Operation Tainted Love | Chinese APTs Target Telcos in New Attacks

Aleksandar Milenkoski

By Aleksandar Milenkoski, Juan Andres Guerrero-Saade, and Joey Chen, in collaboration with QGroup

Executive Summary

- In Q1 of 2023, SentinelLabs observed initial phases of attacks against telecommunication providers in the Middle East.
- We assess that this activity represents an evolution of tooling associated with Operation Soft Cell.
- While it is highly likely that the threat actor is a Chinese cyberespionage group in the nexus of Gallium and APT41, the exact grouping remains unclear.
- SentinelLabs observed the use of a well-maintained, versioned credential theft capability and a new dropper mechanism indicative of an ongoing development effort by a highly-motivated threat actor with specific tasking requirements.

Overview

In collaboration with QGroup GmbH, SentinelLabs recently observed initial threat activities targeting the telecommunication sector. We assess it is highly likely that these attacks were conducted by a Chinese cyberespionage actor related to the Operation Soft Cell campaign.

The initial attack phase involves infiltrating Internet-facing Microsoft Exchange servers to deploy webshells used for command execution. Once a foothold is established, the attackers conduct a variety of reconnaissance, credential theft, lateral movement, and data exfiltration activities.

The deployment of custom credential theft malware is central to this new campaign. The malware implemented a series of Mimikatz modifications on closed-source tooling. This post details the multi-component architecture and functionality of a sample, referred to as mim221.

We assess that mim221 is a recent version of an actively maintained credential theft capability upgraded with new anti-detection features. The use of special-purpose modules that implement a range of advanced techniques shows the threat actors' dedication to advancing its toolset towards maximum stealth. These techniques include

- in-memory mapping of malicious images to evade EDR API hooks and file-based detections
- surgically terminating Event Log threads instead of the host process to inhibit logging without raising suspicions
- staging a credential theft capability in the LSASS process itself by abusing native Windows capabilities.

Version numbers and build timestamps indicate a maintained software project by designated developers. Closer analysis reveals an element of pragmatism in that the threat actors use modified publicly available code to achieve their goals.

In terms of attribution, the tooling suggests an immediate link to the 'Operation Soft Cell' campaign but remains slightly vague on the specific threat actor. That campaign has been publicly associated with Gallium and possible

connections to APT41 have been suggested by the use of a common code signing certificate and tooling that shares code similarities. APT41 is also known to target telecommunication providers.

Given previous target and TTP overlaps, and an evident familiarity with victim environments, we assess with medium-confidence that Gallium is involved. However, we also recognize the possibility of closed-source tool-sharing between Chinese state-sponsored threat actors, and the possibility of a shared vendor or digital quartermaster.

Regardless of clustering specifics, this finding highlights the increased operational tempo of Chinese cyberespionage actors and their consistent investment in advancing their malware arsenal to evade detection.

Infection Vector and Initial TTPs

As initial attack indicators, we observed command execution through webshells on compromised Microsoft Exchange server deployments. The threat actors used `C:\MS_DATA` as their main working directory for storing malware and staging data for exfiltration. Noting that the Microsoft TroubleShootingScript toolset (TSSv2) uses `C:\MS_DATA` for storing log files, we suspect that its use as a working directory is an attempt to make malicious file system activities look legitimate.

After establishing an initial foothold, the threat actor conducts reconnaissance like querying user and network information using a variety of tools. For example, the attackers used dsquery and query to obtain information about Active Directory objects, including user information, and Remote Desktop user sessions. They also used the Local Group (LG) tool to enumerate all local groups and members in a domain.

```
   "cmd"  /c cd /d C:\MS_DATA\&dsquery * -limit 0 -filter
   "cmd"  /c cd /d C:\MS_DATA\&dsquery * -limit 0 -filter "&(objectClass=User)
(objectCategory=Person)" -attr objectSID sAMAccountName displayName  mail memberOf
>da.back&cd
   "cmd"  /c cd /d c:\windows\system32\inetsrv\&query user&cd
   "cmd"  /c cd /d C:\MS_DATA\&lg.exe \\[IP ADDRESS] -lu >169.txt&cd
```

