

M C L C

M C L C	3
DANH M C T VI T T T	5
DANH M C HÌNH V	6
L I NÓI U	8
CH NG 1. PH NG PHÁP THU TH P VÀ PHÂN TÍCH MÃ C	10
1.1 Gi i thi u v mã c h i	10
1.1.1 Khái ni m mã c h i	10
1.1.2 L ch s mã c h i.....	10
1.2 C ch ho t ng c a mã c	11
1.2.1 C ch ho t ng c a Virus	12
1.2.2 C ch ho t ng c a Worm.....	12
1.2.3 C ch ho t ng c a Trojan Horse	12
1.3 Ph ng pháp thu th p mã c	12
1.3.1 Các ph ng pháp thu th p m u	12
1.3.2 Các công c thu th p m u mã c	16
1.4 Quy trình phân tích mã c h i	30
1.4.1 Các ph ng pháp phân tích mã c.....	30
1.4.2 Các b c c b n phân tích mã c h i	31
1.4.3 Phân tích một m u mã c c th	34

CHƯƠNG 2. NGHIÊN CỨU MÔ TẢ CẤU TRÚC LƯỚI MÃ C...	36
2.1 Các kiểu dữ liệu mã nguồn và kỹ thuật phát hiện lỗi...	36
2.1.1 String – Chuỗi.....	36
2.1.2 Mã byte.....	37
2.1.3 Khung mã nguồn có sẵn.....	38
2.1.4 Phương pháp dựa trên hành vi.....	38
2.1.5 Kỹ thuật tĩnh.....	39
2.1.6 Phát hiện biến cục bộ mã nguồn.....	40
2.1.7 Mã giả lập.....	40
2.2 Nghiên cứu chuyên tra cứu dữ liệu mã nguồn.....	41
2.3 Nghiên cứu cấu trúc dữ liệu mã nguồn ClamAV để xây dựng cấu trúc dữ liệu mã nguồn.....	44
2.3.1 Clam Anti Virus.....	44
2.3.2 ClamAV Virus Databases.....	44
2.3.3 Debug thông tin từ libclamav.....	45
2.3.4 Phân tích cấu trúc của ClamAV.....	51
CHƯƠNG 3. XÂY DỰNG CẤU TRÚC LƯỚI MÃ C.....	61
3.1 Xây dựng chương trình quản lý cấu trúc dữ liệu mã nguồn theo chuỗi.....	61
3.2 Xây dựng chương trình nhận dạng mã nguồn theo chuỗi.....	65
KẾT LUẬN.....	69
TÀI LIỆU THAM KHẢO.....	70
PHỤ LỤC.....	71

DANH M C T VI T T T

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 M C HÌNH V

Hình 1.1 Giao di n chính virustotal	13
Hình 1.2 Giao di n virustotal sau khi quét	13
Hình 1.3 Giao di n tr c khi quét c a ThreatExpert.....	14
Hình 1.4 Giao di n sau khi quét c a ThreatExpert	14
Hình 1.5 H th ng ThreatExpert	15
Hình 1.6 Ng i dung g i m u t i	15
Hình 1.7 ThreatExpert tr k t qu v cho ng i dùng.....	16
Hình 1.8 Giao di n 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 di n Olly Debug.....	25
Hình 1.21 Tùy ch n 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 Nhúng dữ liệu	62
Hình 3.3 Hình ảnh	63
Hình 3.4 Chi tiết mã nguồn	63
Hình 3.5 Giao diện Scan	64
Hình 3.6 Mô hình chương trình quét.....	64
Hình 3.7 Tệp dữ liệu	65
Hình 3.8 Chuỗi String.....	66
Hình 3.3 Mô hình chương trình quét	67
Hình 3.9 Giao diện chính	67
Hình 3.10 Danh sách mã nguồn	68
Hình 3.11 Thêm mã	68

L I NÓI U

Khi nhu cầu và vị trí sử dụng Internet của con người ngày càng tăng thì cũng là lúc nhu cầu mua sắm hàng hóa trực tuyến càng nhiều, nên bắt đầu có các mã độc hại. Mã độc hại xuất hiện bắt đầu trên môi trường của các thiết bị như các máy tính, USB, máy tính xách tay, môi trường Internet trong các website, trong các tin nhắn, trong hình ảnh, âm thanh, video, trong các phần mềm miễn phí.... Khi mã độc hại xâm nhập vào một máy tính nào đó thì nó lây lan sang các máy tính khác là khá nhanh và thiết bị do mã độc hại gây ra là khó có thể ngăn ngừa được.

Hình thức chống lại các loại mã độc hại mà chúng ta thường sử dụng các chương trình Antivirus. Tuy nhiên có một chương trình Antivirus mà cách có hiệu quả hơn có một thuật toán quét sao cho nhanh nhất và quan trọng là miễn phí sử dụng và update thường xuyên hàng ngày. Ngoài ra các sản phẩm có phí miễn phí hoặc bán, nhưng và phần mềm pháp luật mà cách khoa học cho phép các thuật toán kiểm tra mã độc hại mà cách hiệu quả nhất. Nhưng thì nó phải có các chức năng các tiêu chuẩn có thể trao đổi với các sản phẩm của các hãng khác. Vì xây dựng các sản phẩm như vậy có thể sản phẩm phát triển các phần mềm phát hiện và ngăn chặn mã độc hại hiệu quả. Ngoài ra nó cũng có thể sử dụng như một công cụ hỗ trợ cho nhà nghiên cứu làm công việc tác nghiệp phân tích mã độc.

Chính vì những lý do trên, nên mục đích của đề án này là nhằm nghiên cứu nhằm sáng tạo các kỹ thuật phát hiện ra các loại mã độc, các chủng loại mã độc và cấu trúc chung của các sản phẩm mã độc của các hãng phần mềm lớn trên thị trường như ClamAV, Kaspersky, BKAV... những thiết bị thiết kế, xây dựng một sản phẩm mã độc và một chương trình demo quản lý và sử dụng các sản phẩm.

Đề án 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, cách hoạt động của mã độc và quy trình thu thập, phân tích mã độc.

Chương 2. Nghiên cứu về mã độc sử dụng mã độc

Trình bày về mã độc phần mềm phát hiện mã độc cơ bản mà các chương trình anti virus hay dùng tìm và diệt mã độc. Tìm hiểu nghiên cứu chủng loại mã độc, các sản phẩm mã độc của chương trình clamav.

Chương 3. Xây dựng cơ sở dữ liệu mã c

- Xây dựng cơ sở dữ liệu theo chu trình n d ng, theo chu n ã nêu trên
- Xây dựng m t ch ng trình quét mã c s d ng chu i nh n d ng
- Xây dựng ch ng trình qu n lý c s d li u theo chu n.

K t lu n

Trình bày v k t qu t c c a quá trình làm án, nh ng m t h n ch và h ng phát tri n.

Vì i u ki n th i gian làm án có h n c ng nh hi u bi t còn h n ch nên ch c ch n tài không tránh kh i nh ng sai sót. R t mong các th y cô góp ý ki n tài khoa h c c a em c hoàn thi n h n.

Em xin chân thành c m n các th y, cô giáo H c Vi n K Thu t M t Mã ã t n gi ng d y và giúp em có nh ng ki n th c c n thi t hoàn thành án ã nh n. Em c m n th y L ng Th D ng - phó khoa An toàn thông tin, th y Hoàng Thanh Nam gi ng viên khoa An toàn thông tin ã giúp , h ng d n và ch b o em hoàn thành án.

Em xin chân thành c m n!

Hà N i, ngày 11 tháng 06 n m 2013

Sinh viên th c hi n

Trần Văn Khang

CHƯƠNG 1. PHƯƠNG PHÁP THU THẬP VÀ PHÂN TÍCH MÃ C

Có thể xây dựng một cơ sở dữ liệu mã c t yêu cầu thì cần phải trải qua nhiều quá trình khác nhau, trong đó quá trình thu thập và phân tích mã c là quá trình rất quan trọng. Nó giúp cho việc xây dựng cơ sở dữ liệu mã c một cách hiệu quả và nhanh chóng. Cùng với quá trình này đem lại cho ta cái nhìn rõ nét nhất về các hành vi c ng nh là c i m mà mã c ó gây hại cho ng i dung. Vì vậy chúng ta sẽ đi sâu vào quá trình thu thập và phân tích mã c.

1.1 Giới thiệu về mã c hại

1.1.1 Khái niệm mã c hại

Malware (Malicious software) hay còn gọi là mã c hại (Malicious code) là tên gọi chung cho các phần mềm c thi t k , lập trình c bit phá hoại hệ thống cá nhân hoặc làm gián đoạn môi trường hoạt động m ng. Malware thâm nhập vào một hệ thống máy tính mà không có sự đồng ý của chủ sở hữu.

1.1.2 Lịch sử mã c hại

Internet phát triển, nó thể hiện vì c k t n i các máy tính, máy chủ , laptop, mobile phone trên khắp thế giới, thêm vào là sự phát triển của các website vì vậy mã c hại c ng theo đó mà phát triển r ng kh p v i s l ng l n và ngày càng phức tạp hơn.

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

Năm 1949 John von Neuman (1903-1957) phát triển nền tảng lý thuyết 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 i h c M , đã đưa ra định nghĩa đầu tiên về virus: “Là một chương trình máy tính có thể tác động lên hệ thống máy tính khác bằng cách sao chép chúng bằng phương pháp nào đó vào một bộ nhớ sao chép”. Fred Cohen luôn là cái tên c nh c n khi nói về lịch sử virus.

Năm 1986 hai anh em lập trình viên ng i Pakistan là Basit và Amjad thay thế mã thực thi (executable code) trong rãnh ghi khi ng c a m t a m m bằng mã riêng của họ , c thi t k v i m c ích phát tán t m t a m m 360K khi cho

vào bất cứ ai nào. Loại mã mang virus này có mác “© Brain”. Đây chính là nh ng virus MS-DOS xu t hi n s m nh t.

N m 1987 Lehigh, m t trong nh ng virus file u tiên xâm nh p các t p l nh command.com (virus này sau ó ti n hoá thành virus Jerusalem). M t virus khác có tên IBM Christmas, v i t c phát tán c c nhanh (500.000 b n sao/ti ng), là c n ác m ng i v i các máy tính l n (mainframe) c a Big Blue trong su t n m ó. ng h c a máy tính (gi ng bom n ch m cài hàng lo t cho cùng m t th i i m).

Tháng 11 cùng n m, Robert Morris ch ra worm chi m c các máy tính c a ARPANET làm li t kho ng 6.000 máy.

N m 1991 virus a hình (*polymorphic virus*) ra i u tiên là Tequilla. Lo i này bi t t thay i hình th c c a nó, gây ra s khó kh n cho các ch ng trình ch ng virus.

N m 1994 Trò l a qua e-mail u tiên xu t hi n trong c ng ng tin h c. Trò này c nh báo ng i s d ng v m t lo i virus có th xoá toàn b c ng ngay khi m e-mail có dòng ch “Good Times”. M c dù không gây thi t h i gì mà ch có tính ch t do d m, trò l a này v n ti p t c xu t hi n trong chu k t 6 n 12 tháng/l n.

N m 1995 *macro virus* u tiên xu t hi n trong các mã macro trong các t p c a Word và lan truy n qua r t nhi u máy. Lo i virus này có th làm h h i u hành ch .

N m 1999 Bubble Boy sâu máy tính u tiên không d a vào vi c ng i nh n email có m file ính kèm hay không. Ch c n th c m ra, nó v n s t ho t ng.

N m 2003 Slammer m t lo i worm lan truy n v i v n t c k l c, truy n cho kho ng 75 ngàn máy trong 10 phút.

N m 2004 ánh d u m t th h m i c a mã c h i là worm Sasser. V i lo i worm này thì ng i ta không c n ph i m ính kèm c a i n th mà ch c n m lá th là cho nó xâm nh p vào máy. Sasser không hoàn toàn h y ho i máy mà ch làm cho máy ch tr nên ch m h n và ôi khi nó làm máy t kh i ng tr l i. Vi t Nam mã c h i c ng gây ra nh ng thi t h i áng k .

1.2 C ch ho t ng c a mã c

M i lo i mã c có các c ch ho t ng khác nhau tuy nhiên chúng u có cùng m t m c ích là phá ho i gây h i cho máy tính ng i dùng. Đây em s nêu ra c ch ho t ng c a các lo i mã c c b n là Virus, Trojan Horse, worm

1.2.1 Cách 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 dùng. Thông thường virus lây lan vì mục đích là xóa, sửa file trên máy tính. Loại virus nguy hiểm nhất là loại virus ẩn hình chúng đính kèm vào các file exe, các file này vận hoạt động bình thường. Sau khi lây lan các mã virus sẽ thay đổi theo các phương pháp khác nhau để lừa dối người dùng về nó.

1.2.2 Cách hoạt động của Worm

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

1.2.3 Cách hoạt động của Trojan Horse

Loại mã độc này thường khiến mình diễn ra là một chương trình an toàn, vô hại vì máy tính người dùng. Chính vì vậy nó ngoài thực tế thì chức năng trình an toàn nó còn âm thầm thực thi các chức năng ẩn nấp trong đó mà khi nó cần thì mới thực thi.

1.3 Phương pháp thu thập mã độc

1.3.1 Các phương pháp thu thập mã độc

Có nhiều phương pháp thu thập mã độc xây dựng honeypot nhằm thu thập những mã độc mới:

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

Hacker lấy mã độc khách hàng tặng, hoặc lấy từ các nguồn chia sẻ trên mạng, hoặc mua từ các hãng nghiên cứu bảo mật... Đây là những phương pháp thu thập đưa vào máy do người dùng gửi. Phương pháp này thu thập được rất nhiều mã độc hàng ngày có rất nhiều chủ và cá nhân trên thị trường các mã độc có thể là mã độc hoặc không từ các trang web, hàng phishing virus.

Các trang web thu thập mã độc trên thị trường là:

- <https://www.virustotal.com>



VirusTotal là dịch vụ miễn phí, giúp **phân tích tập tin và URL nghi ngờ** và tạo điều kiện cho việc nhanh chóng phát hiện virus, sâu máy tính, trojan và tất cả các loại phần mềm độc hại khác:

Kích thước tập tin tối đa: 64MB

Bằng việc chọn nút 'Quét', bạn đồng ý với **Điều khoản dịch vụ** của chúng tôi và cho phép VirusTotal chia sẻ tập tin này với cộng đồng bảo mật. Xem qua **Chính sách riêng tư** để biết thêm chi tiết.

Có thể bạn cần quét URL hoặc tìm trong cơ sở dữ liệu của VirusTotal

English - Español - Français - Italiano - Português - Türkçe - Deutsch - Nederlands - Dansk - Norsk - Русский - български език - Ελληνικά - Срpski - Tiếng Việt - 日本語 - 한국어 - 中文 - اللغة العربية - فارسی

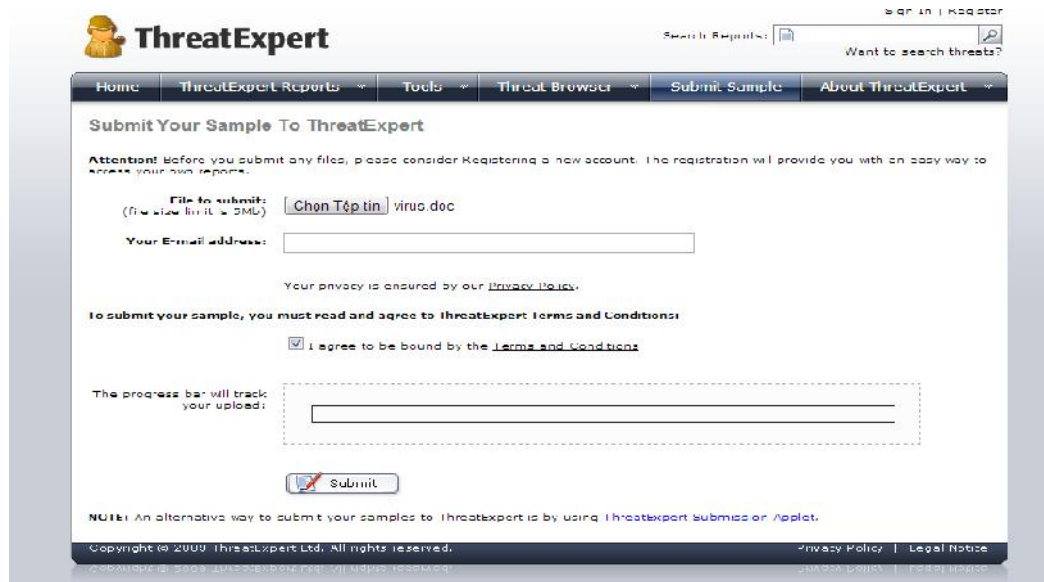
Hình 1.1 Giao diện chính virustotal

<div> <div>Cộng đồng</div> <div>Thống kê</div> <div>Tài liệu</div> <div>FAQ</div> <div>Giới thiệu</div> <div>Liên hệ</div> <div>Tham gia cộng đồng</div> <div>Đăng nhập</div> </div>		
Phân tích	Thông tin khác	Ý kiến
Chương trình	Kết quả	Cập nhật
Agnitum		20130519
AhnLab-V3		20130520
AntiVir		20130520
Antiy-AVL		20130519
Avast	Other:Malware-gen [To]	20130520
AVG	Exploit_c_UEK	20130520
BitDefender		20130520
ByteHero		20130517
CAT-QuickHeal		20130520
ClamAV		20130520
Commtouch		20130520
Comodo		20130520
DrWeb		20130520
Emsisoft		20130520
eSafe		20130516
ESET-NOD32		20130520
F-Prot		20130520
F-Secure		20130520
Fortinet		20130520
QData	Other:Malware-gen	20130520
Ikarus	Exploit:PDF	20130520
Jiangmin		20130520
K7AntiVirus		20130517
K7GW		20130517

Hình 1.2 Giao diện virustotal sau khi quét

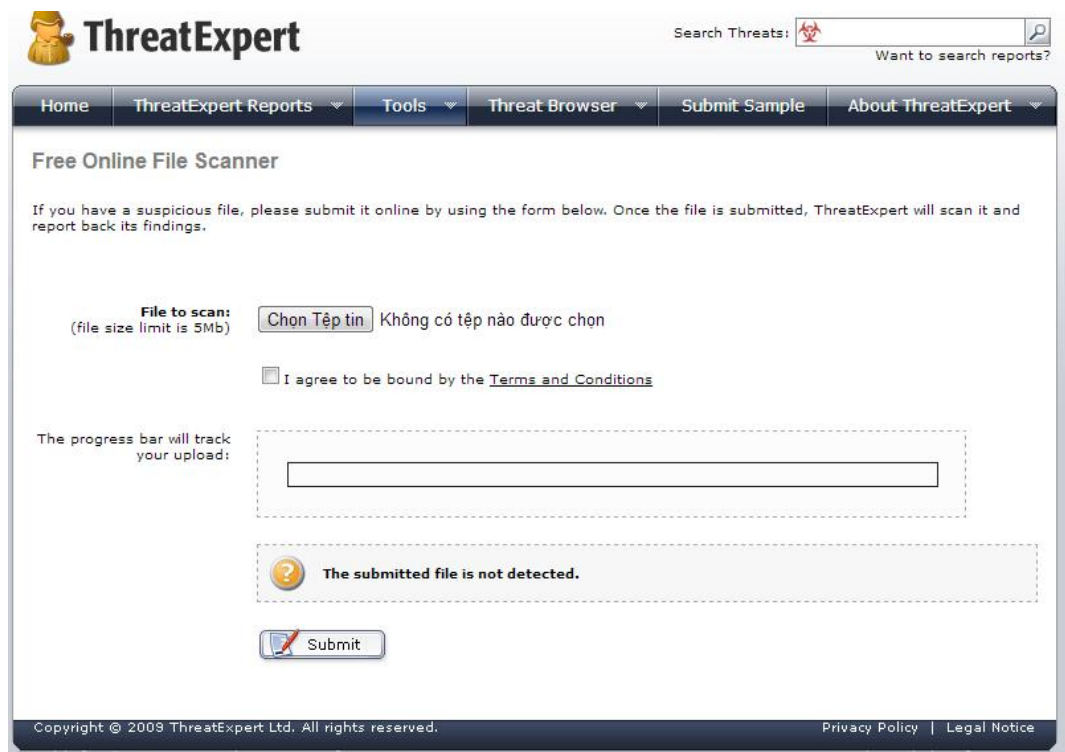
Virus total là trang web được google mua lại từ năm 2012 nó liên kết với gần 50 hãng phần mềm diệt virus lớn trên thế giới nhằm mục đích thu thập và phân tích nhanh chóng các mẫu mã độc hại để đưa ra cảnh báo và cách xử lý kịp thời. Cùng với đó là sự đóng góp của trang này thu thập các dữ liệu.

