BỘ GIÁO DỤC VÀ ĐÀO TẠO
TRƯỜNG ĐẠI HỌC NGOẠI NGỮ – TIN HỌC THÀNH PHỐ HỒ CHÍ MINH
KHOA CÔNG NGHỆ THÔNG TIN

Chuyên ngành: An ninh mạng

MÔN HỌC : PHÂN TÍCH MALWARE

Đề Tài: Phân tích LummaC2 Stealer

GIÁO VIÊN HƯỚNG DẪN: Th.S Phạm Đình Thắng

Thành viên nhóm: Nhóm 2

Nguyễn Lê Văn Quyền - 22DH113040
Huỳnh Gia Hòa - 22DH114543

TP.HCM, tháng 7/ 2025

Điểm phần trình bày – Điểm hệ 10

	CBCT1	CBCT2
Họ tên CBCT	…….. Chữ ký: …….....	…….. Chữ ký: …….....
Điểm	…….. Bằng chữ: ……...	…….. Bằng chữ: ……...
Nhận xét Báocáo: 2đ Ván đáp: 2đ Chức năng và demo: 5đ Mở rộng và ứng dụng thực tiễn: 1đ	Quyển báo cáo: (…) điểm…… Vấn đáp: (…) điểm…… Chức năng: (…) điểm…… Mở rộng: (…) điểm…… …… …… …… …… …… …… ……	Quyển báo cáo: (…) điểm…… Vấn đáp: (…) điểm…… Chức năng: (…) điểm…… Mở rộng: (…) điểm…… …… …… …… …… …… …… ……

Điểm quá trình – Điểm hệ 10

Họ tên CBCT: ……

Điểm tổng kết: ……(Bằng chữ:……...

LỜI CẢM ƠN

Trong bối cảnh chuyển đổi số đang diễn ra mạnh mẽ, bảo mật thông tin cá nhân và dữ liệu người dùng trở thành một trong những yếu tố then chốt để đảm bảo sự an toàn và tin cậy trong các hệ thống công nghệ thông tin. Các vụ tấn công mạng, rò rỉ dữ liệu và hành vi xâm phạm quyền riêng tư ngày càng diễn ra phổ biến, đòi hỏi mỗi cá nhân và tổ chức cần nâng cao nhận thức cũng như năng lực bảo vệ người dùng trước các mối đe dọa ngày càng tinh vi.

Môn học *Phân tích malware* nhằm trang bị cho sinh viên kiến thức nền tảng về các rủi ro bảo mật thường gặp, kỹ thuật tấn công trong môi trường mạng, cùng các biện pháp phòng chống hiệu quả. Thông qua các nội dung lý thuyết kết hợp với bài thực hành, sinh viên sẽ được tiếp cận các công cụ, phương pháp giám sát và tăng cường bảo vệ hệ thống và người dùng khỏi các nguy cơ đánh cắp thông tin, giả mạo danh tính và lừa đảo trực tuyến.

Môn học được giảng dạy bởi Thầy Th.S Phạm Đình Thắng, người có nhiều năm kinh nghiệm trong lĩnh vực an toàn thông tin, sẽ giúp sinh viên có cái nhìn thực tế và ứng dụng được kiến thức vào việc bảo vệ người dùng trong môi trường số hiện nay.

Hy vọng rằng với tinh thần học tập nghiêm túc, chủ động tìm tòi và thực hành, các bạn sinh viên sẽ tiếp thu hiệu quả và vận dụng tốt kiến thức đã học vào thực tiễn.

NHẬN XÉT CỦA GIẢNG VIÊN

.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....

MỤC LỤC

Contents

MỤC LỤC HÌNH ẢNH

MỤC LỤC BẢNG

I. GIỚI THIỆU ĐỀ TÀI

1.1 Lý do chọn đề tài

Trong bối cảnh công nghệ thông tin phát triển mạnh mẽ, các mối đe dọa an ninh mạng cũng ngày càng gia tăng về mức độ phức tạp và tinh vi. Trong số đó, **malware (mã độc)** là một trong những mối nguy hiểm phổ biến và nguy hiểm nhất. Malware không chỉ gây tổn thất về dữ liệu, mà còn ảnh hưởng đến tài chính, uy tín và an toàn của các cá nhân và tổ chức.

Vì vậy, việc **hiểu rõ cách malware hoạt động, cách phát hiện và phân tích** chúng là một kỹ năng quan trọng trong lĩnh vực an ninh mạng. Môn học **Phân tích Malware** trang bị cho sinh viên các kiến thức và kỹ năng cần thiết để tiếp cận, đánh giá và xử lý các phần mềm độc hại một cách bài bản và có hệ thống.

Với mục tiêu củng cố kiến thức đã học, em đã chọn đề tài:**"LummaC2 "** nhằm phân tích một mẫu malware thực tế, qua đó áp dụng các phương pháp **phân tích tĩnh và phân tích động** để hiểu sâu hơn về cách thức hoạt động và hành vi của malware.

1.2 Mục tiêu đề tài

- Phân tích một mẫu malware cụ thể từ góc nhìn kỹ thuật.
- Áp dụng các kỹ thuật phân tích tĩnh và phân tích động để trích xuất thông tin quan trọng như: hành vi, chỉ số IoC (Indicators of Compromise), kỹ thuật lẩn tránh,...
- Sử dụng các công cụ phân tích như: **dnSpy, Procmon, Wireshark, x64dbg, IDA,...**
- Đề xuất các biện pháp phòng tránh hoặc phát hiện tương tự trong thực tế.

II. CƠ SỞ LÝ THUYẾT

2.1 Malware

Malware (malicious software – phần mềm độc hại) là một chương trình máy tính được tạo ra với mục đích gây hại cho hệ thống, đánh cắp thông tin, chiếm quyền điều khiển, gián điệp, phá hoại hoặc thực hiện các hành vi phi pháp khác. Malware thường được sử dụng bởi tin tặc hoặc tổ chức tội phạm mạng để khai thác lỗ hổng, chiếm đoạt tài nguyên hoặc kiểm soát hệ thống mà không được sự cho phép của người dùng.

Malware có thể lây lan qua nhiều con đường khác nhau như email, USB, phần mềm crack, website độc hại hoặc thậm chí là qua các phần mềm hợp pháp đã bị chỉnh sửa. Hiểu rõ các loại malware là bước đầu tiên để nhận diện, phân tích và đưa ra biện pháp phòng chống hiệu quả.

2.1.1 Các loại Malware

Loại Malware	Mô tả	Hành vi điển hình
Virus	Tự nhân bản và chèn mã độc vào các file hoặc chương trình hợp lệ.	Lây lan qua file thực thi, phá hoại dữ liệu, thay đổi hệ thống.
Worm	Tự động sao chép và lan truyền qua mạng mà không cần file chủ.	Gây tắc nghẽn mạng, làm tràn băng thông, phát tán nhanh chóng.
Trojan (Ngựa thành Troy)	Ngụy trang dưới dạng phần mềm hợp pháp nhưng thực hiện hành vi độc hại.	Mở cửa hậu, cho phép truy cập từ xa, đánh cắp dữ liệu.
Ransomware	Mã hóa dữ liệu và yêu cầu tiền chuộc để giải mã.	Khóa toàn bộ file hệ thống, hiển thị yêu cầu thanh toán tiền chuộc.
Spyware	Theo dõi hoạt động của người dùng mà không có sự đồng ý.	Ghi log bàn phím, chụp màn hình, lấy thông tin cá nhân và mật khẩu.
Adware	Hiển thị quảng cáo không mong muốn, thường kèm theo phần mềm miễn phí.	Popup, chuyển hướng trình duyệt, thu thập dữ liệu người dùng.
Rootkit	Ẩn giấu sự hiện diện của malware khác và cung cấp quyền truy cập cao.	Né tránh công cụ bảo mật, kiểm soát sâu hệ thống.
Keylogger	Ghi lại toàn bộ phím bấm của người dùng.	Đánh cắp mật khẩu, số thẻ tín dụng, thông tin nhạy cảm.
Botnet	Tập hợp các thiết bị bị điều khiển từ xa như một mạng máy tính zombie.	Gửi spam, tấn công DDoS, đào tiền mã hóa.
Fileless Malware	Không lưu trữ mã độc dưới dạng file mà chạy trực tiếp trong bộ nhớ RAM.	Khó phát hiện, thường sử dụng PowerShell hoặc script chạy nền.

Bảng 1: Các loại malware

2.1.2 Tại sao phải phân tích Malware?

Phân tích malware là một hoạt động quan trọng trong lĩnh vực an ninh mạng nhằm hiểu rõ cách thức hoạt động, mục tiêu và kỹ thuật mà phần mềm độc hại sử dụng. Việc phân tích này mang lại nhiều lợi ích thiết thực như:

- **Phát hiện và phòng chống kịp thời**: Hiểu được hành vi và dấu hiệu của malware giúp các chuyên gia bảo mật phát hiện sớm các cuộc tấn công và thiết lập hệ thống cảnh báo tự động.
- **Truy vết nguồn gốc tấn công**: Phân tích kỹ lưỡng có thể giúp xác định cách thức phát tán, kỹ thuật khai thác và thậm chí cả nhóm tấn công đứng sau (APT, hacker,...).
- **Cải thiện hệ thống phòng thủ**: Từ thông tin phân tích được, các tổ chức có thể vá lỗ hổng, cập nhật chính sách firewall/antivirus và tăng cường các lớp bảo vệ hiệu quả hơn.

- **Phục vụ nghiên cứu và đào tạo**: Giúp các nhà nghiên cứu, sinh viên, kỹ sư bảo mật hiểu rõ hơn về kỹ thuật tấn công hiện đại, từ đó nâng cao năng lực phòng chống.
- **Phục hồi sau tấn công (Incident Response)**: Giúp xác định mức độ thiệt hại, các hệ thống bị ảnh hưởng, và hỗ trợ quá trình phục hồi, khôi phục dữ liệu, điều tra pháp lý (forensics).
- **Hạn chế thiệt hại lây lan**: Một malware nếu không được phát hiện kịp thời có thể lây lan sang hệ thống khác, đặc biệt là trong môi trường doanh nghiệp. Phân tích giúp cô lập và dập tắt sớm nguồn lây.

2.2 Các công cụ phân tích Malware

Trong quá trình phân tích malware, ta thường chia làm hai hướng chính:

- **Phân tích tĩnh (Static Analysis):** Xem mã thực thi mà **không cần chạy chương trình**, tập trung vào cấu trúc PE, luồng thực thi, string, API,...
- **Phân tích động (Dynamic Analysis):** Theo dõi **hành vi runtime** khi malware được chạy trong môi trường kiểm soát (debugger hoặc sandbox).

Mỗi hướng cần các công cụ đặc thù để trích xuất thông tin và hỗ trợ phân tích hiệu quả.

IDA – Phân tích tĩnh

IDA (Interactive Disassembler) là công cụ disassembler mạnh mẽ, hỗ trợ nhiều kiến trúc (x86, x64, ARM, v.v.). IDA cho phép dịch ngược mã máy (machine code) thành hợp ngữ, giúp người dùng:

- Tìm **Entry Point** của chương trình
- Phân tích luồng gọi hàm (Call Graph, Control Flow)
- Theo dõi các API quan trọng như WinExec, GetProcAddress, VirtualAlloc,…
- Truy ngược thuật toán mã hóa, unpack, hoặc loader
- Đặt breakpoint tĩnh, gán nhãn, comment mã cho dễ hiểu

=> IDA là công cụ **bắt buộc** trong phân tích mã độc dạng PE/EXE/DLL, đặc biệt khi cần phân tích thủ công không phụ thuộc sandbox.

x64dbg – Debugger cho phân tích động

x64dbg là một trình **gỡ lỗi mã nhị phân mã nguồn mở** cho Windows (hỗ trợ cả x86 và x64), thường dùng để:

- Đặt **breakpoint** tại EntryPoint, API, vùng pack
- Theo dõi dòng thực thi thật (runtime trace)
- Quan sát thanh ghi, stack, vùng bộ nhớ
- Dump vùng RAM chứa payload hoặc shellcode
- Bypass các kỹ thuật anti-debug đơn giản
- Kết hợp với **Scylla** để dump PE đã unpack

=> x64dbg cho phép bạn "nhìn thấy" chính xác những gì malware đang làm tại thời điểm thực thi, rất hiệu quả trong giai đoạn **unpacking** hoặc giải mã shellcode động.

Một số công cụ hỗ trợ khác

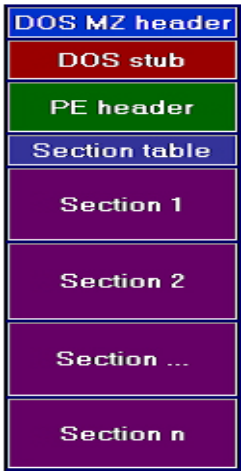
Công cụ	Loại	Chức năng chính
Detect It Easy (DIE)	Static	Nhận diện compiler, packer, entropy
PE-bear	Static	Xem chi tiết PE Header, Section Table, EntryPoint
Resource Hacker	Static	Xem/chỉnh sửa resource (icon, manifest, string table, dialog,...)
BinText / FLOSS	Static	Trích xuất chuỗi ASCII, Unicode, XOR hoặc động
ProcMon	Dynamic	Giám sát hành vi file, registry, process tại runtime
Wireshark	Dynamic	Bắt gói mạng, phân tích kết nối ra ngoài (C2, exfiltration)

Bảng 2: Một số công cụ hỗ trợ

2.3 Tệp PE

Tệp PE (Portable Executable) là định dạng file riêng của Win32 và cũng được mở rộng cho Win64. Các tệp thực thi được trên Win32 đều có định dạng PE, trừ các file VxDs (Virtual Device Drivers) và DLLs 16-bit. Kể cả các file như : Dlls 32 bit và 64 bit, COMs, khiển OCX, các chương trình nhỏ trong Control Panel, các chương trình điều khiểu ở Kernel mode và các ứng dụng đều có định dạng PE.

