

**BAN CƠ YẾU CHÍNH PHỦ**

**HỌC VIỆN KỸ THUẬT MẬT MÃ**

¯ ¯ ¯ ¯ ¯ ¯ ¯ ¯ ¯ ¯ ¯ ¯ ¯ ¯ ¯ ¯

**CHUYÊN ĐỀ AN TOÀN THÔNG TIN**

**Tìm hiểu về kỹ thuật Bypass AV**

**Giáo viên hướng dẫn : Hoàng Thanh Nam**

**Sinh viên thực hiện: Bùi Thị Hương - AT170227**

**Phùng Tiến Đạt – AT170309**

Hà Nội, 2023

# Lời nói đầu

Trong thế giới số ngày càng phát triển, việc bảo vệ dữ liệu và hệ thống thông tin đã trở thành một phần không thể thiếu trong cuộc sống hằng ngày. Sự xuất hiện của các phần mềm độc hại (malware) không chỉ gây ra những tổn thất lớn về mặt tài chính mà còn ảnh hưởng đến sự an toàn của thông tin cá nhân và doanh nghiệp. Trong bối cảnh đó, antivirus và các phương pháp bypass antivirus không ngừng được cải tiến và phát triển.

Bản báo cáo này được soạn thảo với mục tiêu chính là cung cấp một cái nhìn toàn diện và sâu sắc về hệ thống Antivirus (AV) - một công cụ không thể thiếu trong bối cảnh kỹ thuật số hiện đại cũng như các kỹ thuật Bypass antivirus. Báo cáo tập trung vào ba khía cạnh chính:

(1) **Hiểu rõ về cách thức hoạt động và sự phát triển của các hệ thống AV**: Báo cáo sẽ giải thích cách mà các hệ thống AV hoạt động để phát hiện và ngăn chặn các mối đe dọa.

(2) **Phân tích các kỹ thuật mà các loại malware sử dụng để bypass các hệ thống AV**: Bạn đọc sẽ được làm quen với các chiến lược và kỹ thuật mà malware sử dụng để tránh bị phát hiện bởi các chương trình antivirus. Điều này sẽ giúp nhìn nhận những thách thức mà cộng đồng an ninh mạng đang đối mặt.

(3) **Khám phá xu hướng phát triển của AV trên thế giới nói chung và tại Việt Nam nói riêng**: Báo cáo sẽ đánh giá sự phát triển của các giải pháp antivirus trên toàn cầu và trong bối cảnh cụ thể của Việt Nam. Điều này sẽ giúp người đọc hiểu rõ về những tiến triển và thách thức đặt ra trong lĩnh vực an ninh mạng.

Qua đó, nhóm em hy vọng sẽ mang đến cho người đọc cái nhìn cập nhật và đa chiều về lĩnh vực an ninh mạng và vai trò của AV trong việc bảo vệ thông tin số. Bằng cách này, bản báo cáo không chỉ là nguồn thông tin hữu ích cho những người làm trong lĩnh vực an ninh mạng mà còn đem lại sự hiểu biết sâu sắc về tầm quan trọng của việc duy trì và cập nhật các giải pháp bảo mật trong môi trường số ngày nay.

# Mục Lục

**Lời nói đầu**

**Mục lục**

**Danh mục hình ảnh**

**Danh mục ký hiệu viết tắt**

**CHƯƠNG I. TỔNG QUAN LÝ THUYẾT**

1. Khái niệm về Antivirus

2. Khái niệm về Bypass Antivirus

3. Sự phát triển của Antivirus.

3.1. Từ Creeper đến Brain: Bình Minh của Antivirus.

3.2. Bước Chuyển Mình trong Công Nghệ Antivirus: Những Năm 1990 đến 2000.

3.3. Đổi Mới và Phát Triển trong Thế Kỷ 21: AI và Học Máy.

Kết luận Chương I

**CHƯƠNG II. PHÂN TÍCH CÁC KỸ THUẬT ANTIVIRUS VÀ BYPASS AV**

**Kỹ thuật Antivirus**

1. Dựa trên Chữ Ký (Signature-Based):

1.1. Tổng quan về kỹ thuật Signature- Based

1.2. Phân tích hành vi thực hiện

1. Dựa trên Học (Behavior-Based):

2.1. Tổng quan về kỹ thuật Behaviour- Based

2.2. Phân tích hành vi thực hiện

3. Dựa trên Hành Vi (Heuristic-Based):

3.1. Tổng quan về kỹ thuật quét Heuristic- Based

3.2. Phân tích hành vi thực hiện

**Kỹ thuật Bypass AV**

4. Antivirus Evasion Techniques: On Disk

4.1. Obfuscators

4.2. Packers, Cryptors, Protectors

5. Antivirus Evasion Technique: In Memory

5.1. Classic DLL Injection Via createremotethread And Load Library

5.2. Process Hollowing ( A.K.A Process Replacement And runpe )

5.3. IAT Hooking And Inline Hooking (A.K.A userland Rootkits )

Kết luận Chương II

**CHƯƠNG III. THỰC NGHIỆM KỊCH BẢN ANTIVIRUS VÀ BYPASS AV**

1. Kịch bản 1: Tạo dựng 1 chương trình antivirus và 1 chương trình Bypass antivirus

2. Kịch bản 2: Phân tích Signature-Based -Kaspersky Core Malware

3. Kịch bản 3: Phân tích mã độc sử dụng packer bằng phần mềm UPX

4. Kịch bản 4: Phân tích mã độc plugx, sử dụng kỹ thuật mã hóa để mã hóa thông tin file cũng như nội dung thực thi.

Kết luận Chương III

**KẾT LUẬN**

**Tài liệu tham khảo**

# DANH MỤC HÌNH VẼ

# DANH MỤC KÝ HIỆU VIẾT TẮT

# Chương I. Cơ sở lý thuyết

## 1. Giới thiệu chung về Antivirus

1.1. **Định Nghĩa và Mục Đích**

**Antivirus** là một loại phần mềm được thiết kế để phát hiện, ngăn chặn, và loại bỏ các loại phần mềm độc hại (malware). Mục đích chính của nó là bảo vệ hệ thống máy tính khỏi virus, trojan, worm, spyware, và adware.

1.2. **Cách Thức Hoạt Động**

**- Phát Hiện Dựa Trên Chữ Ký (Signature-Based Detection)**: Phương pháp này sử dụng cơ sở dữ liệu chữ ký của các phần mềm độc hại đã biết để phát hiện và ngăn chặn chúng.

**- Phát Hiện Dựa Trên Hành Vi (Behavior-Based Detection)**: Đây là cách tiếp cận hiện đại hơn, nơi a antivirus phát hiện ra hành vi đáng ngờ (như sự thay đổi đột ngột trong hệ thống tập tin) thay vì dựa vào chữ ký.

**- Quét Heuristic**: Là kỹ thuật phát hiện những mối đe dọa mới và không biết trước bằng cách xem xét mã của chúng để tìm ra những đặc điểm nghi ngờ.

1.3. **Các Tính Năng và Công Nghệ Khác**

**- Tường lửa tích hợp (Firewall Integration)**: Một số antivirus cung cấp tường lửa tích hợp để kiểm soát dữ liệu vào và ra khỏi máy tính.

**- Bảo vệ Email và Web**: Chống lại các mối đe dọa từ email và trang web độc hại.

**- Cập Nhật Tự Động**: Các cập nhật tự động đảm bảo rằng phần mềm có thể nhận diện các mối đe dọa mới nhất.

## 2. Giới thiệu chung về Bypass Antivirus

2.1. Định Nghĩa và Mục Đích

**Bypass Antivirus** là các phương pháp và kỹ thuật mà hacker hoặc phần mềm độc hại sử dụng để tránh bị phát hiện bởi các phần mềm antivirus. Mục tiêu là triển khai mã độc mà không bị phát hiện.

2.2. Phương Pháp Phổ Biến

Các kỹ thuật bypass Antivirus (AV) như Obfuscatiors, Packers, Cryptors, Protectors, Process Memory Injection, DLL Injection, Inline Hooking, và Process Hollowing, đều là những phương pháp mà các tác giả malware sử dụng để tránh bị phát hiện bởi phần mềm AV. Dưới đây là giới thiệu chung từng kỹ thuật:

**- Obfuscatiors**:

Mục Đích: Làm rối mã độc để ngăn chặn phân tích.

Cách Thức: Thay đổi mã sao cho khó đọc và phân tích, có thể bao gồm việc thêm mã chết hoặc thay đổi logic mã.

* **Packers**:

Mục Đích: Giảm kích thước và ẩn mã độc.

Cách Thức: Nén mã độc và thực thi nó trong môi trường nén, thường sử dụng các công cụ như winrar.

* **Cryptors**:

Mục Đích: Mã hóa mã độc để tránh bị phát hiện.

Cách Thức: Mã hóa mã và sử dụng hàm giải mã, thực thi mã trong bộ nhớ và giải mã trên đĩa.

* **Protectors**:

Mục Đích: Ngăn chặn mã bị đảo ngược hoặc gỡ lỗi.

Cách Thức: Bảo vệ mã độc dưới lớp bảo vệ để đánh lừa AV.

* **Process Memory Injection**:

Mục Đích: Chèn mã độc vào quá trình đang chạy.

Cách Thức: Sử dụng API Windows để thay đổi bộ nhớ của quá trình đang chạy và chèn mã độc.

* **DLL Injection**:

Mục Đích: Chèn thư viện DLL độc hại vào quá trình.

Cách Thức: Tương tự như Process Memory Injection nhưng tập trung vào việc chèn DLL.

* **Inline Hooking**:

Mục Đích: Thay đổi hành vi của hàm hệ thống.

Cách Thức: Thêm mã độc vào giữa quá trình thực thi của một hàm hệ thống.

* **Process Hollowing**:

Mục Đích: Thay thế mã của một quá trình hợp lệ bằng mã độc.

Cách Thức: Tạm ngưng một quá trình, thay đổi bộ nhớ của nó bằng mã độc, sau đó tiếp tục quá trình.

Mỗi kỹ thuật này đều có những đặc điểm riêng biệt nhằm che giấu hoặc biến đổi mã độc để tránh bị phát hiện bởi các cơ chế bảo vệ thông thường của phần mềm AV.

2.3 Thách thức đối với phần mềm Antivirus

Thách thức đối Với Phần Mềm Antivirus: Các kỹ thuật bypass antivirus liên tục phát triển, buộc các nhà sản xuất phần mềm antivirus phải liên tục cập nhật và nâng cấp công nghệ của họ.

## 3. Sự phát triển của Antivirus.

3.1. Từ Creeper đến Brain: Bình Minh của Antivirus.

**- Virus Creeper (1971)**: Virus máy tính đầu tiên được biết đến. Creeper chỉ tự sao chép và không gây hại cho dữ liệu. Điều này đã khởi đầu cho nhu cầu phát triển phần mềm antivirus.

**- Virus Brain (1986)**: Là một trong những virus máy tính đầu tiên gây hại thực sự, Brain lây nhiễm vào sector khởi động của đĩa mềm. Sự xuất hiện của Brain đã thúc đẩy sự phát triển của các giải pháp antivirus đầu tiên.

**=>** Sự Ra Đời của Antivirus: Trong thời kỳ này, các chương trình như IBM's Antivirus, mcafee's virusscan, và Norton Antivirus được phát triển để chống lại virus.

3.2. Bước Chuyển Mình trong Công Nghệ Antivirus: Những Năm 1990 đến 2000.

Các bước chuyển mình trong Công nghệ AV từ những năm 1990 đến 2000 đã có sự chuyển biến rõ rệt, mọi thứ bắt đầu đều do sự phát triển công nghệ, dẫn đến mọi thứ cũng đều được phát triển theo.

- Sự Gia Tăng của Malware: Thập niên 90 chứng kiến sự gia tăng nhanh chóng về số lượng và độ phức tạp của malware, bao gồm cả virus, trojan và worm.

- Phát Triển Công Nghệ: Công nghệ antivirus bắt đầu tích hợp các kỹ thuật phức tạp hơn như heuristic analysis và behavior-based detection, không chỉ dựa vào database chữ ký.

- Internet và Bảo Mật Mạng: Sự phổ biến của Internet dẫn đến sự phát triển của các giải pháp bảo mật mạng và cloud-based antivirus, giúp cập nhật nhanh chóng và phân phối chữ ký virus mới.

3.3. Đổi Mới và Phát Triển trong Thế Kỷ 21: AI và Học Máy.

Công nghệ phát triển dẫn đến sự đổi mới và phát triển diễn ra liên tục và thường xuyên.

- AI và Học Máy: Gần đây, AI và học máy đã trở thành các công cụ quan trọng trong việc phát hiện và chống lại malware. Chúng giúp phân tích và phát hiện các mối đe dọa mới mà không cần chữ ký cố định.

- Tự Động Hóa và Thích Ứng: Antivirus ngày nay có khả năng tự động học và thích ứng với các loại mối đe dọa mới, thông qua việc phân tích mô hình hành vi và dữ liệu lớn.

- Phản Ứng Trước Các Mối Đe Dọa Mới: Nhờ AI và học máy, các phần mềm antivirus có thể phản ứng nhanh chóng trước các mối đe dọa chưa từng thấy, đồng thời cung cấp cảnh báo sớm và bảo vệ chống lại các cuộc tấn công zero-day.

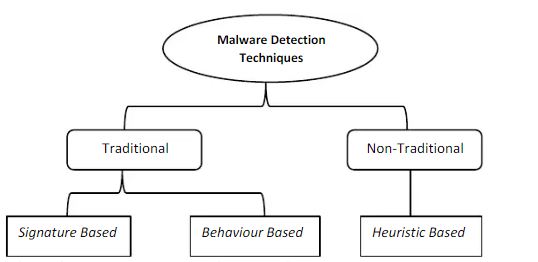
## Kết luận Chương I

Qua chương này, chúng ta có thể thấy rõ sự cần thiết của việc tiếp tục nghiên cứu và phát triển trong lĩnh vực bảo mật máy tính. Sự tiến bộ của công nghệ AV không chỉ phản ánh nhu cầu bảo vệ dữ liệu mà còn là cuộc đua không ngừng với các hacker và phần mềm độc hại. Sự cân nhắc giữa phát triển công nghệ bảo mật và hiểu biết về cách thức mà kẻ tấn công bypass những hệ thống bảo mật này là yếu tố quan trọng cho sự an toàn của thông tin trong kỷ nguyên số hiện nay.

# Chương II. PHÂN TÍCH CÁC KỸ THUẬT ANTIVIRUS VÀ BYPASS ANTIVIRUS

## Kỹ thuật Antivirus

Trong phần này, chúng ta sẽ đi sâu vào việc khám phá các kỹ thuật mà phần mềm antivirus sử dụng để phát hiện và ngăn chặn malware. Kỹ thuật phát hiên mã độc được chia làm 2 loại: Traditional và Non- Traditional.



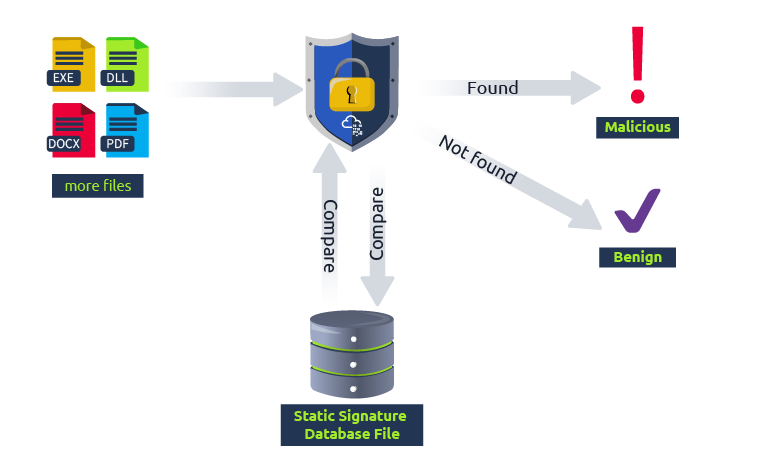
Hình 1. Ảnh phân loại kỹ thuật phát hiện mã độc

Các phương pháp này bao gồm phát hiện qua : Signature Based, Behaviour Based và Heuristic Based.

## 1. Dựa trên Chữ Ký (Signature-Based):

### 1.1. Tổng quan về kỹ thuật Signature-Based

Signature Based là một thuật toán hoặc một mã hash định danh độc nhất cho một tệp tin, do đó, Signature Malware là một đặc điểm độc nhất được trích xuất từ mã viral (các đặc tính). Mỗi một chữ ký là một đặc điểm độc nhất cho mỗi tệp tin, tương tự như dấu vân tay của chúng ta. Chữ ký thường được trích xuất với sự nhạy cảm đặc biệt để đảm bảo tính duy nhất, vì vậy các phương pháp phát hiện sử dụng chữ ký này có tỷ lệ lỗi nhỏ. Dựa trên chữ ký cũng có thể được gọi là dựa trên Hash.



Hình 2: Ảnh cách hoạt động dựa trên chữ ký

### 1.2. Phân tích hành vi thực hiện

Phương pháp này hoạt động như sau: Khi một tệp tin đến, chương trình Anti-malware phân tích thống kê tệp tin đến và so sánh xem có trùng khớp nào với các chữ ký đã được lưu trữ sẵn. Nếu có trùng khớp với các chữ ký malware trong tập dữ liệu, một tín hiệu báo động sẽ được kích hoạt để cho biết rằng tệp tin này là độc hại.

* Phân loại: Phương pháp signature-based có thể được thực hiện thông qua ba loại phân tích:
  + Phân tích Tĩnh (Static Analysis): Quét và phân tích mã mà không cần thực thi nó. Công cụ giải mã sẽ được sử dụng để thu thập thông tin chi tiết từ mã nguồn. Phương pháp này có thể không hiệu quả với mã nguồn đã được mã hóa hoặc làm rối.
  + Phân tích Động (Dynamic Analysis): Thực thi file để nghiên cứu hành vi của nó trong hệ điều hành. Phương pháp này sử dụng các kỹ thuật như máy ảo hoặc sandbox để phân tích mã độc.
  + Phân tích Kết hợp (Hybrid Analysis): Kết hợp cả hai phương pháp trên, tăng cường khả năng phát hiện và độ chính xác.
* Các thuật toán phổ biến: Aho-Corasick, Boyer-Moore, và Wu-Manber là một số thuật toán nổi bật được sử dụng trong signature-based detection. Mỗi thuật toán có ưu và nhược điểm riêng về hiệu suất và độ chính xác.

=> Kỹ thuật dựa trên chữ ký trong sản phẩm antivirus là một phương pháp phát hiện malware truyền thống và vẫn còn rất quan trọng, dù có một số hạn chế. Các phương pháp phân tích tĩnh, động và kết hợp cung cấp các cách tiếp cận khác nhau để tăng cường khả năng phát hiện và giảm thiểu rủi ro do các loại malware mới gây ra.

Nhà phát triển Malware đã sử dụng tính đa hình và biến đổi để tránh phát hiện dựa trên chữ ký; do đó, họ tạo ra các virus có thể thay đổi chữ ký của họ. Và với điều này, các virus đa hình và biến đổi đã làm cho việc phát hiện dựa trên chữ ký trở nên không hiệu quả.

Các ưu điểm của kỹ thuật phát hiện malware dựa trên Signature-Based như sau:

* Tỷ lệ lỗi nhỏ
* Quét và phân tích rất nhanh chóng
* Phát hiện các malware đã biết một cách chính xác với ít tài nguyên cần thiết
* Không có báo cáo false positive (cảnh báo sai)
* Nhanh chóng phát hiện sự xâm nhập

Phương pháp này cũng có nhược điểm hoặc hạn chế của nó. Chúng là:

* Yêu cầu cập nhật thường xuyên cho bộ dữ liệu chữ ký, nếu không, không có phát hiện hiệu quả.
* Không thể phát hiện các biến thể malware không xác định.
* Không thể phát hiện các virus biến đổi (virus đa hình, biến đổi và oligomorphic).
* Yêu cầu nhiều thời gian, nhân lực và tiền bạc để trích xuất chữ ký.
* Tạo chữ ký mất thời gian.
* Không hiệu quả nếu mã bị thay đổi một chút (được che khuất).

John (2017) trong một bài báo trực tuyến cho biết Báo cáo Bảo mật Cyber hàng năm của Cisco năm 2017 đã phát hiện ra rằng 95% các tệp malware mà họ đã phân tích không còn tồn tại trong vòng 24 giờ trước khi tiến hóa. Loại malware này được gọi là malware zero day và ví dụ là jigsaw và hddcryptor. Vì các malware đã được phát hiện bằng cách sử dụng kỹ thuật dựa trên chữ ký trong những năm đó, các nhà phát triển đã bắt đầu che giấu mã của họ. Không còn khả năng của các chương trình chống malware dựa trên chữ ký có thể phát hiện được mã malware đã được che giấu. Do đó, các nhà nghiên cứu đã phát triển một kỹ thuật mới để có thể phát hiện mã code đã được che giấu, đó là kỹ thuật phát hiện malware dựa trên hành vi.