The attackers then check connectivity with both the Internet and specific local machines of interest.

```
   "cmd"  /c cd /d c:\windows\system32\inetsrv\&ping 8.8.8.8 -n 1&cd
   "cmd"  /c cd /d c:\windows\system32\inetsrv\&ping -n 1 [IP ADDRESS/HOSTNAME]&cd
```

They also retrieve networking information, like network adapters, specific machines, and network services like Remote Desktop Protocol (RDP).

```
   "cmd"  /c cd /d C:\MS_DATA\&ipconfig /all&cd
   "cmd"  /c cd /d c:\windows\system32\inetsrv\&net use&cd
   "cmd"  /c cd /d c:\windows\system32\inetsrv\&netstat.exe -nob
   "cmd"  /c cd /d c:\windows\system32\inetsrv\&netstat -aon |find "3389"&cd
   "cmd"  /c cd /d C:\MS_DATA\&netstat -aon |find "[IP ADDRESS]"&cd
```

The threat actor made use of the native makecab tool to compress information gathered for exfiltration.

```
   "cmd"  /c cd /d C:\MS_DATA\&makecab da.back d.zip >1.txt&cd
```

For lateral movement, the attackers made use of the PsExec tool and the net use command for accessing shared resources on remote machines.

```
   "cmd"  /c cd /d C:\MS_DATA\&net use \\[IP ADDRESS] [PASSWORD] /u:[DOMAIN]\
[USERNAME]
```

A Penchant for Credential Theft

In order to steal credentials, the attackers employ custom modified versions of Mimikatz, including an executable named `pc.exe`.

```
default:
        PRINT_ERROR(L"Unknow Struct Key revision (%u)", pDomAccF->keys1.Revision);
```

```
text "UTF-16LE", 'ERROR kuhl_m_lsadump_getSamKey ; Unknow Struct Key '
text "UTF-16LE", 'revision (%u)',0
```

Mimikatz publicly available code (top); strings from a Mimikatz modification (bottom)

The `pc.exe` executable stages the execution of three other components that ultimately result in stealing credentials from the Local Security Authority Subsystem Service (LSASS) process.

We refer to the four component chain as ‘mim221’ based on the version number that the tool displays (2.2.1).

We observed the threat actors deploying individual chunks of `pc.exe` in the working directory and merging these into `pc.exe` using the `type` command.

