Study of Vulnerability

Unvalidated Input

Vivek Khajuria

Aug 29th, 2011

Contents

INTRODUCTION

Vulnerability

It is a weakness which allows an attacker to reduce a system's information assurance. Vulnerability is the intersection of three elements: a system susceptibility or flaw, attacker access to the flaw, and attacker capability to exploit the flaw. To be vulnerable, an attacker must have at least one applicable tool or technique that can connect to a system weakness. In this frame, vulnerability is also known as the attack surface.

Unvalidated Input

Information from web requests is not validated before being used by a web application. Attackers can use these flaws to attack back-end components through a web application.

Attacker manipulates the input thread and submit to the input field.

When an web site has been compromised, there is a good chance the attacker used unvalidated input as an element of the attack. If information submitted via a Web site is not validated before it's processed, an attacker can obtain sensitive information or attack the site.

Programmers should be cautious when writing code that accepts user input. Programmers tend to assume that all users will enter in Web forms only the data they are prompted for (such as a six-digit invoice number) and wouldn't think to enter anything different. It is this trust that has led many a Web site to ruin.

HOW DOES IT WORK

Two main types of validation exist: client-side and server-side.

Client-side validation - which is pushed to the browser and uses the client machine to validate input, should only be used as a method to reduce unneeded calls to the server. It may serve some useful purposes though. For example, the entire burden of input validation could conceivably be offset to the client. One could, for instance, set an HTML input field to perform data validation and policy enforcement. The policy could state that alphanumeric input must be of a certain length, or must have at least one capitalized character, and so on. In your pen testing endeavors, look for too much logic on the client side, because that may represent a high-risk area.

Server-side validation - accepts client-submitted data and processes it. Failure to ensure server-side validation can lead to a number of potentially serious security vulnerabilities. Many threats, including XSS and SQL Injection, can manifest themselves if proper server-side controls are not implemented. In today's web environments, server-side processing is inevitable and quite useful. For instance, any data transformations that need to take place will probably do so based on other data pulled from a DB or from some data sitting on the live stack. Doing this type of data processing on the client side is far too expensive and in some cases just downright impossible.

Web applications use input from HTTP requests (and occasionally files) to determine how to respond. Attackers can tamper with any part of an HTTP request, including

