CS-527 Software Security

Web security

Asst. Prof. Mathias Payer

Department of Computer Science
Purdue University

TA: Kyriakos Ispoglou

https://nebelwelt.net/teaching/17-527-SoftSec/

Spring 2017

Table of Contents

Dynamic web pages

- Dynamic web pages execute code on the server.
- This allows the web server to add content from other sources (e.g., databases) and provide rich interfaces back to the user.
- Build and combine complex parts dynamically and send the final result to the user (e.g., a content management system that loads contents from the database, intersects it with the site template, adds navigation modules and other third party modules).

Command injection

display.php

```
<html><head><title>Display a file</title></head>
<body>
<? echo system("cat ".$_GET['file']); ?>
</body></html>
```

- How can this script be attacked?
- `display.php?file=info.txt%3bcat%20%2fetc%2fpasswd`
- `;` allows chaining of individual commands.
- `system` is a powerful command that executes full shell scripts.

Command injection mitigation

- Can we just block ;?
- Blacklisting is *usually* not a good solution as attack space may be infinite.
- What about using a pipe.
- What about using a backtick.
- What about using alternative commands (e.g., cat instead of rm).
- Even the shell itself has many inbuilt commands.

Mitigation through validation

- Ensure that the filename matches a set of allowed filenames.
- Non-alphanumeric characters are needed to execute alternate functionality.
- Fix both directory and set of allowed files.
- Disallow special characters in the file name.

Mitigation through escaping

- Escape parameters so that interpreter can distinguish between data (channel) and control (channel).
- Escaped form: `system("cat 'file.txt'")`.
- How do you write such an escape function?
- You don't – there's a huge potential for error. Use built-in ones.
- Each language has its own flavours of escape functions.

Mitigation through reduction of privileges

- The system command is immensely powerful as it launches a new shell interpreter.
- Fall down to simplest possible API: open the file yourself and read it into a buffer or, if you *must* execute a command, launch it directly and not through the shell.

Generalized injection attacks

- What enables injection attacks?
- Both code and data share the same channel.
- In the system example above, cat and file are specified as part of the same "shell script" where ; starts a new command.
- In code injection the data on the stack and the executed code share the same channel (as do code pointers).

Table of Contents

SQL injection example

User authentication

```
$sql = "SELECT * FROM users WHERE email='" . $_GET['email']
. "' AND pass='" . $_GET['pwd'] . ';"
```

- What is wrong with this query?
- An attacker may inject ' to escape queries and inject commands.
- (Also, the password is not hashed but stored in plaintext.)

More SQL injection

- An attacker may set email to "root@acme.com'–".
- Or "root@acme.com'; DROP TABLE users;–".
- Or insert a new user into the database.
- SQL injection is, in spirit, the same attack as code injection or command injection.

SQL injection mitigation

- The same as before: validation, escaping, or reduction of privileges.
- Always ensure to check/validate/protect *all parameters*.
- Validation checks for valid input of values.
- Escaping ensures that illegal values are escaped.
- Reduction of privileges is best strategy.

SQL injection mitigation

- Separate control and data channel: prepared SQL statements.
- Similar to printf, define "format" string and supply arguments.
- Supported by all web frameworks and DB connectors.
- `sql("SELECT * FROM users WHERE email=$1 AND pwd=$2", email, pwd)`

Table of Contents

Cross Size Scripting (XSS)

Definition

XSS allows an attacker to inject and execute JavaScript (or other content) in the context of another web page (e.g., malicious JavaScript code that is injected into the banking web page of a user to extract user name and password or to issue counterfeit transactions).

The XSS family

1. Persistent/stored XSS (i.e., code is stored at the server).
2. Reflected XSS (i.e., the code is reflected through the server but not stored there).
3. Client-side XSS (i.e., the code leverages a client-side vulnerability).

Persistent XSS

- The attacker stores the attack data on the server itself.
- A simple chat application allows users to store arbitrary text that is then displayed to other logged in users.
- The attacker may send a message that contains “
  `<script>alert('Mr. Evil here');</script>`
  ”.
- This code is then executed on all logged in clients when the server sends them the chat message of Mr Evil.
- Common use case: feedback forms, blog comments, or even product meta data (you don't have to see it to execute it).
- The bug is on the server side (usually missing sanitization).

Reflected XSS

- The attacker encodes the attack data in the link that is then sent to the user (e.g., through email or on a compromised site).
- A web interface may return your query as part of the results (i.e., "Your search for 'query' return 23 results.").
- This attack requires the user to follow the attacker controlled link!
- This attack is often possible in search forms or other input that is not validated/cleaned.
- Many scripts also forget to screen both GET and POST requests.
- The bug is on the server side (usually missing sanitization).

Client-side XSS

- Larger applications contain a lot of JavaScript code. This code itself may also contain vulnerabilities on the *client side*.
- JavaScript may use URL parameters to, e.g., display user names or process information (think JSON requested data that is processed in the background).
- Attacker must make user follow the compromised link but, compared to reflected XSS, the server does not embed the JavaScript code into the page through server side processing but the user-side JavaScript parses the parameters and misses the attack.
- Missing input validation and sanitization in the client side code allows the attacker to inject code into the client-side processing.

Table of Contents

Summary

- The web can be attacked through different attack vectors, not just memory corruption.
- Command injection allows an attacker to inject shell commands into the execution flow.
- SQL injection allows an attacker to inject SQL commands from a web application into the database.
- XSS allows the attacker to execute malicious JavaScript code in the context of another web application.
- All three types of attacks share the common problem that code and data are not separated.

Questions?

?