به نام خدا

مهندسی معکوس نرم‌افزار

تألیف: سید داود ملک حسینی

(گروه امنیتی هامان)

انتشارات پندار پارس

سرشناسه	:	ملک‌حسینی، سید داود، ۱۳۶۷-
عنوان و نام پدیدآور	:	مهندسی معکوس نرم‌افزار/ تالیف داود ملک‌حسینی.
مشخصات نشر	:	تهران : پندار پارس ، ۱۳۹۸.
مشخصات ظاهری	:	۲۷۰ ص.: مصور، جدول.
شابک	:	978-600-8201-66-3
وضعیت فهرست نویسی	:	فیپا
یادداشت	:	کتابنامه.
موضوع	:	مهندسی معکوس-- نرم‌افزار
موضوع	:	Reverse engineering-- Software
رده بندی کنگره	:	TA۱۶۸/۵/م۶۶م۹ ۱۳۹۷
رده بندی دیویی	:	۰۰۴۲۰۲۸۵/۶۲۰
شماره کتابشناسی ملی	:	۵۴۶۱۴۰۰

انتشارات پندارپارس

دفتر فروش: انقلاب، ابتدای کارگرجنوبی، کوی رشتچی، شماره ۱۴، واحد ۱۶ **www.pendarepars.com**

تلفن: ۶۶۵۷۲۳۳۵ - تلفکس: ۶۶۹۲۶۵۷۸همراه: ۰۹۱۲۲۴۵۲۳۴۸ **info@pendarepars.com**

نام کتاب	:	**مهندسی معکوس نرم‌افزار**
ناشر	:	**انتشارات پندار پارس**
تالیف	:	**سید داود ملک حسینی**
چاپ نخست	:	**بهمن ۹۸**
شمارگان	:	**۵۰۰ نسخه**
طرح جلد	:	**سارا یعسوبی**
چاپ، صحافی	:	**روز**
قیمت	:	**۲۵۰.۰۰۰ تومان**

شابک : ۳-۶۶-۸۲۰۱-۶۰۰-۹۷۸

فهرست

پیشگفتار مؤلف

کتابی که در دستان شماست شامل کلیدی‌ترین و تخصصی‌ترین مباحث مهندسی معکوس نرم‌افزار در شاخه کرک می‌باشد که پس از تست کامل و دقیق همه مثال‌ها و دستورات مندرج در کتاب، اقدام به چاپ آن نمودیم. جلد دوم آن نیز در دست آماده‌سازی است که امیدوارم به زودی به عرصه چاپ و نشر برسد.

اینجانب سید داود ملک حسینی دارای بورد تخصصی هک و امنیت و مهندسی معکوس نرم‌افزار می‌باشم و نیز مسلط به زبان‌های برنامه‌نویسی سطح بالا و سطح پایین و متخصص در شبکه‌های مخابراتی و ارتباطی هستم. این کتاب، حاصل دانش چندین ساله و سال‌ها تجربه‌ام در این زمینه است و آنرا به تمام اعضای جامعه مهندسی کامپیوتر تقدیم می‌کنم و امیدوارم روزی کشور عزیزمان به سمت پیشرفت واقعی در مسیر علوم کامپیوتر برود.

جا دارد از همه عزیزانی که طی این سال‌ها حامی و پشتیبانم بوده‌اند، به‌ویژه پدر و مادر گرانقدرم تقدیر و تشکر نمایم. همچنین، بدون انجام زحمات سرکار خانم مهندس مریم قارونی، فارغ التحصیل کارشناسی ارشد امنیت اطلاعات از مالزی، این کتاب به مرحله تألیف و چاپ نمی‌رسید و لازم می‌دانم از ایشان قدردانی کنم و امیدوارم در تمام مراحل زندگی موفق و پیروز باشند.

به امید رسیدن به فردایی بهتر و ایرانی پیشرفته و قدرتمند

در پناه حق

سید داود ملک حسینی

گروه امنیتی هامان

بخش نخست

کیژن (Keyggen)

فصل نخست

دو روش در ساخت کیژن مربوط به الگوریتم MD5

مقدمه

مجاز هستید همه کدهای ارائه شده در این آموزش را استفاده کرده و تغییر دهید. تنها از شما می‌خواهیم به مکانی که آنها را پیدا کرده‌اید اشاره‌ای داشته باشید. به علاوه، این آموزش با تمام الحاقیات و مکمل‌هایش و در همین قالب فعلی و بدون تغییر، جهت توزیع، مجاز و آزاد محسوب می‌شود.

تمام برنامه‌های تجاری استفاده شده در این آموزش تنها به جهت اثبات راه کارها و تئوری‌های موجود، آورده شده‌اند و هیچ توزیعی از نرم‌افزارهای وصله نرم‌افزاری[۱] شده تحت هیچ رسانه‌ای[۲] انجام نشده است. نرم‌افزارهای استفاده شده بیشتر وقت‌ها توسط دیگر افراد، قبلا وصله نرم‌افزاری شده‌اند و نسخه‌های کرک شده آنها از مدت‌ها پیش قابل استفاده و دسترسی بوده‌اند. گروه "هامان" یا نویسندگان این کتاب نباید بابت خسارت وارد شده به حقوق شرکت‌های آن برنامه‌ها مسئول شناخته شوند. هدف اصلی این مطلب همانند دیگر آموزش‌های گروه هامان، انتشار و اشتراک دانش و آموزش چگونگی پچ کردن نرم‌افزارها، نحوه دور زدن محافظت‌های مختلف و به طور کلی توضیح این است که چگونه هنر مهندسی معکوس را ارتقاء دهیم. در واقع ما هیچ نرم‌افزار کرک شده‌ای را منتشر نمی‌کنیم.

---

[1] patch
[2] media

کیژن MD5

۱-۱ پیشگفتار

در طول زمان برنامه‌ها و الگوریتم‌های محافظتی زیادی آمدند و رفتند و هر کدام به نوعی قابل شکسته شدن و آنپک شدن بودند اما در این میان تکنیک کیژن برای ما راهکاری جدید و جدی محسوب می‌شد. این آموزش راه کاملی را به شما نشان نمی‌دهد در واقع افرادی که مقداری دانش در زمینه مهندسی معکوس دارند قادرند که این موضوع محافظت را متوجه شوند اما اگر شما یک تازه وارد هستید این آموزش نمی‌تواند مناسب شما باشد.

ابزارهای مورد استفاده عبارتند از یک pc و نسخه v1.10 نرم‌افزار OllyDebug.

۱-۲ پیش از شروع

در آموزش ارائه شده، خواهید دید که توضیحاتی را در مورد دیس اسمبل[1] ارائه می‌دهیم. در این حالت نیاز است که نرم‌افزار هدف ما از الگوریتم MD5 hashing استفاده کند. اما چگونه متوجه شویم؟ در ابتدا نیاز است که برنامه را خط به خط دنبال کنیم و نهایتا به قطعه‌های مشخصی از کد خواهیم رسید که شما را به سمت جوابتان راهنمایی می‌کنند.

در زیر نخستین نشانه را که از RFC1351 گرفته شده است (http://tools.ietf.org/htm/rfc1321) ببنید:

```
00CFA540  $  8B4424 04     MOV EAX,DWORD PTR SS:[ESP+4]
00CFA544  .  33C9          XOR ECX,ECX
00CFA546  .  C700 0123456  MOV DWORD PTR DS:[EAX],67452301
00CFA54C  .  C740 04 89AB  MOV DWORD PTR DS:[EAX+4],EFCDAB89
00CFA553  .  C740 08 FEDC  MOV DWORD PTR DS:[EAX+8],98BADCFE
00CFA55A  .  C740 0C 7654  MOV DWORD PTR DS:[EAX+C],10325476
00CFA561  .  8948 10       MOV DWORD PTR DS:[EAX+10],ECX
00CFA564  .  8948 14       MOV DWORD PTR DS:[EAX+14],ECX
00CFA567  .  8948 58       MOV DWORD PTR DS:[EAX+58],ECX
00CFA56A  L. C3            RETN
```

[1] Disasmble

```
/* MD5 initialization. Begins an MD5 operation, writing a new context.
 */
void MD5Init (context)
MD5_CTX *context;                                        /* context */
{
  context->count[0] = context->count[1] = 0;
  /* Load magic initialization constants.
*/
  context->state[0] = 0x67452301;
  context->state[1] = 0xefcdab89;
  context->state[2] = 0x98badcfe;
  context->state[3] = 0x10325476;
}
```