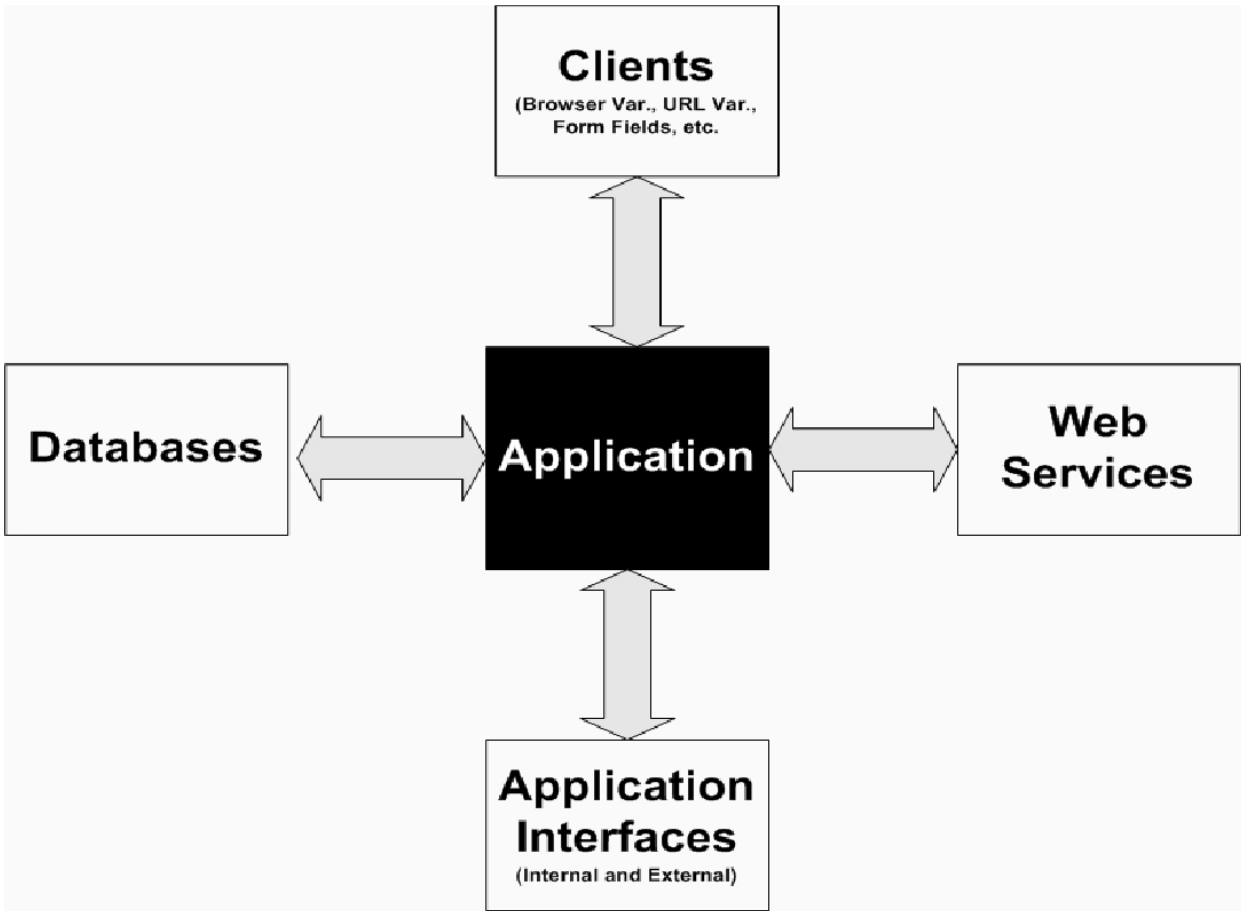
1. The URL

2. Query string
3. Headers
4. Cookies
5. Form fields
6. Hidden fields

to try to bypass the site's security mechanisms.

Below is the logical view of four of the ways in which input is received by an application.

Input from any one of these sources impacts how an application accesses, processes, and



displays data. For example, attackers might add, delete, or modify URL parameters in a query string. Hidden form fields can be changed and the form resubmitted. Database information, especially fields written by other applications, might be either purposely or accidentally tainted.

DAMAGES IT CAUSES

Common input tampering attacks include:

1. Improper HTML Display
2. By-Passed Client Side Validation
3. Cross-site Scripting
4. Generation of informational error messages
5. Buffer overflow
6. Unauthorized access
7. Injecting Flaws
8. Forced Browsing
9. Cookie Poisoning
10. Format String Attack
11. Hidden Field Manipulation

There are many possible negative outcomes if input containing either improperly formatted or invalid/malicious content reaches a business critical web application. Few of them are :

1. Improper HTML display – The lowest impact display issue would be an unprofessional portrayal of your site. On the other end of the spectrum, tainted input can render your display unusable. Further, attackers can force errors in application output to gain some idea of the diligence with which your developers tighten-up their code. If it's easy to hack a display, it's a good bet that it's just as easy to crack other application components.

2. By-passed client side validation – Client side validation is not really validation. It isn't very difficult for an attacker to disable script execution on her workstation, enter malicious invalid form input, and then submit the form. If there's no validation on the server side of the transaction, server crashes and the execution of rogue commands are just two of the possible outcomes.

3. Cross-site scripting – The web application can be used as a mechanism to transport an attack to an end user's browser. A successful attack can disclose the end user's session token,attack the local machine or spoof content to fool the user. Text fields that are not validated might contain HTML tags--like <script> and <object>--that execute code unexpectedly.

4. Generation of informational error messages – By forcing certain types of error messages, an attacker can obtain application and operating system version and patch level information about your system. This information is commonly used in the first phase of an attack—footprinting.

