Binary Code Modification
[Patching Vulnerabilities]

Hellcode Research

Celil ÜNÜVER

celilunuver[n0sp4m]gmail.com

http://tcc.hellcode.net/musashi

Giriş

Güvenlik dünyasında şüphesiz uygulama güvenliğinin önemi tartışılamaz. Bugtraq , Full-disc vb. Mail listelerini takip ediyorsanız, birçok “0-day” güvenlik açığının yayınladığını görebilirsiniz.

Bu yazıda güvenlik zaafiyeti bulunan programları nasıl yamalayabileceginiz temel duzeyde gosterilecektir.

Patching ??

Bu makalede anlatacaklarım literatürde “**hotpatching**” veya “**runtime patching”** olarak da bilinmekte. Disassembler vb. Programlar yardımıyla bir programın “binary” kodlarının belli bir amaç için değiştirilmesine “**patching”** diyebiliriz.

Tersine mühendislerin (reverse engineers) bir çok işlemde kullandığı bir teknik. (API Hooking, Cracking , Code injection vb.) Ancak bu makalede “patching” tekniğini uygulama açıklarının yamalanması açısından ele alacağız.

Alet Çantası

Her tersine mühendisin kullandığı bilinen programlar (debuggers, disassemblers, hex editor) bu makalede anlatacağımız tekniği uygulamada bize yardımcı olabilir.
Ancak IDA Pro , Ollydbg gibi “inline assembler” ve “binary edit” özelliği olan debugger/disassembler yazılımları işimizi kolaylaştırmakta. Tabi bu “inline assemble” ve binary edit özelliklerinin “x86” çalıştırılabilir dosyaları için geçerli olduğunu unutmayın. Örneğin ARM, Xbox vb. Executable dosyaları üzerinde patch işlemi yapacaksanız, IDA sadece disassemble kısmında size yardımcı olacak, “patching” için işlemci kaynak kitaplarındaki Opcode (instruction encoding) gibi konulara göz atmanızda fayda var çünkü işiniz Hex Editor'e kalacak :) [Bu konu hakkında ilerde birşeyler karalamayı planlıyorum .]

Uygulama

Aşağıdaki hafıza taşması zaafiyeti barındıran programımız üzerinde patching işlemi yapacağız. Programı derleyin öncelikle ve artık kaynak kodunu unutun çünkü bu bizim kapalı kaynak kodlu yazılımımız.

```
#include <stdio.h>

int main()
  {
  char buf[16];
  printf("\nString giriniz:");
  scanf("%s", &buf);
  return 0;
  }
```

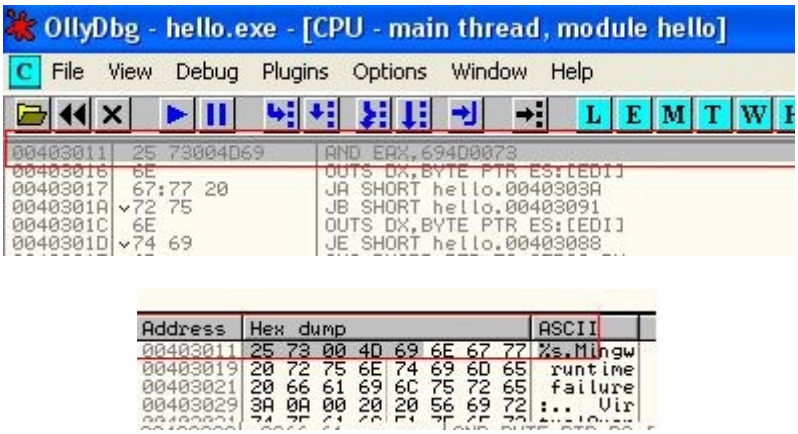
*Hafıza taşması nedir, nasıl oluşur , etkileri vb. konularına bildiğinizi varsayıp girmiyorum.

```
mov     [ebp+var_1C], eax
mov     eax, [ebp+var_1C]
call    sub_401AC0
call    sub_401770
mov     [esp+38h+var_38], offset aStringGiriniz ; "\nString giriniz:"
call    printf
lea     eax, [ebp+var_18]
mov     [esp+38h+var_34], eax
mov     [esp+38h+var_38], offset aS ; "%s"
call    scanf
```

