

درس ۱

آغازی برای کراکینگ

بهترین روش برای فراگیری کراک کردن یا نفوذ، استفاده از کاربردهای قدیمی‌تر که طرح‌های حفاظتی قدیمی‌تری نیز دارند می‌باشد. (منظور از کراک یا نفوذ، یعنی اختصاصی کردن، احاطه پیدا کردن، حذف یا به تعویق انداختن یا تسلیم شدن در برابر یک یا چند طرح حفاظتی داخل یک نرم‌افزار کاربردی) وقتی که شما کد اصلی برنامه را در دسترس ندارید (می‌باشد) بدینوسیله شما تکنیک اصلی مزاحمت را به سرعت فرا می‌گیرید. فراموش نکنید که سیر تکاملی طرح‌های حفاظتی همیشه به یک صورت نیست.

شما احتمالاً در برخی موارد حقه‌های زیرکانه جدیدی را، درخواهید یافت اما در بیشتر مواقع شما این حقه را به زانو درخواهید آورد.

مشکلی نیست، اطلاعات واقعی برنامه نویسان اغلب خیلی محدود است و آنها بیشتر تمایل دارند به جای تکنیک‌های جدید از همان روش‌های حفاظتی قدیمی استفاده کنند. (گرچه ممکن است کمی آنها را تغییر دهند یا حتی در برخی موارد پیشرفتهایی هم بدست آورند). این حالت معمولاً زمانی اتفاق می‌افتد که کسی در عوض پول یا برای لذت خودش کاری را انجام دهد. این نوع تمایلات تجاری، کورکورانه توسط افراد نادان و پول پرست تشویق می‌شود، یعنی همانهایی که ما مجبوریم با آنها زندگی کنیم.

بهمین دلیل من با استفاده از مثالها، برخی کاربردهای قدیمی و برخی حقه‌های قدیمی از درس سوم به بعد، آنها را به شما یاد میدهم بطوریکه بعدها ما خواهیم توانست حقه‌های حفاظتی جدیدتر را نیز با آن پوشش دهیم و شما یادخواهید گرفت که چگونه بر آنها پیروز شوید. و به شما خواهم گفت که در چه جاهایی می‌توانید برنامه‌های زیادی برای کراک کردن پیدا کنید. البته نه برای پولدار شدن!

این خودآموز برای افرادی است که میخواهند کراک کردن را شروع کنند. شاید شما به فکر کراک کردن باشید و هدف شما هم موفقیت در این زمینه باشد، پس استفاده از این خودآموز و شروع با حقه‌ها و پروسیجرهای آن، شما را در همان مسیری که میخواهید قرار میدهد.

البته من قول نمیدهم آنچه میخواهید با این خودآموز بدست آورید اما من بهترین کار ممکن را برای شما انجام میدهم. بعبارت دیگر اگر شما قبلاً کدهایی را به کمک اسمبلر کراک کرده باشید یا به برخی طرحهای حفاظتی نفوذ کرده باشید، این خودآموز برای شما بعنوان یک مبتدی مفید خواهد بود. (اگر میخواهید کمی مفاهیم پایه‌ای را تکرار کنیم، حرکتی نکنید تا “go” را فشار دهیم، پس همینجا بمانید).

برای یک کراک موفقیت آمیز، شما به چهارچیز اصلی احتیاج دارید:

- یکسری اطلاعات در مورد زبان یک اسمبلر (برای داشتن معلومات بیشتر، و همچنین یک کراک سریعتر و بهتر).

- مقداری بینش

- کمی کمک از یک کراکر با تجربه

- یک تفکر غیرتجارتی (که در این مورد بعداً بیشتر صحبت خواهیم کرد).

کاربردهایی که شما برای فراگیری از آنها استفاده خواهید کرد به چند دسته زیر تقسیم می‌شوند:

