

Neden Tersine Mühendislik ?

written by Mert SARICA | 6 July 2010

Eskiden tersine mühendislik nedense bana çok uzak gelirdi. IDA Pro uygulamasını ilk kurup çalıştırdığımda her yerde bir buton olduğunu görünce “hiç işim olmaz” diyerek kapattığımı hatırlıyorum ama aradan geçen zaman sonrasında şimdi masaüstüne bakıyorum da, vazgeçilmez kısayollardan bir tanesi oluvermiş ve “hep işim olmuş” :)

Neden bu kadar insan assembly gibi anlaşılması zor bir dil ile uğraşıyor, neden saatlerini tek bir fonksiyonun ne iş yaptığını anlamak için harcıyor kısaca neden tersine mühendislik ? Hemen aklıma gelenleri sıralayıp kısaca açıklayayım;

- Assembly öğrenmek için – Bilmeden nasıl öğrenilir demeyin. Debugger ile programları izleye izleye emin olun birşeyler öğreniyor ve komutların ne iş yaptığını az çok anlayabiliyorsunuz ve üzerine güzel bir kitap okuduğunuz zaman taşlar yerine oturuyor ve bir bakmışsınızki matrixi yorumlar olmuşsunuz.
- Gizli şifreleri, arka kapıları tespit etmek için – Zaman zaman programlardaki gizli menülere erişmek için programı hazırlayanlar dokümantasyonda yer almayan gizli fonksiyonlara programlarında yer veriyorlar ve kaynak kodu açık olmadığı için bunun sadece kendileri tarafından bilineceği yanılgısına düşüyorlar ve eninde sonunda tüm dünya bunu tersine mühendislik ile uğraşan bu meraklı insanlar sayesinde duyuyor.
- Doğrulama mekanizmalarını aşmak için – Daha önce bu örneği vermişmiydim hatırlamıyorum fakat bir zaman şifreleme anahtarı saklamak amacıyla kullanılan bir yazılımı incelediğimde anahtarı görüntüleyene kadar iki defa şifre doğrulama adımından geçmeniz gerekiyordu. Fakat bu adımları assembly seviyesinde yamadığımda (son adıma gitmesini sağladım) doğrulama mekanizmalarının tersine mühendislik ile aşılabildiğini işte o zaman öğrenmiştim.
- Kapalı kaynak kodlu yazılımlardaki güvenlik açıklarını bulmak için – Bildiğiniz veya bilmediğiniz üzere Microsoft firmasının o aylık meşhur yama günü gelip çattığında dünyanın dört bir yanındaki meraklı insanlar yamaların farklarını çıkartarak tersine mühendislik ile güvenlik

bültenlerine konu olmamış güvenlik zafiyetlerini ortaya çıkartarak istismar kodu geliştirilmesini sağlamaktalar.

- Zararlı yazılımları analiz etmek için – Bu konuda çok fazla söze gerek yok sanırım, yazılarımı takip edenler bilirler :)
- Can sıkıntısına karşı bire bir :) – Canım ne zaman sıkılsa ve bir programı oturup incelemeye başlasam saatlerin nasıl geçtiğin bir türlü anlamam, 6 saatin 6 dakika gibi geçtiğine çok defa tanık olduğumu söyleyebilirim.

Tersine mühendislik denilince çoğu kişinin aklına nedense hep kod seviyesinde olanı gelir fakat aslında tersine mühendislik sistem seviyesi ve kod seviyesi olmak üzere ikiye ayrılır ve günlük işlerinizde oldukça zaman zaman kullandığınız o meşhur sysinternals araçları ile sistem seviyesinde tersine mühendislik yaptığınızın farkında bile olmazsınız.

Sistem seviyesinde tersine mühendislik yaparken çoğu bilgiye işletim sistemi üzerinden erişirsiniz çünkü incelemiş olduğunuz hedef program eninde sonunda işletim sistemi ile etkileşime girer. Sistem seviyesinde tersine mühendislik için kullanılan araçlar regmon, filemon, lsof, ptrace ve benzerleridir.

Kod seviyesinde tersine mühendisliğe ise sistem seviyesindeki yeterli olmadığı zaman başvurursunuz çünkü sistem seviyesinde bir programa gireni, çıkanı ve işletim sistemindeki etkileşimini görebilirsiniz fakat aslında o gizem dolu olup ve biten yazılımın içinde gerçekleşmektedir ve olup bitenden emin olmak için tek seçeneğiniz kod seviyesidir.