Cấu trúc cơ bản:



Hình 1: Các sections

The DOS Header

Các tệp PE đều bắt đầu bằng DOS Header, vùng này là 64 bytes đầu tiên của file. Nó có 19 thành phần, trong đó 2 thành phần lưu ý nhất là lfanew và magic.

Phần magic của DOS Header chứa giá trị 4Dh, 5Ah (là ký tự “MZ” viết tắt là Mark Zbikowsky một trong người sáng tạo của MS-DOS) nó cho ta biết đây là một file hợp lệ để nhận diện cùng một số thông tin khác.

Phần lfanew chỉ đến vị trí của PE Header trong file, giá trị của nó là một DWORD (Double Word = 4 bytes) và nằm ở vị trí cuối cùng của DOS Header nơi bắt đầu DOS Stub. Vì nó có vị trí PE Header nên , có liên quan đến phần đầu nên Windows Loader sẽ tìm kiếm offset này đi trực tiếp tới PE Header bỏ qua Dos Stub.

Dos Stub chỉ là một chương trình DOS EXE nhỏ hiện thông báo lỗi như "This Program cannot be run in DOS mode"

The PE Header

PE Header là thuật ngữ chung để chỉ cấu trúc IMAGE_NT_HEADERS trong tệp PE. Chứa các thông tin cần thiết về file, bao gồm kiến trúc, entry point và các sections trong file. Gồm 3 thành phần chính:

Signature là một DWORD chứa giá trị 50h, 45h, 00h, 00h (PE\0\0) để xác định đây là PE files. Đây là nơi bắt đầu của PE Header sau khi DOS Header kết thúc.

IMAGE_FILE_HEADER (FileHeader) gồm 20 bytes tiếp theo, nó chứa thông tin sơ đồ bố trí vật lý và những đặc tính của file như số lượng sections, nằm ở sau 6 bytes đầu của PE Header. Gồm các trường như trong hình.

```
IMAGE_FILE_HEADER STRUCT
  Machine                 WORD    ?
  NumberOfSections        WORD    ?
  TimeDateStamp           DWORD   ?
  PointerToSymbolTable    DWORD   ?
  NumberOfSymbols         DWORD   ?
  SizeOfOptionalHeader    WORD    ?
  Characteristics         WORD    ?
IMAGE_FILE_HEADER ENDS
```

Hình 2: PE Header

IMAGE_OPTIONAL_HEADER (Optional Header) Đây là phần không thể thiếu trong PE file, nó gồm các thông tin quan trọng như entry point, kích thước của stack hay heap, các Data Directories (như Import Table và Export Table). Các trường quan trọng như:

AddressOfEntryPoint: địa chỉ nơi chương trình bắt đầu thực thi

ImageBase: Địa chỉ ảo mà tệp PE mong muốn được nạp

SectionAlignment và File Alignment: Xác định cách các section được căn chỉnh trong bộ nhớ và trong file.

Data Directories: Chứa các phần liên quan đến dữ liệu quan trọng trong file như import table, export table,…

The Section Table

Section Table là một bảng quan trọng nằm ở phần tiếp theo sau PE Header kết thúc. Nó là một mảng những cấu trúc IMAGE_SECTION_HEADER, mỗi mục trong Section Table đại diện cho một section và chứa thông tin về thuộc tính của nó và offset ảo. Trong file PE có bao nhiêu sections thì đó là số lượng của mục Section Table. Có các trường chính như :

Name: Tên của section, có 8 bytes ký tự. Tên này có thể được bỏ trống

VirtualSize: Kích thước thật của section trong bộ nhớ tính theo bytes. Đây là kích thước của section khi đã được nạp vào bộ nhớ và có thể khác kích thước trên đĩa.

VirtualAddress: RVA của section, địa chỉ ảo mà section được nạp vào bộ nhớ. Địa chỉ này tương đối so với ImageBase.

SizeOfRawData: Kích thước thực tế của section trong PE file, nhỏ hơn hoặc bằng VirtualSize.

PointerToRawData (Raw Offset): Điểm bắt đầu của section trong file.

Characteristics: Bao gồm các flags xác định thuộc tính section như executable, readable, writable.

The PE File Sections

Trong PE file, đây là những sections chứa nội dung chính của file, bao gồm code, data, resources và những thông tin khác của file thực thi. Mỗi section có một Header và body (Đây là raw data, dữ liệu thô chưa được xử lý, nó chưa được xử lý hoặc chưa được biểu diễn lại dạng dễ truy tìm hay phân tích). Những Section Headers được chứa trong Section Table nhưng Section Bodies lại không có cấu trúc nhất định, nó có thể được sắp xếp theo bất kì cách nào với điều kiện là header được điền thông tin đầy đủ để giải mã dữ liệu. Một chương trình đặc thù trên Windows NT có 9 sections có tên là .text, .bss, .rdata, .rsrcm .edata, .idata, .pdata và .debug, một vài chương trình không cần có đủ các sections này, trong khi một số thì định nghĩa thêm nhiều sections để phù hợp với chúng.

Executable Code Section

Trong cấu trúc file PE, đây là section quan trọng nhất lưu trữ code của chương trình được tập trung vào một section đơn lẻ gọi là .text hoặc CODE. Đây cũng là nơi chứa điểm đột nhập (entry point).

Data Sections

Section .bss (Block Started by Symbol) chứa các biến chưa được khởi tạo, nó chứa các biến toàn cục chưa được gán giá trị cụ thể (Ví dụ : int y;). Khi chương trình chạy, các biến này sẽ được cấp phát trong không gian bộ nhớ và thường được gán mặc định là 0.

Section .rdata (Read-only Data) chứa dữ liệu chỉ đọc như hằng số, chuỗi và thông tin thư mục debug.

Section .data lưu trữ các biến toàn cục và dữ liệu đã được khởi tạo.

Resources Section

Section .rsrc chứa thông tin tài nguyên của chương trình. Cung cấp tài nguyên hợp lý cho giao diện đồ hoạ và các yếu tố hình ảnh của chương trình. 16 bytes đầu tiên gồm Header giống các sections khác nhưng dữ liệu của .rsrc được cấu trúc vào một resource tree, có thể dễ dàng quan sát thông qua chương trình resource editor.

Export Data Section

Section e.data chứa Export Directory Table cho một chương trình hoặc file Dll. Nó giúp chương trình khác biết đến các hàm hoặc biến trong Dll mà chúng có thể gọi hoặc truy cập.

Import Data Section

Section i.data chứa thông tin thư viện và hàm sẽ import khi thực thi. Section này cho phép gọi cá hàm từ các liên kết động đến thư viện mà không cần toàn bộ mã của hàm đó. Khi chương trình chạy, Windows sử dụng .idata để thêm những thư viện cần thiết và ánh xạ nó vào các hàm import để chương trình sử dụng.

Debug Information Section

Thông tin debug thường được đặt trong .debug. Chứa các thông tin gỡ lỗi của chương trình, tuy nó không ảnh hưởng đến việc thực thi chương trình nhưng rất hữu ích với các nhà phát triển và các công cụ phân tích. Section debug chưa thông tin debug nhưng những thư mục debug lại trong Section .rdata. Mỗi thư mục sẽ liên quan đến thông tin Debug trong Section .debug.

Base Relocations Section

Khi các trình linker tạo ra file Exe, nó sẽ có thông tin địa chỉ mong muốn nó được lưu. Nếu vì lý do gì đó khiến nó không nạp được ở nơi nó muốn thì nhưng địa chỉ nào sẽ bị trình linker đặt vào trong image sai. Thông tin được lưu trong .reloc giúp cho chương trình chạy đúng ở vị trí mới, cần phải có Base Relocations để điều chỉnh tất cả các địa chỉ tuyệt đối. Trường hợp nếu nó thuận lợi được nạp vào nơi nó muốn thì .reloc sẽ bị lờ đi.

III. GIỚI THIỆU MALWARE

LummaC2 Stealer (viết tắt là Lumma hoặc Lumma Stealer) là một loại mã độc đánh cắp thông tin (**info-stealer malware**) rất phổ biến trên Windows, được phát triển và phát hành dưới mô hình **Malware-as-a-Service (MaaS)** từ năm 2022.

Lịch sử và nguồn gốc

- **Xuất hiện lần đầu**: Tháng 8 năm 2022
- **Nguồn phát hành**: Các diễn đàn ngầm như Exploit, BreachForums, Telegram
- **Tác giả / Nhóm phát triển**: Được cho là do một nhóm lập trình viên Nga phát triển, đứng sau biệt danh **Lumma Author** hoặc **"lummadev"**
- Lumma được phát hành dưới dạng **gói thuê bao theo tháng (subscription)**, người mua có thể tùy biến cấu hình C2, cơ chế mã hóa, payload,...

Các phiên bản phát hành đáng chú ý

Phiên bản	Ngày phát hành	Tính năng nổi bật
v1.x	Cuối 2022	Stealer cơ bản (password, cookie, wallet)

Phiên bản	Ngày phát hành	Tính năng nổi bật
v2.x	Giữa 2023	Giao diện web C2 nâng cấp, ẩn mình tốt hơn
v3.x	Cuối 2023–2024	Tùy biến packing, anti-debug, loader mới
v4.0+	2024	Tích hợp thêm module anti-analysis phức tạp, cải tiến luồng shellcode

Bảng 3: Các phiên bản LummaC2

Tính năng chính của LummaC2

- **Đánh cắp thông tin trình duyệt**:
  - Cookie, mật khẩu, token từ Chrome, Edge, Brave, Firefox,...
- **Trích xuất thông tin hệ thống**:
  - User, hostname, địa chỉ IP, thông tin mạng
- **Thu thập dữ liệu ví tiền mã hóa (crypto wallet)**:
  - MetaMask, Exodus, Electrum, Coinomi...
- **Chiếm quyền truy cập các nền tảng**:
  - Discord token, Telegram session, Steam session,...
- **Tùy biến giao tiếp C2**: mã hóa, nén, định dạng zip hoặc base64
- **Tự xóa sau khi gửi dữ liệu về server**

Kỹ thuật ẩn mình & chống phân tích

- Sử dụng **mutex** và **sleep trick** để tránh bị sandbox phân tích
- **Packing** custom + nhiều lớp obfuscation
- Dùng **control flow flattening** qua biến trạng thái (v25, v51) để làm rối luồng chương trình
- **Anti-VM, Anti-Debug**: sử dụng IsDebuggerPresent, NtGlobalFlag, CPUID, GetCursorPos, QueryPerformanceCounter

Tình hình sử dụng thực tế

- Lumma được sử dụng trong **nhiều chiến dịch tấn công thực tế**, từ các chiến dịch spam email đến giả phần mềm crack/installer.
- Nhiều **công cụ khai thác sơ hở (loader)** phân phối Lumma như: PrivateLoader, SmokeLoader, NullMixer,...
- Các domain C2 liên tục thay đổi như: curtainjors.fun, trycsrv.net, lookmeup.fun.

Lý do chọn phân tích LummaC2

- Là **một trong những info-stealer đang hoạt động phổ biến nhất hiện nay**
- Sử dụng **packing và shellcode loader phức tạp**, phù hợp để thực hành phân tích thủ công
- Có hành vi rõ ràng, API dễ theo dõi qua **x64dbg và IDA**
- Tích hợp nhiều **kỹ thuật né tránh phân tích (anti-analysis)** hiện đại

IV. PHÂN TÍCH

4.1 Phân tích tĩnh

4.1.1 Thông tin cơ bản

Dùng PE-Bear để kiểm tra thông tin cơ bản của file PE

File info	
Path	C:/Users/quyen/Desktop/b14ddf64ace0b5f0d7452be28d07355c1c6865710dbed84938e2af...
Is Truncated?	No
File size	550400
Loaded size	550400
File Alignment Units	1075
ImpHash	1dad3e8a25605e3da6c89bdbab10b2a1
Rich Header Hash	not available
Checksum	8f4bc
MD5	1a6d38a9dfaf650971d6522f3f06ba12
SHA1	ea606f7b695c8826e291136423e8caa100dbca73
SHA256	b14ddf64ace0b5f0d7452be28d07355c1c6865710dbed84938e2af48ccaa46cf

Hình 3: File information

Cái thư viện động nó gọi

Disasm: .text | General | Strings | DOS Hdr | File Hdr | Optional Hdr | Section Hdrs | Imports | Resources | BaseReloc

Offset	Name	Func. Count	Bound?	OriginalFirstThunl	TimeDateStamp	Forwarder	NameRVA	FirstThun
E25C	KERNEL32.dll	69	FALSE	F8BC	0	0	FAFE	E010
E270	GDI32.dll	1	FALSE	F8B4	0	0	FB20	E008
E284	ADVAPI32.dll	1	FALSE	F8AC	0	0	FB3A	E000

Hình 4: Import dlls

Imports 71 hàm, 10 hàm là nguy hiểm do có khả năng ghi đè file, phân quyền vùng nhớ ảo, tác động tới Reg.