- ThreatExpert



The screenshot shows the ThreatExpert website interface. At the top, there is a navigation bar with links: Home, ThreatExpert Reports, Tools, Threat Browser, Submit Sample, and About ThreatExpert. The main heading is 'Submit Your Sample To ThreatExpert'. Below this, there is a section titled 'Attention! Before you submit any files, please consider Registering a new account. The registration will provide you with an easy way to access your own reports.' The form includes a 'File to submit:' field with a file size limit of 5MB and a 'Chọn Tập tin' button. Below this is a 'Your E-mail address:' field. A checkbox is present for 'I agree to be bound by the Terms and Conditions'. A progress bar is shown with the text 'The progress bar will track your upload:'. At the bottom, there is a 'Submit' button. The footer contains copyright information for 2009 ThreatExpert Ltd. and links to Privacy Policy and Legal Notice.

Hình 1.3 Giao diện trước khi quét của ThreatExpert



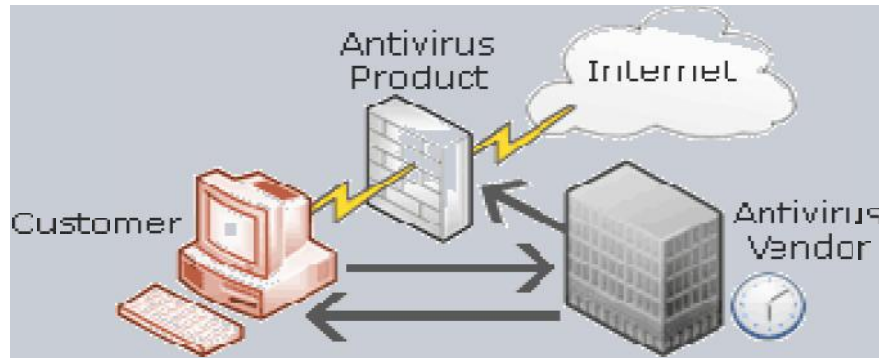
The screenshot shows the ThreatExpert website interface after a file upload attempt. The main heading is 'Free Online File Scanner'. Below this, there is a section titled 'If you have a suspicious file, please submit it online by using the form below. Once the file is submitted, ThreatExpert will scan it and report back its findings.' The form includes a 'File to scan:' field with a file size limit of 5MB and a 'Chọn Tập tin' button. Below this is a checkbox for 'I agree to be bound by the Terms and Conditions'. A progress bar is shown with the text 'The progress bar will track your upload:'. A message box with a question mark icon states 'The submitted file is not detected.' At the bottom, there is a 'Submit' button. The footer contains copyright information for 2009 ThreatExpert Ltd. and links to Privacy Policy and Legal Notice.

Hình 1.4 Giao diện sau khi quét của ThreatExpert

ThreatExpert là một hệ thống phân tích mã độc dựa trên các thuật toán 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 môi trường hoàn toàn tự động trong

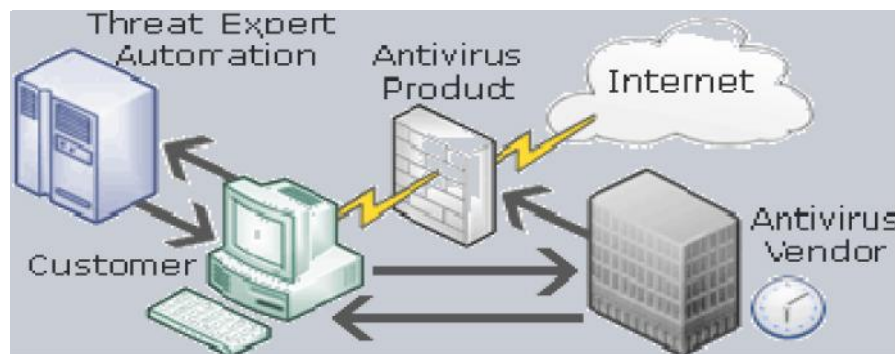
vài phút ThreatExpert có thể xử lý mọi và tất cả các mối đe dọa báo cáo rất chi tiết và chính xác chi tiết kỹ thuật phù hợp với quá trình chuyên công nghệ phòng chống virus chính xác như như những bình thường tìm thấy trong bách khoa toàn thư virus trực tuyến.

Hệ thống của ThreatExpert:



Hình 1.5 Hệ thống ThreatExpert

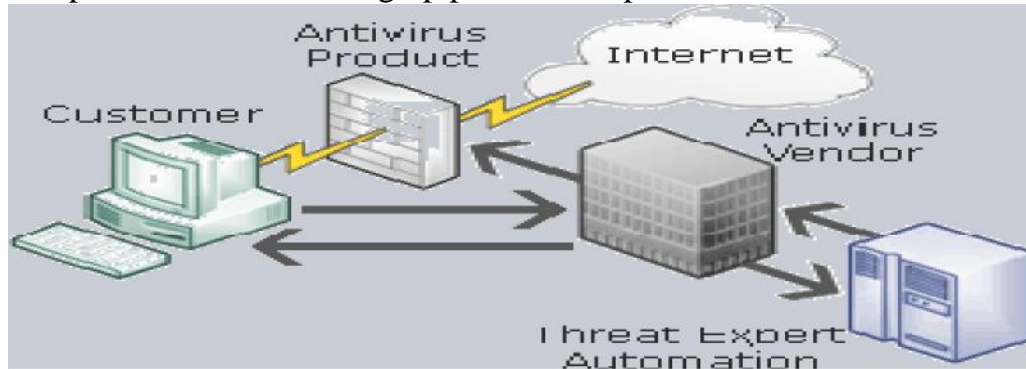
- Người sử dụng gửi dữ liệu ThreatExpert.
- ThreatExpert cung cấp phân tích mô tả chi tiết mối đe dọa ngay lập tức.
- Mô tả mối đe dọa có thể sử dụng bởi khách hàng để thực hiện giao dịch mua thêm mối đe dọa (như là thông tin hoặc báo cáo về mối đe dọa hoặc phòng) trực tiếp khi gửi cung cấp phần mềm diệt virus cập nhật.



Hình 1.6 Người dùng gửi dữ liệu

- Ngay như như những bình thường hệ thống nhận các dữ liệu từ các khách hàng, tham gia ThreatExpert, ThreatExpert cung cấp phân tích ngay lập tức.
- Mô tả mối đe dọa mới có thể cập nhật trực tiếp trên trang web của công ty của các nhà cung cấp, trực tiếp khi các nhà cung cấp khác có khả năng làm như vậy.

- Khách hàng khác của nhà cung cấp có thể cập nhật các nhà báo về mã mới để đảm bảo môi trường mới để đảm bảo.
- Nhà cung cấp có thể sử dụng các báo cáo chi tiết hành vi hacker trong phân tích phần mềm để giúp phát hiện ra phần mềm hacker.



Hình 1.7 ThreatExpert trợ giúp kiểm tra và ngăn chặn mã độc

1.3.2 Các công cụ thu thập mã độc

Thu thập mã độc trước tiên cần phải có những câu hỏi chuyên ngành. Có rất nhiều công cụ trình bày hướng dẫn thì không thể, trong phần này em sẽ không đi sâu vào các công cụ tìm mã độc hay các hành vi mà sẽ đi sâu vào các công cụ mà sẽ nêu phần “2.3.3 Mã giả lập” và một số công cụ bổ sung thêm.

1.3.2.1 IDA

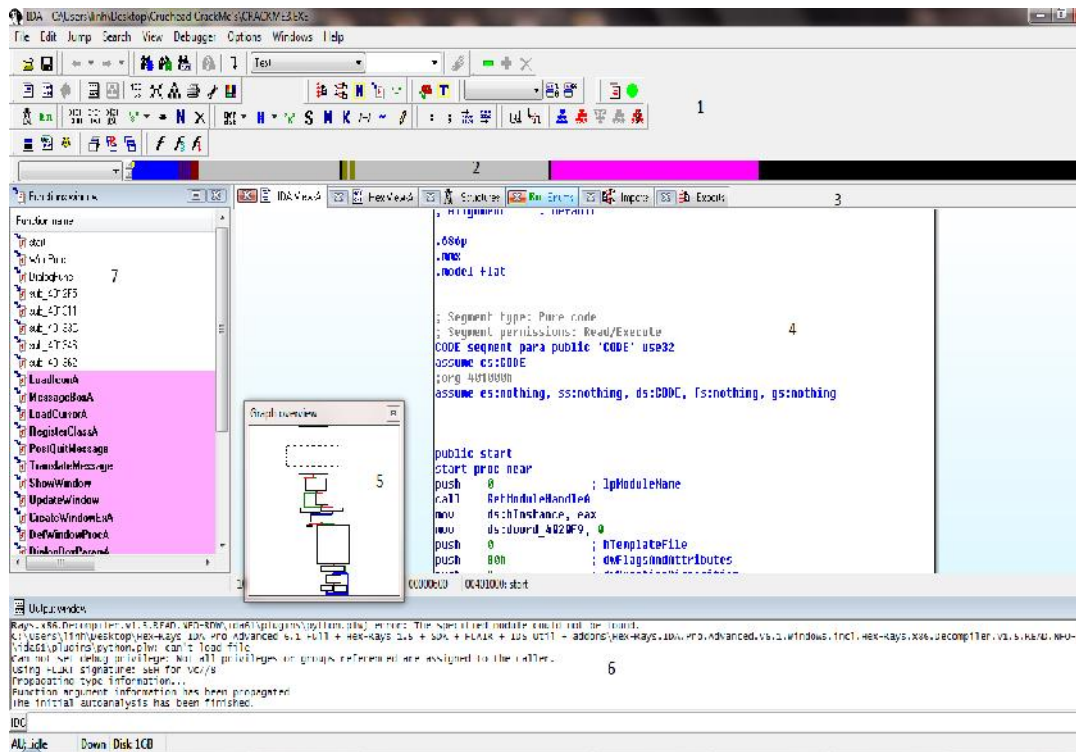
IDA là một công cụ để disassembly và debug. Tính năng nổi bật của nó là cho phép dịch ngược mã nguồn mà không cần load vào bộ nhớ. Đây là công cụ rất mạnh và hữu ích cho công việc lập trình và debug.

Cách sử dụng và tính năng:

- Sau khi load vào sẽ hiện lên một hộp thoại bên trong đó có chứa danh sách file mà chúng ta làm việc. Ta thường quen thuộc với PE(pe.ldw) làm việc. Trong Option là các tùy chọn cho IDA để phân tích file vào cho ra kết quả chính xác nhất. Sau đó nhấn OK vào giao diện chính làm việc. Giao diện là vì có 7 cửa sổ:

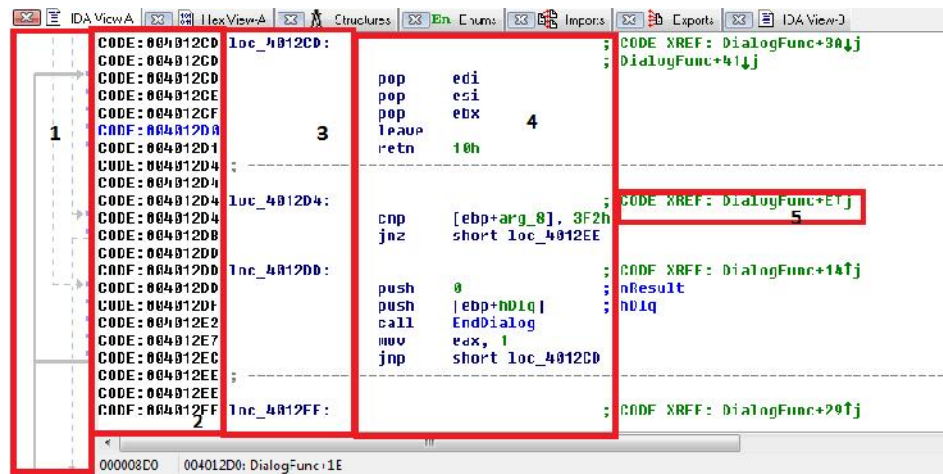
- (1) Toolbar chứa các thanh công cụ sử dụng trong hoạt động của IDA, thể hiện các chức năng ta nhập vào các biểu tượng trên đó.
- (2) Navigation band nơi ta có thể vào và ra khi cần biết cách nhập chuỗi di chuyển bằng cách nhấp chuột theo hướng màu vàng. Mỗi màu sắc khác nhau thể hiện vùng dữ liệu mà ta làm việc.
- (3) Tabs có chứa các danh sách thông tin chi tiết về file hiện tại, vì phân tích phụ thuộc vào những tabs này. Giao diện IDA View-A, Hex View-A, Struct, Enums, Imports, Exports, String.

- (4) Disassembly hiển thị dữ liệu chúng ta phân tích theo 2 loại text hoặc graph
- (5) Graph overview mô tả thu nhập mô tả cấu trúc cơ bản của dữ liệu. Màn hình chức năng cho thấy màn hình hiển thị vị trí hiển thị tương ứng làm việc.
- (6) Output window nhận các thông tin, tin nhắn từ IDA sau khi load file xong.



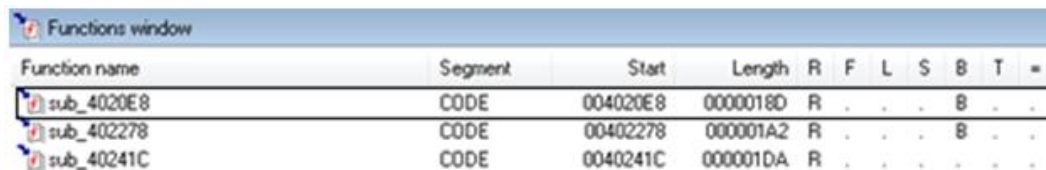
Hình 1.8 Giao diện IDA

- (7) Function window đây là các hiển thị tất cả các hàm API các và tìm thấy trong quá trình phân tích.
- IDA Text view màn hình hiển thị kiểu text trình bày toàn bộ công việc disassembler có thể cung cấp để xem các dữ liệu thu được vùng nào.
 - (1) Các mã tên hiển thị trong khi nó mã và nhận ra các vòng lặp.
 - (2) Hiển thị các địa chỉ Virtual Address
 - (3) Các vị trí so sánh ảnh hưởng hoặc các bit tham chiếu trong stack



Hình 1.9 IDA Text view

- (4) Code của chương trình disassembled
- (5) Code tham chiếu hiện tại các hàm khi truy cập, nhúng vào sáo ta nên nhớ, hoặc hàm các ghi hoặc nhúng lên phía trên. Hoặc nhúng vị trí tham chiếu khác ta nhúng chuốt phi->jump to cross reference.
- Function Window hiện tại các hàm của chương trình nhúng vào IDA



Hình 1.10 Function Windows

Bằng cách nháp chu trình hoán vị enter là ta có thể di chuyển các hàm vào trong các khối 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 (library), S (static), B(BP tham chiếu đến biến cục bộ), T (type thông tin).

- String Window t i c a s này hi n th ra thông tin t t c các chu i xu t hi n trong file th c thi. C a s này hi n th chi ti t v a ch , dài, ki u, tên chu i ó là gì. Ta có th truy c p n các chu i này b ng cách nh n úp chu t nó s di chuy n n c a s Disassembly chúng ta th c hi n.
- Import window c a s hi n th chi ti t a ch , tên hàm c import và th v i n ch a hàm import ó. ây là m t c a s r t quan tr ng b i ta có th th y c ch ng trình s d ng các DLL khác nhau và ch c n ng c a các hàm c g i nh c, vi t ho c registry T i ây ta có th th p c các thông tin v các hàm import mà mã c h i hay dùng chèn vào v i m c ích x u nh l y c p thông tin, theo dõi ...