Programları incelemeyi seven biri olarak geçtiğimiz ay Wirofon uygulamasını sistem seviyesinde incelemiştim. Filemon ve regmon ile kısa bir takipten sonra kurulduğu klasörde yer alan userSettings\loginSettings.dat dosyası ile etkileşim halinde olduğunu görmüştüm. Canınım sıkıldığı bir gün bu dosya ile ne işi olduğunu merak ettim ve incelemeye başladım. Aradan kısa bir zaman geçtikten sonra programda yer alan “Beni hatırla” ve “Şifremi hatırla” seçeneği işaretlendiği zaman bu dosyanın içeriğinin değiştiğini fark ettim. loginSettings.dat dosyasını metin düzenleme programı (wordpad) ile incelediğimde okunaklı olmadığını yani şifrelenmiş olduğunu gördüm. Immunity Debugger ile dosya ile ilişkili kısımları incelediğimde “Not valid TEA encoded data” mesajı hemen dikkatimi çekti.

```
00462C76 . 805424 4C LEA EDX, DWORD PTR SS:[ESP+4C]
00462C7A . 68 BCAB5F00 PUSH wirofon.005FABBC
00462C7F . 52 PUSH EDX
00462C80 . C68424 DC0000 MOV BYTE PTR SS:[ESP+DC], 2
00462C88 . E8 E3EBFAFF CALL wirofon.00411870

[Arg2 = 005FABBC ASCII "Not valid TEA encoded data"
Arg1
wirofon.00411870]
```

Bu programı inceleyene kadar TEA şifreleme algoritması ile ilgili hiçbir bilğim yoktu. (İşte tersine mühendisliğin güzel yanlarından biride bu, mutlaka yeni birşeyler öğreniyorsunuz.) Google’da ufak bir araştırma yaptıktan sonra şifreleme algoritması olduğunu ve zayıflıkları olması nedeniyle XTEA olarak güncellendiğini ve en sonunda XXTEA olarak güncellendiğini gördüm. TEA şifrelemesi ile ilgili internette bir çok makale ve hazır kod parçacıkları olduğu için loginSettings.dat dosyasında yer alan şifreli veriyi kolaylıkla çözebileceğimi düşünüyordum fakat çok geçmeden yanıldığımı fark ettim. TEA ve XTEA algoritmalarını incelediğimde şifrelemenin bir bölümünde SHL 4 (shift left, çarpma işlemi için kullanılır) ve SHR 5 (shift right, bölme işlemi için kullanılır) dikkatimi çekti fakat şifreli verinin neden çözülmediği ile ilgili uygulama üzerinde araştırma yaptığımda ufak bir detay gözüme ilişti.

Öncelikle şifrelemeyi çözmeden sorumlu olan fonksiyona giden ve anahtar olarak kullanılan dizinin “ARGELA Technologies” olduğunu ve bu dizinin (string) 4 parçaya bölünerek anahtar olarak kullanıldığını tespit ettim.

```
00462D5A . 8B0D 54036500 MOV ECX,DWORD PTR DS:[650354]      wirofon.005FAB28
00462D60 . 8B40 0C      MOV EAX,DWORD PTR DS:[EAX+C]
00462D63 . 51          PUSH ECX
00462D64 . 8D14F0      LEA EDX,DWORD PTR DS:[EAX+ESI*8]
00462D67 . 52          PUSH EDX
00462D68 . 8BCB      MOV ECX,EBX
00462D6A . E8 51FCFFF CALL wirofon.004629C0      wirofon.004629C0
                                [Arg2 => 005FAB28 ASCII "ARGELA Technologies"
                                Arg1
```

