Zellic

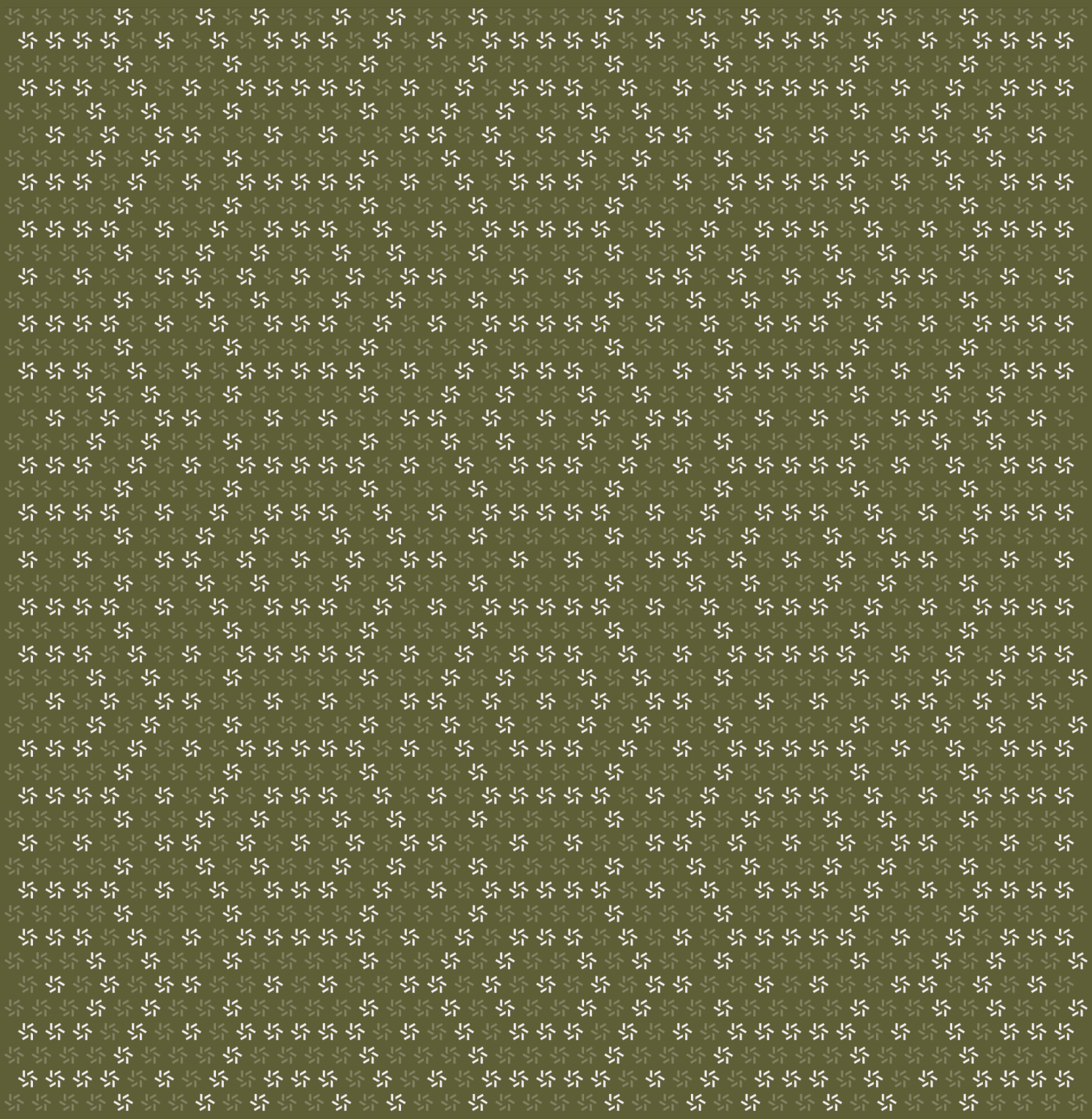
Prepared for
Madhavan Malolan Chetlur
CreatorOS Inc

Prepared by
Junghoon Cho
Juchang Lee
Philip Papurt
Zellic

April 2, 2024

Reclaim Protocol

Web Security Assessment



Contents

About Zellic

Zellic is a vulnerability research firm with deep expertise in blockchain security. We specialize in EVM, Move (Aptos and Sui), and Solana as well as Cairo, NEAR, and Cosmos. We review L1s and L2s, cross-chain protocols, wallets and applied cryptography, zero-knowledge circuits, web applications, and more.

Prior to Zellic, we founded the #1 CTF (competitive hacking) team ↗ worldwide in 2020, 2021, and 2023. Our engineers bring a rich set of skills and backgrounds, including cryptography, web security, mobile security, low-level exploitation, and finance. Our background in traditional information security and competitive hacking has enabled us to consistently discover hidden vulnerabilities and develop novel security research, earning us the reputation as the go-to security firm for teams whose rate of innovation outpaces the existing security landscape.

For more on Zellic's ongoing security research initiatives, check out our website zellic.io ↗ and follow @zellic_io ↗ on Twitter. If you are interested in partnering with Zellic, contact us at hello@zellic.io ↗.

1. Overview

1.1. Executive Summary

Zellic conducted a security assessment for CreatorOS Inc from Feb 27th to March 21st, 2024. During this engagement, Zellic reviewed Reclaim Protocol's code for security vulnerabilities, design issues, and general weaknesses in security posture.

