Zellic

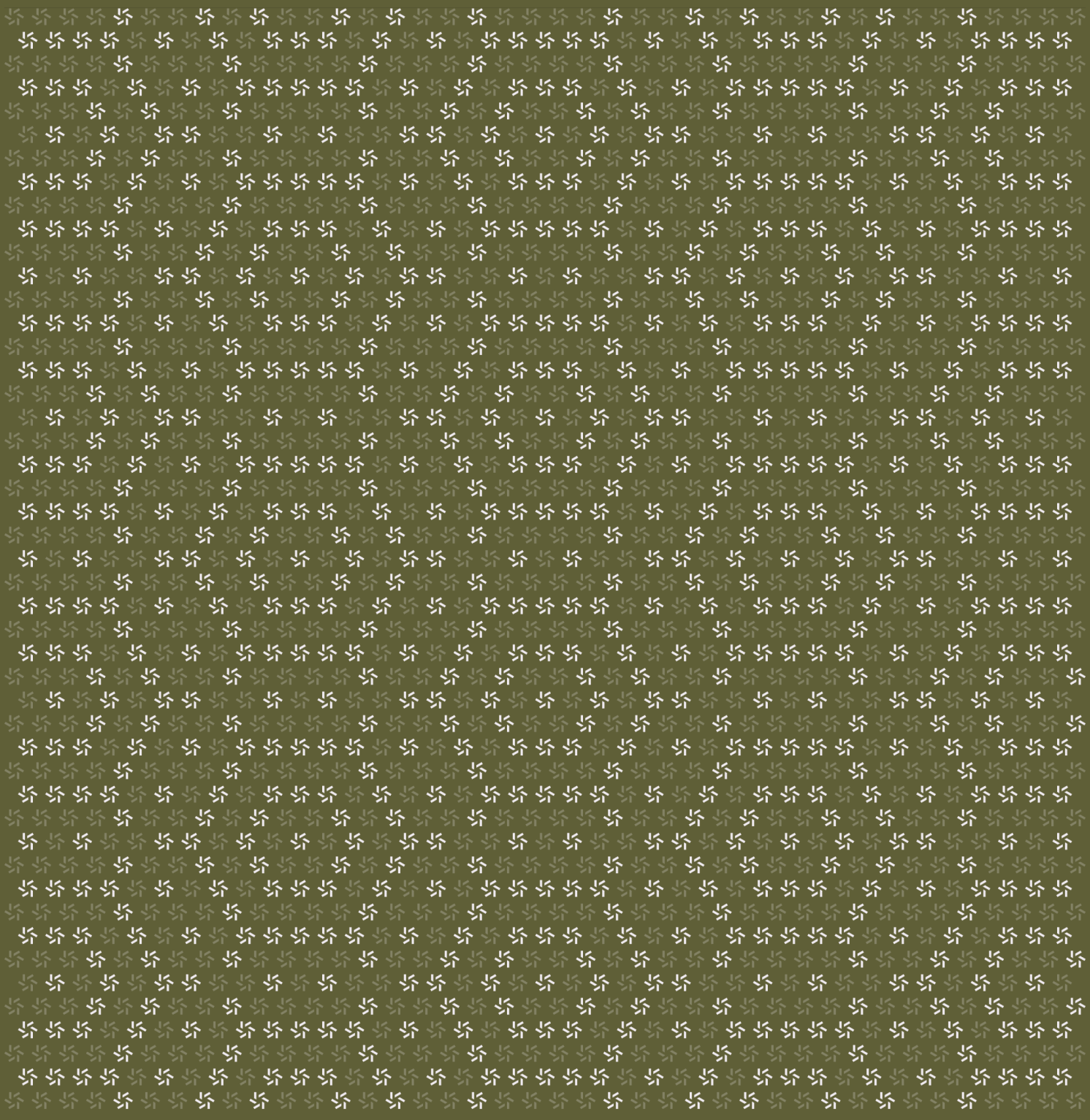
Prepared for
Kristov Atlas
All In Bits Inc.

Prepared by
Jinseo Kim
Seungjun Kim
Zellic

January 21, 2025

AtomOne Staking Portal

Web Application Security Assessment



Contents

About Zellic **3**

1. Overview **3**

1.1. Executive Summary 4

1.2. Goals of the Assessment 4

1.3. Non-goals and Limitations 4

1.4. Results 4

2. Introduction **5**

2.1. About AtomOne Staking Portal 6

2.2. Methodology 6

2.3. Scope 8

2.4. Project Overview 9

2.5. Project Timeline 9

3. Detailed Findings **9**

3.1. Insufficient validator lists 10

3.2. Handling large numbers with `Number` type of JavaScript 12

3.3. Showing user balance with hardcoded precision 14

3.4. Inserting an address into CLI command without validation 16

4. Assessment Results **17**

4.1. Disclaimer 18

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 All In Bits Inc. from January 3rd to January 10th, 2025. During this engagement, Zellic reviewed AtomOne Staking Portal'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:

- Could a malicious attacker launch web-based attacks, such as XSS, CSRF, or clickjacking?
- Does the codebase correctly fetch, process, and provide the information?

