1.1.1 Şifresiz İletişim Kullanılması (Cleartext Submission of Sensitive Information) (CWE-319)

Açıklık Önem Derecesi: Orta

Açıklığın Etkisi: Hassas bilgilerin ele geçirilebilmesi

Açıklığın Barındıran Dosyalar/Satırlar:

Proje Dosyası/Dosya Adı	Satır Numarası

Açıklığın Açıklaması:

Ağ üzerinde veri gönderme eğer iletişim kanalı güvenli bir protokolle korunmuyorsa risklidir. Özellikle güvensiz ağlara ve hotspot'lara sıklıkla bağlanan mobil cihazlar için bu durum daha da kritik öneme sahiptir. Bir saldırgan gizlice ve kolaylıkla havada gönderilen bilgileri dinleyebilir, elde edebilir ve bazı durumlarda ayrıca dinlediği veriyi manipüle ederek örneğin kullanıcı ekranına yansıyacak legal bir web site arayüzüne zararlı bir sahte login sayfası enjekte edebilir.

Hassas ve kişisel veriler (örn; parolalar, tc kimlik numaraları, kredi kart verileri ve diğer PII (Personally Identifiable Information) veriler) ağ üzerinden gönderilirken her zaman korunmalıdır. Özel verileri şifresiz kanal üzerinden gönderme (örn; SSL/TLS şifrelemesi veya diğer tür şifrelemeler kullanılmadan gönderme) kullanıcıların gizli bilgilerinin açığa çıkmasına ve ayrıca kullanıcı hesaplarının devralınmasına, mali dolandırıcılıklara maruz bırakabilir.

Not:

Eğer SSL/TLS kanalı kullanımı front-end'de, reverse proxy'de veya buna benzer konumlarda sonlandırılıyorsa bu durum risk içermez.

Uygulamalarda şifresiz bağlantı başlatılması açıklığına örnek olarak şunlar verilebilir:

Java - Güvensiz Kod Bloğu (1):

```java
// Using Standard Socket (No Encryption) for a Basic Authentication Server

public void runServer() {
    ServerSocket server = new ServerSocket(PORT);
    Socket client;

    while (true) {
        client = server.accept();
        MyUser user = handleRequest(client.getInputStream());

        PrintWriter output = new PrintWriter(server.getOutputStream());
        output.println(user.AccountId);
        output.flush();
    }
}
```

Java - Güvenli Kod Bloğu (1):

```java
// Using Encrypted SSLSocket for a Basic Authentication Server

public void runServer() {
    try {
        SSLServerSocketFactory factory =
            (SSLServerSocketFactory) SSLServerSocketFactory.getDefault();

        ServerSocket server = factory.createServerSocket(PORT);
        Socket client;

        while (true) {
            client = server.accept();
            MyUser user = handleRequest(client.getInputStream());

            PrintWriter output = new PrintWriter(server.getOutputStream());
            output.println(user.AccountId);
            output.flush();
        }
    }
    catch (IOException ex) {
        handleException(ex);
    }
    finally {
        if (output != null) output.close();
        if (client != null)
            if (!client.isClosed()) client.close();
        if (server != null)
            if (!server.isClosed()) server.close();
    }
}
```

Bu örnekte güvensiz java kod bloğunda standart bir (şifresiz) socket bağlantısı başlatılmaktadır. Dolayısıyla araya giren adam saldırılarına karşı koruma yoktur. Güvenli java kod bloğunda ise sslsocket (şifreli socket) bağlantısı başlatılmaktadır. Yani araya giren adam saldırılarına karşı koruma vardır.

