Fine penetration tests for fine websites

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

Pentest-Report Ente Platform, Cryptography & Infra 10.2025

Cure53, Dr.-Ing. M. Heiderich, Y. Yuan, Dr. N. Kobeissi, S. Rajbhar

Index

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

Introduction

“Ente is a service that provides a fully open source, end-to-end encrypted platform for you to store your data in the cloud without needing to trust the service provider.”

From https://github.com/ente-io/ente/?tab=readme-ov-file#ente

This report (ID ENT-02) has been compiled by Cure53 to verify the results of a penetration test and source code audit targeting the Ente platform, server cryptography, underlying Compute Instances infrastructure, and selected feature sets.

For background information, the project was commissioned by Ente in August 2025 and conducted soon thereafter in CW42 October 2025. The scope examinations comprised a total of fourteen evaluation days, which were dispersed to a four-person review team.

The aforementioned focus elements were placed into four distinct Work Packages (WPs) for test execution efficiency. These were defined as follows:

- **WP1:** White-box code audits & reviews against ente server crypto functions
- **WP2:** White-box pen.-tests & code audits against ente authn & authz, ACL
- **WP3:** White-box pen.-tests & code audits against ente specific functionalities
- **WP4:** Gray-box pen.-tests & assessments against ente compute instances

Certain aspects of the Ente platform have already been previously investigated by Cure53. Specifically, the crypto designs were vetted in March 2023; the outcomes of which have been documented in the report entitled *ENT-01*.

The pentesting methodology deemed most appropriate for this exercise was a dual white- and gray-box approach. To fulfill these requirements, the internal maintainers granted the Cure53 analysts access to sources, documentation, and other assorted assets. Preliminary initiatives were finalized in CW41 2025 to ensure maximal and unimpeded coverage.

A dedicated Discord server was established for cross-organization communications. All involved personnel from both parties were invited to join this platform in advance. The collaborative process was handled efficiently, with minimal queries required and no significant blockers encountered.

Cure53 kept the in-house team in the loop with regard to the progress of the audit and interesting findings, furnishing frequent status updates when required. Live reporting was also requested and conducted via the shared Discord server.

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

The Cure53 consultants achieved satisfactory analysis depth over the WP1-WP4 scope features, detecting a total of fifteen negative security implementations (subsequently provided in ticket format). In terms of their categorization, ten were filed as security vulnerabilities and five pertained to general weaknesses with lower exploitation potential.

In summary, the Ente platform is affected by a variety of security vulnerabilities across its authentication processes, email management, file handling, and infrastructure elements. Despite the strong foundational elements, such as robust server-side ACL enforcement and generally sound coding practices, multiple major flaws were uncovered that necessitate urgent remediation, including the uncovered XSS pitfalls and fake session mechanism, which expose the platform to unnecessary risk.

Nonetheless, the Ente team's proactive responsiveness in swiftly rremediating twelve of the fifteen issues, including all Critical and High severity issues, is praiseworthy. These efforts, in tandem with the astutely architected system, demonstrates their commitment to security and should be commended. Notably, some of the located pitfalls were classified as out-of-scope, such as the desktop client RCE. Given their presence, a dedicated review of these components is advised for future engagements.

The subsequent sections of this report will detail the scope, test setup, and available testing resources. The report will then present all findings in chronological order, beginning with identified vulnerabilities and subsequently addressing general weaknesses. Each finding will include a technical description, a proof of concept (PoC) where applicable, and mitigation or remediation advice. Finally, the report will conclude with a summary of Cure53's overall impressions and an assessment of the perceived security posture of the Ente platform and respective components.

