How to Hack a Django Website

Adam Johnson

🍀 Four Stories

🍀 Four Stories

1. **Facebook** - Pickle Remote Code Execution

🍀 Four Stories

1. **Facebook** - Pickle Remote Code Execution
2. **GitHub** - Unicode Case Collision Account Takeover

🍀 Four Stories

1. **Facebook** - Pickle Remote Code Execution
2. **GitHub** - Unicode Case Collision Account Takeover
3. **YPlan** - HTML Injection

🍀 Four Stories

1. **Facebook** - Pickle Remote Code Execution
2. **GitHub** - Unicode Case Collision Account Takeover
3. **YPlan** - HTML Injection
4. **Lotsa Sites** - Javascript HTML Injection

Story 1: Facebook

2018 Blog Post by Blaklis at SCRT, a Swiss infosec company

SCRT Information Security

Sec Team Blog

RECENT POSTS

- Continuous Pentesting
- Engineering antivirus evasion (Part II)
- Engineering antivirus evasion
- SCRT on Covid-19 and Remote Access / Working From Home
- Combining Request Smuggling and CBC Byte-flipping to stored-XSS

ARCHIVES

Select Month

CATEGORIES

- Antivirus bypass
- CTF
- Events
- exploit
- Fail
- forensics

Remote Code Execution on a Facebook server

I regularly search for vulnerabilities on big services that allow it and have a Bug Bounty program. Here is my first paper which covers a vulnerability I discovered on one of Facebook's servers.

While scanning an IP range that belongs to Facebook (199.201.65.0/24), I found a Sentry service hosted on 199.201.65.36, with the hostname sentryagreements.thefacebook.com. Sentry is a log collection web application, written in Python with the Django framework.

While I was looking at the application, some stacktraces regularly popped on the page, for an unknown reason. The application seemed to be unstable regarding the user password reset feature, which occasionally crashed. Django debug mode was not turned off, which consequently prints the whole environment when a stacktrace occurs. However, Django snips critical information (passwords, secrets, key...) in those stacktraces, therefore avoiding a massive information leakage.

TypeError at /account/recover/

object of type 'long' has no len()

Request Method:	POST
Request URL:	https://sentryagreements.thefacebook.com/account/recover/
Django Version:	1.6.11
Exception Type:	TypeError
Exception Value:	object of type 'long' has no len()

🔍 Step 1: Scan Facebook IP range

Step 2: Find Sentry

TA The Tardis
River Song

- Projects
- Issues
- Events
- Releases
- User Feedback
- Dashboards
- Discover
- Activity
- Stats
- Settings

All Projects | All Environments | Last 14 days

Unresolved Issues (450)

Sort by: Last Seen | is:unresolved

Resolve | Ignore | Merge | GRAPH: 24h 14d | EVENTS | USERS | ASSIGNEE

	EVENTS	USERS	ASSIGNEE
TypeError func(components/App) in components/App.js this.myCodeIsPerfect is not a function FRONTEND-REACT-V · an hour ago – 2 months old · SI-20	500	45	NM
Error cart.forEach(app) in /Users/vu/Documents/sdk_demos/demos/express/app.js No inventory for nails EXPRESS-DEMO-4 · 3 hours ago – 2 months old	490	80	
Error apply(components/App) in components/App.js 500 - Internal Server Error FRONTEND-REACT-R · 3 hours ago – 3 months old	140	49	NM
TypeError null.<anonymous>(app) in /app/app.js obj.doesNotExist is not a function EXPRESS-DEMO-6 · 21 hours ago – a month old	86	38	

Step 3: Browse Sentry

Wow, `DEBUG=True`

TypeError at /account/recover/

object of type 'long' has no len()

Request Method:	POST
Request URL:	https://sentryagreements.thefacebook.com/account/recover/
Django Version:	1.6.11
Exception Type:	TypeError
Exception Value:	`object of type 'long' has no len()`
Exception Location:	/opt/app/sentry/venv/local/lib/python2.7/site-packages/simplejson/decoder.py in raw_decode, line 394
Python Executable:	/opt/app/sentry/venv/bin/uwsgi

🤓 Step 4: Read Debugged Settings

```
SESSION_SERIALIZER = "django.<snip>.PickleSerializer"
SESSION_ENGINE = "django.<snip>.signed_cookies"
SENTRY_OPTIONS = {
    ...,
    "system.secret-key": "some-not-so-secret-key",
}
```

🍪 Step 5: Replace contents of signed cookie

```
data = django.core.signing.loads(
    current_cookie,
    key="some-not-so-secret-key",
    serializer=PickleSerializer,
    ...
)
data['testcookie'] = PickleRce()
print(django.core.signing.dumps(
    data,
    key="some-not-so-secret-key",
    serializer=PickleSerializer,
    ...
))
```

🚨 PICKLE IS CODE 🚨

```
import os


class PickleRce:
    def __reduce__(self):
        return os.system, ("sleep 30",)
```

```
In [2]: pickle.dumps(PickleRce())
Out[2]:
b'\x80\x04\x95#\x00\x00\x00\x00\x00\x00\x00\x8c\x05posix\x94\x
 30\x94\x85\x94R\x94.'
```

Runs `sleep 30` *on load*!

