

TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN
KHOA CÔNG NGHỆ THÔNG TIN

TPHCM, 2019



CRACK PHẦN MỀM

ĐỀ 1

MÔN HỌC: KIẾN TRÚC MÁY TÍNH

Sinh viên thực hiện:

Huỳnh Văn Tú: 1712856

Bùi Thị Cẩm Nhung: 1712645

Nguyễn Ngọc Băng Tâm: 1712747

MỤC LỤC

1. Giới thiệu	3
1.1. Mô tả đồ án.....	3
1.2. Đánh giá mức độ hoàn thành.....	3
1.3. Phân công công việc.....	3
2. Nội dung thực hiện.....	3
2.1. Target1: Grinder.....	3
2.1.1. Phân tích	3
2.1.2. Thuật toán mã hóa	8
2.2. Target2: Crackme#3.....	10
2.2.1. Phân tích	10
2.2.1. Thuật toán mã hóa	16
2.3. Target3: Beam.....	16
2.3.1. Phân tích	16
2.3.1. Thuật toán mã hóa	16
3. Thử nghiệm.....	16
3.1. Target 1	16
4. Tham khảo	17

1. Giới thiệu

1.1. Mô tả đề 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: 60%
- Chi tiết từng yêu cầu:

STT	Yêu cầu	Đã hoàn thành
1	Mô tả thuật toán phát sinh key của chương trình.	60%
2	Minh họa một key hợp lệ với một username bất kỳ.	40%
3	Viết chương trình phát sinh keygen (nếu có).	40%

1.3. Phân công công việc

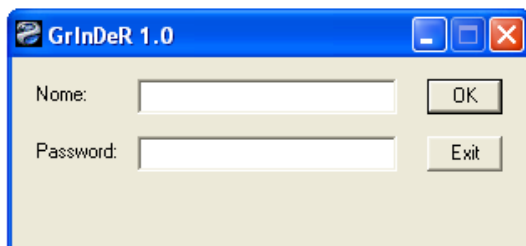
MSSV	Họ tên	Công việc thực hiện
1712645	Bùi Thị Cẩm Nhung	Thực hiện crack target 1 - Grinder
1712747	Nguyễn Ngọc Băng Tâm	Thực hiện crack target 2 - Crackme#3
1712856	Huỳnh Văn Tú	Thực hiện crack target 3 – Beam, Viết keygen cho target 1.

2. 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.

Đồ án 3: Crack phần mềm

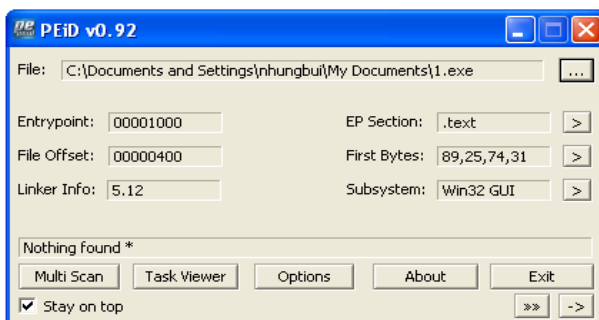
Với target được cho, hoàn toàn có khả năng khi nhập vào một "Nome", target sẽ tự phát sinh một "Password" tương ứng và so khớp giá trị được sinh với giá trị do người dùng nhập vào. Hơn nữa với các chương trình Keygemme, có thể có rất nhiều thủ thuật anti-debug và anti-routine được áp dụng nhằm làm tăng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



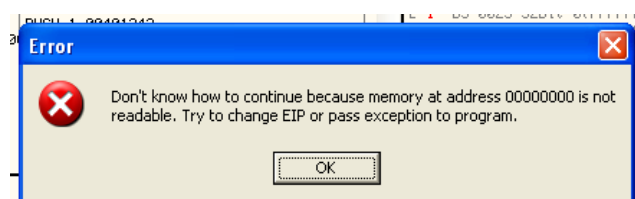
Như vậy, ta biết được BadBoy của chương trình này là "Try again, Sir!".

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.



Từ kết quả scan, ta phát hiện target được patch bằng chương trình UPXShit 0.06. Để đơn giản, chúng ta sẽ chọn cách 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.

0040101E	. 6A 00	PUSH 0	
00401020	. 51	PUSH EAX	
00401021	. 68 47104000	PUSH 1.00401047	
00401026	. 68 D8154000	PUSH 1.004015D8	
0040102B	. 64:FF35 0000	PUSH DWORD PTR FS:[0]	SE handler installation
00401032	. 64:8925 0000	MOV DWORD PTR FS:[0],ESP	
00401039	. 68 43124000	PUSH 1.00401243	
0040103E	> FF15 64314000	CALL DWORD PTR [403164]	

Ta tạm thời patch lại đoạn lệnh này như sau:

1712645_1712747_1712856

Đồ án 3: Crack phần mềm

0040103E	AS 00000000	MOV DWORD PTR [0],EAX
00401043	90	NOP

Sau khi quan sát các lệnh tiếp theo, ta thấy câu lệnh tại địa chỉ 4017FE có thể cung cấp một số thông tin hữu ích. Thử đặt breakpoint ở lệnh này và chạy tiếp chương trình.

004017EB	5B	POP EBX	
004017EC	C9	LEAVE	
004017ED	C2 1000	RET 10	
004017F0	EB 01	JMP SHORT patched.004017F3	
004017F2	46	INC ESI	
004017F3	80 00	CMP BYTE PTR [ESI],0	
004017F6	75 FA	JNZ SHORT patched.004017F2	
004017F8	8B 0D 68314000	MOV ECX,DWORD PTR [403168]	
004017FE	8FC8	BSWAP EAX	kernel32.GetModuleHandleA
00401800	89 04 99	MOV DWORD PTR [ECX+EBX*4],EAX	
00401803	43	INC EBX	

Registers (FPU)
EAX: 7C80B741 kernel32.GetModuleHandleA
ECX: 00870000
EDX: 7C97E174 ntdll.7C97E174
EBX: 00000000
ESP: 0012FB44
EBP: 0012FBCC
ESI: 0040304B patched.0040304B
EDI: 00000000
EIP: 004017FE patched.004017FE

Rõ ràng chương trình vẫn chạy tới breakpoint mà không xảy ra lỗi gì. Nhấn F9 thêm vài lần nữa, ta thấy giá trị EAX thay đổi: kernel32.GetModuleHandleA, kernel32.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 có địa chỉ 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 trên.

00401047	6A 00	PUSH 0	
00401049	33C0	XOR EAX,EAX	
0040104B	B8 00000000	MOV EAX,0	
00401050	C1E0 10	SHL EAX,10	
00401053	E8 49000000	CALL patched.004010A1	

Lúc này nếu ta chạy tiếp sẽ lại nhận được một hộp thoại báo Access Violation tương tự như đã đề cập trước đó ở câu lệnh có đại 4010AF gọi procedure tại địa chỉ 4010A1.

