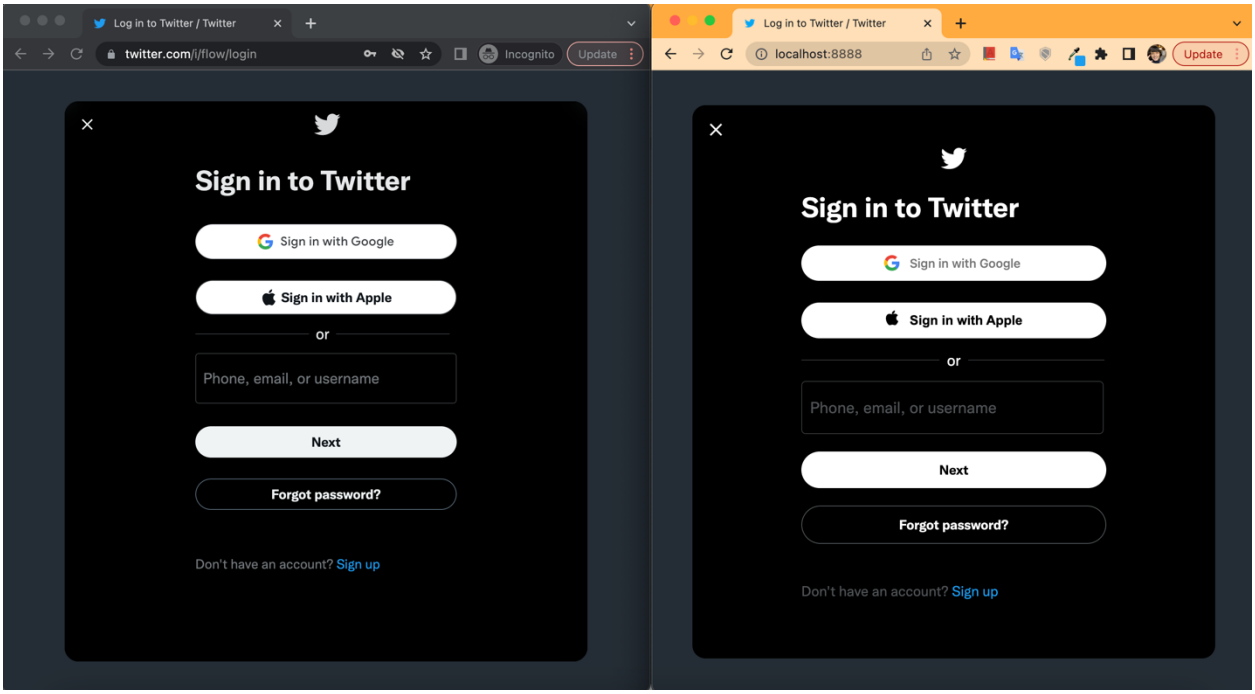
Swasti Mishra
Dr. Sun
COSC 366
15 November 2022

Twitter Phishing Project

For my project, I chose to build a phishing site based on one of my favorite social media platforms, Twitter. I am actually really proud of how effectively I was able to replicate the front-end. While it would have just been easier to just use inspect element and developer controls to vacuum up the HTML, CSS, and JS of the page, Twitter's login page is written abstrusely, and it was easier to just re-write it from scratch, which is what I did! I have included a comparison between my page and the actual Twitter login page below.



