

Web Application Security: Unvalidated Input

Tom Olzak
June 2006

According to the OWASP Guide, unvalidated input is the most common weakness found in web applications. Tainted input leads to almost all other vulnerabilities in these environments ([OWASP, 2005](#)). Before we look at how to prevent this weakness from spreading throughout your web solutions, let's examine the potential threats to your business when tainted input is allowed to reach your processing components.

Figure 1 is a logical view of four of the ways in which input is received by an application.

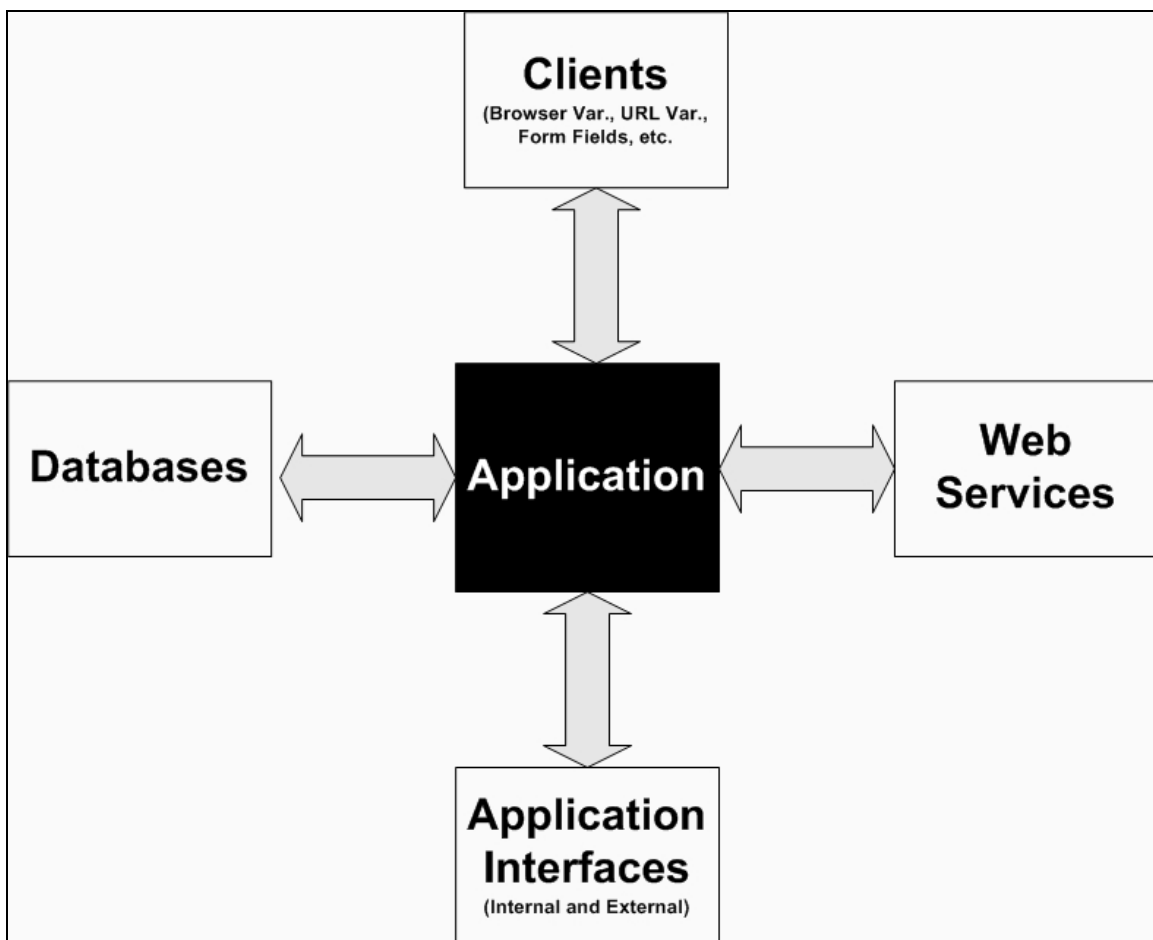


Figure 1

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.

Potential Vulnerabilities

There are many possible negative outcomes if input containing either improperly formatted or invalid/malicious content reaches a business critical web application. The following are just a few:

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.

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.

Cross-site scripting – Text fields that are not validated might contain HTML tags--like [<script>](#) and [<object>](#)--that execute code unexpectedly.

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](#).

Buffer overflow – Attackers can use [buffer overflows](#) to crash a system or to execute malicious code.

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=235">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.

SQL Injection – This type of attack exposes company databases to unauthorized modification, deletion, or addition of data. It's caused by an attacker including

SQL code in the input to a form field or by directly manipulating query strings. An example can be found at http://en.wikipedia.org/wiki/Sql_injection.

Validation as a Defense

This is quite a list of vulnerabilities, but it isn't complete. Vulnerabilities are limited only by your system's unique weaknesses and the creativity of the attacker. The only truly effective defense against tainted input is to secure the code used to deploy the application. The following are some principles your development teams should consider.

1. Validate everything. Inspect what you expect, and reject anything unexpected.
2. Perform all validation on the server. Client side validation is suspect for reasons given earlier in this article.
3. Use *positive filtering* instead of *negative filtering*. In other words, check for what should be present instead of for what shouldn't. Validating for what shouldn't be in the input leaves too many open holes in the validation process; there are too many variations to check. Possible filtering checks include:
 - a. Is it the right data type (string, integer, etc.)
 - b. Is the parameter required
 - c. Is the character set allowed
 - d. Does the input meet minimum and maximum length constraints
 - e. Is NULL allowed
 - f. If numeric, is the input within range constraints
 - g. Does the input cause data duplication, and if so is it acceptable
 - h. Does the input meet format requirements (e.g., when compared to a regular expression)
 - i. If selected from a list, does the parameter contain a valid value
4. Perform internal code reviews or "buddy checks". The first line of defense is having your programmers check each other's work. As we'll see later in this series, there are also automated ways to perform this task.
5. Make sure that Quality Assurance processes include testing not only valid input but also invalid input.

Some development environments provide ready-made controls for validation. For example, the following validators are available in ASP.NET:

- RequiredFieldIndicator – Forces entry of a value
- CompareValidator – Compares input values
- RangeValidator – Checks input for compliance with range constraints
- RegularExpressionValidator – Validates user input with regular expressions
- CustomValidator – Allows creation of customized validation routines

Conclusion

Your approach to validating input will be as unique as your development environment. The one common requirement across all environments, however, is development of appropriate policies, standards, guidelines, practices, and developer awareness.

Next week, we examine the second OWASP Top Ten vulnerability—broken access control.

© 2006 Thomas W. Olzak. Tom Olzak, MBA, CISSP, MCSE, is President and CEO of Erudio Security, LLC. He can be reached at tom.olzak@erudiosecurity.com. Additional security management resources are available at <http://adventuresinsecurity.com>.

Check out his book, *Just Enough Security*, at Amazon.com

Visit Tom's blog at <http://blogs.ittoolbox.com/security/adventures/>

Listen to Tom's podcasts at <http://blastpodcast.com/viewpodcast.html?id=441>

Free security training available at <http://adventuresinsecurity.com/SCourses>

Additional Resources

- [Untrusted Data Sources](#)
- [Input Validation with ASP.NET, Part 1](#)
- [A Security Checklist for Web Application Design](#)
- [Top 7 PHP Security Blunders](#)
- [Have Your Cake and Eat It Too](#)