🚨 PICKLE IS CODE 🚨

Table of Contents

`pickle` — Python object serialization

Source code: Lib/pickle.py

The `pickle` module implements binary protocols for serializing and de-serializing a Python object structure. *"Pickling"* is the process whereby a Python object hierarchy is converted into a byte stream, and *"unpickling"* is the inverse operation, whereby a byte stream (from a binary file or bytes-like object) is converted back into an object hierarchy. Pickling (and unpickling) is alternatively known as "serialization", "marshalling," [1] or "flattening"; however, to avoid confusion, the terms used here are "pickling" and "unpickling".

Warning: The `pickle` module **is not secure**. Only unpickle data you trust.

It is possible to construct malicious pickle data which will **execute arbitrary code during unpickling**. Never unpickle data that could have come from an untrusted source, or that could have been tampered with.

Consider signing data with `hmac` if you need to ensure that it has not been tampered with.

Safer serialization formats such as `json` may be more appropriate if you are processing untrusted data. See Comparison with json.

💰 Step 6: Bounty

- 18 hours from report to patch.
- 20 hours to bounty.
- $5000 reward - lower tier bounty due to no user data at risk.

How to Hack

- Deliberately 404 to check for `DEBUG = True`.
- Look for things using pickle.
- Build evil pickle payloads.

🦸‍♀️ How to Protect

- *Never* deploy with `DEBUG = True`.
- Run `manage.py check --deploy`.
- Avoid pickle - no “safe mode”.
- Deprecate `PickleSerializer` (#29708).

🐙 Story 2 - GitHub

2019 Blog Post by John Gracey

Wisdom Session Replay Engineering JavaScript User Experience •••

Hacking GitHub with Unicode

Hacking GitHub with Unicode's dotless 'i'.

