TOBB EKONOMİ VE TEKNOLOJİ ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

MOBİL KİMLİK DOĞRULAMA UYGULAMALARININ GÜVENLİK İNCELEMESİ

YÜKSEK LİSANS TEZİ

Can ÖZKAN

Bilgisayar Mühendisliği Anabilim Dalı

Tez Danışmanı: Prof. Dr. Ali Aydın SELÇUK

ARALIK 2020

TEZ BİLDİRİMİ

Tez içindeki bütün bilgilerin etik davranış ve akademik kurallar çerçevesinde elde edilerek sunulduğunu, alıntı yapılan kaynaklara eksiksiz atıf yapıldığını, referansların tam olarak belirtildiğini ve ayrıca bu tezin TOBB ETÜ Fen Bilimleri Enstitüsü tez yazım kurallarına uygun olarak hazırlandığını bildiririm.

.

Can ÖZKAN

ÖZET

Yüksek Lisans

MOBİL KİMLİK DOĞRULAMA UYGULAMALARININ GÜVENLİK İNCELEMESİ

Can Özkan

TOBB Ekonomi ve Teknoloji Üniveritesi
Fen Bilimleri Enstitüsü
Bilgisayar Mühendisliği Anabilim Dalı

Danışman: Prof. Dr. Ali Aydın Selçuk

Tarih: Aralık 2020

İkili kimlik doğrulamanın uygulanması hesap ele geçirilme saldırılarına karşı şiddetle tavsiye edilen güvenlik mekanizmalarından bir tanesidir. İkili kimlik doğrulaması uygulamanın en yaygın yöntemlerinden bir tanesi bildiğiniz ve sahip olduğunuz faktörlerin bir araya getirilmesidir. Sahip olunan faktör için USB bellekler, SMS doğrulaması, mobil uygulamalar tarafından oluşturulan tek kullanımlık parola değerleri gibi seçeneklerimiz mevcuttur. Maliyet ve kolaylık nedenlerinden dolayı ikili kimlik doğrulama sürecini mobil uygulamalar tarafından üretilen tek kullanımlık parola değerleri ile dağıtmak daha yaygındır. Bununla birlikte kurcalamaya karşı dayanıklılık özelliğine sahip olan akıllı kartların aksine, saldırganlar akıllı telefonlara uzaktan ve ya fiziksel olarak erişebilir ve gizli tohum değerini okuyabilirler. Bu da mobil kimlik doğrulama uygulamaları için önemli bir güvenlik riskidir. Bu nedenle mobil kimlik doğrulama uygulamalarının bu bağlamda incelenmesi kritik önem taşımaktadır. Araştırmamızda on bir tane Android kimlik doğrulama uygulamasını inceledikten sonra bulgularımızı raporladık. Standart tersine mühendislik yöntemleri ve açık kaynak kodlu araçlar kullanarak beş uygulamada depolamadan ve yedi uygulamada bellekten gizli tohum değerini şifresiz metin olarak elde ettik.

Anahtar Kelimeler: Android, Mobil güvenlik, Tersine mühendislik, Kod karmaşıklaştırma, Kriptografik kontroller, ProGuard, Android Keystore, Mobil kimlik doğrulama, Kimlik doğrulama, İkili kimlik doğrulaması

ABSTRACT

Master of Science

SECURITY ANALYSIS of MOBILE AUTHENTICATOR APPLICATIONS

Can Özkan

TOBB University of Economics and Technology
Institute of Natural and Applied Sciences
Computer Engineering Programme

Supervisor: Prof. Dr. Ali Aydın Selçuk

Date: December 2020

Deploying Two-Factor Authentication (2FA) is one of the highly-recommended security mechanism against account hijacking attacks. One of the common methods for 2FA is to bring something you know and something you have factors together. For the later we have options including USB sticks, smart cards, SMS verification, and one-time password values generated by mobile applications (soft OTP). Due to the cost and convenience reasons, deploying 2FA via soft OTPs is more common. However, unlike smart cards which have tamper resistance property, attackers can access smartphones remotely or physically so that they can fetch shared secret seed value - an important security risk for mobile authenticators. For this reason, it is critical to analyze mobile authenticator applications in this context. In this paper, we report our findings after analyzing eleven different Android authenticator applications. We report that we have fetched cleartext shared secret seed value from storage in five applications and from memory in seven applications using standard reverse engineering techniques and open-source tools.

Keywords: Android, Mobile security, Reverse engineering, Obfuscation, Cryptographic controls, ProGuard, Android keystore, Mobile authenticator, Authentication, Two factor authentication

TEŞEKKÜR

Çalışmalarım boyunca değerli yardım ve katkılarıyla beni yönlendiren hocam Ali Aydın SELÇUK'a, kıymetli tecrübelerinden faydalandığım TOBB Ekonomi ve Teknoloji Üniversitesi Bilgisayar Mühendisliği Bölümü öğretim üyelerine ve destekleriyle her zaman yanımda olan aileme ve arkadaşlarıma çok teşekkür ederim.

İÇİNDEKİLER

ŞEKİL LİSTESİ

Sayfa

ÇİZELGE LİSTESİ

Sayfa

KISALTMALAR

2FA : İkili kimlik doğrulama (Two factor authentication)
SMS : Kısa mesaj servisi (Short message service)
OTP : Tek kullanımlık parola (One time password)
APK : Android paketi (Android package)
iOS : iPhone işetim sistemi (iPhone OS)
IPA : iOS uygulama depolama paketi (iOS application store package)
TOTP : Zaman tabanlı tek kullanımlık parola
ART : Android çalışma zamanı (Android runtime)
DVM : Dalvik sanal makinesi (Dalvik virtual machine)
JIT : Dinamik çeviri (Just in time)
AOT : Erken çeviri (Ahead of time)
PID : İşlem tanımlayıcı (Process identifier)
UID : Kullanıcı tanımlayıcı (User identifier)
IPC : Süreçler arası iletişim (Inter Process Communication)
API : Uygulama programlama arayüzü
SDK : Yazılım geliştirme kiti (Software development kit)
MAC : Ortam erişim kontrolü (Media Access Control)
IMEI : Uluslararası mobil ekipman kimliği
OWASP : Açık web uygulaması güvenlik projesi
PIN : Kişisel kimlik numarası (Personal identification number)
IETF : İnternet mühendisliği görev gücü
TEE : Güvenilir yürütme ortamı
SE : Güvenli unsur
JAR : Java arşivi
SSH : Güvenli kabuk
DEX : Dalvik çalıştırılabiliri
GUI : Grafiksel kullanıcı arayüzü
IDE : Entegre geliştirme ortamı
VM : Sanal makine
SSL : Güvenli yuva katmanı

1. GİRİŞ

21. yüzyılda kullandığımız teknoloji önemli ölçüde değişti. Çevrimiçi servisler büyük bir popülerlik kazandı [1]. Çevrimiçi servisleri tabletler, mobil cihazlar gibi taşınabilir aygıtlar aracılığıyla önceye kıyasla daha çok kullanmaya başladık ve sosyal medya, mali işlemler, çevrimiçi eğitim gibi yeni ve gitgide gelişmekte olan uygulamalar için kullanmaya başladık. Faturalarımızı öderken, alışveriş yaparken, bankacılık işlemleri yaparken, rezervasyon yaparken ve hatta uzaktan çalışırken çevrimiçi sistemlerden faydalanıyoruz [2]. İnsan hayatında İnternet bu kadar yaygınlaşırken doğal olarak kullanıcı hesapları siber saldırganlar tarafından büyük bir hedef haline geldi. Bunun sonucunda günümüzde kullanıcı hesaplarına karşı yapılan siber saldırılar yaygın hale geldi. Kullanıcı hesaplarının siber saldırganlar tarafından ele geçirilmesinin sonucu olarak kullanıcılar ve firmalar mali ve itibar kaybı olarak etkilenebilir.

Kullanıcı hesaplarının güvenliğini artırmak için uygulanan önlemlerden bir tanesi ikili kimlik doğrulamasıdır. İkili kimlik doğrulaması, iki farklı unsur aracılığıyla kimlik doğrulama işleminin yapılmasıdır. İkili kimlik doğrulama uygulandığında, iki tane farklı ve bağımsız kimlik doğrulama unsuru olduğundan dolayı kurban kişinin sadece kullanıcı adı ve parola bilgisini bilmek hesabı ele geçirmek için yeterli olmayacaktır. Kullanıcı ilk olarak, kullanıcı adı ve parola bilgilerini girer. Bu unsur "bildiğin bir şey" unsurudur ve parola, PIN gibi değerler olabilir. Ardından, "sahip olduğun bir şey" ya da "olduğun bir şey" unsuru olacaktır. Genel olan durum ise kullanıcı USB bellek, akıllı kart, donanım belirteci ya da cep telefonu gibi bir şeye sahip olduğunu kanıtladığı durum olan "sahip olduğun bir şey" durumudur. Mobil cihaz tabanlı ikili kimlik doğrulama yöntemi kolaylık ve maliyet ucuzluğu sebebiyle tercih sebebi olabilir. Kısa mesaj servisi (SMS) doğrulaması olarak uygulanabilir ve şöyle çalışır : ikinci unsur olarak sistem tek kullanımlık parola (OTP) değerini kullanıcıya cep telefonu ağı üzerinden kısa mesaj servisi olarak gönderir. İkili kimlik doğrulamasında kullanılan bir diğer uygun yöntem (ve daha güvenli) ise bir yazılım tarafından oluşturulmuş tek kullanımlık paroladır. Tek kullanımlık parola üreten

yazılımlar cep telefonlarına yüklenebilir ve çevrimdışıyken bile çalışabilir [3]. Günümüzde neredeyse herkes bir cep telefonuna sahip ve ikili kimlik doğrulamayı bu şekilde uygulamak ücretsizdir. Kısa mesaj servisi ve donanım tabanlı çözümlerin aksine, ikili kimlik doğrulamayı bu şekilde uygulamak için ek bir masrafa gerek yoktur. Bu sebeple yazılım tabanlı kimlik doğrulama uygulamalarının sağladığı güvenlik seviyesini incelemek önemli bir hal amıştır.

Yazılım tabanlı tek kullanımlık parola üreten mobil uygulamaların bir güvenlik sorunu uygulamanın genelde Android tarafı için APK, iOS için IPA dosyasına derlenen kaynak kodunu içermesidir. Tek kullanımlık parola değeri üretmek için (Örneğin TOTP algoritmasını kullanarak), gizli tohum değeri kullanılır. Eğer bu gizli tohum değer saldırganlar tarafından ortaya çıkarılırsa, saldırganlar bu değeri kullanarak uygulamanın bütün ömrü boyunca tek kullanımlık parola değerini üretebilir. Tersine mühendislik yöntemleri kullanılarak saldırganlar kullanıcıların cep telefonlarına yüklü olan uygulamaları inceleyebilir ve bu gizli tohum değerini şifresiz metin halinde elde edebilir.

Android işletim sistemi üzerinde çalışan kimlik doğrulama uygulamalarının güvenliğini inceledik. Android işletim sisteminin varsayılan olarak uyguladığı bir takım güvenlik mekanizmaları vardır. Bunlardan bir tanesi mobil güvenlik kapsayıcısıdır ve Android cihaz üzerinde yüklü olan ve çalışan bir uygulamayı, aynı cihaz üzerinde çalışan diğer uygulamalardan ve işletim sisteminin kendisinden izole eder. Buna ek olarak, yazılım geliştirme firmaları kendi uyguladıkları ek güvenlik önlemlerini de uygulayabilirler. Kod karmaşıklaştırma, kurcalama önleyici, root hesabı kontrol etme, kriptografi ve cihaz bağlama ek önlemlerden bazılarıdır fakat yazılımcıların alabileceği ek önlemler bunlarla da sınırlı değildir. Mobil güvenlik kapsayacısı ve uygulamayı geliştiren firma tarafından uygulanan güvenlik önlemlerinin birleşimi en azından teoride OTP üreten uygulamaları daha güvenli hale getirir. Tezimizin amacı seçilen kimlik doğrulama uygulamaları için pratikte bu durumları incelemek ve anlamaktadır.

Yaklaşımımız Android tersine mühendislik yöntemlerini (temelde apktool, Jadx-GUI, Frida, Objection gibi araçları kullanarak hem statik hem de dinamik bir şekilde inceleme yapmak) açık kaynak kodlu ve ücretsiz olarak sağlanan araçlarla uygulamayı içerir. Mobil kimlik doğrulama uygulamalarını mobil inceleme

ortamımıza yükledikten sonra, depolama ve/ya hafızadan gizli tohum değerine şifresiz metin olarak erişilip erişilmediğini araştırdık.