004010A1	87D9	XCHG ECX,EBX	
004010A3	83C0 00	ADD EAX,0	
004010A6	87CA	XCHG EDI,ECX	
004010A8	83C1 00	ADD ECX,0	
004010AB	EB 08	JMP SHORT patched.004010B5	
004010AD	87CA	XCHG EDI,ECX	
004010AF	FF15 64314000	CALL DWORD PTR [403164]	
004010B5	87DA	XCHG EDI,EBX	
004010B7	EB F4	JMP SHORT patched.004010AD	

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.

004011D7	8B41 4C	MOV EAX,DWORD PTR [ECX+4C]	
004011DA	0FC8	BSWAP EAX	
004011DC	8983 B8000000	MOV DWORD PTR [EBX+B8],EAX	
004011E2	8B83 C4000000	MOV EAX,DWORD PTR [EBX+C4]	
004011E8	8168 08 C800	SUB DWORD PTR [EAX+8],0C8	
004011EF	8368 0C 64	SUB DWORD PTR [EAX+C],64	
004011F3	33C0	XOR EAX,EAX	
004011F5	5B	POP EBX	
004011F6	C9	LEAVE	
004011F7	C2 1000	RET 10	
004011FA	C1E8 10	SHR EAX,10	
004011FD	8B 0D 68314000	MOV ECX,DWORD PTR [403168]	
00401203	8B 04 81	MOV EAX,DWORD PTR [ECX+EAX*4]	
00401206	0FC8	BSWAP EAX	
00401208	8983 B8000000	MOV DWORD PTR [EBX+B8],EAX	
0040120E	EB 27	JMP SHORT patched1.00401237	
00401210	A1 74314000	MOV EAX,DWORD PTR [403174]	
00401215	8983 C4000000	MOV DWORD PTR [EBX+C4],EAX	
0040121B	A1 78314000	MOV EAX,DWORD PTR [403178]	
00401220	8983 B8000000	MOV DWORD PTR [EBX+B8],EAX	
00401226	C783 B0000000	MOV DWORD PTR [EBX+B0],0	
00401230	33C0	XOR EAX,EAX	
00401232	5B	POP EBX	
00401233	C9	LEAVE	
00401234	C2 1000	RET 10	

Đồ án 3: Crack phần mềm

Tại câu lệnh 401208, giá trị trong thanh EIP được ghi vào địa chỉ của kernel32.GetModuleHandleA. Thử đặt breakpoint ở đây để nhận thấy điều này rõ hơn.

004011FA	>	C1E8 10	SHR EAX,10		EIP 00401208 patched1.00401208
004011FD	.	8B80 68314000	MOV ECX,DWORD PTR [403168]		C 0 ES 0023 32bit 0(FFFFFFFF)
00401203	.	8B0481	MOV EAX,DWORD PTR [ECX+EAX*4]		P 1 CS 001B 32bit 0(FFFFFFFF)
00401206	.	9F03	BSWAP EAX		R 0 SS 0023 32bit 0(FFFFFFFF)
00401208	.	9983 B8000000	MOV DWORD PTR [EBX+B8],EAX	kernel32.GetModuleHandleA	Z 1 DS 0023 32bit 0(FFFFFFFF)
0040120E	>	EB 27	JMP SHORT patched1.00401237		S 0 FS 003B 32bit 7FFDF000(FFF)
00401210	>	A1 74314000	MOV EAX,DWORD PTR [403174]		T 0 GS 0000 NULL
00401215	.	9983 C4000000	MOV DWORD PTR [EBX+C4],EAX		n n

0012FF84	00000000	
0012FF88	00406C04	
0012FF8C	0012D730	
0012FF90	00000026	
0012FF94	00000000	
0012FF98	00000000	
0012FF9C	00401058	RETURN to patched1.00401058 from patched1.004010A1
0012FFA0	00000000	
0012FFA4	0012FFE0	Pointer to next SEH record
0012FFA8	004010BC	SE handler
0012FFAC	00401047	patched1.00401047
0012FFB0	7C81776F	RETURN to kernel32.7C81776F
0012FFB4	00000000	

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 "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.

004010AF	50	PUSH EAX
004010B0	A3 00000000	MOV DWORD PTR [0],EAX

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

Trong cả quá trình phân tích trên, ta nhận thấy có khá nhiều câu lệnh gọi tới địa chỉ 4010A1. Nếu để ý kỹ một chút, ta sẽ nhận ra các câu lệnh này đều có chung một cấu trúc là:

PUSH ...

PUSH ...

...

MOV EAX, N

SHL EAX, 10

CALL 1.0040A1

Cấu trúc trên làm ta liên tưởng đến các imported function. Trong đó ta sử dụng câu lệnh PUSH để truyền các tham số API và lệnh MOV xác định API được gọi tới.

Đồ án 3: Crack phần mềm

00401249	. 817D 0C 1001	CMP DWORD PTR [EBP+C],110	WM_INITDIALOG
00401250	~ 0F85 CA000000	JNZ patched1.00401320	
00401256	. 68 F4010000	PUSH 1F4	
0040125B	. FF35 60314000	PUSH DWORD PTR [403160]	
00401261	. B8 04000000	MOV EAX,4	
00401266	. C1E0 10	SHL EAX,10	
00401269	. E8 33FEFFFF	CALL patched1.004010A1	LoadIconA
0040126E	. 50	PUSH EAX	
0040126F	. 6A 01	PUSH 1	
00401271	. 68 80000000	PUSH 00	WM_SETICON
00401276	. FF75 08	PUSH DWORD PTR [EBP+8]	
00401279	. B8 05000000	MOV EAX,5	
0040127E	. C1E0 10	SHL EAX,10	
00401281	. E8 1BF0FFFF	CALL patched1.004010A1	SendMessageA
00401286	. 68 D10B0000	PUSH 0BD1	
0040128B	. FF75 08	PUSH DWORD PTR [EBP+8]	
0040128E	. B8 06000000	MOV EAX,6	
00401293	. C1E0 10	SHL EAX,10	
00401296	. E8 06FEFFFF	CALL patched1.004010A1	GetDlgItem
0040129B	. 50	PUSH EAX	
0040129C	. B8 07000000	MOV EAX,7	
004012A1	. C1E0 10	SHL EAX,10	
004012A4	. E8 F8FDFFFF	CALL patched1.004010A1	SetFocus
004012A9	. B8 0A000000	MOV EAX,0A	
004012AE	. C1E0 10	SHL EAX,10	
004012B1	. E8 EBF0FFFF	CALL patched1.004010A1	GetCurrentProcessId
004012B6	. 50	PUSH EAX	

Trong các API được gọi tới, ta thấy có API CreateThread. Ta quan sát các câu lệnh có địa chỉ từ 401819 đến 40194B và rút ra nhận xét như sau: tại câu lệnh 40188A, chương trình mở một thread. Nếu không thành công, gọi hàm thoát ExitProcess. Sau đó chương trình đọc các giá trị trong bộ nhớ ReadProcessMemory và kiểm tra tổng tại câu lệnh có địa chỉ 4018DB. So sánh tổng mới với tổng đã được xác định trước đó để nhảy tới hàm tiếp theo. Do đó, ta sẽ patch lại câu lệnh ở địa chỉ 4018F2.