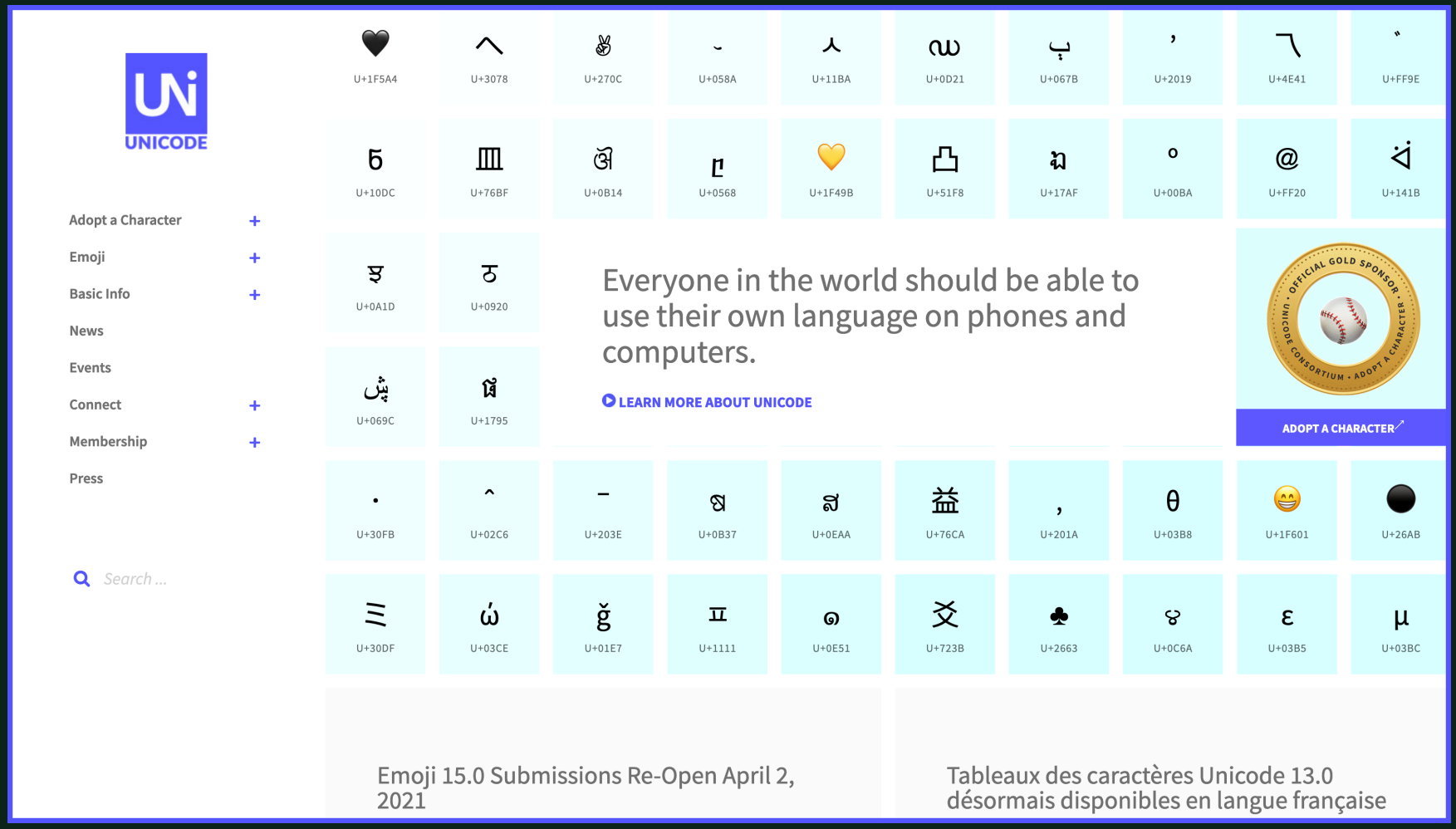
security · Nov 28, 2019

From combining emoji marks and astral planes, Unicode is under appreciated and poorly understood. The importance of understanding Unicode extends beyond localization and diversity. Failing to understand Unicode may lead to vulnerabilities in your code.

One lesser known occurrence is Unicode *Case Mapping Collisions*. Loosely speaking, a collision occurs when two *different* characters are uppercased or lowercased into the same character. This effect is found commonly at the boundary between two different protocols, like email and domain names.

💥 Step 1 - Find Case Collision

- github -> GITHUB
- github -> GITHUB



All Case Collisions with English:

Lowercase	Uppercase
ß	SS
ı	I
ſ	S
ﬀ	FF
ﬁ	FI
ﬂ	FL
ﬃ	FFI
ﬄ	FFL
ﬅ	ST
ﬆ	ST

Step 2 - Register Domain

- gıthub.com
- punycode: xn--gthub-n4a.com

Step 3 - Reset Password

- Reset password for john@github.com
- Receive reset link for john@github.com

[GitHub] Please reset your password Inbox ×

GitHub <noreply@github.com> 5:19 PM (0 minutes ago)

to me

We heard that you lost your GitHub password. Sorry about that!

But don't worry! You can use the following link to reset your password:

https://github.com/password_re...

If you don't use this link within 3 hours, it will expire. To get a new password reset link, visit https://github.com/password_reset

Thanks,
The GitHub Team

🔥 Step 4 - Unleash Chaos

Export account data

Export all repositories and profile metadata for **@adamchainz**. Exports will be available for 7 days.

