Bil 553
İnternet ve Veri Güvenliği

Mustafa Simav
091101036

Burak Dikili
091101062

Yaz 2013

Bölüm 1

Giriş

İnternet ve Veri Güvenliği dersi kapsamında, kullanıcıların kayıt olup giriş yapacakları, şifre değiştirme, şifre sıfırlama, kullanıcı hesabı silme gibi temel fonksiyonları olan bir sistemi güvenli bir şekilde tasarlamamız ve implement etmemiz gerekmektedir.

Bu raporda, proje kapsamında uyguladığımız güvenlik prensiplerinden bahsedip, bu prensipleri nasıl implement ettiğimizin detaylarını anlatacağız.

1.1 Yapı

Bu proje kapsamında gerçekleştireceğimiz uygulama bir web uygulaması olacaktır. Kullanıcılar internet tarayıcılarını kullanarak sistem ile etkileşim içinde olacaklardır. Değerli bilgiler, güvenli olmayan internet kanalı ile sunucu ve istemci arasında gidecektir. Bu nedenle güvenliğini sağlamamız gereken 3 yer vardır.

1. Sunucu güvenliği
2. İstemci güvenliği
3. Haberleşme kanalı güvenliği

1.2 Kullanılan Araçlar ve Kütüphaneler

Bu sistemi yaparken kullanacağımız pek çok araç vardır. İlk olarak geliştireceğimiz uygulamayı *Python* dili ile *flask* framework'ünü kullanarak yazacağız. Ayrıca kullanıcı arayüzü için *HTML* ve *JavaScript* kullanacağız. Web uygulamamızı çalıştırabilmek için, web sunucu uygulaması olarak da *uwsgi* ve *nginx* kullanacağız. SSL desteğini de *nginx* sayesinde kazanacağız. Bütün bu uygulamaların üzerinde çalıştığı işletim sistemi olarak da *Debian* tercih ettik. Kullanıcı bilgilerini tutacağımız veri tabanı olarak da *PostgreSQL*'i seçtik. Python ve PostgreSQL arasındaki iletişim için *python-psycopg2* kütüphanesini, kripto araçları için *pycrypto* kütüphanesini, kullanıcının IP adresine göre konumunu almak için *pygeoip* kütüphanesini kullanıyoruz.

Kullandığımız bütün bu araçları liste halinde sunmamız gerekirse:

1. Debian 7 Wheezy 64 bit
2. python 2.7.3
3. uwsgi 1.9.13
4. flask 0.10

5. python-psycopg2 2.5.1
6. PostgreSQL 9.1
7. nginx 1.2.1
8. pycrypto 2.6
9. pygeoip 0.2.7

Bölüm 2

Temel Güvenlik

2.1 Hazır Kripto Kütüphanesi Kullanmak

Kripto araçlarının tanımının açıkça yapılmış olması, bu araçların güvenliği için olmazsa olmaz şartlardan biridir. Auguste Kerckhoffs, "Bir kripto sistemi, sistem hakkındaki anahtar hariç her detay açıkça bilindiği halde bile güvenli olmalıdır."[1] diyerek *Kerckhoffs İlkesi* olarak bilinen ilkeyi ortaya atmıştır.

Fakat tanımı açık olsa da kripto araçlarını deneyimsiz insanların implement etmesi çeşitli güvenlik zaaflarına neden olmaktadır. Örnek vermek gerekirse, iki hash değeri karşılaştırılırken, fark görüldüğü anda algoritmanın sonlanması, zamanlama saldırılarına temel hazırlamaktadır.

Zamanlama saldırılarının ciddiyeti konusundaki en önemli örnek Java'nın standart kütüphanesinde yer alan `java.security.MessageDigest` sınıfının `isEqual` methodudur. Bu method implementasyonundaki nedeniyle zamanlama saldırılarına karşı zaafiyeti keşfedilmiştir. Bu zaafiyet Java SE 6 Update 17 güncellemesi ile kapatılmıştır [2].

Bu nedenlerden dolayı kripto ihtiyaçlarımızı, deneyimli insanlar tarafından implement edilmiş ve iyi test edilmiş kütüphaneler ile sağlamamız tavsiye edilmektedir. Biz de bu nedenle *pycrypto* kütüphanesini kullanmaya karar verdik.