1.2. Goals of the Assessment

In a security assessment, goals are framed in terms of questions that we wish to answer. These questions are agreed upon through close communication between Zellic and the client. In this assessment, we sought to answer the following questions:

- Is the Reclaim Protocol app secure against common web-application vulnerabilities?
- Do the web and mobile apps properly handle the confidentiality and integrity of user data?
- Do the Reclaim Protocol server systems provide necessary security guarantees?

1.3. Non-goals and Limitations

We did not assess the following areas that were outside the scope of this engagement:

- On-chain contracts
- Underlying cloud infrastructure
- Third-party dependencies (e.g., NPM packages)
- Published Reclaim data providers

Due to the time-boxed nature of security assessments in general, there are limitations in the coverage an assessment can provide.

Based on the number of severe findings uncovered during the audit, it is our opinion that the project is not yet ready for production. We strongly advise a comprehensive reassessment before deployment to help identify any potential issues or vulnerabilities introduced by necessary fixes or changes. We also recommend adopting a security-focused development workflow, including (but not limited to) augmenting the repository with comprehensive end-to-end tests that achieve 100% branch coverage using any common, maintainable testing framework, thoroughly documenting all function requirements, and training developers to have a security mindset while writing code.

1.4. Results

During our assessment on the scoped Reclaim Protocol files, we discovered 19 findings. Four critical issues were found. Two were of high impact, five were of medium impact, six were of low impact, and the remaining findings were informational in nature.

Additionally, Zellic recorded its notes and observations from the assessment for CreatorOS Inc's benefit in the Discussion section (4. ↗) at the end of the document.

Breakdown of Finding Impacts

Impact Level	Count
Critical	4
High	2
Medium	5
Low	6
Informational	2

2. Introduction

2.1. About Reclaim Protocol

CreatorOS Inc contributed the following description of Reclaim Protocol:

Reclaim protocol allows for computationally efficient, secure and private generation of proofs of provenance (PoP) completely on client side that users can then share to any third-party. Furthermore, users can generate zero-knowledge proofs of features of their data to avoid sharing sensitive information.

2.2. Methodology

During a security assessment, Zellic works through standard phases of security auditing, including both automated testing and manual review. These processes can vary significantly per engagement, but the majority of the time is spent on a thorough manual review of the entire scope.

Alongside a variety of tools and analyzers used on an as-needed basis, Zellic focuses primarily on the following classes of security and reliability issues:

Unsafe code patterns. Many critical vulnerabilities in web-based applications arise from oversights during development. Missing an authentication check on a single API endpoint can circumvent the entire authentication model of the application. Failure to properly sanitize and encode user input can lead to vulnerabilities like SQL injection or cross-site scripting. We use automated tools to identify unsafe code patterns and perform a thorough manual review for vulnerable code patterns.

Business logic errors. Business logic is the heart of all applications. We manually review logic to ensure that the code implements the expected functionality specified in the platform's design documents. We also thoroughly examine the specifications and designs for inconsistencies, flaws, and vulnerabilities. This involves use cases that open the opportunity for abuse.

External interactions. Web projects contain third-party dependencies and interact with APIs and code that are not under the developer's control. Auditors will review the project's external interactions and identify potentially dangerous behavior, such as implicitly trusting API responses or assuming third-party code functionality.

Code maturity. We review for possible improvements in the codebase in general. We look for violations of industry best practices and guidelines and code quality standards. We also suggest possible improvements to code clarity, documentation, and usability.

For each finding, Zellic assigns it an impact rating based on its severity and likelihood. There is no hard-and-fast formula for calculating a finding's impact. Instead, we assign it on a case-by-case basis based on our judgment and experience. Both the severity and likelihood of an issue affect its impact. For instance, a highly severe issue's impact may be attenuated by a low likelihood. We assign the following impact ratings (ordered by importance): Critical, High, Medium, Low, and Informational.

Zellic organizes its reports such that the most important findings come first in the document, rather than being strictly ordered on impact alone. Thus, we may sometimes emphasize an "Informational" finding higher than a "Low" finding. The key distinction is that although certain findings may have the same impact rating, their *importance* may differ. This varies based on various soft factors, like our clients' threat models, their business needs, and so on. We aim to provide useful and actionable advice to our partners considering their long-term goals, rather than a simple list of security issues at present.

Finally, Zellic provides a list of miscellaneous observations that do not have security impact or are not directly related to the scoped files itself. These observations — found in the Discussion (4. ↗) section of the document — may include suggestions for improving the codebase, or general recommendations, but do not necessarily convey that we suggest a code change.

2.3. Scope

The engagement involved a review of the following targets:

Reclaim Protocol Files

Repositories	https://gitlab.reclaimprotocol.org/reclaim/tls ↗ https://gitlab.reclaimprotocol.org/reclaim/tls-receipt-verifier ↗ https://gitlab.reclaimprotocol.org/reclaim-clients/witness-sdk ↗ https://gitlab.reclaimprotocol.org/reclaim-clients/reclaim-app ↗ https://gitlab.reclaimprotocol.org/reclaim-developer-experience/reclaim-sdk ↗
Versions	tls: `5e310480ebaef6f04ba4f9d5bda222ed70f578cb` tls-receipt-verifier: `7ef383a279d5ce345f4a194028fa4201a41aabf8` witness-sdk: `40b320be40a10b124bb025628155c68798c51170` reclaim-app: `508d4e690b0b3a0fa83fb1d7853e95b0b77be533` reclaim-sdk: `dc5a567d2544fd0aeb62c81c9feee02d78c1ef7a`
Programs	• tls • tls-receipt-verifier/node • witness-sdk • reclaim-app • reclaim-sdk
Type	TypeScript
Platforms	Web2, Mobile

2.4. Project Overview

Zellic was contracted to perform a security assessment with three consultants for a total of nine person-weeks. The assessment was conducted over the course of four calendar weeks.