Start export

Delete account

Your account is currently an owner in these organizations: **...**

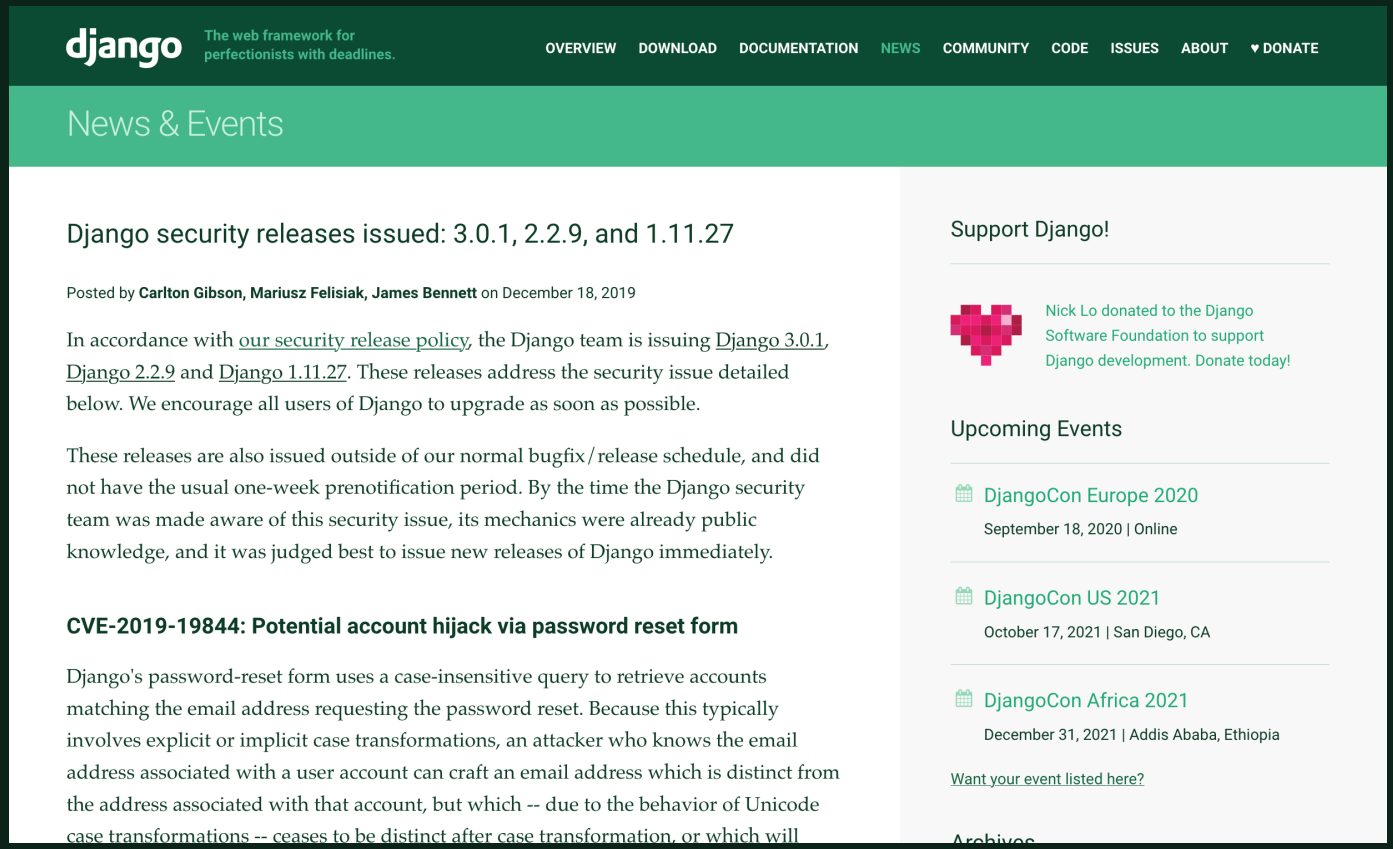
You must remove yourself, transfer ownership, or delete these organizations before you can delete your user.

Delete your account

Are you sure you don't want to just downgrade your account to a **FREE** account? We won't charge your credit card anymore.

😱 django.contrib.auth too!

Fixed in 1.11.27 / 2.2.9 / 3.0.1. Spotted by Simon Charette after reading about Github exploit.



django The web framework for perfectionists with deadlines.

OVERVIEW DOWNLOAD DOCUMENTATION NEWS COMMUNITY CODE ISSUES ABOUT ♥ DONATE

News & Events

Django security releases issued: 3.0.1, 2.2.9, and 1.11.27

Posted by **Carlton Gibson, Mariusz Felisiak, James Bennett** on December 18, 2019

In accordance with our security release policy, the Django team is issuing Django 3.0.1, Django 2.2.9 and Django 1.11.27. These releases address the security issue detailed below. We encourage all users of Django to upgrade as soon as possible.

These releases are also issued outside of our normal bugfix/release schedule, and did not have the usual one-week prenotification period. By the time the Django security team was made aware of this security issue, its mechanics were already public knowledge, and it was judged best to issue new releases of Django immediately.

CVE-2019-19844: Potential account hijack via password reset form

Django's password-reset form uses a case-insensitive query to retrieve accounts matching the email address requesting the password reset. Because this typically involves explicit or implicit case transformations, an attacker who knows the email address associated with a user account can craft an email address which is distinct from the address associated with that account, but which -- due to the behavior of Unicode case transformations -- ceases to be distinct after case transformation, or which will

Support Django!

Nick Lo donated to the Django Software Foundation to support Django development. Donate today!

Upcoming Events

DjangoCon Europe 2020
September 18, 2020 | Online

DjangoCon US 2021
October 17, 2021 | San Diego, CA

DjangoCon Africa 2021
December 31, 2021 | Addis Ababa, Ethiopia

Want your event listed here?