00401880	. 6A 10	PUSH 10	
00401882	. B8 0B000000	MOV EAX,0B	
00401887	. C1E0 10	SHL EAX,10	
0040188A	. E8 12F8FFFF	CALL patched1.004010A1	OpenProcess
0040188F	. 0BC0	OR EAX,EAX	
00401891	~ 75 19	JNZ SHORT patched1.004018AC	
00401893	. 6A 00	PUSH 0	
00401895	. B8 03000000	MOV EAX,3	
0040189A	. C1E0 10	SHL EAX,10	
0040189D	. 83EC 04	SUB ESP,4	
004018A0	. C70424 381A4	MOV DWORD PTR [ESP],patched1.00401A38	
004018A7	^ E9 F5F7FFFF	JMP patched1.004010A1	ExitProcess
004018AC	> 8945 FC	MOV DWORD PTR [EBP-4],EAX	
004018AF	> 8D45 F0	LEA EAX,DWORD PTR [EBP-10]	
004018B2	. 50	PUSH EAX	
004018B3	. FF75 F4	PUSH DWORD PTR [EBP-C]	
004018B6	. FF75 F8	PUSH DWORD PTR [EBP-8]	
004018B9	. 68 00104000	PUSH patched1.<ModuleEntryPoint>	
004018BE	. FF75 FC	PUSH DWORD PTR [EBP-4]	
004018C1	. B8 0C000000	MOV EAX,0C	
004018C6	. C1E0 10	SHL EAX,10	
004018C9	. E8 D3F7FFFF	CALL patched1.004010A1	ReadProcessMemory
004018CE	. B9 00000000	MOV ECX,0	
004018D3	. 8B75 F8	MOV ESI,DWORD PTR [EBP-8]	
004018D6	. 33C0	XOR EAX,EAX	
004018D8	. 50	PUSH EAX	
004018D9	~ EB 0B	JMP SHORT patched1.004018E6	
004018DB	> 0FB606	MOVZX EAX,BYTE PTR [ESI]	
004018DE	. C1E0 10	SHL EAX,10	
004018E1	. 010424	ADD DWORD PTR [ESP],EAX	

Với API SetTimer, ta có thể đoán rằng [403008] là một cờ quan trọng, do đó không thể bằng không. Tuy nhiên để đơn giản, ta cứ patch lại câu lệnh tại địa chỉ 401352.

00401320	> 817D 0C 1301	CMP DWORD PTR [EBP+C],113	WM_TIMER
00401327	~ 75 49	JNZ SHORT patched1.00401372	
00401329	. 817D 10 9101	CMP DWORD PTR [EBP+10],191	nIDEvent
00401330	~ 0F85 5E020000	JNZ patched1.00401594	
00401336	. 68 91010000	PUSH 91	
0040133B	. FF75 08	PUSH DWORD PTR [EBP+8]	
0040133E	. B8 12000000	MOV EAX,12	
00401343	. C1E0 10	SHL EAX,10	
00401346	. E8 56FDFFFF	CALL patched1.004010A1	KillTimer
0040134B	. 833D 09304000	CMP DWORD PTR [403008],0	
00401352	~ 75 19	JNZ SHORT patched1.0040136D	
00401354	. 6A 00	PUSH 0	
00401356	. B8 03000000	MOV EAX,3	
0040135B	. C1E0 10	SHL EAX,10	
0040135E	. 83EC 04	SUB ESP,4	
00401361	. C70424 381A4	MOV DWORD PTR [ESP],patched1.00401A38	
00401368	^ E9 34FDFFFF	JMP patched1.004010A1	
0040136D	~ E9 22020000	JMP patched1.00401594	
00401372	> 817D 0C 0104	CMP DWORD PTR [EBP+C],401	
00401379	~ 75 49	JNZ SHORT patched1.004013C4	

Với API TickCount, ta sẽ patch lại câu lệnh tại địa chỉ 401394.

Như vậy ta đã xử lý xong các thủ thuật anti-patches.

Đồ án 3: Crack phần mềm

2.1.2. Thuật toán mã hóa

Dựa vào các câu lệnh từ địa chỉ 401415 đến 401435, ta biết rằng một username hợp lệ phải nhiều hơn 6 ký tự.

00401415	. 8D45 EB	LEA EAX,DWORD PTR [EBP-15]	Username!!!
00401418	. 50	PUSH EAX	
00401419	. 6A 11	PUSH 11	
0040141B	. 6A 0D	PUSH 0D	WM_GETTEXT
0040141D	. 68 C0B00000	PUSH 0BCC	
00401422	. FF75 08	PUSH DWORD PTR [EBP+8]	
00401425	. B8 0E000000	MOV EAX,0E	
0040142A	. C1E0 10	SHL EAX,10	
0040142D	. E8 6FFCFFFF	CALL patched1.004010A1	SendDlgItemMessageA
00401432	. 83F8 06	CMP EAX,6	
00401435	~v 73 12	JNB SHORT patched1.00401449	username.length() > 6

Tiếp tục khảo sát các câu lệnh tiếp theo từ địa chỉ 401449 đến 401469, ta lại biết thêm một thông tin hữu ích khác: password phải có độ dài lớn hơn 16 ký tự.

00401449	> 8D45 DA	LEA EAX,DWORD PTR [EBP-26]	Password!!!
0040144C	. 50	PUSH EAX	
0040144D	. 6A 11	PUSH 11	
0040144F	. 6A 0D	PUSH 0D	
00401451	. 68 D10B0000	PUSH 0BD1	
00401456	. FF75 08	PUSH DWORD PTR [EBP+8]	
00401459	. B8 0E000000	MOV EAX,0E	
0040145E	. C1E0 10	SHL EAX,10	
00401461	. E8 3BFCFFFF	CALL patched1.004010A1	SendDlgItemMessageA
00401466	. 83F8 10	CMP EAX,10	
00401469	~v 73 12	JNB SHORT patched1.0040147D	password.length() > 16

Tạm thời patch cả hai câu lệnh JNB - mặc định "Nome" và "Password" ta điền vào đã thỏa điều kiện trên. Chạy từng câu lệnh một, ta đến được tới đoạn quan trọng nhất quyết định kết quả hộp thoại MessageBoxA trả về sẽ là GoodBoy hay BadBoy.