1.1 Tezin Amacı

Araştırmamızda, kullandığımız metodoloji yardımıyla hem hafızadan hem de depolama alanından yüksek miktarda kimlik doğrulama uygulamasında kullanılan gizli tohum değerlerinin şifresiz metin olarak elde edilebileceğini raporladık.

1.2 LiteratürAraştırması

TOTP algoritması üzerine bir çok araştırma yapılmasına rağmen, bunlardan bir çoğu direkt olarak gizli kök değerini elde etmeye yönelik değildi [4] [5] [6] [7]. Philip Polleit ve Michael Spreitzenbarth gizli tohum değerinin çekilmesi üzerine bir çalışmaları bulunmaktadır [8]. Araştırmalarında mobil kimlik doğrulama uygulamalarının klonlanabilip klonlanamaması, PIN koruması olup olmaması, güvenli SSL bağlantı yapıp yapmadıklarını incelediler. Ayrıca, Bernhard Mueller mobil yazılımlar tarafından üretilen tek kullanımlık parola değerlerinin Android platformunda ele geçirilmesi için kullanılan metotlar üzerine bir araştırma yayımladı. Bu yayımda araçların tanımları, araçların nasıl kullanıldığı ve hem statik hem de dinamik analiz yaparken kullanılan stratejilerden bahsetti [9].

Android Keystore bileşeninin kriptografik anahtarları güvenli bir şekilde saklamak için iyi bir yöntem olduğu üzerine çalışmalar yapılmıştır [10] [11] [12] [13]. Ayrıca, bir araştırmada ikili kimlik doğrulamayı tamamen ele geçirilmiş bir sistemde bile güvenli hale getirmeyi hedefleyen SecurePay isminde bir sistem tasarlandı ve herhangi bir ikili kimlik doğrulama işleminde aslına uygunluk ve bütünlük sağlamaktadır [14].

Bir dizi akademik çalışma güvenlik çözümleri için TOTP algoritmasının geliştirilmesi üzerine yapıldı. Imran, Mardi ve Kiki SHA256 ve TOTP algoritmalarını kullanarak mobil cihazlarda üretilen tek kullanımlık parolaların rastgeleleştirmek üzerine bir çalışma yaptı [15]. Sdf Jianxun ise sayacı zamanlayıcı ile değiştirip bir TOTP üreten algoritma tasarladı [16].

Azhari, Lucgu ve Carolus Android mobil bankacılık uygulamalarının ağ dinlenme saldırıları ve tersine mühendislikle incelenmesi zorlaştırmak üzerine araştırma yaptılar. Android mobil uygulamalar kod karmaşıklaştırması, statik katar şifreleme yöntemi, yerel C/C++ uygulaması ve ölü kod ekledikleri zaman, uygulamaların tersine mühendislik yöntemleri ile incelenmesinin daha zor olduğunu gösterdiler [17].

Katkılarımız

Aşağıda sıralanan katkılarda bulunduk;

1) Depolama tarafında incelemeye ek olarak, hafıza tarafını da ele aldık.
2) Tehdit modellememizi olabildiğince gerçeğe yakın tasarlayıp, saldırganların neler yapabilecekleri sorusuna cevap vermeye çalıştık.
3) 11 tane kimlik doğrulama uygulamasını geliştirenler tarafından uyguladıkları ek güvenlik mekanizmalarına (ProGuard, cihaz bağlama, KeyStore kullanımı vb.) göre inceledik.
4) Tersine mühendislik yöntemleri ile Android uygulamaların incelenmesini zorlaştırılması için bir kontrol listesi hazırladık.

2. ARKA PLAN

Bu bölümde tehdit modellememizi anlamak için gerekli arka plan anlatılacaktır.

2.1 Android İşletim Sistemi ve Varsayılan Güvenlik Önlemleri

Android açık kaynak kodlu ve Linux tabanlı bir işletim sistemidir. Açık kaynak kodlu olması sebebiyle geliştiriciler ve kullanıcılar tarafından tercih sebebi olmaktadır. Android, Linux çekirdeğinin üzerinde çalışır. Android işletim sisteminde Linux çekirdeği akıllı telefonlar ve tabletlerde daha verimli çalışıp, daha az pil tüketimi yapacak şekilde düzenlenmiştir. Uygulama geliştiricileri uygulamalarını Java ve Kotlin dillerinde geliştirmelerine ragmen, Android işletim sistemi servisleri genellikle C/C++ dillerinde yazılır. Android uygulamalar Android Runtime ortamı üzerinde çalışır. Android uygulamalar aslında Dalvik çalıştırılabilir dosyalarıdır. Classes.dex dosyası uygulamanın çalışması için gerekli byte kodları barındırır. Android Runtime (ART), Android 5.0 sürümünden sonra Android işletim sisteminin varsayılan işleyiş süreci ortamıdır. Android 5.0 öncesinde, Dalvik çalıştırılabilir dosyalar Dalvik Virtual Machine (DVM) üzerinde çalışıyordu.

Android Run Time öncesinde, Dalvik byte kodları çalışma zamanında makine diline uygulama çalışırken o an çeviriyordu. Bu süreç Just-in-Time (JIT) olarak adlandırılır. Android Run Time gelmesiyle birlikte, uygulamanın byte kodları makine kodlarına her uygulama çalıştığında dönüştürülmesine gerek kalmadı. Çünkü bu yeni süreç uygulamayı yükleme zamanında yapılıp, tekrardan bu işlemin yapılmasına ihtiyaç duymaz. Bu süreç de Ahead-of-Time derlemesi olarak isimlendirilir.

Android işletim sisteminin varsayılan olarak uyguladığı bir takım güvenlik mekanizmaları vardır. Aşağıdaki gibi sıralanabilir :

- **Linux Kullanıcı Tabanlı Erişim Modeli :** Android işletim sistemi üzerindeki depolama mekanizması geleneksel Linux çekirdek depolamasına kıyasla bir takım farklılıklar barındırır. Android işletim sistemi üzerinde her

bir uygulamanın /data/data dizini altında kendine özel bir dizini vardır ve uygulama verileri burada saklanır. Android işletim sistemi her bir uygulamaya farklı birer kullanıcı gibi davranır. Her bir uygulama Android işletim sistemi üzerinde farklı bir kullanıcı olarak çalıştığından dolayı, her bir uygulama için erişim kontrolü uygulama tabanlı yani kullanıcı tabanlı olarak sağlanır. Bir uygulamanın sahibi aynı zamanda PID'in de sahibidir. Bir diğer deyişle, her bir uygulamaya farklı bir süreç UID değerine göre Linux çekirdeği tarafından atanır. Android işletim sistemi böylece root olmayan kullanıcılar için dosya sistemi üzerinde erişim kontrolü uygular. Bir başka deyişle, bir uygulama başka bir uygulamaya özel veri dizinine kullanıcı tabanlı erişim modeli kapsamında erişemez.

- **Android Kum Havuzu :** Android işletim sisteminde her bir uygulama kendi kum havuzu ortamında çalışır. Varsayılan olarak uygulamalar birbirleriyle etkileşim kuramaz ve işletim sistemiyle de limitli bir etkileşim kurabilir. Bir uygulama başka bir uygulamanın kaynaklarına Android kum havuzunun kendi sanal makinesi üzerindeki her bir uygulamayı izole etmesi sebebiyle erişemez. Android, çekirdek seviyesinde kum havuzu uygulayabilmek için her uygulamaya farklı bir UID atanmasından faydalanır. Fakat normal Linux çekirdeğinde, her bir süreç o anki mevcut kullanıcının UID değeri ile çalışır. Android bu durumu alıp, bir seviye ileriye götürüp her bir uygulamaya farklı bir UID değeri atayarak uygulama kaynaklarının izolasyonuna gitmiştir.

- **Güvenli IPC :** Süreçler arası iletişim Android uygulamanın bileşenler ve cihaz üzerinde bulunan diğer uygulamalar ile nasıl güvenli bir şekilde iletişim kuracağı üzerinde süreçleri belirler.

- **Uygulama İmzalama :** Bütün Android uygulamalar Android cihaz üzerine yüklenmeden önce, gizli anahtarı geliştiricide bulunan bir sertifika ile dijital olarak imzalanmalıdır. Aksi takdirde, uygulama Android cihaz üzerinde çalışmayacaktır. Bir başka ifadeyle, imzalı olmayan hiç bir uygulama Android cihaz üzerine yüklenmeyecektir. Dolayısıyla uygulamalar yüklenmeden önce imzalanmalıdır. Android sistemi sertifikayı uygulamanın geliştiricisini tanımlamak için bir mekanizma olarak kullanır.

- **İzinler :** Uygulamalar istedikleri izinleri tanımlar fakat kullanıcılar o istenilen izinleri kabül eder ve ya etmezler. İzinler uygulamayı geliştiren geliştiriciler tarafından belirlenir. AndroidManifest.xml dosyası uygulama tarafından istenen izinlere ilişkin bütün detayları kapsamaktadır. Yüklenen uygulamalar ilgili API kullanmak için izinlere ihtiyaç duyar. İstenen izinler kullanıcı tarafından reddedilirse, uygulama API isteklerini gerçekleştiremez. Android 5.1 ve öncesi izinleri yükleme esnasında isterken, Android 6 ve sonrası izinleri ihtiyaç anında ister.

- **Google Bouncers :** Kötücül Android uygulamalarının Google Play Store üzerinde bulunmasını önlemek amacıyla, Google şirketi Google Bouncers hizmetini duyurdu ve böylelikle otomatik bir şekilde Google Play Store üzerinde bulunan uygulamaları ve geliştirici hesaplarını zararlı aktivitelere karşı izlemeye başladılar.

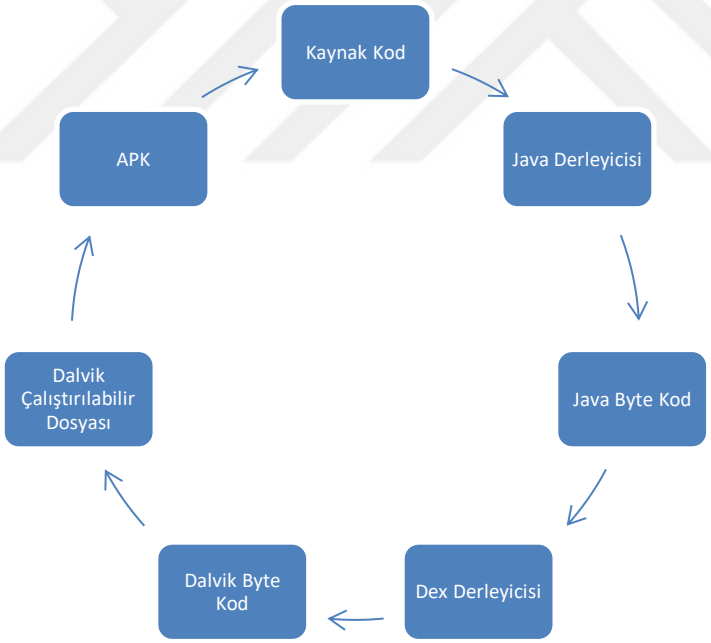
2.2 Android Uygulamalarına Genel Bakış ve Uygulama Mimarisi

Bir Android uygulaması Android işletim sistemi üzerinde çalışan uygulamadır. Android uygulamalar temelde Java ve Kotlin dilleri kullanılarak geliştirilebilir. Android ayrıca C/C++ dillerini de destekler. C/C++ dillerindeki geliştirme yerel uygulama geliştirme olarak da bilinir. Bir diğer taraftan, çapraz platform geliştirme yöntemi de vardır ve bu yöntemde geliştiriciler kodlarını geliştirdikten sonra uygulama hem Android hem de iOS platformlarına derleme işlemi gerçekleşir. Bu geliştirme tipine çapraz platform mobil uygulama geliştirme yöntemi denir ve geliştiricler React Native, Xamarin, Flutteri Ionic gibi platformlardan faydalanırlar.