- ۱- کاربردهایی که با یک کلمه عبور محدود شده است (از همه برای کراک کردن ساده تر است).
 - ۲- کاربردهایی که بر اساس زمان، یا تعداد روزی که شما از آن استفاده می کنید محدود شده اند (تقریباً برای کراک کردن آسان است).
 - ۳- کاربردهایی که بر اساس آخرین تاریخی که شما از آنها استفاده کردید، محدود شده اند (آسان برای کراک کردن).
 - ۴- کاربردهایی که دارای تعداد فانکشن (تابع) هستند. غیرفعال (برخی آسان، برخی مشکل).
 - ۵- کاربردهایی که برای کار کردن با آنها، نیاز به دسترسی به دیسک می باشد (این نوع طرح حفاظتی امروزه چندان استفاده نمیشود).
 - با حضور CD-ROM (کم و بیش همان روشها بکار گرفته شده، اما نه به آن صورتی که قبلاً تعریف شده بود) (خیلی آسان برای کراک کردن).
 - ۶- CRYPTOGRAPHIC FEED ADDSON (یعنی یکی از طرحهای حفاظتی قبلی، اما دارای زحمت بیشتر یا استفاده از کدهایی که خودشان، خودشان را اصلاح میکنند. (XOR کردن و SHRL کردن) (تقریباً برای کراک کردن آسان است).
 - ۷- هیچیک از حالات فوق (برخی مواقع برای کراک کردن مشکل است).
- مواد مورد نیاز را از کجا تهیه کنیم.
- CD ROM DEMO رایج روی جلد مجلات، ثروتی برای همه کراکرهاست! بعد از مدت کوتاهی، آنها اعلام میکنند، شما همه کپی های باقیمانده که غیرفروشی هستند را در آینده مجانی دریافت خواهید نمود. DEMO های روی سی دی ها به شما اجازه میدهد که خیلی سریع، کاربردهای زیادی را - چه کهنه و چه نو - جمع آوری کنید که هریک از آنها به طریقی حفاظت شده اند (که در برخی مواقع با طرحهای جالب توجهی حفاظت شده اند). در واقع با یک دنیای

عجیب کراک مواجه می‌شوید. بعداً بدون هیچ پولی شما می‌توانید روی یک سی‌دی کاربردهای لوتوس (یا Microsoft یا wordperfect یا هر نام دیگری) که برای ۳۰ روز، یا ۲۰ بار دسترسی به آن، تنظیم شده باشد، کراک کنید و در آن صورت برای استفاده بیشتر و بیشتر و یا قرار دادن آن بعنوان هدیه‌ای گرانبها روی وب برای تھی دستانی که هیچ پول یا هوشی ندارند، به آن نفوذ کنید. بازیها به نحو آشکاری، از این روش پیروی نمی‌کنند. آنها از این نظر دورنمای جالبی، و رای حفاظت، برای کراکرها دارند. یعنی اینکه طرحهای حفاظتی سطح بالا داخل فایلهای کوچک آنها پنهان شده‌اند. به همین سادگی و آسانی برای شکستن و حذف یک طرح حفاظتی داخل یک فایل ۳۵۰۰۰ بایتی قابل اجرا باید آنها را داخل مجموعه‌ای از چندین DLL طولانی که همدیگر را پوشش می‌دهند و هریک می‌توانند طولی به اندازه ۲۰۰۰۰۰۰ بایت داشته باشند، پیدا کنید. گروههای برنامه‌نویسی پیشرفته و در عین حال تنبل، به طور سیستمیک. طرحهای حفاظتی را طوری پنهان می‌کنند که مثل نیش جانوری در یک بیابان پهناور می‌ماند. در واقع آنها قادر نیستند برنامه را داخل یک اسمبلر بکار گیرند. آنها از زبانهای چرب و نرمی مثل ویژوال بیسیک، دلفی یا ویژوال C++ استفاده میکنند (نگران نباشید، من چگونگی کراک کردن آن هم سریع با این کاربردها را به شما خواهم آموخت).

دلیل دیگر استفاده از بازیها بجای کاربردها در این مطالعه این است که اغلب همان طرح حفاظتی ساده (و کوتاهی) که شما در یک بازی اشتراکی به آن نفوذ کردید بعداً برای یک کاربرد گرافیکی با ارزش و گران قیمت استفاده می‌شود به همین دلیل در این خودآموز ما بیشتر به طرحهای حفاظتی بازیها، نفوذ خواهیم کرد تا اینکه در آینده از آنچه آموختیم، برای نفوذ به طرح حفاظتی یک برنامه کاربردی تجاری، یا برای نفوذ و دسترسی به روتینهای حفاظت شده سرورهای دور (یا remote)، یا BBS، یا حتی ATM (صندوق ناظر هزینه) استفاده کنیم.