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

Scope

- **Pen.-tests & code audits against Ente platform, cryptography & infra**
  - **WP1:** White-box code audits & reviews against ente server crypto functions
    - **Source repository URL:**
      - https://github.com/ente-io/ente
    - **Folder *server* only*:***
      - Branch: *main*
      - Commit hash: *b6c35924fe7b3c4b8792144eff5b0ed235f58411*
  - **WP2:** White-box pen.-tests & code audits against ente authn & authz, ACL
    - **Environment URL:**
      - https://api.ente.io/
    - **Source code:**
      - *See WP1*
  - **WP3:** White-box pen.-tests & code audits against ente specific functionalities
    - **Source code:**
      - *See WP1*
    - *Focus should be placed on the following*
    - **Authentication & Authorization**
      - Token Validation: /pkg/middleware/auth.go:33-90
      - File Access Control: /pkg/controller/access/file.go:34-68
      - Collection Access: /pkg/controller/access/collection.go:27-59
    - **File Operations**
      - File Creation: /pkg/controller/file.go:118-201
      - File Update: /pkg/controller/file.go:204-291
      - File Download: /pkg/controller/file.go:321-336
      - File Trash: /pkg/controller/file.go:439-463
      - File Copy: /pkg/controller/file_copy/file_copy.go:64-167
    - **Collection Sharing**
      - Share Collection: /pkg/controller/collections/share.go:18-60
      - Public Link Creation: /pkg/controller/collections/share.go:184-207
      - Join via Link: /pkg/controller/collections/share.go:61-107
    - **Public Access**
      - Collection Link Auth: /pkg/middleware/collection_link.go:51-125
      - File Link Auth: /pkg/middleware/file_link.go:38-114
  - **WP4:** Gray-box pen.-tests & assessments against ente compute instances
    - *A list of IPs in scope has been shared with Cure53*
  - **Test-supporting material was shared with Cure53**
  - **All relevant sources were shared with Cure53**

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

Identified Vulnerabilities

The following section lists all vulnerabilities and implementation issues identified during the testing period. Notably, findings are cited in chronological order rather than by degree of impact, with the severity rank offered in brackets following the title heading for each vulnerability. Furthermore, all tickets are given a unique identifier (e.g., *ENT-02-001*) to facilitate any future follow-up correspondence.

ENT-02-001 WP2: Weak replay attack protection on WebAuthn *(Medium)*

***Fix Note**: This issue has been fixed by the Ente team and verified by Cure53 to be working as expected. The described issue no longer exists.*

Cure53 confirmed that Ente's current WebAuthn authentication implementation fails to prevent replay attacks outside of the 2-minute timeout limit. An attacker can intercept a WebAuthn *finish* request and replay it to create an arbitrary amount of sessions within the aforementioned time frame.

Steps to reproduce:

1. Set up an Ente account with Webauthn (Passkeys) enabled.
2. Attempt to log in and intercept the request to *https://api.ente.io/users/two-factor/passkeys/finish?sessionID= [...]*
3. Replay the request.
4. Verify that each replayed request generates a new session and accompanying *new session* email.

To resolve this vulnerability, Cure53 recommends either deleting the challenge upon operation completion or marking the challenge as *used* in the database. According to the Web Authentication specification[1], the Relying Party should store the generated challenges "temporarily until the operation is complete."

ENT-02-002 WP3: Cloudflare bypass via Cloudflare Spectrum *(Medium)*

***Client Note:** Ente believes this to be an Info/Low priority issue, since (a) Ente is on an Enterprise Cloudflare account and uses Cloudflare's Advanced "adaptive" DDoS protection, and (b) Ente no longer relies on IP-based protection to prevent brute-force TOTP attacks.*

Ente's backend infrastructure hosted behind Cloudflare currently leverages IP whitelisting to ensure that only Cloudflare-origin traffic is allowed. However, this setup can be bypassed via Cloudflare Spectrum, allowing an attacker to bypass any security rules configured on the Cloudflare WAF, as well as override the *CF-Connecting-IP* header to spoof source IP address. This would also circumvent a number of other security measures provided by Ente, including rate limiting that serves to prevent 2FA TOTP brute force attacks.

[1] https://www.w3.org/TR/webauthn-2/#sctn-cryptographic-challenges

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

To resolve this vulnerability, Cure53 recommends enabling per-hostname authenticated origin pulls (mTLS) between Cloudflare[2] and the backend.

ENT-02-003 WP2: Client IP spoofing for self-hosted instances *(Medium)*

***Fix Note**: This issue has been fixed by the Ente team and verified by Cure53 to be working as expected. The described issue no longer exists.*

Generally speaking, Ente assumes that all self-hosted instances are hosted behind Cloudflare, since the *CF-Connecting-IP* header is leveraged to determine the client's IP address. In light of this, self-hosted Ente instances that do not utilize Cloudflare are hence vulnerable to client IP spoofing attacks if an attacker sets the *CF-Connecting-IP* header themselves. Similarly to ENT-02-002, this exposes self-hosted Ente instances to brute force attacks for TOTP 2FA.