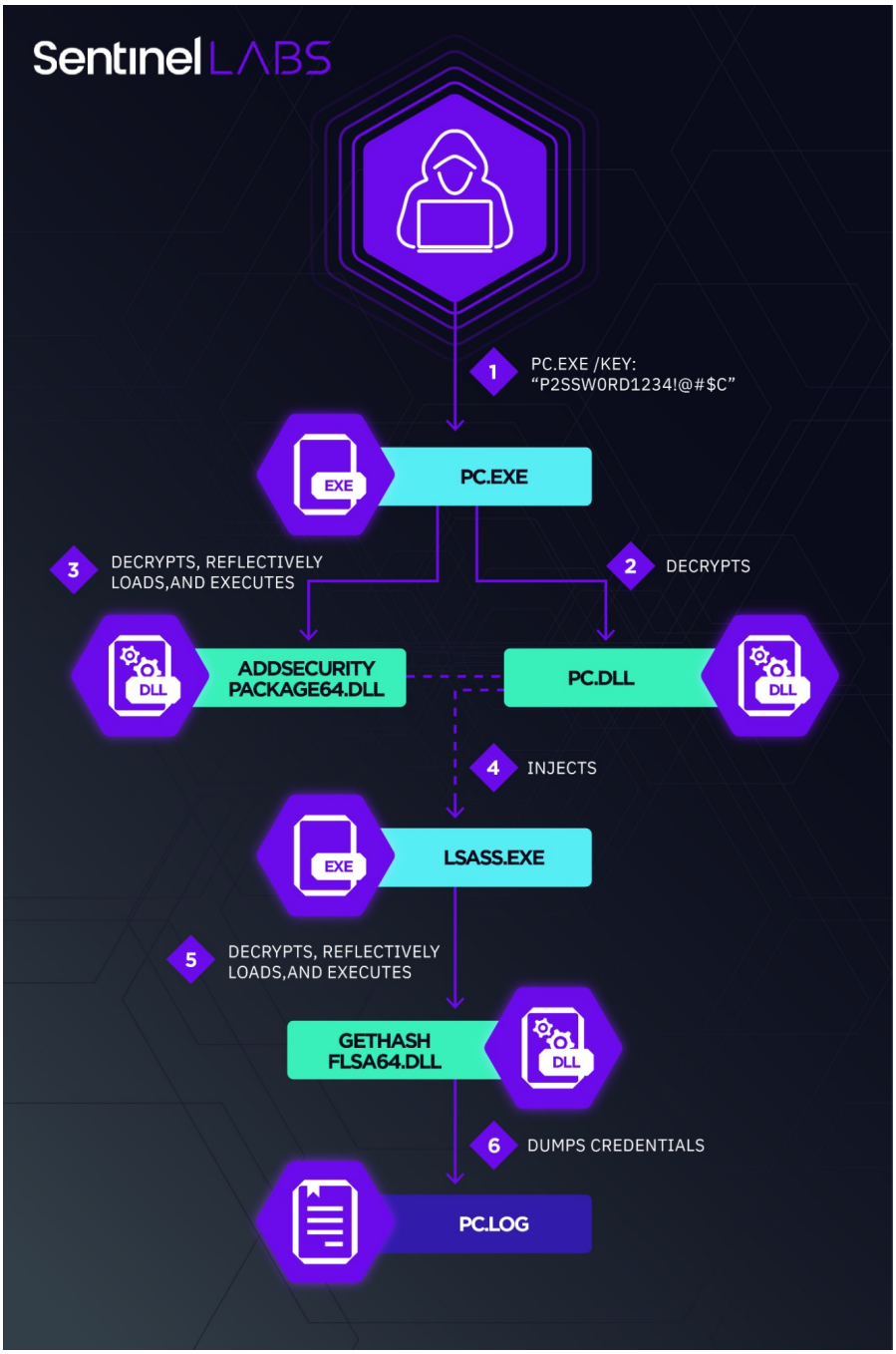
```
C:\MS_DATA\xaa
C:\MS_DATA\xab
C:\MS_DATA\xad
C:\MS_DATA\xac
C:\MS_DATA\xae
C:\MS_DATA\xag
C:\MS_DATA\xaf
C:\MS_DATA\xah
C:\MS_DATA\xaj
C:\MS_DATA\xai
C:\MS_DATA\xak

type x* >pc.exe
```

pc.exe file chunks

We noticed that the attackers ceased their activities after stealing credentials. This could indicate a multi-phase attack strategy, where the deployment of backdoors and further persistence mechanisms is carried out separately after credential theft has ensured continued access. The intrusions were detected and interrupted before the attackers could carry out further phases, such as deploying backdoors.

mim221

The architecture of mim221 consists of four components: the `pc.exe` Windows executable, and the `AddSecurityPackage64.dll`, `pc.dll`, and `getHashFlsa64.dll` DLLs contained therein.



mim221 execution overview

mim221 Component	Size	Compilation timestamp
pc.exe	502 KBs	Thu Jun 09 08:02:12 2022 (UTC)
AddSecurityPackage64.dll	119 KB	Thu Jun 09 08:01:46 2022 (UTC)
pc.dll	297 KB	Tue Jun 07 16:55:05 2022 (UTC)
getHashFlsa64.dll	216 KB	Fri May 27 20:56:26 2022 (UTC)

pc.exe

The main binary executed by the threat actor is `pc.exe`. It decrypts `AddSecurityPackage64.dll` and `pc.dll`, stores `pc.dll` on the file system, and then loads and executes `AddSecurityPackage64.dll` by invoking its exported function, `pathAddPackage`.

The execution of `pc.exe` requires a password supplied by the operator (in this case, `P2sSW0rd1234!@#$C`), which the operator provides through the `key` command-line parameter.

