1.1.1 Doğrulanmamış Yol Girdisi (Input Path Not Canonicalized) (CWE-73) (CWE-36) (CWE-23) (CWE-22)

Açıklık Önem Derecesi: Orta

Açıklığın Etkisi: Hassas Dosyaların Çalınması, SSRF saldırısı düzenlenmesi

Açıklığın Barındıran Dosyalar/Satırlar:

Proje Dosyası/Dosya Adı	Satır Numarası

Açıklığın Açıklaması:

Uygulamalar tasarımı gereği bazen kullanıcı girdisi ile bir dosya yolu bilgisi alabilmektedirler. Uygulamalar bu şekilde uygulama sunucusunun yerel diskindeki dosyalara erişim imkanı sunabilmektedirler. Saldırganlar bu tarz web uygulamalarda bu işleyişleri eğer girdiler kontrol edilmiyorsa kötü yönde kullanabilirler. Saldırganlar özel hazırlanmış dosya yolları uygulamaya girerek uygulamanın kullanıcılarına hizmet olarak sunduğu uygulama sunucusunun yerel diskindeki dosyalarla birlikte ayrıca uygulamanın kullanıcılarına hizmet olarak sunmadığı uygulama sunucusunun yerel diskindeki dosyalara da erişim imkanı elde edebilirler. Bu ise geliştiricileri ciddi bilgi ifşalarıyla veya teknik problemlerle başbaşa bırakabilir.

Saldırganlar uygulamanın kullanması adına keyfi (rastgele) dosya yolu girdisi girerek;

- Uygulama sunucusunun yerel diskindeki hassas dosyaları çalabilirler (örn; konfigürasyon dosyalarını veya sistem dosyalarını)
- Uygulama sunucusunun yerel diskindeki dosyaların üzerine yazma yapabilirler (örn; program binary'leri, konfigürasyon dosyaları veya sistem dosyaları v.b. üzerine)
- Uygulama sunucusunun yerel diskindeki kritik dosyaları silebilirler ve böylece DoS saldırısı gerçekleştirebilirler.

Uygulamalar kullanıcı girdisi olarak dosya yolu aldıklarında ve bu girdi denetlenmeden / filtrelenmeden uygulamada kullanıldığında buna "Doğrulanmamış Yol Girdisi (CWE-73) (CWE-36)" açıklığı adı verilir.

Açıklığı somutlaştırmak adına bazı teknolojilerde örnekler verilmiştir:

Java - Güvensiz Kod Bloğu (1):

```
// TR: "filename" Parametresinde Göreceli Yol Gezinme
// EN: Relative Path Traversal in "filename" Parameter

private String getFileContents(HttpServletRequest request) throws
ServletException, FileNotFoundException, IOException {
    String filename = request.getParameter("filename");
    Path path = Paths.get(SERVED_FILES_DIR + filename);
    byte[] fileContentBytes = Files.readAllBytes(path);
    String fileContents = new String(fileContentBytes,
FILE_CONTENT_ENCODING_STRING);
    return fileContents;
}
```

Java - Güvensiz Kod Bloğu (2):

```
// TR: "filename" Parametresinde Mutlak Yol Gezinme
// EN: Absolute Path Traversal in "filename" Parameter

private String getFileContents(HttpServletRequest request) throws
ServletException, FileNotFoundException, IOException {
    String filename = request.getParameter("filename");
    Path path = Paths.get(filename);
    byte[] fileContentBytes = Files.readAllBytes(path);
    String fileContents = new String(fileContentBytes,
FILE_CONTENT_ENCODING_STRING);
    return fileContents;
}
```