2.2 Rastgele Sayı Üreticisinin Kriptografik Güvenli Olması

Rastgele sayı üretmek, şifre sıfırlama gibi işlemler için gerekli bir fonksiyondur. Şifre sıfırlama işteğinin, gerçekten hesabın sahibi kullanıcıdan geldiğinden emin olmak için, güvenli bir kaynaktan (email gibi) doğrulama gerekmektedir. Bunun için de rastgele bir doğrulama kodu üreterek, kullanıcıya mail atıyoruz.

Fakat bir saldırgan, başka bir hesap için şifre sıfırlama isteği açabilir ve üretilen rastgele sayıyı da tahmin ederek o hesabı ele geçirebilir.

Bu saldırıya karşı, kriptografik olarak güvenli (tahmin edilemez) rastgele sayı üretici kullanmak koruma sağlayacaktır. Biz de *pycrypto* kütüphanesi içindeki *Random* sınıfını kullanarak rastgele sayı ürettik.

Bölüm 3

Sunucu Tarafı Güvenliği

3.1 İşletim Sistemi Güvenliği

Kullanıcı bilgilerinin güvenliğinin sağlanmasının ilk adımı, uygulamanın çalışacağı sunucu üzerinde güvenlik önlemleri almaktır. Bu önlemlerin ilk adımı da sunucu üzerinde çalışan işletim sisteminin güvenliğidir.

Biz işletim sistemi olarak *Debian 7 Wheezy 64 Bit* kullanmayı seçtik. Aşağıdaki maddeler ile debian işletim sistemini nasıl daha güvenli yapabileceğimiz tartışılmaktadır.

3.1.1 Güvenilir Kaynaklardan Yazılım Yüklemek

GNU/Linux sistemler gerek kernel'inin yapısı gerekse üzerinde çalışan programların güvenliği temel ilke kabul etmelerinden dolayı güvenlidir [3]. Sistemin güvenliği önündeki en büyük engel bilinmeyen kaynaklardan yazılım yüklemektir.

Biz yüklediğimiz bütün yazılımları *Debian depolarından* yükleyerek bu sorunun önüne geçtik. Ayrıca debian depolarındaki yazılımlar güvenlik yamaları da yapılmış olduğu için işletim sisteminin yazılım bazında güvenliğini sağladık.

3.1.2 Uzaktan Erişimin Güvenliğini Arttırmak

Sunucu bilgisayara uzaktan erişim, oluşan hataların tespitini ve çözümünü kolaylaştıran çok önemli bir özelliktir. Debian kurulduğunda, şifrelenmiş bir bağlantı üzerinden uzaktan erişime imkan veren *openssh* programı ile beraber gelir. Bu program güvenli iletişimi sağlar fakat saldırganın şifreyi kaba kuvvet ile deneyerek bulma imkanı vardır.

Bruteforce saldırılarını önlemek için yapılabilecek pek çok şey vardır [4]. Bunlardan en önemli ikisi şunlardır.

1. root kullanıcısının login özelliğini kapatmak
2. şifre ile login özelliğini kapatıp, sadece açık anahtar ile login olmak

Biz bu iki yöntemi de uyguladık. Bunu yapmak için önce kendi ssh açık anahtarımızı sunucuya yerleştirdik ve ardından `/etc/ssh/sshd_config` dosyasını aşağıdaki gibi değiştirdik.

Kaynak Kod 3.1: OpenSSH Ayarları

```
PermitRootLogin no
PasswordAuthentication no
PubkeyAuthentication yes
```

Bu yöntemlerden root kullanıcısının login özelliğini kısıtlamanın hiç bir sakıncası yoktur. Şifre ile oturum açmayı iptal etmeknin sakıncaları ile gizli anahtarımızın olmadığı bilgisayarlardan oturum açamama ve gizli anahtarı kaybetme durumunda saldırganın sunucuya erişim sağlamasıdır. Fakat biz güvenlik konusunda bilgili olduğumuz ve gizli anahtarlarımızın güvenliğini sağladığımız için bu bir sorun olmayacaktır.

3.1.3 Güvenlik Duvarı

Linux çekirdeği güçlü bir firewall olan *iptables* ile birlikte gelmektedir. Basitçe ayarlandıktan sonra güçlü bir koruma sağlamaktadır.

