# MỤC LỤC

## MỤC LỤC 3

## DANH MỤC TỪ VIẾT TẮT 5

## DANH MỤC HÌNH VẼ 6

## LỜI NÓI ĐẦU 8

## CHƯƠNG 1. PHƯƠNG PHÁP THU THẬP VÀ PHÂN TÍCH MÃ ĐỘC . 10 1.1 Giới thiệu về mã độc hại 10

* + 1. Khái niệm mã độc hại 10
    2. Lịch sử mã độc hại 10

## Cơ chế hoạt động của mã độc 11

* + 1. Cơ chế hoạt động của Virus 12
    2. Cơ chế hoạt động của Worm 12
    3. Cơ chế hoạt động của Trojan Horse 12

## Phương pháp thu thập mã độc 12

* + 1. Các phương pháp thu thập mẫu 12
    2. Các công cụ thu thập mẫu mã độc 16

## Quy trình phân tích mã độc hại 30

* + 1. Các phương pháp phân tích mã độc 30
    2. Các bước cơ bản phân tích mã độc hại 31
    3. Phân tích môt mẫu mã độc cụ thể 34

CHƯƠNG 2. NGHIÊN CỨU VỀ MỘT SỐ CƠ SỞ DỮ LIỆU MÃ ĐỘC .. 36

* 1. [Các kiểu dấu hiệu mã độc và kỹ thuật phát hiện tương ứng 36](#_TOC_250006)
     1. String – Chuỗi 36
     2. Mã băm 37
     3. Khung mã độc có sẵn 38
     4. Phương pháp dựa trên hành vi 38
     5. Kỹ thuật lọc 39
     6. Phát hiện bằng việc giải mã tĩnh 40
     7. Mã giả lập 40
  2. [Nghiên cứu chuẩn trao đổi dữ liệu mã độc 41](#_TOC_250005)
  3. Nghiên cứu cơ sở dữ liệu mã độc Clam AV từ đố xây dựng cơ sở dữ liệu mẫu mã độc 44
     1. Clam Anti Virus 44
     2. ClamAV Virus Databases 44
     3. Debug thông tin từ libclamav 45
     4. Định dạng chữ ký của ClamAV 51

[CHƯƠNG 3. XÂY DỰNG CƠ SỞ DỮ LIỆU MÃ ĐỘC 61](#_TOC_250004)

* 1. Xây dựng chương trình quản lý cơ sở dữ liệu mẫu mã độc theo chuẩn 61
  2. [Xây dựng chương trình nhận dạng mã độc theo chuỗi 65](#_TOC_250003)

[KẾT LUẬN 69](#_TOC_250002)

[TÀI LIỆU THAM KHẢO 70](#_TOC_250001)

[PHỤ LỤC 71](#_TOC_250000)

**DANH MỤC TỪ VIẾT TẮT**

|  |  |
| --- | --- |
| MS-DOS | Microsoft Disk Operating System |
| Spyware | Spy software |
| FTP | File Transfer Protocol |
| UDP | User Datagram Protocol |
| Ddos | Distributed Denial of Service |
| MD5 | Message-Digest algrorithm 5 |
| CPU | Central Processing Unit |
| HTML | Hypertext Tranfer Protocol |
| ICSG | Industry Connection Sercurity Group |
| XML | Extensible Markup Language |
| DLL | Dynamic-link Library |
| SHA | Message-Digest algrorithm 5 |
| malware | Malicious software |
| PE | Portable Executable |
| TCP/IP | Transmission Control Protocol/Internet Protocol |

**DANH MỤC HÌNH VẼ**

Hình 1.1 Giao diện chính virustotal 13

Hình 1.2 Giao diện virustotal sau khi quét 13

Hình 1.3 Giao diện trước khi quét của ThreatExpert 14

Hình 1.4 Giao diện sau khi quét của ThreatExpert 14

Hình 1.5 Hệ thống ThreatExpert 15

Hình 1.6 Người dung gửi mẫu tới 15

Hình 1.7 ThreatExpert trả kết quả về cho người dùng 16

Hình 1.8 Giao diện IDA 17

Hình 1.9 IDA Text view 18

Hình 1.10 Function Windows 18

Hình 1.11 Import Windows 19

Hình 1.12 Cross – references 19

Hình 1.13Function Call 19

Hình 1.14 Menu Jump 20

Hình 1.15 Menu search 21

Hình 1.16 Text search 22

Hình 1.17 Menu View 22

Hình 1.18 Compiler setup 23

Hình 1.19 Plugin 24

Hình 1.20 Giao diện Olly Debug 25

Hình 1.21 Tùy chọn View 26

Hình 1.22 Debug 27

Hình 2.23 BreakPoint 28

Hình 1.24 Giao diện HxD 29

Hình 2.1 Mô hình chương trình quét hành vi 39

Hình 3.1 Cấu trúc file dữ liệu 61

Hình 3.2 Nhập cơ sở dữ liệu 62

Hình 3.3 Hiển thị 63

Hình 3.4 Chi tiết mẫu mã độc 63

Hình 3.5 Giao diện Scan 64

Hình 3.6 Mô hình chức năng quét 64

Hình 3.7 Tệp dữ liệu 65

Hình 3.8 Chuỗi String 66

Hình 3.3Mô hình chương trình quét 67

Hình 3.9 Giao diện chính 67

Hình 3.10 Danh sách mã độc 68

Hình 3.11 Thêm mẫu 68

**LỜI NÓI ĐẦU**

Khi nhu cầu và việc sử dụng Internet của con người ngày càng tăng thì cũng là lúc những mối đe dọa xuất hiện càng nhiều, nổi bật là đe dọa của mã độc hại. Mã độc hại xuất hiện bất kỳ ở đâu trên môi trường của các thiết bị điện tử như các đĩa mềm, usb, máy tính đến môi trường Internet trong các website, trong các tin nhắn, trong hòm thư điện tử của người dùng, trong các phần mềm miễn phí…. Khi mã độc hại đã nhiễm vào một máy tính nào đó thì nó lây lan sang các máy tính khác là khá nhanh và thiệt hại do mã độc hại gây ra là khó có thể lường trước được.

Hiện tại để chống lại các loại mã độc hại người ta thường sử dụng các chương trình Antivirus. Tuy nhiên để có được một chương trình Antivirus một cách có hiệu quả cần có một thuật toán quét sao cho nhanh nhất và quan trọng là một cơ sở dữ liệu đầy đủ và update thường xuyên những mẫu mới. Ngoài ra cơ sở dữ liệu đó phải đảm bảo kiến trúc, định dạng và phương pháp lưu trữ một cách khoa học để cho phép các thuật toán đối sánh mẫu được thực hiện một cách hiệu quả nhất. Đồng thời nó phải có các cơ chế đáp ứng được các tiêu chuẩn để có thể trao đổi với các cơ sở dữ liệu cả các hãng khác. Việc xây dựng được cơ sở dữ liệu như vậy có thể sử dụng để phát triển các phần mềm phát hiện và ngăn chặn mã độc hiệu quả. Ngoài ra nó cũng có thể sử dụng như một công cụ để hỗ trỡ đắc lực cho những người làm công việc tác nghiệp phân tích mã độc.

Chính vì những lý do trên, nên mục đích của dồ án này là để nhằm nghiên cứu để nhằm sang tỏ các kiểu dấu hiệu để phát hiện ra các loại mã độc, các chuẩn dấu hiệu để trao đổi mã độc và cấu trúc chung của cơ sở dữ liệu mã độc của một số hãng phần mềm lớn trên thế giới như ClamAV, Kaspersky,BKAV… Đồng thời thiết kế, xây dựng một cơ sở dữ liệu mã độc và một chương trình demo để quản lý và sử dụng cơ sở dữ liệu đó.

Đồ án được trình bày theo bố cục:

## Chương 1. Quy trình thu thập và phân tích mã độc hại

Trình bày tổng quan về mã độc hại, cơ chế hoạt động của mã độc và quy trình thu thập, phân tích mã độc

## Chương 2. Nghiên cứu về một số cơ sở dữ liệu mã độc

Trình bày về một số phương pháp phát hiện mã độc cơ bản mà các chương trình anti virus hay dùng để tìm và diệt mã độc. Tìm hiểu nghiên cứu chuẩn trao đổi dữ liệu mã độc, cơ sở dữ liệu mã độc của chương trình clamav.

## Chương 3.Xây dựng cơ sở dữ liệu mã độc

* + - Xây dựng cơ sở dữ liệu theo chuỗi nhận dạng, theo chuẩn đã nêu ở trên
    - Xây dựng một chương trình quét mã độc sử dụng chuỗi nhận dạng
    - Xây dựng chương trình quản lý cơ sở dữ liệu theo chuẩn.

## Kết luận

Trình bày về kết quả đạt được của quá trình làm đồ án, những mặt hạn chế và

hướng phát triển.

Vì điều kiện thời gian làm đồ án có hạn cũng như hiểu biết còn hạn chế nên chắc chắn đề tài không tránh khỏi những sai sót. Rất mong các thầy cô góp ý kiến để đề tài khoa học của em được hoàn thiện hơn.

Em xin chân thành cảm ơn các thầy, cô giáo Học Viện Kỹ Thuật Mật Mã đã tận giảng dạy và giúp đỡ em có những kiến thức cần thiết để hoàn thành đồ án đã nhận. Em cảm ơn thầy Lương Thế Dũng - phó khoa An toàn thông tin, thầy Hoàng Thanh Nam giảng viên khoa An toàn thông tin đã giúp đỡ, hướng dẫn và chỉ bảo để em hoàn thành đồ án.

Em xin chân thành cảm ơn!

*Hà Nội, ngày 11 tháng 06 năm 2013*

## Sinh viên thực hiện

Trần Văn Khang

# CHƯƠNG 1. PHƯƠNG PHÁP THU THẬP VÀ PHÂN TÍCH MÃ ĐỘC

Để có thể xây dựng một cơ sở dữ liệu mẫu mã độc đạt yêu cầu thì cần phải trải qua nhiều quá trình khác nhau, trong đó quá trình thu thập và phân tích mã độc là quá trình rất quan trọng. Nó giúp cho việc xây dựng cơ sở dữ liệu mẫu mã độc một cách hiệu qả và nhanh chóng. Cùng với đó quá trình này đem lại cho ta cái nhìn rõ nét nhất về các hành vi cũng như là đặc điểm mà mã độc đó gây hại cho người dung. Vì vậy chương trình sẽ đi sâu vào quá trình thu thập và phân tích mã độc.

## Giới thiệu về mã độc hại

* + 1. Khái niệm mã độc hại

Malware (Malicious software) hay còn gọi là mã độc hại (Malicious code) là tên gọi chung cho các phần mềm được thiết kế, lập trình đặc biệt để phá hoại hệ thống của bạn hoặc làm gián đoạn môi trường hoạt động mạng. Malware thâm nhập vào một hệ thống máy tính mà không có sự đồng ý của chủ sở hữu.

* + 1. Lịch sử mã độc hại

Internet phát triển, nó thực hiện việc kết nối các máy tính, máy chủ, laptop, mobile phone trên khắp thế giới, thêm vào là sự phát triển của các website vì vậy mã độc hại cũng theo đó mà phát tri ển rộng khắp với số lượng lớn và ngày càng phức tạp hơn.

Một số mốc lịch sử của mã độc hại trên thế giới.

Năm 1949 John von Neuman (1903-1957) phát triển nền tảng lý thuyết tự nhân bản của một chương trình cho máy tính.

Năm 1981 các virus đầu tiên xuất hiện trong hệ điều hành của máy tính Apple II.

Năm 1983 Fred Cohen, một sinh viên đại học Mỹ, đã đưa ra định nghĩa đầu tiên về virus: “Là một chương trình máy tính có thể tác động những chương trình máy tính khác bằng cách sửa đổi chúng bằng phương pháp đưa vào một bản sao của nó”. Fred Cohen luôn là cái tên được nhắc đến khi nói về lịch sử virus.

Năm 1986 hai anh em lập trình viên người Pakistan là Basit và Amjad thay thế mã thực hiện (executable code) trong rãnh ghi khởi động của một đĩa mềm bằng mã riêng của họ, được thiết kế với mục đích phát tán từ một đĩa mềm 360K khi cho

vào bất cứ ổ đĩa nào. Loại đĩa mềm mang virus này có mác “© Brain”. Đây chính là

những virus MS-DOS xuất hiện sớm nhất.

Năm 1987 Lehigh, một trong những virus file đầu tiên xâm nhập các tệp lệnh command.com (virus này sau đó tiến hoá thành virus Jerusalem). Một virus khác có tên IBM Christmas, với tốc độ phát tán cực nhanh (500.000 bản sao/tiếng), là cơn ác mộng đối với các máy tính lớn (mainframe) của Big Blue trong suốt năm đó. đồng hồ của máy tính (giống bom nổ chậm cài hàng loạt cho cùng một thời điểm).

Tháng 11 cùng năm, Robert Morris chế ra worm chiếm cứ các máy tính của ARPANET làm liệt khoảng 6.000 máy.

Năm 1991 virus đa hình (*polymorphic virus*) ra đời đầu tiên là Tequilla. Loại này biết tự thay đổi hình thức của nó, gây ra sự khó khăn cho các chương trình chống virus.

Năm 1994 Trò lừa qua e-mail đầu tiên xuất hiện trong cộng đồng tin học. Trò này cảnh báo người sử dụng về một loại virus có thể xoá toàn bộ ổ cứng ngay khi mở e-mail có dòng chủ đề “Good Times”. Mặc dù không gây thiệt hại gì mà chỉ có tính chất doạ dẫm, trò lừa này vẫn tiếp tục xuất hiện trong chu kỳ từ 6 đến 12 tháng/lần.

Năm 1995 *macro virus* đầu tiên xuất hiện trong các mã macro trong các tệp của Word và lan truyền qua rất nhiều máy. Loại virus này có thể làm hư hệ điều hành chủ.

Năm 1999 Bubble Boy sâu máy tính đầu tiên không dựa vào việc người nhận email có mở file đính kèm hay không. Chỉ cần thư được mở ra, nó vẫn sẽ tự hoạt động.

Năm 2003 Slammer một loại worm lan truyền với vận tốc kỉ lục, truyền cho khoảng 75 ngàn máy trong 10 phút.

Năm 2004 đánh dấu một thế hệ mới của mã độc hại là worm Sasser. Với loại worm này thì người ta không cần phải mở đính kèm của điện thư mà chỉ cần mở lá thư là đủ cho nó xâm nhập vào máy. Sasser không hoàn toàn hủy hoại máy mà chỉ làm cho máy chủ trở nên chậm hơn và đôi khi nó làm máy tự khởi động trở lại. Ở Việt Nam mã độc hại cũng gây ra những thiệt hại đáng kể.

## Cơ chế hoạt động của mã độc

Mỗi loại mã độc có các cơ chế hoạt động khác nhau tuy nhiên chúng đều có cùng một mục đích là phá hoại gây hại cho máy tình người dùng. Ở đây em sẽ nêu ra cơ chế hoạt động của các loại mã độc cơ bản là Virus, Trojan Horse, worm

* + 1. Cơ chế hoạt động của Virus

Virus không thể tồn tại độc lập nên cách hoạt động duy nhất là chúng lây lan qua các file trên máy tính người dung. Thường thì virus lây lan với mục đích là xóa, sửa file trên máy tính. Loại virus nguy hiểm nhất là loại virus đa hình chúng tự đính kèm vào các file exe, các file này vẫn hoạt động bình thường. Sau mỗi lần lây lan các đoạn mã virus được thay đổi theo các phương pháp khác nhau để che dấu đặc điểm nhận dạng của nó.

* + 1. Cơ chế hoạt động của Worm

Do worm tồn tại một cách độc lập nên nó cách thức hoạt động của nó tuy là cũng tự nhân đôi, sao chép chính nó nhưng không cần lân lan vào một file nào. Worm hoạt động không cần tác động của người dung và thường lây lan qua mạng LAN hoặc Internet.

* + 1. Cơ chế hoạt động của Trojan Horse

Loại mã độc này thường ẩn mình dưới dạng là một chương trình an toàn, vô hại với máy tính người dung. Chính vì vậy nó ngoài thực thực thi chức năngchương trình an toàn nó còn âm thầm thực thi các chức năng độc hại ẩn trong đó mà khi nó cần đến mới thực thi.

## Phương pháp thu thập mã độc

1.3.1 Các phương pháp thu thập mẫu

Có nhiều phương pháp thu thập mẫu mã độc xây dựng honeypot như một số hãng phần mềm diệt virus:

* + - <http://www.honeyclient.org/trac>
    - <http://nepenthes.carnivore.it/>
    - <http://sourceforge.net/projects/amunhoney/>

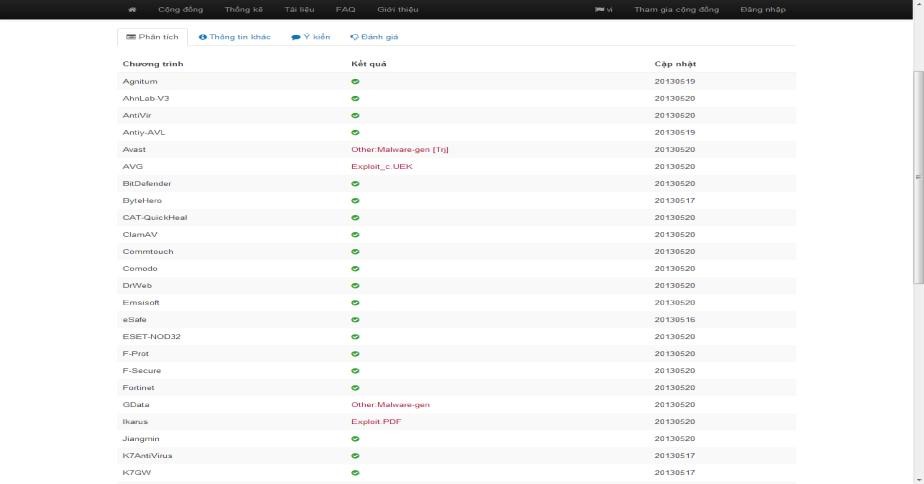
Hoặc lấy mẫu từ khách hàng tự gửi đến, hoặc lấy từ các nguồn chia sẻ trên mạng, hoặc mua từ các hãng nghiên cứu bảo mât… Ở đây em sẽ trình bày phương pháp thu thập dựa vào mẫu do người dùng gửi đến. Phương pháp này thu thập được rất nhiều mẫu do hằng ngày có rất nhiều tổ chức và cá nhân trên thế giới gửi tới các mẫu có thể là mã độc hoặc không tớ các trang web, hang phần mềm diệt virus.