Android uygulama geliştirme süreci aşağıdaki gibidir :

İlk olarak, geliştirici kaynak kodu yazar ve sonrasında kaynak kod Java derleyicisi aracılığıyla Java byte kodu üretmek amacıyla derlenir. Bu .class uzantılı dosyalar Android işletim sistemi üzerinde çalışmaz. Android işletim sisteminin Dalvik adı verilen kendi byte kod formatı vardır. Sonrasında, üretilen Java byte kodlar Dex derleyicisine girdi olarak verilir ve bu işlemin çıktısı olarak Dalvik byte kodları elde edilir. Bu işlem sonrasında Classes.dex dosyası üretilir. Java byte kodları Dalvik byte

kodlarına çevrilmiş olur çünkü Android uygulamalar temelde Dalvik çalıştırılabilir dosyalarıdır. Dalvik çalıştırılabilir dosyaları elde edilince, AndroidManifest.xml, resources, libraries ve dalvik çalıştırılabilir dosyaları bir apk paketine sıkıştırılır. APK üretildikten sonra bir sertifika ile dijital olarak imzalanması gerekir böylece bir Android cihaza yüklenmeye hazır hale gelir. Android uygulamalar kendinden imzalı olabilir yani uygulamayı geliştiren geliştirici kendi sertifikasını üretip uygulamayı o sertifika ile imzalayabilir. Geliştiricilerin Sertifika Otorite'lerinden açık anahtarlarını kayıt ettirmek için bir sertifika almalarına gerek yoktur. AndroidManifest.xml dosyası ve diğer bileşenler Android Tersine Mühendislik bölümünde daha detaylı olarak ele alınacaktır. Fakat kısaca, her bir Android uygulaması AndroidManifest.xml dosyasını barındırmak zorundadır ve bu dosya bir uygulamanın özetidir. Uygulamanın istediği izinleri, uygulamanın bileşenlerini, minimum SDK versiyonu, hedeflenen SDK versiyonu ve diğer önemli bilgileri içerir. Yukarıda anlatılan süreç Şekil 2.1 üzerinde özetlenebilir.



Şekil 2.1 : Android Uygulama Geliştirme Döngüsü

2.3 Android Uygulama Bileşenleri

Android uygulamalar Activity, Intent, Service, Broadcast Receiver ve Content Provider bileşenlerinden oluşur. Bir Android uygulama bu bileşenlerden bir kaçını ya da tamamını içerir.

- **Activity :** Aktiviteler kullanıcının etkileşimde bulunabileceği ekranlar sağlar ve böylece kullanıcılar işlem yapabilir. Genelde, bir aktivite bir ekrana karşılık gelir. Dahası, bir aktivite başka bir aktiviteyi başlatabilir. Bir uygulamadaki bütün ekranlar aslında birer aktivitedir.

- **Intent :** Intent, başka bir Android bileşeninden bir işlem istemek için kullanılan nesnedir. Bu işlemler aktivite başlatabilir, bir servis başlatabilir ve bir broadcast iletebilir.

- **Content Provider :** Content Provider bir uygulamanın verisini başka bir uygulamaya da mevcut hale getirmek için kullanılan bir bileşendir. Bir uygulama, verisini aynı cihaz üzerindeki başka bir uygulama ile paylaşmak istediğinde, Content Provider uygulama verisini cihaz üzerideki başka uygulamalarla paylaşmanın bir yoludur ve standart veritabanı fonksiyonlarını (ekleme, silme, çekme, güncelleme) gerçekleştirmek için bir ara yüz görevi görür. Fakat bir uygulama başka bir uygulamanın verisine Content Provider aracılığıyla erişmek isterse öncesinde gerekli izinleri alması gerekir.

- **Broadcast Receiver :** Sistem genelinde yayın mesajlarına yanıt veren bir bileşendir. Şarj tükeniyor ve kulaklık takıldı uyarıları bu mesajlara örnek olarak verilebilir.

2.4 Ek Uygulama Güvenliği Önlemleri

Bölüm 2.1'de Android işletim sisteminin varsayılan olarak uyguladığı güvenlik mekanizmalarını inceledik. Bir saldırgan Android işletim sisteminin uyguladığı güvenlik önlemlerini atlattığı takdirde siber saldırganın önünde başka bir güvenlik mekanizması kalmaz eğer uygulama ek olarak başka güvenlik uygulamazsa. Android işletim sisteminin varsayılan olarak sunduğu güvenlik mekanizmalarına ek olarak, uygulamayı geliştiren geliştiricinin ekleyebileceği güvenlik önlemlerinden bazıları aşağıdaki gibidir :

- **ProGuard Kod Karmaşıklaştırıcısı :** ProGuard uygulamayı küçültmek ve optimize etmek için kullanılsa da, uygulamanın kodlarını karmaşıklaştırmak

için de kullanılır ve böylece siber saldırganların uygulamayı tersine mühendislik yöntemleri ile incelenmesini zorlaştırırlar. ProGuard uygulanırken asıl amaç uygulamanın paket isimlerinin, sınıf isimlerinin, alan isimlerinin, metot isimlerinin, parameter isimlerinin kısacası kaynak koda ilişkin birçok alanı yeniden isimlendirmeye dayanır.

- **Gelişmiş Kod Karmaşıklaştırıcıları :** Gelişmiş kod karmaşıklaştırıcıları sayesinde, derlenmiş kod tabanına bilerek sözdizimsel ve anlamsal hatalar eklenir. Kod tabanına ek olarak ölü kod eklenir. Fakat bu işlemler yapılırken uygulamanın asıl davranışı korunur.

- **Android KeyStore :** Geliştiricilerin kriptografik anahtarları bir konteynırda tutmasına izin veren Android KeyStore, kriptografik anahtarların çıkartılma sürecini zorlaştırır [23]. Buradaki amaç kriptografik anahtarları kriptografik işlemlerde kullanılırken ihraç edilemez yapmaktır.

- **Yeniden Paketleme Önleyici / Yama Önleyici :** Yama işlemi yazılımı güncellemek, düzeltmek ve iyileştirmek amacıyla bir uygulama üzerinde yapılan bir dizi değişikliktir. Bir yazılımı yamamanın iyi huylu sebebi güvenlik açıklarını ve hataları düzeltmek, yeni özellikler eklemek, kullanılabilirliği ve performanı artırmak ve dahasıdır. Yamama işlemi, kaynak kodun mevcut olmadığı durumlarda derlenmiş kodun değiştirilmesini mümkün kılar. Fakat bu değişikliği yapmak için derlenmiş kodu çok iyi anlamak ve uygulamanın fonksiyonlarını detaylı analiz etmek gerekir. Bu durum yüksek seviyeli sözde kodları okumanın yanı sıra genelde assembly ve smali kodu gibi düşük seviyeli kodlarla uğraşmayı da gerektirir. Tersine, kötü niyetli amaç ise tersine mühendislik önlemlerini devre dışı bırakmak, yazılıma kötücül kod eklemek, yıllık üyelik vermemek için yıllık abonelik ücretini devre dışı bırakıp ticari yazılımları kullanmayı, yazılım korsanlığı yapmayı ve dahasını kapsar. Yama önleyici bütünlük denetimi ve anti-hata ayıklama şeklinde uygulanabilir ve uygulamanın ya hatalı çalışması ya da hiç çalışmaması ile sonuçlanır.

- **Device Binding :** Bazı değerlerin diğerlerine göre daha rastgele üretilmesi istenebilir. MAC adresi, IMEI numarası gibi cihaza özgü değerler okunup kriptografik konksiyonlara girdi olarak verildiğinde çıktılara ekstra bir rastgelelik eklenmiş olur.

- **Hassas Verileri Depolama Tarafında Şifreleme :** Gizli tohum değeri bir kimlik doğrulama uygulamasındaki en önemli değerlerden bir tanesidir. Bu değerin şifresiz metin halde tutulmaması önemlidir. Gizli tohum değerinin depolama tarafında şifreli metin halinde tutulması bir mobil kimlik doğrulama uygulamasının güvenlik postürünü artıracak ek güvenlik mekanizmalarından bir tanesidir. Bu durum OWASP en yaygın 10 mobil güvenlik zafiyetinden ikincisi olan güvensiz veri depolamaya haritalandırılabilir.

2.5 İkili Kimlik Doğrulama Teknikleri ve TOTP Protokolü

İkili kimlik doğrulama, iki farklı kimlik doğrulama unsurunun birlikte kullanıldığı kimlik doğrulama sürecidir. Bir unsur bir kullanıcı hesabını bilişim sistemine başarılı bir şekilde gerçekten söylediği hesap olduğunu doğrulama işlemidir. Günümüzde en yaygın olarak kullanılan kimlik doğrulama yöntemi kullanıcı adı / parola çiftidir. Parolalar kaba kuvet, ortadaki adam saldırısı, sözlük saldırları, Rainbow tablosu saldırılarına zafiyetli olduğundan, bireylere dijital hesaplarını güçlendirmek için ikili kimlik doğrulama uygulaması önerilir. Ek olarak, bireylerin bir çok sistemde parolaları olduğundan dolayı parolalar aynı zamanda parolanın tekrardan kullanılması, basit parola seçilmesi gibi durumlara da zafiyetlidir. Bu sebeple ikili kimlik doğrulama kullanılarak hesapların ele geçirilmesi ve olası bir veri sızıntısı engellenmesi hedeflenir.

Kimlik doğrulama yöntemleri aşağıdaki gibi sıralanabilir :

- **Bildiğin Bir Şey :** Parola, PIN, gizli soruya cevap şeklinde bilinen bir şey ile sisteme kimlik doğrulandığı durumdur.

- **Sahip Olduğun Bir Şey :** Kullanıcının elinde kredi kartı, donanım belirteci, kod gönderilebilen bir mobil cihaz, tek kullanımlık parola üreten bir uygulama gibi bir şey vardır.

- **Olduğun Bir Şey :** Bu kimlik doğrulama türü, parmak izi, retina taraması ve ses tanıma gibi insanın doğasında olan benzersiz fiziksel nitelikleri ifade eder.

- **Olduğun Bir Yer :** Bu kimlik doğrulama türünde kullanıcının kimliğinin nerede olduğuna göre kanıtlayacağına bağlıdır.

İki faktörlü kimlik doğrulamanın yaygın bir örneği, bilinen ve sahip olunan bir şeyi birlikte kullanmayı gerektirir. Bildiğiniz ve sahip olduğunuz birşey genellikle sırasıyla kimlik bilgileri ve SMS metin doğrulamasıyla elde edilebilir. SMS tabanlı 2FA, SMS mesajlarının saldırganlar tarafından ele geçirilebilmesi nedeniyle çok güvenli değildir [24] [25]. Mobil cihaz kullanarak daha güvenli ikili kimlik doğrulama gerçekleştirmenin yolları da vardır. Bunlardan biri mobil uygulama aracılığıyla doğrulama kodunu üretmektir. Google, Microsoft ve diğer büyük yazılım tedarikçileri zamana dayalı tek kullanımlık parola (TOTP) algoritmasını kullanarak OTP üreten mobil uygulamaları vardır. Kullanıcının cep telefonuna yüklenen bir mobil uygulama, TOTP protokolü yardımıyla günün saatine bağlı olarak yaklaşık 30 saniye geçerli tek kullanımlık bir kod oluşturur. Bu katı zaman çizelgesi saldırganın OTP değerini yani ikinci unsurdan kaçınmasını zorlaştırır.

2.6 TOTP Yaşam Döngüsü Yönetimi

IETF tarafından RFC 6288 olarak yayımlanan TOTP protokolü, token olarak adlandırılan ve yalnızca belli bir süre geçerli olan tek kullanımlık parola üretmede kullanılır. Oluşturulan bu token değerler, gizli tohum değerine dayanır. Ayrıca, TOTP algoritması HOTP algoritmasının bir uzantısıdır ve HOTP, HMAC based OTP anlamına gelir. RFC 4226 olarak yayımlanmıştır. HOTP algoritması bir sayaç değeri ve gizli tohum değerine bağlı olarak tek kullanımlık parolalar üretir.

Kullanıcının Google Authenticator uygulamasını mobil cihazına yüklediğini farzedelim. Kullanıcının gizli tohum değerini sağlaması için iki farklı yöntem vardır. İlk olarak, servis sağlayıcısı (Google, Microsoft, Banka vb.) gizli kök değerini ve diğer ilgili bilgileri üretir. Gizli kök değer ve diğer bilgiler (servis sağlayıcı ismi, kullanılan algoritma vb.) QR kodu içerisine gömülür. Ardından kullanıcı bu QR kodunu, Google Authenticator gibi bir kimlik doğrulama uygulaması ile tarar. İkinci yöntem ise, kullanıcı paylaşılan gizli kök değerini manuel olarak girer. Gizli tohum değerinde mutabık kalındıktan sonra, Google Authenticator gibi bir kimlik doğrulama uygulaması gizli kök değeri ve güncel saat bilgisine göre tek kullanımlık parola üretmeye başlar.

TOTP Nasıl Hesaplanır?

Daha önceki bölümlerde tek kullanımlık parola değerinin gizli tohum değeri ve güncel saat bilgisine göre üretildiğinden bahsettik. Güncel zaman damgası, Unix Epoch Time değerine dönüştürülür. 1 Ocak 1970 00:00:00 tarihinden bu yana geçen saniyeleri sayar. Artık saniyeler sayılmaz. Güncel zaman hep değiştiğinden dolayı, tek kullanımlık parola değeri her saniye değişecektir. Bu durumu engellemek için time step kavramı kullanılır ve aşağıdaki denkleme göre hesaplanır :

N = floor(T(unix) / Ts)

N = Unix Epoch zamanından beri time step sayısıdır.

floor() = Bir kesirli sayının kendisine en yakın bir küçük tam sayıya yuvarlanmasıdır.