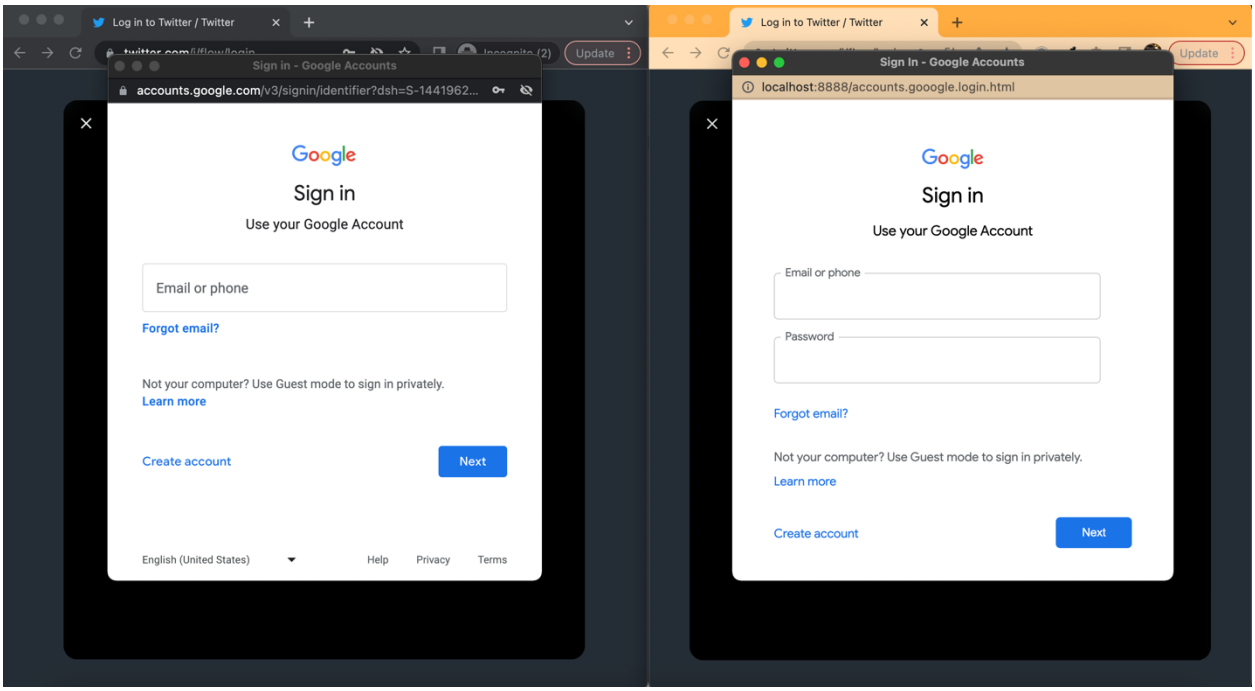
Twitter's authentic login page on the left, my re-creation on the right.

There are a few other bells and whistles that are not pictured in the above screenshot that I recreated, including the "Sign in with Apple" button leading to a legitimate popup, and the "Phone, email or username" field minimizing and floating the placeholder text to the top of the box in the exact same way that the real Twitter page does. All the other links, including the close button, the "Forgot password?" button, and the "Sign up" link lead to their corresponding links on the real Twitter page. All these features convince an unsuspecting victim that this is, in fact, the real Twitter login page.

Two things to note- the phishing attempt does not actually happen on this page. I thought that it may be too difficult to have a user input text in the "Phone, email or username" field and hit "Next", because the next page is still in this popup, but many of the features disappear and are

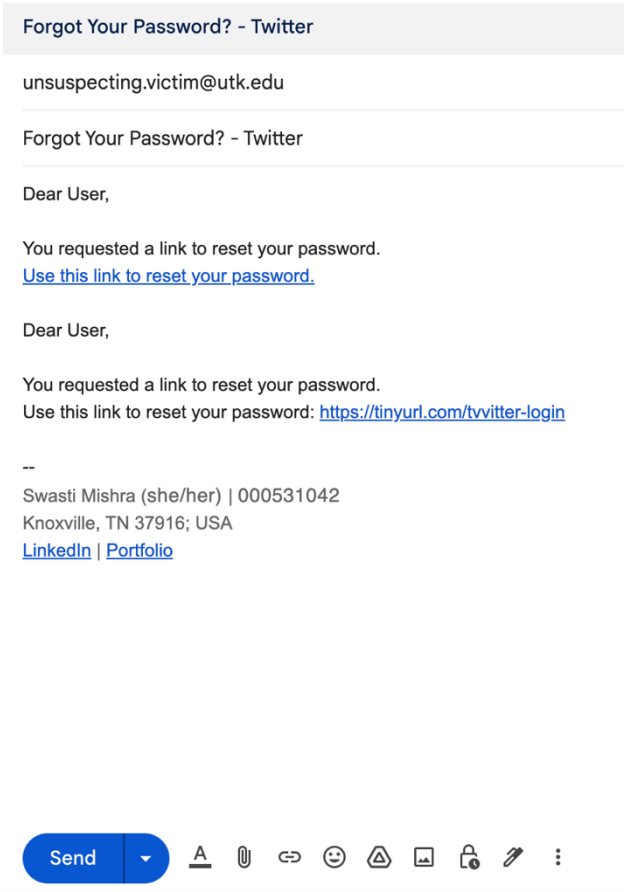
replaced by a username and password field. For this reason, the victim can put in their phone, email, or username, but clicking “Next” will actually lead to the legitimate twitter login page.

The real phishing attempt happens when a user clicks “Sign in with Google”. When the victim clicks this button, I wrote a popup page that looks exactly like the real Google sign in. While the victim puts their login information in the Google popup, the fake site “refreshes” and loads in the legitimate Twitter login page. I have included screenshots of my fake Google popup below.



Google’s authentic login page on the left, my re-creation on the right.