این هم از اثبات نهایی:

```
00CFA570 r$  8B5424 0C     MOV EDX,DWORD PTR SS:[ESP+C]
00CFA574 .   8B4C24 04     MOV ECX,DWORD PTR SS:[ESP+4]
00CFA578 .   53            PUSH EBX
00CFA579 .   8BC2          MOV EAX,EDX
00CFA57B .   8B59 08       MOV EBX,DWORD PTR DS:[ECX+8]
00CFA57E .   55            PUSH EBP
00CFA57F .   8B69 04       MOV EBP,DWORD PTR DS:[ECX+4]
00CFA582 .   4A            DEC EDX
00CFA583 .   56            PUSH ESI
00CFA584 .   8B71 0C       MOV ESI,DWORD PTR DS:[ECX+C]
00CFA587 .   85C0          TEST EAX,EAX
00CFA589 .v  0F84 C606000( JE 00CFAC55
00CFA58F .   8B4424 14     MOV EAX,DWORD PTR SS:[ESP+14]
00CFA593 .   57            PUSH EDI
00CFA594 .   83C0 38       ADD EAX,38
00CFA597 .   42            INC EDX
00CFA598 .   895424 18     MOV DWORD PTR SS:[ESP+18],EDX
00CFA59C >   8B78 C8       MOV EDI,DWORD PTR DS:[EAX-38]
00CFA59F .   8BD6          MOV EDX,ESI
00CFA5A1 .   33D3          XOR EDX,EBX
00CFA5A3 .   23D5          AND EDX,EBP
00CFA5A5 .   33D6          XOR EDX,ESI
00CFA5A7 .   03D7          ADD EDX,EDI
00CFA5A9 .   8BFB          MOV EDI,EBX
00CFA5AB .   8BF2          MOV ESI,EDX
00CFA5AD .   8B11          MOV EDX,DWORD PTR DS:[ECX]
00CFA5AF .   33FD          XOR EDI,EBP
00CFA5B1 .   8D9416 78A46( LEA EDX,DWORD PTR DS:[ESI+EDX+D76AA478]   Look at this
00CFA5B8 .   8B70 CC       MOV ESI,DWORD PTR DS:[EAX-34]
00CFA5BB .   C1C2 07       ROL EDX,7
00CFA5BE .   03D5          ADD EDX,EBP
00CFA5C0 .   23FA          AND EDI,EDX
00CFA5C2 .   33FB          XOR EDI,EBX
00CFA5C4 .   03FE          ADD EDI,ESI
00CFA5C6 .   8B71 0C       MOV ESI,DWORD PTR DS:[ECX+C]
00CFA5C9 .   8DB437 56B7C( LEA ESI,DWORD PTR DS:[EDI+ESI+E8C7B756]   Look at this
00CFA5D0 .   8BFD          MOV EDI,EBP
00CFA5D2 .   C1C6 0C       ROL ESI,0C
00CFA5D5 .   03F2          ADD ESI,EDX
00CFA5D7 .   33FA          XOR EDI,EDX
00CFA5D9 .   23FE          AND EDI,ESI
00CFA5DB .   33FD          XOR EDI,EBP
00CFA5DD .   8B68 D0       MOV EBP,DWORD PTR DS:[EAX-30]
00CFA5E0 .   03FD          ADD EDI,EBP
00CFA5E2 .   8BEE          MOV EBP,ESI
00CFA5E4 .   33EA          XOR EBP,EDX
00CFA5E6 .   8DBC1F DB702( LEA EDI,DWORD PTR DS:[EDI+EBX+242070DB]   Look at this
```

این شکل نیز از RFC1351 گرفته شده است.

```
/* MD5 basic transformation. Transforms state based on block.
 */
static void MD5Transform (state, block)
UINT4 state[4];
unsigned char block[64];
{
  UINT4 a = state[0], b = state[1], c = state[2], d = state[3], x[16];

  Decode (x, block, 64);

  /* Round 1 */
  FF (a, b, c, d, x[ 0], S11, 0xd76aa478); /* 1 */
  FF (d, a, b, c, x[ 1], S12, 0xe8c7b756); /* 2 */
  FF (c, d, a, b, x[ 2], S13, 0x242070db); /* 3 */
  FF (b, c, d, a, x[ 3], S14, 0xc1bdceee); /* 4 */
  FF (a, b, c, d, x[ 4], S11, 0xf57c0faf); /* 5 */
  FF (d, a, b, c, x[ 5], S12, 0x4787c62a); /* 6 */
  FF (c, d, a, b, x[ 6], S13, 0xa8304613); /* 7 */
  FF (b, c, d, a, x[ 7], S14, 0xfd469501); /* 8 */
  FF (a, b, c, d, x[ 8], S11, 0x698098d8); /* 9 */
  FF (d, a, b, c, x[ 9], S12, 0x8b44f7af); /* 10 */
  FF (c, d, a, b, x[10], S13, 0xffff5bb1); /* 11 */
  FF (b, c, d, a, x[11], S14, 0x895cd7be); /* 12 */
  FF (a, b, c, d, x[12], S11, 0x6b901122); /* 13 */
  FF (d, a, b, c, x[13], S12, 0xfd987193); /* 14 */
  FF (c, d, a, b, x[14], S13, 0xa679438e); /* 15 */
  FF (b, c, d, a, x[15], S14, 0x49b40821); /* 16 */
```

این مدرکی است که ثابت می‌کند ما در نخستین دور از MD5 هستیم. همچنین می‌توانید از پلاگین IDA findcrypt نیز استفاده کنید. البته ابزارهای دیگری نیز وجود دارند که به صورت مشابه عمل می‌کنند اما اینکه شما قادر باشید خودتان MD5 را تشخیص دهید در وقت‌تان صرفه جویی کرده‌اید.

۳-۱ آمادگی لازم برای آغاز کار

فایل drg2pdf.exe را اجرا کنید و به قسمت Registration dialog بروید. در شکل زیر پنجره‌ای را می‌بینید.

وقتی قسمت نام (name) و سریال (serial) را پر کردید دکمه ok را بزنید. پیامی[1] را دریافت خواهید کرد با این مضموم که سریال نامعتبر است. بنابراین باید بریک‌پوینت‌هایی[2] را در MessageBOXA و MessageBOXEXA و MessageBOXEXW قرار دهیم و دوباره برنامه را اجرا کنیم.

