1.1.1 Denetlenmemiş Verinin Tersine Serileştirilmesi (Deserialization of Untrusted Data) (CWE-502)

Açıklık Önem Derecesi: Yüksek

Açıklığın Etkisi: Uzaktan kod çalıştırma

Açıklığın Barındıran Dosyalar/Satırlar:

Proje Dosyası/Dosya Adı	Satır Numarası

Açıklığın Açıklaması:

Uygulamalar ihtiyaca göre serileştirme (serialization) ve deserileştirme (deserialization) işlemini kullanabilmekteler. Uygulama kaynak kodlarındaki bir veri yapısının veya nesnenin sonradan ağ üzerinden başka bir platforma iletilmek veya sonradan kullanılmak amacıyla dönüştürülerek bir dosyaya veya veritabanına depolanması işlemine serileştirme adı verilir. Serileştirme ile uygulama kaynak kodlarındaki bir veri yapısı veya nesne JSON, XML veya binary (byte[]) veri formatlarında dosyaya veya BLOB veri formatında veritabanına kaydolur. Uygulama kaynak kodlarındaki bir veri yapısının ve nesnenin serileştirme işlemi ile oluşan formunu tersine alarak tekrar uygulama kaynak kodlarında denk bir veri yapısı veya nesne oluşturmaya ise deserileştirme adı verilir.

Örneğin serileştirmeye python'dan örnek verilmiştir:

Python:

```python
class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age
```

Person nesnesi uygulamada bir serileştirme fonksiyonu kullanılarak aldığı değerler ile beraber şu şekilde JSON formata dönüştürülüp depolanabilir.

JSON Dosyası:

```
{"name": "John", "age": 25}
```

Örneğin deserileştirmeye yine python'dan bir örnek verilmiştir:

JSON Dosyası:

```
{"name": "John", "age": 25}
```

Python:

```
json_veri = '{"name": "John", "age": 25}'
person_nesne = json.loads(json_data)          # Python nesnesi ile sonuçlanır
                                              # {'name': 'John', 'age': 25}
```

Bir person'ın JSON temsili verisi alınarak ve json.loads() ile deserileştirme uygulayarak tekrardan kaynak kodda Person nesnesi elde edilir.

Deserileştirme sırasında eğer kaynak JSON, XML, Binary temsili verileri denetlenmeden kaynak kodda denk bir veri yapısı ve nesneye dönüştürülürse bu durum keyfi komut çalıştırma veya işletim sistemi komutları çalıştırma ile sonuçlanabilir. Saldırganlar bu sonuca deserileştirme işlemi sırasında nesnede mevcut olan olası sınıfları ve metotları çağırarak ulaşabilirler. Dolayısıyla denetlenmemiş veri deserileştirmeye tabi tutulursa bu bir güvenlik açıklığı olarak ele alınır. Çeşitli teknolojilerde bu açıklığın varlığını ve kapatılma yöntemini gösteren bazı kod bloklarına yer verilmiştir:

C# - Güvensiz Kod Bloğu:

```
// TR: Kullanıcı Girdisinden Zafiyetl Json.NET Nesne Deserileştirme
// EN: Vulnerable Json.NET Deserialization of Object from User Input

string objString = Request.Form["object"];

Newtonsoft.Json.JsonSerializerSettings settings = new
Newtonsoft.Json.JsonSerializerSettings {
        TypeNameHandling = TypeNameHandling.All //Any value except
TypeNameHandling.None is vulnerable
};

object o = Newtonsoft.Json.JsonConvert.DeserializeObject(objString, settings);
```

C# - Güvenli Kod Bloğu - Söz Dizimi (Syntax) (1):

```
// TR: Tür Zorlaması Uygulayan Kullanıcı Girdisinden
//     Json.NET Nesne Deserileştirmesi
// EN: Json.NET Deserialization of Object from User Input That Enforces Type

string objString = Request.Form["object"];

TrustedObject oo =
Newtonsoft.Json.JsonConvert.DeserializeObject<TrustedObject>(objString);
```

