 🙠🙟🕮🙝🙢

**BỘ GIÁO DỤC VÀ ĐÀO TẠO**

**TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN**

**KHOA CÔNG NGHỆ THÔNG TIN**

**

**CRACK PHẦN MỀM**

**Môn học**: Kiến trúc máy tính và hợp ngữ

**Giáo viên**: Chung Thùy Linh

**Lớp**: 18CTT5

**Người thực hiện**:

1. Ngô Nhật Tân – 18120547
2. Quách Hải Thanh – 18120561

**new2 Hồ Chí Minh, ngày 01 tháng 07 năm 2020**

**MỤC LỤC**

[**I.** **Bảng phân công công việc** 3](#_Toc44588801)

[**II.** **Phần mềm hỗ trợ** 3](#_Toc44588802)

[**III.** **Chương trình 1.3** 3](#_Toc44588803)

[**1.** **Tìm manh mối** 3](#_Toc44588804)

[**2.** **Giải quyết vấn đề** 6](#_Toc44588805)

[**IV.** **Chương trình 1.4** 8](#_Toc44588806)

[**1.** **Tìm manh mối** 8](#_Toc44588807)

[**2.** **Giải quyết vấn đề** 13](#_Toc44588808)

# **Bảng phân công công việc**

|  |  |  |
| --- | --- | --- |
| **Công việc** | **Người thực hiện** | **Tiến độ hoàn thành** |
| Chương trình 1.1 | Quách Hải Thanh |  |
| Chương trình 1.2 |  |
| Chương trình 1.3 | Ngô Nhật Tân | 100% |
| Chương trình 1.4 | 100% |
| Chương trình 1.5 | Quách Hải Thanh |  |

# **Phần mềm hỗ trợ**

Để hỗ trợ việc crack phần mềm, nhóm có sử dụng 2 chương trình là PEiD và OllyDbg

# **Chương trình 1.3**

1. **Tìm manh mối**

Đầu tiên chúng ta kiểm tra chương trình có bị pack hay không bằng phần mềm PeiD

A screenshot of a cell phone

Description automatically generated

Nhận thấy rằng chương trình không bị pack, chúng ta tiến hành phân tích mã hợp ngữ của chương trình bằng OllyDg.

A screen shot of a computer

Description automatically generatedChúng ta chạy thử chương trình và thấy hiện ra một màn hình console như sau, yêu cầu chúng ta nhập mật mã. Nhập ngẫu nhiên mật mã “1234” và thấy chương trình báo trả kết quả là “**bad password**” và đây có thể là một badboy để chúng ta giải quyết chương trình này.

Tuy nhiên có thể thấy chuỗi được in ra màn hình console không xuất hiện trong vùng nhớ của chương trình hoặc có khả năng không thể truy xuất khi chưa chạy chương trình. Ta lại tiếp tục tìm kiếm các chuỗi string (*Search for All referenced text string*)và nhận thấy rằng kết quả tìm được không có những chuỗi được in ra màn hình console.

A screenshot of a social media post

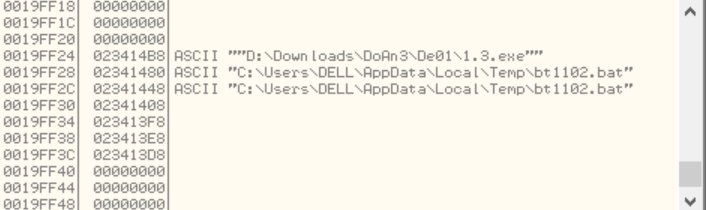
Description automatically generatedTiếp tục tìm đến Memory Map của chương trình, một cửa sổ được hiện ra có rất nhiều dòng dữ liệu trong đó bao gồm địa chỉ, kích thước, đối tượng quản lý vùng nhớ đó, … Ta tiến hành tìm kiếm từng vùng và phát hiện vùng nhớ này có điều đặc biệt.

Truy cập vào ta phát hiện chuỗi được in ra màn hình console xuất hiện trong vùng nhớ này.

