1.1.1 Kodda Açık Bir Şekilde Parola Bulundurulması (Use of Hardcoded Password) (CWE-798) (CWE-259)

Açıklık Önem Derecesi: Düşük

Açıklığın Etkisi: Hassas Bilgilere Yetkisiz Erişim, Bilgi İfşası, Sızma girişimlerinde saldırının boyutunun artması

Açıklığın Barındıran Dosyalar/Satırlar:

Proje Dosyası/Dosya Adı	Satır Numarası

Açıklığın Açıklaması:

Uygulama geliştiricileri uygulama kaynak kodlarına parola bilgisi gömebilmektedirler. Bu hassas veriler kimlik bilgilerini doğrulama, bir web api hizmetine istek yapma (api key) v.b. nedenler için yer alabilir. Güvenliğe derinlikli defans (depth of defence) perspektifinden yaklaşacak olursak kaynak kodda bu tarz hassas verilerin açık bir şekilde yer alması güvenlik riski teşkil etmektedir.

Uygulama güvenliğinde temel esas saldırganların içeriye hiçbir şekilde girememesidir. Fakat bazen güvenlik ne kadar üst seviyede olursa olsun saldırganın içeri girmesi devasa bir yüzeyde sadece bir boşluk yakalamasına bakabilmektedir. Dolayısıyla güvenliği daha garanti hale getirmenin yolu güvenliğe katman stratejisiyle yaklaşmaktan geçer. Bu ise örneğin uygulamanıza veya ağınıza gelecekte olası bir sızma girişimi yapıldığında şayet girişim başarılı olursa saldırganın içerideyken verebileceği zararı minimize edecek, belki de tam manasıyla sıfırlayacak iç katmanınızdaki güvenliğinizle mümkündür. Uygulama sunucunuza sızıldığı senaryoyu ele alacak olursak saldırgan örneğin uygulama kaynak kodunu okuyarak oturum açma kullanıcı adı ve şifre bilgilerini veya web servis erişim bilgilerini (api anahtarını), v.b. ele geçirebilir. Bu sayede belki sadece web sunucu içerisinde uygulamanın kök dizininde sıkışıp kalacakken şimdi edindiği bilgi doğrultusunda uygulamanın saklı arayüzlerine atlama yapabilir veya diğer bağlantı noktası web servise erişebilir. Uygulama üzerinden yetki ölçüsünde bu şekilde daha da ilerleyebilir ve uzanabildiği ölçüde atlamalar yapabilir.

Uygulamanızın deployment metoduyla servis edildiği durumu ele alacak olursak ve saldırganın yine web sunucunuza sızdığını düşünecek olursak saldırganın uygulama kök dizininde sadece deploy edilmiş web uygulamanızı görmesi onu engellemeyecektir. Saldırgan edindiği deploy edilmiş uygulama dosyalarına tersine mühendislik araçları (örn; Microsoft Documents resmi web sitesinde bahsedilen Ildasm.exe (IL Disassembler) aracını)

kullanarak içerlerinde şifre addedilebilecek string taraması yapabilir ve kritik verileri yine elde edebilir. Ayrıca eğer saldırgan parola bilgisini ele geçirecek olursa bu parola uygulamanın yeni bir sürümü derlenmediği sürece (yani parola kolayca değiştirilemeyeceğinden) kötüye kullanımı devam edecektir. Dahası eğer bu uygulama sayısız sisteme komple veya modül olarak dağıtılırsa sistemin çalınan parolası otomatik olarak dağıtılan tüm sistemlere girilmesine olanak verecektir. Dolayısıyla bu v.b. bilgi toplamalar ile saldırganların derine inme ihtimallerini engellemek / asgariye çekmek için kaynak kodlardaki parola hassas verisinin saklanması önerilmektedir.

> Microsoft resmi sayfası der ki:
>
> "Uygulama kodlarına gömülen bağlantı string'leri güvenlik zafiyetlerine ve bakım problemlerine yol açabilmektedir. Uygulamanın kaynak kodu içerisinde derlenmiş şifrelenmemiş bağlantı string'leri Ildasm.exe (IL Disassembler) aracı kullanılarak görüntülenebilir. Dahası her bağlantı string değişimi gerektiği zaman uygulamanız tekrar tekrar derlenmek zorunda kalabilir. Bu nedenlerden dolayı uygulama içerisinde depolu olan bağlantı string'lerinin uygulama yapılandırma dosyasına taşınmasını önermekteyiz." (https://docs.microsoft.com, 03/30/2017)

Uygulama parolaları açık metin olarak kaynak kodlarda yer aldığında "Use Of Hardcoded Password" açıklığı olarak işaretlenirler. Uygulamalardaki bu açıklığa şöyle bir örnek verilebilir:

> Java - Güvensiz Kod:

```
bool isAdmin(String username, String password) {

    bool isMatch = false;

    if (username.equals("admin")) {
        if (password.equals("P@ssw0rd"))
            return isMatch = true;
    }

    return isMatch;
}
```

Bu örnekte kullanıcı kimlik doğrulaması yapılmaktadır ve parola kıyaslamasındaki geçerli parola hardcoded (açık bir şekilde) yer almaktadır. Bu güvensiz bir kullanımdır. Güvenli hale dönüştürülmüş hali ise şu şekildedir:

Java - Güvenli Kod:

```
bool isAdmin(String username, String password) {

    bool adminPrivs = false;

    if (authenticateUser(username, password)) {
        UserPrivileges privs = getUserPrivileges(username);

        if (privs.isAdmin)
            adminPrivs = true;
    }

    return adminPrivs;

}
```

Bu güvenli kullanım örneğinde yapılan kimlik doğrulamada geçerli parola hardcoded (açık bir şekilde) yer almamaktadır. authenticateUser() metodu gelen kullanıcı adı ve parolayı içerisinde veritabanından geçerli kullanıcı adı ve parola bilgilerini çekerek kıyaslamaktadır.

Uygulamalardaki bu açıklığa bir başka örnek olarak şu da verilebilir:

Javascript - Güvensiz Kod:

```
[Vulnerable] Hardcoded Account Password

var username = request.body.username;
var password = request.body.password;
var admin_username = "admin";
var admin_password = "5up3r53cr3t";

if (username == admin_username && password == admin_password) {
    // Authenticate
}
else {
    // Reject
}
```

Bu örnekte yine kullanıcı kimlik doğrulaması yapılmaktadır ve parola kıyaslamasındaki geçerli parola hardcoded (açık bir şekilde) yer almaktadır. Bu güvensiz bir kullanımdır. Güvenli hale dönüştürülmüş hali ise şu şekildedir:

Javascript - Güvenli Kod:

```
[SECURE] Authenticating by Querying the Database with Credentials

var username = request.body.username;
var password = secureHashImplementation(request.body.password);

connection.query('SELECT * FROM users WHERE name=? AND
hashed_password=?',[username, password], function(err,results) {

    if (error) {
        // handle error
    }
    if (results.length == 1) {
        // Authenticate
    }

});
```

Bu güvenli kullanım örneğinde yapılan kimlik doğrulamada geçerli parola hardcoded (açık bir şekilde) yer almamaktadır. Parola veritabanında hash'li bir şekilde yer almaktadır. Kimlik doğrulamada kullanıcıdan gelen parolanın önce hash'i alınmaktadır ve sonra veritabanına sorgu yapılarak kullanıcı adı ve parola hash'i kıyaslanmaktadır.

Parola v.b. hassas veriler kaynak koda gömülmemelidirler. Diğer türlü kaynak kodda açık bir şekilde parola bulundurulması açıklığı şeklinde ele alınırlar. Bu açıklıktan her çeşit uygulama etkilenir. Fakat "özellikle" dağıtılan uygulamalar (yani masaüstü uygulamalar, router firmware uygulamaları, firewall firmware uygulamaları, yazıcı firmware uygulamaları v.b.) veya açık kaynak kodlu uygulamalar daha çok etkilenir.

Eğer şu sorulardan herhangi bir tanesine evet deniyorsa kurum uygulama tarif edilen bu açıklıktan etkileniyordur:

- Kaynak koddaki hesap bilgileri hassas bir bileşene mi erişim izni veriyor (örn; Bir veritabanına, bir dosya depolamasına, bir API'ye veya bir servise) ?

- Kaynak koddaki hesap bilgileri production (yayın) ortamında mı kullanılıyor?

- Kaynak koddaki hesap bilglerini güncellemeden önce uygulamayı yeniden dağıtmak mı gerekiyor?

Uygulama kaynak kodlarında açık bir şekilde parola olunca;

- Nadiren değişecekleri için güvenlik noktasında daha az güvenlidirler.

- Kodda açık metin halinde yer aldıklarından çok görünürdürler. Eğer kötü niyetli bir kimse (öreğin harici bir kurum çalışanı) bunları görürse (uygulama admin parolaları, web api parolaları gibi) uygulama risk altına girer.

Kurum uygulama kaynak kodlarının açık bir şekilde parola bilgilerini barındırdığı tespit edilmiştir.

::::::::: BULGU ::::::::::

Açıklığın Önlemi:

“Kullanıcı parolaları” veritabanında veya bir dizin hizmetinde (linux veya windows directory service’te - örn; Active Directory’de) güçlü bir tek yönlü şifreleme algoritması (örn; bcrypt, scrypt, PBKDF2 veya Argon2 v.b.) ile şifrelenerek tutulmalıdır. Kullanıcı parola kıyaslamalarında kullanıcı taraflı gelen parola aynı tek yönlü şifreleme algoritması kullanılarak şifrelenmelidir ve oluşan hash ile veritabanındaki / dizin hizmetindeki hash kıyaslanmak suretiyle işlemlerin yürütülmesi yolu takip edilmelidir.

