درس ٦

ترفندهای عجیب و غریب

قبل از برداشتن قدم بعدی،چکیده‌ای از آنچه که در درسهای ۳ تا ۵. کـه بـا تمرینـات سـاده کـراک کردن، شروع می‌شود آمده را بیان کنیم : سرچ کردن مدل (مـدل ۱,۱.). ایـن توالـی قدیمـی برنامـه محافظت larry ساده‌ترین و ابتدایی‌ترین برنامه است. این برنامه بسـیار رایجـی اسـت (و یـافتن آن بسیار ساده است) . و یکی از اولین برنامه‌هایی است که به جای خواستن کلمـه‌های رمـز معنـی‌دار (که به مان امکان ضبط نمودن سریع آنـها در حافظـه را پیشـنهاد می‌کنـد) سـراغ عـدد تصـادفی را میگیرد که خریدار براحتی می‌تواند آن را در کاتالوگ پیدا کند، در صورتیکه کراکر بد نمیتوانـد. (در اینجا، بوسیله ماوس، یک عدد غیر از ۵ برای وسیله انتخاب شده بطور اتفاقی انتخـاب کنیـد). مـن لازم نمیدانم که چیزهای بیشتری در مورد چگونگی یافتن بخش مربوط بـه کـد بـه شـما بیـاموزم. وقتی،محافظت را پیدا کردید، انچه که دست یافته‌اید این است :

```
:protection_loop
 :C922 8E0614A3        MOV     ES,[A314]
...
 :C952 50 0E           PUSH    AX & CS
 :C954 E81BFF          CALL    C872      <- call protection scheme
 :C957 5B              POP     BX twice
 :C959 8B76FA          MOV     SI,[BP-06] <- prepare store_room
```

```
 :C95C D1E6           SHL     SI,1       <- final prepare
 :C95E 8942FC         MOV     [BP+SI-04],AX  <- store AX
 :C961 837EFA00       CMP     Word Ptr [BP-06],+00  <- good_guy?
 :C965 75BB           JNZ     C922           <- loop, bad guy
 :C967 8E0614A3       MOV     ES,[A314]
 :C96B 26F606BE3501   TEST    Byte Ptr ES:[35BE],01  <- bad_guy?
 :C971 74AF           JZ C922                <- loop, bad guy
 :C973 8B46FC         MOV     AX,[BP-04]...  <- go on good guy
Let's see now the protection scheme called from :C954
 :C872 55             PUSH    BP
...
 :C8F7 90             NOP
 :C8F8 0E             PUSH    CS
 :C8F9 E87234         CALL    FD6E <- call user input
 :C8FC 5B             POP     BX
 :C8FD 5B             POP     BX
 :C8FE 8B5E06         MOV     BX,[BP+06]
 :C901 D1E3           SHL     BX,1
:C903 39872266       CMP     [BX+6622],AX  <- right answer?
 :C907 7505           JNZ     C90E      <- no, beggar_off
 :C909 B80100         MOV     AX,0001   <- yes, AX=1
 :C90C EB02           JMP     C910
 :C90E 2BC0           SUB     AX,AX     <- beggar_off with AX=0
 :C910 8BE5           MOV     SP,BP
 :C912 5D             POP     BP
 :C913 CB             RETF              <- back to main
```

حال، به پنج سؤال زیر پاسخ دهید:

۱) در کدام قسمت از حافظه (در کدام محلها)، شماره رمزهای «صحیح» ذخیره شده است؟

در کجای حافظه، بخش مربوط به آدرسها، ذخیره شده است؟ این طرح،چگونه افست را بدست می‌آورد؟

۲) آیا دستورالعملهای NOP در کراک C965 : و C971 : باید قرار گیرد؟

به نظر شما، این طرح خوبی است؟

۳) آیا کراکC907 : باید به کراک JZ تبدیل شود؟ آیا به نظر شما این طرح خوبی است؟

٤) آیا کراک C907: به کراک JNZ C909 باید تغییر کند؟ آیا با این طرح موافقید؟

۵) سعی کنید برای کراک کردن این طرح در Spades، حداقل ۷ وصله مختلف دیگر بنویسید.

