Zellic

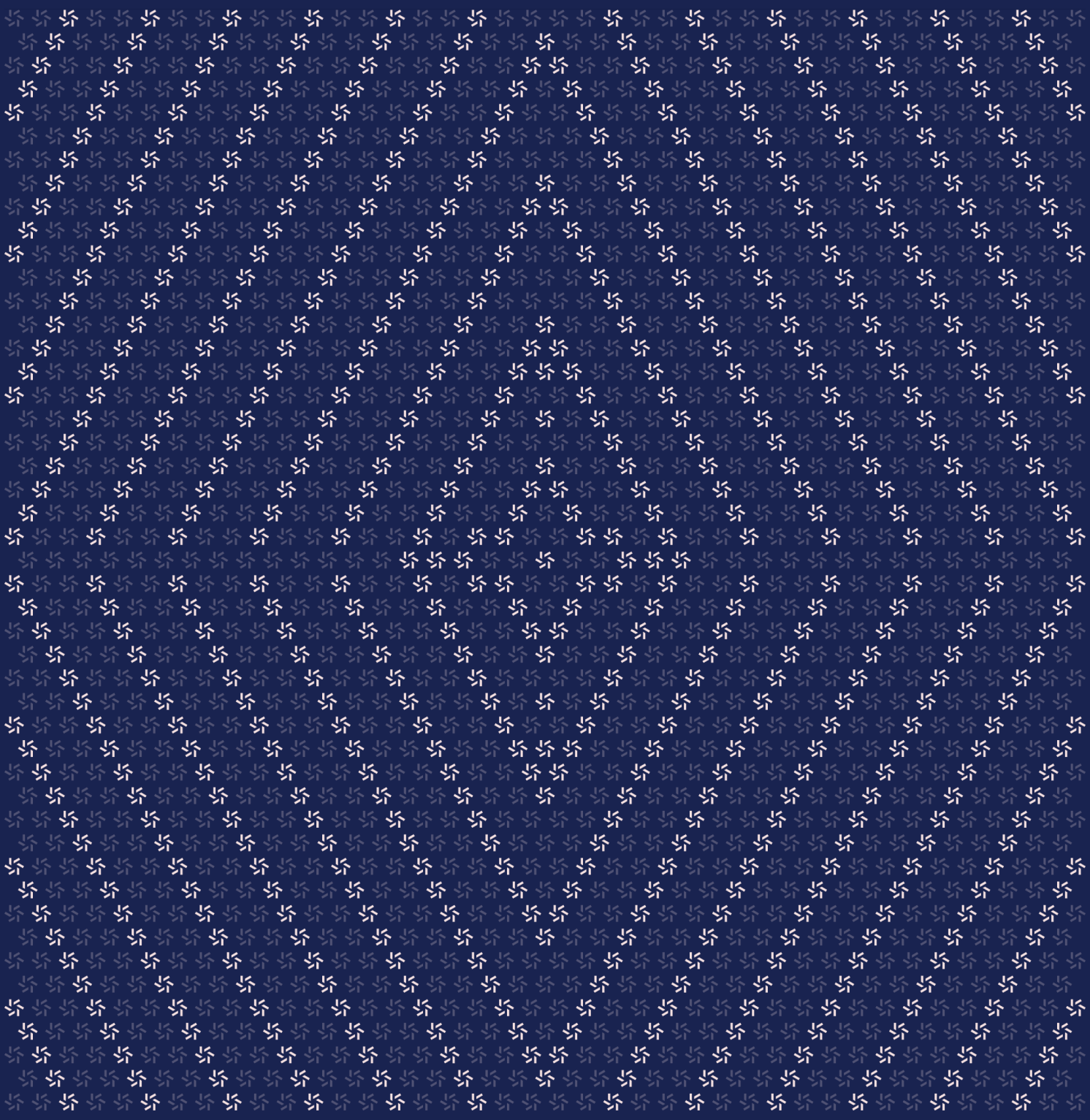
**Prepared for**
**Alex Clement de l'Epine**
Deep Blue

**Prepared by**
**Vlad Toie**
**Varun Verma**
Zellic

**October 21, 2024**

# Extended ERC20

## 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 Deep Blue from October 17th to October 18th. During this engagement, Zellic reviewed Extended ERC20'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:

- What does ExtendedERC20 add over the standard ERC-20 token?
- Are there any edge cases to the KYC verification process?
- What are the implications of limiting transfers to non-blacklisted addresses?

