YAZILIM MÜHENDİSLİĞİ AÇISINDAN UYGULAMALARDAKİ ARA BELLEK TAŞMASI ZAFİYETİNİN İNCELENMESİ

Fatma AKGÜN[1] Ercan BULUŞ[2] H. Nusret BULUŞ[3]
[1,2,3] Bilgisayar Mühendisliği Bölümü
Mühendislik-Mimarlık Fakültesi
Trakya Üniversitesi, 22100, Edirne
[1]e-posta: fatmaa@trakya.edu.tr [2] e-posta: ercanb@trakya.edu.tr
[3]e-posta: nusretb@trakya.edu.tr

Anahtar Sözcükler: Bellek (Buffer), Taşma (Overflow), Hata(Bug), Security (Güvenlik)

ÖZET
Bu bildiride kullanmakta olduğumuz birçok uygulamalarda karşılaşabileceğimiz güvenlik açıklarından ve önlemlerinden bahsedilmektedir. Özellikle mevcut yazılımları alıp kullanmak ya da uygulamalarda bellek kontrolü yapmamak önemli ölçüde açık oluşturmaktadır. Bu açıklar vasıtasıyla saldırganlar çeşitli komutları çalıştırıp sistem üzerinde istediği işlemi gerçekleştirmektedirler. Bu tür durumlardan korunmak için mevcut uygulamaların kurulmadan önce incelenmesi, programcıların yazılımlarında bellek boyutunu ayarlaması vs. gibi aşağıda anlattığımız işlemleri yapmak gerekmektedir.

1. GİRİŞ
Uygulamalardaki zafiyetler nedeniyle sistemler üzerinde yaratılan açıklarla saldırganların işi daha da kolaylaşmıştır. Özellikle yazılım güvenliğinin en büyük tehlikelerinden biri ara bellek taşması olayıdır. Bu açık ile saldırgan uzak ve yakın ağdan sisteme erişip kötü amaçlı, genellikle virüs olarak nitelenen kodlarını çalıştırmayı amaçlar.

2. YAZILIM MÜHENDİSLİĞİ KAVRAMI
Bilgisayarların, hard disklerinde yüklü bir yazılım olmadan kullanılmaları hiçbir anlam ifade etmez. Ancak mevcut bir işletim sistemi sayesinde işlevlerini gerçekleştirebilirler. Yazılımlar sayesinde, bir bilgisayara uygun arabirimler kullanarak bilgi girişi ve bilgi çıkışı yapılabilir. Bu sayede bir bilgisayar, kullanıcılar için istenen hizmeti verebilir. Yazılım geliştirme süreci bazı durumlarda çok maliyet ve uzun zaman gerektirebilmektedir. Yazılım mühendisliği son yıllarda çok fazla karşılaşılan bir kavram olarak karşımıza çıkmıştır. Birçok çalışma, uzman bir proje yöneticisi önderliğinde, çeşitli gruplar oluşturularak ortaya çıkarılmaya başlanmıştır. Grup çalışması ile bu çalışmaların daha hızlı olmasının yanında hataların oluşma olasılığı da artmaktadır. Yazılım yaparken birçok programcı uygulama zayıflıklarına neden olan hatalar yapmaktadır. Bu hatalar vasıtasıyla da saldırgan bir açık yakalayıp sistem üzerinde söz sahibi olabilmektedir.