در اینجا، یک مثال کراک کردن را در ذیل می‌بینید که من امیدوارم به این وسیله شما به هنر ما پی ببرید. اجازه دهید بعنوان مقدمه، با هم به این کاربرد حفاظت شده، نفوذ کنیم. بعداً در درس ۴ شما یاد خواهید گرفت که کلیه کاربردهایی که بر اساس زمان محدود شده‌اند (یعنی مدت زمانی که شما می‌توانید کار کنید یا تعداد دفعاتی که می‌توانید کار کنید) طرحهای حفاظتی آنالوگی دارند. (اگرچه با وجود بزرگی، تنوع چندانی ندارند).

۱- ممکن است آنها شمارنده‌ای داشته باشند که هر بار یکی اضافه می‌شود: آن را بیابید و غیر فعال کنید.

۲- ممکن است آنها وقفه‌های زمان ساعت را FETCH کنند و در ماشین شما قرار دهند: خودتان از اینکار جلوگیری کنید.

۳- ممکن است آنها یک مقدار تصادفی را با یک متغیر مقایسه کنند: با این حالت هیچ کاری نمی‌توانید بکنید.

۴- ممکن است آنها بصورت تصادفی تاریخ سایر فایل‌های روی هارد شما را (حتی آنهایی که بی ربط هستند) بررسی کنند: در این حالت شما باید روتینی که اینکار را انجام می‌دهد پیدا کند و JUMP‌های آن را بر عکس کنید!

حالا من می‌خواهم با یک مثال پیشرفته از این حالت حفاظتی استفاده از شمارنده‌ها، شروع کنم. در این وضعیت احساس کراک کردن به شما دست خواهد داد و من یک DEMO خیلی رایج را برای اینکار انتخاب کرده‌ام که شما هم خیلی راحت می‌توانید آن را پیدا کنید. برای نمایش برخی مشکلاتی که ممکن است شما با آن روبرو شوید، ما از این مثال استفاده می‌کنیم. (شما روش نفوذ کار و اثربخش را طی درسهای آینده خواهید آموخت).

ARCADE POOL, Demonstration version, PC Conversion مثال:

by East Point Software Ltd, (c) Team 17 Software Ltd 1994

این DEMO از طریق تعدادی از مجلات روی سی دی رام در طول سال ۱۹۹۵ منتشر شده است. آنچه در ذیل می آید، حتی اگر شما این مثال را نداشته باشید، برای شما قابل استفاده می باشد. اما در عین حال شما باید یک کپی از این DEMO را به منظور فهم بهتر نکات زیر تهیه کنید. این DEMO زیبا، یک بازی بلیارد است که بر اساس مدت زمان استفاده از آن محدود شده. یعنی شما فقط ۲ دقیقه میتوانید بازی کنید. و بدنبال آن در یک پنجره به شما اعلام میشود که از کجا و چگونه میتوانید نسخه اصلی این برنامه را تهیه کنید. این بهترین حفاظت درهم و برهم می باشد. خب حالا چطور و از کجا میخواهید شروع کنیم؟ آنچه شما میتوانید انجام دهید، در ادامه آمده است:

[SOFT-ICE] را بگیرید و آن را در فایل Config.sys تان بارگذاری کنید. درس ۲ یعنی ابزارهای

مزاحمت در مورد این دیباگر را ببینید. نسخه ۲/۶ برنامه [SOFT-ICE] بوسیله MARQUIS DE

SOIREE، کراک شده و بصورت مجانی روی وب، یافت میشود.

vecs s- (ذخیره همه بردارها قبل از بارگذاری آنها)

[Pooldem.exe] start-

vecs c- (مقایسه بردار، ذخیره یک پرینت از کلیه بردارهای hook شده).

- برای فهم اینکه چه چیزی در [pooldemo.exe]، کجا می رود و ما کجا پرسه می زنیم، چند بار

وارد soft-ice شوید و از آن خارج شوید.

(شما همیشه باید بیش از یکبار یافته هایتان را موقع جستجو کنکاش کردن، چک کنید هیچ چیز به

اندازه همان dos غیرفعال و قدیمی نمی تواند نکات و روشهای پیچیده را برای ما روشن کند).

- نظری به (نقشه یا) map حافظه مورد استفاده بیاندازید.

- حالا snap حافظه اصلی را. جایی که [pooldemo.exe] در آن مستقر شده ذخیره کنید. Snap

کردن یعنی ذخیره تصاویری از نواحی حافظه (snap-sane)

- هیچ کاری انجام ندهید و اجازه دهید ثانیه‌ها سپری شوند.