T (unix) = 1 Ocak 1970 00:00:00 tarihinden itibaren günümüze kadar geçen saniye sayısıdır.

Ts = Time step değeri, varsayılan olarak 30 dur.

Farzedelim ki güncel zaman damgası değeri 1600519285

T (unix) = 1600519285

N (floor) = (1600519285 / 30) = 53350642

N değeri hex formatına çevrilir. Bu hex değer 16 hex karakterli yani 8 byte uzunluğunda olmalıdır. Eğer bu değer 16 hex karakter uzunluğunda değilse, değerin başına 0'lar eklenmelidir.

N(hex) = 0x0000 0000 5F65 FC75

OTP değerini hesaplamak için izlenmesi gereken adımlar aşağıdak gibidir :

1. N(hex) değerini hesaplayın.

2. Hex değerini 8 byte uzunluğundaki bir array'e çevirin ve bu değeri m (mesaj) değişkenine atayın.

3. Gizli tohum değerini (örneğin, cozkan23456cozkan) 20 byte uzunluğundaki bir array'e dönüştürün ve bu değeri bir K değişkenine atayın. Gizli tohum değeri base32 ile kodlanmış 20 byte uzunluğundaki bir değerdir. Okunabilirlik için bu gizli değer 4 karakterlik gruplara bölünmüştür hepsi küçük harften oluşmaktadır.

4. HMAC-SHA1 algoritmasını kullanarak Şekil 2.2 üzerinde gösterildiği gibi HMAC özet değerini hesaplayın. Bu HMAC özet değeri 160 bit (20 byte) uzunluğundadır. Farzedelim ki HMAC özet değeri aşağıdaki gibidir :

   9a ee 6a 13 70 32 a0 d9 b5 e5 **37 8f 89 28** 76 2f 68 3c 0f 0a

5. Üretilen özet değerin son 4 bitini alın ve 10 luk sisteme çevirin. Bizim durumumuzda, 0xA = 10. Bu tam sayı değeri bizim offset değerimizdir.

6. Offset değerine göre HMAC özetinden 4 byte alın.

7. 4 byte uzunluğundaki değer ile 7FFFFFFF değerine AND işlemi yapın.

378F8928 AND 7FFFFFFF = 5F65 FC75. Bu değer de onluk sistemde 1600519285 değerine denk gelmektedir.

Şekil 2.2 : TOTP İçerisinde HMAC Hesaplaması

8. Tek kullanımlık parola değerini hesaplayın. n = OTP boyutu, bizim örneğimizde n değeri 6 sayısıdır.

   $1600519285 \% 10^{n} = 519285$

Bu işlem hem mobil uygulamada hem de sunucu tarafında yapılır. Eğer üretilen değerler birbiriyle eşleşirse, kimlik doğrulama işlemi başarıyla gerçekleşir. Değerler birbiriyle eşleşmezse, kimlik doğrulama işlemi başarısız sonuçlanır.

2.7 Android KeyStore

Tam disk şifreleme ve dosya tabanlı şifreleme Android işletim sisteminde depolama alanı için güvenlik sağlasa da konu gizli tohum değer, kullanıcı adı/parola, oturum ID değeri, kullanıcı tokenı gibi hassas verileri korumada en iyi seçenek olmayabilir. Bu senaryodaki sorun cihaz bir saldırgan tarafından tamamen ele geçirilirse ve saldırgan cihaz üzerinde root hakkı elde ederse, kullanıcı ekran parolasını çözünce veya diskin / dosyanın şifresini çözerse saldırgan artık dosya sistemini şifresiz metin olarak görecektir. Sonuç olarak, saldırgan hassas verileri şifresiz olarak görebilecektir. Bu duruma ilişkin çözüm önerilerinden bir tanesi verileri uygulamaya

özel veri dizini altında saklamadan önce şifrelemektir. Bu durumda ise bir diğer problem ortaya çıkar : Verilerin bir noktada şifrelerinin çözülmesi gerekeceğinden dolayı şifre çözme anahtarına ihtiyaç duyulacaktır. Buradaki problem bu şifre çözme anahtarı nerede ve nasıl bir şekilde saklanacaktır? Burada Android Keystore devreye giriyor. Android Keystore'un resmi dokümantasyonunu okursak, "Android Keystore sistemi şifreleme anahtarlarını bir konteynerde saklamanıza izin verir ve cihazdan çıkartılmasını zorlaştırır. Anahtarlar KeyStore'a girdikten sonra anahtarlar kriptografik işlemler için kullanılabilir ve ihraç edilemez. Dahası, anahtarların ne zaman ve nasıl kullanılacağını kısıtlayan özellikler sunar." Buradan kriptografik anahtarları saklayabileceği (açık anahtar, özel anahtar) ve saklanan kriptografik anahtarlarda bir takım kriptografik işlemler yapılabileceğini (şifreleme, şifreyi çözme, imzalama, doğrulama vb.) çıkarsayabiliriz.

Anahtarlar Android Keystore üzerinde oluşturulur ve saklanır. Açık anahtar uygulamanın hassas verilerini şifrelemek için kullanılır. Aynı şekilde, özel anahtar gerektiğinde verilerin şifresini çözmek için kullanılır. Veriler üzerinde kriptografik işlemler yapılırken, kriptografik anahtarlar Android Keystore içerisinden asla çıkmazlar. Ek olarak, bir uygulama sadece kendi uygulamasının anahtarlarını kullanabilir.

Anahtarların çıkarılması iki farklı şekilde engellenebilir. İlk olarak, anahtarlar uygulama sürecine asla girmezler. Diğer bir deyişle, anahtar Android Keystore içerisinden asla çıkmaz. Değerler üzerinden kriptografik işlemler yapıldığından, değerler tüm şifreleme işlemlerini işleyen ve sonucu uygulamaya döndüren bir system uygulamasına sağlanır. Sonuç olarak, KeyStore'da depolanan anahtar hiç bir zaman KeyStore içerisinden çıkmaz. Diğeri, anahtarın bağlı olabileceği Trusted Execution Environment (TEE) ve Secure Element (SE) gibi bir güvenli donanım bulunur.

2.8 Android Tersine Mühendislik

Tersine mühendislik, bir sistemin bileşenlerinin tanımlaması, sistemin bir planının çıkarılması sürecidir. Buradaki amaç uygulamanın nasıl inşa edildiğini anlamaktır [26]. Android uygulama geliştirme süreci x bölümünde tartışıldı. Kısaca burada da

bahsetmek gerekirse, geliştiricilerin yazdıkları Java kodu sonunda DEX byte kodlarına derlenir. Tersine mühendisler Şekil 1' de gösterilen adımların ters yönünde çalışır. APK uzantılı olan Android uygulamaları aşağıda detaylı anlatılan bileşenlerin arşivlenmiş halidir. Bu arşivi açtığımızda, aşağıdaki gibi dosya ve klasörlerin bir listesini göreceğiz : Classes.dex, AndroidManifest.xml, META-INF, res, assets, ve lib. Fakat, bir APK dosyasını arşivden çıkarmak ve geri derlemek aynı şey değildir. Aynı şey olmamaların sebebi ise APK dosyasını sadece arşivinden çıkardığımızda hala derli olan kaynak kodu ve AndroidManifest.xml dosyalarını görürüz. Bu sorunun üstesinden gelmek için, apktool aracını kullanarak APK dosyasını geri derlememiz gerekmektedir.

Bir APK dosyasının içeriği aşağıdaki gibidir :

AndroidManifest.xml : Her bir Android uygulaması AndroidManifest.xml dosyası bulundurur. Uygulamanın özetidir ve uygulamanın ana bileşenleri, Android uygulamaya yönelik gerekli yapılandırma ayarları gibi detayları barındırır.

META-INF : Sertifikalar bu dizindedir. Uygulamayı yazan geliştirici hakkında bilgiler (isim, şirket ismi, geçerlilik süresi vb.) bilgiler tutulur. Eğer sertifika süresi dolarsa, uygulama Google PlayStore üzerinden silinir. Ayrıca, paket içerisinde bulunan dosyaların bütünlüğünü kontrol eder.

res : Bu klasör görüntüler, uygulama simgeleri, bitmapler ve katar gibi uygulama için gerekli olan ham kaynakları barındırır.

Classes.dex : Bunlar dex dosya formatındaki Dalvik byte kodlardır. Geliştiriciler tarafından yazılan kaynak kodlar günün sonunda DEX dosya formatında derlenir. Bu dex dosyası aslında Android cihaz üzerinde çalıştırılır.

lib : Bu dizin genelde C/C++ dilinde uygulanmış yerel kütüphaneleri barındırır.

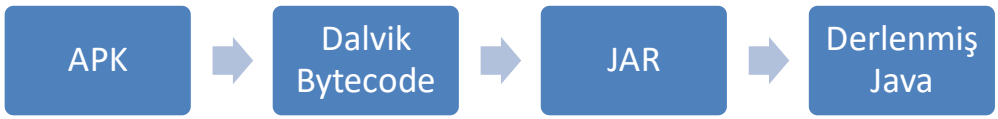
assets : Bu dizinde muzik ve video gibi dosyalar bulunur.

APK dosyamızı apktool ile geri derlediğimizde, tersine mühendisler yukarıda listelenen APK içeriklerini ve ek olarak smali kodlarının bulunduğu smali klasörünü

görürler. AndroidManifest.xml insanlar tarafından okunabilir bir formatta olacaktır. Smali, Dalvik byte kodlarının insanlar tarafından okunabilir formatıdır ve Android komutlarını ifade eder. C/C++ dillerindeki derleme kodu ile Android tarafındaki smali arasında bir analoji yapabiliriz. Smali, farklı bilgisayar mimarilerindeki Assembly dili gibidir. Daha sonra, Classes.dec dosyasını analiz etmek içim iki seçeneğimiz vardır. İlk olarak, dex dosyasını java arşiv (JAR) formatına çevirmek için dex2jar aracından faydalanırız ve java arşiv kodunu okumak için JD-GUI gibi bir Java geri derleyici kullanılırız. İkinci olarak, dex2jar aracı ve JD-GUI'yi ayrı ayrı kullanmak zorunda kalmamak için Jadx-GUI aracını kullanabiliriz. Bir diğer deyişle, hem dex2jar hem de JD-GUI kullanmak yerine sadece Jadx-GUI kullanabiliriz. Bu adımları tamamladıktan sonra, ilgili Java kaynak kodunu görebileceğiz. Ayrıca, apktool yardımı ile apk dosyasını çözdüğümüzde smali adlı bir klasör de vardır. Bu klasörde uygulamanın smali kodları bulunmaktadır. Java kodunda bir gizleme varsa, kodları eşleştirmek için hem karmaşık Java kodunu hem de smali kodu kullanılabilir. Sonuç olarak, karmaşık hale getirilmiş kodu çözersiniz.

Tersine mühendislik yaparken başarının en önemli faktörlerinden biri, analize nereden başlanacağını bulmaktır. Android uygulamalar büyük olabilir; bu nedenle, tersine mühendisler uygulamalanın tüm yönlerini analiz edemezler. Bu nedenle tersine mühendislerin analize nereden başlayacaklarını bilmeleri önemlidir. Analize nereden başlanacağına dair üç ana temel vardır. Birincisi, tersine mühendislik yaparken amacınız nedir? Çoğu durumda belirli bir soruyu yanıtlamak için tersine mühendislik uygularız. Bizim araştırmamızda bunlar OTP değerini üreten algoritmanın ne olduğu, gizli tohum değerinin nereden okunacağı, cihaz bağlama olup olmadığı, OTP değeri üretme süreci ve daha fazlasıdır. Hedefimizin ne olduğunu unutmamalıyız ve tersine mühendislik yaparken kaybolduğumuzda o amaca geri dönmeliyiz. İkinci olarak, tersine mühendisler API çağrılarını izlemelidir. Tersine mühendisler hedefleriyle ilgili API çağrılarını takip etmelidir. Son olarak, uygulama giriş noktaları. Bazen analize nereden başlayacağımızı bilemeyiz. Bu durumda, uygulamanın giriş noktaları iyi birer başlangıç noktaları olabilir [27].

Bu çalışmadaki amacımız gizli tohum değerini hem depolama hem de bellekten açık metin biçimde almaktır. Şekil 2.3 üzerinde gösterilen adımları gerçekleştireceğiz.