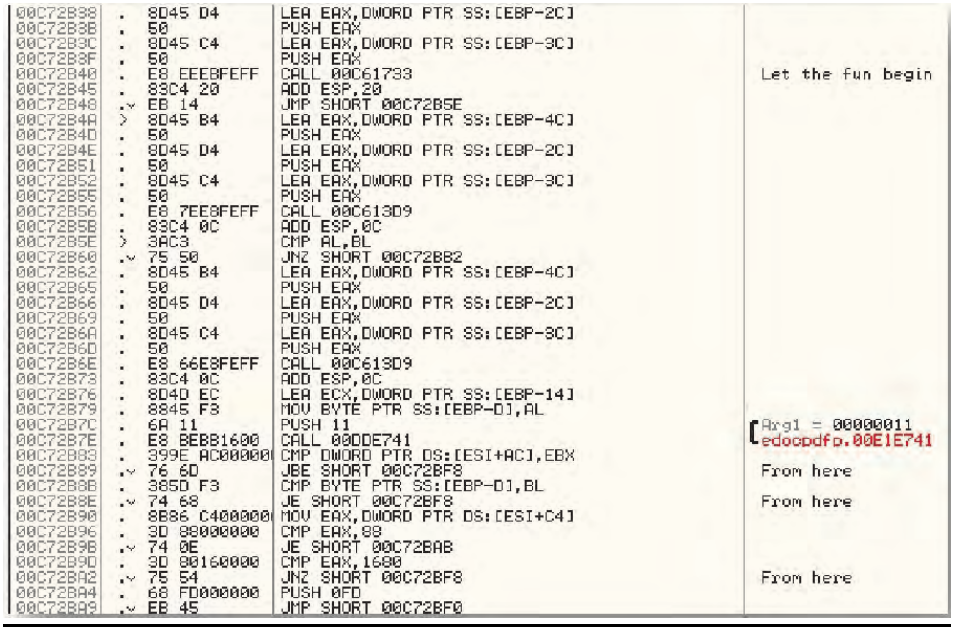
ما آنرا در MessageBOXAEXW به دام انداختیم. دوباره برنامه را جست‌وجو[3] کنید و در قسمت code به انتها برسید.

```
00C72B76  .  8D4D EC      LEA ECX,DWORD PTR SS:[EBP-14]
00C72B79  .  8845 F3      MOV BYTE PTR SS:[EBP-D],AL
00C72B7C  .  6A 11        PUSH 11                          Arg1 = 00000011
00C72B7E  .  E8 BEBB1600  CALL 00DDE741                    edocpdfp.00E1E741
00C72B83  .  399E AC00000 CMP DWORD PTR DS:[ESI+AC],EBX
00C72B89  .v 76 6D        JBE SHORT 00C72BF8               From here
00C72B8B  .  385D F3      CMP BYTE PTR SS:[EBP-D],BL
00C72B8E  .v 74 68        JE SHORT 00C72BF8                From here
00C72B90  .  8B86 C400000 MOV EAX,DWORD PTR DS:[ESI+C4]
00C72B96  .  3D 88000000  CMP EAX,88
00C72B9B  .v 74 0E        JE SHORT 00C72BAB
00C72B9D  .  3D 80160000  CMP EAX,1680
00C72BA2  .v 75 54        JNZ SHORT 00C72BF8               From here
00C72BA4  .  68 FD000000  PUSH 0FD
00C72BA9  .v EB 45        JMP SHORT 00C72BF0
00C72BAB  >  68 FE000000  PUSH 0FE
00C72BB0  .v EB 3E        JMP SHORT 00C72BF0
00C72BB2  >  8B45 B8      MOV EAX,DWORD PTR SS:[EBP-48]
00C72BB5  .  3BC3         CMP EAX,EBX
00C72BB7  .v 75 05        JNZ SHORT 00C72BBE
00C72BB9  .  B8 EC88E000  MOV EAX,00E088EC
00C72BBE  >  50           PUSH EAX
00C72BBF  .  6A 06        PUSH 6                           Arg1 = 00000006
00C72BC1  .  E8 81E7FEFF  CALL 00C61347                    edocpdfp.00CA1347
00C72BC6  .  59           POP ECX
00C72BC7  .  8BF8         MOV EDI,EAX
00C72BC9  .  59           POP ECX
00C72BCA  .  E8 29990500  CALL 00CCC4F8
00C72BCF  .  85C0         TEST EAX,EAX
00C72BD1  .v 75 39        JNZ SHORT 00C72C0C
00C72BD3  .  81FF 3730000 CMP EDI,3037
00C72BD9  .v 74 10        JE SHORT 00C72BEB
00C72BDB  .  81FF 3830000 CMP EDI,3038
00C72BE1  .v 74 08        JE SHORT 00C72BEB
00C72BE3  .  81FF 3E30000 CMP EDI,303E
00C72BE9  .v 75 21        JNZ SHORT 00C72C0C
00C72BEB  >  68 CE000000  PUSH 0CE
00C72BF0  >  8D4D EC      LEA ECX,DWORD PTR SS:[EBP-14]
00C72BF3  .  E8 49BB1600  CALL 00DDE741                    edocpdfp.00E1E741
00C72BF8  >  6A 30        PUSH 30
00C72BFA  .  8BCE         MOV ECX,ESI
00C72BFC  .  FF75 E8      PUSH DWORD PTR SS:[EBP-18]
00C72BFF  .  FF75 EC      PUSH DWORD PTR SS:[EBP-14]
00C72C02  .  E8 339E1600  CALL 00DDCA3A
00C72C07  .v E9 95000000  JMP 00C72CA1
00C72C0C  >  6A 10        PUSH 10                          Arg1 = 00000010
00C72C0E  .  8D4D EC      LEA ECX,DWORD PTR SS:[EBP-14]
Jumps from 00C72B89, 00C72B8E, 00C72BA2
```

در نظر داشته باشید از هر سه موقعیت موجود، منبع اصلی مورد نظر است و صدا زده می‌شود. بنابراین اجازه دهید کمی جست‌وجو کنیم تا به بهترین قسمت برخورد کنیم.

---

[1] messagebox
[2] Break point
[3] trace

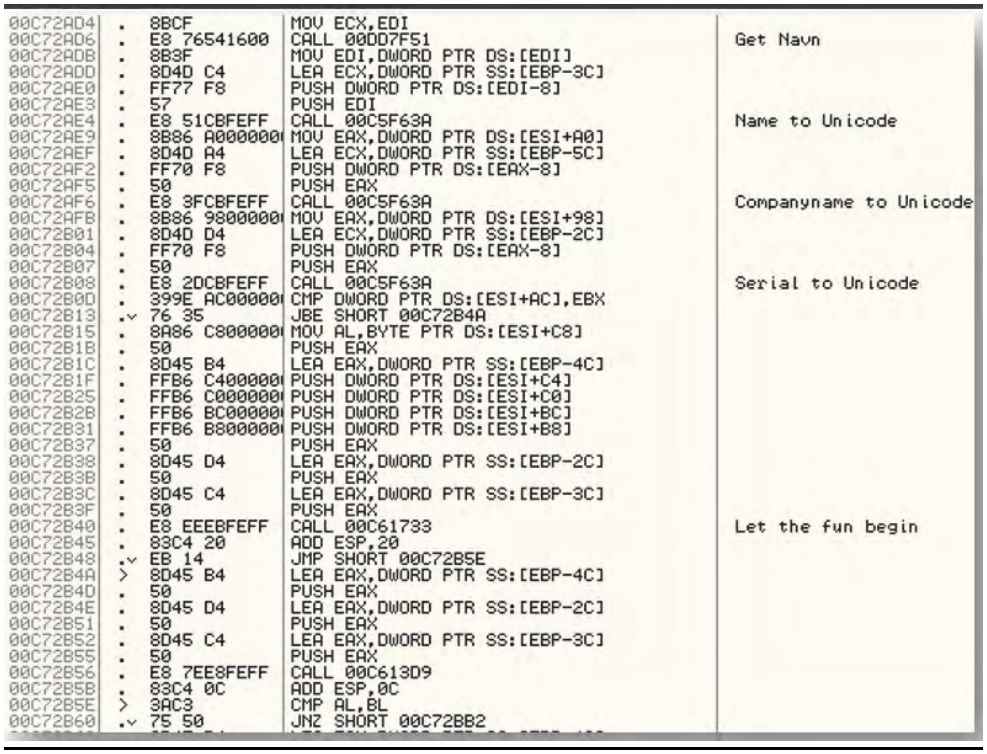


صدا زدن[1] تابع در 0C72B40 در واقع می‌تواند یک صدا زدن قابل توجه باشد. پس به شروع این پروسه می‌رویم و اتفاقات رخ داده را پیدا می‌کنیم. فعلا عمل جست‌وجو در0C72B40 را انجام ندهید.

۱-۴ تجزیه و تحلیل الگوریتم

هنگامی که متوجه عملکردمان شدیم در واقع به اینجا رسیده‌ایم:

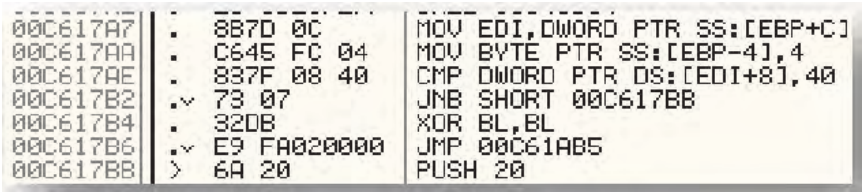
[1] call



برخی از توضیحات، توضیحاتی هستند که هنگام دیباگ کردن نرم‌افزار هدف، خودمان اضافه کرده‌ایم. شما می‌توانید آنها را جست‌وجو کنید بدون اینکه بخواهید وارد جزئیات شوید.

جالبترین قسمت دستورات عمل صدا زدن در 0C72B40 و CMP، در 0C72B5E است. اگر عمل جست‌وجو را روی صدا زدن در 0C72B40 انجام دهید خواهید دید که این صدا زدن کلمه FALSE را برمی‌گرداند. این مقدار برگردانده شده سپس در 0C72B5E چک می‌شود. بنابراین عمل صدازدن در0C72B40 برای روشن (ON) شدن مجبور است که TRUE را برگرداند.

جست‌وجو کردن صدا زدن در 0C72B40 ما را در ابتدای کار به اینجا می‌رساند.



به 0C617AE نگاه کنید. در واقع مقایسه‌ای بین رشته نام (name-string) و40h(64bytes) انجام می‌دهد. پس می‌توانیم نتیجه بگیریم که name-string باید حداقل ۶۴ بایتی باشد. یک بریک پوینت در 0C617AE قرار می‌دهیم و کلید F9 را می‌زنیم.

دوباره یک name- string به طول ۶۴ بایت را وارد می‌کنیم. پیشنهاد ما این است که جمله زیر را تایپ کنید:

“1111111111111111222222222222222233333333333333334444444444444444”

زیرا هنگام جست‌وجو کردن به راحتی قابل تشخیص است.

دوباره در 0C617AE متوقف خواهید شد. اگر به جست‌وجو کردن ادامه دهید در ابتدا به ۲ حلقه برخورد می‌کنید. نخستین حلقه، ۳۲ بایت آخر را کپی می‌کند، به این صورت که بایت به بایت از سریال را در حافظه قرار می‌دهد. دومین حلقه، نخستین ۳۲ بایت مربوط به سریال را به هگز تبدیل می‌کند و در حافظه قرار می‌دهد. هر دو قسمت از سریال، بعدها در محاسبه یک کلید معتبر استفاده می‌شود.