1.3. Non-goals and Limitations

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

- Infrastructure relating to the project
- Key custody

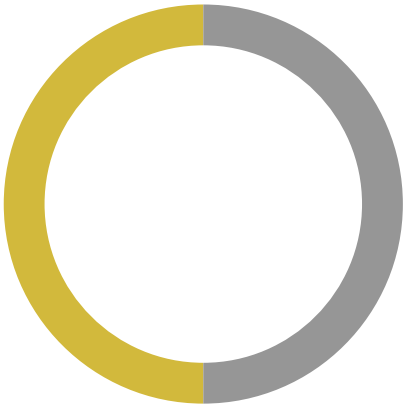
Due to the time-boxed nature of security assessments in general, there are limitations in the coverage an assessment can provide.

1.4. Results

During our assessment on the scoped AtomOne Staking Portal files, we discovered four findings. No critical issues were found. Two findings were of medium impact and the remaining findings were informational in nature.

Breakdown of Finding Impacts

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



2. Introduction

2.1. About AtomOne Staking Portal

All In Bits Inc. contributed the following description of AtomOne Staking Portal:

> The AtomOne Staking Portal is a seamlessly integrated web based interface empowering secure and efficient delegation, re-delegation, un-delegation operations and claiming of rewards on the AtomOne chain.

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 vulnerabilities in web-based applications arise from oversights during development. Missing an authentication check on a single API endpoint can critically impact the security of the overall application. Failure to properly sanitize and encode user input can lead to vulnerabilities like SQL injection, server-side request forgery, and more. We use both automated tools and extensive manual review to identify unsafe code patterns.

Business logic errors. Business logic is the heart of all web applications. We carefully review logic to ensure that the code securely and correctly implements the specified functionality. We also thoroughly examine all specifications for inconsistencies, flaws, and vulnerabilities, including for risks of fraud and abuse.

Integration risks. Web projects frequently have many third-party dependencies and interact with services and libraries that are not under the developer's control. We review the project's external interactions and identify potentially dangerous behavior resulting from compatibility issues and data inconsistencies.

Code maturity. We look for possible improvements across the entire codebase, reviewing violations of industry best practices and code quality standards. We suggest improvements to code clarity, documentation, and testing practices.

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.

2.3. Scope

The engagement involved a review of the following targets:

AtomOne Staking Portal Files

Type	TypeScript
Platform	Web
Target	Changes between c5969a80...@atomone-staking-portal and 2efa134e...@govgen-governance-dapp
Repository	https://github.com/allinbits/atomone-staking-portal ↗
Version	`c5969a80153959a535e6bc61905b1d990421b095`
Programs	`src/*`
Target	Changes between 2efa134e...@govgen-governance-dapp through latest master commit 6b3fd24e...@govgen-governance-dapp
Repository	https://github.com/allinbits/govgen-governance-dapp ↗
Version	`6b3fd24e96a6ce54afa9a6280cee9bc08aa653ff`
Programs	`src/*`

2.4. Project Overview

Zellic was contracted to perform a security assessment for a total of eight person-days. The assessment was conducted by two consultants over the course of six calendar days.

Contact Information

The following project managers were associated with the engagement:

Jacob Goreski
Engagement Manager
jacob@zellic.io ↗

Chad McDonald
Engagement Manager
chad@zellic.io ↗

The following consultants were engaged to conduct the assessment:

Jinseo Kim
Engineer
jinseo@zellic.io ↗

Seungjun Kim
Engineer
seungjun@zellic.io ↗

2.5. Project Timeline

The key dates of the engagement are detailed below.

January 3, 2025	Kick-off call
January 3, 2025	Start of primary review period
January 10, 2025	End of primary review period

3. Detailed Findings

3.1. Insufficient validator lists

Target	HomeView.vue, Redelegate.vue		
Category	Business Logic	**Severity**	Medium
Likelihood	High	**Impact**	Medium

Description

This project provides UI for staking on the AtomOne chain. A user can see a list of the validators on the dashboard, claim the reward, and perform delegate/redelegate/undelegate operations. Each row on the dashboard represents a validator, and there are "Claim", "Delegate", "Redelegate", and "Undelegate" buttons for each row.

We observed that the dashboard shows the truncated addresses of the validators:

```
<template v-for="val in orderedValidators" :key="val.operator_address">
  <span class="text-grey-50 text-100 py-2">{{
    val.description.moniker }}</span>
  <span class="text-grey-50 text-100 py-2">{{ shorten(val.operator_address)
    }}</span>
  <!-- ... -->
</template>
```

```
export function shorten(text: string) {
  if (text.length <= 20) {
    return text;
  } else {
    return text.slice(0, 8) + "..." + text.slice(-8);
  }
}
```

Although it also provides the name of the validator, note that the name of a validator can be arbitrarily chosen by them.

