HW2 Final Assignment

Andrea Stojanovski

NOTE: Signed GitHub Commits

Only the my last commit is signed, as I did this last. Note that the signed commit at tag **part_2_complete** contains everything.

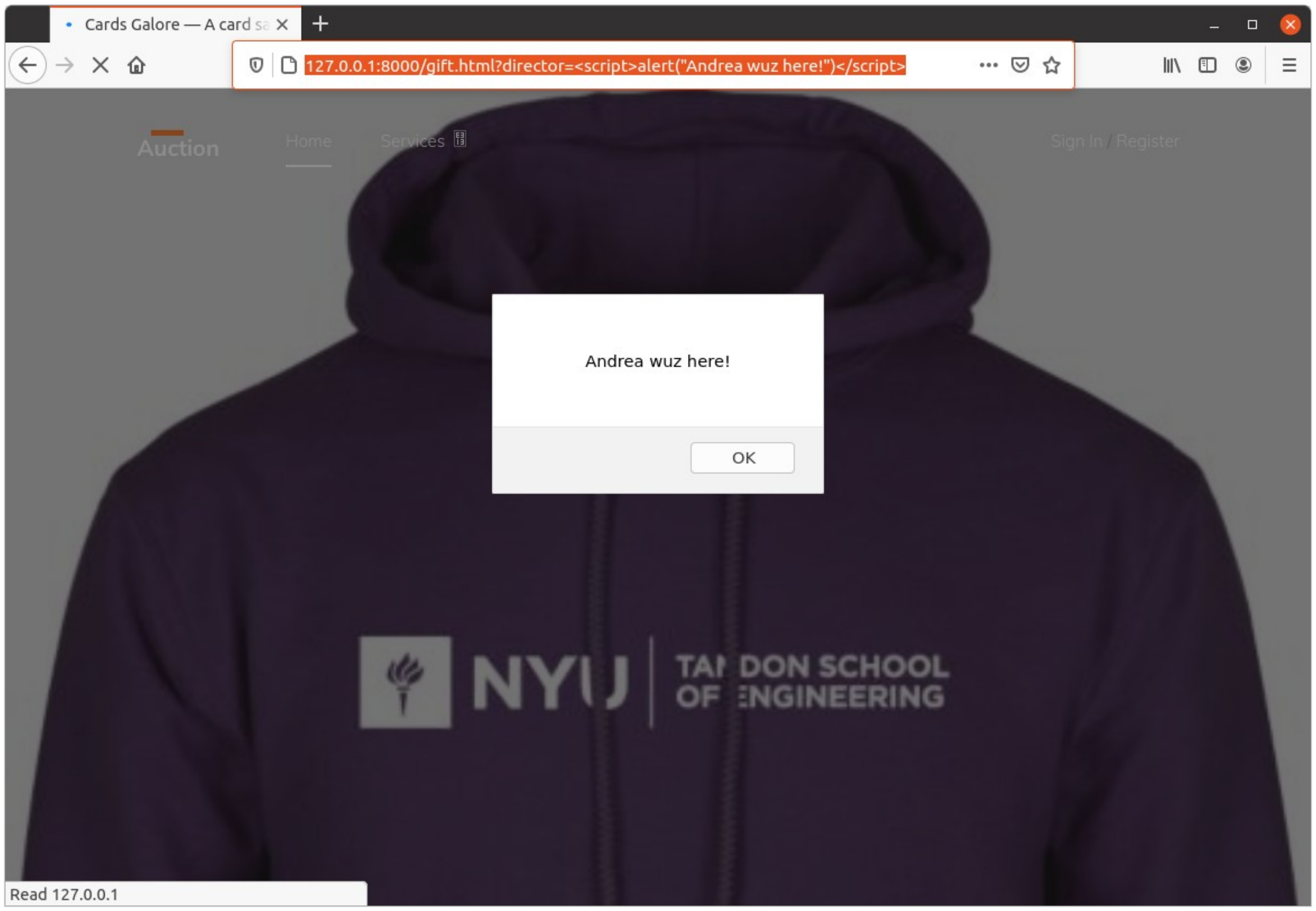
Part 1 – Auditing and Test Cases

Cross-Site Scripting Attack

If we can lure the naive trusting user to the site:

http://127.0.0.1:8000/gift.html?director=%3Cscript%3Ealert(%22Andrea%20wuz%20here!%22)%3C/script%3E

we get:



We see that we can execute a script, which in turn could collect the session, cookie etc.