اه! حا،باید بتوانید ۵ تمرین بالا را در کمتر از ۱۵ دقیقه بدون استفاده از اشکال‌زدا انجام دهید! فقط به داده‌های بالا نگاه کنید و پاسخهای صحیح را پیدا کنید. حال به موضوع این درس می‌پردازیم:

(xoring) (روشهای ساده سری کردن)

یک روش ساده برای سری کردن داده‌ها، روش XOR است. XOR، دستورالعمل دستکاری یک بیت است که می‌تواند به منظور رمزگذاری و رمزگشایی داده با کلید یکجور مورد استفاده قرار گیرد:

```
Byte to encrypt                    key            result
     FF                  XOR       A1               5E
     5E                  XOR       A1               FF
As you can see XOR offers a very easy way to encrypt or to
decrypt data, for instance using the following routine:
 encrypt_decrypt:
     mov  bx, offset_where_encryption/decryption_starts
 xor_loop:
     mov  ah, [bx]            <-   get current byte
     xor  ah, encrypt_value   <-   engage/disengage xor
     mov [bx], ah             <-   back where you got it
     inc  bx                  <-   ahead one byte

     cmp  bx, offset_start_+_size  <- are we done?
     jle  xor_loop            <-   no, then next cycle
     ret                      <-   back where we came from
```

کمیت یا عدد سری کردن همیشه یکجور است (ثابت) یا بطور تصادفی انتخاب می‌شود، مثلاً با استفاده از INT21، سرویس 2ch (بدست آوردن زمان فعلی) و انتخاب کردن بعنوان عدد سری، این رقم در DL گزارش شده است (اما به خاطر داشته باشید که عدد نهایی صفر را کنار بگذارید. در غیر اینصورت، عمل اپراتور XOR نیز انجام نخواهد شد.

عدد تصادفی:

```
     mov  ah,2Ch
     int  21h
     cmp  dl,0
```

```
je   random_value
mov  encrypt_value,dl
```

مشکل موجود در رابطه با متد XORing (و سایر روشهای سری کردن) این است که بخشی از کد که روتین سری سازی را فراخوانی می‌کند، خود نمیتواند سری شود. بهر جهت، بـــاید کلیـد سـری سازی مشخصی را داشته باشید.

محافظه کاران، بهترین راه را، مخفی نمودن روتین سری کردن (کدگذاری) میداننـد، متدهـای رایـج دیگری هم وجود دارند.

Junk Filling ،کلیدهای اسلایدینگ، آشکارسازهای جهش دهنده.

این متدها محافظت رایجتری برای بخش کوچک کشف رمز کد برنامه هستند.

این متدها، که در اصل برای اسکنرهای ویروس برنامه‌های احمقانه توصیـــه شـده، از پردازنده‌هـای ویروس پلی مورف نویسندگان ویروسها، گرفته شده و هنوز هم از آنها برای بســیاری از طرحـهای ساده محافظه کشف رمز استفاده می‌شود. در مورد بخش بعدی، لازم است مراتب تشکر خـــود را از (بلاک بارون) اعلام دارم، بسی جـــای تأسـف اسـت کـه بسـیاری از کرکرهـای بـالقوه خـوب، بـا اختصاص دادن وقت خود به نوشتن ویروس بی مصرف وقت خود را تلف میکنند. بنابراین، مطالعه ویروس برای کراکرها، بسیار حائز اهمیت می‌باشد. کد ویروسها عبارت است از:

- فوق حفاظت شده
- محکم و مؤثر
- مخفیانه و سری

حال، برای نشان دادن نمونه تاکتیکهای حفاظت فوق‌الذکر، به ذکر یک مثال از آشکارساز بسیار ساده می‌پردازیم:

```
MOV      SI,jumbled_data     ;Point to the jumbled data
```

```
          MOV     CX,10               ;Ten bytes to decrypt
mn_loop:  XOR     BYTE PTR [SI],44    ;XOR (un_scramble!) a byte
          INC     SI                  ;Next byte
          LOOP    mn_loop             ;Loop the 9 other bytes
```

این برنامه کوچک، ده بایت به محل (آدرس) ذکر شده توسط SI، با رمز ٤٤ را XOR خواهد کرد. با فراهم نمودن ده بایت XOR شده با رمز ٤٤ قبل از آنکه این آشکارساز برنامه خود را اجرا کند، این ده بایت، مجدداً در محل اصلیشان ذخیره میشوند.

در این مورد بسیار ساده، کلید (رمز) عدد ٤٤ است. اما ترفندهای دیگری هم هست که به کلیدها نیاز دارند. ساده ترین آنها، استفاده از کلید یا رمز «اسلایدینگ» است:

رمزی که در هر گذر حلقه (LOOP)، زیاد، کم، یا چندبرابر و تغیر بیت و یا هر گونه عملیات دیگری روی آن انجام می‌شود.

در هر محافظت ممکن هم میتوان یک آشکارساز پلی مورف بوجود آورد. روتین آشکارسازی که کاملاً در هر نسل متفاوت به نظر میرسد ترفند اینست: پر کردن از مقادیر تصادفی دستورالعملهای کلاً تصادفی، که شامل Jumps و Calls بوده و بر روی شبکه‌هایی که برای کشف رمز استفاده میشوند هیچ تأثیری نمیگذارند. همچنین،این نوع حفاظت، غالباً از آشکارساز اصلی متفاوتی استفاده نموده (احتمالاً از پیش کدگذاری شده) و در هر نسل فرق می‌کند که کد Junk را که آن بوجود می‌آورد، هیچ یک از ثبتهای مورد استفاده آشکارساز واقعی را خراب نمی‌کند. بنابراین، با وجود چنین قوانینی، آشکارساز ساده‌ای وجود خواهد داشت.

```
          MOV     DX,10               ;Real part of the decryptor!
          MOV     SI,1234             ;junk
          AND     AX,[SI+1234]        ;junk
```

```
        CLD                         ;junk
        MOV     DI,jumbled_data     ;Real part of the decryptor!
        TEST    [SI+1234],BL        ;junk
        OR      AL,CL               ;junk
mn_loop: ADD    SI,SI               ;junk instr, but real loop!
        XOR     AX,1234             ;junk
        XOR     BYTE PTR [DI],44    ;Real part of the decryptor!
        SUB     SI,123              ;junk
        INC     DI                  ;Real part of the decryptor!
        TEST    DX,1234             ;junk
        AND     AL,[BP+1234]        ;junk
        DEC     DX                  ;Real part of the decryptor!
        NOP                         ;junk
        XOR     AX,DX               ;junk
        SBB     AX,[SI+1234]        ;junk
        AND     DX,DX               ;Real part of the decryptor!
        JNZ     mn_loop             ;Real part of the decryptor!
```

همانطور که می‌بینید. کاملاً بهم ریخته است! اما هنوز هم کد قابل اجرایی است . لازم است تا هر کد Junk ایجاد شده به وسیله محافظت پلی مورف، قابل اجرا باشد. همانگونه که توسط آشکارساز پر می‌شود. در این مثال، برخی از دستورالعملهای Junk از ثبتهایی استفاده میکنند که واقعاً در آشکارساز استفاده میشوند. مشروط بر اینکه اعداد در این ثبتها از بین نروند. همچنین قابل ذکر است که هم اکنون دارای ثبتهای تصادفی و دستورالعملهای تصادفی در هر نسل هستیم. بنابراین، یک موتور محافظت پلی مورف، می‌تواند در سه بخش عمده خلاصه شود:

۱- ... ژنراتور عدد تصادفی

۲- ..ژنراتور کد Junk

۳- ...ژنراتور آشکارساز

بخشهای دیگری هم وجود دارند اما سه بخش فوق، بخشهایی هستند که بیشتر کارهـا بـا آنـها ادامه می‌یابد.

