BLM4011
Bilişim Sistemleri Güvenliği

Take Home

Öğrenci Adı: Sinem SARAK
Öğrenci Numarası: 22011647
Dersin Eğitmeni: Ali Gökhan YAVUZ

İçindekiler

Problem 1: Jump Oriented Programming (JOP) 3
Problem 2: Stack Canaries 5
Problem 3: Integer Underflow Vulnerability 6
Problem 4: Privilege Escalation 7
Problem 5: Android Isolation 8
Problem 6: Race conditions 9
Problem 7: Setuid 10
Referanslar 11

Problem 1: Jump Oriented Programming (JOP)

İstenilen yazma işlemini gerçekleştirebilmek için, kod içerisinde bir bellek adresine yazma işlemi uygun değerler kullanılarak tetiklenmelidir. Bunun için `mov bellek_adresi, register` benzeri bir komut aranmalıdır. Verilmiş olan kod parçaları içerisinde 0x6000 adresinde bu formata uygun bir komutun yer aldığı görülür. Bu komutta yer alan eax registerına 0x8888 ve ebx 0x2222 değerleri verilerek komut çalıştırıldığında istenilen yazma işlemi mümkün olabilir. Ancak eax ve ebx registerlarına doğrudan erişilemediğinden verilen kodları ve 0x9000 – 0x9014 adres aralıklarındaki bellek içeriğini kullanarak komutları uygun sırada çalıştırarak ilerlemek gerekir.

ebx registerına 0x222 değerini atayabilmek için öncelikle 0x9004 adres alanına, yazılmak istenen değer olan 0x2222 yerleştirilir ve edx registerının değeri 0x9000 olarak ayarlanır. Bu adımın sonunda registerların ve kontrol edilebilen hafıza alanının temsili aşağıda verilmiştir:

ecx	
edx	0x9000
eip	**0x4000**

0x9000	
0x9004	0x2222
0x9008	
0x900c	
0x9010	
0x9014	

eip registerı 0x4000 değerini gösterdiğinden 0x4000'de bulunan `add edx, 4` komutu çalıştırılır ve edx'in değeri 0x9004 olarak güncellenir. Sonrasında sıradaki komut olan `mov eax, [edx]` çalıştırılır ve 0x9004 adresinde bulunan 0x2222 değeri eax registerına yüklenmiş olur. Bu değerin ebx registerına geçirilebilmesi için 0x5000 adresindeki kod çalıştırılmalıdır. Bu adrese erişebilmek için ecx içerisine 0x5000 yerleştirilir. Bu sayede `jmp ecx` satırı çalıştırıldığında eax registerının değerini ebx registerına aktaracak komut çalıştırılabilecektir. jmp komutu çalıştırıldıktan hemen sonra registerların durumlarının temsilli aşağıda verilmiştir:

eax	0x2222
ecx	0x5000
edx	0x9004
eip	0x5000

jmp komutu sayesinde eip registerına ecx registerında bulunan değerin atanması ve 0x5000 adresindeki komutun çalıştırılması mümkün olmuştur. `mov ebx, eax` komutu çalıştırılarak ebx registerına 0x222 atanması tamamlanmış olur.

eax registerına 0x8888 değerini atamak için `mov eax, [edx]` komutu kullanılır. Bu komutu kullanmadan önce 0x9008 adresine 0x8888 yerleştirilir. Bu komuta erişebilmek için en son çalıştırılmış olan komuttan hemen sonra yer alan `jmp ecx` komutu kullanılır. ecx registerına 0x4000 değeri yerleştirilerek istenen adrese gitme işlemi sağlanır. jmp komutu çalıştırıldıktan hemen sonra registerların değerlerinin temsilli aşağıda verilmiştir:

eax	0x2222
ebx	0x2222
ecx	0x4000
edx	0x9004
eip	0x4000

0x9000	
0x9004	0x2222
0x9008	0x8888
0x900c	
0x9010	
0x9014	

jmp komutu sayesinde eip registerına ecx registerında bulunan değerin atanması ve 0x4000 adresindeki komutun çalıştırılması mümkün olmuştur. Bu komuttan sonra edx registerının değeri 0x90008 olarak güncellenir ve bir sonraki komut çalıştırıldığında 0x9008 adresinde bulunan 0x8888 değeri eax registerına yerleştirilmiş olur.