0040155D	. B8 64000000	MOV EAX,64	
00401562	. 8B4D 08	MOV ECX,DWORD PTR [EBP+8]	
00401565	. 68 90154000	PUSH patched1.00401590	
0040156A	^ E9 32FBFFFF	JMP patched1.004010A1	MessageBoxA

Để biết được kết quả trả về, ta buộc phải hiểu điều gì đang xảy ra trong Structured exception handler. Ta quan sát các câu lệnh có địa chỉ từ 4010BC tới 4010F2.

004010BC	§ 55	PUSH EBP	Structured exception handler
004010BD	. 8BEC	MOV EBP,ESP	
004010BF	. 53	PUSH EBX	
004010C0	. 8B5D 08	MOV EBX,DWORD PTR [EBP+8]	
004010C3	. 813B 050000C0	CMP DWORD PTR [EBX],C0000005	
004010C9	~v 0F85 41010000	JNZ patched1.00401210	
004010CF	. 8B5D 10	MOV EBX,DWORD PTR [EBP+10]	
004010D2	. 8383 C4000000	ADD DWORD PTR [EBX+C4],4	
004010D9	. 8B83 B0000000	MOV EAX,DWORD PTR [EBX+B0]	
004010DF	. 83F8 64	CMP EAX,64	
004010E2	~v 0F85 12010000	JNZ patched1.004011FA	
004010E8	. 81BB A8000000	CMP DWORD PTR [EBX+A8],1010100	
004010F2	~v 75 78	JNZ SHORT patched1.0040116C	BadBoy

Nhìn chung sẽ có hai trường hợp xảy ra khi MessageBoxA được gọi:

- Nếu giá trị trong EDX = 1010100, kết quả trả về sẽ là GoodBoy.
- Ngược lại, kết quả trả về sẽ là BadBoy.

Tiếp tục quan sát các câu lệnh từ địa chỉ 401469, ta có thể phần nào đoán được cách phát sinh key của target:

Đồ án 3: Crack phần mềm

00401470	. B8 08000000	MOV EAX,8	
00401475	. C1E0 10	SHL EAX,10	
00401478	. E8 24FCFFFF	CALL grinder_.004010A1	
0040147D	> 33C9	XOR ECX,ECX	
0040147F	. 8D75 DA	LEA ESI,DWORD PTR [EBP-26]	
00401482	~ EB 33	JMP SHORT grinder_.004014B7	
00401484	> 803C31 61	CMP BYTE PTR [ECX+ESI],61	
00401488	~ 72 0C	JB SHORT grinder_.00401496	< 'a'
0040148A	. 803C31 66	CMP BYTE PTR [ECX+ESI],66	
0040148E	~ 77 06	JA SHORT grinder_.00401496	> 'f'
00401490	. 802C31 20	SUB BYTE PTR [ECX+ESI],20	'a' -> 'A'
00401494	~ EB 20	JMP SHORT grinder_.004014B6	
00401496	> 803C31 30	CMP BYTE PTR [ECX+ESI],30	
0040149A	~ 72 08	JB SHORT grinder_.004014A4	< '0'
0040149C	. 803C31 39	CMP BYTE PTR [ECX+ESI],39	
004014A0	~ 77 02	JA SHORT grinder_.004014A4	> '9'
004014A2	~ EB 12	JMP SHORT grinder_.004014B6	
004014A4	> 803C31 41	CMP BYTE PTR [ECX+ESI],41	< 'A'
004014A8	~ 72 08	JB SHORT grinder_.004014B2	
004014AA	. 803C31 46	CMP BYTE PTR [ECX+ESI],46	> 'F'
004014AE	~ 77 02	JA SHORT grinder_.004014B2	
004014B0	~ EB 04	JMP SHORT grinder_.004014B6	
004014B2	> C60431 41	MOV BYTE PTR [ECX+ESI],41	
004014B6	> 41	INC ECX	i++
004014B7	> 803C31 00	CMP BYTE PTR [ECX+ESI],0	
004014BB	^ 75 C7	JNZ SHORT grinder_.00401484	
004014BD	. 33C9	XOR ECX,ECX	
004014BF	. 8D75 EB	LEA ESI,DWORD PTR [EBP-15]	
004014C2	. C745 D4 0000	MOV DWORD PTR [EBP-2C],0	
004014C9	~ EB 51	JMP SHORT grinder_.0040151C	
004014CB	> 803C31 61	CMP BYTE PTR [ECX+ESI],61	

Password sẽ được kiểm tra dựa vào Name.

Khởi tạo một biến tổng. Lần lượt duyệt qua từng ký tự trong Name:

- Nếu ký tự đó là ký tự chữ cái in thường: lấy mã cần thiết id bằng cách trừ mã ASCII của ký tự đó cho ký tự 'a', sau đó dịch phải 1 bit.
- Nếu ký tự đó là ký tự số: lấy mã cần thiết id bằng cách trừ mã ASCII của ký tự đó cho ký tự '0', sau đó dịch phải 1 bit.
- Nếu ký tự đó là ký tự chữ cái in hoa: lấy mã cần thiết id bằng cách trừ mã ASCII của ký tự đó cho ký tự 'A', sau đó dịch phải 1 bit.
- Ngược lại, gán ký tự đó bằng rỗng (hay xóa ký tự đó), chuyển sang xét ký tự kế tiếp.

Biến tổng được dịch phải 1 bit, sau đó cộng cho giá trị tại vị trí id của chuỗi Password do người dùng nhập vào, chuyển sang xét ký tự kế tiếp.

Sau khi đã có được tổng theo công thức trên từ toàn bộ các ký tự trong chuỗi Name, ta thực hiện AND tổng với giá trị 0FF. Lưu lại 8-bit thấp hơn của tổng, hoán đổi 8-bit thấp và 8-bit cao trong tổng. Với 8-bit thấp được lưu trước đó, thực hiện AND với F0. Lấy kết quả đó OR với 8-bit thấp hiện tại.

Cuối cùng kiểm tra tổng của hai Password do target tính và người dùng nhập vào có khớp nhau không để hiển thị thông báo.

2.2. Target2: Crackme#3

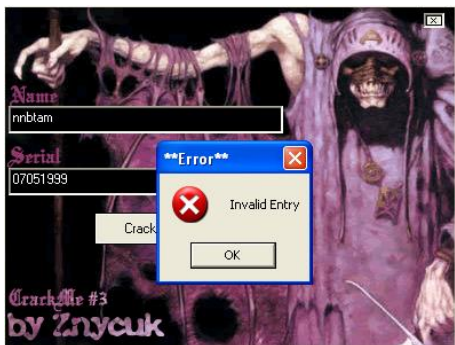
2.2.1. Phân tích

Trước hết, tiến hành chạy chương trình Crackme trên máy ảo có hệ điều hành WinXP Pro SP3.

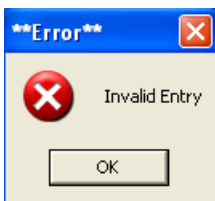
Ta thử nhập một cặp username – serial (nnbtam – 07051999) và nhấn nút Crackme. Kết quả phần mềm bị đứng!