## 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
- Code outside the scope of the project

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

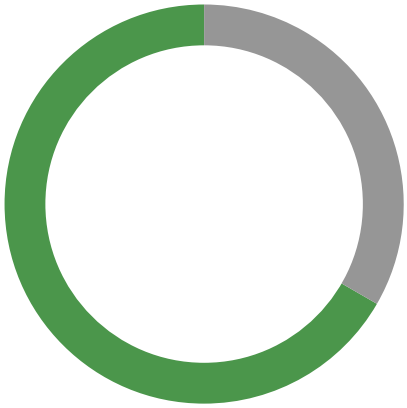
## 1.4. Results

During our assessment on the scoped Extended ERC20 contracts, we discovered three findings. No critical issues were found. Two findings were of low impact and the remaining finding was informational in nature.

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

## Breakdown of Finding Impacts

| Impact Level | Count |
|---|---|
| Critical | 0 |
| High | 0 |
| Medium | 0 |
| Low | 2 |
| Informational | 1 |



# 2. Introduction

## 2.1. About Extended ERC20

Deep Blue contributed the following description of Extended ERC20:

> Deep Blue is a stablecoin issuance company that is launching its first stablecoin, DBUSD. A USD fiat backed 1:1 stablecoin. Deep Blue is based out of Jersey in the Channel Islands and has been given consent to issue by the JFSC. Additionally, the company is working with a large global network of key industry partners including CEXs, protocols and chains to build out a strong ecosystem of users and partners.

## 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:

### Extended ERC20 Contracts

| | |
|---|---|
| **Type** | Solidity |
| **Platform** | EVM-compatible |
| **Target** | Zip File provided by Deep Blue |
| **Repository** | Zip File ↗ |
| **Version** | `0827f6acffd0460d72919e725493a04cf2dce4b00275e38323cb52d60741fc21` |
| **Programs** | `ExtendedERC20` |

## 2.4. Project Overview

Zellic was contracted to perform a security assessment for a total of 4 person-days. The assessment was conducted by two consultants over the course of one calendar week.

### 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:

**Vlad Toie**
Engineer
vlad@zellic.io ↗

**Varun Verma**
Engineer
varun@zellic.io ↗

## 2.5. Project Timeline

The key dates of the engagement are detailed below.

| | |
|---|---|
| **October 17, 2024** | Start of primary review period |
| **October 19, 2024** | End of primary review period |

# 3. Detailed Findings

## 3.1. The `changeOwner` function does not update the `UPGRADER_ROLE`

| | | | |
|---|---|---|---|
| **Target** | ExtendedERC20 | | |
| **Category** | Coding Mistakes | **Severity** | Low |
| **Likelihood** | Low | **Impact** | Low |

### Description

The `changeOwner` function is used to change the owner of the contract. The function revokes the current owner's roles and grants the roles to the new owner. However, the `UPGRADER_ROLE` is not revoked from the current owner, nor is it granted to the new owner.

```
function changeOwner(address newOwnerAddress)
    public virtual onlyRole(DEFAULT_ADMIN_ROLE) {
    _revokeRole(DEFAULT_ADMIN_ROLE, _msgSender());
    _revokeRole(PAUSER_ROLE, _msgSender());
    _revokeRole(MINTER_ROLE, _msgSender());
    _revokeRole(WIPER_ROLE, _msgSender());
    _revokeRole(KYC_ROLE, _msgSender());
    // @audit-issue UPGRADER_ROLE is not revoked from the current owner, nor is
    it granted to the new owner
    _grantRole(DEFAULT_ADMIN_ROLE, newOwnerAddress);
    _grantRole(PAUSER_ROLE, newOwnerAddress);
    _grantRole(MINTER_ROLE, newOwnerAddress);
    _grantRole(WIPER_ROLE, newOwnerAddress);
    _grantRole(KYC_ROLE, newOwnerAddress);
}
```

### Impact

This allows for the previous owner to still have the ability to upgrade the contract, even after transferring ownership.

### Recommendations

We recommend revoking the `UPGRADER_ROLE` from the current owner and granting it to the new owner in the `changeOwner` function.

```
function changeOwner(address newOwnerAddress)
    public virtual onlyRole(DEFAULT_ADMIN_ROLE) {
    _revokeRole(DEFAULT_ADMIN_ROLE, _msgSender());
    // ...
  _revokeRole(UPGRADER_ROLE, _msgSender());
    _grantRole(DEFAULT_ADMIN_ROLE, newOwnerAddress);
    // ...
  _grantRole(UPGRADER_ROLE, newOwnerAddress);
}
```

### Remediation

This issue has been acknowledged by Deep Blue.

## 3.2. Duplicate values of registry

| | | | |
|---|---|---|---|
| **Target** | DocumentHashRegistry | | |
| **Category** | Coding Mistakes | **Severity** | Low |
| **Likelihood** | Low | **Impact** | Low |

### Description

The contract's document-management system contains an error in its hash-removal process. When removing or updating a document hash, the function sets `registeredHashes[existingHash] = false` without verifying if other keys still reference that hash. This leads to `isHashRegistered()` incorrectly returning false for a hash that may still be associated with other keys in the `documentHashes` mapping.