درست پس از اجرای هر دو حلقه، به شکل زیر می‌رسیم.

```
00C6190B  .  8D45 CC      LEA EAX,DWORD PTR SS:[EBP-34]      Name-string
00C6190E  .  50           PUSH EAX
00C6190F  .  8D45 BC      LEA EAX,DWORD PTR SS:[EBP-44]
00C61912  .  50           PUSH EAX
00C61913  .  8D45 8C      LEA EAX,DWORD PTR SS:[EBP-74]
00C61916  .  FF75 08      PUSH DWORD PTR SS:[EBP+8]
00C61919  .  50           PUSH EAX
00C6191A  .  E8 DBF1FFFF  CALL 00C60AFA
```

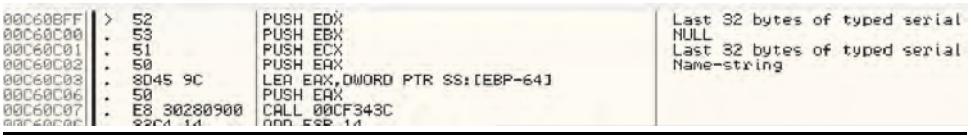
از آنجایی که name-string در پشته قرار گرفته شده، این می‌تواند یک صدا زدن برای پیگیری باشد؛ پس به سمت آن می‌رویم (jump).

به طور خلاصه این صدا زدن همان عمل تولید یک کلید است. قصد نداریم در مورد این صدا زدن توضیحات زیادی را بیان کنیم پس تنها به قسمت‌های مهم اشاره خواهیم کرد.

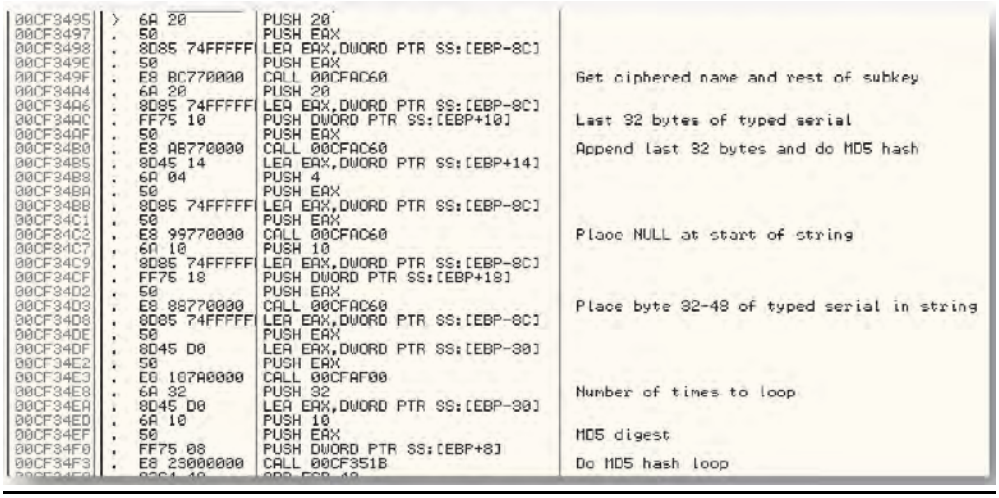
```
00C60BB9  >  8B45 0C      MOV EAX,DWORD PTR SS:[EBP+C]
00C60BBC  .  B9 00010000  MOV ECX,100
00C60BC1  .  0FB700       MOVZX EAX,WORD PTR DS:[EAX]
00C60BC4  .  99           CDQ
00C60BC5  .  F7F9         IDIV ECX
00C60BC7  .  8B45 F0      MOV EAX,DWORD PTR SS:[EBP-10]
00C60BCA  .  FEC2         INC DL
00C60BCC  .  47           INC EDI
00C60BCD  .  8810         MOV BYTE PTR DS:[EAX],DL
00C60BCF  .  3B7D E4      CMP EDI,DWORD PTR SS:[EBP-1C]
00C60BD2  .^ 72 9F        JB SHORT 00C60B73
```

نخستین چیزی که به آن برخورد می‌کنیم یک حلقه است. اینطور به نظر می‌رسد که برنامه نویس در اینجا بایستی یک عمل modulo را روی بایت‌های name-string انجام دهد. در عوض به سادگی مقدار بایت را یکی افزایش می‌دهد.

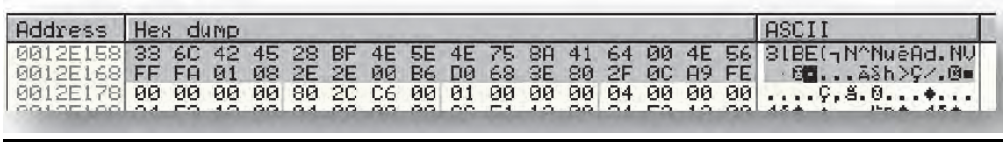
ادامه می‌دهیم و در اینجا متوقف می‌شویم:



بیایید آدرس 0CF343C را جست‌وجو کنیم زیرا به زودی این کار، مهم و ضروری خواهد شد.

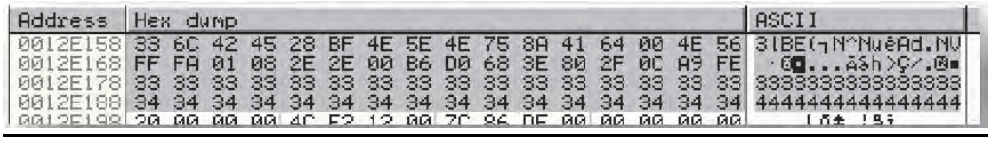


در ادامه تلاش خواهیم کرد که نشان دهیم چه روی داده است.



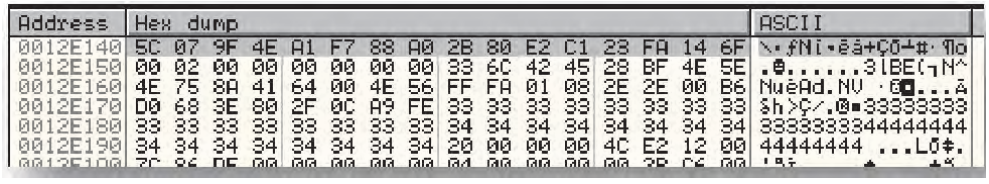
عمل صدا زدن در 0CF34B0، ۳۲ بایت آخر را به name-string اضافه می‌کند و سپس عمل هش[1] MD5 را روی string انجام می‌دهد.

[1] hash

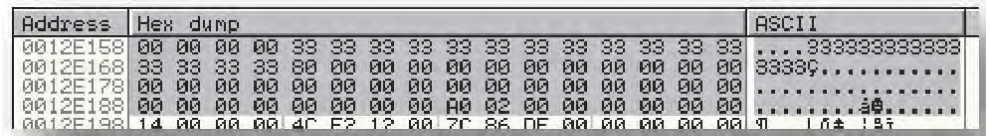


در اینجا هش MD5 اجرا شده است:

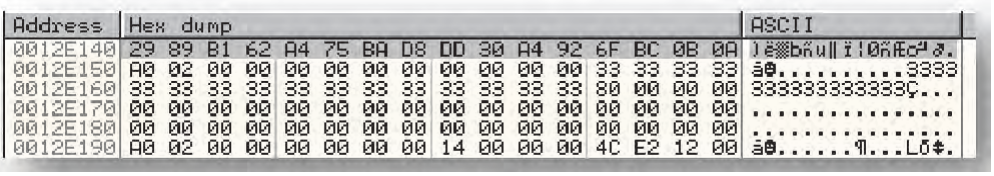
در شکل خلاصه‌ای از آنچه که پس از هش MD5 اتفاق می‌افتد نشان داده شده است:



عمل صدا زدن در 0CF34C2 و صدا زدن در 0CF34D3 رشته‌ای (string) طبق شکل زیر می‌سازد:



دوباره یک هش MD5 اجرا شده است، در نتیجه:



توجه کنید که عملیات صدا زدن پیش از هش کردن این قسمت MD5 را مقداردهی اولیه نمی‌کند!!

