SAST Jargonunda “Taint Source”, “Taint Sink”, “Data-Flow Analysis” Nedir?

İçindekiler

a. Taint Source Nedir?

“Taint Source”, harici bir girdinin (örn; bir dosyanın, bir servlet request’in, bir konsol girdisinin, veya bir socket girdisinin) uygulamanın kaynak kodlarında alındığı noktaya denir. Yani istemci girdisini getiren fonksiyona taint source (kusur kaynağı) denir. Taint source (kusur kaynağı) ile gelen istemci girdisinin kendisine ise tainted data (kusurlu veri) denir.

PHP’deki istemci girdisini getiren fonksiyonlara, yani taint source’lara (kusur kaynaklarına) örnek olarak aşağıdaki global değişkenler verilebilir:

```
$_GET
$_POST
$_COOKIE
$_REQUEST
$_FILES
$_SERVERS
$HTTP_GET_VARS                  // deprecated. $_GET is new one.
$HTTP_POST_VARS                 // deprecated. $_POST is new one.
$HTTP_COOKIE_VARS               // deprecated. $_COOKIE is new one.
$HTTP_REQUEST_VARS              // depracated. $_REQUEST is new one.
```

Java’daki istemci girdisini getiren fonksiyonlara, yani taint source’lara (kusur kaynaklarına) örnek olarak ise aşağıdaki fonksiyonlar verilebilir:

javax.servlet.http.HttpServletRequest Kütüphanesi Metotları:

```
getHeader()
getQueryString()
getRequestURI()
...
```

javax.servlet.http.Cookie Kütüphanesi Metotları:

getDomain()
getPath()
getName()
getValue()
...

java.io.Console Kütüphanesi Metotları:

readLine
readPassword
...

javax.servlet.ServletRequest Kütüphanesi Metotları:

getRemoteAddr()
getParameterValues()
...

...

b. Taint Sink Nedir?

“Taint sink” taint source’un (kusur kaynağının) getirdiği tainted data’nın (kusurlu verinin) uygulamada anahtar bir noktaya yerleştiği yere denir. Yani istemci girdisinin yürütüldüğü / işletildiği “hassas fonksiyonlara” taint sink (kusur çıkışı) denir.

> Not: “Sink” bilgisayar teknolojileri alanında çıkış düğümü anlamına gelmektedir. Normal anlamı ise lavabodur.

Sink fonksiyonları; veritabanı çağrı fonksiyonları, komut çalıştırma fonksiyonları, dosya çıktılama fonksiyonları, konsol çıktılama fonksiyonları, veya socket cevap gönderme fonksiyonları,... olabilir.

PHP’deki istemci girdilerinin işletilebileceği hassas fonksiyonlara, yani taint sink’lere (kusur çıkışlarına) örnek olarak aşağıdakiler verilebilir:

mysql_query()
mysqli_query()
fopen()
file_get_contents()
file_put_contents()
file()
copy()
unlink()
move_upload_file()
shell_exec()
exec()
require_once
include
echo
print

printf
die()
eval()
...

Java’daki istemci girdilerinin işletilebileceği hassas fonksiyonlara, yani taint sink’lere (kusur çıkışlarına) örnek olarak aşağıdakiler verilebilir:

java.lang.Runtime Kütüphanesi Metotları:

exec()
...

java.sql.PreparedStatement Kütüphanesi Metotları:

addBatch()
execute()
executeQuery()
executeUpdate()
...

javax.servlet.http.Cookie Kütüphanesi Metotları:

Cookie()
setDomain()
setPath()
setValue()
...

javax.servlet.jsp.JspWriter Kütüphanesi Metotları:

print()
println()
...

javax.servlet.http.HttpServletResponse

setHeader()
addHeader()
...

...

c. Data-Flow Analysis Nedir?

