poisoned forever
- Unable to determine a peer's complete peer list
  - Bots only return up to 10 entries
  - Peer lists are usually bigger
  - No way to deterministically enumerate it

Poisoning Example

- Bot peer list before the attack:

Bot ID	IP address	Port
c2ad2c7621e8cc9057e8ee0fe678acdf216f8d0f	186.88.196.115	10355
c28df459e506e3fbaf0fe4e09c3e8a1fcc697f39	142.163.184.154	12631
3e6684b8016ad93410bc94803d1da9502239f582	208.41.173.138	13850
c19aff3ecf6a2e0443640baad118ee528ccd43ce	95.104.110.191	15550
3d0445ac21017cf284191485fc045e23a4d65dba	75.38.136.56	10169
5b68273785dc1a0e19d1461ccb5688e150528697	98.203.40.174	21918
e10fa5a555f3653837ceef2380da034dc7190261	174.134.88.28	19433
c1ff72dda4362153a43079ed35301537aaf56634	74.234.107.231	25975
93b2028482d876a9dd4a3b01b2265956f189aed4	190.206.20.161	29346
c3575bcd52b97c1484bee81dfa1bfcf5d3fd1343	79.113.161.10	16824

Poisoning Example

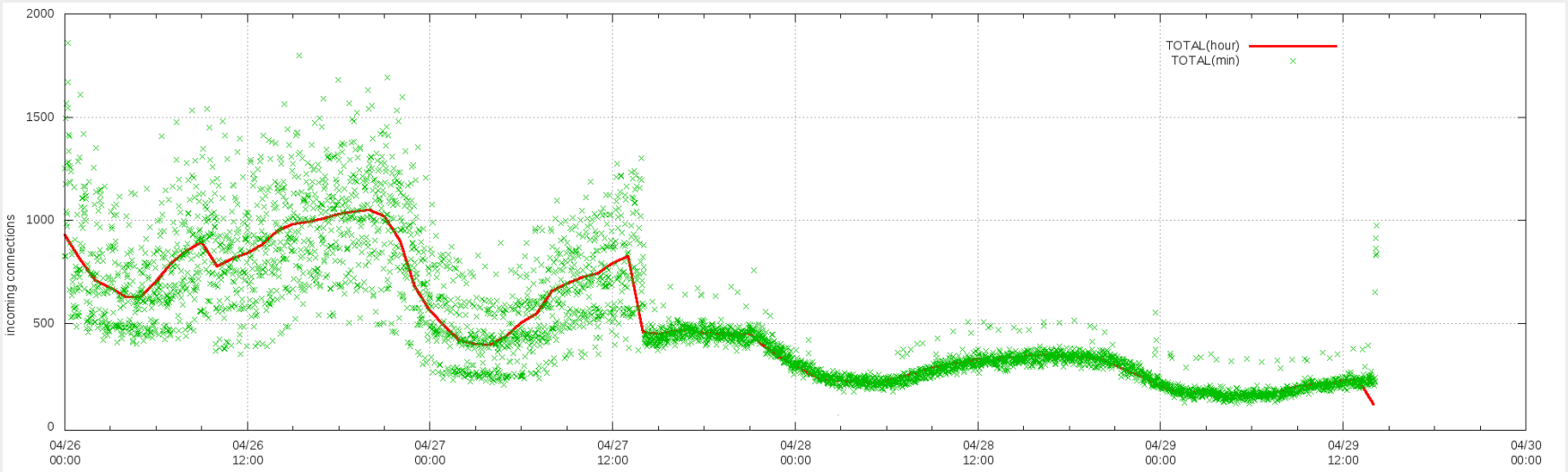
- Bot peer list *after* the attack:

Bot ID	IP address	Port
c2ad2c7621e8cc9057e8ee0fe678acdf216f8d0f	192.168.1.1	13337
c28df459e506e3fbaf0fe4e09c3e8a1fcc697f39	192.168.1.2	13337
3e6684b8016ad93410bc94803d1da9502239f582	192.168.1.3	13337
c19aff3ecf6a2e0443640baad118ee528ccd43ce	192.168.1.4	13337
3d0445ac21017cf284191485fc045e23a4d65dba	192.168.1.5	13337
5b68273785dc1a0e19d1461ccb5688e150528697	192.168.1.6	13337
e10fa5a555f3653837ceef2380da034dc7190261	192.168.1.7	13337
c1ff72dda4362153a43079ed35301537aaf56634	192.168.1.8	13337
93b2028482d876a9dd4a3b01b2265956f189aed4	192.168.1.9	13337
c3575bcd52b97c1484bee81dfa1bfcf5d3fd1343	192.168.1.10	13337

First Sinkholing Attack

Results

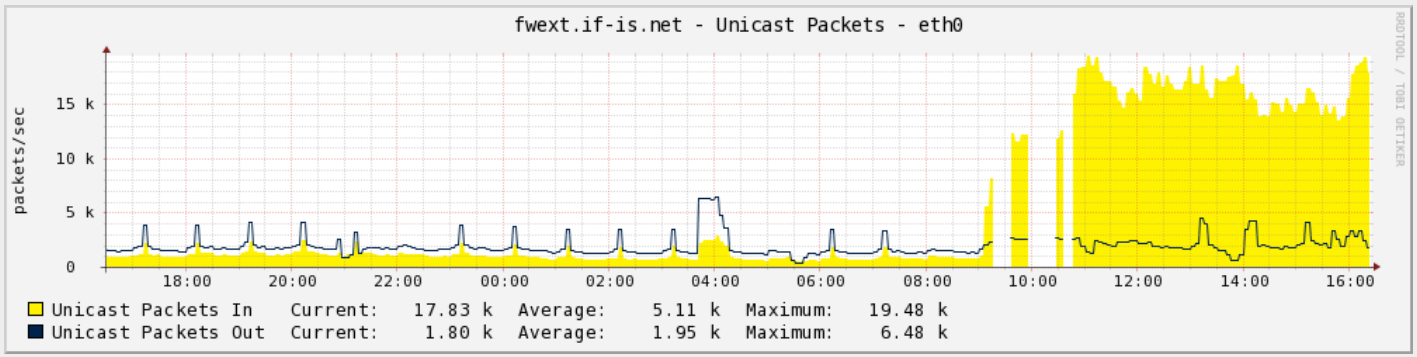
- First sinkholing attack launched on April 27th 2012
- Sharp decline in number of bots contacting CERT.pl's monitor