این موتور چگونه کار می‌کند؟ خوب، یک حفاظت خوب باید یک گزینه تصادفی از ثبتها برای استفاده آشکارساز انتخاب کند،و ثبتهای باقیمانده دیگر را بعنوان ثبتهای Junk برای ژنراتور کـد Junk جا بگذارد.

- یکی از آشکارسازهای از پیش کدگذاری شده فشرده را انتخاب کنید.
- وارد حلقه تولید کننده آشکارساز واقعی شوید که با کد Junk کار می‌کند.

از دیدگاه محافظه کاران، مزایای این متد عبارتند از:

- کراکر تصادفی، با زحمت زیاد، آشکار ساز را پیدا می‌کند.
- کراکر تصادفی نمیتواند یک «وصله» برای لامرها فراهم کند، مگر اینکه ژنراتورها را در یک محل قرار داده و آنها را بهم متصل کند. در غیر اینصورت آشکار ساز لحظه به لحظه تغیـیر خواهد کرد. به منظور رد کردن این نوع محافظت، نیاز بـه کمـی حسـن Zen و اطلاعـات جزیی در مورد اسمبلر خواهیـد داشـت. وقتـی بـه برخـی از دسـتورالعملهای Junk نـگاه میکنید، میبینید که برخی از آنها کاملاً عجیب و غریبند. (به درس B مراجعه کنید). بعـلاوه، هم اکنون ممکن است چه اتفاقی بیفتد و نقـاط توقـف حافظـه، بلافاصلـه در کشـف رمـز متوقف خواهند شد. راه باز است و ادامه آن آسان.

جادوی عدد نقطه شروع

مثلاً اگر بگوییم، کد سری شده در آدرس 10h شروع شده، آدرس ذیل را میتوان به ایـن آدرس ضمیمه نمود:

```
MOV   SI,10h          ;Start address
```

```
MOV   AL,[SI]         ;Index from initial address
```

گاهی اوقات ،به جای این آدرس، آدرسی را شبیه به آن خواهید یافت، که بر اساس کـد سـری شده‌ای می‌باشد که در آدرس 10H شروع می‌شود:

```
MOV   DI,0BFAAh       ;Indirect start address
MOV   AL,[DI+4066h)   ;4066h + 0BFAAh = 10010h (and FFFF = 10h)!!
```

ترکیبهای بیشمار دیگری نیز، امکانپذیر است.

(کلید اصلی) (روشهای دشوار سری کردن)

فاکتورگیری عدد پریم، به طورسری برای محافظت از داده‌هـای حسـاس و کاربردهـای بسـیار گران استفاده شده است. مسلماً، برای رمزهای رقمی (عددی)، رمزگشـایی،اسـانتر از کلیدهـای رقمی (عددی) ۱۲۹ یا ۲۵۰ می‌باشد. با این وجود، شما میتوانید آن سری کردن عظیم را هم بـا استفاده از پردازش توزیع شده معادلات درجه دو، به منظـور متوقـف سـاختن کلیـد در اعـداد پریم، کراک کنید (که به منظور کراک کردن با متدهای پردازش ترتیبی، بهتر است).

methods) in order to break the key into prime numbers. To teach you how to do this sort of "high" cracking is a little outside the scope of my tutorial: you'll have to write a specific short dedicated program, linking together more or less half a thousand PC for a couple of hours, for a 250 bit key, this kind of things have been done quite often on Internet, were you can also find many sites that do untangle the mysteries (and vagaries) of such techniques.