Affected file:
server/pkg/utils/network/network.go

Affected code:

```
func GetClientIP(c *gin.Context) string {
	ip := c.GetHeader("CF-Connecting-IP")
	if ip == "" {
		ip = c.ClientIP()
	}
	return ip
}
```

To resolve this vulnerability, Cure53 recommends forcing all self-hosting Ente users to specify their reverse proxy setup. In addition, Ente should consider applying the standardized *X-Forwarded-For* header for compatibility with a wide range of reverse proxies, rather than parsing *CF-Connecting-IP*.

ENT-02-005 WP3: Email boundary injection in *report abuse* endpoint *(Medium)*

***Fix Note**: This issue has been fixed by the Ente team and verified by Cure53 to be working as expected. The described issue no longer exists.*

When sending emails using SMTP, Ente manually generates emails using basic string interpolation and a static email MIME boundary value. In the event that email bodies include user input, an attacker can inject a boundary value and additional attachments to the sent emails. This behavior enables phishing techniques by adding malicious PDFs or other attachments to an abuse report email.

[2] https://developers.cloudflare.com/ssl/origin-configuration/authenticated-origin-pull/set-up/per-hostname/

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

Affected file:
server/pkg/utils/email/email.go

Affected code:
```
func sendViaSMTP(toEmails []string, fromName string, fromEmail string,
subject string, htmlBody string, inlineImages []map[string]interface{})
error {
	[...]
	header := "From: " + fromName + " <" + fromEmail + ">\n" +
		"To: " + emailAddresses + "\n" +
		"Subject: " + subject + "\n" +
		"MIME-Version: 1.0\n" +
		"Content-Type: multipart/related; boundary=boundary\n\n" +
		"--boundary\n"
	htmlContent := "Content-Type: text/html; charset=us-ascii\n\n" +
htmlBody + "\n"

	emailMessage = header + htmlContent
```

PoC:
```
curl -v \
  https://api.ente.io/public-collection/report-abuse \
  -H "x-auth-access-token: <album access token>" \
  --json '{
  "url": "https://example.com",
  "reason": "MALICIOUS_CONTENT",
  "details": {
    "fullName": "CURE53 TEST",
    "email": "larry@volt.cure53.de",
    "signature": "CURE53 TEST",
    "comment": "\n--boundary\nContent-Type: text/html\n\nCURE53TEST\n",
    "onBehalfOf": "CURE53 TEST",
    "jobTitle": "CURE53 TEST",
    "address": {
      "street": "CURE53 TEST",
      "city": "CURE53 TEST",
      "state": "CURE53 TEST",
      "postalCode": "CURE53 TEST",
      "country": "CURE53 TEST",
      "phone": "CURE53 TEST"
    }
  }}'
```

To resolve this vulnerability, Cure53 recommends utilizing an appropriate library to generate MIME emails, rather than string interpolation. Alternatively, a cryptographically secure random boundary could be generated to remediate this specific flaw.

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

ENT-02-007 OOS: Full read SSRF in Uploader Worker *(High)*

Fix note: *This issue was promptly mitigated by the Ente team during the testing phase. The fix was verified by Cure53 and the problem no longer exists.*

During the assessment, Cure53 identified a Server-Side Request Forgery (SSRF) vulnerability in the Uploader Worker used via *https://web.ente.io/* to upload user files. The *handlePOSTOrPUT* function reads the *UPLOAD-URL* header directly from the incoming request and passes it for fetching without any validation or restriction. As such, an adversary can issue arbitrary HTTP requests from the server's network context and read full responses, leading to a full read SSRF inside the worker.

Affected file:
infra/workers/uploader/src/index.ts

Affected code:

```
const handlePOSTOrPUT = async (request: Request) => {
    [...]
    const uploadURL = request.headers.get("UPLOAD-URL");
    let response: Response;
    switch (url.pathname) {
        case "/file-upload":
            response = await fetch(uploadURL, {
        [...]
        case "/multipart-upload":
            response = await fetch(uploadURL
```

cURL command:

```
curl -i -X POST 'https://uploader.ente.io/file-upload' \
  -H 'Upload-Url: https://cure53.de' \
  -H 'Content-Length: 0'
```

To resolve this vulnerability, Cure53 recommends enforcing strict allow-listing of upload URL domains and validating the *UPLOAD-URL* header prior to performing the request.

ENT-02-008 WP2: TOTP authentication vulnerable to code reuse *(Low)*

Fix Note*: This issue has been fixed by the Ente team and verified by Cure53 to be working as expected. The described issue no longer exists.*

Cure53 verified that the Two-Factor Authentication (2FA) implementation in Ente's server stack fails to track or prevent the reuse of Time-based One-Time Password (TOTP) codes. Once a valid TOTP code is generated, it can be utilized multiple times within its validity window.