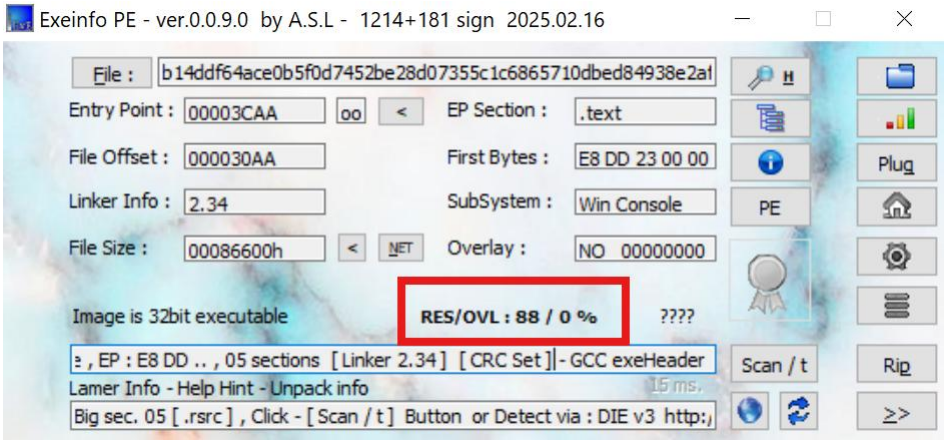
imports (71)	flag (10)	type	ordinal	first-thunk (IAT)	first-thunk-original (INT)	library
VirtualProtect	x	implicit	-	0x0000FA0E	0x0000FA0E	KERNEL32.dll
VirtualAlloc	x	implicit	-	0x0000FA42	0x0000FA42	KERNEL32.dll
FindFirstVolumeW	x	implicit	-	0x0000FADC	0x0000FADC	KERNEL32.dll
WriteFile	x	implicit	-	0x0000FB86	0x0000FB86	KERNEL32.dll
GetEnvironmentStrings	x	implicit	-	0x0000FBD2	0x0000FBD2	KERNEL32.dll
GetEnvironmentStringsW	x	implicit	-	0x0000FC1A	0x0000FC1A	KERNEL32.dll
GetCurrentThreadId	x	implicit	-	0x0000FCD8	0x0000FCD8	KERNEL32.dll
GetCurrentProcessId	x	implicit	-	0x0000FD58	0x0000FD58	KERNEL32.dll
GetCurrentProcess	x	implicit	-	0x0000FDD0	0x0000FDD0	KERNEL32.dll
RegDeleteKeyA	x	implicit	-	0x0000FB2A	0x0000FB2A	ADVAPI32.dll

Hình 5: Hàm nguy hiểm

Kiểm tra bằng Exeinfo PE

Trình Compiler GCC có thể được viết bằng C++

Win Console ứng dụng dòng lệnh.

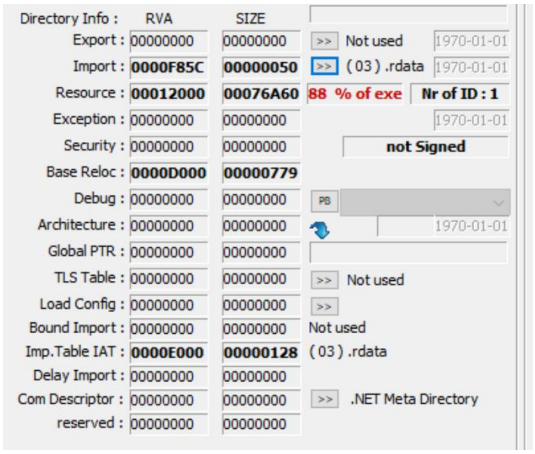


Hình 6: Exeinfo PE

Điều đặc biệt ở đây là **RES/OVL: 88 / 0%**

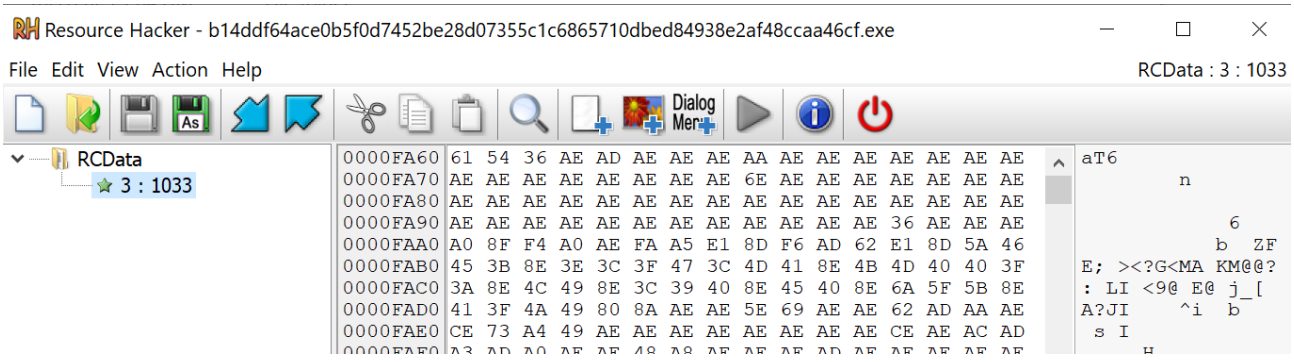
=> .rsrc chiếm 88% bên trong

Các Section



Hình 7: Các sections của LummaC2

Dùng Resource Hacker kiểm tra



Hình 8: Resource LummaC2

Có Resource nhúng nhưng đã bị Encrypt

=> Resource này khả năng là payload cao vì chiếm tới hơn 80% các section khác.

4.1.2 Phân tích trong IDA

```
; Section 2. (virtual address 0000D000)
; Virtual size                  : 00000779 (   1913.)
; Section size in file          : 00000800 (   2048.)
; Offset to raw data for section: 0000C200
; Flags 62300060: Text Data Discardable Executable Readable
; Alignment     : 4 bytes

; Segment type: Pure code
; Segment permissions: Read/Execute
_reloc segment dword public 'CODE' use32
assume cs:_reloc
;org 40D000h
assume es:nothing, ss:nothing, ds:_data, fs:nothing, gs:nothing


; Attributes: bp-based frame

sub_40D000 proc near

var_4= dword ptr -4

push    ebp
mov     ebp, esp
push    ecx
call    ds:FreeConsole
mov     [ebp+var_4], offset loc_402DE0
mov     eax, [ebp+var_4]
push    eax
retn
sub_40D000 endp ; sp-analysis failed
```

Hình 9: Entry Point

Hàm sub_40D000 là Entry Point mà IDA đưa ra.

Gọi hàm FreeConsole để ẩn Console, Anti Debug đơn giản.

Nó dùng kĩ thuật push – ret. Đẩy địa chỉ loc_402DE0 vào var_4

Sau đó đưa vào thanh ghi EAX rồi đẩy vào stack, chạy địa chỉ này.

Mã giã :

```
1 void sub_40D000()
2 {
3   FreeConsole();
4   JUMPOUT(0x402DE0);
5 }
```

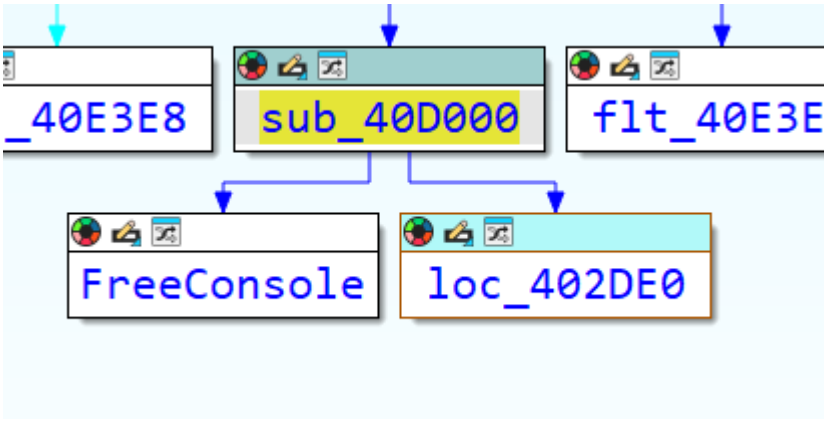
Hình 10: Mã giả Entry Point

```
.text:00402DE0 loc_402DE0:                             ; DATA XREF: sub_40D000+A↓o
.text:00402DE0                 push    ebp
.text:00402DE1                 mov     ebp, esp
.text:00402DE3                 sub     esp, 824h
.text:00402DE9                 push    ebx
.text:00402DEA                 push    esi
.text:00402DEB                 push    edi
.text:00402DEC                 push    0
.text:00402DEE                 lea     eax, [ebp-7E8h]
.text:00402DF4                 push    eax
.text:00402DF5                 call    ds:FindFirstVolumeW
.text:00402DFB                 jz      short near ptr loc_402DFF+3
.text:00402DFD                 jnz     short near ptr loc_402DFF+3
```

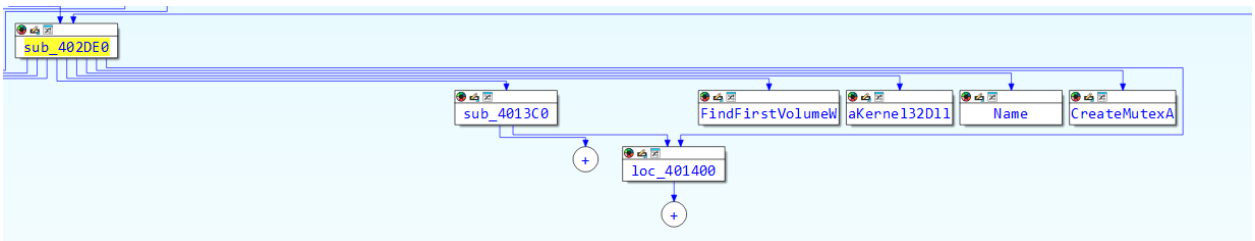
Hình 11: Hàm loc_402DE0

Vào tiếp loc_402DE0, nó gọi hàm FindFirstVolume. Sau đó nhảy tới hàm loc_402DFF bằng kĩ thuật jz + jnz để Anti-Disassembly, khiến trình Disassembly xem đoạn code đó trở về sau khó hiểu.

Proximity View:



Hình 12: Proximity View sub_40D000



Hình 13: Proximity View sub_402DE0

Tiếp tục phân tích mã giả ở sub_402DE0

```
do
{
  hModule = GetModuleHandleA("kernel32.dll");
  MutexA = CreateMutexA(0, 1, "5x8Rdw2cBe7kmj9DTSXmy8BPsXlTG5FUuDw1wqYTLZdr9HAnNDQ6WCDqXiAi%");
}
while ( GetLastError() != 183 );
 RAX = 183LL * (int)(v35 - 1);
```

Hình 14: Vòng lặp kiểm tra instance

Tạo Mutex với chuỗi:

"5x8Rdw2cBe7kmj9DTSXmy8BPsXlTG5FUuDw1wqYTLZdr9HAnNDQ6WCDqXiAi%"

Để tránh chạy nhiều phiên bản (**anti-multi-instance)** , ngoại lệ báo lỗi 183 (183 = ERROR_ALREADY_EXISTS)

Mã giả tiếp theo:

```
ProcName[0] = 18;
ProcName[1] = -57;
ProcName[2] = -91;
ProcName[3] = -97;
ProcName[4] = -99;
ProcName[5] = -16;
ProcName[6] = -59;
ProcName[7] = 56;
ProcName[8] = -79;
ProcName[9] = -65;
ProcName[10] = -83;
ProcName[11] = -24;
ProcName[12] = -10;
ProcName[13] = -77;
ProcName[14] = -122;
for ( j = 0; j < 0xF; ++j )
{
  v53 = ~ProcName[j] - j - 104;
  v54 = -((8 * v53) | ((int)v53 >> 5));
  v55 = ~(33 - ((32 * v54) | ((int)v54 >> 3)) + 79);
  v56 = j + ((v55 << 7) | ((int)v55 >> 1));
  v57 = j + j - ((v56 << 7) | ((int)v56 >> 1));
  v58 = j + ((4 * v57) | ((int)v57 >> 6));
  ProcName[j] = -(char)(j + ~((v58 << 6) | ((int)v58 >> 2))) - 97;
}
ProcAddress = GetProcAddress(hModule, ProcName);
```

Hình 15: Che giấu gọi API

Hàm GetProAddress(), nó dùng biến ProcName[16] để thực thi. ProcName[] gộp nhiều dữ liệu lại để thành ra tên thật, tiến hành giải mã để lấy lại tên thực.

```
decryt.py > ...
1   data = [18, -57, -91, -97, -99, -16, -59, 56, -79, -65, -83, -24, -10, -77, -122]
2
3   def decrypt(data):
4       result = []
5       for j in range(len(data)):
6           byte = data[j] & 0xFF
7           v53 = (~byte & 0xFF) - j - 104
8           v53 &= 0xFF
9
10          v54 = -((8 * v53) | (v53 >> 5)) & 0xFF
11          v55 = ~(33 - ((32 * v54) | (v54 >> 3)) + 79) & 0xFF
12          v56 = (j + ((v55 << 7) | (v55 >> 1))) & 0xFF
13          v57 = (j + j - ((v56 << 7) | (v56 >> 1))) & 0xFF
14          v58 = (j + ((4 * v57) | (v57 >> 6))) & 0xFF
15
16          c = -(j + ~((v58 << 6) | (v58 >> 2))) - 97
17          result.append(chr(c & 0xFF))
18      return ''.join(result)
19
20  print(decrypt(data))
21

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Users\2ien\Desktop> & C:/Users/2ien/AppData/Local/Programs/Python/Python313/python.exe
VirtualProtect
PS C:\Users\2ien\Desktop>
```

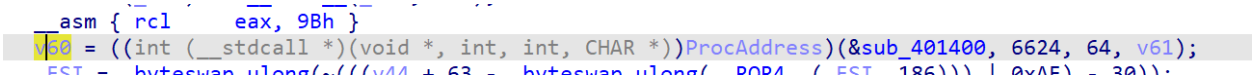
Hình 16: Thuật toán giải mã tên hàm

Malware ẩn hàm VirtualProtect từ kernel32.dll để thực hiện hành vi trái phép mà không bị phát hiện bằng cách cho chạy ở runtime, thay đổi quyền truy cập (memory protection).

