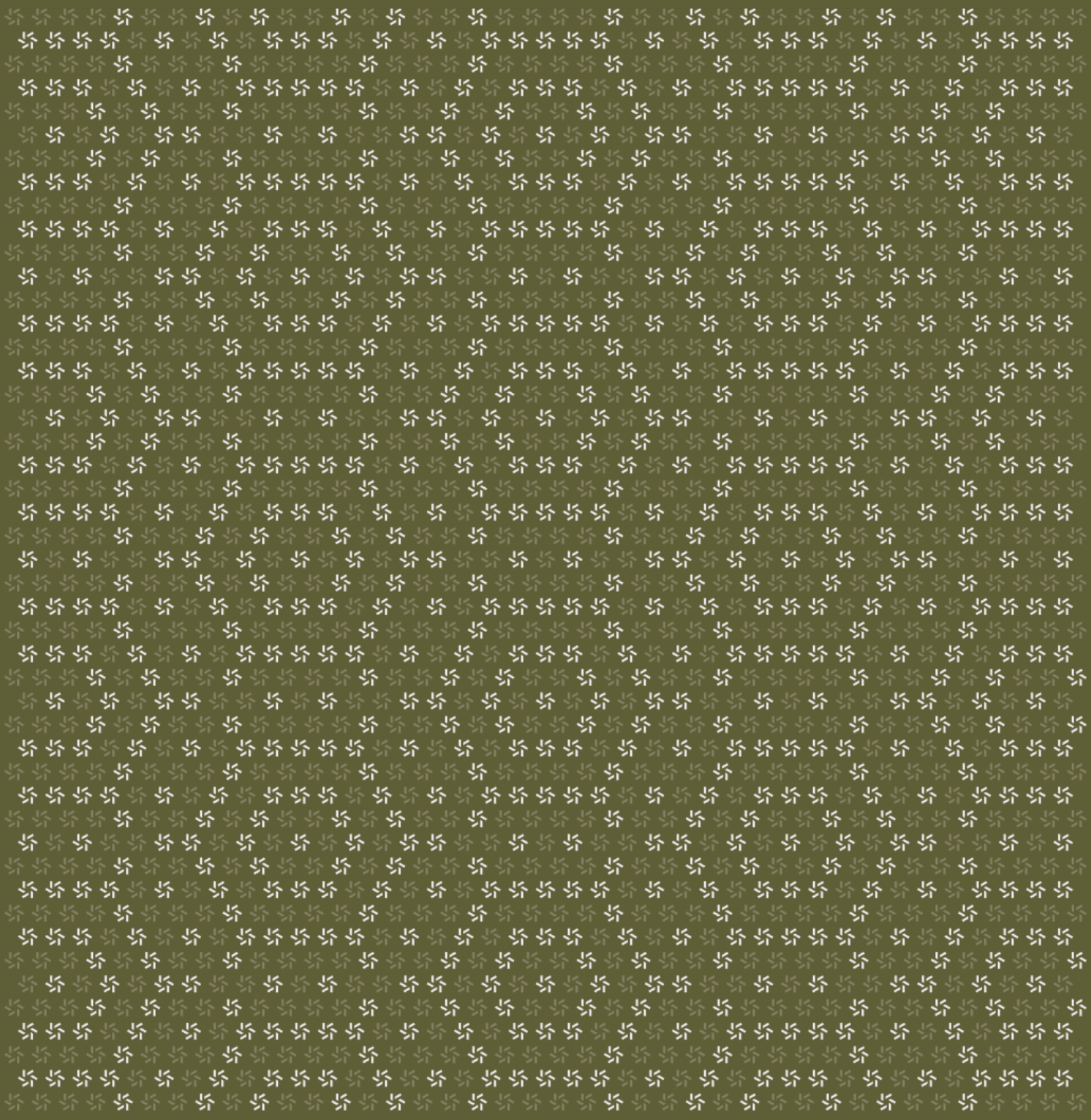
Zellic

March 8, 2024

Prepared for
Avara Labs Ltd.

