ECE568 Lab #3: Web Application Security

Introduction

In this lab you will practice common web vulnerabilities such as cross-site scripting and cross-site request forgery. A working knowledge of javascript is required. Various, javascript references can be found on the web. Further clarifications and hints are posted in the FAQ below.

Lab 3 should be done in groups of 2. It will be due at 11:59pm on Nov 16, 2015.

Your Task

The fictional "Zoobar Foundation" has set up a simple web application at zoobar.csl.toronto.edu, allowing registered users to post profiles and transfer "zoobar" credits between each other. Each registered user starts with 10 zoobars. You will craft a series of attacks on zoobar.csl.toronto.edu that exploit vulnerabilities in the website's design. Each attack presents a distinct scenario with unique goals and constraints, although in some cases you may be able to re-use parts of your code.
Although many real-world attackers do not have the source code for the web sites they are attacking, you are one of the lucky ones: source code is available. You won't actually *need* to look at the site's source code, but it's there if you get stuck. Your attacks will run in a restricted network environment that can only connect to zoobar.csl.toronto.edu and ece568.csl.toronto.edu. We will run your attacks after wiping clean the database of registered users (except the user named "attacker"), so any data you submitted to zoobar.csl.toronto.edu while working on the assignment will **not** be present during grading. **We reserve the right to delete users from the zoobar.csl.toronto.edu database at any time if it gets too large, so please keep local copies of any important data you submit there.**

Setup

Mozilla Firefox. We will grade your project with default settings using the version of the Mozilla Firefox browser that is installed on ECF at the time the lab is due. We chose this browser for grading because it is widely available and can run on a variety of operating systems. There are subtle quirks in the way HTML and JavaScript are handled by different browsers, and some attacks that work in Internet Explorer (for example) may not work in Firefox. In particular, you should use the Mozilla way of adding listeners to events. We recommend that you test your code on ECF before you submit, to ensure that you will receive credit for your work.
Email script. For Attacks A and C, you will need a server-side script to automatically email information captured by your client-side JavaScript code to the TAs for grading. We have provided this script for you. Please review the instructions at http://ece568.csl.toronto.edu/a3/sendmail.php and use that URL in your attack scripts to send emails. You may send as many emails as you like while working on the project, but please do not attack or abuse the email script.

Attack A. Cookie Theft

- Your solution is a URL starting with http://zoobar.csl.toronto.edu/users.php?
- The grader will already be logged in to zoobar.csl.toronto.edu before loading your URL.
- Your goal is to steal the document cookie and email it to yourself and grader@zoobar.csl.toronto.edu using the email script.
- Except for the browser address bar (which can be different), the grader should see a page that looks **exactly** as it normally does when the grader visits users.php. No changes to the site appearance or extraneous text should be visible. Avoiding the **red warning text** is an important part of this attack. (It's ok if the page looks weird briefly before correcting itself.)
- The attack URL will contain some escaped characters. Make sure to escape only the necessary characters. Solutions with all characters escaped will not get credit for this part.
- Hint: Here is an example attack to use as a starting point.

Attack B. Cross-Site Request Forgery

- Your solution is a short HTML document that the grader will open using the web browser.
- The grader will already be logged in to zoobar.csl.toronto.edu before loading your page.
- Transfer 10 zoobars from the grader's account to the "attacker" account. The browser should be redirected to http://ece568.csl.toronto.edu/ as soon as the transfer is complete (so fast the user might not notice).
- The location bar of the browser should not contain zoobar.csl.toronto.edu at any point.
- Hint: Cross-site request forgery was described in lecture.

Attack C. Password Theft

- Your solution is a short HTML document that the grader will open using the web browser.
- The grader will not be logged in to zoobar.csl.toronto.edu before loading your page.
- Upon loading your document, the browser should immediately be redirected to http://zoobar.csl.toronto.edu/ and this is the address that should appear in the location bar. The grader will enter a username and password and press the "Log in" button.
- When the "Log in" button is pressed, send the username and password (separated by a comma) using the email script.
- The login form should appear perfectly normal to the user. No extraneous text (e.g. warnings) should be visible, and assuming the username and password are correct the login should proceed the same way it always does.
- Hint: The site uses htmlspecialchars() to sanitize the reflected username, but something is not quite right.

