Andrew D. Pitt

CIS 4360

Professor Fisher

May 1, 2018

Vulnerability Report

* **SQL Injection**

WebGoat is susceptible to SQL injection on its login page (as well as 10 other pages) because it utilizes a SQL database in the backend, and it has not thoroughly protected itself against SQL injection. Using special logical statements, the attacker is able to determine whether or not such a SQL database exists. If it exists, they can then insert malicious code into a statement and parsed by the SQL server for execution. The attacker can gain access to the SQL database, the data from which they can read, modify, copy, encrypt, or even shut down the whole database. This is marked a high priority alert due to the severity of aforementioned potential damage.

*Mitigation: Prepared statements*

As a primary defense against SQL injection, businesses should use prepared statements with parameterized queries, “which ensure that an attacker is not able to change the intent of a query, even if SQL commands are inserted by an attacker.” In this way, the input is scrubbed and the server and database are protected against probing. Stored procedures are also similarly helpful.

*Pseudocode- prepared statement*

Connect to database  
Generate query such as query = “SELECT username FROM usernames WHERE username = ?”

Prepare new statement(query, connection)  
Bind parameters  
Execute

*Mitigation: Whitelist input validation*

Another defense is to white list input validation, or to create a list of trusted values and block all other cases.

*Pseudocode*

if(input value matches case)  
 Output requested corresponding value  
else  
 Output: “value does not match case”

*Mitigation: Least privilege*  
 Least Privilege is another important characteristic to include in order to protect against SQL injection. Least privilege is a backup measure in case input scrubbing fails and the attacker gains access. Applying least privilege to database accounts ensures that the attacker will have limited power.

* **Path Traversal**

A path traversal happens when a security misconfiguration is present, and a user or attacker is able to manipulate the URL in their browser to access data as files, directories, and commands included in the environment outside the CGI root. This puts contents stored on the server at risk. Since WebGoat uses HTTP it made itself vulnerable to path traversal and has not taken steps necessary to protect against that threat.   
  
 One way to reduce path traversal is to restrict behavior to the CGI root domain, but this does not necessarily protect against other Unicode-encodings of the URL. All possible user input must be properly handled.

This vulnerability is has major potential to compromise assets, especially considering the that Zap reported a vulnerability in Application Error in WebGoat. If this error supplies sensitive path information, the attacker or an unsuspecting user has a roadmap of sensitive resources which they can simply feed into the URL in order to gain access. In other cases, an attacker may know what resource they are looking for, and so they can guess or programmatically find the URL of the resource they want.

*Mitigation: Separate sensitive files from public facing domains* The business must take steps to separate sensitive data from public facing domains. They can accomplish this by storing configuration files and other sensitive resources outside the web root and simply including the file.

*Pseudocode*  
include(“$\_SERVER[DOCUMENT\_ROOT/.. /config.php”) /\*config file stored in higher directory\*/

*Mitigation: Proper URL input handling*

In terms of input scrubbing, businesses should look for classic path traversal input such as something like \..\..\ when they handle the URL. They should take steps to block such requests perhaps through a whitelist of resources to be accepted as public-facing, or a blacklist of resources to be denied as public-facing (although a blacklist is more fragile and thus less secure than a white list).

*Pseudocode*  
Traverse/iterate through URL input

Match occurrences of acceptable URLs (say yes only to “/” style URL)  
Block all other cases  
  
 Other steps to protect assets to take include returning indices in a URL rather than filenames, understanding how the OS will interpret filenames supplied from input, internally ensuring that the user can’t supply all parts of the path, and following a strict resource naming protocol.

* **Buffer Overflow**

Buffer overflow can occur by an attacker sending the server large input (such as an unsigned int instead of a signed int), which causes the program to write beyond the memory allocated for that buffer. If the input is precise enough, attackers can target and access specific memory contents. Since it’s a memory management issue, certain languages and frameworks are more susceptible to buffer overflow than others, such as the C programming language. Businesses should use strongly-typed languages/frameworks that don’t carry this issue (such as such as Java, .Net, PHP, Python, and Perl. Improper foresight and preparation for buffer overflow on behalf of the business can cause the server to be compromised or to execute code that was not intended for public use. There is also potential to deny service, or display sensitive error messages which give the attacker a way in.

*Mitigation: range checking*  
 One way to prevent buffer overflow is to use range checking on input values to prevent overflow. How this is accomplished may vary based on the implementation, but in terms of a more vulnerable language like C, we could use something like the following:

*Pseudocode- line by line example*

char buffer[1024];

fgets(buffer, sizeof buffer, stdin);

or alternatively:

*Pseudocode- character by character example*  
BUFFSIZE = 1024;

buffer[BUFFSIZE] = {0};

int ch;

int i = 0;

while ((ch = getchar()) != /n or EOF && i < BUFFSIZE -1)  
 buffer[i++] = ch;

*Mitigation: use safe libraries* Buffer overflow is common in C because some of the library functions expose low level details of buffers. A secure development practice entails avoiding standard library functions which are not bounds checked, such as gets, scanf, and strcpy. Decent alternatives are fgets and strncpy.

*Mitigation: no stack execution*If the business turns off stack execution, any buffer overflow will be returned as a segfault. It’s best to use systems that have this capability which help further mitigate buffer overflow.

Other mitigations include use of Stackguard/Pointguard (or similar solutions) to check during runtime if buffer overflow has occurred by verifying if the stack has been tampered with.