Prepared by
Jasraj Bedi
Jaeeu Kim
SeungHyeon Kim
Maik Robert
Zellic

Pocket
Web Wallet 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 Avara Labs Ltd. from February 27th to March 6th, 2024. During this engagement, Zellic reviewed Pocket'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 newly implemented pin and passkey support implemented safely?
- Is the SSO pop-up safe from malicious websites attempting to steal session information?
- Do the signing approval modals effectively block potentially unwanted transactions from completing without additional user approval?
- Is the 2FA recovery support secure against unauthorized access?
- Is HKDF a safe design decision over scrypt for auth token hashes?
- Are the cryptographic changes to recovery flow implemented correctly?
- Is the CSP implementation used for the EVM RPC proxy safe?
- Is the usage of the WebCrypto API correct?

1.3. Non-goals and Limitations

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

- Production-deployment infrastructure
- Secrets management in production

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

1.4. Results

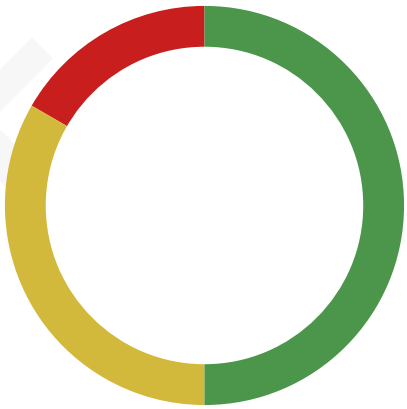
During our assessment on the scoped Pocket code, we discovered six findings. One critical issue was found. Two were of medium impact and three were of low impact.

Additionally, Zellic recorded its notes and observations from the assessment for Avara Labs Ltd.'s

benefit in the Discussion section (4. ↗) at the end of the document.

Breakdown of Finding Impacts

Impact Level	Count
Critical	1
High	0
Medium	2
Low	3
Informational	0



2. Introduction

2.1. About Pocket

Pocket is a noncustodial burner wallet that allows creating a wallet with a username and password but in a secure, encrypted way with a focus on improving the UX of the Ethereum experience.

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:

Basic coding mistakes. Many critical vulnerabilities in the past have been caused by simple, surface-level mistakes that could have easily been caught ahead of time by code review. Depending on the engagement, we may also employ sophisticated analyzers such as model checkers, theorem provers, fuzzers, and so on as necessary. We also perform a cursory review of the code to familiarize ourselves with the code.

Business logic errors. Business logic is the heart of any smart contract application. We examine the specifications and designs for inconsistencies, flaws, and weaknesses that create opportunities for abuse. For example, these include problems like unrealistic tokenomics or dangerous arbitrage opportunities. To the best of our abilities, time permitting, we also review the contract logic to ensure that the code implements the expected functionality as specified in the platform's design documents.

Integration risks. Several well-known exploits have not been the result of any bug within the contract itself; rather, they are an unintended consequence of the contract's interaction with the broader DeFi ecosystem. Time permitting, we review external interactions and summarize the associated risks: for example, flash loan attacks, oracle price manipulation, MEV/sandwich attacks, and so on.

Code maturity. We look for potential improvements in the codebase in general. We look for violations of industry best practices and guidelines and code quality standards. We also provide suggestions for possible optimizations, such as gas optimization, upgradability weaknesses, centralization risks, and so on.

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 code 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:

Pocket Code

Repository	https://github.com/pocket-wallet/pocket-monorepo/ ↗
Version	pocket-monorepo: `93d74094b08a926965ef435ca3283e61dd5b8174`
Programs	• *.ts • *.tsx • *.js
Types	TypeScript, JavaScript
Platform	Web

2.4. Project Overview

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

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:

Jasraj Bedi
Co-Founder
jazzy@zellic.io

Jaeeu Kim
Engineer
jaeeu@zellic.io

SeungHyeon Kim
Engineer
seunghyeon@zellic.io

Maik Robert
Engineer
maik@zellic.io

2.5. Project Timeline

The key dates of the engagement are detailed below.

