

# چگونگی کراک کردن (خودآموز)

## حفاظت (Protection) (۱)

برخی مشکلات با وقفه‌های اینتل:

دستور INT (یا وقفه) منشاء قسمت اعظم سازگاری در معماری PC میباشد. چون توانایی گرفتن و تنظیم بردار وقفه (interrupt vector) به این مفهوم است که سرویسهای سیستم، تا حد زیادی قابل تعمیم، قابل جایگزینی و تحریک کننده می‌باشند.

اما دستور INT (یا وقفه) هنوز با دو مورد زیر سازگاری ندارد:

- یک برنامه کنترل کننده وقفه (دستگیره وقفه HANDLER)، شماره وقفه‌ای که به او روی آورده نمی‌داند.

- دستور INT خودش منتظر یک اپراند (OPERAND) میباشد، یعنی شما نمیتوانید برای صدا زدن وقفه ۲۱ ابتدا ۲۱ را در رجیستر یا ثبات X A بریزید (mov ax, x21) و سپس وقفه ۲۱ را صدا بزنید، شما باید بنویسید: (INT x21)

کراک کردن واقعاً برای ما خیلی خوبست... اما متأسفانه کمپایلر بسیاری از زبانهای سطح بالا، به جای انجام یک وقفه واقعی، آنها را با دستورات FAR CALL, PUSHF کمپایل میکند. روش دیگر، پوش کردن آدرس هندلر (handler) به داخل استک (stack) و سپس انجام RETF روی آن میباشد.

برخی طرحهای حفاظتی منجر به فراخوانیهای متفاوتی از وقفهها میشود:

(۱) camuflag کردن کد

(۲) قرار گرفتن در وضعیتی که دستور وقفه آنها را تغییر دهد. (۳) تکرار روتین وقفه داخل کد. اینها کاملاً شبیه طرح حفاظتی دسترسی به دیسک میباشد که از وقفه 13 (INT 13) یا همان وقفه مربوط به دیسک استفاده میکرد که در درس ۵ بیان شد.

### یک اسمبلر پایه

برای فهم طرحهای حفاظتی و غلبه بر آنها، شما لازم است اطلاعاتی در مورد اسمبلر و کد زبان ماشین یادگیرید. فراگیری آن خوب و رایگان است. مثل ویروسها که حقههایی سخت و غیر قابل نفوذ از کدهای اسمبلر میباشند. شما میتوانید کد منبع اکثر ویروسها را روی اینترنت پیدا کنید. اینهم از شانس همه کسانی است که میخواهند هکر شوند.

میلیونها سطر از این کدها روی اینترنت وجود دارد که باید مثل ماهی آنها را صید کنید و مطالعه کنید. همانطور که همه شما میدانید بهتر است که کراک (crack) کنید. من با بیان این مطالب در این خودآموز، سر خودم کلاه میگذارم. حال اجازه دهید شروع کنیم:

### Strings یا رشتهها

دستوراتی که با رشتهها کار میکنند خیلی قوی هستند. (و کاربرد زیادی در طرح حفاظت کلمه عبور دارند) همه آنها دارای ویژگیهای زیر میباشند:

(۱) منبع data در DS:SI قرار میگیرد. (۲) مقصد data در es:di قرار میگیرد. (۳) و رجیسترها (یا ثباتهای) DI و SI برای OR یا and باهم جمع یا از هم کم میشوند. بنابراین، این عملیات میتواند تکرار شود.

### Jumps یا پرسشها

|           |   |                                             |
|-----------|---|---------------------------------------------|
| JZ ero    | : | که گفته میشود پرسش کن اگر صفر است.          |
| JNZ ero   | : | همانطور که گفته میشود پرسش کن اگر صفر نیست. |
| JG reater | : | به معنی اگر تفاوت SIGNED مثبت است .         |
| JABove    | : | به معنی اگر تفاوت unsigned مثبت است.        |
| JL ess    | : | به معنی اگر تفاوت SIGNED منفی است.          |
| JB elow   | : | به معنی اگر تفاوت UNSIGNED منفی است.        |
| JC arrey  | : | مثل JB اسمبل میشود.                         |

### کراک(نفوذ) کردن به برنامه‌هایی که با کلمه رمز حفاظت شده

به درس یک مراجعه کنید تا متوجه شوید که چرا ما از بازیها بعنوان برنامه‌های کاربردی تجاری، برای آموزش استفاده می‌کنیم. چون آنها هم از همان روشهای حفاظتی (یا BBS و server) استفاده میکنند. هرچند که نفوذ کردن به داخل برنامه‌های کوچک، خیلی زمان نمی برد. یک برنامه کامل، باید از کپی کردن محافظت شود ولی در مورد یک برنامه بزرگ همین که جلو کپی کردن ما گرفته شود تا حد زیادی برای تحریک کردن ما کافی است.

