1.1.1 Log Taklit Etme (Log Forging) (CWE-117)

Açıklık Önem Derecesi: Düşük

Açıklığın Etkisi: Uygulama verisini modifiye etme, aktiviteleri gizleme, yetkisiz kod veya komutlar çalıştırma

Açıklığın Barındıran Dosyalar/Satırlar:

Proje Dosyası/Dosya Adı	Satır Numarası

Açıklığın Açıklaması:

Uygulamalar bir hata meydana geldiğinde hata ayıklama için veya saldırı meydana geldiğinde gereken aksiyonu belirlemek için log kaydı alma mekanizmalarını kullanırlar. Kullanıcı girdilerinin log'lara yazdırıldığı uygulamalarda ise bir güvenlik riski söz konusudur. Eğer kullanıcı girdileri "denetlenmeden" / "olduğu gibi" uygulama veya sistem log dosyalarına yazdırılmaktaysa kötü niyetli kullanıcılar Log Taklit Etme (diğer adıyla; Log Enjeksiyonu) saldırıları düzenleyebilirler.

Log Taklit Etme saldırısını izah etmek adına şu web uygulama sayfası örnek olarak verilebilir.

Örnek Bir Açıklıklı Web Sayfası:

```
...
String val = request.getParameter("val");
try {
    int value = Integer.parseInt(val);
}
catch (NumberFormatException) {
    log.info("Failed to parse val = " + val);
}
...
```

Bu web uygulama sayfasında istemciden gelen val parametresi ile bir integer değer okuması yapılıyor. Eğer istemciden gelen parametredeki değer integer veri tipinde değilse (farklı bir veri tipinde değer ise) catch istisna bloğuna atlanıyor. Ardından catch istisna bloğunda girdinin integer olmadığına dair haber veren hata mesajı ile parametre değerinin (girdinin) kendisi log'lanıyor.

Eğer bir kullanıcı val parametresine “yirmi-bir” string değerini girerse değer integer olmadığından sunucu tarafta aşağıdaki gibi log’lanacaktır:

```
INFO: Failed to parse val=yirmi-bir
```

Fakat eğer bir kullanıcı val parametresine satır atlatma (%0a) karakterleri şu şekilde girerse

“yirmi-bir%0a%0aINFO:+User+logged+out%3dkotuNiyetliKullanici”

sunucu tarafta aşağıdaki hata log’laması gerçekleşecektir:

```
INFO: Failed to parse val=yirmi-bir
INFO: User logged out=kotuNiyetliKullanici
```

Yani görüldüğü gibi kullanıcı girdisinde satır atlatma karakterleri kullanıldığından ve kullanıcı girdisi log dosyasına yazdırıldığından sahte bir ekstra log kaydı log dosyasına eklenmiştir. Bu şekilde saldırganlar girdi parametrelerine satır atlatma karakterleri girerek kendilerine ait keyfi log girdileri sunucu taraftaki log dosyasına ekleyebilirler.

Kötü niyetli kullanıcılar keyfi log kaydı ekleyerek belirli koşullar altında “kod çalıştırma” saldırısı da düzenleyebilirler. Örneğin log forging açıklıklı bir web uygulamada şöyle bir url tetiklendiğinde;

https://www.zafiyetliuygulama.com/index.php?file=`<?php echo phpinfo(); ?>`

ve file parametresinin log’landığı durum ele alındığında parametredeki php kod çıktısı log dosyasına yazdırılacaktır. Eğer log dosyası public bir dizinde konumlandırılmışsa ve http get isteği ile dışarıdan erişilebiliyorsa bu durumda girdideki php kodun çıktısı web tarayıcıda görüntülenebilecektir. Böylelikle girilen her bir php kodunun çıktısı web tarayıcıda log dosyası görüntülenerek elde edilebilecektir.

Log Forging açıklığını izah etmek adına çeşitli teknolojilerde güvensiz ve güvenli kod blokları örneklerine yer verilmiştir:

Java - Güvensiz Kod Örneği:

```
// GÜVENSİZ ÖRNEK

// TR: Kullanıcı Girdisi Log'lamayı Etkiliyor.
// EN: User Input Affects Logging

protected void doPost(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {

    String color = request.getParameter("color");
    logger.info("{} was picked", color);
    if colorList.contains(color){
        // Handle Response
    }else{
        // Handle Response
    }
}
```

Java - Güvenli Kod Örneği:

```
// GÜVENLİ ÖRNEK

// TR: Kullanıcı Girdisi Log'lama Öncesi Encode'lanıyor.
// EN: User Input Encoded Prior Logging

protected void doPost(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {

    String color = request.getParameter("color");
    cleanColor = color.replace('\t', '_').replace('\n', '_').replace('\r',
'_');
    logger.info("{} was picked", cleanColor);
    if colorList.contains(cleanColor){
        // Handle Response
    }else{
        // Handle Response
    }
}
```