3. UYGULAMA ZAYIFLIKLARI
Uygulamalardaki açıklar; genelde yazılımlardaki yapılandırma aşamasında ya da programlama aşamasında ortaya çıkmaktadır. Bu açıklardan yaralanan saldırgan sistemde istediği uygulamayı çalıştırabilir ya da sistemin tamamen kapanmasına neden olabilir. Her bir bilgisayar programının kaynak kodlarının derinliklerinde saldırgan kişiler tarafından konulmuş sayısız korunmasız ortamlar bulunmaktadır. Zayıf programcıların program kodlarında gözden kaçırdıkları açıklar ile saldırganlar erişim kontrolünü, yıllık hesap denetim loglarını, birleşmiş sistemlerin korunmasında sayısız güvenlik ölçümlerini dakikalar içinde ele geçirebilir[1]. Mevcut yapılandırmayı değiştirmeden olduğu gibi kullanmak, karmaşık şifreler belirlememek, bellek taşırmak ve kullanıcıların uygulamalara erişim haklarını belirlememek en sık karşılaşılan programlama hatalarıdır. Bunlar içinde son zamanlarda en yaygın kod tabanlı saldırı "ara bellek" taşmasıdır. Ara bellek taşması uygulamalarda ve sistemin belirli programlarında, genellikle prosedüre çağrılarında ortaya çıkar. Ara bellek taşması belirli işletim sistemleri ve uygulamalar için yazılmıştır ve tipik olarak hedeflenen donanıma bağlı olarak kodları sömürmek için genel değişikliklere gerek duyar. Birçok durumda ara bellek taşması bütün olarak sistemlere kontrolsüz erişimlere izin verir ve önceki güvenli iş bilgisini içerir[1].

4. ARA BELLEK TAŞMASI NEDİR?
Ara bellek taşmasının ne olduğunu bilmeden önce bu konuda belli başlı kavramları bilmek gerekmektedir. Öncelikle bellek kavramına bakacak olursak;

4.1. Bellek: Bazı uygulamaları ve bu uygulamalarda kullanılan verileri geçici olarak depolamak için kullanılan birimdir. Bilgisayar sektöründe genelde RAM (Read Access Memory- Sadece okunabilir bellek) olarak adlandırılır. Ayrıca hafızada ardı ardına dizili türdeş veri tipi (örn; char, int, vs.) depolayan hafıza bloğu da diyebiliriz. Yapılan işlemler sırasında CPU (işlemci) ihtiyaç duyduğu verilere RAM

üzerinden kolayca ulaşabilir. Bu sayede daha az bekleme ile işler daha hızlı bir şekilde yapılabilir.

4.2. Ara bellek taşması: C programlama dilinde bellek array olarak geçer. Diğer bütün veri türleri gibi array'ler de static(sabit) ya da dinamik(değişken) olarak sınıflandırılabilirler. Statik değişkenler, program hafızaya yüklenirken, programın veri kısmına yerleştirilirler. Dinamik değişkenler ise program çalışırken, dinamik olarak 'stack(yığın)' denen hafizada program için hazırlanmış özel bölümde yaratılıp, yok edilirler.

Ara bellek taşması da bu dinamik değişkenlerin taşıyabilecekleri veri miktarından fazlasını yükleyerek değişkenin sınırlarını aşması olayıdır. Örneğin, 10 byte veri taşıyabilecek bir diziye 20 byte veri kopyalamak, bu belleğin taşmasına neden olmaktadır.

Örnek:

```
main(){
    char array[10];
    Strcpy (array,"bu mesaj ara bellek taşmasına
yol açar");
return 0;
}
```

Bu durumda programın çalışması sırasında ayrılan hafıza boyutu kadar veri işlem görür, diğerleri ise yasak bölge diye nitelenen yığın bölgesine yerleşmek zorunda kalır. Bu işlem bazı durumlarda programın çökmesine ya da çalışmanın durdurulmasına neden olur ve "geçersiz işlem yürütüldü" uyarısı verilmesine neden olarak önemli ölçüde açık oluşturur. Bazı programlarda ise herhangi bir değişim olmaksızın çalışmaya devam edilir. Yapılan bu işlemlerde en önemlisi de, yığın denen hafıza alanına taşan veri kötü amaçlar için yapılandırılırsa bu durumda saldırganın sistem üzerinde istediği işlemi yapması çok kolaylaşır. Bu işlemler sonucunda bu alanda "virüs" işlemleri bile oluşturulabilir. Aslında ağ güvenliği dışında başka bir açıdan bu olaya eğilirsek bu tip yazılım hatalarına "bug" denmektedir.

Bug: Yazılımın içindeki görünmeyen hatalar olarak adlandırılabilir. Belirli şartlarda ortaya çıkan hatalar veya ayrılan hafıza alanına ayrılandan daha fazla verinin yüklenmesi gibi durumlar örnek verilebilir.