Şekil 2.3 : Android Tersine Mühendislik Süreci

Sonuç olarak, sözde kaynak kodunu görüyoruz ve “secret”, “totp”, “hotp”, “hash”, ”generate” gibi anahtar kelimeleri sözde kodda arayacağız. Daha sonra, nasıl çalıştıklarını derinlemesine anlamak ve paylaşılan gizli tohum değeriyle ilgili ipuçlarını aramak için API çağrılarını araştırıyor ve API çağrısı izlemesi yapıyoruz. Ek olarak, paylaşılan gizli tohum değerini değeriyle ilgili dosya dizinini görebilmek için koddaki sınıf tanımlarını ve örneklerini araştırıyoruz. AndroidManifest.xml’de mevcut OTP değerini görüntüleyen ekranla ilgili aktiviteler vardır. Ayrıca, bu faaliyetlerle ilgili intent filtrelerini ve aktiviteleri araştırıyoruz.

2.9 Android Tersine Mühendislik Araçları

Bu bölümde tehdit modelimiz ve saldırı simülasyonumuz için gerekli gerekli araçları tartışıyoruz. Uygulamaları çalıştırmadan analiz yapmak için static analiz araçları kuruyoruz. Ayrıca uygulama çalışırken analiz yapmak için dinamik analiz araçları kuruyoruz.

apktool : Kapalı kaynak derlenmiş APK’ların tersine mühendislik işlemleri için kullanılan apktool, Android uygulamalarının kodunu neredeyse original haline çözebilir ve bazı değişiklikler yaptıktan sonra yeniden oluşturabilir. Araştırmamizda uygulamaları geri derlemek ve AndroidManifest.xml, dex sınıfları, kütüphaneler ve sertifikaları analiz etmek için apktool aracını kullanıyoruz. Ek olarak, kurcalamaya karşı direnci test etmek için smali kodunda bazı değişiklikler yaptıktan sonra uygulamayı yeniden oluşturmak için apktool kullanıyoruz. Bu işlem aynı zamanda yeniden paketleme olarak da bilinir.

adb : Android Debug Bridge'i temsil eden adb, Android Emulator (bizim araştırmamızda Genymotion) ile iletişim kurmamıza izin veren bir araçtır. Sistem yönetimi dünyasındaki Secure Shell (SSH) gibidir. ADB'yi uygulamaları yükleme, Android emülatörüne uzaktn bağlantı, Android dosya sisteminde dizin geçişi, /data/data altındaki uygulamaya özel veri dizinlerini derinlemesine analiz etme, Android akıllı telefona / Android akıllı telefondan dosya gönderme ve çekme gibi görevleri gerçekleştirmek için kullanıyoruz. Özetlemek gerekirse, Unix kabuğu sağlar. İstemci sunucu modeline göre çalışır. Sunucu bağlanması için istemciyi dinler.

Jadx-GUI : Jadx-GUI (Dex'ten Java'ya geri derleyici) Android uygulamaların tersine mühendisliği için kullanılan bir araçtır. Android dex ve apk dosyalarından Java kaynak kodu üretmek için kullanılan Graphical User Interface (GUI) tabanlı bir araçtır. Dex sınıflarını ve AndroidManifest.xml kdunu çözebilmek için gennelikle Jadx-GUI aracını kullanıyoruz. Dahası, ProGuard kullanımını gizlemek için yerleşik gizleme giderici içerir. GUI tabanlı özellikler sayesinde tam metin arama, kullanım bulma, bildirime atlama, vurgu sözdizimi ile derleme kodunu görüntüleme gibi işlerimleri gerçekleştirebiliriz.

Android Studio : Android Studio, Intellij IDEA tabanlı Android uygulama geliştirme için resmi Entegre Geliştirme Ortamıdır (IDE) ve Android uygulamaları oluşturmak için gereklidir. Intellij'in güçlü kod düzenleyici ve geliştirici araçlarının yanı sıra Android Studio, Android uygulamaları oluştururken üretkenliğimizi artıran daha da fazla özellik sunar. .hprof uzantılı yığın dökümü dosyalarını analiz etmek için Android Studio kullanıyoruz.

Memory Profiler : Bellek Profilcisi, Android Profiler'da bellek sızıntılarını belirlemenize yardımcı olan bir bileşendir. Uygulamanın bellek kullanımını gerçek zamanlı grafik halinde gösterir ve bir yığın dökümü yakalamanıza, çöp toplamalarınızı zorlamanıza ve bellek ayırmalarınızı izlemenize olanak tanır.

JD-GUI : JD-GUI .class dosyalarının Java kaynak kodlarını görüntüleyen bağımsız bir grafik yardımcı programıdır. Metotlara ve alanlara anında erişim için yeniden

yapılandırılmış kaynak koduna JD-GUI ile göz atılabilir. Sınıf dosyalarına ve Java modülleri hiyerarşisine göz atmanıza izin verir.

Frida : Frida, geliştiriciler, tersine mühendisler ve güvenlik araştırmacıları için çalışma zamanında JavaScript kodu enjekte etmelerine olanak tanıyan dinamik bir araç setidir. Bu araç, araştırmacıların uygulamalara bağlanmasına ve çalışma süresianalizi gerçekleştirmesine olanak tanır. Frida'nın yetenekleri arasında bunlarla sınırlı olmamak üzere çalışma zamanı uygulama analizi, Java neslerin nesnelerinin oluşturulması, Java metotlarının ezilmesi, bir katarın oluşumları için işlem belleğini tarama, yerel fonksiyon çağrılarını analiz etme yer alır.

Objection : Nesne enjeksiyonunun kısaltması olan Objection, tersine mühendislerin SSL sabitlemeyi atlatmasına, keystore analizine, sınıfları keşfetmesine, uygulamaları yamamaya, dosya sistemiyle etkileşime girmesine, yüklenen sınıfları keşfetmesine ve daha fazlasına olanak sağlayan bir çalışma zamanı mobil keşif araç setidir. Objection, Frida dinamik enstrümantasyon araç seti tarafından desteklenmektedir.

SQLite Browser : Android yerleşik veritabanı uygulaması olan SQLite veritabanına sahiptir. SQLite kullanıcıların ve geliştiricilerin veritabanı üzerinde CRUD (Oluşturma, okuma, güncelleme ve getirme) işlemlerini gerçekleştirmesine olanak tanıyan açık kaynaklı bir veritabanıdır. SQLite Browser, SQLite veritabanına bağlanmak ve veritabanı işlemlerini gerçekleştirmek için kullanılan bir araçtır.

3. PROBLEM TANIMI, TEHDİT MODELLEMESİ ve VARSAYIMLAR

3.1 Genel Bakış

Bu bölümde problemimizin detaylı anlatımı, tehdit modellemimizin adımarı ve tehdit modellemimizi uygulamadan önceki varsayımlarımız anlatılmaktadır.

OTP değerinin temel olarak iki parametreli TOTP algoritmasının bir ürünü olduğunu hatırlayın. Bu iki parametre, paylaşılan gizli tohum değeri ve geçerli zaman damgası olan hem sunucu hem de mobil istemci için aynıdır. Tek kullanımlık parola üretebilmekiçin provizyon olarak da bilinen kayıt işlemi yapılmalıdır. Kayıt işlemi sırasında paylaşılan gizli tohum değeri üzerinde anlaşmaya varılır. Hazırlık tamamlandıktan sonra, paylaşılan gizli tohum değeri iki varlık arasında paylaşılmış olur.

Kullanıcıların hesaplarında oturum açtıklarında veya 2FA korumalı bir işlem gerçekleştirdiklerinde, tek kullanımlık parola değerine sahip olduklarını kanıtlamaları gerekir. Paylaşılan gizli tohum değeri olarak, özet fonksiyon ve geçerli zaman damgası iki varlık için aynıdır ve hem sunucu hem mobil istemci kimlik doğrulama uygulaması aynı tek kullanımlık parola değerini üretir. Bu durumda sorun saldırganlar için her bir kullanıcının paylaşılan gizli tohum değeri dışındaki diğer değerlerin bilinmesidir. Saldırganlar bir şekilde paylaşılan gizli tohum değerini öğrenirse, ikinci faktörü atlatabilirler. Ana sorunumuzu “paylaşılan gizli tohum değerini bilmek, OTP uygulanan yazılım tabanlı kimlik doğrulama uygulamalarını atlatmak için yeterli olabilir” şeklinde sonuçlandırabiliriz.

Varsayımlara gelince, provizyon adımının başarıyla tamamlandığını ve mobil kimlik doğrulama uygulamasının en az bir kez kapatıldığını varsayıyoruz. Mobil kimlik doğrulama uygulaması en az bir kez kapatılınca atak simülasyonumuz daha gerçekçi olacaktır. Ardından uygulamayı çalıştırın ve uygulama OTP üretmeye başlayacaktır.

Ayrıca, bu araştırmada saldırganın sisteme root erişimi vardır. Bu durum aşağıdaki şekilde gerçekleştirilebilir:

1) Android cihaz rootludur.

2) Saldırgan sistemdeki bir zafiyetten faydalanarak sistem üzerinde root yetkisi ile kod çalıştırır.

3) Cihaza fiziksel erişim vardır.

Günümüzde, Android cihazı root işlemi uygulamak nadir değildir ve her geçen gün root haklarıyla kod çalıştırılmasına izin veren zararlı yazılımlar ve zafiyetler çıkmaktadır [28] [29] [30] [31] [32] [33]. Google PlayStore dahil olmak üzere çeşitli mağazalardaki bazı uygulamaların çalışması için root iznine ihtiyaç duyar. Tehdit modellememiz günümüz dünyası için gerçekçi bir senaryodur.

Araştırmamızda tehdit modellememiz gizli anahtarı hem depolamadan hem de bellekten almaya çalışır. Kara kutu testleri gerçekleştirdik, yani uygulamalarla ilgili herhangi bir bilgimiz ve kaynak koda erişimimiz yoktur. APK dosyasını alınır ve analiz edilir. Akıllı telefon olarak Google Nexus 5 (API Level 23), Samsung Galaxy S 10 (API Level 29) ve emülatör olarak da Genymotion kullandık. VirtualBox ile sanallaştırılmış Ubuntu Virtual Machine (VM) üzerine gerekli araçları (apktool, adb, jadx-GUI, Android Studio, Frida, Objection) kurduk. Ayrıca Windows 10 istemci makineye Android Studio kurulur. Bu makineler aynı alt ağdadır.Tehdir modellememiz şöyle çalışır : APK indirildikten sonra, APK geri derlenir. Ardından, APK'nın işlevlerini, metotlarını ve OTP işlem süreciyle ilgili daha derin bilgi edinmek için statik kod analizi yapılır. APK'yı derin bir şekilde inceledikten sonra, APK adb kullanarak emülatöre yüklenir. Devamında manuel olarak gizli tohum değerini girip provizyon işlemini tamamlıyoruz. Araştırmamızda uygulamanın kullandığı gizli tohum değerini elde ettiğimizden emin olmak için uygulamanın seçeneği varsa gizli tohum değerini manuel olarak elle girilir. Ardından kimlik doğrulayıcı uygulama kapatılır. Sonra telefon kapatılır. Telefonu açtıktan sonra, hedeflenen kimlik doğrulama uygulamasını başlatır ve paylaşılan gizli kök değerine saldırılır. Bu adımlar tamamlandıktan sonra bir sonraki adıma geçilir.

3.2 Hafızadan Gizli Tohum Değerini Okumanın Tehdit Modellemesi

Hafızadan gizli kök değerini almak söz konusu olduğunda iki farklı prosedür vardır. İlk prosedür aşağıdaki gibidir ve adb aracından faydalanılır.

1) OTP değeri ekranda görülür.

2) OTP değeri ekranda görüldükten sonra adb kullanılarak emülatöre bağlantı yapılır.

3) Kimlik doğrulayıcı uygulamanın işlem kimliğini, yani PID değerini ps komutu aracılığı ile tanımlanır.

4) Geri dönmek için exit komutu uygulanır.

5) adb kullanılarak bir yığın dökümü oluşturulur.

6) Analiz için oluşturulan dosya ana makineye çekilir.

7) Bellek Profiler özelliğini kullanarak Android Studio'daki yoğın dökümü analiz edilir.

Diğer yol ise doğrudan Android Studio uygulamasını kullanmaktır. Bu prosedür kapsamında gizli tohum değerini almak için izlediğimiz adımlar aşağıdaki gibidir:

1) Android Studio'yu açın.

2) Android Profiler düğmesine tıklayın.

3) "Yeni bir profil oluşturma oturumu başlat" düğmesine basın.

4) Uygun emulatörü seçin ve ardından kimlik doğrulayıcı uygulamasını açın.

5) Android Profilleme başlatılır.

6) Uygulamanın ilk kurulum ve provizyon süreci tamamlanır.

7) Android Profiler'a geri dönülür ve bellek bölümü seçilir.

8) Java yığın dökümüne basılır.

9) Hafıza analiz edilir.

İstenilen kadar yığın dökümü alınabilir. Ancak Java yığın dökümünü art arda saniyelerle alınamaz. Bir yığın dökümü alınır, en az 3 - 4 saniye ardından bir sonraki yığın dökümü alınabilir. Bu durum, Tehdit Modellememizin Eksikleri bölümünde tartışılan açık kaynak araçlarının temel sınırlamasıdır.

