1.1.1 Yansıtılmış Siteler Arası Betik Çalıştırma (Reflected XSS All Clients)

Açıklık Önem Derecesi: Yüksek

Açıklığın Etkisi: Kullanıcı hesaplarının çalınması, kullanıcıların dos saldırılarında bot olarak kullanılması, kullanıcıların oltalama saldırılarına maruz kalması

Açıklığın Barındıran Dosyalar/Satırlar:

Proje Dosyası/Dosya Adı	Satır Numarası

Açıklığın Açıklaması:

XSS web uygulamalarda kullanıcının uygulama girdi noktasına girmesi umulan değer yerine güvensiz bir veriyi (Javascript kodunu) girmesiyle girilen verinin yanıt olarak dönen html sayfanın bir parçası olarak geri dönmesi saldırılarına denir. XSS saldırılarında saldırgan Javascript dilinin verdiği tüm olanakları hedef web uygulama ve kullanıcısı üzerinde deneyebilir.

Reflected (Yansıyan) XSS ise kullanıcının uygulama girdi noktasına girdiği güvensiz girdinin web sunucuya kadar gittiği ve veri tabanına kaydolmadan web tarayıcıya geri döndüğü / yansıtıldığı XSS türüne denir. Reflected XSS'de saldırganlar mağdurlara bir e-posta mesajı veya başka bir web sitesi (forum / sosyal medya v.b.) üzerinden saldırı mesajını iletirler. Kullanıcı kötü niyetli bir bağlantıyı tıklamak veya kötü amaçlı bir web siteye göz atmak üzere kandırıldığında enjekte edilen kod güvenlik açığından etkilenen web sitesi sunucusuna gidip gelerek web tarayıcıda çalışır. Bu saldırı neticesinde kullanıcıların hesapları ele geçirilebilir, kullanıcılar çeşitli adreslere yapılan servis dışı bırakma saldırılarında bot olarak kullanılabilir veya kullanıcılar bir oltalama web sitesine yönlendirilebilirler. Saldırganlar Reflected XSS açıklığını URL'i modifiye ederek, kullanıcı girdisinde zararlı veri göndererek veya başka http talep girdi alanlarında zararlı veri göndererek sömürebilirler.

Reflected XSS saldırısını somutlaştırmak için C#, Java, PHP ve Ruby örneklerine yer verilmiştir:

C# - Güvensiz Hal (I):

```
// Outputting Unsanitized User Input with Html.Raw()

@Html.Raw("<h1>Welcome, " + Request.Params["var"] + "</h1>")
```

C# - Güvensiz Hal (II):

```
// Retrieve User Address from DB and Present It in View inside
// a Javascript Context using Razor, Which Does Not Properly
// Encode Certain Meta-Characters in the JS Context

<script>alert(`Welcome, @Request.Params["var"]`);</script>
```

C# - Güvenli Hal:

```
// Outputting User Input to HTML Using Razor, Which Properly Encodes HTML Tags

<h1>Welcome, @Request.Params["var"]</h1>
```

C# kodu yorumlayacak olursak “C# - Güvensiz Hal (I)” kod bloğunda GET var parametresi denetime tabi tutulmadan dönen html yanıtta gönderiliyor. Bu xss açıklığını meydana getirecektir.

“C# - Güvensiz Hal (II)” kod bloğunda GET var parametresi Razor tarafından html için özel anlam ifade eden karakterlerden ayıklanmaktadır (encode’lanmaktadır), fakat GET var parametresinin bulunduğu context (bağlam) javascript context’i (bağlamı) olduğundan GET var parametresi şu payload’u aldığında

```
`); alert(1); //
```

güvenliği atlatır ve alert(1) popup'ı gelerek xss açıklığı meydana gelir.

“C# - Güvenli Hal” kod bloğunda GET var parametresi Razor tarafından html için özel anlam ifade eden karakterlerden ayıklanmaktadır (encode’lanmaktadır). Parametrenin bulunduğu context (bağlam) ise html context olduğundan encode’lama ilgili context’e (bağlama) uygundur. Dolayısıyla xss açıklığı meydana gelmeyecektir.