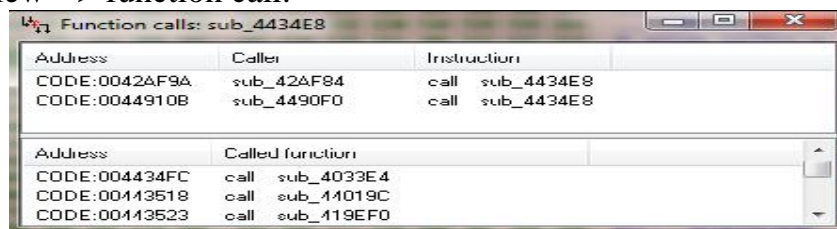
Hình 1.11 Import Windows

- Export window là cửa sổ liệt kê các entrypoint của tệp tin. Trong cửa sổ này người dùng có thể liệt kê theo tên, địa chỉ và thuộc tính (nếu có). Khi xem các file thì export window cần phải chứa ít nhất một entry point đây là địa chỉ mà chương trình bắt đầu thực thi.
- Cross-references cửa sổ hiển thị tất cả các tham chiếu, là tất cả các vị trí mà code gọi đến hàm. Khi xem cửa sổ này ta chỉ cần nhìn 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

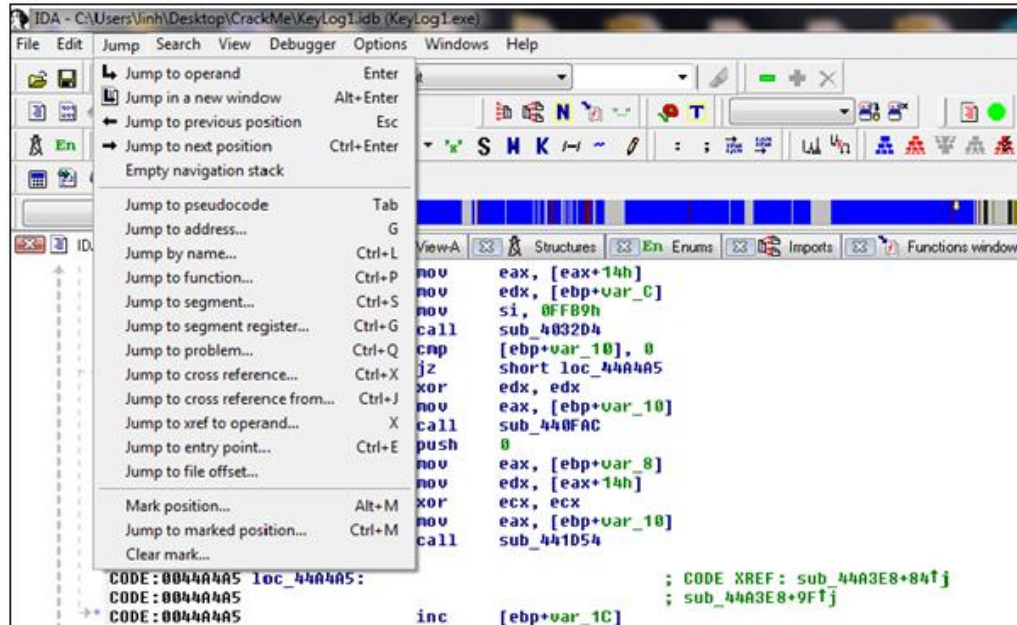
- Name Window là cửa sổ cung cấp danh sách các tên biến, tên có thể được sắp xếp theo bảng chữ cái hoặc virtual address. Khi nhìn thấy tên có các chữ cái in hoa khác nhau A(string data), D(data), C(Name code), I(import name), L (library function), F(regular function). Bảng vì chúng giúp chúng ta vào tên biến khi ta muốn tìm, chúng trình bày chuyển đổi vị trí ở trong cửa sổ chính hiển thị làm việc. Chọn view --> option subview --> name.
- Function Calls cửa sổ hiển thị 2 chức năng caller và called function. Xác định các hàm gọi xung quanh đó là gì. View --> option subview --> function call.



Hình 1.13 Function Call

Còn rất nhiều những cách khác với các cách trên khác nhau, những cách sẽ được đề cập ở phần tiếp theo là những cách quan trọng nhất trong việc làm việc với công cụ IDA pro.

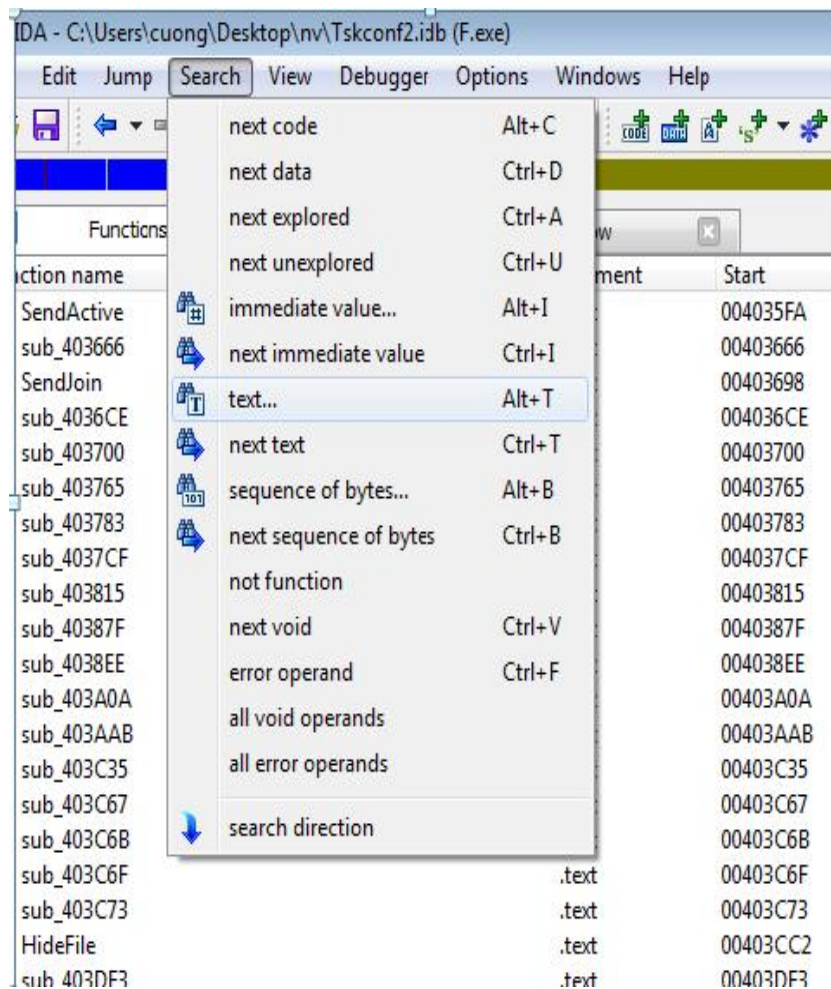
- Menu Jump



Hình 1.14 Menu Jump

- Jump to address(G) khi bạn muốn nhảy đến địa chỉ chính xác mà bạn muốn, bạn cần nhập địa chỉ này vào trường địa chỉ của menu jump.
- Jump to entry point(ctrl-E) hiển thị vị trí các entry point bạn muốn nhảy đến.
- Sử dụng forward/ backward arrows bằng cách sử dụng nút jump trên thanh công cụ để nhảy đến vị trí trước (Esc) hoặc nhảy đến vị trí tiếp theo (ctrl-enter), hoặc sử dụng nút trên màn hình làm việc.

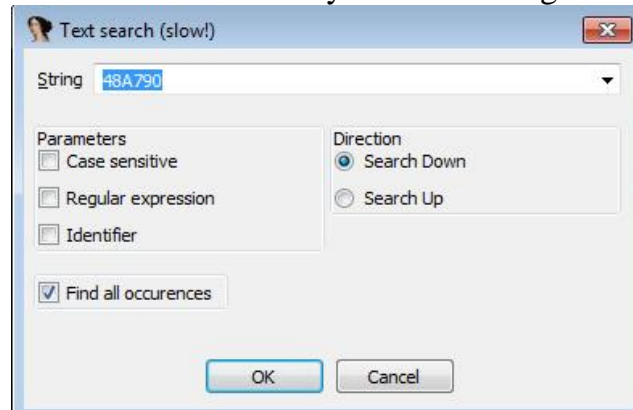
- Menu Search với các tùy chọn tìm kiếm:



Hình 1.15 Menu search

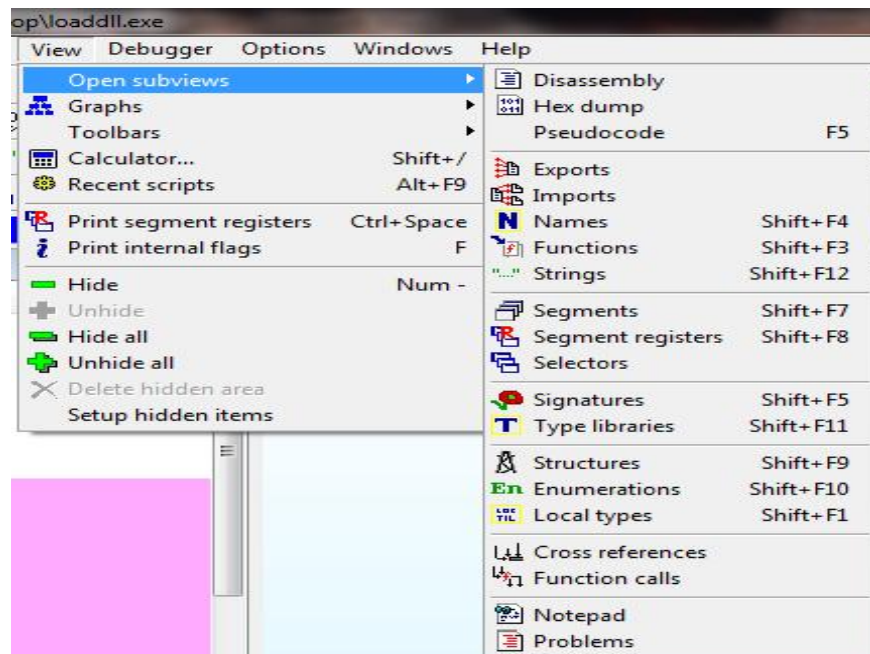
- Next code sẽ tìm kiếm vùng code tiếp theo (vùng này đã được phân tích, Disasm).
- Next data là sẽ tìm kiếm vùng chứa dữ liệu tiếp theo.
- Next explored sẽ tìm kiếm vùng dữ liệu tiếp theo đã được phân tích, đã được thành struct.
- Next unexplored thì ngược với next explored.
- Immediate value sẽ hiển thị ra các tìm kiếm các giá trị trong các struct, data.

- Text tìm kiếm các chuỗi có ký tự đặc biệt trong tất cả các hàm, dữ liệu.



Hình 1.16 Text search

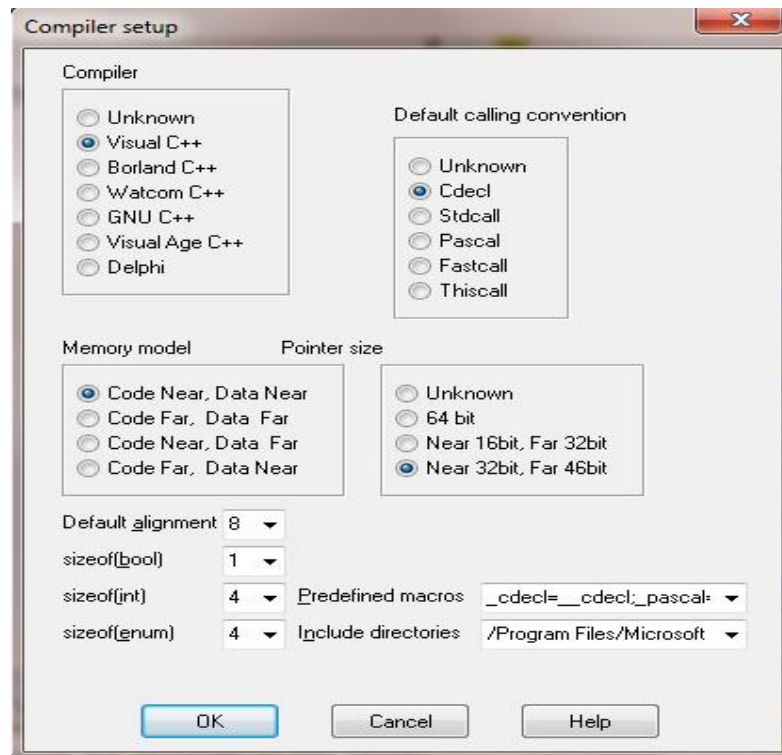
- Menu View



Hình 1.17 Menu View

- View open subview là menu các cửa sổ view con trên các tab view: DisAssembly, hexdump, export import ... là các cửa sổ đã có sẵn. Có thể thêm các cửa sổ như :
- PseudoCode là cửa sổ chứa mã C để tái tạo lại các function hiển thị các Disassembly.

- Segmen, SegmenRegister chứa các thông tin về các segmen, các thanh ghi, các vùng dữ liệu.
- Signature, Type Lib chứa các ký xác nhận các chương trình dùng thử vì n gì, xây dựng trên ngôn ngữ, c s gì.
- Tiếp theo sẽ là menu debugger chứa các tùy chọn liên quan đến debugger gắn kèm theo IDA, nhưng ta tạm thời không xét vì các debugger này khá phức tạp, ta sẽ tìm hiểu debugger n gì n h n là OllyDebugger.
 - Menu Options
 - Menu này chứa các tùy chọn về compiler:



Hình 1.18 Compiler setup

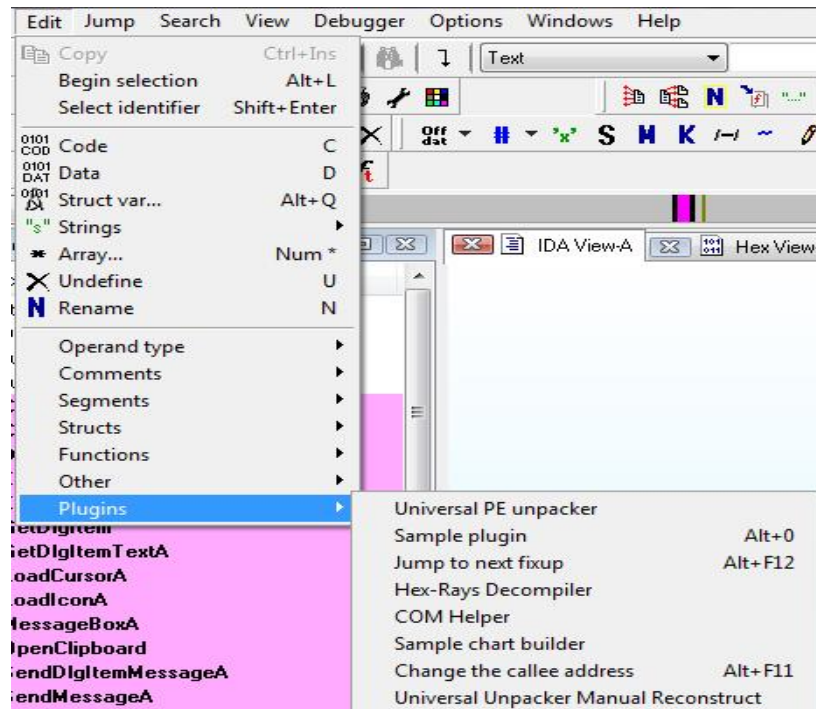
- Disassembly cho phù hợp với các trình biên dịch
- Tùy chọn string style chứa các kiểu string tùy theo trình biên dịch, kiểu dữ liệu.
- Ngoài ra còn rất nhiều các tùy chọn liên quan đến cách đặt tên cách hiển thị biến, chú thích...
- Menu Edit
 - Chọn các tùy chọn:
 - Code: chuyển mã vùng dữ liệu bytecode thành AsmCode.
 - Data: chuyển mã vùng dữ liệu từ AsmCode thành các bytecode.
 - String: chuyển mã 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 để cài các Plugin mà ta cài thêm vào.

Các chú ý nhất là Plugin HexRays Decompiler và phím tắt là F5 sẽ dịch mã thành mã Assembly thành mã C.

- Plugin BinDiff cũng là Một plugin rất quan trọng cho phép ta so sánh các dữ liệu hint và mã lệnh khác. Chạy các function tương tự, gì nhau...