Java - Güvenli Kod Bloğu:

```java
// TR: "filename" Parametresinin Filtrelenmesi Üzerinden Giderilen Yol Gezinme
// EN: Path Traversal Mitigated via Sanitization of Path Variable

private static String sanitizePathTraversal(String filename) {
    Path p = Paths.get(filename);
    return p.getFileName().toString();
}

private String getFileContents_fixed(HttpServletRequest request) throws
ServletException, FileNotFoundException, IOException {
    String filename = sanitizePathTraversal(request.getParameter("filename"));
// Ensures access only to files in a given folder, no traversal
    Path path = Paths.get(SERVED_FILES_DIR + filename);
    byte[] fileContentBytes = Files.readAllBytes(path);
    String fileContents = new String(fileContentBytes,
FILE_CONTENT_ENCODING_STRING);
    return fileContents;
}
```

Java güvensiz kod bloğu (1)'de istemci taraflı "filename" parametresi ile bir dosya yolu alınmaktadır. Ardından sunucu taraflı SERVED_FILES_DIR değişkenin tuttuğu bir dosya yolunun sonuna istemci girdisi (göreceli dosya yolu) eklenmektedir. Bu şekilde oluşan nihai dosya yolundaki dosyanın içeriği okuması yapılmaktadır ve içerik döndürülmektedir. Bu örnek bir "doğrulanmamış dosya yolu" zafiyeti, bu örnek özelinde daha spesifik adıyla "göreceli yol gezinme" zafiyeti örneğidir. İstemci taraflı girdi "filename" denetlenmediği için ve olduğu gibi dosya okuması işleyişinde kullanıldığı için suistimale açıktır.

Java güvensiz kod bloğu (2)'de istemci taraflı "filename" parametresi ile yine bir dosya yolu alınmaktadır. Ardından bu dosya yolu bütün haliyle okunacak dosyanın belirlenmesi ve okunması adına kullanılmaktadır. Bu örnek bir "doğrulanmamış dosya yolu" zafiyeti, bu örnek özelinde daha spesifik adıyla "mutlak yol gezinme" zafiyeti örneğidir. İstemci taraflı girdi "filename" denetlenmediği için ve olduğu gibi dosya okuması işleyişinde kullanıldığı için suistimale açıktır.

Java güvenli kod bloğunda ise istemci taraflı "filename" parametresi ile bir dosya yolu alınmaktadır, fakat bu sefer bu dosya yolu dosya okumasında kullanılmadan önce filtreleme işleminden (sanitizePathTraversal() metodundan) geçmektedir. Bu sayede istemci tarafın gireceği kural dışı karakterler elimine olacaktır ve suistimal yolu kapatılmış olacaktır.

C# - Güvensiz Kod Bloğu (1):

```
// TR: StreamReader'da Absolute Path Traversal Açıklığı
// EN: Absolute Path Traversal in StreamReader

public string GetFile()
{
  string filename = Request.Params["filename"];
  string path = filename;
  StreamReader sr = new StreamReader(path);
  return sr.ReadToEnd();
}
```

C# - Güvensiz Kod Bloğu (2):

```
// TR: StreamReader'da Relative Path Traversal Açıklığı
// EN: Relative Path Traversal in StreamReader

public string GetFile()
{
  string filename = Request.Params["filename"];
  string path = DIRECTORY_TO_SERVE + filename;
  StreamReader sr = new StreamReader(path);
  return sr.ReadToEnd();
}
```

C# - Güvenli Kod Bloğu:

```
// TR: GetFileName kullanarak olası ön dizin isimlerini
//     Kırpma ile Giderilmiş Path Traversal Açıklığı
// EN: Path Traversal Mitigated by Trimming Potential
//     Prefix Directories, using GetFileName

public string GetFileByNumber()
{
  string filename = Path.GetFileName(Request.Params["filename"]);
  string path = DIRECTORY_TO_SERVE + filename;
  StreamReader sr = new StreamReader(path);
  return sr.ReadToEnd();
}
```

C# güvensiz kod bloğu (1)'de istemci taraflı "filename" parametresi ile bir dosya yolu alınmaktadır. Ardından StreamReader ile dosyanın içeriği okuması yapılmaktadır. Bu örnek bir "doğrulanmamış dosya yolu" zafiyeti, bu örnek özelinde daha spesifik adıyla "mutlak yol gezinme" zafiyeti örneğidir. İstemci taraflı girdi "filename" denetlenmediği için ve olduğu gibi dosya okuması işleyişinde kullanıldığı için suistimale açıktır.