Attacks Against the Sinkholes

- On May 10th 2012, the botmasters launched a counterattack
  - DDoS attack against our sinkholes
  - Hardcoded blacklist filters several IP ranges
  - IP address /20 Filter

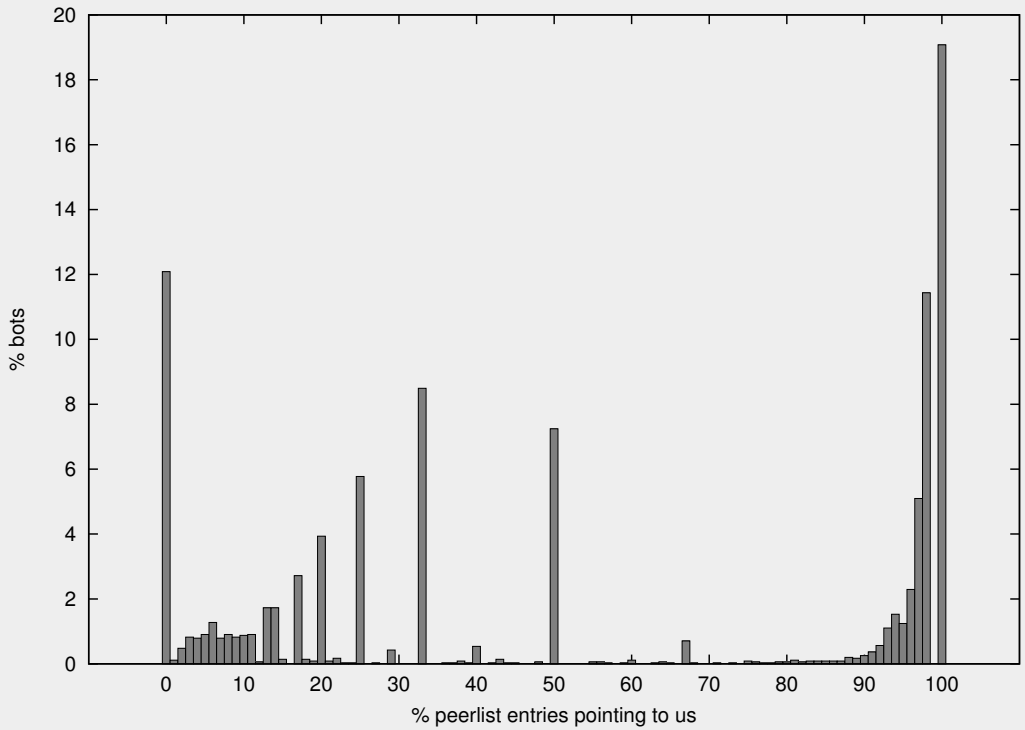
```
64.233.160.0/19    91.212.136.0/24    195.168.53.48/16
64.88.164.160/27   91.213.143.0/24     195.74.76.0/24
   65.52.0.0/14     128.130.0.0/15    207.46.130.0/16
  66.148.64.0/18    131.107.0.0/16    208.118.60.0/20
   84.74.14.0/24     150.26.0.0/24     212.5.80.0/26
 85.222.116.0/22   193.175.86.0/24    212.67.88.64/19
  91.103.64.0/22    193.71.68.0/24    91.199.104.0/24
  195.164.0.0/16
```



Effect of the First Attack

Assessment Results

- We lost some bots, but retained contact with ∼20%
  - Fully poisoned NATed nodes are unable to recover
  - They did not receive the May 10^{th} update
- Poisoned peers are still continuously contacting our sinkholes



Second Sinkholing Attack

Second Sinkholing Attack

Improving the Attack Strategy

- New strategy: Shrink peer lists
  - Gain access to the whole list
  - Fewer sinkhole IP addresses needed
  - Reduce load on the sinkholes
- Overwrite a few peer list entries with sinkhole addresses
- Invalidate the rest (IP spoofing)
  - Probe bot for peer list
  - Overwrite received entries, *only change ports*
  - Set some of the entries to sinkhole IPs
- Much more difficult for botmasters to notice

Poisoning Example

- Bot peer list before the attack:

Bot ID	IP address	Port
c2ad2c7621e8cc9057e8ee0fe678acdf216f8d0f	186.88.196.115	10355
c28df459e506e3fbaf0fe4e09c3e8a1fcc697f39	142.163.184.154	12631
3e6684b8016ad93410bc94803d1da9502239f582	208.41.173.138	13850
c19aff3ecf6a2e0443640baad118ee528ccd43ce	95.104.110.191	15550
3d0445ac21017cf284191485fc045e23a4d65dba	75.38.136.56	10169
5b68273785dc1a0e19d1461ccb5688e150528697	98.203.40.174	21918
e10fa5a555f3653837ceef2380da034dc7190261	174.134.88.28	19433
c1ff72dda4362153a43079ed35301537aaf56634	74.234.107.231	25975
93b2028482d876a9dd4a3b01b2265956f189aed4	190.206.20.161	29346
c3575bcd52b97c1484bee81dfa1bfcf5d3fd1343	79.113.161.10	16824

Poisoning Example

- Bot peer list *after* the attack:

Bot ID	IP address	Port
c2ad2c7621e8cc9057e8ee0fe678acdf216f8d0f	186.88.196.115	22945
c28df459e506e3fbaf0fe4e09c3e8a1fcc697f39	142.163.184.154	10361
3e6684b8016ad93410bc94803d1da9502239f582	192.168.1.1	14521
c19aff3ecf6a2e0443640baad118ee528ccd43ce	95.104.110.191	24540
3d0445ac21017cf284191485fc045e23a4d65dba	75.38.136.56	12954
5b68273785dc1a0e19d1461ccb5688e150528697	98.203.40.174	13953
e10fa5a555f3653837ceef2380da034dc7190261	10.0.0.1	25486
c1ff72dda4362153a43079ed35301537aaf56634	74.234.107.231	21953
93b2028482d876a9dd4a3b01b2265956f189aed4	190.206.20.161	17435
c3575bcd52b97c1484bee81dfa1bfcf5d3fd1343	79.113.161.10	12653

Poisoning Example

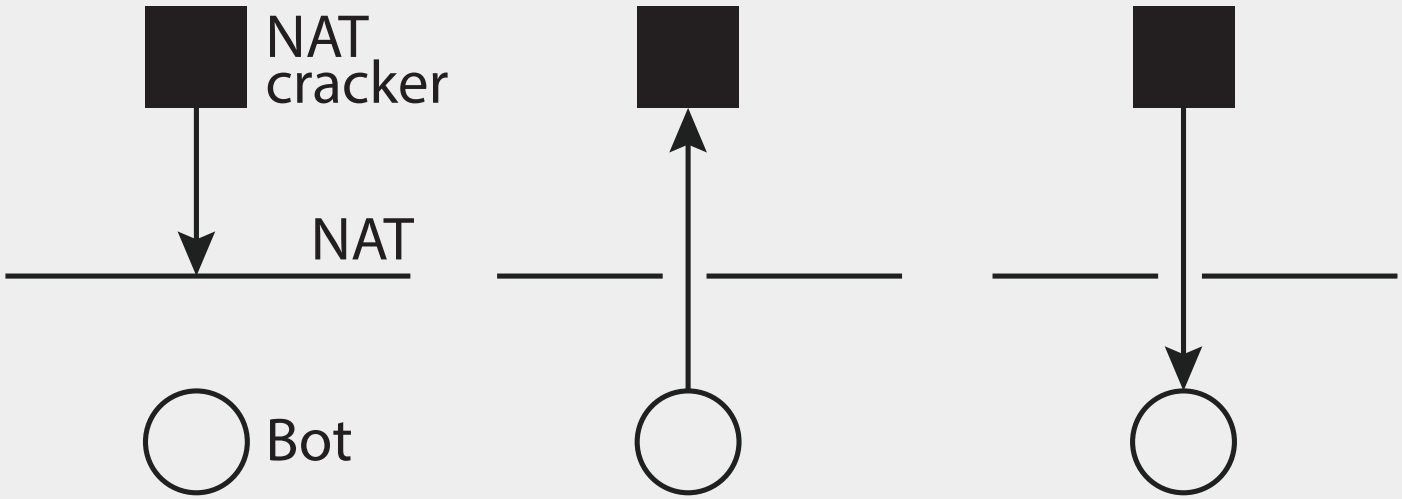
- Bot peer list after three validation rounds:

Bot ID	IP address	Port
3e6684b8016ad93410bc94803d1da9502239f582	192.168.1.1	14521
e10fa5a555f3653837ceef2380da034dc7190261	10.0.0.1	25486

Dealing with NAT

Introducing NAT Crackers

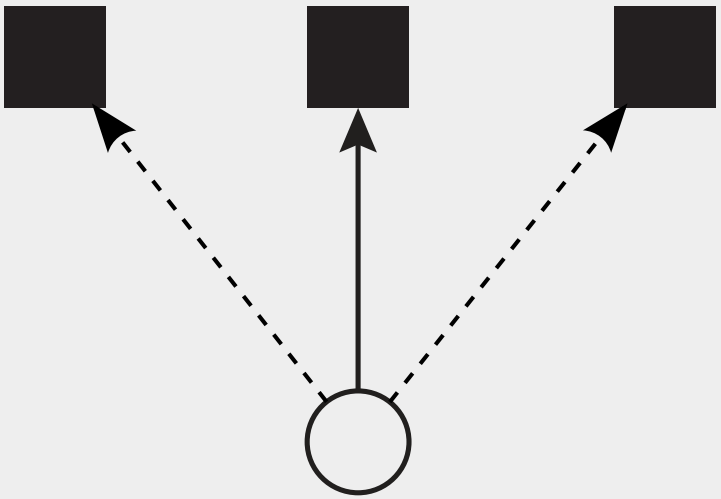
- Sinkholes cannot shrink peer lists of NATed peers
- Sinkholes also inject NAT Crackers
  - Bots contact NAT Crackers before accepting them
  - NAT crackers deliberately fail verification
  - But keep punch hole open
  - Peer list entries can be poisoned through that hole



Preventing DGA Fallback

Introducing Moncheris

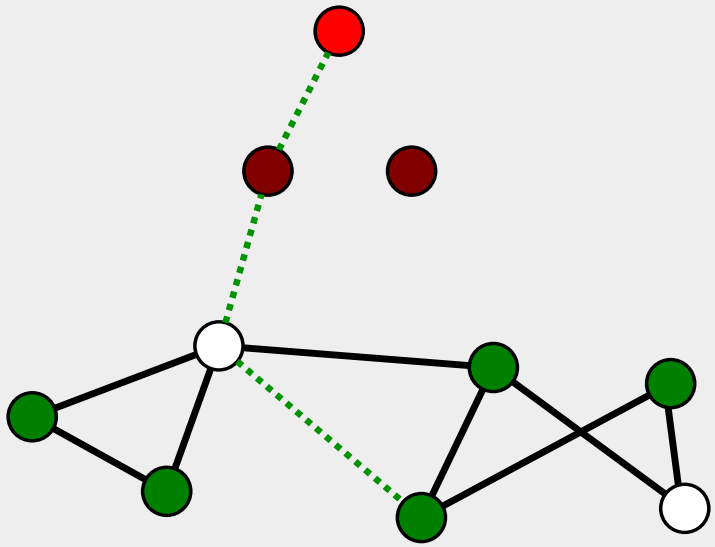
- No peer list changes between two validation rounds: DGA
- Need to make sure that the peer list always changes
  - Sinkholes inject special entries called *Moncheris*
  - **Do** respond to peer list requests
  - **Do not** respond to version requests
  - They are rotated every 30 minutes