Tác dụng chính	Mô tả
Đổi quyền truy cập bộ nhớ	Có thể đổi từ READONLY sang READWRITE, hoặc từ NOACCESS sang EXECUTE_READWRITE
Cho phép ghi mã vào vùng code	Malware thường dùng để ghi **shellcode** (mã độc) vào bộ nhớ
Cho phép thực thi code đã ghi	Sau khi ghi shellcode vào vùng nhớ, VirtualProtect dùng để chuyển vùng đó sang trạng thái có thể thực thi (PAGE_EXECUTE_READWRITE)

Bảng 4: Chức năng hàm VirualProtect

Ở biến v60, gọi VirtualProtect để sửa quyền vùng nó muốn chạy

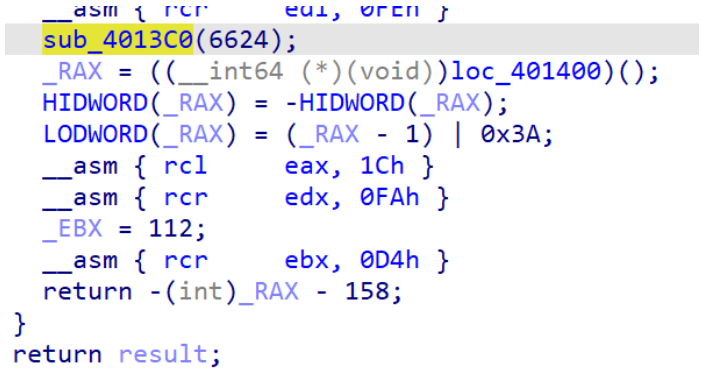


Hình 17: Gọi hàm VirtualProtect sửa quyền vùng nhúng payload

```
((int (__stdcall *)(void *, int, int, CHAR *))ProcAddress)(&sub_401400, 6624, 64, v61);
VirtualProtect(
  &sub_401400,      // địa chỉ vùng nhớ cần cấp quyền thực thi
  6624,             // size
  64,               // PAGE_EXECUTE_READWRITE
  &oldProtect);     // lưu quyền cũ vào biến `v61`
```

=> Cho phép shellcode hoặc PE thứ 2 được thực thi trong RAM

=> &sub_401400 size 6624 có thể là nơi chứa mã độc vì được cấp quyền



Hình 18: Gọi hàm sub_4013C0

Tiếp tục hàm sub_4013C0(6624)

```
1 int __cdecl sub_4013C0(int a1)
2 {
3   return sub_401200(&loc_401400, a1);
4 }
```

Hình 19: Hàm sub_4013C0

Trả về là sub_401200(tham số địa chỉ loc_401400, size), size = 6624

Vào bên trong sub_401200, Ta thấy được thuật toán giải mã địa chỉ loc_401400

```
do
{
  *a4 = ~*a4;
  *a4 += 120;
  *a4 += 73;
  result = 0;
  ++a4;
  --a5;
}
while ( a5 );
return result;
}
```

Hình 20: Thuật toán giải mã loc_401400

a4 = &loc_401400

a5 = size

Nó lấy giá trị bên trong hàm loc_401400. Đổi hexcode tính trong IDA từ 401400 đến 6624 byte là 402DD0.

==> Hàm này **giải mã payload** hoặc **shellcode** đã bị obfuscate.

→ Nghĩa là **hàm sub_4013C0 gọi sub_401200** để bắt đầu quá trình **giải mã payload thứ 2**.

Hàm loc_401400 khi bị obfuscate

```
.text:00401400 loc_401400:                             ; CODE XREF: .text:00403664↓p
.text:00401400                                         ; DATA XREF: sub_4013C0+7↑o ...
.text:00401400                 imul    esi, ds:7D5408D4h, -4Fh
.text:00401407                 rcr     al, 4Dh
.text:0040140A                 call    far ptr 4C69h:6A6DC0C0h
.text:00401411                 mov     ebx, 0FD08BD4Bh
.text:00401416                 xor     [eax], cl
.text:00401418                 fild    qword ptr [ebp-3E873F80h]
.text:0040141E                 lock jnz short near ptr word_401426
.text:00401421                 jno     short loc_4013E3
.text:00401423                 rol     al, 7Ah
```

Hình 21: loc_401400 bị obfuscate

```
.text:00401400 loc_401400:                             ; CODE XREF: .text:00403664↓p
.text:00401400                                         ; DATA XREF: sub_4013C0+7↑o ...
.text:00401400                 push    ebp
.text:00401401                 mov     ebp, esp
.text:00401403                 mov     eax, 0F436Ch
.text:00401408                 call    __alloca_probe
.text:0040140D                 push    ebx
.text:0040140E                 push    esi
.text:0040140F                 push    edi
.text:00401410                 jz      short near ptr loc_401414+3
.text:00401412                 jnz     short near ptr loc_401414+3
```

Hình 22: loc_401400 deobfuscate

Hàm sau khi đã patch

Kiểm tra thấy được :

```
.text:0040164C                 call    ds:GetModuleHandleW
.text:00401652                 mov     [ebp-0F42FCh], eax
.text:00401658                 push    0Ah
.text:0040165A                 push    3
.text:0040165C                 mov     eax, [ebp-0F42FCh]
.text:00401662                 push    eax
.text:00401663                 call    ds:FindResourceW
.text:00401669                 mov     [ebp-0F42E8h], eax
.text:0040166F                 mov     ecx, [ebp-0F42E8h]
.text:00401675                 push    ecx
.text:00401676                 mov     edx, [ebp-0F42FCh]
.text:0040167C                 push    edx
.text:0040167D                 call    ds:SizeofResource
.text:00401683                 mov     [ebp-0F42F0h], eax
.text:00401689                 mov     eax, [ebp-0F42E8h]
.text:0040168F                 push    eax
.text:00401690                 mov     ecx, [ebp-0F42FCh]
.text:00401696                 push    ecx
.text:00401697                 call    ds:LoadResource
.text:0040169D                 mov     [ebp-0F4304h], eax
.text:004016A3                 mov     edx, [ebp-0F4304h]
.text:004016A9                 push    edx
.text:004016AA                 call    ds:LockResource
.text:004016B0                 mov     [ebp-0F42E4h], eax
.text:004016B6                 mov     dword ptr [ebp-0F4320h], 0
.text:004016C0                 jmp     short loc_4016D1
```

Hình 23: Hàm gọi Resource

Gọi những API Resource, Malware là resource nhúng.

Quy trình chuẩn để truy cập Resource :

```
HRSRC hRes = FindResourceW(hModule, L"3", RT_RCDATA);
HGLOBAL hResLoad = LoadResource(hModule, hRes);
LPVOID lpResLock = LockResource(hResLoad);
DWORD dwSize = SizeofResource(hModule, hRes);
```

API	Chức năng
FindResourceW	Tìm resource theo tên/ID và loại (RT_RCDATA, RT_ICON, v.v.)
LoadResource	Nạp resource vào bộ nhớ (trả về handle HGLOBAL)
LockResource	Trả về con trỏ thực tế tới vùng dữ liệu resource
SizeofResource	Lấy kích thước của resource (để xử lý đúng byte)

Bảng 5: API Gọi Resource

```
.text:00401658                  push    0Ah
.text:0040165A                  push    3
.text:0040165C                  mov     eax, [ebp-0F42FCh]
.text:00401662                  push    eax
.text:00401663                  call    ds:FindResourceW
```

Hình 24: FindResourceW

Gọi FindResourceW

Type = 0A (RT_RCDATA)

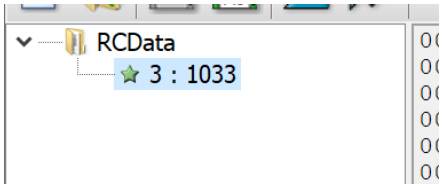
Resoure id = 3

Sau khi nó lấy đủ các biến ở hàm thì vào sub_401000 để giải mã resource

```
do
{
  *a3 = ~*a3;
  *a3 += 71;
  *a3 += 104;
  result = 0;
  ++a3;
  --a4;
}
while ( a4 );
```

Hình 25: Thuật toán trong sub_401000

Dùng Resource Hacker lấy resource nó ra rồi giải mã



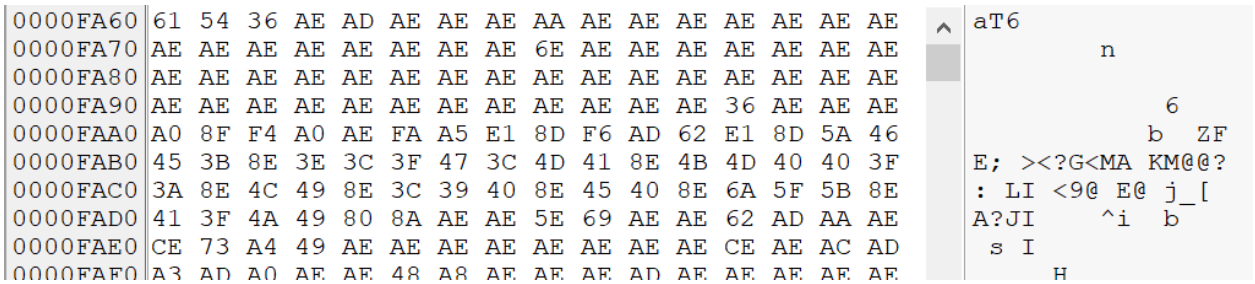
Hình 26: Resouce trong Lumma

Script decrypt

```python
def decrypt(data: bytes) -> bytes:
    decrypted = bytearray()
    for b in data:
        decoded = ~(b - 0xAF) & 0xFF
        decrypted.append(decoded)
    return decrypted

with open("RCData3.bin", "rb") as f:
    encrypted = f.read()

decrypted = decrypt(encrypted)

with open("resource_decrypted.bin", "wb") as f:
    f.write(decrypted)
```

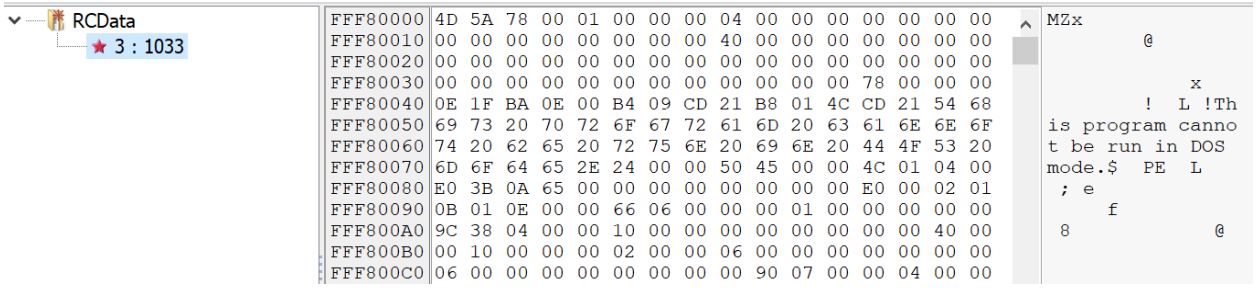
Hình 27: Thuật toán giải mã resourcec

Trước decrypt



Hình 28: Resource encrypt

Sau decrypt



Hình 29: Resource decrypt

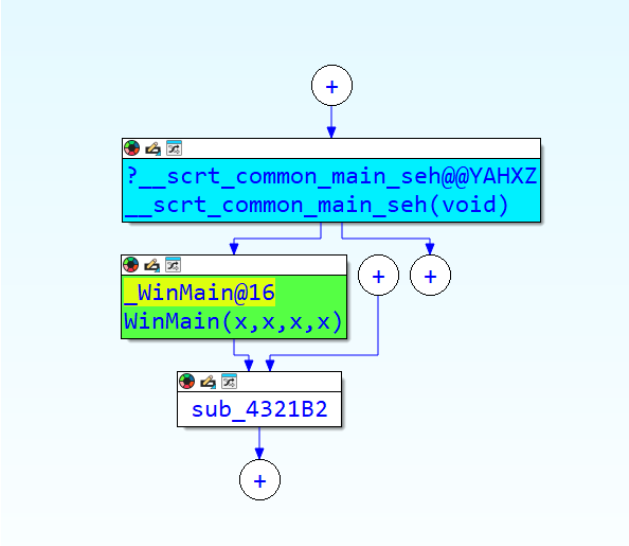
=> Bên trong Resource là file PE, đây là payload thật sự.