Các trang web thu thập mẫu nổi tiếng trên thế giới là:

* + - * https://[www.virustotal.com](http://www.virustotal.com/)



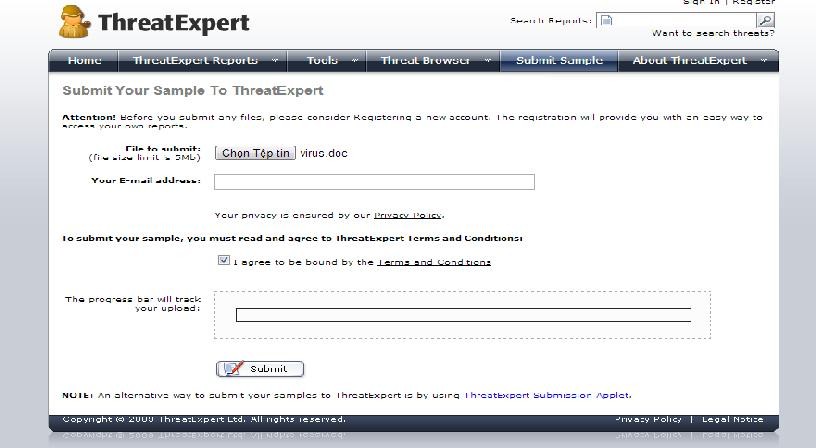
*Hình 1.1 Giao diện chính virustotal*



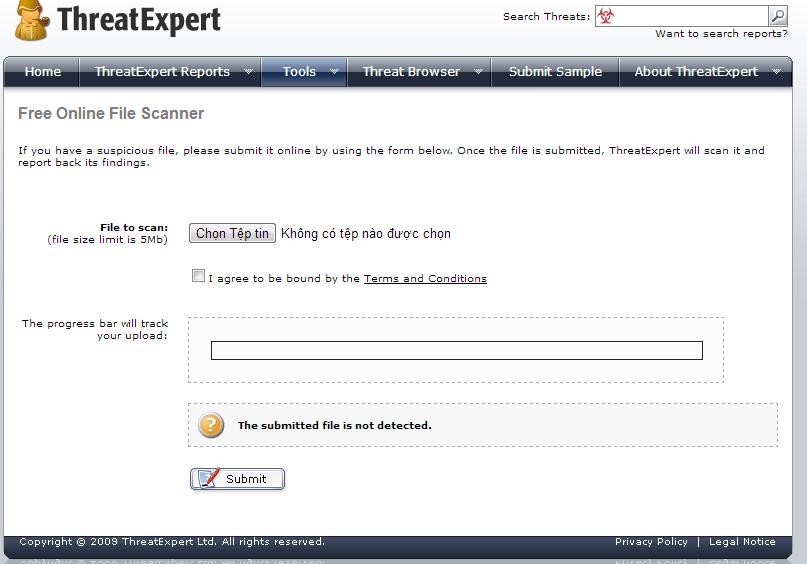
*Hình 1.2 Giao diện virustotal sau khi quét*

Virus total là trang web được google mua lại từ năm 2012 nó liên kết với gần 50 hãng phần mềm diệt virus lớn trên thế giới nhằm mục đích thu thập và đưa ra những cảnh báo về các mẫu mã độc người dùng gửi lên một cách nhanh chóng. Cùng với đó là số lượng mẫu của trang này thu thập được rất nhiều.

* + - * Threatepert



*Hình 1.3 Giao diện trước khi quét của ThreatExpert*

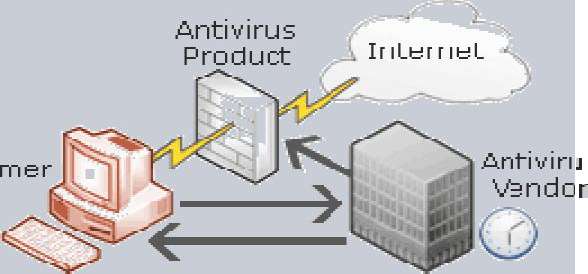


*Hình 1.4 Giao diện sau khi quét của ThreatExpert*

ThreatExpert là một hệ thống phân tích mối đe dọa tự động được thiết kế để phân tích và báo cáo hành vi của virus máy tính, sâu, trojan, adware, spyware, và các rủi ro liên quan đến bảo mật khác trong một chế độ hoàn toàn tự động chỉ trong

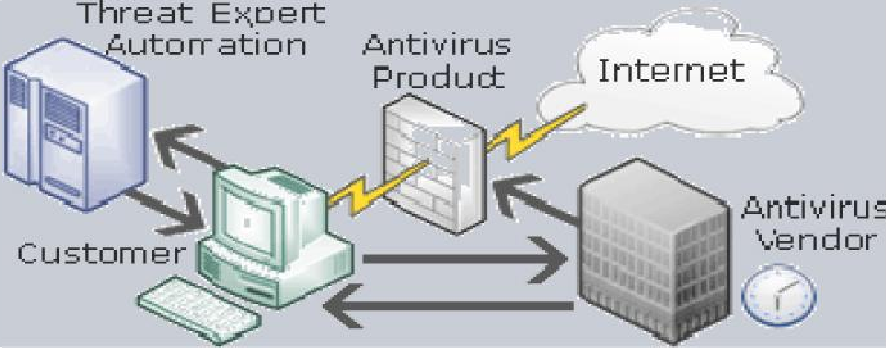
vài phút ThreatExpert có thể xử lý mẫu và tạo ra một mối đe dọa báo cáo rất chi tiết với mức độ chi tiết kỹ thuật phù hợp hoặc vượt quá tiêu chuẩn công nghiệp chống virus chẳng hạn như những người bình thường tìm thấy trong bách khoa toàn thư virus trực tuyến.

Hệ thống của ThreatExpert:



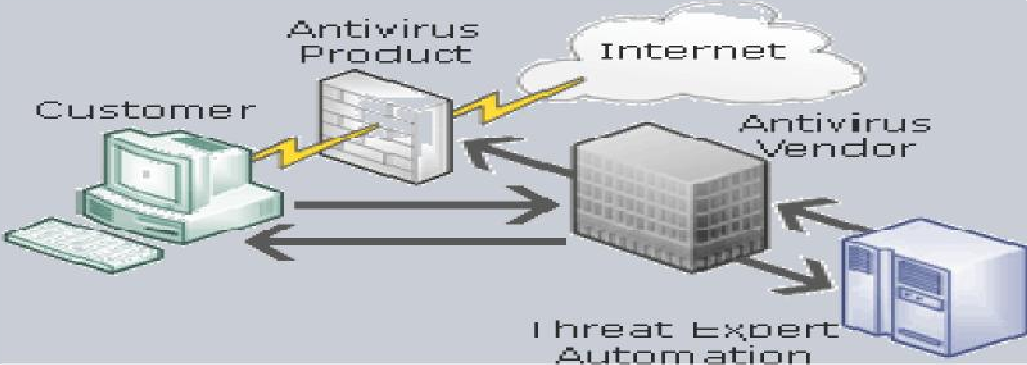
*Hình 1.5 Hệ thống ThreatExpert*

* + - Người sử dụng gửi mẫu tới ThreatExpert.
    - ThreatExpert cung cấp một phân tích mô tả chi tiết mối đe dọa ngay lập tức.
    - Mô tả mối đe dọa có thể được sử dụng bởi khách hàng để thực hiện giai đoạn giảm thiểu mối đe dọa ( như tự động hoặc bằng tay loại bỏ mối đe dọa hoặc phòng) trước khi người cung cấp phần mềm diệt virus đáp ứng.



*Hình 1.6 Người dung gửi mẫu tới*

* Ngay như những người hỗ trợ nhận được một mẫu từ các khách hàng, tham gia ThreatExpert, ThreatExpert cung cấp một phản ứng đe dọa mô tả ngay lập tức.
* Mô tả mối đe dọa mới có thể được ngay lập tức được đăng trên trang web của công ty của các nhà cung cấp, trước khi các nhà cung cấp khác có khả năng làm như vậy.
* Khách hàng khác của nhà cung cấp có thể được ngay lập tức cảnh báo về một mối đe dọa mới với mô tả đầy đủ mối đe dọa.
* Nhà cung cấp có thể sử dụng các báo cáo chi tiết hành vi hỗ trợ trong phân tích phần mềm độc hại để giúp phát hiện ra phần mềm độc hại.



*Hình 1.7 ThreatExpert trả kết quả về cho người dùng*

* + 1. Các công cụ thu thập mẫu mã độc

Để thu thập được mẫu mã độc trước tiên cần phải có những câu cụ chuyên dụng. Có rất nhiều công cụ để trình bày hết thì không thể ,trong phần này em sẽ không đi sâu vào các công cụ tìm mã băm hay các hành vi mà sẽ đi sâu về các công cụ em đã nêu ở phần “2.3.3 Mã giả lập” và một số công cụ bổ sung thêm.

* + - 1. IDA

IDA là một công cụ hỗ disasembly và debug. Tính năng nổi bật của nó là cho phép dịch ngược mã nguồn mà không cần load vào bộ nhớ. Đây là công cụ rất mạnh và hữu ích cho cả công việc lập trình và dịch ngược.

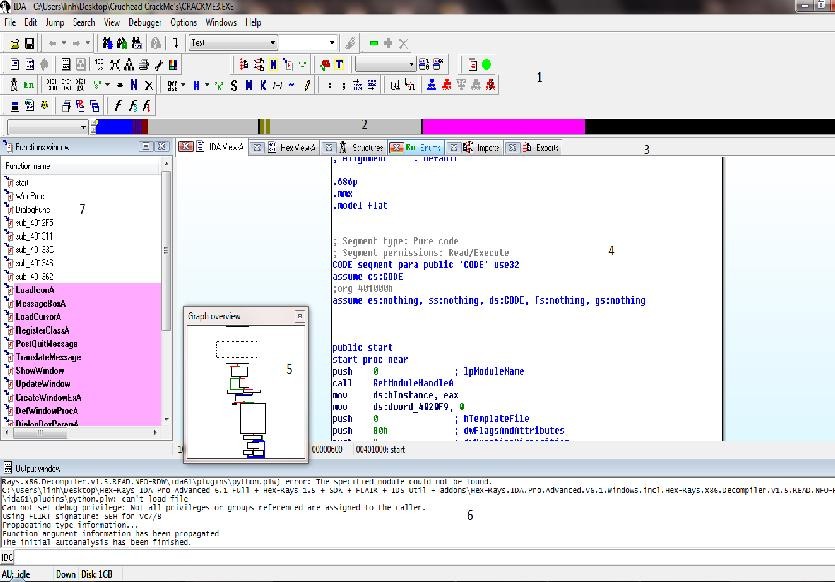
Cách sử dụng và tính năng:

* + - * + Sau khi load vào sẽ hiện lên một hộp thoại bên trong đó có chứa định dạng file mà chúng ta làm việc. Ta thường quen thuộc chọn PE( pe.ldw) để làm việc. Trong Option là các tùy chọn để cho IDA tự phân tích file vào cho ra hiển thị ở cửa sổ chính. Sau đó nhấn OK để vào giao diện chính để làm việc. Giao diện là việc của IDA gồm có 7 cửa sổ:

1. Toobar chứa các thanh công cụ sử dụng trong hoạt động của

IDA, để thực hiện các chức năng ta nhấp vào các biểu tượng trên đó.

1. Navigation band nơi ta có thể vào và ra khỏi địa chỉ bằng cách nhấp chuột di chuyển bằng cách điều chỉnh theo hướng mũi tên màu vàng. Mỗi màu sắc của cửa sổ thể hiện vùng dữ liệu mà ta làm việc.
2. Tabs có chứa cửa sổ nhỏ những thông tin chi tiết về file đối tượng, việc phân tích phụ thuộc vào những tabs này. Gồm có IDA View-A, Hex View-A, Struct, Enums, Imports,Exports, String.
3. Disassembly hiển thị dữ liệu để chúng ta phân tích theo 2 loại text hoặc graph
4. Graph overview một đồ thị thu nhỏ mô tả cấu trúc cơ bản của dữ liệu. Màn hình chữ nhật chấm nhỏ cho thấy màn hình hiển thị vị trí hiện tại đang làm việc.
5. Output window nhận được các thông tin, tin nhắn từ IDA sau khi load file xong.



*Hình 1.8 Giao diện IDA*

1. Function window đây là cửa sổ hiện thị tất cả các hàm API các

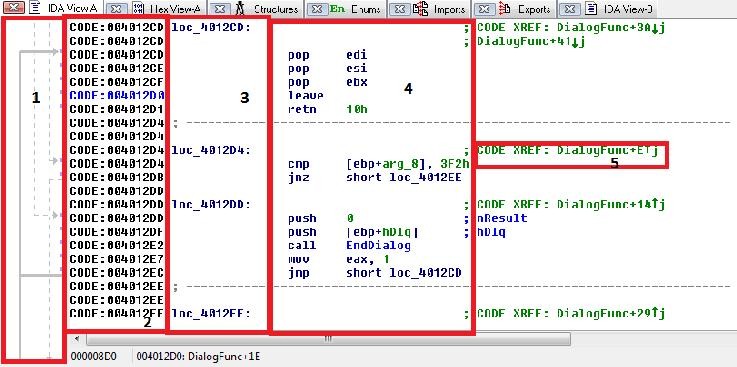
địa chỉ tìm thấy trong quá trình phân tích.

* + - * + IDA Text view màn hình hiển thị kiểu text trình bày toàn bộ về công việc disassembler cũng như cung cấp địa chỉ để xem các dữ liệu thuộc vùng nào.

1. Các mũi tên thể hiện nơi nhảy đến trong khối đoạn mã và nhận

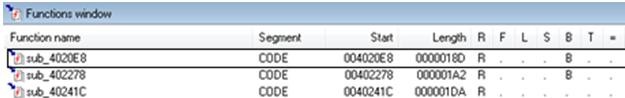
ra các vòng lặp nhỏ.

1. Hiện ra các địa chỉ Virtual Address
2. Các vị trí được so sánh để đánh dấu nhảy hoặc các biến tham chiếu trong stack



*Hình 1.9 IDA Text view*

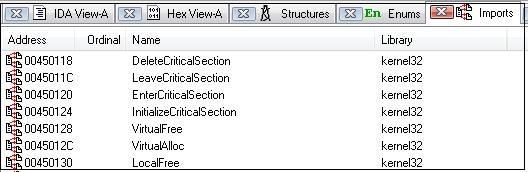
1. Code của chương trình disassembled
2. Code tham chiếu hiển thị các điểm đến khi truy cập, nhấp đúp vào sẽ đưa ta đến đoạn mã, hoặc hàm được gọi hoặc nhảy lên phía trên. Hoặc để nhảy đến vị trí tham chiếu khác ta nhấp chuột phải->jump to cross reference.
   * + - * Fucntion Window hiển thị các hàm được nhận định bởi IDA



*Hình 1.10 Function Windows*

Bằng cách nhấp chuột hoặc enter là ta có thể di chuyển đc đến hàm đó trong cửa sổ chính disassembly. Ta nhìn thấy bên trong function window gồm có function name, segment, start, length, R( return hàm trả về khi gọi),F ( far),L (libarary), S (static), B( BP tham chiếu đến biến cục bộ),T ( type thông tin).

* + - * + String Window tại cửa sổ này hiển thị ra thông tin tất cả các chuỗi xuất hiện trong file thực thi. Cửa sổ này hiện thị chi tiết về địa chỉ, độ dài, kiểu, tên chuỗi đó là gì. Ta có thể truy cập đến các chuỗi này bằng cách nhấn đúp chuột nó sẽ di chuyển đến cửa sổ Disassembly để chúng ta thực hiện.
        + Import window cửa sổ hiển thị chi tiết địa chỉ, tên hàm được import và thư viện chứa hàm import đó. Đây là một cửa sổ rất quan trọng bởi ta có thể thấy được chương trình sử dụng các DLL khác nhau và chức năng của các hàm được gọi như đọc, viết hoặc registry .... Tại đây ta có thể thập được các thông tin về các hàm import mà mã độc hại hay dùng để chèn vào với mục đích xấu như lấy cắp thông tin, theo dõi …



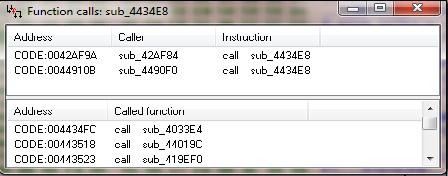
*Hình 1.11 Import Windows*

* + - * + Export window là cửa sổ liệt kê các entrypoint của tập tin. Trong cửa sổ này gồm có liệt kê theo tên, địa chỉ ảo và thứ tự( nếu có). Đối với các file thực thi export window cần phải chứa ít nhất một entry point đây là địa chỉ điểm đầu vào đầu tiên trong lúc thực thi.
        + Cross-references cửa sổ hiện thi tất cả các tham chiếu , là tất cả các điểm code nơi mà hàm được gọi. Để mở được cửa sổ này ta chỉ nhấn vào header của hàm, kích chuột phải chọn jump to reference xref hoặc chọn trên thanh công cụ view --> Open subview --> cross reference.



*Hình 1.12 Cross – references*

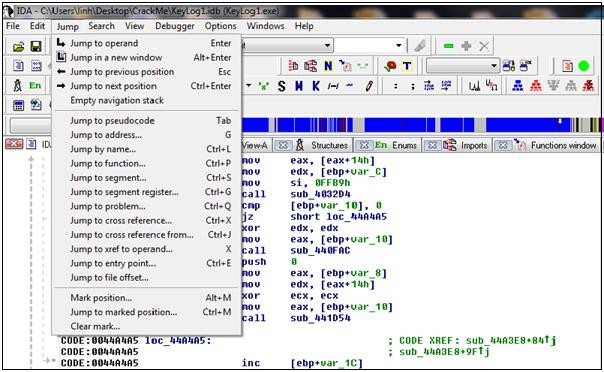
* + - * + Name Window là cửa sổ cung cấp danh sách các tên được sử dụng, tên có thể được sắp xếp theo bảng chữ cái hoặc để virtual address. Hiển thị tên có các chữ cái in hoa khác nhau A( string data), D( data), C( Name code), I( import name), L (library fuction), F( regular function). Bằng việc nhấp đúp chuột vào tên bất kì ta muốn tìm, chương trình sẽ chuyển đến vị trí đó trong cửa sổ chính hiển thị để làm việc. Chọn view --> option subview --> name.
        + Function Calls cửa sổ thể hiện được 2 chức năng caller và called function. Xác định được các hàm được gọi xung quanh đó là gì. View --> option subview --> function call.



