# BÁO CÁO ĐỒ ÁN 3 – CRACKING

1. Thành viên nhóm

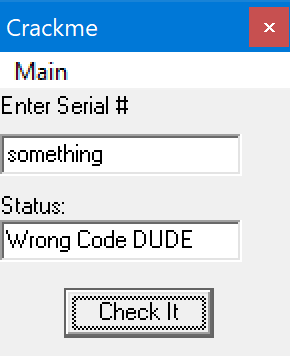
|  |  |  |
| --- | --- | --- |
| STT | Tên thành viên | MSSV |
| 1 | Nguyễn Bảo Long | 18120201 |
| 2 | Võ Thế Minh | 18120211 |
| 3 | Phạm Văn Minh Phương | 18120227 |
| 4 | Trà Anh Toàn | 18120662 |
| 5 | Mai Ngọc Tú | 18120253 |

1. Phân công công việc và đánh giá mức độ hoàn thành

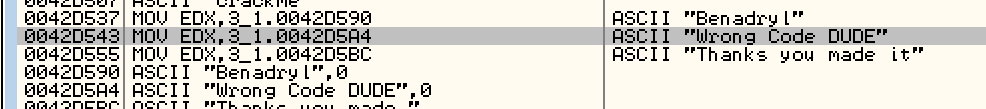
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| STT | Tên công việc | Yêu cầu | Người thực hiện | Đánh giá |
| 1 | Câu 3.1 |  | Phạm Văn Minh Phương | 100% |
| 2 | Câu 3.2 | Cơ bản | Trà Anh Toàn | 100% |
| Nâng cao | Mai Ngọc Tú | 100% |
| 3 | Câu 3.3 | Cơ bản | Nguyễn Bảo Long | 100% |
| Nâng cao | Võ Thế Minh | 100% |
| 4 | Viết báo cáo |  | Nguyễn Bảo Long | 100% |

1. Quá trình crack phần mềm
2. Bài 3.1

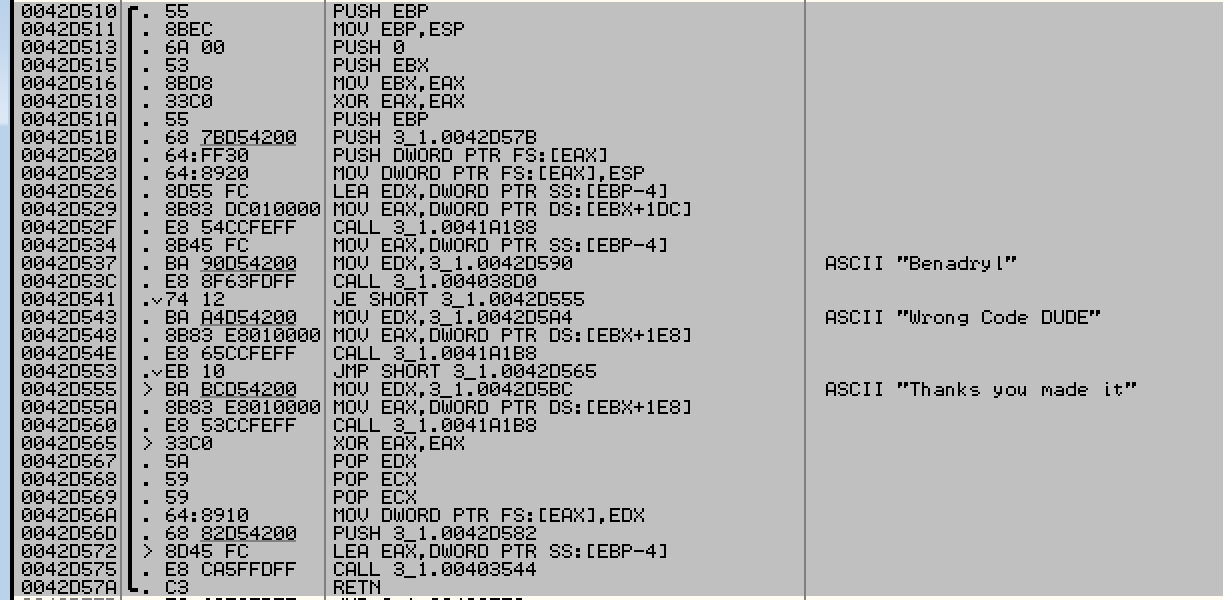
* Dựa vào thông báo lỗi, ta tìm chuỗi **Wrong Code DUDE**