Java güvensiz kod örneğinde kullanıcı girdisi hiçbir kontrol olmadan, doğrudan log dosyasına eklenecek kaydın sonuna eklenmektedir. Bu durum kullanıcının boşluk ekleme (tab) ve satır atlatma karakterlerini (line feed karakterlerini) kullanarak log dosyasına ekstradan keyfi log kaydı girebilmesi ile sonuçlanabilir. Java güvenli kod örneğinde ise kullanıcı girdisi log dosyasında log kaydının sonuna eklenmeden önce kodlama (encode'lama) işlemine tabi

tutuluyor. Bu kodlama işlemi boşluk ekleme (tab) ve satır atlatma (line feed) karakterlerini etkisizleştireceğinden olası bir keyfi log enjeksiyonuna fırsat vermeyecektir.

C# - Güvensiz Kod Örneği:

```
// GÜVENSİZ ÖRNEK

// TR: Kullanıcı Girdisini Log Dosyalarına Yazdırma - MVC 5
// EN: Writing User Input Into Log Files - MVC 5

public ActionResult Index()
{
    string Id = Request.QueryString["Id"];
    Logger logger = new Logger(LogType.File);

    //Id parametresi URL adresten alınıyor. Dolayısıyla kurcalanabilir.
    logger.TraceStart("User entered to homepage, user id: " + Id);

    return View();
}
```

C# - Güvenli Kod Örneği:

```
// GÜVENLİ ÖRNEK

// TR: Sunucu taraflı çekilen User Id'nin Log Dosyasına Yazdırılması - MVC 5
// EN: Writing User Id From Server Into Log File - MVC 5

public ActionResult Index()
{
    Logger logger = new Logger(LogType.File);

    //Id is retrieved from the server
    logger.TraceStart("User entered to homepage, user id: " +
User.Identity.GetUserId());

    return View();
}
```

C# güvensiz kod örneğinde kullanıcı hesabın kullanıcı id'si istemciden çekilerek log kaydının sonuna ekleniyor ve log dosyasına yazdırılıyor. Bu durumda istemci taraftaki kullanıcı id kurcalanarak (tab veya line feed karakterleri girilerek) log enjeksiyonu gerçekleştirilebilir. C# güvenli kod örneğinde ise kullanıcı hesabın kullanıcı id'si halihazırda sunucu taraftan elde edilebiliyor olduğundan sunucu taraftaki değişkenden çekiliyor ve log kaydının sonuna

eklenerek log dosyasına yazdırılıyor. Bu güvenli örnekte eğer imkan varsa değerleri istemci taraflı girdilerden almak yerine sunucu taraftan alarak güvenliğin sağlanabileceği örneklenmiştir. Değerler sunucu taraftan elde edildiğinden güvenlidir ve istemci kurcalayamayacağından log enjeksiyonu yaşanmayacaktır.

PHP - Güvensiz Kod Örneği (1):

```php
<?php

// Güvensiz Örnek (1)

// TR: Ayıklama Uygulanmadan Kullanıcı Adının Syslog Kaydına Eklenmesi
// EN: Concatenate Username In SysLog Without Sanitization

syslog(LOG_EMERG, "Attempting to Login with User: " . $_POST['username']);
```

PHP - Güvensiz Kod Örneği (2):

```php
<?php

// Güvensiz Örnek (2)

// TR: Ayıklama Uygulanmadan Kullanıcı Adının Hata Log Kaydına Eklenmesi
// EN: Concatenate Username In Error Log Without Sanitization

error_log("Attempting to Login with User: " . $_POST['username']);
```

PHP - Güvenli Kod Örneği:

```php
<?php

// Güvenli Örnek

// TR: Kullanıcı Girdisinin Log'a Yazdırılmadan
//     Önce Satır Atlatmalardan Temizlenmesi
// EN: Removing Line Breaks in User Input Prior to Log Write

$username = str_replace(["\r\n", "\r", "\n"], " ", $_POST['username']);
error_log("Attempting to Login with User: " . $username);
```