`pc.exe` decrypts `AddSecurityPackage64.dll` and `pc.dll` using the AES encryption algorithm, providing the operator-provided execution password as an initialization vector.

`pc.exe` loads and executes the decrypted `AddSecurityPackage64.dll`using reflective image loading. This technique involves first mapping a Windows PE image in memory and then executing the image’s main entry point or an export function.

Among other activities, the image mapping process includes allocating memory for the image, storing the image headers and sections in the memory, populating the images' import and delay import tables, adding exception handlers, and executing TLS callback and export routines. The Phant0m tool provides a complete implementation of this process.

While reflective image loading is a known technique at this time, its use was first observed in the DoublePulsar and subsequently the SlingShot frameworks in 2017 and 2018, respectively. This technique enables the fully fileless loading and execution of a malicious image without invoking the standard Windows API, such as `LoadLibrary`. This eliminates detection based on API hooking and file artifacts.

When it is finished executing, `pc.exe` displays a message indicating a version number and build timestamp: `Version 2.2.1  - build on Jun  9 2022 16:02:12`.

AddSecurityPackage64.dll

`AddSecurityPackage64.dll`, which is the original filename of this mim221 component, is responsible for:

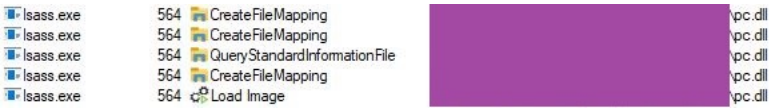
- Obtaining the `SeDebugPrivilege` and `SYSTEM` privilege by access token impersonation. This allows mim221 to inspect and extract credentials from the LSASS process.
- Disabling Windows event logging in an attempt to evade detection; and
- Injecting `pc.dll` into LSASS as a Security Package. Security Packages are used to extend the Windows authentication mechanism and can be abused to execute malicious code in the context of LSASS.

In an attempt to remain undetected, `AddSecurityPackage64.dll` disables Windows event logging by killing threads of the Windows `Event Log` service without stopping the execution of the service itself. This is achieved by locating the process that hosts the `Event Log`, enumerating the processes' threads, identifying the threads assigned to the service by their service tag (`eventlog`), and terminating them.

```
advapi32_lib = LoadLibraryW(L"advapi32.dll");
[...]
if ( advapi32_lib )
{
  I_QueryTagInformation = GetProcAddress(advapi32_lib, "I_QueryTagInformation");
  if ( I_QueryTagInformation )
  {
    [...]
    ((void (__fastcall *)(_QWORD, __int64, _QWORD *))I_QueryTagInformation)(0i64, 1i64, tagInfo);
    [...]
```

Querying service tag information

`AddSecurityPackage64.dll` injects `pc.dll` into LSASS by deploying `pc.dll` as a Security Package. To this end, `AddSecurityPackage64.dll` issues an RPC call to LSASS – to the `ncalrpc:[lsasspirpc]` RPC endpoint, providing the file path to `pc.dll` to LSASS. This call instructs LSASS to load and execute `pc.dll`, which then stages the `getHashFlsa64.dll` credential theft component.



getHashFlsa64.dll conducts credential theft in the context of LSASS

pc.dll and getHashFlsa64.dll

In the context LSASS, `pc.dll` decrypts, reflectively loads, and executes the code credential theft component `getHashFlsa64.dll` in a manner similar to `pc.exe`. `pc.dll` and `getHashFlsa64.dll` share the same original filename: `getHashFlsa64.dll`.

`pc.dll` is implemented such that its main routine returns **FALSE**, making LSASS execute `pc.dll` and then unload it. This is a detection evasion technique making LSASS load `pc.dll` while avoiding appearing as an added (registered) Security Package. LSASS normally creates registry entries when adding Security Packages and does not unload them once loaded. This provides an opportunity for defenders to detect the loading of malicious Security Packages. Previous research provides more detail on this topic.