*Hình 1.13Function Call*

Còn rất nhiều những cửa sổ khác với các chức năng khác nữa, những cửa sổ giới thiệu bên trên là những cửa sổ quan trọng nhất trong việc làm việc với công cụ IDA pro.

Menu Jump



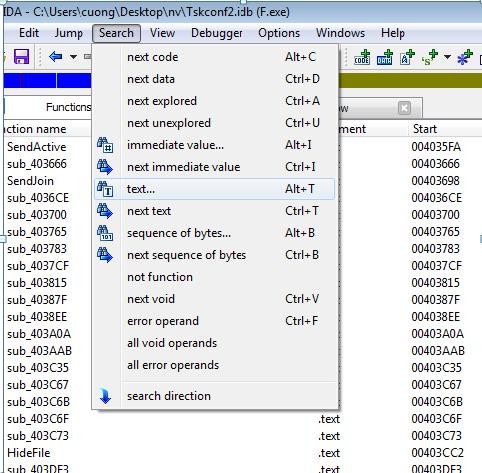
*Hình 1.14 Menu Jump*

Jump to address(G) nhiều lúc ta sẽ biết được địa chỉ chính xác mà ta muốn đến để thuận tiện cho việc này IDA cũng tạo ra 1 cửa sổ để nhập vào địa chỉ muốn jump.

Jump to entrey point( ctrl-E) hiện lên vị trí các entry point ban đầu chỉ cần nhấp vào sẽ đưa ta đến vị trí làm việc ban đầu của nó.

Sử dụng forward/ backward arrows bằng cách sử dụng nút jump trên thanh công cụ chọn jump to previous( Esc) hoặc jump to next( ctrl- enter), hoặc sử dụng nút trên màn hình làm việc.

Menu Search với các tùy chọn tìm kiếm:



*Hình 1.15 Menu search*

Next code sẽ đi tìm đến vùng code tiếp theo(vùng này đã được định nghĩa, Disasm.

Next data là sẽ đi đến vùng chứa dữ liệu tiếp theo.

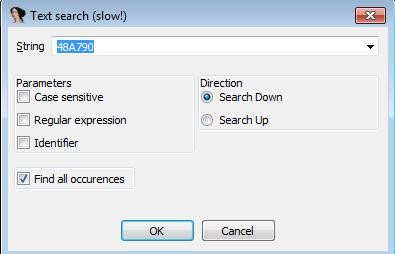
Next explored sẽ đi dến vùng dữ liệu tiếp theo được định nghĩa, sử

dụng thành struct.

Next unexplored thì ngược với next explored.

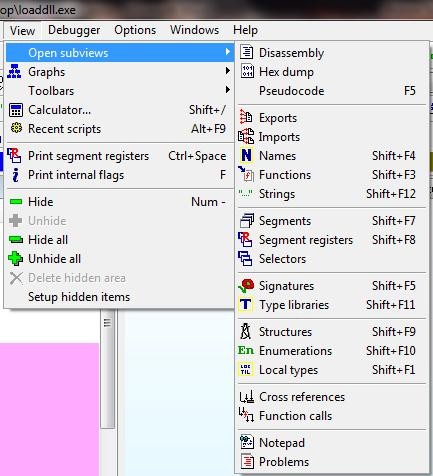
Immediate valua sẽ hiện ra cửa sổ tìm kiếm cá giá trị trong các struct, data.

Text tìm kiếm các chuỗi có ký tự cụ thể trong tất cả các hàm, dữ liệu.



*Hình 1.16 Text search*

Menu View



*Hình 1.17 Menu View*

Với open subview là mở các cửa sổ view con trên các tab với: DisAssemBly, hexdump, export import … là các cửa sổ đã có sẵn. Có thể mở thêm các cửa số như:

PseudoCode là cửa sổ chứa mã C được tái tạo lại của function hiện tại của sổ Disaseembly.

Segmen,SegmenRegister chứa các thông tin về các segmen, các thanh

gi đoạn, các vùng dữ liệu.

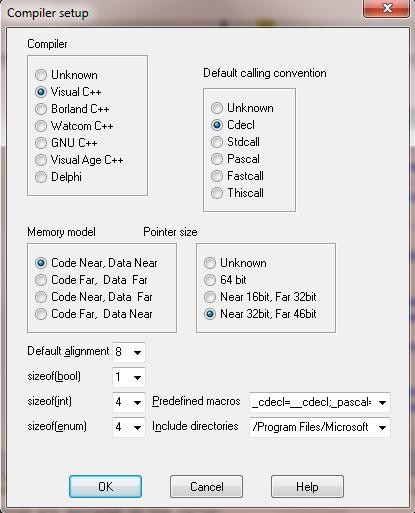
Signature,Type Lib chứa các chữ ký để xác định các chương trình dùng

thư viện gì, xây dựng trên ngôn ngữ, cơ sở gì.

Tiếp theo sẽ là menu debuger chứa các tùy chọn liện quan đến debuger gắn kèm theo IDA, nhưng ta tạm thời không xét vì các debuger này khá phức tạp, ta sẽ tìm hiểu debuger đơn giản hơn là OllyDebuger.

Menu Options

Menu này chứa các tùy chọn về complier:



*Hình 1.18 Compiler setup*

Disasembly cho phù hợp với các trình biên dịch

Tùy chọn string style chọn các kiểu string tùy theo trình biên dịch, kiểu dữ liệu.

Ngoài ra còn rất nhiều các tùy chọn liên quan dến cách dặt tên cách hiển thị biểu đồ, chú thích…

Menu Edit

Chứa các tùy chọn:

Code để định nghĩa một vùng dữ liệu bytecode thành AsmCode.

Data để địch nghĩa một vùng dữ liệu từ AsmCode thành các bytecode.

String sẽ định nghĩa lại kiểu của các string.

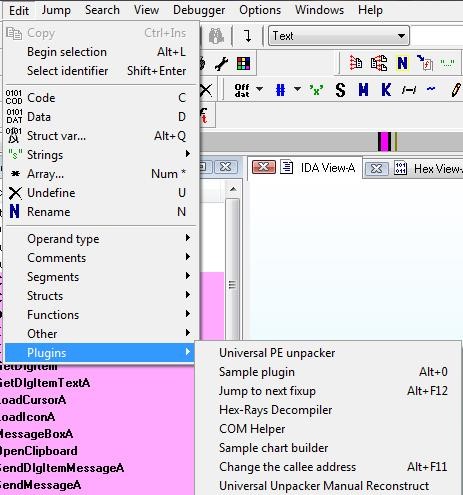
Và nhiều tùy chọn khác liên quan đến việc chỉnh sửa các segments,struct,

function…

Nhưng quan trọng nhất là Plugin nơi chứa các Pluign mà ta cài thêm vào.

Đáng chú ý nhất là Plugin HexRays Decomplier với phím tắt là F5 sẽ dịch một đoạn Assembly thành một đoạn C.

Plugin BinDiff cũng là Một plugin rất quan trọng cho phép ta so sánh cở sở dữ liệu hiện tại với một cơ sở dữ liệu khác. Chỉ ra các function tương tự, giống nhau…



*Hình 1.19 Plugin*

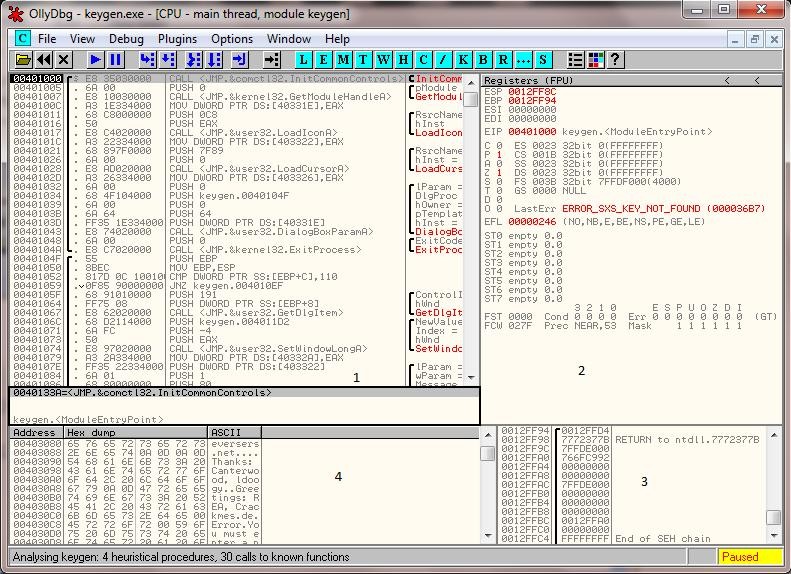
* + - 1. Olly Debug

Olly Debug là công cụ debug có từ khá lâu và phổ biến trong cộng đồng cracker và dịch ngược, nó giúp cho người sử dụng toàn quyền can thiệp vào chương trình mã độc sau khi nó được load vào bộ nhớ. Ưu điểm của Olly Debug là cho chúng ta Trace từng dòng code.

Giao diện và tính năng:

* + - * + Đầu tiên chúng ta load file vào trong ollydbg ta sẽ được giao diện chương

trình như sau:



*Hình 1.20 Giao diện Olly Debug*

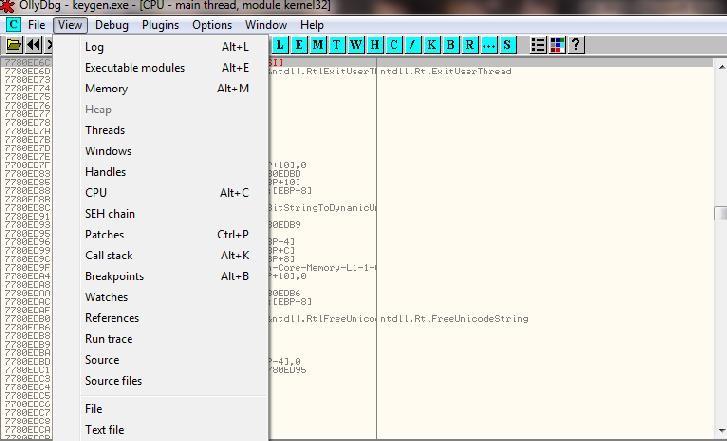
* The Dissembler Window: ở cửa sổ này ta nhìn thấy các đoạn chương

trình ở dạng asm cùng với những lời giải thích cho các đoạn mã asm.

* The Registry Window: đây là cửa sổ chứa thông tin chi tiết về các thanh ghi ,các cờ trạng thái.
* The Stack Window: hiển thị trạng thái của stack, lưu dữ tạm thời các dữ liệu và địa chỉ.
* The Dump Window: cửa sổ hiển thị nội dung của bộ nhớ hoặc file, cửa sổ này cho phép ta tìm kiếm thực hiện các chức năng chỉnh sửa…
  + - * + Các tùy chọn với View
* View --> log ( cửa sổ L ) cho chúng ta biết thông tin mà Olly ghi lại. Theo mặc định cửa sổ này sẽ lưu thông tin về các module, import libarary hoặc các Plugins được load cùng chương trình tại thời điểm đầu tiên khi load chương trình vào Olly, cùng với ghi lại các thông tin về BP. Và 1 chức năng nữa là khi ta muốn lưu lại thông tin về file Log ta chỉ viếc nhấp chuột phải trong cửa sổ L và chọn Log to file
* View --> Executable module( cửa sổ E ) cửa sổ đưa ra danh sách những file có khả năng thực thi được chương trình sử dụng như file exe, dlls, …
* View --> Memory ( cửa sổ M ) cửa sổ này cho ta biết thông tin về bộ nhớ đang được sử dụng. Tại cửa sổ này chúng ta có thể sử dụng tính năng search để tìm kiếm thông tin về các string, các đoạn hexa cụ thể

hay unicode… thêm vào đó cùng cấp cho chúng ta những kiểu thiết đặt BP khác nhau tại section.

* View --> Threads ( cửa sổ T) liệt kê các thread của chương trình.
* View --> window( cửa sổ W) để mở cửa sổ Window.
* View --> Handles( cửa sổ H) để mở cửa sổ Handles.

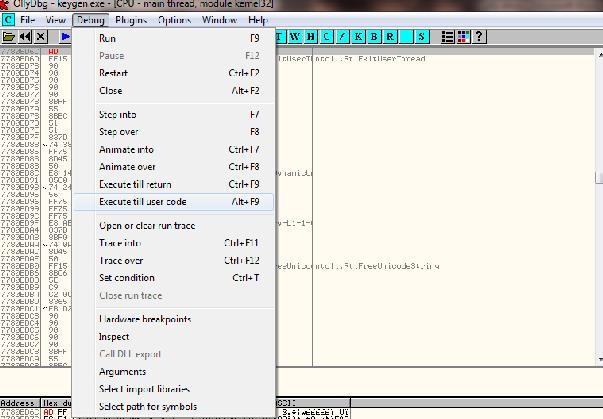


*Hình 1.21 Tùy chọn View*

* View --> CPU( cửa sổ C) cửa sổ đang làm việc hiện tại.
* View --> Patches( cửa sổ /) cửa sổ này cho chúng ta biết thông tin về những gì mà chúng ta đã edit trong chương trình.
* View --> Call stack( cửa sổ K) hiển thị một danh sách các lệnh Call mà chương trình của chúng ta đã thực hiện khi chúng ta Run bằng F9 và dùng F12 để tạm dừng chương trình.
* View --> Breakpoint( cửa sổ B) hiển thị tất cả các BP mà chúng ta đặt trong chương trình, Nó sẽ hiển thị những BP được set bằng F2 còn đối với những dạng Bp khác như HWBP và Memory BP sẽ không hiển thị.
* View --> References( cửa sổ R) hiển thi kết quả khi thực hiện chức

năng search trong Olly.

* + - * + Các tùy chọn đối với Debug
* F9( Run) khi nhấn olly sẽ tìm xem co breakpoint nào được set không, chương trình có tung ra các exception gì không, hay nếu chương trình có cơ chế chống debug thì sẽ terminate ngay lập tức.

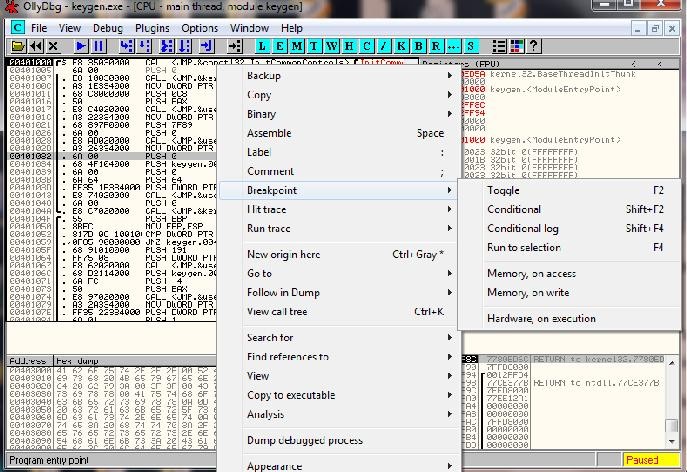


*Hình 1.22 Debug*

* F12( Pause) tạm dừng chương trình
* Ctrl-F9( Execute till return ) câu lệnh sẽ dừng lại ở RET
* Alt-F9( Execute till user code ) khi trong quá trình phân tích bị lạc vào sâu bên trong trong khi debug thì ta sẽ dùng chức năng này để đưa ta về vị trí hiện tại.
* F7( step into) thực thi trace từng dòng lệnh, trong quá rình trace gặp lệnh Call sẽ nhảy vào bên trong lệnh Call và thực thi lệnh bên trong Call, khi nào gặp return sẽ trở về chương trình chính.
* F8( step over) thực thi trace từng dòng lệnh nhưng khác với F7 là khi gặp lệnh Call nó sẽ không nhảy vào bên trong trương trình mà dừng lại ngay ở câu lệnh tiếp theo dưới lệnh Call.
* Tùy chọn Plugins
* Plugins --> Ollydump --> dump debugged process dump một vùng nhớ trong process.
* Option --> Debugging Option( Alt-O) theo mặc định chon auto start analysis thì chương trình khi đư ợc load vào sẽ được Olly tiến hành phân tích và đưa ra các comment thích hợp. Nếu không chọn chức năng auto start analysic thì chúng ta sẽ phải thực hiện manual sau khi chương trình được load vào.
* Chuột phải vào cửa sổ Disassambler có các tùy chọn đang chú ý sau:
* Binary --> edit thay đổi giá trị trong hex dump
* Goto --> expression( Ctrl+G) đưa ta đến địa chỉ ta cần.
* Goto --> previous quay trở lại call sau khi chọn follow
* New origin here( Ctrl+ Gray\*) đưa chương trình đến thực hiện ở một

địa chỉ khác tại vị trí mà ta chọn Crtl+Gray\*

* Follow( enter) cho phép ta xem câu lệnh trong call ,mà bản thân nó không hề thực thi bất kì câu lệnh nào của chương trình.Follow in dump cho ta xem giá trị tại ví trị ta chọn trong cửa sổ dump.
  + - * + Để tìm kiếm các function hay string trong ollydebug:
* Với các function ta chọn search for->all intermodular calls, hay chọn go to --> expression to fllow và nhập tên hàm vào bên trong.
* Với tìm kiếm các string ta chọn search for-> all referenced text string
* Từ việc tìm kiếm các hàm và chuỗi ta sẽ đặt một BreakPoint( BP) tại đó như một điểm đánh dấu cho ta thực hiện chương trình . Đặt BreakPoint được chia ra làm các loại sau: Common BreakPoint, Memory BreakPoint, Hardware BreakPoint,Conditional.