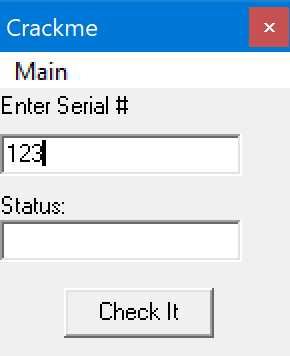
* Vô tình, chúng ta thấy được dòng thông báo **Thanks you made it** – ám chỉ việc active thành công phần mềm



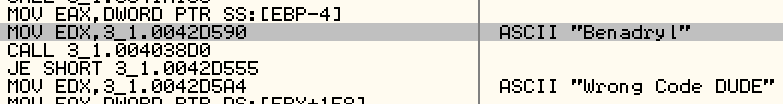
* Sau đó chúng ta đi đến đoạn code gọi đến thông báo **Thanks you made it** và **Wrong Code DUDE**

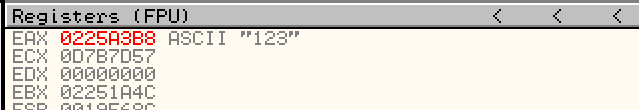


* Đặt breakpoint ở dòng **0042D510 PUSH EBP**. Ta nhập một giá trị bất kỳ vào khung Enter Serial rồi chạy debug từng dòng.

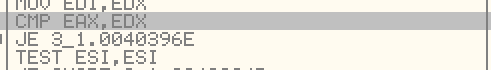


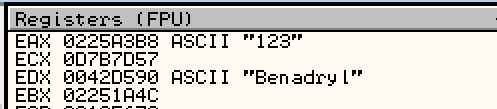
* Để ý rằng, khi chạy qua dòng **MOV EDX,3\_1.0042D590**thì thanh ghi **EAX** mang giá trị vừa nhập vào **123**



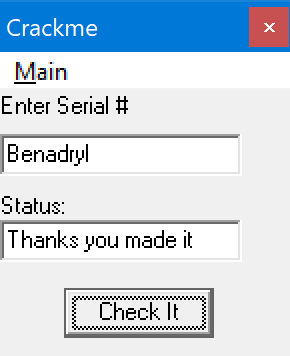


* Tiếp tục debug từng dòng lệnh, ta đến được đoạn code chương trình so sánh 2 thanh ghi **EAX** chứa giá trị **123** với thanh ghi **EDX** chứa giá trị **Benadryl**. Tới đây, ta hoàn toàn có quyền nghi ngờ rằng **Benadryl** chính là giá trị cần tìm.





* Thử nhập **Benadryl** vào khung Enter Serial, ta được kết quả sau



1. Bài 3.2

* Tìm GOOD BOY, BAD BOY: Thử nhập bất kỳ 1 Username và 1 Registration Code, ta không thấy thông báo. Tiến hành đọc code hợp ngữ và tìm kiếm với những string khả nghi như “Success”, “Successfully”,… ta tìm được:

A screenshot of text

Description automatically generated

* Nhìn vào địa chỉ của dòng **FreshUI has been registered successfully**, ta nhận thấy ở phía trên có một lệnh nhảy chạy đến thông báo này. Đặt breakpoint tại đó.

A screenshot of a cell phone

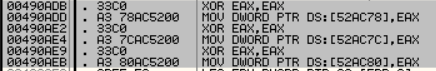
Description automatically generated

* Trên dòng **JE SHORT 3\_2.00490D33** có lệnh **CMP AL, 1**, nghĩa là sau khi thực hiện hàm nào đó, nếu giá trị **AL = 1** thì nhảy tới thông báo **FreshUI has been registered successfully**.
* Tiếp tục debug trên từng dòng, ta nhận thấy, để thực thi lệnh **JE SHORT 3\_2.00490D33**, ta cần phải thực hiện lệnh **JE SHORT 3\_2.00490D13**. Như vậy, phải cho giá trị ở ô nhớ **52AC80** bằng 0 (**[52AC80] = 0**)
* Ta xét lệnh **CALL 3\_2.00490ABC** – hàm gần nhất để nhảy tới lệnh thông báo nhập key đúng. Đặt breakpoint ở dòng này và nhấn F9 để nhảy đến đó. Sau đó ngưng lại.
* Nhập ngẫu nhiên 1 giá trị User Name và Registration Code.