Kaynak Kod 3.2: IPtables Ayarları

```
*filter
:INPUT DROP [0:0]
:FORWARD DROP [0:0]
:OUTPUT ACCEPT [748:52299]
:TCP - [0:0]
:UDP - [0:0]
-A INPUT -m conntrack --ctstate RELATED,ESTABLISHED -j ACCEPT
-A INPUT -i lo -j ACCEPT
-A INPUT -m conntrack --ctstate INVALID -j DROP
-A TCP -p tcp --dport 80 -j ACCEPT
-A TCP -p tcp --dport 443 -j ACCEPT
-A TCP -p tcp --dport 22 -j ACCEPT
-A INPUT -p icmp -m icmp --icmp-type 8 -m conntrack --ctstate NEW -j ACCEPT
-A INPUT -p udp -m conntrack --ctstate NEW -j UDP
-A INPUT -p tcp -m tcp --tcp-flags FIN,SYN,RST,ACK SYN -m conntrack
                              --ctstate NEW -j TCP
-A INPUT -p udp -j REJECT --reject-with icmp-port-unreachable
-A INPUT -p tcp -j REJECT --reject-with tcp-reset
-A INPUT -j REJECT --reject-with icmp-proto-unreachable
COMMIT
```

Yukarıdaki gibi ayarlandığında iptables sadece 80, 443 ve 22 numaralı portlardan gelen istekleri kabul edecektir. Diğer bütün istekler *iptables* tarafından düşürülecektir.

3.2 Veri Tabanı Güvenliği

Veritabanı, kullanıcı bilgilerinin saklandığı yer olduğu için saldırganlar veritabanına da saldıracaktır. Bu nedenle veritabanın güvenliğinin de sağlanması gerekmektedir. PostgreSQL veritabanı daha güvenli hale getirmek için yapılabilecek bir kaç küçük işlem vardır.

3.2.1 Veri Tabanına Erişimi Kısıtlamak

Veritabanına erişimin kısıtlanması, sunucuya sızmış olan saldırganın işlerini zorlaştıracaktır. Bu nedenle yapılması gereklidir.

Biz vertabanı olarak *PostgreSQL* kullanıyoruz. *PostgreSQL*'de kullanıcı oluşturmak için öncelikle *postgres* kullanıcısı olarak oturum açmamız gereklidir. Bu nedenle `su - postgres` komutunu çalıştırılmalı ve ardından kullanıcı ve veritabanı eklenmelidir.

Kaynak Kod 3.3: PostgreSQL Kullanıcı ve Verıtabanı Ekleme

```
$ createuser guvenlik
$ createdb -O guvenlik guvenlik
```

Bu komutlar guvenlik isminde bir veritabanı ve bu veritabanını kullanma hakkı olan bir kullanıcı oluşturur. PostgreSQL'in varsayılan ayarlarına göre sadece *guvenlik* ismindeki sistem kullanıcısı bu veritabanına erişebilir.

3.2.2 SQL Injection Saldırılarına Karşı Önlem Almak

Veritabanı üzerindeki diğer bir güvenlik tehtidi de *SQL Injection* yöntemidir. Bu yöntem, kötü programcıların SQL ifadelerini string işlemleri ile oluşturmalarından dolayı oluşan bir açıktır.

Bu hatalara örnek, SQL ifadelerinin python string format operatörü olan % işareti ile oluşturmaktır.

Kaynak Kod 3.4: Python-psycopg2 SQL Injection

```
SQL = "SELECT * FROM users WHERE mail = %s AND passwd = %s" %
                (mail, passwd)
cursor.execute(SQL)
```

Bu hatalardan korunmanın yöntemi ise argümanları `execute` fonksiyonuna parametre olarak vermektir.

Kaynak Kod 3.5: Python-psycopg2 SQL Injection

```
SQL = "SELECT * FROM users WHERE mail = %s AND passwd = %s"
cursor.execute(SQL, (mail, passwd))
```

Biz de programın kodlaması sırasında SQL ifadelerimizi bu prensibe uygun oluşturduk.

3.3 Uygulamanın Güvenliği

Sunucu bilgisayarın ve veritabanının güvenliğini sağladıktan sonra, geliştirdiğimiz uygulama içinde de güvenlik önlemleri almamız gerekmektedir.

3.3.1 Şifrelerin Açık Bir Şekilde Saklanmaması

Şifrelerin veritabanında açıkça saklanması, veritabanının ele gecirilmesi durumunda kullanıcılar açısından sorun oluşturabilir. Bu nedenle şiflerelerin gizli bir şekilde saklanması daha uygundur.

