INF226 – Software Security

Håkon Robbestad Gylterud

2019–10–23

CRSF Tokens on login forms

Classic cross-site request forgery

Web form, as sent to browser:

```
<form action="/url/profile.php" method="post">
  <input type="text" name="firstname"/>
  <input type="text" name="lastname"/>
  <br/>
  <input type="text" name="email"/>
  <input type="submit" name="submit" value="Update"/>
</form>
```

Server-side handling:

```
session_start();
// Check session cookie
if (! session_is_registered("username")) {
   echo "invalid session detected!";
   [...]
   exit;
}
update_profile();

function update_profile {
  SendUpdateToDatabase($_SESSION['username']
                       , $_POST['email']);
[...]
  echo "Your profile has been updated.";
}
```

Meanwhile on a different website. . .

https://attacker.com/attack/:

```
<SCRIPT>
function SendAttack () {
  form.email = "attacker@example.com";
  form.submit();
}
</SCRIPT>

<BODY onload="javascript:SendAttack();">
<form action="http://victim.example.com/profile.php"
      id="form" method="post">
  <input type="hidden"
         name="firstname" value="Funny">
  <input type="hidden"
         name="lastname" value="Joke">
  <br/>
  <input type="hidden" name="email">
</form>
```

Confused deputy and the session cookie

CSRF tricks the browser to use its session cookie to approve actions initiated by a third party site.

Confused deputy and the session cookie

CSRF tricks the browser to use its session cookie to approve actions initiated by a third party site.

Best practise defense

- Add **anti-CSRF tokens** on any form.
- Make sure that any authority-bearing token-cookies (such as session cookies) have the **`SameSite` flag** set to `strict` (or `lax` if GET requests do not have **any** side effects)

Question

But, what about *login forms*? There is no session cookie, yet?

Example 1: Internet routers/modems

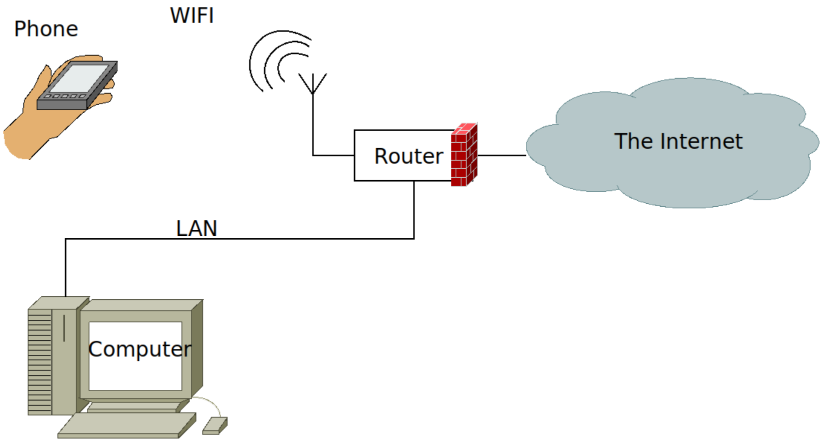
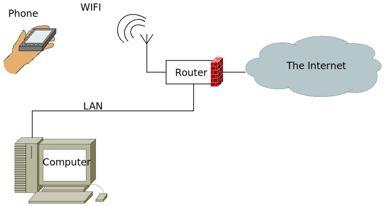


Figure 1: The router

Example 1: Internet routers/modems



- The router has a configuration interface. (Basically, a wep application)
- The router often have insecure default passwords
- . . . but only allows access from local network.

Cross site-request forgeries will allow an attacker to send requets to the router originating from the local network.

Example 2: Google search history

While you are logged in, Google collects data on your search history – and you can access that data later.

Example 2: Google search history

While you are logged in, Google collects data on your search history – and you can access that data later.

But what if you (somehow) get logged in as another user? Then that user can access your search history later.

Example 2: Google search history

While you are logged in, Google collects data on your search history – and you can access that data later.

But what if you (somehow) get logged in as another user? Then that user can access your search history later.

CRSF on the login form will allow an attacker to log you into an account which they control.

Mandatory assignment

The third mandatory assignment

Is available on `git.app.uib.no`.

The third mandatory assignment

Is available on `git.app.uib.no`.

The pitch:

- Your organisation has developed a forum, but the programmers knew nothing about software security.
- After several security incidents, you (an expert on security) is given the task to improve the security of the web application.
- The most critical security issues have been identified, and you now have a task-list of things to improve.

Demo

First we look at it from a user perspective.

Demo

First we look at it from a user perspective.

Then from an attacker perspective.

CERT Top 10 Secure Coding Practises

Practise defence in depth

- Keep the number of linchpins down.
- Plan for failure of individual components.
- Program defensively.

Validate input

Regard all input with suspicion!

- Map the surface of the program and identify all input points.
- Formulate explicit descriptions of the possible inputs. (Protocol, format, $\cdots$)
- Validate input according to these descriptions.

Be strict in what you accept, and even stricter in what you output!

Sanitize data to other systems

Covers large classes of vulnerabilities:

- SQL injection
- XSS
- Command injection
- File paths

(Basically, any point where strings are concatenated!)

Sanitize data to other systems

Covers large classes of vulnerabilities:

- SQL injection
- XSS
- Command injection
- File paths

(Basically, any point where strings are concatenated!)

Whenever a string is transfered:

- Identify the protocol or format.
- Identify which parts of the string come from untrusted sources.
- Sanitize (escape) the data appropriately.

Deny by default

Base access control on denying by default, and describe the allowed cases carefully.

This also goes for functions and code.

```
Result function(arugments···) {

  if(···) {
     return DENIED;
  }
  return ALLOWED;
}
```

Is worse than:

```
Result function(arugments···) {

  if(···) {
     (···)
     return ALLOWED;
  }
  return DENIED;
```

Because the first one can quickly develop into:

```
Result function(arugments···) {
  try {
    if(···) {
       return DENIED;
    }
  } catch (Exception e) {···}
  return ALLOWED;
}
```

Adhere to the principle of least priviledge

(We discussed this already)

Archtect and design for policy enforcement

(Example i code)

Keep it simple

Complex security is likely to be weak security.

Keep it simple

Complex security is likely to be weak security.

If you find that you are making a lot of *ad hoc* additions or exceptions, you may have to rethink the fundamental design.

Keep it simple

Complex security is likely to be weak security.

If you find that you are making a lot of *ad hoc* additions or exceptions, you may have to rethink the fundamental design.

How have other's solved similar challenges?

Adopt a secure coding standard

How to make secure and correct programs vary from language to language and platform to platform.

Always make your self familiar with how security challenges are tackled on your platform.

Adopt a secure coding standard

How to make secure and correct programs vary from language to language and platform to platform.

Always make your self familiar with how security challenges are tackled on your platform.

Misunderstanding the security mechanisms on your platforum leads to security holes!

Heed compiler warnings

Crucial in the case of C or C++, where undefined behaviour will violate security.

Also important for other languages.

Use effective quality assurance tools

We have discussed dynamic and static progam analysis. Also worth checking out:

- Fuzzers
- Property based checking

Muddest point

Answer on `mitt.uib.no`.