Daha sonra ise şifreli veriyi çözen fonksiyonu incelediğimde TEA ve XTEA algoritmasında yer alanın aksine stackte yer alan değerlerin bilinenin SHL 5 ve SHR 4 işlemine tabi tutulduğunu gördüm.

```

004629C0 . 83EC 0C      SUB ESP,0C
004629C3 . 8B4424 10    MOV EAX,DWORD PTR SS:[ESP+10]
004629C7 . 8B08         MOV ECX,DWORD PTR DS:[EAX]
004629C9 . 8B40 04      MOV EAX,DWORD PTR DS:[EAX+4]
004629CC . 53          PUSH EBX
004629CD . 56          PUSH ESI
004629CE . 8B7424 1C    MOV ESI,DWORD PTR SS:[ESP+1C]
004629D2 . 57          PUSH EDI
004629D3 . 8B3E         MOV EDI,DWORD PTR DS:[ESI]
004629D5 . 897C24 14    MOV DWORD PTR SS:[ESP+14],EDI
004629D9 . 8B7E 04      MOV EDI,DWORD PTR DS:[ESI+4]
004629DC . 897C24 10    MOV DWORD PTR SS:[ESP+10],EDI
004629E0 . 8B7E 08      MOV EDI,DWORD PTR DS:[ESI+8]
004629E3 . 8B7E 0C      MOV ESI,DWORD PTR DS:[ESI+C]
004629E6 . 897C24 0C    MOV DWORD PTR SS:[ESP+C],EDI
004629EA . BA 2037EFC6 MOV EDX,C6EF3720
004629EF . 897424 20    MOV DWORD PTR SS:[ESP+20],ESI
004629F3 . BF 20000000 MOV EDI,20
004629F8 . EB 06       JMP SHORT wirofon.00462A00
004629FA . 8D9B 00000000 LEA EBX,DWORD PTR DS:[EBX]
00462A00 > 8BF1        MOV ESI,ECX
00462A02 . C1EE 05     SHR ESI,5
00462A05 . 037424 20    ADD ESI,DWORD PTR SS:[ESP+20]
00462A09 . 8BD9        MOV EBX,ECX
00462A0B . C1E3 04     SHL EBX,4
00462A0E . 035C24 0C    ADD EBX,DWORD PTR SS:[ESP+C]
00462A12 . 33F3        XOR ESI,EBX
00462A14 . 8D1C0A      LEA EBX,DWORD PTR DS:[EDX+ECX]
00462A17 . 33F3        XOR ESI,EBX
00462A19 . 2BC6        SUB EAX,ESI
00462A1B . 8BF0        MOV ESI,EAX
00462A1D . C1EE 05     SHR ESI,5
00462A20 . 037424 10    ADD ESI,DWORD PTR SS:[ESP+10]
00462A24 . 8BD8        MOV EBX,EAX
00462A26 . C1E3 04     SHL EBX,4
00462A29 . 035C24 14    ADD EBX,DWORD PTR SS:[ESP+14]
00462A2D . 33F3        XOR ESI,EBX
00462A2F . 8D1C02      LEA EBX,DWORD PTR DS:[EDX+EAX]
00462A32 . 33F3        XOR ESI,EBX
00462A34 . 2BCE        SUB ECX,ESI
00462A36 . 81C2 4786C861 ADD EDX,61C88647
00462A3C . 83EF 01     SUB EDI,1
00462A3F . 75 BF       JNZ SHORT wirofon.00462A00
00462A41 . 8B5424 1C    MOV EDX,DWORD PTR SS:[ESP+1C]
00462A45 . 5F          POP EDI
00462A46 . 5E          POP ESI
00462A47 . 890A        MOV DWORD PTR DS:[EDX],ECX
00462A49 . 8942 04     MOV DWORD PTR DS:[EDX+4],EAX
00462A4C . 5B          POP EBX
00462A4D . 83C4 0C     ADD ESP,0C
00462A50 . C2 0800     RETN 8

```

İnternette ufak bir araştırma yaptığımda algoritmanın bu şekilde kullanımına uzak doğu sayfalarında rastlasamda çok fazla vakit harcamayarak şifreli veriyi çözen bu fonksiyonu Python'a taşımaya karar verdim ve sonunda şifreli veriyi çözmeyi başardım ve bu vesileyle şifresini unutan Wirofon kullanıcıları için ufak bir şifre kurtarma programı (şifremi hatırla seçeneğini işaretlemişseniz şifreli olarak loginSettings.dat dosyası içinde saklanan şifrenizi size gösterir) hazırlamış oldum. Programa buradan ulaşabilirsiniz.

```
C:\Documents and Settings\Administrator\Desktop\wiropwn\dist\wiropwn.exe

=====
Wirofon Şifre Kurtarma Aracı [http://www.mertsarica.com]
=====
Kullanıcı adı: http://www.mertsarica.com!http://www.hack4career.com
Şifre: wiropwn
-
=====
```

Bir sonraki yazıda görüşmek dileğiyle...