Bu işlem için de kripto bize *özet (hash) fonksiyonları* sunar. Hash fonksiyonu, $H : X \to Y$ üzerinde tanımlı, bire bir eşleme yapan, deterministik bir fonksiyondur. Yapısı gereği her şifre stringini bir hash değerine eşler. Bu nedenle şifre bilindiğinde, şifre kontrolü mümkün iken, hash bilindiğinde şifreye ulaşmak imkansızdır.

Biz de bu projede *SHA256* algoritmasını kullanarak şifreleri saklıyor. *SHA256*'yı şu anda kullanılması tavsiye edilen algoritma olduğu için seçtik.

Kaynak Kod 3.6: Python SHA256 Kullanımı

```
hash = SHA256.new()
hash.update(passwd)
message = hash.hexdigest()
```

3.3.2 Hash İşlemi Sırasında Salt Kullanılması

Şifrelerin sadece hash olarak saklanması yöntemi, çevrimdışı bir saldırı olan *Rainbow Saldırılarına* karşı zaafiyet içermektedir. Bu nedenle şifrenin başına rastgele üretilen bir *tuz (salt)* değeri koyularak, saldırganın elindeki önceden hesaplanmış tablo geçersiz hale getirilir.

Unix işletim sistemlerinde şifreler saklanırken 128 bitlik bir salt değeri kullanılmaktadır [5]. Biz de bu nedenle *pycryto* kütüphanesi içindeki *kriptografik rastgele sayı üretici* ile 128 bitlik sayı üreterek bunu salt olarak kullanıyoruz.

Kaynak Kod 3.7: Python salt ile SHA256 Kullanımı

```
salt = Random.new().read(128)
hash = SHA256.new()
hash.update(salt + passwd)
message = hash.hexdigest()
```

3.3.3 Hash İşlemini İteratif Yapılması

Tuzlama yöntemi de elinde yeterince hesaplama gücü olan saldırganlara karşı tam güvenlik sağlamamaktadır. Bu işlemi zorlaştırmak için hash işlemi iteratif yapılabilir. Bu durumda saldırganın her girdi için harcayacağı işlem gücü artacak ve dolayısıyla şifreyi öğrenme olasılığı azalacaktır.

Hash işlemini 1000 kez yapmak, sunucu için çok küçük bir yük getircektir. Bu nedenle bu tekniği uygulamak sorun oluşturmayacaktır.

Kaynak Kod 3.8: Python iteratif SHA256 Kullanımı

```
def digest_passwd(passwd, salt):
    hash = SHA256.new()
    hash.update(salt + passwd)
    message = hash.digest()

    for i in range(1000):
        hash = SHA256.new()
        hash.update(message)
        message = hash.digest()

    return hash.hexdigest()
```

Yukarıdaki kod bizim implementasyonumuzdan alınmıştır. Şifreyi *salt* kullanarak *1000* kez hash'ler.

3.3.4 Hash Karşılaştırırken Zamanlama Saldırılarından Kaçınma

Zamanlama saldırıları iki hash değeri karşılaştırılırken yapılabilecek saldırılardır. Bu nedenle karşılaştırma işlemi yapılacak yerde çalışan algoritma bu saldırılara karşı güvenli olmalıdır. Bu güvenlik algoritmanın her zaman aynı sürede karşılaştırma işlemi yapması ile sağlanabilir.

Örnek vermek gerekirse string karşılaştırma işlemi greedy bir algoritma olduğu için ilk farkı gördüğünde sonlanacaktır. Bu da saldırdanın her seferinde 1 karakter doğru bularak en sonunda tüm hash değerine ulaşmasına imkan sağlayacaktır.

Bunu önlemek için biz tüm değerleri tek tek XOR işlemine sokup sonuç sıfıra eşit mi diye baktık. Aşağıda bu yöntemi nasıl implement ettiğimiz gözükmektedir.

Kaynak Kod 3.9: Zamanlama Saldırılarına Karşı Sabit Zamanlı Karşılaştırma

```
def verify_passwd(passwd, salt, digest):
    computed_digest = digest_passwd(passwd, salt)
    result = 0
    for x, y in zip(computed_digest, digest):
        result |= ord(x) ^ ord(y)
    return result == 0
```

3.3.5 Sahte İsteklerin CAPTCHA ile Tesipti

İnternet üzerinde duran ve 7/24 hizmet veren bir servise, pek çok ilgisiz veya sahte istek gelebilir. Sadece kullanıcı tarafından yapılan isteklerin tespit edilmesi ve diğer isteklerin engellenmesi uygulama güvenliği için olmazsa olmazdır.