5. Buffer overflow – Web application components in some languages that do not properly validate input can be crashed and, in some cases, used to take control of a process. These components can include CGI, libraries, drivers, and web application server components.

6. Unauthorized access – Access to files might be obtained by manipulating paths. For example, the following is a path to a file to which the user has authorized access:

```
<a href="display.cfm?paper_id=999">Customer List</a>
```

By manipulating the paper_id parameter, an attacker could potentially retrieve sensitive information to which he has not been granted official access.

7. Injecting Flaws - Web applications pass parameters when they access external systems or the local operating system. If an attacker can embed malicious commands in these parameters, the external system may execute those commands on behalf of the web application.

8. Forced Browsing - is an attack where the aim is to enumerate and access resources that are not referenced by the application, but are still accessible. An attacker can use Brute Force techniques to search for unlinked contents.

9. Cookie Poisoning - is the modification of a cookie (personal information in a Web user's computer) by an attacker to gain unauthorized information about the user for purposes such as identity theft.

10. Format string attack - The Format String exploit occurs when the submitted data of an input string is evaluated as a command by the application. In this way, the attacker could execute code, read the stack, or cause a segmentation fault in the running application, causing new behaviors that could compromise the security or the stability of the system.

11. Hidden Field Manipulation - In this during a client session, developers take the help of hidden fields to store information related to the client.

PRECAUTIONS WE SHOULD TAKE

The best way to prevent parameter tampering is to ensure that all parameters are validated before they are used. A centralized component or library is likely to be the most effective, as the code performing the checking should all be in one place. Each parameter should be checked against a strict format that specifies exactly what input will be allowed. "Negative" approaches that involve filtering out certain bad input or approaches that rely on signatures are not likely to be effective and may be difficult to maintain.

Parameters should be validated against a "positive" specification that defines:

- Data type (string, integer, real, etc...)
- Allowed character set
- Minimum and maximum length
- Whether null is allowed
- Whether the parameter is required or not
- Whether duplicates are allowed
- Numeric range
- Specific legal values (enumeration)
- Specific patterns (regular expressions)

A new class of security devices known as web application firewalls can provide some parameter validation services. However, in order for them to be effective, the device must be configured with a strict definition of what is valid for each parameter for your site. This includes properly protecting all types of input from the HTTP request, including URLs, forms, cookies, query strings, hidden fields, and parameters.

TESTING VULNERABILITY

To test if controls are active for the applications you are screening, you can submit data that is known to be invalid. Input manipulation basically means you must force anomalies onto the target application and see how it reacts. When investigating potential input manipulation vulnerabilities, use the following general checklist of things to look for or use:

- Can you find any limitations in the defined/used variables and protocol payload, that is, accepted data length, accepted data types, data formats, and so on?
- Use exceptionally long character-strings to find buffer overflow vulnerability in the application code base or the web server itself.
- Use concatenation techniques in the input strings to try to get the target application to behave incorrectly.
- Inject specially crafted SQL statements in the input strings of database tiered Web applications
- Try to force Cross-Site Scripting(XSS) functionality.
- Look for unauthorized directory or file access with path or directory traversal in the input strings of the target application.
- Try using specific URL-encoded strings and Unicode-encoded strings to bypass input validation mechanisms used within the target application.
- If you detect the use of server-side includes, try executing remote commands.
- Try to manipulate the session management techniques to fool or modify the server-side logic
- Try to manipulate (hidden) field variables in HTML forms to fool server-side logic.
- Try to manipulate the “Referrer” value in the HTTP “Host” header in order to fool or modify server-side logic.
- Try to force illogical or illegal input so as to test the target’s error-handling routines.

It is also possible to find tainted parameter use by using tools like OWASP's Web-scarab. By submitting unexpected values in HTTP requests and viewing the web application's responses, you can identify places where tainted parameters are used.

CONCLUSION

Unvalidated input is a very severe security threat which include manipulating the input thread and submitting it to find the flaws in the system. It can cause enormous damage to the user as hacker can play with the confidential data of the user. So taking necessary precautions and properly testing the web service is imperative to ensure safety of the user.