Java - Güvensiz Hal:

```
// Returning Data To Clients Without Encoding

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

    response.setContentType("text/html;charset=UTF-8");

    PrintWriter out = response.getWriter();
    String loc = request.getParameter("location");

    out.println("<h1> Location: " + loc + "<h1>");
}
```

Java - Güvenli Hal:

```
// Returning Data to Clients After Encoding The User Input
// Using HtmlEscapers by Google Guava

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

    response.setContentType("text/html;charset=UTF-8");

    PrintWriter out = response.getWriter();
    String loc = request.getParameter("location");
    String escapedLocation = HtmlEscapers.htmlEscaper().escape(loc);

    out.println("<h1> Location: " + escapedLocation + "<h1>");
}
```

Java kodu yorumlayacak olursak “Java - Güvensiz Hal” kod bloğunda GET location parametresi loc nesnesine çekiliyor ve loc nesnesi denetime tabi tutulmadan dönen html yanıtta gönderiliyor. Bu denetimsizlik xss açıklığını meydana getirecektir. “Java - Güvenli Hal” kod bloğunda ise GET location parametresi yine loc nesnesine çekiliyor, fakat loc nesnesi yanıt olarak html yanıtta döndürülmeden önce htmlEscaper() ve escape() metotları ile encode’lanıyor (düzenleniyor). Bu düzenleme sonrası yanıt olarak döndürülecek html yanıtta gönderiliyor. Dolayısıyla bu denetim uygulaması xss açıklığını önleyecektir.

PHP - Güvensiz Hal (I):

```
// Outputting Unsanitized Inputs into HTML Results in XSS

if (isset($_GET['name'])) {
    echo "<h1>Welcome," . $_GET['name'] . "!</h1>";
}
```

PHP kodu yorumlayacak olursak “PHP - Güvensiz hal (I)” kod bloğunda GET name parametresi hiçbir denetime tabi olmadan doğrudan html yanıtta döndürüldüğünden xss meydana gelecektir.

PHP - Güvensiz Hal (II):

```
// Insecure Use of "htmlspecialchars" Without a Secure Flag

if (isset($_GET['name'])) {

    echo "<script> var name = '" . htmlspecialchars($_GET['name']) .
"';</script>\r\n";

}
```

“PHP - Güvensiz hal (II)” kod bloğunda GET name parametresi html encode’lama yapan (html özel karakterlerini etkisiz hale getiren) htmlspecialchars() fonksiyonuna girmektedir. Fakat htmlspecialchars() fonksiyonunda güvenli herhangi bir flag (bayrak) kullanılmamaktadır. Bu haliyle htmlspecialchars() fonksiyonu sadece çift tırnak karakterini filtrelemektedir, fakat tek tırnak karakterlerini filtrelememektedir. Dolayısıyla mevcut javascript context’inde (bağlamda) GET name parametresine

```
'; alert(1); //
```

payload’u girildiğinde htmlspecialchars() fonksiyonu payload’daki tek tırnağı filtrelemeyeceğinden enjeksiyon başarılı olacaktır ve alert(1) popup’ı gelecektir. Yani xss açıklığı meydana gelecektir.

PHP - Güvensiz Hal (III):

```
// Insecure Use of "htmlspecialchars" With "ENT_QUOTES" Flag

if (isset($_GET['name'])) {

    echo "<script> var name = `" . htmlspecialchars($_GET['name'], ENT_QUOTES,
'UTF-8') . "`;</script>";

}
```

"PHP - Güvensiz hal (II)" kod bloğunda GET name parametresi html encode'lama yapan (html özel karakterlerini etkisiz hale getiren) htmlspecialchars() fonksiyonuna girmektedir. htmlspecialchars() fonksiyonu güvenli ENT_QUOTES bayrağını kullanmaktadır. ENT_QUOTES bayrağı çift tırnak (") ve tek tırnak (') karakterlerini encode'lar (etkisiz hale getirir). Fakat htmlspecialchars() fonksiyonu mevcut javascript context'indeki (bağlamındaki) konumu gereği ENT_QUOTES bayrağına rağmen güvensiz durumdadır. Çünkü mevcut javascript context'inde (bağlamında) GET name parametresine

```
`; alert(1); //
```

backtick'li payload girildiğinde htmlspecialchars() fonksiyonunun ENT_QUOTES bayrağı payload'daki backtick'i filtrelemeyecektir, enjeksiyon başarılı olacaktır ve alert(1) popup'ı gelecektir. Yani xss açıklığı meydana gelecektir.