C# güvensiz kod bloğu (2)'de istemci taraflı "filename" parametresi ile yine bir dosya yolu alınmaktadır. Ardından sunucu taraflı DIRECTORY_TO_SERVE değişkenin tuttuğu bir dosya yolunun sonuna istemci girdisi (göreceli dosya yolu) eklenmektedir. Bu şekilde oluşan nihai dosya yolundaki dosyanın içeriği okuması yapılmaktadır. Bu örnek bir "doğrulanmamış dosya yolu" zafiyeti, bu örnek özelinde daha spesifik adıyla "göreceli yol gezinme" zafiyeti örneğidir. İstemci taraflı girdi "filename" denetlenmediği için ve olduğu gibi dosya okuması işleyişinde kullanıldığı için suistimale açıktır.

C# güvenli kod bloğunda ise istemci taraflı "filename" parametresi ile bir dosya yolu alınmaktadır, fakat bu sefer bu dosya yolu dosya okumasında kullanılmadan önce illegal karakterlerin kırpıldığı işlemden (Path.GetFileName() metodundan) geçmektedir. Bu sayede istemci tarafın gireceği kural dışı karakterler elimine olacaktır ve suistimal yolu kapatılmış olacaktır.

PHP - Güvensiz Kod Bloğu (1):

```php
<?php

    // TR: "filename" Parametresinde Absolute Path Traversal Açıklığı
    // EN: Absolute Path Traversal in "filename" Parameter

    if (isset($_GET['filename'])) {
        $filename = $_GET['filename'];
        if (is_readable($filename)) {
            $fp = fopen($filename, 'rb');
            fpassthru ($fp);
        }
        else {
            // return 404
        }
    } else {
        // return 404
    }

?>
```

PHP - Güvensiz Kod Bloğu (2):

```php
<?php

    // TR: "filename" Parametresinde Relative Path Traversal Açıklığı
    // EN: Relative Path Traversal in "filename" Parameter

    if (isset($_GET['filename'])) {
        $filename = "public_files/".$_GET['filename'];
        if (is_readable($filename)) {
            $fp = fopen($filename, 'rb');
            fpassthru ($fp);
        }
        else {
            // return 404
        }
    } else {
        // return 404
    }

?>
```

PHP - Güvensiz Kod Bloğu (3):

```php
<?php

    // TR: Olası SSRF Saldırısına Yol Açan "filename" Parametresinde
    //     Absolute Path Traversal Açıklığı
    // EN: Absolute Path Traversal in "filename" Parameter
    //     Leading to Possible SSRF

    /**
     * TR: Bir HTTP GET parametresi olarak Sağlanmış FileName
     * EN: Filename supplied as an HTTP GET request parameter
    **/
    if(isset($_GET["filename"])) {

        // TR: Burası olası bir taint sink'tir. Çünkü kullanıcı
        //     "http://" karakterleri filename parametresine gi-
        //     rebilir ki bu SSRF'e yol açar.
        // EN: Possible sink as the user can supply a filename
        //     with "http://", causing SSRF

        $content = file_get_contents($_GET["filename"]);

        // display $content

    }

?>
```

PHP - Güvenli Kod Bloğu:

```php
<?php

    // TR: Basename Kullanılarak Giderilmiş Path Traversal Açıklığı
    // EN: Path Traversal Mitigated by Utilizing Basename

    if (isset($_GET['filename'])) {
        $filename = "public_files/".basename($_GET['filename']);
        if (is_readable($filename)) {
            $fp = fopen($filename, 'rb');
            fpassthru ($fp);
        }
        else {
            // return 404
        }
    } else {
        // return 404
    }
?>
```

PHP güvensiz kod bloğu (1)'de istemci taraflı "filename" parametresi ile bir dosya yolu alınmaktadır. Ardından fopen() ile dosyanın içeriği okuması yapılmaktadır. Bu örnek bir "doğrulanmamış dosya yolu" zafiyeti, bu örnek özelinde daha spesifik adıyla "mutlak yol gezinme" zafiyeti örneğidir. İstemci taraflı girdi "filename" denetlenmediği için ve olduğu gibi dosya okuması işleyişinde kullanıldığı için suistimale açıktır.