Attack D. Profile Worm

- Your solution is a profile that, when viewed, transfers 1 zoobar from the current user to a user called "attacker" (that's an actual username) and replaces the profile of the current user with itself.

- Your malicious profile may include a witty message to the grader (optional, but it helps us see that it replicated).
- To grade your attack, we will cut and paste the submitted profile file into the profile of the "attacker" user and view that profile using the grader's account. We will then view the copied profile with more accounts, checking for the transfer and replication.
- The transfer and replication should be reasonably fast (under 15 seconds). During that time, the grader will not click anywhere.
- During the transfer and replication process, the browser's location bar should remain at http://zoobar.csl.toronto.edu/users.php?user=**username**, where **username** is the user whose profile is being viewed. The visitor should not see any extra graphical user interface elements (e.g. frames), and the user whose profile is being viewed should appear to have 10 zoobars.
- You will not be graded on the corner case where the user viewing the profile has no zoobars to send.
- Hint: The site allows a sanitized subset of HTML in profiles, but you can get around it. This MySpace vulnerability may provide some inspiration.

Deliverables

Create files named a.txt, b.html, c.html, and d.txt, containing each of your four attacks. For each attack, please explain in at most 100 words what you did. These explanations must be in the file explanations_lab3.txt. This file must also contain the names and student numbers of your group members prefixed by #. Do not prefix other lines by # as this would confuse the automated marking scripts.

```
#first1 last1, studentnum1, e-mail address1
#first2 last2, studentnum2, e-mail address2
```

It is very important that this information is correct as your mark will be assigned by student number, and the e-mail you give here will be how we will get the results of the lab back to you.

Submission is done using the ECF submit command:

```
submitece568f 3 a.txt b.html c.html d.txt explanations_lab3.txt
```

Grading notes

Beware of Race Conditions: Depending on how you write your code, all four of these attacks attacks could potentially have race conditions that affect the success of your attacks. Attacks that fail on the grader's browser during grading will receive less than full credit. To ensure that you receive full credit, you should wait after making an outbound network request rather than assuming that the request will be sent immediately.

Honor code

In this assignment, we are asking you to craft attacks to further your understanding of web application security and the consequences of poor input validation. It is highly illegal and unethical to send malicious code to unwitting recipients. Do not distribute your attacks outside your group, and choose non-obvious usernames on zoobar.csl.toronto.edu so that other groups will not accidentally stumble upon your work.

FAQ

General

Is register_globals enabled on the web server?
No, we've turned it off on zoobar.csl.toronto.edu. The login page is vulnerable to an attack if register_globals is on.
Why do I get an error when I try to visit login.php with my web browser?
Actually, login.php isn't intended to be viewed directly. It's just a library of functions that get called if the user tries to view one of the other pages and isn't logged in yet.
Um... Same Origin Principle? Do I need to know what that is?
You should be familiar with the Same Origin Principle before attempting this project. The Same Origin Principle is the most important idea behind web application security; ignoring it will cause you to run into lots of browser security exceptions.

Resources

Where can I find more information about JavaScript?
You can find a lot of references on Google. There's a decent one at W3Schools. Pay particular attention the DOM examples.
My JavaScript isn't working, and I don't know why. What can I do?
First, you should look at the page source to make sure that the javascript executed by the browser corresponds to what you believe you have injected. You can see the source of a page via Ctrl+U in Firefox. In addition, two extremely useful tools for debugging in Mozilla Firefox are the JavaScript console and the DOM Inspector. Both can be accessed from the Tools menu. The JavaScript console lets you see which exceptions are being thrown and why. The DOM Inspector lets you peek at the structure of the page and the properties and methods of each node it contains. (If the DOM Inspector isn't installed, make sure it's selected when you install Mozilla Firefox.) You might also want to try Firebug.
What do I need to know about CSS?
You only need to know enough to make your attacks disappear. You should know what basic syntax like <style>.warning{display:none}</style> means, and you should feel free to use stealthy attributes like style="display: none; visibility: hidden; height: 0; width: 0; position: absolute" in the HTML of your attacks. Beware that frames and images may behave strangely with display: none, so you might want to use visibility: hidden instead.