February 27, 2024	Start of primary review period
March 6, 2024	End of primary review period
TBD	Closing call

3. Detailed Findings

3.1. Cross-site scripting via image proxy

Target	apps/pocket-web/app/api/proxy-image/route.ts		
Category	Coding Mistakes	**Severity**	Critical
Likelihood	High	**Impact**	Critical

Description

The nextJS frontend web app has a single nextJS server endpoint. This was used on a temporary basis by the client to proxy images for the UI and lives at `pocker-web/app/api/proxy-images`. A URL can be passed to this endpoint via the `url` `GET` parameter, which will make the server initiate an HTTP request to the given URL and display the response.

```
export async function GET(request: Request) {
  try {
    const { searchParams } = new URL(request?.url);
    const url = searchParams.get('url');

    if (!url) throw new Error('No url provided');

    const res = await fetch(url);
    const data = await res.arrayBuffer();

    return new Response(data, {
      status: 200,
      headers: res.headers,
    });
  } catch (err) {
    return Response.json({
      error: 'invalid query',
    });
  }
}
```

This endpoint reflects the response from the server-initiated request without a `Content-Security-Policy` in the response headers. This behavior creates the conditions for a cross-site scripting vulnerability.

Impact

In scenarios where a user that has logged into Pocket in the last 7 days navigates to a domain which an attacker controls, the attacker can leverage the cross-site scripting vulnerability to access sensitive information in the `localStorage` which could be used to decrypt a user's mnemonic key.

Recommendations

Unless the proxying functionality is critical to the functioning of the application, it would be advised to disable the endpoint entirely.

Remediation

It was noted that the client had planned to remove the endpoint in question prior to any official launch, as the UI is still in the development phase. Furthermore, to prevent future CSP bypasses, the client has prohibited the addition of any nextJS same-domain server endpoints. Lastly, the CSP was additionally reviewed and determined to provide a significant mitigation to a majority of cross-site scripting vulnerabilities.

This issue has been acknowledged by Avara Labs Ltd., and a fix was implemented in commit `96e52542` ↗.

3.2. Insecure random-number generation for validation codes

Target	/packages/custom-types/src/validation-code.type.ts		
Category	Coding Mistakes	**Severity**	Medium
Likelihood	Medium	**Impact**	Medium

Description

It was discovered that the validation codes, which are used in account validation steps, are generated in an insecure manner.

```
public static generate() {
    return new ValidationCode(Math.floor(100000 + Math.random() * 900000));
}
```

The pattern `Math.floor(CONSTANT * Math.random())` leaks a lot of information that can be used to reverse engineer the internal state of `Math.random(..)`, which allows an attacker to predict the future values of `Math.random()` remotely. An attacker only requires a few sequentially generated validation codes, which can trivially be obtained by repeatedly taking an action that would result in the attacker receiving a validation code per email or SMS. These tokens can then be used to reverse engineer the internal `Math.random` seed values using a tool such as `v8-rand-buster`.

Impact

It is possible to accurately predict all future validation codes, which allows an attacker to bypass the account validation step.

Recommendations

Replace the existing implementation of the validation-code generation with a cryptographically secure pseudorandom number generator that cannot be predicted by an attacker.

Remediation

This issue has been acknowledged by Avara Labs Ltd., and a fix was implemented in commit `24e8ba86` ↗.

3.3. Stored cross-site scripting in image upload

Target	apps/server/src/services/file/file.service.ts		
Category	Coding Mistakes	**Severity**	Medium
Likelihood	High	**Impact**	Medium

Description

The wallet-picture functionality allows users to upload an image for their wallet. This functionality is implemented by generating a presigned file-upload URL that the client then uploads a file to, found in the `generatePresignedUploadUrl` function. This presigned URL points to an S3 bucket that the contents of are accessible on a subdomain, usually `statics.<root>` of the root domain.