عمل صدا زدن[1] در 0CF34F3، یک حلقه MD5 هش را انجام می‌دهد. در این زمان آنها مقداردهی اولیه MD5 را پیش از هش شدن انجام می‌دهند. این حلقه[2] مانند شکل زیر است:

---

[1] call
[2] loop

```
00CF3545  .   50              PUSH EAX
00CF3546  .   8975 FC         MOV DWORD PTR SS:[EBP-4],ESI
00CF3549  .   FF75 10         PUSH DWORD PTR SS:[EBP+10]
00CF354C  .   FF75 0C         PUSH DWORD PTR SS:[EBP+C]
00CF354F  .   E8 EC7A0000     CALL 00CFB040
00CF3554  .   8B45 14         MOV EAX,DWORD PTR SS:[EBP+14]
00CF3557  .   83C4 0C         ADD ESP,0C
00CF355A  .   3BC6            CMP EAX,ESI
00CF355C  .v  7E 18           JLE SHORT 00CF3576
00CF355E  .   8D78 FF         LEA EDI,DWORD PTR DS:[EAX-1]
00CF3561  >   8D45 D0         LEA EAX,DWORD PTR SS:[EBP-30]
00CF3564  .   50              PUSH EAX
00CF3565  .   8D45 D0         LEA EAX,DWORD PTR SS:[EBP-30]
00CF3568  .   6A 10           PUSH 10
00CF356A  .   50              PUSH EAX
00CF356B  .   E8 D07A0000     CALL 00CFB040
00CF3570  .   83C4 0C         ADD ESP,0C
00CF3573  .   4F              DEC EDI
00CF3574  .^  75 EB           JNZ SHORT 00CF3561
00CF3576  >   8D45 D0         LEA EAX,DWORD PTR SS:[EBP-30]
```

صدا زدن در 0CF354F و 0CF356B هر دو، هم عملیات مقداردهی و هم عملیات هش را انجام می‌دهد. نتیجه حلقه در زیر نشان داده شده است:

```
Address   Hex dump                                        ASCII
0012E0BC  A5 09 6C B4 20 27 85 7A 3D 8E 94 B2 E8 D5 A4 F3  ¥.l´ '…z=Ž”²èÕ¤ó
0012E0CC  00 2C C6 00 00 00 00 00 00 00 00 00 00 00 00 00  .,Æ.............
0012E0DC  00 00 00 00 C0 E1 12 00 9B 89 DF 00 01 00 00 00  ....Àá..›‰ß.....
```

این کلید بسیار مهم است و برای معتبر کردن نسخه گواهی (license) استفاده می‌شود که در آینده توضیح خواهیم داد.

هنگامی که ما با یک MD5 تازه برگردیم می‌توانیم عمل جست‌وجو[1] را دوباره ادامه دهیم و در مراحل پیش رو چیزی مشابه شکل زیر می‌بینید:

```
00C60C4A  >   53              PUSH EBX                    NULL
00C60C4B  .   68 80000000     PUSH 80
00C60C50  .   6A 10           PUSH 10
00C60C52  .   56              PUSH ESI                    MD5 Digest
00C60C53  .   6A 10           PUSH 10
00C60C55  .   50              PUSH EAX
00C60C56  .   8D45 9C         LEA EAX,DWORD PTR SS:[EBP-64]   First 32 bytes of typed serial (hex)
00C60C59  .   50              PUSH EAX
00C60C5A  .   E8 DE360900     CALL 00CF433D
00C60C5F  .   83C4 1C         ADD ESP,1C
```

بیایید 0CF433D را جست‌وجو کنیم و نهایتا به اینجا می‌رسیم:

[1] *tracing*

```
00CF4379  .  8D45 F0      LEA EAX,DWORD PTR SS:[EBP-10]
00CF437C  .  FF75 1C      PUSH DWORD PTR SS:[EBP+1C]
00CF437F  .  FF75 18      PUSH DWORD PTR SS:[EBP+18]
00CF4382  .  FF75 14      PUSH DWORD PTR SS:[EBP+14]      MD5 digest (key)
00CF4385  .  50           PUSH EAX
00CF4386  .  E8 E4FEFFFF  CALL 00CF426F                   Create magic table
00CF438B  .  FF75 10      PUSH DWORD PTR SS:[EBP+10]
```

دوباره 0CF426F را جست‌وجو می‌کنیم و دو مورد صدا زدن مهم که باید از آنها مطلع باشیم، وجود دارد. نخستین آن این است:

```
00CF42A2  .  FF75 18      PUSH DWORD PTR SS:[EBP+18]
00CF42A5  .  8D45 D4      LEA EAX,DWORD PTR SS:[EBP-2C]
00CF42A8  .  FF75 14      PUSH DWORD PTR SS:[EBP+14]
00CF42AB  .  FF75 10      PUSH DWORD PTR SS:[EBP+10]
00CF42AE  .  FF75 0C      PUSH DWORD PTR SS:[EBP+C]
00CF42B1  .  50           PUSH EAX
00CF42B2  .  E8 D3EFFFFF  CALL 00CF328A                   Do MD5 hash on key
00CF42B7  .  83C4 14      ADD ESP,14
```

این صدا زدن، هش MD5 را روی کلید اجرا می‌کند.

```
00CF33BD  .  51           PUSH ECX
00CF33BE  .  56           PUSH ESI
00CF33BF  .  50           PUSH EAX                        MD5 digest (key)
00CF33C0  .  E8 7B7C0000  CALL 00CFB040                   Do MD5 hash on key
00CF33C5  .  83C4 0C      ADD ESP,0C
```

عملیات صدا زدنی که MD5 جدید را ساخته است همانند شکل می‌باشد:

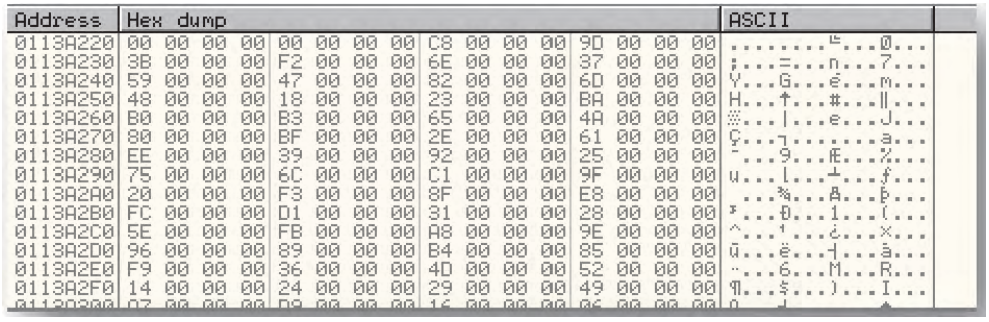
```
Address   Hex dump                                        ASCII
01495E21  C8 D4 9C B4 7D FC 10 9F 4A E2 CE C8 FF C3 AF CE  ╚╘£┤}ⁿ►ƒJΓ╬╚ ├»╬
01495E31  00 AD BA 0D F0 AD BA 0D F0 AD BA 0D F0 AD BA 0D  .¡║.≡¡║.≡¡║.≡¡║.
01495E41  F0 0D BA 0D F0 0D BA 0D F0 0D BA 0D F0 0D BA 0B  ≡♪║♪≡♪║♪≡♪║♪≡♪║◙
```

کلیدی که قبلا ساخته شده است تغییر نمی‌کند این یک MD5 جدید است که برای ساخت آخرین ۳۲ بایت سریال استفاده می‌شود.

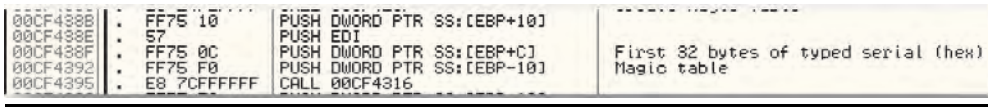
دومین صدا زدن مهم در زیر آمده است. اگر عملیات جست‌وجو را نسبت به این صدا زدن انجام دهیم به این نقطه می‌رسیم.

```
00CF42E9  >  50           PUSH EAX                        MD5 digest
00CF42EA  .  FF75 EC      PUSH DWORD PTR SS:[EBP-14]      Size of digest
00CF42ED  .  56           PUSH ESI                        Output buffer (magic table)
00CF42EE  .  E8 4D730000  CALL 00CFB640                   Create magic table
00CF42F3  .  8B4D 08      MOV ECX,DWORD PTR SS:[EBP+8]
```

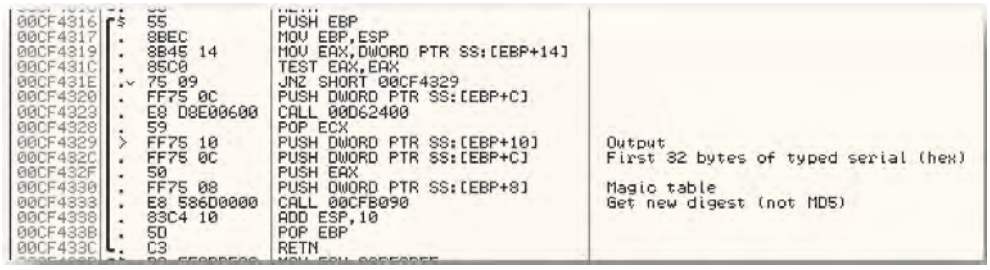
این صدا زدن درآدرس افست 0CF42EE یک «جدول جادویی»[1] می‌سازد. جدول ابتدا ساخته می‌شود سپس با استفاده از MD5 digest به هم ریخته و مخلوط می‌گردد. در این مثال شروع جدول همانند این شکل است:



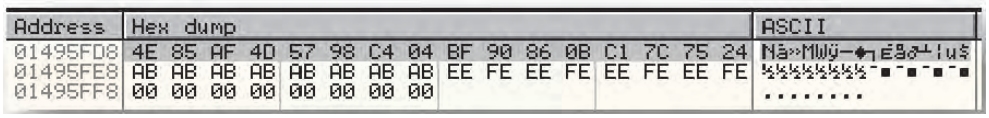
به 0CF438B بر گردید، دیس اسمبل همانند زیر است:



صدا زدن در 0CF3416، کاری را انجام می‌دهد که آنرا به عنوان عملیات xor توضیح داده‌ایم. خودتان روی آن بررسی کوچکی انجام دهید. اگرچه کمی گیج کننده به نظر می‌رسد اما در حقیقت ساده است. شکل زیر پروسیجر 0CF4316 را نشان می‌دهد:



خروجی همانند این شکل است:



[1] magic table

باید این احتمال را داده باشید که بدترین قسمت تمام شده است، اما نه، هنوز کار را به اتمام نرسانده‌ایم. به نظر می‌رسد که تولیدکنندگان، این الگوریتم را به منظور پراکنده کردن ذهن ما ساخته‌اند. اما موفق نخواهند شد!

قبلا ما صدا زدن در 0C6191A را دنبال کردیم. اکنون به 0C6191F برگشتیم. کمی به سمت پایین جست‌وجو می‌کنیم تا به این شکل برخورد کنیم:

```
00C61946  .  8D45 AC       LEA EAX,DWORD PTR SS:[EBP-54]
00C61949  .  50            PUSH EAX
00C6194A  .  8D45 9C       LEA EAX,DWORD PTR SS:[EBP-64]
00C6194D  .  50            PUSH EAX
00C6194E  .  8D45 8C       LEA EAX,DWORD PTR SS:[EBP-74]
00C61951  .  FF75 08       PUSH DWORD PTR SS:[EBP+8]
00C61954  .  50            PUSH EAX
00C61955  .  E8 1AEEFFFF   CALL 00C60774
```

عملیات جست‌وجو را در 0C60774 انجام دهید و در ابتدای این پروسه یکبار دیگر عمل هش modudoرا روی نام انجام دهید و آنرا در حافظه ذخیره کنید. بعدا به آن خواهیم پرداخت. در ادامه به اینجا می‌رسیم:

```
00C608C3  >  8D45 B0       LEA EAX,DWORD PTR SS:[EBP-50]
00C608C6  .  50            PUSH EAX
00C608C7  .  8D45 D0       LEA EAX,DWORD PTR SS:[EBP-30]
00C608CA  .  50            PUSH EAX
00C608CB  .  8D45 8C       LEA EAX,DWORD PTR SS:[EBP-74]
00C608CE  .  50            PUSH EAX
00C608CF  .  E8 5C330900   CALL 00CF3C30
```

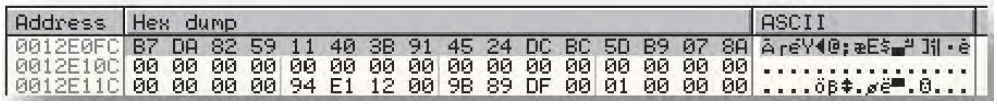
این روند اجرایی در 0CF3C30 همچنان ممکن است برای ما آزار دهنده باشد. اما اجازه دهید که آنرا جست‌وجو کنیم. کمی که جلوتر برویم به شکل زیر می‌رسیم:

```
00CF3CD9  >  6A 33         PUSH 33                          Number of loops
00CF3CDB  .  6A 20         PUSH 20                          Size of string to hash
00CF3CDD  .  50            PUSH EAX
00CF3CDE  .  8D45 B0       LEA EAX,DWORD PTR SS:[EBP-50]
00CF3CE1  .  50            PUSH EAX
00CF3CE2  .  E8 34F8FFFF   CALL 00CF351B                    Do yet another MD5 hash loop
```

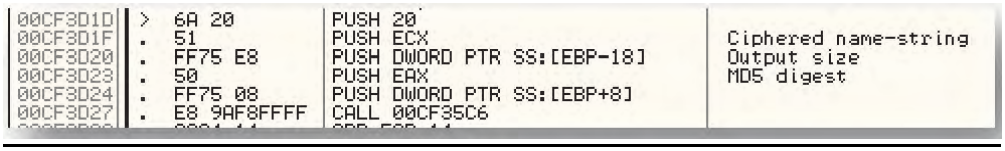
به خاطر داشته باشید که digest جدید شبیه زیر است:

```
Address   Hex dump                                          ASCII
0012E09C  4E 85 AF 4D 57 98 C4 04 BF 90 86 0B C1 7C 75 24   N…»MW˜─.┐É…¡|u$
0012E0AC  28 BF 4E 5E 4E 75 8A 41 64 00 4E 56 FF FA 01 08   (┐N^NuèAd.NV ·☺
0012E0BC  80 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   Ç...............
```

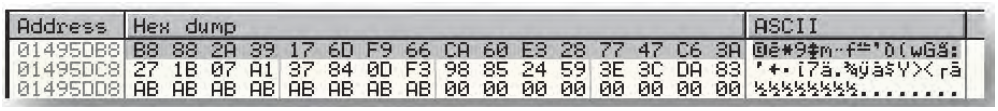
و این باید در حلقه‌های MD5، ۵۱ بار دیگر اجرا شود. نتیجه این است:



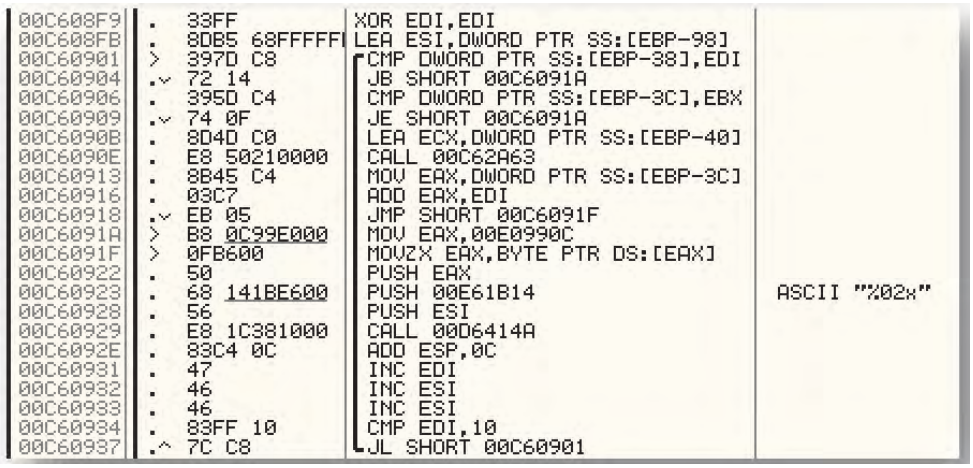
به 0C608D4 برمی‌گردیم و می‌توانیم دوباره کمی جست‌وجو کنیم و به اینجا برسیم:



اگر به جست‌وجو کردن این صدا زدن ادامه دهید خواهید دید که از MD5 digest برای ساختن یک جدول جادویی دیگر استفاده شده است و سپس دوباره عمل xor انجام می‌گیرد. این قضیه ۲۰ مرتبه انجام شده و نتیجه زیر ظاهر شده است:



سرانجام این برنامه پایان پذیرفت. اکنون ما به 0C608D4 برمی‌گردیم. کمی که جلوتر برویم این شکل را خواهیم دید:



۲ عدد حلقه وجود دارد. نخستین آن (حلقه بالایی)، آخرین کلید ما را به یک text-string تبدیل می‌کند و نتیجه زیر پدیدار می‌گردد:

```
Address   Hex dump                                          ASCII
0012E1C0  62 38 38 38 32 61 33 39 31 37 36 64 66 39 36 36  b8882a39176df966
0012E1D0  63 61 36 30 65 33 32 38 37 37 34 37 63 36 33 61  ca60e3287747c63a
0012E1E0  00 0C C6 00 00 00 00 00 00 00 00 00 00 00 00 00  ..Æ.............
```

حلقه دوم زیاد مهم نیست. حداقل ما نیاز چندانی به آن نداریم!