Archives

 How to Hack

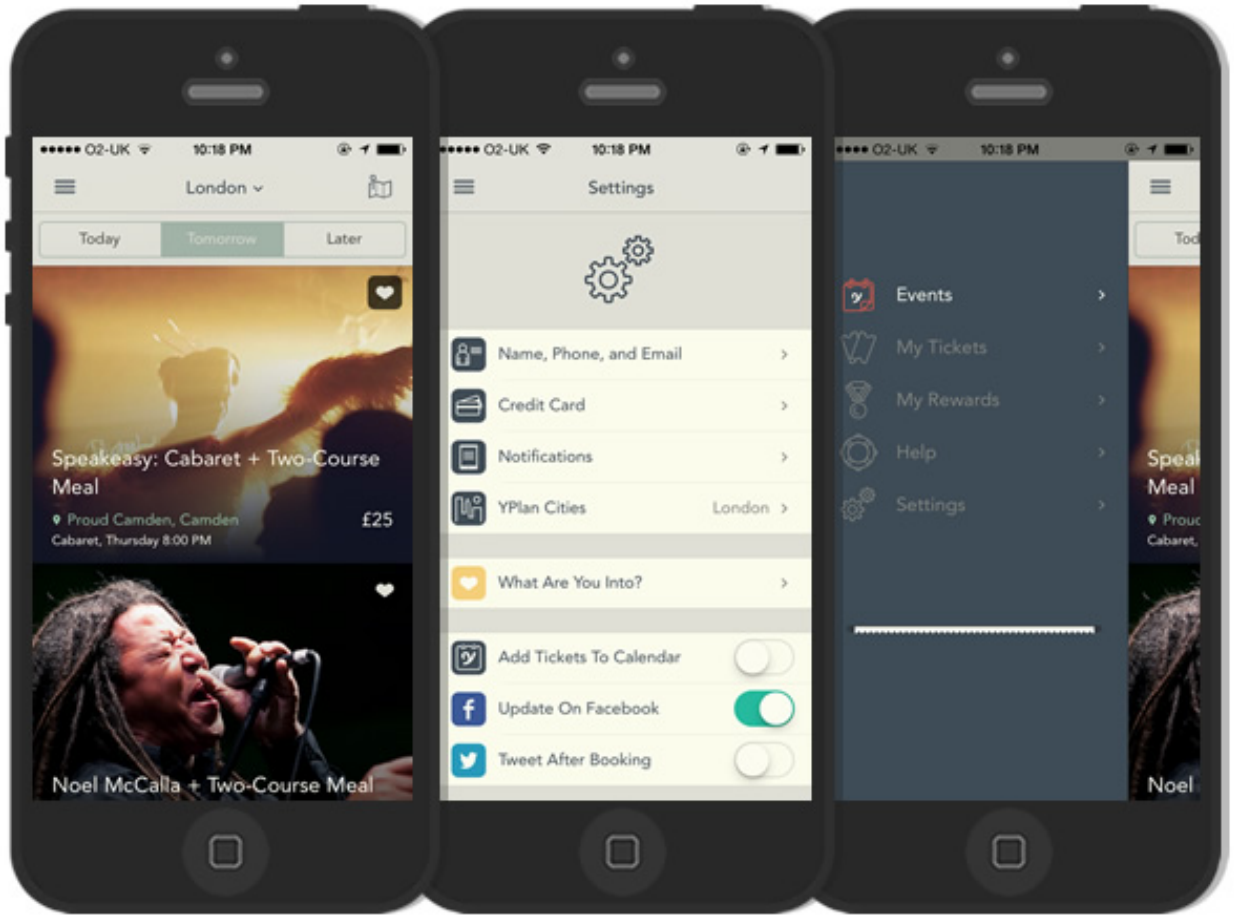
- Look for targets with unicode case collisions.
- Know other unicode features.
- Detect target site's Django version.

🦸‍♀️ How to Protect

- Upgrade Django.

🎉 Story 3: YPlan

Happened to me.



