BÁO CÁO ĐỒ ÁN THỰC HÀNH 3:

**CRACK**

Môn: Kiến trúc máy tính và Hợp ngữ

Sinh viên thực hiện:

|  |  |
| --- | --- |
| Mã số sinh viên | Họ tên |
| 1712645 | Bùi Thị Cẩm Nhung |
| 1712747 | Nguyễn Ngọc Băng Tâm |
| 1712856 | Huỳnh Văn Tú |

# I. Giới thiệu

## 1.1. Mô tả đồ án

Sinh viên ứng dụng các kiến thức đã học để crack chương trình:

* Mô tả thuật toán phát sinh key của chương trình.
* Minh họa một key hợp lệ với một username bất kỳ.
* Viết chương trình phát sinh keygen (nếu có).

## 1.2. Đánh giá mức độ hoàn thành

* Đánh giá tổng thể:
* Chi tiết từng yêu cầu:

|  |  |  |
| --- | --- | --- |
| STT | Yêu cầu | Đã hoàn thành |
| 1 | Mô tả thuật toán phát sinh key của chương trình. |  |
| 2 | Minh họa một key hợp lệ với một username bất kỳ. |  |
| 3 | Viết chương trình phát sinh keygen (nếu có). |  |

## 1.3. Phân công công việc

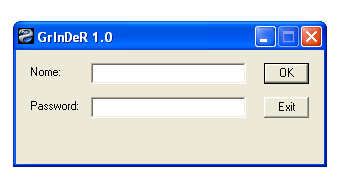
|  |  |  |
| --- | --- | --- |
| MSSV | Họ tên | Công việc thực hiện |
| 1712645 | Bùi Thị Cẩm Nhung | Thực hiện crack target 1 - Grinder |
| 1712747 | Nguyễn Ngọc Băng Tâm | Thực hiện crack target 2 - |
| 1712856 | Huỳnh Văn Tú | Thực hiện crack target 3 - |

# II. Nội dung thực hiện

## 2.1. Target1: Grinder

### 2.1.1. Phân tích

Trước tiên, ta tiến hành một số phân tích bằng cách chạy thử target:



Đây là một Keygemme - một chương trình được thiết kế để người giải không chỉ tìm ra được thuật toán bảo vệ đang được sử dụng mà còn có thể tự viết ra một chương trình keygen phát sinh các cặp key hợp lệ bằng ngôn ngữ lập trình.

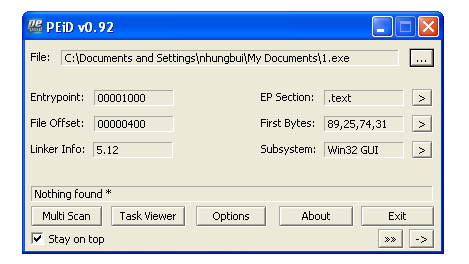
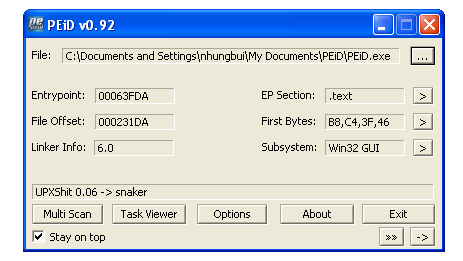
Với target được cho, ta chưa thể loại bỏ khả năng khi nhập vào một "Nome", target sẽ tự phát sinh một "Password" tương ứng và so khớp giá trị được sinh với giá trị do người dùng nhập vào. Hơn nữa với các chương trình Keygemme, có thể có rất nhiều thủ thuật anti-debug và anti-routine được áp dụng nhằm làm tăng khó khăn trong quá trình giải.

Ta thử nhập một cặp Nome - Password bất kỳ (ví dụ btcnhung - 121299), sau đó nhấn nút OK. Một hộp thoại Nag hiện ra:



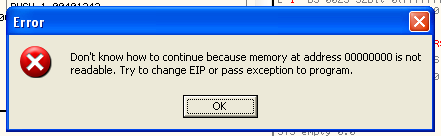
Như vậy, ta biết được BadBoy của chương trình này là "Try again, Sir!".

Thử kiểm tra xem target này có bị pack hay protect không. Ta sử dụng PEiD với chế độ "Normal Scan", tuy nhiên không thu được kết quả gì. Thử lại với chế độ "Deep Scan".

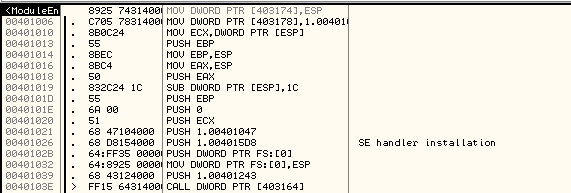


Từ kết quả scan, ta phát hiện target được patch bằng chương trình UPXShit 0.06. Để đơn giản, chúng ta sẽ unpatch trực tiếp vào target.