Uygun değerler bu şekilde ayarlandıktan sonra 0x6000 adresindeki komutun tetiklenmesi yeterlidir. Bu işlem için jmp komutundan bir kez daha faydalanılır. ecx registerının içerisine 0x6000 değeri yazdırılır ve jmp komutu çalıştırıldıktan sonra 0x6000 adresindeki `mov [eax],ebx` komutu çalıştırılır. Bu komut 0x8888 adresine 0x2222 değerinin yazılmasını bu şekilde sağlamış olur.

Problem 2: Stack Canaries

a) -fstack-protector kullanılarak derlenen kodlarda; stack içerisinde lokal değişkenler ile return adresi arasında yer alan ve ret komutu kullanıldığında değeri kontrol edilerek stack'in taşıp taşmadığının anlaşılmasını sağlayan bir kanarya değeri bulunur[1]. Bu tanımdan da anlaşılabileceği gibi kanarya sadece return addres üzerine yapılan bir taşmayı algılayabilmektedir. Bu koruma yönteminin etrafından dolanabilmek için return addresse doğrudan müdahale etmek yerine bir struct kullanılabilir. Structlar bellekte genellikle art arda yer alan farklı elemanlardan oluşur. Örneğin şu şekilde bir struct tanımlanabilir:

```
struct VulnerableS {
     void (*func_ptr)();
     char buffer[8];
};
```

Bu struct içerisinde yer alan buffer değerine gets() veya strcpy() gibi güvenliği az fonksiyonlarla değer okuma yapılırsa bu durumda okunan değer buffer'ı aşarak func_ptr içerisine de yazılmış olur. Struct içerisindeki değerlerin arasında kanarya bulunmadığından ötürü bu şekilde planlanmış bir taşma kanarya kontrol edilerek tespit edilemez. Kullanıcı 8 baytlık buffer kapasitesini aşacak `AAAAAAAA<malicious_code_address>` gibi bir girdi verdiği zaman func_ptr değişkenini istenmeyen bir kodu çalıştırabilecek şekilde manipüle edebilir veya stack üzerinden yükleyeceği bir shellcode çalıştırabilir. Bu nedenle derleme işlemi sırasında -fstack-protector bayrağı kullanılsa bile ilgili saldırı yapılabilir.

b) NX marked bir stack kullanmak stack içerisine yüklenebilecek bir kod bloğunun çalıştırılmasını önler ancak stack'in manipüle edilmesinin önüne geçemez. Saldırgan shellcodelar çalıştırmayı denemek yerine doğrudan sistem fonksiyonlarını hedef alacak şekilde stack'i manipüle edebilir. Örneğin istediği parametreleri kullanarak doğrudan exec() sistem çağrısını çalıştırarak başka programlar da çalıştırabilir. [2] Saldırganın benzer bir metotla yapabileceği saldırıların en yaygınları arasında Return-to-libc / ret2libc saldırısı yer alır. Bu saldırı kısaca saldırganın halihazırda derlenmiş ve maplenmiş olan standart C kütüphanesindeki kodları çalıştırmasına dayanır.[3]

Problem 3: Integer Underflow Vulnerability

Verilen kod parçasında integer underflow gerçekleştirmek için unsigned integer olarak tanımlanmış iki değişken üzerinden gidilebilir. Bu durumda integer underflow'u kullanmak için hdr içerisine negatif sayılar atanabilir ve bu sayılar unsigned olarak cast edildiğinde büyük sayılara dönüştürülüp taşma tetiklenebilir.

Yapılabilecek ilk senaryo hdr->nlen = -1 vermektir. Bu durumda `if ( hdr->nlen <= 8192)` koşulu doğru olacak ve `nlen = hdr->nlen` satırında nlen değişkenine -1 atanması gerçekleşecektir. Ancak nlen sayısı unsigned olduğundan atanırken -1 i temsil eden 0xFFFFFFFF değeri bu değişken üzerinden okunurken 4294967295 sayısına karşılık gelir. Bu durum bir sonraki satırda memcpy çalıştırılırken buffer'ın taşmasına neden olur.