Second Sinkholing Attack

Results

- About 97% sinkholed
- DGA prevents a lasting effect
- C2 communication still possible
- Data dropping still possible

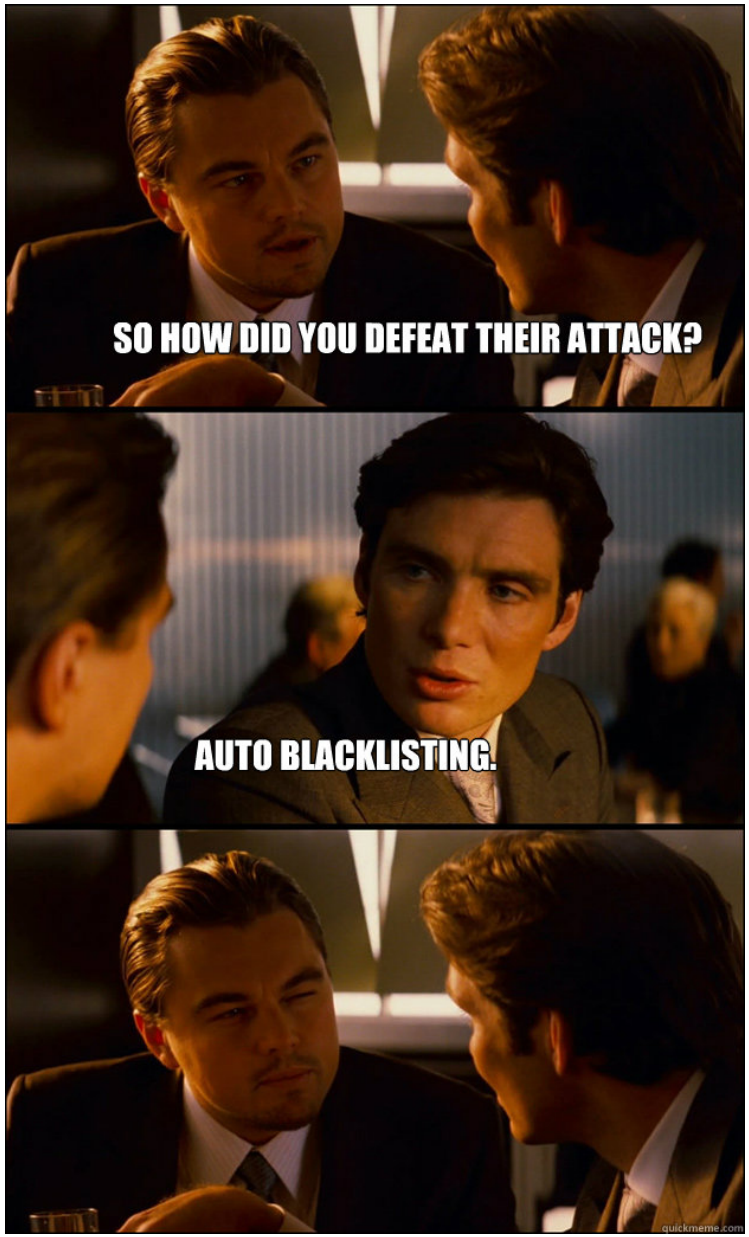


Third Attack(?)

Auto Blacklisting

Reaction to second attack

- Each bot now tracks IPs that make frequent requests
- Hard hitters automatically blacklisted