A screenshot of a cell phone

Description automatically generated

* Chạy từng dòng lệnh để chương trình chạy đến đoạn lệnh mà **CALL 3\_2.00490ABC** gọi tới



* Ta thấy những ô nhớ **[52AC78], [52AC7C]** và **[52AC80]** đều là những ô nhớ mang giá trị quyết định cho điều kiện lệnh nhảy để nhảy tới GOODBOY như những hình trên. Gán giá trị những ô nhớ đó bằng **EAX**, mà **XOR EAX, EAX** nghĩa là gán **EAX = 0**

🡪 **Có thể đây là ô nhớ chứa biến Boolean dùng kiểm tra xem key nhập vào đúng hay sai.**

* Để kiểm tra giả thuyết, ta chọn dòng có ô nhớ **[52AC78]** 🡪 nhấn chuột phải 🡪 Find references to 🡪 Address constant. Một hộp thoại với những dòng lệnh có gọi ô nhớ **[52AC78]** sẽ xuất hiện.

A screenshot of a cell phone

Description automatically generated

* Giá trị của ô nhớ này chỉ thay đổi bởi 3 lệnh, nhấn đúp vào từng lệnh để xem vị trí thì thấy nó đều nằm trong hàm chúng ta đang xét. Giá trị của ô nhớ được khởi tạo lần đầu khi gán với **EAX**, và chỉ thay đổi xoay quanh giá trị 0 và -1

🡪 **Giả thuyết đặt ra có thể đúng (nếu key đúng thì gán 0, key sai gán -1).**

* Làm tương tự với hai ô nhớ còn lại cũng cho kết quả tương tự. Tại sao lại cần tới 3 biến kiểm tra key?

🡪 **Có thể có 3 loại key.**

* Tiếp tục chạy từng dòng lệnh, ta bắt gặp 3 vòng lặp như sau

A circuit board

Description automatically generated

* Đây chính là đoạn code kiểm tra giá trị key nhập vào có khớp với các key có sẵn trong chương trình hay không. Do đó, chúng quyết định việc đoạn code tiếp theo có quay lại GOOD BOY hay không.

A screenshot of a cell phone

Description automatically generated

* Như đã trình bày ở trên, ba ô nhớ **[52AC80]**, **[52AC78]** và **[52AC7C]** là những ô nhớ lưu kết quả so sánh key từng loại với registration code người dùng nhập vào. Ở đây sau khi thực hiện hàm tại dòng lệnh **3\_2.00490ABC**, ta có 3 lệnh **CMP** tương ứng để kiểm tra xem liệu registration code có khớp với key của hệ thống hay không.
* Sau khi thực hiện lệnh **CALL 3\_2.00490*ABC***), kết quả sẽ trả về 3 ô nhớ **[52AC80]**, **[52AC78]** và **[52AC7C]**. Tuy nhiên giá trị ô nhớ không phải là cái quyết định registration code ta nhập đúng hay sai, mà là giá trị **AL**.
* Lệnh **CMP <ô nhớ>, 1** chỉ để tác động tới cờ **CF**. Nếu **ô nhớ** có giá trị 1, cờ **CF = 0**. Do đó **SBB EAX, EAX** cho kết quả **EAX = 0**. Tiếp tục thực hiện lệnh **INC EAX** cho kết quả **EAX = 1**. Suy ra **AL = 1**. Lúc đó ta mới tới được lệnh thông báo thành công.
* Chú ý là các ô nhớ không hề được gán giá trị 1 trong bất cứ dòng lệnh nào, nếu registration code ta nhập đúng thì nó sẽ lưu giá trị thành **-1**. Khi dùng lệnh **CMP <ô nhớ>, 1**, nếu ô nhớ có giá trị **-1** thì không thay đổi cờ **CF**, từ đó **AL = 1** và chương trình vẫn sẽ thông báo ta nhập đúng.
* Ta đã biết có 3 loại key trong chương trình nhưng chưa biết cụ thể đó là những loại key gì. Tìm trong bảng Search for all referenced text strings, ta thấy có dòng sau:

A screenshot of a cell phone

Description automatically generated

* Với những thông tin đã có, ta có thể suy ra: Chương trình có 3 loại key bao gồm **Personal, Business** và 1 loại key khác sẽ được đề cập đến sau.
* Quay trở lại đoạn code sau khi gọi lệnh **CALL 3\_2.00490ABC**(hàm kiểm tra registration code ta nhập vào với key của hệ thống). Đặt breakpoint ở lệnh **CMP [52AC80], 0** và cho chương trình chạy tới lệnh này, sau đó tiếp tục chạy tới lệnh **JE SHORT 3\_2.00490D13*.*** Thay đổi cờ **ZF** bằng cách nhấn đúp vào giá trị cờ ZF. Như vậy lệnh **JE** sẽ không được thực hiện, tức là ta đang giả sử registration code đúng ở loại key này. Tiếp tục nhấn F7, chương trình FreshUI hiện ra thông báo như sau:

A screenshot of a cell phone

Description automatically generated

* Vậy loại key cuối cùng là **Old** và do ô nhớ **[52AC80]** đánh dấu.
* Bây giờ ta sẽ tìm cách load key hệ thống và tìm tập hợp các key hệ thống trong chương trình. Thực hiện lại các bước đặt breakpoint tại lệnh **CALL 3\_2.00490ABC**, nhấn F9, nhập User Name, Registration Code và nhấn F7 để chạy vào lệnh CALL. Đặt breakpoint ở tất cả các câu lệnh **CALL DWORD PTR [EDX+14]**để xem số lượng key mỗi loại là bao nhiêu, nhấn F9 để chạy tới dòng đó, sau đó nhấn F8.

A close up of a receipt

Description automatically generated

* Quan sát bên thanh ghi Register ta có kết quả sau
* **CALL DWORD PTR [EDX+14]** đầu tiên

A screenshot of a cell phone

Description automatically generated

🡪 Key loại **Old** có số lượng là **4** key

* **CALL DWORD PTR [EDX+14]** thứ hai

A screenshot of a cell phone

Description automatically generated

🡪 Key loại **Personal** có **1F0 = 496** key

* **CALL DWORD PTR [EDX+14]** thứ ba

A screenshot of a cell phone

Description automatically generated

🡪 Key loại **Business** có **1F4 = 500** key

* Để tìm được danh sách các key, ta đặt breakpoint ở dòng lệnh **CALL DWORD PTR [EDI+C]**, nhấn giữ F7 cho đến khi tới được dòng sau

A screenshot of a social media post

Description automatically generated

* Nhấn chuột phải vào ô nhớ **EDI**, chọn Follow in Dump, giá trị của ô nhớ đó sẽ hiện ra ở bảng dưới

A screenshot of a cell phone

Description automatically generated

* Đây chính là danh sách các key. Ta xuất ra file bằng cách nhấn chuột phải vào ô nhớ bên cột Address 🡪 Backup 🡪 Save data to file. OllyDbg sẽ xuất dữ liệu ra cho chúng ta. Vào file vừa xuất ra, ta thấy có danh sách key ở đó

A screenshot of a computer

Description automatically generated

* Nhập 1 trong 3 loại key vào chương trình, ta nhận được các thông báo như sau:

A screenshot of a cell phone

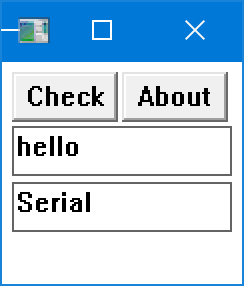
Description automatically generated

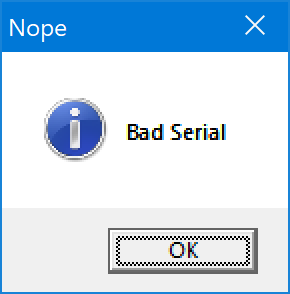
A screenshot of a cell phone

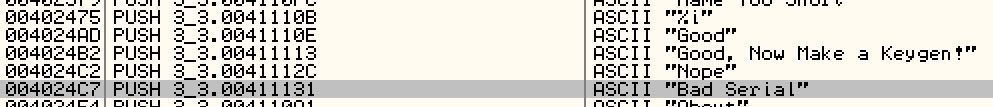
Description automatically generated

1. Bài 3.3

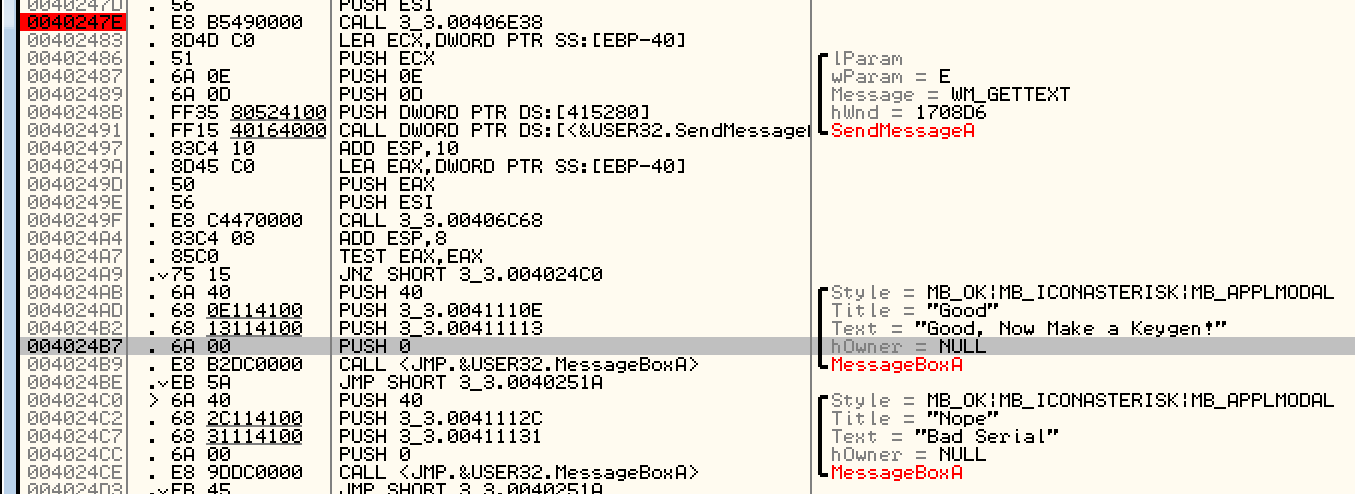
* Dựa vào thông báo lỗi, ta tìm đến chuỗi **Bad Serial**







* Đi đến đoạn code liên quan, ta được



* Đặt breakpoint ở dòng **0040247E**, chạy debug từng dòng lệnh. Nhập liệu đầu vào như hình sau

A screenshot of a cell phone

Description automatically generated

* Sau khi chạy qua lệnh **CALL 3\_3.00406E38**, ta thấy giá trị khả nghi trong thanh ghi **EDX “803218662”**, **ESI = “803218662”**

A screenshot of a cell phone

Description automatically generated

A screenshot of a cell phone

Description automatically generated

* Tới lệnh **JNZ 3\_3.004024C0** là lệnh nhảy đến GOODBOY nên ta có thể đoán là lệnh **TEST EAX, EAX** là kiểm tra điều kiện chuỗi ta nhập vào

A screenshot of a cell phone

Description automatically generated

* **CALL 3\_3.004024C0** là lệnh so sánh giá trị trong **EDX = “123456789”** với giá trị trong thành ghi **ECX = “803218662”** nếu đúng thì **EAX = 0**, nếu sai thì **EAX = 1** như hình sau

A screenshot of a cell phone

Description automatically generated

* Chạy tiếp tục tiếp theo chắc chắn sẽ chạy vào BAD BOY.
* Nhập lại User Name là **votheminh** và Serial là **803218662**, ta nhận được kết quả sau

A screenshot of a cell phone

Description automatically generated

* Vậy với Username là **votheminh** thì Serial tương ứng là **803218662**
* Để tạo KeyGen, chúng ta chú ý 3 đoạn code sau
* Đoạn thứ nhất



* Lệnh **0040235B** là hàm mã hóa lần 1
* Mã hóa chuỗi ta nhập vào từ ký tự thành **hệ Thập Lục Phân** trong bảng mã ASCII
* Đoạn code thứ 2

A close up of text on a white background

Description automatically generated

* Các lệnh từ dòng **004023A3** đến **0040246A** là tập các hàm mã hóa lần 2
* Mã hóa **chuỗi** **Thập Lục Phân từ hàm mã hóa lần 1** thành 1 **chuỗi thập lục phân mới**, thuật toán đã được trình bày trong file **Solution3\_3.pdf**
* Đoạn code thứ 3



* Dòng lệnh **00402491** gọi đến hàm mã hóa lần 3
* Mã hóa từ chuỗi thập lục phân từ hàm mã hóa lần 2 thành 1 chuỗi thập lục phân mới, thuật toán đã được trình bày trong file **Solution3\_3.pdf**
* Hình ảnh đoạn code mã hoá lần 3

A close up of text on a white background

Description automatically generated