*Hình 2.23 BreakPoint*

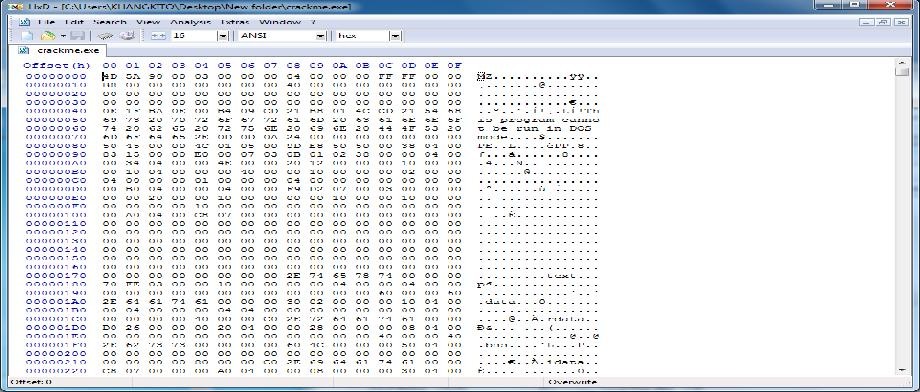
* Common BP ta đặt bằng cách tìm đến function hoặc string rồi nhấn F2, hoặc đặt thông qua command. Olly sẽ lưu dữ điểm đặt BP này tại cửa sổ BreakPoint ta có thể mở cửa sổ này ra để kiểm tra. Việc đặt BP ở trên chỉ đối với những opcode không bị thay đổi trong suốt quá trình thực hiện chương trình.
* Memory BP việc đặt BP này áp dụng cho những opcodes có thể bị thay đổi và Olly hỗ trợ chúng ta 2 kiểu đặt BP này trên memory là Breakpoint Memory on access và BreakPoint Memory on write. BreakPoint Memory on access việc đặt BP này lên một vùng nhớ sẽ cho phép ta dừng thực thi chương trình khi có bất kì sự thực thi nào, đọc hay ghi đè lên vùng dữ liệu mà ta đặt BP.BreakPoint Memory on write dừng chương trình thực thi khi có bất kì dữ liệu nào được ghi lên vùng nhớ mà đặt BP. Việc đặt BP tại memory sẽ không được lưu dữ

thông tin tại cửa sổ BreakPoint. Olly chỉ cho đặt duy nhất 1 Bp tại

memory nên khi đặt Bp thứ 2 vào thì Bp 1 sẽ tự được remove.

* Hardware BreakPoint( HWBP) ta có thể đặt được 4 HWBP nhiều hơn so với memory breakpoint tại một thời điểm chỉ đặt được BP, không sử dụng ngắt INT3 mà sử dụng ngắt INT1. HWBP được hỗ trợ trực tiếp bởi CPU và sử dụng một số thanh ghi đặc biệt gọi là debug registry. Chung ta sử dụng HWBP bởi vì nó không làm thay đổi các đoạn mã, stack. Chúng cũng không làm chậm tốc độ thực hiện. Chúng ta đặt HWBP tại đó nhưng không thấy dấu hiệu nào chứng tỏ là đã đặt cả, nên để kiểm tra xem ta đặt nó ở đâu chọn debug-> Hardware 3 Breakpoint.Với HWBP on write và HWBP on access thì ta chỉ cần bôi đen byte, word, Dword tùy ý muốn ở cửa sổ dump, sau đó đặt BP.
* Conditional Breakpoint( shift+F2) cũng giống BP thông thường, tuy nhiên việc đặt Conditional BP phải thỏa mãn một điều kiện đã đư ợc thiết lập từ trước. Ta ấn chọn BP-> Conditional nhập vào điều kiện bên trong ô đó. Được lưu trong bảng Breakpoint.
* Conditional log Breakpoint( shift+F4) cũng giống như conditional BP nhưng nó có thêm tùy chọn cho phép ta lưu vết giá trị của biểu thức hoặc các tham số funtion mỗi khi xảy ra Bp hoặc khi thỏa mãn điều kiện mà ta yêu cầu. Những thông tin lưu dấu vết này sẽ được lưu tại Log(L) của Olly
  + - 1. HxD

Là công cụ biên tập mã hex nhanh chóng, cho phép người dùng chỉnh sửa nội dung file, xác định các thành phần của PE file nếu người dung có kiến thức về PE file. Đây là công cụ hiệu quả khi đã xác định được vị trí của chuỗi nhận dạng để đưa vào cơ sở dữ liệu mà không cần dung đến công cụ bên thứ ba.



*Hình 1.24 Giao diện HxD*

## Quy trình phân tích mã độc hại

Đầu tiên ta sẽ tìm hiểu về các phương pháp phân tích mã độc từ đó sẽ đưa ra

quy trình phân tích mã độc.

* + 1. Các phương pháp phân tích mã độc

Phương pháp phân tích thì có 2 phương pháp chính :

* + - * Dynamic analysis (phân tích hành vi mã độc hại).
      * Static analysis (phân tích mã độc hại bằng cách xem mã dịch ngược của mã

độc hại).

Tuy nhiên không phải mã độc nào cũng phân tích theo hai phương pháp trên, bởi vì có những mã độc có thể chỉ cần phân tích động hoặc có những loại chỉ cần phân tích tĩnh. Vì vậy cần phải có cách thức phân tích riêng của mỗi người dựa trên kinh nghiệm và thực tế phân tích để tiến hành tránh mất thời gian cũng như công sức bỏ ra. Chính vì thế hai phương pháp trên chỉ là tương đối.

* + - 1. Static analysis

Static analysis thì ta sẽ xem code để xem thực sự mã độc hại này sẽ làm gì trên hệ thống. Không cần phải chạy mã độc hại đó, và hiểu rõ hơn thực sự về hoạt động của malware. Các công cụ thường sử dụng trong quá trình phân tích tĩnh có thể kể đến các chương trình dịch ngược :

* + - * + Disassembler
        + Decompiler
        + Source Code Analyzer

Ưu điểm :

Static analysis là có thể phát hiện ra hoạt động, cách ứng xử của chương

trình trong những điều kiện không tồn tại thực tế.

Static analysis sẽ cho chúng ta cái nhìn rất đúng nhất về một chương trình. Sở dĩ gọi là gần đúng bởi rất hiếm khi chúng ta có thể nắm được toàn bộ hoạt động của một chương trình, trừ các chương trình đủ nhỏ.

Nhược điểm :

Phương pháp này đòi hỏi người phân tích phải am hiểu sâu về hệ thống và lập trình.

Phương pháp này đôi khi rất mất thời gian thậm chi có những loại mã độc mà thời gian phân tích của nó tính bằng tháng.

* + - 1. Dynamic analysis

Dynamic analysis thì là quan sát xem mã độc hại khi thực thi thì sẽ làm những gì, nó chạy ra sao, làm gì trên máy tính mình qua các công cụ monitor, cách này thì có nhược điểm với các dòng mã độc hại chạy theo lịch trình. Nghiên cứu hoạt động của chương trình bằng cách thực thi chương trình đó. Các công c ụ được sử dụng trong trường hợp này có thể kể đến như :

* + - * + Debugger
        + Function call tracer
        + Machine emulator
        + Logic analyzer
        + Network sniffer

*Ưu điểm :*

Dynamic analysis thì nhanh và thông tin chính xác. Tuy nhiên dynamic analysis có một khuyết điểm là: "Tất cả những thấy không phải là tất cả những gì chương trình có". Nói cách khác dynamic analysis không thể dự đoán được những hành vi của chương trình trong các điều kiện "đặc biệt" không tồn tại trong thực tế. Có thể lấy ví dụ với các mã độc hại chạy theo thời gian, tức là ở thời điểm này mình cho nó chạy nó không có hoạt động gì, nhưng ở một thời điểm khác nó lại chạy.

Mỗi phương pháp đều có một đặc điểm riêng nhưng khi phân tích mã độc hại thì người ta thường phải sử dụng cả 2 phương pháp để hỗ trợ lẫn nhau:

Dù mục đích chính thì giống nhau nhưng các công cụ hỗ trợ việc phân tích, thời gian bỏ ra của từng phương pháp là khác nhau.

* + 1. Các bước cơ bản phân tích mã độc hại
       1. Phân tích thông tin sơ bộ

Đầu tiên nên nhìn nhận dấu hiệu trên máy bị nhiễm có những hiện tượng gì ? Máy chậm hay lưu lượng mạng tăng, từ đó kiểm tra xem có các processes lạ trên máy bị nhiễm không, có những bất thường xảy ra trên máy tính đó không?... Sau đó là thu thập mẫu nghi ngờ là mã độc hại.

Nhận diện hệ thống bị nhiễm mã độc hại :

* + - * + Thông tin về hệ điều hành đang sử dụng.
        + Trình duyệt Web.
        + Firewall.
        + Các trình bảo vệ máy hiện thời đã có.

Từ đó ta xác định được các modules, các processes, các dịch vụ, các drivers, các add-on trình duyệt, phiên bản hệ điều hành… của hệ thống. Mục đích thu thập thông tin này để sau này còn có thể xác định nguồn gốc lây lan và tại sao lại bị nhiễm thông qua đâu. Đồng thời cũng ra soát lại một lượt xem hệ thống mình có dính lỗi bảo mật nào không.

Dự đoán đặc điểm, phân loại mã độc hại, chia dòng, đặt tên. Bằng cách xem mã hex, xem properties của files, kích thước file… hoặc thông tin đơn giản khác để chúng ta tiến hành phân loại sơ bộ mã độc hại.

Tìm kiếm thông tin : Có thể upload mã độc hại lên các trang như virustotal, mcafee hoặc scan bằng các chương trình antivirus xem chúng đã được nhận diện chưa. Nếu mã độc hại đã được nhận diện, chúng ta sẽ tìm kiếm thêm thông tin về mã độc hại đó làm cơ sở để thực hiện việc phân tích chi tiết.

* + - 1. Quan sát hành vi mã độc hại (Dynamic analysis)

Thiết lập môi trường thử nghiệm: Sử dụng máy ảo, sandbox, vitural PC để cho mã độc hại chạy. Nếu mã độc hại có cơ chế phát hiện môi trường ảo có thể phải sử dụng môi trường thật đã được giới hạn (trong LAN) để thử nghiệm. Thiết lập mạng Internet, web, mail, cài đặt các hệ thống lỗi nếu mã độc hại có khai thác lỗi... Môi trường thử nghiệm càng đầy đủ, càng "thật" thì sẽ quan sát được càng nhiều đặc điểm của mã độc hại. Nếu xây dựng được hệ thống máy như máy bị nhiễm thì là tốt nhất.

Thiết lập các công cụ giám sát: Vì không thể monitor virus bằng mắt nên

trong môi trường thử nghiệm cần có các công cụ monitor chuyên dụng như:

* + - * + File monitor
        + Registry monitor
        + Process monitor
        + Network monitor
        + Các tool phát hiện rootkit

 .......

Bộ công cụ này sẽ giúp chúng ta quan sát mã độc hại tốt hơn. Bắt đầu chạy mã độc hại và quan sát các thông tin :

* + - * + Khảo sát processes xem có processes lạ nào đáng ngờ không ?
        + Khảo sát các modules dlls có gắn vào các process hệ thống không?
        + Khảo sát registry xem có process nào cùng khởi động với hệ điều hành có

key nào được sinh ra và bị sửa đổi không ?

* + - * + Kiểm tra MD5, CRC, SHA các modules (files) đang chạy trên hệ thống xem có bị mã độc hại gắn vào không ?
        + Khảo sát các files, các folder ẩn để tìm các file nghi ngờ.
        + Khảo sát các driver, tìm rootkit làm ẩn tiến trình, ẩn key.
        + Khảo sát lưu lượng mạng (Dùng Dumetter) xem có tăng đột ngột không?
        + Kiểm tra xem các kết nối TCP/IP trên máy.

Chạy mã độc hại và ghi log, quan sát càng lâu kết quả càng chính xác. Thông thường mã độc hại sẽ được quan sát cho đến khi chúng không còn hoạt động gì đáng kể hoặc hành động lặp đi lặp lại. Việc ghi log chủ yếu do các công cụ monitor chuyên dụng ở trên thực hiện.

Phân loại virus và ghi nhận các đặc điểm: Từ các thông tin trên chúng ta sẽ tiến hành phân loại virus: là worm hay trojan, backdoor, virus lây file... có rootkit hay không.

* + - 1. Phân tích mã dịch ngược ( Static Analynis)

Sau khi phân tích mã độc hại bằng cách quan sát hành vi thì ta sẽ thu được một số thông tin, nhưng như vậy chưa đủ, ta vẫn cần dịch ngược mã độc hại để xem chi tiết rõ hơn liệu mã độc hại còn làm gì nữa không vào thời điểm nào nữa.

Do mã độc hại khi ta thu được là ở dạng thực thi không có mã nguồn đi kèm theo để phân tích nên thường phải dịch ngược mã độc hại đó về mã assembly để phân tích code. Thường ở bước này sẽ ta sẽ làm những việc như sau:

* + - * + Xem mã độc hại được viết bằng ngôn ngữ gì ?
        + sử dụng các packer để nén lại hay các protecter nào không ?
        + Khi biết mã độc hại được pack bằng trình nào thì có thể dùng trình đó unpack

ra, tương tự cũng unprotecter file mã đ ộc hại.

* + - * + Dùng các công phụ phù hợp với công cụ viết mã độc hại để decompile, disassemble, debugging.
        + Đọc và phân tích mã code assembly để tìm thêm hoạt động của mã độc hại.
      1. Trace Code Debug

Khi việc dịch ngược tỏ ra khó khăn hoặc thiếu hiệu quả. Hoặc khi cần làm thật rõ một đoạn mã người ta phải debug mã độc hại. Có nghĩa là sẽ lần từng lệnh nhỏ của mã độc hại để xem chính xác là mã độc hại đã làm những gì. Mã độc hại

thường có các kỹ thuật chống dịch ngược thì cũng sẽ có các kỹ thuận chống debug

(anti debug) và người phân tích phải vượt qua nó (anti anti-debug).

* + - 1. Tìm signature của mã độc hại và đưa vào c ơ sở dữ liệu

Tìm đặc điểm nhận diện mã độc hại trong hệ thống sau khi phân tích (ví dụ tại offset nào đó chứa string gì, hoặc MD5 của file).

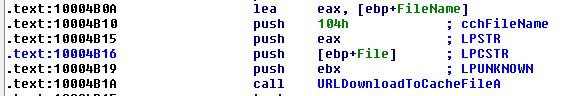
Từ những đặc điểm nhận dạng trên sẽ đưa vào cơ sở dữ liệu.

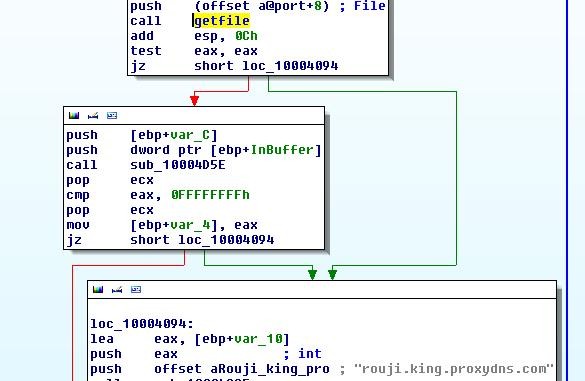
* + 1. Phân tích môt mẫu mã độc cụ thể

Dưới đây là quá trình phân tích một mẫu mã độc cụ thể có tên là netui.dll được trích xuất ra từ một file excel. Ở đây em chỉ dung chương trình IDA vì đề tài không đi sâu vào phân tích.

* + - * Sử dụng IDA tiến hành disasembly

Get file “ )(!@PORT” về máy tính nạn nhân sử dụng hàm : URLDownloadToCacheFile từ địa chỉ *rouji.king.proxydns.com*





* + - * Ta thấy có hàm GetKeystate được dùng:

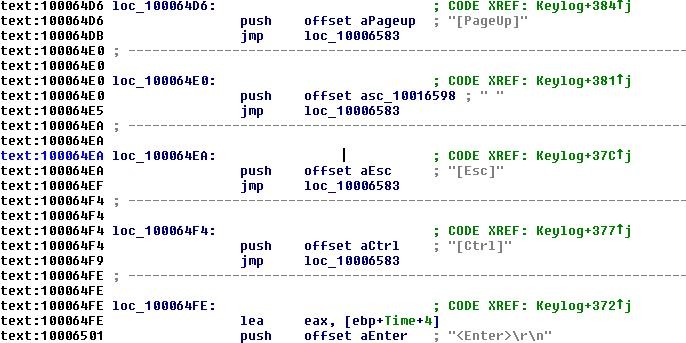


Từ đây ta tìm ra được đoạn code sử dụng hàm này và thấy rằng nó được dùng với mục đích là ghi lại trạng thái bàn phím.

Dưới đấy là đoạn chứa hàm keylog



Một phần đoạn code của hàm keylog



**Kết Luận**: Do không phải là đề tài đi sâu vào phân tích mã độc nên mẫu trên em chỉ phân tích sơ qua và đưa ra kết luận là mẫu rên là mọt thư viện động chưa các hàm có những hành động là một keyloger.

# CHƯƠNG 2. NGHIÊN CỨU VỀ MỘT SỐ CƠ SỞ DỮ

**LIỆU MÃ ĐỘC**

Để xây dựng một cơ sở dữ liệu mã độc trước tiên cần phải có kiến thức về việc phát hiện mã độc bằng cách nào, các kỹ thuật thu thập mã và phát hiện mã độc. Chương này sẽ trình bày các dấu hiệu phát hiện mã độc, chuẩn trao đổi mã độc đang được sử dụng hiện nay.

## Các kiểu dấu hiệu mã độc và kỹ thuật phát hiện tương ứng

Phương pháp này sử dụng một cơ sở dữ liệu có sẵn khi quét các chương trình quét chỉ việc quét các tập tin và so sánh với cơ sở dữ liệu có sẵm. Cơ sở dữ liệu mỗi hãng phát triển phần mềm diệt virus được tạo ra theo những cách khác nhau cũng với đó là thuật toán quét sao cho nhanh và hiệu quả nhất. Chúng đều được các hãng giữ kín và thường không tiết lộ ra ngoài.

* + 1. String – Chuỗi

Phương pháp phát hiện theo chuỗi là phương pháp đơn giản nhất để phát hiện mã độc. Nó sử dụng một chuỗi các byte hoặc kỹ tự chỉ có trong mã độc mà không có trong các chương trình bình thường khác. Các chuỗi sau khi đã được trích xuất ra được lưu trong một cơ sở dữ liệu, các chương trình phát hiện mã độc sẽ sử dụng cơ sở dữ liệu này để phát hiện các mã độc.

Phương pháp này có nhược điểm là đòi hỏi thuật toán quét phải tốt nếu không thời gian quét sẽ rất lâu nếu lượng tệp tin cần quét là lớn hoặc cơ sở dữ liệu lơn. Một nhược điểm nữa là việc tìm chuỗi nhận dạng này cần đối với mỗi hãng phát triển phần mềm diệt virus là khác nhau nên quá trình trao đổi dữ liệu giữa các hãng là không có.

Phương pháp này có ưu điểm là đối với các biến thể khác nhau của cùng một loại mã độc thì chuỗi nhận dạng vẫn giống nhau nếu việc tìm chuỗi nhận dạng tốt. Thường các loại mã độc ngay cả các mã độc tự biến đổi mã của mình vẫn giữa nguyên các chuôi nhận dạng của chúng.

