Zellic

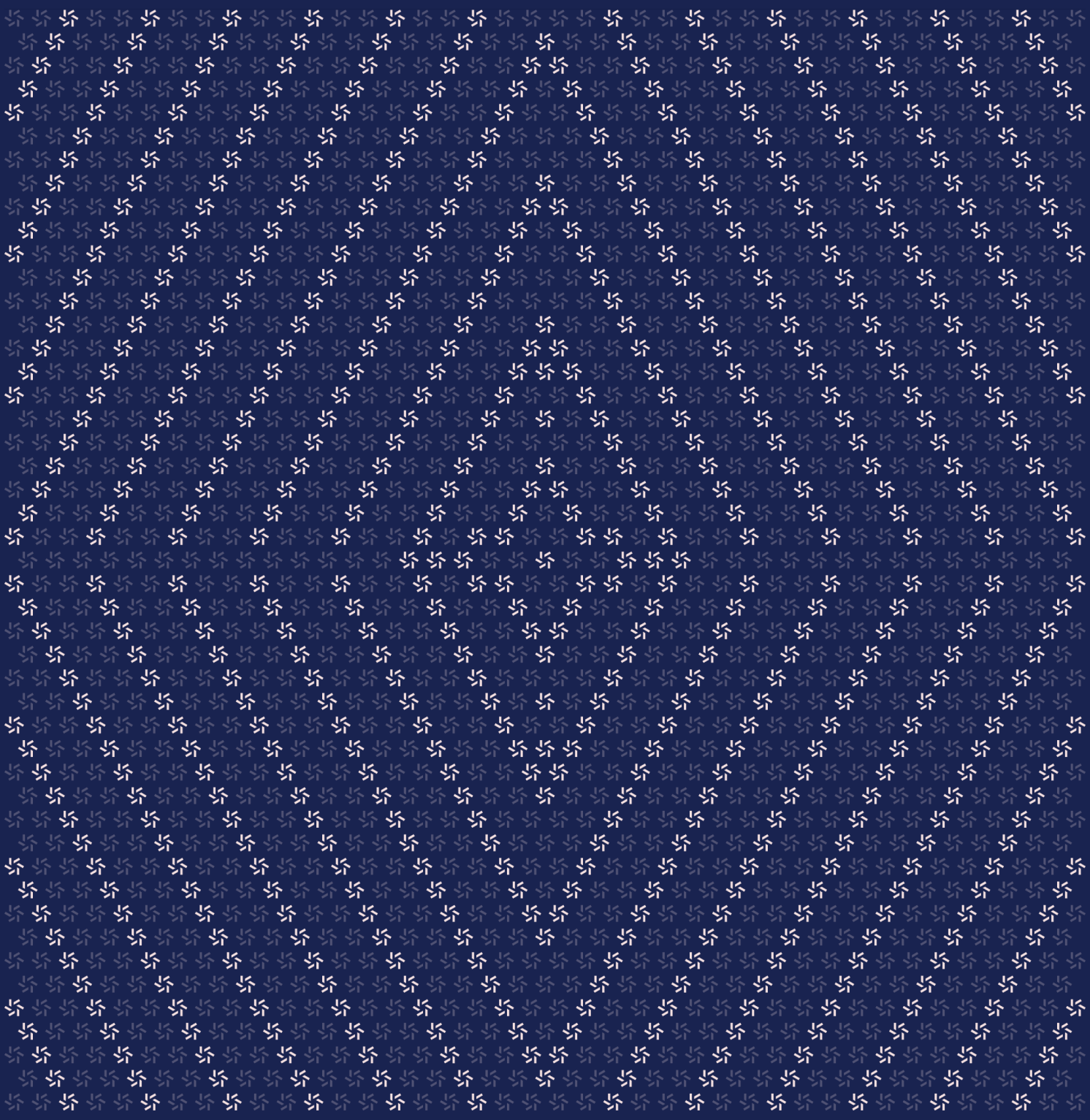
Prepared for
Tom Lehman
Facet Computing, Inc.

Prepared by
Jade Han
Avraham Weinstock
Zellic

September 22, 2024

Facet Geth

Blockchain 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 Facet Computing, Inc. from September 5th to September 9th, 2024. During this engagement, Zellic reviewed Facet Geth'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:

- Are there any risks of assets being frozen or leaked on L2 (Facet Protocol) due to the differences from op-geth?
- Could the differences from op-geth cause malicious actions that slow down or halt the Facet chain?

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 Facet Geth modules, we discovered one finding, which was critical.