We can also see that the drop-down in the redelegate modal only shows the names of the validators:

```
<span>Redelegate to:</span>
<DropDown
  :values="validatorList.map((x) => x.description.moniker)"
  :model-value="validatorIndex"
  @select="handleValChange"
```

```
/>
```

Impact

An attacker can create a validator that has the same truncated address and name of an existing validator. Because the web UI only provides the truncated address and the name of a validator, it is challenging for users to differentiate between the authentic validator and the fraudulent one, increasing the risk of phishing attacks.

Recommendations

Consider providing the full address of the validators on the dashboard and the drop-down of the validator list.

Remediation

This issue has been acknowledged by All In Bits Inc., and fixes were implemented in the following commits:

- 5742a09c ↗
- b1d2a6d5 ↗

3.2. Handling large numbers with `Number` type of JavaScript

Target	text.ts, Delegate.vue, Redelegate.vue, Undelegate.vue		
Category	Coding Mistakes	**Severity**	Medium
Likelihood	Medium	**Impact**	Medium

Description

The following `formatAmount` function receives the amount with the precision, and it returns the formatted string of the given amount:

```
export function formatAmount(amount: string | number | undefined, precision:
    number) {
  const n = parseInt(amount?.toString() ?? "0") / 10 ** precision;
  return n.toLocaleString(undefined, { maximumFractionDigits: 6 });
}
```

This function first converts the given amount variable into a string then reconverts it into an integer. However, JavaScript uses scientific notation when a number greater than 10^{21} is converted into a string:

```
> (10 ** 21).toString()
'1e+21'
```

Because the `parseInt` function ignores all characters after the first nonnumeric character, the calculation logic of the `formatAmount` silently returns the incorrect value if the given amount is in the `Number` type of JavaScript.

Note that the current codebase does not incorporate the `formatAmount` function with the parameter in the `Number` type. Nonetheless, we recommend to fix the issue since the function signature (incorrectly) explains that this function works with the `Number` type parameter.

Also, we can see that the Delegate, Redelegate, and Undelegate components also handle the large numbers in this way:

```
const delegationOptions: Partial<MsgDelegate> = {
  validatorAddress: props.validatorAddress,
  amount: {
    denom: chainConfig.stakeCurrency.coinMinimalDenom,
    amount: (delegationAmount.value * Math.pow(10,
    delegationDenomDecimals.value))?.toString() ?? "",
```

```
  },
};
```

Impact

The service may show or utilize incorrectly calculated amounts.

Recommendations

Consider handling large numbers as a `String` or `BigInt` type.

Remediation

This issue has been acknowledged by All In Bits Inc., and fixes were implemented in the following commits:

- 3242e641 ↗
- e2209502 ↗

3.3. Showing user balance with hardcoded precision

Target	UserBalance.vue		
Category	Code Maturity	**Severity**	Informational
Likelihood	N/A	**Impact**	Informational

Description

The UserBalance component shows the balance of the current user. We can see this component formats the balance with the function `formatAmount`:

```
<template>
  <span>{{ formatAmount(balance.amount, 6) }}</span>
</template>
```

The precision of the amount is hardcoded into six. However, there is the separate configuration file chain-config.json, which specifies the precision of the token:

```
{
  // ...
  "currencies": [
    {
      "coinDenom": "ATONE",
      "coinMinimalDenom": "uatone",
      "coinDecimals": 6
    }
  ],
  // ...
}
```

Impact

The UserBalance component using a hardcoded precision value instead of the configuration complicates maintenance of the codebase.

Recommendations

Consider using the precision defined in the configuration file.

Remediation

This issue has been acknowledged by All In Bits Inc., and a fix was implemented in commit `5f7fd0b6`.

3.4. Inserting an address into CLI command without validation

Target	commandBuilder.ts		
Category	Business Logic	**Severity**	Informational
Likelihood	N/A	**Impact**	Informational

Description

The project allows users to copy the generated CLI command so they can sign and broadcast the transaction themselves. The following code generates a CLI command:

```
finish() {
  this.command.push("--fees");
  let feeString = "";
  for (let i = 0; i < this.fees.length; i++) {
    feeString = feeString + this.fees[i].amount + this.fees[i].denom;
    if (i + 1 != this.fees.length) {
      feeString = feeString + ",";
    }
  }
  this.command.push(feeString);
  this.command.push("--generate-only");
  this.command.push("--from");
  this.command.push(this.address);
  this.command.push("--chain-id");
  this.command.push(this.chainId);
  this.command.push("--gas");
  this.command.push("auto");
  this.command.push("--sequence");
  this.command.push(this.sequence.toString());
  this.command.push(">");
  this.command.push("tx.unsigned.json");
  return this.command.join(" ");
}
```

Note that the address, which is fetched from the REST server, does not have validation on whether it is a valid address or not.

Impact

If an attacker takes control of the REST server, they can inject a malicious command to the address field in order to execute the malicious command on the computer of a user.

Recommendations

Consider checking if the address provided from the REST server is valid address.

Remediation

This issue has been acknowledged by All In Bits Inc., and a fix was implemented in commit 98996bfa ↗.

4. Assessment Results

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

During our assessment on the scoped AtomOne Staking Portal files, we discovered four findings. No critical issues were found. Two findings were of medium impact and the remaining findings were informational in nature.

4.1. 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.