- هر دو یا سه ثانیه بدون اینکه چیزی را روی صفحه بازی جابجا کنید snapها را مقایسه کنید

(snap-compar) (هیچ چیزی، حتی کلیک موس) بنابراین امیدواریم تنها تغییراتی که باعث فعال

شدن شمارنده می شوند

ذخیره شوند.

- مقایسه snap دوبار در ثانیه -مقایسه snap در ثانیه: 00:۵۹ و در ثانیه ۱:۰۱

- مقایسه snap، درست قبل و بعد از محدودیت زمانی و snap صفحه nag (یا nag screen)

- حالا با دقت پرینت dataهای snapهایتان را جمع آوری کنید. حالا بطور واضح روی صفحات

مختلف snapهای روی داده را بنویسید.

- حالا لحظه زیبای کراک کردن فرا رسیده. حالا با یک ودکای مارتینی بنشینید (فقط ودکای

روسی) و به محلهای مختلفی که روی پریتتها دارید فکر کنید. محلی را پیدا کنید که در آن مقایسه

snapدچار تغییر شده، آنالیز، تغییر و معادل سازی کنید.

هی، چیزهایی اینجا و آنجا صید شده‌اند (شما خوش شانس هستید با کمی تغییرات شما اینکار را

انجام دادید، در این حالت فقط دو دوجین).

- در حال اجرا روی محلی که پیدا کرده‌اید، یک break point بگذارید. یعنی محلی که فکر

می کنید شمارنده از آنجا فعال می شود. بعبارت دیگر محلی که در یک زمان خاصی، با صفر

شدن آن اجرای برنامه خاتمه می‌یابد.

- حالا معادل شمارنده BPX را بوسیله break کردن در لحظه‌ایکه محتویات محل مورد نظر در حال تعویض است، در آن جا قرار دهید. بدین وسیله، شما باعث می شوید برنامه از اول شروع به اجرا کند. (به اندازه یک ثانیه برای شروع مجدد طول می کشد). استفاده از شمارنده معادل، مثل یک اجرای یک مرحله‌ای در طول حیات برنامه شما می‌باشد!

در این حالت شما به سرعت در محل تعویض یعنی 3DD0، قرار می‌گیرید. سطر تعویض در زیر آمده است:

```
xxxx:3DCC C706F1013C00 MOV WORD PTR [01F1], 003C
```

که بایت 3C در آدرس xxxx:3DD0 یک شمارنده بایت می‌باشد. یعنی برنامه 3C را در این محل بارگذاری می‌کند و سپس دستورهای DEC آن را مرحله به مرحله. کم می‌کنند یعنی 3B,3A,39,38 و تا 0 و وقتی که به صفر رسید، بینگو! کاربر نادان یک ثانیه از وقتش برای بازی را از دست داده است.

اگر شما جستجوی بیشتری برای یافتن محل صحیح مربوط به nag screen، برای حذف حفاظت انجام داده‌اید، حالا وقت آن است که همان اولین سطح ویزارد مربوطه را حفظ کنید. اما اگر فکر می‌کنید آن را قبلاً بدست آورده‌اید ولی خاطرتان نیست که از چه روشی استفاده کرده‌اید، این اصل را بخاطر بسپارید: "فقط یکبار شما می‌توانید اثر یک حفاظت را حذف کنید، پس بیش از این کاری انجام ندهید".

اکثر مواقع این گفته درست است که : شما همیشه احتیاجی به حذف یک حفاظت ندارید (مگر آنکه شما دقیقاً برای اینکار مطالعه کرده باشید). معمولاً حذف اثر یک طرح حفاظتی آسانتر (و سریعتر) میباشد که متأسفانه در مثال ما کار درستی نخواهد بود.

در اینجا شما تصور می کنید شمارنده را قبلاً پیدا کرده اید. در واقع شما، شمارنده ای که ساعت طرح حفاظتی را تنظیم می کند، پیدا کرده اید [pooldemo.exe]. حالا شاید فکر کنید که می توانید مقدار یدکی را پیدا کنید و آن را به جای 3C به EE تغییر بدهید، (ها، FF را هم تمرین کنید... اما همیشه تمرین کردن روی مقادیر اضافی در موقع کراک کردن خیلی خوبست) شما باید چهار دفعه برای بازی تان زمان بگیرید تا طرح حفاظتی غیرفعال شود.