Contact Information

The following project manager was associated with the engagement:

Chad McDonald
Engagement Manager
chad@zellic.io

The following consultants were engaged to conduct the assessment:

Junghoon Cho
Engineer
junghoon@zellic.io

Juchang Lee
Engineer
lee@zellic.io

Philip Papurt
Engineer
philip@zellic.io

2.5. Project Timeline

The key dates of the engagement are detailed below.

February 26, 2024	Start of primary review period
March 21, 2024	End of primary review period

3. Detailed Findings

3.1. Claim spoofing via lack of SNI/host validation

Target	witness-sdk		
Category	Coding Mistakes	**Severity**	Critical
Likelihood	High	**Impact**	Critical

Description

The Reclaim node does not validate that either the server name indication (SNI) extension in the ClientHello message or the `Host` header in the HTTP request match the expected host. Most popular websites use a reverse proxy service such as Cloudflare, Fastly, or Akamai to serve their content. These services use these two values to determine the origin server to route the request to. Thus, for most websites, Reclaim claims can be spoofed by modifying the SNI/`Host` header in the request to be a server that the attacker controls.

To test this vulnerability, we can first modify `createRequest` in `witness-sdk/src/providers/http-provider/index.ts`:

```
const httpReqHeaderStr = [
  reqLine,
- `Host: ${hostPort}`,
+ `Host: fastly.okay.blue`,
  `Content-Length: ${contentLength}`,
```

Then, we modify `packServerNameExtension` in `witness-sdk/node_modules/@reclaimprotocol/tls/lib/utils/client-hello.js`:

```
function packServerNameExtension(host) {
+ host = 'fastly.okay.blue'
    return packExtension({
```

Then, we can use the below code to create a claim that https://www.nytimes.com is serving the content `reclaim sni poc`:

```
import { createClaim } from './src/index'

const claim = await createClaim({
    name: 'http',
    params: {
        url: 'https://www.nytimes.com',
```

```
        method: 'GET',
        responseMatches: [{
            type: 'contains',
            value: 'reclaim sni poc',
        }],
        responseRedactions: [],
    },
    secretParams: {
        cookieStr: 'a'
    },
    ownerPrivateKey: '0x...'
})

console.log(claim)
```

This issue occurs in `witness-sdk/src/providers/http-provider/index.ts`, where only the address that the node connected to is validated:

```
assertValidProviderReceipt(receipt, paramsAny) {
  // ...
  const expHostPort = `${hostname}:${port || DEFAULT_PORT}`
  if(receipt.hostPort !== expHostPort) {
    logTranscript()

    throw new Error(
      `Expected hostPort: ${expHostPort}, found: ${receipt.hostPort}`
    )
  }
```

Impact

This allows an attacker to spoof claims for most websites that use reverse proxy services.

Recommendations

The Reclaim node should validate that the SNI extension in the ClientHello message and the `Host` header in the HTTP request match the expected host.

Remediation

This issue has been acknowledged by CreatorOS Inc, and fixes were implemented in the following commits:

- 37fa5179
- a30a097d

3.2. Claim spoofing via improper node HTTP parsing

Target	witness-sdk		
Category	Coding Mistakes	**Severity**	Critical
Likelihood	High	**Impact**	Critical

Description

Reclaim's custom HTTP parsing is brittle and insufficiently general. This allows for a vulnerability where a crafted request can be interpreted as `Connection: close` by Reclaim but `Connection: keep-alive` by the target server. The Reclaim node will then perform response checks against the combined, pipelined HTTP response, allowing for claim spoofing.

To test this vulnerability, we can modify `witness-sdk/src/providers/http-provider/index.ts` to add the additional request:

```
  'Accept-Encoding: identity',
  ...buildHeaders(pubHeaders),
  ...secHeadersList,
+ 'Connection:keep-alive', // note the lack of space after the colon
+ '\r\n',
+ 'GET /reclaim-http-parse-poc HTTP/1.1',
+ 'Host: soundcloud.com',
+ 'Content-Length: 0',
+ 'Accept-Encoding: identity',
+ 'Connection: close',
  '\r\n',
].join('\r\n')
```

We can then modify `witness-sdk/src/utils/http-parser.ts` to continue reading until the server ends the connection:

```
streamEnded() {
  if(!res.headersComplete) {
    throw new Error('stream ended before headers were complete')
  }

- if(remaining.length) {
- throw new Error('stream ended with remaining data')
- }
-
```

```
- if(remainingBodyBytes > 0) {
- throw new Error('stream ended before all body bytes were received')
- }

  res.complete = true
}

// ...

// take the number of bytes we need to read, or the number of bytes remaining
// and append to the bytes of the body
bytesToCopy = Math.min(remainingBodyBytes, remaining.length)
- remainingBodyBytes -= bytesToCopy
```