“Sistem parolaları” ise bir dosyada (config dosyasında / properties dosyasında,… veya v.b. dosyada) veya veritabanında tersine çözümlenebilir bir şekilde şifrelenerek tutulmalıdır. Şifrelemeyi açmak için bir deşifreleme anahtarı kullanılmalıdır. Şifreleme anahtarları hardcoded (açık bir şekilde) olmamalıdır ve güvenli bir şekilde yönetilmelidir. Uygulama kaynak kodları yapılandırma dosyalarından veya veritabanından sistem parolalarını deşifreleme ile çekerek işlemlerini yürütmelidir.

Referanslar:

1. https://www.owasp.org/index.php/Preventing_LDAP_Injection_in_Java
2. https://www.webguvenligi.org/wp-content/uploads/2007/11/authenticationtrk.pdf
3. https://www.owasp.org/index.php/Use_of_hard-coded_password
4. https://docs.microsoft.com/en-us/dotnet/framework/data/adonet/connection-strings-and-configuration-files
5. https://stackoverflow.com/questions/4227921/how-can-we-have-two-connection-strings-in-web-config-and-switch-betweeen-them-in
6. https://stackoverflow.com/questions/5642474/setting-up-connection-string-in-asp-net-to-sql-server
7. https://www.c-sharpcorner.com/UploadFile/7d3362/various-ways-to-specify-connection-string-in-Asp-Net-web-app/
8. https://www.youtube.com/watch?v=v1EWUAYNA1g
9. https://www.youtube.com/watch?v=Gf3kFBAovXw
10. https://stackoverflow.com/questions/15365210/failed-to-encrypt-the-section-connectionstrings-using-provider-rsaprotectedco
11. https://superuser.com/questions/401495/equivalent-of-unix-find-command-on-windows

12. https://docs.microsoft.com/en-us/previous-versions/aspnet/hh8x3tas(v%3Dvs.100)
13. https://www.youtube.com/watch?v=lzm_uTW9Ug8
14. http://www.sortedset.com/how-to-encrypt-decrypt-a-password-stored-in-a-properties-file-with-java-jasypt-apache-commons-configuration/
15. https://github.com/sortedset/jasyptblog
16. https://commons.apache.org/proper/commons-configuration/userguide/upgradeto2_0.html
17. https://commons.apache.org/proper/commons-configuration/apidocs/org/apache/commons/configuration2/ex/ConfigurationException.html
18. https://commons.apache.org/proper/commons-configuration/apidocs/org/apache/commons/configuration2/PropertiesConfiguration.html
19. https://www.programcreek.com/java-api-examples/?class=org.apache.commons.configuration2.builder.FileBasedConfigurationBuilder&method=save
20. https://stackoverflow.com/questions/22541455/warning-the-method-assertequals-from-the-type-assert-is-deprecated
21. https://www.tutorialspoint.com/junit/junit_suite_test.htm
22. http://www.jasypt.org/dependencies.html
23. https://www.youtube.com/watch?v=2xA82u3Q84M
24. http://docs.vizrt.com/viz-content-pilot-guide/5.6/configuration_database_service_names_and_sid.html
25. https://docs.oracle.com/database/121/ODPNT/featConnecting.htm#ODPNT199
26. https://docs.microsoft.com/en-us/dotnet/framework/data/adonet/oracle-and-adonet
27. https://www.oreilly.com/library/view/adonet-cookbook/0596004397/ch01s09.html
28. https://docs.microsoft.com/tr-tr/dotnet/framework/data/adonet/connection-string-syntax#sql-server-authentication-with-sqlclient
29. https://www.c-sharpcorner.com/UploadFile/nipuntomar/connection-strings-for-oracle/
30. https://www.oracle.com/webfolder/technetwork/tutorials/obe/db/12c/r1/appdev/dotnet/Web_version_Fully_Managed_ODPnet_OBE/odpnetmngdrv.html
31. https://stackoverflow.com/questions/46408085/oracle-connection-in-c-sharp-connection-string
32. https://stackoverflow.com/questions/8202999/encrypt-a-connection-string-how
33. https://stackoverflow.com/questions/8195099/java-how-to-store-password-used-in-application/8195195#8195195
34. https://rules.sonarsource.com/typescript/RSPEC-2068
35. https://www.appmarq.com/public/tqi,1020820,Avoid-hardcoded-passwords-TypeScript
36. https://docs.snyk.io/scan-application-code/snyk-code/security-rules-used-by-snyk-code/javascript-and-typescript
37. https://rules.sonarsource.com/php/RSPEC-2068
38. https://offensive360.com/how-to-prevent-hardcoded-passwords/