A screenshot of a cell phone

Description automatically generatedTuy nhiên sau khởi động lại chương trình thì vùng nhớ này đã không còn tồn tại nữa, tìm kiếm những vùng nhớ lân cân đó ta vẫn không tìm được vùng nhớ đặc biệt đó.

A picture containing bird

Description automatically generatedTrong quá trình debug, khi dò tìm ta phát hiện có một stack đã lưu địa chỉ của vùng nhớ dump, nơi đang lưu chuỗi: “C:\Users\DELL\AppData\Local\Temp\” và một stack khác đang lưu chuỗi: “C:\Users\DELL\AppData\Local\Temp\bt1102.bat”

Đặt breakpoint và follow vào hàm “Call <JMP.&kernel32.WaitForSingleObject”

A screenshot of a cell phone

Description automatically generated Sau khi truy cập vào bên trong hàm, ta nhìn thấy có một vài câu lệnh khả nghi như &kernel32. WriteFile, … Ta đặt ra một nghi vấn, liệu chuỗi được in ra màn hình console có liên quan đến nội dung của file “C:\Users\DELL\AppData\Local\Temp\bt1102.bat” hay không?

Chúng tìm lần theo file path này và nhận thấy rằng file bt1102.bat có tồn tại. Thời gian file này được tạo ra cùng với thời gian hiện tại của hệ thống. Chúng ta chạy thử file này và nhận ra một điều đặc biệt rằng kết quả chúng ta nhận được tương tự với khi chúng A screenshot of a computer

Description automatically generatedta cho chạy chương trình 1.3.exe

Vậy ta có thể kết luận rằng file bat này được tạo ra khi chúng ta cho chạy chương trình 1.3.exe

1. **Giải quyết vấn đề**

A screenshot of a social media post

Description automatically generatedChúng ta mở code của file bat này lên để xem xét thử. Nhìn qua chúng ta thấy có một đoạn kiểm tra password (dòng 16, 17). Vậy chúng ta sẽ có được mật mã sau khi giải mã chuỗi “**%o%%llo%%he%%h%%t%%windir%billgates..2006**”.

A screenshot of a computer

Description automatically generatedCó 2 hướng để chúng ta giải mã chuỗi này, chúng ta có thể giải tay chuỗi này dựa trên quy tắc biến đổi chữ mà chương trình đã nêu ra (dòng 5 đến dòng 10) hoặc có thể chạy cmd để tìm ra password. Sau khi giải mã xong chúng ta được một chuỗi “**thellouyC:\WINDOWSbillgates..2006**”.

A screenshot of a computer

Description automatically generatedBây giờ chúng ta sẽ chạy lại chương trình và nhập chuỗi trên vào để xem kết quả. Và chuỗi sau khi giải mã chính xác là password của chương trình.

Vậy pass của chương trình là: **thellouyC:\WINDOWSbillgates..2006**

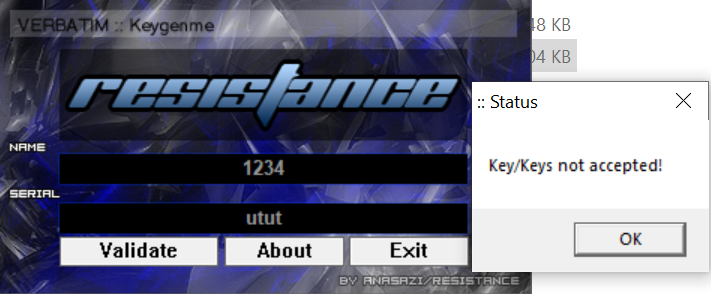
Quay trở lại một chút về vùng nhớ đặc biệt, nếu chúng ta để ý kỹ sẽ nhìn ra từ khóa “password” và đoạn lệnh so sánh password khi truy cập vùng nhớ này. Chúng ta vẫn có thể từ đây giải mã chuỗi để tìm ra password chính xác.

