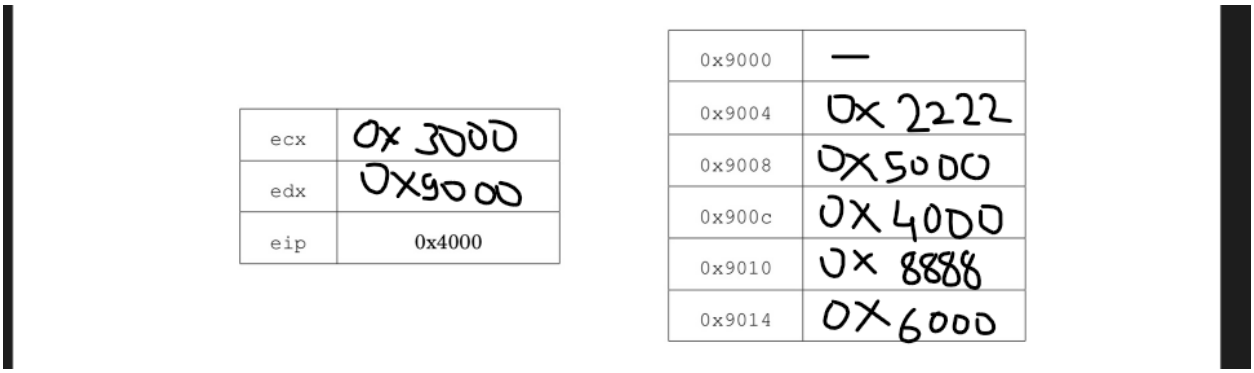
Problem 1: Jump Oriented Programming

Amaç 0x8888 adresine 0x2222 değerini yazmak. Bunu yapmak için son olarak mov [eax], ebx komutu kullanılıyor. Yani eax'de 0x8888 , ebx'de 0x2222 olmalı.



ecx	0x 3000
edx	0x9000
eip	0x4000

0x9000	—
0x9004	0x 2222
0x9008	0x 5000
0x900c	0x 4000
0x9010	0x 8888
0x9014	0x 6000

Adım adım ilerleyerek çözdüm. Eip'nin gösterdiği yerden kod çalışmaya başlar. Ecx değerini 0x3000 yaptım çünkü oradaki kodta edx'i 4 artırmak var. Edx ise 9000 çünkü memory adreslerimiz 9000'li değerler. Edx üzerine 4 ekleyerek gideceğiz. Bu bizim bellekte atlamamızı sağlayacak. İlk önce eax'e 0x2222 atıyorum sonrasında bu değeri 0x5000'de ebx'e geçiriyorum ve ebx'İ ayarlamamız bitiyor. Sonrasında tekrar eax'i asıl olması gereken değere getiriyorum. Kod aşama aşama tabloda doldurduğum değerlerle takip edilince istenilen sonucu yani 0x8888 adresine 0x2222 değerini yazma işini yerine getiriyor.

Problem 2: Stack canaries

A

B

Evet, çünkü nonexecutable stack saldırganların stacke zararlı kod enjekte edip doğrudan çalıştırmalarını engeller ve klasik stack smashing saldırılarına karşı savunma sağlar. Ancak saldırganlar bu durumu aşmak için Return Oriented Programming gibi daha gelişmiş teknikler kullanabilir. ROP, gadget olarak adlandırılan mevcut kod parçalarını kullanarak zararlı işlevler gerçekleştirmeyi mümkün kılar. Saldırgan, yığındaki geri dönüş adreslerini manipüle ederek programın kontrol akışını değiştirir ve bu sayede küçük kod parçalarının ardışık olarak çalıştırılmasını sağlar. [1],[2]

Problem 3: Integer underflow vulnerability

baştaki if (hdr->nlen <= 8192) nlen = hdr->nlen; ifadesi nedeniyle nlen en fazla 8192 olabilir. Eğer saldırgan hdr->nlen değerini tam olarak 8192'ye ayarlarsa, (nlen + 1) ifadesi 8193 olur ve sonraki if (8192 - (nlen + 1) <= vlen) { vlen = 8192 - (nlen + 1);} işlemi devreye girer. Bu durumda (8192 - 8193) negatif bir sonuç döndürür. Ancak işaretsiz tamsayılar negatif değer alamaz ve taşma meydana gelir ve buradan büyük bir pozitif sayı gelir ve vlene atanır. Ardından gelen memcpy(&buf[nlen + 1], hdr->vdata, vlen); ifadesi, belleğin sınırlarının dışına taşarak veri yazılmasına neden olur. Saldırgan bu durumu, hdr->vdata içerisine istediği zararlı veri veya komutları yazarak belleğin kontrolünü ele geçirmek ve sistemi bozmak için kullanabilir. [3]

Problem 4: Privilege Escalation

Burada potansiyel güvenlik zafiyeti, /sbin/ping dosyasının root kullanıcısına ait olmasına rağmen dosyanın laura grubuna yazma yetkisi verilmiş olmasıdır. Dosyanın izinlerinde setuid bitinin aktif olduğunu görüyoruz yani dosya çalıştırıldığında root yetkileriyle işlem yapabiliyor. Burada sıkıntı olan durum laura grubundan bir kullanıcının bu dosyayı değiştirebilmesidir. Bu zaafiyet şu şekilde kullanılabilir: laura grubundaki bir kullanıcı, /sbin/ping dosyasının içeriğini değiştirerek zararlı bir kod yerleştirir. Daha sonra bu dosya çalıştırıldığında, zararlı kod root yetkileriyle çalıştırılır. Çünkü setuid biti, dosyanın her durumda root yetkileriyle çalışmasını sağlar. Böylece, kullanıcı root yetkilerini ele geçirir