Görsel olan veya olmayan herhangi bir programlama dili ile yazılım yaptığımız zaman derleyici, program kodlarımızı makine dili olan assembly kodlarına çevirir. Bu kodlar genelde birincil derecedeki komutlar ve verinin kendisini içerir. Temel olarak, her bir proses çalışmaya 3 kısımla başlar.

1-) Text Bölümü: Programın komutlarını (instruction) içeren ve sadece okunabilen bölgedir.
a-) for (x=0; x<100; x++); b-) y+=x;
Örneklerdeki C koduna denk gelen assembly komutlarını yığın bölgesinde bulabiliriz.
2-)Veri (Data) Bölümü: tanımlanmış ve tanımlanmamış verilerin bulunduğu bölgedir.
Örn: int i; Veri bölgesinin tanımlanmamış veriler kısmında bulunur. int j=10; bu ise veri bölgesinin tanımlanmış veriler için ayrılmış kısmında yer alır.
3-) Yığın olarak adlandırılan, dinamik değişkenlerin kendisinde oluşturulduğu JMP, CALL gibi fonksiyonların geri dönüş adreslerinin de geçici olarak saklandığı bölgedir. Aşağıdaki örnekteki fonksiyonda, i değişkeni yığında yaratılır ve fonksiyon çıkışında yok edilir.

```
int yaz(void)
{ int i;
   for(i=0; i<10; i++)
   putchar('X');
}
```

Yığın, LIFO (Last In First Out- son gelen ilk çıkar) mantığına göre düzenlenmiş bir veri yapısıdır. İşlemci, PUSH ve POP gibi komutlarla yığın'a veri aktarımı veya yığın'dan veri çıkarılması işlemleri yapılabilir. Korumalı moda yığın'dan veri çıkarılması veya eklenmesi 4 byte'lık (32 bit/ double word) değerler halinde olur. PUSH komutu ile ESP'de (Extended Stack Pointer) 4 byte çıkararak, yığın'a bir double word ekler ve double word'u ESP registerının içindeki adrese yerleştirir. POP komutu ise ESP registerındaki adresi okur, oradaki değeri yığın'dan çıkarır ve ESP'nin değerini 4 artırır. Yığın geçici veri depolamak, dinamik değişkenleri saklamak, fonksiyon çağrıları yaparken ana programda kalındığı yerin dönüş adresini saklamak, yerel değişkenleri depolamak ve fonksiyonlara parametre yollamak için kullanılır[2][3].

Küçük bir program üzerinde inceleme yapacak olursak;

```
1.main () {
2. printf( "Hello World" );
```

Yukarıda görünen küçük program örneği 300294 byte olarak çalıştırılabilir program haline getirilmiştir. İşletim sistemi windows NT ve derleyici program 'C' dir. Bu binary çalışabilir dosya aşağıdaki gibi bir görünüme sahiptir:

```
0x401000    55 89 e5 83 ec 10 83 3d 00 20 40 00 00 74 01 cc
0x401010    d9 7d fe 66 8b 45 fe 25 c0 f0 ff ff 66 89 45 fe
…
…
0x401040    48 65 6c 6f 20 57 6f 72 6c 64 00 55 89 55 89
0x401050    e5 e8 94 01 00 00 68 40 10 40 00 e8 92 01 00 00
0x401060    83 c4 …
```

Burada gerekli açıklamalar yapılacak olunursa;
İlk sütunda yer alan sayılar ile programın o bölümünün hangi hafıza bölgesinde yer alacağı tanımlanmaktadır. Bu hafıza adreslerinin yanında ise o hafıza alanının tutacağı değer yazılıdır. Her sütundaki veri, hafıza adres değerini bir arttırmaktadır. Örneğin 0x401000 adresinde 55 yer almaktadır. 0x401001 de ise 89 değeri saklanmaktadır. Bu sıradaki tüm adresler aynı

buradaki gibi birer artırılarak saklanırlar. Bu sayılardan bazıları komuttur. Bilgisayara belirli bir görevi yapmasını emrederler. Geriye kalanlar ise saf veridir. Burada 55 bir komuttur ve bilgisayar bu komutu alarak belirli bir görevi yerine getirir. Ardından da ikinci sütundaki 89 sayısına geçer. Bilgisayar 89 komutunun bir parametre gerektirdiğini bilmektedir. 89 komutu için gerekli parametre bir sonraki sütundaki e5 ile o komuta sağlanmaktadır. Bu durumda bilgisayar aslında 89 e5 komutunu isletir. Bilgisayar bu komutları her defasında bir tane olacak şekilde işletir. 0x401040 adresindeki hexadecimal kodları ASCII değer olarak "Hello World" verisini ifade etmektedir.