Dưới đây là một chuỗi nhận dạng của virus Stoned:

*0400 B801 020E 07BB 0002 33C9 8BD1 419C*

Chuỗi này là chuỗi nhận dạng 16 byte thường được sử dụng với virus 16 bit. Đối với virus 32 bit thường sử dụng chuỗi 32 byte.

Em sẽ tiến hành xây dựng một chương trình demo quét mã độc theo một cơ

sở dữ liệu là chuỗi nhận dạng có sẵ n.

* + 1. Ký tự đại diện

Ký tự đại diện cũng giống như chuỗi nhận dạng tuy nhiên khi các chương trình phát hiện mã độc sử dụng cơ sở dữ để quét thì nó được phép bỏ qua một byte hoặc một dãy byte.

Ví dụ:

*0400 B801 020E 07BB ??02 %3 33C9 8BD1 419C*

Bắt đầu duyệt:

* + - * Nếu khớp với 04 thì tiếp tục
      * Khớp với 00 thì tiếp tục
      * Tiếp tục cho đến khi gặp ?? thì bỏ qua byte này
      * Khớp với 02 thì tiếp tục
      * Gặp %3 33 thì nghĩa là tìm khớp 33 với 3 vị trí sau nếu khớp thì tiếp tục
      * Duyệt đến hết chuỗi nếu khớp hết thì gửi lại kết quả

Ký tự đại điện thường được sử dụng đối với các byte bị mất cho phép quét một cách chính xác hơn. Đối với với mã độc được mã hóa, virus đa hình thời đầu có thể dễ dàng bị phát hiện bởi phương pháp này.

* + 1. Mã băm

Mã băm là thuật ngữ chung cho kỹ thuật tăng tốc tìm kiếm các thuật toán. Nó có thể được thực hiện trên các byte đầu tiên từ 16 bit và 32 bit của chuỗi quét. Điều này cho phép thêm byte để chứa ký tự đại diện. Các nhà nghiên cứu mã độc có thể kiểm soát mã băm tốt hơn bằng cách chọn lọc những byte bắt đầu của chuỗi chứa nó.

Việc lấy mã băm có thể là băm toàn bộ tệp tin hoặc một phần tệp tin như phần đầu hay phần cuối tệp tin. Mã độc thường chèn đoạn code thực thi của chúng vào đầu hoặc vào cuối tệp tin nên việc băm một phần sẽ giúp giảm thời gian băm và cơ sở dữ liệu băm có hiệu suất cao hơn.

Nhược điểm của mã băm là đối với các mã độc có khả năng tự biến đổi hay xáo trộn mã thì sẽ gặp khó khăn vì có thể cùng một loại mã độc nhưng sẽ có nhiều mã băm khác nhau.

Dưới đây là một đoạn code lấy mã băm là MD5 viết bằng C# :

*private void ToMD5(string FilePath)*

*{*

*//tạo đối tượng mã hóa MD5 MD5 MyMD5 = MD5.Create();*

*//đọc file*

*FileStream fs = new FileStream(FilePath, FileMode.Open);*

*//mã hóa mảng byte bằng MD5*

*byte[] HashCode = MyMD5.ComputeHash(fs);*

*//chuyển mảng byte thành chuỗi StringBuilder SB = new StringBuilder(); for (int i = 0; i < HashCode.Length; i++)*

*SB.Append(HashCode[i].ToString("x2")); stMD5 =SB.ToString();*

*fs.Close();*

*}*

* + 1. Khung mã độc có sẵn

Phương pháp này được tạo bởi Eugene Kaspersky. Phương pháp khung mã độc có sẵn rất hữu ích trong việc phát hiện họ virus macro. Thay vì lựa chọn một chuỗi đơn giản hoặc một kiểm tra của tập các virus macro, chương trình quét phân tích các dòng báo cáo vĩ mô để lược bỏ các thông báo không cần thiết. Kết quả là xác định được các mã vĩ mô nằm trong virus macro sau đó so sánh với cơ sở dữ liệu có sẵn.

Phương pháp này dung để phát hiện rất tốt các biến thể của một họ virus.

* + 1. Phương pháp dựa trên hành vi

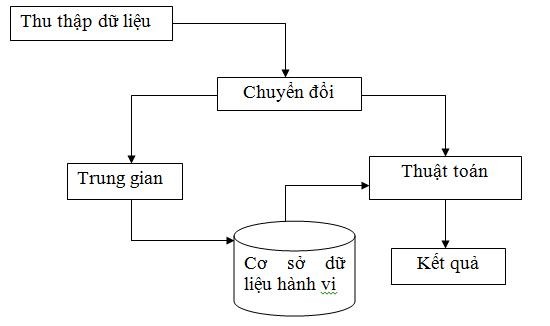
Phương pháp phát hiện dựa trên hành vi khác với việc phát hiện đựa vào bề ngoài, cấu trúc tệp tin có sẵn là nó xác định các hành động thực hiện của mã độc

hơn là việc xác định cấu trúc nhị phân của chương trình. Các chương trình không giống với cú pháp hay cấu trúc nhưng có hành vi giống với những hành vi đã xác định trước là đã xác định được nó là mã độc hay không.

Cơ chế này giúp cho việc xác định mã độc một cách hiệu quả đối với các loại mã độc không ngừng tạo ra các biến đổi moeis của nó. Phương pháp phát hiện hành vi luôn theo dõi các biến đổi về tài nguyên hệ thống và các dịch vụ mà các mã độc khi sử dụng sẽ ngay lập tức bị theo dõi và quan sát hành vi.

Một chương trình phát hiện hành vi gồm các thành phần sau:

* + - * Thu thập dữ liệu: Thành phần này thu thập các thông tin động và tĩnh được ghi lại.
      * Chuyển đổi: Thành phần này sẽ chuyển các thông thu thập được bởi module thu thập dữ liệu vào nơi trung gian để lưu vào cơ sở dữ liệu.
      * Thuật toán so sánh: Được sử dụng để so sánh các phần đại điện với chữ ký hành vi.



*Hình 2.1 Mô hình chương trình quét hành vi*

* + 1. Kỹ thuật lọc

Kỹ thuật lọc sử dụng trong các chương trình quét sau này. Các mã độc lây nhiễm thường là tập hợp con của các loại đối tượng đã biết. Do đó các chương trình quét có thể tiến hành quét một cách rất nhanh sử dụng kỹ thuật này. Ví dụ như chữ ký của một loại virus bị giới hạn để tự khởi động, trong hệ điều hành DOS thì chữ ký tệp tin exe có đặc điểm khác. Vì vậy một cờ được thêm vào chuỗi nhận dạng để

xác định xem có chuỗi ký tự này trong đối tượng cần quét hay không. Do đó nó

làm giảm thời gian quét của chương trình phải thực hiện

Chương trình quét phục thuộc rất nhiều vào bộ lọc. Bộ lọc có tốt thì thuật toán quét mới hiệu quả, thời gian mới rút ngắn. Một bộ lọc có thể là : các loại thực thi, dấu nhận dạng mã độc, tên mã…Tuy nhiên có một số mã độc việc tìm kiếm bộ lọc rất phức tạp thậm chí là không thể tìm được điển hình là các virus đa hình hay mã độc tự biến đổi mã hay mã hóa.Những loại này được phát hiện tốt hơn với kỹ thuật phát hiện giải mã tĩnh

* + 1. Phát hiện bằng việc giải mã tĩnh

Kỹ thuật này sử dụng việc giải mã tĩnh trong một mã độc cụ thể sử dụng tất cả các cách trong toàn bộ phần mã của nó. Như vậy tốc độ quét phục thuộc nhiều vào kích thước của phần mã chương trình được quét. Kỹ thuật này tương đối nhanh nếu kết hợp với kỹ thuật lọc. Tuy nhiên nó cũng dễ gấy ra các thông báo sai và không đảm bảo với những mã độc khi chạy mà không cần giải mã.

* + 1. Mã giả lập

Là kỹ thuật rất mạnh mẽ trong việc phát hiện mã độc. một máy ảo được mô phỏng các hệ thống CPU và bộ nhớ để bắt trước các thực thi mã. Do đó mã độc hại được mô phỏng trong các máy ảo quét và không có mã của mã độc thực tế nào thực hiện bởi bộ vi xử lý.

Công cụ mã giả lập:

* + - * IDA pro là bộ đa xử lý dung để disassembler và debug cũng cấp các tính năng. Chương trình này sẽ được nêu chi tiết ở phần “**3.2 Các phương pháp và công cụ thu thập và tạo mẫu của các hãng trên thế giới**”.
      * Ollydebug là chương trình debug mạnh mẽ giúp cho ta có thể trace từng dòng code, theo dõi từng hành vi của mã độc. Chi tiết về olly debug cũng sẽ được nêu ở phần “**3.2 Các phương pháp và công cụ thu thập và tạo mẫu của các hãng trên thế giới**”

Phương pháp đầu tiên của mã giả lập là sử dụng trình debug để theo dõi các mã sử dụng bộ vi xử lý. Tuy nhiên giải pháp này không đủ an toàn do mã các mã độc có thể nhảy ra ngoài nơi mô phỏng trong môi trường phân tích với việc sử dụng các kỹ thuật antidebug hay anti disassembly hoặc anti máy ảo.

Dưới đây là một ví dụ về các thanh ghi và cơ 16 bit định nghĩa với các cấu trúc trong ngôn ngữ C:

*Typedef struct*

*{ byte ah,al,bh,bl,ch,cl,dh,dl;*

*word si,di,sp,bp,cs,ds,es,ss,ip;*

*} Emulator\_Registers\_t;*

*typedef struct {*

*byte c,z,p,s,o,d,i,t,a;*

*} Emulator\_Flags\_t;*

## Nghiên cứu chuẩn trao đổi dữ liệu mã độc

Hiện nay do việc tìm và diệt mã độc càng ngày càng trở lên cấp thiết đối với không chỉ người dung mà cả đối với an ninh quốc gia nên việc chia sẻ cơ sở dữ liệu mã độc giữa các hang hay các tổ chức, quốc gia là cần thiết. Chính vì vậy một chuẩn trao đổi dữ liệu đã được ra đời.

Chuẩn trao đổi dữ liệu mã độc tạo ra bởi ICSG Malware Working Group. ICSG là viết tắt của Industry Connection Sercurity Group, nó ra đời nhằm thúc đẩy việc hợp tác và chia sẻ thông tin trong ngành bảo mật.Mục tiêu của Malware Working Group là giải quyết các vấn đề về phần mềm độc hại mà ngành công nghệ thông tin hiện nay phải đối mặt.

Trọng tâm ban đầu có mục đích là thành lập cách thông minh hơn để chia sẻ các mẫu mã độc hại và các thông tin liên quan tới chúng trong ngành bảo mật máy tính đạt hiệu quả cao hơn.

Các nhóm làm việc giải quyết các vấn đề của việc đóng gói mã độc.

* + - Các tài liệu phục vụ tốt nhất trong việc sử dụng được đống gói bởi các nhà phát triển phần mềm.
    - Xác định các thuộc tính của việc đống gói tập trung vào các tính năng

mà mã độc thường sử dụng.

* + - Tạo một khóa đăng ký của việc đóng gói và một tập hợp các tên các việc đống gói đó.
    - Thiết lập một định dạng chia sẻ dữ liệu để chia sẻ thông tin đóng gói.
    - Phát triển và thực hiện “Hệ thống Taggant” – nhúng một mã băm mạnh vào từng đối tượng để đóng gói để phát hiện ra nguồn các tập tin đóng gói. Taggant là một dấu hiệu được bổ sung trong quá trình tạo ra sản phẩm. Một khi thư viện taggant được thực hiện các nhóm sẽ thúc đẩy

và tạo điều kiện giám stas việc triển khai nó đối với cả bên phát triển Anti Virus và bên cung cấp mẫu đống gói.

Để tham gia nhóm Malware Working Group thì các tổ chức cá nhân cần trở thành thành viên của ICSG. Chỉ có các thành viên ICSG mới có quyền tham gia trao đổi đưa ra các ý tưởng giải pháp trong việc đưa ra chuẩn. Tuy nhiên cũng có những chuyên gia được mời nhưng họ chỉ tham gia việc phát triển mà không có quyền quyết định.

Dưới đây là dạng của chuẩn trao đổi mã độc:

<malwareMetaData [xmlns="http://xml/metadataSharing.xsd](http://xml/metadataSharing.xsd)" xmlns:xsi="[http://www.](http://www/) w3.org/2001/XMLSchema- instance"xsi:schemaLocation="<http://xml/metadataSharing.xsd> file:metadataSharing.xsd" version="1.200000" id="10000">

<company>N/A</company>

<author>MMDEF Generation Script</author>

<comment>Test MMDEF v1.2 file generated using genMMDEF</comment>

<timestamp>2011-08-19T13:50:21.721000</timestamp>

<objects>

<file id="c7ae4ffe33fc841aea2e0113afa05fdf">

<md5>c7ae4ffe33fc841aea2e0113afa05fdf</md5>

<sha1>25daac9d19f18b5ac19769dcf7e5abc154768641</sha1>

<sha256>e3b0c44298fc1c149afbf4c8996fb92427ae41e4649b934ca495 991b7852b855</sha256>

<sha512> cf83e1357eefb8bdf1542850d66d8007d620e4050b5715dc83f4a921d36c e9ce47d0d13c5d85f2b0ff8318d2877eec2f63b931bd47417a81a538327a f927da3e

</sha512>

<size>1546790</size>

<filename>ProcessExplorer.zip</filename>

<MIMEType>application/zip</MIMEType>

</file>

<file id="d22ff2cc70fa2eec94aaa6c6f49e6eb0">

<md5>d22ff2cc70fa2eec94aaa6c6f49e6eb0</md5>

<sha1>2458a3d696698e2c4550b91e54ff63f4b964198d</sha1>

<sha256> 6ff22c87fb5ee105b33346dbb3f13f3049a292981e9df1eb8591e858ccf4 d782

</sha256>

<sha512> 34e18bf9679c71189383bcd89c9f723383715bbf63f051edd659c57e14d0 12987c33ba67fbbb0faeca962b3ec7b12b0aa24b3c134ddbb9f905aa2660 4718f375

</sha512>

<size>7005</size>

<crc32>1185414000</crc32>

<filename>Eula.txt</filename>

<filenameWithinInstaller>Eula.txt</filenameWithinInstaller>

<MIMEType>text/plain</MIMEType>

</file>

<file id="ae846553d77284da53abcd454b4eaedf">

<md5>ae846553d77284da53abcd454b4eaedf</md5>

<sha1>782333340d56a8f020a74bd2830e68f31310a5b7</sha1>

<sha256>

a8f4bd956816960691bc08bf94be342a6d62bf6d91c92f7e7506903ffda5 0b83

</sha256>

<sha512> bf1fd7b27234d5605731d21358bba01738098cb363a6deee79ed88699a39 27b3a09d9044767db24ff6ddd028fbe0f4a1572f9af4d4ab4996bfdefe2b 950a9b49

</sha512>

<size>72268</size>

<crc32>2807815698</crc32>

<filename>procexp.chm</filename>

<filenameWithinInstaller>procexp.chm</filenameWithinInstalle r>

</file>

<file id="4edc50d3a427566d6390ca76f389be80">

<md5>4edc50d3a427566d6390ca76f389be80</md5>

<sha1>9cb1bd5dc93124f526a1033b1b3f37cc0224a77e</sha1>

<sha256> e942d28c0e835b8384752731f1b430cb3fbd571381666ded7637a2db47fa fcc0

</sha256>

<sha512> 3ceb1bd07af9e470ff453ef3dd4b97f9228856cb78eb5cddb7b81796b4b8 30368e3ed2f0c6a9ce93009397e8158c68dba67e398f58df87137d8872cb 0bb3b53b

</sha512>

<size>3412856</size>

<crc32>1119775926</crc32>

<filename>procexp.exe</filename>

<filenameWithinInstaller>procexp.exe</filenameWithinInstalle r>

<MIMEType>application/octet-stream</MIMEType>

</file>

<softwarePackage id="procexp">

<vendor>Sysinternals</vendor>

<product>Process Explorer</product>

<version>14.11</version>

<language>English</language>

</softwarePackage>

</objects>

<relationships>

<relationship type="createdBy" id="1">

<source>

<ref>file[@id="c7ae4ffe33fc841aea2e0113afa05fdf"]</ref

>

</source>

<target>

<ref>file[@id="d22ff2cc70fa2eec94aaa6c6f49e6eb0"]</ref

>

<ref>file[@id="ae846553d77284da53abcd454b4eaedf"]</ref

>

<ref>file[@id="4edc50d3a427566d6390ca76f389be80"]</ref

>

</target>

<timestamp>2011-08-19T13:50:21.924000</timestamp>

</relationship>

<relationship type="partOfPackage" id="2">

<source>

<ref>softwarePackage[@id="procexp"]</ref>

</source>

<target>

<ref>file[@id="d22ff2cc70fa2eec94aaa6c6f49e6eb0"]</ref

>

<ref>file[@id="ae846553d77284da53abcd454b4eaedf"]</ref

>

<ref>file[@id="4edc50d3a427566d6390ca76f389be80"]</ref

>

<ref>file[@id="c7ae4ffe33fc841aea2e0113afa05fdf"]</ref

>

</target>

<timestamp>2011-08-19T15:50:21.924000</timestamp>

</relationship>

</relationships>

</malwareMetaData>

Chuẩn trao đổi được lưu dưới dạng file XML. XML viết tắt của Extensible Markup Language (ngôn ngữ đánh dấu mở rộng) là một ngôn ngữ đánh dấu sử dụng để tạo thẻ riêng cho người sử dụng. Nó được tạo bởi liên minh mạng toàn cầu nhằm khắc phục những hạn chế của HTML. File dữ liệu chuẩn trao đổi có thuộc tính cha là “malwareMetaData” và nó có những thuộc tính con: company,author, objects,relationships… và chúng lại chứa các thuộc tính con khác nữa như: name, md5,sha1,sha256 …Vì thời lượng và kiến thức nên chương trình quản lý dữ liệu của em sẽ thực hiện với một số các thuộc tính trên mà sẽ không dùng hết các thuộc tính. Em sẽ trình bày chi tiết hơn khi bảo vệ đồ án.