Simple fix would be to force escaping of the “director” by changing:

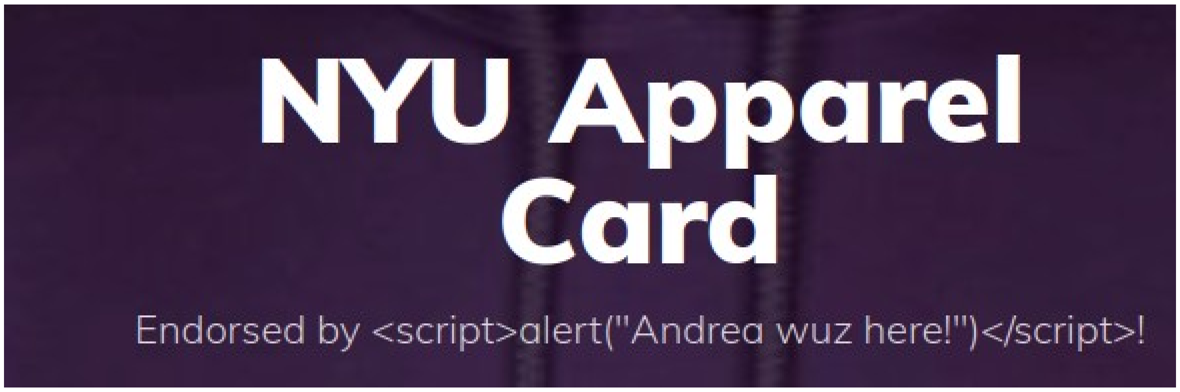
```
<p>Endorsed by {{director|safe}}!</p>
```

to

```
<p>Endorsed by {{director}}!</p>
```

in the page item-single.html and gift.html (there may be other places).

Now, instead of my script, an escaped text would show on the page, which is benign and thus the problem is fixed:

