A paper on

Web-Form Submission Security

From SolidBlue Software Inc.

Severity: Unknown, Depends on system targeted.
Author: Doug W. Styles
Date: March 23, 2003

Table of Contents:

A paper on

Web-Form Submission Security

From SolidBlue Software Inc.

Foreword: The study this paper discusses could not collect quantitative information on the number of systems lacking appropriate security. This paper discusses an ability to reverse engineer unseen software at work through submitting invalid data to be processed. To test public systems for security would require analyzing their response to real attacks – and where possible this was done on a very limited scale to relatively unimportant systems.

Overview: This paper will discuss automated attack methods against web sites utilizing "HTML Forms". An HTML Form is used on web sites requesting data from visitors. Traditionally web sites have been used as a limited medium whereby feedback and user information was requested by email or file uploads. It is now very common to see web sites allowing users to post messages, create accounts, enter information for processing, sign up for mailing lists, vote on topics, etc. All of these are examples of HTML Forms in use.

Once information is submitted to a web site, it is processed on the server. The processing software will identify the different information submitted, database it, calculate a response, etc. This second-level of web development, the processing level, requires a more advanced developer to setup. This is the area where security becomes a major concern.

The number of web developers learning this form of development is growing very fast, while the security implications and ability to spoof data for web sites of this nature is still relatively unknown or unguarded against due to the relatively low number of attackers on the Internet. Almost all of the web pages analyzed in this study had no effective security guarding against data spoofing.

Some examples of data-spoofing targets are listed below. An exhaustive list of targets is beyond the scope of this paper.

Attack Targets:

- Online voting systems
- Contact Forms for submitting comments and requesting information
- Message forums
- Etc.

Attack Methods:

- Method 1: Using an automated script or program to submit information repeatedly.
  (Proof of Concept: fig 1)
- Method 2: Submitting variables beyond their intended scope.
  (Proof of Concept: fig. 2)

Attack Types:

- Changing voting data…… (Changing weight of vote data for polls or record listings)
- Denial of service…… (Filling database to limit or increasing access time)
- Obfuscation of existing data…. (Inserting useless records throughout a database)
- Modifying existing data……. (Editing records intended for other people)

Beginning an Attack

For both Attack Method's this paper will discuss, it is important to understand the relationship between both levels of the target. Level one, the web page, is configured to accept information from visitors. It is coded in HTML and must send visitors questions along with code to accept input. The second-level is always hidden, and is most commonly coded in the PHP Hypertext Markup language, Microsoft ASP, or a server side executable. This second-level will process the information, database it in some form, and will usually generate a web page to display the result of its processing.

The commonality between these two levels is the variables used to represent the data. A variable is an identifier for information so that it can be accessed from different areas of a program by the same name. The following pseudo-code explains this relationship.

Example:
$Name = "John Smith"
Print $Name

The output of this program would be, "John Smith".

Now let's examine a two-level program in pseudo-code, meant to represent the relationship between an HTML page and the program that processes it.

Example:
Level 1 pseudo-code:

Print "How old are you?"
Request as $Age
Send Level 2 $Age

Level 2 outputs:

"You are between the age of 20 and 30. The average age of our visitors is 37 years old."

Explanation:

While we can not see the program at level 2, we know from its output that:
- It has recorded our age.
- It has recorded the age of other people.
- It calculates statistical information.

Most importantly
- We know that it is passed information as $Age

Vulnerability of Online Polls and Voting Systems

Let's discuss the information it calculates. While we can not see how it calculates the information, we know that to find an average -- data sets are added up and the sum is divided by the amount of data records. Therefore, if this example were a target it would be susceptible to both Attack Methods.

An attacker could repeatedly submit ages slightly higher than the average (1000 records at 45 years old) or an attacker could submit a very large age (1000 years old). Both of these methods would increase the average. The first would be hard to identify among a database of similar records, but the second would require very little work.

Not being able to view the level 2 program, it is impossible to know immediately what type of security the second level program uses. It is possible that it only accepts a certain range of ages, 1 – 100 years old.

Some programs will double check the data for its scope, and some will not. Therefore the first attack method is the one more often used, but the second is still common and in some situations, preferred.

Data Obfuscation and Denial of Service

Let's examine a web page that accepts feedback from users. The web page asks for a full name, email address, and comments. From using this web page, an attacker identifies that the email address is used to send marketing material to visitors (identifying them by name), and comments are sent to the owner of the page.

An attacker utilizing a simple script or program could repeatedly send feedback through this page with changing data. The database storing names with respective email addresses would be flooded with fake data.

Flooding the database with useless records would slow down access time, make valid records harder to identify, and possibly fill the database to its capacity. Making the addition of further records impossible.

Figure 1 is a PHP program that generates HTML output. By pointing multiple web browser's to a site hosting the Figure 1 program, a distributed denial of service attack against the target is possible. Furthermore, the attack would appear to come from the systems visiting the page – while the site hosting it would not be identifiable to the target.