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

The vulnerable code in *VerifyTwoFactor()* and *EnableTwoFactor()* validates the TOTP code; however, the implementation never records that the code has been utilized.

Affected file:
server/pkg/controller/user/twofactor.go

Affected code:

```
// server/pkg/controller/user/twofactor.go (lines 96-99)
valid := totp.Validate(otp, secret)
if !valid {
    return ente.TwoFactorAuthorizationResponse{},
stacktrace.Propagate(ente.ErrIncorrectTOTP, "")
}
// No tracking of used codes - same OTP can be validated multiple times

// Similarly in EnableTwoFactor():
for index, encryptedSecret := range encryptedSecrets {
    secret, err := crypto.Decrypt(encryptedSecret.Cipher,
c.SecretEncryptionKey, encryptedSecret.Nonce)
    if err != nil {
        return stacktrace.Propagate(err, "")
    }
    valid = totp.Validate(request.Code, secret)
    if valid {
        // No tracking - same code can enable multiple 2FA setups
```

A threat actor that intercepts or observes a valid TOTP code can reuse it multiple times within the validity window, enabling several attack scenarios:

- **Man-in-the-Middle replay attack:** An attacker intercepting network traffic can capture valid TOTP code and leverage it to authenticate prior to expiration, even if the legitimate user has already used it.
- **Race condition exploitation:** An attacker can issue multiple parallel authentication requests using the same intercepted TOTP code, potentially bypassing other security controls.

To resolve this vulnerability, Cure53 advises implementing TOTP code tracking, thus ensuring that each code can only be used once. This can be achieved by storing code in a Redis cache structure or similar entity. Alternatively, other database-level tracking mechanisms can be employed for this purpose.

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

ENT-02-012 WP3: Stored XSS on *files.ente.io* allows phishing ATO *(High)*

***Fix Note**: This issue has been fixed by the Ente team and verified by Cure53 to be working as expected. The described issue no longer exists.*

Cure53 noticed that the *Content-Type* header remains unsigned when signing S3 URLs for file uploads. As a result, an attacker could upload a text/html document that leads to XSS when served using *files.ente.io*. This can subsequently be escalated to a phishing account takeover (even if a user enables Passkey 2FA), since *accounts.ente.io/passkeys/verify* allows redirects back to any **.ente.io*. In combination, these factors permit an adversary to serve a phishing page on *files.ente.io* for the purpose of obtaining the victim's password, then bypass passkey 2FA using the redirect for a full account takeover.

Affected file:
server/pkg/controller/file.go

Affected code:
```
func (c *FileController) getObjectURL(s3Client *s3.S3, dc string, bucket
*string, objectKey string) (ente.UploadURL, error) {
	r, _ := s3Client.PutObjectRequest(&s3.PutObjectInput{
		Bucket: bucket,
		Key:    &objectKey,
	})
	url, err := r.Presign(PreSignedRequestValidityDuration)
```

Steps to reproduce:

1. Create an Ente account and enable Passkey 2FA.
2. Upload an image to Ente, noting the S3 pre-signed URL and *fileID*.
3. Create an HTML file with the following content and save it:

   Content:
   ```
   <script>alert(location.href)</script>
   ```

4. Run the following command:

   Command:
   ```
   curl <s3 url> --upload-file <html file> -H 'Content-Type: text/html'
   ```

5. Attempt to log in using another browser without an active Ente session.
6. Proceed to the Passkey authentication step.
7. Replace the *redirect* query parameter with the following value:

   Parameter replacement:
   ```
   https://files.ente.io/?fileID=<fileId>&token=<ente auth token>
   ```

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

8. Verify that an alert box appears after completing Passkey authentication, which includes the account's encrypted master/recovery keys.

To resolve this vulnerability, Cure53 recommends signing all S3 upload URLs with a *Content-Type* header such as *application/octet-stream*. In addition, a similar header should be present when serving user-generated content. Lastly, a stricter allow-list for the *redirect* query parameter during Passkey authentication should be enforced, rather than permitting any **.ente.io* subdomain.

ENT-02-013 WP3: Storage limit bypass via reuploading *(Medium)*

***Fix Note**: This issue has been fixed by the Ente team and verified by Cure53 to be working as expected. The described issue no longer exists.*

The observation was made that Ente generates S3-compatible pre-signed upload links for user photo uploads. After the user uploads the data to the storage, a request is issued to */files* with the photo's metadata. Ente verifies that the uploaded file comprises the specified size and the metadata is subsequently stored, which is the method by which a user's storage quota is calculated.