# **A close up of a newspaper Description automatically generatedChương trình 1.4**

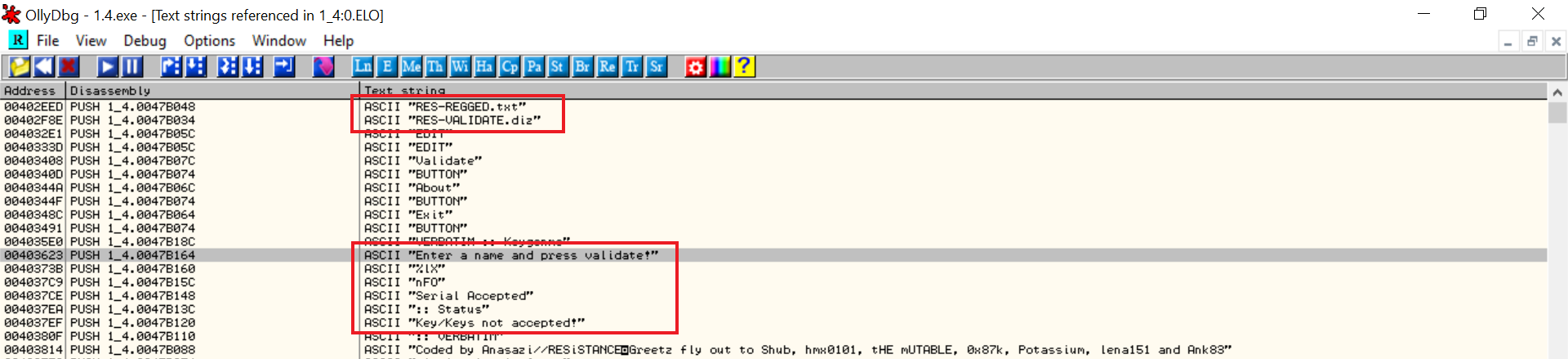
1. **Tìm manh mối**

Tương tự như ở các chương trình sau, bước đầu tiên vẫn cần phải dùng PeiD để kiểm tra xem chương trình này có bị pack hay không. Sau khi xác định chương trình không bị pack, ta tiến hành xem xét mã hợp ngữ của của nó bằng OllyDg.

Ta khởi động chương trình và nhận thấy chương trình yêu cầu chúng ta phải nhập Name và Serial, nhập ngẫu nhiên một cặp giá trị, ta nhận được kết quả trả về là **“Key/Keys not accepted”.** Vậy đây rất có thể là một badboy của chương trình.

A screenshot of a cell phone

Description automatically generated

Tìm kiếm các text string bên trong chương trình (search for all referenced text string), ta tìm được badboy mà chúng ta đã dự đoán ban đầu, bên cạnh đó ta nhìn thấy 2 text string có vẻ “đặc biệt sau”.

A screenshot of a social media post

Description automatically generatedQuay trở lại xem xét chương trình, ta tìm thấy 2 dòng GetDlgItemText A, có thể hiểu đây chính là hai dòng lấy dữ liệu của chương trình. Giá trị **“Name”** sẽ được lưu trữ tại địa chỉ [EBP -14] và giá trị **“Serial”** sẽ được lưu trữ tại địa chỉ [EBP -28].

Nhận thấy rằng cả hai đoạn chương trình có một điểm tương đồng nhau chính là đều gọi 2 hàm **CALL 1\_4.00415640** và **CALL 1\_4.00416040.** Lệnh **CALL 1\_4.00415640** được thực hiện **trước khi nhập dữ liệu** và lệnh **CALL 1\_4.00416040** được thực hiện **sau khi nhập dữ liệu.** Ta sẽ lần lượt follow vào bên trong 2 hàm này để xem các dòng lệnh của nó.