معمولاً در شروع یک برنامه، یک صفحه آزار دهنده یا nag screen ظاهر یمشود و از کاربر کلمه‌ای را میخواهد که او میتواند آن را در هر جایی از داخل دفترچه راهنما پیدا کند، و چیزهایی شبیه این میبیند : «لطفاً اولین حرف سطر سوم از قسمت 3.3.2 را تایپ کنید» . در اغلب موارد به منظور

جلوگیری از اشتباه کاربر، برنامه اولین حرف کلمه عبور را تعیین میکند و کاربر باید تنها حروف باقیمانده را پر کند.

برخی مثالها، از برخی کراکها ( cracks نفودها ) :

UMS (Universal Military Simulator) version 1

by Dr Ezra SIDRAN

(c) 1987 Intergalactic Development

European Union: Rainbird Software

United States: Firebird Software

برنامه‌های EGA قدیمی، یکی از اولین برنامه‌هایی هستند که من در جوانی به داخل آنها نفوذ کردم (کراک کردم) آنها خیلی جالب هستند و طرحهای حفاظتی بنیادی‌ای را به کار می‌گیرند. تا امروزه (ژوئن ۱۹۹۶) بیش از ۸۰ درصد طرحهای حفاظتی، همانهایی هستند که در گذشته استفاده می‌شدند. یک nag screen (یا صفحه آزار دهنده) در ابتدای برنامه ناگهان ظاهر میشود و منتظر جواب شما میماند و فقط ctrl+C میتواند شما را نجات دهد و دوباره به dos برگرداند. که در طرحهای حفاظتی قدیمی از آن استفاده می‌شده. در طرحهای جدیدتر، در برخی موارد تا سه بار یا فقط یکبار به شما اجازه میدهد که امتحان کنید و اگر موفق نشدید، شما را به سیستم عامل برمیگرداند. در UMS، هیچ راهنمایی به شما نمیشود و حتی حرف اول کلمه عبور نمایش داده نمیشود.

پروسیجر Cracking برای برنامه‌هایی که با کلمه رمز حفاظت میشوند، در رأس همه قرار دارد که برای پیدا کردن محل حروف ذخیره شده، بکار میرود. بنابراین شما باید حافظه را جستجو کنید و جایی را که

برنامه‌ها در حافظه جایگزین میشوند پیدا کنید. با ذخیره‌سازی این ناحیه از حافظه و انجام یکسری عملیات مقایسه، شما میتوانید کلمه عبور تایپ شده را پیدا کنید.

در حالت UMS، همانطور که کلمه را تایپ میکند، آن کلمه در محل‌های مختلف حافظه، یعنی جاییکه برنامه قرار گرفته، مستقر میشود. میتواند برای پیدا کردن آن بردارهای hook شده را جستجو کنید.

در محل مورد نیاز hook شده VECS 00,02,22

در XXXX شده VECS 34-3D

در hook, XXXX:C CA شده VECS 3E

حالا اجازه بدهید دنبال کلمه‌ای بگردیم که در nag screen یا همان پنجره‌ایکه در آغاز برنامه ظاهر میشود، وارد شده در UMS در آدرس 3E+7656 خواهد بود و محتویات آن پنجره، بلافاصله وارد آن می‌شود. و شما میتوانید با کنار هم قرار دادن آنها، کلمه رمز را پیدا کنید. البته این یک روش قدیمی است و ما حالا سایر روشهای پیشرفته‌تر در دیگر برنامه‌ها را برایتان خواهیم گفت. اجازه دهید کمی ریشه‌ای‌تر بموضوع نگاه کنیم و سایر روشهای کراک (نفوذ crack) کردن را باهم مقایسه کنیم.