`getHashFlsa64.dll` accesses the memory of its host LSASS process and stores stolen credentials in a Mimikatz log file named `pc.log` for later exfiltration.

```
Authentication Id : 0 ; 136556 (00000000:0002156c)
Session           : Interactive from 1
User Name         : [redacted]
Domain            : DESKTOP-[redacted]
Logon Server      : DESKTOP-[redacted]
Logon Time        : 09/03/2023 20:50:53
SID               : S-1-5-21-3555187505-637998078-3648899888-1001
    msv :
     [00000003] Primary
     * Username : [redacted]
     * Domain   : DESKTOP-[redacted]
     * NTLM     : 92f0d9bf382e71482b[redacted]
     * SHA1     : 9f6b862361b59812d3[redacted]
[...]
```

Example *pc.log* content

`getHashFlsa64.dll` exports a function named `GetMyVersion`, which displays a version number and build timestamp (`Version 2.2.0  - build on May 28 2022 04:56:23`), in a format consistent with the output from `pc.exe`. The credential theft functionality of `getHashFlsa64.dll` is implemented in its export function `GetLogonInfo`.

```
int GetMyVersion()
{
  [...]
  wcscpy(v1, L"Version 2.2.0  - build on %hs %hs");
  swprintf(Format, v1, "May 28 2022", "04:56:23");
  fwprintf(logfile_stream, Format);
  return fflush(logfile_stream);
}
```

The *GetMyVersion* function

Additional Information

Error Messages and Public Code Reuse

The mim221 components implement error logging. The error messages follow a consistent output format.

```
wcscpy(Format, L"[ERROR] Error in EnablePrivilege\n");

fwprintf(v19, L"CryptSetKeyParam MODE_ECB Fail.(0x%08X)\n", v29);

wcscpy(Format, L"[ERROR] not find RpcBindingFromStringBinding\n");
```

Example error messages

It is important to note that we observed code segments that seem to be modified versions of publicly available code. For example, the implementation of `AddSecurityPackage64.dll` looks like an adaptation of public code that demonstrates injection of a Security Package into LSASS using RPC calls.

```
printf("\nAddSecurityPackage Raw RPC Example... by @_xpn_\n\n");


if (argc != 2) {
        printf("Usage: %s PACKAGE_PATH\n");
        return 1;
}
```

```
if ( !path_to_dll )
{
  wcscpy(Format, L"Usage:  PACKAGE_PATH\n");
  wprintf(Format);
  return 1i64;
}
```

Similarity between publicly available code (top) and *AddSecurityPackage64.dll* (bottom)

Timestamp Information

The mim221 components that reflectively load other executables, `pc.exe` and `pc.dll`, patch beforehand a string in the loaded executable, which provides further timestamp information: `====A!B@C#0-2022-05-23 16:33:03S`. The patching involves replacing the string with configuration information, such as the mim221 execution password and a path to the log file for storing stolen credentials.

```
.data:000000001004A990                  text "UTF-16LE", '====A!B@C#0-2022-05-23 16:33:03S',0
.data:000000001004A9D2                  db    0
.data:000000001004A9D3                  db    0
[...]
-------
.data:000000001004A990 aP2ssw0rd1234C   db 'P2sSW0rd1234!@#$C',0
.data:000000001004A990                                  ; DATA XREF: DllMain+179↑o
.data:000000001004A990                                  ; DllMain+1C4↑r
.data:000000001004A9A2                  db ':',0
.data:000000001004A9A4 [...]:
.data:000000001004A9A4                  text "UTF-16LE", '\Users\[...]'
.data:000000001004A9A4                  text "UTF-16LE", '[...]\pc.log',0
[...]
```

Patched timestamp string

Attribution Analysis

We assess it is highly likely the initial attack phases we observed were conducted by Chinese threat actors with cyberespionage motivations. Telecommunication providers are frequent targets of espionage activity due to the sensitive data they hold. Our analysis identified indicators that point to the operation Soft Cell actors.

Operation Soft Cell has been associated with the Gallium group based on TTPs and some of the domains the group has been using.

Active since at least 2012, Gallium is likely a Chinese state-sponsored group that is targeting telecommunication, financial, and government entities in Southeast Asia, Europe, Africa, and the Middle East. While the group's original focus has been on telecommunication providers, recent reports suggest that Gallium has recently expanded targeting across other sectors.