Then, we can use the below code to create a claim that https://soundcloud.com is serving the content `reclaim http parse poc`:

```
import { createClaim } from './src/index'

const claim = await createClaim({
  name: 'http',
  params: {
    url: 'https://soundcloud.com/discover',
    method: 'GET',
    responseMatches: [{
      type: 'contains',
      value: 'reclaim http parse poc',
    }],
    responseRedactions: [],
  },
  secretParams: {
    cookieStr: 'a',
  },
  ownerPrivateKey: '0x...'
})

console.log(claim)
```

This issue occurs in `witness-sdk/src/utils/http-parser.ts`:

```
export function makeHttpResponseParser() {
  // ...
  } else if(!res.complete) { // parse the header
    const [key, value] = line.split(': ')
    res.headers[key.toLowerCase()] = value
```

Note that there may be other ways to exploit similar issues to inject headers or body content, depending on server behavior.

Impact

This allows an attacker to spoof claims for most websites that use reverse proxy services. In particular, any website that uses AWS CloudFront is affected.

Recommendations

Reclaim should strictly construct and validate HTTP requests based on an internal (such as JSON) representation of the request, rather than directly validating the flexible textual HTTP format.

Remediation

This issue has been acknowledged by CreatorOS Inc, and a fix was implemented in commit `e6a8f1c5` ↗.

3.3. Lack of TLS `CertificateVerify` validation

Target	tls		
Category	Coding Mistakes	**Severity**	Critical
Likelihood	High	**Impact**	Critical

Description

The Reclaim TLS implementation only validates the server's certificate chain when it receives a `CertificateVerify` record. If the record is never received, the certificate chain is never validated and the connection proceeds normally. In particular, for TLS 1.2 connections where the server only uses its certificate for signing (all modern cipher suites), the `CertificateVerify` record is never sent by the server. This means that the client never validates the server's certificate chain.

To test this vulnerability, we can use the following:

```
import { createClaim } from '@reclaimprotocol/reclaim-node'

const claim = await createClaim({
  name: 'http',
  params: {
    url: 'https://self-signed.badssl.com/', // negotiates TLS 1.2
    method: 'GET',
    responseMatches: [{
      type: 'contains',
      value: '',
    }],
    responseRedactions: [],
  },
  secretParams: {
    cookieStr: 'a'
  },
  ownerPrivateKey: '0x...'
})

console.log(claim)
```

This issue occurs in `tls/src/make-tls-client.ts`:

```
case SUPPORTED_RECORD_TYPE_MAP.CERTIFICATE_VERIFY:
  // ...
```

```
await verifyCertificateChain(certificates, host, rootCAs)
```

Impact

This allows an attacker who can intercept network packets between the node and the origin server to perform a man-in-the-middle (MITM) attack, compromising the confidentiality and integrity of all data transmitted.

Recommendations

The Reclaim TLS client should do the following:

- When using TLS 1.2, validate the certificate chain after it receives a `ServerKeyExchange` message.
- When using TLS 1.3, refuse to proceed with the connection if it does not receive a `CertificateVerify` message.

Remediation

This issue has been acknowledged by CreatorOS Inc, and a fix was implemented in commit `62dada51` ↗.

3.4. Lack of `extractedParameterValues` validation in callback

Target	witness-sdk		
Category	Coding Mistakes	**Severity**	Critical
Likelihood	High	**Impact**	Critical

Description

The Reclaim documentation recommends websites use the values in `extractedParameterValues`. However, the SDK does not validate this data and only validates the claim `claimData.parameters`. A malicious user can modify the `extractedParameterValues` of a valid, signed claim to contain any value they like.

To test this vulnerability, we can use the following:

```
import { Reclaim } from '@reclaimprotocol/js-sdk'

// a valid, signed claim
const claim = {
  identifier: '0x...',
  claimData: {
    provider: 'http',
    parameters: '{...}',
    context: '...',
    ...,
  },
  signatures: [
    '0x...',
  ],
  witnesses: [{
    id: '0x...',
    url: 'https://...',
  }],
  extractedParameterValues: { ... },
}

claim.extractedParameterValues = {
  malicious: 'values',
}

console.log(await Reclaim.verifySignedProof(claim)) // true
```

This issue occurs in `reclaim-sdk/packages/js/src/witness.ts` where the `extractedParameterValues` are not included in the identity computation:

```
export function getIdentifierFromClaimInfo(info: ClaimInfo): ClaimID {
  const str = `${info.provider}\n${info.parameters}\n${info.context || ''}`;
  return ethers.keccak256(strToUint8Array(str)).toLowerCase();
}
```

Impact

This issue allows all claim parameter values to be easily spoofed by malicious users.

Recommendations

We recommend that either

- reclaim-sdk should validate `extractedParameterValues` against `claimData.parameters`, or
- the node should include the extracted parameters in the signed identifier (which is then validated by `verifySignedProof`).