برنامه‌های حفاظت شده با کلمه رمز (و همچنین روتینهای دسترسی به آنها برای سرویس دهنده server و BBS) نقاط سست و بی اساس زیادی دارند. مشهورترین آنها (که وقتی شما یک کراکر (cracker) سطح بالا شدید، آن را متوجه خواهید شد) به این صورت کار میکند که شما باید کلمه عبور کاربر را با اصل آن مقایسه کنید. پس نیازی به دزدیدن نخواهید داشت. فقط کافی است ناحیه انعکاس آن در محل‌های حافظه را برای مقایسه پیدا کنید. و درست‌ترین محل را برای نفوذ کردن بوسیله مکانیسمهای مقایسه پیدا کنید حتی اگر کلمه رمز را اشتباه حدس بزنید.

مکانیسمهای مقایسه در UMS میتواند مجموعه‌ای از (نقاط نفوذ) break point روی محدوده‌ای از حافظه باشد که سه نقطه‌ایکه کلمه رمز در آن ذخیره میشود را تحت پوشش قرار میدهد (و شما آن را از طریق جستجو مقایسه پیدا خواهید کرد).

ES: 0F8E در اینجا شما یک کپی از کلمه رمزی که مورد نظر برنامه می باشد پیدا می کنید.

ES: 0F5C در اینجا شما یک کپی از کلمه رمزی که کاربر تایپ کرده پیدا می کنید.

INT-3E-hook-address+7656 در اینجا کلیه کلمات رمز ممکن تعریف شده را پیدا میکنید.

حال ببینیم طرح حفاظتی چطور عمل می کند:

MOV CX, FFFF بزرگترین مقدار ممکن را در رجیستر CX می ریزیم:

REPZ CASBS scan ED: DI کلمه رمز کاربر را جستجو میکنیم

NOT CX حالا CX محتوی تعداد کاراکترهایی که کاربر تایپ کرده می باشد

MOV DI, SI افست کلمه رمز واقعی را در DI میریزیم

LDS SI, [BP+0A] افست کلمه رمز کاربر را در SI میریزیم

REPZ CMPSB (توضیح در پایین)

DS: SI را با ES:DI (یعنی کلمه رمز واقعی با آنچه که کاربر تایپ کرده) مقایسه می شود، پس از آن

CX=0 یا هر کاراکتری دیگری، میشود. یا اولین حرفی که بعداً بیاید. روی سطح پایه بمانیم ... شما به

کدی که در ادامه برای جستجوی CMPSB آمده نگاهی بیندازید. این همان حالتی است که در قدیم

اکثراً استفاده می شد. بخاطر داشته باشید که CMPSB اولین کاراکتر مورد اختلاف را پیدا میکند یا

اینکه بر اساس تعداد کاراکترهای کاربر، نتیجه گیری میکند.

MOV AL, [SI-01] کاراکترهای کلمه عبور کاربر در AL بارگذاری میشود.

SUB AL, ES:[DI-01] کاراکترهای کلمه عبور واقعی از AL کم میشود.

CBM Flag مربوط به صفر شدن نتیجه، چک می شود

اگر حاصل True یعنی کاراکترها باهم cbw مطابق بوده اند و کاربر کلمه عبور را صحیح تایپ کرده.

خب حالا اجازه دهید به JZ بعدی نگاه بیندازیم (آن کد ۷۴ است)

CS; IP 740 D موقعیت نامناسب JZ

صبر کنید، کمی ادامه می‌دهیم یک چک دیگر می‌کنیم (در اغلب موارد شما دوبار DI را چک کنید).

CS: ID 7590 JZN

کراک کردن خیلی آسان است. فقط لازم است ۷۵ را به ۷۴ تغییر دهید و ۷۴ را به ۷۵.

همچنین JZ را به JNZ و بالعکس تبدیل کنید. حالا شما موفق خواهید شد و در مورد آنچه نوشته‌اید

مشکلی ندارید. مگر اینکه شما واقعاً کلمه عبور را حدس زده باشید.

حالا اجازه دهید به سرعت کراک کنیم

-----  
CRACKING UMS.EXE (by +ORC, January 1996)

```
ren ums.exe ums.ded
```

```
symdeb ums.ded
```

```
- s (cs+0000):0 Lffff 74 0D 1E B8 C2 3F
```

```
(nothing)
```

```
- s (cs+1000):0 Lffff 74 0D 1E B8 C2 3F
```

```
(nothing)
```

```
- s (cs+2000):0 lffff 74 0D 1E B8 C2 3F
```

```
xxxx:yyyy (this is the answer of the debugger)
```

```
- e xxxx:yyyy 75
```

```
- e xxxx:yyyy+17 74
```

```
- w
```

```
- q
```

```
ren ums.ded ums.exe
```

-----  
در degug/symdeb کراک فوق، ما از بایتها بعنوان رشته جستجو استفاده می‌کنیم و در ادامه بلافاصله

اولین JZ را می‌بینیم. من میدانم ... ما آنها را در [saft-ice] دیدیم میتوانیم آنها را همان جا تغییر دهیم

و اصلاح کنیم. اما من می‌خواهم به شما یاد دهم که در صورت نبودن [soft-ice] چکار کنید.