پس محل xxxx:3DD0 را از 3C به EE تغییر دهید. برای کار روی بایتها شما باید از یک ویرایشگر هگز مثل PSEDIT یعنی [Psedit.exe] (یا parity solution) که به صورت اشتراکی هم اجرا می شود استفاده کنید (بخش ابزار مزاحمت را ببینید) ولی همیشه امکان کار با دیباگرهایی مثل [debug] یا [symdeb] برای شما فراهم نیست (درس ۲ را ببینید). حالا فایل dead[*].exe] یک با کپی آن کار کردید بخاطر بیاورید.

```
ren POOLDEMO.EXE POOLDEMO.DED
symdeb POOLDEMO.DED
-s (cs+0000):0 Lffff C7 06 F1 01 C3 <- this string
corresponds to the
refill line).
cs:3E85 <- symdeb gives you two locations as answer
cs:3EEA
-e cs:3E85+4 EE <- refill changed from C3 to EE
-w
ren POOLDEMO.DED POOLDEMO.EXE
```

حالا شما pooldemo موقتی خودتان را اجرا کنید. گمان می کنید آن را کراک کرده اید و احساس خشنودی می کنید.. اما لوو! هیچ چیزی در مجموع تغییر نکرده است. همه چیز به سختی قبل است.

شما فقط دو نوبت برای بازی وقت دارید. چطور می خواهید حریف را ناکام کنید بطوریکه نتواند هیچ کاری انجام دهد؟

خب برای شروع شما به اندازه کافی مراقب نبودید! جستجو در دیباگر دو محل را به شما می دهد، شما نادانی کردید و آن را که درست بود تشخیص ندادید. چک کنید، آن وقت است که خواهید دید دومین محل (CS:3EEA) یک منطقه با کنترل آینه ای می باشد (در این مورد بعداً بیشتر صحبت خواهیم کرد). در برخی مواقع دو محل وجود دارد که با استفاده از یک روتین دوبل، کار سریعتر می شود. گاهی دومین محل آینه اولی می باشد و در مواقع لزوم آن را تصحیح می کند. پس شما باید این یکی را هم اصلاح کنید. مثل قبل رفتار کنید. اما این بار به کمک یک دیباگر سطر زیر را وارد کنید:

-E CS: 3EEA+4 EE

قبل از نوشتن برگردید به فایل deal و آن را به exe تغییر نام دهید و بعد آن را اجرا کنید. لوو! هوو!

یواش! شما آنجا هستید؟ pooldemo.exe شما، حالا بدون حفاظت است. فکر می کنید حالا بتوانید مثل احمقها ۱۲ دقیقه واقعی بازی کنید، حتی اگر معتقد باشید طرح حفاظتی (و شمارنده) تنها دو دقیقه به شما وقت می دهند.

پس شروع به بازی کنید، ثانیه ها خیلی آهسته می گذرند و همه چیز خوب به نظر می رسد. اما نه! در صفحه ثانیه ۲۸، شما تصویری را می بینید که به شما میگوید «دو دقیقه تمام شد». یعنی شما واقعاً اشتباه کردید، برنامه زمان صحیح را از تایمر می گیرد و شما بطرز احمقانه ای فقط شمارنده روی صفحه نمایش را اصلاح کرده اید.

پس باید برگردید برای کراک کردن و اوقاتتان تلخ است بطوریکه راههایی را که از آغاز کراک کردن طی کرده‌اید فراموشتان شده. پس باید حالا کارتان را تکمیل کنید. شروع کنید به چک کردن بردارهای hook شده (قبلاً روتین vecs-save را دیدید که pooldemo را بداخل [soft-ice] بارگذاری می کرد و بعد از آن شما می توانستید از vecs-compare استفاده کنید). و بعضی چیزهای جالب را ببینید.

این بیشتر شبیه آن چیزی است که شما چیزی در مورد آن نمی دانید. INT-8 بطور غیر مستقیم وقفه تایمر INT-1C را کنترل می کند. کلیپ ساعت 8253، یک وقفه سخت افزاری IRQ-0 را در محدوده 18.2 وقفه در ثانیه تولید می کند که باعث انتقال کنترل به ISR (یا روتین سرویس دهنده وقفه) که در اینجا به INT-08 اشاره می کند، می شود و ... این باید در آدرس 0CD1:17AC باشد اما در اینجا بوسیله pooldemo.exe در 1EFD:84C6 گیر افتاده.