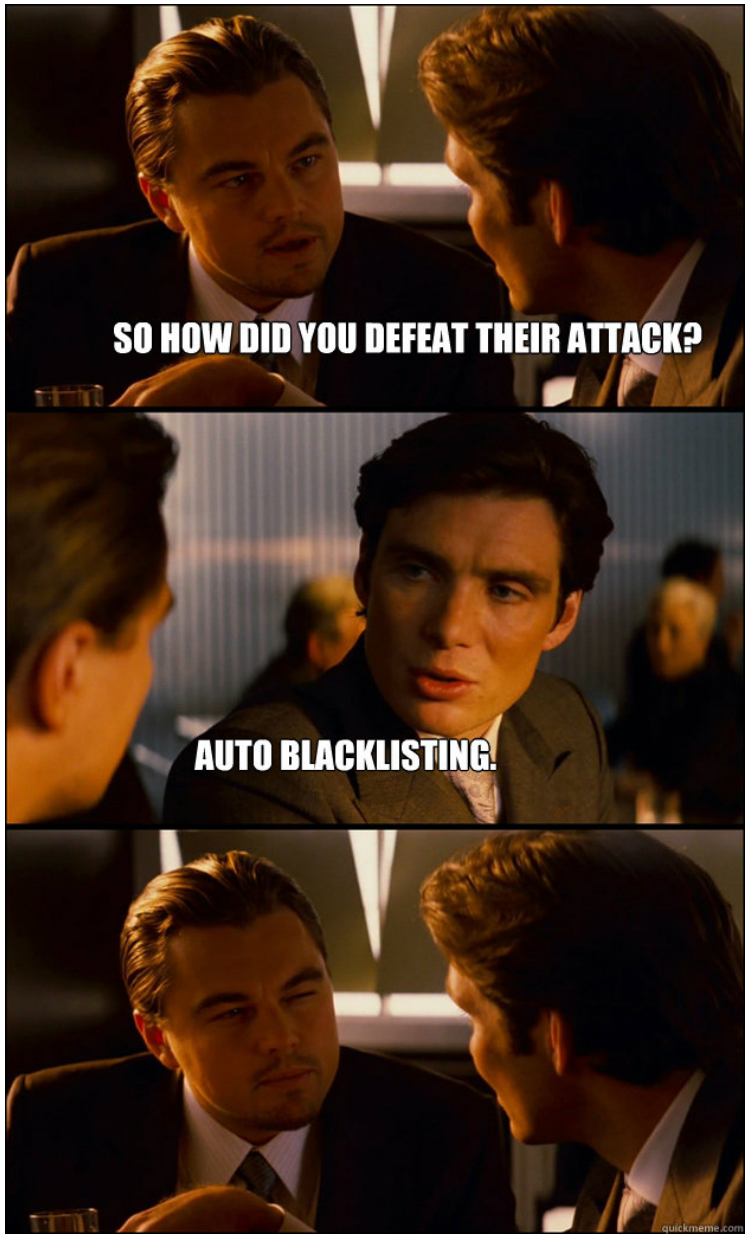
Auto Blacklisting

Reaction to second attack

- Each bot now tracks IPs that make frequent requests
- Hard hitters automatically blacklisted

Overreaction

- Can be used against the botnet
- Send spoofed messages to peer
- That peer will blacklist the sender
- Also affects outgoing messages



Zeus Improves its P2P Resilience

Zeus Improves its P2P Resilience

New encryption strategy

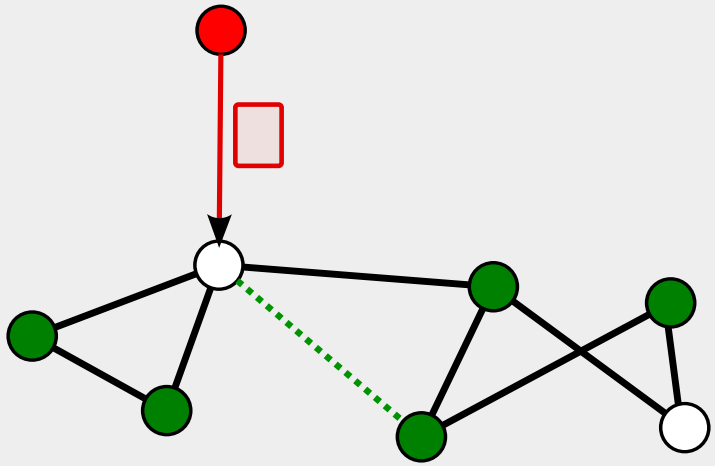
- In June 2013, the botmasters updated the message encryption
- Instead of chained-XOR, use RC4 with recipient's ID as the key
- Creates problems for botnet infiltration
  - To avoid detection, our poisoners use dynamic bot IDs
  - When a bot contacts us, don't know what ID it *thinks* we have
  - Need to know this ID to decrypt

Attacking Zeus Proxy Nodes

Attacking Zeus Proxy Nodes

Announcements

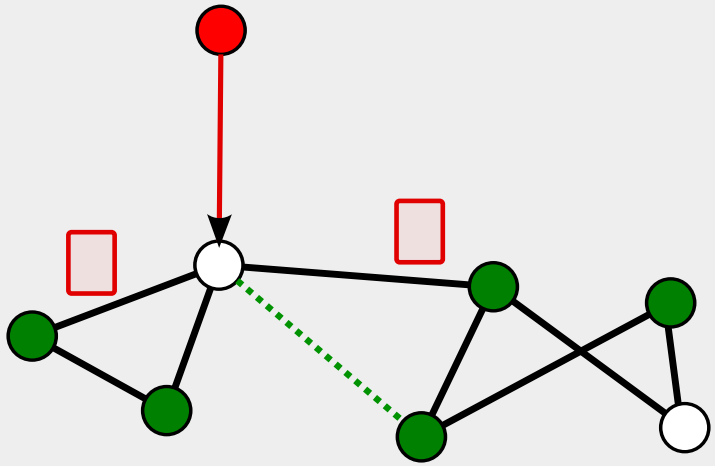
- Periodically, some nodes are appointed as proxies
  - Provided with signed proxy announcement (2048 bit RSA)
    - 2^{nd} header byte is TTL
    - Cannot reach nodes behind NAT
  - Force switch to our proxy by blocking others (ISP collab)



Attacking Zeus Proxy Nodes

Announcements

- Periodically, some nodes are appointed as proxies
  - Provided with signed proxy announcement (2048 bit RSA)
    - 2^{nd} header byte is TTL
    - Cannot reach nodes behind NAT
  - Force switch to our proxy by blocking others (ISP collab)