توجه داشته باشید که طول برنامه x421A0 بایت است. بنابراین دارای ثباتهای BX=4 به همراه CX = 31A0 خواهد بود. .. در نتیجه من باید همه سکتورها را امتحان کنم. (حتی اگر بدانم که snap در سکتور CS+2000 رخ داده). این یک تمرین خوب است و اگر شما رشته را در اولین سکتور پیدا نکردید، باید سکتورهای بعدی را جستجو کنید. تا اینکه آن را بیابید. در بیشتر برنامه‌ها شما مجبور هستید که طرح را چند بار امتحان کنید.

(که بعداً در مورد آن بیشتر توضیح خواهیم داد. این راه قدیمی کراک کردن است [UMS.exe]. اجازه بدهید ادامه دهیم. امروزه طرحهای حفاظتی کلمه عبور استادانه‌تر و پیشرفته‌تر می‌باشند.

---

برنامه LIGHT SPEED از Microprose (ما در اینجا از نسخه 461.01 آن برای کراک استفاده می‌کنیم).

---

-

این برنامه در سال ۱۹۹۰ منتشر شد و نسبت به طرحهای قبلی خیلی مدرنتر و پیشرفته‌تر عمل میکند. شما متوجه این تفاوتها طی دسترسی به سرور (server) دور (remote) خواهید شد. (حقیقتاً همین چیزها هستند که آن را جالب میکند).

طبق معمول اجازه دهید بردارهایمان تمرین و مقایسه‌ها را شروع کنیم. بردارهای hook شده: 00 , 08 , 1B , 22 , 23 : که هیچ چیز خاصی در آنها نیست. مقایسه حافظه اصلی، یعنی همان که شما کلمه عبور را در آن تایپ می‌کنید، خیلی طول میکشد... درست مثل اینکه شما در حال تمرین هستید.

خب حالا چه؟

بنشینید و یک برودکای ماریتنی بنویسید (من فقط از moskovskaja می‌ترسم) و راحت باشید.

تصویر برنامه‌تان را در حافظه بگیرید، و از اینجا شروع کنید. ABCDE را بعنوان کلمه رمز وارد کنید، راحت بنشینید و ودکای مارتینی را مزمه کنید. میدانید که کد حرف A، X41، کد حرف B، X42، کد حرف C، X43 و .... میباشد. و در مقایسه حروف کاربرد دارد و شما تنها با استفاده از این مقادیر خواهید توانست عمل مقایسه را انجام دهید. پس روی آن خیلی دقت کنید.

شما بزودی درمی‌یابید که برنامه LIGHTSPEED دارای یک آدرس مطلق و یک آدرس نسبی می‌باشد. بعنوان مثال در کامپیوتر من در آدرس مطلق 404307 و آدرس نسبی BE:F857 30 یا 4043:0007 قرار دارد که کاراکترهایی که شما تایپ کرده‌اید در آنها قرار می‌گیرند و میتوانید آنها را چاپ کنید.

|      |      |      |      |      |
|------|------|------|------|------|
| F856 | F857 | F858 | F859 | ...  |
|      |      |      |      | F855 |

دومین حرفیکه تایپ کرده‌اید. اولین حرفیکه تایپ کرده‌اید. اولین حرف حاضر 3E 41

با بررسی همان پرینتها، متوجه میشوید که آدرس مطلق 30C64 محتوی آخرین کاراکترهایی است که شما تایپ کرده‌اید. خط کد نسبی بصورت زیر است:

کد حرف تایپ شده SS:F83E=00+ جاییکه CS: 0097 MOV AX, [BP-08]

حالا اجرای برنامه در این محلها را متوقف کنید و ببینید که کاراکتر تایپ شده کجا میرود. برای مثال دستور زیر را در نظر بگیرید: CS:009A MOV [BX],AX مفهوم آن، این است که کد حرف تایپ شده شما در آدرسی که رجیستر BX به آن اشاره می‌کند، یعنی F85A، قرار می‌گیرد. غیر از این چه میتواند باشد؟