Yapılabilecek bir diğer çözüm ise hdr->nlen sayısını büyük bir değer vermektir. 9000 sayısını ele alalım. Bu durumda if koşulu yanlış olacağından nlen=8192 değeri korunur ve memcpy işlemi ve sonrasındaki : atama işlemi taşma olmaksızın gerçekleşir. `if (8192 - (nlen+1) <= vlen)` koşulunun sol tarafı nlen değişkeninin 8192 olması nedeniyle -1 olarak bulunur. -1<=x koşulunu doğru döndüren bir unsigned int değeri yoktur çünkü -1 hexadecimal olarak 32 bitlik bir sayının alabileceği en büyük değeri barındırır. Bu nedenle if koşulu sağlanmaz ve vlen değeri güncellenmez. Bu durumda kontrol edilen hdr->vlen değişkeni üzerinden, bufferın uzunluğu olan 8264-nlen+1 değerinden daha büyük olmak koşulu ile istenilen herhangi bir sayı verilerek memcpy içerisinde taşma işlemi gerçekleştirilebilir.

Problem 4: Privilege Escalation

Verilen dosyanın izinleri incelendiğinde bu dosyada dosya sahibi için rws şeklinde izinlerin olduğu görülür. Buradaki s SUID (Set User ID) bitinin açık olduğu anlamına gelir. Bu bitin açık olması ilgili komut çalıştırıldığında dosya sahibinin yetkileri kullanılarak çalıştırılmasını sağlar. Bu komutta dosya sahibi root'tur. Yani ping komutu çalıştırıldığında root yetkileriyle çalışır. Dosyanın diğer izinleri incelendiğinde grup izinleri kısmında rwx görülmektedir. Bu izin laura grubundaki herhangi birinin okuma yazma veya çalıştırma yapabilmesine imkan tanır. Root yetkilerini kullanarak çalışabilen bir dosyanın düzenlenebilmesi root yetkilerine sahip olmak için kullanılabilir. Laura grubunda yer alan bir kişi dosya içeriğini aşağıdaki gibi bir kod kullanarak değiştirebilir:

```
echo '/bin/bash' > /sbin/ping
```

Terminalden bu kodun yazılmasıyla dosya içeriği tamamen değiştirilerek sadece /bin/bash satırını barındıracak şekilde güncellenebilir. Bu komut çalıştırıldığında shell açılmasını sağlar. Dosya çalışırken root yetkileri ile çalıştığından, dosyayı çalıştırabilecek kişiler bu dosyayı çalıştırdıklarında root yetkisiyle çalışan bir shell açabilmiş olurlar.

Problem 5: Android Isolation

Android, uygulamaları korumak, tanımlamak veya izole edebilmek için Linux tabanlı bir koruma sisteminden faydalanır. Bu sistemde her uygulamaya eşsiz birer ID (UID) atanır ve bu sayede uygulamanın dosyalara ve işletim sistemine erişimi kısıtlanarak uygulamaların izolasyonu sağlanır.[4] Bu yapı, uygulamaların birbirinden tamamen izole bir şekilde çalışmasını sağlar. İzolasyon sayesinde, uygulamalar birbirlerinin bellek alanlarına veya dosyalarına doğrudan müdahale edemez ve aralarındaki iletişim yalnızca kullanıcı tarafından verilen izinlerle sınırlıdır. Bu sayede uygulamalar birbirlerinin dosyalarına ve kullandıkları bellek alanlarına müdahale edemezler. Bir uygulama başka bir uygulamanın alanına müdahale ettiğinde uidsi tutarsız olacağından tespit edilebilir ve erişimi önlenebilir. Bu şekilde uid kullanılarak uygulamalar, zararlı uygulamalardan korunmuş olur.