However, a user can overwrite the uploaded file with a greater volume of data after this process, which would not be calculated as part of the storage quota. Accordingly, users can utilize Ente as an unlimited file storage service, even with a free account.

Steps to reproduce:

1. Upload an image to Ente and note the pre-signed URL.
2. Run the following command:

   Command:
   ```
   curl <s3 url> --upload-file <bigger file>
   ```

3. Note that viewing the images is no longer possible, but one can retrieve the raw data via Chrome DevTools or a similar request inspector.
4. Confirm that the user's Ente storage usage status has not altered.

To resolve this vulnerability, Cure53 recommends determining the size of the upload prior to uploading it, then signing the *content-length* header of the S3 upload URL.

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

ENT-02-014 OOS: XSS on *accounts.ente.io* via Passkey recovery *(High)*

Fix note: *This issue was promptly mitigated by the Ente team during the testing phase. The fix was verified by Cure53 and the problem no longer exists.*

In general, Ente's Passkey authentication flow adopts a URL query parameter to ascertain the user's redirection location for recovery. However, the URL is never validated when the user clicks the *Recover two-factor* button. As such, an attacker could utilize a *javascript:* URI to run arbitrary JS when the button is clicked.

Affected file:
web/apps/accounts/src/pages/passkeys/verify.tsx

Affected code:

```
const handleRecover = (() => {
    const searchParams = new URLSearchParams(window.location.search);
    const recover = nullToUndefined(searchParams.get("recover"));
    if (!recover) {
        // [Note: Conditional passkey recover option on accounts]
        //
        [...]
        return undefined;
    }

    return () => redirectToPasskeyRecoverPage(new URL(recover));
})();
```

PoC:
https://accounts.ente.io/passkeys/verify?clientPackage=io.ente.photos.web&passkeySessionID=fake&redirect=https%3A%2F%2Fweb.ente.io%2Fpasskeys%2Ffinish&recover=javascript:alert(origin)

To resolve this vulnerability, Cure53 recommends validating that the redirect constitutes an HTTPS URL leading to a valid Ente recovery page. For additional defense-in-depth, the dev team should incorporate a Content-Security-Policy (CSP) for all Ente web applications.

ENT-02-015 WP2: SRP fake session mechanism enables user enum *(High)*

Fix Note*: This issue has been fixed by the Ente team and verified by Cure53 to be working as expected. The described issue no longer exists.*

The Secure Remote Password (SRP) authentication implementation attempts to prevent user enumeration by returning fake session data for non-existent users via the *fCreateSession()* function. However, this protection mechanism is flawed, rendering user enumeration attempts more viable than no protection whatsoever.

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

Affected file:
server/pkg/controller/user/srp.go

Affected code:

```
func (c *UserController) CreateSrpSession(context *gin.Context, req
ente.CreateSRPSessionRequest) (*ente.CreateSRPSessionResponse, error) {
    srpAuthEntity, err :=
c.UserAuthRepo.GetSRPAuthEntityBySRPUserID(context, req.SRPUserID)
    if err != nil {
        if errors.Is(err, sql.ErrNoRows) {
            return fCreateSession(req.SRPUserID.String(), req.SRPA)  //
Returns fake session
        }
        return nil, stacktrace.Propagate(err, "failed to get srp auth
entity")
    }
    // ... real session creation
}

[...]

// server/pkg/controller/user/srp.go (lines 327-342)
func fCreateSession(srpUserID string, srpA string)
(*ente.CreateSRPSessionResponse, error) {
    srpABytes := convertStringToBytes(srpA)
    [...]
    srpBBytes := make([]byte, 512)
    _, err := rand.Read(srpBBytes)
    if err != nil {
        return nil, stacktrace.Propagate(err, "failed to generate random
bytes")
    }

    return &ente.CreateSRPSessionResponse{
        SessionID: uuid.New(),  // NOT STORED IN DATABASE
        SRPB:      convertBytesToString(srpBBytes[:512]),  // RANDOM, NOT
VALID SRP
    }, nil
}
```

The fake session mechanism is affected by three fundamental vulnerabilities that enable reliable user enumeration. Firstly, fake session IDs are never stored in the database. When a client attempts to verify the session, the database lookup fails with *sql.ErrNoRows*, immediately revealing that the user does not exist. Secondly, timing differences are detectable. Real user session creation takes variable time (estimated at 50-100ms), while fake user session creation requires a consistent 20ms due to a fixed 20ms sleep delay and

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

minimal random variation, which renders detection trivial. Thirdly, the take SRPB value consists of 512 bytes of random data, rather than a valid SRP public ephemeral value that should be computed as $B = (k*v + g^b) \bmod n$ according to the SRP protocol. A cryptographically sophisticated attacker could detect this by analyzing the mathematical properties of the returned values.