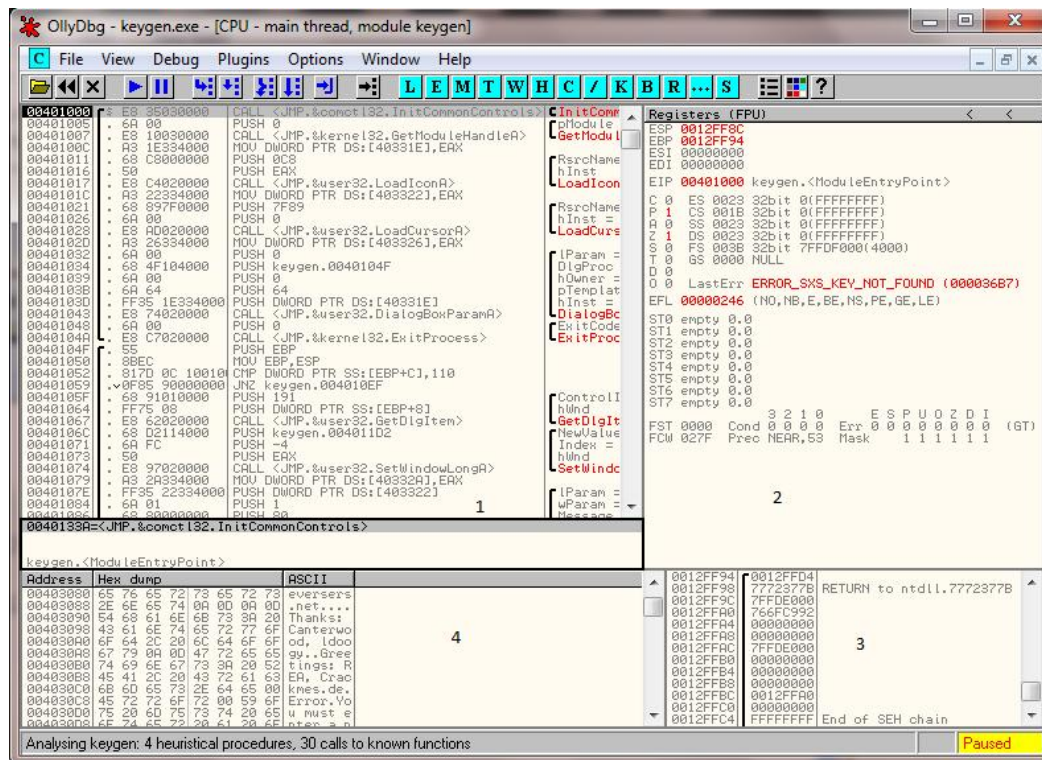
Hình 1.19 Plugin

1.3.2.2 Olly Debug

Olly Debug là công cụ debug có tuổi đời khá lâu và phổ biến trong cộng đồng cracker và developer, nó giúp cho người sử dụng toàn quyền can thiệp vào chương trình mã nguồn 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 có giao diện như sau:

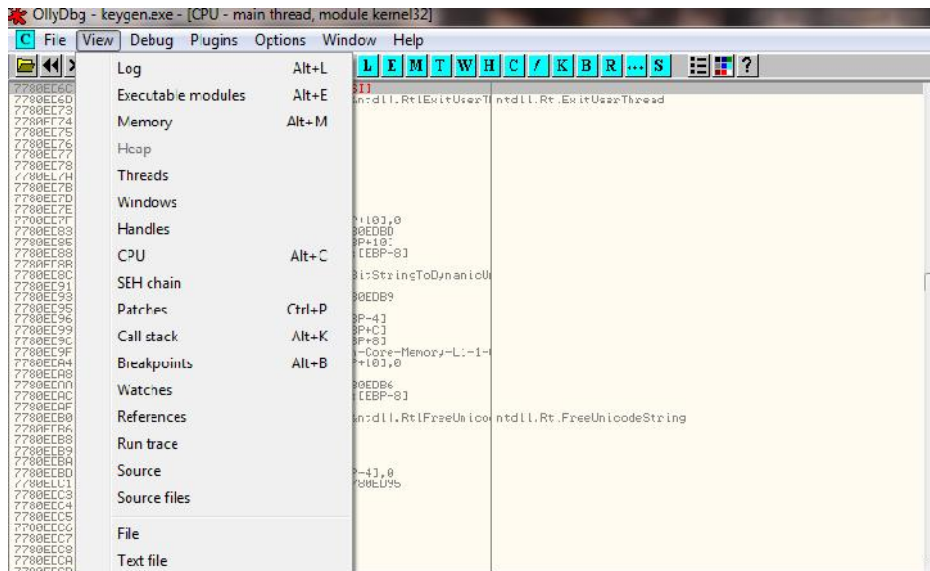


Hình 1.20 Giao diện Olly Debug

- The Disassembler Window: cửa sổ này ta nhìn thấy các dòng chương trình disasm cùng với nhúng lại gì thích cho các dòng mã asm.
- The Registry Window: đây là cửa sổ chứa thông tin chi tiết về các thanh ghi, các cấu trúc.
- The Stack Window: hiển thị trạng thái của stack, lưu trữ dữ liệu 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 thấy các chức năng chương trình s.a...
- Các tùy chọn 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 library hoặc các Plugins được load cùng chương trình thì khi mà ưu tiên khi load chương trình vào Olly, cùng với ghi lại các thông tin về BP. Và lúc chức năng này là khi ta muốn lưu lại thông tin về file Log ta chỉ cần nhấp chuột phải trong cửa sổ L và chọn Log to file
 - View --> Executable module (cửa sổ E) cửa sổ này ra danh sách những file có khả năng thực thi các 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 sử dụng. Tại cửa sổ này chúng ta có thể sử dụng tính năng search tìm kiếm thông tin về các string, các dòng hexa code

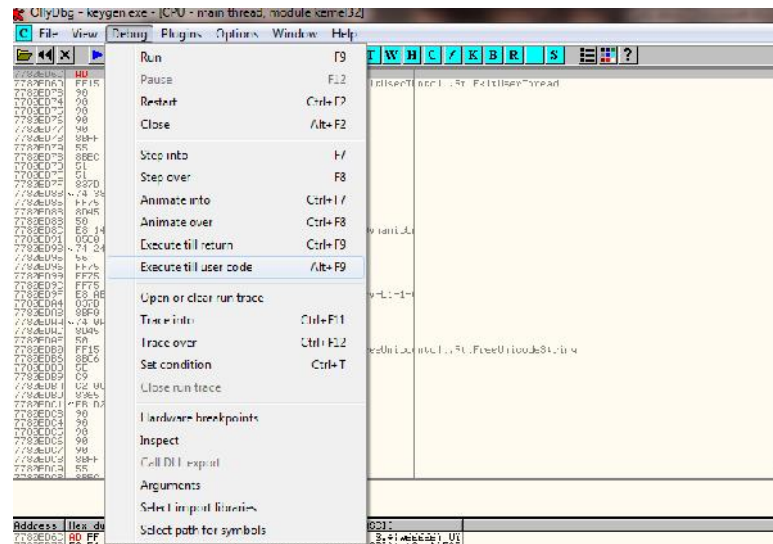
hay unicode... thêm vào đó cùng cấp cho chúng ta những kiểu thì t BP khác nhau thì section.

- View --> Threads (c a s T) liệt kê các thread của chương trình.
- View --> window(c a s W) mở các Window.
- View --> Handles(c a s H) mở các Handles.



Hình 1.21 Tùy chọn View

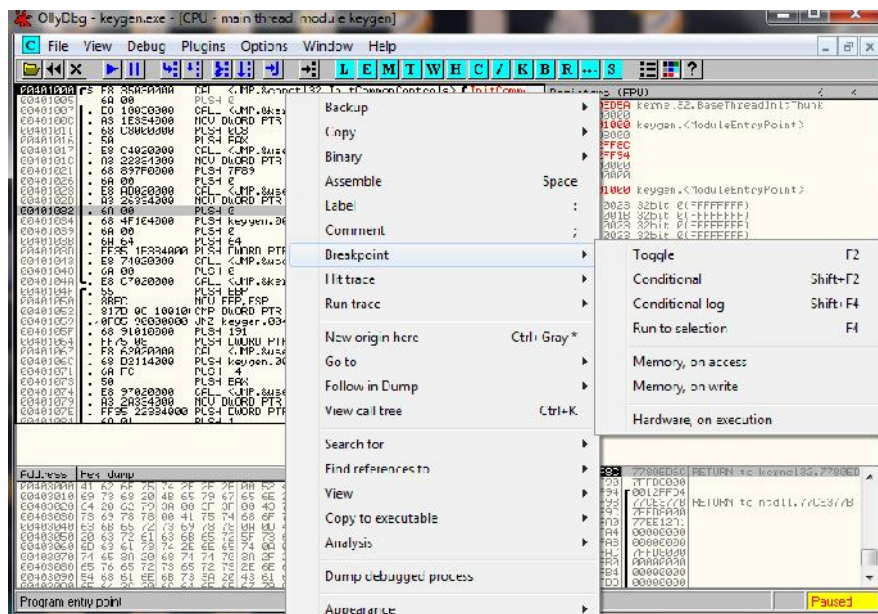
- View --> CPU(c a s C) của sang làm việc hiện tại.
 - View --> Patches(c a s /) của này cho chúng ta biết thông tin về những gì mà chúng ta đã edit trong chương trình.
 - View --> Call stack(c a s K) hiển thị danh sách các lệnh Call mà chương trình của chúng ta đã thực hiện khi chúng ta Run bằng F9 và dùng F12 để tạm dừng chương trình.
 - View --> Breakpoint(c a s B) hiển thị tất cả các BP mà chúng ta đặt trong chương trình, Nó sẽ hiển thị những BP đã set bằng F2 còn lại về những điểm BP khác như HWBP và Memory BP sẽ không hiển thị.
 - View --> References(c a s R) hiển thị kết quả khi thực hiện chức năng search trong Olly.
- Các tùy chọn ở View Debug
 - F9(Run) khi nhấn olly sẽ tìm xem có breakpoint nào đã set không, chương trình có tung ra các exception gì không, hay nếu chương trình có cách 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 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 thoát ra và vị trí hiện tại.
- F7(step into) thì sẽ trace từng dòng lệnh, trong quá trình trace gặp lệnh Call sẽ nhảy vào bên trong lệnh Call và sẽ trace lệnh bên trong Call, khi nào gặp return sẽ trở về chương trình chính.
- F8(step over) thì sẽ trace từng dòng lệnh nhưng khác với F7 là khi gặp lệnh Call nó sẽ không nhảy vào bên trong trình mà dừng lại ngay câu lệnh tiếp theo của lệnh Call.
- Tùy chọn Plugins
- Plugins --> Ollydump --> dump debugged process dump một vùng nhớ trong process.
- Option --> Debugging Option(Alt-O) theo mặc định chọn auto start analysis thì chương trình khi được load vào sẽ tự động phân tích và sẽ có các comment thích hợp. Nếu không chọn chức năng auto start analysis thì chúng ta sẽ phải thực hiện manual sau khi chương trình được load vào.
- Chuột phải vào các Disassembler có các tùy chọn như chú ý sau:
- Binary --> edit thay đổi giá trị trong hex dump
- Goto --> expression(Ctrl+G) để tìm kiếm địa chỉ.
- Goto --> previous quay trở lại call sau khi chọn follow
- New origin here(Ctrl+ Gray*) để chọn chương trình mới thì sẽ hiện ra một địa chỉ khác để vị trí mà ta chọn Ctrl+Gray*

- Follow(enter) cho phép ta xem câu lệnh trong call ,mà bên thân nó không hiển thị bất kỳ câu lệnh nào của chương trình.Follow in dump cho ta xem giá trị vị trí ta chọn trong cửa sổ dump.
- tìm kiếm các function hay string trong ollydebug:
 - Với các function ta chọn search for->all intermodular calls, hay chọn go to --> expression to follow và nhập tên hàm vào bên trong.
 - Với tìm kiếm các string ta chọn search for-> all referenced text string
 - Để tìm kiếm các hàm và chuỗi ta sử dụng BreakPoint(BP) thì có nhiều loại ảnh hưởng cho ta thể hiện chương trình. BreakPoint chia ra làm các loại sau: Common BreakPoint, Memory BreakPoint, Hardware BreakPoint,Conditional.



Hình 2.23 BreakPoint

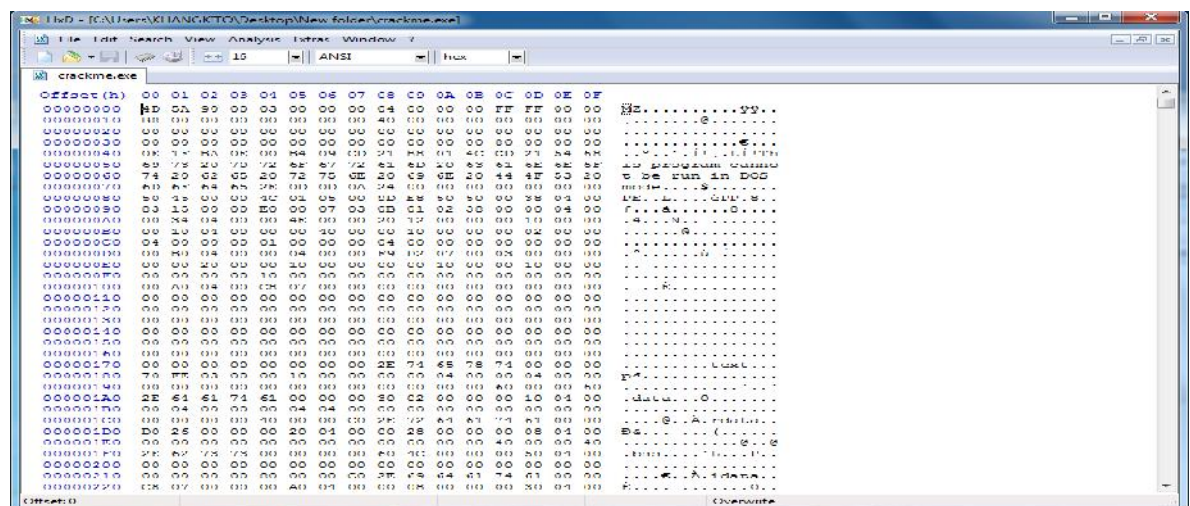
- Common BP ta sử dụng cách tìm kiếm function hoặc string rồi nhấn F2, hoặc nhập thông qua command. Olly sẽ lưu lại mã BP này tại cửa sổ BreakPoint ta có thể mã này ra kiểm tra. Vì code BP trên chỉ ảnh hưởng opcode không thay đổi trong suốt quá trình thực hiện chương trình.
- Memory BP vì code BP này áp dụng cho những opcodes có thể thay đổi và Olly hỗ trợ chúng ta 2 kiểu BP này trên memory là Breakpoint Memory on access và BreakPoint Memory on write. BreakPoint Memory on access vì code BP này lên mã vùng nhớ cho phép ta debug thể hiện chương trình khi có bất kỳ sự thể hiện nào, hoặc ghi lên vùng dữ liệu mà ta đặt BP.BreakPoint Memory on write debug chương trình thể hiện khi có bất kỳ dữ liệu nào ghi lên vùng nhớ mà đặt BP. Vì code BP tại memory sẽ không lưu lại

thông tin tức của BreakPoint. Ollympic chỉ cho một duy nhất 1 BreakPoint trong memory nên khi thêm BreakPoint thứ 2 vào thì BreakPoint thứ 1 sẽ bị xóa.

- Hardware BreakPoint(HWBP) ta có thể thêm 4 HWBP nhưng chỉ có 1 trong số vị trí memory breakpoint thì mới có thể thêm được BP, không sử dụng ngắt INT3 mà sử dụng ngắt INT1. HWBP chỉ hoạt động trên CPU và sử dụng một số thanh ghi cụ thể là debug registry. Chúng ta sử dụng HWBP bởi vì nó không làm thay đổi các biến, stack. Chúng ta không làm chương trình bị lỗi. Chúng ta thêm HWBP thì nó sẽ không thay đổi dữ liệu nào cho nên nó là tốt nhất, nên khi kiểm tra xem ta thêm nó ở đâu thì debug-> Hardware Breakpoint. Với HWBP on write và HWBP on access thì ta chọn tùy ý muốn của dump, sau đó thêm BP.
- Conditional Breakpoint(shift+F2) có nghĩa là breakpoint thông thường, tuy nhiên vì có thể Conditional BP phải thỏa mãn một điều kiện nào đó thì mới phát tác. Ta thêm BP-> Conditional nhập vào điều kiện bên trong ô đó. Chọn loại trong bảng Breakpoint.
- Conditional log Breakpoint(shift+F4) có nghĩa là breakpoint điều kiện nhưng nó có thêm tùy chọn cho phép ta lưu trữ giá trị của biến hoặc các tham số function mà khi xảy ra breakpoint hoặc khi thỏa mãn điều kiện mà ta yêu cầu. Nhập thông tin lưu trữ và giá trị này sẽ lưu trữ Log(L) của Ollympic.

1.3.2.3 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 và người dùng có thể kiểm tra và sửa đổi file. Đây là công cụ hữu dụng khi đã xác định được vị trí của chuỗi ký tự cần sửa đổi vào các dữ liệu mà không cần dùng đến công cụ bên thứ ba.



Hình 1.24 Giao diện HxD

1.4 Quy trình phân tích mã nhị phân

Đầu tiên ta sẽ tìm hiểu về các phương pháp phân tích mã nhị phân để có thể đưa ra quy trình phân tích mã nhị phân.

1.4.1 Các phương pháp phân tích mã nhị phân

Phương pháp phân tích thì có 2 phương pháp chính :

- Dynamic analysis (phân tích hành vi mã nhị phân).
- Static analysis (phân tích mã nhị phân bằng cách xem mã disassembly của mã nhị phân).

Tuy nhiên không phải mã nhị phân nào cũng phân tích theo hai phương pháp trên, bởi vì có những mã nhị phân có thể chỉ cần phân tích một hoặc có những lỗi chỉ cần phân tích một. Vì vậy chúng ta có cách thức phân tích riêng của mình dựa trên kinh nghiệm và thực tiễn phân tích để tránh mất thời gian cũng như công sức bỏ ra. Chính vì thế hai phương pháp trên chỉ là tổng quát.

1.4.1.1 Static analysis

Static analysis thì ta sẽ xem code disassembly của mã nhị phân này sẽ làm gì trên hệ thống. Không cần phải chạy mã nhị phân đó, và hiểu rõ hơn về các virus hoặc mã độc hại. Các công cụ thường sử dụng trong quá trình phân tích thì có thể kể đến các chương trình disassembler :

- Disassembler
- Decompiler
- Source Code Analyzer

Ưu điểm :

Static analysis là có thể phát hiện ra hoạt động, cách thức của chương trình trong những lúc không thể thực thi.

Static analysis sẽ cho chúng ta cái nhìn rất đúng đắn về mặt chức năng của chương trình. Sẽ dễ dàng nhận biết được khi chúng ta có thể nắm được toàn bộ hoạt động của chương trình, từ các chương trình con.

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 chí có những lỗi mà thời gian phân tích của nó tính bằng tháng.

1.4.1.2 Dynamic analysis

Dynamic analysis thì là quan sát xem mã của hệ 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 những điểm về các dòng mã của hệ chạy theo lịch trình. Nghiên cứu hoạt động của chương trình bằng cách thực thi chương trình đó. Các công cụ sử dụng trong trình này có thể kể 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: "Thực nghiệm thực tế không phải là thực nghiệm gì chương trình có". Nói cách khác dynamic analysis không thể đoán được những hành vi của chương trình trong các "input" "output" không tồn tại trong thực tế. Có thể lấy ví dụ về các mã của hệ chạy theo thời gian, thực tế là thời điểm này mình cho nó chạy nó không có hoạt động gì, nhưng mà thực tế thì điểm khác nó lịch sự.

Mỗi phương pháp đều có một điểm riêng nhưng khi phân tích mã của hệ thì chúng ta thường phải sử dụng 2 phương pháp hỗ trợ lẫn nhau:

Dù mục đích chính thì gì nhưng nhau nhưng các công cụ hỗ trợ về phân tích, thời gian của các phương pháp là khác nhau.

1.4.2 Các bước phân tích mã của hệ

1.4.2.1 Phân tích thông tin sơ bộ

Đầu tiên nên nhìn nhận đầu tiên trên máy tính có những gì ? Máy chủ hay là ứng dụng mạng, tất nhiên khi mà tra xem có các processes gì trên máy tính không, có những bộ thực thi xảy ra trên máy tính không?... Sau đó là thu thập dữ liệu nghi ngờ là mã của hệ.

Nhận diện những bộ phận mã của hệ :

- Thông tin về hệ thống đang sử dụng.

- Trình duyệt Web.
- Firewall.
- Các trình bảo vệ máy hiện tại đã có.

Tôi đã xác định 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 sau này còn có thể xác định nguồn gốc lây lan và tìm sao lại bản nháp thông qua đó. Nhưng tôi cũng ra soát lại một lần nữa để xem hệ thống mình có dính lại ổ đĩa nào không.