While the generated URL is meant for image upload, it enforces no restrictions on the mime type of the uploaded file. Because of this missing restriction, a malicious party can upload an HTML file, which is then accessible on the `statics` subdomain.

Impact

The primary impact of this vulnerability is likely phishing. Password managers, such as 1Password, will also suggest credentials on subdomains, which would make for a particularly convincing phishing page.

Recommendations

Add a restriction on the content type for the presigned URL during generation. This will prevent clients from uploading, and ultimately serving, files with disallowed content types.

Remediation

This issue has been acknowledged by Avara Labs Ltd., and a fix was implemented in commit `90453fc1` ↗.

3.4. Rate-limit bypass

Target	apps/server/src/middleware/get-ip-address-hash.ts		
Category	Coding Mistakes	**Severity**	Low
Likelihood	High	**Impact**	Low

Description

Rate limiting within the application concerns itself with two parts: per-user lockout and per–IP address lockout. The purpose of the per-user lockout is evident — it is to ensure that attackers are not attempting to break into a specific user's account. The per–IP address lockout is to prevent an attacker from trying to hit multiple accounts in a larger campaign where the success of breaking into any individual account is minimal, but across enough accounts, the expected success goes way up.

We have three distinct findings related to how IP-based lockout works.

1. The `X-Forwarded-For` header is inaccurately assumed to be an array.

This might appear to be a small nit, but it has serious security implications on how this feature works. The logic assumes that if multiple values are present, it will be available as an array. This is incorrect in all circumstances. If multiple IP addresses are presented in a comma-delimited list, they are returned as a single string. Additionally, despite this being a pattern in other frameworks, multiple instances of a given header are treated as one single, command-delimited list of IP addresses.

```
const ip = req.headers['X-Forwarded-For']
          || req.headers['x-forwarded-for']
          || req.socket.remoteAddress;

invariant(ip, 'Not possible to resolve and IP address');

if (Array.isArray(ip)) {
  return IpAddressHash.fromRawIpAddress(ip[0]);
}
return IpAddressHash.fromRawIpAddress(ip);
```

The impact of this is that the application logic will treat the entire string (e.g., "`1.1.1.1,2.2.2.2`") as an IP address. This means that an attacker merely needs to add an `X-Forwarded-For` (XFF) header and put any value they so desired. This value is then hashed and used as the IP address hash for IP-based lockout. This provides an attacker a simple way of getting around this lockout.

2. Incorrect IP address is fetched from the `X-Forwarded-For` header

Assuming that the previous issue of the header's value not being treated as an array is addressed, there still exists an issue with the remaining logic.

The logic will check the XFF header or remote address from the socket. This is needed as the service will be deployed behind a reverse proxy, which typically needs to forward the real client IP address along in the XFF header. The logic assumes that if multiple values are present, either through multiple XFF headers or from multiple values in the XFF header, the first entry `ip[0]` is the address that should be used. This is incorrect.

CloudFront, which fronts the service, will append the real client IP address to the XFF header if an XFF header is already present. This means that the correct IP address is the last entry. That is also not necessarily trustworthy if there are additional reverse proxies in the request path that may additionally manipulate this header. It is all dependent on the organization of the infrastructure.

At minimum, though, the first entry is attacker controllable via the XFF header and should not be used: https://docs.aws.amazon.com/AmazonCloudFront/latest/DeveloperGuide/RequestAndResponseBe

3. Failure to consider IPv6 addresses

IPv6 addresses are supported by CloudFront and will be forwarded along. Because the space of IPv6 is so massive (128 bit), typical rate-limiting strategies focus on blocking entire IPv6 blocks. Cloudflare specifically takes the approach of blocking on `/64`, which is a relatively solid standard to apply.

Impact

These three issues allow an attacker to easily bypass the IP-based rate-limiting controls without the added cost of additional IP addresses.