PHP - Güvenli Hal (I):

```
// Secure Use of "htmlspecialchars" With "ENT_QUOTES" Flag

if (isset($_GET['name'])) {

    //ENT_QUOTES flag sanitizes apostrophe

    echo "<script> var name = '" . htmlspecialchars($_GET['name'], ENT_QUOTES,
'UTF-8') . "';</script>";

}
```

"PHP - Güvenli Hal (I)" kod bloğunda GET name parametresi html encode'lama yapan (html özel karakterlerini etkisiz hale getiren) htmlspecialchars() fonksiyonuna girmektedir. htmlspecialchars() fonksiyonunda güvenli ENT_QUOTES bayrağı kullanılmaktadır. Tekrar edilecek olursa ENT_QUOTES bayrağı çift tırnak (") ve tek

tırnak (') karakterlerini encode'lar (etkisiz hale getirir). Bu örnekte htmlspecialchars() güvenli durumdadır. Çünkü htmlspecialchars() fonksiyonunun ENT_QUOTES bayrağı mevcut javascript context'iyle (bağlamıyla) uyumludur. Mevcut javascript context'inde (bağlamında) GET name parametresi tek tırnak arasında kullanıldığından ve tek tırnaklar filtrelendiğinden xss için ENT_QUOTES encode'lamasını atlatacak uygun bir payload bulunamaz. Dolayısıyla xss meydana gelmez.

PHP - Güvensiz Hal (IV):

```
// Insecure Use of "htmlspecialchars" With "ENT_COMPAT" Flag

if (isset($_GET['name'])) {

    echo "<script> var name = '" . htmlspecialchars($_GET['name'], ENT_COMPAT,
'UTF-8') . "';</script>";

}
```

"PHP - Güvensiz hal (IV)" kod bloğunda GET name parametresi html encode'lama yapan (html özel karakterlerini etkisiz hale getiren) htmlspecialchars() fonksiyonuna girmektedir. htmlspecialchars() fonksiyonunda ENT_COMPAT adlı bir bayrak kullanılmaktadır. Bu bayrak fonksiyonun varsayılan bayrağıdır. Sadece çift tırnakları (") encode'lar (etkisizleştirir). htmlspecialchars() fonksiyonu mevcut javascript context'inde (bağlamında) bu bayrak ile güvensiz haldedir. Çünkü mevcut javascript context'i (bağlamı) ENT_COMPAT bayrağını bypass'lama imkanı sunmaktadır. Mevcut javascript context'inde (bağlamında) GET name parametresine

```
'; alert(1); //
```

tek tırnaklı payload girildiğinde htmlspecialchars() fonksiyonunun ENT_COMPAT bayrağı tek tırnak karakterini filtrelemeyeceğinden enjeksiyon başarılı olacaktır ve alert(1) popup'ı gelecektir. Yani yine xss meydana gelecektir.

PHP - Güvenli Hal (II):

```
// Secure Use of "htmlspecialchars" With "ENT_COMPAT" Flag

if (isset($_GET['name'])) {

    //ENT_COMPAT flag sanitize quotation marks

    echo "<script> var name = \"" . htmlspecialchars($_GET['name'],
ENT_COMPAT, 'UTF-8') . "\";</script>";

}
```

“PHP - Güvenli Hal (II)” kod bloğunda GET name parametresi html encode’lama yapan (html özel karakterlerini etkisiz hale getiren) htmlspecialchars() fonksiyonuna girmektedir. htmlspecialchars() fonksiyonunda ENT_COMPAT adlı bayrak kullanılmaktadır. Bu örnekte htmlspecialchars() foksiyonu mevcut javascript context’inde (bağlamında) güvenli durumdadır. Çünkü ENT_COMPAT çift tırnak (") karakterini filtreler ve mevcut javascript context’inde (bağlamında) GET name parametresi çift tırnak arasında eklenir durumdadır. Dolayısıyla html özel karakteri çift tırnak etkisizleştirildiğinden bu bölgeye enjekte işlemi için uygun xss payload’u bulunamayacaktır ve xss meydana gelmeyecektir.