Đầu tiên, phân loại mã độc hại, chia dòng, đặt tên. Bằng cách xem mã hex, xem properties của files, kích thước file... hoặc thông tin ngắn gọn khác 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 nhận diện chưa. Nếu mã độc hại đã nhận diện, chúng ta sẽ tìm kiếm thêm thông tin về mã độc hại đó làm cơ sở để phân tích chi tiết.

1.4.2.2 Quan sát hành vi mã độc hại (Dynamic analysis)

Thiết lập môi trường thử nghiệm: Sử dụng máy ảo, sandbox, virtual PC cho mã độc hại chạy. Nếu mã độc hại có chức năng phát hiện môi trường ảo có thể phớt lờ môi trường thử nghiệm thì cần giả lập (trong LAN) thử nghiệm. Thiết lập mạng Internet, web, mail, cài đặt các hệ thống liên quan mã độc hại có khai thác liên lạc... Môi trường thử nghiệm càng đầy đủ, càng "thật" thì sự quan sát càng hiệu quả và chính xác mã độc hại. Nếu xây dựng các hệ thống máy nháp máy bả nháp 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
-

Bằng công cụ này sẽ giúp chúng ta quan sát mã độc hại thật hơn. Bắt đầu với 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 khi khởi động và khi shutdown có key nào được sinh ra và bị xóa đi không?
- Kiểm tra MD5, CRC, SHA các modules (files) đang chạy trên hệ thống xem có bị mã hóa hay không?
- Khảo sát các files, các folder để tìm các file nghi ngờ.
- Khảo sát các driver, tìm rootkit làm nhiệm vụ, key.
- Khảo sát lưu lượng mạng (Dùng Dumetter) xem có đáng ngờ không?
- Kiểm tra xem các kết nối TCP/IP trên máy.

Chạy mã hóa và ghi log, quan sát càng lâu kết quả càng chính xác. Thông thường mã hóa 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 trên thiết bị.

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.

1.4.2.3 Phân tích mã độc tĩnh (Static Analysis)

Sau khi phân tích mã độc bằng cách quan sát hành vi thì ta sẽ thu được một số thông tin, nhưng nhược yếu của nó, ta vẫn cần đến kỹ thuật mã hóa để xem chi tiết rõ hơn liệu mã độc còn làm gì nữa không vào thời điểm nào nữa.

Do mã độc khi ta thu được là dạng thực thi không có mã nguồn kèm theo phân tích nên thường phải dùng kỹ thuật mã hóa để xem mã assembly phân tích code. Thường bước này sẽ ta sẽ làm như sau:

- Xem mã độc bằng cách viết bằng ngôn ngữ gì?
- Sử dụng các packer nén lại hay các protector nào không?
- Khi bị mã độc pack bằng trình nào thì có thể dùng trình để unpack ra, dùng công cụ unprotector file mã độc.
- Dùng các công cụ phù hợp với công cụ viết mã độc để decompile, disassemble, debugging.
- Đọc và phân tích mã code assembly để tìm thêm hoạt động của mã độc.

1.4.2.4 Trace Code Debug

Khi viết kỹ thuật sẽ rất khó khăn hoặc thì hiểu quá. Hoặc khi cần làm thì rõ ràng nó mã hóa thì phải debug mã độc. Có nghĩa là sẽ lần lượt nhìn nhận mã độc xem chính xác là mã độc đã làm như thế nào.

thường có các kỹ thuật chống debug thì cũng sẽ có các kỹ thuật debug (anti debug) và ngược lại phân tích phần virus qua nó (anti anti-debug).

1.4.2.5 Tìm signature của mã chèn và đưa vào cơ sở dữ liệu

Tìm kiếm minh chứng mã chèn trong hệ thống sau khi phân tích (ví dụ tìm offset nào có chứa string gì, hoặc MD5 của file).

Thì những minh chứng trên sẽ đưa vào cơ sở dữ liệu.

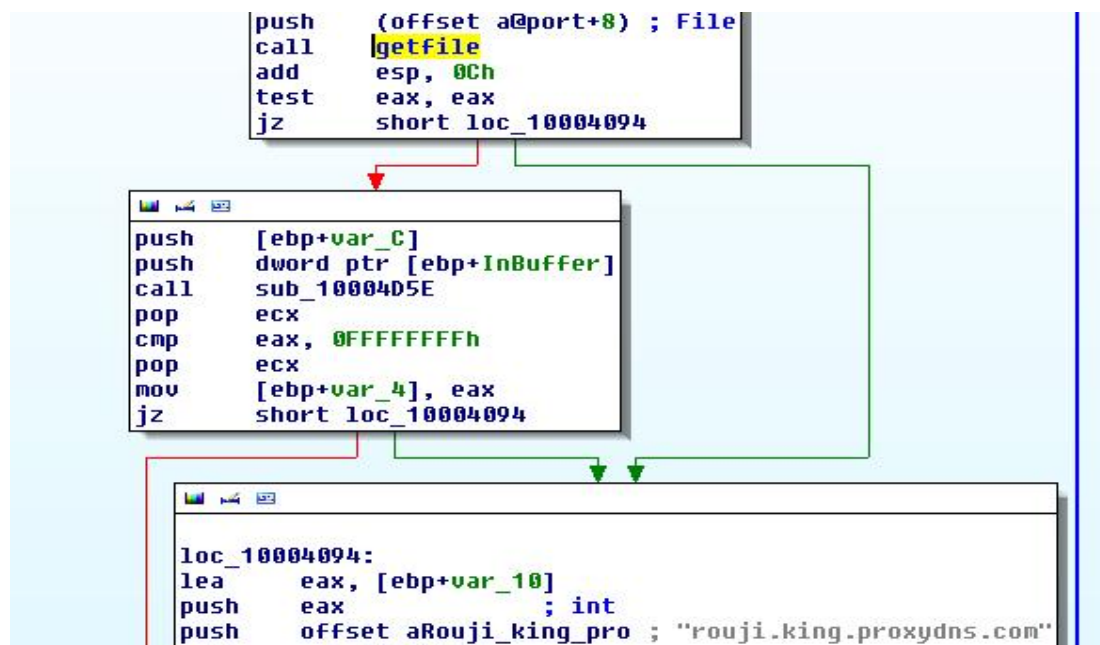
1.4.3 Phân tích một mẫu mã độc thực

Đây đây là quá trình phân tích mẫu mã độc thực có tên là netui.dll trích xuất ra từ một file excel. Đây em chỉ dùng chương trình IDA vì tài không đi sâu vào phân tích.

- Sử dụng IDA tiến hành disassembly

Get file “)(!@PORT” và máy tính nhận sẽ gửi hàm :
URLDownloadToCacheFile tới *rouji.king.proxydns.com*

```
.text:10004B0A      lea     eax, [ebp+FileName]
.text:10004B10      push   104h          ; cchFileName
.text:10004B15      push   eax            ; LPSTR
.text:10004B16      push   [ebp+File]     ; LPCSTR
.text:10004B19      push   ebx            ; LPUNKNOWN
.text:10004B1A      call   URLDownloadToCacheFileA
```



- Ta thấy có hàm GetKeyState có dùng:

1000F350	GetKeyState	USER32
1000F188	GetLastError	KERNEL32
1000F1B8	GetLocaleInfoA	KERNEL32

Tây ta tìm ra các mã nguồn code sử dụng hàm này và thấy rằng nó có dùng với mục đích là ghi lại trạng thái bàn phím.

Đi tiếp là nguồn của hàm keylog

```

mov     esi, ds:SetWindowsHookExA
push    ebx                ; dwThreadId
push    hmod               ; hmod
push    offset Keylog      ; lpfn
push    2                  ; idHook
call    esi ; SetWindowsHookExA
push    ebx                ; dwThreadId
mov     ds:hkh, eax
push    hmod               ; hmod
push    offset sub_10006105 ; lpfn
push    7                  ; idHook
call    esi ; SetWindowsHookExA

```

Một phần mã nguồn của hàm keylog

```

text:100064D6 loc_100064D6:                ; CODE XREF: Keylog+384Tj
text:100064D6                push    offset aPageup ; "[PageUp]"
text:100064DB                jmp     loc_10006583
text:100064E0                ; -----
text:100064E0 loc_100064E0:                ; CODE XREF: Keylog+381Tj
text:100064E0                push    offset asc_10016598 ; ""
text:100064E5                jmp     loc_10006583
text:100064EA                ; -----
text:100064EA loc_100064EA:                ; CODE XREF: Keylog+37CTj
text:100064EA                push    offset aEsc       ; "[Esc]"
text:100064EF                jmp     loc_10006583
text:100064F4                ; -----
text:100064F4 loc_100064F4:                ; CODE XREF: Keylog+377Tj
text:100064F4                push    offset aCtrl      ; "[Ctrl]"
text:100064F9                jmp     loc_10006583
text:100064FE                ; -----
text:100064FE loc_100064FE:                ; CODE XREF: Keylog+372Tj
text:100064FE                lea     eax, [ebp+Time+4]
text:10006501                push    offset aEnter     ; "<Enter>\r\n"

```

Kết Luận: Do không phải là tài liệu sâu vào phân tích mã nên mục trên em chỉ phân tích sơ qua và đưa ra kết luận là mục trên là một thủ vi phạm khi các hàm có nhúng hành vi là một keylogger.

CHƯƠNG 2. NGHIÊN CỨU VIRUS C S D LI U MÃ C

xây dựng một cơ sở dữ liệu mã để tra cứu trước tiên để biết có khả năng virus phát hiện mã để biết cách nào, các kỹ thuật thu thập mã và phát hiện mã. Chương này sẽ trình bày các dấu hiệu phát hiện mã, chú ý trao đổi mã để tránh cơ sở dữ liệu này.

2.1 Các kỹ thuật dấu hiệu mã và kỹ thuật phát hiện mã

Phương pháp này sẽ dựng một cơ sở dữ liệu có sẵn khi quét các chương trình quét để biết quét các tệp tin và so sánh với cơ sở dữ liệu có sẵn. Cơ sở dữ liệu mã sẽ hỗ trợ phát triển phần mềm diệt virus để tạo ra theo những cách khác nhau để virus là thu thập quét sao cho nhanh và hiệu quả nhất. Chúng ta sẽ xem các hãng giới thiệu và những không thể tìm ra ngoài.

2.1.1 String – Chuỗi

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

Phương pháp này có nhược điểm là đòi hỏi thu thập quét phải tốn nhiều thời gian quét sẽ rất lâu nên việc quét là làm cho cơ sở dữ liệu lớn. Mặt khác nhược điểm là vì tìm chuỗi nên độ nhạy của việc tìm mã sẽ hỗ trợ 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 vì các biến thể khác nhau của cùng một loại mã thì chuỗi nên độ nhạy của việc tìm mã sẽ hỗ trợ phát triển phần mềm diệt virus là khác nhau nên quá trình trao đổi dữ liệu giữa các hãng là không có.

Đây là một chuỗi nên độ nhạy của virus Stoned:

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

Chuỗi này là chuỗi nên độ nhạy 16 byte thì sẽ sử dụng virus 16 bit. Vì virus 32 bit thì sẽ sử dụng chuỗi 32 byte.

Em sẽ tiến hành xây dựng một chương trình demo quét mã theo một cách sơ đẳng là chu trình nhúng có sẵn.

2.1.2 Kỹ thuật nhúng

Kỹ thuật nhúng là kỹ thuật chèn chuỗi nhúng tùy nhiên khi các chương trình phát hiện mã độc đang cố gắng quét thì nó sẽ phép bỏ qua một byte hoặc một dãy byte.

Ví dụ :

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

Bảng duy trì:

- Nếu khi gặp vị 04 thì tiếp tục
- Khi gặp vị 00 thì tiếp tục
- Tiếp tục cho đến khi gặp ?? thì bỏ qua byte này
- Khi gặp vị 02 thì tiếp tục
- Gặp %3 33 thì nghĩa là tìm khi gặp 33 vị 3 vị trí sau nếu khi gặp thì tiếp tục
- Duy trì một chuỗi nếu khi gặp thì gộp lại

Kỹ thuật nhúng là kỹ thuật chèn vào các byte bít một cho phép quét một cách chính xác hơn. Kỹ thuật này mã hóa, virus sẽ hình thức nếu có thể đăng bị phát hiện bởi phần mềm pháp này.

2.1.2 Mã bít

Mã bít là thuật toán chung cho các thuật toán tìm kiếm các thuật toán. Nó có thể được thực hiện trên các byte ưu tiên 16 bit và 32 bit của chuỗi quét.

Kỹ thuật này cho phép thêm byte chèn ký thuật nhúng. Các nhà nghiên cứu mã độc có thể kiểm soát mã bít để thay đổi cách chuyển đổi byte bít của chuỗi chèn nó.

Vì vậy mã bít có thể là bít toàn bộ tập tin hoặc một phần tập tin hay phần của tập tin. Mã bít thường chèn vào code thì các chúng vào hoặc vào cuối tập tin nên vì vậy mã bít phần sẽ giúp giảm thiểu gian lận và các sơ đẳng là bít có hiệu suất cao hơn.

Như vậy mã bít là kỹ thuật chèn vào các mã độc có khả năng bị nhúng hay xáo trộn mã thì sẽ gặp khó khăn vì có thể cùng một loại mã độc nhưng sẽ có nhiều mã bít khác nhau.

Đây là một đoạn code lấy mã băm là MD5 với bằng C#:

```
private void ToMD5(string FilePath)
{
    //tạo một đối tượng mã hóa MD5
    MD5 MyMD5 = MD5.Create();

    // mở 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();
}
```

2.1.3 Khung mã có sẵn

Phương pháp này do tác giả Eugene Kaspersky. Phương pháp khung mã có sẵn rất hữu ích trong việc phát hiện virus macro. Thay vì lựa chọn một chuỗi ký tự nào đó để kiểm tra các virus macro, chương trình quét phân tích các dòng báo cáo về một số các thông báo không cần thiết. Kết quả là xác định các mã virus trong virus macro sau đó so sánh với cơ sở dữ liệu có sẵn.

Phương pháp này dùng để phát hiện và loại bỏ các biến thể của virus.

2.1.4 Phương pháp dựa trên hành vi

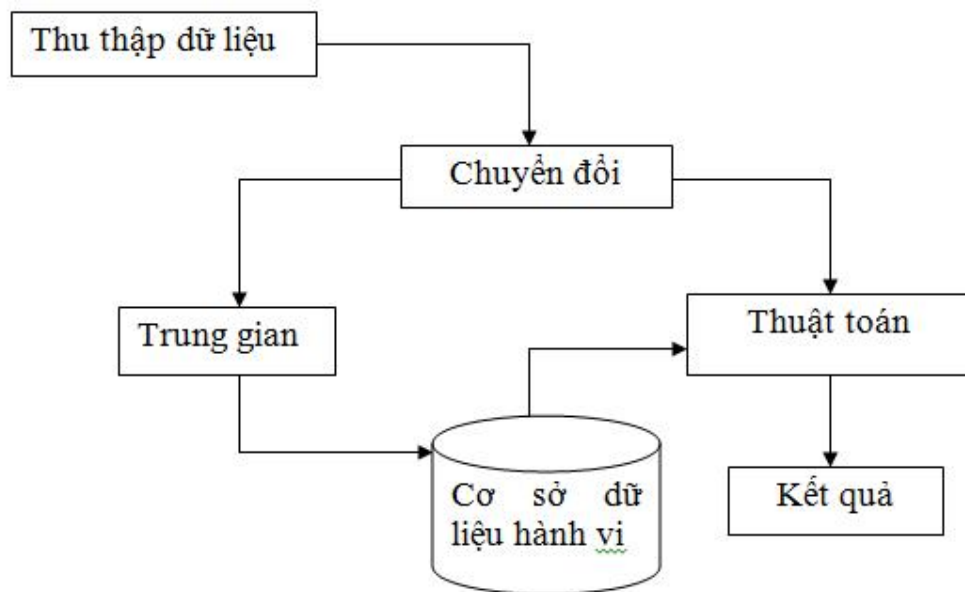
Phương pháp phát hiện dựa trên hành vi khác với việc phát hiện dựa vào bên ngoài, cấu trúc tệp tin có sẵn là nó xác định các hành vi thực hiện của mã

hành vi là việc xác định cấu trúc phân cấp của chương trình. Các chương trình không ghi nhận cú pháp hay cấu trúc nên có hành vi ghi nhận hành vi đã xác định trước là đã xác định nó là mã độc hay không.

Cách này giúp cho việc xác định mã độc một cách hiệu quả vì với các loại mã độc không ngừng tạo ra các biến đổi mới của nó. Phương pháp phát hiện hành vi luôn theo dõi các biến đổi 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 hệ thống và tính các ghi lại.
- Chuyển đổi: Thành phần này sẽ chuyển các thông tin thu thập được về module thu thập dữ liệu vào nơi trung gian lưu vào cơ sở dữ liệu.
- Thuật toán so sánh: Cơ sở dữ liệu so sánh các phần tử với nhau để ký hành vi.



Hình 2.1 Mô hình chương trình quét hành vi

2.1.5 Kỹ thuật

Kỹ thuật sử dụng trong các chương trình quét sau này. Các mã độc lây nhiễm thường là tập hợp của các loại mã độc khác nhau. Do đó các chương trình quét có thể tiến hành quét một cách rất nhanh sử dụng kỹ thuật này. Ví dụ như ký các mã độc virus bằng mã độc khác, trong hệ điều hành DOS thì ký tập tin exe có mã độc khác. Vì vậy mã độc được thêm vào chuỗi mã độc

xác định xem có chu kỳ ký tự này trong chuỗi ký tự quét hay không. Do đó nó làm gì mà tìm kiếm chuỗi ký tự này trong chuỗi ký tự quét.

Chuỗi ký tự quét phải được chuyển vào bộ nhớ. Bộ nhớ có thể là một chuỗi ký tự, một mảng ký tự, một chuỗi ký tự, một mảng ký tự... Tuy nhiên có một số mã để tìm kiếm một chuỗi ký tự trong một chuỗi ký tự là không thể tìm kiếm được vì nó là các virus mã nguồn mở hoặc mã nguồn đóng. Nhưng lỗi này có thể phát hiện ra những vấn đề về kỹ thuật phát hiện mã độc.

2.1.6 Phát hiện bằng việc ghi mã độc

Kỹ thuật này sử dụng việc ghi mã độc trong một mã độc để sử dụng tất cả các cách trong toàn bộ phần mã độc của nó. Như vậy việc quét phải được chuyển vào kích thước của phần mã độc để quét. Kỹ thuật này cũng rất nhanh nhưng lại có nhược điểm là nó có thể gây ra các thông báo sai và không thể biết được mã độc khi chỉ có mã độc mà không có việc ghi mã.