C2 Protocol: Bot → C2 Server

```
000000  c7 d0 e2 7f e6 75 bd 0f  02 b1 f6 e2 90 ec 9b 72  |.....u.........r|
000010  a7 5b b8 e8 11 24 35 bf  30 82 cc 1a 03 78 a1 70  |.[...$5.0....x.p|
000020  d3 96 ee 80 e4 40 1e 7f  9d 80 ab 35 fb 0f fe 57  |.....@.....5...W|
000030  7c 27 6a b2 a2 e0 42 8e  aa 7c df 17 3c 3e 98 13  ||'j...B..|..<>..|
000040  bd 4e 33 f7 5c da e8 80  92 58 69 ee 5b e8 d4 ce  |.N3.....Xi.[...|
000050  ca ed e8 20 5a b8 42 a0  66 b8 c0 99 25 4e f2 ee  |... Z.B.f...%N..|
000060  08 f0 47 07 ce fb 7d 6e  0d 03 ca 25 27 2a fc 71  |..G.....n...%'*.q|
000070  5a 43 41 41 ee 10 d7 7b  03 98 1b 5d f6 40 cb 95  |ZCAA.......].@..|
000080  92 32 d1 86 76 46 68 0a  61 a7 17 de 55 e8 2f 89  |.2..vFh.a...U./.|
000090  46 0e 3d 1b 3c ca 4d cf  58 14 6e 77 97 2d 04 3a  |F.=.<.M.X.nw.-.:|
0000a0  9d 58 77 d9 5c be c0 99  1c a6 78 99 6c 7a 75 a6  |.Xw.....x.lzu.|
0000b0  36 8d 78 0b bf 53 a9 df  fe cf e9 79 58 be e1 7b  |6.x..S.....yX...|
0000c0  44 d6 42 0a 00 48 e8 96  97 49 6c 71 52 5a 4d 40  |D.B..H...IlqRZM@|
0000d0  bb c2 43 0a 47 0c 8c 68  3f 5b 97 61 8d a2 4e af  |..C.G..h?[.a..N.|
0000e0  dd 6a b5 c7 d4 46 53 4f  0c 4d a0 0b 02 e9 51 9b  |.j...FSO.M....Q.|
0000f0  28 21 78 e8 37 37 95 cf  c3 0a 26 bb 42 aa c1 95  |(!x.77....&.B...|
000100  4c 75 21 42 60 68 e8 a6  b1 b6 76 fb 23 db 5d 0d  |Lu!B`h....v.#.].|
000110  d0 6f 0f 87 4a 86 c7 5a  b4 c0 86 1f ba 32 ba 89  |.o..J..Z.....2..|
000120  d7 06 d8 e7 d0 f5 9b 0d  c1 ff fa b4 54 80 7e c1  |............T.~.|
000130  02 cc 94 e6 c6 58 ab f2  54 b9 6c ac 28 1f 5a 75  |.....X..T.l.(.Zu|
000140  5e 4b 5e b2 1d 35 3c 81  03 64 39 fc 8b db 7b 15  |^K^..5<..d9.....|
000150  fd 66 01 61 02 9d d1 24  f6 56 0d 8c 58 95 1f b2  |.f.a...$.V..X...|
000160  db 03 23 9b 23 d7 e8 7b  75 08 61 a8 42 a5 ec 79  |..#.#...u.a.B..y|
000170  47 9a 2f 37 1a 3a 50 3e  31 79 40 d3 28 99 80 2e  |G./7.:P>1y@.(...|
000180  ca 35 ac 28 a5 9f 53                              |.5.(..S|
```

C2 Protocol: Bot → C2 Server (decrypted)

```
000000  50 4f 53 54 20 2f 77 72  69 74 65 20 48 54 54 50  |POST /write HTTP|
000010  2f 31 2e 31 0d 0a 48 6f  73 74 3a 20 64 65 66 61  |/1.1..Host: defa|
000020  75 6c 74 0d 0a 41 63 63  65 70 74 2d 45 6e 63 6f  |ult..Accept-Enco|
000030  64 69 6e 67 3a 0d 0a 43  6f 6e 6e 65 63 74 69 6f  |ding:..Connectio|
000040  6e 3a 20 63 6c 6f 73 65  0d 0a 43 6f 6e 74 65 6e  |n: close..Conten|
000050  74 2d 4c 65 6e 67 74 68  3a 20 32 33 38 0d 0a 58  |t-Length: 238..X|
000060  2d 49 44 3a 20 32 38 32  38 0d 0a 0d 0a 14 19 f4  |-ID: 2828.......|
000070  55 13 e7 98 b8 f0 35 01  e3 9a 94 96 2a 11 5c be  |U.....5.....*.┼ |
000080  aa ee 00 00 00 00 00 00  00 07 3c d6 3f 15 81 00  |..........<.?...|
000090  8a b7 2f 62 c4 1a 5e d4  3f 9b 5e 88 8e 65 00 00  |../b..^.?.^..e..|
0000a0  00 00 00 00 00 17 00 00  00 17 00 00 00 36 42 7c  |.............6B||
0000b0  9a 24 45 60 94 51 43 79  e1 53 36 0e 95 23 35 7d  |.$E‘.QCy.S6..#5.|
0000c0  95 52 42 7c 66 00 00 00  00 00 00 00 14 00 00 00  |.RB|f...........|
0000d0  14 00 00 00 81 4c f2 55  b1 13 1d b1 4f ad f8 61  |.....L.U....O..a|
0000e0  d4 3f cd 9b ef c8 69 3d  67 00 00 00 00 00 00 00  |.?....i=g.......|
0000f0  08 00 00 00 08 00 00 00  04 6f 5d a5 02 74 0e e2  |.........o]..t..|
000100  c9 00 00 00 00 00 00 00  04 00 00 00 04 00 00 00  |................|
000110  ee 07 3c d6 c8 00 00 00  00 00 00 00 10 00 00 00  |..<.............|
000120  10 00 00 00 15 36 0e a8  f1 06 82 54 f3 9f 6e 0f  |.....6.....T..n.|
000130  9a df 4a 5e ca 00 00 00  00 00 00 00 04 00 00 00  |..J^............|
000140  04 00 00 00 ca 07 3c d6  cb 00 00 00 00 00 00 00  |......<.........|
000150  03 00 00 00 03 00 00 00  84 4c 11                 |.........L.     |
```

C2 Protocol: Bot → C2 Server (decoded)

An *OK* Message

```
0x65: MYBOXX_B4DF7611CF2BC7EC
0x66: e74bce83d714216729aac4b7b238f14d89cf55eb
0x67: chases24
0xC9: 39
0xC8: dd31327e3901be823b9852d952d87688
0xCA: 0
0xCB: OK.
```