While my popup looks slightly different, the fonts all match Google’s typical branding, and only a very attentive user would be able to notice that something is off. On this page, the victim inputs their email or phone and password, and clicks “Next”, which puts their information into my database. The popup window closes, and the victim is left with the Twitter login page that refreshed earlier, on the legitimate page.

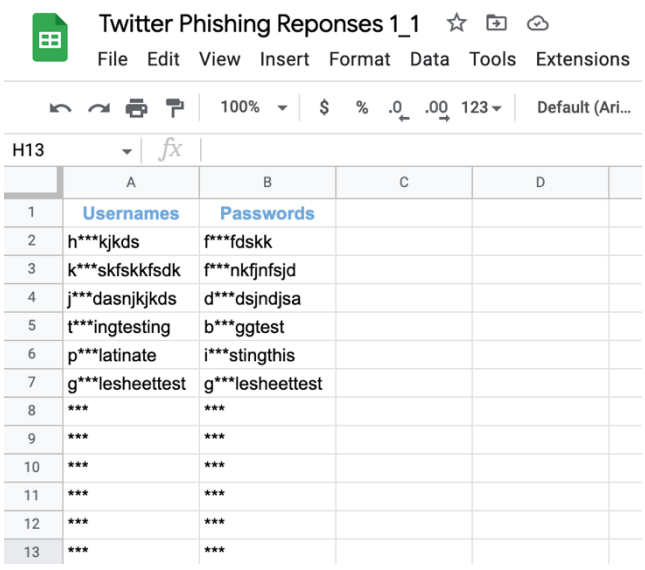


1. Have two defined methods of tricking victims into using the phishing site, e.g., shortened URL sent via email, or posted on social media, etc.

For this requirement, I drafted two possible methods of tricking victims into using the phishing site. The easiest would be through an email, for example, like this one. In one method, an attacker could conceal the link in text because many users do not mouse over a link and confirm their authenticity before clicking on them. In another method, a user could purchase a realistic looking domain, e.g., tvvitter.com, and then configure the files in such a way that the link looks exactly like Twitter's actual domain. Where Twitter's real login page is twitter.com/i/flow/login, a hacker could use tvvitter.com/i/flow/login. For the purposes of this email, I simply shortened the link to a tinyurl, but the concept is essentially limitless. A hacker could also conceal the link in PowerPoints or things like that.

2. Store credentials in a database to prove validity of attack, but passwords should be slightly obfuscated before storage, so that no real credentials are stored, e.g., only store the first and/or last two characters of the password.

To achieve this goal, I set up a Google Sheet and connected it to a service called "API Spreadsheet" to essentially treat inputs as queries. I specifically did this because I wanted Dr. Sun and the TAs to be able to see the database update in real time. The spreadsheet/database is publicly viewable here. The first sheet is protected and contains the "real" credentials. The second sheet is unprotected and contains the censored credentials.

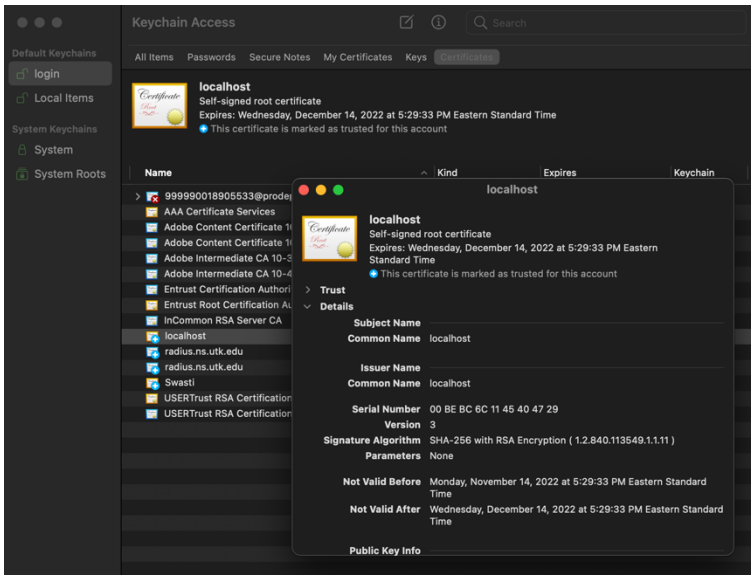


3. Redirect from phishing site to the real site, without leaving evidence to the victim that they were at a phishing site. There are multiple ways to accomplish this using JavaScript among other languages.