2.1.7 Mã ghi lại

Là kỹ thuật rất đơn giản trong việc phát hiện mã độc. Một máy ảo có mô phỏng các hệ thống CPU và bộ nhớ để thực hiện các thao tác mã. Do đó mã độc khi chạy trong các máy ảo quét và không có mã độc của mã độc thực tế nào thì sẽ bị phát hiện.

Công cụ mã ghi lại:

- IDA pro là bộ phân tích dung lượng disassembler và debug có thể phân tích các tính năng. Chuỗi ký tự này sẽ nêu chi tiết phần **“3.2 Các phương pháp và công cụ thu thập và tổ chức các dữ liệu trên thị trường”**.
- Ollydebug là chuỗi ký tự debug đơn giản giúp cho ta có thể trace từng dòng code, theo dõi từng hành vi của mã độc. Chi tiết về Ollydebug có thể nêu phần **“3.2 Các phương pháp và công cụ thu thập và tổ chức các dữ liệu trên thị trường”**

Phương pháp ưu tiên của mã ghi lại là sử dụng trình debug để theo dõi các mã sử dụng bộ xử lý. Tuy nhiên việc ghi lại này không an toàn do mã độc có thể nhúng vào ngoài môi trường phân tích vì 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 các 16 bit nhúng lại 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;
```

2.2 Nghiên cứu chuẩn trao đổi dữ liệu mã độc

Hiện nay do việc tìm và diệt mã độc càng ngày càng trở nên phức tạp thì việc không chỉ dùng mà còn việc an ninh quốc gia nên việc chia sẻ dữ liệu mã độc giữa các hãng hay các tổ chức, quốc gia là cần thiết. Chính vì vậy mà chuẩn trao đổi dữ liệu mã độc ra đời.

Chuẩn trao đổi dữ liệu mã độc do ICSG Malware Working Group. ICSG là viết tắt của Industry Connection Security 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 gì? quy tắc các vấn đề về phần mềm độc hại mà ngành công nghệ thông tin hiện nay phải đối mặt.

Trên tầm bản đồ có mục đích là thành lập cách thông minh hơn chia sẻ các mẫu mã độc hại và các thông tin liên quan tới chúng trong ngành bảo mật máy tính để hiểu rõ hơn.

Các nhóm làm việc gì quy tắc các vấn đề và việc đóng gói mã độc.

- Các tài liệu phục vụ tốt nhất trong việc sử dụng công cụ gói bộ các nhà phát triển phần mềm.
- Xác định các thuộc tính của việc đóng gói tập trung vào các tính năng mà mã độc thường sử dụng.
- Tóm tắt khóa ngữ ký của việc đóng gói và một tập hợp các tên các việc đóng gói.
- Thiết lập một nền tảng chia sẻ dữ liệu chia sẻ thông tin đóng gói.
- Phát triển và thực hiện “Hệ thống Taggant” – nhúng mã bìa bên trong vào từng gói đóng gói phát hiện ra nguồn các tập tin đóng gói. Taggant là một dấu hiệu bổ sung trong quá trình tạo ra sản phẩm. Một khi thấy được taggant sẽ thực hiện các nhóm sẽ thúc đẩy

và tạo ra các ứng dụng giám sát virus và các trình khai thác virus bên phát triển Anti Virus và bên cung cấp mẫu virus mới.

tham gia nhóm Malware Working Group thì các thành viên cá nhân cũng trở thành thành viên của ICSG. Chỉ có các thành viên ICSG mới có quyền tham gia trao đổi và các ý tưởng gì pháp trong virus và ra chu trình. Tuy nhiên cũng có những chuyên gia cũng minh chứng họ tham gia virus phát triển mà không có quyền quyết định.

Dưới đây là dữ liệu của chu trình mã:

```
<malwareMetaData xmlns="http://xml/metadataSharing.xsd" xmlns:xsi="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>e3b0c44298fc1c149afbf4c8996fb92427ae41e4649b934ca495991b7852b855</sha256>
      <sha512>cf83e1357eefb8bdf1542850d66d8007d620e4050b5715dc83f4a921d36ce9ce47d0d13c5d85f2b0ff8318d2877eec2f63b931bd47417a81a538327af927da3e</sha512>
      <size>1546790</size>
      <filename>ProcessExplorer.zip</filename>
      <MIMEType>application/zip</MIMEType>
    </file>
    <file id="d22ff2cc70fa2eec94aaa6c6f49e6eb0">
      <md5>d22ff2cc70fa2eec94aaa6c6f49e6eb0</md5>
      <sha1>2458a3d696698e2c4550b91e54ff63f4b964198d</sha1>
      <sha256>6ff22c87fb5ee105b33346dbb3f13f3049a292981e9df1eb8591e858ccf4d782</sha256>
      <sha512>34e18bf9679c71189383bcd89c9f723383715bbf63f051edd659c57e14d012987c33ba67fbbb0faeca962b3ec7b12b0aa24b3c134ddbb9f905aa26604718f375</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 dữ liệu định file XML. XML viết tắt của Extensible Markup Language (ngôn ngữ đánh dấu mở rộng) là một ngôn ngữ đánh dấu định nghĩa riêng cho người sử dụng. Nó có thể biểu diễn minh bạch toàn bộ những khía cạnh của hệ thống như cấu trúc 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 liệt kê các thuộc tính con khác nhau như: name, md5, sha1, sha256 ... Vì thế việc và kiểm tra nên chọn trình quản lý dữ liệu của em sẽ thể hiện vì mỗi các thuộc tính trên mà sẽ không dùng hết các thuộc tính. Em sẽ trình bày chi tiết hơn khi báo cáo.

2.3 Nghiên cứu sản phẩm mã của Clam AV để xây dựng sản phẩm mã

2.3.1 Clam Anti Virus

Clam Anti Virus - viết tắt là ClamAV - là một phần mềm diệt virus mã nguồn mở có thể kiểm tra phát hiện các loại mã độc như: trojan, virus,... và các mối đe dọa khác. Nó có chức năng quét theo thời gian. Nó cung cấp một hệ thống quét tự động cao, tiện ích dòng lệnh cho việc quét file theo yêu cầu và một công cụ thông minh để phân tích ký hiệu. Có lỗi của ClamAV là cung cấp nhiều cách nhận định phần mềm phát hiện, hỗ trợ ghi nhận phần mềm, hỗ trợ lưu trữ và có nhiều định dạng ký hiệu phát hiện các mối đe dọa. Có lỗi là vì của ClamAV có sản phẩm trong Immunity 3.0, có hỗ trợ ClamAV. Nó xử lý nhanh, yêu cầu tính năng là gì để pháp cho máy tính sản phẩm Windows.

2.3.2 ClamAV Virus Databases

ClamAV Virus Databases - viết tắt là CVD – là liệt kê các ký hiệu bao gồm các sản phẩm ký hiệu trong các định dạng và khác nhau. Các tiêu chí của các liệt kê này là chuỗi ký hiệu có độ dài 512 byte và 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 hi n th thông tin chi ti t t p 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

*JVh4vYmW8mZ62ZHYMlM903TMZFg5hZlxcjQB3SX0TapdF1SFNzoWjsyH53eXv
MDY*

eaPVNe2ccXLfEegoda4xU2TezbGfbSEGoU1qolyQYLX674sNA2Ni6l6/CEKYYh

Verification OK.

Project ClamAv phân ph i d i các t p tin c a file CVD g m: main.cdv và daily.cvd

2.3.3 Debug thông tin t libclamav

t o ra ch ký có hi u qu cho clamAV, i u quan tr ng nh t là làm th nào hi u c công c x lý t p tin u vào. Cách t t nh t xem nó ho t d ng c hay không là có m t cái nhìn v các thông tin debug libclamav. Chúng ta có th làm vi c này b ng cách g i clamscan v i tùy ch n –debug và –leavee-temps. Tùy ch n u tiên làm cho clamscan hi n th các thông tin t libclamav và tùy ch n th hai giúp tránh vi c xóa t p tin t m th i có th phân tích ti p. D i ây là thông tin hi n th c a m t 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 c ghi nh n m t c a s th c 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)*

C u trúc các section hi n th trên cho th y nó c óng gói b ng 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-90d2d25c9dca42bae6fa9a764a4bcde

*LibClamAV debug: ***** Scanning decompressed file ******

LibClamAV debug: Recognized MS-EXE/DLL file

LibClamAV debug: Matched signature for file type PE

Libclamav ghi d li u óng gói d ng UPX và l u mã gi i nén t i th c thi t i
/tmp/clamav-90d2d25c9dca42bae6fa9a764a4bcde. Sau ó nó ti p t c quét t p tin
m i 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 tệp tin nào có thể tạo ra tệp libclamav. Vì vậy tệp tin được ký cho các tệp tin gì thì nên ta có thể hiểu rõ hơn mà các công cụ có thể phát hiện ra được liệu khi nó có nên bị các bên đóng gói khác.

Phương pháp này nên áp dụng cho tất cả các tệp tin muốn tạo tệp tin. Bằng cách debug các thông tin ta có thể nhanh chóng xem các công cụ ghi nhận và thể hiện được liệu và những tệp tin khác có thể tạo ra. Có thể tạo ra cho các tệp tin tệp tin thì có thể dễ dàng chung chung hơn và vì có thể phát hiện mã có thể gặp nhau trong vài các hình thức khác nhau.

2.3.4.1 Kiểm tra ClamAV

2.3.4.1.1 MD5

Cách để kiểm tra tệp tin cho ClamAV là sử dụng mã checksum MD5, tuy nhiên phương pháp này chỉ có thể sử dụng cho mã có thể. 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 để kiểm tra:

```
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)
```

2.3.4.2 Tạo MD5 cho mã tệp PE file

ClamAv hỗ trợ tính toán ký MD5 cho mã tệp PE file. Chức năng này được lưu trữ trong file mdb với nội dung sau:

PESectionSize:MD5:MalwareName

Cách dễ dàng nhất để tính toán ký MD5 cho mã tệp PE file là trích xuất phần đầu ra mã tệp riêng biệt và sau đó copy vào tùy chọn -mdb của ClamAV.

2.3.4.3 Chuyển ký mã trên phần thân chương trình

ClamAV lưu trữ tất cả các ký mã trên phần thân của file trong mã nhị phân dạng hex (hexa). Mã ký mã hexa nghĩa là phần thân của file được chuyển sang dạng chuỗi hex và có thể mã rỗng khi sử dụng các ký mã khác nhau.

2.3.4.3.1 Nội dung hexa

Sử dụng công cụ sigtool --hex-dump để chuyển dữ liệu nhị phân sang dạng chuỗi hexa

```
zolw@localhost:/tmp/test$ sigtool --hex-dump
```

How do I look in hex?

```
486f7720646f2049206c6f666b20696e206865783f0a
```

2.3.4.3.2 Ký mã nhị phân

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

- ?? : Phù hợp với bất kỳ byte nào
- a? : Phù hợp với mã nibble cao (bít cao)
- ?a : phù hợp với nibble thấp (bít thấp)
- * : Phù hợp với bất kỳ số byte
- {} : Phù hợp với n byte
- {-n} : phù hợp với n hoặc nhiều byte
- {n-} : phù hợp với l hoặc nhiều 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 gán vào mã ký mã hex
- (B): phù hợp với t (bao gồm mã tin nhắn)

- (L): phù hợp với CR, CRLF hoặc tập tin ghi h n

Ch ký ph m vi * và {} h u nh tách m t hex thành hai ph n.

2.3.4.3.3 nh d ng ch ký c b n

Ví d v nh d ng ch ký :

MalwareName=HexSignature

ClamAv s quét t t c t p tin tìm ch ký hex. T t c ch ký ki u này c l u d i d ng file .db.

2.3.4.3.4 nh d ng ch ký m r ng

nh d ng ch ký m r ng cho phép b sung thông tin c i m k thu t nh m t lo i t p tin m c tiêu, offset c a virus hoặc phiên b n công c , giúp phát hi n t t h n. nh d ng nh sau:

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

TargetType có giá tr là m t trong các s sau i di n cho ki u file c th :

- 0: b t k file nào
- 1: file th c thi di ng i v i c 32 bit và 64 bit
- 2: file n m trong OLE2: file nh, VBscript ...
- 3: HTML
- 4: T p tin mail
- 5: h a
- 6: ELF
- 7: file text mã ASCII
- 8: không s d ng
- 9: File Mach – O

Offset là m t d u hoặc s th p phân n có th k t h p v i v i c thay i c bi t:

- *: b t k
- n: offset tuy t i
- EOF – n: k t thúc file tr i n byte

Ch ký cho PE, ELF, Mach-O file h tr :

- EP + n: entry point c ng n byte

- EP – n: entry point tr n byte
- Sx + n: b t u t section x c a d li u c ng n byte
- Sx – n: b t u t section x c a d li u tr n byte
- SL + n: k t thúc c a section c a d li u c ng n byte
- SL – n: k t thúc c a section c a d li u tr n byte

2.3.4.4 Ch ký bi u t ng cho PE file

ClamAV 0.96 có m t kho ng bi u t ng m giúp phát hi n các t p tin th c thi mã c ã ng y trang b ng cách tìm ki m các t p tin hình nh, office, v n b n, hay t ng t th .

Bi u t ng phù h p ch c thông qua ch ký .ldb b ng cách s d ng thu c tính c bi t th IconGroup1 ho c IconGroup2. nh danh hai nhóm các bi u t ng c l u trong t p tin c s d li u .ldb. nh d ng c a t p tin .ldb là :

ICONNAME:GROUP1:GROUP2:ICON_HASH

- ICON_NAME là m t chu i nh danh duy nh t cho bi u t ng c th
- Group1 là m t chu i t nh danh nhóm u tiên c a các bi u t ng (Icongroup1)
- GROUP2 là m t chu i nh danh nhóm th hai c a các bi u t ng (IconGroup2)
- ICO_HASH là m t mã b m mò c a hình nh bi u t ng

2.3.4.5 Ch ký cho các siêu d li u thông tin phiên b n trong PE file

T phiên b n CLamAV 0.96, nó ã d dàng k t h p m t s thông tin xây d ng vào các PE file (g m file th c thi và th vi n lien k t ng). B t k khi nào tra c u thu c tính c a m t t p tin th c thi PE trong windows, chúng ta s th y m t lo t các chỉ ti t ó.

Nh ng thông tin c l u tr trong m t khu v c c bi t c a ngu n t p tin kèm theo tên c a VS_VERSION_INFOMATION (ho c versioninfo). Nó c chia làm hai ph n. Ph n u tiên là m t lo t cá s và c cho ta th y phiên b n t p tin. Lúc u nó c s d ng v i các trình cài t sau khi phân tích nó s có th xác nh xem vi c th c thi ho c th vi n nh t nh s c nâng c p, ghi è ho c phát tri n lên. Ph ng pháp này không bao gi th c s c s d ng.

Ph n th hai là m t danh sách n gi n c a khóa, chu i giá tr .Nó dành cho thông tin ng i dùng và hoàn toàn b qua b i h di u hành. Ví d khi nhìn vào ping.exe ta s th y công ty là “Microsoft Corpo-ration”, ph n mô t là “TCP/IP

Ping command”, tên n i b là “ping.exe”.... Tùy thu c vào phiên b n h i u hành, m t s khóa s có th hi n th thông tin c bi t trong h p tho i thu c tính, tuy tên n i b là nh nhau.

phù h p v i m t c p khóa/giá tr thông tin phiên b n các offset c bi t c a file g n VI ã c a ra. i u này t ng t nh cá ki u g n khác ngoài tr vì c thay i phù h p v i m u hex l p l i c a m t offset riêng bi t, nó s ki m tra chính nó và m i c p khóa/giá tr trong t p tin. Th VI không c n và c ng không ch p nh n m t +/- offset gi ng nh các ví d EP +1. i v i các ch ký hex c a chính nó, nó ch là UTF 16 k t xu t c a khóa và giá tr . Ch ký t i di n ?? và (aa||bb) c cho phép trong ch ký. Thông th ng chúng ta không c n ph i b n tâm tìm ra áp án: b i vì m i c p khóa/giá tr cung v i ch ký VI d a trên t ng ng c in b ng clamscan khi dùng tùy ch n –debug.

Ví d : *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

069006f006e0000000

VersionInfo (d32a): 'FileDescription'='Entertainment Pack

FreeCell Game' - VI:460069006c006500440065007300630072006900700

0740069006f006e0000000000045006e007400650072007400610069006e006d

0065006e00740020005000610063006b0020004600720065006500430065006

c006c002000470061006d00650000000

VersionInfo (d396): 'FileVersion'='5.1.2600.0 (xpclient.010817

-1148)' - VI:460069006c006500560065007200730069006f006e00000000
 0035002e0031002e0032003600300030002e003000200028007800700063006
 c00690065006e0074002e003000310030003800310037002d00310031003400
 380029000000

VersionInfo (d3fa): 'InternalName'='freecell' - VI:49006e007400
 650072006e0061006c004e0061006d006500000066007200650065006300650
 06c006c0000000

VersionInfo (d4ba): 'OriginalFilename'='freecell' - VI:4f007200
 6900670069006e0061006c00460069006c0065006e0061006d0065000000660
 0720065006500630065006c006c0000000

VersionInfo (d4f6): 'ProductName'='Sistema operativo Microsoft
 Windows' - VI:500072006f0064007500630074004e0061006d006500000000
 000530069007300740065006d00610020006f00700065007200610074006900
 76006f0020004d006900630072006f0073006f0066007400ae0020005700690
 06e0064006f0077007300ae0000000

VersionInfo (d562): 'ProductVersion'='5.1.2600.0' - VI:50007200
 6f006400750063007400560065007200730069006f006e000000035002e00310
 02e0032003600300030002e0030000000

[...]

M c dù ch ký VI trên c s d ng trong ch ký logic nh ng ta v n có
 th th dung chúng bình th ng trong file .ndb:

my_test_vi_sig: I:VI:paste_your_hex_sig_here

N u mu n gi i mã m t ch ký VI c b n ta dùng :

echo hex_string | xxd -r -p | strings -el

2.3.4.6 Ch ký d a trên siêu d li u

ClamAV 0.96 cho phép t o ch ký chung phù h p v i các t p tin c l u tr trong các ki u ch a khác nhau áp ng các i u ki n c th . nh d ng nh sau:

VirusName:ContainerType:ContainerSize:FileNameREGEX:

FileSizeInContainer:FileSizeReal:IsEncrypted:FilePos:

Res1:Res2[:MinFL[:MaxFL]]

- VirusName: Tên c a virus hi n th khi kh p ch ký
- ContainerType g m các nh d ng 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) ho c *
- ContainerSize: Kích th c n i ch a t p tin
- FileNameREGEX: Mô t tên file ích
- FileSizeInContainer: th ng c nén kích th c,
- FileSizeReal: th ng không n kích th c, MAIL, TAR và CPIO thì ng c l i
- IsEncrypted: 1n u file mã hóa, 0, n u không mã hoá, * b qua
- FilePos: V trí file trong n i ch a
- Res1: khi ContainerType là CL_TYPE_ZIP or CL_TYPE_RAR
- Res2: không s d ng v i ClamAV0.96
- Ch ký c l u tr trong file .cdb.