Bunu yapmak için en çok kabul gören yöntem *captcha* ismi verilen ve sadece insanlar tarafından çözülebilecek bilmeceler sormaktır.

Bu yöntem için çeşitli kütüphaneler ve servisler mevcuttur. Özellikle google tarafından sunulan *recaptcha* oldukça popülerdir. Fakat malesef biz bu özelliği implement edemedik.

Bölüm 4

Haberleşme Kanalı Güvenliği

4.1 İletişimin Şifrelenmesi için SSL protokolünü kullanmak

İnternet güvensiz bir iletişim ortamı olduğu için, önemli bilgileri taşımak için uygun değildir. İnterneti güvenli hale getirmek için *SSL protokolü* ortaya atılmıştır. Biz de kullanıcının şifresi gibi önemli bilgileri taşıdığımız için bu protokolü kullanmak zorundayız.

SSL protokolü kullanmak için *nginx*'in yeteneklerinden faydalandık. nginx SSL desteği ile gelen yetenekli bir web sunucudur. SSL desteğini aktif hale geçirmek için aşağıdaki ayarı yaptık.

Kaynak Kod 4.1: nginx SSL ayarları

```
 server {
    listen 80;
    server_name bil553.com;
    return 301 https://bil553.com$request_uri;
   }

server {
    listen 443;
    ssl on;
    ssl_certificate      /etc/nginx/ssl/server.crt;
    ssl_certificate_key  /etc/nginx/ssl/server.key;

    server_name bil533.com;


    location / {
        uwsgi_pass 127.0.0.1:8889;
        include uwsgi_params;
    }
  }
```

Bölüm 5

Kullanıcı Tarafı Güvenliği

Tüm sunucu ve iletişim kanalı güvenlik önlemlerinin yanında kullanıcı tarafında da güvenlik önlemi almak gereklidir.

5.1 Keylogger'lara Karşı Sanal Klavye

Kullanıcının bilgisayarında, klavye hareketlerini takip eden zararlı yazılım (keylogger) kurulu olabilir. Bu nedenle kullanıcıya sanal klavye sunmak önemli bir güvenlik önlemidir.

Biz de *JavaScript* kullanarak kullanıcıya bir sanal klavye sunuyoruz ve şifre bilgilerini bu klavye ile yazmasını zorunlu tutuyoruz.

Bu yöntem kullanışlılık açısından biraz zorluk getirse de keylogger'lara karşı kullanıcıyı korumaktadır.

5.2 Fare Hareketlerinin Dinlenmesine Karşı Rastgele Tuşlar

Kullanıcının bilgisayarında, gelişmiş bir keylogger yazılımı olabilir ve bu yazılım fare hareketlerini de kaydediyor olabilir. Bu nedenle sanal klavyenin tuşlarının yerlerini karıştırmak bu zararlılara karşı koruma için önemlidir.

Biz de kullanıcıya sunduğumuz sanal klavyenin tuşlarını karıştırarak kullanıcıyı bu zararlılardan korumayı amaçlıyoruz.

5.3 Farklı Ülkelerden Oturum Açmaya Karşı Hesap Kilitleme

Kullanıcı her zaman oturum açtığı ülke dışından oturum açtıysa bir saldırı ile karşı karşıya olabilir. Bu yüzden biz, farklı ülkeden oturum açan kullanıcının hesabını geçici olarak pasif hale geçiriyoruz. Ve kendisine bu durumu belirten bir mail atıyoruz.

Kullanıcı bu mail içindeki linke tıklayarak kendi hesabını aktif hale geçirip kullanmaya devam edebiliyor. Ya da bir saldırı ile karşı karşıya kalmışsa önlemini alabiliyor. Bu süreç içinde kullanıcı alım yapamıyor.

5.4 Şifre Sıfırlayan Kullanıcıya Mail İle Haber Verme

Önceden de belittiğimiz gibi, şifre sıfırlama esnasında saldırganın rastgele değeri tahmin etmeye şalışması nedeniyle şifre sıfırlama işlemlerinin aktif saldırılara zaafiyeti vardır. Bu olasılık 48 bitlik rastgele değer için 2^{48} olsa da hala bir risk teşkil etmektedir.

Bu nedenle biz şifresini sıfırlayan kullanıcıya, şifre sıfırlama işlemi gerçekleştiğine dair mail göndererek, kendisini haberdar ediyoruz. Gönderdiğimiz mail'de işlemin gerçekleştiği IP adresini de göndererek kullanıcının bu işlemi kimin tarafından yaptıldığını tayin etmesini kolaylaştırmayı hedefliyoruz. Eğer işlem kendisi tarafından yapılmamışsa bize ulaşmasını söylüyoruz.