## 2. Dựa trên Hành vi (Behaviour Based )

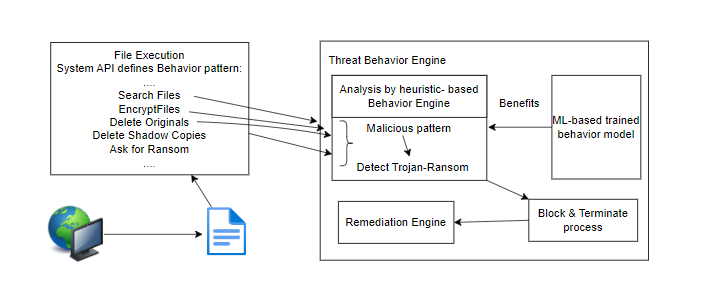
### **2.1 Tổng quan về kỹ thuật dựa trên hành vi:**

**Behaviour Based** là một phương pháp tiên tiến trong lĩnh vực an ninh mạng, nhằm mục đích phát hiện và bảo vệ chống lại hoạt động độc hại bằng cách giám sát hành vi của người dùng. Đây là một cách tiếp cận chủ động, theo dõi tất cả hoạt động liên quan để nhanh chóng xác định và xử lý bất kỳ sự lệch lạc nào khỏi mẫu hành vi bình thường. Loại an ninh này ngày càng trở nên quan trọng khi dấu chân số của các tổ chức mở rộng và việc quản lý an ninh mạng tập trung trở nên khó khăn hơn. Theo báo cáo tội phạm mạng của Symantec, các cuộc tấn công mạng trở nên hấp dẫn và có thể gây ra hậu quả nghiêm trọng hơn khi sự phụ thuộc vào công nghệ thông tin của chúng ta tăng lên.

Mckinsey khám phá ba xu hướng an ninh mạng mới nhất, bao gồm an ninh dựa trên hành vi và hậu quả của chúng đối với các tổ chức đối mặt với các rủi ro mạng mới và nổi lên. Nhìn về phía trước vào năm 2023, các chuyên gia dự đoán rằng kẻ tấn công sẽ chuyển sự chú ý từ các doanh nghiệp lớn sang các doanh nghiệp nhỏ hơn với ít tài nguyên hơn để bảo vệ. Các Howlers đã xác định tám xu hướng an ninh quan trọng cho năm 2022 bao gồm phát hiện mối đe dọa thông qua trí tuệ nhân tạo (AI), bảo vệ ứng dụng bản địa đám mây, triển khai kiến trúc không tin cậy (zero trust), tăng cường sử dụng phương pháp xác thực sinh trắc học như nhận diện khuôn mặt hoặc quét dấu vân tay, cải thiện khả năng phát hiện điểm cuối thông qua thuật toán học máy, tăng cường sử dụng công nghệ mã hóa như mật mã lượng tử hoặc hệ thống mã hóa đồng dạng, chú trọng hơn vào quy định bảo mật dữ liệu như tuân thủ GDPR hoặc CCPA, và cuối cùng là sự tăng cường trong giải pháp phản ứng tự động trước sự cố. Tất cả những xu hướng này đều chỉ ra một tương lai nơi an ninh dựa trên hành vi sẽ đóng một vai trò còn lớn hơn trong việc bảo vệ doanh nghiệp khỏi các tác nhân độc hại trực tuyến.

### ****2.2 Phân tích hành vi thực hiện:****

Behaviour Based là một loại công nghệ an ninh theo dõi hành vi người dùng và xác định bất kỳ hoạt động độc hại nào. Nó dựa trên ý tưởng rằng nếu người dùng có hành vi đáng ngờ hoặc bất thường, hệ thống có thể phát hiện và có hành động. Chẳng hạn, nếu người dùng cố gắng truy cập vào hệ thống mà họ không có quyền, hệ thống có thể phát hiện điều này và cảnh báo cơ quan thích hợp. Loại NIDS này sử dụng phân tích hành vi để xác định liệu có hoạt động đáng ngờ nào đã xảy ra. Nếu hành vi được phân tích đáp ứng các tiêu chí nhất định, thì một cảnh báo sẽ được kích hoạt.



Hình 3: Ảnh Behavior-based malware detection schema

Phân tích hành vi sử dụng phân tích dữ liệu lớn và trí tuệ nhân tạo trên dữ liệu hành vi người dùng để xác định mẫu, xu hướng, dị thường, hoặc các chỉ báo khác của hoạt động độc hại. Điều này cho phép phát hiện chính xác hơn so với các hệ thống phát hiện xâm nhập dựa trên chữ ký chỉ dựa vào việc xác định chữ ký tấn công đã biết để kích hoạt cảnh báo. Các giải pháp IDS dựa trên dị thường cũng có thể phát hiện các mối đe dọa không biết vì chúng không bị giới hạn bởi các chữ ký tấn công được định nghĩa trước như các hệ thống dựa trên chữ ký.

Một hệ thống phát hiện xâm nhập (IDS) theo dõi lưu lượng mạng cho hoạt động đáng ngờ và cảnh báo khi hoạt động như vậy được phát hiện. IDS là một trong những công cụ đầu tiên được sử dụng cho mục đích này và nó tiếp tục được sử dụng ngày nay như một phần của chiến lược an ninh của nhiều tổ chức do khả năng theo dõi lỗ hổng trong hệ thống và phân tích lưu lượng mạng để tìm kiếm dấu hiệu của ý định độc hại hoặc nỗ lực truy cập trái phép từ các nguồn bên ngoài như hacker hoặc chương trình malware cố gắng truy cập trái phép vào mạng hoặc hệ thống với thông tin nhạy cảm được lưu trữ bên trong chúng.

Đối với các nhà cung cấp dịch vụ quản lý (msps), việc hiểu những khác biệt chính giữa các hệ thống phát hiện xâm nhập dựa trên chữ ký và dựa trên dị thường có thể giúp họ bảo vệ mạng của khách hàng một cách tốt hơn trước các mối đe dọa tiềm tàng đồng thời cung cấp bảo vệ toàn diện hơn chống lại các cuộc tấn công không biết mà chưa được xác định bởi các giải pháp dựa trên chữ ký truyền thống. An ninh dựa trên hành vi cung cấp một lớp bảo vệ bổ sung chống lại các loại mối đe dọa này bằng cách theo dõi hành vi người dùng để xác định bất kỳ hoạt động bất thường nào có thể chỉ ra ý định độc hại trước khi thiệt hại xảy ra - cho phép msps yên tâm hơn khi bảo vệ mạng của khách hàng trước các cuộc tấn công mạng tiềm tàng.

**Ưu điểm của An ninh Dựa trên Hành vi:**

* **Khả năng Phát Hiện Hành Vi Độc Hại**: An ninh dựa trên hành vi theo dõi hành vi người dùng và phát hiện các hoạt động đáng ngờ hoặc bất thường, giúp phát hiện sớm các mối đe dọa.
* **Phản Ứng Nhanh Chóng**: Hệ thống tự động có khả năng cảnh báo hoặc chặn truy cập ngay lập tức khi phát hiện hoạt động đáng ngờ, giảm thiểu thiệt hại.
* **Hiệu Quả Cao**: So với các biện pháp an ninh thủ công, an ninh dựa trên hành vi tự động hiệu quả hơn, khó bị kẻ tấn công lách qua và phát hiện đe dọa một cách nhanh chóng và chính xác.
* **Bảo Vệ Toàn Diện**: Cung cấp sự bảo vệ rộng rãi cho toàn bộ mạng từ hành vi bất thường hoặc đáng ngờ.

**Nhược điểm của An ninh Dựa trên Hành vi:**

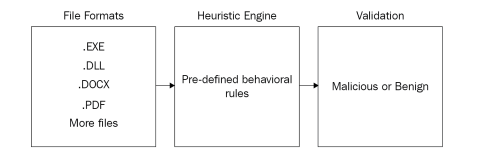
* **False Positives và False Negatives**: Có khả năng xảy ra lỗi khi hệ thống nhận dạng sai hoạt động lành tính là độc hại hoặc không phát hiện hoạt động độc hại.
* **Chi Phí Giám Sát và Bảo Dưỡng**: Đòi hỏi giám sát đáng kể và bảo dưỡng để duy trì hiệu quả, làm tăng chi phí triển khai và bảo trì.
* **Vulnerability to Attacks**: Do là hệ thống tự động, có thể bị tấn công như denial of service, làm giảm hiệu quả bảo vệ.

Trong khi an ninh dựa trên hành vi cung cấp một lớp bảo vệ chủ động bằng cách theo dõi hành vi người dùng, một phương pháp khác đóng vai trò quan trọng trong việc bảo vệ các hệ thống mạng và thông tin là **An ninh Dựa trên Phỏng đoán** (Heuristic Based Security). Phương pháp này sử dụng các thuật toán phỏng đoán để phân tích hành vi và các thuộc tính của tệp, nhằm xác định khả năng chứa mã độc hại. Tiếp theo, chúng ta sẽ khám phá sâu hơn về cách thức hoạt động và những lợi ích mà An ninh Dựa trên Phỏng đoán mang lại, cũng như cách nó bổ sung cho các kỹ thuật dựa trên hành vi để tạo nên một hệ thống an ninh mạng đa tầng và hiệu quả.

## 3. Quét Heuristic ( Heuristic Based)

### 3.1 Tổng quan về kỹ thuật quét Heuristic (Heuristic Based):

Kỹ thuật Heuristic trong phần mềm diệt virus ngày càng trở nên tiên tiến hơn. Loại động cơ này xác định điểm số cho mỗi tệp bằng cách tiến hành phân tích thống kê kết hợp giữa phương pháp động cơ tĩnh và động. Phát hiện dựa trên Heuristic là phương pháp dựa trên các quy tắc hành vi được định nghĩa trước để phát hiện hành vi có thể gây hại của các quá trình đang chạy.



Hình 4: Ảnh Antivirus heuristic engine illustration

Trong thập kỷ đầu của những năm 1990, với số lượng virus tăng lên vượt quá một vài trăm, chuyên gia diệt virus nghiên cứu khả năng phát hiện các virus chưa biết và không có chữ ký. Kết quả là, các trình phân tích heuristics được tạo ra.

*Các Phương Pháp Phân Tích Heuristic:*

* **Phân Tích Heuristic Tĩnh:**
  + Tìm kiếm chữ ký ngắn chung có tính đặc biệt cho hầu hết virus (các lệnh nghi ngờ).
  + Dùng để phát hiện các hoạt động như tìm kiếm file sử dụng mask \*.EXE và ghi mã vào file này.
* **Phân Tích Heuristic Động:**
  + Phát triển đồng thời với việc đưa máy ảo vào các chương trình diệt virus.
  + Mô phỏng hiệu suất chương trình và ghi lại tất cả các hoạt động nghi ngờ.

### 3.2 Phân tích hành vi thực hiện:

Phân tích heuristic có mặt trong hầu hết các giải pháp antivirus chính thống trên thị trường ngày nay. Tương tự như quét chữ ký, quét heuristic tìm kiếm mối đe dọa bằng cách tìm kiếm các chuỗi hoặc lệnh cụ thể không thường thấy trong một ứng dụng. Các lệnh tiềm năng gây hại này có thể thực hiện các chức năng như payload của trojan, cơ chế nhân bản của virus, hoặc mô hình phân phối của worm. Quá trình phân tích antivirus heuristic thường sử dụng hệ thống dựa trên quy tắc hoặc trọng số để xác định mức độ nguy hiểm của chức năng chương trình. Nếu các quy tắc vượt quá ngưỡng đã định trước, một cảnh báo được kích hoạt và hành động phòng ngừa được thực hiện.

Các công cụ phân tích antivirus dựa trên Heuristic sử dụng nhiều kỹ thuật quét khác nhau, bao gồm phân tích tệp, mô phỏng tệp (còn gọi là quét động hoặc thử nghiệm cát), và phát hiện chữ ký di truyền. Những kỹ thuật này có thể được sử dụng để phát hiện virus đang nằm yên trên bộ lưu trữ tệp hoặc di chuyển giữa hai điểm cuối.

Ví dụ khi Heuristic quét được các hành vi có thể là mã độc :

* + Là khi một quá trình cố gắng tương tác với quá trình LSASS.exe chứa băm NTLM, vé Kerberos của người dùng, và hơn thế nữa.
  + Khi một quá trình không được ký bởi nhà cung cấp uy tín cố gắng tự ghi vào một vị trí cố định.
  + Và nếu một quá trình mở một cổng lắng nghe và chờ nhận lệnh từ máy chủ điều khiển và chỉ huy (C2).

Ưu điểm:

* **Phát hiện Malware Mới và Biến Thể:** Heuristic có khả năng phát hiện các loại malware mới và các biến thể chưa được biết đến trước đây, giúp bảo vệ chống lại các mối đe dọa mới và phức tạp.
* **Phân tích Hành vi:** Kỹ thuật này phân tích hành vi của các tệp và chương trình, cho phép nó phát hiện các mối đe dọa dựa trên cách chúng hoạt động thay vì dựa trên chữ ký đã biết.
* **Chủ Động và Đáp Ứng Nhanh:** Heuristic chủ động phát hiện các mối đe dọa thay vì chỉ phản ứng với các chữ ký đã biết, giúp giảm thời gian phản ứng đối với các mối đe dọa mới.
* **Phát hiện Dựa trên Hành vi Động:** Các kỹ thuật như mô phỏng tệp (sandboxing) cho phép phân tích hành vi của tệp trong môi trường ảo, giúp phát hiện mối đe dọa mà không làm ảnh hưởng đến hệ thống thực.
* **Bảo vệ Dài Hạn:** Kỹ thuật này không phụ thuộc vào cơ sở dữ liệu chữ ký cập nhật, do đó nó cung cấp một lớp bảo vệ lâu dài hơn chống lại các mối đe dọa đang phát triển.

Nhược điểm:

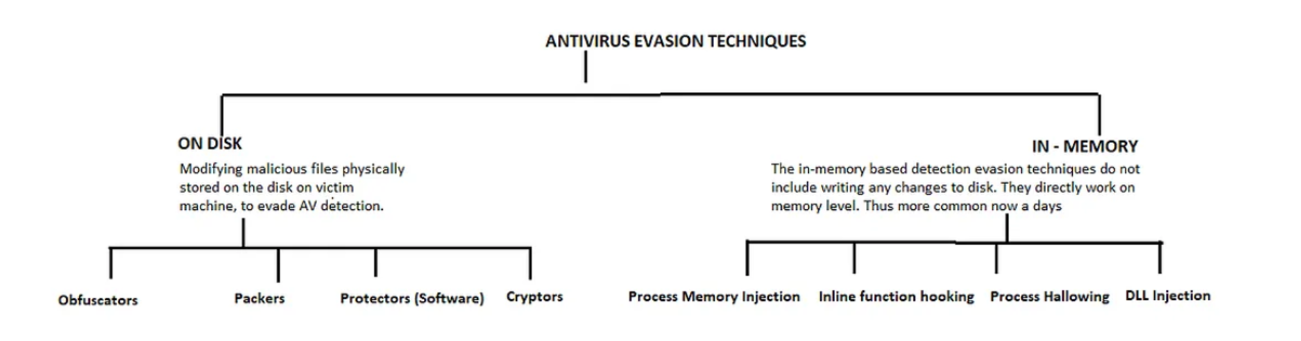
* **Dương Tính Giả Cao:** Do phụ thuộc vào hành vi phỏng đoán, heuristic có thể phát hiện nhầm các chương trình an toàn là mối đe dọa (dương tính giả), gây phiền toái và làm gián đoạn cho người dùng.
* **Phức Tạp trong Cấu Hình và Quản Lý:** Việc cài đặt và cấu hình phần mềm heuristic có thể phức tạp hơn so với các phương pháp truyền thống, đòi hỏi kiến thức kỹ thuật cao hơn từ người quản trị.
* **Tài Nguyên Hệ Thống:** Quét heuristic thường đòi hỏi nhiều tài nguyên hệ thống hơn, có thể làm chậm máy tính, đặc biệt là khi thực hiện phân tích hành vi động.
* **Khả năng Bị Bypass:** Mặc dù heuristic có thể phát hiện nhiều loại malware, nhưng các tác giả malware ngày càng tinh vi hơn trong việc phát triển các chiến thuật để tránh né phát hiện heuristic.
* **Khó Khăn trong Việc Phân Biệt:** Phân biệt giữa các hành vi bình thường và độc hại có thể khó khăn, đòi hỏi hệ thống phải có thuật toán phức tạp và thông minh để giảm thiểu dương tính giả.

Mặc dù quá trình phân tích và phát hiện heuristic có thể không hoàn hảo và đôi khi gây ra dương tính giả, nhưng phương pháp quét virus chủ động này có thể là cách hiệu quả để bổ sung cho các giải pháp quét chữ ký truyền thống. Phần mềm antivirus phân tích heuristic đang được cải thiện liên tục, đảm bảo quy trình hoạt động hiệu quả hơn và sử dụng tài nguyên máy tính tốt hơn. Đối với các tổ chức tìm kiếm sự bảo vệ tối ưu từ malware và virus đã biết và chưa biết, phân tích antivirus heuristic chắc chắn là một khoản đầu tư đáng giá.

## Kỹ thuật Bypass antivirus

Mã độc hiện đang ngày càng trở nên tinh vi và phức tạp, đặt ra thách thức lớn đối với các chương trình diệt virus (AV) trong việc nhận diện và ngăn chặn chúng. Để đối mặt với sự tiến bộ này, những kẽ hở trong các hệ thống an ninh ngày càng trở nên rộng lớn, tạo nên môi trường lý tưởng cho việc triển khai các kỹ thuật bypass AV. Trong bối cảnh này, việc hiểu rõ về những kỹ thuật này và mối tương quan của chúng đối với mã độc không chỉ là quan trọng mà còn là cực kỳ cần thiết để bảo vệ hiệu suất và an toàn của hệ thống.

Chúng em sẽ khám phá sâu hơn về các kỹ thuật bypass AV, tập trung vào hai khía cạnh chính: ON DISK và IN-MEMORY. Trên đĩa (ON DISK), chúng ta sẽ nói về Obfuscators, Packers, Protectors và Cryptors, trong khi IN-MEMORY sẽ bao gồm Process Memory Injection, DLL Injection, Inline Hooking và Process Hollowing. Mỗi kỹ thuật đều đóng góp vào cách mã độc có thể tránh được phát hiện, tạo ra môi trường nguy hiểm mà chương trình diệt virus phải đối mặt hàng ngày.



Hình 5: Biểu đồ phân loại các kỹ thuật Bypass AV

Với mỗi kỹ thuật này, chúng em xin phép được đào sâu vào từng kỹ thuật này dựa trên các nguồn tài liệu tham khảo và hơn hết, với mỗi 1 kỹ thuật, chúng em sẽ phân tích 1số mã độc tương ứng và giải thích cách mà tác giả sử dụng kỹ thuật đó.

## 4. Antivirus Evasion Techniques: On Disk

Nhắc lại 1 chút khái niệm về các kỹ thuật Bypass Antivirus trên ổ đĩa:

* **Obfuscators:**
  + **Mục Tiêu:** Che giấu mã nguồn của chương trình, làm cho mã trở nên khó đọc và hiểu.
  + **Cách Hoạt Động:** Sử dụng các biện pháp như thay đổi tên biến, hàm, và câu lệnh để làm rối mã nguồn.
  + **Ưu Điểm:** Khó khăn hóa việc phân tích động và tĩnh, làm chậm quá trình phân tích mã độc.
  + **Nhược Điểm:** Không thay đổi cấu trúc logic, chỉ tạo sự rối mã nguồn.
* **Packers:**
  + **Mục Tiêu:** Nén và đóng gói mã nguồn để giảm kích thước của tệp tin thực thi.
  + **Cách Hoạt Động:** Sử dụng thuật toán nén để giảm dung lượng tệp tin thực thi, có thể bao gồm cả mã độc.
  + **Ưu Điểm:** Giảm kích thước của tệp tin, giảm bề ngoài của mã độc.
  + **Nhược Điểm:** Chương trình chống virus có thể phát hiện sự đóng gói, đòi hỏi giai đoạn giải nén trước khi thực thi.
* **Protectors:**
  + **Mục Tiêu:** Bảo vệ mã nguồn khỏi sự phân tích và thay đổi.
  + **Cách Hoạt Động:** Sử dụng các kỹ thuật như mã hóa, chống gỡ bỏ, và giả mạo để bảo vệ mã nguồn.
  + **Ưu Điểm:** Tăng cường độ an toàn của mã nguồn trước sự kiểm tra của chương trình chống virus.
  + **Nhược Điểm:** Tăng khả năng phát hiện của các giải pháp an ninh nâng cao.
* **Cryptors:**
  + **Mục Tiêu:** Mã hóa mã độc để ngăn chặn việc phát hiện dựa trên chữ ký.
  + **Cách Hoạt Động:** Sử dụng thuật toán mã hóa để biến đổi mã nguồn thành dạng khó đọc.
  + **Ưu Điểm:** Ngăn chặn chương trình chống virus phát hiện dựa trên cơ sở chữ ký.
  + **Nhược Điểm:** Yêu cầu giai đoạn giải mã trước khi thực thi, có thể tăng thời gian khởi động.

Ở phần này, nhóm mình sẽ chia làm kĩ thuật Bypass AV On Disk làm 2 phần, phần 1 là về Obfucators và phần 2 là về 3 kĩ thuật còn lại: Packer, Protectors, Cryptors. Bởi lẽ Obfucators nó gần như là kĩ thuật cha của 3 kĩ thuật còn lại, việc chia như vậy, nó có thể sẽ dễ hiểu và hình dung hơn

#### 1.1 Obfuscators:

**Kỹ thuật Obfuscation** là hành động làm cho mã nguồn của một chương trình khó khám phá hoặc hiểu—cả đối với con người và máy tính—nhưng không làm thay đổi cách chương trình hoạt động. Mục tiêu không chỉ là làm cho một chương trình trở nên không đọc được mà còn là để hoàn toàn ẩn đi sự tồn tại của nó.