## Nghiên cứu cơ sở dữ liệu mã độc Clam AV từ đ ó xây dựng cơ sở dữ liệu

**mẫu mã độc**

* + 1. Clam Anti Virus

Clam Anti Virus - viết tắt là là ClamAV - là một phần mềm diệt virus mã nguồn mở được thiết kế để phát hiện các loại mã độc như: trojan, virus,… và các mối đe dọa khác. Nó có chuẩn cho việc quét thư điện tử. Nó cung cấp một hiệu suất quét đa luồng cao, tiện ích dòng lệnh cho viêc quét file theo yêu cầu và một công cụ thông minh để cập nhật chữ ký tự động. Cốt lõi của LamAV là cung cấp nhiều cơ chế định dạng tập tin phát hiện, hỗ trợ giải nén tập tin, hỗ trợ lưu trữ và có nhiều dạng kiểu chữ ký để phát hiện các mối đe dọa. Cốt lõi thư viện của ClamAV được sử dụng trong Immunet 3.0, được hỗ trợ bởi ClamAV. Nó xử lý nhanh, đầy đủ các tính năng là giải pháp cho máy tinh sử dụng Windows.

* + 1. ClamAV Virus Databases

ClamAV Virus Databases - viết tắt là CVD – là nơi chứa các chữ ký số bao gồm cơ sở dữ liệu chữ ký trong các định dạng văn bản khác nhau. Các tiêu đề của các nơi chứa này là chuỗi ký tự có độ dài 512 byte với các trường riêng biệt :

*ClamAV-VDB:build time:version:number of signatures:functionality level required:MD5 checksum:digital signature:builder name:build*

*time (sec)*

*sigtool –info* hiện thị thông tin chi tiết tệp tin trong CVD*: zolw@localhost:/usr/local/share/clamav$ sigtool -i main.cvd File: main.cvd*

*Build time: 09 Dec 2007 15:50 +0000*

*Version: 45*

*Signatures: 169676*

*Functionality level: 21 Builder: sven*

*MD5: b35429d8d5d60368eea9630062f7c75a*

*Digital signature: dxsusO/HWP3/GAA7VuZpxYwVsE9b+tCk+tPN6OyjVF/U8*

*JVh4vYmW8mZ62ZHYMlM903TMZFg5hZIxcjQB3SX0TapdF1SFNzoWjsyH53eXv MDY*

*eaPVNe2ccXLfEegoda4xU2TezbGfbSEGoU1qolyQYLX674sNA2Ni6l6/CEKYYh Verification OK.*

Project ClamAv phân phối dưới các tập tin của file CVD gồm: main.cdv và daily.cvd

* + 1. Debug thông tin từ libclamav

Để tạo ra chữ ký có hiệu quả cho clamAV, điều quan trọng nhất là làm thế nào hiểu được công cụ xử lý tập tin đầu vào. Cách tốt nhất để xem nó hoạt dộng được hay không là có một cái nhìn về các thông tin debug libclamav. Chúng ta có thể làm việc này bằng cách gọi clamscan với tùy chọn –debug và –leavee-temps. Tùy chọn đầu tiên làm cho clamscan hiển thị các thông tin từ libclamav và tùy chọn thứ hai giúp tránh việc xóa tập tin tạm thời để có thể phân tích tiếp. Dưới đây là thông tin hiện thị của một libclamav:

*$ clamscan --debug attachment.exe [...]*

*LibClamAV debug: Recognized MS-EXE/DLL file LibClamAV debug: Matched signature for file type PE LibClamAV debug: File type: Executable*

Công cụ ghi nhận một cửa sổ thực thi: *LibClamAV debug: Machine type: 80386 LibClamAV debug: NumberOfSections: 3*

*LibClamAV debug: TimeDateStamp: Fri Jan 10 04:57:55 2003 LibClamAV debug: SizeOfOptionalHeader: e0*

*LibClamAV debug: File format: PE LibClamAV debug: MajorLinkerVersion: 6 LibClamAV debug: MinorLinkerVersion: 0 LibClamAV debug: SizeOfCode: 0x9000*

*LibClamAV debug: SizeOfInitializedData: 0x1000 LibClamAV debug: SizeOfUninitializedData: 0x1e000 LibClamAV debug: AddressOfEntryPoint: 0x27070 LibClamAV debug: BaseOfCode: 0x1f000 LibClamAV debug: SectionAlignment: 0x1000 LibClamAV debug: FileAlignment: 0x200*

*LibClamAV debug: MajorSubsystemVersion: 4 LibClamAV debug: MinorSubsystemVersion: 0 LibClamAV debug: SizeOfImage: 0x29000 LibClamAV debug: SizeOfHeaders: 0x400 LibClamAV debug: NumberOfRvaAndSizes: 16*

*LibClamAV debug: Subsystem: Win32 GUI*

*LibClamAV debug: ------------------------------------LibClamAV debug: Section 0 LibClamAV debug: Section name: UPX0*

*LibClamAV debug: Section data (from headers - in memory) LibClamAV debug: VirtualSize: 0x1e000 0x1e000 LibClamAV debug: VirtualAddress: 0x1000 0x1000 LibClamAV debug: SizeOfRawData: 0x0 0x0*

*LibClamAV debug: PointerToRawData: 0x400 0x400 LibClamAV debug: Section’s memory is executable LibClamAV debug: Section’s memory is writeable*

*LibClamAV debug: ------------------------------------LibClamAV debug: Section 1 LibClamAV debug: Section name: UPX1*

*LibClamAV debug: Section data (from headers - in memory) LibClamAV debug: VirtualSize: 0x9000 0x9000*

*LibClamAV debug: VirtualAddress: 0x1f000 0x1f000 LibClamAV debug: SizeOfRawData: 0x8200 0x8200 LibClamAV debug: PointerToRawData: 0x400 0x400 LibClamAV debug: Section’s memory is executable LibClamAV debug: Section’s memory is writeable*

*LibClamAV debug: ------------------------------------LibClamAV debug: Section 2 LibClamAV debug: Section name: UPX2*

*LibClamAV debug: Section data (from headers - in memory) LibClamAV debug: VirtualSize: 0x1000 0x1000*

*LibClamAV debug: VirtualAddress: 0x28000 0x28000 LibClamAV debug: SizeOfRawData: 0x200 0x1ff LibClamAV debug: PointerToRawData: 0x8600 0x8600 LibClamAV debug: Section’s memory is writeable*

*LibClamAV debug: ------------------------------------LibClamAV debug: EntryPoint offset: 0x8470 (33904)*

Cấu trúc các section hiển thị ở trên cho thấy nó được đóng gói bằng trình

UPX

*LibClamAV debug: ------------------------------------LibClamAV debug: EntryPoint offset: 0x8470 (33904)*

*LibClamAV debug: UPX/FSG/MEW: empty section found - assuming compression*

*LibClamAV debug: UPX: bad magic - scanning for imports LibClamAV debug: UPX: PE structure rebuilt from compressed file LibClamAV debug: UPX: Successfully decompressed with NRV2B LibClamAV debug: UPX/FSG: Decompressed data saved in*

*/tmp/clamav-90d2d25c9dca42bae6fa9a764a4bcede LibClamAV debug: \*\*\*\*\* Scanning decompressed file \*\*\*\*\* LibClamAV debug: Recognized MS-EXE/DLL file LibClamAV debug: Matched signature for file type PE*

Libclamav ghi dữ liệu đóng gói dạng UPX và lưu mã giải nén tại thực thi tại

*/tmp/clamav-90d2d25c9dca42bae6fa9a764a4bcede.* Sau đó nó tiếp tục quét tệp tin mới này:

*LibClamAV debug: File type: Executable LibClamAV debug: Machine type: 80386*

*LibClamAV debug: NumberOfSections: 3*

*LibClamAV debug: TimeDateStamp: Thu Jan 27 11:43:15 2011 LibClamAV debug: SizeOfOptionalHeader: e0*

*LibClamAV debug: File format: PE LibClamAV debug: MajorLinkerVersion: 6 LibClamAV debug: MinorLinkerVersion: 0 LibClamAV debug: SizeOfCode: 0xc000*

*LibClamAV debug: SizeOfInitializedData: 0x19000 LibClamAV debug: SizeOfUninitializedData: 0x0 LibClamAV debug: AddressOfEntryPoint: 0x7b9f LibClamAV debug: BaseOfCode: 0x1000 LibClamAV debug: SectionAlignment: 0x1000 LibClamAV debug: FileAlignment: 0x1000 LibClamAV debug: MajorSubsystemVersion: 4 LibClamAV debug: MinorSubsystemVersion: 0 LibClamAV debug: SizeOfImage: 0x26000 LibClamAV debug: SizeOfHeaders: 0x1000 LibClamAV debug: NumberOfRvaAndSizes: 16 LibClamAV debug: Subsystem: Win32 GUI*

*LibClamAV debug: ------------------------------------LibClamAV debug: Section 0 LibClamAV debug: Section name: .text*

*LibClamAV debug: Section data (from headers - in memory) LibClamAV debug: VirtualSize: 0xc000 0xc000*

*LibClamAV debug: VirtualAddress: 0x1000 0x1000 LibClamAV debug: SizeOfRawData: 0xc000 0xc000 LibClamAV debug: PointerToRawData: 0x1000 0x1000 LibClamAV debug: Section contains executable code LibClamAV debug: Section’s memory is executable*

*LibClamAV debug: ------------------------------------LibClamAV debug: Section 1 LibClamAV debug: Section name: .rdata*

*LibClamAV debug: Section data (from headers - in memory) LibClamAV debug: VirtualSize: 0x2000 0x2000*

*LibClamAV debug: VirtualAddress: 0xd000 0xd000 LibClamAV debug: SizeOfRawData: 0x2000 0x2000 LibClamAV debug: PointerToRawData: 0xd000 0xd000*

*LibClamAV debug: ------------------------------------LibClamAV debug: Section 2 LibClamAV debug: Section name: .data*

*LibClamAV debug: Section data (from headers - in memory) LibClamAV debug: VirtualSize: 0x17000 0x17000 LibClamAV debug: VirtualAddress: 0xf000 0xf000 LibClamAV debug: SizeOfRawData: 0x17000 0x17000 LibClamAV debug: PointerToRawData: 0xf000 0xf000 LibClamAV debug: Section’s memory is writeable*

*LibClamAV debug: ------------------------------------LibClamAV debug: EntryPoint offset: 0x7b9f (31647)*

*LibClamAV debug: Bytecode executing hook id 257 (0 hooks) attachment.exe: OK*

*[...]*

Như vậy không có một tập tin nào được tạo ra từ libclamav. Việc viết một chữ ký cho các tập tin giải nén ta có nhiều co hội mà các công cụ có thể phát hiện ra dữ liệu khi nó được nén bởi các bên đóng gói khác.

Phương pháp này nên áp dụng cho tất cả các tập tin muốn tạo chữ ký. Bằng cách debug cá thông tin ta có thể nhanh chóng xem các công cụ ghi nhận và thực hiện dữ liệu và những tập tin khác được tạo ra.Chữ ký được tạo ra cho các tập tin tạm thời ở cấp dưới thường chung chung hơn và việc phát hiện mã độc tương tự nhau trong với các hình thức khác nhau.

* + 1. Định dạng chữ ký của ClamAV
       1. MD5

Cách dễ nhất để tạo chữ ký cho ClamAV là sử dụng mã checksum MD5, tuy

nhiên phương pháp này chỉ có thể sử dụng cho mã độc tĩnh. Tạo mã MD5: *zolw@localhost:/tmp/test$ sigtool --md5 test.exe > test.hdb zolw@localhost:/tmp/test$ cat test.hdb 48c4533230e1ae1c118c741c0db19dfb:17387:test.exe*

Sử dụng chữ ký:

*zolw@localhost:/tmp/test$ clamscan -d test.hdb test.exe test.exe: test.exe FOUND*

*----------- SCAN SUMMARY -----------Known viruses: 1*

*Scanned directories: 0*

*Engine version: 0.92.1*

*Scanned files: 1*

*Infected files: 1*

*Data scanned: 0.02 MB Time: 0.024 sec (0 m 0 s)*

* + - 1. Tạo MD5 cho một phần PE file

ClamAv hỗ trợ việc tạo chữ ký MD5 cho một phần cụ thể của PE file. Chữ ký này được lưu trữ trong file mdb với định dạng sau:

*PESectionSize:MD5:MalwareName*

Cách dễ dàng nhất để tạo chữ ký MD5 cho một phần PE file là trích xuất phần đó ra một file riêng biệt và sau đó chạy với tùy chọn -mdb với ClamAV.

* + - 1. Chữ ký đựa trên phần thân chương trình

Clam AV lưu trữ tất cả các chữ ký đựa trên phần thân của file trong một định dạng hệ 16 (hexa). Một chữ ký dạng hexa nghĩa là phần thân của file được chuyển đổi sang dạng chuỗi hệ 16 và có thể mở rộng khi sử dụng các ký tự khác nhau.

* + - * 1. Định dạng hexa]

Sử dụng công cụ sigtool –hex-dump để chuyển dữ liệu bất kỳ sang dạng chuỗi hexa

*zolw@localhost:/tmp/test$ sigtool --hex-dump How do I look in hex?*

*486f7720646f2049206c6f6f6b20696e206865783f0a*

* + - * 1. Ký tự đại diện

ClamAv hỗ trợ các phần mở rộng dưới đây cho chữ ký hex:

?? : Phù hợp với bất kỳ byte nào

a?: Phù hợp với một nibble cao (bốn bít cao)

?a: phù hợp với nibble thấp ( bốn bít thấp)

\*: Phù hợp với bất kỳ số lượng byte

{}: Phù hợp với n byte

{-n}: phù hợp với nhỏ hơn hoặc bằng n byte

{n-}: phù hợp với lớn hơn hoặc bằng n byte

{n-m}: phù hợp n-m byte với n>m

(aa||bb||cc||…): phù hợp với aa hoặc bb hoặc cc ...

! (aa||bb||cc||…): phù hợp với bất kỳ byte nào trừ aa, bb,cc,…

HEXSIG[x-y] aa or aa[x-y]HEXSIG : phù hợp với aa gắn vào một chữ ký hex

(B): phù hợp với từ (bao gồm tập tin giới hạn)

(L): phù hợp với CR, CRLF hoặc tập tin giới hạn

Chữ ký phạm vi \* và {} hầu như tách một hex thành hai phần.

* + - * 1. Định dạng chữ ký cơ bản

Ví dụ về định dạng chữ ký :

*MalwareName=HexSignature*

ClamAv sẽ quét tất cả tập tin để tìm chữ ký hex. Tất cả chữ ký kiểu này được lưu dưới dạng file .db.

* + - * 1. Định dạng chữ ký mở rộng

Định dạng chữ ký mở rộng cho phép bổ sung thông tin đặc điểm kỹ thuật như một loại tập tin mục tiêu, offset của virus hoặc phiên bản công cụ, giúp phát hiện tốt hơn. Định dạng như sau:

*MalwareName:TargetType:Offset:HexSignature[:MinFL:[MaxFL]]*

TargetType có giá trị là một trong các số sau đại diện cho kiểu file cụ thế:

0: bất kỳ file nào

1:file thực thi di động đối với cả 32 bit và 64 bit

2: file nằm trong OLE2: file ảnh, VBscript …

3: HTML

4: Tập tin mail

5: đồ họa

6: ELF

7: file text mã ASCII

8: không sử dụng

9: File Mach – O

Offset là một dấu hoặc số thập phân n có thể kết hợp với việc thay đổi đặc

biệt:

\* : bất kỳ

n:offset tuyệt đối

EOF – n: kết thúc file trừ đi n byte

Chữ ký cho PE,ELF, Mach-O file hỗ trợ:

EP + n:entry point cộng n byte

EP – n: entry point trừ n byte

Sx + n: bắt đầu từ section x của dữ liệu cộng n byte

Sx – n: bắt đầu từ section x của dữ liệu trừ n byte

SL + n: kết thúc của section của dữ liệu cộng n byte

SL – n: kết thúc của section của dữ liệu trừ n byte

* + - 1. Chữ ký biểu tượng cho PE file

ClamAV 0.96 có một khoảng biểu tượng mờ để giúp phát hiện các tập tin thực thi mã độc đã ngụy trang bằng cách tìm kiếm các tập tin hình ảnh, office, văn bản, hay tương tự thế.

Biểu tượng phù hợp chỉ được thông qua chữ ký .ldb bằng cách sử dụng thuộc tính đặc biệt thẻ IconGroup1 hoặc IconGroup2. Định danh hai nhóm các biểu tượng được lưu trong tập tin cơ sở dữ liệu .ldb. Định dạng của tập tin .ldb là :

*ICONNAME:GROUP1:GROUP2:ICON\_HASH*

* ICON\_NAME llà một chuỗi định dannh duy nhất cho biểu tượng cụ thể
* Group1 là một chuỗi từ định danh nhóm đầu tiên của các biểu tượng (Icongroup1)
* GROUP2 là một chuỗi định danh nhóm thứ hai của các biểu tượng (IconGroup2)
* ICO\_HASH là một mã băm mò của hình ảnh biểu tượng
  + - 1. Chữ ký cho các siêu dữ liệu thông tin phiên bản trong PE file

Từ phiên bản CLamAV 0.96, nó đã dễ dàng kết hợp một số thông tin xây dựng vào các PE file (gồm file thực thi và thư viện lien kết động). Bất kỳ khi nào tra cứu thuộc tính của một tập tin thực thi PE trong windows, chúng ta sẽ thấy một loạt các chi tiết đó.

Những thông tin được lưu trữ trong một khu vực đặc biệt của nguồn tập tin kèm theo tên của VS\_VERSION\_INFOMATION (hoặc versioninfo). Nó được chia làm hai phần. Phần đầu tiên là một loạt cá số và cờ cho ta thấy phiên bản tập tin. Lúc đầu nó được sử dụng với các trình cài đặt sau khi phân tích nó sẽ có thể xác định xem việc thực thi hoặc thư viện nhất định sẽ được nâng cấp, ghi đè hoặc phát triển lên. Phương pháp này không bao giờ thực sự được sử dụng.

Phần thứ hai là một danh sách đơn giản của khóa, chuỗi giá trị.Nó dành cho thông tin người dùng và hoàn toàn bỏ qua bởi hệ diều hành. Ví dụ khi nhìn vào ping.exe ta sẽ thấy công ty là “Microsoft Corpo-ration”, phần mô tả là “TCP/IP

Ping command”,tên nội bộ là “ping.exe”…. Tùy thuộc vào phiên bản hệ điều hành, một số khóa sẽ có thể hiển thị thông tin đặc biệt trong hộp thoại thuộc tính, tuy tên nội bộ là như nhau.