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

Breakdown of Finding Impacts

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

2. Introduction

2.1. About Facet Geth

Facet Computing, Inc. contributed the following description of Facet Geth:

> Facet Protocol is an EVM-compatible rollup that offers a novel approach to scaling Ethereum without introducing new dependencies or trust assumptions. As a fork of Optimism's OP Stack, the framework behind many of the largest Layer 2 rollups, Facet differentiates itself by eliminating all sources of centralization and privilege, resulting in the first rollup that preserves Ethereum's liveness, censorship resistance, and credible neutrality.
>
> Facet geth, the focus of this audit, is a forked version of OP Stack's go-ethereum (geth), specifically adapted to process Facet transactions. It is responsible for executing Facet state transitions in a deterministic manner, ensuring that Facet nodes can derive and maintain consistent state from Ethereum's transaction history.

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:

Facet Geth Modules

Type	Go
Platform	EVM-compatible
Target	facet-geth
Repository	https://github.com/0xFacet/facet-geth ↗
Version	`06b5f6e039ccd79c34c3fb01333d3ccbc8b952db`
Programs	`Geth`

2.4. Project Overview

Zellic was contracted to perform a security assessment for a total of 0.9 person-weeks. The assessment was conducted by two consultants 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:

Jade Han
Engineer
hojung@zellic.io ↗

Avraham Weinstock
Engineer
avi@zellic.io ↗

2.5. Project Timeline

The key dates of the engagement are detailed below.

September 5, 2024	Start of primary review period
September 6, 2024	Kick-off call
September 9, 2024	End of primary review period

3. Detailed Findings

3.1. DOS in precompiled contract due to nil dereference

Target	facet/core/vm/contracts.go		
Category	Coding Mistakes	**Severity**	Critical
Likelihood	High	**Impact**	Critical

Description

A panic issue has been identified in the newly added precompiled contract. The following function, `Run`, lacks input validation, leading to a nil dereference when excessively large numbers are passed as input:

```
func (c *sqrtWithFloatPrecision) Run(input []byte) ([]byte, error) {
    n := new(big.Int).SetBytes(input)
    nFloat64, _ := new(big.Float).SetInt(n).Float64()
    sqrtFloat64 := stdmath.Sqrt(nFloat64)
    sqrtStr := fmt.Sprintf("%.0f", stdmath.Floor(sqrtFloat64))
    sqrtInt, _ := new(big.Int).SetString(sqrtStr, 10)

    return common.LeftPadBytes(sqrtInt.Bytes(), 32), nil // panic occurs here!
}
```

Specifically, the function does not handle input values that convert to a floating-point infinity (e.g., values exceeding 10^{309}), causing a nil dereference and resulting in a panic.

Impact

If a number larger than 10^{309} is passed as input to the precompiled contract, it triggers a nil dereference, leading to a panic during the execution of the transaction. While the `eth_call` RPC handler recovers from the panic, during actual transaction execution, there is no recovery function in place, causing the facet-geth process to forcibly terminate.

Recommendations

To prevent this issue, it is recommended to add input validation that enforces a limit ensuring that the input value is less than 10^{309}.

Remediation

This issue has been acknowledged by Facet Computing, Inc., and was fixed with a length check in commit `bf0753a1` ↗ before the precompile was removed in commit `c82cc6e6` ↗.

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. Address-generation method in create-opcode handler

Zellic's discussion with Facet during the audit raised concerns about the difference in address generation using the create-opcode handler on the Facet chain compared to other EVM chains. The concern focused on user experience, specifically regarding potential confusion when users mistakenly send assets to the wrong address. On most EVM chains, the address-generation method is uniform, which could allow for the recovery of missent assets. However, on Facet, the modified address generation could make such recovery more difficult.

Facet's approach to address generation was modified to prioritize security by ensuring that L1 and L2 contracts do not share the same address. This prevents potential security risks such as packet replay or ownership control issues, where having identical addresses across L1 and L2 could allow unintended contract control or transaction replay. By ensuring that different addresses are generated for contracts across layers, Facet mitigates the risk of cross-layer exploits.

While this approach enhances security, it does introduce some trade-offs in terms of user convenience. For example, deterministic address generation, which could help recover missent assets across chains, becomes more difficult. However, the Facet team has carefully evaluated this trade-off and determined that the benefits of preventing cross-layer attacks outweigh the potential inconvenience of nondeterministic addresses in certain edge cases.