Problem 6: Race conditions

a) Bu kod parçası root olarak çalıştırıldığında bir saldırgan araya girerek yetkisi olmayan dosyalar üzerinde değişiklik yapabilir. Bunun için sleep(10) komutu çalıştırıldığında araya girebilir ve file.dat dosyası üzerinde bir sembolik link oluşturabilir. Bu sembolik linki ve kodun root yetkileriyle çalışmasını kullanarak saldırgan, normalde erişemeyeceği dosyalara erişebilir, içeriklerini değiştirebilir. Örneğin saldırgan sleep komutu esnasında /etc/passwd gibi önemli sistem dosyasını `ln -s /etc/passwd file.dat` komutunu kullanarak file.dat dosyasına bağlayabilir. Bu sayede normalde erişip değiştiremeyeceği önemli bir sistem dosyasına kod içerisinden erişmiş ve değiştirebilmiş olur.

b) sleep(10) komutu kaldırılarak bir önceki öncülde belirtilmiş olan güvenlik açığı sınırlandırılabilir ancak tamamen kaldırılamaz. Bunun nedeni yapılan işlemlerin atomik olmamasıdır. Saldırganın 10 saniye gibi uzun bir süresi olmasa da yine de uygun bir zamanlama ile if satırı ile fopen komutu arasına girerek bir önceki öncülde belirtilen linkleme işlemini gerçekleştirmesi ve ilgili kodu kullanarak normalde erişemeyeceği yerlere erişebilmesi mümkündür.

c) a şıkkındaki güvenlik açığının en önemli nedenlerinden biri işlemin atomik olmaması ve araya girilmesinin mümkün olmasıdır. Bu durumda olası bir saldırının önüne geçmek ancak atomik bir komut kullanılarak dosyanın açılmasıyla mümkün hale gelir.

```
int fd = open("file.dat", O_CREAT | O_EXCL | O_WRONLY, 0600);
if (fd == -1) {
    printf("File already exists or cannot be created");
    return;
}
```

Bu şekilde flaglerle desteklenmiş bir open komutu kullanmak atomik bir işlem yapısı sağlayarak araya girilmesini önleyecektir. Burada kullanılan O_CREAT bayrağı bir dosya yoksa oluşturulmasını sağlar. Dosya sistemde mevcutsa açılması sağlanır. Ancak dosyaya yapılan bir yazma işlemi veri tutarsızlıklarına neden olabilir. Bu nedenle O_EXCL bayrağı kullanılarak dosya üzerinde oluşacak yarış durumları da önlenmiş olur. Ayrıca duruma uygun olduğunda ek bir güvenlik önlemi olarak O_NOFOLLOW bayrağı kullanılarak sembolik bağlantılar da engellenebilir.

Problem 7: Setuid

Buradaki sorun `seteuid(100)` ifadesidir. Seteuid sadece effective uid'i değiştirir ve real ve saved uidleri değiştirmeden bırakır. Sunucu çalışmaya root olarak başladığı için tüm uid değerleri başlangıçta 0 olarak atanır. Daha sonrasında seteuid(100) komutu kullanılarak **yalnızca euid** değiştirilir. Bu büyük bir sorun oluşturur çünkü yetkisiz herhangi bir kullanıcı euid değerini ruid veya suid değerine değiştirebilir. Bu değerler seteuid kullanımından kaynaklı 0 olarak bulunmaktadır. Yani kullanıcı serve fonksiyonu içerisinde bir zafiyet bulur ve `setuid(0)` gibi bir komut çalıştırmayı başarabilirse euid değerini 0 olarak ayarlayıp normalde yetkisi olmayan işlemleri root yetkilerini kullanarak gerçekleştirebilir. Bu durumu önlemek için `seteuid(100)` ifadesi yerine `setuid(100)` komutu kullanılarak tüm uid değerleri 100'e set edilmeli ve bu sayede saldırganın serve içindeki olası bir zafiyeti kullanarak root yetkileri kazanmasının önüne geçilmelidir.

Referanslar

[1] https://www.sans.org/blog/stack-canaries-gingerly-sidestepping-the-cage/
[2] https://security.stackexchange.com/questions/47807/nx-bit-does-it-protect-the-stack
[3] https://www.ired.team/offensive-security/code-injection-process-injection/binary-exploitation/return-to-libc-ret2libc
[4] https://source.android.com/docs/security/app-sandbox