How can I see cookies and form data that the browser sends?
Try the LiveHTTPHeaders browser extension or HttpFox (my recommendation).
Where can I find more information about PHP?
The definitive resource on PHP is php.net. You can find some introductory tutorials there.

Hints

Am I allowed to load scripts or images from other domains?
No, your attacks should not load data from domains other than ece568.csl.toronto.edu and zoobar.csl.toronto.edu. For example, this file is ok, but this one is not. We are enforcing this policy so that the submission deadline is a hard one.
Are we allowed to include additional files?
Please limit yourself to the files requested. One file per attack (a.txt b.html c.html d.txt), the explanations file (explanations_lab3.txt).
Should we submit a tar file?
No, that's not necessary.

Attack A.

How are the graders going to test our URL?
We will put it into the browser's address bar and click the "Go" button.
The example attack doesn't seem to do anything. What's wrong?
You need to be logged in to zoobar.csl.toronto.edu before the attack will work. When you click the link, you should get a browser alert with the contents of document.cookie.
Why would someone want to steal document.cookie?
The cookie is the user's authentication credential. If you steal someone else's cookie, it is easy to hijack that user's session (although we won't ask you to do so in this project).
How am I supposed to email document.cookie? I didn't know browsers could send email.
You have to convince the browser to send a GET request to the ECE 568 email script, which will cause an email to be sent by the ece568.csl.toronto.edu server to you and the grader. The grader gets the email automatically, so just put a group member's email address (or a comma-delimited list of addresses) in the "to" field.
How do I convince the browser to send a GET request to an URL of my choosing?
The email script provides some example code. You allocate a JavaScript Image object, and set its src attribute to be the URL you want the browser to retrieve. The browser immediately tries to fetch the content of this image, even though the URL isn't actually pointing at an image and the image hasn't been inserted into the page anywhere. Pretty sneaky.
How do I put code into a URL? I want to have newlines and stuff.
It helps if you URL-encode it (Google "URL encoding").
Should I include the javascript:void part in my code?
If you do, your attack might still work, but it's not necessary if you're already within a <script> tag. Just copy the part starting with `(new Image()).src=...`
What's up with this random query parameter?

If you try your attack more than once, the browser might think that it already has the image in its cache, and so it wouldn't send a second GET request. Adding a random number to the end of the URL ensures that the browser will think the new URL is different, and won't use the cache. The email script ignores this parameter, so it's only useful for bypassing the cache.

Why are the characters reflected back different from the ones in the URL?

Your query parameter is URL decoded by the server before being reflected back at the user. You'll need to made sure that your attack code is URL encoded. For example, use + instead of space and %2b instead of +. Here is a URL encoding reference and a handy conversion tool.

Are there any restrictions on the length of my solution URL?

No. Internet Explorer has a limit of 2083 characters or so for URLs, but the limit is much longer in Firefox, which is what we are using for grading. The grader's solution was 297 characters, and shorter solutions are possible.

Since sending an email is a side effect of the request, wouldn't it make more sense for the browser's request to be a POST instead of a GET?

Yes, but that would prevent the Image trick from working. Real-world attackers are rarely bothered by such semantic distinctions.

What should the email look like if the attack worked?

```
The parts that might be different are in red.
From: Apache <apache@zoobar.csl.toronto.edu>
To: grader@zoobar.csl.toronto.edu, youremail@utoronto.ca
Subject: Message from group 'yourgroupid'

Payload:

ZoobarLogin=YToyOntpOjA7czo4OiJhbnRpZ3JhdiI7aToxO3M6MzI6
ImZlOGIzMTM3YjgyNTFlMDQ2YTIzMzRkNTgxOWM2YWZiIjt9=
```

I'm afraid of spamming the grader's email account. Should I be careful not to test the attack until I'm ready?

Don't worry, we will be ignoring emails sent to grader@zoobar.csl.toronto.edu until we start actually grading the assignment.