With the already large and increasing storage space of modern web servers, a distributed attack against web sites that database information is the only viable method of causing denial of service in a short period of time.

Modifying Existing Data

This last attack type is theoretical. A very large amount of computer security exploits deal with modifying existing data records. This paper is meant to discuss the modification of existing records through passing incorrect variables to a level-two program.

The systems identified in this study did have security restrictions on modifying existing data. However, it is possible and likely that systems exist which are susceptible to this attack type.

Since database records are usually stored with a unique identification number, it is possible that poorly coded web pages or the underlying software would actually allow database record numbers to be specified in an attack.

This may allow a record to be overwritten.

Under some circumstances a second record could be created that had the same identity as the first. Since this type of scenario would likely be unforeseen by the developer, the result is completely unpredictable and would vary from system to system.

The Actual Attack: Identifying Form Variables in HTML

The following HTML code is from a web site contact form. Code has been removed to simplify the example. For this example, let's assume this code was taken from http://www.abcd.com/contact/contact.php

```
<FORM method="post" action="email.php">
 Your Name: <INPUT TYPE="text" NAME=”Customer” SIZE="30" value=””>
<INPUT TYPE="submit" NAME="submit" VALUE="Send Your Comment.">
</FORM>
```

Analysis of the Form

1. First this is a FORM for accepting input, because the code is between a <FORM> and </FORM> tag.

2. Secondly, this information is “posted” to a file called “email.php” which resides in the same directory as this web page.

3. Lastly, the <INPUT> tags identify the data type of each variable, as well as the variables NAME.

The Actual Attack: Passing Data Directly to the Second Level

Using the HTML sample above:

We want to pass data to http://www.abcd.com/contact/email.php

This can be done in any standard web browser. This is accomplished by loading the page with variables specified after the page location. In the address bar with the example above we would write.

http://www.abcd.com/contact/email.php?Customer="John%20Smith"&submit=1

Notes:

- In HTML spaces are replaced with the string "%20" (without quotation marks).
- To pass variables a "?" is used for the first variable, and "&" for every variable afterwards.
- If the example above did not list email.php as the POST ACTION, and instead had empty quotation marks – we would want to post this data directly to the original page "contact.php"

The Actual Attack: Passing Changing Data Multiple Times

The following program is written in PHP. For those unfamiliar with PHP the connection between PHP and HTML in the same page may be a little confusing. I recommend the PHP tutorial available at php.org. This example generates a web page that reloads itself every second, and passes mostly random information to a target site on every reload. The only variable that is not random information is the comment.

Figure 1:

```
<?php
rand(0,1000);
function randstring($min,$max) {
 $len=rand($min,$max);
 for($i=0;$i<$len;$i++) $rval.=chr(rand(97,122));
       return $rval;
}

?> <!–The following variables are going to be passed to the second level, the actual
variable names the second level expects are farther below in the form
'contact_form[name]' →

<?php
$name=randstring(3,5);
$company=randstring(5,10);
$fromemail=randstring(5,15).'@'.randstring(5,10).'.com';
$city=randstring(7,20);
$comment="This%20is%20a%20comment."; ?>

$thisfile = "[Insert URL to this web page]";
$targetfile = "[Insert URL to target page]";
?>
<html>
<head>
<META HTTP-EQUIV=Refresh CONTENT="1; URL=<?php echo $thisfile; ?>">
</head>

<frameset rows="50%,50%">

  <frame>

Email Sent:<br /><br />
Name: <? echo $name; ?><br />
Company: <? echo $company; ?><br />
From Email Address: <?echo $fromemail; ?><br />
City: <? echo $city; ?><br />
Comment: <? echo $comment; ?>

<FRAME src="<?php echo $targetfile; ?>?contact_form[user_name]='<?php echo $name;
?>'&contact_form[company]='<?php echo $company; ?>'&contact_form[email_address]='<?php
echo $fromemail; ?>'&contact_form[city]='<?php echo $city;
?>'&contact_form[question]='<?php echo $comment; ?>'&submit=1">
</FRAME>
</FRAMESET>

</html>
```

The Actual Attack: Passing Variable Data Beyond it's Intended Scope

The following example has been discussed briefly in the section, "Vulnerability of Online Polls and Voting Systems". The following example is the string to manually enter in a web browser for a single submission.

Figure 2:

An attacker has visited the web page http://www.abcd.com/vote.php and viewed the source code in Internet Explorer by clicking on, "View", "Source". As explained above, the attacker wrote down variable names and the variable types. The attacker discovered information is passed directly back to the original file, not a second file.

There are three variables in this example, "vote", "record", and "submit". Submit is always set to 1 when passing data back.

The example page accepts votes for files in a shareware database. The different programs available have a unique identifying record number. Votes are allowed between 1 and 10.