Problem 5: Android Isolation

Androidde her uygulamanın kendi UIDsi ile çalışmasının en önemli amacı uygulamaları birbirinden izole ederek güvenliği sağlamaktır. UID sayesinde bir uygulama, diğer uygulamaların süreçlerine veya verilerine doğrudan erişemez. Bu da yetkisiz veri erişimini önler. Her uygulama yalnızca kendi UID'si ile ilişkilendirilen dosyalara erişebilir. Bu mekanizma, uygulama tarafından üretilen verileri korur. UID sistemi, Androidin least privilege prensibini de destekler. Her uygulama yalnızca kendi ihtiyaç duyduğu kaynaklara eriştiği için bir güvenlik açığı durumunda zarar yalnızca ilgili uygulama ile sınırlı kalır. Eğer bir uygulama ele geçirilirse, UID sayesinde zararlı kodun sisteme veya diğer uygulamalara müdahale etmesi engellenir. Ayrıca uid kullanarak Androidde uygulamalara özel güvenlik politikaları uygulanabilir. Örneğin, sistem uygulamaları daha yüksek ayrıcalıklara sahip olabilirken, bilinmeyen kaynaklardan yüklenen uygulamalar daha sıkı kısıtlamalara tabi tutulabilir.[4]

Problem 6: Race conditions

A)

Bu kod parçası, time of check time of use olarak geçen yarış durumu olarak bilinen bir güvenlik açığına neden olabilir, bu root olarak çalışan programlar için ciddi bir güvenlik riski oluşturur. Kodun başında, stat fonksiyonu ile dosyanın mevcut olup olmadığı kontrol ediliyor, ardından 10 saniyelik sleep ile beklemeden sonra fopen ile dosya yazma modunda açılıyor. Bu iki işlem arasında olan zamanda, saldırgan dosya sistemi üzerinde değişiklikler yapabilir. Saldıran file.dat adında bir sembolik bağlantı oluşturarak bu bağlantıyı bir sistem dosyasına yönlendirebilir. Program fopen çağrısını gerçekleştirdiğinde, sembolik bağlantı çözülerek link ile yönlendirilen dosya yazma modunda açılır ve hello world ifadesi bu dosyaya yazılır.

B)

Evet, sleep(10) kısmı kaldırıldığında güvenlik açığı ortaya çıkabilir. Çünkü bu istenmeyen durum dosyanın kontrolü ve kullanım işlemleri arasında geçen herhangibi zaman diliminde meydana gelebilir. sleep(10) komutu sadece saldırganın müdahale etmek için daha fazla zaman kazanmasını sağlar. Asıl sorun stat ve fopen işlemlerinin atomik olmamasından kaynaklanır ve bu durum sleep(10) olmasa da mevcut.

C)

A şıkkındaki güvenlik probleminin önüne geçmek için dosyanın varlığını kontrol etme ve oluşturma işlemleri tek bir atomik işlemde gerçekleştirilmelidir. Unix sistem çağrısında kullanılan O_CREAT ve O_EXCL bayrakları bu sorunu çözmek için kullanılabilir. O_CREAT, dosya mevcut değilse oluşturur, O_EXCL dosya zaten mevcutsa işlemi başarısız hale getirir. Bu bayrakların birlikte kullanılması, dosyanın kontrol ve oluşturma işlemlerinin aynı anda atomik yapılmasını sağlar böylece yarış durumu oluşmaz. [5]

Problem 7: Setuid

Serve fonksiyonu, buffer overflow ve komut enjeksiyonu vb güvenlik açıklarına sahipse, kötü niyetli kişiler tarafından sistem kontrolünü ele geçirmek için kullanılabilir. Örneğin, buffer overflow kullanarak saldırgan, hazırladığı özel bir veriyi input olarak göndererek fonksiyonun stack yapısını bozabilir ve return adresini değiştirerek kendi zararlı kodunu çalıştırabilir. Eğer fonksiyon işletim sistemi komutlarını çalıştırıyorsa ve kullanıcıdan alınan input bu komutların bir parçası haline geliyorsa komut enjeksiyonu yoluyla zararlı işlemler gerçekleştirebilir. setuid(100) çağrısının başarısız olması durumunda işlem root yetkileriyle çalışmaya devam edeceği için, saldırgan bu açığı kullanarak root ayrıcalıklarıyla zararlı kod çalıştırabilir ve tüm sistemi ele geçirebilir.

Yararlanılan Kaynaklar

1. https://bluegoatcyber.com/blog/the-role-of-non-executable-stacks-in-preventing-buffer-overflows/
2. https://en.wikipedia.org/wiki/Return-oriented_programming
3. https://cwe.mitre.org/data/definitions/191.html
4. https://medium.com/@security.tecno/comprehensive-research-of-android-permission-mechanisms-09f89e6d75d8
5. https://pubs.opengroup.org/onlinepubs/7908799/xsh/open.html