Zellic

Prepared for

All in Bits

Prepared by

Maik Robert

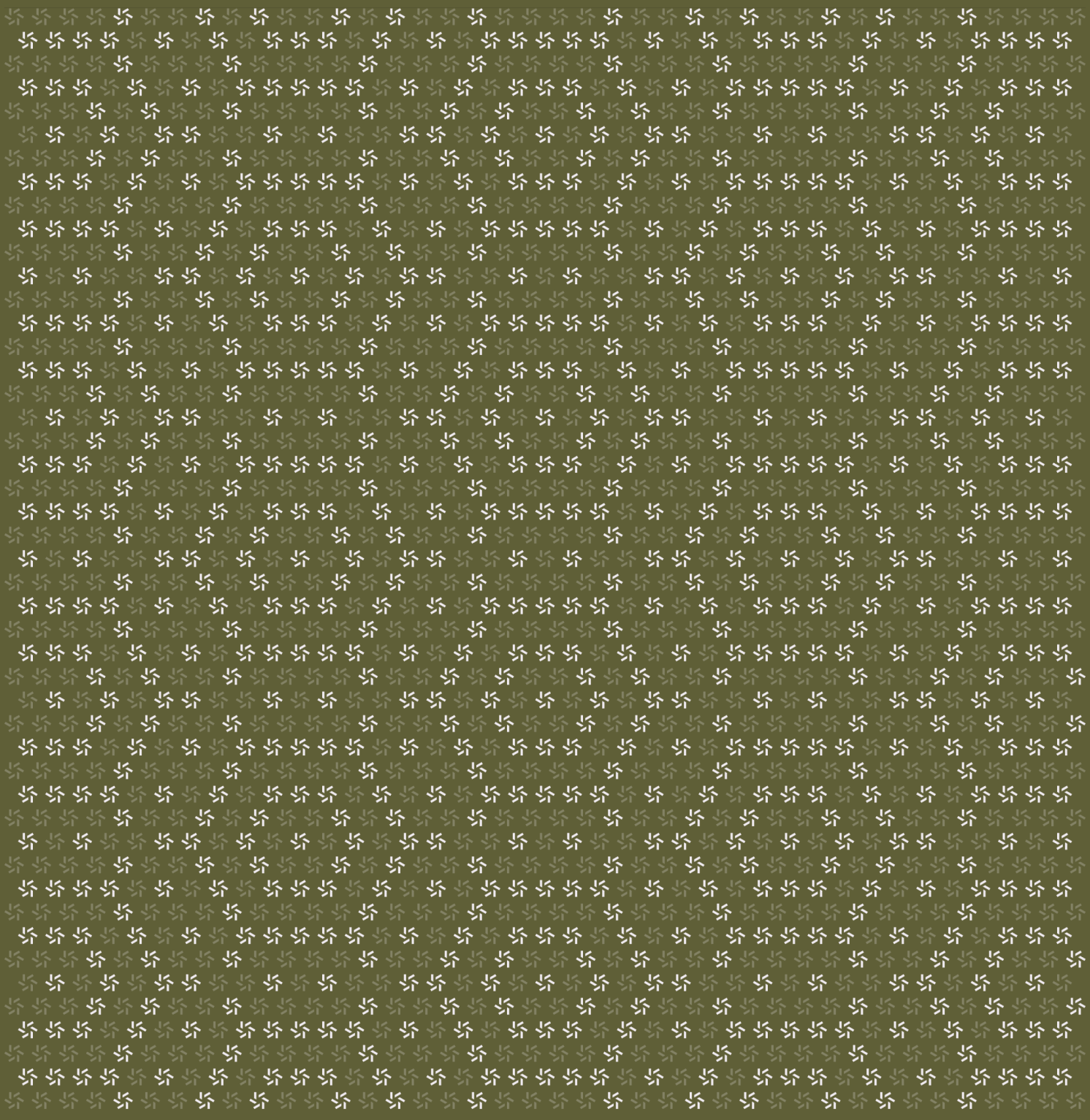
Juchang Lee

Zellic

May 7, 2024

GovGen Governance dApp

Web Application 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 All in Bits from May 1st to May 3rd, 2024. During this engagement, Zellic reviewed GovGen Governance dApp'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:

- Would it be possible to compromise user wallet keys via attacker-controlled discussion messages tricking a user into entering their recovery mnemonic?
- Is it possible to manipulate or misrepresent the status of votes and their contents or otherwise subvert the voting process?