Free

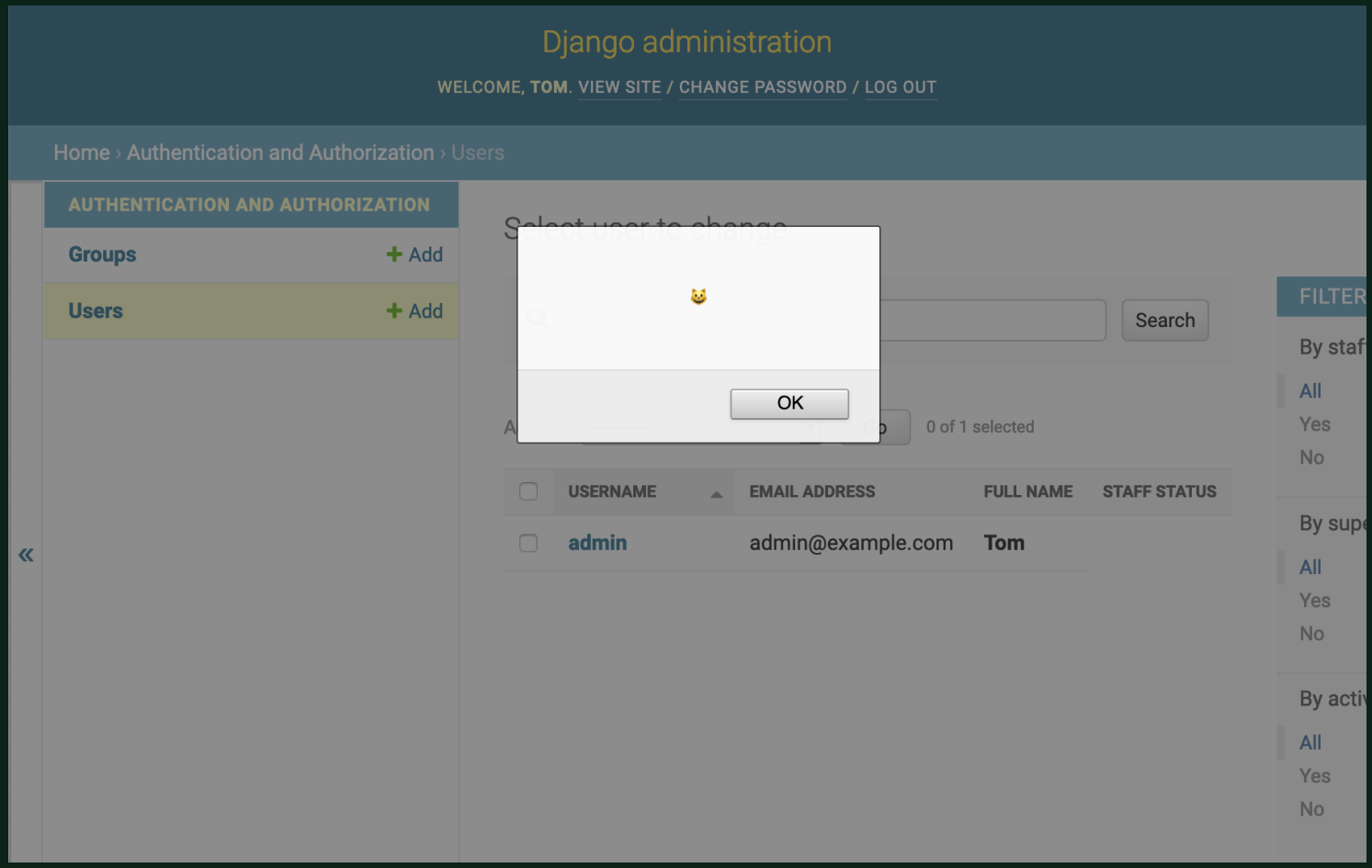
Available on the App Store

✍️ Step 1: Set name to include script tag

For example:

Tom `<script>alert("😸")</script>`

👀 Step 2: Someone browses the admin



🚨 mark_safe is not safe 🚨

```
class CustomUserAdmin(UserAdmin):
    list_display = ["username", "email",
                    "full_name", "is_staff"]

    def full_name(self, obj):
        return mark_safe(
            "<strong>"
            + f"{obj.first_name} {obj.last_name}"
            + "</strong>"
        )
```

HTML in user data *unsafely* injected into admin.

✅ Use `format_html`

```
class CustomUserAdmin(UserAdmin):
    list_display = ["username", "email",
                    "full_name", "is_staff"]

    def full_name(self, obj):
        return format_html(
            "<strong>{} {}</strong>",
            obj.first_name,
            obj.last_name,
        )
```

For lists: `format_html_join`

How to Hack

- Try adding HTML in every field you can.
- Beacon resources - script/image/etc.

🦸‍♀️ How to Protect

- Don’t copy code with `mark_safe`.
- Rename `mark_safe`?
- Block untrusted resources with `Content-Security-Policy`

📚 Story 4 : Lotsa sites

Recurring audit issue.