Ta đặt giả thuyết phần mềm này chỉ tương thích với một số hệ điều hành nhất định và tiếp tục thử lại với máy ảo có hệ điều hành WinXP Pro SP2.



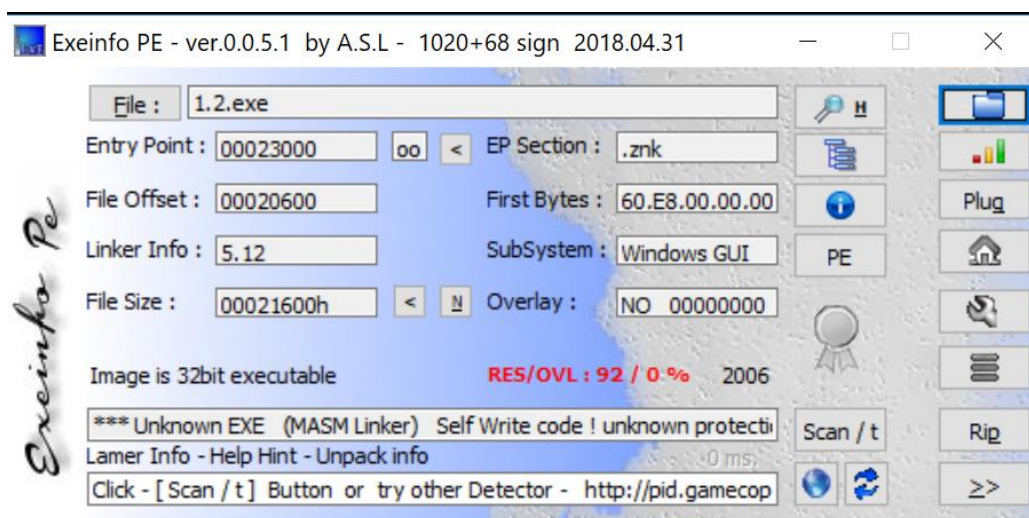
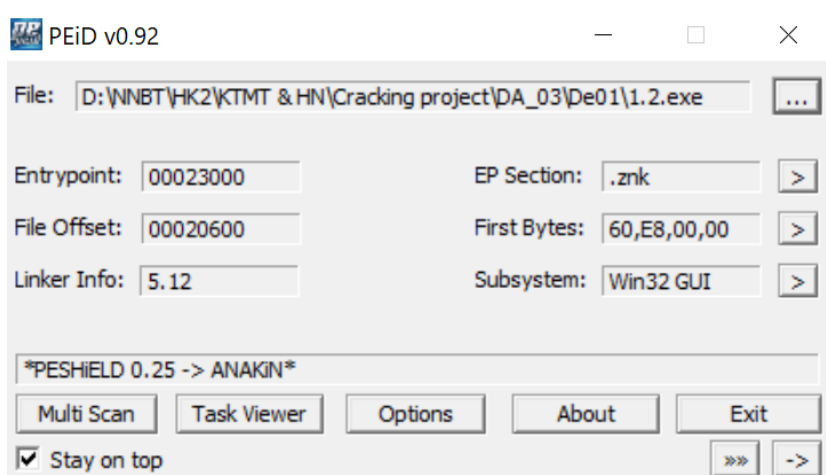
Chương trình thực thi và một hộp thoại Nag hiện ra. Vậy giả thuyết trên là đúng!



Như vậy, ta biết được Badboy của chương trình này là “Invalid Entry”.

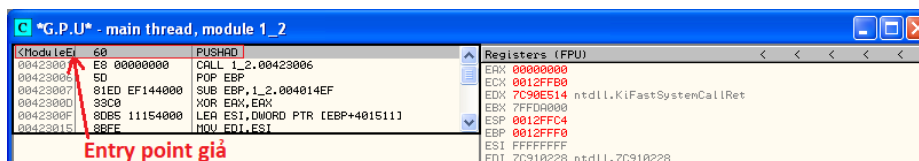
Tiếp đến, ta kiểm tra xem tập tin Crackme#3 có bị pack hay không bằng các phần mềm packer detector. Ở đây ta sử dụng 2 phần mềm detector là PEiD và PE.

Đồ án 3: Crack phần mềm



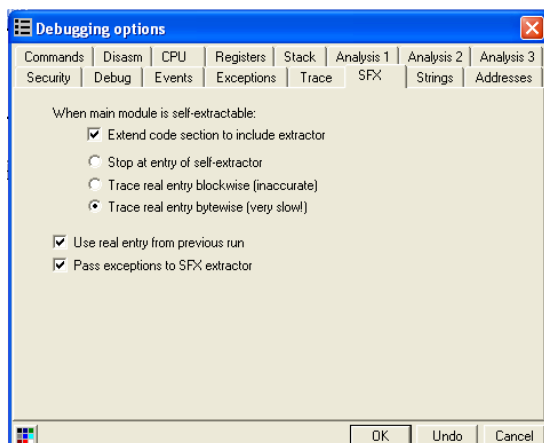
Sau khi kiểm tra tình trạng của tập tin Crackme#3, có lẽ tập tin này đã bị pack thủ công, nên ta sẽ tiến hành unpack bằng phần mềm OllyDbg.

Mở file Crackme. Ban đầu chương trình vẫn hiện entry point giả.

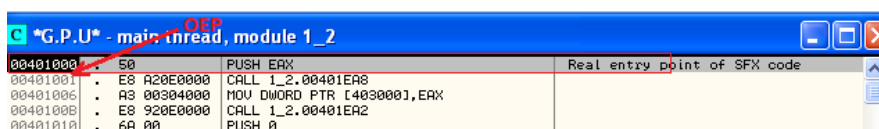


Vào bảng chọn Options → Debugging options → SFX và chọn chức năng “Trace real entry bitwise (very slow)” → OK

Đồ án 3: Crack phần mềm



OllyDbg sẽ tìm đến OEP của chương trình.



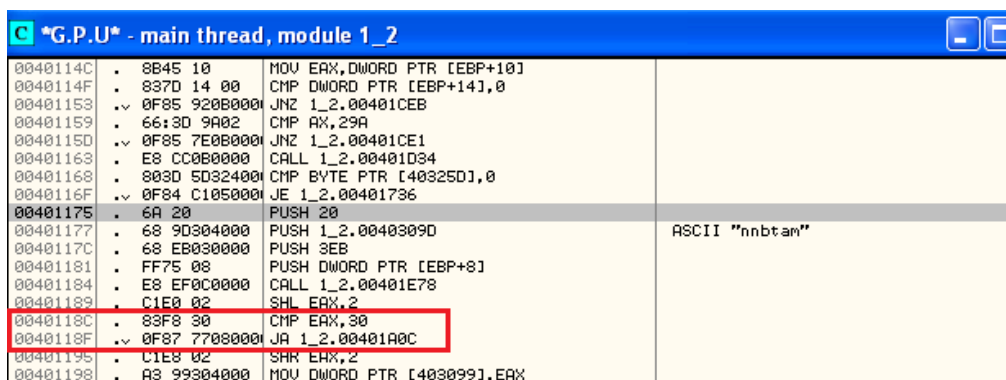
Ta tiến hành tìm quy luật để chương trình tự phát sinh serial dựa trên username mà người dùng nhập vào.