2.3.4.7 Ch ký trên siêu d li u ZIP/RAR

Nó ch c s d ng l u tr ch ký i v i các nh d ng Zip ho c Rar và có nh d ng:

virname:encrypted:filename:normal size:csize:crc32:cmethod:

fileno:max depth

- Virname: Tên virus
- Encrypted: c mã hóa, 1- mã hóa , 0 – không mã hóa
- Filename: Tên file
- Normal size : kích th c file không nén (* b qua)
- Compressed size: kích th c file khi nén(* b qua)
- CRC32(* b qua)
- Cmethod: ph ng th c nén (* b qua)

- Fileno: vị trí file trong kho lưu trữ (* b qua)
- Max depth: Số tầng tiếp diễn lưu trữ chồng nhau (* b qua)

Các số liệu này sẽ xuất diễn ra file .zmd hoặc .rmd

2.3.4.8 Danh sách tệp

Danh sách tệp là một tệp tin chứa các thông tin về ký MD5 và vị trí trong một tệp tin các số liệu có thể diễn ra .fb

Danh sách tệp chứa các thông tin về các số liệu và các tên của nó vào một tệp tin các thông tin là local.ign2 của lưu trữ trong thư mục các số liệu. Các tệp tin có thể hiển thị theo tên của ký MD5 của phần mềm phần mềm các số liệu cho các tệp tin. Ví dụ :

Eicar-Test-Signature:bc356bae4c42f19a3de16e333ba3569c

2.3.4.9 Danh sách các tệp theo tên

ClamAv sẽ diễn ra các tệp tin để lấy làm tên của tệp tin:

- Worm: Sâu mìn
- Trojan: Chạy trình của họ
- Adware: adware
- Flooder: flooder
- HTML: tệp tin HTML
- Email: Thư điện tử
- IRC: trojan IRC
- JS: Mã của Java Script
- PHP: Mã của PHP
- ASP: Mã của ASP
- VBS: Mã của VBS
- BAT: Mã của BAT
- W97M, W2000M: virus macro trên word
- X97M, X2000M: virus macro trên excel
- O97M, O2000M: virus macro chung trên office
- DoS: phân mảnh mạng công nghệ để chặn
- DOS: mã của DOS trên máy
- Exploit: lờn ng b o m t ph bi n
- VirTool: Công cụ để virus
- Dialer: dialer
- Joke: l a o

Quy tắc đặt tên của ClamAV:

- Luôn sử dụng mật khẩu zippwd vào tên mã để chỉ cho loại ZMD
- Luôn sử dụng mật khẩu rarpwd vào tên mã để chỉ cho loại RMD
- Chỉ sử dụng các ký tự chữ và số, dùng chèn ngang “-“, dùng chấm “.”, dùng chèn dưới “_” trong tên mã để không bao gồm sử dụng dấu, kép, khoảng trống.

2.3.4.10 File mã bí mật

2.3.4.10.1 HTML

ClamAV có chứa mã HTML mã bí mật bình thường, giúp phát hiện lỗi HTML. Chỉ cần sigtool -html chuyển hóa trên mã tệp tin HTML sẽ tạo ra các tệp tin dưới đây:

- Nocmcomment.html: tệp tin bình thường, vị trí ghi chép phần nội dung các nhận xét và khoảng trống bị xóa
- Notags.html: ghi chép lại trên những các thẻ HTML bị xóa

Cần tạo ra mã để ký hiệu vị trí các tệp tin để tạo ra. Loại bị khả năng nhận báo ghi các mã để tìm kiếm mã để giá trị là 3.

2.3.4.10.2 Tệp tin văn bản

Cần ghi chép lại HTML các tệp tin văn bản mã ASCII để bình thường hóa tất cả các khoảng trống và kiểm soát các ký tự bị xóa trừ khi quét

Câu lệnh: clamscan - leave-temp có mã tệp tin bình thường và sau đó tạo ra chỉ ký hiệu vị trí có ký hiệu là 7

2.3.4.10.3 Tệp tin thực thi để nén

Nếu tệp tin để nén bằng UPX, FSG, Petite hoặc mã trình biên soạn gói PE file khác để chứa mã libclamav, cần chỉ cần clamscan với tùy chọn --debug --leave-temps. Ví dụ xuất ra tệp 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

Ti p theo t o m t ch ký t :

/tmp/clamav-f592b20f9329ac1c91f0e12137bcce6c

CHƯƠNG 3. XÂY DỰNG CẤU TRÚC DỮ LIỆU MÃ C

Chương cụ thể sẽ hướng dẫn xây dựng hai chương trình nhúng và quản lý cấu trúc dữ liệu mã C trong chương này. Ưu tiên là chương trình nhúng theo chuỗi sau đó sẽ là chương trình quản lý cấu trúc dữ liệu theo chuỗi và sẽ được tổng kết.

3.1 Xây dựng chương trình quản lý cấu trúc dữ liệu mã C theo chuỗi

Xây dựng một phần mềm quản lý cấu trúc dữ liệu mã C có cấu trúc dữ liệu 1 bộ dữ liệu chuỗi. Vì vậy nó giúp loại bỏ công việc chuyển đổi cấu trúc dữ liệu bình thường sang dạng chuỗi.

Chương trình sẽ viết bằng ngôn ngữ C# dựa trên nền tảng .Net với Net Framework 4.0 và biên dịch bằng Visual studio 2010.

Ưu tiên là xây dựng cấu trúc dữ liệu theo chuỗi

File cấu trúc dữ liệu 1 bộ dữ liệu nhúng XML có tên là “malware.xml” và kết cùng thêm các chương trình quản lý

```

1 <malware version="1.0">
2   <company>AV/AV/company</company>
3   <author>KhangKiller/author</author>
4   <comment>Malware Data v1.0 file</comment>
5   <timestamp>19/05/2018 8:28:39 CH</timestamp>
6   <id>0</id>
7   <file ID="1">
8     <md5>4e0b0d3a427b66d8390aa7623890e90</md5>
9   </file>
10  <sha1>9cb1bd5dc50124f526e1003b1b3f37cc0224a77ec</sha1>
11  <sha256>
12    e942d28e0e83918384732731711430b33f13713816663a17637a2b47f7c70d1
13  </sha256>
14  <sha512>
15    9e01107a79e470f433e3b14197f9228836a378e0c0b0b7181798b41830366e3e12f0d0693e93f09397e8138e48d1e67e398f70d1f87137a18872d0f0a304531
16  </sha512>
17  <filename>procomp.exe</filename>
18  <size>5412856</size>
19  <MIMEType>application/octet-stream</MIMEType>
20  <MalwareName>NULL</MalwareName>
21  <Sign>NULL</Sign>
22  </file>
23  <file ID="2">
24    <md5>50c7ae4f7e332c841ee2e0113afa05f0f</md5>
25  </file>
26  <sha1>20daa09d1918bba019769d0f7ebab01c4768e11</sha1>
27  <sha256>
28    c3b0c44290f0c1c149afbf4c08596fb92427ae41e4649b334ca495591b7052b055
29  </sha256>
30  <sha512>
31    c5f03e1357ee0b0df1542e50c66c0007d620e4050b5715dc03f4e321d030e30e47d0d13c5d05f2b0ff0310d2077ee02f62b901bd47417e01e530327af927da0e
32  </sha512>
33  <filename>ProcessExplorer.zip</filename>
34  <size>1546790</size>
35  <MIMEType>application/zip</MIMEType>
36  <MalwareName>NULL</MalwareName>
37  <Sign>NULL</Sign>
38  </file>

```

Hình 3.1 Cấu trúc file dữ liệu

Cấu trúc dữ liệu gồm các trường chính như sau :

- ID : Số thống nhất ngành sản phẩm mã
- MD5 : Mã băm MD5 trích xuất file
- SHA1: mã SHA1
- SHA1: mã SHA256
- SHA1: mã SHA512
- Filename : Tên file
- Size: Kích thước file
- MINEType : Kiểu định dạng file : exe, rar, mp3 ...
- Malwarename: Tên mã độc định dạng file
- Sign: Chuỗi định dạng file (nếu có)

Các dữ liệu kèm với chương trình có thể được kiểm tra bằng các hãng Antivirus trên thị trường và Việt Nam:

- Mã độc là mã độc mang tên “FunnyIM” do BKAV phát hiện khá lâu có định dạng file .dat, đây là mã độc dạng worm
- Mã độc có tên là “nettui.dll” là mã độc vì nó sử dụng bí mật sử dụng lỗ hổng bom tấn office: CVE-2010-3333 của Microsoft có tên là “Trojan.Win32.Genome.akudf” do Kaspersky phát hiện

Chiến lược của chương trình:

- Nhập các dữ liệu:

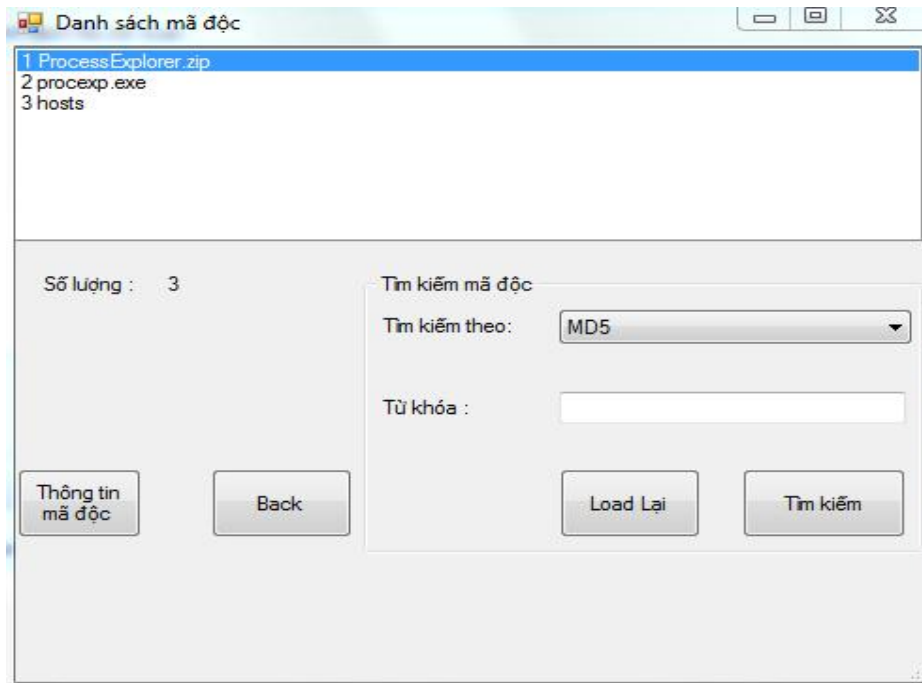
Chiến lược này khi hoạt động sẽ nhập ID sản phẩm vào các dữ liệu thông tin hành lý mã băm MD5, SHA1, SHA 256, SHA 512, tên file, kích thước, kiểu file, tên mã độc và chuỗi định dạng (nếu có). Trong đó tên file có ký tự khung nhúng và tên mã độc do người nhập dữ liệu đưa vào. Tên của chương trình này là module nhập dữ liệu

Hình 3.2 Nhập các dữ liệu

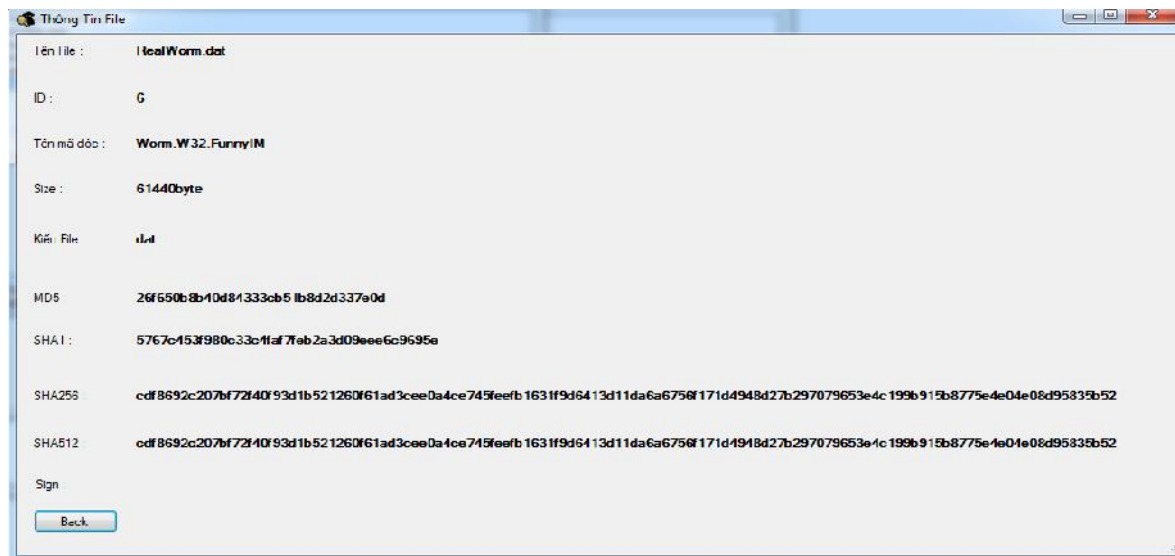
- Danh sách các dữ liệu:

Chức năng này hiển thị tên các mục chứa trong cơ sở dữ liệu dựa trên 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 từng mục mã chứa trong cơ sở dữ liệu.

Module cách chức năng như hình dưới đây:

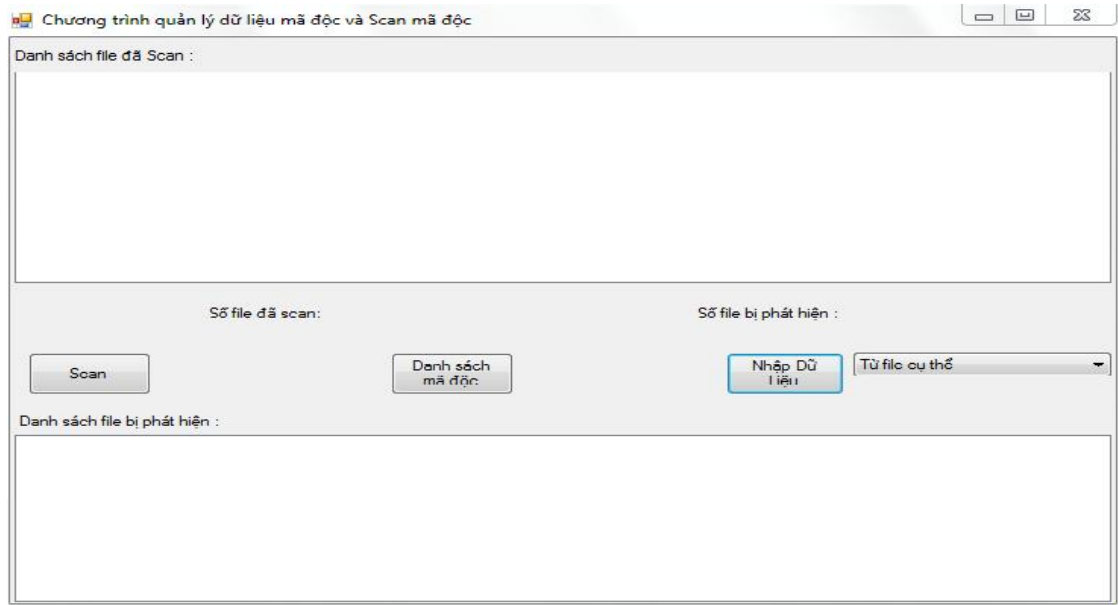


Hình 3.3 Hiển thị



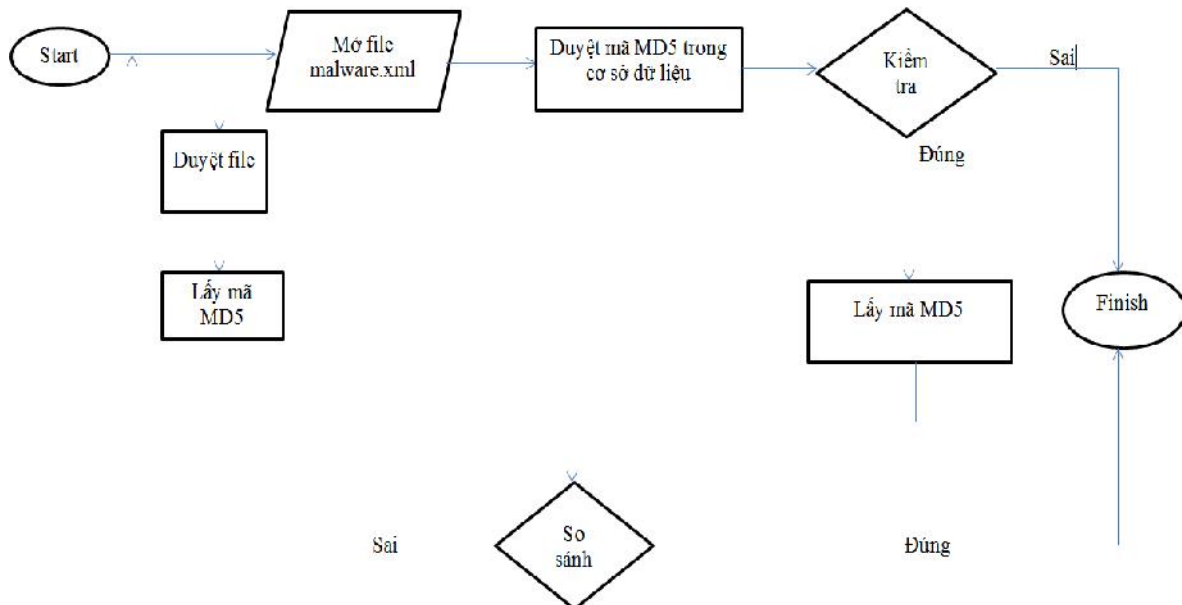
Hình 3.4 Chi tiết mục mã

Chức năng thứ 3 của chương trình là sử dụng cơ sở dữ liệu trên nền mạng mã độc trong máy. Chức năng này cho phép quét các file trong hệ thống có khả năng quét tất cả các file trên máy tính người dùng. 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 trích xuất mã MD5 duy nhất trong mã MD5. Nếu không tìm thấy mã MD5 hoặc trống thì sẽ kết thúc chương trình