Để phù hợp với một cặp khóa/giá trị thông tin phiên bản các offset đặc biệt của file gắn VI đã được đưa ra. Điều này tương tự như cá kiểu gắn khác ngoài trừ việc thay đổi để phù hợp với mẫu hex lặp lại của một offset riêng biệt, nó sẽ kiểm tra chính nó và mỗi cặp khóa/giá trị trong tập tin.Thẻ VI không cần và cũng không chấp nhận một +/- offset giống như các ví dụ EP +1. Đối với các chữ ký hex của chính nó, nó chỉ là UTF 16 kết xuất của khóa và giá trị. Chỉ ký tự đại diện ?? và (aa||bb) được cho phép trong chữ ký.Thông thường chúng ta không cần phải bận tâm tìm ra đáp án: bởi vì mỗi cặp khóa/giá trị cung với chữ ký VI dựa trên tương ứng được in bằng clamscan khi dùng tùy chọn –debug.

Ví dụ: *clamscan --debug freecell.exe*

*[...]*

*Recognized MS-EXE/DLL file in cli\_peheader*

*versioninfo\_cb: type: 10, name: 1, lang: 410, rva: 9608 cli\_peheader: parsing version info @ rva 9608 (1/1)*

*VersionInfo (d2de): ’CompanyName’=’Microsoft Corporation’ - VI:43006f006d00700061006e0079004e0061006d006500000000004d006900*

*630072006f0073006f0066007400200043006f00720070006f0072006100740*

*069006f006e000000*

*VersionInfo (d32a): ’FileDescription’=’Entertainment Pack*

*FreeCell Game’ - VI:460069006c006500440065007300630072006900700 0740069006f006e000000000045006e007400650072007400610069006e006d*

*0065006e00740020005000610063006b0020004600720065006500430065006 c006c002000470061006d0065000000*

*VersionInfo (d396): ’FileVersion’=’5.1.2600.0 (xpclient.010817*

*-1148)’ - VI:460069006c006500560065007200730069006f006e00000000 0035002e0031002e0032003600300030002e003000200028007800700063006 c00690065006e0074002e003000310030003800310037002d00310031003400 380029000000*

*VersionInfo (d3fa): ’InternalName’=’freecell’ - VI:49006e007400 650072006e0061006c004e0061006d006500000066007200650065006300650*

*06c006c000000*

*VersionInfo (d4ba): ’OriginalFilename’=’freecell’ - VI:4f007200 6900670069006e0061006c00460069006c0065006e0061006d0065000000660*

*0720065006500630065006c006c000000*

*VersionInfo (d4f6): ’ProductName’=’Sistema operativo Microsoft*

*Windows’ - VI:500072006f0064007500630074004e0061006d00650000000 000530069007300740065006d00610020006f00700065007200610074006900*

*76006f0020004d006900630072006f0073006f0066007400ae0020005700690*

*06e0064006f0077007300ae000000*

*VersionInfo (d562): ’ProductVersion’=’5.1.2600.0’ - VI:50007200 6f006400750063007400560065007200730069006f006e00000035002e00310*

*02e0032003600300030002e0030000000 [...]*

Mặc dù chữ ký VI ở trên được sử dụng trong chữ ký logic nhưng ta vẫn có thể thử dung chúng bình thường trong file .ndb:

*my\_test\_vi\_sig:1:VI:paste\_your\_hex\_sig\_here*

Nếu muốn giải mã một chữ ký VI cơ bản ta dùng :

*echo hex\_string | xxd -r -p | strings –el*

* + - 1. Chữ ký dựa trên siêu dữ liệu

ClamAV 0.96 cho phép tạo chữ ký chung phù hợp với các tập tin được lưu

trữ trong các kiểu chứa khác nhau đáp ứng các điều kiện cụ thể. Định dạng như sau: *VirusName:ContainerType:ContainerSize:FileNameREGEX: FileSizeInContainer:FileSizeReal:IsEncrypted:FilePos: Res1:Res2[:MinFL[:MaxFL]]*

* VirusName: Tên của virus hiển thị khi khớp chữ ký
* ContainerType gồm các định dạng CL\_TYPE\_ZIP, CL\_TYPE\_RAR, CL\_TYPE\_ARJ,CL\_TYPE\_CAB,CL\_TYPE\_7Z, CL\_TYPE\_MAIL,CL\_TYPE\_(POSIX|OLD)\_TAR, CL\_TYPE\_CPIO\_(OLD|ODC|NEWC|CRC) hoặc \*
* ContainerSize: Kích thước nơi chứa tập tin
* FileNameREGEX: Mô tả tên file đích
* FileSizeInContainer: thường được nén kích thước,
* FileSizeReal: thường không nẽ kích thước, MAIL,TAR và CPIO thì

ngược lại

* IsEncrypted: 1nếu file mã hóa, 0, nếu không mã hoá, \* để bỏ qua
* FilePoss: Vị trí file trong nới chứa
* Res1: khi ContainerType là CL\_TYPE\_ZIP or CL\_TYPE\_RAR
* Res2: không sử dụng với ClamAV0.96
* Chữ ký được lưu trữ trong file .cdb.
  + - 1. Chữ ký trên siêu dữ liệu ZIP/RAR

Nó chỉ được sử dụng lưu trữ chữ ký đối với các định dạng Zip hoặc Rar và có

định dạng:

*virname:encrypted:filename:normal size:csize:crc32:cmethod: fileno:max depth*

* Virname: Tên virus
* Encrypted: cờ mã hóa, 1- mã hóa , 0 – không mã hóa
* Filename: Tên file
* Normal size : kích thước file không nén (\* để bỏ qua)
* Compressed size: kích thước file khi nén(\* để bỏ qua)
* CRC32(\* để bỏ qua)
* Cmethod: phương thức nén (\* để bỏ qua)
* Fileno: vị trí file trong kho lưu trữ (\* để bỏ qua)
* Max depthSố lượng tối đa dữ liệu lưu trữ chồng nhau(\* để bỏ qua)

Cơ sở dữ liệu này được xuất dưới dạng file .zmd hoặc .rmd

* + - 1. Danh sách trắng

Danh sách trắng là một tập tin cụ thể sử dụng chữ ký MD5 và đặt trong một tập tin cớ sở dữ liệu có định dạng .fb

Để danh sách một chữ ký cụ thể từ cơ sở dữ liệu ta chỉ cần thêm tên của nó vào một tập tin cục bộ gọi là local.ign2 được lưu trữ trong thư mục cơ sở dữ liệu. Ta cũng có thể thực hiện theo tên chữ ký với MD5 của phần nhập toàn bộ cơ sở dữ liệu cho chữ ký .Ví dụ:

*Eicar-Test-Signature:bc356bae4c42f19a3de16e333ba3569c*

* + - 1. Danh sách chữ ký theo tên

ClamAv sử dụng các tiền tố dưới đây làm tên chữ ký:

* Worm: Sâu mạng
* Trojan: Chương trình cửa hậu
* Adware: adware
* Flooder: flooder
* HTML: tập tin HTML
* Email: Thư điện tử
* IRC: trojan IRC
* JS: Mã độc Java Script
* PHP: Mã độc PHP
* ASP: Mã độc ASP
* VBS: Mã độc VBS
* BAT: Mã độc BAT
* W97M,W2000M:virus macro trên word
* X97M, X2000M: virus macro trên excel
* O97M,O200M: virus macro chung trên office
* DoS: phân mềm tấn công từ chối dịch vụ
* DOS: mã độc DOS trước đây
* Exploit: lỗ hổng bảo mật phổ biến
* VirTool: Công cụ tạo virus
* Dialer: dialer
* Joke: lừa đảo

Quy tắc đặc tên của ClamAV:

* Luôn sử dụng một hậu tố zippwd vào tên mã độc hại cho loại ZMD
* Luôn sử dụng một hậu tố rarpwd vào tên mã độc hại cho loại RMD
* Chỉ sự dụng các ký tự chữ và số, dấu gạch ngang “-“, dấu chấm “.”, dấu gạch dưới “\_” trong tên mã độc không bao giờ sử dụng dấu đơn, kép, khoảng trống.
  + - 1. File đặc biệt
         1. HTML

ClamAV có chứa một mã HTML đặc biệt binhg thường, giúp phát hiện lỗ hổng HTML. Chạy sigtool –html chuẩn hóa trên một tập tin HTML sẽ tạo ra các tập tin dưới đây:

Nocmomment.html: tập tin bình thường, với trường hợp thấp hơn tất cả các nhận xét và khoảng trắng bị xóa

Notags.html: giống như trên nhưng các thẻ HTML bị xóa

Cần tạo ra một chữ ký đối với các tập tin được tạo ra. Để loại bỏ khả năng

cảnh báo giả các mục tiếu cần được đặt giá trị là 3.

* + - * 1. Tập tin văn bản

Cũng giống như HTML các tập tin văn bản mã ASCII được bình thường hóa tất cả các khoảng trống và kiểm soát các ký tự bị xóa trước khi quét

Câu lệnh : clamscan – leave-temp để có một tập tin bình thường và sau đó tạo ra chữ ký với đích có kiểu là 7

* + - * 1. Tập tin thực thi di động đã nén

Nếu tập tin được nén bởi UPX, FSG, Petite hoặc một trình đống gói PE file khác được hỗ trợ bởi libclamav, cần chạy clamscan với tùy chọn --debug --leave- temps. Ví dụ xuất ra từ trình nén file FSG:

*LibClamAV debug: UPX/FSG/MEW: empty section found - assuming compression LibClamAV debug: FSG: found old EP @119e0*

*LibClamAV debug: FSG: Unpacked and rebuilt executable saved in*

*/tmp/clamav-f592b20f9329ac1c91f0e12137bcce6c*

Tiếp theo tạo một chữ ký từ :

*/tmp/clamav-f592b20f9329ac1c91f0e12137bcce6c*

# CHƯƠNG 3. XÂY DỰNG CƠ SỞ DỮ LIỆU MÃ ĐỘC

Chương cuối của đồ án sẽ xây dựng hai chương trình để nhận dạng và quản lý cơ sở dữ liệu mã độc được trình bày trong chương này. Đầu tiên là chương trình nhận dạng theo chuỗi sau đó sẽ là chương trình quản lý cơ sở dữ liệu theo chuẩn và sử dựng cơ sở dữ liệu đó để quét.

## Xây dựng chương trình quản lý cơ sở dữ liệu mẫu mã độc theo chuẩn

Xây dựng một phần mềm quản lý cơ sở dữ liệu mã độc có cơ sở dữ liệu được lưu dưới dạng chuẩn. Việc lưu này giúp loại bỏ công việc chuyển từ cơ sở dữ liệu bình thường sang dạng chuẩn.

Chương trình được viết bằng ngôn ngữ C# dựa trên nền tảng .Net với Net

Framework 4.0 và được biên dịch bằng Visual studio 2010.

Đầu tên là xây dựng cơ sở dữ liệu theo chuẩn

File cơ sở dữ liệu được lưu dưới định dạng XML có tên là “malware.xml” và được đặt ở cùng thư mục của chương trình quản lý



*Hình 3.1 Cấu trúc file dữ liệu*

Cơ sở dữ liệu gồm các trường chính như sau :

* + - ID : Số thực tự cũng như số hiệu của mẫu mã độc
    - MD5 : Mã băm MD5 trích xuất từ file mẫu
    - SHA1: mã SHA1 của mẫu
    - SHA1: mã SHA256 của mẫu
    - SHA1: mã SHA512 của mẫu
    - Filename : Tên file mẫu
    - Size: Kích thước của file mẫu
    - MINEType : Kiểu định dạng file mẫu : exe, rar, mp3 …
    - Malwarename: Tên mã độc được nhận dạng của file mẫu đó
    - Sign: Chuỗi nhận dạng của file mẫu đó (nếu có)

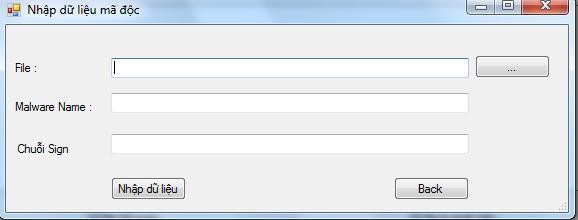
Cơ sở dữ liệu đi kèm với chương trình có một số mẫu đã được cập nhật bởi các hãng Antivirus trên thế giới và Việt Nam:

* + - * Một mẫu là mã độc mang tên “*FunnyIM”* do BKAV cập nhật đã khá lâu có định dạng file .dat, đây là một mã độc dạng worm
      * Mã độc có tên là “*nettui.dll*” là một thư viện được sử dụng bởi mã độc sử dụng lỗ hổng bảo mật office: CVE-2010-3333 của Microsoft có tên là “*Trojan.Win32.Genome.akudf*” do Kaspersky cập nhật

Chức năng của chương trình:

* + - Nhập cơ sở dữ liệu:

Chức năng này khi hoạt động sẽ lần lượt nhập ID của mẫu vào cơ sở dữ liệu đồng thời tiến hành lấy mã băm MD5, SHA1, SHA 256, SHA 512, tên mẫu, kích thước, kiểu file, tên mã độc và chuỗi nhận dạng (nếu có). Trong đó tên file được lấy từ khung nhập và tên mã độc do người nhập dữ liệu đưa vào. Tương ứng với chứ năng này là module nhập dữ liệu

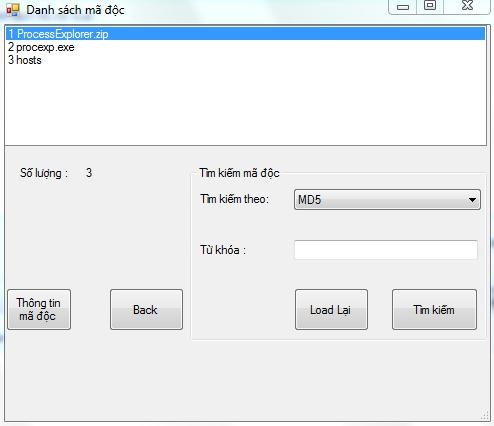


*Hình 3.2 Nhập cơ sở dữ liệu*

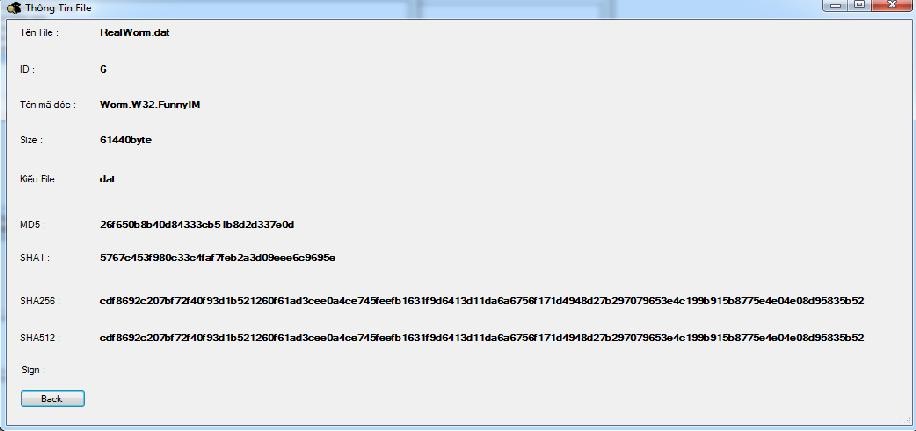
* + - Danh sách cơ sở dữ liệu:

Chức năng này hiển thị tên các mẫu được lưu trong cơ sở dữ liệu dưới dạng tên của file mà người nhập cơ sở dữ liệu đưa vào. Nó cho phép tìm kiếm theo các trường và hiển thị chi tiết tựng mẫu mã độc được lưu trong cơ sở dữ liệu.

Module của chức năng như hình dưới đây :

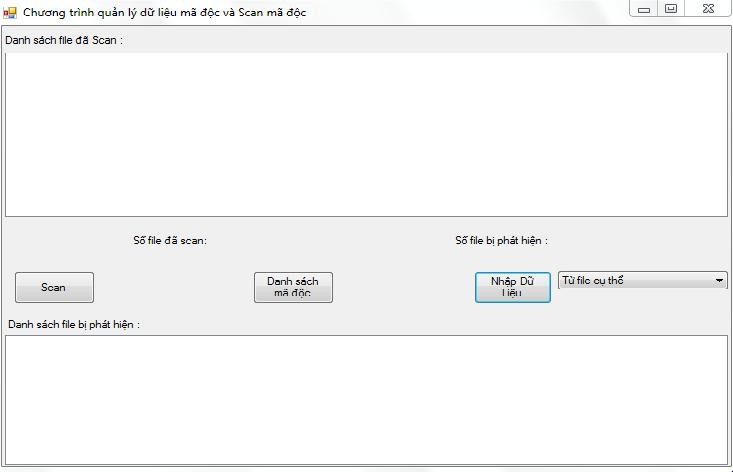


*Hình 3.3 Hiển thị*



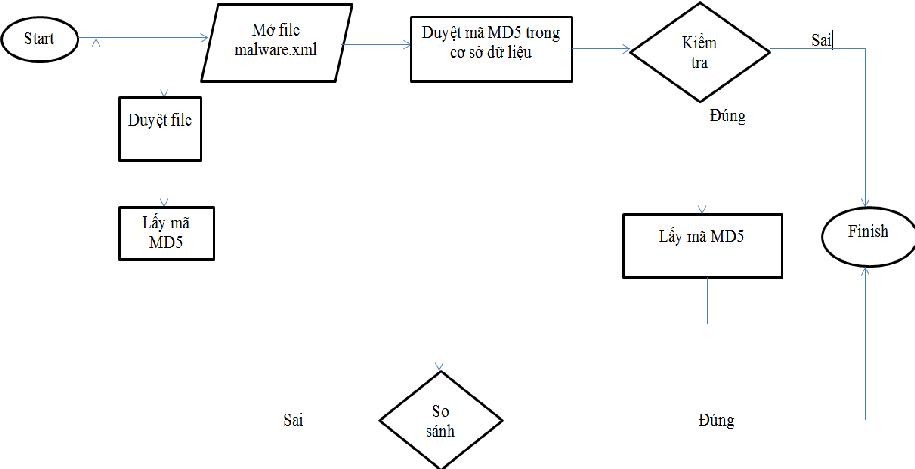
*Hình 3.4 Chi tiết mẫu mã độc*

Chức năng thứ 3 của chương trình là sử dụng cơ sở dữ liệu trên để nhận dạng mã độc trong máy. Chức năng này chỉ cho phép quét các file trong từng thư mục chưa có khả năng quét tất cả các file trên máy tính người dung. Giao diện của chức năng này như sau:



*Hình 3.5 Giao diện Scan*

Mô hình quét của chương trình:



*Hình 3.6 Mô hình chức năng quét*

Thuật toán : Khi nhấn nút scan thì chức năng bắt đầu hoạt động

Bước 1: Mở file malware.xml sau đó trỏ tới trường MD5 duyệt từng trường MD5. Nếu không tồn tại trường MD5 hoặc trống thì sẽ kết thúc chương trình

Bước 2: Duyệt file trong thư mục sau đó lấy mã MD 5 của từng file

Bước 3: So sánh hai mã MD5 lấy từ cơ sở dữ liệu và của file được duyệt

* + - Nếu bằng nhau thì thong báo phát hiện , kết thúc chương trình
    - Nếu không bằng nhau thì lặp lại bước 2
    - Sau hi duyệt hết file trong thư mục thì quay lại bước 1 lặp đến khi kết

thúc file cơ sở dữ liệu và kết thúc chương trình

## Xây dựng chương trình nhận dạng mã độc theo chuỗi

Tệp tin cơ sở dữ liệu là “mn.kkt”, được để cùng với thư mục chứa chương

trình quét.Cấu trúc tệp tin cơ sở dữ liệu gồm hai phần :

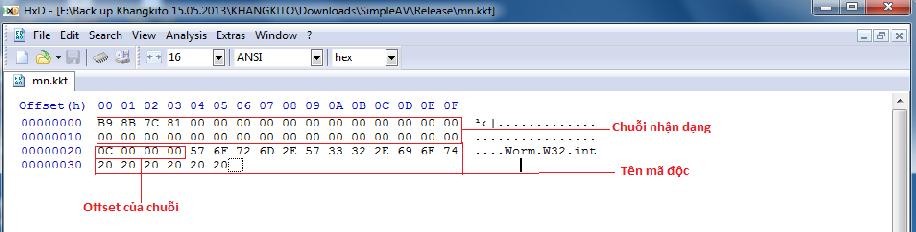
* + - Chuỗi nhận dạng gồm 32 ký tự
    - Vị trí của chuỗi trong mẫu mã độc
    - Tên mã độc gồm 18 ký tự

*struct m\_Sign*

*{*

*char Sign[32]; unsigned int lPos; char Name[18];*

*};*

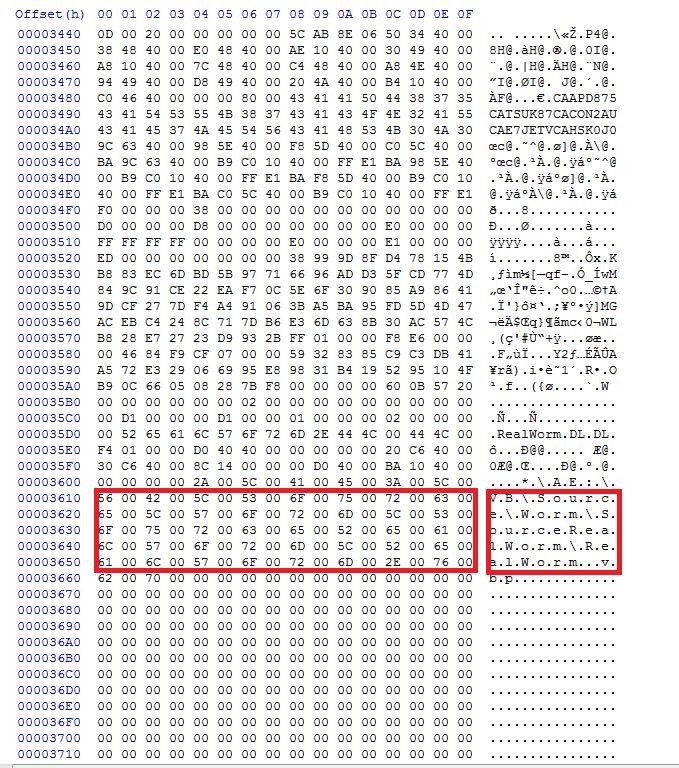


*Hình 3.7 Tệp dữ liệu*

Cách xác định chuỗi nhận dạng của một mẫu cụ thể là mẫu có tên FunnyIM

do BKAV trước đây có cập nhật :

* + - Sử dụng tool HxD.exe để xác định chuỗi như hình dưới đây :



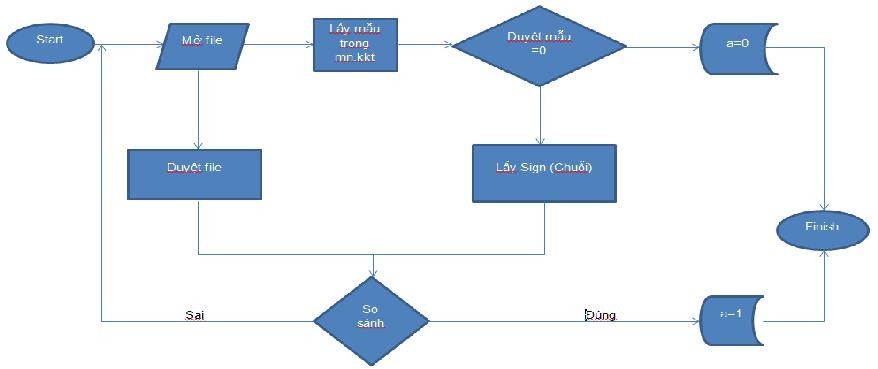
*Hình 3.8 Chuỗi String*

Đây là chuỗi đặc trưng của loại mã độc này mà không lẫn với các mã

độc hay tệp tin nào khác.

* + - Sau khi xác định được chuỗi nhận dạng thì ta sử dugnj chức năng nhập dữ liệu mã độc vào cơ sở dữ liệu bằng module được trình bày ở dưới đây.

Mô hình tổng quan chương trình

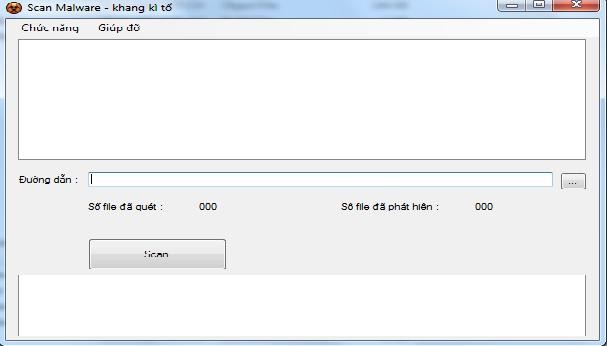


*Hình 3.3Mô hình chương trình quét*

Chương trình được viết bằng ngôn ngữ C++ và biên dịch bằng Visual Studio 2010. Nó chạy trên nền tảng Net Framework 4.0 trở lên

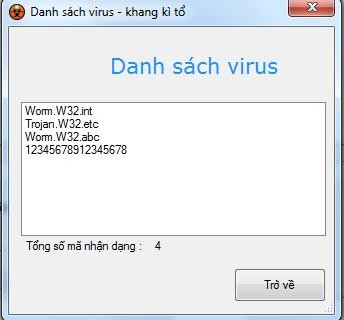
Chương trình gồm ba Module:

* + - Module 1: Giao diện chính – quét mã độc



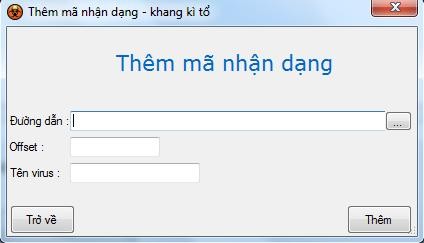
*Hình 3.9 Giao diện chính*

* + - Module 2: Danh sách mã độc có trong tệp dữ liệu



*Hình 3.10 Danh sách mã độc*

* + - Module 3: Thêm mẫu vào tệp dữ liệu



*Hình 3.11 Thêm mẫu*

Hoạt động: Chi tiết hoạt động của chương trình em sẽ trình bày khi tiến hành bảo vệ đồ án.

Code chương trình được trình bày ở phần “MỤC LỤC”

# KẾT LUẬN

Sau một thời gian tập trung nghiên cứu, do còn hạn chế về kiến thức nên đồ án còn nhiều hạn chế.Kết quả đạt được là đồ án này đã thực hiện được một số nội dung sau:

Về phần lý thuyết:

* + - Tìm hiểu tổng quan về mã độc hại, tình hình phát triển mã độc hại trên thế giới những năm gần đây.
    - Tìm hiểu được các cách thức phát hiện mã độc cơ bản
    - Tìm hiểu về cách thức thu thập mẫu mã độc mới.
    - Tìm hiểu và xây dựng cơ sở dữ liệu mã độc theo chuẩn chung Về phần thực nghiệm:
    - Xây dựng một chương trình quét mã độc theo chuỗi nhận dạng
    - Xây dựng chương trình quản lý cơ sở dữ liệu mã độc.

Hạn chế:

Chưa xây dựng được cơ sở dữ liệu một cách đầy đủ các thành phần, cơ sở dữ liệu xây dựng trên yếu tổ chủ quan, cơ chế sử dụng cơ sở dữ liệu chưa hoàn thiện, thực hiện thuật toán chưa tốt

Hướng phát triển

Áp dụng thêm các phương pháp phát hiện mã độc khác để hoàn thành chương trình quét mã độc dần tiến tới thử nghiệm và áp dụng trong thực tế đi từ quy mô nhỏ tới lớn. Xây dựng một cơ sở dữ liệu mã độc đầy đủ hơn, truy xuất nhanh hơn và có tính áp dụng thực tế cao. Xây dựng chương trình nhận dạng mã độc sử dụng được không chỉ cơ sở dữ liệu có sẵn mà sử dụng được cả những cơ sở dữ liệu của các hãng, tổ chức cá nhân khác .

# TÀI LIỆU THAM KHẢO

[1]. Addison and Wesley – *The Art of Computer Virus Research and Defense* –

Feb.2005

[2]. Michael Sikorski and Andrew Honig – *Practical Malware analysis* – 2012 [3]. Chris Eagle – *The IDA Pro Book* - 2011

[4]. https://[www.hex-rays.com](http://www.hex-rays.com/)

[5]. [www.ollydbg.de](http://www.ollydbg.de/)

[6]. [www.threatexpert.com/](http://www.threatexpert.com/)

[7]. [www.mcafee.com](http://www.mcafee.com/)

[8]. [www.codeproject.com/](http://www.codeproject.com/)

[9]. [http://www.securelist.com](http://www.securelist.com/)

# PHỤ LỤC

## Code chương trình quét theo chuỗi nhận dạng

* + - Code phần quét mã độc

private: System::Boolean compareStr(char \*str1, char \*str2)

{

*//Hàm dùng để so sánh 2 chuỗi ký tự có giống nhau không*

*//Hàm trả về TRUE khi giống nhau và FALSE khi khác nhau*

int i=0;

while (\*(str1+i) == \*(str2+i))

{ i++;

if (i == 32) return true;

}

return false;

}

private: System::Boolean compareStr1(char \*str1, char \*str2, int m, int n)

{

*//Hàm dùng để so sánh 2 chuỗi ký tự có giống nhau không*

*//Hàm trả về TRUE khi giống nhau và FALSE khi khác nhau*

int i=0; int j;

for (j=0;j<=m;j++)

{

while (\*(str1+i) == \*(str2+i))

{ i++;

if (i == n ) return true;

}

return false;

}

}

private: System::Void scan(System::String ^strPath)

{

if (isScan)

{

System::String ^strFile;

System::String ^tmp; char strDatFile[32];

*//Liệt kê file trong thư mục hiện tại*

for each (strFile in System::IO::Directory::GetFiles(strPath))

{

//Chuyen kieu String sang kieu char int lenght = strFile->Length;

char \*chTmpData = new char[lenght+1]; for(int i = 0; i<lenght;i++)

{

\*(chTmpData + i) = strFile[i];

};

\*(chTmpData + lenght) = '\0';

*//Mở file cần được scan ra làm việc* FILE \*fs; fopen\_s(&fs,chTmpData,"rb");

if (fs != NULL)

{

FILE \*f;

char strPathSign[]="mn.dat"; fopen\_s(&f,strPathSign,"rb"); m\_Sign sign;

int i=0;

int posEnd;

*//Xác định kích thước file* fseek(f,0,SEEK\_END); posEnd=ftell(f);

*//Trả về vị trí ban đầu* fseek(f,0,SEEK\_SET); fseek(fs,0,SEEK\_END); int posend=ftell(fs);

*//Trả về vị trí ban đầu*

fseek(fs,0,SEEK\_SET);

*//Tiến hành vòng lặp liệt kê mã nhận dạng*

int j=0;

while (ftell(f) < posEnd)

{

*//Đọc mẫu tin lưu mã nhận dạng virus*

fread(&sign,sizeof(m\_Sign),1,f);

//*Duyet file can quet*

while(ftell(fs)<posend)

{

fseek(fs,j++,SEEK\_SET);

int cur1;

cur1=ftell(fs)

fread(strDatFile,32,1,fs);

if (compareStr(strDatFile,sign.Sign) == true)

{

//*Nhận dạng đúng là virus*

tmp = gcnew System::String(sign.Name);

lstRe->Items->Add(L"Phát hiện : " + tmp + L" đường dẫn : " + strFile); detected++;

lblDetected->Text = Convert::ToString(detected); System::Windows::Forms::Application::DoEvents();

}

}

//

int cur2;

cur2=ftell(f)

fread(strDatFile,32,1,fs);

if (compareStr(strDatFile,sign.Sign) == true)

{

System::Windows::Forms::Application::DoEvents();

}

}

fclose(f); fclose(fs);

}

*//Ghi ra listBox thông báo đã quét file này* tmp = gcnew System::String(strFile); lstFile->Items->Add(tmp);

scaned++;

lblScaned->Text = Convert::ToString(scaned); System::Windows::Forms::Application::DoEvents();

}

*//Tiến hành liệt kê các tập tin trong thư mục*

for each (strFile in System::IO::Directory::GetDirectories(strPath))

{

try

{

*//Lặp đệ qui*

scan(strFile);

}

catch(Exception ^e)

{

*//Bắt lỗi*

}

}

}

}

* + - Code nhập dữ liệu

System::String ^strFile; System::String ^tmp; FILE \*f;

*//Mở file sẽ được lấy mẫu*

FILE \*fs;

char strPathSign[]="mn.kkt"; char strDatFile[32];

m\_Sign sign;

System::Boolean isHave=false; int iTmpOffset;

int i;

strFile = txtPath->Text;

*//Convert từ chuỗi trong TextBox qua chuỗi dạng chuẩn của chương trình*

//Từ đưa về chuỗi với 18 ký tự

*//Chuyển từ dạng String -> Char* char tmpName[18]; System::String ^strTmpName; strTmpName=txtName->Text;

for (i=0; i < strTmpName->Length;i++)

{

tmpName[i] = strTmpName[i];

}

for (int i=strTmpName->Length; i <= 18;i++)

{

tmpName[i] = 32;

}

*//Do trong strFile là kiểu string*

*//cần phải chuyển đổi ra thành kiểu Char để đưa vào hàm*

int itmp = strFile->Length;

char \*chTmpData = new char[itmp+1]; for(int i = 0; i<itmp;i++)

{

\*(chTmpData + i) = strFile[i];

};

\*(chTmpData + itmp) = '\0'; fopen\_s(&fs,chTmpData,"rb"); fopen\_s(&f,strPathSign,"a+b");

int posEnd;

*//Xác định kích thước file* fseek(f,0,SEEK\_END); posEnd=ftell(f);

*//Trả về vị trí ban đầu*

fseek(f,0,SEEK\_SET);

*//Tiến hành vòng lặp liệt kê mã nhận dạng*

while (ftell(f) < posEnd)

{

*//Đọc mẫu tin lưu mã nhận dạng virus*

fread(&sign,sizeof(m\_Sign),1,f);

*//Nhảy trong file muốn quét tới offset muốn xét*

*//Offset này lấy từ sign.lPos*

fseek(fs,sign.lPos,SEEK\_SET);

*//Lấy 32 byte tại offset đó*

fread(strDatFile,32,1,fs);

if (compareStr(strDatFile,sign.Sign,32))

{

*//Nhận dạng trùng mã nhận dạng*

tmp = gcnew System::String(sign.Name); isHave=true;

System::Windows::Forms::MessageBox::Show(L"Mã nhận dạng này đã có trong bảng mã",L"Trùng mã nhận dạng");

break;

}

if (compareStr(tmpName,sign.Name,18))

{

*//Nhận dạng trùng tên virus đã lưu*

tmp = gcnew System::String(sign.Name); isHave=true;

System::Windows::Forms::MessageBox::Show(L"Tên virus này đã có trong bảng mã nhận dạng",L"Trùng mã nhận dạng");

break;

}

}

if (!isHave)

{

*//Sau khi kiểm tra là chưa có mã nhận dạng này + Tên virus là thao tác ghi mẫu tin dữ liệu virus mới vào*

*//Lấy dữ liệu tại Offset cần lấy (Tại txtOffset)* iTmpOffset=System::Int32::Parse(txtOffset->Text); fseek(fs,iTmpOffset,SEEK\_SET); fread(strDatFile,32,1,fs);

*//Đưa dữ liệu vào một biến theo Struct m\_Sign*

*//Hàm strcpy xử lý lỗi trong những trường hợp này (Do có ký tự NULL)*

*//Vì vậy hàm copy 2 mảng char sẽ cần tự code lại*

for (i=0;i < 32;i++) sign.Sign[i]=strDatFile[i]; sign.lPos=iTmpOffset; strcpy(sign.Name,tmpName);

*//Nhảy đến cuối file*

fseek(f,0,SEEK\_END);

*//Ghi dữ liệu vào cuối file*

fwrite(&sign,sizeof(sign),1,f);

System::Windows::Forms::MessageBox::Show(L"Đã lưu xong mẫu tin dữ liệu virus mới",L"Thành công");

}

fclose(f);

fclose(fs);

* + - Liệt kê mẫu trong tệp dữ liệu

FILE \*f;

char strPathSign[]="mn.kkt"; fopen\_s(&f,strPathSign,"rb"); m\_Sign sign;

char chuoi[18]=""; System::String ^tmp=""; int i=0;

int posEnd;

*//Xác định kích thước file* fseek(f,0,SEEK\_END); posEnd=ftell(f);

*//Trả về vị trí ban đầu*

fseek(f,0,SEEK\_SET);

*//Tiến hành vòng lặp liệt kê mã nhận dạng*

while (ftell(f) < posEnd)

{ i++;

fread(&sign,sizeof(m\_Sign),1,f); strncpy(chuoi,sign.Name,17);

tmp = gcnew System::String(chuoi); lstVirus->Items->Add(tmp);

};

fclose(f);

lblCount->Text=Convert::ToString(i);