Đặt breakpoint ở OEP và tiến hành debug bằng f8. Khi nhập vào bộ username – serial (nnbtam – 07051999) → Chọn search for → All referenced text strings, ta tìm được 2 vị trí có xuất hiện chuỗi username và serial.

Address	Disassembly	Text string
0040103C	CALL 1_2.00401E9C	(Initial CPU selection)
00401177	PUSH 1_2.0040309D	ASCII "nnbtam"
0040119F	PUSH 1_2.004030BD	ASCII "07051999"

Address	Disassembly	Text string
00401346	ASCII "wrong serial",0	
00401379	ASCII "registered",0	
00401412	PUSH 1_2.0040309D	ASCII "nnbtam"
0040144E	PUSH 1_2.004030BD	ASCII "07051999"
00401460	PUSH 1_2.00403068	ASCII "yep yep !"

Click chuột để đến địa chỉ 00401177, ta thấy như sau:



Chương trình thực hiện push username và các đối số khác sau đó thực hiện gọi hàm ở vùng nhớ 00401E78. Ta đặt giả thuyết hàm này để kiểm tra điều kiện của username, vì sau khi gọi

Đề án 3: Crack phần mềm

hàm có sự so sánh giá trị ở thanh ghi EAX với số 30 (hệ hex) tương đương 48 (hệ thập phân) kèm theo lệnh nhảy JA (nhảy nếu giá trị ở EAX lớn hơn)

Click chuột nhảy đến địa chỉ 00401A0C.

004019EE	. 58	POP EAX	
004019EF	.^ E9 A6FEFFFF	JMP 1_2.0040189A	
004019F4	. 99	DB 99	
004019F5	. 94	DB 94	
004019F6	. 08	DB 08	
004019F7	. 3D FF03498D	CMP EAX,8D4903FF	
004019FC	. 68 1A304000	PUSH 1_2.0040301A	ASCII "Sorry, it seems ur machine i
00401A01	. 8F05 90304000	POP DWORD PTR [403090]	1_2.0040305H
00401A02	. E9 01020000	JMP 1_2.00401C0D	
00401A0C	> 68 10304000	PUSH 1_2.00403010	ASCII "***Error**"
00401A11	. 8F05 8C304000	POP DWORD PTR [40308C]	1_2.00403010
00401A17	.v E9 59020000	JMP 1_2.00401C75	

Kết quả kiểm tra là “Error” thay vì “Invalid Entry” như chúng ta đã thấy. Vậy hàm kiểm tra điều kiện ban này có thể là so sánh độ dài của username với số 48 (vì so sánh với số mà liên quan đến chuỗi thì có khả năng là độ dài). Ở đây ta lại phát hiện thêm một chuỗi “Sorry, it seems ur machine is not compatible with this crackme”.

Điều này phù hợp với kết luận chương trình chỉ phù hợp với một số máy nhất định.

```
00403010=1_2.00403010 (ASCII "***Error**")
Jumps from 0040118F, 004011B9, 0040145A
```

Từ chuỗi “Error”, ta thấy được chuỗi này được gọi từ các vùng nhớ 0040118F (vùng nhớ kiểm tra điều kiện username), 004011B9 (vùng nhớ kiểm tra điều kiện serial) và 0040145A (tạm thời ta chưa xét vùng nhớ này).

Quay lại hàm kiểm tra độ dài chuỗi username (như ta giả thuyết), từ hàm SHL EAX, 2; CMP EAX, 30 và JA, ta thấy nếu username dài hơn $48 / 4 = 12$ thì bị lỗi. Sau hàm nhảy JA khôi phục chiều dài username.

00401195	. C1E8 02	SHR EAX,2	
00401198	. A3 99304000	MOV DWORD PTR [403099],EAX	
0040119D	. 6A 20	PUSH 20	
0040119F	. 68 BD304000	PUSH 1_2.004030BD	ASCII "07051999"
004011A4	. 68 EA030000	PUSH 3EA	
004011A9	. FF75 08	PUSH DWORD PTR [EBP+8]	
004011AC	. E8 C70C0000	CALL 1_2.00401E78	
004011B1	. C1E0 03	SHL EAX,3	
004011B4	. 3D E0000000	CMP EAX,0E0	
004011B9	.v 0F82 4D080000	JB 1_2.00401A0C	
004011BF	.v E9 6E010000	JMP 1_2.00401332	

Tương tự với chuỗi serial, ta so sánh độ dài serial với E0 (hệ hex) tương ứng với 224 (hệ thập phân). Hàm nhảy JB là nhảy nếu nhỏ hơn, nên nếu độ dài serial nhỏ hơn $224 / 8 = 28$ thì bị lỗi.

Vậy ta tìm được điều kiện: **chuỗi username dài không quá 12 ký tự, chuỗi serial dài ít nhất 28 ký tự.**

Ta tiếp tục xem vị trí thứ 2 xuất hiện chuỗi username và serial, vùng nhớ 0040145A. Vị trí đầu tiên xuất hiện hai chuỗi trên là nhằm kiểm tra tính độ dài hợp lệ, vậy có thể vị trí thứ 2 là nơi kiểm tra tính hợp lệ để mở khóa.

Đề án 3: Crack phần mềm

00401402	> 68 2F324000	PUSH 1_2.0040322F	
00401407	. 68 29324000	PUSH 1_2.00403229	
0040140C	. FF35 99304000	PUSH DWORD PTR [403099]	
00401412	. 68 90304000	PUSH 1_2.0040309D	ASCII "nnbta"
00401417	. E8 F1090000	CALL 1_2.00401E0D	
0040141C	. 68 35324000	PUSH 1_2.00403235	
00401421	. 68 2F324000	PUSH 1_2.0040322F	
00401426	. 68 29324000	PUSH 1_2.00403229	
0040142B	. E8 16090000	CALL 1_2.00401E46	
00401430	. 68 41324000	PUSH 1_2.00403241	Arg2 = 00403241
00401435	. 68 35324000	PUSH 1_2.00403235	Arg1 = 00403235
0040143A	. E8 79090000	CALL 1_2.00401D88	1_2.00401D88
0040143F	. 68 41324000	PUSH 1_2.00403241	
00401444	. E8 56090000	CALL 1_2.00401D9F	
00401449	. 68 41324000	PUSH 1_2.00403241	
0040144E	. 68 BD304000	PUSH 1_2.004030BD	ASCII "07051999"
00401453	. E8 62090000	CALL 1_2.00401EBA	
00401458	. 85C0	TEST EAX,EAX	
0040145A	. 0F85 AC050000	JNZ 1_2.00401A0C	
00401460	. 68 68304000	PUSH 1_2.00403068	ASCII "yep yep *"
00401465	. 8F05 8C304000	POP DWORD PTR [40308C]	1_2.00403010
0040146B	. E9 E5000000	JMP 1_2.00401555	
00401470	. 2077 08	AND BYTE PTR [EDI+8],DH	
00401473	. 3D E0FBC0A1	CMP EAX,A10FB0A1	
00401478	. EB 0B	JMP SHORT 1_2.00401485	
0040147A	. 72 65 67 69	ASCII "registered",0	
00401485	. 0FAFC0	INUL EAX,EAX	

Ta sẽ lần lượt kiểm tra từng block gọi hàm.