As References I would advocate the works of Lai Xueejia, those swiss guys can crack *everything*. Begin with the following:

Xuejia Lai, James Massey, Sean Murphy, "Markov Ciphers and Differential Cryptanalysis", Advances in Cryptology, Eurocrypt 1991.

Xuejia Lai, "On the Design and Security of Block Ciphers", Institute for Signal and Information Processing,

ETH-Zentrum, Zurich, Switzerland, 1992
Factoring and primality testing is obviously very important for
this kind of crack. The most comprehensive work I know of is:
(300 pages with lengthy bibliography!)
W. Bosma & M. van der Hulst
Primality Testing with Cyclotomy
Thesis, University of Amsterdam Press.
A very good old book you can incorporate in your probes to build
very effective crack programs (not only for BBS accesses :=) is
the "pomerance" catalog:
Pomerance, Selfridge, & Wagstaff Jr.
The pseudoprimes to 25*10^9
Math. Comp. Vol 35 1980 pp. 1003-1026
Anyway... make a good search with Lykos, and visit the relevant
sites... if encryption really interests you, you'll be back in
two or three (or thirty) years and you'll resume cracking with
deeper erudite knowledge.
[PATENTED PROTECTION SYSTEMS]
The study of the patented enciphering methods is also *quite*
interesting for our aims :=) Here are some interesting patents,
if you want to walk these paths get the complete texts:

بعنوان مرجع، به شما اثرهای Lai xueejia را معرفی می‌کنیم، با آنها میتوان هر چیزی را کـراک کرد.

(سیستم های محافظت ثبت شده)

تحقیق و بررسی روشهای ثبت شده سری کردن، برای اهداف ما کاملاً جذاب و جالب است.

=: در این قسمت ثبتهای جالبی برایتان ذکر می‌کنیم، اگر میخواهید در این مسـیر گام بردارید، باید نسخه‌های کامل آن را تهیه کنید.

USPat 4168396 ،متعلق به سیستم Best ،ریزپردازنده‌ای برای اجرای برنامه‌های سری شده را نشان میدهد. برنامه‌های کامپیوتری، که در طی ساخت به منظور تعیین اجرای برنامه‌ها در کامپیوترهای غیرمجاز سری میشوند، باید قبل از اجرای برنامه، داده‌هایشان افشا میشد. ریزپردازنده افشا شده. برنامه سری شده را کراک نموده و سپس آن را با یک دستورالعمل،از طریق مجموعه‌ای از جایگزینی‌ها و ضمیمه‌های خاص OR، اجرا می‌کند. بنحویکه آدرس هر دستورالعملی، با این دستورالعمل، تلفیق می‌شود.

هر واحدی از یک مجموعه جداگانه جایگزینیها استفاده می‌کند. به گونه‌ای که، برنامه‌ای که توسط یک ریزپردازنده اجرا می‌شود، ریزپردازنده دیگر قادر به اجرای آن نخواهد بود. بعلاوه، Best نمیتواند مخلوطی از برنامه‌های سری شده و پیامهای عادی در شکل معنادار اصلی آن را در خود جای دهد.

USP[JOHNSTONE] با شماره ثبت 4120030 متعلق به Johnstone، کامپیوتری را شرح میدهد که در آن بخش دستورالعملهای داده‌ها، به رمز درآمده بطوریکه داده‌ها،الزاماً در حافظه جداگانه‌ای ذخیره می‌شود. در این سیستم اصلاً از نحوه کار کردن با دستورالعملهایی، که کاملاً با کد عملیاتی و بخش آدرس داده سری شده‌اند، و بدون هسته کلیدی قابل خواندن نیستند، چیزی عنوان نشده است.