Remediation

This issue has been acknowledged by CreatorOS Inc, and fixes were implemented in the following commits:

- bf2f0d45 ↗
- 0a911146 ↗

3.5. Server-side request forgery via insufficient address validation

Target	tls-receipt-verifier/node		
Category	Coding Mistakes	**Severity**	High
Likelihood	High	**Impact**	High

Description

The function `initializeSession` of an externally opened gRPC server allows the user to establish a socket with any desired server. In particular, `localhost` can be accessed due to an insufficient address check, leading to a server-side request forgery vulnerability.

```
const initialiseSession: RPCPromiseHandler<'initialiseSession'> = async(
    {
        receiptGenerationRequest,
        // ...
    }
) => {
  // ...
    if(receiptGenerationRequest?.host) {
        host = receiptGenerationRequest.host
        port = receiptGenerationRequest.port
        // ...
    }
    // ...
    const { id } = await newSession({
        host,
        port,
        // ...
    })
    // ...
}

export default async function newSession({
    host,
    port,
    // ...
}: NewSessionOpts) {
    // ...
    const socket = await getSocket({ host, port, geoLocation }, logger)
    // ...
}
```

```
async function getSocket(
    {
        host,
        port,
    }
    // ...
) {
    const socket = new Socket()
    // ...
    if(!geoLocation) {
        socket.connect({ host, port, })
        return socket
    }
    // ...
}
```

Impact

Arbitrary socket sends are available, allowing attackers to communicate with internal servers. This allows the following:

- The function `initializeSession` may be called multiple times, allowing an attacker to search for open ports on `localhost` (or an internal network).
- The Reclaim node can initiate a connection to each port and write packets with `PushToSession`.

Recommendations

Access to `localhost` and nonglobal IP addresses should be restricted. Care should be taken to ensure that this validation is also safe against DNS rebinding attacks.

Remediation

CreatorOS Inc provided the following response:

> Reclaim Protocol does not consider this as an issue at the moment since there are no protected internal services.

3.6. Node ReDoS in claim receipt validation

Target	witness-sdk		
Category	Coding Mistakes	**Severity**	High
Likelihood	High	**Impact**	High

Description

Reclaim nodes can validate claim receipts using a regular expression (RegEx) provided by the user. The node does not place a limit on the time or complexity of the RegEx execution. This allows an attacker to perform a ReDoS attack by providing a RegEx and a website that the RegEx is matched on.

To test this vulnerability, we can use the following:

```
import { createClaim } from '@reclaimprotocol/reclaim-node'

const claim = await createClaim({
  name: 'http',
  params: {
    url: 'https://...', // responds with
    aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa!
    method: 'GET',
    responseMatches: [{
      type: 'regex',
      value: '([a-z]+)+$',
    }],
    responseRedactions: [],
  },
  secretParams: {
    cookieStr: 'a'
  },
  ownerPrivateKey: '0x...'
})

console.log(claim)
```

Impact

An attacker can cause the Reclaim node to hang indefinitely, preventing access from any other Reclaim user.

Recommendations

Reclaim should use a RegEx library that limits the time and complexity of the RegEx execution.

Remediation

This issue has been acknowledged by CreatorOS Inc, and a fix was implemented in commit `da111cad`↗.

3.7. Cross-site scripting via `postMessage`

Target	witness-sdk		
Category	Coding Mistakes	**Severity**	Medium
Likelihood	Medium	**Impact**	Medium

Description

The witness SDK receives incoming window messages and insecurely passes them to `JSONPath` without validation. This allows an attacker to execute arbitrary JavaScript code.

This issue occurs in `witness-sdk/src/providers/http-provider/utils.ts` where `JSONPath` evaluates any JavaScript code wrapped in `()`:

```
import { JSONPath } from 'jsonpath-plus'

export function extractJSONValueIndex(json: string, jsonPath: string) {
    const pointers = JSONPath({
        path: jsonPath,
        json: JSON.parse(json),
        // ...
    })
    // ...
}
```

To test this vulnerability, we can use the following:

```
<iframe src="https://sdk-rpc.reclaimprotocol.org" id="frame"></iframe>
<script>
  frame.onload = () => {
    frame.contentWindow.postMessage(JSON.stringify({
      module: 'witness-sdk',
      channel: 'parent',
      id: 1,
      type: 'extractJSONValueIndex',
      request: {
        json: '{"asdf": 1}',
        jsonPath: '("(alert(origin))")'
      }
    }), '*')
  }
```

```
</script>
```

Impact

An attacker can execute arbitrary JavaScript code on the `https://sdk-rpc.reclaimprotocol.org` origin, compromising the confidentiality and integrity of any witness SDK RPC calls and potentially allowing for phishing due to the trusted nature of reclaimprotocol.org.

Recommendations

The witness SDK should use a safe alternative to the jsonpath-plus library. Alternatively, the witness SDK can validate that incoming messages do not contain malicious JavaScript before passing them to `JSONPath`.

Remediation

This issue has been acknowledged by CreatorOS Inc, and a fix was implemented in commit `20e87d54` ↗.

3.8. Address bar not updated after failed navigation

Target	reclaim-app		
Category	Coding Mistakes	**Severity**	Medium
Likelihood	Medium	**Impact**	Medium

Description

After a top-frame navigation fails (such as due to a TLS error), the address bar displayed above the WebView component in the Reclaim app displays the origin of the failed navigation rather than the origin of the currently displayed page.

This issue occurs in `reclaim-app/src/components/DevToolWebview/index.tsx`:

```
const DevToolWebView: React.FC<DevTollWebViewProps> = forwardRef(
  // ...
  return (
    <WebView
    // ...
    onNavigationStateChange={({ url }) => {
      ref.current?.injectJavaScript(INJECTION)
      urlInputUpdate(url)
    }}
    onLoadEnd={({ nativeEvent }) => {
      urlInputUpdate(nativeEvent.url)
```

Impact

Malicious pages may trigger a failed navigation to disguise themselves as a trusted origin. This can convince the user to enter their credentials or reveal other sensitive information to the attacker's website.