Sau khi mở file bằng OllyDbg, nhấn F9 để chạy chương trình thì nhận được lỗi sau:



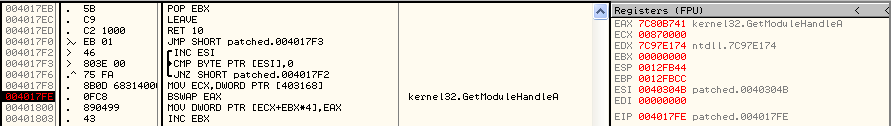
Thử chạy từng lệnh một, ta phát hiện được lỗi phát sinh bởi Exception Handler khi chạy đến lệnh 40103E. Lệnh này yêu cầu gọi procedure tại [403164] vốn đang là địa chỉ rỗng. Như vậy ta đoán được lỗi phát sinh là Access Violation.



Ta tạm thời patch lại đoạn lệnh này để chạy tiếp.



Sau khi quan sát các lệnh tiếp theo, ta thấy câu lệnh 004017FE có thể cung cấp một số thông tin hữu ích. Thử đặt breakpoint ở lệnh 004017FE.

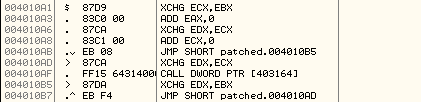


Nhấn F9 thêm vài lần nữa, ta thấy giá trị EAX thay đổi: kernal32.GetModuleHandleA, kernal32.VirtualAlloc, USER32.DialogBoxParamA, ..., USER32.MessageBoxA. Tương ứng, giá trị EBX cũng thay đổi 00000000, 00000001, 00000002, ..., 00000013. Nói cách khác, EAX lưu địa chỉ của hàm được gọi và EBX lưu index của hàm tương ứng dùng để hiển thị cửa sổ chương trình.

Đặt thêm một breakpoint ở lệnh 401047 và nhấn F9. Ta nhận thấy rằng sau khi thực hiện xong các hàm nói trên, ta lại quay trở về để thực hiện tiếp lệnh ở 401047.

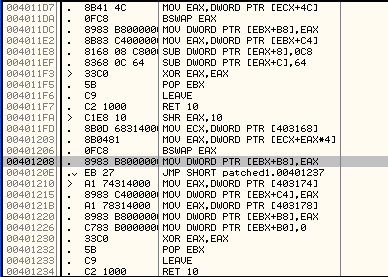


Lúc này nếu ta chạy tiếp sẽ lại nhận được một hộp thoại báo Access Violation tương tự như trên ở câu lệnh gọi procedure tại địa chỉ 4010A1.



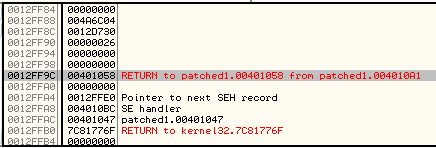
Khảo sát procedure tại địa chỉ 4010A1, không nghi ngờ nữa gì đây cũng là một Structured Exception Handler. Đọc qua đoạn code trên, ta phát hiện lại có câu lệnh 4010AF gọi tới procedure tại [403164] vốn đang là một địa chỉ rỗng. Để đơn giản, ta xử lý bằng cách patch đoạn code này lại bằng cách thức tương tự như đã trình bày ở trên.

Thử đặt breakpoint tại câu lệnh 4010BC và nhấn F9 để chạy. Chương trình ngừng lại ở breakpoint đã đặt. Lúc này ta cần quan sát kỹ hơn các câu lệnh kế tiếp.



Tại câu lệnh 401208, giá trị trong thanh EIP được ghi vào địa chỉ của kernal32.GetModuleHandleA. Thử đặt breakpoint ở đây để nhận thấy điều này rõ hơn.





Dựa vào các giá trị trong bộ nhớ, ta biết rằng nếu chạy tiếp chương trình chắc chắn sẽ lại báo lỗi Access Violation. Thế nhưng tại sao chương trình gốc lại chạy được? Sau khi thử lại, ta phát hiện ra trigger gốc "CALL 0" mới thực sự đưa địa chỉ của câu lệnh tiếp theo vào stack chứ không phải "MOV [0], EAX" như ta đã làm. Ta tiến hành patch lại procedure tại 4010AF.



Tới đây ta đã có thể cơ bản chạy được chương trình.

# III. Thử nghiệm

# IV. Tài liệu tham khảo