3.3 Depolamadan Gizli Tohum Değerini Okumanın Tehdit Modellemesi

Depolamadan gizli tohum değerinin elde edilmesi için izlenen adımlar aşağıda verilmiştir.

1) OTP değeri ekranda görülür.

2) Mobil kimlik doğrulama uygulamasının gizli tohum değerini nerede sakladığını bulmak için analiz yapılır.

3) Dosya sisteminde o dizine gidilir.

4) Dosya sisteminde o dizine gidilir ve dosya analiz edilir.

3.4 Tehdit Modellememizin Eksikleri

Araştırmamizda sadece açık kaynak kodlu araçlar kullanıldı. Herhangi bir ticari yazılım kullanılmadı. Tehdit modellememizde ki temel eksiklik gizli tohum değerinin bellekten alınmasıdır. adb kullanılarak bir yığın dökümü alınabilir. Komut: adb shell am dumpheap PID /data/local/tmp/appname.prof

Bu komut istenilen zaman çalıştırılır ve o zamanın yığın dökümü alınır. Bu zamana t0 diyelim. En az 3 – 4 saniye sonra başka bir yığın dökümü alınabilir. Ne yazık ki ardışık iki saniyeninyığın dökümü alınamaz. Bu, bellekten gizli tohum değerini

getirmeye ilgili ana sınırlamamızdır. Bununla birlikte, uygulamanın OTP değerini üretmesi için, gizli tohum değerinin bir noktada açık metin olarak bellekte olması gerekir. Ardışık saniyelerin yığın dökümü alınamadan, bellekten açık metin olarak gizli tohum değerini getirilemeyebilir.

4. SALDIRI SİMÜLASYONU

4.1 Seçilmiş Uygulamalar

Seçilen uygulamalar şunlardır : Google Authenticator, Microsoft Authenticator, Red Hat Free OTP, Blizzard Authenticator, Epic Authenticator, Oracle Authenticator, SAP Authenticator, Sophos Authenticator, Twilio Authy Authenticator, Pixplicity Authenticator ve SaasPass Authenticator uygulamalarıdır.

Google PlayStore'da daha fazla kimlik doğrulama uygulaması bulunmasına rağmen, bütün kimlik doğrulama uygulamalarını analiz etmek mümkün değildir. Temsile ilişkin 11 tane kimlik doğrulama uygulaması seçmek yeterlidir. Google PlayStore üzerindeki popülerliklerine göre 11 tane kimlik doğrulama uygulaması seçtik. Ayrıca seçilen uygulamalar PlayStore üzerinde ücretsiz olarak dağıtılmaktadır. Ek olarak, seçilen bütün kimlik doğrulama uygulamaları TOTP ve HOTP algoritmalarını uygulamaktadır.

4.2 İncelenen Güvenlik Önlemleri (Ne ve Nasıl?)

Bu bölümde ek güvenlik mekanizmalarının uygulamalara uygulanıp uygulanmadıklarını nasıl test edildikleri açıklanır.

ProGuard : ProGuard, uygulamanın sınıflarının ve üyelerinin isimlerini A, B gibi basit harflerle ve p003io, p002 gibi önemsiz katarlarla kısaltır. Uygulamayı geri derledikten böyle sonra sınıf isimleri, metot isimleri, değişken isimler, veya parametreler görürsek, ProGuard uygulandığını çıkarsayabiliriz.

Gelişmiş Gizleme : Jadx-GUI aracında Araçlar bölümünün altındaki gizleme kaldırma düğmesi seçildikten sonra, büyük bir ölçüde ProGuard gizlemesi açılır. Ancak, gelişmiş gizleyici uygulandığında tersine mühendisi şaşırtmak için çoğunlukla önemsiz, ilgisiz sınıf isimleri, değişkenler, parametreler ve ölü kod eklenir. Bu durumlar varsa uygulamanın gelişmiş gizleme uyguladığı çıkarsanabilir.

Android Keystore : Android Keystore kullanılıp kullanılmadığını öğrenmenin yollarından bir tanesi Objection aracı ile test edilmesidir. Önce Frida aracı kurulup yapılandırılır. Sonrasında Frida'nın desteklediği Objection kurulur, yapılandırılır ve

kullanılır. Mobil kimlik doğrulama uygulaması çalışırken, Objection ile uygulamaya çalışma zamanında bağlanılır ve ilgili komutlar çalıştırılır. Böylece uygulamanın Android Keystore kullanıp kullanmadığı öğrenilir.

Kurcalama Önleme / Yeniden Paketleme : Bir takım smali kodları değiştirilerek yama önleme özelliği test edilir. Uygulama geri derlendikten sonra, smali kodlarında bir takım değişiklikler yapılır. Değişiklikler yapıldıktan sonra, apktool ile uygulama tekrardan derlenir. Uygulama derlenirse kurcalama önleme / yeniden paketleme uygulanmadığı sonucuna varılabilir.

Hassas Verilerin Depolama Tarafında Şifrelenmesi : Gizli tohum değeri depolama tarafında şifreli bir şekilde tutulup tutulmadığı kontrol edilir.

Cihaz Bağlama : Uygulama geri derlendikten sonra, IMEI, MAC adres gibi cihaza özel değerlerin kullanılıp kullanılmadığı incelenir.

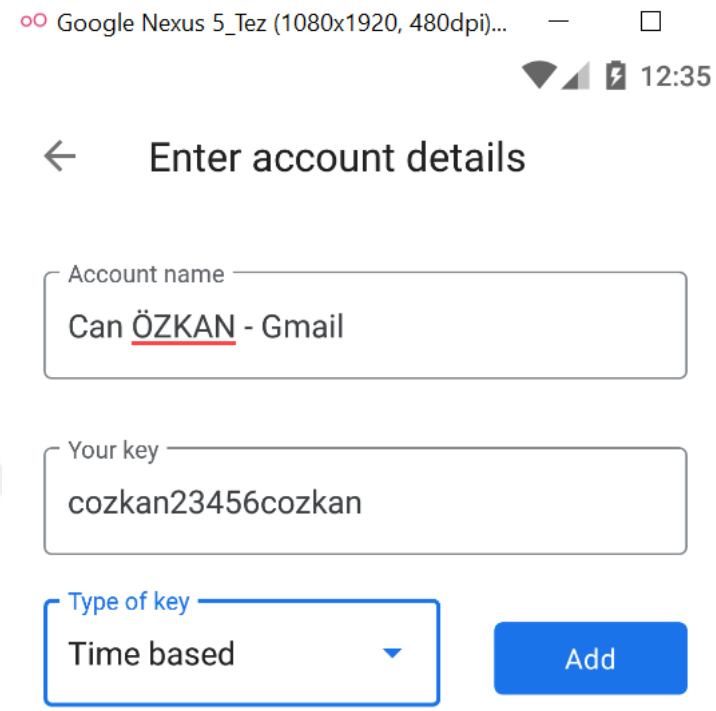
4.3 Simüle Edilmiş Saldırı Adımları

Yukarıda listelenen tüm mobil kimlik doğrulama uygulamalarına yönelik saldırı simüle etmek mümkün değildir. Bu bölümde hem depolama hem de belleğe hem de depolamaya yönelik gizli tohum değerinin elde edilmesine yönelik saldırı simüle edilir.

Tüm tersine mühendislik araçlarını yapılandırdıktan sonra Şekil 4.1 üzerinde gösterilen provizyon süreci tamamlanır, Google Authenticator uygulaması kapatılır, cep telefonu kapatılır, ardından cep telefonu açılır ve Google Authenticator uygulaması açılır. Bu adımlar tamamlandıktan sonra asldırı simülasyonu gerçekleştirilmeye hazır hale gelinir.

4.3.1 Depolamaya yönelik saldırı simülasyonu

Objection ile dinamik analiz sırasında bir veritabanı dosyası ile iletişim kurulduğu gözlemlendi ve ardından statik analiz yapıldı. Statik analiz sonucunda Şekil 4.2 üzerinde görüldüğü üzere databases isminde bir veritabanı dosyası oluşturulduğu gözlemlendi. Dosyanın yoluna gidildi ve veritabanı



Şekil 4.1 : Google Authenticator Provizyon Süreci

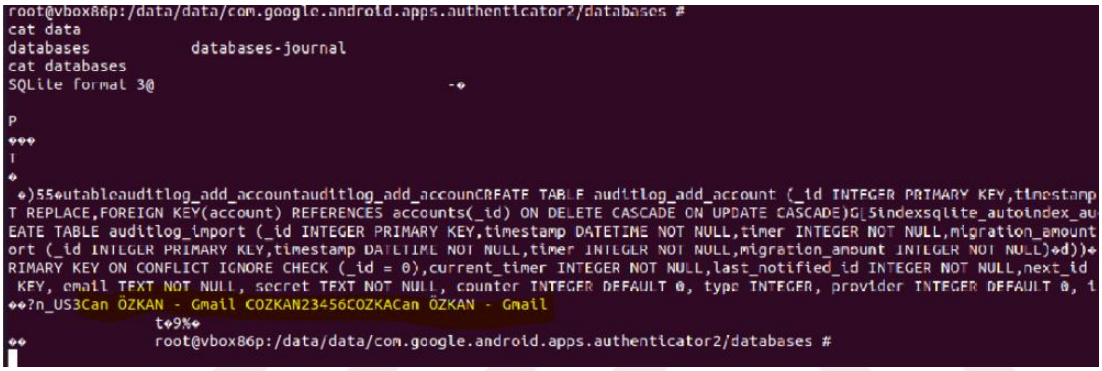
dosyası Şekil 4.3 ve 4.3 üzerinde görüldüğü üzere incelendi. Şifresiz metin halinde gizli tohum değerine ulaşıldı.

Şekil 4.2 : Google Authenticator Statik Tersine Mühendisliği

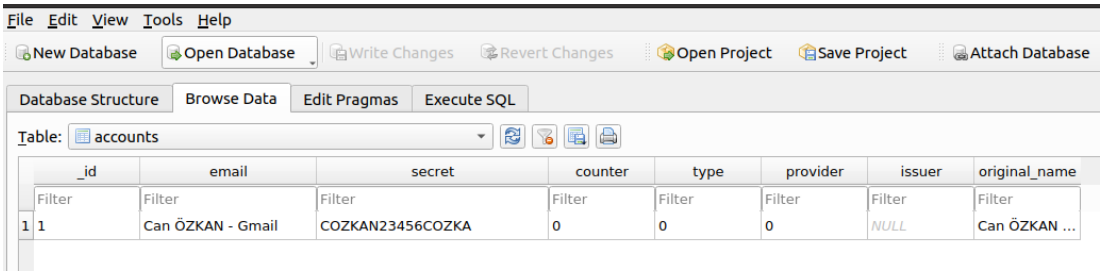
4.3.2 Hafızaya yönelik saldırı simülasyonu

Bu bölümde Google Authenticator uygulamasına hafıza üzerinden gizli tohum değerini şifresiz metin halinde elde edilmesi amacıyla saldırılır. Varsayımlarımızı

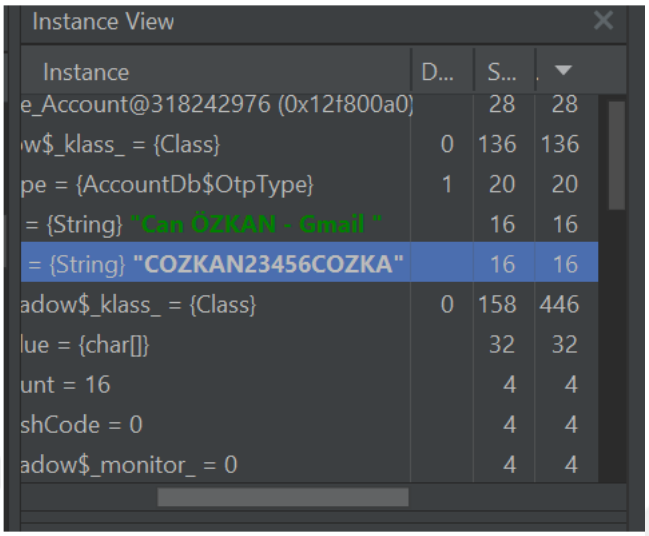
yerine getirdikten sonra tehdit modellememizi uygulamaya hazırız. T zamanında yığın boşaltma işlemi gerçekleştirilir. T ile aktarılmak istenen zamanlama ile ilgili herhangi bir kuralımız olmadığıdır. T rastgele zamanında örnek yığın dökümü alınır. Bir diğer deyişle, T zamanı rastgele seçilir ve yığın dökümleri alınır. Bu konuda belirli bir algoritmamız ve kuralımız yoktur. Bu durumda önemli olan ardışık her saniyenin yığın dökümünün alınamamasıdır. Bir yığın dökümü alındıktan en az 2 – 4 saniye sonra başka bir yığın dökümü alınabilir. T rastgele zamanında bir yığın dökümü alındıktan sonra Şekil 4.5 üzerinde görüldüğü üzere Android Studio Memory Profiler üzerinde incelenir ve Google Authenticator uygulamasının gizli tohum değerini şifresiz metin olarak tuttuğu gözlemlenir.