یکی از کارهایی که روتین وقفه مربوط به INT-08 داخل بایاس انجام می دهد آنست که یک وقفه نرم افزاری برای INT-1C فراخوانی می شود که درست در همین حال جلو پاسخگویی ماژولهای نرم افزاری داخل سیستم، گرفته می شود. و چنانچه ممانعتی بعمل نیاید، محتویات پیش فرض مربوطه به بردار وقفه 1C وقوع یک دستور نابجا در بایاس سیستم را اعلام می کند. بنابراین هیچ کاری انجام نمی گیرد.

بطور نرمال یک محافظ باید از INT-1C جلوگیری کند تا هر ISR از INT-08 مربوط به CPU محتویات بردار وقفه صحیح را از حافظه فراخوانی کند (یا fetch) و یک شیوه فراخوانی وقفه برای که آن ادرس (که باید شامل دستور مربوطه در آدرس F000:9876 باشد ولی بجای آن محتوی هر چیزی که فکرش را بکنید شده است) می باشد.

پس - شما گمان می کنید - طرح حفاظتی اینجا مستقیماً با وقفه 08، hook شده در حالیکه یک طرح حفاظتی پیوسته و تشکیل در اینجا استفاده شده. خب حالا چکار کنیم؟ به جای اینکار به توضیحات زیر توجه کنید:

وقفه تایمر سطح IRQ-0 که به وسیله بیت صفرم رجیستر mask کنترل شده، و در آدرس I/O 0021h قرار دارد باید غیرفعال شود. وقتی که بیت صفرم رجیستر mask به 1، تنظیم شد (یا set شد) هیچ کمکی به وقفه‌ها برای تشخیص سطح IRQ نخواهد شد که متأسفانه نباید اینطور باشد. اما این تفکیک جالبی است که بهتر است شما آن را یاد بگیرید، که در این حالت شما در جای دیگر به آن نیاز دارید.

...حقه‌ای برای غیرفعال کردن تایمر (IRQ-0 masking، نوشته +ORC)

اعلان سیستم را \$t (یعنی نمایش زمان) تغییر دهید و enter بزنید، کمی بعد خواهید دید که ساعت dos در طول اجرای این حقه چطور عمل خواهد کرد. DEBUG.com را اجرا کنید. با استفاده از فرمان 'a' اسمبل کنید:

```
* Assemble using the command 'a'
- a
in al,21
or al,1
out 21,al
ret
RETURN
RETURN    <- twice to exit immediate assembler
- g 100    <- to run the tiny program.
- q        <- to quit debug.
```

اعلان فرمان \$t (نمایش زمان) هنوز فعال است، کمی بعد از زدن enter: هورا! پیشرفت ساعت متوقف شده!

توضیحات: شما محتویات رجیستر mask جاری را به داخل AL، بارگذاری کردید، بیت mask را در محل بیت صفرم، تنظیم کردید (که مربوط به وقفه IRQ-0 بود) و بالاخره مقادیر برگشتی به رجیستر mask را بروز رساندید. وقتی که شما آماده فعال کردن مجدد رخداد IRQ-0 هستید دوباره دیباگ را اجرا کنید. و سپس ساعت متوقف شده DOS را با فرمان IME مجدداً راه اندازی کنید.

```
- a
in al,21
and al,fe
out 21,al
ret
RETURN twice
- g 100
- q
```

در واقع با غیرفعال کردن کلیک تایمر، برخی پروسه‌ها، صحیح عمل نمی کنند. یکبار شما به درایو دیسکت دسترسی پیدا می کنید، پس از آن موتور، اجرای برنامه را تا بینهایت ادامه خواهد داد و الی آخر.

متأسفانه تکنیک فوق نمی تواند با [pooldemo.exe] ی ما که شما حالا در حال جستجوی آن هستید و فکر می کنید طرح حفاظتی را پنهان کرده، کار کند. در این مورد، شما باید بلافاصله EOI یعنی انتهای وقفه را از طریق MOV AL,20h , OUT 20h,AL را پیدا کنید. هر دوی کنترل کننده‌ها، دارای یک پورت دوم آدرس در 20 (یا 0a0h) می باشند، که دستورات از آن گرفته می شود که مهمترین آنها فرمان EOI (20h) میباشد. این دستور، انتهای هندلر وقفه را تعیین میکند و معادل کنترل کننده را برای وقفه بعدی، آزاد میکند. اگر کسی یک هندلر وقفه بهینه شده بنویسد

(همانطور که برخی طرحهای حفاظتی انجام می دهند). این امر، او را موفق به دیدن انتهای هندلر

فرمان EOI (20h) نوشته شده برای پورت 20h یا پورت 0a0h می کند.

