

# Input validation of client-server web applications through static analysis

Francis Hsu  
University of California, Davis  
fhsu@cs.ucdavis.edu

## 1 Motivation

While early web applications were created with all data processing done on the server, the expansion in the use of scripting languages embedded in web browsers (specifically, dialects of ECMAScript - JavaScript and JScript) in techniques such as AJAX have allowed for change in the design of web applications. They are no longer run solely on the server-side with a limited input interface of static boxes in an HTML form, but are combinations of two programs – a client program run by the web browser communicating with a server program. While new applications of this type bring benefits in usability, the additional complexity may introduce security problems.

These new web applications designed with client and server components face the traditional problems of classic client-server programs, such as validating the input to the client or server program. However, the web applications' server and client components are usually designed with ad-hoc application level protocols only to operate with its counterpart and no other programs. This may lead to dangerous assumptions about the internal state of the counterpart and any data being transmitted. For example, in some cases application writers attempt to handle input validation with JavaScript in the client browser. When the input is then transmitted to the server-side part of the application, the server code continues to operate on the data with the assumption that the client's input validation had successfully completed. A malicious party could simply construct a client without these checks and submit input without validation, leading to security failures such as SQL injection attacks. With the client source code made accessible to attackers in script form, such vulnerabilities are even more easily exploited. Since the web application programmer had the intention of performing these checks on data to be transmitted to the server, input validation code done on the client should also be present in the server code.

## 2 Input validation

Static analysis has been used in many instances to automatically find security vulnerabilities in source code. and specifically for this domain, it has been successfully applied to find vulnerabilities such as SQL injection and XSS in web applications [5].

We can use static analysis techniques to ensure consistency in input validation checks between the client and server portions of a web application. One interesting thing to note is that the client portion of the application can be partially or even completely generated by the server code at runtime. The output of the server application is an HTML document with the client application embedded within. Techniques to create static approximations of the dynamic webpages [4] can reliably extract the client application.

With the client application code, we can perform source analysis to identify the points of interaction with the server. While a client application running on a web browser with AJAX has much richer interactions with the server, all interactions are still limited by traditional GET and POST requests through form submissions or XMLHttpRequest objects. Form submissions simply submit the value of a set of variables entered by the user and/or manipulated by the client. Before the submission is allowed to continue, the client portion may alter or block the submitted data. For example, only strings conforming to an email addresses regular expression may be passed. The client code thereby reduces the space of allowable values to be sent. This may seem not to be applicable to XMLHttpRequest objects, since it seems to be more likely to allow arbitrary data to be sent over to the server. However, conventions like

```

<script>
function validateEmail(field) {
    pattern = /^(\.)+@(\.)+\.(com|net|org|edu)$/;
    return pattern.test(field);
}
</script>
<form action="register.php" onSubmit="validateEmail()">
<input type="text" name="email">
</form>

```

#### **register.html**

```
mysql_query("INSERT INTO users VALUES ('" . $_POST["email"] . "')");
```

#### **register.php**

Figure 1: Example code based on web application vulnerable to CVE-2005-3365

JavaScript Object Notation (JSON) encapsulate the variables being transmitted, and allows the sent data to be mapped to variables used in the construction of the JSON object.

Even though the sent data from the client may perform this validation, it should be duplicated on the server side, since a malicious party may use a modified client or generate the inputs without one. We can apply constraint analysis [1] to those variables at two points: on the clients before sending, and on the sever before use in some security relevant function (such as part of a database query or storage). If value range constraint at the server side allows for a larger domain than that of the client, it indicates there exist some inputs that the server will accept that cannot be generated by the client. Inputs not properly sanitized by the server are the source of security vulnerabilities such as XSS or command injection. For example, the code in Figure 1 is based on a web application with known SQL injection vulnerabilities [2]. The client side Javascript suggests the developers were concerned with input validation, but may have forgotten replicate their validation code during the server processing. However, these constraint checks can cover even larger classes of programming errors. We can uncover higher level semantic errors – for example, if the shopping cart web application in [3] had calculated quantities or prices for an item in the client, the numbers should also be checked on the server, lest an enterprising customer purchase a store’s inventory for a dollar.

## References

- [1] A. Aiken. Introduction to set constraint-based program analysis. *Sci. Comput. Program.*, 35(2-3):79–111, 1999.
- [2] Common Vulnerabilities and Exposures. CVE-2005-3365. <http://nvd.nist.gov/nvd.cfm?cvename=CVE-2005-3365>.
- [3] Common Vulnerabilities and Exposures. CVE-2006-6464. <http://nvd.nist.gov/nvd.cfm?cvename=CVE-2006-6464>.
- [4] Y. Minamide. Static approximation of dynamically generated web pages. In *WWW '05: Proceedings of the 14th international conference on World Wide Web*, pages 432–441, New York, NY, USA, 2005. ACM Press.
- [5] Y. Xie and A. Aiken. Static detection of security vulnerabilities in scripting languages. In *USENIX Security '06: Proceedings of the 15th USENIX Security Symposium*, 2006.