ÖN BİLGİ

Bu belge

- https://www.bgasecurity.com/makale/binary-modification-patching/

resmi adresindeki veya linkin kırık olması ihtimaline karşın alternatif olarak

- https://www.includekarabuk.com/kitaplik/indirmeDeposu/Siber_Guvenlik_Teknik_Makaleler/Teori/BaskalarinaAitMakaleler/Binary%20Modification%20[Patching].pdf

adresindeki makaleye çalışılarak elde edilen notlarımı kapsamaktadır. Bu çıkarılan notlar belgemde alıntılar ve/veya kişisel ilavelerim mevcuttur.

1)

Bugtraq, Full-disc gibi mail listeleri birçok zero day güvenlik açığını yayınlamaktadır. Takipte kalın.

(Page 3)

2)

Bu makalede “disassembler” yardımıyla güvenlik zafiyeti barındıran yazılımların nasıl yamalanabileceğini temel düzeyde öğreneceksiniz.

(Page 4)

3)

Patching Nedir?
Disassembler gibi programlar yardımıyla bir programın binary kodlarının belli bir amaç için değiştirilmesine patching denmektedir. Patching kavramı literatürde hotpatching veya runtime patching olarak da geçmektedir. Patching tersine mühendislerin birçok işlemde kullandığı bir tekniktir. Örneğin API Hooking, Cracking, Cod Injection gibi işlemlerde... Fakat bu makalede patching tekniği hedef uygulamanın açıklarını kapatmaya yönelik yamalama açısından ele alınacaktır.

(Page 3)

4)

Alet Çantası (Gereksinimler)
Hemen hemen tüm tersine mühendislik yapan mühendislerin kullandığı debuggers, disassemblers ve hex editor bu makalede anlatılacak tekniği uygulamada bize yardımcı olacaktır. Ancak inline assembler ve binary edit özelliğine sahip olan IDA Pro, Ollydbg gibi debugger'lar (disassembler'lar) işimizi hayliyle kolaylaştırmaktadır. Tabi bu arada ufak bir ayrıntı şu ki inline assembler ve binary edit özelliklerine sahip debugger (disassembler) yazılımları sadece x86 sistemlerinde çalıştırılabilmektedir.

5)
Disassembler Nedir?
Disassembler makine kodundan assembly komutlarını çıkarma işini yapan programların adıdır. Disassembler decompiler yazılımlarından ayrışmaktadır. Decompiler yazılımları makine kodundan high-level dilin kodlarını çıkarmaya çalışırken disassembler makine kodundan assembly komutlarını çıkarmaya çalışır. Disassembler'ın sunduğu çıktı, yani makine kodundan tespit edilebilen assembly komutları biz insanlar tarafından okunabilir formattadır. Fakat bir assembler'a verip makine koduna dönüştürüp çalıştıracak derecede hassas bir formatta değildir. Zaten öyle

olsaydı reverse-engineering terimi olmazdı. İşte bu okunabilen assembly komutlarından genel işleyişe hakim olmaya reverse-engineering denmektedir.

(https://en.wikipedia.org/wiki/Disassembler)

6)

Diyelim ki ARM, Xbox gibi çalıştırılabilir dosyalar üzerinde patch işlemi yapacaksınız. Bu işlem için sözgelimi IDA Pro sadece disassemble kısmında size yardımcı olacaktır. Yani makine kodundan çıkarılabildiği ölçüde bir assembly çıktısı elde etmenizi sağlayacaktır. Patching işlemi için ise CPU kaynak kitaplarını karıştırmanız gerekmektedir. Çünkü işiniz makina kodu deryası içerisinde yer alan assembly komutlarını “manuel” olarak inceleyebileceğiniz Hex Editör'lerine kalacaktır. :)

Benim NOT: Anladığım kadarıyla Patching işlemi nasıl bir yazılımın güvenlik açığını kapatan bir teknikse aynı zamanda crack'li yazılımları var eden tekniğin de ta kendisi.
Yani bir patching işlemi ile yazılımın lisansının bypass edildiği bir crack'li sürüm oluşturulabilir.

(Page 3)

7)

Uygulama
Hafıza taşması zafiyeti barındıran aşağıdaki programın üzerinde patching işlemi yapılacaktır. Böylece bu zafiyet ortadan kaldırılacaktır. Öncelikle aşağıdaki kodu bir C derleyicisinde derleyelim ve çalıştırılabilir dosyayı oluşturalım.

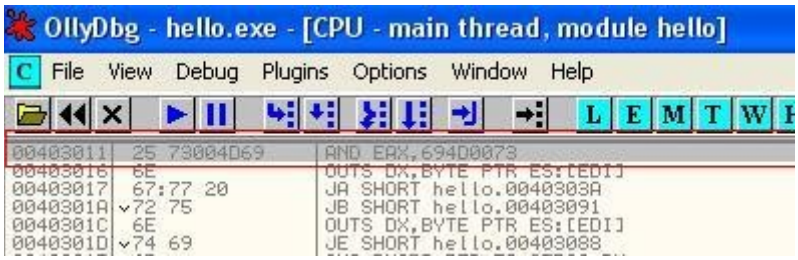
```
#include <stdio.h>

int main() {
	char buf[16];

	printf(“\nString giriniz:”);
	scanf(“%s”, &buf);

	return 0;
}
```

Bildiğiniz gibi scanf , sprintf gibi fonksiyonlarda format karakterinin önüne eklenilecek sayı ile uzunluk kontrolü sağlanabilmektedir. (%15s veya %.15s gibi) Halbuki yukarıdaki kodda scanf() fonksiyonu aldığı string'in uzunluğunu kontrol etmemektedir (%s). Yukarıdaki kod binary koda derlendikten sonra tersine muhendislik yaklaşımıyla IDA Pro adlı disassembler tarafından assembly formatına dönüştürülür.