بهتر است بهانه نگیرید و آن را قبول کنید. یک دستور مثل CM P AX, 000D را در نظر بگیرید که زمانی رخ میدهد که کاربر کلید enter را بزند. که با زدن کلید enter به x1D تبدیل میشود و باید یک

جایی همین نزدیکیها باشد و شما را به سطر CS: 0073 CMP AX, 000D

3D0D00 میرساند. حالا راه نفوذ باز شده ولی شما نیازی به همه اینها ندارید. از آنجا که طرحهای

حفاظتی کلمه عبور (همانطور که من گفتم) تقریباً مشابه هستند. من حدس میزنم که شما از اولین حقه‌ایکه الان میگویم استفاده خواهید کرد.

با استفاده از نقشه حافظه محل سکونت برنامه را پیدا کنید و سپس در بزرگترین قسمت برنامه بدنبال F36A بگردید. این همان دستور REPZ CMPSB میباشد.

در حالت، استفاده از Lightspd شما با دستورات زیر، ۴ آدرس خواهید داشت. (سگمنت اصلی برنامه=pgsg)

Pgsg:C6F9

pgsg:E5CA

pgsg:E63E

pgsg:EAB0

ممی‌بینید فقط چهارتا... که با یک نظر شما میتوانید تشخیص دهید که دومی یعنی E5CA از همه بهتر است. مکانیسم مقایسه در این برنامه ۱۹۹۰، کم و بیش شبیه همان UMS سال ۱۹۸۷ میباشد. البته بنظر من همان مکانیسمها، امروزه (یعنی سال ۱۹۹۶) قابل استفاده هستند

قرار دادن بزرگترین عدد در رجیستر CX      MOV    CX,FFFF      B9FFFF

این سطر ES را از محل DI جستجو میکند. (کلمه عبور اصلی)      REPNZ    SCASB

F2AE

تعداد کاراکترهای کلمه عبور اصلی،      NOT    CX      F7DI

تغییر و تنظیم DI برای مقایسه      SUB    DI,CX      2BF9

مقایسه DS از محل SI با ES از محل DI      REPZ    CMPSB      F3A6

(یعنی مقایسه کلمه عبور واقعی با کلمه عبور کاربر)

که وقتی CX=0 شود یا آنکه به یک کاراکتر متفاوت برسد، متوقف خواهد شد.

حالا دیدید چقدر آسان است؟ اینها همه همان حلقه‌های قدیمی است. در اینجا فقط یک روتین کوچک برای تست حروف کوچک الفبا لازم است چون معمولاً حروف کلمات عبور کوچک هستند.

حالا اگر دوست داشته باشید اجرای برنامه را در یکی از این محلها متوقف کنیم البته بجز ناحیه Snap چون در این ناحیه ما نمیتوانیم یک breakpoint (نقطه نفوذ) ثابت پیدا کنیم، که معمولاً محل آن همانطور که قبلاً دیدید با offset : segment (افست: سگمنت) های مختلفی آدرس‌دهی میشود. به همین دلیل شما ابتدا باید یک break point نوشتنی / خواندنی (Real/wirte) روی حافظه برای این محلها داشته باشید و بعد از آن، آنها را برای snap بگیرید. الان با استفاده از segment: offset (آدرس دهی افست: سگمنت) میتوانید محل snap را پیدا کنید و یک breakpoint ثابت روی آن بگذارید (بعنوان مثال میتوانید اینکار را با دستور NOT CX انجام دهید).

خب حالا برنامه را اجرا کنید. یک beak point روی رجیستر ES از محل DI (ES:DI) داشته باشید و کلمه عبور واقعی را ببینید. چه زیبا! ما حالا کلمه عبور اصلی را داریم. میتوانیم آن را روی پنجره dump ببینیم که به آن میگویند یک انعکاس (echo) این echoها برای علاقمندان کراک کردن، یک مدرسه واقعی است که میتوانند با مسیرهای مختلف کار کنند که علاوه بر آنکه کلمه عبور را بدست می‌آورند میتوانند پی ببرند که محتویات سایر قسمتها از کجا می‌آید.