اکنون تا زمانی که به 0C6195A برگردیم عمل جست‌وجو را انجام می‌دهیم. در بالا یک حلقه مقایسه‌ای می‌بینم، این مقایسه‌ای است بایت به بایت از کلید با ۳۲ بایت آخر که در سریال ما تایپ شده است.

```
00C61967  >  3975 D4        CMP DWORD PTR SS:[EBP-2C],ESI
00C6196A  .v 72 16          JB SHORT 00C61982
00C6196C  .  837D D0 00     CMP DWORD PTR SS:[EBP-30],0
00C61970  .v 74 10          JE SHORT 00C61982
00C61972  .  8D4D CC        LEA ECX,DWORD PTR SS:[EBP-34]
00C61975  .  E8 E9100000    CALL 00C62A63
00C6197A  .  8B45 D0        MOV EAX,DWORD PTR SS:[EBP-30]
00C6197D  .  8D3C30         LEA EDI,DWORD PTR DS:[EAX+ESI]
00C61980  .v EB 05          JMP SHORT 00C61987
00C61982  >  BF 0C99E000    MOV EDI,00E0990C
00C61987  >  3975 B4        CMP DWORD PTR SS:[EBP-4C],ESI
00C6198A  .v 72 1A          JB SHORT 00C619A6
00C6198C  .  837D B0 00     CMP DWORD PTR SS:[EBP-50],0
00C61990  .v 74 14          JE SHORT 00C619A6
00C61992  .  8D4D AC        LEA ECX,DWORD PTR SS:[EBP-54]
00C61995  .  E8 EC29FEFF    CALL 00C44386
00C6199A  .  8B45 B0        MOV EAX,DWORD PTR SS:[EBP-50]
00C6199D  .  BB EC88E000    MOV EBX,00E088EC
00C619A2  .  03C6           ADD EAX,ESI
00C619A4  .v EB 07          JMP SHORT 00C619AD
00C619A6  >  BB EC88E000    MOV EBX,00E088EC
00C619AB  .  8BC3           MOV EAX,EBX
00C619AD  >  8A0F           MOV CL,BYTE PTR DS:[EDI]      byte-to-byte check
00C619AF  .  3A08           CMP CL,BYTE PTR DS:[EAX]
00C619B1  .v 75 0D          JNZ SHORT 00C619C0
00C619B3  .  46             INC ESI
00C619B4  .  83FE 20        CMP ESI,20
00C619B7  .^ 7C AE          JL SHORT 00C61967
```

```
Registers (FPU)
EAX 0113A221 ASCII "b8882a39176df966ca60e3287747c63a"
ECX 0012E29C
EDX 01130608
EBX 00E088EC edocpdfp.00E088EC
ESP 0012E270
EBP 0012E2F0
ESI 00000000
EDI 01495C79 ASCII "33333333333333334444444444444444"
EIP 00C619AD edocpdfp.00C619AD
```

هم اکنون وقت آن است که قسمت آخر سریال را که باید در جدول جادویی (نامی است که ما بر آن گذاشته‌ایم) محاسبه شود پاک کنید. بهتر است آنچه را که می‌دانیم جمع کرده و به طور خلاصه بیان کنیم:

آخرین ۳۲ بایت از سریال با استفاده از نخستین کلید، نام، MD5 hashing و جدول جادویی محاسبه می‌شود. این عملیات محاسباتی چک می‌شود و اگر نتیجه این محاسبات با آخرین ۳۲ بایت سریال مساوی نباشد خطا دریافت می کنیم.

نخستین کلید در واقع از آخرین ۳۲ بایت سریال با استفاده ازقسمت نام ما (name) که در تکس باکس است وMD5 hashing محاسبه شده است.

در این وضعیت چگونه می‌توانیم هر کدام از اینها را محاسبه کنیم در حالی که هر دوی آنها به یکدیگر وابسته‌اند؟

این سوال مطرح می‌شود که این کار امکان دارد یا نه؟ از آنجایی که کلید ۱۲۸ بیتی است عمل بروت فورس[1] خارج محدوده ماست و نیز ساختن یک collision MD5 به نظر دور از تصور است. پس باید چه کار کرد؟

پس از کمی فکر کردن جواب مشخص می‌شود!! پاسخ به این سوال، جدول جادویی و نخستین کلید است. در قسمت‌های بعد چگونگی عملکرد آن را نشان می‌دهیم.

۵-۱ پیدا کردن یک کلید معتبر

می‌دانیم که هدف ما مقایسه‌ای است بایت به بایت روی آخرین ۳۲ بایت از سریال. اگر موفقیت حاصل نشد، برنامه هدف یک AND را اجرا می‌کند و یک موقعیت پیغام خطا را می‌سازد.

```
00C619AB  .  8BC3          MOV EAX,EBX
00C619AD  >  8A0F          MOV CL,BYTE PTR DS:[EDI]      byte-to-byte check
00C619AF  .  3A08          CMP CL,BYTE PTR DS:[EAX]
00C619B1  .v 75 0D         JNZ SHORT 00C619C0
00C619B3  .  46            INC ESI
00C619B4  .  83FE 20       CMP ESI,20
00C619B7  .^ 7C AE         JL SHORT 00C61967
00C619B9  .  BB EC88E000   MOV EBX,00E088EC
00C619BE  .v EB 04         JMP SHORT 00C619C4
00C619C0  >  8065 F3 00    AND BYTE PTR SS:[EBP-D],0     Create BAD-BOY
```

برای اینکه یک کلید معتبر را بدست بیاوریم نیاز است که بررسی‌هایی را انجام دهیم. از نقطه نظر ما این کار در IDA با استفاده از شکل گرافیکی آسان‌تر انجام می‌شود. بنابراین پس از استفاده از قسمت OLLY قسمت IDA خواهد بود.

[1] *Brute-forcing*

```
00C619E2  .  8BC3          MOV EAX,EBX
00C619E4  >  50            PUSH EAX
00C619E5  .  57            PUSH EDI
00C619E6  .  E8 5CF9FFFF   CALL 00C61347
00C619EB  .  8945 08       MOV DWORD PTR SS:[EBP+8],EAX
00C619EE  .  59            POP ECX
00C619EF  .  66:C1E8 0D    SHR AX,0D
00C619F3  .  66:3D 0100    CMP AX,1
00C619F7  .  59            POP ECX
00C619F8  .v 75 03         JNZ SHORT 00C619FD
00C619FA  .  8845 F2       MOV BYTE PTR SS:[EBP-E],AL
```

در شکل بالا key[0] را می‌گیرد (سایز WORD) و بیت به اندازه ۱۳ بیت به سمت راست شیفت پیدا می‌کند. اگر سیزدهمین بیت تنظیم شود آنگاه یک نسخه تستی[1] را اجرا می‌کنیم و اگر تنظیم نشود، یک نرم‌افزاری واقعی را اجرا می‌کنیم. فعلا قصد نداریم درباره آن با جزئیات توضیح دهیم. خودتان آزادانه آنرا تست کنید. اگر سیزدهمین بیت تنظیم شود، هدف ما تلاش برای این است که متوجه شویم چند روز از این نسخه آزمایشی باقی مانده است و این کار را با استفاده از key[6] (word) انجام می‌دهیم.

```
00C61A0B  .  8BC3          MOV EAX,EBX
00C61A0D  >  50            PUSH EAX
00C61A0E  .  6A 04         PUSH 4
00C61A10  .  E8 32F9FFFF   CALL 00C61347
00C61A15  .  807D F2 00    CMP BYTE PTR SS:[EBP-E],0        Are we a test-version ?
00C61A19  .  59            POP ECX
00C61A1A  .  59            POP ECX
00C61A1B  .v 74 11         JE SHORT 00C61A2E
00C61A1D  .  57            PUSH EDI
00C61A1E  .  50            PUSH EAX
00C61A1F  .  E8 28CD0600   CALL 00CCE74C
00C61A24  .  59            POP ECX
00C61A25  .  84C0          TEST AL,AL
00C61A27  .  59            POP ECX
00C61A28  .v 74 04         JE SHORT 00C61A2E
00C61A2A  .  8065 F3 00    AND BYTE PTR SS:[EBP-D],0
```

همانطور که ممکن است متوجه شوید EBP-0E اگر FALSE باشد چک می‌شود. اگر چنین باشد، ادامه می‌دهیم؛ وگرنه تست می‌کنیم که چند روز از نسخه آزمایشی باقی مانده است. اگر چیزی باقی نمانده بود، پیغام خطا را در 0C61A2A می‌سازیم. مرحله بعد key[4] (word) را بررسی می‌کنیم:

[1] test-version

```
00C61A35  .  8BC3          MOV EAX,EBX
00C61A37  >  50            PUSH EAX
00C61A38  .  6A 02         PUSH 2
00C61A3A  .  E8 08F9FFFF   CALL 00C61347
00C61A3F  .  59            POP ECX
00C61A40  .  59            POP ECX
00C61A41  .  33C9          XOR ECX,ECX
00C61A43  .  8ACC          MOV CL,AH
00C61A45  .  25 FF000000   AND EAX,0FF
00C61A4A  .  394D 1C       CMP DWORD PTR SS:[EBP+1C],ECX   CMP ecx == 5
00C61A4D  .v 74 0D         JE SHORT 00C61A5C
00C61A4F  .  3BC7          CMP EAX,EDI
00C61A51  .v 74 05         JE SHORT 00C61A58
00C61A53  .  3945 1C       CMP DWORD PTR SS:[EBP+1C],EAX   CMP eax >= 5
00C61A56  .v 76 04         JBE SHORT 00C61A5C
00C61A58  >  8065 F3 00    AND BYTE PTR SS:[EBP-D],0
```

فرض کنید که کلید این است: xxxxYYZZxxxxxxxxxxxxxxxxxxxxxxxx

می‌توان دید که هم ZZ باید ۰۵ باشد و هم YY باید بالاتر از ۰۵ باشد.