Recommendations

The WebView component should properly account for failed navigations and always set the address-bar text to the origin of the displayed page.

Remediation

This issue has been acknowledged by CreatorOS Inc, and fixes were implemented in the following commits:

- 9003b788
- 06b9ce6e

3.9. Callback address bar not updated after navigation

Target	reclaim-app		
Category	Coding Mistakes	**Severity**	Medium
Likelihood	Medium	**Impact**	Medium

Description

The Reclaim app uses a WebView component to display the content of the callback webpage. It displays an address bar above the WebView. But, this address bar does not update whenever the WebView navigates to a new page.

This issue occurs in `reclaim-app/src/screens/Submit.tsx`:

```
const Submit: React.FC<Props> = (props) => {
  // ...
  return (
    // ...
    <ProviderSubheading
    numberOfLines={1}>{template.callbackUrl}</ProviderSubheading>
```

Impact

Untrusted JavaScript running on callback pages (such as in an iframe) may trigger a top frame navigation to a malicious page that disguises itself as the trusted origin. This can convince the user to enter their credentials or reveal other sensitive information to the attacker's website.

Recommendations

The Submit component should update the address bar to correctly reflect the origin of the displayed page.

Remediation

This issue has been acknowledged by CreatorOS Inc, and a fix was implemented in commit `c5b5a7bd` ↗.

3.10. Lack of WebView `postMessage` origin validation

Target	reclaim-app		
Category	Coding Mistakes	**Severity**	Medium
Likelihood	Medium	**Impact**	Medium

Description

The Reclaim app uses a WebView to render the callback and authentication pages. The WebView components listen to message events sent by injected JavaScript running on the embedded web-pages. However, Reclaim does not validate the origin of these messages against an expected value, allowing a malicious cross-origin webpage to impersonate the expected origin.

This issue occurs in `reclaim-app/src/lib/webviewRpc/webviewRpc.tsx`:

```
export function useWitnessWebview() {
    // ...
    <WebView
        // ...
        onMessage={(data) => {
            // ...
            const rpcResult = data.nativeEvent.data
            const parsed = JSON.parse(rpcResult)
```

This issue also occurs in `reclaim-app/src/provider/screens/HttpAuthentication.tsx`:

```
const HttpAuthentication = forwardRef<object, refProps>((props, compRef) => {
    // ...
    <WebView
        // ...
        onMessage={async (event) => {
            const rawMessage = event.nativeEvent.data
            const messageData = JSON.parse(rawMessage)
```

Impact

Malicious websites can prompt the user to sign a message about an account they do not own. This may lead to the user inadvertently connecting their account to the attacker's profile.

Recommendations

The Reclaim app should validate the origin of `postMessages` sent to the WebView components.

Remediation

This issue has been acknowledged by CreatorOS Inc, and a fix was implemented in commit `9148852d` ↗.

3.11. Lack of TLS ALPN validation

Target	witness-sdk		
Category	Coding Mistakes	**Severity**	Medium
Likelihood	Medium	**Impact**	Medium

Description

The Reclaim node does not validate the Application-Layer Protocol Negotiation (ALPN) extension in the server's `ServerHello` message. TLS clients and servers use this field to determine what protocol (for example, HTTP/1.1 or HTTP/2) to use.

This value should be validated in `tls/src/make-tls-client.ts`:

```
async function processRecord(
  // ...
) {
  // ...
  const hello = await parseServerHello(content)
```

Impact

If the Reclaim node does not validate the ALPN field, an attacker may be able to use a protocol other than HTTP/1.1, leading to parsing errors or other vulnerabilities.

Recommendations

The Reclaim node should validate the ALPN field in the `ServerHello` message.

Remediation

This issue has been acknowledged by CreatorOS Inc, and fixes were implemented in the following commits:

- 5682b8e4 ↗
- 45407b89 ↗

3.12. Cross-site scripting via ignored content-disposition header

Target	witness-sdk		
Category	Coding Mistakes	**Severity**	Low
Likelihood	High	**Impact**	Low

Description

The Reclaim app WebView ignores the `content-disposition` HTTP header on responses. Many websites serve potentially untrusted files using the `content-disposition: attachment` header to trigger a download rather than rendering the content in the browser. By ignoring this header, the Reclaim WebView may be vulnerable to cross-site scripting attacks on some websites.

Impact

An attacker can compromise the integrity and confidentiality of user data on any website that serves untrusted file downloads with the `content-disposition: attachment` header.

Recommendations

The Reclaim WebView should fail to navigate to any URL that responds with the `content-disposition: attachment` header.

Remediation

CreatorOS Inc provided the following response:

> Since the severity of the issue is low and due to some limitations, Reclaim Protocol will leave this fix for future work.

3.13. Address bar not updated before page load

Target	witness-sdk		
Category	Coding Mistakes	**Severity**	Low
Likelihood	High	**Impact**	Low

Description

The Reclaim WebView updates the address bar to the new origin while displaying the previous origin's content without any indication that the new page is loading. This may confuse users and lead to phishing attacks.