* Hàm tại địa chỉ 1\_4.00415640

A screenshot of a social media post

Description automatically generatedXem xét đoạn lệnh, ta tạm thời có thể nhận thấy không có câu lệnh nào làm thay đổi giá trị input chúng ta nhập vào, thử follow vào địa chỉ 00418A10 của chương trình cũng nhận thấy điểm nào khả nghi. Có khả năng đây chỉ là một hàm từ hệ thống và để hỗ trợ việc nhập xuất nên ta có thể tạm thời bỏ qua hàm tại địa chỉ này.

* A close up of a piece of paper

  Description automatically generatedHàm tại địa chỉ **1\_4.00416040**

Để tìm hiểu xem hàm này thực hiện thao tác gì ta tiến hành phân tích vòng lặp (loop) của chương trình. Tại mỗi vòng lặp, ta thấy chương trình thực hiện các câu lệnh:

* So sánh 2 giá trị ESI và EDI. Nếu bằng nhau thì thoát khỏi vòng lặp và return. Ngược lại, sẽ vẫn tiếp tục thực hiện vòng lặp.
* Gán AL = BYTE[EDI], AH = BYTE[ESI]
* Gán BYTE[EDI] = AH, BYTE[ESI] = AL
* Tăng ESI 1 đơn vị và giảm EDI 1 đơn vị

Nhận thấy rằng trong vòng lặp thực hiện hành động hoán vị 2 byte và song song đó index hoán vị sẽ lần lượt hoán vị kí tự đầu tiên và kí tự cuối cùng. Vậy ta rút ra một kết luận đây chính là hàm đảo ngược một chuỗi.

Quay trở lại đoạn lệnh nhập dữ liệu, xem xét đoạn lệnh sau khi nhập giá trị Name vào địa chỉ [EBP - 14]

|  |  |
| --- | --- |
| **Câu lệnh** | **Ý nghĩa** |
| LEA EDX, DWORD PTR [EBP -14] | Lưu name vào EDX |
| PUSH EDX | Lưu EDX và push vào stack |
| CALL 1\_4.00416040 | Thực hiện đảo chuỗi Name nhập vào (reversed name) |
| ADD ESP, 4 | ESP = reversed name |
| MOV ESI, ESP | ESI = ESP |

Tiếp tục xem xét đoạn lệnh sau khi chúng ta nhập Serie

|  |  |
| --- | --- |
| **Câu lệnh** | **Ý nghĩa** |
| CMP ESI, ESP |  |
| CALL 1\_4.0015640 |  |
| LEA EDX, DWORD PTR [EBP - 28] | Gán EDX = Serie |
| PUSH EDX | Lưu serie vào trong stack |
| CALL 1\_4.00416040 | Đảo ngược serie nhập vào |
| ADD ESP, 4 | Gán ESP = reversed serie |
| LEA EAX, DWORD PTR [EBP - 14] | Gán EAX = reversed name |
| PUSH EAX | Lưu reversed name vào stack |
| CALL 1\_4.00415FCO |  |
| ADD ESP, 4 |  |

A screenshot of a cell phone

Description automatically generatedTại đây chúng ta có thể thấy chương trình lại gọi thêm 1 lệnh CALL 1\_4.00415FCO, để kiểm tra xem hàm này thực hiện công việc gì, ta follow vào bên trong hàm.

Nhìn vào vòng lặp trên, ta thấy rằng tại đây AL sẽ lưu từng index của BYTE PTR và ECX tăng 1 đơn vị. Từ đó ta có thể suy luận rằng đây là hàm lấy chiều dài của chuỗi. Để làm tăng thêm tính đúng đắn của suy luận này ta có thể nhìn vào dòng (1) và (2) ở đoạn code bên dưới (đoạn code ở sau nhập serial), khi [EBP–54] = 0 và sau mỗi lần lặp thì so sánh với [EBP - 54], tức là index so với chiều dài.