My attack is working. What should I do to make it invisible to the user?

The text box should be at its usual size and in the normal place. No warning text or characters that are normally part of the page should be visible. From the point of view of the visitor, it should appear as if they just went to users.php and didn't put in a username yet (with the possible exception of the address bar, which can be whatever you want). It's ok if the page briefly looks weird before correcting itself. Don't worry about the fact that the header at the top of the page changes with each page view — that's normal site behavior.

That sounds hard.

There are actually several quick and easy ways to do it, so try to think outside the box on this one. If you can't figure it out, try moving on to the other attacks and come back to this one when you're done.

Attack B.

Is this a cross site scripting attack?

No, this is a cross site request forgery attack. You are exploiting the fact that zoobar.csl.toronto.edu uses only a cookie to authenticate requests, even ones with side effects.

Can I use the vulnerability in user.php from Attack A?

No. Since this not a cross site scripting attack, you do not need to use the vulnerability in user.php. All you have to do convince the user's browser to post malicious form data to transfer.php.

How do I convince the browser to send a malicious POST request to transfer.php?

Put together a form in your HTML document, with http://zoobar.csl.toronto.edu/transfer.php as the action attribute.

How can I get the form to be submitted with no user interaction?

You can call the "Send" button's click method. Or, you can use JavaScript to call the form's submit method. Keep in mind that the site is looking for a parameter named "submission" in the form data, so if it's not there, the transfer won't happen.

What <input> fields should the form contain?

Use the browser's view source function on transfer.php and you'll get a pretty good idea.

How can I submit a form to zoobar.csl.toronto.edu without causing the browser's address bar to change to zoobar.csl.toronto.edu?

Create a hidden <iframe> and make sure the form's target attribute matches the frame's name attribute.

Uh oh, iframes. Will the grader have third-party cookie blocking enabled?

No, third party cookie blocking will not be enabled. It's probably off by default on your browser, but if your attack isn't working because the login cookie isn't sent, you might want to check to make sure. Edit > Preferences > Privacy > Cookies > Uncheck "for the originating site only"

How do I make the iframe hidden?

There are lots of ways to do it, but the easiest is probably `<iframe style="visibility: hidden" ...>`.

How do I redirect the browser to the ece568.csl.toronto.edu home page?

Change the document.location property. Note that it is **required** for the browser's address bar to change to http://ece568.csl.toronto.edu/ once your attack is complete.

How do ensure that the redirect doesn't happen until after the form data has been posted?

You can trigger the redirect from the frame's onload handler. Depending on how your code is written, this onload handler may get called twice — once when the page initially loads and once when the form is submitted. If this is the case, you'll have to make sure that you change document.location on the second time only.

Attack C.

What can we assume the grader will do when logging in?

The grader will type a username, then click the password field, type a password, then move the mouse over the login button and click it. The grader will then wait for the login to complete.

Why is the site using JavaScript to focus the username field?

It's a convenience for the visitor, so they don't have to select the username field manually when they first come to the page. You may find it useful as well.

How do I get the browser to call my injected code?

Because your code is sanitized with `htmlspecialchars`, you won't be able to inject a simple `<script>` tag like you did in Attack A. Trick the browser into running your code another way.

I think I figured it out, but when I call alert to test my attack, nothing happens.

It turns out that calling alert would lead to an infinite loop of dialog boxes, so Firefox is trying to be helpful by preventing it. Try using something like `document.login_form.loginusername.value=42` to test whether your attack is working.

How am I supposed to invoke the email script from my script without using any characters that will get escaped?

There are numerous static methods of String that you might find usable. Also, don't forget about escape, unescape, and eval.

Encoding by hand is incredibly tedious. Help?

You can save yourself some headaches if you write out your attack as a string in your attack page and then encode it programmatically, using those static String methods.

What do I need to do to make my attack invisible to the user?

You'll have to get a handle on the relevant DOM nodes and make the extraneous text disappear, either by deleting the text or setting the style.display property to "none".

How can I get a handle on the warning message? It doesn't have an id attribute.