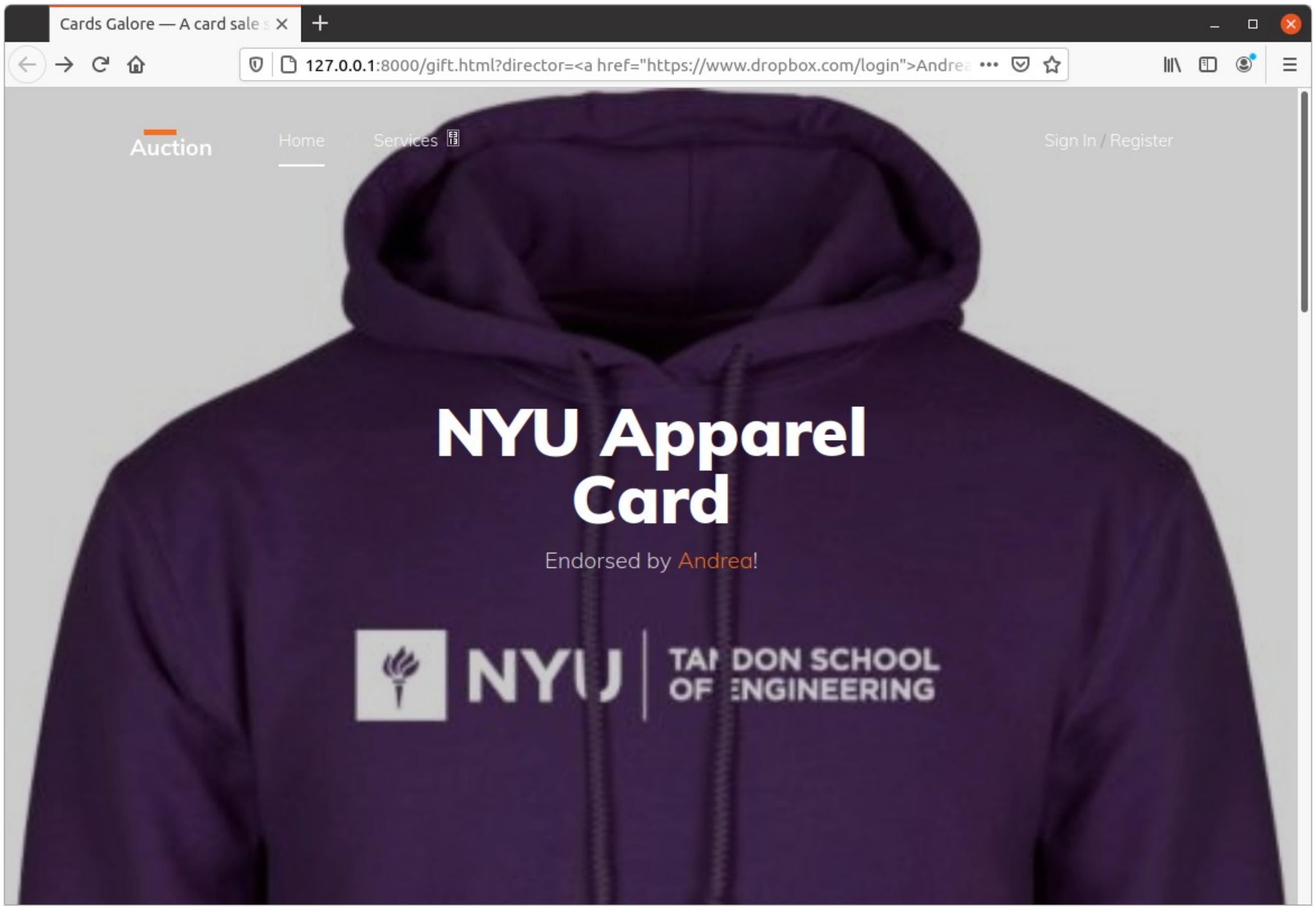

NYU Apparel Card

Endorsed by <script>alert("Andrea wuz here!")</script>!