An adversary can enumerate all user email addresses in the system with 100% accuracy using timing analysis to distinguish real users (50-100ms response time) from fake users (20-22ms). This can be achieved by attempting to verify the session, whereby fake sessions return database errors while real sessions return authentication errors, or via statistical analysis exploiting the low variance in fake responses versus high variance in real responses.

Plausible ramifications of a successful exploit include targeted phishing, since attackers will know with certainty which email addresses use Ente; password spraying focused only on valid accounts; privacy violations by revealing user account existence; competitive intelligence allowing competitors to identify Ente's customer base; and correlation attacks that cross-reference with other data breaches.

To resolve this vulnerability, Cure53 advises improving the fake session mechanism or removing it entirely, as the half-broken protection may actually incur greater risk than no protection at all. Ideally, fake sessions should be stored in the database with a deterministic fake verifier generated via the *srpUserID*, therefore ensuring consistency across requests. Alternatively, for simpler implementations whereby perfect enumeration protection is non-critical, the fake session logic should be removed and consistent error messages returned for all authentication failures.

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

Miscellaneous Issues

This section covers any and all noteworthy findings that did not incur an exploit but may assist an attacker in successfully achieving malicious objectives in the future. Most of these results are vulnerable code snippets that did not provide an easy method by which to be called. Conclusively, while a vulnerability is present, an exploit may not always be possible.

ENT-02-004 WP2: OTTs for specific operations valid for all *(Info)*

Cure53 confirmed that Ente's One Time Tokens (OTTs) sent by email are only ever associated with the receiving email address. As a result, an OTT sent for registration validation and another sent for email change user verification are both valid for either operation, even if the two operations are associated with different sessions. When sending an OTT email, the associated correspondence specifies its intended purpose, which may mislead users into believing that the token is limited to one purpose only.

To resolve this issue, Cure53 recommends associating OTTs with specific sessions and operations, ensuring that the server enforces the intended OTT purpose.

ENT-02-006 WP2: Unexploitable email header injection *(High)*

***Fix Note**: This issue has been fixed by the Ente team and verified by Cure53 to be working as expected. The described issue no longer exists.*

Similarly to the flaw outlined in ticket ENT-02-005, Cure53 noticed that the subject header line for sending emails is generated using string interpolation. If an injection point exists that allows an attacker to specify an unconstrained value that is eventually added to the subject header, they may be able to inject arbitrary content and present any layout of their choosing.

While Cure53 did not locate any corresponding injection points during this audit, they may be inadvertently integrated in the future and hence should be monitored.

Affected file:
server/pkg/utils/email/email.go

Affected code:

```
func sendViaSMTP(toEmails []string, fromName string, fromEmail string,
subject string, htmlBody string, inlineImages []map[string]interface{})
error {
	[...]
	header := "From: " + fromName + " <" + fromEmail + ">\n" +
		"To: " + emailAddresses + "\n" +
		"Subject: " + subject + "\n" +
		"MIME-Version: 1.0\n" +
		"Content-Type: multipart/related; boundary=boundary\n\n" +
```

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

```
            "--boundary\n"
        htmlContent := "Content-Type: text/html; charset=us-ascii\n\n" +
htmlBody + "\n"

        emailMessage = header + htmlContent
```

To resolve this issue, Cure53 recommends leveraging a library for MIME email generation, rather than appending strings.

ENT-02-009 OOS: Potential RCE in desktop application *(**Critical**)*

***Fix Note**: This issue has been fixed by the Ente team and verified by Cure53 to be working as expected. The described issue no longer exists.*

Cure53 noticed that *shell.openExternal* is utilized in the Ente desktop application to open external links in a browser, which is inherently unsafe since *shell.openExternal* can lead to RCE in certain cases.

This target was Out-of-Scope (OOS) for this pentest. As a result, the test team refrained from investigating the vector's exploitation conditions.