I achieve this in a few ways (outlined in depth in my description of the front-end design), but I think the most effective way is timing a "page refresh" after the victim clicks the "Sign in with Google" button. Pages refresh all the time for a variety of reasons, and while the victim is focused on the new popup, they do not even notice the page reloading to the authentic Twitter page. I used JavaScript to achieve this in the following code.

```
<script>
    function buttonClick() {
        var timer = setTimeout(function() {
        window.location='https://twitter.com/i/flow/login'
    }, 3000);
    }
</script>
```

JavaScript used to redirect back to the authentic page on a button click.



4. Use a TLS certificate so that the site appears even more legitimate. You can set up a self-signed certificate to achieve this.

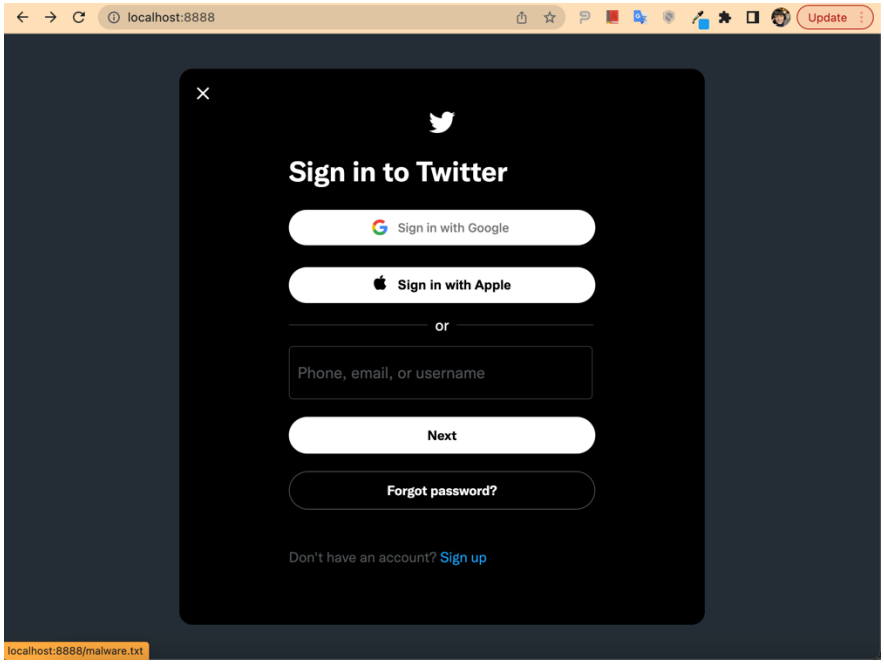
I struggled with this requirement a little bit. I have generated a self-signed SSL certificate which you can see in the repository files, localhost.crt and localhost.key. Further, these files have been added to my local security keychain and are trusted, as pictured in the screenshot below. For some reason, though, my browser refuses to display the little lock that would make the site look more legitimate.

5. Implement two SQL injection vulnerabilities for your website using the tricks covered in class, e.g.,' or 1=1--,'; Drop Table Users--, and show how to exploit them.

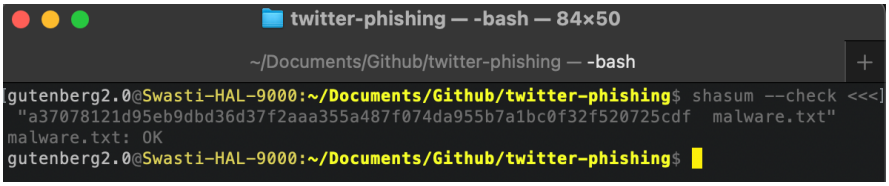
Because my data is sent to a public spreadsheet, there is no SQL database with vulnerabilities. (More about this in requirement 2 above.) But if one were to connect this site to a mySQL database, it should be vulnerable- I didn't program in any text sanitization or anything like that.

6. Have a download link to "malware" on the site. You can use any benign file or program on your site for download. Include a SHA-256 hash of the "malware" visible on the site so that victims trust it.

There was not really a good place to put a download link, seeing as Twitter's login page does not have one, but I concealed one in the on the "Forgot password?" button, on the text of the button. I did also include an SHA-256 hash malware, which you can verify in the image below. I have also included an image of where the "Forgot password?" malware is concealed.



The malware is concealed in the "Forgot password?" text.



The hash of malware.txt.

Conclusion

I learned a lot during this project! Primarily some fun front-end things, which will be useful in a few other projects I'll be working on. Also, this project gave me a little bit more experience with JavaScript, which is something I've been wary of in the past. Connecting to the database through API Spreadsheets is also something I'll take advantage of in the future. If I had more time, I would love to actually make the "Next" button lead to a full sign in phishing page instead of just reloading. I would also like to spend more time experimenting with SQL injection vulnerabilities and sanitizing input.

Here's a link to my walkthrough video.
Here's a link to my GitHub repository.