PHP güvensiz kod bloğu (2)'de istemci taraflı "filename" parametresi ile yine bir dosya yolu alınmaktadır. Ardından sunucu taraflı "public_files/" dosya yolunun sonuna istemci girdisi (göreceli dosya yolu) eklenmektedir. Bu şekilde oluşan nihai dosya yolundaki dosyanın içeriği okuması yapılmaktadır. Bu örnek bir "doğrulanmamış dosya yolu" zafiyeti, bu örnek özelinde daha spesifik adıyla "göreceli yol gezinme" zafiyeti örneğidir. İstemci taraflı girdi "filename" denetlenmediği için ve olduğu gibi dosya okuması işleyişinde kullanıldığı için suistimale açıktır.

PHP güvensiz kod bloğu (3)'de istemci taraflı "filename" parametresi ile tekrardan bir dosya yolu alınmaktadır. Ardından file_get_contents() ile dosyanın içeriği okuması yapılmaktadır. file_get_contents() metodu argüman olarak yerel dosya yolları alabileceği gibi uzak dosya yolları (http://example.com v.b.) da alabilmektedir. Dolayısıyla istemci taraflı "filename" parametresine http:// ön eki girilebilir ve SSRF saldırısı düzenlenebilir. Bu örnek bir "doğrulanmamış dosya yolu" zafiyetidir ve suistimale açıktır. Diğer doğrulanmamış dosya yolu zafiyetlerine kıyasla SSRF saldırısına da ilaveten olanak vermektedir.

PHP güvenli kod bloğunda ise istemci taraflı “filename” parametresi ile bir dosya yolu alınmaktadır, fakat bu sefer bu dosya yolu dosya okumasında kullanılmadan önce illegal karakterlerin kırpıldığı işlemden (basename() metodundan) geçmektedir. Bu sayede istemci tarafın gireceği kural dışı karakterler elimine olacaktır ve suistimal yolu kapatılmış olacaktır.

Ruby - Güvensiz Kod Bloğu (1):

```ruby
# TR: "filename" Parametresinde Absolute Path Traversal Açıklığı
# EN: Absolute Path Traversal in Parameter "filename"
def getFile
    if params[:filename]
        filename = params[:filename]
        if File.file?(filename) and File.readable?(filename)
            send_file filename
        end
    end
end
```

Ruby - Güvensiz Kod Bloğu (2):

```ruby
# TR: "filename" Parametresinde Relative Path Traversal Açıklığı
# EN: Relative Path Traversal in Parameter "filename"
def getFile
    if params[:filename]
        filename = "public_files/" + params[:filename]
        if File.file?(filename) and File.readable?(filename)
            send_file filename
        end
    end
end
```

Ruby - Güvenli Kod Bloğu:

```ruby
# TR: Olası Dizin Ön Eklerini Kırpmak için File.basename Kullanarak
#     Giderilmiş Path Traversal
# EN: Mitigated Path Traversal using File.basename to Trim Potential
#     Directory Prefix
def getFile_fixed
    if params[:filename]
        filename = "public_files/" + File.basename(params[:filename])
        if File.file?(filename) and File.readable?(filename)
            send_file filename
        end
    end
end
```

Ruby güvensiz kod bloğu (1)'de istemci taraflı "filename" parametresi params[:filename] ile bir dosya yolu filename nesnesine alınmaktadır. Ardından send_file ile dosyanın içeriği okuması yapılmaktadır ve istemciye döndürülmektedir. Bu örnek bir "doğrulanmamış dosya yolu" zafiyeti, bu örnek özelinde daha spesifik adıyla "mutlak yol gezinme" zafiyeti örneğidir. İstemci taraflı girdi "filename" denetlenmediği için ve olduğu gibi dosya okuması işleyişinde kullanıldığı için suistimale açıktır.