In the following, we see the removal process:

```
else if (value == bytes32(0) && existingHash != bytes32(0)) {
    documentHashes[key] = bytes32(0);
    registeredHashes[existingHash] = false;
    emit DocumentHashRemoved(key, existingHash);
}
```

The issue occurs in the following scenario:

1. Two different keys point to the same hash value.
2. The hash value for one key is deleted or changed.
3. The contract incorrectly assumes the hash value is no longer present anywhere, even though it still exists for the other key.

### Impact

This bug can lead to an inconsistent state in the contract, where a hash is still in use but reported as unregistered. This could potentially disrupt operations that rely on accurate hash-registration status.

### Recommendations

We recommend enforcing uniqueness of hash values across all keys. This can be achieved by checking if the hash is already registered before adding a new one via the following in the function body:

```
require(!registeredHashes[value], "Hash already registered");
```

### Remediation

This issue has been acknowledged by Deep Blue.

### 3.3. The `__DocumentHashRegistry_init()` function should be internal

| | | | |
|---|---|---|---|
| **Target** | DocumentHashRegistry | | |
| **Category** | Coding Mistakes | **Severity** | Informational |
| **Likelihood** | N/A | **Impact** | Informational |

#### Description

The `__DocumentHashRegistry_init()` function in the DocumentHashRegistry contract is currently declared as public. This allows it to be called directly from outside the contract, which is not the intended use for initialization functions in upgradable contracts.

```
function __DocumentHashRegistry_init() public virtual initializer {
    __AccessControl_init();
    __DocumentHashRegistry_init_unchained();
}
```

#### Impact

While the `initializer` modifier prevents multiple initializations, having this function public unnecessarily exposes an internal implementation detail.

#### Recommendations

We recommend changing the visibility of `__DocumentHashRegistry_init()` from public to internal.

#### Remediation

This issue has been acknowledged by Deep Blue.

# 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. Comprehensive test coverage and upgradability

We provided the client and tested the code with a test suite that provides extensive coverage of the ExtendedERC20 contract, including

- Basic ERC-20 functionality
- KYC and blacklist features
- Minting, burning, and wiping tokens
- Pausing and unpausing transfers
- Role-based access control
- Document-hash registry
- Disabling/enabling KYC and blacklist features
- Batch role management
- Contract initialization and reinitialization prevention

While the client's tests focused primarily on blacklist functionality, we recommend that the client reproduces and expands upon these tests to ensure full coverage of all contract features.

Regarding upgradability, while we have included basic tests for the upgrade process, role preservation, and storage integrity, we strongly advise conducting more extensive upgradability testing.

# 5. System Design

This provides a description of the high-level components of the system and how they interact, including details like a function's externally controllable inputs and how an attacker could leverage each input to cause harm or which invariants or constraints of the system are critical and must always be upheld.

Not all components in the audit scope may have been modeled. The absence of a component in this section does not necessarily suggest that it is safe.

## DocumentHashRegistry

The DocumentHashRegistry contract is designed to store and manage hashes of documents. Its main function, `updateDocumentHashes`, can only be called by the admin and is used to add/update or remove existing document hashes.

## ExtendedERC20

This implements additional functionalities on top of the ERC-20 standard. It allows the admin to blacklist and KYC addresses. The following changes are noted:

- The `transfer` function has been modified to check if the sender and receiver are blacklisted/KYC'd.
- The `transferFrom` function has been modified to check if the sender and receiver are blacklisted/KYC'd.
- The `mint` function checks if the receiver is blacklisted/KYC'd. It can only be called by the admin.

It also introduces the following main actors within the system:

- `DEFAULT_ADMIN_ROLE` — Overall contract administrator
- `PAUSER_ROLE` — Can pause/unpause token transfers
- `MINTER_ROLE` — Can create new tokens
- `WIPER_ROLE` — Can burn tokens from any account
- `KYC_ROLE` — Manages KYC approvals
- `UPGRADER_ROLE` — Can upgrade the contract

With the key important state transitions being

- `Minting` — Increases the total supply and recipient's balance
- `Burning/Wiping` — Decreases total supply and account's balance
- `Pausing` — Halts all token transfers
- `Changing KYC/Blacklist status` — Affects transfer permissions

# 6. Assessment Results

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

During our assessment on the scoped Extended ERC20 contracts, we discovered three findings. No critical issues were found. Two findings were of low impact and the remaining finding was informational in nature.

## 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.