PHP güvensiz kod bloğu (1)'de istemci taraftan çekilen kullanıcı adı syslog log kaydına olduğu gibi satır atlatma (line feed) karakterlerinden ayıklanmadan eklenmektedir ve log kaydı yazdırılmaktadır. Bu durum kullanıcı adı parametresi kurcalanarak log enjeksiyonuna dönüşebilir. Aynı şekilde PHP güvensiz kod bloğu (2)'de de istemci taraftan çekilen kullanıcı adı bu sefer hata log kaydına olduğu gibi satır atlatma (line feed) karakterlerinden ayıklanmadan eklenmektedir ve log kaydı yazdırılmaktadır. Bu durumda da kullanıcı adı parametresi kurcalanarak log enjeksiyonuna dönüşebilir. PHP güvenli kod bloğu örneğinde ise istemci taraftan çekilen kullanıcı adı girdisi satır atlatma karakterlerinden ayıklanarak hata log kaydına eklenmektedir ve log dosyasına yazdırılmaktadır. Bu durum olası kullanıcı ad parametresinin kurcalandığı durumlarda log enjeksiyonunun yaşanmasını önleyecektir.

Vb.NET - Güvensiz Kod :

```
' GÜVENSİZ ÖRNEK

' TR: Kullanıcı Girdisinin Log Dosyasına Yazdırılması - MVC 5
' EN: Writing User Input Into Log Files - MVC 5

Function Index() As ActionResult

    'Id paramatresi query-string'den alınıyor, dolayısıyla kurcalanabilir
    Dim id As String = Request.QueryString("id")
    My.Application.Log.WriteEntry("User entered to homepage, user id: " + id)
    Return View()

End Function
```

Vb.NET - Güvenli Kod:

```
' GÜVENLİ ÖRNEK

' TR: Sunucu Taraftan Elde Edilen Kullanıcı Id'sinin
'     Log Dosyasına Yazdırılması - MVC 5
' EN: Writing User Id From Server Into Log File - MVC 5

Function Index() As ActionResult

    'Id sunucu taraftan elde ediliyor.
    My.Application.Log.WriteEntry("User entered to homepage, user id: " +
User.Identity.GetUserId())
    Return View()

End Function
```

Vb.NET güvensiz kod örneğinde url adresteki parametreden kullanıcı girdisi çekilmektedir ve hiçbir ayıklamaya tabi tutulmadan olduğu gibi log kaydının sonuna eklenerek log dosyasına yazdırılmaktadır. Bu durum log enjeksiyonu riskini doğuracaktır. Vb.NET güvenli kod örneğinde ise url adresten gelen parametredeki değer sunucu taraftan da elde edilebildiğinden sunucu taraftan elde edilmiştir ve bu sayede log enjeksiyonu ihtimali ortadan kalkmıştır. Çünkü sunucu taraftan elde edilen değer güvenlidir. İstemci kurcalayamaz.

Log Forging (Keyfi log ekleme / log enjekte etme) açıklığı ile kötü niyetli kullanıcılar;

- Uygulama sunucusundaki log dosyalarına log girişlerini taklit edecek sahte / yeni log girişleri ekleyebilirler. Bu sayede;
    - Bozuk hale getirilmiş log dosyaları ile log istatistiklerini çarpıtarak kendisinin veya bir başka saldırganın izlerini kapatabilirler.
    - Girişleri taklit edilmiş log dosyaları ile ayrıca kötü bir eylemin sorumlusu olarak bir başka kişiyi gösterebilirler.
- Uygulama sunucusundaki zararlı log event'lerinin geliştiricilerce web tarayıcıda görüntülendiği durum ele alındığında log dosyalarına xss payload'u girerek XSS saldırıları düzenleyebilirler.
- Uygulama sunucusunda komut çalıştırabilen log parser'lar (örn; PHP log parser'lar) kullanıldığı durum ele alındığında komut enjeksiyonu saldırıları düzenleyebilirler.

Kurum uygulamada Log Forging (CWE-117) açıklığı tespit edilmiştir:

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

Şekil XXX. Girdi Doğrulaması Yapılmamış Log Alma

Açıklığın Önlemi:

Tavsiyeler şu şekildedir:

- Tüm girdiler kaynağına bakılmaksızın denetlenmelidir. Denetleme sadece belirtilen yapıya uygun verileri kabul edecek şekilde beyaz liste temelli olmalıdır. Belirtilen yapıların reddedildiği siyah liste temelli olmamalıdır.
- Girdilerin denetlenmesi encode'lamanın yerine geçecek bir önlem değildir. Tüm girdiler ayrıca encode'lanmalıdır.
- Güvenli log alma mekanizmaları kullanılmalıdır.

Referanslar:

1. https://knowledge-base.secureflag.com/vulnerabilities/inadequate_input_validation/log_injection_vulnerability.html
2. https://kelumkp.medium.com/what-is-log-forging-5cfdf045979a
3. https://cwe.mitre.org/data/definitions/117
4. https://owasp.org/www-community/attacks/Log_Injection
5.