Packers, Cryptors, Protector cũng là một số trong những phương pháp obfuscation phổ biến nhất được sử dụng bởi các nhóm hacker . Chúng thường xuyên sử dụng nhiều phương pháp cùng một lúc để tránh qua nhiều loại công cụ an ninh mạng khác nhau tại điểm xâm nhập ban đầu.

Obfuscation có sức mạnh lớn đến mức hacker có thể sử dụng Cobalt Strike, một công cụ kiểm thử xâm nhập thương mại, để rút dữ liệu từ hàng ngàn công ty trên khắp thế giới trong một trong những vụ việc xâm nhập an ninh mạng lớn nhất của thế kỷ 21—vụ việc xâm nhập solarwinds vào năm 2020.Chỉ cần hacker ẩn malware trong một hình ảnh có nhãn dính không đáng chú ý (ví dụ, gracious\_truth.jpg, festive\_computer.jpg) và mã hóa nó bằng một mã hóa XOR đơn giản quay vòng. Một khi bên trong mạng mục tiêu, payload sau đó được kích hoạt để tải xuống và cài đặt các thành phần malware bổ sung.

Ví dụ các kỹ thuật Malware Obfuscation:

1. BINARY PADDING Mã rác được tạo ra bằng một hàm và lưu dưới dạng nhị phân để vượt quá giới hạn dung lượng tệp tối đa mặc định (thông thường là 25-200 MB) của các công cụ quét malware. Điều này ngăn chặn công cụ quét malware kiểm tra nó do rủi ro thời gian cao và thời gian chờ từ phía khách hàng.
2. SOFTWARE PACKING Một payload độc hại được đóng gói (nén) vào một tệp thực thi bằng cách sử dụng một công cụ đóng gói phổ biến như UPX. Điều này thay đổi kích thước và chữ ký của payload, cũng như làm phức tạp hóa bất kỳ nỗ lực phân tích nào ngược. Tệp thực thi cũng có thể được mã hóa để làm khó khăn hơn trong quá trình giải mã.
3. COMPILE AFTER DELIVERY Một phần ransomware được gửi dưới dạng mã không được biên dịch (mã nguồn) bằng cách sử dụng một email rác. Khi được kích hoạt bởi người dùng, nó triệu hồi một bộ biên dịch native như csc.exe để biên dịch payload trên thiết bị, đằng sau các phòng thủ biên giới như tường lửa. Ransomware thực thi và tiếp tục mã hóa tất cả các tệp trên ổ đĩa cứng của nạn nhân.

**Code obfucation** là một kỹ thuật được sử dụng để biến đổi một chương trình, làm trở ngại cho việc phân tích vì nó thay đổi mã nguồn nội bộ trong khi vẫn duy trì tính năng của chương trình. Có bốn loại mã hóa mã nguồn [7]: bố cục (layout), dữ liệu (data), luồng điều khiển (control flow), và ngăn chặn (preventive).

* Layout Obfuscation: Kỹ thuật này sửa đổi hoặc loại bỏ các yếu tố chi tiết không ảnh hưởng đến việc thực thi của chương trình. Nó chủ yếu tương ứng với một kỹ thuật làm khó khăn việc biến đổi tên biến hoặc loại bỏ các chú thích.
* Data Obfuscation: Kỹ thuật này biến đổi giá trị dữ liệu trong chương trình hoặc xây dựng lại cấu trúc dữ liệu. Nó chủ yếu tương ứng với một kỹ thuật biến đổi giá trị biến một cách phức tạp hoặc xây dựng lại cấu trúc mảng.
* Control Flow Obfuscation: Kỹ thuật này chèn mã nguồn giả mạo hoặc sửa đổi luồng điều khiển từ bên trong chương trình. Nó chủ yếu tương ứng với một kỹ thuật chèn các câu lệnh nhánh không có ý nghĩa hoặc biến đổi đồ thị luồng điều khiển.
* Preventive Obfuscation: Kỹ thuật này chèn các quy trình ngăn chặn phân tích hoặc mã ngăn chặn phân tích vào bên trong các chương trình. Nó chủ yếu tương ứng với một kỹ thuật ngăn chặn hoặc chấm dứt hoạt động của một công cụ phân tích (ví dụ: một bộ gỡ lỗi hoặc một bộ giải mã).

Trong thực tế, Preventive Obfuscation dựa trên cùng một khái niệm như các kỹ thuật chống phân tích – Tương ứng là các kĩ thuật Packers, Cryptor và Protectors.

Phát hiện và bảo vệ khỏi malware được obfuscate có thể là một công việc khó khăn vì nhiều kỹ thuật obfuscation lợi dụng các công cụ và tính năng tự nhiên như các thuộc tính thẻ HTML và bộ biên dịch tích hợp, không thể nhận biết được là độc hại vì lý do thực tế.

Khám phá mã nguồn độc hại đã được obfuscated là khó khăn. Đây là lý do tại sao việc áp dụng một phương pháp bảo vệ toàn diện bao gồm phân tích hành vi của tất cả các tài sản và người dùng trong môi trường của bạn là một cách tốt để đảm bảo rằng các ứng dụng hoạt động lạ lẫm được phát hiện, ngay cả khi mã nguồn độc hại không được bắt trước khi thực thi. Bắt đầu bằng cách triển khai nhiều loại phòng thủ, đặc biệt là ba trong tam giác quan sát SOC: Phát hiện và phản ứng trên mạng (NDR), phát hiện và phản ứng ở điểm cuối (EDR), và hệ thống quản lý thông tin và sự kiện an ninh (SIEM).

Hơn nữa, bạn nên tăng cường ý thức về cuộc tấn công lừa đảo trong tổ chức vì obfuscation thường được kết hợp với email lừa đảo và rác để đánh lừa người dùng tải xuống malware. Cuối cùng, hãy thực hiện các bước tích cực để chặn các phương tiện phương tiện giao thông phổ biến cho các cuộc tấn công obfuscate. Điều này bao gồm các tệp đính kèm được bảo vệ bằng mật khẩu, tệp ZIP chứa javascript và các tệp đính kèm có kích thước lớn bất thường.

#### 1.2 Packers, Protectors, Cryptors:

Payload, tức là phần malware thực sự mà nhóm đe dọa muốn chạy trên máytính của nạn nhân, được bảo vệ khỏi việc đảo ngược và phát hiện (bởi phần mềm bảo mật). Điều này được thực hiện bằng cách thêm mã không hoàn toàn độc hại, nhưng chỉ được thiết kế để che giấu mã độc hại. Vì vậy, mục tiêu là ẩn payload khỏi nạn nhân và khỏi những nhà nghiên cứu nắm giữ tập tin.

**a)Packer**

Thường ngắn gọn cho "runtime packers" còn được biết đến là "self-extracting archives". Phần mềm này tự giải nén trong bộ nhớ khi "tệp được đóng gói" được thực thi. Đôi khi kỹ thuật này cũng được gọi là "executable compression". Loại nén này được phát minh để làm cho các tệp nhỏ hơn. Vì vậy, người dùng không cần giải nén chúng thủ công trước khi chúng có thể được thực thi. Tuy nhiên, do kích thước hiện tại của phương tiện di động và tốc độ internet, nhu cầu về tệp nhỏ hơn không còn quá cấp thiết nữa. Do đó, khi bạn thấy một số packer được sử dụng ngày nay, nó hầu như luôn dành cho mục đích độc hại. Đơn giản là để làm cho việc đảo ngược khó khăn hơn, với lợi ích thêm vào là kích thước nhỏ hơn trên máy tính bị nhiễm.

**Packing**

Packing là một phần của kỹ thuật Obfuscators . Một packer là một công cụ thay đổi định dạng mã bằng cách nén hoặc mã hóa dữ liệu.

Mặc dù thường được sử dụng để trì hoãn việc phát hiện mã độc hại, nhưng vẫn có sử dụng hợp lệ cho việc packing. Một số sử dụng hợp lệ bao gồm bảo vệ tài sản trí tuệ hoặc dữ liệu nhạy cảm khác khỏi việc bị sao chép.

Một stub là một phần nhỏ của mã chứa giải mã hoặc giải nén được sử dụng để giải nén tệp đã được đóng gói.

Quá trình packing bao gồm:

1. Mã gốc được tải lên công cụ packer và trải qua quá trình packer để nén hoặc mã hóa dữ liệu.
2. Header thực thi di động gốc (PE header, bao gồm hình ảnh thực thi và các tệp đối tượng) và mã gốc được nén hoặc mã hóa và lưu trữ trong phần đã đóng gói của tệp thực thi mới.
3. Tệp đã đóng gói bao gồm:
   * PE header mới
   * Các phần đã đóng gói
   * Decompression stub — được sử dụng để giải nén mã
4. Trong quá trình packing, điểm nhập gốc được di chuyển/đánh lạc quan trong phần đã đóng gói. Điều này quan trọng đối với bất kỳ ai cố gắng phân tích mã. Quá trình này khiến việc xác định bảng địa chỉ nhập (IAT) và điểm nhập gốc trở nên khó khăn.
5. Decompression stub được sử dụng để giải nén mã khi giao hàng.

Một số tạo malware sử dụng packer tùy chỉnh, nhưng cũng có các packer thương mại/mã nguồn mở phổ biến như:

* UPX
* Themida
* The Enigma Protector
* Vmprotect
* Obsidium
* MPRESS
* Exe Packer 2.300
* Exestealth

**b)Crypter**

Là kỹ thuật lý tưởng nhất cho crypter thường được gọi là obfuscation.. Obfuscation cũng thường được sử dụng trong các script, như javascripts và vbscripts. Nhưng hầu hết thời gian này chúng không quá khó để tránh hoặc giải mã. Phương pháp phức tạp hơn sử dụng mã hóa thực sự. Hầu hết crypters không chỉ mã hóa tệp, mà phần mềm crypter còn cung cấp người dùng nhiều tùy chọn khác để làm cho tệp thực thi ẩn khó phát hiện nhất có thể đối với các nhà cung cấp bảo mật. Điều tương tự cũng đúng đối với một số packers. Một điều khác bạn sẽ được tìm hiểu là thuật ngữ FUD (Fully Undetectable), đó là mục tiêu tối thượng cho các tác giả malware. Có khả năng đi ngang qua mọi nhà cung cấp bảo mật là mục tiêu cuối cùng cho các tác giả malware. Nhưng nếu họ có thể tồn tại mà không bị phát hiện một thời gian và sau đó dễ dàng thay đổi các tệp của họ một khi chúng bị phát hiện, họ sẽ chấp nhận điều đó.

Quá trình của một cryptor liên quan chủ yếu đến việc mã hóa và giải mã dữ liệu để bảo vệ thông tin khỏi sự truy cập trái phép. Dưới đây là mô tả chi tiết về quá trình cryptor:

1. **Chuẩn bị Dữ Liệu:** Trước khi bắt đầu quá trình mã hóa, cryptor sẽ chuẩn bị dữ liệu cần được bảo vệ. Điều này có thể bao gồm việc chọn các tệp tin, thư mục hoặc thậm chí là toàn bộ ổ đĩa.
2. **Chọn Thuật Toán Mã Hóa:** Cryptor sẽ sử dụng một thuật toán mã hóa để biến đổi dữ liệu thành dạng không đọc được. Các thuật toán này có thể bao gồm AES, DES, RSA, hoặc các thuật toán khác tùy thuộc vào mức độ bảo mật mong muốn.
3. **Tạo Chìa Khóa:** Một chìa khóa được tạo ra để sử dụng trong quá trình mã hóa. Chìa khóa này có thể là một số ngẫu nhiên hoặc có thể dựa trên mật khẩu từ người dùng.
4. **Mã Hóa Dữ Liệu:** Cryptor sử dụng chìa khóa và thuật toán mã hóa để biến đổi dữ liệu thành dạng mã hóa. Quá trình này là quá trình chính của cryptor và tạo ra một phiên bản mã hóa của dữ liệu gốc.
5. **Lưu Trữ Dữ Liệu Mã Hóa:** Dữ liệu sau khi được mã hóa sẽ được lưu trữ tại một vị trí an toàn, thường là một tệp tin mới hoặc một vùng bộ nhớ được bảo vệ.
6. **Ghi Chìa Khóa hoặc Mã Thông Tin:** Cryptor có thể ghi chìa khóa hoặc mã thông tin liên quan vào một vị trí an toàn khác. Điều này là quan trọng để việc giải mã dữ liệu sau này có thể được thực hiện.
7. **Giải Mã Dữ Liệu (Tuỳ Chọn):** Nếu có yêu cầu, cryptor có thể cung cấp chức năng giải mã. Quá trình này đảo ngược quá trình mã hóa để khôi phục lại dữ liệu gốc.
8. **Kiểm Tra Quyền Truy Cập:** Cryptor thường đi kèm với cơ chế kiểm tra quyền truy cập để đảm bảo rằng chỉ những người có quyền mới có thể giải mã và truy cập vào dữ liệu.

Quá trình trên đều được thực hiện một cách an toàn và có tính bảo mật để đảm bảo rằng dữ liệu không thể dễ dàng bị truy cập hay đọc được bởi những người không có quyền. Cryptor chủ yếu được sử dụng để bảo vệ dữ liệu quan trọng, và việc chọn thuật toán và chìa khóa đúng là quyết định quan trọng trong quá trình này.

Công cụ sử dụng để triển khai cryptors:

* **Cryptlib:** Một thư viện mã nguồn mở cung cấp các chức năng mã hóa và bảo mật cho ứng dụng.
* **Crypto++ (cryptopp):** Một thư viện mã nguồn mở rất mạnh mẽ với nhiều thuật toán mã hóa và chức năng khác nhau.
* **Dịch vụ bên thứ 3:** Có rất nhiều dịch vụ, hay phần mềm mất phí được tạo ra nhằm mục đích mã hóa và được rao bán trên các cộng đồng nhiều người theo dõi.

**c)Protectors**

Một protector trong ngữ cảnh này là phần mềm được thiết kế để ngăn chặn sửa đổi và đảo ngược của các chương trình. Các phương pháp được sử dụng có thể, và thường sẽ, bao gồm cả việc đóng gói và mã hóa. Sự kết hợp đó cộng với một số tính năng bổ sung tạo thành điều được thường gọi là một protector. Vì vậy, một nhà nghiên cứu sẽ phải đối mặt với các lớp bảo vệ xung quanh payload, làm cho quá trình đảo ngược trở nên khó khăn.

Trước khi tìm hiểu các kĩ thuật Protectors, ta cần biết khái niệm về DBI:

**DBI** (DYNAMIC BINARY INSTRUMENTATION ) là một kỹ thuật chủ yếu được sử dụng cho phân tích động của các chương trình, và nó có thể được phân tích bằng cách chèn mã công cụ vào trong quá trình thực thi chương trình. Do đó, DBI được sử dụng để đo lường hiệu suất của chương trình, và các nhà phân tích có thể sử dụng các công cụ DBI để quan sát giá trị bộ nhớ và thanh ghi trong quá trình thực thi chương trình, phát hiện lỗi cấp phát bộ nhớ và thực hiện phân tích mối đe dọa về an ninh. Do những ưu điểm này, gần đây, nó cũng được sử dụng để phân tích phần mềm độc hại bằng DBI. Các công cụ như PIN [9], dynamorio [10] và Valgrind [11] thường cung cấp một cấu trúc để thực hiện DBI.

Ta có thể hiểu rằng, Protector là quá trình bảo vệ mã nguồn của phần mềm – mã độc, và nó có rất nhiều các kỹ thuật liên quan đến Protector malware, trong đó, phổ biến nhất, đó là ANTI- ANALYSIS, ANTI- VM, ANTI- DBI.

**ANTI- ANALYSIS**

ANTI- ANALYSIS là một kỹ thuật ngăn chặn hoặc can thiệp vào việc phân tích chương trình. Để làm cho việc phân tích chương trình khó khăn hơn, một chương trình mà đã áp dụng kỹ thuật chống phân tích có thể phát hiện môi trường phân tích hoặc công cụ phân tích và tự chấm dứt nó một cách bắt buộc. Các bảo vệ thương mại cũng cung cấp nhiều kỹ thuật chống phân tích đa dạng, trong đó có những kỹ thuật đại diện nhất như Anti Debug, Anti- VM (VM), Anti- patching và Anti- dumping.

* Anti- debug là một kỹ thuật ngăn chặn phân tích bằng cách sử dụng bộ gỡ lỗi, và kỹ thuật chống máy ảo đạt được điều này trong môi trường ảo.
* Anti- patching chủ yếu phát hiện xem một tệp đã được patch thông qua giá trị kiểm tra tổng hợp, và cuối cùng
* Anti- dumping phát hiện và bảo vệ khỏi các bản ghi bộ nhớ thực hiện bởi các công cụ dump.

**ANTI- VM**

Anti- VM là một kỹ thuật phát hiện môi trường máy ảo và chấm dứt chương trình để không thể phân tích trong môi trường máy ảo. Kỹ thuật này hiện đang được cung cấp bởi nhiều bảo vệ thương mại và được sử dụng phổ biến nhất để ngăn chặn phân tích trong một môi trường ảo. Các phương pháp phát hiện môi trường ảo được sử dụng hiện nay trong các bảo vệ thương mại có thể được phân loại thành ba loại cơ bản như sau:

- Dựa trên Registry: Khi Windows được cài đặt trong một môi trường ảo, registry được thiết lập như thông tin về máy ảo. Do đó, có sự khác biệt giữa giá trị registry của hệ điều hành khách và hệ điều hành chủ, dẫn đến việc có thể thực hiện phát hiện môi trường ảo bằng cách này.

- Dựa trên Hardware: Trong môi trường ảo, phần cứng được tách biệt logic thông qua hypervisor, không phải là phần cứng vật lý được sử dụng trong máy tính chủ. Do đó, có sự khác biệt giữa thông tin phần cứng của hệ điều hành khách và hệ điều hành chủ. Phát hiện môi trường ảo được thực hiện thông qua sự khác biệt giữa thông tin phần cứng của hệ điều hành khách và hệ điều hành chủ.

- Dựa trên Process/Service: Trong môi trường ảo, các chương trình cụ thể được cài đặt để sử dụng mỗi máy ảo một cách hiệu quả hơn. Do đó, có một số chương trình chỉ tồn tại trong hệ điều hành khách và không có trong hệ điều hành chủ. Ví dụ, phát hiện môi trường ảo được thực hiện bằng cách sử dụng một chương trình cụ thể chỉ tồn tại trong hệ điều hành khách.

Đến thời điểm này, có hạn chế nghiên cứu về kỹ thuật chống máy ảo và chỉ có một số nghiên cứu thực nghiệm giới hạn đã khám phá những kỹ thuật nào được sử dụng trong các bảo vệ thương mại thực tế và làm cách nào để vượt qua chúng.

**ANTI- DBI**

Các kỹ thuật chống DBI phát hiện tình trạng đang được phân tích bằng công cụ DBI và bắt buộc chương trình dừng lại, có nghĩa là không thể thực hiện phân tích động bằng công cụ DBI. Chống DBI thực hiện phát hiện công cụ DBI bằng cách sử dụng nguyên tắc hoặc đặc điểm của một công cụ DBI cụ thể nào đó. Hiện nay, các bảo vệ thương mại không cung cấp bất kỳ tùy chọn chống DBI độc lập nào. Trên thực tế, các kỹ thuật chống DBI được cung cấp bởi các bảo vệ thương mại đơn giản chỉ trùng với các kỹ thuật được cung cấp trong các tùy chọn chống gỡ lỗi. Đến nay, chỉ có vài trường hợp được nghiên cứu một cách có hệ thống để chặn các công cụ DBI. Các kỹ thuật chống DBI hiện đang được sử dụng trong các bảo vệ thương mại có thể được phân loại thành bốn loại cơ bản.

- Dựa trên Chi phí: Sự khác biệt giữa thời gian thực thi chung của chương trình và thời gian thực thi của một chương trình đang được phân tích bằng công cụ DBI là có thể phát hiện được công cụ DBI thông qua sự khác biệt trong thời gian thực thi của chương trình mục tiêu phân tích.

- Dựa trên Bộ biên dịch JIT: Khác với các chương trình thông thường, công cụ DBI sử dụng bộ biên dịch just-in-time (JIT) để patch và biên dịch một số hướng dẫn cụ thể trong chương trình đang được phân tích để thực hiện phân tích chương trình theo thời gian thực. Một công cụ DBI có thể được phát hiện thông qua việc patch và biên dịch các hướng dẫn trong bộ đệm công cụ DBI.

- Dựa trên API: Windows cung cấp các giao diện lập trình ứng dụng (API) chống gỡ lỗi. Một số API thực hiện chống gỡ lỗi có thể phát hiện không chỉ bộ gỡ lỗi mà còn các công cụ DBI, và chống DBI sử dụng những API này để phát hiện công cụ DBI.