یک کار که در روشهای حفاظتی موسوم است، پنهان کردن آنها در فایل‌های مختلف، دور از دسترس، باید در بردارها، یا در قسمت‌های SMC می‌باشد. در اینجا یک برنامه از ۱۹۹۰ می‌بینید که از نظراتی با UMS فرق میکند.

کلمه عبور داخل یک بردار پنهان شده، از حفاظت زیبا و در عین حال ساده لوحانه‌ای در آن استفاده شده که با یک یوتیلیتی هگز(کار جادویی) شما می‌توانید آنها را ببینید. در اینجا کلمه عبور، کدگذاری شده (مثل روشهای قبلی) که می‌توانید با استفاده از همان brak pointهای قبلی آنها را پیدا کنید. شما خیلی سریع می‌توانید بخشی از برنامه را که بصورت زیر است پیدا کنید:

Sg: 0118      8C 91 9D 95 9B 8D 00 B8 EC 94 9B 8D 8F 8B 9B  
Sg: 0 128      94 9B 8D 00 AE EC 9C 9B 8A 9B 86 00 A9 EC 91

این یک نوع ماتریس کدگذاری شده می‌باشد که با پاک کننده "۰۰" (دو صفر) کلمه عبور کدگذاری شده را در بر گرفته .

ها! اگر همه این جور کدگذاری می‌کردند، کراک کردن چه آسان بود! رمزهای بچه گانه هم از اینها بهتر است!

معادل ماتریس فوق بصورت زیر است :

$$\begin{matrix} 93=6D='m' & ; & 92=6E='n' & ; & 91=6F='0' \\ & & & & \text{و الی آخر ....} \end{matrix}$$

حالا اجازه دهید کلمه عبور پنهان شده را رها کنیم و کراک خودمان را انجام دهیم.. اجازه دهید نگاهی به پروسیجر snap زیر بعد از دستور CMPSB REPZ که به دستور «پرش به OK» میرسد، بیندازیم

```

F3A6    REPZ    CMPSB    ; compares DS:SI with ES:DI
7405    JZ     preserved_AX=0000 <--- Here the first JZ
1BC0    SBB    AX,AX
        AFFFFF SBB    AX,FFFF
        :preserved_AX=0000
        8BF3    MOV    SI,BX
        8BFA    MOV    DI,DX
        5D      POP    BP
        CB      RETF
        ....
        83C404  ADD    SP,+04
        0BC0    OR     AX,AX
7509    JNZ    0276      <----- And here it is!

```

. حالا کراک UMS را بخاطر بیاورید که شما می‌خواستید دستور JZ را به JNZ تغییر دهید (شما سعی داشتید از داخل [SOFT-ICE] پرش کنید و همین کار را هم کردید!) و همچنین ۷۴ را به ۷۵ تغییر دهید. و بعد از آن دوست داشتید JNZ را به JZ تغییر دهید... لطفاً خودتان را خسته کنید. در اینجا کار عملی نیست. (شما حتی دومین JNZ را هم در برنامه پیدا نخواهید کرد) شما همیشه باید مواظب SMC (یعنی کدهایی که خودشان خودشانرا تغییر میدهند) باشید. در واقع کدی که موقع اجرای یک برنامه بدست می‌آید، با کد مطلق برنامه خیلی تفاوت دارد.

حال در اینجا ما یک اصلاحیه کوچک روی کدهای اولیه خواهیم داشت. مشابه دستوری که اجباراً برای دستکاری دیگر قسمت‌های برنامه استفاده کردید، چنانچه آن را به JNZ تغییر دهید، با یک پیغام سرریزی مواجه خواهید شد. بنابراین اصلاح دستور JNZ به این راحتیها نخواهد بود. نگاهی بیندازید به قسمتی که بعد از RETF، در حالت پرش، بوسیله برنامه light speed کمپایل خواهد شد. بنابراین شما باید جستجویی برای ترجمه مکانیسم و اصلاح رمزگذاری بایتهایی که پیدا میکنید داشته باشید و در برخی موارد ممکن است آنها را بیش از یکبار رمزگذاری کنید. آن وقت است که شب پرزحمت و سختی را سپری خواهید کرد.