0x401040 48(**H**) 65(**e**) 6c(**l**) 6c(**l**) 6f(**o**) 20( )
57(**W**) 6f(**o**) 72(**r**) 6c(**l**) 64(**d**) 00

Bu sayılar ve harfler diğerleri ile benzerdir. Bilgisayar hafızada yer alan bu sayı ve harfleri programın akışına göre işlemeye başlar. İşte "bellek taşması" hassaslığı da bu yapıdan kaynaklanmaktadır. Eğer bir hata yapılırsa veri değerleri komutmuş gibi işlem görebilir. Onların gerçekte "Hello World" gibi "gerçek veriler" olduğunun program farkına varamaz. Program çalıştığında komutlar teker teker işlenmeye başlar, eğer ana programda bir fonksiyon çağrıldı ise bu durumda Instruction Pointer(IP) o fonksiyonun başlangıcını gösterir ve ona konumlanıp, işlem görür. Fonksiyonun çalışması bittikten sonra IP fonksiyon çağrılmadan önceki komuttan sonraki komutun adresini gösterir. Örneğin;

```
void main()
{ int i;
  i=5*5;
  toplama(a,b);
  i=8+8;
}
```

i=5+5; için gerekli assembly komutu çalıştıktan sonra, fonksiyonun bulunduğu adrese JMP vs. bir komut ile dallanılır. Bu işlemden önce çalışacak bir sonraki komutun adresi yani i=8+8; yığın'a kopyalanır. Fonksiyonun çalışması bittikten sonra daha önce saklanılan adrese konumlanılır. CALL komutu ile bu işlemler dizisi otomatik hale getirilmiştir. Bu komut çağrıldığında, önce bir sonraki komutun adresi yığın'a PUSH(ekleme) edilir, arkasından da işlenecek fonksiyonun geri dönüş adresi, yığın'dan POP edilir ve o adres EIP'ye yazılır. Böylece programın çalışmasına kalındığı yerden devam edilir. Belleğe ayrılan boyutta veri girişi olur diğerleri de bellekte programda daha sonra gelecek değişkenlerin ve daha da önemlisi fonksiyonların dönüş adreslerinin üzerine yazılırlar. Derleyici taşan bu bölgedeki verileri komutmuş gibi işlemek isteyecektir. Örneğe bakarsak program gerçekte "Hello World" e karşılık gelen alanda her bir harfin karşılığı olan hexadecimal değerini komutmuş gibi işleyecektir. Bu değerler veri olduğu için ve bunlar için herhangi bir bilgisayar komutu tanımlı olmadığı için işletim sistemi o programı "Geçersiz işlem yürüttü" uyarısı ile kapatacaktır. Bilgisayar saldırganı bu zayıf durumdan yararlanarak bu noktadaki program verilerini, kendi istediği biçimde çeşitli bilgisayar komutları ile değiştirebilir. Bu şekilde bilgisayar üzerinde kendisine açık bir kapı bırakılmasını sağlayacaktır[4].

Bu örnekte ise;

```
int i;
Void function (void)
{
    char buffer[256];
    for (i=0;i<512;i++);
            buffer[i]='a';
}
```

Veri buffer alanından taştığı zaman, veri çalışma yolunu gösteren instruction pointer (EIP) gibi kritik değerleri içeren yığındaki depolanmış verilerin bitişik alanlarının üzerine yazılır. EIP'nin üzerine yazma suretiyle, bir saldırgan programın bir sonra ne çalıştıracağını rahatlıkla değiştirebilir[5].