- Dựa trên Ngoại lệ: Trong hầu hết các tình huống thông thường, chương trình thực hiện xử lý ngoại lệ khi ngoại lệ xảy ra. Trong quá trình phân tích sử dụng công cụ DBI, chương trình hoạt động mà không thực hiện xử lý ngoại lệ trong phần nơi ngoại lệ xảy ra. Do đó, nếu có một ngoại lệ xảy ra, một quy trình được chèn vào chương trình chung và mã hoạt động bình thường của chương trình được chèn vào phần xử lý ngoại lệ. Sau đó, một công cụ DBI có thể được phát hiện khi nó thực hiện chương trình mà không xử lý ngoại lệ.

## 5. Antivirus Evasion Techniques: In Memory

IN-MEMORY (Trong Bộ Nhớ) sẽ bao gồm các kỹ thuật như Tiêm Bộ Nhớ Quy Trình (Process Memory Injection), Tiêm DLL (DLL Injection), Inline Hooking và Đào Tạo Quy Trình (Process Hollowing). Mỗi kỹ thuật này đều đóng góp vào cách mã độc có thể tránh được phát hiện, tạo ra một môi trường nguy hiểm mà các chương trình diệt virus phải đối mặt hàng ngày.

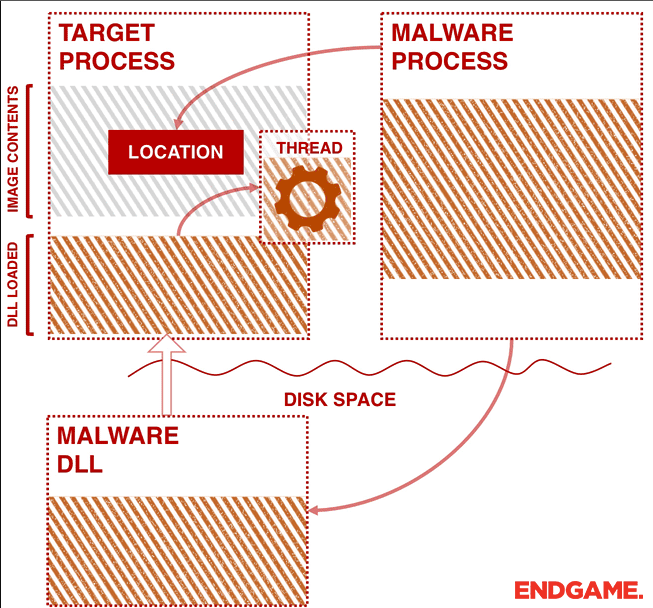
* **Tiêm Bộ Nhớ Quy Trình (Process Memory Injection):**
  + Mục Tiêu**:** Tiêm mã độc vào không gian bộ nhớ của một quy trình khác.
  + Ưu Điểm**:** Không tạo ra tệp tin thực thi mới, làm tăng khả năng tránh phát hiện.
  + Nhược Điểm**:** Có thể tạo ra các dấu vết trong không gian bộ nhớ.
* **Tiêm DLL (DLL Injection):**
  + Mục Tiêu**:** Tiêm mã độc dưới dạng DLL vào quy trình khác.
  + Ưu Điểm**:** Không tạo ra một tiến trình mới, khó phát hiện.
  + Nhược Điểm**:** Có thể tạo ra xung đột với các DLL khác trong quy trình.
* **Đào Tạo Quy Trình (Process Hollowing):**
  + Mục Tiêu**:** Thay thế nội dung của một quy trình khác bằng mã độc.
  + Ưu Điểm**:** Khó phát hiện, không tạo ra entitý mới.
  + Nhược Điểm**:** Có thể tạo ra các dấu vết, làm giảm sự ổn định của quy trình.
* **Inline Hooking:**
  + Mục Tiêu**:** Thay đổi luồng điều khiển của chương trình bằng cách chèn mã độc vào nơi chạy của nó.
  + Ưu Điểm**:** Không tạo ra entitý mới, khó phát hiện.
  + Nhược Điểm**:** Cần kiến thức sâu rộng về mã máy và có thể tạo ra xung đột.

Trước hết, ta cần hiểu 1 chút về Process Injection:

Process Injection (Tiêm quy trình) là một kỹ thuật Bypass AV phổ biến nhất thường được sử dụng trong mã độc và nghệ thuật fileless adversary tradecraft ( có thể hiểu là thay đổi thuộc tính, nội dung của file) , bao gồm việc chạy mã tùy chỉnh trong không gian địa chỉ của một quy trình khác. Việc tiêm quy trình cải thiện tính âm thầm, và một số kỹ thuật còn đạt được sự kiên trì. Mặc dù có nhiều kỹ thuật tiêm quy trình, trong bài viết này, nhóm em sẽ trình bày 3 kỹ thuật thường gặp trong thực tế để chạy mã độc thay mặt cho một quy trình khác. Nhóm em sẽ cung cấp ảnh chụp minh họa cho nhiều trong số những kỹ thuật này và các kỹ thuật tương tự có liên quan ( nhưng vẫn chủ yếu là DLL Injection, Inline Hooking và Process Hollowing) để hỗ trợ việc đảo ngược và phân tích mã độc, bao gồm cách các kỹ thuật này né tránh Antivirus và phát hiện, phòng thủ chống lại những kỹ thuật phổ biến này.

#### 2.1 CLASSIC DLL INJECTION VIA CREATEREMOTETHREAD AND LOADLIBRARY

Kỹ thuật này là một trong những kỹ thuật phổ biến nhất được sử dụng để tiêm mã độc vào một quy trình khác. Mã độc viết đường dẫn đến thư viện động-link (DLL) độc hại của nó vào không gian địa chỉ ảo của một quy trình khác và đảm bảo rằng quy trình từ xa sẽ tải nó bằng cách tạo một luồng từ xa trong quy trình mục tiêu.

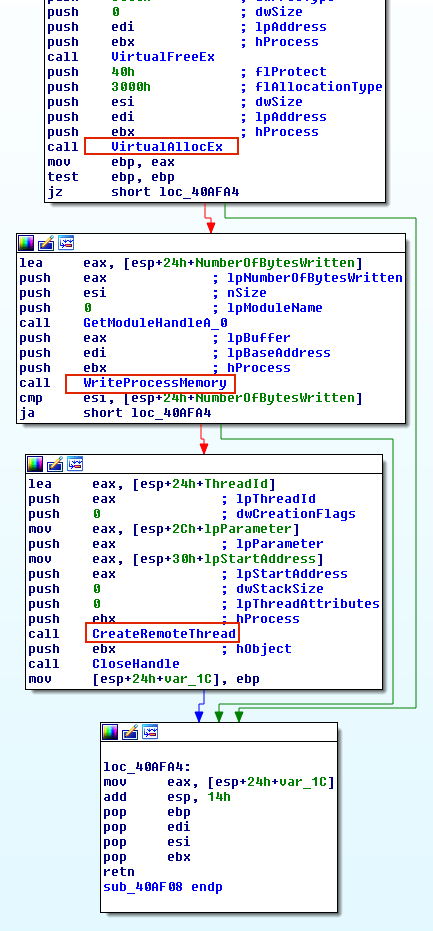


Hình 6: Ảnh Process Injection (1)

Ban đầu, mã độc cần chọn một quy trình để tiêm (ví dụ: svchost.exe). Điều này thường được thực hiện bằng cách tìm kiếm qua các quy trình thông qua việc gọi một bộ ba giao diện ứng dụng (apis): createtoolhelp32snapshot, Process32First và Process32Next. Createtoolhelp32snapshot là một API được sử dụng để liệt kê trạng thái heap hoặc module của một quy trình cụ thể hoặc tất cả các quy trình và nó trả về một bản chụp. Process32First trả về thông tin về quy trình đầu tiên trong bản chụp, sau đó Process32Next được sử dụng trong một vòng lặp để duyệt qua chúng. Sau khi tìm thấy quy trình mục tiêu, mã độc có được handle của quy trình mục tiêu bằng cách gọi openprocess.

Như thể hiện trong ảnh , mã độc gọi virtualallocex để có một không gian để viết đường dẫn đến DLL của mình. Sau đó, mã độc gọi writeprocessmemory để viết đường dẫn vào bộ nhớ được cấp phát. Cuối cùng, để có mã được thực thi trong một quy trình khác, mã độc gọi các apis như createremotethread, ntcreatethreadex hoặc rtlcreateuserthread. Hai cái cuối cùng là không được công bố. Tuy nhiên, ý tưởng chung là truyền địa chỉ của loadlibrary cho một trong những apis này để một quy trình từ xa phải thực thi DLL thay mặt cho mã độc.

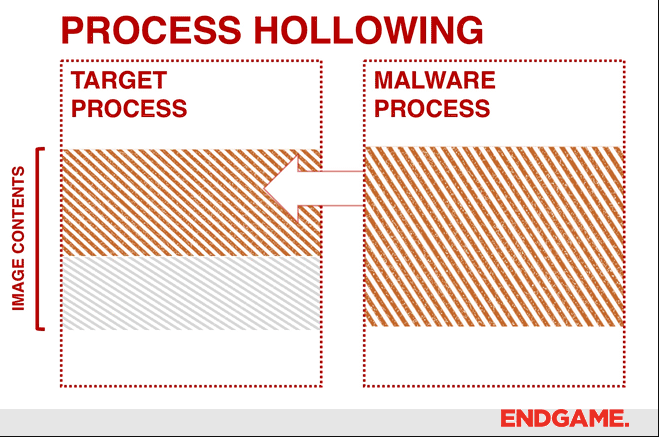
Createremotethread được theo dõi và đánh dấu bởi nhiều sản phẩm an ninh. Hơn nữa, nó yêu cầu có một DLL độc hại trên đĩa có thể được phát hiện. Xét đến việc những kẻ tấn công thường xuyên tiêm mã để tránh phòng thủ, những kẻ tấn công tinh vi có thể sẽ không sử dụng phương pháp này. Ảnh chụp màn hình dưới đây hiển thị một mã độc có tên là Rebhip thực hiện kỹ thuật này.



Hình 7: Ảnh Process Injection (2)

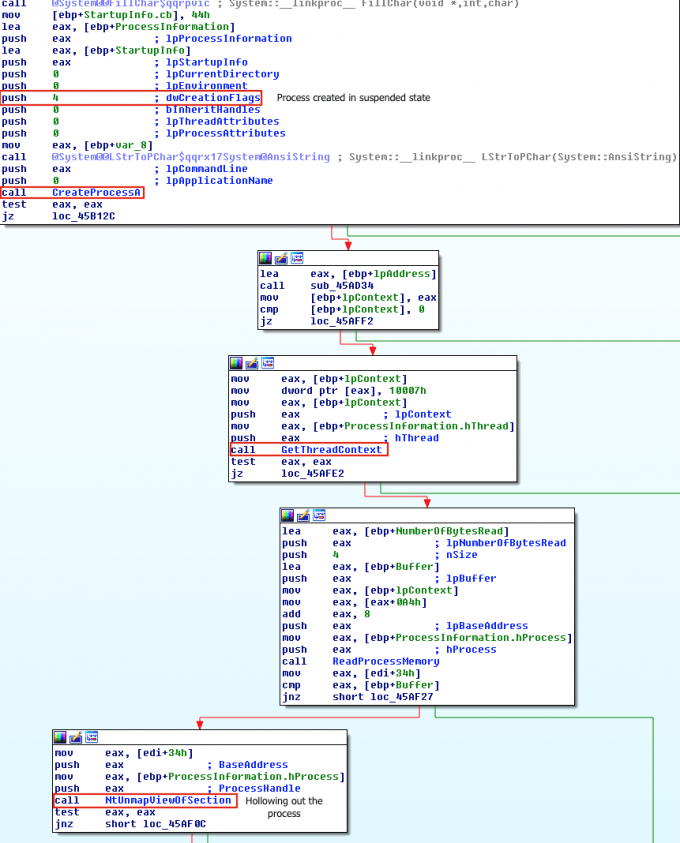
#### 2.2 PROCESS HOLLOWING (A.K.A PROCESS REPLACEMENT AND RUNPE )

Thay vì tiêm mã vào một chương trình máy chủ (ví dụ: tiêm DLL), mã độc có thể thực hiện một kỹ thuật được biết đến là "process hollowing" (đào tạo quy trình). Đào tạo quy trình xảy ra khi mã độc hủy bỏ bản mã chính thức khỏi bộ nhớ của quy trình mục tiêu và ghi đè lên không gian bộ nhớ của quy trình mục tiêu (ví dụ: svchost.exe) với một tệp thực thi độc hại.



Hình 8 : Ảnh Process Hollowing

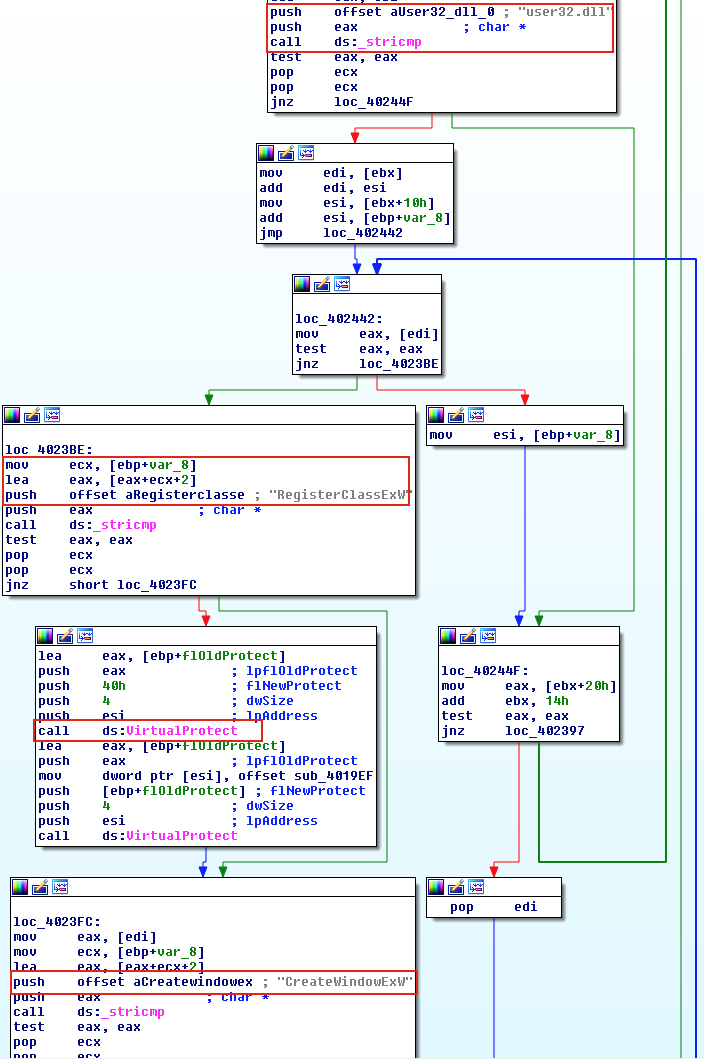
Mã độc đầu tiên tạo một quy trình mới để chứa mã độc trong trạng thái đình chỉ. Như thể hiện trong Hình 3, điều này được thực hiện bằng cách gọi createprocess và đặt Cờ Tạo Quy Trình thành CREATE\_SUSPENDED (0x00000004). Luồng chính của quy trình mới được tạo trong trạng thái đình chỉ và không chạy cho đến khi hàm resumethread được gọi. Tiếp theo, mã độc cần phải thay thế nội dung của tệp chính thức bằng dữ liệu độc hại của nó. Điều này được thực hiện bằng cách hủy bỏ ánh xạ bộ nhớ của quy trình mục tiêu thông qua việc gọi zwunmapviewofsection hoặc ntunmapviewofsection. Cả hai API này cơ bản là giải phóng tất cả bộ nhớ được chỉ định bởi một phần. Bây giờ khi bộ nhớ đã được hủy bỏ ánh xạ, trình tải thực hiện virtualallocex để cấp phát bộ nhớ mới cho mã độc và sử dụng writeprocessmemory để ghi từng phần của mã độc vào không gian của quy trình mục tiêu. Mã độc gọi setthreadcontext để trỏ đến điểm nhập cho một phần mã mới mà nó đã viết. Cuối cùng, mã độc khôi phục lại luồng đang đình chỉ bằng cách gọi resumethread để đưa quy trình ra khỏi trạng thái đình chỉ.



****Hình 9: Ảnh****  Ransom.Cryak performing process hollowing  
*****Sha256****:***eae72d803bf67df22526f50fc7ab84d838efb2865c27aef1a61592b1c520d144

#### 2.3 IAT HOOKING AND INLINE HOOKING (A.K.A USERLAND ROOTKITS)

IAT hooking và inline hooking thường được biết đến là các rootkit ở mức người dùng. IAT hooking là một kỹ thuật mà mã độc sử dụng để thay đổi bảng địa chỉ nhập. Khi một ứng dụng chính thức gọi một API nằm trong một DLL, hàm thay thế sẽ được thực thi thay vì hàm gốc. Ngược lại, với inline hooking, mã độc sửa đổi chính hàm API. Trong Hình 11, mã độc finfisher thực hiện IAT hooking bằng cách sửa đổi nơi createwindowex trỏ đến.



Hình 10: Ảnhfinfisher performing IAT hooking by changing where createwindowex points to  
****Sha256***:*** f827c92fbe832db3f09f47fe0dcaafd89b40c7064ab90833a1f418f2d1e75e8e

**Inline hooking**

**Inline hooking** là 1 kỹ thuật rất rất khó. Nó là một phương pháp để chặn cuộc gọi đến các hàm mục tiêu, chủ yếu được sử dụng bởi các phần mềm diệt virus, hộp cát và mã độc. Ý tưởng chung là chuyển hướng một hàm đến hàm của chúng ta, để chúng ta có thể thực hiện xử lý trước và/hoặc sau khi hàm thực hiện nhiệm vụ của mình; điều này có thể bao gồm: kiểm tra tham số, shimming, logging, giả mạo dữ liệu trả về và lọc cuộc gọi. Rootkit thường sử dụng hook để sửa đổi dữ liệu được trả về từ các cuộc gọi hệ thống để che giấu sự tồn tại của chúng, trong khi phần mềm bảo mật sử dụng chúng để ngăn chặn/giám sát các hoạt động có thể là độc hại.

Hook được đặt bằng cách trực tiếp sửa đổi mã trong hàm mục tiêu (sửa đổi trực tiếp), thường bằng cách ghi đè lên vài byte đầu tiên bằng một lệnh nhảy; điều này cho phép chuyển hướng thực thi trước khi hàm thực hiện bất kỳ xử lý nào. Hầu hết các engine hooking sử dụng một lệnh nhảy tương đối 32-bit (opcode 0xe9), chiếm 5 byte.

**Những vấn đề chúng ta phải đối mặt**

* Có thể cần gọi hàm gốc mà chúng ta đã ghi đè bằng lệnh nhảy.
* Điều kiện đua.
* Không phù hợp với gọi hàm.
* Đệ quy vô hạn.

**Làm thế nào nó hoạt động**

Chúng ta sẽ sử dụng một hook dựa trên trampoline, nó cho phép chúng ta chặn các hàm, đồng thời vẫn có thể gọi các hàm gốc (mà không cần gỡ bỏ chúng trước). Hook này gồm 3 phần:

1. **Hook –** Một nhảy tương đối 5 byte được viết vào hàm mục tiêu để hook nó, nhảy sẽ chuyển từ hàm đã được hook đến mã của chúng mình.
2. **Proxy –** Đây là hàm được chỉ định của chúng mình (hoặc mã) mà hook đặt trên hàm mục tiêu sẽ nhảy đến.
3. **Trampoline –** Được sử dụng để tránh hook để chúng mình có thể gọi một hàm đã được hook một cách bình thường.

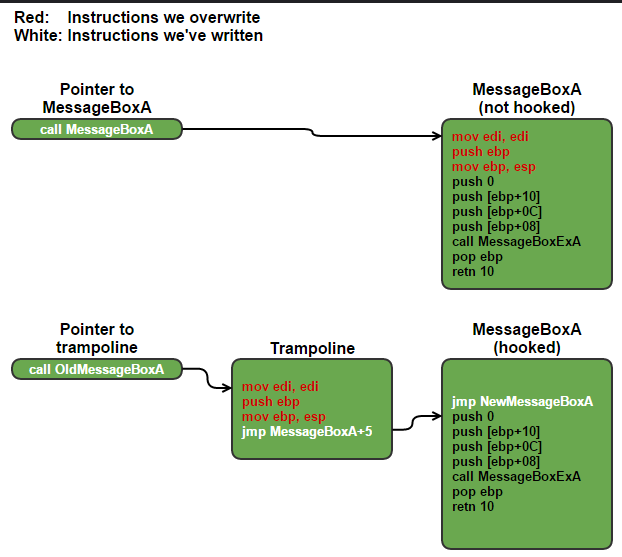
**Tại sao lại cần Trampoline?**

Giả sử chúng ta muốn hook messageboxa, in ra các tham số từ bên trong hàm proxy, sau đó hiển thị hộp thoại thông báo: Để hiển thị hộp thoại thông báo, chúng ta cần gọi messageboxa (mà chuyển hướng đến hàm proxy của chúng ta, rồi lại gọi messageboxa). Dĩ nhiên, việc gọi messageboxa từ bên trong hàm proxy của chúng ta sẽ chỉ gây ra đệ quy vô hạn và chương trình cuối cùng sẽ bị đánh văng do quá trình đệ quy vượt quá ngăn xếp.