Java - Güvensiz Kod Bloğu (2):

```java
// Authenticating Server to Server via HTTP

public static boolean authenticateServerToServer(String username, String
password) throws IOException {
    String stringUrl = "http://" + HOSTNAME + "/" + URI_PATH;
    URL url = new URL(stringUrl);
    HttpURLConnection conn = (HttpURLConnection)url.openConnection();
    conn.setRequestMethod("POST");
    conn.setDoOutput(true);

    String postParameters  = "username=" + username + "&password=" + password;
    byte[] postRequestBytes = postParameters.getBytes();
    OutputStream os = conn.getOutputStream();
    os.write(postRequestBytes);
    os.flush();
    os.close();
    return conn.getResponseCode() == HttpURLConnection.HTTP_OK;
}
```

Java - Güvenli Kod Bloğu (2):

```java
// Authenticating Server to Server via HTTPS

public static boolean authenticateServerToServer(String username, String
password)  throws IOException {
    String stringUrl = "https://" + HOSTNAME + "/" + URI_PATH;
    URL url = new URL(stringUrl);
    HttpsURLConnection conn = (HttpsURLConnection)url.openConnection();
    conn.setRequestMethod("POST");
    conn.setDoOutput(true);

    String postParameters  = "username=" + username + "&password=" + password;
    byte[] postRequestBytes = postParameters.getBytes();
    OutputStream os = conn.getOutputStream();
    os.write(postRequestBytes);
    os.flush();
    os.close();
    return conn.getResponseCode() == HttpURLConnection.HTTP_OK;
}
```

Bu örnekte güvensiz java kod bloğunda http bağlantısı başlatılmaktadır. Dolayısıyla araya giren adam saldırılarına karşı koruma yoktur. Güvenli java kod bloğunda ise https bağlantısı başlatılmaktadır. Yani araya giren adam saldırılarına karşı koruma vardır.

Python - Güvensiz Kod Bloğu (1):

```
# Making an HTTP Request via http.client

conn = http.client.HTTPConnection(domain)
resp = conn.request("GET",url).getresponse()
```

Python - Güvenli Kod Bloğu (1):

```
# Making an HTTPS Request via http.client

conn = http.client.HTTPSConnection(domain)
resp = conn.request("GET",url).getresponse()
```

Python - Güvensiz Kod Bloğu (2):

```
# Making an HTTP Request using Python Requests

url = "http://" + domain
resp = requests.get(url)
```

Python - Güvenli Kod Bloğu (2):

```
# Making an HTTPS Request using Python Requests

url = "https://" + domain
resp = requests.get(url)
```

Python - Güvensiz Kod Bloğu (3):

```
# Making an HTTP Request using urllib

resp = urllib.request.urlopen("http://" + domain)
```

Python - Güvenli Kod Bloğu (3):

```
# Making an HTTPS Request using urllib

resp = urllib.request.urlopen("https://" + domain)
```

Bu örneklerde ise güvensiz python kod bloklarında çeşitli şekillerde http bağlantısı başlatıldığı görülmektedir. Bu durum araya giren adam saldırılarına imkan verir ve

güvensizdir. Güvenli python kod bloklarında ise çeşitli şekillerde https bağlantısı başlatıldığı görülmektedir. Bu durum araya giren adam saldırılarını önler ve güvenlidir.

Uygulamalarda şifresiz bağlantı başlatılıyorsa ve/veya bu bağlantıda veri gönderiliyorsa "Cleartext Submission of Sensitive Information (CWE-319)" açıklığı olarak ele alınırlar. Kurum uygulamada bu açıklık tespit edilmiştir:

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

Açıklığın Önlemi:

Bu açıklık şu şekilde önlenir:

- Tüm PII (Personally Identifiable Information) ve diğer hassas verileri ağ üzerinden gönderirken daima koruyun.
- Hassas veri iletirken SSL/TLS kullanın. Alternatif olarak diğer şifreleme protokolleri (örn; IPsec veya SSH v.b.) de kullanılabilir.
- Uygulamanın iletilen kişisel verilere gerçekten ihtiyacı var mı tekrardan gözden geçirin. Çünkü veri iletilmezse ifşa ihtimali olmaz.
- Web uygulamalarda kanalın güvenilir olduğunu doğrulamadan kişisel verileri yanıt çıktısı olarak doğrudan göndermeyin.
- Ayrıca kişisel verileri doğrudan standart socket'lere yazdırmayın. Bunun yerine kanalın SSL/TLS'den yararlanmasını temin etmek için daima SSLSocket kullanın.

Referanslar:

1. https://cwe.mitre.org/data/definitions/319.html