Data-Flow Analysis (diğer adıyla; Taint Analysis) taint source’dan (kusur kaynağından) ilgili taint sink’e (kusur çıkışına) doğru takip etme ve izleme sürecine denir. Data-Flow Analysis (taint analysis) yapıldığında girdi doğrulamaya (input validation’a) dair bir güvenlik açığının var olup olmadığı anlaşılır.

Girdi doğrulama (input validation) açıklıklarına örnek olarak şunlar verilebilir:

- SQL Injection (SQLI)
- Cross-site scripting (XSS)
- Remote File Inclusion (RFI)
- Local File Inclusion (LFI)
- Directory Traversal or Path Traversal (DT/PT)
- Source Code Disclosure (SCD)
- OS Command Injection (OSCI)
- PHP Code Injection
- ...

Data-Flow analysis, diğer adıyla taint analysis ise bu v.b. girdi doğrulama açıklıklarını tespit etmede uygulanır.

d. Sonuç

Sonuç olarak taint source (kusur kaynağı) istemci taraftan gelen girdileri getiren fonksiyonlara denir. Taint sink (kusur çıkışı) gelen tainted data'nın (kusurlu verinin) yerleştiği exploit edilebilen fonksiyonlara denir. Taint Analysis ise taint source'dan (kusur kaynağından) taint sink'e (kusur çıkışına) kadar takip yapıldığında tainted data'nın (kusurlu verinin) denetimsiz bir şekilde taint sink'e (kusur çıkışına) ulaşıp ulaşmadığının tespitine denir.

Taint source (kusur kaynağı) açıklığın olmasını sağlar. Taint sink (kusur çıkışı) ise açıklığın gerçekleştiği yerdir. Taint source (kusur kaynağı) ve taint sink (kusur çıkışı) data-flow (taint) analizlerinde kullanılmaktadır.

e. Sanitizing İle Önlem

Untainting işlemi, yani kusurun giderilmesi işlemi için taint source'lara (kusur kaynaklarına) sanitizing fonksiyonları uygulanması gerekir. Yani taint source'larla (kusur kaynaklarıyla) gelen tainted data'lar (kusurlu veriler) sanitizing edilmiş halde taint sink (kusur çıkışı) fonksiyonlarına ulaştırılmalıdır.

Bu işlem için örneğin PHP'de taint source'lara (kusur kaynaklarına) şu sanitizing fonksiyonları uygulanabilir:

mysql_escape_string()
mysql_real_escape_string()
htmlentities()
htmlspecialchars()
strip_tags()
urlencode()
…

Veya örneğin Java'da taint source'lara (kusur kaynaklarına) şu sanitizing fonksiyonları uygulanabilir:

org.owasp.encoder.Encode Kütüphanesi Metotları:

forHtml()
forCssString()
forUri()

forXml()
forCDATA()
forJava()
forJavaScript()
...

java.net.URLEncoder Kütüphanesi Metotları:

encode()
...

java.net.URLDecoder Kütüphanesi Metotları:

decode()
...

org.apache.commons.lang.StringEscapeUtils Kütüphanesi Metotları:

escapeJava
unescapeJava
escapeJavaScript
unescapeJavaScript
escapeHtml
unescapeHtml
escapeXml
unescapeXml
escapeCsv
unescapeCsv
escapeSql
...

...

f. Ek. Taint Analizi Örneği

i) Taint Source

Harici veriler potansiyel olarak tainted (kusurlu) olarak nitelendirilirler. Potansiyel olarak tainted (kusurlu) olan verileri getiren taint source'lara örnek olarak örneğin aşağıdaki gibi bir http request'teki parametrenin değerini getirme operasyonu verilebilir.

```
void ProcessRequest(HttpRequest request)
{
        ...
        string name = request.Form["name"];          // Taint Source
        ...                                          // Variable "name"
                                                     // contains
                                                     // potentially
                                                     // "tainted data".
}
```

