



Binary Modification

[Patching Vulnerabilities]

Celil Ünüver
cunuver@bga.com.tr
www.securityarchitect.org

İçindekiler

Giriş	3
Patching.....	3
Alet Çantası.....	3
Uygulama.....	4

Giriş:

Güvenlik dünyasında şüphesiz uygulama güvenliğinin önemi tartışılmaz. Bugtraq , Full-disc vb. Mail listelerini takip ediyorsanız, birçok Oday güvenlik açığının yayınladığını görebilirsiniz.

Bu makalede “*disassembler*” yardımıyla güvenlik zaafiyeti bulunan programların nasıl yamalayabileceğiniz temel düzeyde gösterilecektir.

Patching ?:

Bu makalede anlatılacak olanlar 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 , Cod 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 unutmamalıyız.

Örneğin ARM, Xbox vb. Executable dosyaları üzerinde patch işlemi yaparsanız, 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 konuda yakın zamanda 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. konularını bildiğinizi varsayıp o noktalara değinmeyeceğiz.

```
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)

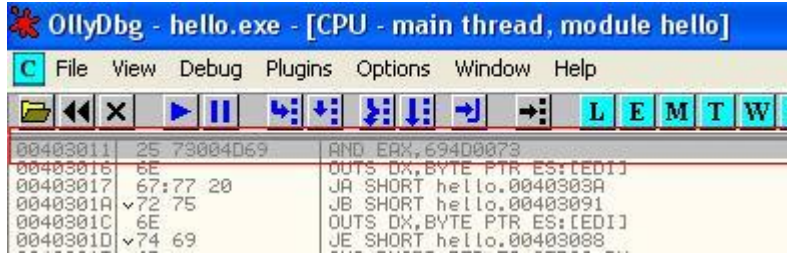
Şimdi adım adım bu hatayı düzeltmeye çalışalım. Ben bu düzeltme işlemi için alet olarak Ollydbg ‘ ı kullanmayı tercih ediyorum. Patching işlemlerinde daha kolay bir ortam sunmakta. IDA Pro yu ise analiz işlemlerinde kullanmayı tercih ediyorum.

Ollydbg ile vulnerable programımızı açalım;

Disassembler’da gördüğümüz gibi program stringi hafızaya taşımak için “00403011” adresini

304012E0	. 8B45 E4	MOV EAX, DWORD PTR SS:[EBP-1C]	
304012F0	. E8 CB070000	CALL hello.00401AC0	
304012F5	. E8 76040000	CALL hello.00401770	
304012FA	. C70424 003040	MOV DWORD PTR SS:[ESP],hello.00403000	ASCII 0A,"String gir"
30401301	. E8 22000000	CALL <JMP.&msvcrt.printf>	printf
30401306	. 8D45 E8	LEA EAX, DWORD PTR SS:[EBP-18]	
30401309	. 894424 04	MOV DWORD PTR SS:[ESP+4],EAX	
3040130D	. C70424 113040	MOV DWORD PTR SS:[ESP],hello.00403011	ASCII "%s"
30401314	. E8 07000000	CALL <JMP.&msvcrt.scanf>	scanf

çağırıyor.

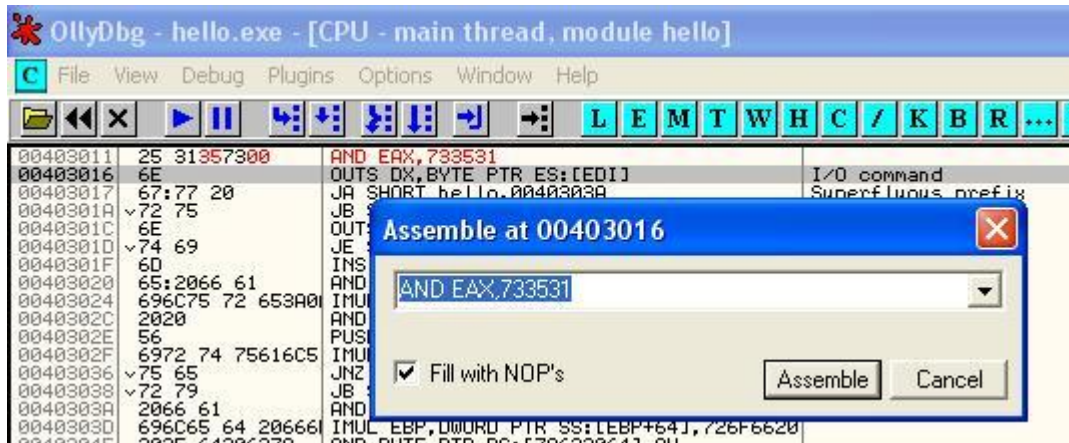


```
OllyDbg - hello.exe - [CPU - main thread, module hello]
File View Debug Plugins Options Window Help
L E M T W H C / K B R ... S
00403011 25 73004D69 AND EAX,694D0073
00403016 6E OUTS DX, BYTE PTR ES:[EDI]
00403017 67:77 20 JA SHORT hello.0040303A
0040301A 72 75 JB SHORT hello.00403091
0040301C 6E OUTS DX, BYTE PTR ES:[EDI]
0040301D 74 69 JE SHORT hello.00403088
```

Address	Hex dump	ASCII
00403011	25 73 00 4D 69 6E 67 77	%s, Mingw
00403019	20 72 75 6E 74 69 6D 65	runtime
00403021	20 66 61 69 6C 75 72 65	failure
00403029	3A 0A 00 20 20 56 69 72	... Uir

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 uzunluğunu kontrol etmesini sağlayacağız.



```
OllyDbg - hello.exe - [CPU - main thread, module hello]
File View Debug Plugins Options Window Help
L E M T W H C / K B R ... S
00403011 25 31357300 AND EAX,733531
00403016 6E OUTS DX, BYTE PTR ES:[EDI]
00403017 67:77 20 JA SHORT hello.0040303A
0040301A 72 75 JB SHORT hello.00403091
0040301C 6E OUTS DX, BYTE PTR ES:[EDI]
0040301D 74 69 JE SHORT hello.00403088
0040301F 6D OUTS DX, BYTE PTR ES:[EDI]
00403020 65:2066 61 INSDI ECX, DWORD PTR DS:[EAX]
00403024 696C75 72 653A0 INSDI ECX, DWORD PTR DS:[EAX]
0040302C 2020 AND EAX, 0
0040302E 56 PUSH ESI
0040302F 6972 74 75616C5 INSDI ECX, DWORD PTR DS:[EAX]
00403036 75 65 JNZ SHORT hello.0040303A
00403038 72 79 JB SHORT hello.0040303A
0040303A 2066 61 AND EAX, 0
0040303D 696C65 64 20666 INSDI ECX, DWORD PTR DS:[EAX]
00403045 2025 64306379 AND EAX, 0
```

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

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 kaydettikten sonra taşıp taşmadığını kontrol edebilirsiniz.

Binary Modification [Patching Vulnerabilities]

IDA ile son 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
```

Şimdiye kadar değindiğimiz noktalar temel anlamda bazı kavramlara ışık tutmaktır. İlerleyen makalelerde daha geniş kavramlara değineceğiz.