بعد از ادامه EOI، پوش (push) های معمول، سپس برخی call (فراخوانی)ها، بعد از یک

فراخوانی ای منجر به AL, 40 out میشود، که به نظر میرسد مشابه احیاء تایمر باشد (دستور out،

data را به یک پورت خروجی و پورتهای 40-42 معادل شمارنده / تایمر، انتقال می دهد). برای

نگهداری، یک فراخوانی دابل، یک فراخوانی شرطی و سپس یک فراخوانی مرموز از نوع FAR

CS:[AA91] لازم است که بستگی به بایت PTR[970C] خواهد داشت که فراخوانی نهایی را

تعیین خواهد کرد. پس روتین کلیه رجیسترها را POP میکند.

شما آهی میکشید و شروع به dis assembling می کنید. بداخل هریک از فراخوانی ها نظر

مهندسی ای می اندازید، (روش سریعتر در این حالت، استفاده از breakpoint، در محل ورودی

می باشد که ببینید که آیا call دیگری از این طرح حفاظتی محدود به زمان پیدا میکنید یا نه).

شما کار کردید و کار کردید و کار کردید. و در مجموع هیچ چیزی پیدا نکردید. حفاظت این برنامه

اینجا نیست.

برگردید به آنالیز پرینت snapها، ما خیلی زود آن را به حال خود رها کردیم.

اگر با دقت بیشتری به مقایسه محل های محدوده DS:FFFF و DS:0 نگاه کنید، متوجه خواهید شد

که یکی از آنها خیلی آهسته از 0 به 1 به 2 به 3 و ... تغییر می کند. در حالیکه تغییر مکانهای قبلی

خیلی سریع بوده و سیکل کامل FF..0 را طی میکرد، یک شمارنده در محل DS:0009 و

DS:0004! حالا این چقدر طول میکشد؟ خب ما در بالا دیدیم که شارژ کردن در هر ثانیه، 3C

می باشد بنابراین در کل $X3C * x78 = X1C20$ خواهد شد که X78، معادل ۱۲۰ ثانیه است. یعنی

محدوده زمانی دو دقیقه.

حالا داخل کد بدنبال مقدار 1C20 بگردید (زمان حفاظتها، اکثراً در شروع بخش CS:offset، قرار دارند) و شما به سرعت به آنچه در ذیل آمده خواهید رسید:

حفاظت در [pooldemo.exe]، در محل کد زیر قرار دارد:

```
CS:0A8A 813E20A7201C CMP WORD PTR [A720], 1C20
compare location A720 with limit 1C20
CS:0A90 7C07 JL okay_play_a_little_more
CS:0A92 E834FD CALL beggar_off_time_is_up
BINGO!: FOUND!
```

Now let's quickly crack it:

حالا اجازه دهید به آن سریعاً کراک کنیم:

```
CRACKING POOLDEMO.EXE (by +ORC, January 1996)
ren pooldemo.exe pooldemo.ded
symdeb pooldemo.ded
- s cs:0 Lffff 81 3E 20 A7 20 1C
xxxx:yyyy <- this is the answer of the debugger
- e xxxx:yyyy+5 4C <- this time limit is much better
- w
- q
ren pooldemo.ded pooldemo.exe
```

در اینجا ما یک کراک طولانی انجام داده‌ایم ولی در عوض خودمان را با یک محدوده زمانی (بهتر) وفق دادیم، آن را از 1C20 به 4C20 (۴ دقیقه) به جای ۲ دقیقه تغییر دادیم. اگر دستور JL (Jump lower) را به دستور JMP (پرش به هر طریق) تغییر می‌دادیم ما بطور وضوح توانستیم یک کراک ریشه‌ای انجام دهیم. در این حالت برای گرفتن نتیجه‌ای که در درس ۴ بیان

خواهد شد، شما لازم است یک نفوذ کاملاً ظریف را در برخورد با یک طرح حفاظتی محدود به زمان، انجام دهید.