1.3. Non-goals and Limitations

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

- Backend components

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

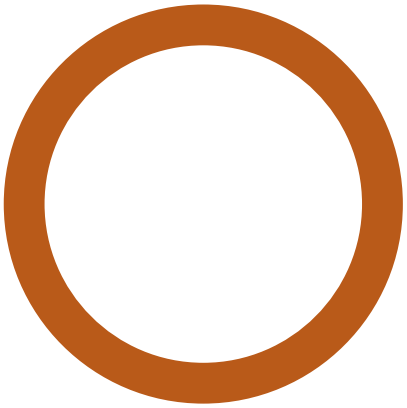
1.4. Results

During our assessment on the scoped GovGen Governance dApp modules, we discovered three findings, all of which were high impact.

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

Breakdown of Finding Impacts

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



2. Introduction

2.1. About GovGen Governance dApp

All in Bits contributed the following description of GovGen Governance dApp:

> GovGen governance dApp is a platform facilitating web-based access to GovGen governance where users can seamlessly engage with proposals, view proposal statistics, participate in discussions as well as perform transactions to vote or deposit on proposals.

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 modules.

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 modules 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:

GovGen Governance dApp Modules

Repository	https://github.com/allinbits/govgen-governance-dapp/ ↗
Version	govgen-governance-dapp: `2efa134e3adab63055d2c5a66766ee1ad67f3c03`
Program	*.ts
Type	TypeScript
Platform	Web

2.4. Project Overview

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

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:

Maik Robert
Engineer
maik@zellic.io ↗

Juchang Lee
Engineer
lee@zellic.io ↗

2.5. Project Timeline

The key dates of the engagement are detailed below.

May 1, 2024	Start of primary review period
May 3, 2024	End of primary review period

3. Detailed Findings

3.1. Cross-site scripting via link-parsing function

Target	src/utility/text.ts		
Category	Coding Mistakes	**Severity**	High
Likelihood	Medium	**Impact**	High

Description

The application makes use of GitHub Discussions via the Discussion API. It does this using a custom module to communicate with the GitHub API and display the discussion comments and links on the web page. Links are parsed from the received comment body with the `getLinks` function.

```
export function getLinks(rawHtml: string): string[] {
  const doc = document.createElement("html");
  doc.innerHTML = rawHtml;
  const links = doc.getElementsByTagName("a");
  const urls: string[] = [];
  for (let i = 0; i < links.length; i++) {
    const link = links[i].getAttribute("href");
    if (!link) {
      continue;
    }

    urls.push(link);
  }

  return urls;
}
```

The raw comment body HTML is passed to the function and inserted into a client-side DOM where `a` tags are referenced and the `href` attribute value is extracted. As no prior sanitization is happening to the comment body, and the comment is coming from a user-controlled and untrusted source, it may lead to cross-site scripting.

Impact

If a malicious comment that contains a cross-site–scripting payload is received from the GitHub API, the payload could execute in the context of the victim's browser session. This could lead to malicious actions being taken on behalf of the victim, for example changing votes or tricking the user into entering their mnemonic phrase.

Recommendations

If possible, use a different way to parse links from the received discussion comment that does not involve evaluating attacker-controlled HTML/JavaScript. Sanitize any attacker-controlled input before evaluating or using it in the dApp. As a further defense against cross-site scripting, a content security policy may be introduced.