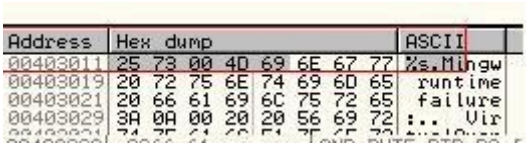
```
mov     [ebp+var_1C], eax
mov     eax, [ebp+var_1C]
call    sub_401AC0
call    sub_401770
mov     [esp+38h+var_38], offset aStringGiriniz ; "\nString giriniz:"
call    printf
lea     eax, [ebp+var_18]
mov     [esp+38h+var_34], eax
mov     [esp+38h+var_38], offset aS ; "%s"
call    scanf
```

Yukarısı zafiyet barındıran programımızın exe'sinden çıkarılan assembly kodlarıdır. Şimdi bu programımızın sahip olduğu buffer overflow açığını (hatayı) kapamak için Ollydbg programını açalım. Bu tür manipulasyon (patching) işlemleri için genellikle Ollydbg tool'u kullanılmaktadır. O nedenle Ollydbg kullanılacaktır.

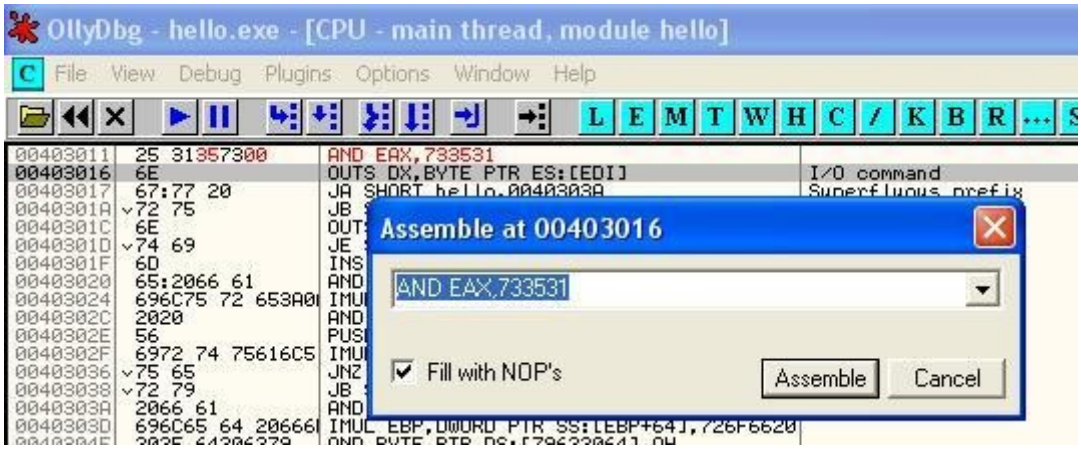
Açılan Ollydbg adlı disassembler'ın sunduğu arayüzdeki kırmızı çereveli yere dikkat edin: O kısımda program string'i aktrarmak için 00403011 adresini çağırmaktadır. Bunun üzerine Ollydbg'da CTR+G kombinasyonunu tuşlayalım ve 00403011 adresine gidelim.



AND EAX, 694D0073 kodunun bulunduğu satırın ascii karşılığına bakacak olursak

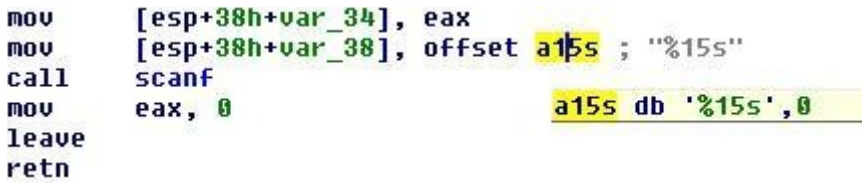


%s görünmektedir. Bu nedenle o satıra sağ tıklayıp Assembly seçeneğine tıklayalım ve %s'i %15s olarak değiştirelim. Ardından sağ tıkladığımız AND EAX,694D0073 kodunu AND EAX,733531 olarak değiştirelim (733531 sayısı nereden geldi, anlaşılmadı. PDF lifo tekniğinden bahsediyor).



Kırmızı satırdan görebileceğiniz üzere 00403011 adresindeki kod AND EAX,733531 olmuştur. Böylece programımızın açığını yamalamış olduk. Programızı bu haliyle kaydetmek için değiştirdiğimiz satıra sağ tıklayıp “Copy to executable > Selection” seçeneğine tıklamamız gerekir. Böylece programınızın yamalı halinin taşıp taşmadığını kontrol edebilirsiniz.

Yamalı programın son haline IDA Pro ile bakacak olursak:



Sarı ile vurgulanan a15s assembly kodu C'deki %15s anlamına gelmektedir. Yani yaptığımız yamayı teşkil etmektedir.

NOT: IDA Pro tool'u analiz işlemlerinde kullanıma uygunken Ollydbg ise yama türü işlemler için daha uygundur.