Ruby - Güvensiz Hal:

```
# Reflecting Request Data To Clients in "Raw" Output

# Controller:
class PagesController < ApplicationController
    def home
        @greeting = "<h1>Welcome, new user!</h1>"
        if params[:name]
            @greeting = "<h1>Welcome " + params[:name] + "!</h1>"
        end
    end
end

# View Fragment:
<%=raw @greeting %>
```

Ruby - Güvenli Hal:

```
# Relying on Rails Output to HTML Encode Outputs

# Controller:
class PagesController < ApplicationController
    def home
        @greeting = "<h1>Welcome, new user!</h1>"
        if params[:name]
            @greeting = params[:name]
        end
    end
end

# View Fragment:
<h1>Welcome <%=@greeting %>!</h1>
```

Ruby on Rails framework'üne ait ruby kodu yorumlayacak olursak "Ruby - Güvensiz Hal" kod bloğunda Controller'da @greeting değişkenine bir string atanmaktadır. Sonra bir if kontrolü ile eğer istemciden name parametresi gönderilmişse @greeting değişkenine name parametresi değeri atanmaktadır. İstemciden name parametresi geldiği durumu ele alacak olursak @greeting değişkenine name parametresi değeri atanacaktır ve view katmanında @greeting değişkeni html yanıt olarak döndürülecektir. @greeting değişkeni ham şekilde (olduğu gibi) html yanıtta gönderildiğinden xss açıklığı meydana gelecektir. "Ruby - Güvenli Hal" kod bloğunda ise @greetings değişkeni html encode (kodlama) edilmiş halde html yanıt olarak döndürüldüğünden xss açıklığı meydana gelmeyecektir.

Kurum web uygulamada XSS açıklığı tespit edilmiştir:

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

Açıklığın Önlemi:

- Kaynağa bakılmaksızın tüm istemci girdileri çıktıya gömülmeden önce encode'lanmalıdır (kodlanmalıdır / etkisizleştirilmelidir).
- Encode'lama (kodlama / etkisizleştirme) işlemi içeriğe / bağlama (context'e) uygun olmalıdır. Örneğin HTML context'i (bağlamı) için HTML encode'laması (kodlaması), sunucu tarafından üretilen Javascript context'i (bağlamı) için Javascript encode'laması (kodlaması) gibi. Eğer içeriğe / context'e (bağlama) uygun encode'lama (kodlama) yapılmazsa encode'lama arzu edilen XSS önlemini sunamaz.

- OWASP ESAPI kütüphanelerinin kullanılması ya da ASP.NET'in eski sürümleri için AntiXSS kütüphanesinin kullanılması önerilmektedir.

  Örneğin eski bir ASP.NET web uygulaması HttpResponse oluşturmak için "Referer" alan dizesini kullanıyor olsun.

```
public class ReflectedXssAllClients {
    public static void foo(HttpRequest Request, HttpResponse Response)
        {
                string Referer = Request.QueryString["Referer"];
                Response.BinaryWrite(Referer);
        }
}
```

  Bu dizeyi güvenli hale getirmek için AntiXSS kütüphanesi kullanılır.

```
public class ReflectedXssAllClientsFixed
{
        public static void foo(HttpRequest Request, HttpResponse Response,
AntiXss.AntiXssEncoder encoder)
        {
                string Referer = Request.QueryString["Referer"];
                Response.BinaryWrite(encoder.HtmlEncode(Referer, true));
        }
}
```

  Böylece bu kod bloğunda XSS önlenir.

- XSS açıklıkları çıktığında derinlikli defans (defense-in-depth) prensibi gereği çerez çalınmasını önlemek adına Set-Cookie http yanıt başlığı için “httpOnly” bayrağının yapılandırma dosyalarında “true” olarak tanımlanması önerilmektedir.

- Content Security Policy (CSP) http yanıt başlığı eklenmelidir ve whitelist denetimi olarak sadece uygulamaya has kaynaklar (js dosyaları) açıkça çalıştırılabilir şekilde tanımlanmalıdır.

- Encoding (kodlama / etkisizleştirme) yanında ilaveten whitelist olarak girdi doğrulama uygulanmalıdır. Girdi doğrulama beklenen veriler geliyorsa işleme sokulması, beklenen veriler gelmiyorsa işleme sokulmaması şeklinde whitelist mantığıyla çalışmalıdır. Girdi doğrulama beklenen veriler geliyorsa işleme sokulmaması,

beklenen veriler gelmiyorsa işleme sokulması şeklinde blacklist mantığıyla çalışmamalıdır.

Referanslar

1. https://www.owasp.org/index.php/Cross-site_Scripting_(XSS)
2. https://www.w3schools.com/php/func_string_htmlentities.asp
3. https://www.php.net/manual/en/function.htmlentities.php
4. https://cwe.mitre.org/data/definitions/79.html