C# - Güvenli Kod Bloğu - Söz Dizimi (Syntax) (2):

```
// TR: Tür Zorlaması Uygulayan Kullanıcı Girdisinden
//     Json.NET Nesne Deserileştirmesi
// EN: Json.NET Deserialization of Object from User Input That Enforces Type

using Newtonsoft.Json;
using Newtonsoft.Json.Converters;

...

string objString = Request.Form["object"];

TrustedObject oo = JsonConvert.DeserializeObject<TrustedObject>(objString);
```

C# güvensiz kod bloğu örneğinde kullanıcı girdisinden gelen json veri denetlenmeden deserileştirmeye tabi tutuluyor. Kullanıcı, girdisini manipüle ederek deserileştirme sonrası kaynak kodda bu vaziyette keyfi komut çalıştırabilir. C# güvenli kod bloğu örneklerinde ise kullanıcı girdisinden gelen json veri içeriği kendi veri tipinde deserileştirmeye <TrustedObject> ile zorlanıyor. Yani kendi veri tipinde deserileştirmeye zorlanarak kullanıcı taraftan gelen json veri içerisinde var olabilecek komut niteliğindeki kodların kaynak kodda çalıştırılması önleniyor.

PHP - Güvensiz Kod Bloğu:

```
<?php

// TR: Güvensiz bir şekilde Unserialize() Fonksiyonu
//     ile Yeni $user Nesnesi oluşturma
// EN: Create New User with unserialize() insecurely

$attributes = $_GET['user_attributes'];    // Gelen Input JSON Verisi.
$user = unserialize($attributes);          // Deserileştirme ile Nesne Oluşur.

?>
```

PHP - Güvenli Kod Bloğu:

```php
<?php

    // TR: Parametreleri Kullanarak JSON'dan Yeni Bir $user Nesnesi
    // EN: Create New User from JSON using Parameters

    // .. JSON Validity Kontrolü - JSON Çıktı Döner .. //
    $user_params = json_decode($_GET['user_attributes']);

    // .. Parametre Kontrolü – JSON Çıktı Parçaları String'leşir .. //
    $name = $user_params['Name'];
    $email = $user_params['Email'];
    $phone = $user_params['Phone'];

    // .. String'lerle Yeni Bir $user Nesnesi Oluşur .. //
    $user = new User($name, $email, $phone);

?>
```

PHP güvensiz kod bloğu örneğinde kullanıcı girdisinden gelen json veri denetlenmeden deserileştirmeye tabi tutuluyor. Kullanıcı, girdisini manipüle ederek deserileştirme sonrası kaynak kodda bu vaziyette keyfi komut çalıştırabilir. PHP güvenli kod bloğu örneğinde ise kullanıcı girdisinden gelen json veri içeriği doğrudan deserileştirme ile nesneye dönüştürülmüyor. Kullanıcı girdisi önce json formatta ayıklanarak çekiliyor. Ardından ayıklanmış json formattaki veride yer alan json parametreler teker teker değişkenlere "string" olarak atanıyor. En sonunda bu string formattaki değişkenler yeni bir User nesnesi için değer olarak aktarılıyor ve $user nesnesi oluşturuluyor. Yani kullanıcı girdisi ayıklanarak ve string'leştirilerek gelen json veri içerisinde var olabilecek komut niteliğindeki kodların kaynak kodda çalıştırılması önleniyor.

JAVA - Güvensiz Kod Bloğu:

```java
// TR: ServletRequest Input Stream'den Bir Nesne Okuma
// EN: Reading an Object from ServletRequest Input Stream

ServletInputStream sis = request.getInputStream();
ObjectInputStream ois = new ObjectInputStream(sis);
Object obj = ois.readObject();
```

