Free2z Mini Audit

February 9, 2023
Updated: September 12, 2023
Taylor Hornby
zecsec@defuse.ca
ZecSec

Introduction

This is the result of a 3-day timeboxed security audit of Free2z. I was able to skim over most of the code looking for classic kinds of web bugs (e.g. XSS) and other problems specific to free2z, like the security of the 2z tokens. I did not spend very much time looking at the deployment infrastructure (docker and k8s stuff). I reviewed commit 4fdc17f1aefeb6cc2a0a7b065bd3ea951213d2ab of the tuzi repo.

Issues & Recommendations

XSS in Twitter Callback URL

Severity: High

The twitter callback URL will reflect its `error` parameter into the HTML of the error page without escaping it. This would have allowed for a cross-site scripting attack against free2z users if scripts weren't blocked by free2z's CSP headers:

https://free2z.com/api/twitter/callback?error=foobar%3Ch1%3Ei%20think%20some%20security%20headers%20are%20preventing%20scripts%20from%20running?%3C/h1%3E

Update 2023-09-12: This has been fixed by applying Django's strip_tags() function to the error message before printing the string. This *should* defend against XSS attacks, but according to Django's documentation, this is not guaranteed: “Absolutely NO guarantee is provided about the resulting string being HTML safe. So NEVER mark safe the result of a `strip_tag` call without escaping it first, for example with escape().

Relaying Image URLs Allows User Tracking

Severity: Medium

Rather than uploading profile pictures or profile background images, users are allowed to specify any URL for those images, which will be reflected into free2z pages.

This is a privacy risk for free2z users because any user can host their profile picture on a domain they control and see all of the requests for the image, including free2z users' IP addresses.

Since users' profile pictures are displayed alongside their comments, this allows an attacker to track which free2z pages users are viewing over time.

There is a similar issue for image URLs provided in Markdown, but there, users are warned about fetching third-party content and have the option not to fetch those images.

Update 2023-09-12: This kind of tracking has not been prevented, but is now documented in Free2Z's privacy information.

TLS Configuration Improvements

Severity: Low

Free2z's TLS configuration could be improved. I used SSL labs to scan its configuration:

https://www.ssllabs.com/ssltest/analyze.html?d=free2z.com

The main issue is that TLS 1.0 and 1.1 are still supported. These versions of TLS are vulnerable to some attacks and should be forbidden.

TLS 1.2 is supported by default on IE >= 11, Chrome >= 29, Firefox >= 23, so it is pretty safe to require >= TLS 1.2 these days. If compatibility with old browsers is a concern, it is definitely safe and worthwhile to disable TLS 1.0.

Once that is fixed, re-run the SSL labs test using the URL above, and it will probably tell you that your cipher suite ordering can be improved to make sure connections have forward secrecy.

Update 2023-09-12: This has been resolved.

Use a Consistent Domain Name

Severity: Low

The same Free2z website is accessible from different domains: free2z.com, free2z.cash, etc. I recommend redirecting all users to one domain name. Otherwise, if users are accustomed to accessing Free2z through many different domain names, it's easier for an attacker to set up a phishing site (e.g. free2z.io) and get users to believe it.

Update 2023-09-12: This is currently unresolved as Free2Z determines which domain to use.

Race Conditions When Checking Balance

Severity: Medium

A common pattern in the Free2z code is to check if the user has enough balance for some action, and if they do, deduct funds from their account. For example, from apps/comments/serializers.py:

```
def create(self, validated_data):
    user = self.context['request'].user
    # Set the author field to the username of the current user
    validated_data['author'] = user

    tuzis = validated_data.get('tuzis', 1)
    if user.tuzis < tuzis: (***)
```

```
          # Return a "bad request" response if the user doesn't have
enough tuzis
          return Response(status=status.HTTP_400_BAD_REQUEST)

     from django.db import transaction
     with transaction.atomic():
          user.tuzis -= tuzis
          user.save()
          return super().create(validated_data)
```

A race condition might allow users to spend more tuzis than they own. For example, suppose two requests come in simultaneously so there are two threads running this code. Both threads execute the balance check at (***) simultaneously. Both threads think the user has enough balance, and both threads deduct from the user's balance, even if the user only had enough balance for *one* of the deductions to happen.

To fix this, the entire operation including the balance check needs to be atomic, i.e. inside the transaction.atomic().

There are several other instances of this kind of race condition:

```
py/dj/apps/comments/serializers.py:83
py/dj/apps/comments/views.py:27
py/dj/apps/comments/views.py:139
py/dj/apps/storytime/views.py:76
py/dj/apps/storytime/views.py:115
py/dj/apps/g12f/views/zpage.py:75
py/dj/apps/g12f/views/zpage.py:124
py/dj/apps/g12f/views/creator.py:147
py/dj/apps/g12f/views/creator.py:213
py/dj/apps/openai/views.py:30
py/dj/apps/openai/views.py:53
```

I did not confirm that this is actually exploitable; the details of how Python/Django schedules requests and threads may prevent its exploitation in practice. However even in that case, it is worth fixing the code in case the website is scaled with multiple backends connecting to the same database.

Update 2023-09-12: This has been resolved by adding a middleware to ensure requests are executed atomically.

Possible Negative Amount Bug

Severity: Medium

The following code in py/dj/apps/g12f/views/zpage.py doesn’t check if the tuzi amount coming from the request is negative. As a result, it seems possible for creators to give themselves free tuzis by funding their pages with negative amounts:

```
class FundPageView(APIView):

      def post(self, request: Request, format=None) -> Response:
      creator = request.user
      # print(dir(request.data.keys()))
      # print(request.user.tuzis)
      addr = request.data.get('id')
      amt = request.data.get('amount')
      # print(addr, amt)
      if not (addr and amt):
            return Response(
                  status=status.HTTP_400_BAD_REQUEST,
                  data="id and amount required"
            )
      amt = min(request.user.tuzis, amt)
…
      page.tuzi_total += amt
      page.save()
      page.cache_total()
      creator.tuzis -= amt
      creator.save()
      creator.cache_total()
      return Response(page.f2z_score)
```

I didn't test this attack on the live server, but if `amt` were -1 for example, min(request.user.tuzis, amt) = -1, and it should result in adding 1 to the creator's balance and subtracting one from the page's balance.

Update 2023-09-12: This has been fixed.

Search Queries are Not URL-escaped

Severity: Low

In src/components/Find.tsx, the URL for the search page is constructed as follows:

```
let params = `search=${q.search}&page=${q.page}`
```

If a search query contains an & character, it will be interpreted as a separate URL parameter rather than part of the search query. For example, searching for `foo&bar=1` generates the following URL which searches for "foo", with an extra `bar=1' parameter which is ignored:

https://free2z.com/find?search=foo&bar=1&page=1

This is not a security issue, but `q.search` (and `q.page`) should be properly URL-escaped to avoid this problem. Escaping should also be added to any other instances where URLs are being constructed in a similar way. Use `urllib.parse.quote()` or similar.

Update 2023-09-12: This has been fixed.

Good Things

Using CSP, HSTS, security.txt

It's awesome to see Free2z using a good CSP header setup (which prevented the XSS attack!) and also HSTS. You can also consider adding your domains to browsers' TLS preload lists so they will absolutely refuse to use a plaintext connection even before seeing the HSTS header:

https://hstspreload.org/

It’s very nice to see PRs adding security.txt as well! All of this is suggestive of a top-notch focus on security.

Django, TypeScript

Using Django makes it pretty much impossible to have SQL injection problems, and the use of TypeScript makes the JS code a lot less error-prone. Nice!