In conclusion, while the concern raised focused on user experience, particularly in terms of address consistency and asset recovery, Facet's approach emphasizes security. Their decision to modify the address-generation method helps prevent potential risks associated with shared addresses across L1 and L2, ensuring safer and more reliable contract interactions. Though it may introduce some inconveniences for users, the enhanced security benefits are a key consideration in the overall design.

Facet Response:

This issue has been acknowledged by Facet Computing, Inc., and a change was implemented to adopt Ethereum's standard CREATE address-generation method. This requires aliasing L1 contract addresses in Facet calls and Facet follows Optimism's approach here.

4.2. Transaction count limits to mitigate spam in Facet chain

A potential issue regarding spam attacks on the Facet chain was raised, focused on the ability to create multiple Facet transactions within a single L1 transaction. The concern was that this could lead to a large volume of minimal transactions being processed in a single block, potentially overwhelming the network and causing disruptions.

After reviewing the situation, we recognized that while L1 calldata costs make this setup less problematic than initially thought, the risk of spam still exists. To address this, we suggested implementing a transaction count limit on the number of Facet transactions that can be included in a single L1 transaction. This was presented as a simple, effective way to mitigate the potential for network overload due to spam.

The Facet team acknowledged the concern, noting that contract-initiated transactions using the `call()` function have minimal base-cost overhead, which could introduce a denial-of-service (DOS) risk. While they considered requiring extra "junk" data in transactions to raise the base cost, they preferred to avoid introducing unnecessary mechanisms.

The team emphasized that the existing L1 and L2 gas systems, along with the L2 gas limit, already impose costs on attackers, making sustained spam attacks less likely. They highlighted the importance of ensuring that users can always determine what happened to their initiated transactions without inadvertently blocking legitimate ones, as doing so could introduce new DOS issues.

We agreed that adding more complex mechanisms could increase system vulnerabilities and reduce the user experience. However, we believe that a simple transaction count limit would not introduce significant complexity and could be an effective way to prevent spam.

We believe that the concerns mentioned are unlikely to occur in the future because attackers would have to spend significant amounts of money without gaining any tangible benefits. The reason for discussing this is to aim for a better system design.

> Facet Response:
>
> Facet has implemented a restriction limiting users to a single Facet transaction per Ethereum transaction. This approach directly addresses the concern raised about potential spam attacks due to multiple Facet transactions within a single L1 transaction. By enforcing a one-to-one ratio between Ethereum and Facet transactions, we mitigate the risk of spam-induced network congestion without introducing complex mechanisms or unnecessary overhead.

5. Threat Model

This provides a full threat model description for various functions. As time permitted, we analyzed each function in the modules 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. Differences between Optimism and Facet Protocol

The Facet Protocol and Optimism share the goal of creating Layer-2 scaling solutions, but they approach this in distinct ways, especially in their transaction processing, gas mechanisms, and architectural design. Below are the key differences between the two protocols.

1. Transaction processing (L1 interaction)

Optimism. In Optimism, users can interact with L2 by deploying contracts on Ethereum (L1) and using these contracts to transfer assets or make calls between L1 and L2. Optimism's L2 system relies heavily on sequencers—centralized entities responsible for ordering and batching transactions on L2.

Facet Protocol. Facet removes the reliance on L1 contracts to facilitate L1-to-L2 interactions. Instead of using an L1 contract for interactions, Facet extracts L2 transactions from the data provided in the Ethereum transaction. This allows execution of L2 transactions on the facet-geth. Facet also removes the centralized sequencer that Optimism relies on, resulting in a more decentralized L2 solution. All transactions on Facet (L2) are processed directly through Ethereum (L1).

2. Gas and cost mechanism

Optimism. Like many L2 solutions, Optimism uses an Ether bridge mechanism where L1 Ether is deposited into an L1 contract and L2 Ether is minted in return. This minted L2 Ether is then burned when withdrawing L1 Ether from the contract. Optimism sequencers order transactions and manage gas fees for these L2 transactions, and the system relies on upgrades to the L1 bridge contracts to maintain its functionality, leading to potential centralization risks.

Facet Protocol. Facet's gas mechanism revolves around a unique concept called Facet Compute Token (FCT). Users earn FCT by burning Ether during the creation of Facet transactions. This differs from Optimism, where gas payments are primarily handled through minting and burning L2 Ether. FCT is credited to users before processing their Facet transaction, meaning in most cases, users do not need to worry about paying for gas separately.

3. Native token and security

Optimism. Optimism's L2 Ether is backed by L1 Ether, which is deposited in an L1 bridge contract. The centralized control over the L1 contracts presents security concerns, as the keys controlling the upgradability of these contracts are in the hands of select operators.