Click vào lệnh call ở vùng nhớ 00401412, gọi tới vùng nhớ 00401E0D:

Hàm này gồm 4 tham số, các giá trị nằm ở vùng nhớ 40322F, 403229, 403099 (chiều dài username) và 40309D (username). Do thứ tự push argument ngược với C++ nên ta hiểu thứ tự tham số là function (username, len(username), <403229>, <40322F>).

00401E0D	§ 55	PUSH EBP	
00401E0E	. 8BEC	MOV EBP,ESP	
00401E10	. 33C0	XOR EAX,EAX	
00401E12	. 33D2	XOR EDI,EDI	
00401E14	. 33C9	XOR ECX,ECX	
00401E16	. 8B75 10	MOV ESI,DWORD PTR [EBP+10]	
00401E19	. 8B5D 08	MOV EBX,DWORD PTR [EBP+8]	
00401E1C	. 8B7D 14	MOV EDI,DWORD PTR [EBP+14]	
00401E1F	> 90431	MOV AL,BYTE PTR [ECX+ESI]	
00401E22	. 8A13	MOV DL,BYTE PTR [EBX]	
00401E24	. 32C2	XOR AL,DL	
00401E26	. 8B439	MOV BYTE PTR [ECX+EDI],AL	
00401E29	. FEC1	INC CL	
00401E2B	. 80F9 05	CMPL CL,5	
00401E2E	. 76 04	JBE SHORT 1_2.00401E34	
00401E30	. B1 00	MOV CL,0	
00401E32	. 8BF7	MOV ESI,EDI	
00401E34	> 8B45 0C	MOV EAX,DWORD PTR [EBP+C]	
00401E37	. 48	DEC EAX	
00401E38	. 8945 0C	MOV DWORD PTR [EBP+C],EAX	
00401E3B	. 85C0	TEST EAX,EAX	
00401E3D	. 74 03	JE SHORT 1_2.00401E42	
00401E3F	. 43	INC EBX	
00401E40	. EB DD	JMP SHORT 1_2.00401E1F	
00401E42	> C9	LEAVE	
00401E43	. C2 1000	RET 10	

Từ hàm ở vùng nhớ 00401E0D, ta thấy:

EDI mang giá trị vùng nhớ 40322F, ESI mang giá trị vùng nhớ 403229, EBX là chuỗi username. Sau hàm này, username được xor với vùng nhớ 403229.

Ta tiếp tục thử debug lệnh call ở vùng nhớ 00401444, và hàm nhảy đến vùng nhớ 00401D9F:

00401D9F	§ 55	PUSH EBP	
00401DA0	. 8BEC	MOV EBP,ESP	
00401DA2	. 8B7D 08	MOV EDI,DWORD PTR [EBP+8]	
00401DA5	. 83C7 18	ADD EDI,18	
00401DA8	. BE 94304000	MOV ESI,1_2.00403094	ASCII "-ZNK"
00401DAD	. B9 05000000	MOV ECX,5	
00401DB2	. F3:A4	REP MOVSB BYTE PTR ES:[EDI],BYTE PTR [ESI]	
00401DB4	. C9	LEAVE	
00401DB5	. C2 0400	RET 4	

Hàm này thực hiện nối chuỗi "-ZNK" vào đối số của hàm. Ta nhận thấy vùng nhớ 00403241 vừa làm đối số cho hàm nối chuỗi "-ZNK" lại tiếp tục làm đối số cho hàm liên quan đến chuỗi serial người dùng nhập.

Đồ án 3: Crack phần mềm

00401430	. 68 41324000	PUSH 1_2.00403241	Arg2 = 00403241 Arg1 = 00403235 1_2.00401DB8
00401435	. 68 35324000	PUSH 1_2.00403235	
0040143A	. E8 79090000	CALL 1_2.00401DB8	
0040143F	. 68 41324000	PUSH 1_2.00403241	ASCII "07051999"
00401444	. E8 56090000	CALL 1_2.00401D9F	
00401449	. 68 41324000	PUSH 1_2.00403241	
0040144E	. 68 BD304000	PUSH 1_2.004030BD	
00401453	. E8 620A0000	CALL 1_2.00401EBA	

Ta dự đoán vùng nhớ 00403241 chính là chuỗi serial được phát sinh và hàm 00401EBA là hàm so sánh serial phát sinh từ username và serial nhập từ bàn phím.

Dễ thấy trước đó vùng nhớ 00403241 lại tham gia một hàm khác. Rất có thể 00401DB8 là hàm phát sinh serial từ username người dùng nhập, và 00403235 là vùng nhớ chứa username sau khi xor.

Hàm phát sinh serial có tham số lần lượt là username và serial được phát sinh.

00401DB8	55	PUSH EBP
00401DB9	. 8BEC	MOV EBP,ESP
00401DBB	. 83C4 FC	ADD ESP,-4
00401DBE	. 8B75 08	MOV ESI,DWORD PTR [EBP+8]
00401DC1	. 8B7D 0C	MOV EDI,DWORD PTR [EBP+C]
00401DC4	. C745 FC 0600	MOV DWORD PTR [EBP-4],6
00401DCB	> 66:AD	LODS WORD PTR [ESI]
00401DCD	. 66:C1C8 08	ROR AX,8
00401DD1	. 33D2	XOR EDX,EDX
00401DD3	. 66:8BD8	MOV BX,AX
00401DD6	. B9 04000000	MOV ECX,4
00401DD8	> 24 0F	AND AL,0F
00401DD0	. 3C 09	CMP AL,9
00401DDF	~ 7E 02	JLE SHORT 1_2.00401DE3
00401DE1	. 04 07	ADD AL,7
00401DE3	> 04 30	ADD AL,30
00401DE5	. 8AD0	MOV DL,AL
00401DE7	. 83F9 01	CMP ECX,1
00401DEA	~ 74 03	JE SHORT 1_2.00401DEF
00401DEC	. C1C2 08	ROL EDX,8
00401DEF	> 66:C1EB 04	SHR BX,4
00401DF3	. 8AC3	MOV AL,BL
00401DF5	^ E2 E4	LOOPD SHORT 1_2.00401DD8
00401DF7	. 8917	MOV DWORD PTR [EDI],EDX
00401DF9	. 8B45 FC	MOV EAX,DWORD PTR [EBP-4]
00401DFC	. 48	DEC EAX
00401DFD	~ 74 0A	JE SHORT 1_2.00401E09
00401DFF	. 8945 FC	MOV DWORD PTR [EBP-4],EAX
00401E02	. 33C0	XOR EAX,EAX
00401E04	. 83C7 04	ADD EDI,4
00401E07	^ EB C2	JMP SHORT 1_2.00401DCB
00401E09	> C9	LEAVE
00401E0A	. C2 0000	RET 8