USPat 4183085(TWINPROGS)، تکنیکی برای نرم افزار حمایت کننده از طریق تهیه دو حافظه جداگانه برنامه، را بیان می‌کند. اولین حافظه برنامه، یک حافظه (امن) حفاظت شده بوده و دومین حافظه،یک حافظه ازاد می‌باشد. منطق حفاظتی، برای کنترل اینکه آیا یک دستورالعمل خروجی در حافظه امن بوجود آمده و همچنین برای جلوگیری از عمل یک واحد خروجی که دستورالعملهای خروجی را از حافظه آزاد دریافت می‌کند، فراهم شده است.

این باعث می‌شود تا تولید اطلاعـات از طریـق بـارگذاری یـک برنامـه در حافظـه آزاد قـدری مشکلتر شود.

[Authenticator]USPat 3996449 با عنوان سیستم عامل Authenticator تکنیکی برای تـأیید صحت و اعتبار برنامه پیامهای عادی در شکل معنادار اصلی آن، می‌باشد که در کامپیوتر خوانده می‌شود، آنهم از طریق عمل OR پیامهای عادی برنامه در شکل معنادار اصلی آن با کلیدی برای ایجاد کلمه رمز،که این کلمه باید کلمه رمز استاندارد قابل تشـخیص باشـد کـه بتـوان آن را بـا موفقیت با کلمه رمز مطابق استاندارد، که در کامپیوتر ذخیره شده، مقایسه نمود.

اگر مقایسه با موفقیت صورت گیرد،پیامهای عادی در شکل معنادار اصلی برنامه، تـأیید شـده و اجرای آن مجاز خواهد بود، در غیر اینصورت، اجرای برنامه، مجاز نیست.

عوامل کراکینگ [PGP]

به منظور کراک کردن PGP، باید طرز کار سیستم کلیدهای عمومی / خصوصی را یـاد بگیریـد. به نظر میرسد کراکینگ PGP، مشکل باشد . من یک کـامپیوتر اختصـاصی attack دارم کـه بـه همین منظور. ٢٤ ساعت کار می‌کند و فقط زمانی کار خود را شروع می‌کند هدفش رسـیدن بـه نقطه مشهور مورد نظرش است.

این کار سخت است، اما کراکرهای خوب هرگز، به آن تن نمیدهند.

در سیستمهای رمزی با کلید عمومی، مثل PGP،هـر اسـتفاده کننـده، دارای کلیـد سـری کـردن مربوطه E=(e,n) و کلید آشکارسازی D=(d,n) می‌باشد، که در آن کلیدهای سری کـردن بـرای تمام استفاده‌کنندگان در فایل عمومی مهیاست. در حالیکه کلیدهای آشکارسازی بـرای اسـتفاده کننده‌های خاص در نظر گرفته شده است. به منظور ایجاد امنیـت در سـطح بـالا، کلیـد کـراک

استفاده کننده، را نمیتوان به روش خاصی با توجه به کلید کـدگذاری اسـتفاده کننـده مشـخص نمود. بطور کلی، در چنین سیستمهایی، با توجه به :

```
e.multidot.d.ident.1 (mod(1 cm((p-1),(q-1)))),
(where "1 cm((p-1),(q-1))" is the least common multiple of the
numbers p-1 and q-1)
```

d را میتوان از روی e مشخص کرد، p و q تعیین شده‌اند. بـر ایـن اسـاس، امنیـت سیسـتم بـه قدرت تعیین pو q بستگی دارد که فاکتورهای پریم n هستند. با انتخاب pو q بعنوان پریمـهای بزرگ، برآیند عدد n نیز بزرگ بوده و به همین نسبت، فاکتور دیگری نیز مشـکل خواهـد بـود مثلاً استفاده از روشهای معروف تجزیه عاملی (فاکتوریزاسـیون) انجـام شـده در کـامپیوتر، بـه فاکتور یک عدد طولانی ۲۰۰ رقمی نیاز دارد. پس اگرچه کلیـد سـری سـازی E(e,n) اسـتفاده کننده، کلید عمومی است , فاکتورهای پریم pو qی n به طرز مؤثری بخـاطر وجـود مشـکلات عمده در فاکتورگیری n، از همه مخفی نگاه داشته می‌شود. چنیـن جنبـه‌هایی، بطـور کـامل در نشریه‌های فراوانی پیرامون امضاهای دیجیتالی و سیستمهای رمزی با کلید عمومـی، شـرح داده شده‌اند . اکثر سیستمهای عمومی یا خصوصی به الگوریتم خلاصه پیام، تکیه میکنند.