Recommendations

There are distinct fixes for each of the identified issues.

1. On the XFF header, split on the comma (`,`) and trim each value. This will provide the expected array of IP addresses.
2. Use the last IP address sent in the array as that represents the real IP address of the client.
3. Either block IPv6 altogether or apply `/64` block-level rate limiting.

Remediation

This issue has been acknowledged by Avara Labs Ltd., and a fix was implemented in commit `8713a7bc` ↗.

3.5. Path-based server-side request forgery in NFT service

Target	apps/server/src/services/nft/simplehash-client.ts		
Category	Coding Mistakes	**Severity**	Low
Likelihood	Low	**Impact**	Low

Description

The application has an NFT service that will request various metadata about NFTs contained in a user's wallet. It does this by querying a service called SimpleHash. The code builds a SimpleHash API URL using user-controlled data, which is then sent along with an API key.

```
private getNftUrl(
    chain: SupportedChain,
    collectionAddress: EthereumAddress,
    tokenId: string,
  ): string {
    const url = new URL(
      `https://api.simplehash.com/api/v0/nfts/${
        simpleHashChainMap[chain]
      }/${collectionAddress.toHexString()}/${tokenId}`,
    );

    return url.toString();
  }

  async getNft(
    chain: SupportedChain,
    collectionAddress: EthereumAddress,
    tokenId: string,
  ): Promise<NFTUnion> {
    const options = {
      method: 'GET',
      headers: { accept: 'application/json', 'X-API-KEY':
    SIMPLEHASH_API_KEY },
    };

    const resultJSON = await fetch(this.getNftUrl(chain, collectionAddress,
    tokenId), options);

    const result = await resultJSON.json();
```

```
    return mapToNft(result);
  }
```

By using path-traversal characters (like `../`), it is possible to control the path entirely and request arbitrary API endpoints, authenticated with the API key on the `api.simplehash.com` host. This may leak data from the API, which is not intended to be publicly accessible, like webhooks configured on the account.

Impact

User-controlled data is used to build a path that will be requested from the SimpleHash API. Using path-traversal characters, it is possible to fully control the path and request endpoints that may not be accessible otherwise as well as leak data not meant to be public.

Recommendations

Sanitize the path components by rejecting requests containing `..` or `/` characters.

Remediation

This issue has been acknowledged by Avara Labs Ltd., and a fix was implemented in commit 2096089c ↗.

3.6. Using arbitrary `ethereumAddress` as wallet address

Target	apps/server/src/graphql/resolvers/wallet/index.ts		
Category	Coding Mistakes	**Severity**	Low
Likelihood	Low	**Impact**	Low

Description

The function `createNewWallet` is used to add a new wallet to the user. And the function `saveNewWalletIndexDb` is called from `saveNewWalletIndex`, which is called `createNewWallet`; `saveNewWalletIndexDb` is used to store the user's wallet index to the database.

But in this process, there is no logic to check whether the wallet address is owned by the user. This allows users to use addresses they do not actually own.

Impact

Wallets that are not owned by the user may be treated as the user's on the database.

Recommendations

We recommend that the wallet signs the request or challenge generated from the server when calling createNewWallet allowing the server to verify this signature before adding it to the database, this avoids a mismatch of incorrect ethereum addresses in the database.

Remediation

This issue has been acknowledged by Avara Labs Ltd., and a fix was implemented in commit `ee993da9` ↗.

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. Low-entropy pin authentication

The wallet introduced two new authentication methods to replace the old password authentication, these being passkey and pin authentication. While Zellic did not find any technical issues with the implementation of these two, we feel it necessary to mention that moving from the previous default authentication method of passwords to six-digit pins is a regression in overall security. The UI mandates that a user set a six-digit pin code consisting of only numbers, while the backend code actually verifies that the received pin consists of four to 10 digits.

```
public static criteria: PinValidationCriteria = {
    minLength: 4,
    maxLength: 10,
};

public static regex: PinValidationRegex = {
    minLength: new RegExp(`^.{${Pin.criteria.minLength},}$`),
    maxLength: new RegExp(`^.{0,${Pin.criteria.maxLength}}$`),
    digitsOnly: new RegExp(`^[0-9]*$`),
};
```

Should a user choose to bypass the UI restriction, it would technically be possible to set a four digit pin, which decreases the security of the user's wallet even further.