|  |  |
| --- | --- |
| **Câu lệnh** | **Ý nghĩa** |
| MOV DWORD PTR [EBP – 40], EAX | Gán [EBP – 40] = length |
| MOV ECX, DWORD PTR [EBP - 48] | Lưu reversed name vào ECX |
| XOR ECX, DWORD PTR [EBP – 48] |  |
| MOV DWORD PTR [EBP – 48], ECX | Khởi tạo biến tạm [EBP – 48] = ECX = 0 |
| MOV DWORD PTR [EBP – 54 ], 0 | Khởi tạo index = 0 (1) |
| JMP SHORT 1-4.0040370E | Nhảy đến địa chỉ để thực hiện vòng lặp |

Xét các câu lệnh bên trong vòng lặp tại địa chỉ 0040370E

|  |  |
| --- | --- |
| **Câu lệnh** | **Ý nghĩa** |
| MOV EDX, DWORD PTR [EBP – 54] | Biến đếm |
| ADD EDX, 1 | Tăng biến đếm lên 1 đơn vị |
| MOV DWORD PTR [EBP – 54], EDX | Gán giá trị cho biến đếm mới |
| MOV EAX, DWORD PTR [EBP – 54] | Bắt đầu vòng lặp tại đây  EAX = index = 0 |
| CMP EAX, DWORD PTR [EBP – 40] | So sánh index với chiều dài chuỗi (2) |
| JGE SHORT 1-4.00403735 | Nếu bằng sẽ thoát khỏi vòng lặp |
| MOV ECX, DWORD PTR [EBP – 54] | Ngược lại gán ECX = index |
| MOVSX EDX, BYTE PTR[EBP+ECX-14] | Gán EDX = char[index] |
| MOV DWORD PTR [EBP-44], EDX | Lưu EDX và gán [EBP – 44] = EDX |
| MOV EAX, DWORD PTR [EBP – 44] | Gán EAX = [EBP – 44] |
| SUB EAX, 20 | EAX = EAX – (0x20) |
| MOV DWORD PTR [EBP – 44], EAX | Gán [EBP – 44] = EAX |
| MOV ECX, DWORD PTR [EBP – 48] | Lưu biến tạm vào ECX |
| SUB ECX, DWORD PTR [EBP – 44] | Lấy ECX – [EBP – 44] |
| MOV DWORD PTR [EBP – 48], ECX | Lưu ngược lại vào thanh ghi |
| JMP SHORT 1-4.00403705 | Quay trở lại vòng lặp |
| MOV ESI, ESP |  |
| MOV EDX, DWORD PTR [EBP – 48] | In chuỗi serial cuối cùng |
| PUSH EDX | Đẩy EDX vào stack |
| PUSH 1-4.0047B160 |  |
| LEA EAX, DWORD PTR [EBP - 3C] |  |
| PUSH EAX |  |
| CALL DWORD PTR [4A1494] |  |

1. **Giải quyết vấn đề**

A picture containing monitor, screen, television, table

Description automatically generatedChúng ta viết một chương trình C để sinh ra keygen như sau:

A screenshot of a cell phone

Description automatically generatedNhập thử một case bất kỳ để kiểm tra, ta được kết quả như sau:

Đây chính là Name và Serial chúng ta cần nhập vào tuy nhiên khi chúng ta thử nhập vào chương trình để kiểm tra, ta vẫn thấy chương trình thông báo cho chúng ta một “badboy”. Vậy muốn đạt được kết quả cuối cùng (goodboy), chúng ta cần phải trải qua một số bước nữa, cụ thể là:

CALL 1\_4.00415F30 (3)

ADD ESP, 8

MOV DWORD PTR [EBP - 58], EAX

CALL 1\_4.00401456 (4)

CMP DWORD PTR [4962F4], 1 (5)

JNZ SHORT 1\_4.004037E6

CMP DWORD PTR [4962F8], 1 (6)

JNZ SHORT 1\_4.004037E6

CMP DWORD PTR [EBP-58], 0 (7)

JNZ SHORT 1\_4.004037E6

MOV ESI, ESP

Để đạt được kết quả, ta phải thông qua 2 câu lệnh CALL (3) và (4), tiếp đó tùy thuộc vào điều kiện của (5), (6) và (7) sẽ dẫn đến địa chỉ chứa badboy: 004037E6.