ii) Tainted Data İletimi

Taint analizini yönetmenin bir önemli özelliği potansiyel tainted data'nın (kusurlu verinin) uygulamada yol aldığı rotayı saptamaktır. Örneğin buradaki potansiyel tainted data (kusurlu veri) bir taint source'dan (kusur kaynağından) name değişkenine iletilmektedir. Bunu takiben aşağıda gösterildiği gibi veri bir başka değişken olan $sql değişkenine iletilmektedir. Bunun neticesinde $sql değişkeni de potansiyel bir tainted data (kusurlu veri) olmaktadır. Son olarak bu veri bir fonksiyona arguman olarak iletilmektedir.

```
void ProcessRequest(HttpRequest request)
{
        ...

        string name = request.Form["name"];                          // Taint Source

        // Potentially "Tainted Data" concatenated with another string.
        string sql = $"SELECT * FROM Users WHERE name='{name}'";


        // Potentially "Tainted Data" passed as an argument to a function.
        ExecuteReaderCommand(sql);

        ...
}

void ExecuteReaderCommand(string sql)
{
        // sql variable contains potentially "tainted data".
        ....
}
```

iii) Taint Sink

Taint analizi bakış açısına göre bir uygulamada eğer tainted data (kusurlu veri) uygulamanın anahtar noktalarına girmekteyse açıklık vardır. Bu anahtar noktalar taint sink'lerdir. Her bir potansiyel açıklıklık kendi taint sink'ine sahiptir. Örneğin bir SQL Enjeksiyonu açıklığında taint sink sql sorgu string'ini sql komut nesnesi yapan bir fonksiyondur. Bu örnekte ise potansiyel olarak tainted data (kusurlu veri) olan $sql değişkeni SqlCommand() fonksiyonuna girmektedir. Yani bir taint sink'e girmektedir.

```
void ProcessRequest(HttpRequest request)
{
        ...

        string name = request.Form["name"];                          // Taint Source

        // Potentially "Tainted Data" concatenate with another string.
        string sql = $"SELECT * FROM Users WHERE name='{name}'";

        // Potentially "Tainted Data" passed as an argument to a function.
        ExecuteReaderCommand(sql);

        ...
}

void ExecuteReaderCommand(string sql)
{
        using (var command = new SqlCommand(sql, _connection))       // Taint Sink
        {
                using (var reader = command.ExecuteReader())
```

```
                {
                        ...
                }
        }
        ....
}
```

Taint analizi tainted data'nın (kusurlu verinin) taint source'dan (kusur kaynağından) taint sink'e (kusur çıkışına) doğru takip edebileceği bir yol var olup olmadığını kontrol eder. Bu örnekte taint source'dan bir taint sink'e doğru tainted data'nın yolculuğu vardır. Dolayısıyla bir açıklık vardır. Bu açıklığın adı SQL Enjeksiyonu açıklığıdır.

iv) Parameterize Kullanım ile Önlem

Potansiyel bir açıklığı önlemek için harici veri kontrol edilmelidir ve güvenli bir form'a dönüştürülmelidir. Bu güvenli form'a dönüştürme saldırının türüne göre değişiklik gösterir. Örneğin SQL Enjeksiyonu açıklığı durumunda parameterize sorgu kullanılabilir.

```
void ProcessRequest(HttpRequest request)
{
        ...

        String userName = Request.Form["userName"];
        String query = "SELECT * FROM Users WHERE UserName = @userName";

        using (var command = new SqlCommand(query, _connection))
        {

                // SQL query parameter value is showed.
                var userNameParam = new SqlParameter("@userName", userName);
                command.Parameters.Add(userNameParam);

                // SQL query is run safely.
                using (var reader = command.ExecuteReader())
                {
                        ...
                 }
                ...
        }

        ...
}
```

g. Ek. Taint Analizi Örneği (2)