To Spoof voting data:

http://abcd.com/vote.php?vote="120"&record="87"&submit=1

This string would vote 120 for record #87. Obviously the intended scope for votes is 1 to 10, but in this example the attacker attempts to vote 120. If the attacker then checks the resulting votes for that project and finds a dramatically increased poll result – he knows the hidden processor of this voting system did not check to ensure votes were only between 1 and 10.

The Actual Attack: Passing Invisible Variables to PHP and Analyzing Results (Advanced Attack Method, explanation is for developers only)

As this paper has discussed, when visiting a web site that utilizes PHP for processing – the PHP code is never sent to the visitor. In analyzing HTML forms we can discover variables that the PHP script uses. The developers intention is to have these variables modified by the user according to a certain set of restrictions: e.g. Variable data should be within a certain scope.

The majority of PHP programs use additional variables the visitor is never meant to see or modify. These variables are completely hidden, and discovering them is very hard without having access to the PHP code. We will refer to these as, "isolated variables" or variables that are only used within the second-level.

Attacks against isolated variables are much harder to perform, and the outcome is completely unknown. This means even a successful attack might go unrecognized by the attacker, so that they do not know how to recreate that successful attack to their advantage.

With a background in software development, it may be possible to guess where isolated variables are being used in the second-level. For example, if the output from the second-level most likely required a WHILE loop, it is possible that WHILE-LOOP used an isolated variable. Usually simple parts of a program will utilize very common variable names, such as, "a", "x", "loop", or "flag".

Furthermore, these simple isolated variables will not be initialized with a starting value inside the program. If an attacker could modify these variables before they are called on in the program, the attacker could cause unintended results.

Take the following example of a pseudo-code WHILE-LOOP utilizing a simple isolated variable.

```
WHILE $X != 5
  $X = $X + 1
  PRINT $X
REPEAT
```

This Pseudo-Code program assumes that X has a starting value of 0, and is intended to print out the numbers 1 to 5.

If an attacker started this program with $X equal to the value 6, it would never reach its exit condition and consequently would loop forever.

Real second-level software often has much more complicated examples, and changing unknown variables would have completely unpredictable results.

To perform an attack of this nature, an attacker could utilize a script as shown in figure 1, modified to pass additional variables with random names **and** random values.

The Actual Attack: Just Deleting Cookies

Some systems rely on security through cookies. A cookie must be set on the visitor's computer before they can vote, and then after voting that cookie remains in place to identify the user as having already voted.

Because the cookie must be set for voting, an automated program must also be able to submit back valid cookie information for each vote. However, if cookies are deleted these systems will treat the same attacker as a new user and allow them to vote again – possibly after a second registration process.

Attacking systems relying on cookies for voting takes more work, but it is still very possible. Where a single record may only have a few votes, an attacker only needs to manually delete their cookies and register a few fake accounts to double the number of votes for that record.

Guarding Against Invalid Data Submissions

The following recommendations cover guarding against all attacks discussed. It is up to each web developer to decide which recommendations are most appropriate for their site, depending on the sensitivity of data their site handles and its respective vulnerability to attacks.

- All variables passed to the second-level should be checked for scope, even though the web site may only allow a certain scope when passed through regular means.
- Variables in second-level programs should always be initialized with a starting value, and ensure exit conditions occur even when the variable manages to be out of scope. E.g. where loops are required the exit condition should utilize the less than or greater than operators rather than the equal to operator.
- Single record-submission sites should record IP addresses, possibly even MAC addresses of visitors to ensure they submit information only once.
- Cookie's identifying a user to ensure one-time only submissions are not very effective against advanced attacks. However they should be utilized in conjunction with IP records.
- Second-Level programs should record the average submission rate over time and notify an administrator immediately if abnormally high data submissions occur.
- Human Only Authentication Systems are very effective against automated, repeat submission attacks. They work by displaying a picture of numbers or letters and requiring the user manually enter the text they see into a form to prove they are human. This should not be considered an end-all solution.
- A complicated protection method involves using changing variable names generated by a PHP script. The PHP script generates random variable names and ONLY accepts data back from the variable name generated in that instance. This is for more advanced developers on systems particularly vulnerable to repeat submission attacks.

Conclusion:

The amount of online systems vulnerable to these attack methods is unknown. From personal experience, I would guess a very large number of systems are vulnerable in some way to attack methods listed in this paper. The relatively small number of systems actually passed invalid data for the purposes of this study were not effected in a serious way. It is interesting to note however that the systems analyzed for this study were the first chosen. No system encountered has been invulnerable to either out-of-scope data sets, or repeat submission attacks.

It would have been interesting to test a system for vulnerability to randomly guessing isolated variable names.

If you would like more information, please feel free to contact myself or SolidBlue Software Inc.

Doug W. Styles
[**President**] SolidBlue Software Inc.
email: doug at solidblue.biz