Additionally, while a six-digit pin would still take a considerable amount of time to bruteforce via the web application due to mitigating factors like strict rate limits, the `clientKey`, an important component of the cryptosystem used to secure the `masterKey`, is made significantly weaker. This means if the `encryptedClientInfo` properties were ever stolen from the server-side (due to insider threat actors, future SQLI-style attacks, or other means), users with pincode-based auth are likely at significant risk. We would strongly recommend users examine their personal threat models and take into consideration this risk when deciding on an authentication scheme.

4.2. Usage of the `X-XSS-Protection` header

To secure the RPC proxy, a strict CSP has been implemented in the Next.js configuration file. The configuration also contains security headers that will be set, which includes the `X-Content-Type-Options` header, to prevent MIME-type sniffing, as well as the `X-XSS-Protection` header, which was previously used to prevent certain types of cross-site scripting attacks in the browser.

```
const securityHeaders = [
  {
    key: 'X-XSS-Protection',
    value: '1; mode=block',
  },
  {
    key: 'X-Content-Type-Options',
    value: 'nosniff',
  },
];
```

The latter has been deprecated and is no longer used by any relevant browser today. It is generally recommended to discontinue the use of this header to prevent potentially unwanted interactions that could lead to vulnerabilities like cross-site scripting, which it is supposed to prevent. More information can be found in Mozilla's mdn web docs ↗.

This issue has been acknowledged by Avara Labs Ltd., and a fix was implemented in commit 33e57c58 ↗.

4.3. Inconsistent regular-expression usage for data validation

The wallet makes use of regular expressions to verify that user input matches the expected format. Two examples of this are a user's email address and phone number. Different parts of the application are using different regular expressions to validate the same data. Here is an example found in `apps/pocket-web/popups/auth/AuthIdentity.tsx`:

```
const validateEmail = (email: string) => {
  return /^[^\s@]+@[^\s@]+\.[^\s@]+$/.test(email);
};
const validatePhoneNumber = (phoneNumber: string) => {
  // May not be the best regex, but it's a start
  // https://stackoverflow.com/questions/4338267/validate-phone-number-with-
    javascript
    (2017-08-03)
  return /^[+]?[(]?[0-9]{3}[)]?[-\s.]?[0-9]{3}[-\s.]?[0-
    9]{4,6}$/im.test(phoneNumber);
};
```

Here is an example found in `packages/utils/src/validation/validate-email.ts` and `packages/utils/src/validation/validate-phone-number.ts`:

```
export const validateEmail = (email: string) => {
  return /^(([^<>()[\]\\.,;:\s@"]+(\.[^<>()[\]\\.,;:\s@"]+)*)|(".+"))@((\[[0-
    9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}])|(([a-zA-Z\-0-9]+\.)+[a-zA-
    Z]{2,}))$/.test(
    email,
  );
};

export const validatePhoneNumber = (phoneNumber: string) => {
  return /^\+[1-9]\d{1,14}$/.test(phoneNumber);
};
```

This could lead to the same data matching one regular expression and not the other, which could result in unintended application states.

This issue has been acknowledged by Avara Labs Ltd., and a fix was implemented in commit 98a5544f ↗.

5. Assessment Results

At the time of our assessment, the reviewed code was not deployed to production.

During our assessment on the scoped Pocket code, we discovered six findings. One critical issue was found. Two were of medium impact and three were of low impact. Avara Labs Ltd. acknowledged all findings and implemented fixes.

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.