پس اینکار را انجام دهید: یک جرعه ودکای ماریتنی بنوشید و کمی فکر کنید. تنها چیزی که بعد از JZ رخ میدهد تنظیم رجیستر یا ثابت AX به مقدار False، بعنوان flag (پرچم) می‌باشد. ( $Ax = 1 \dots$ ) همان چیزی است که دو دستور SBB انجام میدهد). بعنوان مثال اگر شما کلمه عبور را ندانید، با یک مقدار غیر صفر از این قسمت خارج می‌شویم. پس بگذارید با پنج بایت دو دستور SBB هیچ کاری انجام ندهیم و یا در حالت بهتر از آن بگذارید به جای دو دستور SBB سری دستورات INC AX، (یکی به رجیستر Ax اضافه کن، DEC AX (یکی از Ax کم کن)، NOP (هیچ کاری انجام نده)، INC AX، DEC AX را داشته باشیم. یعنی پنج دستور یک بایتی معادل آن قرار داده‌ایم. مزیت استفاده از دستورات فوق نسبت به یکسری دستور nop (no operation) یا هیچ کاری انجام نده) در طرح‌های

حفاظتی جدید آن است که کنار هم گذاشتن nopها داخل یک برنامه و دور ریختن هر آنچه که آن دستورها بدست می‌آورند چندان جالب نخواهد بود. شما همیشه باید سعی کنید در زمان کراک کردن، از حداقل دستورات بیشترین کارایی را بگیرید. عبارتی حذف دو دستور SBB ما را در واقع یک کراکر می‌کند. نه لازم نیست با دومین JNZ خودتان را به زحمت بیندازید. برنامه خودش کار میکند و کلمه عبور را می‌گیرد. چه شما اینکار را انجام بدهید و چه انجام ندهید. البته نوع قبلی کراک مشابه آنچه برای UMS گفتیم - برای شما بهتر است. وقتی شما برای دسترسی به کامپیوتری، کراک می‌کنید. به کاربر اصلی هیچ گونه گمانی برده نخواهد شد و سیستم او را کنار نمی‌گذارد. هر دو به کامپیوتر دسترسی دارید، یکی خوب و دیگری بد... خیلی جالب است نه؟

حالا اجازه دهید یک کراک سریع با LIGHTSPD داشته باشیم:

-----

CRACKING LIGHTSPEED.EXE (by +ORC, January 1996)

ren lightspd.exe lightspd.ded

symdeb lightspd.ded

- s (cs+0000):0 Lffff 2B F9 F3 A6 74  
xxxx:yyyy (this is the answer of the debugger)
- s (cs+1000):0 Lffff 2B F9 F3 A6 74  
(nothing, but do it nonetheless, just to be sure)
- s (cs+2000):0 lffff 2B F9 F3 A6 74  
(nothing, just to be sure, now it's enough)

- e xxxx:yyyy+6 40 [SPACE] 48 [SP] 90 [SP] 40 [SP] 48

- w

- q

ren lightspd.ded lightspd.exe

-----

همه این CMP SB ها مثل هم هستند اما با اینحال برخی برنامه ها از طرح های حفاظتی متفاوتی که چندان با ارزش هم نیستند استفاده میکنند. پس نباید اعتماد چندان به دستور REPZ F3A6 CMPSB داشته باشیم. بیایید طرح حفاظتی مورد استفاده در اولین نسخه (version) General (از QQP-White wolf جولای ۱۹۹۲۹ را باهم آنالیز کنیم.

وقتی که شما اجرای برنامه را در حین اجرای یک پنجره متوقف می کنید، در واقع شما وسط یکی از پروسیجرهای BIOS هستید. برنامه منتظر ورودی شما (مثلاً کلمه عبور) می باشد. شما به سرعت نقشه حافظه مربوط به محل قرارگیری برنامه . exe General را پیدا میکنید. Beak point را در حالت نوشتن روی حافظه تنظیم میکنید.

در آن وقت است که محل مورد استفاده حافظه برای طرح حفاظتی، برای شما آشکار میشود. در دستور زیر

xxxx:1180 to xxxx:11c0

xxxx محل دومین سگمنت حافظه که برنامه در آن ساکن شده نشان داده می شود. حالا مطابق زیر انجام دهید : (یک پروسیجر کراک کردن خیلی رایج):

\* یک break point نوشتنی روی محدوده کوچکی از حافظه که شما در آن بدنبال کلمه عبور می گردید:

\* trace با breakpoint روی محدوده‌ای از حافظه که کد اصلی قرار دارد (trace یعنی اجرای مرحله به مرحله یک برنامه).