Ruby güvensiz kod bloğu (2)'de istemci taraflı "filename" parametresi params[:filename] ile yine bir dosya yolu filename nesnesine alınmaktadır. Ardından sunucu taraflı "public_files/" dosya yolunun sonuna istemci girdisi (göreceli dosya yolu) eklenmektedir. Bu şekilde oluşan nihai dosya yolundaki dosyanın içeriği okuması yapılmaktadır ve istemciye dönülmektedir. Bu örnek bir "doğrulanmamış dosya yolu" zafiyeti, bu örnek özelinde daha spesifik adıyla "göreceli yol gezinme" zafiyeti örneğidir. İstemci taraflı girdi "filename" denetlenmediği için ve olduğu gibi dosya okuması işleyişinde kullanıldığı için suistimale açıktır.

Ruby güvenli kod bloğunda ise istemci taraflı "filename" parametresi params[:filename] ile bir dosya yolu alınmaktadır, fakat bu sefer bu dosya yolu dosya okumasında kullanılmadan önce illegal karakterlerin kırpıldığı işlemden (File.basename() metodundan) geçmektedir. Bu sayede istemci tarafın gireceği kural dışı karakterler elimine olacaktır ve suistimal yolu kapatılmış olacaktır.

Kurum web uygulamada "Doğrulanmamış Yol Girdisi (CWE-73) (CWE-36) (CWE-23) (CWE-22)" açıklığı tespit edilmiştir.

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

Açıklığın Önlemi:

Tavsiyeler şu şekildedir:

- Tüm girdiler kaynağı neresi olursa olsun doğrulamadan geçirilmelidir. Bu doğrulama belirli yapıdaki girdileri reddetme işlemi yerine sadece belirli yapıya uyanların kabul edildiği şeklinde olmalıdır. Yani whitelist (beyaz liste) kullanılmalıdır. Siyah liste (blacklist) önlemi kullanılmamalıdır. Bir girdiyi doğrulamada şunlar kontrol edilebilir:
    - Veri türü (int, string, float, byte, … v.b. tipte mi geliyor kontrolü)
    - Boyutu (x KB, y MB,…v.b. boyutta mı geliyor kontrolü)
    - Aralığı (Veri ölçülebilir bir sayısal değerse aralık sınırlarını aşıyor mu kontrolü)
    - Formatı (çözümlenebilir şifreli, tek yönlü şifreli, açık metin, … v.b. biçimde mi geliyor kontrolü)
    - Beklenen Değerlerden Biri Olup Olmadığı (Whitelist'te tanımlı desenlerden biri mi geliyor kontrolü)
- Dosya yolunun doğrulandığından (canonicalized edildiğinden) emin olunmalıdır.
- Uygulama "uygulamalar binary klasörü"nden ayrı olan tasarlanmış bir klasörü kullanacak şekilde sınırlandırılmalıdır.
- Uygulamanın çalıştığı işletim sistemi kullanıcısının ayrıcalıkları gerekli dosyalar ve klasörler için kısıtlanmalıdır. Uygulama "uygulama binary klasörü"ne yazma yapamıyor olmalıdır ve uygulama dosya ve veri klasörleri dışındaki herhangi bir şeyi okumamalıdır.

Referanslar:

1. https://cwe.mitre.org/data/definitions/73
2. https://cwe.mitre.org/data/definitions/36
3. https://cwe.mitre.org/data/definitions/23.html
4. https://cwe.mitre.org/data/definitions/22.html
5. https://bhartee-tech-ror.medium.com/send-data-and-send-file-methods-in-ruby-on-rails-61ed94923ba6