* **Phân tích quá trình dẫn đến badboy và goodboy**

Để dẫn đến goodboy thì phải qua điều kiện ở các câu lệnh (5), (6) và (7) và các điều kiện trên phải trả về kết quả là TRUE (vì lệnh nhảy JNZ sẽ nhảy về địa chỉ của badboy khi kết quả là FALSE)

Bây giờ chúng ta sẽ phân tích 2 hàm CALL :

* Ở câu lệnh (3), ta follow vào bên trong hàm và phát hiện không có gì đặc biệt, vòng lặp của hàm cũng không làm thay đổi thanh ghi hay biến. Chính vì vậy tạm thời chúng ta sẽ bỏ qua hàm này.
* Ở câu lệnh (4), có thể thấy đây cũng chính là CALL cuối cùng của chúng ta, có khả năng hàm này chính là nơi dẫn đến kết quả.

Follow vào trong hàm (4), ta thấy chương trình nhảy đến 1 địa chỉ và tại đó ta thấy 1 chuỗi các câu lệnh, để ý các text string trong hàm này có liên quan đến các file RES-REGGED.txt và RES-VALIDATE.diz mà ta đã lưu ý ở ban đầu khi tìm kiếm các text string.

|  |  |
| --- | --- |
| **Câu lệnh** | **Ý nghĩa** |
| LEA EAX, DWORD PTR [EBP-20] | EAX = [EBP-20] = input từ file text |
| JE SHORT 1-4.00402FF1 |  |
| CMP DWORD PTR [EBP-20], 1 |  |
| JL SHORT 1-4.00402FDD | Kiểm tra xem có < 1 không |
| CMP DWORD PTR [EBP-20], 5 |  |
| JG SHORT 1-4.00402FDD | Kiểm tra xem có > 5 không |
| CALL 1-4.004015D7 | Nếu giá trị thỏa đoạn [1,5] thì gán [4962F4] = 1 (câu lệnh số (5)) |

Tại những dòng tiếp theo chương trình lại tiếp tục mở file RES-VALIDATE.diz lên để đọc

|  |  |
| --- | --- |
| **Câu lệnh** | **Ý nghĩa** |
| PUSH 1 |  |
| PUSH 1-4.0047B034 |  |
| LEA ECX, DWORD PTR [EBP-140] |  |
| CALL 1-4.00401168 |  |
| MOV EDX, DWORD PTR [EBP -140] |  |
| CALL 1\_4.00401226 |  |
| AND EAX,0FF |  |
| TEST EAX, EAX | Kiểm tra xem EAX = 0  Nghĩa là đầu vào rỗng => [4962F8] = 1 |
| JE SHORT 1\_4.00402FC8 |  |
| MOV DWORD PTR [4962F8], 0 |  |
| JMP SHORT 1\_4.00402FD2 |  |
| MOV DWORD PTR [4962F8], 1 | TEST EAX , EAX dẫn ra đến goodboy thỏa câu lệnh (6) |
| LEA ECX,DWORD PTR [EBP-B0] |  |
| CALL 1\_4.00401393 |  |

Vậy để đạt được goodboy, bên cạnh việc tìm được serial chúng ta cần phải tạo thêm 2 file như sau:

* File RES-VALIDATE.txt với nội dung là số từ [1,5]
* File RES-VALIDATE.diz với nội dung rỗng ( EAX = 0)

A screen shot of a social media post

Description automatically generatedTrong chương trình C tạo keygen, ta viết thêm đoạn lệnh tạo 2 file đó như sau:

Bây giờ chúng ta sẽ cho chạy lại chương trình cũng với test case là “1234”, lấy 2 file được tạo ra để cùng 1 thư mục với chương trình 1.4.exe, ta thử nhập serial mà ta đã tìm ra và xem kết quả như thế nào.

Kết quả là một goodboy. Vậy chúng ta đã thành công

A screenshot of a computer

Description automatically generated