The address-bar change occurs in `reclaim-app/src/components/DevToolWebview/index.tsx`:

```
const DevToolWebView: React.FC<DevTollWebViewProps> = forwardRef(
  // ...
  return (
    <WebView
    // ...
    onNavigationStateChange={({ url }) => {
      ref.current?.injectJavaScript(INJECTION)
      urlInputUpdate(url)
    }}
```

Impact

An attacker can intentionally slow down the loading of the new page, allowing them to display a phishing page with a legitimate-looking address bar. This may lead users to believe they are visiting a trusted website and choose to enter their credentials or other sensitive information.

Recommendations

The WebView component should indicate when the page is loading with a spinner or similar UI element.

Remediation

This issue has been acknowledged by CreatorOS Inc, and fixes were implemented in the following commits:

- 9003b788
- 06b9ce6e

3.14. Improper TLS certificate parsing

Target	tls/src/utils/x509.ts		
Category	Coding Mistakes	**Severity**	Low
Likelihood	Low	**Impact**	Low

Description

The @peculiar/x509 library used by the Reclaim TLS library can parse some invalid certificate files as valid. This includes certificates with invalid hostnames and expired certificates.

Impact

The Reclaim client can be communicating with a site that has an invalid certificate. Attackers could attempt an MITM attack and compromise the integrity or confidentiality of the TLS connection.

Recommendations

Additional certificate validation is required, including validation of the hostname and expired date.

Remediation

This issue has been acknowledged by CreatorOS Inc, and a fix was implemented in commit `af6e4ced` ↗.

3.15. Improper host validation in QR-code proof request

Target	src/components/QRScanner/index.tsx		
Category	Coding Mistakes	**Severity**	Low
Likelihood	High	**Impact**	Low

Description

Due to the usage of an improper regular expression in processing QR-code data, the app may identify a malicious URL such as https://rclm.link.evil.example.com as the shortened URL of a legitimate proof request.

This issue occurs in `reclaim-app/src/components/QRScanner/index.tsx`:

```
const RECLAIM_SHORTENED_URL_REGEX = /^https:\/\/rclm\.link/

const retrieveDataType = (data: string): RetreivedDataType => {
  if (data.match(RECLAIM_SHARE_REGEX)
    || data.match(RECLAIM_SHORTENED_URL_REGEX)) {
    return 'requested-proofs'
  }
```

Impact

Users can be more susceptible to phishing since the URL looks trusted and the JSON payload is hidden.

Recommendations

The Reclaim app should properly validate the host of the URL by ensuring the link starts with https://rclm.link/.

Remediation

This issue has been acknowledged by CreatorOS Inc, and a fix was implemented in commit 0e0b3568 ↗.

3.16. Production credentials in Git source repositories

Target	witness-sdk		
Category	Coding Mistakes	**Severity**	Low
Likelihood	High	**Impact**	Low

Description

The `witness-sdk/src/providers/irs/irs-address.ts` file was found to contain a credential for a proxy service:

```
'Authorization: Basic VTA...TZa',
```

Impact

If the Reclaim source code is ever publicly released, an attacker may compromise Reclaim's proxy service credentials.

Recommendations

The credential should be rotated, removed from the Git repository, and instead moved into an environment variable.

Remediation

This issue has been acknowledged by CreatorOS Inc, and the credentials have been revoked.

3.17. Lack of sufficient claim-parameter validation

Target	witness-sdk		
Category	Coding Mistakes	**Severity**	Low
Likelihood	High	**Impact**	Low

Description

The witness SDK HTTP provider does not strictly validate the claim parameters in a number of ways. These include the following:

- The scheme of the `url` parameter is not validated and is assumed to be `https`. Thus, a Reclaim node may sign incorrect claims for other schemes, such as `http` or `file`.
- The `responseMatches[*].type` parameter is not validated to be either `contains` or `regex`. Thus, a Reclaim node may sign a claim with an invalid `responseMatches[*].type` value.
- Additional, unknown properties are allowed in parameter signatures. If Reclaim ever adds additional properties, previously signed claims may be incorrect.

Impact

An attacker may receive signed claims that appear to be invalid and have not been properly checked.

Recommendations

Reclaim should strictly validate all claim parameters using a JSON schema or other validation system to ensure that nodes do not sign invalid claims that may become valid in the future.

Remediation

This issue has been acknowledged by CreatorOS Inc, and fixes were implemented in the following commits:

- `f08b723a` ↗
- `bf2f0d45` ↗
- `3e48df70` ↗

3.18. Session check bypass via JavaScript prototype

Target	tls-receipt-verifier/node		
Category	Coding Mistakes	**Severity**	Informational
Likelihood	N/A	**Impact**	Informational

Description

The node session check can be bypassed if the ID is set to a key that exists natively in JavaScript objects, such as `__proto__` or `toString`.

This issue occurs in `tls-receipt-verifier/node/src/sessions/assert-session.ts`:

```
export function getSession(id: string) { // id is user's input
    return storage[id]
}

export function assertSession(id: string) { // id is user's input
    const session = getSession(id)
    if(!session) {
        throw new ServerError(Status.NOT_FOUND, `session "${id}" not found`)
    }

    return session
}
```

Impact

If features like session ID–based login sessions are later implemented, they may be able to be bypassed.

Recommendations

The session ID's format should be validated in `getSession` and `assertSession`. Alternatively, the value may be checked using `hasOwnProperty()`.