Các thư viện được gọi:

Offset	Name	Func. Count	Bound?	OriginalFirstThunl	TimeDateStamp	Forwarder	NameRVA	FirstThunk
70EE0	KERNEL32.dll	98	FALSE	725A8	0	0	733E0	727E4
70EF4	USER32.dll	8	FALSE	72734	0	0	733ED	72970
70F08	ADVAPI32.dll	5	FALSE	72758	0	0	733F8	72994
70F1C	GDI32.dll	9	FALSE	72770	0	0	73405	729AC
70F30	SHLWAPI.dll	1	FALSE	72798	0	0	7340F	729D4
70F44	WINHTTP.dll	9	FALSE	727A0	0	0	7341B	729DC
70F58	IPHLPAPI.DLL	1	FALSE	727C8	0	0	73427	72A04
70F6C	WININET.dll	2	FALSE	727D0	0	0	73434	72A0C
70F80	CRYPT32.dll	1	FALSE	727DC	0	0	73440	72A18

Hình 30: Thư viện payload gọi

Flow Payload



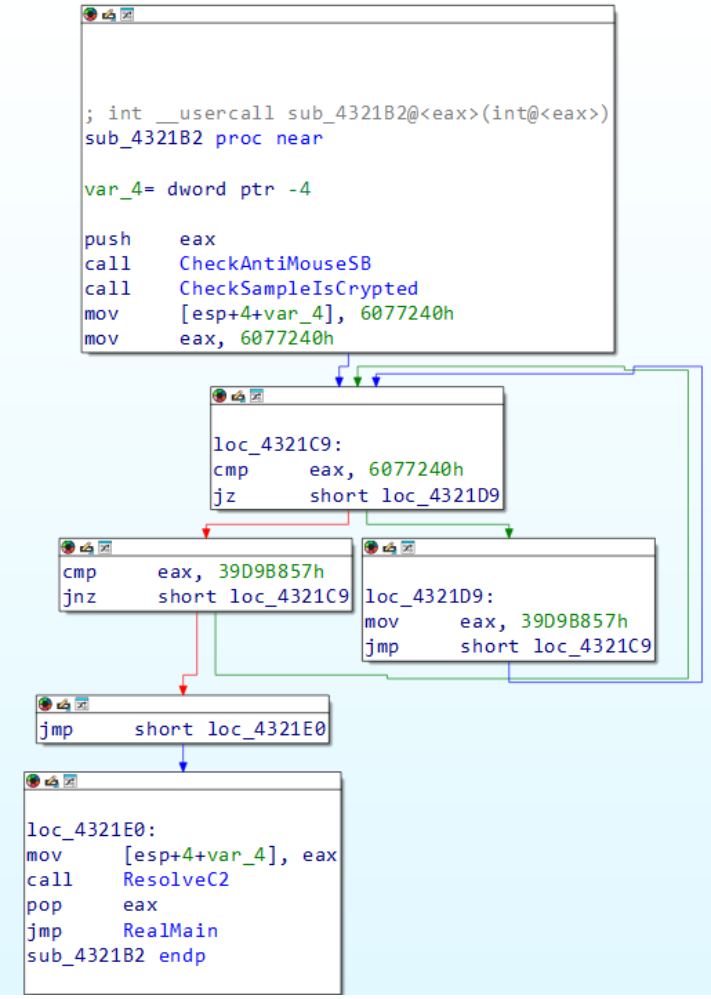
Hình 31: Proximity View Payload

Sub_4321B2 là hàm chính mà nó gọi.

```
; int __stdcall WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpCmdLine, int nShowCmd)
_WinMain@16 proc near

hInstance= dword ptr  4
hPrevInstance= dword ptr  8
lpCmdLine= dword ptr  0Ch
nShowCmd= dword ptr  10h

call    sub_4321B2
xor     eax, eax
retn    10h
_WinMain@16 endp
```

Hình 32: Sub_4321B2

WorkFlow Malware



Hình 33: WorkFlow Main

Nó có hai hàm Anti trước khi vào các bước chính.

Hàm Calculator_Angle

```
    {
      v31 = v47 + 1;
      state = -2080608904;
    }
    else if ( state == -639667699 )
    {
      Angle = Calculator_Angle(oldCursorPos, currentCursorPos);// Tinh Angle
      state = -1784748870;
    }
    else
    {
      state = -1605264192;
    }
  }
  else if ( state > -445357630 )
  {
```

Hình 34: Gọi hàm Calculator_Angle

double __cdecl Calculator_Angle(int *Pos1, int *Pos2);

- Pos1: con trỏ trỏ đến tọa độ vector thứ nhất (x1, y1)
- Pos2: con trỏ trỏ đến tọa độ vector thứ hai (x2, y2)

Mục đích là để tính toán và trả về góc (Angle)

tính toán dựa trên công thức toán học.

Đầu tiên nó sẽ tính **TichVoHuong** là tích vô hướng giữa hai vector A và B:

```
{
  TichVoHuong = (double)(y1y2 + x1x2);
  x1_sq = x1 * x1;                      // x^2 ( dung de xai sau )
  i = 720375612;
}
```

Hình 35: Tính tích vô hướng

Tiếp theo là **len1** độ dài vector A, được tính từ TongBinhPhuong:

```
else if ( i == 1096958478 )
{
  TongBinhPhuongVector1 = x1_sq + y1 * y1;
  i = -1621766071;
}
```

Hình 36: Tổng bình phương

```
else
{
  v9 = (double)TongBinhPhuongVector1;// Copy bien TongBinhPhuongVector1
  v10 = sqrt(v9);                    // Can bac 2 de lay do dai vector1
  if ( v12 )
    v10 = sqrt(v9);
  len1 = v10;                        // Luu do dai
  x2_sq = *Pos2 * *Pos2;
  i = 1951384782;
}
}
```

Hình 37: len1

Kế tiếp là **len2** độ dài vector B, được tính từ X:

```
}
X = (double)(x2_sq + *y2 * *y2);    // x^2 + y^2 (Vector2)
i = 466405770;
}
{
  v4 = sqrt(X);                     // Can bac hai
  if ( v6 )
    v4 = sqrt(X);
  len2 = v4;                        // Do dai Vector2
  i = -201145649;
}
}
```

Hình 38: len2

Cosin là giá trị cosine của góc:

```
{
  Cosin = TichVoHuong / (len1 * len2);// Cosin cua goc 2 vector
  i = 1751782807;
}
```

Hình 39: Cosin

AngleCosin là góc giữa hai vector (đơn vị radian):

```
{
  AngleCosin = acos(Cosin);
  i = 1299569257;
}
```

Hình 40: Góc giữa hai vector

Giá trị trả về cuối cùng của hàm là góc theo độ:

```
  }
  return AngleCosin * 180.0 / 3.141592653589793;// tra gia tri goc theo do
}
```

Hình 41: AngleCosin

Mục đích của hàm

Hàm Calculate_Vector có nhiệm vụ tính toán vector hiệu giữa hai điểm trong không gian tọa độ 2 chiều, được truyền vào qua con trỏ Pos1 và Pos2. Kết quả được lưu vào con trỏ Vector.

Cụ thể:

$$\text{Vector}_x = \text{Pos2}[0] - \text{Pos1}[0] = x_2 - x_1$$

$$\text{Vector}_y = \text{Pos2}[1] - \text{Pos1}[1] = y_2 - y_1$$

Hình 42: Công thức tính Pos

Hàm AntiSandBoxMouse

Bước 1 – Kiểm tra chuột có di chuyển hay không

Mô tả	State
Khởi tạo biến state	2072157566
Gọi GetCursorPos(&cursorPosBefore) để lấy vị trí chuột ban đầu	526711574
Tạm dừng 300ms để chờ chuột có thể di chuyển	1115581339
Gọi lại GetCursorPos(&cursorPosAfter) để lấy vị trí mới	—

Mô tả	State
Nếu chuột đã di chuyển (x ≠ cursorPosAfter.x) → chuyển sang bước tiếp theo	-2079276511 → -830812616 → -1172704950
Kiểm tra chỉ số vị trí (v31 == 4) để xác nhận điều kiện ban đầu	463218770

Bảng 6: Kiểm tra vị trí chuột

```
if ( state == 526711574 )
{
  v21 = (junkState2 + 27) * (junkState2 + 27);
  junkState2 = (143 * junkState2 + 10243) >> 6;
  v22 = -196161614;
  if ( v21 != 26043 * v21 - 1 )
    v22 = -2074784303;
  state = v22;
  v39 = v42;
  v38 = v31;
  v37 = v48;
}
else
{
  GetCursorPos(&cursorPosBefore);// Lay vi tri khoi dau
  state = -799310047;
}
```

Hình 43: GetCursorPos

Bước 2 – Ghi lại 5 vị trí chuột liên tục (P0 → P4)

Mô tả	State
Tạm dừng 50ms trước mỗi lần ghi	state ∈ (-2030460371, -1936032714) → vào nhánh else → Sleep(0x32u)
Ghi vị trí chuột vào cursorLog[2 * v45]	1380941766
Tăng biến v45, chuẩn bị ghi điểm tiếp theo	-946801794 → 1830671050
Lặp lại đến khi ghi đủ 5 điểm (kiểm tra v41 == 5)	-1936032714 → -1070947362

Bảng 7: Ghi lại vị trí chuột liên tục

```
  break;
if ( state > -2030460371 )
{
  if ( state <= -1936032715 )
  {
    v7 = 420 * junkState1;
    goto LABEL_163;
  }
  if ( state == -1936032714 )
  {
    v16 = -1070947362;
    if ( v41 != 5 )
      v16 = -1010690354;
    state = v16;
    v40 = v41;
  }
  else
  {
    Sleep(0x32u);                    // sleep 50ms de lay P0 - P4
    state = 1380941766;              // state de qua cho lay vi tri chuot
  }
}
```

Hình 44: Sleep

```
if ( state == 1380941766 )
{
  GetCursorPos((LPPOINT)&cursorLog[2 * v45]);// Luu vi tri chuot tu P0 - P4
  state = -946801794;
}
```

```
if ( state == -946801794 )
{
  v41 = v45 + 1;                // Tang bien dem len 1
                                // max la 5 lan
  state = 1830671050;
}
```

Hình 45: Lưu vị trí chuột theo con trỏ

Bước 3 – Tính vector chuyển động giữa các điểm

Mô tả	State
Tính vector từ P0 đến P1	-1605264192
Tính vector từ P1 đến P2 (lặp lại quy trình)	-1605264192
Tính vector từ P2 đến P3	-1605264192
Tính vector từ P3 đến P4	-1605264192

Bảng 8: State tính Vector

```
else if ( state == -1605264192 )
{
  v43 = v32 + 1;
  v44 = &cursorLog[2 * v32 + 2];
  Calculate_Vector(&cursorLog[2 * v32], v44, oldCursorPos);// Tinh Vector P0 - P4
  v56 = v32 + 2;
  state = -941373361;
}
else
```

Hình 46: Calculate_Vector

Bước 4 – Tính góc giữa các vector

Mô tả	State
Gọi hàm Calculator_Angle() để tính góc	-639667699
Lưu kết quả góc vào biến Angle	—

Bảng 9: Tính góc Vector

```
else if ( state == -639667699 )
{
  Angle = Calculator_Angle(oldCursorPos, currentCursorPos);// Tinh Goc
  state = -1784748870;
}
else
```

Hình 47: Calculator_Angle

Bước 5 – So sánh góc với ngưỡng hành vi người dùng

Mô tả	State
So sánh giá trị Angle > 45.0 (góc ngưỡng mặc định)	-1282056325
Gán giá trị isCursorMoved = true hoặc false	—
Nếu isCursorMoved == true → vượt qua kiểm tra	1115581340 → -1118832980
Nếu false → quay lại từ đầu (bắt đầu lại từ GetCursorPos ban đầu)	-448511932

Bảng 10: State so sánh hành vi con người

```
if ( state != -1282056325 )
  return result;
isCursorMoved = Angle > AngleToCheck;       // Check Goc ( True/False )
state = 1115581340;
}
```

Hình 48: Kiểm tra xem chuột có di chuyển

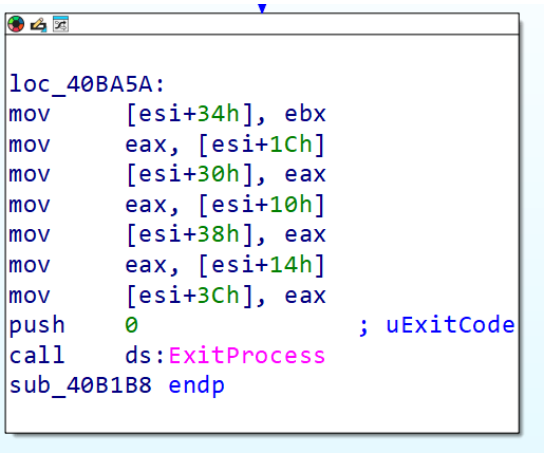
```
if ( state != 1115581340 )
{
  if ( state == 1145771008 )
  {
    v11 = (65 * junkState2 + 104) * (65 * junkState2 + 104);
    junkState2 = 65 * junkState2 - 147;
    v5 = 1105295811;
    if ( v11 != 10325 * v11 - 1 )
      v5 = 1661683787;
  }
  else
  {
    junkState1 = 173 * ((((junkState1 >> 2) & 0x7FFFFFF) + 257) >> 13);
    v5 = 1194212544;
    if ( junkState1 >= 0x97 )
      v5 = -1307007686;
  }
  goto LABEL_162;
}
v10 = -1118832980;            // State dung thi break vi nho hon -262831088
if ( !isCursorMoved )
  v10 = -448511932;           // State sai thi tiep tuc
state = v10;
v35 = v43;
v33 = 0;
}
```

Hình 49: Lưu kết quả vào biến

Đúng thì lưu vào v10, sai thì lưu v35 = 0.