00401412	. 68 9D304000	PUSH 1_2.0040309D	ASCII "nnbtam"
00401417	. E8 F1090000	CALL 1_2.00401E0D	
0040141C	. 68 35324000	PUSH 1_2.00403235	
00401421	. 68 2F324000	PUSH 1_2.0040322F	
00401426	. 68 29324000	PUSH 1_2.00403229	
0040142B	. E8 160A0000	CALL 1_2.00401E46	

00403235 và 0040322F là 2 vùng nhớ làm đối số trong hàm xor username, đến lượt này lại tiếp tục tham gia vào hàm 00401E46. Thứ tự đối số là 00403229, 0040322F và username.

00401E46	55	PUSH EBP
00401E47	. 8BEC	MOV EBP,ESP
00401E49	. 33C9	XOR ECX,ECX
00401E4B	. 33C0	XOR EAX,EAX
00401E4D	. 8B75 08	MOV ESI,DWORD PTR [EBP+8]
00401E50	. 8B5D 0C	MOV EBX,DWORD PTR [EBP+C]
00401E53	. 8B7D 10	MOV EDI,DWORD PTR [EBP+10]
00401E56	> 8A0431	MOV AL,BYTE PTR [ECX+ESI]
00401E59	. 8807	MOV BYTE PTR [EDI],AL
00401E5B	. 47	INC EDI
00401E5C	. 8A0419	MOV AL,BYTE PTR [ECX+EBX]
00401E5F	. 8807	MOV BYTE PTR [EDI],AL
00401E61	. 47	INC EDI
00401E62	. 41	INC ECX
00401E63	. 83F9 06	CMP ECX,6
00401E66	^ 72 EE	JB SHORT 1_2.00401E56
00401E68	. C9	LEAVE
00401E69	. C2 0C00	RET 0C

Đồ án 3: Crack phần mềm

2.2.1. Thuật toán mã hóa

2.3. Target3: Beam

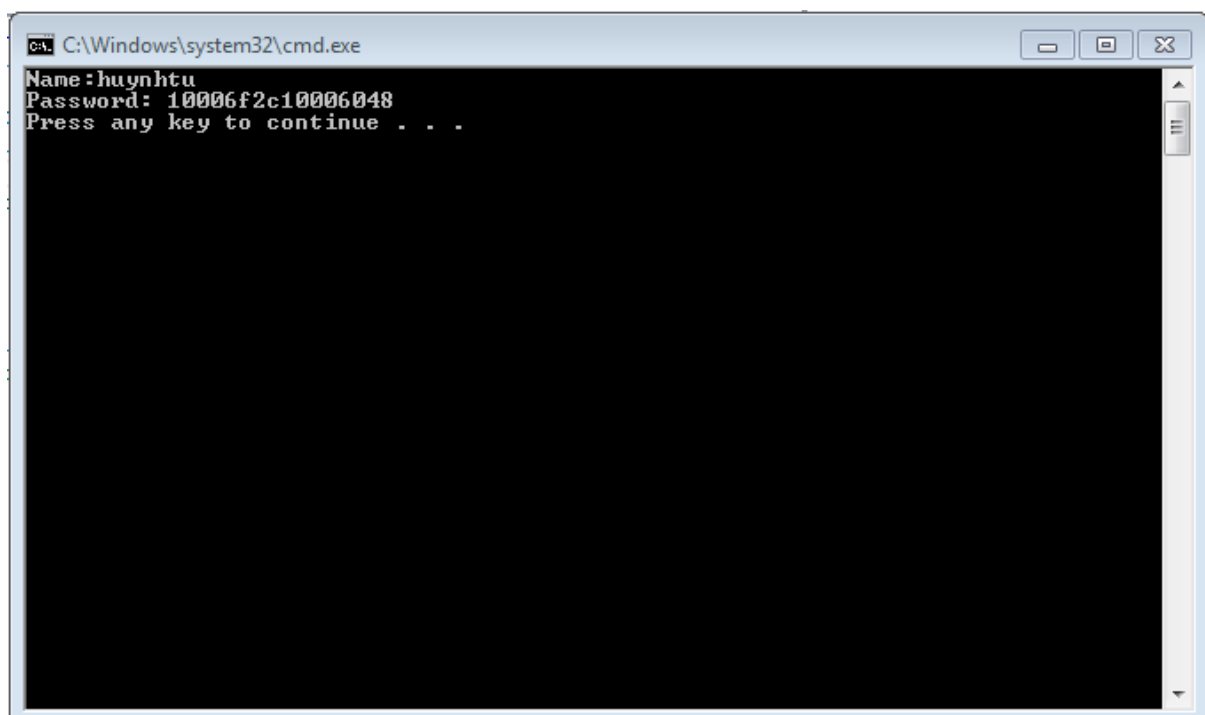
2.3.1. Phân tích

2.3.1. Thuật toán mã hóa

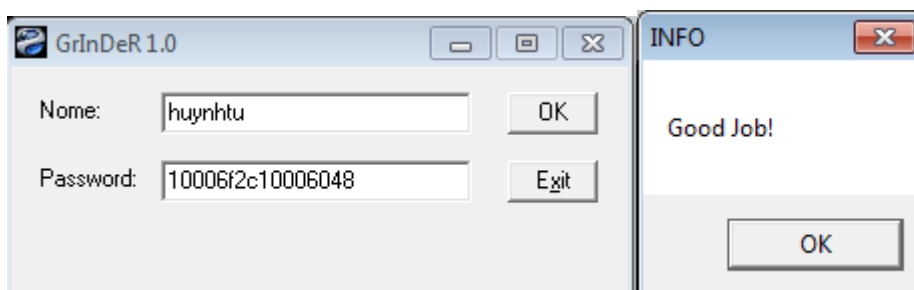
3. Thử nghiệm

3.1. Target 1

Ta chạy file 1_1.exe để lấy password từ của 1 name nhập vào từ bàn phím.



Sau đó nhập name và password đã lấy được vào chương trình cần crack.



Cuối cùng nhấn OK và chương trình hiển thị thông báo “Good Job!”. Như vậy, ta đã crack phần mềm thành công!!!

4. Tham khảo

[1] X-Treem's Grinder 1.0: Solution by red477.

[2] znycuk's crackMe#3: Solution by aallove.

[3] HMX0101's Beam: Solution by Crosys.