Şekil 4.3 : Google Authenticator, Depolamadan Gizli Tohum Değerinin Okunması



Şekil 4.4 : Google Authenticator Veritabanının SQLite Tarayacısına Aktarılması



Şekil 4.5 : Google Authenticator, Hafızadan Gizli Tohum Değerinin Okunması

Saldırı modellemimiz uygulayınca Çizelge 4.1 ve Çizelge 4.2 üzerinde görülen sonuçlar elde edilmiştir ve bulgular ilerleyen bölümde tartışılacaktır.

Çizelge 4.1 : Kimlik Doğrulama Uygulamaları ve Uyguladıkları Ek Güvenlik Mekanizmaları

Uygulama	Free OTP	SAP	Sophos	Oracle	Epic	Pixplicity	Blizzard	Authy	Google	Microsoft	SaasPass
ProGuard	×	×	×	√	×	×	×	×	×	×	×
Gelişmiş Gizleyici	×	×	×	×	×	×	×	×	×	×	×
Cihaz Bağlama	×	×	×	×	×	×	×	×	×	×	×
Anti Yama	×	×	×	×	×	×	×	×	×	√	×
Güvensiz Depolama	×	×	√	×	√	×	×	√	√	√	×

Android Keystore	×	×	×	√	×	×	×	√	×	√	×

Çizelge 4.2 : Kimlik Doğrulama Uygulamaları ve Gizli Tohum Değerinin Şifresiz Metin Halinde Elde Edilmesi

Uygulama	Free OTP	SAP	Sophos	Oracle	Epic	Pixplicity	Blizzard	Authy	Google	Microsoft	SaasPass
Depolama	×	×	√	×	√	×	×	√	√	√	×
Hafıza	×	×	√	√	√	×	×	√	√	√	√

5. BULGULAR

Red Hat Free OTP : 1.5 sürümü incelenen Red Hat Free OTP kimlik doğrulayıcısı gizli tohum değerini /data/data/org.fedorahosted.freeotp/shared_prefs dizini altındaki tokens.xml dosyasında kodlanmış formda tutar. T rastgele zamanında alınan yığın dökümünde gizli tohum değerinin yine kodlu bir şekilde tutulduğu gözlenmiştir.

SAP Authenticator : 1.2.8 sürümü incelenen SP Authenticator /data/data/com.sap.csi.authenticator/databases dizini altında DataVault isminde bir veritabanı dosyası barındırır. Bu veritabanı dosyası şifrelidir. Bu nedenle gizli tohum değerine depolamadan erişemedik. Veritabanı dizinine ek olarak, uygulamaya özel veri dizini altındaki diğer dizinler de incelendi. Diğer taraftan, T zamanında gizli tohum değeri hafızadan çekilemedi. Buna ek olarak, bu kimlik doğrulama uygulamasında getIMEI() isimli bir metot bulunmaktadır fakat boş katar döndürmektedir. Bu metot uygulamada hiç bir zaman çağrılmamaktadır. Yani ölü kod olarak uygulamada bulunmaktadır.

Sophos Authenticator : 3.2 sürümü incelenen Sophos Authenticator gizli tohum değerini /data/data/com.sophos.sophtoken/databases dizini altında bulunan databases dosyasında şifresiz metin olarak tutar. Ayrıca, T rastgele zamanında gizli tohum değerine hafızada şifresiz metin olarak tuttuğu gözlenmiştir.

Oracle Authenticator : 9.2 sürümü incelenen Oracle Authenticator gizli tohum değerini /data/data/oracle.idm.mobile.authenticator/databases dizini altında bulunan OMADb.db veritabanı dosyasında şifreli metin olarak tuttuğu gözlenmiştir. Kullanıcının eposta bilgisini açık olarak tutmasına ragmen gizli tohum değeri şifreli metin olarak tutulmaktadır. Ayrıca, T rastgele zamanında gizli tohum değerine hafızada şifresiz metin olarak tuttuğu gözlenmiştir. OAMAccount sınıfı gizli tohum değerini hafızada şifresiz metin olarak tutmaktadır. Bu kimlik doğrulama uygulaması kurulu olduğu mobil cihazın MAC adresi, IMEI numarası gibi değerleri okur ve uyumlulukla alakalı bir takım işlemlerden geçirir.

Epic Authenticator : Epic Authenticator uygulaması gizli tohum değerini /data/data/com.epic.authenticator/shared_prefs dizini altında bulunan com.epic.otpkit.keystore.OtpAccountStore_ACCOUNTS.xml dosyasında şifresiz metin olarak tuttuğu gözlenmiştir. Ayrıca, T rastgele zamanında gizli tohum değerine hafızada şifresiz metin olarak tuttuğu gözlenmiştir.

Pixplicity Authenticator : 1.0.4 sürümü analiz edilen Pixplicity Authenticator gizli tohum değerini depolama tarafında şifresiz metin olarak tutmaz. Gizli tohum değerinin /data/data/com.pixplicity.auth/databases dizini altında bulunan tokens.db veritabanı dosyasında BLOB tipinde saklandığı gözlenmiştir. T zamanında alınan yığın dökümü incelendiğinde gizli tohum değerinin hafızada byte array'lerine ayrıldığı ve değeri şifresiz metin olarak tutmadığı gözlenmiştir.

Blizzard Authenticator : 2.4.2.9 sürümü incelenen Blizzard Authenticator gizli tohum değerini depolama tarafında şifresiz metin olarak tutmaz. Bu kimlik doğrulama uygulaması IMEI numarası, android kimliği ve cep telefonu modeli bilgisini okur ve katar birleştirmesi işlemi uyguladıktan sonra özet değerini alır. Hesabınına giriş yapmaya çalışan kullanıcının o an telefonun değiştirip değiştirmediğini kontrol eder. Eğer mobil cihaz değişmişse, kullanıcı geri yükleme sayfasına yönlendirilir. Cihaza özel bu değerler OTP üretimindeki algoritmaya rastgelelik eklemese de hesap güvenliği için kullanılır.

Twilio Authy Authenticator : 24.3.4 sürümü incelenen Twilio Authy Authenticator gizli tohum değerini /data/data/com.authy.authy/shared_prefs dizini altında bulunan com.authy.storage.tokens.authenticator.xml dosyasında şifresiz metin olarak saklar. Ayrıca, T rastgele zamanında gizli tohum değerine hafızada şifresiz metin olarak tuttuğu gözlenmiştir. OtpAuthPayload sınıfı gizli tohum değerini hafızada şifresiz metin olarak tutmaktadır.

Google Authenticator : 5.10 sürümü incelenen Google Authenticator gizli tohum değerini /data/data/com.google.android.apps.authenticator2/databases dizini altında bulunan databases dosyasında şifresiz metin olarak tutar. Ayrıca, T rastgele zamanında gizli tohum değerine hafızada şifresiz metin olarak tuttuğu gözlenmiştir. AutoValue_Account sınıfı gizli tohum değerini hafızada şifresiz metin olarak tutmaktadır.

Microsoft Authenticator : 6.2006.4198 sürümü incelenen Microsoft Authenticator gizli tohum değerini /data/data/com.azure.authenticator/databases dizini altında bulunan PhoneFactor-wal dosyasında şifresiz metin olarak tutar. Ayrıca, T rastgele zamanında gizli tohum değerine hafızada şifresiz metin olarak tuttuğu gözlenmiştir. SecretKeyBasedAccount sınıfı gizli tohum değerini hafızada şifresiz metin olarak tutmaktadır.

SaasPass Authenticator : 24.3.4 sürümü incelenen SaasPass Authenticator gizli tohum değerini depolama tarafında şifresiz metin olarak tutmaz. Fakat, T rastgele zamanında gizli tohum değerine hafızada şifresiz metin olarak tuttuğu gözlenmiştir. OTPAuthBarcode sınıfı gizli tohum değerini hafızada şifresiz metin olarak tutmaktadır.

Tablo 4.1 ve Çizelge 4.2' de görüldüğü üzere, 11 tane mobil kimlik doğrulama uygulamasından 5 tanesi gizli tohum değerini depolamada şifresiz metin olarak saklar. Diğer bir deyişle, 5 tane mobil kimlik doğrulama uygulaması depolama tarafında gizli tohum değerini saklarken herhangi bir şifreleme algoritması, özet fonksiyonu vb. uygulamamıştır. 11 uygulamadan 7 tanesi gizli tohum değerini T rastgele zamanında bellekte şifresiz metin olarak tutmaktadır. Bu da uygulamaların yarısından fazlasının gizli tohum değerini hafıza şifresiz metin olarak tuttuğunu göstermektedir. Ayrıca, Blizzard Authenticator uygulaması kullanıcılarını mobil cihazın rootlu olmasından dolayı doğabilecek güvenlik risklerine karşın uyarmaktadır.

Microsoft Auhtenticator uygulaması dışındaki uygulamaların hiçbiri kurcalamaya karşın önlem almamıştır. Uygulamaları başarıyla tersine yeniden paketleyebildik. Sonuç olarak saldırganlar uygulamaları geri derleyip, kötücül amaçlı kod yerleştirip, yeniden paketleyip, uygulamayı imzaladıktan sonra kötü bir amaçla uygulamaları sosyal mühendislik yöntemlerinden faydalanarak dağıtabilirler veya root algılama gibi güvenlik mekanizmalarını kaldırarak uygulamayı yeniden paketleyebilirler.

Dahası, yetenekli tersine mühendislerden kaçınmak için hiçbir uygulamanın gelişmiş gizleyici uygulamadığını unutmamak önemlidir. Doğası gereği OTP üretim aşamasına ilişkin herhangi bir cihaz bağlama durumu yoktur. Ancak Blizzard Authenticator OTP değerini değil, kullanıcının hesabını güvenceye almak için MAC adresini, IMEI numarasını ve telefon modelini okur. Test edilen bütün kimlik doğrulama uygulamaları TOTP veya HOTP protokollerini uyguladığı gözlemlendi.

Android Keystore şifreleme anahtarlarının güvenliğini sağlamak için fiili bir yöntem olsa da, 11 kimlik doğrulama uygulamasının sadece 3 tanesinin Android Keystore uyguladığı gözlenmiştir.

6. TARTIŞMA

Android Keystore şifreleme anahtarlarını güvenli bir şekilde saklamak için kullanılır. Android Keystore bir anahtar oluşturur ve ardından bu anahtarı o anahtarı güvenli bir şekilde saklar. Uygulamalar bu anahtarla hassas bilgilerini, bizim araştırmamız için gizli tohum değerini şifrelemek için kullanabilirler. Twilio Authy Authenticator uygulaması Android Keystore kullansa da gizli tohum değerini paylaşılan tercihler altında açık metin olarak saklamaktadır. Bu durum Android Keystore konusunun hatalı kullanıldığı veya yedek alma konusunda oluşabilmesi muhtemel bir takım durumları önlemek için olabilir. Öte yandan Oracle Authenticator uygulamasının AndroidKeystore kullandığı ve gizli tohum değerini depolamada açık metin olarak saklamadığı gözlenmiştir. Aklımıza gelen ilk durum yedekleme durumu oldu. Şifreleme anahtarlarından dolayı bir uygulama Android Keystore kullanırsa düzgün bir şekilde yedekleme özelliği uygulamayabilir. Çünkü Microsoft ve Twilio Authy Android Keystore kullanmasına ragmen gizli tohum değerini şifresiz metin olarak tutmaktadır ve yedeklemeye izin vermektedir. Başka bir deyişle, Microsoft ve Twilio authy uygulaması gizli tohum değerlerini başka bir cihaza aktarmaya olanak tanır.

İncelenen bütün mobil kimlik doğrulama uygulamalarının TOTP ve HOTP algoritmalarını kullanarak tek kullanımlık parola ürettikleri gözlemlenmiştir. Bu algoritmalar MAC adresi IMEI numarası gibi cihaza özel değerler gerektirmez. Böylece kullanıcılar mobil cihazlarını kaybettiklerinde veya yeni bir cihaz satın aldıklarında hesaplarının başka cihazlara aktarılmasını daha kompakt ve taşınabilir hale getirir. Bir uygulama cihaza özel benzersiz değerler isterse, OTP hesaplarının başka bir cihaza aktarımı ve yedeklenmesi daha zor oalcaktır.