JAVA - Güvenli Kod Bloğu:

```
// TR: Güvenli Object Input Stream Kullanarak ServletRequest
//     Input Stream'den bir Nesne Okuma
// EN: Reading an Object from ServletRequest Input Stream
//     Using a Secure Object Input Stream

/* TrustedObject'nin güvenilir bir nesne sınıfı olduğunu varsayalım */

// ObjectInputStream'den Güvenli Kalıtım
public class SafeObjectInputStream extends ObjectInputStream {
    @Override
    protected Class<?> resolveClass(ObjectStreamClass desc) throws
IOException, ClassNotFoundException {

        if (!desc.getName().equals(TrustedObject.class.getName()))
        {
            throw new ClassNotFoundException("Unsupported Class: ",
desc.getName());
        }

        return super.resolveClass(desc);

    }
    /* Constructors */
}

// SafeObjectInputStream ile güvenli stream okuma kodu
ServletInputStream sis = request.getInputStream();
ObjectInputStream ois = new SafeObjectInputStream(sis);
TrustedObject obj = (TrustedObject)ois.readObject();
```

JAVA güvensiz kod bloğu örneğinde kullanıcı girdisinden gelen json veri denetlenmeden ObjectInputStream.readObject() ile deserileştirmeye tabi tutuluyor. Kullanıcı, girdisini manipüle ederek deserileştirme sonrası kaynak kodda bu vaziyette keyfi komut çalıştırabilir. JAVA güvenli kod bloğu örneğinde ise kullanıcı girdisinden gelen json veri içeriği doğrudan deserileştirme ile nesneye dönüştürülmüyor. Önce deserileştirme class'ından kalıtım yoluyla güvenli bir deserileştirme class'ı oluşturuluyor. Ardından güvenli deserileştirme class'ı ile deserileştirme uygulanarak ve ilgili TrustedObject nesnesi ile (güvenli olduğu varsayılıyor) casting işlemi uygulanarak nesne oluşturuluyor. Yani kullanıcı girdisi güvenli bir şekilde deserileştirilerek gelen json veri içerisinde var olabilecek komut niteliğindeki kodların kaynak kodda çalıştırılması önleniyor.

Eğer bir uygulamada denetlenmemiş bir veri doğrudan deserileştirmeye tabi tutuluyorsa “Denetlenmemiş Verinin Tersine Serileştirilmesi (CWE-502)” açıklığı vardır denir. Kurum uygulamada bu açıklık tespit edilmiştir:

:::::BULGU:::::

Açıklığın Önlemi:

Bu açıklığın giderilmesi için tavsiyeler şu şekildedir:

- Mümkün olduğunca serileştirilmiş nesneyi uzak platformlar arasında iletmeyin. Bunun yerine platformlar arasında primitive (temel) değerleri iletmeyi ve bu değerleri yeni platformda oluşturulmuş nesnenin özelliklerine (instance variable’larına / field’larına) atamayı (set etmeyi) değerlendirin.
- Eğer gerekiyorsa iletilen nesneler için whitelist önlemi kullanın. Daima iletilen nesnenin bilinen, güvenilen ve beklenilen nesne olduğundan emin olun. Herhangi bir kaynaktan dinamik olarak bir nesne oluşturmayın. Yalnızca nesne onaylanmışsa, güvenilen ve bilinen bir türdeyse ve içerisinde güvensiz nesneler içermediği düşünülüyorsa herhangi bir kaynaktan dinamik olarak nesne oluşturulabilir.
- Bir serileştirme yöntemi seçerken serileştirme yönteminin defans edilebilir, güvenli yapılandırılabilir olduğundan ve olası tehlikeli herhangi bir nesneye izin vermediğinden emin olmak adına daima serileştirme kütüphanesi dağıtıcısının dokümantasyonuna, best practice’lerine ve hatta bilinen sızma tekniklerine başvurulması gerekir.

Referanslar:

1. https://hazelcast.com/glossary/serialization/
2. https://medium.com/@khemanta/data-serialization-and-deserialization-what-is-it-29b5ca7a756f
3. https://stackoverflow.com/questions/1360632/what-are-object-serialization-and-deserialization
4. https://knowledge-base.secureflag.com/vulnerabilities/unsafe_deserialization/unsafe_deserialization_php.html
5. https://www.w3schools.com/php/func_var_unserialize.asp
6. https://www.php.net/manual/tr/function.serialize.php
7. https://cwe.mitre.org/data/definitions/502.html