Affected file:
desktop/src/main.ts

Affected code:

```
const allowExternalLinks = (webContents: WebContents) =>
    [...]
    webContents.setWindowOpenHandler(({ url }) => {
        if (!url.startsWith(rendererURL)) {
            [...]
            void shell.openExternal(url);
            return { action: "deny" };
        } else {
            return { action: "allow" };
        }
    });
```

To resolve this issue, Cure53 recommends ensuring that all external links constitute specific trusted protocols prior to opening them, such as HTTPS.

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

ENT-02-010 WP2: Non-constant-time comparison for Passkey recovery *(Low)*

***Fix Note**: This issue has been fixed by the Ente team and verified by Cure53 to be working as expected. The described issue no longer exists.*

Cure53 observed that the *!=* operator is employed when determining if a user's input recovery key matches the entity in the database for Passkey recovery. However, this is not a constant-time comparison. In specific circumstances, an attacker may be able to leverage this information as a side-channel to brute force for the user's recovery key. Albeit, exploitation is considered highly unlikely at present.

Affected file:
server/pkg/repo/two_factor_recovery/repository.go

Affected code:

```
// ValidatePasskeyRecoverySecret checks if the passkey skip secret is valid
for a user
func (r *Repository) ValidatePasskeyRecoverySecret(userID int64, secret
string) (bool, error) {
	[...]
	if secret != serverSkipSecretKey {
		logrus.Warn("invalid passkey skip secret")
		return false, nil
	}
	return true, nil
}
```

To resolve this issue, Cure53 recommends employing a constant-time comparison function that nullifies comparison side-channel attacks.

ENT-02-011 WP2: Disabling 2FA does not require additional auth *(Info)*

During the analysis, Cure53 confirmed that additional authentication measures are not required when disabling/altering TOTP 2FA or adding new Passkeys, which is suboptimal from a security viewpoint.

To resolve this issue, Cure53 recommends enforcing additional verification when altering critical account security settings. This revised approach will help to guarantee that the operation is authorized by the user in the question, rather than a malicious party.

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

Conclusions

This penetration test and source code audit of the Ente platform confirmed the presence of a range of security vulnerabilities spanning the authentication mechanisms, access controls, email handling, file operations, and infrastructure components. While the platform offers a solid foundation characterized by optimal server-side ACL enforcement and generally secure coding practices, several *High*-severity pitfalls were identified that require immediate attention.

Cure53 extensively probed the Passkey 2FA authentication and recovery protocols, yielding two distinct findings in this area. The first confirms that the Passkey challenge-response system used by Ente is vulnerable to challenge reuse/replay attacks (ENT-02-001), while the second denotes an OOS XSS vulnerability in the Passkey recovery mechanism (ENT-02-014).

While considering plausible attacks against TOTP 2FA systems, Cure53 determined that Ente's protection against brute force attacks is subpar, as it relies on spoofable IP addresses (ENT-02-002 and ENT-02-003). The dev team must ensure that request-related IP address information is obtained from trusted sources.

Following a comprehensive evaluation of the various sharing features, Cure53 noted the viability of a MIME email boundary injection attack when sending emails for abuse reported received on a shared collection (ENT-02-005). In addition, an unexploitable header injection issue was noted (ENT-02-006). Both of these drawbacks originate from the same root cause; the outgoing email generation process should be augmented to ensure that injection attacks are definitively nullified.

Cure53's review of the image upload process and S3-compatible upload mechanism yielded two defects emanating from lax header signing (ENT-02-013 and ENT-02-012). Greater strictness should be applied when generating user-employed pre-signed URLs.

A detailed examination of Ente's authentication and authorization mechanisms revealed several security concerns in the 2FA implementation. The TOTP authentication system is susceptible to code reuse (ENT-02-008), permitting multiple usage of the same TOTP within its validity window. This weakness could enable replay attacks and race condition exploitation if an attacker intercepts valid TOTP codes.

The SRP authentication construct attempts to prevent user enumeration via a fake session mechanism; however, this safeguard is fundamentally flawed (ENT-02-015). The *fCreateSession()* function generates fake session data for non-existent users, yet the implementation remains easily detectable via multiple vectors.

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

Fake session IDs are never stored in the database; timing differences are consistent and measurable (20ms for fake vs 50-100ms for real); and the fake SRPB values are cryptographically invalid. Accordingly, user enumeration is increasingly reliable in this setup compared with no protection mechanism at all.