i) Taint Source

Potansiyel olarak tainted data'ları (kusurlu verileri) getiren taint source'lara örneğin bir http request'teki parametrenin değerini getirme operasyonu yine örnek olarak verilebilir.

```
protected void Page_Load(object sender, EventArgs e)
{
        ...


        var userName = Request.Params["userName"];                      // Taint Source

                                                                        // Variable "userName"
        ...                                                             // contains potentially
                                                                        // “tainted data”.
}
```

ii) Tainted Data İletimi

Buradaki potansiyel tainted data (kusurlu veri) bir taint source'dan (kusur kaynağından) aşağıda gösterildiği gibi userName değişkenine iletilmektedir. Bunu takiben veri bir başka değişken olan $message değişkenine iletilmektedir. Bunun neticesinde $message değişkeni de potansiyel bir tainted data (kusurlu veri) olmaktadır.

```
protected void Page_Load(object sender, EventArgs e)
{
	....

	var userName = Request.Params["userName"];                    // Taint Source

	string message;

	if (string.IsNullOrWhiteSpace(userName))
	{
		message = "Empty 'userName' parameter";
	}
	else
	{
		message = $"'{userName}' data has been processed.";
	}

	...
}
```

iii) Taint Sink

Potansiyel olarak tainted data (kusurlu veri) olan $message değişkeni Response.Write() html çıktılama fonksiyonuna girmektedir. Yani bir taint sink'e girmektedir.

```
protected void Page_Load(object sender, EventArgs e)
{
	....

	var userName = Request.Params["userName"];                    // Taint Source

	string message;

	if (string.IsNullOrWhiteSpace(userName))
	{
	  message = "Empty 'userName' parameter";
	}
	else
	{
	  message = $"'{userName}' data has been processed.";
	}

	Response.Write(message);                                      // Taint Sink
}
```

Taint analizi tainted data'nın (kusurlu verinin) taint source'dan (kusur kaynağından) taint sink'e (kusur çıkışına) doğru takip edebileceği bir yol var olup olmadığını kontrol eder. Bu örnekte taint source'dan bir taint sink'e doğru tainted data'nın yolculuğu vardır. Dolayısıyla bir açıklık vardır. Bu açıklığın adı Cross Site Scripting açıklığıdır.

iv) Sanitizing ile Önlem

Cross Site Scripting açıklığını önlemek için taint source'un (kusur kaynağının) getirdiği tainted data (kusurlu veri) sanitizing işlemine tabi tutulmalıdır ve o şekilde taint sink'e (kusur çıkışına) ulaştırılmalıdır.

```
protected void Page_Load(object sender, EventArgs e)
{
        String userName = Request.Form["userName"];

        // Sanitizing variable userName.
        var encodedUserName = System.Net.WebUtility.HtmlEncode(userName);

        String message;

        if (string.IsNullOrWhiteSpace(encodedUserName))
        {
                message = "Empty 'encodedUserName' parameter";
        }
        else
        {
                message = $"'{encodedUserName}' data has been processed.";
        }

        // Output request parameter as response safely.
        Response.Write(message);
}
```

Kaynaklar

https://tureng.com/en/turkish-english/sink
https://blog.shiftleft.io/how-to-review-code-for-vulnerabilities-1d017c21a695
https://pvs-studio.com/en/blog/terms/6496/
https://security.stackexchange.com/questions/123854/terminology-entry-point-data-source-sink
https://help.hcltechsw.com/appscan/Source/9.0.3/topics/views_source_sink_2.html
https://thecodemaster.net/methods-considered-sources-sinks-sanitization/
https://www.researchgate.net/figure/LIST-OF-SOURCES-AND-SINKS-USED-FOR-THE-INFORMATION-FLOW-ANALYSIS_tbl1_319895349
https://wiki.owasp.org/index.php/OWASP_LAPSE_Project
https://docs.sonarqube.org/latest/user-guide/security-rules/
https://www.sitepoint.com/community/t/http-cookie-vars-vs--cookie/3444/3
https://wiki.owasp.org/index.php/OWASP_WAP-Web_Application_Protection
http://awap.sourceforge.net/support.html