Yığın'da meydana gelen ara bellek taşmasının nedeni programlama dillerinin birçoğunun, tanımladıkları verilerin boyutunun belleğe sığıp sığmayacağını kontrol etmemelerinden kaynaklanmaktadır. Özellikle de C programlama dilinin bu konuda açıkları çoktur. Örneğin C programlama dilindeki **strcpy** komutu bu tip bir fonksiyondur. Veriler arasında kopyalama yaparken, kopyalanacak verinin boyutunu kontrol etmez. Programlama dillerindeki bu açık ile saldırgan kişi önce kendi kodlarını hafızanın içine yerleştirip, ara bellek taşması açığı ile bilgisayarı kandırarak kendi kodlarının işlenmesini amaçlar. Bu şekilde sistem üzerinde istediğince söz sahibi olma hakkını ele geçirir.

5. ARA BELLEK TAŞMASI NEDEN OLUŞUR?

Ara bellek taşmasının başlıca nedenleri arasında önde geleni programlama dillerinin bellek ve dizi ya da veri boyutunu kontrol etmemesinden kaynaklanır. Örnek verecek olursak C ve C++ hiçbir sınır kontrolüne sahip değildir. Sınır kontrolü bunlar gibi diğer bazı dillerde de programcılara bırakılmıştır. Fakat programcılarda bu gibi detaylara fazla dikkat göstermeyip, uygulamaların kullanımında ara bellek taşması açığına meydan bırakmaktadır. C'de kullanılan Sprintf() fonksiyonu bu sınır kontrolü yaparak bu tür açıklara engel olurken daha çok kullanılan Strcpy gibi fonksiyonlar ise tam tersi durum olarak bu tür ara bellek taşması açıklarına davetiye çıkarmaktadır. Ara bellek taşması son yıllarda %50 olarak güvenlik problemleri arasında başı çekmiştir. En erken ve en ün yapmış olanı 1988 yılında ortaya atılan Internet solucanı olmuştur ve binlerce bilgisayarda zayıflık yaratarak, kötü yönde etkilemiştir. Son yıllarda da Microsoft Index Service DDL içindeki ara bellek zayıflıkları Code Red, Code Red II ve bunları varyasyonlarını ortaya çıkarmıştır. Diğer örnek ise 2003 Ocak ayının sonlarında oraya çıkan Sapphire veya SQL Slammer olmuştur. Bunlar

ağ üzerinde çok fazlaca trafik oluşturarak, iletimin çok ağırlaşmasına neden olmuştur. Özellikle Asya'yı büyük ölçüde etkilemiştir. MS SQL Server içinde de ara bellek taşmaları görülmüştür [6].

6. ARA BELLEK TAŞMASI NASIL ENGELLENİR?

Bu türde, sistem üzerinde önemli ölçüde açık yaratan durumu engellemek için, çeşitli metotlar uygulamak gerekir. Buna çözüm olarak aşağıda sıralanan yöntemleri örnek verebiliriz.

6.1. Çalıştırılamazyığın'lar

Linux üzerinde çekirdek kısım olarak nitelenen kerneli geliştirerek bir işlemin yığın bölümünü işlemez hale getirebiliriz. Bellekten taşan veriler yığın içerisine yerleştiğinden, programın çalışması esnasında bu adreslerdeki verilerin işlenme durumu ortaya çıktığında saldırgan kişinin bu adreslere yerleştirdiği kötü amaçlı kodun çalıştırılmaması için genel bir savunma sistemi oluşacaktır.

6.2. Rastgele yığın erişimi

EIP (Extended Instruction Pointer) programın çalışması esnasında belleğe yerleştirilmiş verilerin adreslerini gösterir. EIP, taşma durumunda ESP'yi (Extended Stack Pointer) gösterir. Eğer yığın'ın bu kısmında kötü amaçlı kod var ise bu kodların çalıştırılması kaçınılmaz bir durum oluşturur. Bu durumdan sakınmak için, eğer yığın adresini her işlem çalıştığında rasgele değişen bir şekilde yapılandırırsak, saldırgan IP'yi nereye konumlandıracağını bilemez. Bu durumda, karmaşık bir yapı oluştuğundan bu tür kodların çalıştırılma olasılığı bir parçada olsa azalmış olur.