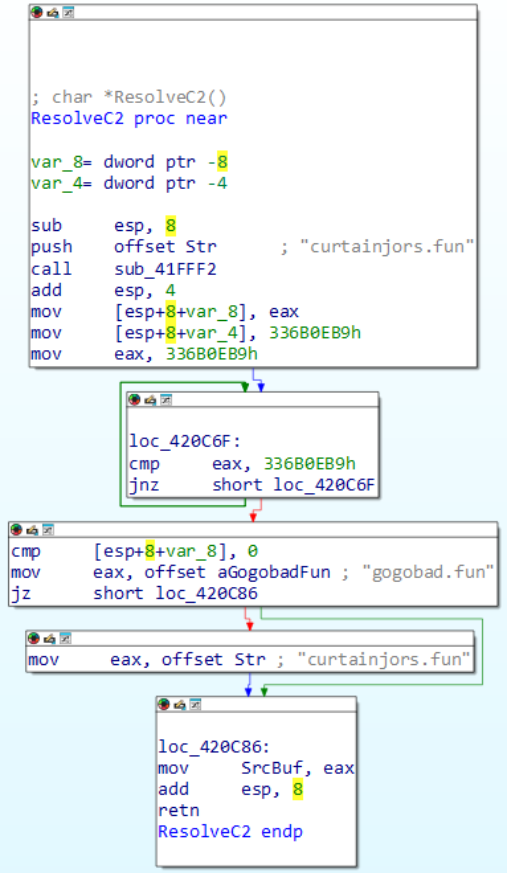
Hàm CheckSampleIsCrypted

Hàm này có thể làm bước cuối cùng xử lý anti, kiểm tra tính toàn vẹn của payload, có thể lưu cờ vào thanh ghi, khi payload được lấy riêng ra thì sẽ hiện GUI của nhà phát triển báo yes no. Ở đây ta đã trích payload ra phân tích.



Hình 50: Kiểm tra tính toàn vẹn

Kiểm tra ResolveC2, phân giải tên miền C2



Hình 51: Kiểm tra tên miền C2

Hàm sub_41FFF2(CheckDomain()) kiểm tra coi domain có ổn định, còn hoạt động hay kết nối có thành không hay không, nếu địa chỉ “curtainjors.fun” không tốt thì “gogobad.fun”.

Xem kỹ hơn về API liên quan đến mạng gọi:

Hình 52: API mạng Lumma Call

API	Mục đích
WinHttpOpen	Khởi tạo session HTTP
WinHttpConnect	Kết nối đến server C2 (curtainjors.fun)
WinHttpOpenRequest	Tạo một HTTP request (GET/POST)
WinHttpSendRequest	Gửi request lên C2 server
WinHttpReceiveResponse	Nhận phản hồi từ server
WinHttpReadData	Đọc nội dung từ phản hồi
WinHttpQueryDataAvailable	Kiểm tra còn dữ liệu chưa đọc không
WinHttpCrackUrl	Phân tích URL (thường dùng để trích phần domain/port)
WinHttpCloseHandle	Giải phóng tài nguyên

Bảng 11: Giải thích API Mạng

Hàm RealMain

Đây là hàm làm rối rất nặng nên phân tích tĩnh sẽ gây rất rối, nó còn rất nhiều hàm gọi và các chuỗi string nhưng không để lộ, chủ yếu là dùng hàm lstrcatw() để ghép lại, Ví dụ:

```
else if ( i == -1405015833 )
{
  st6_0 = sub_4322A0(v267, 1, &unk_46A756, 1);
  wcscat(v202, v267);
  i = -214026213;
}
```

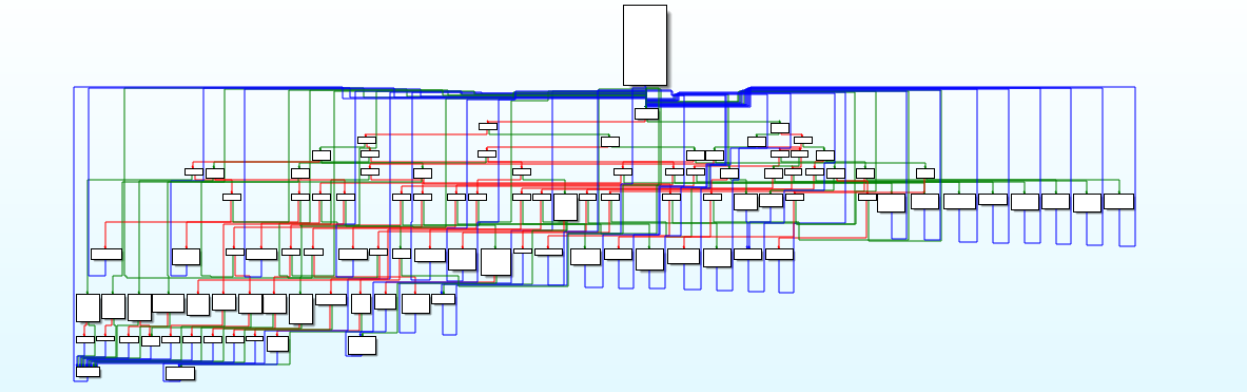
Hình 53:Ghép các chuỗi khi được chạy

```
sub_432338 proc near

var_20= dword ptr -20h
var_1C= dword ptr -1Ch
lpString1= dword ptr -18h
var_14= dword ptr -14h
var_10= dword ptr -10h
lpString2= dword ptr  8

push    ebp
mov     ebp, esp
push    ebx
push    edi
push    esi
sub     esp, 14h
mov     [ebp+var_1C], 81FA5C77h
mov     eax, 81FA5C77h
mov     esi, [ebp+var_20]
mov     ebx, ds:lstrcatW
```

Hình 54: API lstrcatW

=> Mã bị làm rối tuy không khó hiểu nhưng rất nhiều khiến người phân tích nản, muốn tìm được các string hay các hàm mà nghiêm trọng mà nó gọi phải debug.

Tổng quan phân tích tĩnh



Hình 55: Proximity Hàm

Payload sử dụng một **biến trạng thái (i) để điều hướng luồng thực thi** thông qua nhiều vòng lặp while lồng nhau và nhánh if – cấu trúc này được gọi là:

State Machine thủ công (manual Finite State Machine)

Trong đó:

- Biến i đóng vai trò như **chỉ báo trạng thái hiện tại**.
- Mỗi giá trị của i tương ứng với một **trạng thái cụ thể**, điều khiển đoạn mã được thực thi kế tiếp.
- Trạng thái được thay đổi bằng các phép gán mới cho i trong thân hàm.

Đây là một kỹ thuật phổ biến trong **obfuscation**, **control-flow flattening**, hoặc **anti-decompilation**. Nó khiến mã khó đọc và làm công cụ phân tích tĩnh (decompiler) như IDA, Ghidra khó tái dựng logic gốc.

4.2 Phân tích động

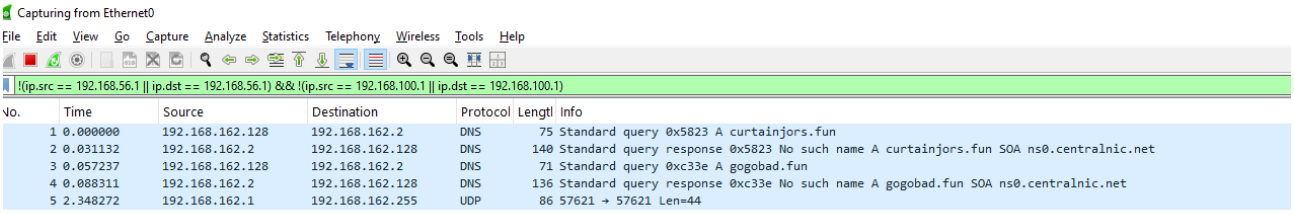
Để thực tế hơn về cách nó hoạt động thì phân tích động cũng là một điều quan trọng. Đầu tiên ta sẽ tiến hành chuẩn bị tool và đặt trên máy ảo, take snapshot ,để mạng host để tránh lan sang máy thật.

Cài đặt cơ bản:

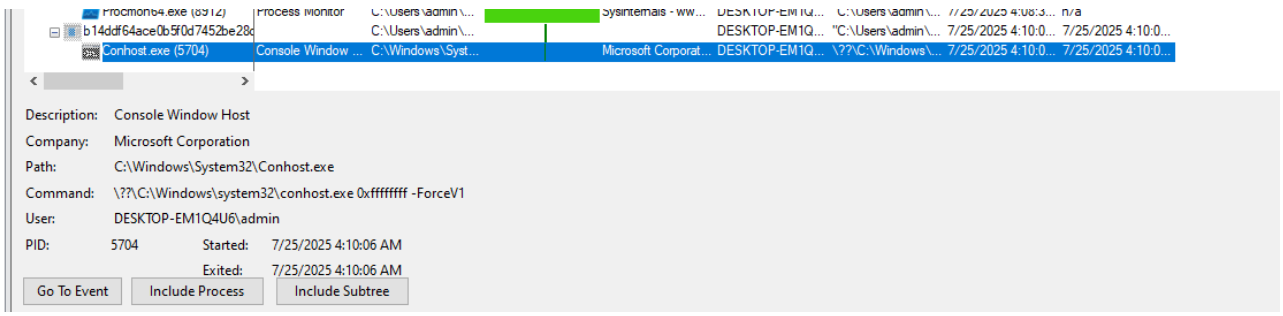
- Snapshot sạch trước khi chạy mẫu.
- Ngắt kết nối internet thực (dùng NAT ảo hoặc Host-Only).
- Không cài antivirus.
- Tắt Windows Defender và cập nhật.
- Cài công cụ phân tích.

Cài đặt công cụ

Hình 56: Công cụ phân tích động



Hình 57: Bắt DNS bằng wireshark

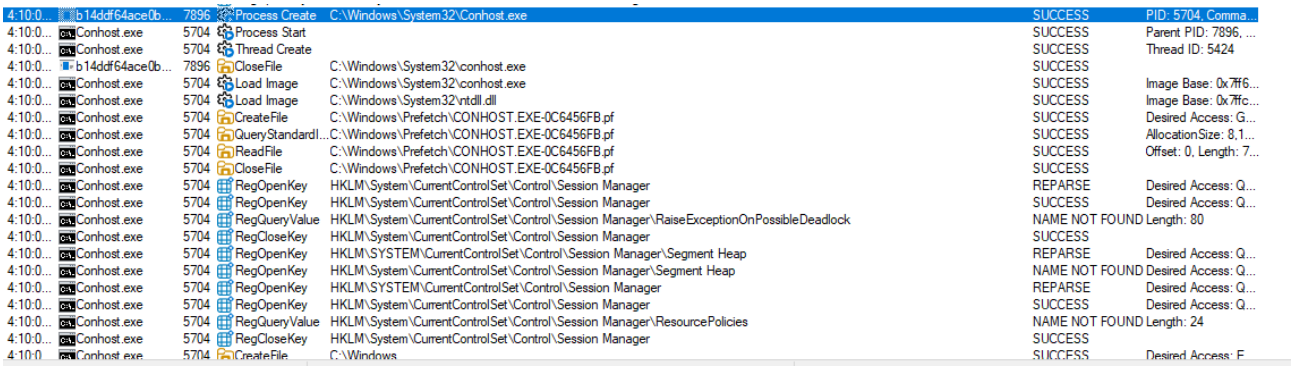


Hình 58: Process tree LummaC2

Tạo conhost.exe từ malware (không phải do cmd hoặc shell):

conhost.exe tiếp tục đọc & query Session Manager

Không có command line trong conhost.exe ⇒ Có thể **injected hoặc chạy ngầm**



4:10:0...	b14ddf64ace0b...	7896	Process Create	C:\Windows\System32\Conhost.exe	SUCCESS	PID: 5704, Comma...
4:10:0...	Conhost.exe	5704	Process Start		SUCCESS	Parent PID: 7896, ...
4:10:0...	Conhost.exe	5704	Thread Create		SUCCESS	Thread ID: 5424
4:10:0...	b14ddf64ace0b...	7896	CloseFile	C:\Windows\System32\conhost.exe	SUCCESS	
4:10:0...	Conhost.exe	5704	Load Image	C:\Windows\System32\conhost.exe	SUCCESS	Image Base: 0x7ff6...
4:10:0...	Conhost.exe	5704	Load Image	C:\Windows\System32\ntdll.dll	SUCCESS	Image Base: 0x7ffc...
4:10:0...	Conhost.exe	5704	CreateFile	C:\Windows\Prefetch\CONHOST.EXE-0C6456FB.pf	SUCCESS	Desired Access: G...
4:10:0...	Conhost.exe	5704	QueryStandardI...	C:\Windows\Prefetch\CONHOST.EXE-0C6456FB.pf	SUCCESS	AllocationSize: 8,1...
4:10:0...	Conhost.exe	5704	ReadFile	C:\Windows\Prefetch\CONHOST.EXE-0C6456FB.pf	SUCCESS	Offset: 0, Length: 7...
4:10:0...	Conhost.exe	5704	CloseFile	C:\Windows\Prefetch\CONHOST.EXE-0C6456FB.pf	SUCCESS	
4:10:0...	Conhost.exe	5704	RegOpenKey	HKLM\System\CurrentControlSet\Control\Session Manager	REPARSE	Desired Access: Q...
4:10:0...	Conhost.exe	5704	RegOpenKey	HKLM\System\CurrentControlSet\Control\Session Manager	SUCCESS	Desired Access: Q...
4:10:0...	Conhost.exe	5704	RegQueryValue	HKLM\System\CurrentControlSet\Control\Session Manager\RaiseExceptionOnPossibleDeadlock	NAME NOT FOUND	Length: 80
4:10:0...	Conhost.exe	5704	RegCloseKey	HKLM\System\CurrentControlSet\Control\Session Manager	SUCCESS	
4:10:0...	Conhost.exe	5704	RegOpenKey	HKLM\SYSTEM\CurrentControlSet\Control\Session Manager\Segment Heap	REPARSE	Desired Access: Q...
4:10:0...	Conhost.exe	5704	RegOpenKey	HKLM\System\CurrentControlSet\Control\Session Manager\Segment Heap	NAME NOT FOUND	Desired Access: Q...
4:10:0...	Conhost.exe	5704	RegOpenKey	HKLM\SYSTEM\CurrentControlSet\Control\Session Manager	REPARSE	Desired Access: Q...
4:10:0...	Conhost.exe	5704	RegOpenKey	HKLM\System\CurrentControlSet\Control\Session Manager	SUCCESS	Desired Access: Q...
4:10:0...	Conhost.exe	5704	RegQueryValue	HKLM\System\CurrentControlSet\Control\Session Manager\ResourcePolicies	NAME NOT FOUND	Length: 24
4:10:0...	Conhost.exe	5704	RegCloseKey	HKLM\System\CurrentControlSet\Control\Session Manager	SUCCESS	
4:10:0...	Conhost.exe	5704	CreateFile	C:\Windows	SUCCESS	Desired Access: E...

