not a MANual Tool for Risk Analysis

Risk Analysis Report [Implementation Constraints]

This document describes the features needed to be implemented in the implementation step to avoid the most risky threats to the system

*** Report Generated by: guifre**

*** Date: Fri Oct 26 10:25:54 CEST 2012**

*** Issued by: Computer Architecture and Operating Systems (CAOS)**
Universitat Autonoma de Barcelona.

Department of Architecture and Operating Systems

Universitat Autònoma de Barcelona

This is an alpha release, if for some arbitrary reason this got to you and have any comment suggestion question idea, you can ping me at guifre.ruiz at the gmail dot com server ;-).

1. Time of Check to Time Of Use (TOCTOU)

1.1. Affected Components

* Path[0] Threat Agent[Anonymous User] Asset Element[config].
* Path[1] Threat Agent[Anonymous User] Asset Element[Logs].
* Path[2] Threat Agent[Identified User] Asset Element[config].
* Path[3] Threat Agent[Identified User] Asset Element[Logs].

1.2. Description

Summary

It is fairly common for an application to need to check some condition before undertaking an action. For example, it might check to see if a file exists before writing to it, or whether the user has access rights to read a file before opening it for reading. Because there is a time gap between the check and the use, an attacker can sometimes use that gap to mount an attack. Temporary Files: A classic example is the case where an application writes temporary files to publicly accessible directories. You can set the file permissions of the temporary file to prevent another user from altering the file. However, if the file already exists before you write to it, you could be overwriting data needed by another program, or you could be using a file prepared by an attacker, in which case it might be a hard link or symbolic link, redirecting your output to a file needed by the system or to a file controlled by the attacker.

Example

```
if (access("file", W_OK) != 0) {
  exit(1);
}
fd = open("file", O_WRONLY);
write(fd, buffer, sizeof(buffer));

If an attacker executes symlink("/etc/passwd", "file") after access() and before
open(), a succesful attack will be performed.
```

References

http://capec.mitre.org/data/definitions/29.html
http://cwe.mitre.org/data/definitions/367.html

1.3. Software Specifications (AKA countermeasures)

Summary: Recheck the resource after the use call to verify that the action was taken appropriately.

Recheck the resource after the use call to verify that the action was taken appropriately.

2. SQL Injection Attacks

2.1. Affected Components

* Path[0] Threat Agent[Anonymous User] Asset Element[Relational Database].
* Path[1] Threat Agent[Identified User] Asset Element[Relational Database].

2.2. Description

Summary

A SQL injection attack consists of insertion or injection of a SQL query via the input data from the client to the application. A successful SQL injection exploit can read sensitive data from the database, modify database data (Insert/Update/Delete), execute administration operations on the database (such as shutdown the DBMS), recover the content of a given file present on the DBMS file system and in some cases issue commands to the operating system. SQL injection attacks are a type of injection attack, in which SQL commands are injected into data-plane input in order to effect the execution of predefined SQL commands..

Example

```
sql_query= "SELECT ProductName, ProductDescription FROM Products WHERE ProductNumber =
" & Request.QueryString("ProductID")
It expects users to define the parameter ProductID with a number such as 123. However,
an attacker can execute arbitrary SQL queries by using SQL syntax to change the meaning
of the query.For example: '123 ;DROP ALL tables' would execute
SELECT ProductName, ProductDescription FROM Products WHERE ProductNumber = 123; DROP
all tables
```

References

http://cwe.mitre.org/data/definitions/89.html
https://www.owasp.org/index.php/SQL_injection
http://www.unixwiz.net/techtips/sql-injection.html

2.3. Software Specifications (AKA countermeasures)

Summary: Allowing only alphanumeric characters in all fields of this payload

The fields used for this aim should only allow alphanumeric characters.

you can make sure it does no contain other types of characters by matching the user-controllable data with a regular expression such as "^[a-zA-Z0-9_]*$". For more information refer to http://msdn.microsoft.com/en-us/library/ms525361(v=vs.90).aspx and http://www.informit.com/articles/article.aspx?p=102193&seqNum=12

3. Reflected Cross Site Scripting (RXSS)

3.1. Affected Components

* Path[0] Threat Agent[Anonymous User] Asset Element[VO Admin].
* Path[1] Threat Agent[Anonymous User] Asset Element[Identified User].
* Path[2] Threat Agent[Identified User] Asset Element[VO Admin].
* Path[3] Threat Agent[Identified User] Asset Element[Identified User].