It does have other distinguishing characteristics. You may find getElementsByTagName useful. Depending on how your attack works, you may not see a warning message anyway.

I'm sending the email when the form is submitted, but it isn't working. What's wrong?

There's a race condition here where the form may be getting submitted before the email image is downloaded. Once the form submit starts, the thread that's downloading the image is killed. So, to ensure that your attack always works, you should delay the form submission a little bit. You can use addEventListener with an event handler such `asfunction(evt) { evt.preventDefault(); ... }`. In this way, you can prevent the form submission until you're ready to trigger it yourself.

Ok, I prevented the submission. How long should I wait?

If you're lazy, you can use setTimeout with a reasonable number of milliseconds. But the precise way of doing it is to use addEventListener to wait for the "error" event to fire on the Image object you created. That event indicates that the email script's server has started to respond with a non-image file, meaning that it successfully processed the email send request.

What should I do after the email is sent?

You can trigger the submission manually using the login button's click() method. Use removeEventListener() to avoid infinite loops.

Is it okay if the attack doesn't work when the HTML file is clicked on the desktop, but it does work when the URL of the HTML file is put into the browser address bar directly?

That is fine. We will test your attack by putting in the URL of your HTML file in the browser's address bar.

What should the email look like if the attack worked?

The parts that might be different are in red.

```
From: Apache <apache@zoobar.csl.toronto.edu>
To: grader@zoobar.csl.toronto.edu, youremail@utoronto.ca
Subject: Message from group 'yourgroupid'

Payload:

grader,topsecret
```

Firefox keeps asking if I want to save the password. Is there any way I can turn this off?

You can probably turn it off by setting autocomplete="no" in each of your form fields, but it shouldn't present much of an issue for grading. The grader will have already hit "Never for this site" before testing your attack, so the dialog won't come up.

Attack D.

How does the site sanitize profiles?

It uses strip_tags() to restrict the tags that can be used, and it uses a regular expression to replace certain dangerous words like "onmouseover" with a space character. Note that strip_tags() will remove all tags that are more than 1024 characters. If your solution occurs inside a tag, you will have to make sure it fits inside this limit, or it won't render to the user.

What tags does zoobar.csl.toronto.edu allow in profiles?

`<a> <br> <b> <h1> <h2> <h3> <h4> <i> <img> <li> <ol> <p> <strong> <table> <tr> <td> <th> <u> <ul> <em> <span>`

What strings does zoobar.csl.toronto.edu not allow in profiles?

javascript: `eval setTimeout setInterval window target onAbort onBlur onChange onClick onDblClick onDragDrop onError onFocus onKeyDown onKeyPress onKeyUp onLoad onMouseDown onMouseMove onMouseOut onMouseOver onMouseUp onMove onReset onResize onSelect onSubmit onUnload`

Keep in mind that this is an example of what **not** to do. Blacklisting keywords is a recipe for disaster and will annoy, but not limit, a determined attacker.

How do I transfer the zoobar?

Create an `<iframe>` pointing to transfer.php, set the appropriate form fields inside it, and post the form. Alternatively, you can create a form on the current page with transfer.php as its target, and then post it with the target pointing at an `<iframe>`. Another option is to use XMLHttpRequest, since you're actually making a same-site request this time. Pick whichever approach you prefer.

How do I create an <iframe>?

You can use the DOM methods document.createElement and document.body.appendChild.

How do I get a handle on form fields inside the <iframe>?

It differs by browser, and only works when the frame's domain matches the parent page (that's the Same Origin Principle). Here's the Firefox way of doing it:

```
iframe.contentDocument.forms[0].zoobars.value = 1;
```

Are there any alternatives to target, which is blacklisted?

You can use string concatenation to express "target" without actually saying it. The following are equivalent in JavaScript: `x.target, x["target"], x["tar"+"get"]`.

How do I replace the profile?

Use the same technique you just used on transfer.php, but point at index.php instead.

Is there an easy way to get a copy of the current profile?

You can use `document.getElementById('profile').innerHTML`, but it may mangle quotes in your profile, so be sure to check that the replicated profile is still functional. Also, note that only display:inline tags can be nested inside a `<p>`.