Hình 59: Tạo process conhost.exe

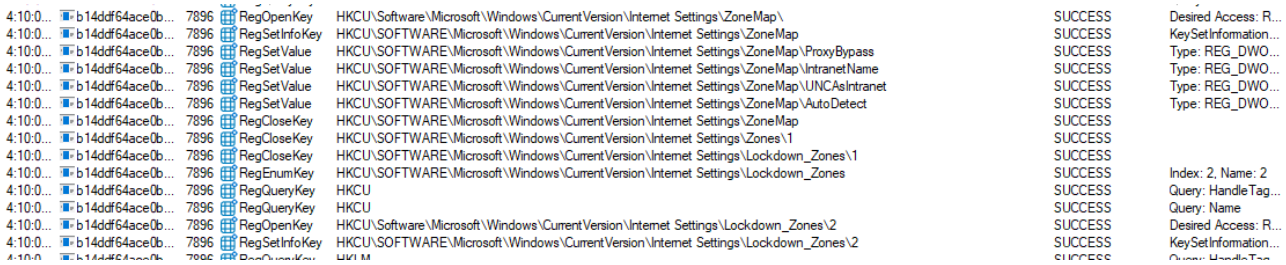
COM hijacking / COM object loading.

Ghi đè CLSID/ProgID trong Registry, dẫn đến các ứng dụng load DLL độc hại mỗi khi khởi động.

4:10:0...	Conhost.exe	5704	RegQueryKey	HKCR\CLSID\{529A9E6B-6587-4F23-AB9E-9C7D683E3C50}
4:10:0...	Conhost.exe	5704	RegQueryKey	HKCR\CLSID\{529A9E6B-6587-4F23-AB9E-9C7D683E3C50}
4:10:0...	Conhost.exe	5704	RegOpenKey	HKCU\Software\Classes\CLSID\{529A9E6B-6587-4F23-AB9E-9C7D683E3C50}
4:10:0...	Conhost.exe	5704	RegQueryValue	HKCR\CLSID\{529A9E6B-6587-4F23-AB9E-9C7D683E3C50}\(Default)
4:10:0...	Conhost.exe	5704	RegQueryKey	HKCR\CLSID\{529A9E6B-6587-4F23-AB9E-9C7D683E3C50}
4:10:0...	Conhost.exe	5704	RegQueryKey	HKCR\CLSID\{529A9E6B-6587-4F23-AB9E-9C7D683E3C50}
4:10:0...	Conhost.exe	5704	RegOpenKey	HKCU\Software\Classes\CLSID\{529A9E6B-6587-4F23-AB9E-9C7D683E3C50}
4:10:0...	Conhost.exe	5704	RegQueryValue	HKCR\CLSID\{529A9E6B-6587-4F23-AB9E-9C7D683E3C50}\(Default)
4:10:0...	Conhost.exe	5704	RegQueryKey	HKCR\CLSID\{529A9E6B-6587-4F23-AB9E-9C7D683E3C50}
4:10:0...	Conhost.exe	5704	RegQueryKey	HKCR\CLSID\{529A9E6B-6587-4F23-AB9E-9C7D683E3C50}
4:10:0...	Conhost.exe	5704	RegOpenKey	HKCU\Software\Classes\CLSID\{529A9E6B-6587-4F23-AB9E-9C7D683E3C50}\InprocServer32
4:10:0...	Conhost.exe	5704	RegQueryKey	HKCR\CLSID\{529A9E6B-6587-4F23-AB9E-9C7D683E3C50}
4:10:0...	Conhost.exe	5704	RegOpenKey	HKCR\CLSID\{529A9E6B-6587-4F23-AB9E-9C7D683E3C50}\InprocServer32
4:10:0...	Conhost.exe	5704	RegQueryKey	HKCR\CLSID\{529A9E6B-6587-4F23-AB9E-9C7D683E3C50}\InProcServer32
4:10:0...	Conhost.exe	5704	RegQueryKey	HKCR\CLSID\{529A9E6B-6587-4F23-AB9E-9C7D683E3C50}\InProcServer32
4:10:0...	Conhost.exe	5704	RegOpenKey	HKCU\Software\Classes\CLSID\{529A9E6B-6587-4F23-AB9E-9C7D683E3C50}\InProcServer32
4:10:0...	Conhost.exe	5704	RegQueryValue	HKCR\CLSID\{529A9E6B-6587-4F23-AB9E-9C7D683E3C50}\InProcServer32\InprocServer32
4:10:0...	Conhost.exe	5704	RegQueryKey	HKCR\CLSID\{529A9E6B-6587-4F23-AB9E-9C7D683E3C50}\InProcServer32
4:10:0...	Conhost.exe	5704	RegQueryKey	HKCR\CLSID\{529A9E6B-6587-4F23-AB9E-9C7D683E3C50}\InProcServer32
4:10:0	Conhost.exe	5704	RegOpenKey	HKCU\Software\Classes\CLSID\{529A9E6B-6587-4F23-AB9E-9C7D683E3C50}\InProcServer32

Hình 60: Tạo cổng COM giao tiếp

Chỉnh sửa các browsers



4:10:0...	b14ddf64ace0b...	7896	RegOpenKey	HKCU\Software\Microsoft\Windows\CurrentVersion\Internet Settings\ZoneMap\	SUCCESS	Desired Access: R...
4:10:0...	b14ddf64ace0b...	7896	RegSetInfoKey	HKCU\SOFTWARE\Microsoft\Windows\CurrentVersion\Internet Settings\ZoneMap	SUCCESS	KeySetInformation...
4:10:0...	b14ddf64ace0b...	7896	RegSetValue	HKCU\SOFTWARE\Microsoft\Windows\CurrentVersion\Internet Settings\ZoneMap\ProxyBypass	SUCCESS	Type: REG_DWO...
4:10:0...	b14ddf64ace0b...	7896	RegSetValue	HKCU\SOFTWARE\Microsoft\Windows\CurrentVersion\Internet Settings\ZoneMap\IntranetName	SUCCESS	Type: REG_DWO...
4:10:0...	b14ddf64ace0b...	7896	RegSetValue	HKCU\SOFTWARE\Microsoft\Windows\CurrentVersion\Internet Settings\ZoneMap\UNCAsIntranet	SUCCESS	Type: REG_DWO...
4:10:0...	b14ddf64ace0b...	7896	RegSetValue	HKCU\SOFTWARE\Microsoft\Windows\CurrentVersion\Internet Settings\ZoneMap\AutoDetect	SUCCESS	Type: REG_DWO...
4:10:0...	b14ddf64ace0b...	7896	RegCloseKey	HKCU\SOFTWARE\Microsoft\Windows\CurrentVersion\Internet Settings\ZoneMap	SUCCESS	
4:10:0...	b14ddf64ace0b...	7896	RegCloseKey	HKCU\SOFTWARE\Microsoft\Windows\CurrentVersion\Internet Settings\Zones\1	SUCCESS	
4:10:0...	b14ddf64ace0b...	7896	RegCloseKey	HKCU\SOFTWARE\Microsoft\Windows\CurrentVersion\Internet Settings\Lockdown_Zones\1	SUCCESS	
4:10:0...	b14ddf64ace0b...	7896	RegEnumKey	HKCU\SOFTWARE\Microsoft\Windows\CurrentVersion\Internet Settings\Lockdown_Zones	SUCCESS	Index: 2, Name: 2
4:10:0...	b14ddf64ace0b...	7896	RegQueryKey	HKCU	SUCCESS	Query: HandleTag...
4:10:0...	b14ddf64ace0b...	7896	RegQueryKey	HKCU	SUCCESS	Query: Name
4:10:0...	b14ddf64ace0b...	7896	RegOpenKey	HKCU\Software\Microsoft\Windows\CurrentVersion\Internet Settings\Lockdown_Zones\2	SUCCESS	Desired Access: R...
4:10:0...	b14ddf64ace0b...	7896	RegSetInfoKey	HKCU\SOFTWARE\Microsoft\Windows\CurrentVersion\Internet Settings\Lockdown_Zones\2	SUCCESS	KeySetInformation...

Hình 61: Chỉnh sửa Internet Settings

Registry Key	Vai trò
ZoneMap\ProxyBypass	Quy định danh sách domain hoặc IP bỏ qua proxy — malware có thể thêm domain C2 ở đây
IntranetName, UNCAsIntranet, AutoDetect	Dùng để "lừa" Internet Explorer/WebView coi domain lạ là intranet (ít bảo mật hơn)
Lockdown_Zones	Áp dụng chính sách chặt hơn khi dùng Protected Mode — malware ghi vào đây có thể **giảm mức bảo mật mặc định** để dễ khai thác hơn

Bảng 12: Các Key liên quan browsers

thực hiện hàng loạt truy vấn và chỉnh sửa vào registry:

HKCU\Software\Microsoft\Windows\CurrentVersion\Internet Settings\ZoneMap

ZoneMap được dùng bởi Internet Explorer / WebBrowser Control để phân chia các vùng bảo mật (Zone 1: Intranet, 2: Trusted, 3: Internet,...)

Có chỉnh sửa cả:

- Zones\3\A110 → viết giá trị REG_DWORD
- Zones\3 → có RegSetInfoKey → cập nhật metadata

Đây có thể là thủ thuật malware dùng để điều chỉnh bảo mật vùng Internet, ví dụ:

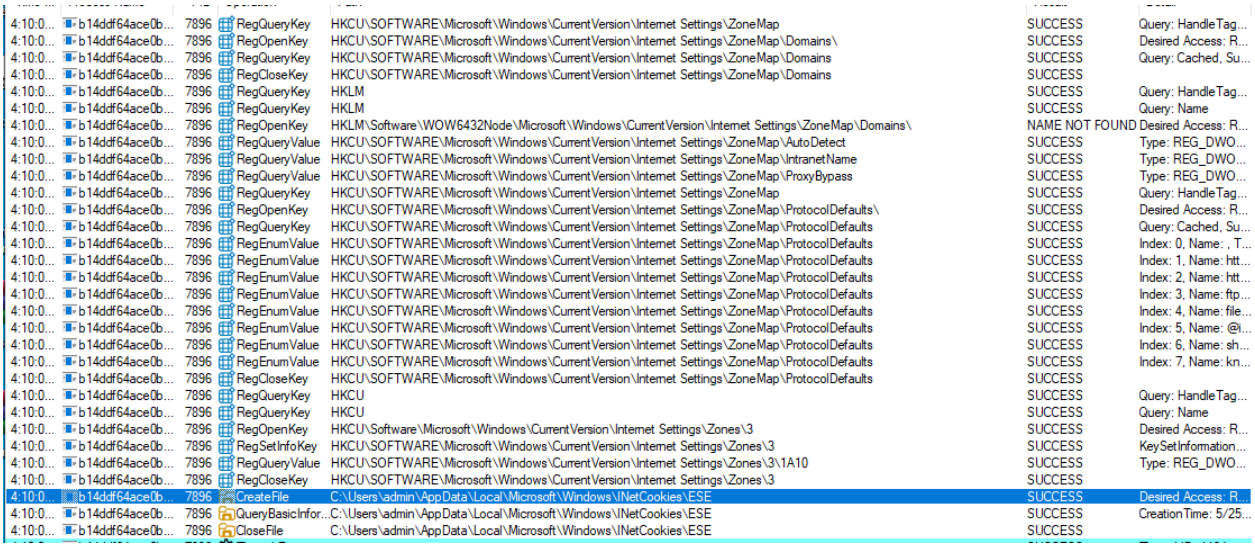
- Cho phép tải ActiveX/Script
- Giảm bảo mật Internet Zone
- Điều khiển WebBrowser behavior

Hành vi đáng chú ý: Tạo file ESE

CreateFile: C:\Users\admin\AppData\Local\Microsoft\Windows\INetCookies\ESE

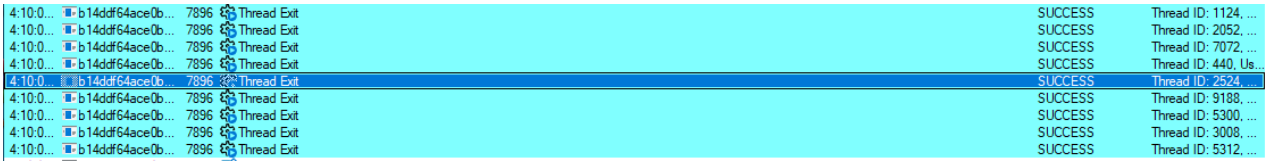
Nhận định:

- Đây là **một file .ESE**: có thể là file ESE (Extensible Storage Engine) — cấu trúc cơ sở dữ liệu nội bộ của Windows.
- **INetCookies** là thư mục lưu **cookie database** của IE/Edge.
- Malware có thể ghi thêm dữ liệu giả vào cookie hoặc **ẩn dữ liệu cấu hình C2**, token, hoặc kỹ thuật **steganography** để tránh bị phát hiện.