Bước 2: Duy trì file trong thư mục sau đó lấy mã MD5 của từng file

Bước 3: So sánh hai mã MD5 lấy được từ dữ liệu và của file duy trì

- Nếu bằng nhau thì thông báo phát hiện, kết thúc chương trình
- Nếu không bằng nhau thì lặp lại bước 2
- Sau khi duy trì hết file trong thư mục thì quay lại bước 1 lặp lại khi kết thúc file dữ liệu và kết thúc chương trình

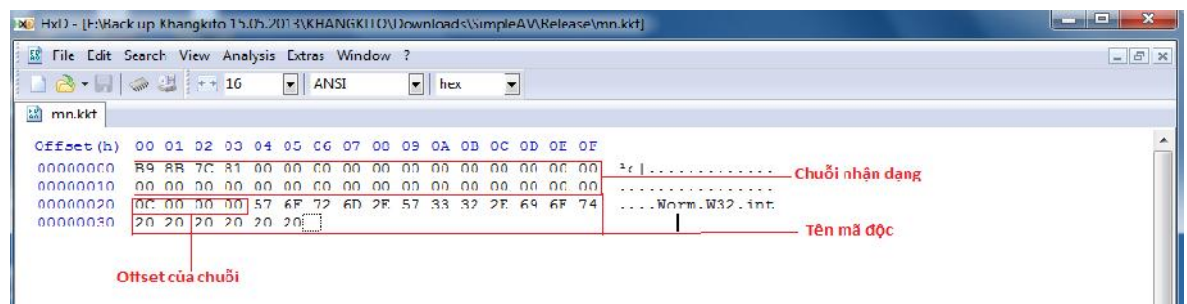
3.2 Xây dựng chương trình nhận dạng mã độc theo chuỗi

Tập tin dữ liệu là “mn.kkt”, cùng với thư mục chứa chương trình quét. Cấu trúc tập tin 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 Tập dữ liệu

Cách xác định chuỗi hình dạng của mã mìn là mìn có tên FunnyIM do BKAV trình bày có cấu trúc như sau:

- Sử dụng tool HxD.exe xác định chuỗi hình dạng như sau:

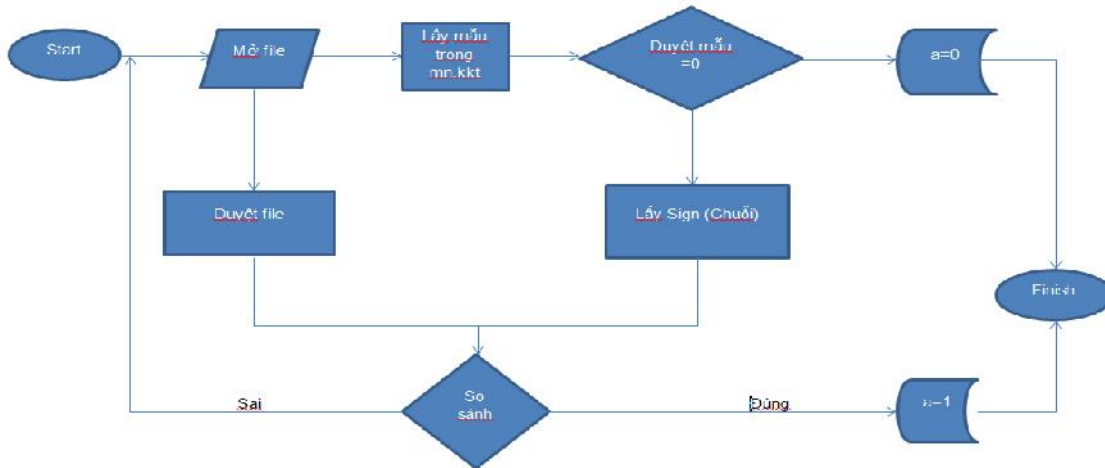
Offset(h)	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	
00003440	0D	00	20	00	00	00	00	00	5C	AB	8E	06	50	34	40	00\«Ž.P4@.
00003450	38	48	40	00	E0	48	40	00	AE	10	40	00	30	49	40	00	8H@.àH@.@.@.0I@.
00003460	A8	10	40	00	7C	48	40	00	C4	48	40	00	A8	4E	40	00	..@. H@.ÀH@.~N@.
00003470	94	49	40	00	D8	49	40	00	20	4A	40	00	B4	10	40	00	"I@.0I@. J@.'@.
00003480	C0	46	40	00	00	00	80	00	43	41	41	50	44	38	37	35	ÀF@...€.CAAPD875
00003490	43	41	54	53	55	4B	38	37	43	41	43	4F	4E	32	41	55	CATSUK87CACON2AU
000034A0	43	41	45	37	4A	45	54	56	43	41	48	53	4B	30	4A	30	CAE7JETVCAHSK0J0
000034B0	9C	63	40	00	98	5E	40	00	F8	5D	40	00	C0	5C	40	00	æc@.~^@.ø]@.À\@.
000034C0	BA	9C	63	40	00	B9	C0	10	40	00	FF	E1	BA	98	5E	40	°æc@.~^@.ýá°~^@
000034D0	00	B9	C0	10	40	00	FF	E1	BA	F8	5D	40	00	B9	C0	10	~^@.ýá°ø]@.~^@.
000034E0	40	00	FF	E1	BA	C0	5C	40	00	B9	C0	10	40	00	FF	E1	@.ýá°À\@.~^@.ýá
000034F0	F0	00	00	00	38	00	00	00	00	00	00	00	00	00	00	00	8...8.....
00003500	D0	00	00	00	D8	00	00	00	00	00	00	00	E0	00	00	00	Đ...Đ.....à...
00003510	FF	FF	FF	FF	00	00	00	00	E0	00	00	00	E1	00	00	00	ÿÿÿÿ....à...á...
00003520	ED	00	00	00	00	00	00	00	38	99	9D	8F	D4	78	15	4B	i.....8™...Ôx.K
00003530	B8	83	EC	6D	BD	5B	97	71	66	96	AD	D3	5F	CD	77	4D	,fìm%[-qf-.Ó ÍwM
00003540	84	9C	91	CE	22	EA	F7	0C	5E	6F	30	90	85	A9	86	41	„æ'í"è÷.^o0...@+A
00003550	9D	CF	27	7D	F4	A4	91	06	3B	A5	BA	95	FD	5D	4D	47	.Í'')ðæ'.;¥°·ý]MG
00003560	AC	EB	C4	24	8C	71	7D	B6	E3	6D	63	8B	30	AC	57	4C	-èÄ\$Qq}qãm<0-WL
00003570	B8	28	E7	27	23	D9	93	2B	FF	01	00	00	F8	E6	00	00	, (ç'#Ü"+ý...øæ..
00003580	00	46	84	F9	CF	07	00	00	59	32	83	85	C9	C3	DB	41	.F„ùí...Y2f...ÉÄÜA
00003590	A5	72	E3	29	06	69	95	E8	98	31	B4	19	52	95	10	4F	¥rã).i·è~1'R·.O
000035A0	B9	0C	66	05	08	28	7B	F8	00	00	00	00	60	0B	57	20	².f..({ø....`W
000035B0	00	00	00	00	00	00	02	00	00	00	00	00	00	00	00	00
000035C0	00	D1	00	00	00	D1	00	00	01	00	00	00	02	00	00	00	.Ñ...Ñ.....
000035D0	00	52	65	61	6C	57	6F	72	6D	2E	44	4C	00	44	4C	00	.RealWorm.DL.DL.
000035E0	F4	01	00	00	D0	40	40	00	00	00	00	00	20	C6	40	00	ó...Đ@@.....Æ@.
000035F0	30	C6	40	00	8C	14	00	00	00	D0	40	00	BA	10	40	00	0E@.E....Đ@.°.@.
00003600	00	00	00	00	2A	00	5C	00	41	00	45	00	3A	00	5C	00*.\.A.E...\.
00003610	56	00	42	00	5C	00	53	00	6F	00	75	00	72	00	63	00	V.B.\.S.o.u.r.c.
00003620	65	00	5C	00	57	00	6F	00	72	00	6D	00	5C	00	53	00	e.\.W.o.r.m.\.S.
00003630	6F	00	75	00	72	00	63	00	65	00	52	00	65	00	61	00	o.u.r.c.e.R.e.a.
00003640	6C	00	57	00	6F	00	72	00	6D	00	5C	00	52	00	65	00	l.W.o.r.m.\.R.e.
00003650	61	00	6C	00	57	00	6F	00	72	00	6D	00	2E	00	76	00	a.l.W.o.r.m...v.
00003660	62	00	70	00	00	00	00	00	00	00	00	00	00	00	00	00	b.p.....
00003670	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00003680	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00003690	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
000036A0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
000036B0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
000036C0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
000036D0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
000036E0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
000036F0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00003700	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00003710	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00

Hình 3.8 Chuỗi String

Đây là chuỗi mã trình của lõi mã mìn này mà không liên quan tới các mã mìn hay tệp tin nào khác.

- Sau khi xác định được chuỗi cần tìm thì ta sử dụng các hàm để kiểm tra chuỗi mã vào các dữ liệu của module để trình bày được đây.

Mô hình trình quan chức trình

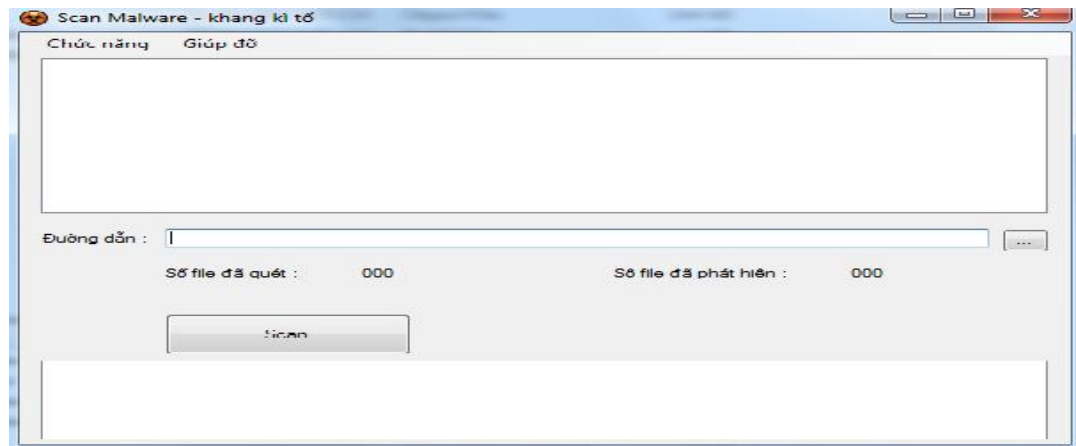


Hình 3.3 Mô hình chức năng trình quét

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



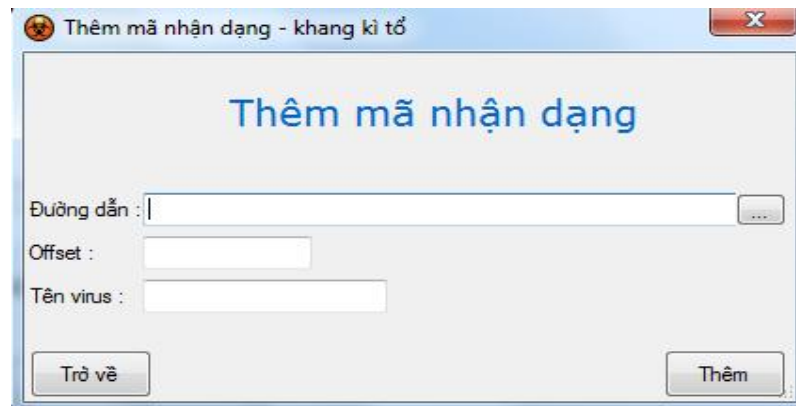
Hình 3.9 Giao diện chính

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



Hình 3.10 Danh sách mã độc

- Module 3: Thêm mã vào tệp dữ liệu



Hình 3.11 Thêm mã

Họ t ng: Chi ti t họ t ng c a ch ng trình em s trình bày khi ti n hành b o v ấn.

Code ch ng trình c trình bày ph n “M C L C”

K T L U N

Sau m t th i gian t p trung nghiên c u, do còn h n ch v ki n th c nên án còn nhi u h n ch .K t qu t c là án này ã th c hi n c m t s n i dung sau:

V ph n lý thuy t:

- Tìm hi u t ng quan v mã c h i, tình hình phát tri n mã c h i trên th gi i nh ng n m g n ây.
- Tìm hi u c các cách th c phát hi n mã c c b n
- Tìm hi u v cách th c thu th p m u mã c m i.
- Tìm hi u và xây d ng c s d li u mã c theo chu n chung

V ph n th c nghi m:

- Xây d ng m t ch ng trình quét mã c theo chu i nh n d ng
- Xây d ng ch ng trình qu n lý c s d li u mã c.

H n ch :

Ch a xây d ng c c s d li u m t cách y các thành ph n, c s d li u xây d ng trên y u t ch quan, c ch s d ng c s d li u ch a hoàn thi n, th c hi n thu t toán ch a t t

H ng phát tri n

Áp d ng thêm các ph ng pháp phát hi n mã c khác hoàn thành ch ng trình quét mã c d n ti n t i th nghi m và áp d ng trong th c t i t quy mô nh t i l n. Xây d ng m t c s d li u mã c y h n, truy xu t nhanh h n và có tính áp d ng th c t cao. Xây d ng ch ng trình nh n d ng mã c s d ng c không ch c s d li u có s n mà s d ng c nh ng c s d li u c a các h ãng, t ch c cá nhân khác .

TÀI LI U THAM KH O

- [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>
- [5]. www.ollydbg.de
- [6]. www.threatexpert.com/
- [7]. www.mcafee.com
- [8]. www.codeproject.com/
- [9]. <http://www.securelist.com>

PH L C

Code ch ng trình quét theo chu i nh n d ng

- Code ph n quét mã c

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

{

    //Hàm dùng so sánh 2 chu i ký t có gi ng nhau không

    //Hàm tr v TRUE khi gi ng 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 so sánh 2 chu i ký t có gi ng 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 để scan ra làm ví dụ

    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);

//Trở về vị trí ban đầu

fseek(f,0,SEEK_SET);

fseek(fs,0,SEEK_END);

int posend=ftell(fs);

//Trở về vị trí ban đầu

fseek(fs,0,SEEK_SET);

//Tiến hành vòng lặp kiểm tra mã nhiễm độc

int j=0;

while (ftell(f) < posEnd)

{

// Kiểm tra tìm kiếm mã nhiễm độc virus
```

```
fread(&sign,sizeof(m_Sign),1,f);

//Duyet file can quet

while(ftell(fs)<posend)

{
    fseek(fs,j++,SEEK_SET);

    int curl;

    curl=ftell(fs)
        fread(strDatFile,32,1,fs);

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

    {

        //Nhấn đống ứng là virus

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

        lstRe->Items->Add(L"Phát hi  n : " + tmp + L"      ng d  n : " + 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();

}

//Ti n hành li t kê các t p tin trong th m c

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

{
```

```
try

{

//L p qui

scan(strFile);

}

catch(Exception ^e)

{

//B t l i }

}

}

}
```


- Code nh p d li u

```
System::String ^strFile;

System::String ^tmp;

FILE *f;

//M file s c l y m u

FILE *fs;

char strPathSign[]="mn.kkt";

char strDatFile[32];

m_Sign sign;

System::Boolean isHave=false;

int iTmpOffset;

int i;

strFile = txtPath->Text;

//Convert t chu i trong TextBox qua chu i d ng chu n c a ch ng trình

//T a v chu i v i 18 ký t

//Chuy n t d ng 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 kí tự string

//còn phải chuyển nó ra thành kí tự Char 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);

//Trở về vị trí ban đầu

fseek(f,0,SEEK_SET);

//Tiến hành vòng lặp kiểm tra mã nhúng

while (ftell(f) < posEnd)

{

//Đọc mã nhúng virus

fread(&sign,sizeof(m_Sign),1,f);

//Nhảy trong file mục quét tới offset mục xét

//Offset này lấy từ sign.lPos

fseek(fs,sign.lPos,SEEK_SET);

//Lấy 32 byte tới offset đó

fread(strDatFile,32,1,fs);

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

{

//Nhúng trùng mã nhúng
```

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

isHave=true;

System::Windows::Forms::MessageBox::Show(L"Mã nh  n d ng này  ã có
trong b ng mã",L"Trùng mã nh  n d ng");

break;

}

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

{

    //Nh  n d ng trùng tên virus  ã l u

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

    isHave=true;

    System::Windows::Forms::MessageBox::Show(L"Tên virus này  ã có trong
b ng mã nh  n d ng",L"Trùng mã nh  n d ng");

    break;

}

}

if (!isHave)

{

    //Sau khi ki m tra là ch  a có mã nh  n d ng này + Tên virus là thao tác ghi
m u tin d  li u virus m i vào
```

```
//Lấy dữ liệu từ Offset cần lấy (Từ txtOffset)

iTmpOffset=System::Int32::Parse(txtOffset->Text);

fseek(fs,iTmpOffset,SEEK_SET);

fread(strDatFile,32,1,fs);

//    Thêm dữ liệu vào mảng bit theo Struct m_Sign

//Hàm strcpy xử lý lỗi trong nhúng trình này (Do có ký tự NULL)

//Vì vậy hàm copy 2 mảng char sang mảng code bit

for (i=0;i < 32;i++)

    sign.Sign[i]=strDatFile[i];

    sign.lPos=iTmpOffset;

    strcpy(sign.Name,tmpName);

//Nhúng vào chuỗi file

fseek(f,0,SEEK_END);

//Ghi dữ liệu vào chuỗi file

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

System::Windows::Forms::MessageBox::Show(L"Đã tải xong mã tin nhắn  
liệu virus mới",L"Thành công");

}

fclose(f);
```

```
fclose(fs);
```

- Li t kê m u trong t p d li u

```
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 nh kích th c file

fseek(f,0,SEEK_END);

posEnd=ftell(f);

//Tr v v trí ban u

fseek(f,0,SEEK_SET);

//Ti n hành vòng l p li t kê mã nh n d ng

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);
```