The initial intrusion vector and the majority of the TTPs we observed closely match those conducted by, or associated with, the Soft Cell actors. This includes deploying webshells at Microsoft Exchange servers for establishing an initial foothold, following same file naming conventions, using the LG tool and the `net`, `query`, and `tasklist` Windows built-in tools for gathering user and process information, and the PsExec Windows Sysinternals tool and `net` for lateral movement and exploration, respectively.

It is worth noting that the attackers' activities at one of the targets suggested previous knowledge of the environment. We had observed activity at the same target a few months prior, which we attributed to Gallium primarily based on the use of the group's PingPull backdoor and TTPs.

By pivoting on the original filename of mim221's `getHashFlsa64.dll`, we observed another sample that steals credentials from LSASS. This sample has the PDB path of `e:\vs_proj\mimkTools\getHashFlsa\getHashFlsa\x64\release\getHashFlsa64.pdb` and has been first submitted to VirusTotal from Vietnam on January 04, 2023.

The path partially overlaps with the PDB path of a Mimikatz Soft Cell executable (`E:\vs_proj\simplify_modify\Win32\simplify.pdb`) and another Mimikatz executable of a Chinese threat actor thought to be part of the Soft Cell activity group arsenal (`E:\vs_proj\mimkTools\dcsync_new\x64\dcsync64.pdb`). This indicates that mim221 and these binaries may originate from the same source.

Closer analysis confirms that the sample we pivoted to is a previous, less-advanced version of mim221 – Version 2.2.0 – that does not include some mim221 components, such as `AddSecurityPackage64.dll` and `pc.dll`. We refer to this sample as mim220.

```
memset(Buffer, 0, 520);
wcscpy(Format, L"Version 2.2.0  - build on %hs %hs");
swprintf(Buffer, Format, "Apr 10 2021", "21:22:10");
wprintf(Buffer);
return 0i64;
```

```
memset(Buffer, 0, 520);
wcscpy(Format, L"Version 2.2.1  - build on %hs %hs");
swprintf(Buffer, Format, "Jun  9 2022", "16:02:12");
wprintf(Buffer);
return 0i64;
```

Output from mim220 (top) and mim221 (bottom)

Previous research indicates possible connections between the Soft Cell actors and APT41, which is known to conduct Chinese state-sponsored espionage activity as well as financially motivated activity targeting multiple sectors with a broad geographical coverage, including telecommunication providers.

The connection between the Soft Cell actors and APT41 that most relates to the activities that we observed is based on the Whizzimo, LLC certificate of the Soft Cell binary with a PDB path `E:\vs_proj\simplify_modify\Win32\simplify.pdb`, a binary that possibly originates from the same source as mim221. This certificate has been reported to be used by APT41. Pivoting on this certificate reveals further Mimikatz modifications, some with filenames very similar to those we observed.

Conclusions

Chinese cyberespionage threat actors are known to have a strategic interest in the Middle East. This is evident from their consistent targeted attacks on various entities including government, finance, entertainment, and telecommunication organizations. The recent activities targeting the telecommunication sector this post discusses are some of the latest such attacks.

Our analysis of mim221 highlights the continuous maintenance and further development of the Chinese espionage malware arsenal. These threat actors will almost certainly continue exploring and upgrading their tools with new techniques for evading detection, including integrating and modifying publicly available code.

SentinelLabs continues to monitor espionage activities and hopes that defenders will leverage the findings presented in this post to bolster their defenses.

Indicators of Compromise

SHA1	Note
f54a41145b732d47d4a2b0a1c6e811ddcba48558	pc.exe
1c405ba0dd99d9333173a8b44a98c6d029db8178	AddSecurityPackage64.dll (unpatched)
df4bd177b40dd66f3efb8d6ea39459648ffd5c0e	AddSecurityPackage64.dll (patched)
814f980877649bc67107d9e27e36fba677cad4e3	pc.dll
508408edda49359247edc7008762079c5ba725d9	getHashFIsa64.dll (unpatched)
97a7f1a36294e5525310f121e1b98e364a22e64d	getHashFIsa64.dll (patched)