C2 Protocol: C2 Server → Bot

```
000000  df 27 43 19 20 2f 57 13  01 4e 7d 1e c5 38 e2 8f  |.'C../W..N...8..|
000010  24 92 65 8b a5 d4 e9 d3  d4 e8 50 ee b1 a9 95 f4  |..e.......P.....|
000020  69 cd 81 6e e5 a2 52 9d  f7 b9 52 81 81 50 6d aa  |i..n..R...R..Pm.|
000030  83 cc 4f 38 24 91 58 2e  8a 4c f3 4b d6 08 c2 1b  |..O8$.X..L.K....|
000040  14 c6 b9 10 0c ad 3b aa  08 f8 9e fa 9d 8f ca f4  |......;.........|
000050  5a 17 c9 22 64 b9 33 f7  fe a1 6f 06 06 3e 0e a9  |Z.."d.3...o..>..|
000060  c4 b1 e5 bd 95 e5 c2 6d  20 f2 aa bd 24 86 81 18  |.......m ...$...|
000070  c1 49 01 7f 54 cd 2a ba  82 7b af d7 35 64 a0 6e  |.I..T.*.....5d.n|
000080  a8 6f e0 e6 73 7e 45 b5  ce 93 fb b4 27 83 31 56  |.o..s~E.....'.1V|
000090  5d 4a 47 65 33 e0 97 12  03 f6 10 de 1a 63 77 c4  |]JGe3........cw.|
0000a0  15 83 b3 96 ff 2a 14 d3  48 ea 2a 82 85 7c 0c fb  |.....*..H.*..|..|
0000b0  85 bd cc 44 fb 30 17 ee  91 db db c7 b6 eb 01 d3  |...D.0..........|
0000c0  7d 71 b1 c5 d7 05 11 c9  93 bc 08 28 11 0b ab 4a  |.q.........(...J|
0000d0  da aa d6 7f 6e 2e 2f f1  16 11 18 19 93 e1 98 51  |....n./........Q|
0000e0  55                                               |U|
```

C2 Protocol: C2 Server → Bot (decrypted)

```
000000  48 54 54 50 2f 31 2e 31  20 32 30 30 20 4f 4b 0d  |HTTP/1.1 200 OK.|
000010  0a 53 65 72 76 65 72 3a  20 6e 67 69 6e 78 0d 0a  |.Server: nginx..|
000020  44 61 74 65 3a 20 54 75  65 2c 20 32 35 20 53 65  |Date: Tue, 25 Se|
000030  70 20 32 30 31 32 20 30  37 3a 30 35 3a 32 37 20  |p 2012 07:05:27 |
000040  47 4d 54 0d 0a 43 6f 6e  74 65 6e 74 2d 54 79 70  |GMT..Content-Typ|
000050  65 3a 20 74 65 78 74 2f  68 74 6d 6c 0d 0a 54 72  |e: text/html..Tr|
000060  61 6e 73 66 65 72 2d 45  6e 63 6f 64 69 6e 67 3a  |ansfer-Encoding:|
000070  20 63 68 75 6e 6b 65 64  0d 0a 43 6f 6e 6e 65 63  | chunked..Connec|
000080  74 69 6f 6e 3a 20 63 6c  6f 73 65 0d 0a 56 61 72  |tion: close..Var|
000090  79 3a 20 41 63 63 65 70  74 2d 45 6e 63 6f 64 69  |y: Accept-Encodi|
0000a0  6e 67 0d 0a 0d 0a 33 30  0d 0a c3 68 94 21 75 68  |ng....30...h.!uh|
0000b0  44 06 21 7a 5d 30 c5 ef  22 35 0d da 58 7e 30 00  |D.!z]0.."5..X~0.|
0000c0  00 00 00 00 00 00 07 3c  d6 3f d4 1d 8c d9 8f 00  |.......<.?......|
0000d0  b2 04 e9 80 09 98 ec f8  42 7e 0d 0a 30 0d 0a 0d  |........B~..0...|
0000e0  0a                                                |.               |
```