Hình 62: Tạo file ESE

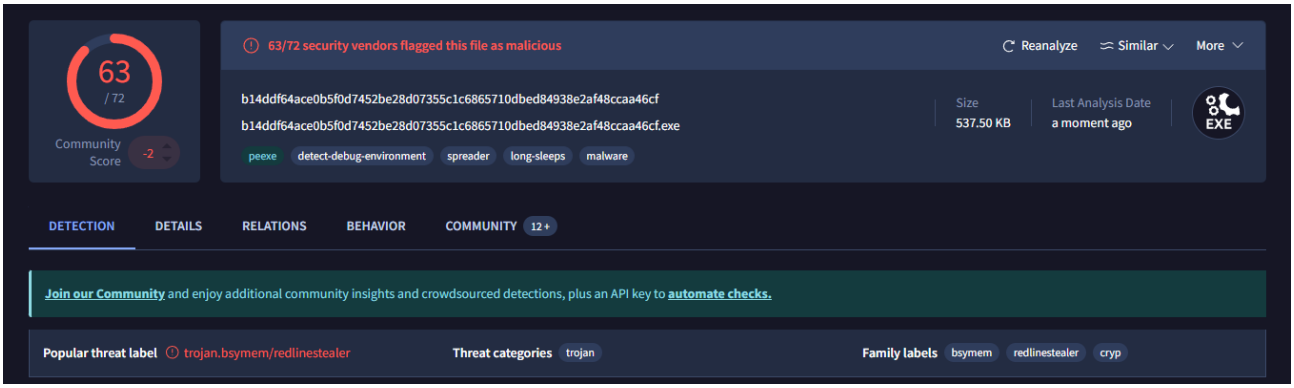
Sau khi nó cài cắm vào hết sẽ clear hết thread



Hình 63: Clear Thread

4.3 Phân tích bằng công cụ trực tuyến

4.3.1 Virus Total



Hình 64: Total Virus Tổng quan

Phát hiện:

- **63/72** hãng AV đánh dấu là **malware**
- Nhãn chính: trojan.bsymem.redlinestealer

- Dạng file: .exe, kích thước ~537.50 KB
- Tag VirusTotal:
    - peexe
    - detect-debug-environment → Có khả năng **anti-analysis**
    - spreader → Có **khả năng lây lan**
    - long-sleeps → Dùng kỹ thuật **ẩn mình chờ đợi** hoặc delay khi debug

Kết nối mạng

Contacted URLs (5)

Scanned	Detections	Status	URL
2025-07-16	13 / 97	-	http://curtainjors.fun/
2025-07-23	0 / 97	200	http://www.microsoft.com/pki/certs/MicCodSigPCA_08-31-2010.crt
2025-07-24	0 / 97	200	http://crt.sectigo.com/SectigoPublicCodeSigningCAR36.crt
2025-07-24	0 / 97	200	http://crt.sectigo.com/SectigoPublicCodeSigningRootR46.p7c
2025-07-03	0 / 97	200	http://www.microsoft.com/pki/certs/MicrosoftTimeStampPCA.crt

Contacted Domains (9)

Domain	Detections	Created	Registrar
crt.sectigo.com	0 / 94	2018-08-16	CSC CORPORATE DOMAINS, INC.
curtainjors.fun	14 / 94	2023-09-24	PDR Ltd. d/b/a PublicDomainRegistry.com
fp2e7a.wpc.2be4.phicdn.net	1 / 94	2014-11-14	MarkMonitor Inc.
fp2e7a.wpc.phicdn.net	1 / 94	2014-11-14	MarkMonitor Inc.
gogobad.fun	13 / 94	2023-09-14	PDR Ltd. d/b/a PublicDomainRegistry.com
microsoft.com	0 / 94	1991-05-02	MarkMonitor Inc.
res.public.onecdn.static.microsoft	0 / 94	2023-05-05	MarkMonitor Inc.
sectigo.com	0 / 94	2018-08-16	CSC CORPORATE DOMAINS, INC.
www.microsoft.com	0 / 94	1991-05-02	MarkMonitor Inc.

Hình 65: Kết nối mạng

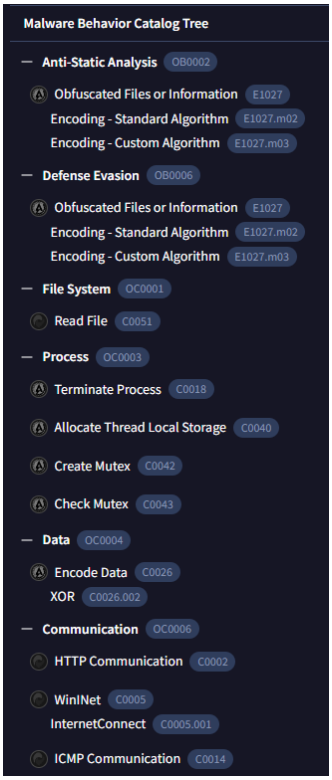
- curtainjors.fun là **domain C2 chính,** Gogobad.fun là domain phụ được malware gọi về lần đầu (dấu hiệu rất rõ: 13/97 engine flag).
- Các URL còn lại là **tải về certificate giả lập hợp lệ** — chiêu của crypter để đánh lừa máy phân tích tự động.

Contacted IP addresses (33)

IP	Detections	Autonomous System	Country
104.18.14.101	0 / 94	13335	-
184.27.218.92	1 / 94	16625	US
192.168.0.2	0 / 94	-	-
192.168.0.50	0 / 94	-	-
192.229.211.108	0 / 94	15133	US
192.229.221.95	0 / 94	-	US
20.69.140.28	0 / 94	8075	US
20.99.133.109	0 / 94	8075	US
20.99.184.37	0 / 94	8075	US
20.99.185.48	0 / 94	8075	US

Hình 66: Kết nối IP

- 184.27.218.92 (Akamai CDN): bị flag **1/94**, có thể chứa hoặc chuyển hướng đến shell / payload phụ.
- Đây rất có thể là relay traffic C2 hoặc stage 2, giúp ẩn ip thật của hacker.



Hình 67: Cây hành vi

Hành vi	**Mức độ nguy hiểm**	**Diễn giải**
Obfuscation + XOR	Rất cao	Dấu hiệu packer/custom loader
Mutex + TLS	Cao	Có cấu trúc bảo vệ, đa tiến trình
HTTP + WinINet	Rất cao	C2 giao tiếp chuẩn thông qua API
ReadFile	Trung bình	Có thể đánh cắp thông tin
Terminate Process	Cao	Diệt các chương trình bảo vệ

Bảng 13: Hành vi và mức độ nguy hiểm

Bảng 14: Hành vi

Processes Created - Stage1 và Stage2 Loader

Tập tin khởi tạo	Vai trò
executable.exe tại Desktop	Likely là Initial Dropper/Loader
%SAMPLEPATH%*.exe (chuỗi hash)	Đã được viết ra disk bởi dropper, chứa stage2
C:\Program Files\Google3008_...\updater.exe	Giả mạo updater, là payload hoặc stage2 loader
conhost.exe 0xffffffff -ForceV1	Tạo console ẩn, thường là để thực thi shellcode/shell loader
wuapihost.exe và UIODetect.exe	Có thể là DLL side-load hoặc giả mạo service hợp pháp

Bảng 15: Processes Created

Shell Commands – Giai đoạn lén lút

Dòng lệnh	Mục đích
--update --system --enable-logging --vmodule=*chrome/updater*	**Fake update logic**, có thể tải stage 2

Dòng lệnh	Mục đích
conhost.exe ... -ForceV1	Có thể để mở process nền kiểu stealth
wuapihost.exe -Embedding	**Giả mạo COM service** để tránh AV

Bảng 16: Shell Commands

Processes Injected - Hành vi rõ của STAGE2/3

Inject vào	Giải thích
updater.exe (Google fake)	Inject chính vào payload giả mạo Google Updater
WMIADAP.EXE	WMI hijack / process hollowing (thường thấy ở SquirrelWaffle, Formbook)

Bảng 17: Processes Injected

Processes Terminated - Tránh phân tích

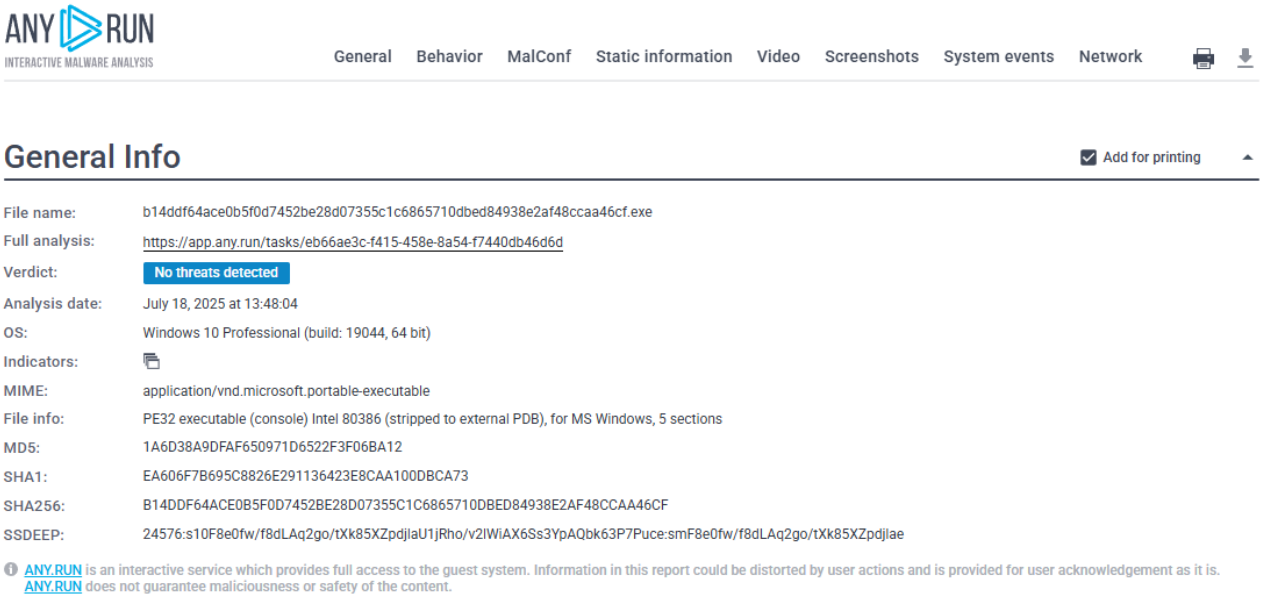
Bị kill	Mục đích
WerFault.exe	Tắt Windows Error Reporting – tránh log crash
svchost.exe -k WerSvcGroup	Dừng dịch vụ lỗi
Dllhost.exe, wmiadap.exe	Inject xong rồi kill để không bị nghi ngờ
conhost, wuapihost, updater.exe	Dọn dẹp dấu vết

Bảng 18: Processes Terminated

Nhận định :

- Đây là malware đa tầng (multi-stage) có hành vi loader → inject → connect to C2.
- Cơ chế giả mạo phần mềm hợp pháp như Chrome Updater, Google Updater.
- Có khả năng là malware dạng stealer / RAT / info-stealer (RedLine, Formbook, Aurora v.v).
- C2 chính sử dụng domain .fun, mới đăng ký gần đây, bị nhiều hãng AV flag.

4.3.2 AnyRun



Hình 68: Tổng quan LummaC2 AnyRun

Ở anyrun vẫn có những hành vi như vậy nhưng nó đã qua mặt được phân tích động. Có thể là do C2 đã thay đổi.

Hình 69: Sơ đồ Process LummaC2

Có tạo cổng COM để giao tiếp nhưng không có gì

V. TỔNG KẾT

Trong quá trình thực hiện phân tích mã độc **Lumma Stealer**, nhóm đã tiến hành đầy đủ các bước từ thu thập mẫu, phân tích hành vi động – tĩnh, truy vết liên lạc đến máy chủ C2, và ghi nhận các hành vi đáng ngờ liên quan đến Registry, mạng, và các kỹ thuật né tránh phân tích. Qua đó, nhóm đã rút ra được một số nhận định quan trọng:

- Lumma Stealer có cơ chế thực thi theo từng **stage (giai đoạn)** rõ rệt, từ dropper cho đến tải mã độc giai đoạn 2.
- Mã độc thực hiện **giao tiếp mã hóa TLS** với C2 và sử dụng **địa chỉ relay** nhằm che giấu nguồn gốc.
- Các hành vi như **ghi Registry để tự khởi động**, **giải mã chuỗi tại runtime**, và **phân mảnh mã lệnh** giúp mã độc tránh bị phát hiện bởi antivirus và sandbox.

- Việc phân tích hành vi với công cụ **Procmon** cho thấy rõ quá trình ghi khóa autorun, cũng như cách malware tìm cách duy trì sự tồn tại sau khi hệ thống khởi động lại.
- Kết quả so sánh từ **VirusTotal** cho thấy mẫu mã độc đã bị nhận diện bởi nhiều antivirus, tuy nhiên vẫn có tỷ lệ nhất định không nhận diện được.

Hạn chế:

- Quá trình phân tích chưa mô phỏng đầy đủ tương tác của attacker phía C2, dẫn đến thiếu một phần về stage 2 và payload cuối cùng.
- Một số kỹ thuật chống phân tích vẫn chưa giải mã hoàn toàn (obfuscation nâng cao hoặc API resolve động).

Định hướng phát triển:

- Tìm hiểu và mô phỏng lại C2 server để phân tích stage 2 trong môi trường kiểm soát.
- Sử dụng các framework như **CAPEv2**, **UnpacMe**, hoặc **IDA Pro plugin** để tự động hóa phân tích.

Qua bài báo cáo này, nhóm đã nâng cao được kỹ năng phân tích mã độc thực tế, từ đó củng cố kiến thức về bảo mật hệ thống và phản ứng sự cố an ninh mạng.