Philip Polleit ve Michael Spreştzenbarth'ın çalışması gizli tohum değerinin depolamadan şifresiz metin olarak çekilmesini kapsar fakat sadece bununla da sınırlı değildir. Buna ek olarak, biz gizli tohum değerinin hafızadan da şifresiz metin olarak elde edilebileceğini gösterdik. Çalışmalarında mobil kimlik doğrulama uygulamalarının klanlanıp klonlanmadığı, cihaz bütünlüğü kontrolü, PIN koruması, güvenli SSL bağlantısı üzerine yogunlaştılar. Paylaşılan tohum değerine ağ üzerinden de saldırı düzenlenebilir. Paylaşılan tohum değeri şifrelenmemiş bir kanal üzerinden taşınırsa yine şifresiz metin olarak elde edilebilir. Ancak bizim tehdit modellememiz bunu kapsamaz. Nitekim paylaşılan gizli tohum değerini elde etmek

için statik ve dinamik tersine mühendislik tekniklerinin kullanımını vurgulaarız. Bunlara ek olarak, ProGuard, gelişmiş gizleyici kullanımı gibi geliştiricilerin ek güvenlik mekanizmaları ekleyip eklemediklerini inceledik. Çünkü başta ProGuard kullanımı ve gelişmiş gizleyiciler olmak üzere, ek güvenlik mekanizmaları tersine mühendislerin uygulama ve kod üzerinde inceleme yapmasını zorlaştırır. Ayrıca, Android Keystore şifreleme anahtarlarını güvenli bir şekilde saklamanın en güvenli yollarından bir tanesidir. Böylece saldırganlar ve tersine mühendisler bu kriptografik anahtarlara erişemezler. Yaptığımız incelemeler işin tersine mühendis yönüne odaklanır.

7. SONUÇLARIN ETKİLERİ

Çoğu durumda gizli tohum değerini bellekten elde etmek depolamadan elde etmeye göre daha kolaydır. Gizli tohum değerini depolamada korumak için Keystore içerisinde saklamak ve beyaz kutu şifrelemesi gibi tekniklerden faydalanılır. Öte yandan, belirsizlikle güvenlik yöntemi ile gizli tohum değeri hafızada şifresiz metin olarak tutulmasına nazaran daha güvenli hale getirilebilir. Belirsizlik yoluyla güvenlik mükemmel bir çözüm olmasa da tersine mühendislerini işini zorlaştıracaktır. Uygulamaların ProGuard ve gelişmiş gizleyici uygulamadıkları durumlarda tersine mühendisler uygulamayı neredeyse original kaynak koduna kadar çevirebilmektedir.

Kimlik doğrulama uygulamaları sadece Android işletim sistemi üzerinde hali hazırda gelen güvenlik korumalarına güvenmemelidir. Uygulamalar ek güvenlik önlemleri almalıdır. Çünkü Android işletim sistemi ile birlikte hali hazırdan gelen güvenlik korumaları atlatıldığında saldırganın ve tersine mühendisin önünde hiç bir güvenlik mekanizması kalmayacaktır. Ek olarak, uygulamalar yazılımın kurcalanmasına ve yamanmasına yönelik ek önlemler almalıdır. Çünkü uygulamalar yeniden paketlenebilir, bir meterpreter tersine kabuğu alacak kod parçacığı eklenebilir, uygulamayı yeniden imzalayıp sosyal mühendislik yöntemleri ile kullanıcılara dağıtılabilir. Kullanıcılar yeniden derlenmiş uygulamayı yüklediklerinde saldırganlara kabuki bağlantısı yapıp, saldırganlar da saldırılarını başlatabilirler. Öte yandan, anti yamama özelliği olmazsa uygulamaların güvenlik seviyelerini artırmak için ekledikleri root algılama yöntemleri atlatılması konusunda da kullanılabilir. Kök tespit yöntemleri silinebilir, uygulama yeniden paketlenip incelemeye devam edilebilir.

8. SONUÇ ve ÖNERİLER

İlk olarak hesap güvenliğini artırmak için ikili kimlik doğrulama uygulanması son derece önemlidir. Çünkü tek faktörlü kimlik doğrulamaya kıyasla güvenlik duruşunu iyileştirmek için ek bir unsur daha ekler.

Çizelge 4.2'de üzerinden çıkarsama yapılacağı üzere çoğu durumda gizli tohum değerini bellekten elde etmek depolamadan elde etmeye göre daha kolaydır. Android Keystore ve beyaz kutu şifrelemesi kullanılmasının gizli kök değerini depolama tarafında korumak için yeterlidir. Fakat bu teknolojiler yeterli olmasına karşın bu teknolojilerin düzgün ve en iyi örneklemelere uygun kullanılması gerekmektedir.

Çok az uygulamanın ek güvenlik mekanizması uygulandığı gözlenmiştir. Bir çok uygulama Android işletim sisteminin varsayılan olarak uyguladığı güvenlik mekanizmasına güvenmektedir. Bu neden root işlemi yapılmış cihazlar, istismar edilebilir zafiyetlere sahip cihazlar ve düşük güvenlik bilincine sahip kullanıcıların gizli tohum değerleri saldırganlar tarafından ele geçirilebilir ve ikinci unsur olarak kullanılabilen tek kullanımlık parola unsuru atlatılabilir.

Gelecekteki çalışmalar ardışık her bir saniyenin yığın dökümünü alan bir araç geliştirmeyi içerebilir.

Mobil kimlik doğrulama uygulaması veya herhangi bir Android işletim sistemi üzerinde çalışacak uygulama geliştirirken, uygulamanın tersine mühendislik yöntemleri ile incelenmesini zorlaştırmak için aşağıda verilen Çizelge 8.1 üzerindeki maddelerden faydalanılması önerilir.

Çizelge 8.1 : Ek Güvenlik Önlemleri ve Avantaj/Dezavantajları

Uygulama	Avantajları	Dezavantajları
ProGuard	Uygulamanın boyutunu düşürür. Verimlilik ve performans artar.	Yok.
Gelişmiş Gizleyici	Statik kod incelemesinde tersine mühendisin işini bir hayli zorlaştırır.	Gelişmiş gizleyici uygulamalar ücretli olabilir ve uygulamanın maliyetini artırabilir.
Cihaz Bağlama	Kriptografik algoritmalar için rastgeleliği artırır.	Cihaza özel değerlerin bir başka cihaza taşınması problem vardır.
Anti Yama Özelliği	Cihazların yamanmasını engeller.	Ek kod parçası ekleneceğinden, uygulamanın performansı negative olarak etkilenebilir.
Android Keystore	Kriptografik anahtarları saklamanın en güvenli yollarından bir tanesidir.	Yok.

KAYNAKLAR

[1] **Sheetal, J.U., Prohit, D.N., Anup, V.,** (2019). Increase in Number of Online Services and Payments through Mobile Applications Post Demonetization

[2] **Url-1** <https://www.theguardian.com/business/2016/jul/22/mobile-banking-on-rise-as-payment-via-apps-soars-by-54-in-2015>, alındığı tarih : 12.10.2020

[3] **Url-2** <https://authy.com/what-is-2fa/> , alındığı tarih : 12.10.2020

[4] **Park, W.S., Hwang, D.Y., Kim, K.H.,** (2018). A TOTP Based Two Factor Authentication Scheme for Hyperledger Fabric Blockchain

[5] **Hasbullah, H.B., Lawal, I.A., Mu'azu A.A.,** (2013). A TOTP-Based Enhanced Route Optimization Procedure for Mobile IPv6 to Reduce Handover Delay and Signalling Overhead

[6] Huseynov, E., Seigneur, J.M., (2019). Hardware TOTP Tokens with Time Synchronization

[7] **Khan, B.U.I., Olanrewaju, R.F., Anwar, F., Yaacob, M.,** (2018). Offline TOTP Based Solution for Secure Internet Banking Access

[8] **Polleit, P., Spreitzenbarth, M.,** (2018). Defeating the Secrets of OTP Apps

[9] **Mueller, B.,** (2016)., Hacking Soft Tokens – Advanced Reverse Engineering on Android

[10] **Poonguzhali, P.P., Dhanokar, M.K., Chaithanya, Patil, M.U.,** (2016). Secure Storage of Data on Android Based Devices

[11] **Cooijmans, T., Ruiter, J.D., Poll, E.,** (2014). Analysis of Secure Key Storage Solutions on Android

[12] **Cooijmans, T.,** (2014). Secure Key Storage and Secure Computation in Android, 15-51

[13] **Kasagiannis, G., Dadoyan, C.,** (2014). Security Evaluation of Android Keystore, 21-29

[14] **Konoth, R.K., Fischer, B., Fokkink, W., Athanasopoulps, E., Razavi, K., Bos, H.,** (2020). SecurePay: Strengthening Two-Factor Authentication for Arbitrary Transactions

[15] **Permana, I., Hardjianto, M., Baihaqi, K.A.,** (2020). Securing the Website Login System with the SHA256 Generating Method and Time-based One-time Password (TOTP)

[16] **Zhao, J.X.,** (2013). Research and Design on an Improved TOTP Authentication

[17] **Azhari, Muhammad, L.Q., Tama, C.G.N.,** (2016). Android Mobile Banking Application Security from Reverse Engineering and Network Sniffing, 462-465

[18] **Kotipalli, S.R., Imran, M.A.,** (2016). Hacking Android, 87-101

[19] **Url-3** < https://source.android.com/security/overview/kernel-security> , alındığı tarih : 12.10.2020

[20] **Granthi, P.K., Bansode, S.M.,** (2016). Android Security: A Survey of Security Issues and Defenses, 543-544

[21] **Url-4 <** https://stuff.mit.edu/afs/sipb/project/android/docs/tools/publishing/app-signing.html> **,** alındığı tarih : 12.10.2020

[22] **Url-5** < https://developer.android.com/studio/build/shrink-code> , alındığı tarih : 12.10.2020

[23] **Url-6** < https://developer.android.com/training/articles/keystore#java> , alındığı tarih : 12.10.2020

[24] **Buttan, D.,** (2020). Hacking the Human Brain: Impact of Cybercriminals Evoking Emotion for Financial Profit, 19-20

[25] **Chaudhari, A.S.,** (2020). Security Analysis of SMS and Related Technologies, 34

[26] **Eilam, E.,** (2011). Reversing: Secrets of Reverse Engineering, 3-12

[27] **Url-7** < https://ragingrock.com/AndroidAppRE/> , alındığı tarih : 12.10.2020

[28] **Url-8** <https://www.nowsecure.com/blog/2017/11/14/oneplus-device-root-exploit-backdoor-engineermode-app-diagnostics-mode/>, alındığı tarih : 12.10.2020

[29] **Url-9** <https://www.androidpolice.com/2017/11/15/oneplus-left-backdoor-devices-capable-root-access/> , alındığı tarih : 12.10.2020

[30] **Url-10** <https://www.trendmicro.com/en%20us/research/17/i/%20zniu-first-android-malware-exploit-dirty-cow-vulnerability.html> , alındığı tarih : 12.10.2020

[31] **Url-11** <https://arstechnica.com/information-technology/2019/10/attackers-exploit-0day-vulnerability-that-gives-full-control-of-android-phones/> alındığı tarih : 12.10.2020

[32] **Url-12** < https://www.trendmicro.com/en_us/research/20/a/first-active-attack-exploiting-cve-2019-2215-found-on-google-play-linked-to-sidewinder-apt-group.html> , alındığı tarih : 12.10.2020

[33] **Davi, L., Dmitrienko, A., Sadeghi A.R., Winandy M.,** (2010). Privilege Escalation Attacks on Android, 5-6

ÖZGEÇMİŞ

Ad-Soyad : **Can ÖZKAN**
Uyruğu : **Türkiye Cumhuriyeti**
Doğum Tarihi ve Yeri : **1994, KEMALPAŞA/İZMİR**
E-posta : **c.ozkan@etu.edu.tr**

ÖĞRENİM DURUMU:

- **Lisans** : 2018, MCBÜ, Mühendislik Fakültesi, Bilgisayar Mühendisliği
- **Yüksek Lisans** : 2020, TOBB ETÜ, Bilgisayar Mühendisliği Anabilim Dalı, Bilgi Güvenliği

MESLEKİ DENEYİM VE ÖDÜLLER:

Yıl	Yer	Görev
2018 -	Roketsan A.Ş	Siber Güvenlik Mühendisi
2018-2018	AKBANK	Siber Güvenlik Analisti

YABANCI DİL: İngilizce (IELTS, 6.0)

TEZDEN TÜRETİLEN YAYINLAR, SUNUMLAR VE PATENTLER:

- **Özkan, C**., (2020). Security Analysis of Mobile Authenticator Applications
- **Özkan, C**., (2020). Security Analysis of Mobile Authenticator Applications: International Conference on Information Security and Technology, December 3-4, Ankara / TURKEY.