5.5 SMS ile satış onayı

Kullanıcının sisteme bildirdiği telefon numarası üzerinden, kullanıcı ile güvenli bir iletişim kanalı kurulabilir. Bu kanal sayasinde de doğrulama işlemleri yapılabilir. Örneğin satın alma işleminin SMS ile onaylanması sisteme sızan saldırganların satın alma yapmasını önleyecektir.

Bu özelliği kullanmak için çeşitli kütüphaneler ve servisler mecvuttur. Örneğin *sms* kütüphanesi enfora gsm modemleri ile mesaj gönderip almayı sağlarken, *twilio.com* gibi kuruluşlar SMS hizmeti sağlamaktadır. Birinci yöntem için de gereken donanımlara, ikinci yöntem için de servis bedelini karşılayacak paraya sahip olmadığımız için bu özelliği implement edemedik.

5.6 Yapay Zeka ile Şifre Politikası Belirleme

Şifrelere yapılan saldırıların önemli bir çoğunluğu *dictionary attack* denilen olası şifre listelerindeki her bir girdiyi tek tek deneyerek yapılmaktadır. Bizim de derslerin lab uygulaması sırasında kullandığımız *john the ripper* aracı bu maktık ile çalışmaktadır.

Bunu göz önünde bulundurarak, şifre politikası şu şekilde belirlenebilir:

1. Örnek bir olası şifre listesi bul
2. Kullanıcının şifresinin herhabgi bir şifre listesinde bulunabilme olasılığını yapay zeka methodları ile hesapla
3. bu olasılık belirlenen oranın üzerinde ise şifreyi reddet

Böylece kullanıcı şifrelerinin çalınması durumunda, şifrelerin hesaplanabilmesi olasılığını iyice düşürebiliriz. Bu yöntemin implementasyonu, başlı başına bir yapay zeka dersi projesi olabileceği için bu yöntemi implement etmedik.

Bölüm 6

Sonuç

Sistem güvenliği, sistem içinde çalışan bütün parçaların güvenliği demektir. Unutmamak gerekir ki bir sistemin güvenliği, en zayıf halkasının güvenliği kadardır. Biz de bu proje kapsamında sunucudan kullanıcıya kadar her parçası ile güvenli bir sistem tasarlamaya çalıştık. Bunu yaparken de kripto araçlarından faydalandık.

Yaptıklarımızı özetlememiz gerekirse:

1. Hazır Kripto Kütüphanesi Kullanmak
2. Rastgele Sayı Üreticisinin Kriptografik Güvenli Olması
3. Güvenilir Kaynaklardan Yazılım Yüklemek
4. Uzaktan Erişimin Güvenliğini Arttırmak
5. Güvenlik Duvarı
6. Veri Tabanına Erişimi Kısıtlamak
7. SQL Injection Saldırılarına Karşı Önlem Almak
8. Şifrelerin Açık Bir Şekilde Saklanmaması
9. Hash İşlemi Sırasında Salt Kullanılması
10. Hash İşlemini İteratif Yapılması
11. Hash Karşılaştırırken Zamanlama Saldırılarından Kaçınma
12. Sahte İsteklerin CAPTCHA ile Tesipti
13. İletişimin Şifrelenmesi için SSL protokolünü kullanmak
14. Keylogger'lara Karşı Sanal Klavye
15. Fare Hareketlerinin Dinlenmesine Karşı Rastgele Tuşlar
16. Farklı Ülkelerden Oturum Açmaya Karşı Hesap Kilitleme
17. Şifre Sıfırlayan Kullanıcıya Mail İle Haber Verme
18. SMS ile satış onayı
19. Yapay Zeka ile Şifre Politikası Belirleme

Kaynakça

[1] Auguste Kerckhoffs, "La cryptographie militaire", *Journal des sciences militaires*, vol. IX, 1883

[2] Java SE 6 Update 17, *Update Release Notes*
`http://www.oracle.com/technetwork/java/javase/6u17-141447.html`

[3] Jeffrey S. Smith, "Why customers are flocking to Linux", *IBM*, 2008-06-03.

[4] Mustafa Simav,
`http://msimav.net/2013/04/13/increasing-ssh-server-security/`

[5] Alexander, Steven, *"Password Protection for Modern Operating Systems"*, 2004