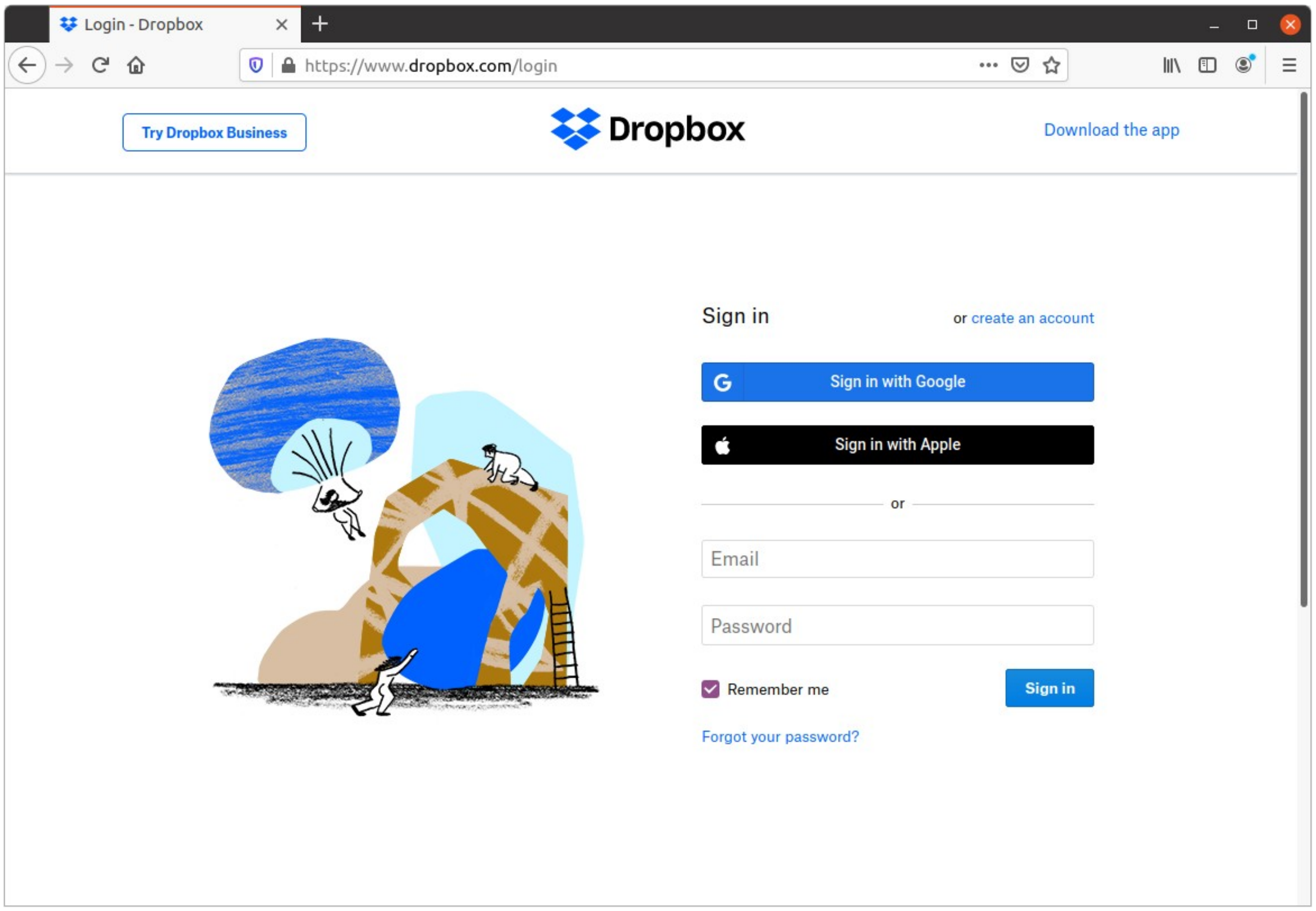
Cross-Site Request Forgery Attack

If we can lure the naive trusting user to the site:

http://127.0.0.1:8000/gift.html?director=%3Ca%20href=%22https://www.dropbox.com/login%22%3EAndrea%3C/a%3E


Cards Galore — A card sale s
127.0.0.1:8000/gift.html?director=<a href="https://www.dropbox.com/login">Andrea
Auction
Home
Services
Sign In / Register
NYU Apparel Card
Endorsed by Andrea!
NYU TANDON SCHOOL OF ENGINEERING

Note the link “Endorsed by **Andrea”** which would take the user to a phishing site, which collects the user’s credentials. I used the Dropbox login site as a sample:



To fix this issue the same fix as in the Cross-site Scripting Attack works. To repeat, I changed