6.3. Sınırları kontrol etme

Ara bellek taşmasına neden olan neden sınır kontrolünün yapılmaması idi. Bunu önleme yöntemi de sınırları kontrol eden güvenli kodlama yapmaktır. Sınır kontrolü belleğe erişen fonksiyonlarda ve kodu değiştirebilen derleyicilerle sağlanabilir. Sınır kontrolü derleme anında olursa, program olası zayıflıklardan haberdar olur ve kodu bu yönde değiştirebilir. Bu durumda bellek taşması gibi durumlara sebebiyet bırakılmaz.

```
strcpy(smallbuff, ptr); // kontrolsüz kodlama
strncpy(smallbuff, sizeof(smallbuff), ptr); // güvenli kodlama
```

7. UYGULAMA ZAYIFLIKLARI ÖNLEME YÖNTEMLERİ

1-) Mevcut sistem üzerinde uygulamaların yeni sürümlerini kullanmak, yayınlanan tüm yamaları uygulamak, güncellemeleri tam zamanında yapmak, kontrolü elden bırakmamak,
2-) Sistemle hazır olarak gelen yapılandırmayı değiştirmek ve sisteme özel bir yapılandırma benimsemek, yapılandırmayı kendimiz adım adım yeniden yüklemek,
3-) Şifre seçiminde dikkatli davranarak kolay tahmin edilmeyecek yani az karakterli ya da kişisel bilgilerimizi içermeyen ve harf-rakam-karakter birleşiminden oluşan şifreler belirlemek ve uygulamaya özel erişim haklarının benimsenmesini sağlamak,
4-) Uygun şekilde yapılandırmak şartıyla, uygulama seviyesinde güvenlik duvarları, uygulama geçitleri ve saldırı tespit sistemleri kullanmak.

6. SONUÇ

Uygulamalardaki zayıflıklar sistem üzerinde güvenlik açıkları yarattığından bu tür durumlar için en başından itibaren önlem almak gerekmektedir. Son yıllarda ara bellek taşması(buffer overflow) çok yaygın hale gelmiştir. Modern programlama dilleri bu probleme bağışıklık kazanmıştır. Örneğin Perl ve Java gibi dillerde dizilerin boyutlanması otomatik olmaktadır. C ve C++ programlama dili bu tür problemlere karşı oldukça zayıftır. Bunun için C/C++ kütüphane fonksiyonlarının dikkatli kullanılması gerekmektedir. Sınırın aşıldığı durumları kontrol etmeyen kütüphaneleri kullanmamak gerekmektedir. Örneğin strcpy(), strcat(), sprintf() ve gets() gibi yardımcı kütüphaneler yerine strncpy(), strncat(), snprintf() ve fgets() kullanılmalıdır. Buffer'a statik ve dinamik yerleşim yapılmaktaydı bundan dolayı önerilen sabit uzunluklu buffer kullanmak yerine dinamik boyutlu buffer kullanmak gerekir [7].

KAYNAKLAR

[1] Michael LEGARY, Understanding Technical Vulnerabilities: Buffer Overflow Attacks, CISSP.

[2] Buffer Overflow'lar Hakkında, http://www.olympos.org/article/articleview/183/1/10, 13.10.2001.

[3] Pierre-Alain FAYOLLE, Vincent GLAUME, A Buffer Overflow Study, Attacks & Defenses, ENSEIRB Network and Distributed Systems 2002.

[4] Buffer Overflow Bilgisayar Yazılımı Zayıflığı Nedir?,http://arsiv.turk-php.com/makaleler/02/01/03/4056028, Sekan Hadi CEYLANİ.

[5] Eric chien and Peter Ször, Blended Attacks Exploits, Vulnerabilities and Buffer-Overflow Techniques In Computer Viruses, Symantec Corporation 2500 Broadway, Suite 200 Santa Monica, CA 90404, USA.

[6] Zili SHAO, Qingfeng ZHUGE, Yi HE, Edwin H.-M. SHA, Defending Embedded Systems Against Buffer Overflow Via Hardware/Software, Department of Computer Science, University ofTexas at Dallas.

[7] ONE Aleph , "Smashing The Stack for Fun and Profit",Phrack 49 Volume 7, Issue 49, File 14 of 16.