Chúng ta có thể đơn giản là bỏ chồng messageboxa từ bên trong hàm proxy, gọi nó, sau đó bỏ chồng lại; nhưng nếu nhiều luồng đang gọi messageboxa cùng một lúc, điều này sẽ tạo ra một điều kiện đua và có thể làm cho chương trình bị đánh văng.

Thay vào đó, chúng ta có thể lưu trữ 5 byte đầu tiên của messageboxa (những byte này bị ghi đè bởi hook của chúng ta), sau đó khi cần gọi messageboxa không bị hook, chúng ta có thể thực hiện 5 byte đầu tiên, sau đó nhảy 5 byte vào messageboxa (ngay sau hook). Nếu bạn gặp khó khăn trong việc tưởng tượng điều này, chúng mình đã tạo một biểu đồ gọi cho dễ hình dung.

**Sử dụng trampoline để gọi một hàm đã bị hook**



Hình 11: Ảnh cuộc gọi bình thường đến messageboxa không bị hook, ở phía dưới là một cuộc gọi sử dụng trampoline của chúng ta để tránh hook.

- Đầu tiên là cuộc gọi bình thường đến messageboxa không bị hook, dưới cùng là cuộc gọi sử dụng trampoline của chúng tôi để tránh hook.

- Trong ví dụ, 5 byte đầu tiên của hàm tạo thành đúng 3 lệnh (mov edi, edi; push ebp; mov ebp, esp), tuy nhiên, nếu chẳng hạn, lệnh đầu tiên dài 10 byte và chúng ta chỉ lưu trữ 5 byte : trampoline sẽ cố gắng thực hiện nửa lệnh và chương trình sẽ phát nổ- crash. Để giải quyết vấn đề này, chúng ta phải sử dụng một trình dissasemble để biết độ dài của mỗi lệnh, vì vậy chúng ta có thể đảm bảo chỉ sao chép toàn bộ instruction. Trường hợp tốt nhất là *n* lệnh đầu tiên cộng lại chính xác 5 byte, trường hợp xấu nhất là nếu lệnh đầu tiên là 4 byte và lệnh thứ hai là 16 (độ dài tối đa của lệnh x86), chúng ta phải lưu trữ 20 byte (4 + 16), nghĩa là cái trampoline của chúng ta phải có kích thước 25 byte (không gian cho các lệnh có giá trị lên tới 20 byte và nhảy trở lại hàm nối 5 byte). Điều quan trọng cần lưu ý là bước nhảy quay lại phải nhảy đến hàm nối *n* byte, trong đó *n* là instructions mà chúng ta đã lưu trữ trong trampoline.

## Kết luận Chương II

Chương II đã cung cấp một cái nhìn đặc sắc và chi tiết về nền tảng của ba kỹ thuật chính trong lĩnh vực phần mềm antivirus: Dựa trên Chữ Ký (Signature-Based), Dựa trên Học (Behavior-Based), và Dựa trên Hành Vi (Heuristic-Based). Cùng với đó, chúng ta đã khám phá hai hướng tiếp cận quan trọng trong lĩnh vực bypass antivirus: ON DISK và IN-MEMORY.

Trên đĩa (ON DISK), chúng ta đã đi sâu vào Obfuscators, Packers, Protectors, và Cryptors, mỗi kỹ thuật đều mang đến những thách thức và đặc điểm riêng biệt. Obfuscators che giấu mã nguồn, Packers nén và đóng gói, Protectors bảo vệ mã nguồn, và Cryptors mã hóa chống lại phát hiện dựa trên chữ ký.

Trong khi đó, IN-MEMORY đã trình bày các kỹ thuật như Process Memory Injection, DLL Injection, Inline Hooking, và Process Hollowing, mỗi phương pháp tạo ra một môi trường nguy hiểm và phức tạp trong bộ nhớ máy tính.

Mỗi kỹ thuật đều đóng góp một vai trò quan trọng trong việc bảo vệ hệ thống khỏi các mối đe dọa mạng. Kỹ thuật dựa trên chữ ký vẫn là nền tảng cơ bản, trong khi các phương pháp dựa trên học và heuristic mang đến khả năng phòng ngừa đối với các mối đe dọa mới. Sự linh hoạt và tích hợp của chúng trong một giải pháp antivirus toàn diện là chìa khóa để đạt được mức độ bảo vệ cao nhất trước các mối đe dọa ngày càng phức tạp và đa dạng.

# CHƯƠNG III.THỰC NGHIỆM KỊCH BẢN ANTIVIRUS VÀ BYPASS AV.

Chương này sẽ tập trung vào việc thực hiện các kịch bản thực nghiệm để nắm vững các khái niệm cơ bản và nâng cao về an ninh mạng. Các kịch bản được thiết kế để đưa ra cái nhìn rõ ràng về quá trình xây dựng và phân tích chương trình chống vi-rút (antivirus) cũng như hiểu biết sâu sắc về cách malware có thể tránh được phát hiện (Bypass antivirus). Mỗi kịch bản đều mang lại kiến thức và kỹ năng quan trọng cho những người muốn theo đuổi lĩnh vực an toàn thông tin.

Trước hết, chúng ta sẽ vào tìm hiểu về cách 1 chương trình antivirus được tạo ra và cách mà nó hoạt động. Từ đó, ta có thể tạo dựng 1 chương trình đơn giản để Bypass antivirus dựa trên những gì mà chúng ta hiểu ( cụ thể ở đây sẽ là Window Defender). Tiếp đó, chúng ta sẽ vào sâu hơn về chức năng chính của 1 phần mềm antivirus phổ biến: Kaspersky, xem cách mà nó hoạt động và các chức năng, các hàm được sử dụng. Chưa hết, chúng ta cũng sẽ vào sâu hơn cái cách mà 1 phần mềm độc hại lâu đời đơn giản sử dụng kỹ thuật Bypass AV phổ biến cũng như cách mà 1 phần mềm độc hại mới – của 1 nhóm tấn công APT sử dụng hàng loạt các kỹ thuật Bypass antivirus. Từ đó đưa ra nhận xét.

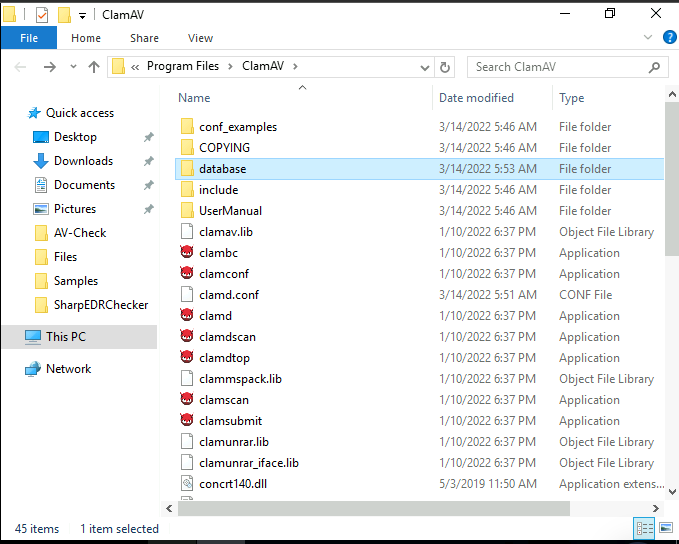
## 1. Kịch bản 1: Tạo dựng 1 chương trình antivirus và 1 chương trình Bypass antivirus.

**Kịch bản 1.1: Tạo 1 chương trình antivirus**

Trong nhiệm vụ này, chúng ta sẽ sử dụng phương pháp phát hiện dựa trên chữ ký để kiểm tra cách các sản phẩm antivirus nhận diện các tệp độc hại. Điều quan trọng cần lưu ý là phương pháp này chỉ hoạt động đối với các tệp độc hại đã biết có chữ ký được tạo sẵn trong cơ sở dữ liệu. Do đó, cơ sở dữ liệu cần được cập nhật thường xuyên.

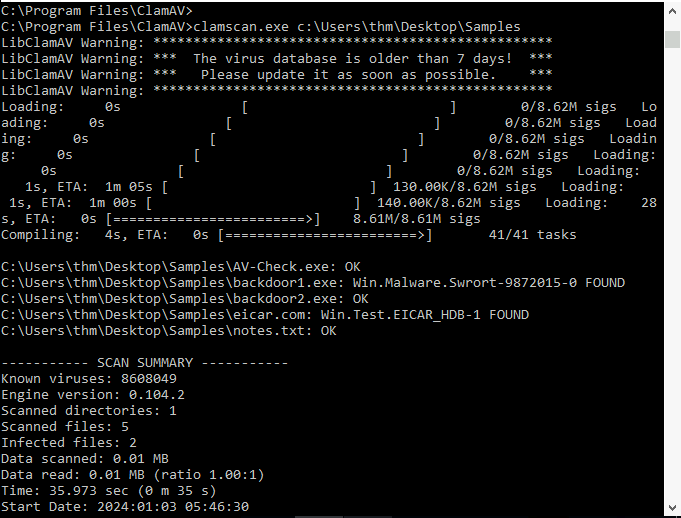
Chúng ta sẽ sử dụng phần mềm antivirus clamav để minh họa cách phát hiện dựa trên chữ ký xác định các tệp độc hại.. Chúng ta cũng sẽ quét một số mẫu malware, có thể được tìm thấy trên desktop. Thư mục Mẫu Malware chứa các tệp sau:

* EICAR là một tệp kiểm thử chứa chuỗi ASCII được sử dụng để kiểm thử hiệu suất của phần mềm antivirus thay vì sử dụng phần mềm độc hại thực tế có thể làm hại máy của bạn. Để biết thêm thông tin, bạn có thể truy cập trang web chính thức của EICAR, ở đây.
* Backdoor 1 là một chương trình C# sử dụng một kỹ thuật nổi tiếng để thiết lập một kết nối ngược, bao gồm việc tạo một tiến trình và thực thi shellcode của Metasploit Framework.
* Backdoor 2 là một chương trình C# sử dụng phương thức tiêm và mã hóa quy trình để thiết lập một reverse shell, bao gồm việc tiêm shellcode của Metasploit vào một quy trình đang chạy.
* AV-Check là một chương trình C# đếm số phần mềm antivirus trên máy mục tiêu. Lưu ý rằng tệp này không phải là độc hại
* Notes.txt là một tệp văn bản chứa một dòng lệnh. Lưu ý rằng tệp này không phải là độc hại.



Hình 12: Ảnh Clamav và database của nó

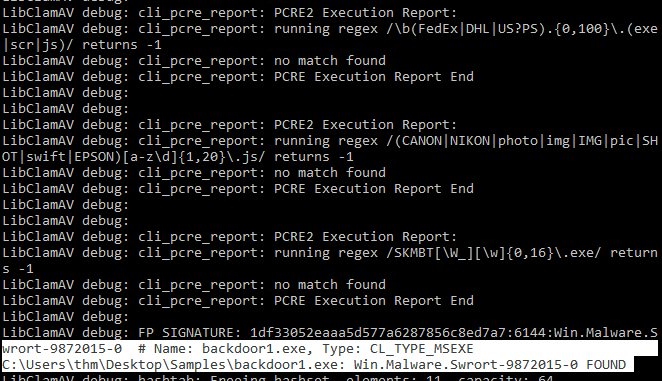
Clamav đi kèm với cơ sở dữ liệu của mình, và trong quá trình cài đặt, chúng ta cần tải xuống phiên bản được cập nhật gần nhất. Sử dụng nó để kiểm tra thử hiệu suất phát hiện của nó.



Hình 14: Ảnh khi sử dụng clamav để scan Folder

Kết quả trên cho thấy phần mềm clamav đã phân tích đúng và đánh dấu hai trong số các tệp thử nghiệm (EICAR, backdoor1, AV-Check và notes.txt) là độc hại. Tuy nhiên, nó đã không đúng lựa chọn backdoor2 là không độc hại trong khi nó thực sự là mã độc

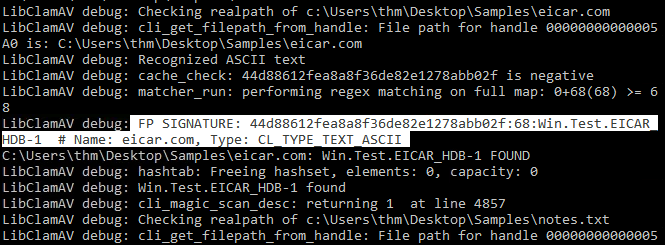
Để rõ hơn mà cách phần mềm này thực hiện, chúng ta có thể chạy lệnh **clamscan.exe --debug <file\_cần\_quét>**, và chúng ta sẽ thấy tất cả các mô-đun được tải và sử dụng trong quá trình quét. Ví dụ, nó sử dụng phương pháp giải nén để chia nhỏ các tệp và tìm kiếm một chuỗi byte code độc hại được xác định trước trong cơ sở dữ liệu, và đó là cách nó có thể phát hiện backdoor 1 viết bằng C#. Giá trị byte code của Metasploit shellcode được sử dụng trong backdoor 1 đã được xác định trước và thêm vào cơ sở dữ liệu của clamav.



Hình 15: ảnh backdoor1 đã bị xác định và thêm vào csdl của clamav

Tuy nhiên, backdoor2 sử dụng một kỹ thuật mã hóa (XOR) cho Metasploit shellcode, dẫn đến các chuỗi giá trị byte code khác mà nó không tìm thấy trong cơ sở dữ liệu của clamav.

Trong khi clamav có thể phát hiện tệp thử nghiệm [**EICAR.COM**](http://eicar.com/) là độc hại bằng kỹ thuật chữ ký dựa trên md5.

. 

Hình 16: Ảnh eicar.com bị phát hiện có trong cơ sở dữ liệu của clamav

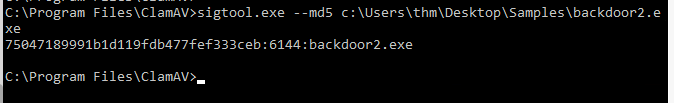
**Tạo Signature Based**

Một trong những tính năng của clamav là tạo cơ sở dữ liệu chữ ký riêng, cho phép ta tạo dữ liệu mới bao gồm các mục không có trong cơ sở dữ liệu chính thức của clamav. Dưới đây là các bước cần thiết:

1. Tạo một chữ ký MD5 cho tệp.
2. Thêm chữ ký đã tạo vào một cơ sở dữ liệu với phần mở rộng ".hdb".
3. Quét lại clamav đối với tệp sử dụng cơ sở dữ liệu mới của chúng ta.

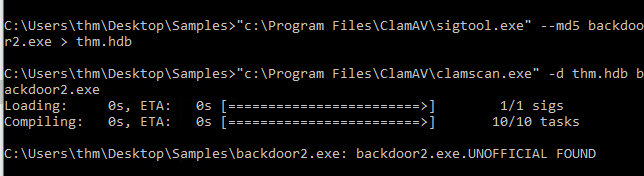
Đầu tiên, chúng ta sẽ sử dụng công cụ sigtool, được bao gồm trong bộ công cụ clamav, để tạo một băm MD5 cho backdoor2.exe bằng đối số --md5.

*Tạo MD5 hash*

 Hình 17: Ảnh sử dụng sigtool.exe để hash file

Như được hiển thị trong đầu ra, chuỗi hash đã tạo bao gồm cấu trúc sau: Hash:Kích-thước-trong-byte:Tên-tệp. Lưu ý rằng clamav sử dụng giá trị được tạo ra trong quá trình so sánh trong quá trình quét.

Bây giờ chúng ta đã có giá trị băm MD5, hãy tạo cơ sở dữ liệu riêng của chúng ta. Chúng ta sẽ sử dụng công cụ sigtool và lưu đầu ra vào một tệp sử dụng > thm.hdb như sau:



Hình 18 : Ảnh tạo dữ liệu và quét lại backdoor2.exe

Kết quả là, một tệp thm.hdb sẽ được tạo ra trong thư mục hiện tại thực thi lệnh. Chúng ta đã biết rằng clamav không phát hiện backdoor2.exe bằng cơ sở dữ liệu chính thứ. Nhưng khi đã cập nhập và quét lại và sử dụng cơ sở dữ liệu chúng ta đã tạo, thm.hdb; Ta thấy nó đã đánh dấu backdoor2.exe khi sử dụng cơ sở dữ liệu mới

**Quy tắc Yara**

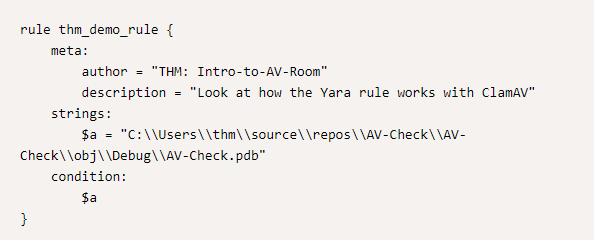
Một trong những công cụ hỗ trợ phát hiện tĩnh là Yara. Yara là một công cụ cho phép kỹ sư malware phân loại và phát hiện malware. Yara sử dụng phương pháp phát hiện dựa trên luật, vì vậy để phát hiện malware mới, chúng ta cần tạo một luật mới. Clamav cũng có thể xử lý các luật Yara để phát hiện các tệp độc hại. Luật sẽ giống như trong cơ sở dữ liệu của chúng ta ở phần trước.

Để tạo một luật, chúng ta cần kiểm tra và phân tích malware; dựa trên những điều kiện đó, chúng ta viết một luật. Hãy lấy AV-Check.exe làm ví dụ và viết một luật cho nó.

Đầu tiên, hãy phân tích tệp và liệt kê tất cả các chuỗi có thể đọc được của con người trong tệp nhị phân bằng công cụ strings. Kết quả sẽ hiển thị tất cả các hàm, biến và chuỗi vô nghĩa. Tuy nhiên, nếu bạn nhìn kỹ, chúng ta có thể sử dụng một số chuỗi duy nhất trong luật của mình để phát hiện tệp này trong tương lai. AV-Check sử dụng cơ sở dữ liệu chương trình (.pdb), chứa thông tin gỡ lỗi kiểu và ký hiệu của chương trình trong quá trình biên dịch.

C:\Users\thm\Desktop\Samples>strings AV-Check.exe | findstr pdb C:\Users\thm\source\repos\AV-Check\AV-Check\obj\Debug\AV-Check.pdb

Chúng ta sẽ sử dụng đường dẫn trong đầu ra của lệnh trước đó làm ví dụ về chuỗi duy nhất trong luật Yara mà chúng ta sẽ tạo. Chữ ký có thể là các khóa Registry, lệnh, v.v. Dưới đây là luật Yara mà chúng ta sẽ sử dụng trong việc phát hiện:

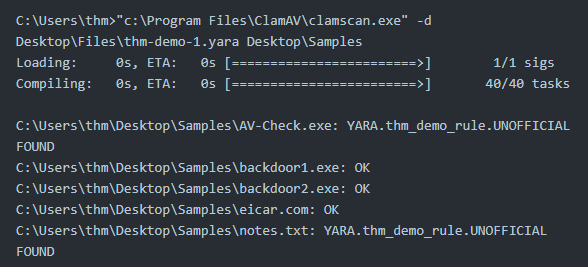


Hình 19: Ảnh demo rule Yara

Hãy giải thích một chút về luật Yara này:

* Luật bắt đầu bằng rule thm\_demo\_rule, đó là tên của luật chúng ta. Clamav sử dụng tên này nếu một luật khớp.
* Phần siêu dữ liệu, chứa thông tin tổng quan, bao gồm tác giả và mô tả, mà người sử dụng có thể điền.
* Phần chuỗi chứa các chuỗi hoặc byte-code mà chúng ta đang tìm kiếm. Trong trường hợp này, chúng tôi đang sử dụng đường dẫn cơ sở dữ liệu của chương trình C#. Lưu ý rằng chúng tôi thêm một \ phụ để tránh ký tự đặc biệt, để nó không làm hỏng luật.
* Trong phần điều kiện, chúng ta chỉ định nếu chuỗi được xác định được tìm thấy trong phần chuỗi, thì đánh dấu tệp.

Lưu ý rằng luật Yara phải được lưu trong một tệp có phần mở rộng .yara để clamav có thể xử lý nó. Hãy quét lại thư mục c:\Users\thm\Desktop\Samples bằng luật Yara chúng ta đã tạo.

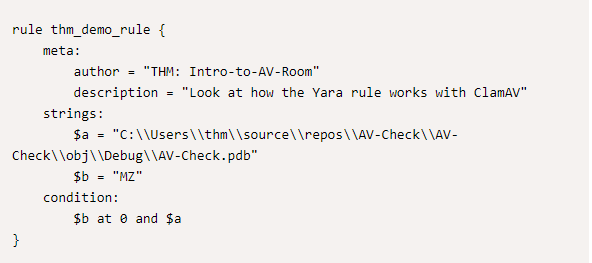


Hình 20: Ảnh sử dụng Yara Rule để quét

Kết quả là, clamav có thể phát hiện tệp nhị phân AV-Check.exe là độc hại dựa trên luật Yara mà chúng ta cung cấp. Tuy nhiên, clamav đưa ra một kết quả giả mạo khi đánh dấu tệp notes.txt là độc hại. Nếu chúng ta mở tệp notes.txt, chúng ta có thể thấy rằng văn bản chứa cùng một đường dẫn mà chúng ta đã chỉ định trong luật.