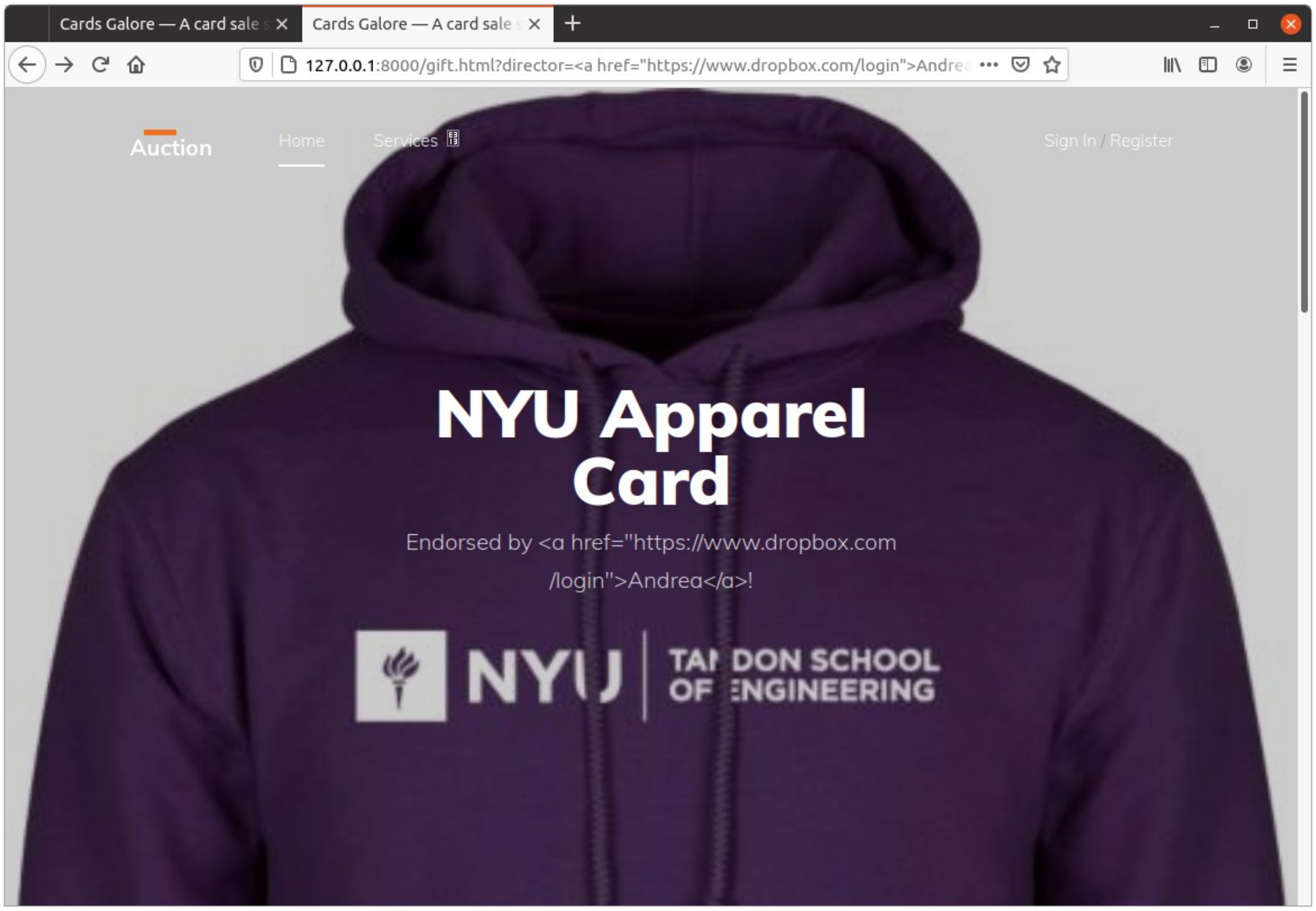
`<p>Endorsed by {{director|safe}}!</p>`

to

`<p>Endorsed by {{director}}!</p>`

in the page item-single.html and gift.html (there may be other places).

Now this fixes the vulnerability. Trying to do the same, we get:


Cards Galore — A card sale
Cards Galore — A card sale
127.0.0.1:8000/gift.html?director=<a href="https://www.dropbox.com/login">Andre
Auction
Home
Services
Sign In / Register
NYU Apparel Card
Endorsed by <a href="https://www.dropbox.com/login">Andrea</a>!
NYU TANDON SCHOOL OF ENGINEERING

SQL Injection to Obtain User's Password

There is an non-encoded SQL query in viewes.py:

```
card_query = Card.objects.raw('select id from LegacySite_card where data = \'%s\'' %
signature)
user_cards = Card.objects.raw('select id, count(*) as count from LegacySite_card where
LegacySite_card.user_id = %s' % str(request.user.id))
```

The user can enter a special string for signature, and then when

<p> Found card with data: {{ card_found }} </p>

will display the result of the injected sub-query.

For example, to get the password we could enter an union attack string instead of signature:

<signature>" UNION select password

as described in https://portswigger.net/web-security/sql-injection/union-attacks

To fix this, in the first query which uses raw “signature”, we should change it to:

signature(encode)

This will escape the signature properly, not allowing injection of closing quotes and opening of another sub-query.

HTTP Man-in-the-middle and Listen-in Attacks

The entire site is exposed in clear text. This includes

```
SESSION_COOKIE_NAME = 'sessionid'
```

(see in global_settings.py)

There are two possible attacks:

1. The traffic can be sniffed at any router on the route between the client and the server. Stealing the Session ID is enough.

2. Man-in-the-middle attack can be orchestrated by putting a proxy on the path from the client browser to the server. To divert the traffic, DNS record can be spoofed to re-direct the request to the attacker, or even a malicious router can be configured to act as a malicious proxy.

One possible (naive) fix is to set up Django to use HTTPS.

However, this should be solved by terminating HTTPS (TLS/SSL) in the load balancer or reverse proxy that should sit in front of the Django web site. This is a better solution, as a combination of reverse proxy for the static content and load balancer for instances that serve dynamic content would make the site perform better with high load. Such fix would be out of the scope of this assignment.

Part 2: Encrypting the Database

Encryption

Considering the limited time, I foresee problems in using a tool that will encrypt all fields in the database. The encryption schemes do not offer ordering (if they did, the encryption would be breakable using binary search. For all encrypted fields, we would lose the following: ordering, sorting, searches with partial text, indexes that span more than one field. In our site, searching would break, and we can only search for exact matches. Any search indexing (like elastic search) would weaken the encryption, as data may be inferred from the search index itself.

To simplify the solution I’ll use the symmetric encryption package django-fernet-fields.

The most vulnerable data is the card itself, so I’ll just encrypt that. I believe this is a good quick compromise, as it is not going to break anything, yet it protects the most important part – the money (the card).

Key management

Storing the symmetric encryption key defeats the purpose of the encryption.

I'll add the package python-decouple and store the key in the .env file.