Zellic

September 16, 2024

Prepared for
Atikh Bana
Issa Hassan
Sergiy Dybskiy
Acctual

Prepared by
Filippo Cremonese
Daniel Lu
Zellic

Acctual Batch Payments

Smart Contract 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 Acctual from September 15th to September 16th, 2024. During this engagement, Zellic reviewed Acctual Batch Payments'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 batch payment's functionality implemented correctly and securely?
- Is the contract securely managing user assets, only allowing authorized payments to be made?
- Is the contract vulnerable to any denial-of-service attack?

The team also requested recommendations for improving gas consumption and suggestions on any important best practices they may not have adopted.

1.3. Non-goals and Limitations

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

- Front-end components
- 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.

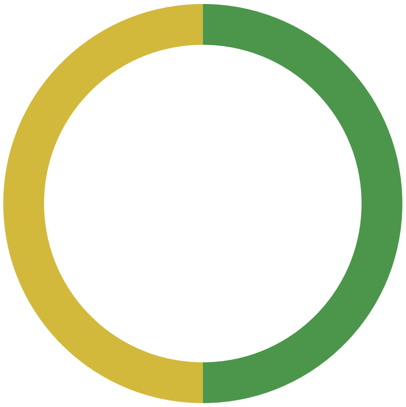
1.4. Results

During our assessment on the scoped Acctual Batch Payments contracts, we discovered two findings. No critical issues were found. One finding was of medium impact and one was of low impact.

Additionally, Zellic recorded its notes and observations from the assessment for the benefit of Acctual in the Discussion section (4. ↗).

Breakdown of Finding Impacts

Impact Level	Count
🟥 Critical	0
🟧 High	0
🟨 Medium	1
🟩 Low	1
⬜ Informational	0



2. Introduction

2.1. About Acctual Batch Payments

Acctual contributed the following description of Acctual Batch Payments:

> We developed a smart contract that allows our users to create batch / multisend transactions of native and ERC-20 tokens to pay their crypto bills.

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 contracts.

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 contracts 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:

Acctual Batch Payments Contracts

Type	Solidity
Platform	EVM-compatible

Target	acctual-smart-contracts
Repository	https://github.com/mango-crypto-accounting/acctual-smart-contracts ↗
Version	`fafc8bdef1b90ca40bd1a00513d7a8bfdd1f53c3`
Programs	`AcctualBatchBillPay`

2.4. Project Overview

Zellic was contracted to perform a security assessment for a total of one person-day. The assessment was conducted by two consultants over the course of two 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:

Filippo Cremonese
Engineer
fcremo@zellic.io ↗

Daniel Lu
Engineer
daniel@zellic.io ↗

2.5. Project Timeline

The key dates of the engagement are detailed below.

September 15, 2024	Start of primary review period
September 16, 2024	End of primary review period

3. Detailed Findings

3.1. Direct usage of `transferFrom`

Target	AcctualBatchBillPay		
Category	Coding Mistakes	Severity	Medium
Likelihood	N/A	Impact	Medium

Description

When processing ERC-20 payments, the `payBills` function calls `transferFrom` directly, requiring the return value of the token to be `true`, indicating success.

Unfortunately, some tokens — including USDT and BNB on some chains — do not strictly adhere to the ERC-20 standard; their `transferFrom` function does not return a value.

Impact

The AcctualBatchBillPay contract has interoperability issues with tokens it may need to operate with, at the time of this writing or in the future.

Recommendations

Use a SafeERC20 library (such as OpenZeppelin) to ensure maximum compatibility.

For reference, this is a list of other ERC-20 tokens with various quirks, including but not limited to missing return values: https://github.com/d-xo/weird-erc20.

Remediation

This issue has been acknowledged by Acctual, and a fix was implemented in commit `3e095382` ↗.

3.2. Unneeded `receive` function

Target	AcctualBatchBillPay		
Category	Coding Mistakes	**Severity**	Low
Likelihood	Low	**Impact**	Low

Description

The contract implements an unneeded, payable `receive` function.

Impact

This function allows to send native ETH to the contract, but it performs no other action. Any ETH sent directly to the contract would be permanently locked; therefore, this represents an unneeded risk to the contract users.

Recommendations

Remove the unneeded `receive` function.

Remediation

This issue has been acknowledged by Acctual, and a fix was implemented in commit `a0818d50` ↗.