Facet Protocol. Facet employs FCT token, a native token that is not bridged from L1 but is instead earned through Ether burning. This eliminates the need for an L1 bridge contract and avoids the risks associated with upgradable L1 contracts.

4. System complexity and design philosophy

Optimism. Optimism uses a relatively more centralized system due to the reliance on sequencers and the upgradable nature of the bridge contracts. While this can offer more control and flexibility in the system, it also introduces risks such as reliance on privileged entities and potential governance issues related to upgrades.

Facet Protocol. Facet's design focuses on simplicity and decentralization. By eliminating sequencers and simplifying the gas mechanism with FCT, Facet reduces the reliance on complex systems and governance overhead.

5. Gas workings on Facet

Optimism. In Optimism, gas fees are paid in bridged L2 Ether, purchased via L1 Ether via the L1 bridge contract. Users are required to maintain a balance of L2 Ether to pay for gas when submitting transactions on the network, which are sent to the `SequencerFeeVault` to pay its operator to post data to the L1.

Facet Protocol. In Facet, users burn L1 Ether to submit Facet transactions as regular Ethereum transactions. Facet mints FCT according to the L1 Ether burn, and then uses a portion of the minted FCT to pay for gas on Facet, crediting the user with the remaining FCT balance. In Facet, users don't need to actively manage their FCT gas balance since FCT minting and FCT gas fee consumption is automatically managed in the background by the protocol.

5.2. Differences in Facet Geth

Facet is based on Optimism, but there are some differences, which result in several distinctions between op-geth and facet-geth. The differences are as follows.

File: core/state_transition.go

Function: `StateTransition.preCheck`

Facet makes several modifications to the `StateTransition.preCheck` function, which checks EVM transactions prior to their execution:

- For deposit transactions, the gas credit is made conditional on the transaction being from the system address (`0xdeaddeaddeaddeaddeaddeaddeaddeaddead0001`).
- If a deposit transaction is both from the system address and separately marked as `IsSystemTx`, this early returns before an `IsOptimismRegolith`; this is not expected to occur in practice due to the `IsSystemTx` field being deprecated.
- For non-deposit transactions that are not from the system address, the transaction's nonce check is skipped.

Function: `StateTransition.TransitionDb`

Facet modifies the `StateTransition.TransitionDb` function to mint FCT (see section 5.1. ↗). Instead of minting all the gas to the transaction's `From` address, it mints the remainder after the amount to be used in the current transaction to the transaction's `L1TxOrigin` address if present (i.e., if the transaction is a deposit transaction). It also ignores the `ErrGasLimitReached` error from `StateTransition.innerTransitionDb` for deposit transactions, allowing them to always be included (since they mint the gas required to pay for their own cost).

Function: `StateTransition.refundGas`

Facet modifies the `StateTransition.refundGas` function to not refund gas to the system address (because system-address transactions have gas credited to them) and to refund gas for deposit transactions to their `L1TxOrigin` instead of their `From` address.

File: core/types/deposit_tx.go

Type: `DepositTx`

Facet adds `L1TxOrigin` and `GasFeeCap` fields to the `DepositTx` type. The `L1TxOrigin` is the origin of the depositor on the L1 and is used when minting FCT. The `GasFeeCap` is used in `StateTransition.buyGas` (called from `StateTransition.preCheck`) for validating the amount of gas for the transaction.

File: core/vm/contracts.go

Function: `sqrtWithFloatPrecision.Run`

Facet adds a `sqrtWithFloatPrecision` precompile, which takes the square root of the arbitrary-precision integer passed as an argument as a 64-bit float and returns it as an arbitrary-precision integer. It panics when handling floating point infinities (see Finding 3.1. ↗). Due to the conversions through arbitrary-precision integers, negatives/NaNs cannot reach the native `Sqrt` call.

> Facet Response:
>
> This precompile has been removed from the production version of facet-geth. It is used only in a separate branch to generate the genesis.json migration data.

File: crypto/crypto.go

Function: `CreateAddress`

Facet modifies the `CreateAddress` function, used to create fresh addresses for deploying smart contracts at, by using the string `"facet"` as a domain separator to ensure that the addresses are disjoint from those generated on the L1.

> Facet Response:
>
> A change was implemented to adopt Ethereum's standard CREATE address-generation method. This requires aliasing L1 contract addresses in Facet calls and Facet follows Optimism's approach here.

6. Assessment Results

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

During our assessment on the scoped Facet Geth modules, we discovered one finding, which was critical.

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.