Hãy cải thiện luật Yara của chúng ta để giảm kết quả giả mạo. Chúng ta sẽ chỉ định loại tệp trong luật của mình. Thường, loại tệp có thể được xác định bằng các magic number, là hai byte đầu tiên của nhị phân. Ví dụ, các tệp thực thi (.exe) luôn bắt đầu bằng giá trị ASCII "MZ" hoặc "4D 5A" ở hệ cơ số 16.

Biết điều này sẽ giúp cải thiện quá trình phát hiện, hãy bao gồm nó trong luật Yara của chúng ta để chỉ đánh dấu tệp .exe chứa chuỗi chữ ký của chúng ta là độc hại. Dưới đây là luật Yara đã được cải thiện:

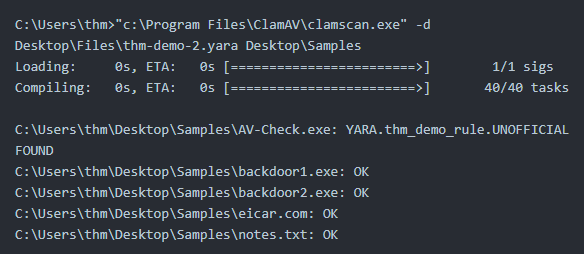


Hình 21: Ảnh sau khi cải thiện Yara Rule

Trong luật Yara mới, chúng ta đã định nghĩa một chuỗi duy nhất ($b) bằng giá trị MZ như một định danh cho loại tệp .exe. Chúng ta cũng đã cập nhật phần điều kiện, bao gồm các điều kiện sau đây:

* Nếu chuỗi "MZ" được tìm thấy tại vị trí 0, đầu tệp.
* Nếu chuỗi đặc biệt (đường dẫn) xuất hiện trong nhị phân.

Trong phần điều kiện, chúng ta đã sử dụng toán tử AND cho cả hai định nghĩa trong 1 và 2 được tìm thấy, sau đó chúng ta có một khớp. Tiến hành kiểm tra lại:



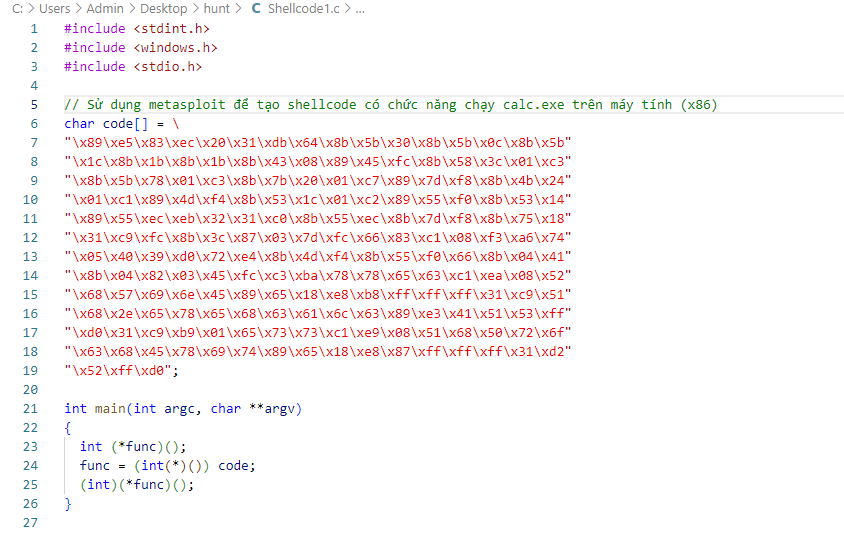
Hình 22: Ảnh sau khi sử dụng lại Yara Rule mới

Kết quả cho thấy chúng ta đã cải thiện luật Yara để giảm thiểu số lượng kết quả dương tính sai. Đây chỉ là một ví dụ đơn giản về cách phần mềm chống virus hoạt động. Do đó, các nhà cung cấp phần mềm antivirus nỗ lực để chống lại phần mềm độc hại và cải thiện sản phẩm cũng như cơ sở dữ liệu của họ để tăng cường hiệu suất và độ chính xác của kết quả.

Nhược điểm của phương pháp phát hiện dựa trên chữ ký là các tệp sẽ có giá trị băm khác nhau nếu tệp nhị phân bị sửa đổi. Do đó, dễ dàng bị Bypass dựa trên chữ ký nếu họ biết phần mềm chống virus tìm kiếm gì và cách phân tích các tệp nhị phân. Đây cũng chính là tiền đề để thực hiện kịch bản 1.2

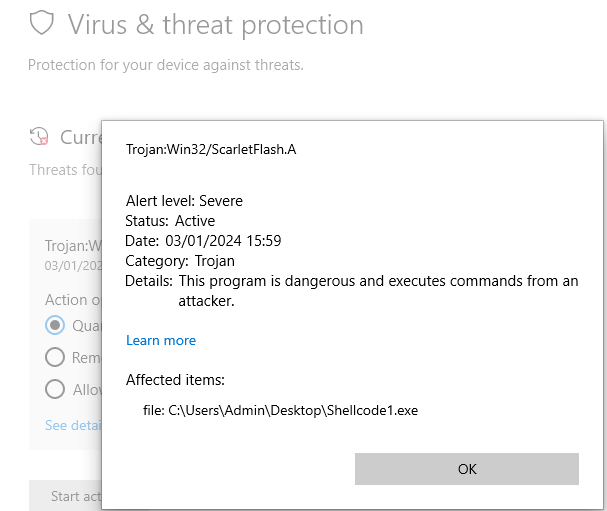
**Kịch bản 1. 2: Tạo 1 chương trình Bypass antivirus**

Dựa theo những gì nhóm em tìm hiểu được, để Bypass antivirus của Window Defender không khó. Tại sao lại như vậy ? Window Defender nó không hề mạnh – phát triển tốt trình antivirus, theo quan điểm cá nhân, 90% độ mạnh antivirus của Window Defender có được là dựa trên Signature –Based, 1 thứ rất dễ để có thể Bypass thông qua việc sử dụng Obfucators Code, để chứng minh điều này là đúng, chúng em đã tạo ra 1 mã nguồn đã được Obfucators và chưa được Obfucators để quan sát xem Window Defender có phát hiện ra được không. Dưới đây là ảnh mã nguồn chưa Obfucators:



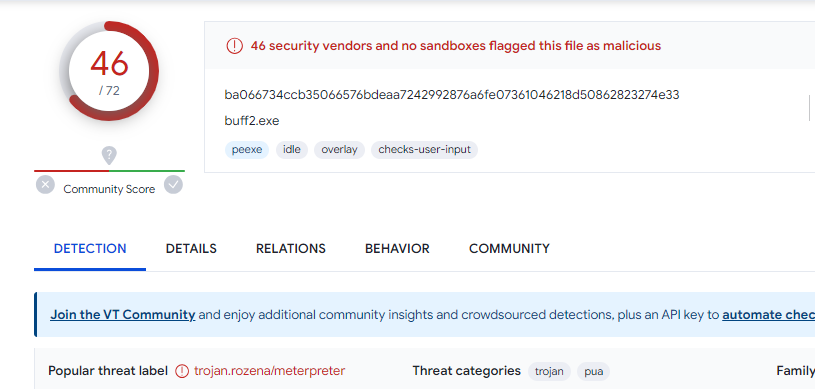
Hình 23: Ảnh mã nguồn khi shell code chưa sử dụng Obfucators

Khi biên dịch chương trình này, ta có thể thấy được Window Defender đã phát hiện và đánh dấu là mã độc.



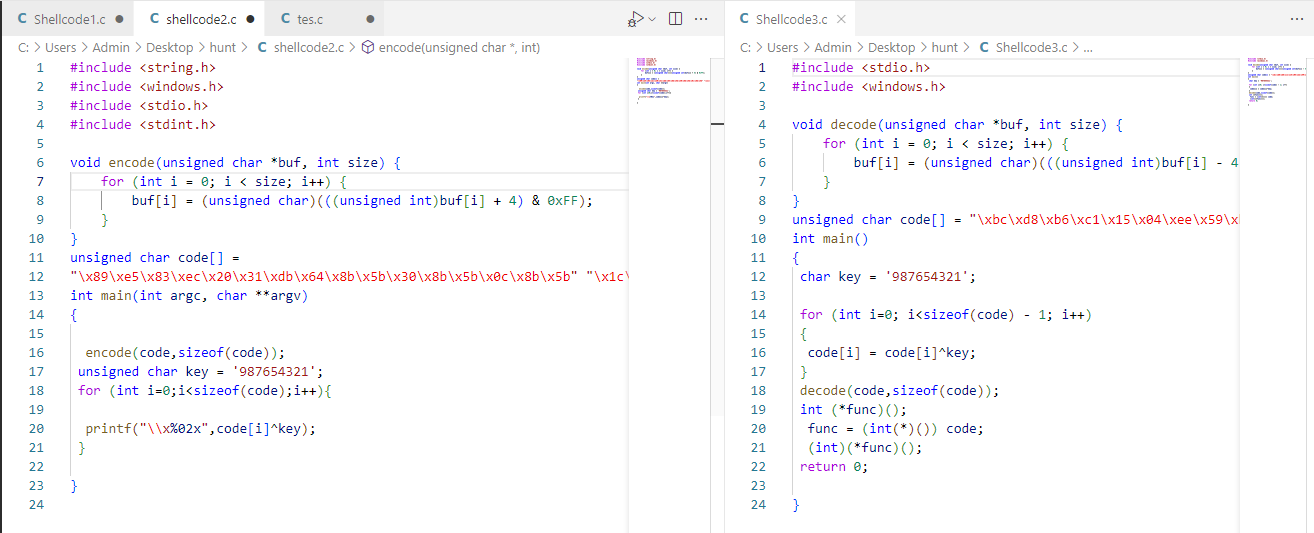
Hình 24: Ảnh Window Defender phát hiện ra shellcode1.exe là mã độc

Việc đặt trực tiếp Msfvenom Shellcode vào một tệp nhị phân sẽ dẫn đến 46/72 nhà cung cấp phát hiện mẫu này là độc hại. Điều này không có gì đáng ngạc nhiên vì đây là phần mềm nguồn mở, nhiều chữ ký đã được phát triển cho nó.



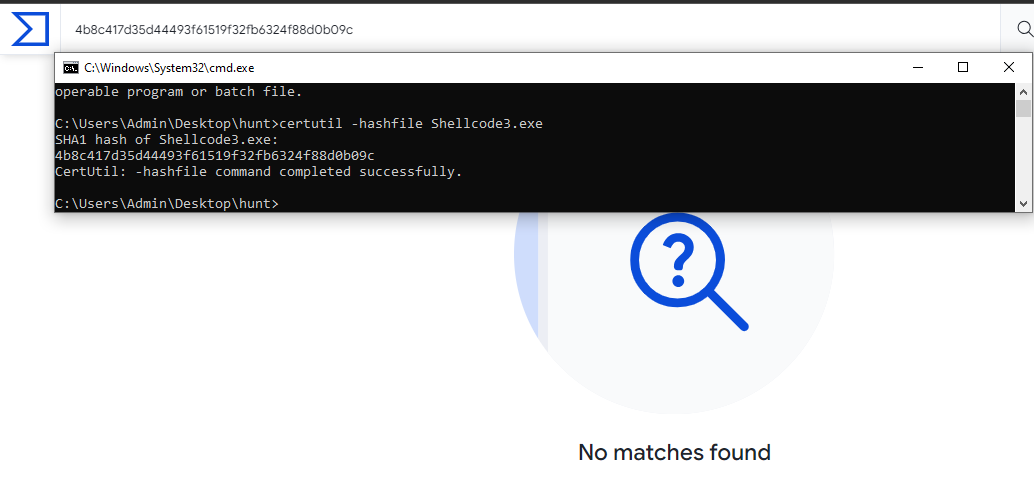
Hình 25: Shellcode1.exe bị virustotal phát hiện

Vẫn là shell code đó, điều gì sẽ xảy ra nếu chúng ta Obfucators Code bằng cách dùng Ceasar + Xor ?



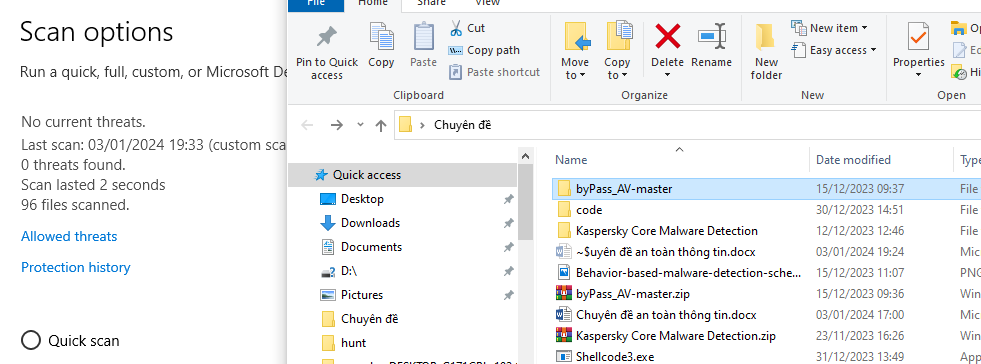
Hình 26: Ảnh mã hóa shell code và ảnh giải mã, thực thi shellcode

Hình trên là hình ảnh mã hóa shellcode ( Shellcode2.c ) sử dụng thuật toán Ceasar với 4 , và sử dụng thuật toán Xor với key là 987654321 để tiến hành mã hóa. Sau đó, tạo 1 chương trình giải mã tương ứng shellcode đó và thực thi. Mục đích chính ở đây là để thay đổi signature ban đầu của shellcode, tránh bị đánh dấu, nhằm kiểm thử hệ thống Window Defender như đã nói ban đầu.



Hình 27: Ảnh check hash và sử dụng virustotal để kiếm tra

Ta có thể thấy, hash của file Shellcode3.exe không có trên virustotal. Tiếp đến, sử dụng Window Defender để kiếm tra file.



Hình 28: Ảnh sử dụng Window Defender để quét folder chứa shellcode3.exe

Như vậy, ta có thể thấy rằng Window Defender không thể detect ra shellcodde đã bị mã hóa.

**Kết Quả Thử Nghiệm:**

1. **Shellcode1.exe - Mã Nguyên Gốc:**
   * Windows Defender phát hiện và đánh dấu là mã độc.
2. **Shellcode3.exe - Mã Đã Được Obfuscated:**
   * Hash của file không xuất hiện trên Virustotal, chỉ ra rằng thay đổi chữ ký đã giúp tránh phát hiện.
   * Windows Defender không phát hiện được mã độc trong trường hợp này.

**Nhận Xét:**

Dựa vào quan điểm của nhóm em và những thử nghiệm thực hiện, nhóm em đã chứng minh rằng việc Bypass antivirus của Windows Defender không khó nếu chúng ta sử dụng kỹ thuật obfuscation code. Windows Defender, theo quan điểm cá nhân của nhóm em , đa phần dựa vào phương pháp Dựa trên Chữ Ký (Signature-Based), và đây là điểm yếu có thể bị tận dụng.

## 2. Kịch bản 2: Phân tích Signature-Based -Kaspersky Core Malware

**Mô tả kịch bản**: Phân tích hành động cách hoạt động về hàm Signature- Based của Kaspersky Core Malware và các chức năng liên quan

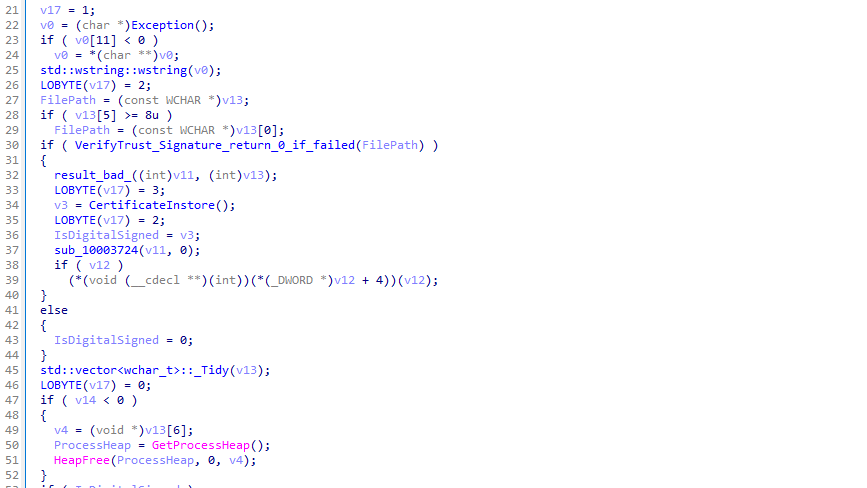
\*Lưu ý: Việc phân tích, tìm hiểu về Signature Based của Kaspersky Core Malware có khuynh hướng chủ quan của người viết, có thế bài phân tích sẽ sai 1 số chỗ, chúng mình không chịu trách nhiệm các vấn đề có liên quan.

**Mục tiêu :** Hiểu được cách chức năng Check Signature của Kaspersky hoạt động nói chung, cũng như các phần mềm antivirus khác nói riêng, từ đó đưa ra nhận xét và kết luận.

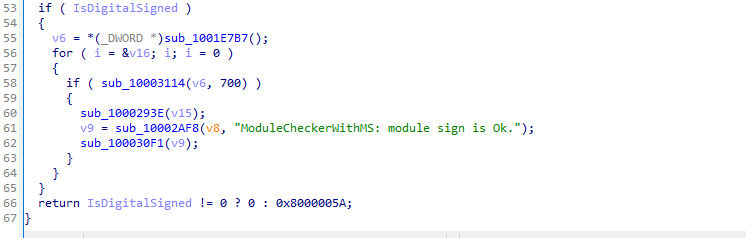
**Tiến hành kịch bản**: Tiến hành phân tích avpmain.dll: nơi này là nơi xử lý chính của core, liên quan tới import các dll khác và export các chức năng của các dll đó: **runloader.**

Chúng ta sẽ bỏ qua các dll phụ và tiến hành tìm 1 số chức năng chính của thư viện avpmain.dll này.

*a) Check Digital Signature Validity Of File:*



Hình 29: Ảnh raw source Signature Based Kaspersky Core Malware (1)



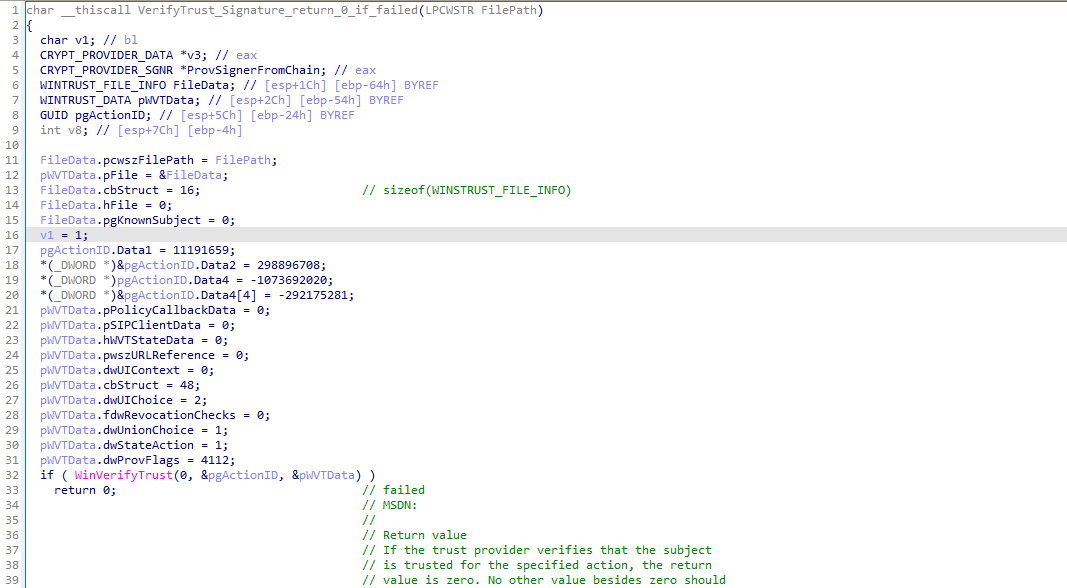
Hình 30: Ảnh raw source Signature Based Kaspersky Core Malware (2)

Đây là hàm xử lý về kiểm tra chữ ký số, cũng như tính hợp lệ chữ ký số của file. Ở trong hàm này, chúng ta cần lưu ý 2 hàm sau:

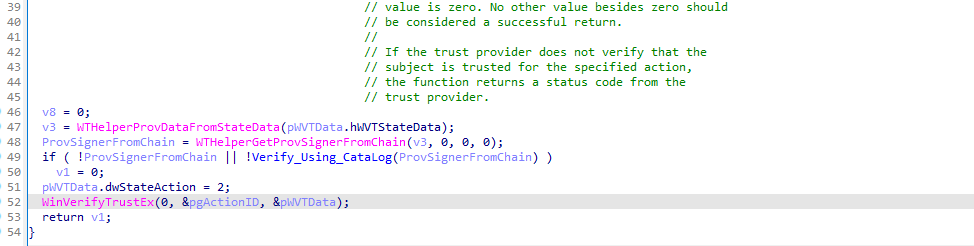
* Hàm **verifytrust\_Signature\_return\_0\_if\_failed(filepath)** để kiểm tra tính hợp lệ của chữ ký số trên tệp.
* Nếu chữ ký số không hợp lệ, hàm **certificateinstore** sẽ được gọi để kiểm tra, sau đó, nó sẽ trả về kết quả.