Unsafe JavaScript Data Passing in Templates

Importance: High

Effort: Medium

Description:

Currently there are a number of instances of unsafe JavaScript data passing in the templates. I didn't audit every `<script>` tag, but I found a number of unsafe inclusions in these templates:

- `example_1.html`
- `example_.html`
- `example_3.html`

These are unsafe because an attacker could set the data to perform HTML injection and include arbitrary content, and potentially steal users' data. I believe most of this data is internally set, but that just reduces the risk and does not eliminate it.

The only safe way to pass data to JavaScript provided by Django is the `json_script` filter. This is explained in my post Safely Including Data for JavaScript in a Django Template.

Such injection potential is also a reason to block inline `<script>` tags with Content-Security-Policy as per the Security Headers issue.

😈 Step 1: Include HTML in field you control

For example:

Adam `</script><script>alert('😸')</script>`

🤖 Step 2: View Forms “JSON” Data

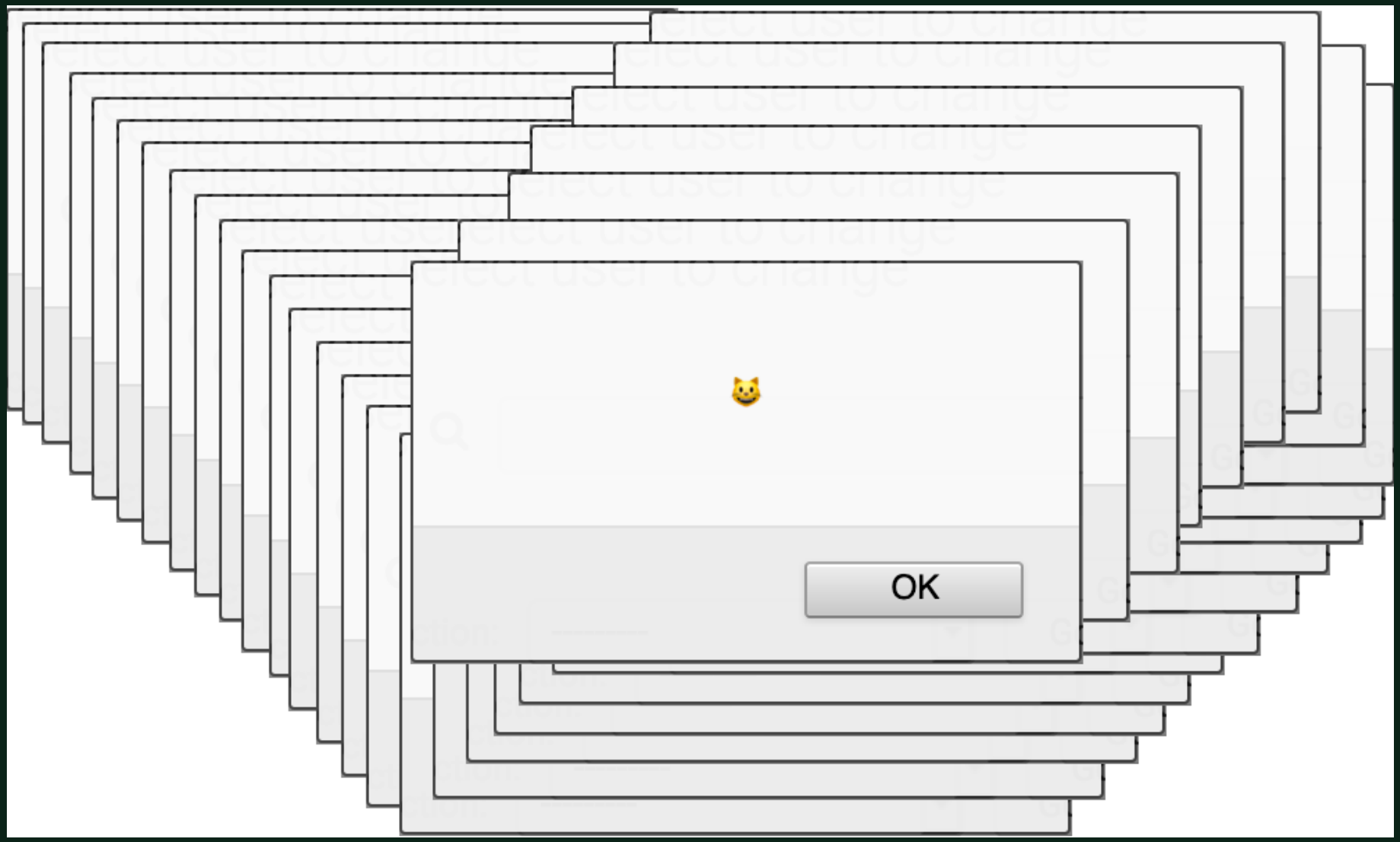
```
def index(request):
    user_json = json.dumps({
        name: request.user.full_name,
    })
    return render(
        request,
        'index.html',
        context={"user_json": user_json}
    )
```

🤖 Step 3: Template Inlines JSON

```
<script>
    const user = {{ user_json|safe }};
</script>
```

`mark_safe` by another name...