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. Usage of events

In the version under review, the contract emits two possible events.

```
event BillsPaid(address indexed payer, uint256 totalPayments);
event Refund(address indexed recipient, uint256 amount);
```

These events are not currently used by any off-chain component. Additionally, the `BillsPaid` event only contains the number of payments performed and the payer, but it does not contain any information about the individual payments made (asset, amount, and recipient).

We also note that the `BillsPaid` event is emitted even if the `payBills` function is invoked with an empty payments array. While this condition is easily recognizable by whoever parses the event, we advise to document this behavior to reduce the risk of a user not considering this edge case.

Remediation

This discussion item was addressed in commit 922d2742 ↗ by emitting an event for each payment that contains the payer, recipient, asset address, and amount.

4.2. Possible gas savings

This section describes some possible ways to reduce the gas usage of the contract, both at deployment and at runtime. We note that the contract does not contain major inefficiencies and all suggestions save a limited amount of gas.

Inheritance from Ownable. The contract inherits from OpenZeppelin Ownable but makes no use of the features implemented by Ownable. This makes the contract use additional bytecode space and storage space. The wasted space is limited, but removing the dependency from Ownable is a trivial way to save a low amount of gas at deployment time.

Reentrancy guard. The contract uses a reentrancy guard; while it is normally recommended to implement reentrancy guards by default on all contracts not explicitly requiring reentrancy, ActtualBatchBillPay does not benefit from reentrancy guards, as the contract has no state or behavior that could be abused. Its complexity is low enough that we can confidently exclude any reentrancy issue. Therefore, the reentrancy guard could be removed to save a small amount of gas at deployment and runtime.

Events. The contract currently emits events that are not used by any off-chain client. These events could be removed to save a small amount of gas.

Remediation

The following commits address this discussion point:

- `9a2e605f` ↗ removes reentrancy guards
- `8aaf8750` ↗ removes inheritance from Ownable
- `40e51fc4` ↗ other miscellaneous gas saving measures

5. Threat Model

This provides a full threat model description for various functions. As time permitted, we analyzed each function in the contracts and created a written threat model for some critical functions. A threat model documents a given function's externally controllable inputs and how an attacker could leverage each input to cause harm.

Not all functions in the audit scope may have been modeled. The absence of a threat model in this section does not necessarily suggest that a function is safe.

5.1. Module: AcctualBatchBillPay.sol

Function: `payBills(Payment[] payments)`

This function can be used to initiate one or more payments. The function supports both native Ethereum as well as ERC-20 tokens.

Inputs

- `payments`
  - **Control**: Arbitrary.
  - **Constraints**: None directly enforced by the contract — user must have sufficient balance and must have granted sufficient allowance for ERC-20 assets.
  - **Impact**: List of payments to be made.

Each entry in the `payments` array is an instance of the `Payment` struct:

```
struct Payment {
    address token; // Use address(0) for native currency (ETH, MATIC, etc.)
    address recipient;
    uint256 amount;
}
```

Branches and code coverage

Intended branches

- Allows to initiate a native ETH payment.
  - ☑ Test coverage
- Allows to initiate an ERC-20 payment.
  - ☑ Test coverage
- Returns any excess ETH sent in the transaction.
  - ☑ Test coverage

Negative behavior

- Reverts if an ERC-20 transfer is unsuccessful.
  - ☑ Negative test
- Reverts if the ETH payment's total is greater than the incoming message value.
  - ☑ Negative test
- Reverts if the call made to refund excess ETH has reverted.
  - □ Negative test

6. Assessment Results

At the time of our assessment, the reviewed code was deployed to the Ethereum Mainnet at address `0xEC41D9b28A020E345dD77bE329699dbF2968B007`.

The updated code which addressed the issues presented in this report was deployed to Ethereum Mainnet address 0xC2ADDa7c829dA5FaFC91AA67784c2e9e11693D97, Arbitrum Mainnet address 0x6B10C1D670a6aF89B9F414EA56E85c9a7DD264fc, Polygon Mainnet address 0x6B10C1D670a6aF89B9F414EA56E85c9a7DD264fc, Base mainnet address 0x6B10C1D670a6aF89B9F414EA56E85c9a7DD264fc, and Optimism Mainnet Address 0xE84EF09967909D942FDAA1Fb7154CBE74347C37D.

During our assessment on the scoped Acctual Batch Payments contracts, we discovered two findings. No critical issues were found. One finding was of medium impact and one was of low impact.

6.1. 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.