3.2. Description

Summary

Cross-Site Scripting attacks are a type of injection problem, in which malicious scripts are injected into the otherwise benign and trusted web sites. Cross-site scripting (XSS) attacks occur when an attacker uses a web application to send malicious code, generally in the form of a browser side script, to a different end user. Flaws that allow these attacks to succeed are quite widespread and occur anywhere a web application uses input from a user in the output it generates without validating or encoding it.

An attacker can use XSS to send a malicious script to an unsuspecting user. The end user’s browser has no way to know that the script should not be trusted, and will execute the script. Because it thinks the script came from a trusted source, the malicious script can access any cookies, session tokens, or other sensitive information retained by your browser and used with that site. These scripts can even rewrite the content of the HTML page.

Example

```
<% String eid = request.getParameter("eid"); %>
  ...
Employee ID: <%= eid %>
In the previous example the eid parameter is displayed to the web browser without
escaping or validating HTML characters.
```

References

http://capec.mitre.org/data/definitions/32.html
http://cwe.mitre.org/data/definitions/79.html
https://www.owasp.org/index.php/Cross-site_Scripting_(XSS)

3.3. Software Specifications (AKA countermeasures)

Summary: Allowing only alphanumeric characters in all fields of this payload

The fields used for this aim should only allow alphanumeric characters.
you can make sure it does no contain other types of characters by matching the user-controllable data with a regular expression such as "^[a-zA-Z0-9_]*$". For more information refer to http://msdn.microsoft.com/en-us/library/ms525361(v=vs.90).aspx and
http://www.informit.com/articles/article.aspx?p=102193&seqNum=12

4. E-mail Headers Injection

4.1. Affected Components

* Path[0] Threat Agent[Anonymous User] Asset Element[Email server].
* Path[1] Threat Agent[Identified User] Asset Element[Email server].

4.2. Description

Summary

An attacker manipulates the headers and content of an email message by injecting data via the use of delimeter characters native to the protocol. Many applications allow users to send email messages by filling in fields. For example, a web site may have a link to "share this site with a friend" where the user provides the recipient's email address and the web application fills out all the other fields, such as the subject and body. In this pattern, an attacker adds header and body information to an email message by injecting additional content in an input field used to construct a header of the mail message. This attack takes advantage of the fact that RFC 822 requires that headers in a mail message be separated by a carriage return. As a result, an attacker can inject new headers or content simply by adding a delimiting carriage return and then supplying the new heading and body information. This attack will not work if the user can only supply the message body since a carriage return in the body is treated as a normal character.

Example

A common function to send emails is the following one:

```
<?php mail($to,$subject,$message,"From: $from
"); ?>
```

As we can see, the subject is user-controllable data. A maliciou user can trick the SMTP server by providing the following string as $from header:

```
haxor@attack.com%0AContent-
Type:text/html%0A%0AMy%20%New%0A<u>HTML%20Anonymous%20Message.</u>%0A
```

The resulting e-mail would look like:

```
To: buddy@pal.xxx
Subject: Visit our site www.website.xxx !
From: haxor@attack.com
Content-Type:text/html

My New
<u>HTML Anonymous Message.</u>

Hello,
A friend thought you might want to see this page : www.website.xxx.
Bye Bye
```

References

http://capec.mitre.org/data/definitions/41.html

https://www.owasp.org/index.php/Testing_for_IMAP/SMTP_Injection

http://www.securephpwiki.com/index.php/Email_Injection

4.3. Software Specifications (AKA countermeasures)

Summary: Allowing only alphanumeric characters in all fields of this payload

The fields used for this aim should only allow alphanumeric characters.
you can make sure it does no contain other types of characters by matching the user-controllable data with a regular expression such as "^[a-zA-Z0-9_]*$". For more information refer to http://msdn.microsoft.com/en-us/library/ms525361(v=vs.90).aspx and
http://www.informit.com/articles/article.aspx?p=102193&seqNum=12