Step 4: Cat Alert



🚨 HTML PARSES FIRST 🚨

```
<script>
    const user = {
        name: "Adam </script><script>alert('😸')</script>"
    };
</script>
```

Two script tags plus some ignored junk!

✅ Use `json_script`

```
{{ user|json_script:"user" }}
<script>
    const user = JSON.parse(
        document.getElementById('user').textContent
    );
</script>
```

✅ Use `json_script`

```
<script id="user" type="application/json">{
    "name": "Adam \u003C/script\u003E\u003Cscript\u003Ealert('
}</script>
<script>
    const user = JSON.parse(
        document.getElementById('user').textContent
    );
</script>
```

See post: Safely Including Data for JavaScript in a Django Template.

How to Hack

- Look for inline <script> tags with data.
- Try adding HTML starting `</script>` in every field you can.

🦸‍♀️ How to Protect

- Use `json_script`.
- Block untrusted resources with `Content-Security-Policy`

🦸‍♀️ Bonus Protection: security.txt

security.txt

A proposed standard which allows websites to define security policies.

Read the latest Internet draft →

https://example.com/.well-kno

example.com/.well-known/security.txt

Guest

```
# Our security address
Contact: mailto:security@example.com

# Our OpenPGP key
Encryption: https://example.com/pgp-key.txt

# Our security policy
Policy: https://example.com/security-policy.html

# Our security acknowledgments page
Acknowledgments: https://example.com/hall-of-fame.html
```

Summary

security.txt view

```
@require_GET
def security_txt(request):
    lines = [
        "Contact: mailto:security@example.com",
    ]
    return HttpResponse(
        "\n".join(lines),
        content_type="text/plain"
    )
```

See post: How to add a .well-known URL to your Django site.

security.txt example

https://adamj.eu/.well-known/security.txt


```
-----BEGIN PGP SIGNED MESSAGE-----
Hash: SHA256

Contact: me@adamj.eu
Contact: https://twitter.com/adamchainz
Encryption: https://keybase.io/adamchainz
Acknowledgments: https://docs.djangoproject.com/en/dev/internals/security/
Preferred-Languages: en
Canonical: https://adamj.eu/.well-known/security.txt
-----BEGIN PGP SIGNATURE-----

iQIzBAEBCAAdFiEEkTFpJKRsVwsHfYzR7HElyTSIO+UFAl5G2NMACgkQ7HElyTSI
O+XZig//XYkCtK8WZMCiYhlfcdKpLZFeDrz5XR7zmJs7x7vsBx5smfIFEGOFXzLw
wuNHIjXBpc6NkPywLPhtda/sLWosiS3jNf8HwfGNVOaP7pgVy56qoL5N0Y1w2S+4
E+34GIsA987cXXi+RIq/Qs1bU3oT21W1lD3ZRc+ZY4EG33b8bZoEMCWLDJ9a3HwW
HN36PIK/b3JVYwQ9+3mhraO5sbtKVH1nTKyXYoPUi4RcA31E68o9iClA8n/N+JpG
C1Ad7XlwEht/soh71/MoDb23KOTWqVYZiJZQAUkOhA5R1/6V3EHxYXtwTFUKdQBr
R5qg8SYKfvjjhfQ7H1eDB/sIiRSvRn9QebFTbIdNVqVBKeOuob5T0Gh6AmrBT2ut
6WsjyWRtCWSEMhu/R835X/G6xwO3vJs3kgp3XjGEU9/AixN8EydnorQ/oplA7ODv
wUm2XggOo0AiKqVaZOrbLm4xlPCg7hKlJ8WG2GLXLWCZiW57Za6zFCIwjCXcYOM2
EaaIOVzktn4nBB+bwsRCotevX8YskwcFtY3vH7O4Dtp+eiFEklv35WjWp6LXkejn
vtifucCzje3eAR3PqN0DkZSteD0CB+9MrNbiPA4xY9KelBf5AuMDRynjJ6zcWxev
ObrxMwfxS4fia7zXtcy15DLFasFdGl62QBnTtXfPZ7gC6qZAPvg=
=djyv
-----END PGP SIGNATURE-----
```

Thank you! 🤗

- Adam Johnson
- @adamchainz on GitHub & Twitter
- me@adamj.eu
- github.com/adamchainz/talk-how-to-hack-a-django-website