Remediation

CreatorOS Inc provided the following response:

The session is an opaque string right now, and will continue to be. So, even if it's overridden – it doesn't compromise anything.

3.19. Unimplemented TLS responses

Target	tls-client		
Category	Code Maturity	**Severity**	Informational
Likelihood	N/A	**Impact**	Informational

Description

The `processRecord` implementation for the `EncryptedExtensions` and `HelloRetryRequest` records are unimplemented.

Impact

Potential communication disruptions may occur during the TLS handshake. This could lead to a connection establishment failure by incompability upon servers that expect a fully implemented TLS handshake.

Recommendations

Processing of the records should be implemented according to the published RFC specifications.

Remediation

This issue has been acknowledged by CreatorOS Inc, and a fix was implemented in commit `a2a91726` ↗.

4. Discussion

The purpose of this section is to document miscellaneous observations that we made during the assessment. These discussion notes are not necessarily security related and do not convey that we are suggesting a code change.

4.1. Spoofing risk via website data format instability

Reclaim parses data from websites using RegEx and contains checks to extract data from the HTML structure. This is inherently unstable as the HTML structure of websites can change over time due to design updates. After these changes, the Reclaim data provider may not function properly and may allow for spoofing. In fact, the majority of published data providers tested from the Reclaim website were outdated and did not function properly.

4.2. Spoofing risk via insecure contains/RegEx checks

An additional risk comes from the limited set of assertions supported by the HTTP provider. It currently only supports checking whether a given string is contained in the response or for a RegEx match. This is insufficient to parse HTML data. This may be exploited by an attacker if they are able to partially control some data displayed on the page that Reclaim fetches. For example, a Reclaim data provider for the user's username may be manipulated if the user is able to set their bio, which is also displayed on the same page.

The Reclaim HTTP provider should be updated to support more complex assertions, such as JSON parsing or CSS/XPath selectors.

4.3. Sensitive data exposure risk

The app debug log file contains sensitive credentials/user data, but the app does not notify the user that the log file contains sensitive information when prompted to share it. This increases the likelihood that users will fall victim to social engineering.

The following code is from `reclaim-app/src/screens/Https.tsx`:

```
const payload = claim.requestedClaim.payload

const createAndShareFile = async () => {
    const fileName = 'logs.txt'
    const filePath = `${RNFS.DocumentDirectoryPath}/${fileName}`
    dispatch(addLog(JSON.stringify({ ...payload, type: 'payload' })))
    // [...]
```

```
    const shareOptions = {
        title: 'Share via',
        subject: 'Debug Logs',
        url: `file://${filePath}`,
        type: 'text/plain',
    }
    await Share.open(shareOptions)
}
```

4.4. User-phishing risk

When users generate a proof of identity or reputation to share with third-party websites, they are required to log in to the website that holds the credential. Attackers can easily steal information by convincing users to generate proofs on elaborately designed fake websites through social engineering.

To mitigate the weakness to phishing in such a structure, it is necessary to implement basic phishing prevention and user-warning measures. These could include

- checking website hosts against the default providers offered by the Reclaim protocol, and
- querying the host on phishing-website databases such as OpenPhish.

4.5. Lack of witness-sdk `postMessage` origin validation

The Reclaim web witness SDK does not validate the origin of incoming `postMessage` events. Thus, anyone can use the witness-sdk in any way to trigger proof generation, which can be a risk if there is a bug in the `postMessage` event.

In fact, this led to a vulnerability, discussed in Finding 3.7. ↗.

4.6. Proxy/scraping service usage

The currently unused IRS provider located at `witness-sdk/src/providers/irs` uses a scraping service, scrape.smartproxy.com, to fetch the user's information from the IRS website. This means that users are trusting a third-party service to fetch their information, which is a potential privacy concern. Additionally, because the TLS connection is terminated at the scrape.smartproxy.com server, Reclaim is only making a claim regarding the response from the scraping service, not the IRS website itself.

4.7. Node centralization risk

Reclaim nodes are currently only operated by CreatorOS Inc. Thus, CreatorOS Inc has full control over the network and can sign invalid claims. This is a significant risk to the network, as it is not decentralized and is not resistant to manipulation.

5. Assessment Results

At the time of our assessment, the reviewed code was deployed.

During our assessment on the scoped Reclaim Protocol files, we discovered 19 findings. Four critical issues were found. Two were of high impact, five were of medium impact, six were of low impact, and the remaining findings were informational in nature.

5.1. Disclaimer

This assessment does not provide any warranties about finding all possible issues within its scope; in other words, the evaluation results do not guarantee the absence of any subsequent issues. Zellic, of course, also cannot make guarantees about any code added to the project after the version reviewed during our assessment. Furthermore, because a single assessment can never be considered comprehensive, we always recommend multiple independent assessments paired with a bug bounty program.

For each finding, Zellic provides a recommended solution. All code samples in these recommendations are intended to convey how an issue may be resolved (i.e., the idea), but they may not be tested or functional code. These recommendations are not exhaustive, and we encourage our partners to consider them as a starting point for further discussion. We are happy to provide additional guidance and advice as needed.

Finally, the contents of this assessment report are for informational purposes only; do not construe any information in this report as legal, tax, investment, or financial advice. Nothing contained in this report constitutes a solicitation or endorsement of a project by Zellic.