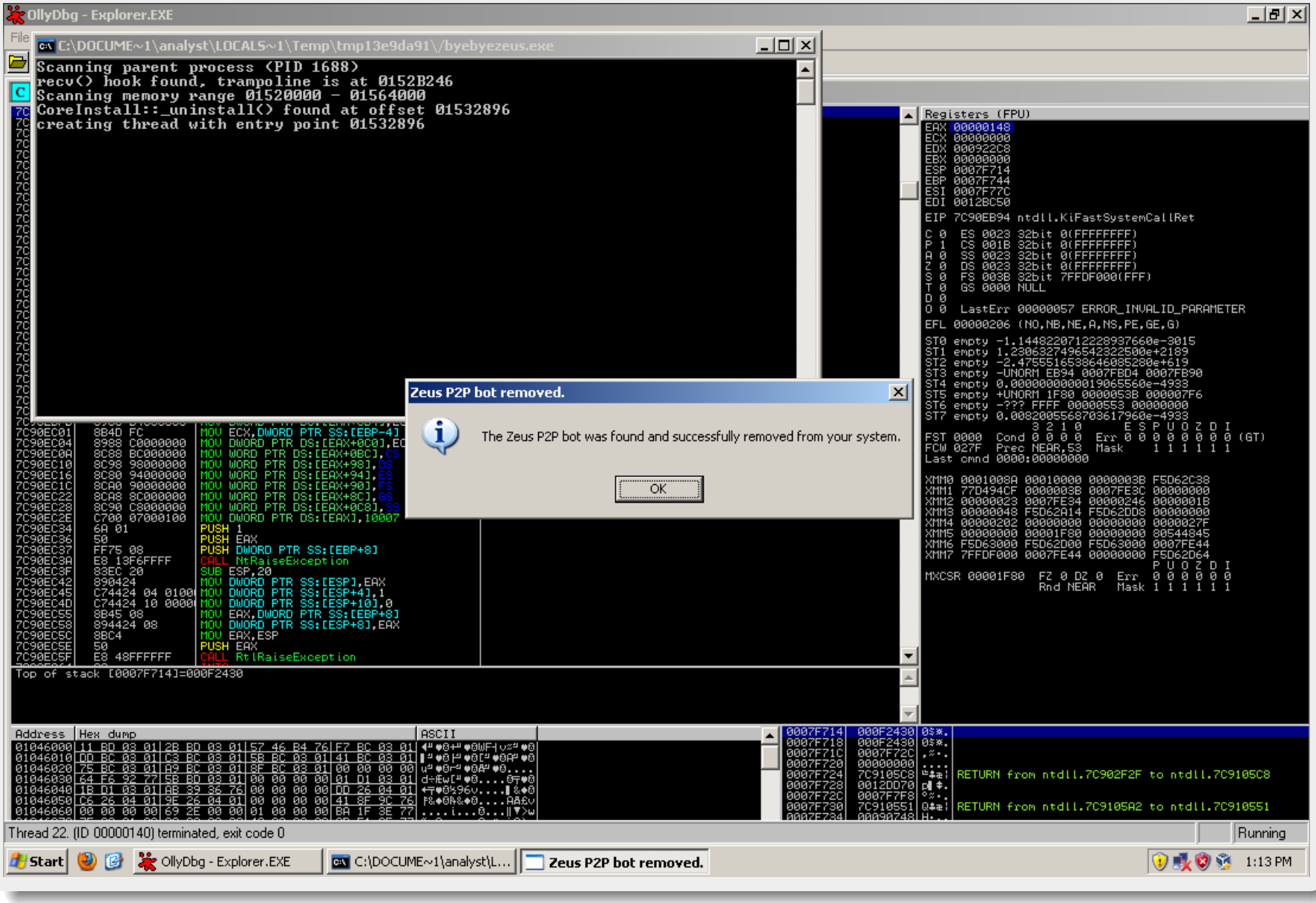
Attacking Zeus Proxy Nodes

Serving Commands

- C2 protocol lacks authentication
- Can in principle serve own commands

```
EncryptedString <0D3h, 9, offset aDdos_type> ; "ddos_type"
EncryptedString <11h, 0Ch, offset aDdos_address> ; "ddos_address"
EncryptedString <97h, 8, offset aDdos_url> ; "ddos_url"
EncryptedString <7Fh, 0Ch, offset aDdos_execute> ; "ddos_execute"
EncryptedString <93h, 0Bh, offset aOs_shutdown> ; "os_shutdown"
EncryptedString <31h, 9, offset aOs_reboot> ; "os_reboot"
EncryptedString <87h, 0Dh, offset aBot_uninstall> ; "bot_uninstall"
EncryptedString <0FEh, 0Ah, offset aBot_bc_add> ; "bot_bc_add"
EncryptedString <0ABh, 0Dh, offset aBot_bc_remove> ; "bot_bc_remove"
EncryptedString <9Ch, 16h, offset aBot_httpinject_di> ; "bot_httpinject_disable"
EncryptedString <99h, 15h, offset aBot_httpinject_en> ; "bot_httpinject_enable"
EncryptedString <32h, 14h, offset aFs_find_add_keywo> ; "fs_find_add_keywords"
EncryptedString <91h, 0Fh, offset aFs_find_execute> ; "fs_find_execute"
EncryptedString <22h, 0Ch, offset aFs_pack_path> ; "fs_pack_path"
EncryptedString <99h, 0Ch, offset aUser_destroy> ; "user_destroy"
EncryptedString <0E7h, 0Bh, offset aUser_logoff> ; "user_logoff"
EncryptedString <9Ah, 0Ch, offset aUser_execute> ; "user_execute"
EncryptedString <7Fh, 10h, offset aUser_cookies_get> ; "user_cookies_get"
EncryptedString <90h, 13h, offset aUser_cookies_remo> ; "user_cookies_remove"
EncryptedString <50h, 0Eh, offset aUser_certs_get> ; "user_certs_get"
EncryptedString <0E0h, 11h, offset aUser_certs_remove> ; "user_certs_remove"
EncryptedString <0D1h, 0Eh, offset aUser_url_block> ; "user_url_block"
EncryptedString <1, 10h, offset aUser_url_unblock> ; "user_url_unblock"
EncryptedString <80h, 11h, offset aUser_homepage_set> ; "user_homepage_set"
EncryptedString <32h, 15h, offset aUser_emailclients> ; "user_emailclients_get"
EncryptedString <42h, 14h, offset aUser_flashplayer_> ; "user_flashplayer_get"
EncryptedString <9Fh, 17h, offset aUser_flashplaye_0> ; "user_flashplayer_remove"
```

Remote Cleanup



Operation Tovar

Operation Tovar

Game over for GameOver

- Early attacks have shown that attacks against isolated components are not effective
- On May 30 2014, we launched a final attack against P2P Zeus
  - FBI, Dell SecureWorks, Crowdstrike, U Saarland, Fox-IT, ...
  - Sinkholing + Proxy layer attack (no uninstalls) + DGA/backend takeover (FBI) + Arrests (FBI)

Final Results

- Attack almost ruined by McAfee info leak, disaster narrowly averted
- GOZ down for good!
- First true P2P botnet takedown $\rightarrow$ precedent for future attacks

Conclusion

Lessons Learned

- P2P botnets come with risks, not just benefits
  - Crawling/sensor injection $\rightarrow$ IP enumeration
  - Bugs allow for infiltration/sinkholing
- Legal issues a major hindrance
  - Botnets survive despite obvious vulnerabilities (Sality v3)
  - Who is responsible for (risky) takedowns?
- Too easy to reboot a botnet when taken down
  - Lasting success depends on coordinated effort (including arrests)
  - Focus on droppers/pay-per-install?