Remediation

The `getLinks` function has been reworked to include sanitization by DOMPurify in commit 773ccc7b ↗

3.2. Insufficient DOMPurify sanitization

Target	src/components/common/MarkdownParser.vue, src/components/proposals/GithubComments.vue, src/components/proposals/GithubLinks.vue		
Category	Coding Mistakes	**Severity**	High
Likelihood	Low	**Impact**	High

Description

DOMPurify is used as a sanitization library to prevent cross-site scripting.

```
<div class="text-grey-100 w-full text-300
    font-normal" v-html="DOMPurify.sanitize(comment.body)" />
```

It is used in the default configuration, which strictly prevents cross-site scripting but is otherwise very lax on HTML. By default, DOMPurify does not sanitize or remove `form` elements. This could be abused by an attacker in a discussion comment to post a `form` element prompting potential victims to enter their mnemonic phrase, which would send the phrase to an attacker-controlled server, compromising the victim's wallet.

Impact

The lax nature of the DOMPurify default configuration allows an attacker to use tags and attributes that will not directly lead to cross-site scripting, but can nonetheless be used to steal information from victims via malicious forms using `form` tags.

Recommendations

Further restrict the allowed HTML tags via a custom DOMPurify configuration, or sanitize HTML tags entirely.

Remediation

A blacklist was added for common tags that could lead to data exfiltration, like the `form` tag, in commit cc0257f6 ↗

3.3. Outdated DOMPurify Version

Target	pnpm-lock.yaml		
Category	Coding Mistakes	**Severity**	High
Likelihood	Low	**Impact**	High

Description

One of the app's main lines of defense against cross-site scripting is the use of DOMPurify. DOMPurify sanitizes attacker-controlled or otherwise untrusted HTML from potentially malicious HTML tags or JavaScript. The pnpm-lock.yaml, which is part of the pnpm package manager the application uses, locks the used DOMPurify version to 3.0.9/2.4.3.

```
/dompurify@2.4.3:
    resolution: {integrity: sha512-q6QaLc...moCHZQ==}
    dev: false

/dompurify@3.0.9:
    resolution: {integrity: sha512-uyb4ND...+jhYOQ==}
    dev: false
```

This version is outdated and contains multiple publicly known bypasses for the sanitizer, leading to cross-site scripting.

Impact

The outdated versions are vulnerable to multiple, publicly known, bypasses of the sanitizer which leads to cross-site scripting.

Recommendations

Update the used DOMPurify version and make sure to always use and update to the latest available version.

Remediation

This was fixed as suggested in commit f3db06b8 ↗

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. Compromising wallet keys via GitHub Discussions

A key focus during this audit was investigating potential vulnerabilities that could compromise user wallets, such as those stemming from social engineering tactics.

A good target for an attacker would be the GitHub Discussions feature, as it would allow an attacker to target many users at once. Discussions are displayed under a proposal and could be commented on by anyone who logs in with a GitHub account. The application receives the comments from the GitHub Discussion API and inserts them into the proposal's page.

DOMPurify was used to sanitize the comment's body, which can contain HTML, to prevent cross-site scripting. However, DOMPurify's default configuration still allows other HTML tags like the `form` tag, which could allow an attacker to trick a victim into entering their mnemonic into the malicious form and sending it to an attacker-controlled server. A fix has been implemented that passes a blacklist of HTML tags, like `form`, to the DOMPurify function to prevent these types of attacks.

Another function was identified that was used to parse links from discussion-comment bodies. It did this by inserting the comment into the client-side DOM of the application and parsing the resulting `a` tags' `href` attribute. An attacker could craft a comment containing a cross-site scripting payload, which would trigger as soon as the function was invoked, as no prior sanitization of the comment's HTML had taken place. This would allow an attacker to take malicious actions on behalf of the victim inside of the dApp or trick a user into revealing their mnemonic via more realistic phishing or social engineering attempts than would be possible with the aforementioned `form` approach.

5. Assessment Results

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

During our assessment on the scoped GovGen Governance dApp modules, we discovered three findings, all of which were high impact.

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.