الگوریتم خلاصه پیام، پیام دارای طول دلخواه را بصورت خلاصه بـا طـول ثـابت و مشـخصی درآورده و دارای سه ویژگی می‌باشد.

محاسبه خلاصه، کار ساده‌ای است . یافتن یک پیام با تغییر خلاصه مشخص شده، کـار سـختی است و یافتن دو پیام با تصادم خلاصه یکجور نیز کار دشواری است .

الگوریتمهای خلاصه پیام، کاربردهیا زیادی علاوه بر امضاهای دیجیتالی و تأیید پیام دارند.

با الگوریتم خلاصه پیام MD5 محافظت داده RSA، توسـط رون ریوسـت، میتـوان پیـام را بـه خلاصه پیام ۱۲۸ بیت تبدیل کرد. محاسبه خلاصه یک پیام یک مگابـایتی، بصـورت یـک پیـام

کوچک و تحت عنوان پیـــام دوم، صـورت میگیـرد. در صورتیکـه هیـچ یـک از الگوریتمـهای خلاصه پیام، محافظت شده نباشند. MD5 معتقد است که تبدیـــل پیـام بـه خلاصـه ۱۲۸ بیـت نسبت به سایر الگوریتمها بهتر است.

بعنوان هدیه آخر، برایتان میگویم که PGP برای عمل مطمئن یک جانبه، به MD5 تکیه می‌کند. این برای PGP بسیار مسئله ساز است و حداقل میتوان گفت، ارتباط نزدیکی بین ٤ تـا ثابتـهای جمع پذیر متوالی وجود دارد. این یعنـی، یکـی از اصـول و ضوابـط ایـن طـرح بدنبـال MD4 (MD5) ، یعنی طـــرح ریـزی عمـل مخـالف تصـادم. مـورد قبـول نیســت. شـما میتوانیـد، دو متغیرزنجیره‌ای و یک بلوک جداگانـــه پیـام بسـازید کـه کـد اختـلاط (برنامه‌نویسـی اختـلاط) یکجوری داشته باشد. Attack روی PC چند دقیقه وقت خواهد گرفت. از اینجـا بـاید شـروع کنید، کاری که من میکنم. [DOS 4GW] کراکینگ ـ این تنها بخش بسیار مشروط ایـــن برنامـه آموزشی است .

کراکینگ DOS 4GW را به گونه بهتری برایتان توضیح خواهیم داد.

اکثر برنامه‌های هر OS، و همچنین DOS 4GW به زبان C نوشته شده، فقط باید ایـــن زبـان را یاد بگیرید. تنها C به شما اجازه میدهد تا محتویات این برنامه را پیدا کنید، البتـه زبـان اسـمبلر نیز بی تأثیر نیست.

پس وقتیکه برنامه کاربردی (tool) خود را آماده نمودیـد و مسـتقیماً هـم از روتینـهای اسـمبلر استفاده نمیکنید، C می‌تواند زبان انتخاب برای کراکرها باشد. پس میتوانید کتابهای بسیار خوبی درمورد C پیدا کنید.

بیشتر استفاده کنندگان. هزینه زیادی صرف خریدن کتابهای بی استفاده و بی فایده فقـط بـزرگ و رنگی میکننـد. کتابـهای جدیـد C (بـه زبـان اسـمبلر) بسـیار کمیـابند. آنـها را پیـدا کنیـد،

بخرید،بخوانید و برای رسیدن به اهدافتان از آنها استفاده کنید. میتوانید برنامه‌های آموزشی C و مطالب C روی وب، به هر شیوه‌ای که انجام میدهید، را پیدا کنید.