Ở các kết quả được kiểm tra, nếu nó có giá trị bằng 0 => file đã được ký, và ngược lại.

*b) verifytrust\_Signature\_return\_0\_if\_failed:*



Hình 31: Ảnh raw source Signature Based Kaspersky Core Malware (3)



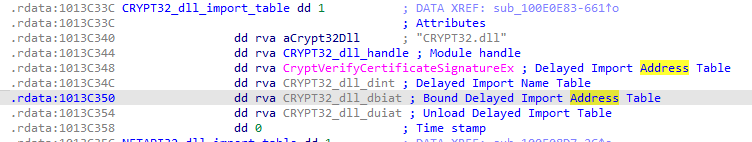
Hình 32: Ảnh raw source Signature Based Kaspersky Core Malware (4)

Hàm này import 1 thư viện : **WINTRUST.dll** và thực hiện các chức năng trong đó:

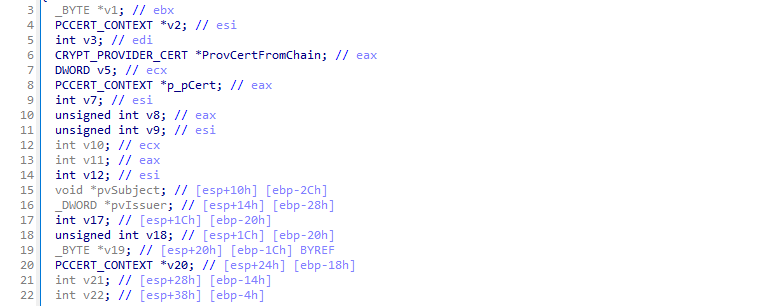
1. Nhận đường dẫn đến tệp cần kiểm tra chữ ký số thông qua tham số **filepath**.
2. Chuẩn bị dữ liệu cho quá trình xác thực chữ ký số bằng cách sử dụng các cấu trúc **WINTRUST\_FILE\_INFO** và **WINTRUST\_DATA**.
3. Thiết lập các thông số cho quá trình xác thực chữ ký số, bao gồm sự chọn lựa giao diện người dùng (**dwuichoice**), hành động trạng thái (**dwstateaction**), và các cờ khác (**dwprovflags**).
4. Sử dụng **winverifytrust** để xác minh chữ ký số trên tệp. Nếu quá trình này không thành công, hàm trả về giá trị 0, đại diện cho việc xác thực không thành công.
5. Nếu xác thực thành công, hàm tiếp tục kiểm tra sự tin cậy sử dụng **wthelperprovdatafromstatedata** và **wthelpergetprovsignerfromchain**.
6. Nếu không tìm thấy thông tin về nhà cung cấp chữ ký hoặc không thành công khi kiểm tra với danh mục (**Verify\_Using\_Catalog**), hàm trả về 0, đại diện cho việc xác thực không thành công.
7. Thực hiện bước tiếp theo của quá trình xác thực bằng cách gọi **winverifytrustex**.

*c) Certificate In Store :*

Kaspersky thực hiện import: **crypt32.dll** để thực hiện quá trình kiểm tra xác thực Certificate



Hình 33: Ảnh raw source Signature Based Kaspersky Core Malware (5)



Hình 34: Ảnh raw source Signature Based Kaspersky Core Malware (6)

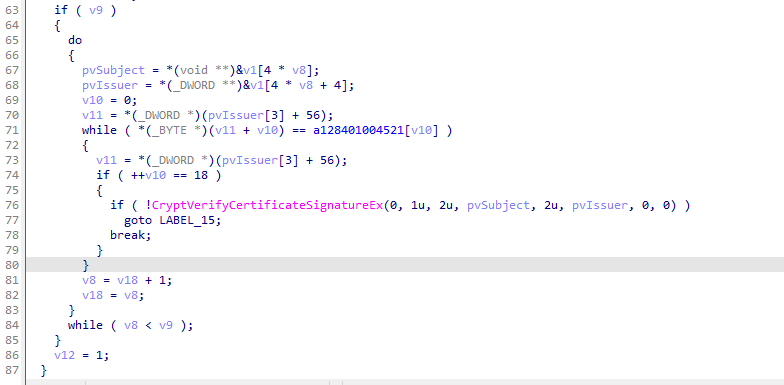
Ở đây cần lưu ý 2 biến :

**CRYPT\_PROVIDER\_SGNR \*this**: Con trỏ đến một cấu trúc

**CRYPT\_PROVIDER\_SGNR**: Đại diện cho một người ký (signer) trong chuỗi chứng thực.



Hình 35: Ảnh raw source Signature Based Kaspersky Core Malware (7)



Hình 36: Ảnh raw source Signature Based Kaspersky Core Malware (8)

Hàm này lấy danh sách các chứng chỉ từ người ký (signer) trong chuỗi chứng thực (**CRYPT\_PROVIDER\_SGNR**).

Lấy danh sách chứng chỉ từ người ký trong chuỗi chứng thực sử dụng **wthelpergetprovcertfromchain**.

Sau đó, với mỗi chứng chỉ, hàm kiểm tra tính hợp lệ của chữ ký số của nó thông qua hàm **cryptverifycertificatesignatureex**.

**Nhận Xét:**

Kịch bản phân tích Signature-Based của Kaspersky Core Malware tập trung vào việc nghiên cứu và phân tích cơ chế hoạt động của hệ thống chữ ký số. Qua việc xem xét mã nguồn raw, chúng ta đã xác định được một số hàm quan trọng liên quan đến quá trình kiểm tra tính hợp lệ của chữ ký số trên các tệp tin.

1. **Hàm verifytrust\_Signature\_return\_0\_if\_failed:**
   * Hàm này đảm nhận việc nhận diện đường dẫn của tệp tin cần kiểm tra chữ ký số.
   * Sử dụng thư viện WINTRUST.dll để thực hiện quá trình xác thực chữ ký số.
   * Trả về giá trị 0 nếu quá trình xác thực không thành công.
   * Nếu xác thực thành công, tiếp tục kiểm tra sự tin cậy và thực hiện các bước khác của quá trình xác thực.
2. **Hàm certificateinstore:**
   * Sử dụng thư viện crypt32.dll để kiểm tra xác thực chứng chỉ.
   * Lấy danh sách chứng chỉ từ người ký trong chuỗi chứng thực.
   * Kiểm tra tính hợp lệ của chữ ký số cho từng chứng chỉ sử dụng hàm cryptverifycertificatesignatureex.

**Kết Luận:**

Kết quả phân tích cho thấy hệ thống của Kaspersky Core Malware sử dụng cơ chế chữ ký số để đảm bảo tính toàn vẹn và nguồn gốc của các tệp tin. Quá trình xác thực này bao gồm việc sử dụng thư viện WINTRUST.dll và crypt32.dll để kiểm tra chữ ký số và chứng chỉ từ người ký, đồng thời kiểm tra với danh mục để đảm bảo tính tin cậy của chúng.

Phân tích này giúp hiểu rõ hơn về cách Kaspersky Core Malware thực hiện quá trình kiểm tra chữ ký số và đảm bảo sự tin cậy của các tệp tin. Nó là một phần quan trọng để hiểu cách các phần mềm chống virus hoạt động khi sử dụng phương pháp Signature-Based. Tuy nhiên, cũng cần lưu ý rằng mô hình này có nhược điểm, nhất là khi đối mặt với các mối đe dọa mới và tiến triển mà chưa có trong cơ sở dữ liệu chữ ký.

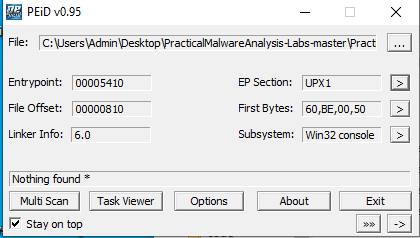
## 3. Kịch bản 3: Phân tích mã độc sử dụng pack bằng phần mềm UPX

**Mô tả:** Phân tích về 1 file có tên Lab01-2.exe ( Trong lab Praticall malware) sử dụng kỹ thuật pack bằng phần mềm UPX.

**Mục tiêu**:Hiểu sâu về cách mà mã độc sử dụng UPX để Anti Analysis, đồng thời hiểu được cách sử dụng các công cụ phân tích phần mềm, cách để unpack được phần mềm mã độc đã pack.

**Phân tích:** Trước hết, ta có thể sử dụng các công cụ quan sát file để đánh giá xem file có bị pack hay không. Công cụ packing nhúng stub vào trong tệp thực thi trong quá trình đóng gói. Vì vậy, nếu bạn có thể xác định công cụ được sử dụng để đóng gói mã, bạn có thể sử dụng cùng một công cụ để trích xuất tệp gốc.

Cách tốt nhất (và nhanh nhất) để giải nén mã malware đã được đóng gói là sử dụng một công cụ. Exeinfo PE là một công cụ như vậy, sẽ phân tích mã để xác định liệu nó đã được đóng gói hay không. Nó thường cũng có thể xác định công cụ đóng gói được sử dụng, hoặc ta cũng có thể sử dụng Detect It Easy.



Hình 37 : Ảnh mã độc sử dụng UPX1 để nén mã

UPX là một công cụ đóng gói thường được sử dụng và bao gồm tính năng giải nén. Nếu malware đã được đóng gói bằng UPX, bạn có thể sử dụng dòng lệnh trong công cụ để giải nén mã độc hại và phân tích thêm với một công cụ kỹ thuật đảo ngược. Một ví dụ về dòng lệnh dựa trên một tệp có tên là packed.exe:

Bashcopy code:

Upx -d -o unpacked.exe packed.exe

Chạy lệnh đó sẽ giải nén tệp packed.exe và tạo một tệp mới có tên là unpacked.exe. Sau đó, bạn có thể đặt tệp malware đã được giải nén vào một trình gỡ lỗi như Ollydbg để thực hiện phân tích tiếp theo.

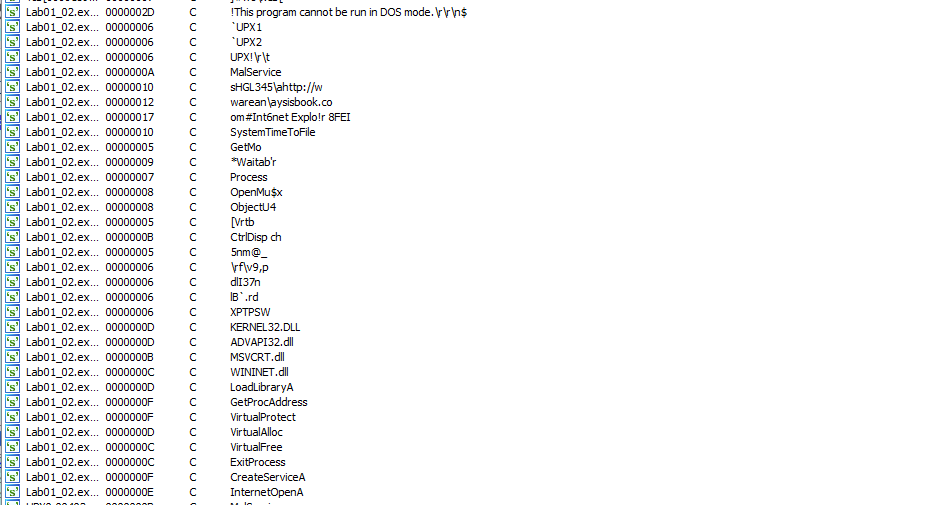
Có cách thủ công để phân tích mã độc hại. Một cách để bắt đầu quá trình phân tích mã độc hại là chạy lệnh strings để phân tích các chuỗi liên quan đến mã độc hại. Tuy nhiên, trong mã đã đóng gói, không có chuỗi có thể xác định được.

Ngoài ra, như đã nói trước đó, điểm nhập gốc được giấu trong tệp đã đóng gói. Biết điểm nhập gốc là quan trọng với bất kỳ nhà phân tích nào cố gắng phân tích mã. Đó là cách họ sẽ có thể khôi phục mã gốc. Bảng địa chỉ nhập được sử dụng bởi các chương trình đang chạy để tham chiếu đến các chức năng mà nó cần sử dụng để chạy đúng cách. Trong các tệp đã đóng gói, thông tin IAT bị che dấu làm cho quá trình dỡ xuống trở nên khó khăn. Điều này là thách thức khác đối với các nhà phân tích mã độc hại.

Ollydbg và IDA PRO có thể được sử dụng để dỡ xuống mã độc hại. Trong khi ở Ollydbg và IDA PRO, bạn có thể đánh giá mã một cách thủ công. Nếu bạn hiểu về ngôn ngữ lập trình assembly, bạn có thể quét qua mã để tìm các lệnh lập trình assembly hợp lệ so với các mục không giống lập trình assembly.

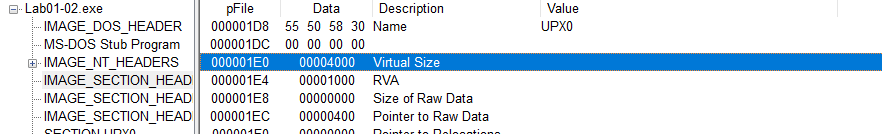
Bạn muốn tìm kiếm lệnh nhảy đuôi, có thể là dấu hiệu của đoạn mã đóng gói. Cuối của một đoạn mã nhảy có thể nhảy đến một vị trí trong bộ nhớ xa xôi, có thể là nơi bộ nhớ đã được giải nén đang nằm.

Một cách để xác định một nhảy là nó có thể được theo sau bởi ngôn ngữ lập trình assembly không hợp lệ.



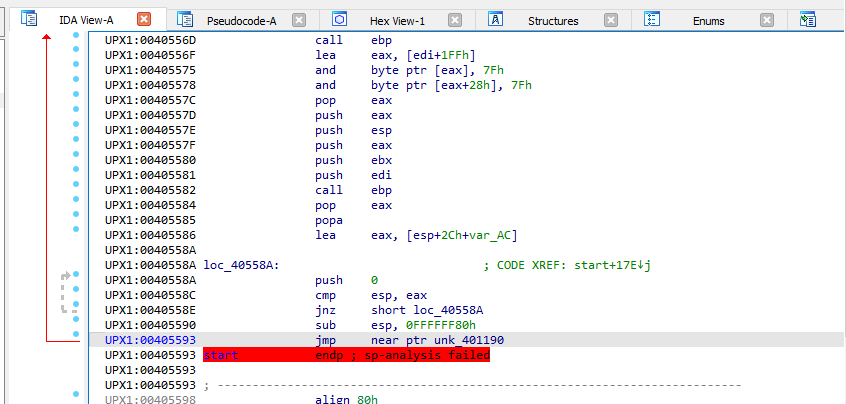
Hình 38: Ảnh String của File Lab01-02.exe khi bị chưa unpack

Khi 1 chương trình bị pack, ta có thể thấy đặc điểm của chúng là rất dễ bị phát hiện khi sử dụng các công cụ detect, hoặc khi chạy, Virtual Size > Raw of size data. Ở trong kịch bản này, Phần .UPX0 có giá trị Raw of size data ( Kích thước dữ liệu thô) là 0, nghĩa là nó không chiếm dung lượng trên đĩa và giá trị Virtual Size (Kích thước ảo của nó là 7000), nghĩa là không gian đó sẽ được phân bổ cho phân đoạn .text. Điều này cho chúng ta biết rằng trình đóng gói sẽ giải nén mã thực thi vào phần .text được phân bổ.:



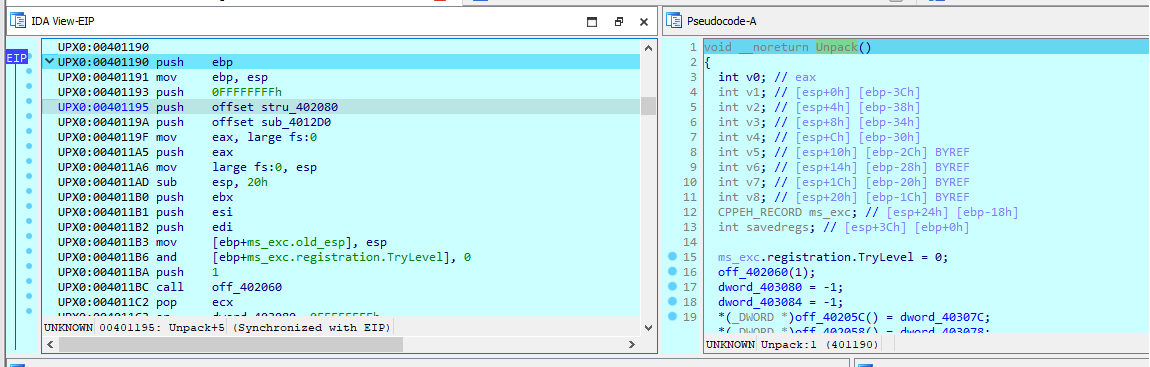
Hình 39: Ảnh khi sử dụng Peview để xem thông tin file Lab01-02.exe

Tiếp đến, ta sử dụng IDA PRO để tiến hành phân tích file Lab01-02.exe, ở đây, ta nhảy đến địa chỉ OEP luôn



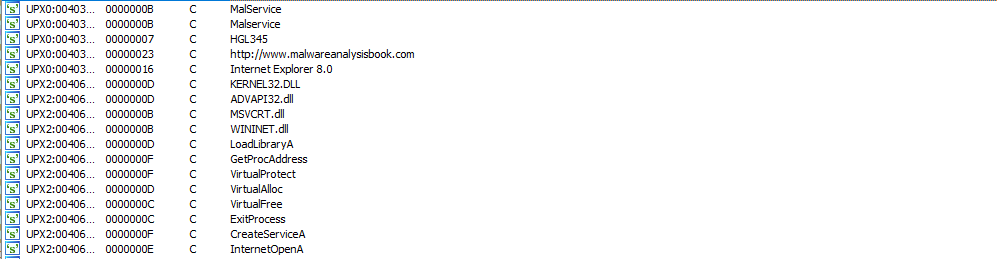
Hình 40: Ảnh đoạn nhảy đến phần được nén

Ấn F2 để tạo điểm dừng, sau đó F9 để thực hiện mã độc hại, sau đó F8 để thực hiện nhảy. Chỗ này là chỗ khu vực đã được giải nén. Nó sẽ đổ dữ liệu mới này.



Hình 41: Ảnh sau khi thực hiện quá trình unpack

Sau khi thực hiện quá trình unpack xong, 1 số chức năng mới của chúng sẽ được hiển thị ra :



Hình 42: Ảnh String của file sau khi unpack

Bạn sẽ tạo ra một tệp mới mà chúng ta có thể tải lên Ollydbg hoặc IDA PRO một lần nữa. Nếu thành công, ta sẽ thấy tất cả các cuộc gọi API có sẵn và toàn bộ IAT. Với tệp này, bạn có thể tiếp tục phân tích mã đã được giải nén.

**Nhận xét**: Kịch bản phân tích mã độc sử dụng pack bằng phần mềm UPX đã mang lại cái nhìn chi tiết và toàn diện về cách mã độc sử dụng kỹ thuật đó để ngăn chặn phân tích. Điều này cũng đã giúp hiểu rõ hơn về quá trình giải nén mã độc và cách các công cụ như Exeinfo PE, Ollydbg, IDA PRO có thể được sử dụng để phân tích mã đã được pack. Thông qua việc xem xét thông tin file và các bước chi tiết trong quá trình phân tích, chúng ta có cái nhìn toàn diện về cách mã độc sử dụng packers và cách chúng ta có thể phân tích, giải nén, nghiên cứu và sử dụng chúng một cách hiệu quả

## 4. Kịch bản 4: Phân tích mã độc plugx, sử dụng kỹ thuật mã hóa để mã hóa thông tin file cũng như nội dung thực thi.

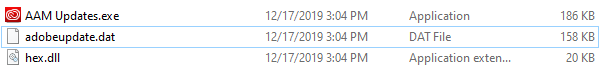
**Giới thiệu**: **Mustang Panda** là một nhóm tấn công được crowdstrike Falcon Intelligence phát hiện vào 2017. Kể từ khi xuất hiện, Mustang Panda đã thực hiện nhiều chiến dịch tấn công vào các doanh nghiệp và tổ chức tại một số quốc gia như Mỹ, Đức , Mông Cổ, Myanmar, Pakistan, Vatican và các quốc gia khác trong khu vực Đông Nam Á, bao gồm cả Việt Nam.

Phương pháp tấn công chính của Mustang Panda là sử dụng kỹ thuật **DLL-Sideloading**, lợi dụng một tệp thực thi sạch để tải một tệp dll độc hại (loader); trong đó, tệp dll có nhiệm vụ giải mã tệp .dat để thực hiện bước tấn công tiếp theo.

**Mô tả:** Phân tích kỹ thuật Bypass AV mà nhóm này sử dụng.

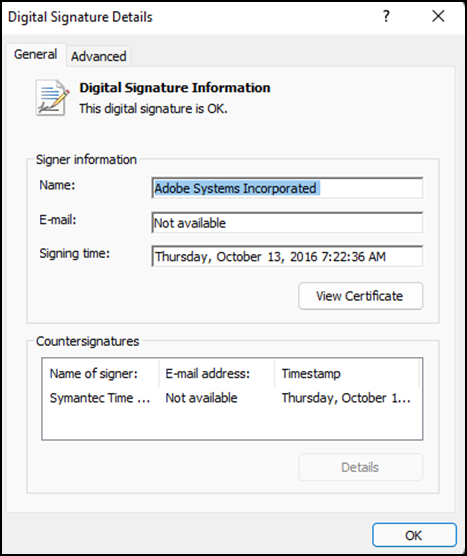
**Mục tiêu:** Hiểu được kỹ thuật mà nhóm tấn công sử dụng, từ đó đưa ra kết luận.