همانطور که دیدید، طی این نفوذ ساختگی، ما طرح حفاظتی را پس از کمی جستجوی مخفیانه در اطراف، پیدا کردیم. اما همانطور که در درسهای آینده خواهید دید، ما برای نفوذ به یک طرح حفاظتی، از روشهای مختلفی میتوانیم استفاده کنیم. بعنوان مثال، همین حفاظت را شما به روشهای دیگری نیز میتوانید انجام دهید: مثلاً روی محدوده‌ای از حافظه برای برنامه، یک trace انجام دهید. البته باید trace را به اولین قسمت آن محدود کنید). با قرار دادن یک break point روی nag screen، گاهی به آخرین دستوره‌ای trace شده 300-400 بیندازید. اگر هیچ چیزی را جابجا نکنید، هرچیزی، یک الگوی تکراری خواهد داشت تا اینکه حفاظت روی کدهای زیر snap شود:

```
...
JL 0A99
CMP BYTE PTR [A72A],01
...
JL 0A99
CMP BYTE PTR [A72A],01
...
for ages and ages and then...
...
JL 0A99
E834FD CALL 0759          <- BINGO! (CALL beggar_off_time_is_up)
```

راههای دیگری هم برای اینکار یافت میشود (اما ظاهراً، این روش بهترین است ولی متأسفانه خیلی بی‌ثبات است. چون بستگی به محدوده breakهای شما و فاصله بین حفاظت و nag

screen دارد. بنابراین ممکن است بطریقی تکمیل شود ولی اطمینانی که به روشهای قبلی داشتیم، را در اینجا نخواهیم داشت.

علت اینکه نزدیکی به محل مورد نظر در کراک کردن موفقیت آمیزتر از کار با بردارها می باشد آنست که بیشتر برنامه ها هیچوت به سختی حفاظت نشده اند. بنابراین، یافتن حفاظت آنها چندان مشکل نخواهد بود. (و اینکه کراک کردن به آنها، برای مطالعه، واقعاً ارزشمند است).

گاهی اوقات شما حتی نیازی به کراک کردن همه چیزها ندارید، برخی کاربردها، تعداد زیادی فانکشن (یا تابع) دارند اما کاملاً عجولانه، برای انتشارشان بصورت یک demo درآورده شده اند. برنامه های تجاری فقط پولشان مهم است و حتی برایشان اهمیتی ندارد که ما چکار می کنیم و چندان مراقبتی برای اینکار ندارند. بهمین منظور مقدار سایر چیزهای روی هارد که ممکن است کاربر آنها را از طریق مازولهای برنامه اصلی بهم بریزد، برای آنها اهمیتی ندارد. یک مثال از این نوع، روش بهم ریخته مربوط به demo [Panzer General] از SSI است که در تابستان ۹۵ به بازار آمد. که آن در واقع تکمیل شده نسخه بتای بازی بود. شما در واقع باید یکی از دو نسخه مجاز را از بین بیش از ۲۰ نسخه بتا، برای بازی مجانی، پیدا می کردید. ... شما حتی نیازی به کراک کردن هم نداشتید.

مثال pooldemo بالا نباید شما را از اصل کراک دلسرده کند. دقت کنید و آنالیز zen را از اول تا آخر، قبل از اینکه مجبور به انجام روشهای پیچیده شوید، انجام دهید. بخاطر داشته باشید که شما می خواهید به طرح حفاظتی بهر طریق ممکن نفوذ کنید و نیازی به خط سیر برنامه نویسی که آن برنامه را نوشته ندارید. فقط از شما خواسته شده به این برنامه نفوذ کنی.

خب خواننده عزیز اینهم از این درس. همه درسهای من روی وب نیستند..

به من mail بزنید. شاید شما کلک‌هایی بلد باشید که من هنوز کشف نکرده‌ام. من همان قبلی‌ها را میدانم اما اگر چیز جدیدی باشد اعتبار شما را خیلی زیاد می‌کند. حتی اگر اینطور هم نباشد من می‌فهم که شما خیلی روی موضوع کار کرده‌اید. در آنصورت من درسهای باقیمانده را برای شما خواهم فرستاد. انتقادات و پیشنهادات شما در مورد چرندیاتی که من نوشتم، همیشه برای من خوش‌آمد خواهد بود.

+ ORC E-mail

an526164@anon.penet.fi