در مرحله بعد با چیزی شبیه به شکل زیر بر خورد می‌کنیم:

```
00C61A63  .  8BC3          MOV EAX,EBX
00C61A65  >  50            PUSH EAX
00C61A66  .  6A 0C         PUSH 0C
00C61A68  .  E8 DAF8FFFF   CALL 00C61347
00C61A6D  .  3945 18       CMP DWORD PTR SS:[EBP+18],EAX   CMP eax == 5900
00C61A70  .  59            POP ECX
00C61A71  .  59            POP ECX
00C61A72  .v 74 04         JE SHORT 00C61A78
00C61A74  .  8065 F3 00    AND BYTE PTR SS:[EBP-D],0
```

(word) key[12] بایستی با 0x5900 مساوی باشد. پس اکنون می‌دانیم که یک کلید تا الان می‌تواند شبیه این باشد:

xxxx1205 xxxxxxxxxxxxxxxx0059xxxx

به عنوان آخرین مورد ولی نسبتا با اهمیت می‌توانیم این شکل را ببینیم:

```
00C61A84  .  8BC3          MOV EAX,EBX
00C61A86  >  50            PUSH EAX
00C61A87  .  6A 0F         PUSH 0F
00C61A89  .  E8 B9F8FFFF   CALL 00C61347
00C61A8E  .  59            POP ECX
00C61A8F  .  59            POP ECX
00C61A90  .  0FB74D 08     MOVZX ECX,WORD PTR SS:[EBP+8]   Get key[0] (word) && 0x1FFF
00C61A94  .  394D 14       CMP DWORD PTR SS:[EBP+14],ECX
00C61A97  .v 74 05         JE SHORT 00C61A9E
00C61A99  .  394D 20       CMP DWORD PTR SS:[EBP+20],ECX
00C61A9C  .v EB 0E         JMP SHORT 00C61AAC
00C61A9E  >  8445 24       TEST BYTE PTR SS:[EBP+24],AL    Test AL = 0x80
00C61AA1  .v 75 0F         JNZ SHORT 00C61AB2
00C61AA3  .v EB 09         JMP SHORT 00C61AAE
00C61AA5  >  0FB745 08     MOVZX EAX,WORD PTR SS:[EBP+8]
00C61AA9  .  3945 14       CMP DWORD PTR SS:[EBP+14],EAX
00C61AAC  >v 74 04         JE SHORT 00C61AB2
00C61AAE  >  8065 F3 00    AND BYTE PTR SS:[EBP-D],0
00C61AB2  >  8A5D F3       MOV BL,BYTE PTR SS:[EBP-D]      Move GOOD BOY / BAD BOY to BL
```

در ابتدا key[15] (byte) را دریافت می‌کنیم. سپس چک می‌کنیم که آیا key[0] (word) هم با 0x1680 و هم با 0x1688 مساوی است یا خیر؟ اگر key[0] با هیچ یک از این مقادیر مساوی نباشد، پیغام خطا

ساخته می‌شود. شبیه همین برای key[15] نیز انجام می‌شود. اگر نه، هشتمین بیت تنظیم می‌شود. قبلا در کد، key[0] با 0x1FFF، AND شده است. هم اکنون یک کلید داریم که به شکل زیر است.

88161205xxxxxxxxxxxxxxxx0059xx80

متاسفانه هنوز کار تمام نشده است و هنوز نیاز داریم که تعداد کمی از جاهای خالی را پر کنیم. برای پیدا کردن قطعات نداشته، یک ایده خوب، تنظیم یک بریک پوینت روی عمل مقایسه بایت به بایت خواهد بود. پنجره ثبت[1] را ببندید و کلید علامت سوال را بزنید. اکنون دوباره در مقایسه بایت به بایت خواهید بود. اینک از طریق معتبرسازی کلید پیش رفته و در آخر به اینجا می‌رسید:

```
00C61E56  .  83C4 24        ADD ESP,24
00C61E59  .  3AC3           CMP AL,BL
00C61E5B  .  8845 F3        MOV BYTE PTR SS:[EBP-D],AL
00C61E5E  .v 74 09          JE SHORT 00C61E69
00C61E60  .  395D E8        CMP DWORD PTR SS:[EBP-18],EBX
00C61E63  .v 0F85 3A010000  JNZ 00C61FA3
00C61E69  >  53             PUSH EBX
00C61E6A  .  FF75 18        PUSH DWORD PTR SS:[EBP+18]
00C61E6D  .  FF75 14        PUSH DWORD PTR SS:[EBP+14]
00C61E70  .  FF75 0C        PUSH DWORD PTR SS:[EBP+C]
00C61E73  .  FF75 08        PUSH DWORD PTR SS:[EBP+8]
00C61E76  .  E8 A5740600    CALL 00CC9320
00C61E7B  .  8A45 0F        MOV AL,BYTE PTR SS:[EBP+F]
00C61E7E  .  83C4 14        ADD ESP,14
00C61E81  .  8D4D C0        LEA ECX,DWORD PTR SS:[EBP-40]
00C61E84  .  8845 C0        MOV BYTE PTR SS:[EBP-40],AL
00C61E87  .  53             PUSH EBX
00C61E88  .  E8 CDC7FEFF    CALL 00C4E65A
00C61E8D  .  68 041BE600    PUSH 00E61B04                    UNICODE "UserID"
00C61E92  .  E8 15181000    CALL 00D636AC
00C61E97  .  59             POP ECX
00C61E98  .  50             PUSH EAX
00C61E99  .  68 041BE600    PUSH 00E61B04                    UNICODE "UserID"
00C61E9E  .  8D4D C0        LEA ECX,DWORD PTR SS:[EBP-40]
00C61EA1  .  E8 A9CFFEFF    CALL 00C4EE4F
```

Shife+F9 را بزنید و شکل زیر را ببینید:

```
00C92499  .  83C4 28        ADD ESP,28
00C9249C  .  84C0           TEST AL,AL
00C9249E  .v 75 3E          JNZ SHORT 00C924DE
00C924A0  .  8A86 6401000(  MOV AL,BYTE PTR DS:[ESI+164]
00C924A6  .  50             PUSH EAX
00C924A7  .  8D45 CC        LEA EAX,DWORD PTR SS:[EBP-34]
00C924AA  .  FFB6 6001000(  PUSH DWORD PTR DS:[ESI+160]
00C924B0  .  FFB6 5C01000(  PUSH DWORD PTR DS:[ESI+15C]
00C924B6  .  FFB6 5801000(  PUSH DWORD PTR DS:[ESI+158]
00C924BC  .  FFB6 5401000(  PUSH DWORD PTR DS:[ESI+154]
00C924C2  .  53             PUSH EBX
00C924C3  .  68 1810E600    PUSH 00E61018                    UNICODE "Software\ITEKSOFT\eDocPrinter\PDF\5.0
00C924C8  .  50             PUSH EAX
00C924C9  .  8D45 BC        LEA EAX,DWORD PTR SS:[EBP-44]
00C924CC  .  50             PUSH EAX
00C924CD  .  57             PUSH EDI
00C924CE  .  E8 DAF7FCFF    CALL 00C61CAD
00C924D3  .  83C4 28        ADD ESP,28
00C924D6  .  84C0           TEST AL,AL
00C924D8  .v 0F84 7406000(  JE 00C92B52
00C924DE  >  8A45 F2        MOV AL,BYTE PTR SS:[EBP-E]
00C924E1  .  6A 00          PUSH 0
00C924E3  .  8D4D 9C        LEA ECX,DWORD PTR SS:[EBP-64]
00C924E6  .  8845 9C        MOV BYTE PTR SS:[EBP-64],AL
00C924E9  .  E8 6CC1FBFF    CALL 00C4E65A
00C924EE  .  BF 041BE600    MOV EDI,00E61B04                 UNICODE "UserID"
00C924F3  .  57             PUSH EDI
00C924F4  .  E8 B3110D00    CALL 00D636AC
00C924F9  .  59             POP ECX
```

[1] *registration*