CS3235 Week 10 Demo

CS3235 Teaching Team

March 2022

This demo presents examples of different types of cross-site scripting (XSS) attacks. It helps you understand what makes XSS attacks possible, what an attacker can do with XSS, and why the defences against cross-site request forgery (CSRF) are not sufficient for preventing XSS attacks.

To get the most out of this demo, you are encouraged to

- Use the "web developer tools" of your web browser to inspect each HTTP request and response as well as the cookies stored by the browser. On both Firefox and Chrome, you can open it by pressing F12.
- Click around on the web pages included in this demo.
- Read the source code (in very simple HTML, PHP and JavaScript).
- Make small changes to the source code and see what will happen.

Setup

1. Set up the "lab environment" (see the released instructions).
2. Download `week10-demo.zip` and unzip it.
3. In your shell, execute `seclab-setup-demo`. This sets up a few domain names in your `/etc/hosts` and might prompt you for your password. Note for WSL users: you might need to add `127.0.0.1 demo.org` to `C:\Windows\system32\drivers\etc\hosts`.
4. In your shell, execute `seclab-serve-demo <folder>/xss` where `<folder>` is the folder you just unzipped the demo files to. Then open your browser, go to `http://localhost/`. A web page should show up. Open the developer tools, go to the "network" tab and disable caching there. Linux users only: if you prefer to use the Firefox bundled in the Docker image, you can use `seclab-serve-browse <folder>/xss` instead. A Firefox window will show up. You don't need to use `seclab-setup-demo` or `seclab-clean-demo` either if you choose this option.
5. You can click around now. Enjoy!
6. You can use `seclab-clean-demo` to clean up the installed entries in `/etc/hosts` after you've had enough of this demo.

A Guided Tour

- `http://demo.org/login.php`: Let's say you are a victim. Please log in first. You will be given a cookie that identifies (and authenticates) you.
- `http://localhost/index.html`: This page is the attacker's web page. Let's say you are browsing this web page. Everything interests you and you want to click on those links. See what surprise you will get. Note that an attacker can as well send those links to you through other means (e.g., emails, WhatsApp messages). Also in some cases you don't even need to click (e.g., redirect).

- `http://demo.org/form-reflected.php`: This page has a form and some text. The webserver parses the parameters in the request and uses the `greeter` content as the text.
- `http://demo.org/post.php`: There is a bulletin board on this website. If you post something here, the message will be stored.
- `http://demo.org/form-stored.php`: In addition to the form, this page also shows the message from the bulletin board. The message might be posted by an attacker, in which case bad things can happen if the attacker has crafted the message carefully.
- `http://demo.org/form-dom.php`: It looks like this page is purely static. Every time your browser visits this page, it gets exactly the same HTML file. However, the browser-side scripts embedded in it update the page with a parameter passed through the URL.
- The `shellinject/shellinject1` file: This is a flawed shell script that is intended to echo your first command line argument to the screen (Windows users please use WSL or Cygwin for this). Try something like `./shellinject1 "hello; whoami"`. Compare this script with `shellinject/shellinject1-patched`.
- The `shellinject/shellinject2` file: This is a flawed shell script that is intended to print the content of a specified file to the screen (try `./shellinject2 document`. Try something like `./shellinject2 "document of=bad`. Compare this script with `shellinject/shellinject2-patched`.