\* اجرای یک چیز جدید

این همان چیزی است که قبلاً انجام دادیم. در اینجا در ۹ سطر، آنچه که طی فراخوانی پروسیجر در حافظه رخ میدهد، دنبال شده.

your memory area:

```
-9  xxxx:0185  7425      JZ  somewhere, not taken

-8  xxxx:0187  2D1103    SUB  AX,0311

-7  xxxx:018A  7430      JZ  somewhere, not taken

-6  xxxx:018C  2DFD04    SUB  AX,04FD

-5  xxxx:018F  7443      JZ  next_trace, taken

-4  xxxx:01D4  E85500     CALL funny_procedure

-3  xxxx:022C  803E8F8C11  CMP  BYTE PTR[8C8F],11

-2  xxxx:0231  750E      JNZ  somewhere, not taken

-1  xxxx:0233  9A0A0AC33E  CALL procedure_that_sniffs
    our_memory_area
```

خب فراخوانی پروسیجر fanny-procedure در زیر دنبال مقایسه یک بایت بعید به نظر

میرسد. پس اجازه دهید بلافاصله نگاهی به بخشی از کد برنامه General. Exe بیندازیم.

:funny\_procedure

```
803E8F8C11  CMP BYTE PTR[8C8F],11
750E      JNZ compare_byte
9A0A0AC333  CALL procedure_that_sniffs
0AC0      OR  AL,AL
7405      J2  compare_byte
```

```
C6068F8C2A  MOV BYTE PTR [8C8F],2A
:compare_byte
```

```
803E8F8C2A  CMP BYTE PTR [8C8F],2A
7504      JNZ after_ret
B001      MOV AL,01
C3        RET
```

با این درس شما باید به اندازه کافی قادر به کراک کردن باشید. توجه داشته باشید که دو دستور متناقض بدنبال هم یعنی MOV 2A و CMP 2A، هیچ تأثیری روی مقایسه 2A برای JNZ نخواهد داشت. چون که شما 2A را درست تنظیم نکرده‌اید. یعنی در واقع با اولین JNZ پرش صورت می‌گیرد. اما بدون اینکه داخل 2A چیزی باشد و داخل 2A هیچ چیزی به جز علامت (\*) نیست که برای برنامه‌نویس‌ها معمولاً حکم OK را دارد. این حفاظت به روش زیر کار می‌کنند (شرح کد بالا در زیر آمده):

- مقایسه محل مورد نظر با 11

- پرش اگر صفر نیست (NON ZERO) به مقایسه محل مورد نظر با (\*)

- در غیر اینصورت فراخوانی پروسیجر sniff

- OR کردن AL با خودش (به منظور 0 کردن آن)

- پرش اگر صفر است، به مقایسه محل مورد نظر با (\*)

- اگر AL صفر باشد، (\*) به داخل محل مورد نظر منتقل میشود.

- محل مورد نظر با (\*) مقایسه می شود.

- اگر تفاوتی دارد، پس JNZ با موفقیت به کار پایان میدهد.

- در غیر اینصورت دوباره از اول شروع می کنیم.

حال اجازه دهید به سرعت کراک کنیم:

---

CRACKING GENERAL.EXE (by +ORC, January 1996)

ren general.exe general.ded

symdeb general.ded

- s (cs+0000):0 Lfff 8C 11 75 0E

xxxx:yyyy (this is the answer of the debugger)

- e xxxx:yyyy+2 EB [SPACE] 09

- w

- q

ren general.ded general.exe

---

این، روش تغییرمیدهد JNZ را به دستور (\*) CMP در یک JMP (پرش) به دستور (\*) MOV.. بنابراین اکثر پنجره ها چندان هم حفاظت نشده اند. همه آنها با استفاده از general.exe، راحت، واضح و دست یافتنی خواهند بود.

خب خواننده عزیز اینهم از این درس. نه همه خودآموزی من روی اینترنت هستند. اگر درسی را فراموش کرده اید، به من mail بزنید. شاید شما کلک هایی بلد باشید که من هنوز کشف نکرده ام. من همان قبلی ها را میدانم اما اگر چیز جدیدی باشد اعتبار شما را خیلی زیاد می کند. حتی اگر اینطور هم نباشد من می فهمم که شما خیلی روی موضوع کار کرده اید. در آنصورت من درسهای باقیمانده را برای

شما خواهیم فرستاد. انتقادات و پیشنهادات شما در مورد چرندیاتی که من نوشتم، همیشه برای من  
خوش آمد خواهد بود.

E-mail +ORC

an526164@anon.penet.fi