**Tiến hành kịch bản**: Mã độc này gồm 3 phần, các chức năng chính của nó đã được mã hóa ở trong file adobeupdate.dat



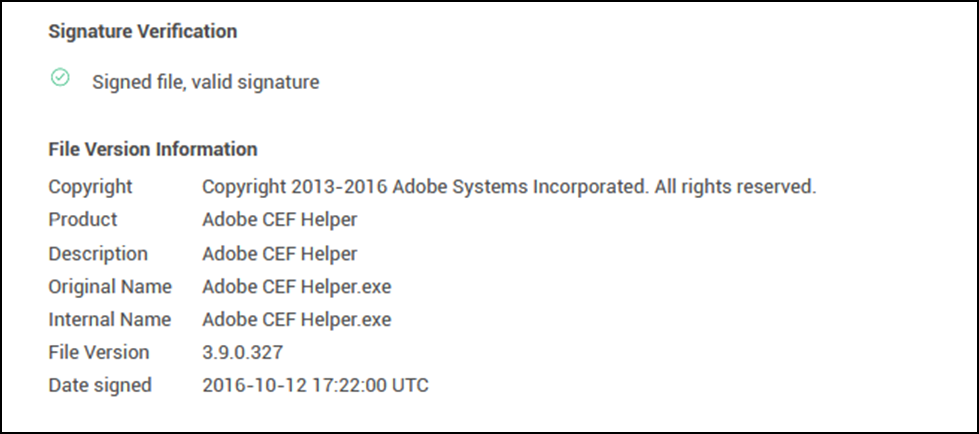
Hình 43: Ảnh các thành phần của mã độc

Mã độc sử dụng kỹ thuật DLL-sideloading qua tệp explorer.exe có chứng chỉ (certificate) hợp lệ của Adobe nhằm qua mặt các phần mềm diệt vi-rút.



Hình 44: Tập tin có chứa chứng chỉ hợp lệ được mã độc sử dụng

Kiểm tra tệp explorer.exe trên Virustotal, ta có thể thấy tên gốc của chương trình là Adobe CEF Helper.exe.



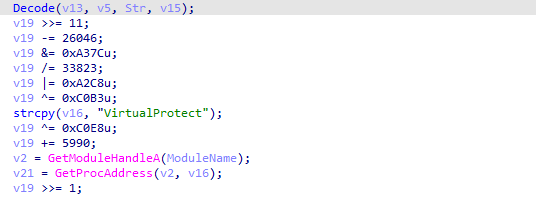
Hình 45: Thông tin về tệp explorer.exe trên Virustotal

Khi được thực thi, tệp này sẽ tải tệp **hex.dll** và gọi tới hàm cefprocessforkhandlerex.



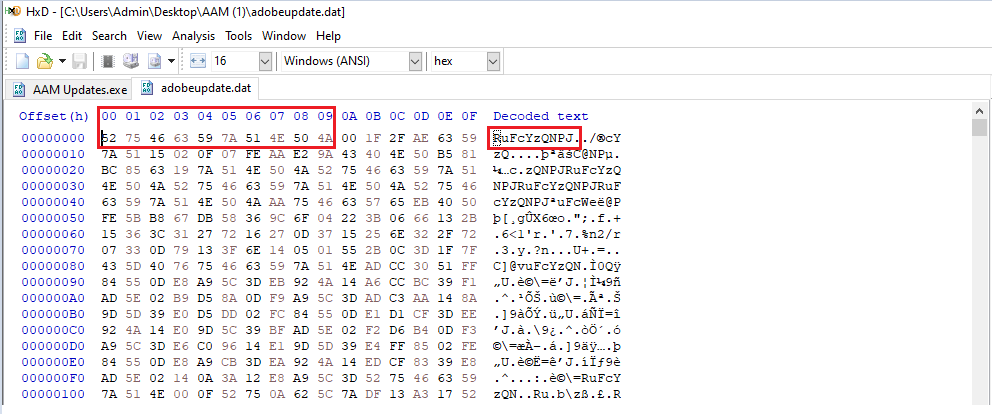
Hình 46: raw source Adobe CEF Helper.exe

Sau khi tải tệp hex.dll và gọi tới hàm cefprocessforkhandlerex, ở trong hàm này, có 1 hàm có chức năng decode:

.

Hình 47: Ảnh hàm decode file adobeupdate.dat

Hàm trên sẽ tiến hành đọc tệp **adobeupdate.dat**, sau đó trích xuất key và payload đã mã hóa và giải mã. Cấu trúc của tệp **adobeupdate.dat** bao gồm 2 phần. Tính từ byte null đầu tiên, các byte phía trước là key và phía sau là payload đã mã hóa.

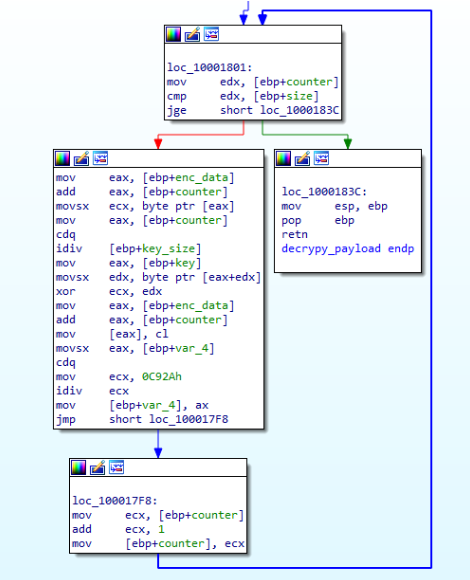


Hình 46: Ảnh cấu trúc tệp adobeupdate.dat khi đang bị mã hóa

DLL có trách nhiệm giải mã tệp DAT và thực thi nó như một shellcode trong bộ nhớ. Nó đọc tải trọng được mã hóa từ đĩa bằng cách gọi các API createfile và readfile, sau đó cấp phát bộ nhớ thông qua virtualalloc. Sau khi giải mã, nó thay đổi bảo vệ thông qua virtualprotect và cuối cùng thực thi nó qua lệnh gọi trực tiếp.

DLL sử dụng các đoạn mã rác và một kỹ thuật làm mờ đơn giản để ẩn các cuộc gọi API thông qua stackstrings và tải chúng động.

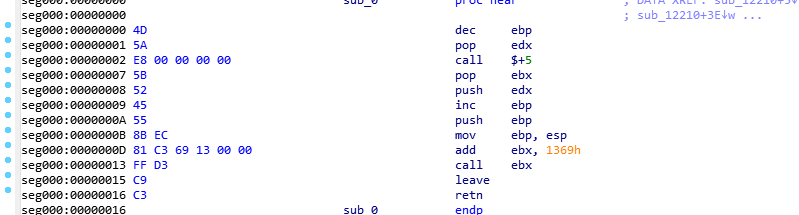
Tệp DAT chứa khóa giải mã bên trong chính nó. Kích thước của khóa có thể thay đổi. Trình tải đọc offset zero của file DAT được mã hóa cho đến khi đạt đến giá trị null và coi đó như một khóa XOR để giải mã phần còn lại của tệp.



Hình 47: Ảnh của hàm giải mã file DAT

Như đã giải thích trước đó, payload được thực thi giống như một shellcode, nhưng đáng chú ý, đó là một tệp nhị phân PE đầy đủ. Trình tải sẽ bắt đầu tải trọng từ offset zero của tải trọng, nghĩa là từ MZ. Hình ảnh dưới đây thể hiện một shellcode được tạo ra rất nhỏ để gọi entrypoint.

Payload đã được giải mã chỉ đơn giản được tải lên trong bộ nhớ và không có dấu vết trên đĩa. Đây là phiên bản tùy chỉnh của plugx với các kỹ thuật làm mờ đơn giản để né tránh phát hiện tĩnh và ẩn các chuỗi.



Hình 48: ảnh shellcode được tạo để gọi entrypoint

Tải trọng là một DLL với một hàm xuất có tên là "Loader." Shellcode ở đầu gọi hàm xuất "Loader."

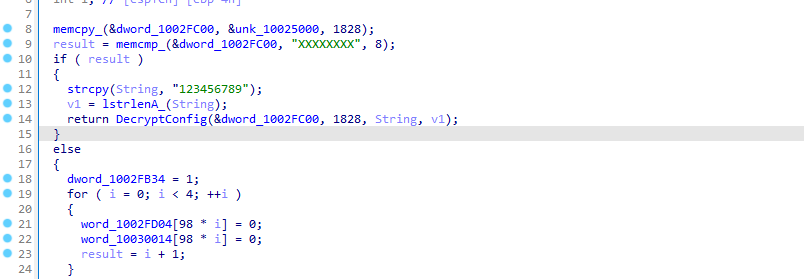
Hàm xuất thực hiện công việc của bộ tải Windows để tải DLL đúng cách và sau đó thực thi entrypoint.

Tải trọng sử dụng hàm bao cho mỗi cuộc gọi API để làm mờ cơ chế gọi API. Trong hàm bao, có stackstrings cho tên chức năng API và tên module. Trong hình ảnh dưới đây, bạn có thể thấy một ví dụ.



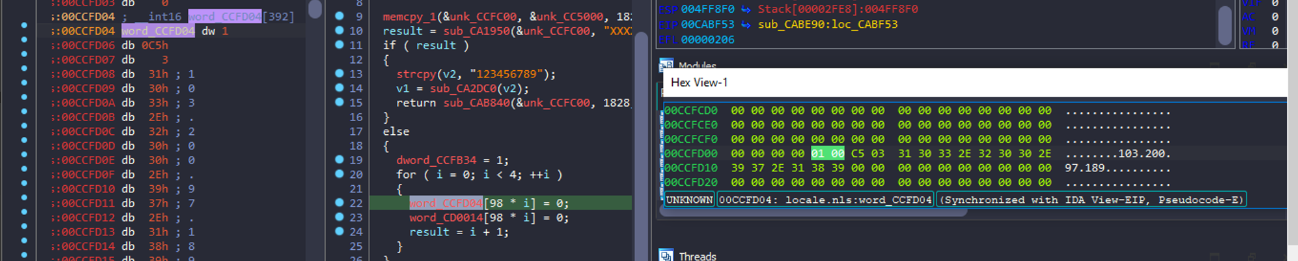
Hình 49 ví dụ mã độc sử dụng stack strings

Tiếp tới, mã độc tiến hành kiểm tra phần đầu cấu hình 8 byte đầu có phải **XXXXXXXX** hay không, nếu không phải thì tiến hành xor file với **123456789** để giải mã lấy c2.

tải IAT và để mọi thứ sẵ

N sàng như Hình 50: mã độc giải mã cấu hình để lấy c2

Sau khi giải mã cấu hình của nó, nó chứa thư mục nơi tệp nhị phân phải được cài đặt, khóa XOR sẽ sử dụng để mã hóa lưu lượng truy cập và danh sách tối đa 4 miền hoặc địa chỉ IP của máy chủ chỉ huy và điều khiển cùng với cổng sẽ được sử dụng. Nói chung, 4 phần tử C2 bao gồm cùng một miền được lặp lại 4 lần hoặc 2 miền được lặp lại mỗi miền hai lần.



Hình 51: Ảnh c2 được mã độc giấu trong config

**Nhận xét**: Về kỹ thuật Bypass AV của nhóm APT Mustang Panda**:**

1. **DLL-sideloading với Chứng chỉ Hợp lệ của Adobe:**
   * Mustang Panda sử dụng kỹ thuật DLL-sideloading thông qua tệp explorer.exe, có chứng chỉ hợp lệ của Adobe, để qua mặt các phần mềm diệt virus.
   * Điều này tạo ra một lớp bảo vệ bổ sung, vì nếu tệp có chứng chỉ hợp lệ, nó có thể được xem xét là an toàn hơn.
   * Chức năng này cho phép mã độc thực hiện tải tệp DLL độc hại và thực hiện các bước tấn công tiếp theo một cách ẩn danh.
2. **Mã hóa và Giải mã Tệp DAT:**
   * Mã độc sử dụng một tệp DAT để chứa key và payload đã mã hóa.
   * Hàm giải mã tệp DAT được sử dụng để trích xuất key và payload đã được mã hóa từ tệp DAT.
   * Mã độc cung cấp một cấu trúc rõ ràng cho tệp DAT, giúp xác định key và payload.
3. **Thực Thi Payload Như Shellcode:**
   * Payload được thực thi như một shellcode, tuy nhiên, nó thực sự là một tệp nhị phân PE đầy đủ.
   * Điều này tạo ra sự nhầm lẫn và khó phát hiện, vì payload đó không được giữ trên đĩa, giúp Mustang Panda tránh được phát hiện tĩnh.
4. **Sử Dụng Wrapper Function và Stackstrings:**
   * DLL sử dụng các đoạn mã rác và kỹ thuật làm mờ đơn giản, chẳng hạn như stackstrings, để ẩn cuộc gọi API.
   * Wrapper function được sử dụng để làm mờ cơ chế gọi API, giúp tăng khả năng tồn tại của mã độc trước các giải pháp phòng thủ.
5. **Giải Mã Cấu Hình và C2:**
   * Mã độc kiểm tra phần đầu cấu hình để xác định C2 và thực hiện việc giải mã cấu hình một cách động.
   * Sử dụng khóa XOR để mã hóa lưu lượng truy cập và giữ danh sách các miền và địa chỉ IP của máy chủ chỉ huy và điều khiển.
6. **Phức Tạp Hóa Cấu Hình để Tránh Phát Hiện:**
   * Cấu hình được giữ trong tệp DAT và được mã hóa, điều này làm tăng độ khó của việc phân tích tĩnh và cung cấp một lớp bảo vệ bổ sung.

Mustang Panda sử dụng một loạt các kỹ thuật thông minh để phá vỡ các biện pháp phòng thủ và phát hiện. Sự kết hợp giữa DLL-sideloading, giải mã động và các kỹ thuật làm mờ giúp mã độc duy trì tính ẩn danh và khó phát hiện. Sự sáng tạo trong việc thực hiện payload như một tệp nhị phân PE đầy đủ tăng cường khả năng xâm nhập. Việc sử dụng chứng chỉ hợp lệ của Adobe trong quá trình tải DLL làm tăng tính thuyết phục và làm giả mạo. Mustang Panda thực sự là một đối thủ đáng gờm, đặt ra thách thức lớn cho các nhóm phòng thủ và chống phát hiện

## Kết luận Chương III

Thực nghiệm kịch bản Antivirus và Bypass Antivirus cung cấp cái nhìn tổng quan về quá trình thử nghiệm và phân tích một loạt các kịch bản liên quan đến hệ thống Antivirus và các phương pháp bypass. Dưới đây là tổng kết các kịch bản đã thực hiện:

1. **Kịch bản 1: Tạo dựng 1 chương trình antivirus và 1 chương trình Bypass antivirus**
   * Thực hiện xây dựng một chương trình antivirus đơn giản để hiểu cơ bản về cách một số chương trình Antivirus có thể hoạt động.
   * Tạo một chương trình Bypass antivirus để thử nghiệm khả năng bypass của nó đối với chương trình Antivirus – Window Defender.
2. **Kịch bản 2: Phân tích Signature-Based - Kaspersky Core Malware**
   * Tiến hành phân tích dựa trên chữ ký để hiểu cách một số sản phẩm Antivirus, ví dụ như Kaspersky, nhận diện các mẫu độc hại thông qua việc so sánh chúng với các chữ ký đã biết.
3. **Kịch bản 3: Phân tích mã độc lỗi thời sử dụng packer bằng phần mềm UPX**
   * Thực hiện phân tích trên một mã độc sử dụng phần mềm UPX để nén và bảo vệ một cách độc lập.
   * Đánh giá khả năng của Antivirus trong việc nhận diện và giải mã mã độc được nén bằng UPX.
4. **Kịch bản 4: Phân tích mã độc plugx, sử dụng kỹ thuật mã hóa để mã hóa thông tin file cũng như nội dung thực thi**
   * Tìm hiểu về một mẫu mã độc cụ thể, plugx, và cách nó sử dụng kỹ thuật mã hóa để ẩn thông tin và thực thi nội dung độc hại.
   * Đánh giá khả năng của Antivirus trong việc nhận diện và giải mã mẫu mã độc này.

Tổng cộng, việc thực hiện những kịch bản này giúp hiểu rõ hơn về cách Antivirus hoạt động, khả năng của chúng trong việc phát hiện và chặn các mẫu độc hại, cũng như những thách thức mà chúng có thể đối mặt trong quá trình đối mặt với các kỹ thuật bypass. Qua đó, kết luận rằng việc cập nhật và nâng cấp liên tục là quan trọng để nâng cao hiệu suất của Antivirus trước các mối đe dọa ngày càng phức tạp.

# Kết luận

Bài báo cáo này đã đưa đến cho chúng ta một hành trình sâu rộng qua lịch sử và kỹ thuật của Antivirus, cũng như những thách thức đối mặt với nó qua các kỹ thuật Bypass Antivirus ngày nay. Chương I giúp ta đảm nhận về tầm quan trọng của Antivirus trong việc bảo vệ hệ thống máy tính khỏi các mối đe dọa, từ bối cảnh lịch sử đến những tiến bộ độc đáo như Trí tuệ Nhân tạo và Học máy.

Chương II phân tích kỹ thuật chi tiết của Antivirus, với dựa trên chữ ký, dựa trên hành vi, và dựa trên heuristics. Đồng thời, nó mở ra cánh cửa cho thế giới Bypass Antivirus, một lĩnh vực đầy thách thức và sáng tạo. Chúng ta đã tìm hiểu về các kỹ thuật tránh, như obfuscators, packers, cũng như các phương pháp đặt mã độc trực tiếp trong bộ nhớ để phá vỡ các cơ chế phòng thủ.

Chương III mang đến cho độc giả những thử nghiệm thực tế, trong đó chúng ta xây dựng và kiểm thử chương trình antivirus, phân tích kỹ thuật Signature-Based với Kaspersky Core Malware, nghiên cứu về packer UPX, và phân tích mã độc plugx với kỹ thuật mã hóa. Những thử nghiệm này không chỉ là một cách để kiểm tra sức mạnh của Antivirus mà còn giúp hiểu rõ hơn về cách các hacker và malware authors nghĩ và hành động.

Cuối cùng, bài viết đặt ra một cảnh báo về sự đối mặt với mối đe dọa ngày càng tinh vi và nguy hiểm. Nó cũng là một kênh để khám phá sự cần thiết của việc duy trì sự cập nhật trong công nghệ bảo mật và đồng thời nâng cao khả năng phát hiện và chống lại những mối đe dọa ngày càng phức tạp. Trong thế giới số ngày nay, sự hiểu biết và chuẩn bị kỹ thuật đều quan trọng để bảo vệ thông tin quan trọng và duy trì tính toàn vẹn của hệ thống.

Hạn chế: Trong phạm vi tìm hiểu của chuyên đề cũng như thời gian hạn chế, đề tài chưa được triển khai và nghiên cứu các trường hợp khác, chưa mở rộng cũng như áp dụng được sâu hơn các kỹ thuật mới nhất và hướng đi của các kỹ thuật này trong tương lai.

Hướng tìm hiểu, nghiên cứu trong tương lai: Tiếp tục dành thời gian để kiểm tra và xây dưng, nghiên cứu các kỹ thuật antivirus của các phần mềm mới nhất, phổ biến trên thị tường, cũng như triển khai các kĩ thuât Bypass antivirus, tích hợp thêm các công cụ Bypass antivirus.

# Tài Liệu Tham Khảo

[1] Malware Obfuscation Techniques: A Brief Survey; Ilsun You, Kangbin Yim - <https://profsandhu.com/cs5323_s18/yk_2010.pdf>

[2] Testing Detection of K-Ary Code Obfuscated by Metamorphic and Polymorphic Techniques; George T. Harter and Neil C. Rowe - https://faculty.nps.edu/ncrowe/oldstudents/harter\_cybersummit21.htm

[3] ALL AV BYPASS TECHNIQUE: EVERYTHING YOU NEED TO KNOW - https://www.techzeroday.com/all-av-bypass-technique-everything-you

[4] Bypassing Defender on modern Windows 10 systems; purpl3f0x - https://www.purpl3f0xsecur1ty.tech/2021/03/30/av\_evasion.html

[5] Defeat the Castle – Bypass AV & Advanced XDR solutions; zux0x3a - <https://0xsp.com/security%20research%20%20development%20srd/defeat-the-castle-bypass-av-advanced-xdr-solutions/>

[6]Understanding And Detecting Dll 1nj3ct0n & Process Hollowing; Alparslan Akyıldız academy -

[Understanding And Detecting Dll 1nj3ct0n & Process Hollowing | by Alparslan Akyıldız academy | Medium](https://alparslanakyildiz.medium.com/understanding-and-detecting-dll-1nj3ct0n-process-hollowing-fcd87676d36b)

[7] Defense Evasion; ATT&CK - https://attack.mitre.org/tactics/TA0005/

[8] <https://www.privacysense.net/terms/behavior-based-security/>

Https://www.bordergate.co.uk/shellcode-obfuscation/