A cryptographic timing side-channel was identified in the Passkey recovery mechanism (ENT-02-010), whereby non-constant-time comparison is adopted for recovery key validation. While exploitation is unlikely in practice, this represents a deviation from security best practices concerning sensitive cryptographic material handling.

The platform generally adheres to contemporary security standards. Session management depends on the *X-Auth-Token* header, which is set and validated by *TokenAuthMiddleware* before being passed to downstream handlers. The critical nature of this authentication mechanism mandated a thorough assessment. The validation process initiates by ensuring that the provided JWT holds a valid signature; thereafter, the corresponding *userId* is retrieved from *UserAuthRepo* and securely configured in the *X-Auth-User-ID* header.

The *GetUserID* method responsible for reading the header was explored for potential discrepancies. The *.Get* and *.Set* methods of the HTTP header are case-insensitive, eliminating the possibility of injecting a duplicate header with an alternate casing. Cure53 attempted to smuggle the header by adding spaces before and after the header name, although these efforts were blocked by the HTTP server prior to reaching the middleware.

The password lock feature enforces access control via a combination of URL whitelisting and JWT validation. The *validatePassword* method sanitizes the request path with a *urlSanitizer* function, which includes additional checks such as maintaining a list of all valid endpoints and verifying that the resolved user-provided path matches one of these endpoints. This setup restricts attackers from applying crafted paths to bypass the whitelist. Requests to any unregistered paths are redirected to */unknown-api*, ensuring that only approved endpoints are accessible.

The cross-platform email handling procedures were thoroughly reviewed for consistency and security. All user-provided emails are normalized by converting to lowercase and trimming spaces prior to further processing. When retrieving user IDs, *c.UserRepo.GetUserIDWithEmail* is called with the normalized email, while a hash is generated using **crypto.GetHash** to safely match against stored email variants. Cure53 tested this construct using emails with uppercase letters, spaces, and Unicode variants. The results confirmed that all email variations are consistently normalized, with no discrepancies found that could allow account takeover or duplication.

The external connector handling was reviewed for security concerns, specifically regarding the *mailing_list* controller and *doListActionZoho* method. Email input is ideally escaped to prevent parameter pollution in the backend Zoho API call. The action parameter that determines the API path is hardcoded, limiting the scope of potential manipulation.

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

Other external connectors such as Wasabi and Listmonk were also vetted, verifying that the input is handled safely.

Elsewhere, an SSRF vector was located in the Uploader Worker, enabling arbitrary HTTP requests from the server's context via the *UPLOAD-URL* header (ENT-02-007). Positively, this shortcoming was promptly mitigated by the Ente team during the testing phase.

In conclusion, Cure53 can confirm that Ente's server-side ACLs are effective, with no major findings stemming from suboptimal ACL enforcement. The inspected functionalities assert that users cannot access alternative user data without authorization. While the Ente maintainers evidently possess ample understanding of modern authentication protocols, the implementation details reveal weaknesses that could be exploited by sophisticated attackers. The fake session mechanism in particular requires significant modification, as the current premise may provide attackers with reliable enumeration capabilities over a simpler error-based approach.

A multitude of findings were either partially or fully OOS for this pentest. In light of this, supplemental reviews of these respective components would prove beneficial. Ente's overall security posture can be significantly strengthened by implementing performant TOTP code tracking; constant-time comparisons for all security-critical operations; stricter header signing for S3 uploads; and email generation upgrades. Despite the detected flaws, the Ente platform reflects the internal development team’s security diligence, underscored by the swift amelioration endeavors and generally well-architected framework.

This security evaluation of the Ente platform resulted in the identification of fifteen total security pitfalls, comprising ten confirmed vulnerabilities and five general weaknesses. The scope of the findings ranged from substantial flaws such as SSRF and XSS, to lower impact implementation concerns within authentication mechanisms. The Ente team exhibited a high degree of security awareness and responsive remediation capabilities, successfully fixing twelve of the fifteen issues, including all *Critical*- and *High*-severity issues prior to the finalization of this report.

The underlying architecture demonstrates solid security fundamentals, especially regarding access control enforcement and authorization logic. However, Ente should strategically prioritize hardening authentication flows, implementing rigorous input validation for interactions with external services, and adopting secure-by-default practices for cryptographic operations.

Cure53 would like to thank Vishnu Mohandas, Manav Rathi, and Neeraj Gupta from the Ente team for their excellent project coordination, support, and assistance, both before and during this assignment.