C را یاد بگیرید. بسیار ارزان و مطمئن است.

به اصل مطلب برگردیم: اکثر مطالب به زبان C نوشته شده و لازم است تا زیرروالهای اصلی در اسمبلر را پیدا کنید. با استفاده از برنامه‌های DOS 4GW ، فایل EXE برای "90 90 90 90 90" را پیدا کنید، این فایل معمولاً در شروع کد کامپایل مشاهده می‌شود . حالا یک INT 21 اجرا شده با 4C در AH . EXEC با کد DOS را سرچ کنید.

(اگر با دستگاه خود نمیتوانید AH=4C BPINT 21 را سرچ کنید. به سرچ کردن توالی b4 4c CD 27 بپردازید. این مساویست با [Mov AH, 4C & int 21] :

این دقیقترین فراخوان است، اما به محض اینکه یاد گرفتید، راهـهای بسـیاری بـرای قـرار دادن 4C در AX وجود دارد. آنها را به ترتیب فرکانسشان امتحان کنید).

با وجود بایتهای موجود در بالای وقفه ۲۱، سرویس 4C، میتوانید زیرروالهای «اصلـی» را پیـدا کنید: "E8 xx xx" حالا، بایت "CC" را، چند بایت بالاتر از فراخوانی در exe قرار داده و فـایل exe را تحت اشکال زدا اجرا کنید.

موقعیکه کامپیوتر سعی می‌کند تا دستورالعمل را اجرا کند، به اشکال زدا مراجعه نموده و بـایت "CC" بعنوان دستورالعمل INT 01 عمل می‌کند. بقیه مراحل مثل همیشه است.

مخفی گاه کلمه رمز STEGONATED

آخرین ترفند بسیار خوب، را توضیح میدهم. سرچ کردن کلمه رمز یا روتنیهای محافظتی که به ظاهر وجود ندارند کمی نگران کننده است . آنها باید درون یک تصویر،پنهان شـوند. (یـا یـک

فایل WAW.*مربوط به آن موضوع) این استگانوگرافی،است،روش تشخیص پیامها در سایر رسانه‌های دیگر.

بسته به اینکه چند تا سایه خاکستری یا رنگهای رنگی میخواهید داشته باشید، یک پیکسل را میتوان با استفاده از ۸، ١٦، ۳۲ یا تعداد بیشتری بیت نشان داد. در صورتیکه، بیت کم اهمیتی تغییر داده شود،سایه پیکسل فقط با ١/٢٥٦، ١/٦٥٠٠٠ بیت یا کمتر، نشان داده می‌شود.

با استفاده از چشم نمیتوان تفاوت آن را تشخیص داد.

آنچه محافظه کاران یا حمایت گرایان انجام میدهند، به سرقت بردن بیت کم اهمیت در هر پیکسل تصویر است . او این بیت را برای ذخیره نمودن بیت محافظت یا کلمه رمز (یا یک فایل، یا یک پیام رمزی) مورد استفاده قرار میدهد. چون، تصاویر دیجیتالی شده، دارای پیکسل زیادی میباشند، امکان ذخیره تعداد زیادی از داده‌ها در یک تصویر جداگانه وجود دارد.

یک الگوریتم ساده، آنها را در صورت نیاز به قسمتهای مربوط به برنامه، منتقل می‌کند و در آنجا، ما آنها را مخفیانه مشاهده خواهیم نمود. برای شما لازم است تا تکنیکهای Zen-cracking را بخوبی یاد بگیرید تا بتوانید چنین چیزهای به درد نخوری را فوراً بشناسید.

خب، این بود درس امروز. اما همه خودآموزهای من روی اینترنت نیستند، شما درسهایی را که فراموش کرده‌اید 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 می‌فهمم 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.

E-mail +OR

an526164@anon.penet.fi (+ORC)