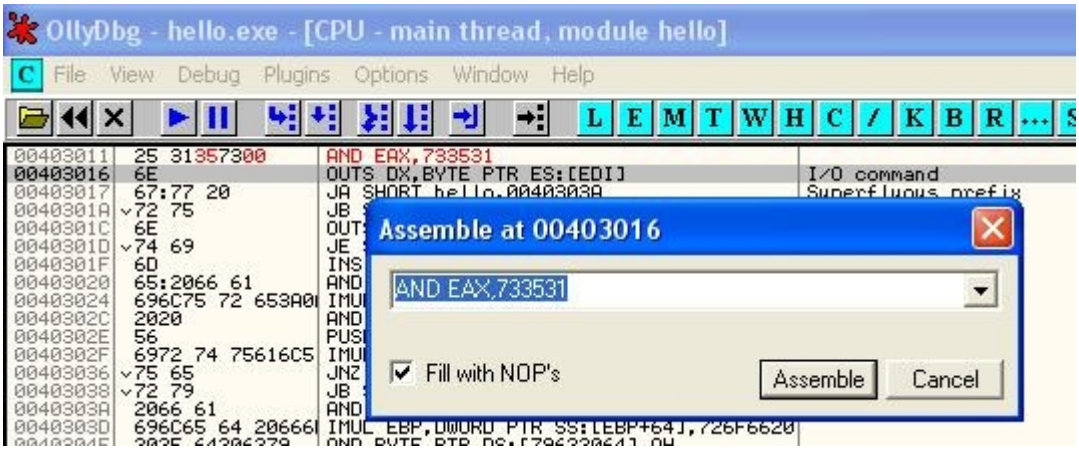
Disassembler çıktısında ve öncesinde kaynak kodda da görüldüğü gibi “scanf” fonksiyonu girilen string'in uzunluğunu kontrol etmiyor. (%s)
Bildiğiniz gibi scanf , sprintf gibi fonksiyonlarda format karakterinin önüne ekleyeceğimiz sayı ile uzunluk kontrolünü yapabiliriz. (%15s veya %.15s gibi)
Şimdi adım adım bu hatayı düzeltmeye çalışalım. Ben bu düzeltme işlemi için alet olarak Ollydbg ' ı kullanmayı tercih ediyorum. Patching işlemlerinde daha kolay bir ortam sunmakta. IDA Pro yu ise analiz işlemlerinde kullanmayı tercih ediyorum.
Ollydbg ile vulnerable programımızı açalım;

Disassembler'da gördüğümüz gibi program stringi hafızaya taşımak için “00403011” adresini çağırıyor.

Ollydbg'da CTRL+G yapıp 00403011 adresine gidelim;



AND EAX, 694D0073 satırına sağ tıklayıp Assemble seçeneğine tıklayalım. %s ' i %15s olarak değiştirip fonksiyonun “buf” değişkenine kopyalanacak stringin uzunluğu kontrol etmesini sağlayacağız.



“AND EAX, 694D0073” kodunu “AND EAX, 733531” ile değiştiriyoruz. (313573 ün ascii karşılığını ve neden tersten girdiğimizi biliyorsunuz.)

Evet bütün işlemimiz bu, programımızın vulnerable kısmını kolayca yamaladık. Programımızı bu haliyle kaydetmek için değiştirdiğimiz satıra sağ tıklayıp “Copy to executable> Selection” seçeneğine tıklıyoruz. Açılan yeni pencereyi kapatırken programın orjinalinden farklı olduğunu bu haliyle kaydetmek isteyip istemediğimizi soracak. Programımızı yeni haliyle kaydettikten sonra overflow kontrolü yapabilirsiniz :)

IDA ile yeni haline bir göz atarsak ;

```
mov     [esp+38h+var_34], eax
mov     [esp+38h+var_38], offset a15s ; "%15s"
call    scanf
mov     eax, 0
leave
retn
```

```
a15s db '%15s',0
```

Sonuç:

Yazıyı burda sonlandırmak istiyorum. Temel anlamda kafanizda birseyler olusmasi acisindan umarim faydasi olmustur. Bir sonraki yazımda aksilik olmazsa JMP / Overwrite tekniğiyle “binary patch” den bahsetmeyi planlıyorum.

Teşekkürler:

murderkey, AhmetBSD (L4M3R) , MDKgroup, C72 crew

Bağlantılar:

http://hellcoderesearch.wordpress.com
http://tcc.hellcode.net