

ZEALYNX

Web3 Security and AI Agent & MCP Audits

Plunid

Smart Contract Audit

FINAL PRIVATE REPORT

Mitigation review through 18 November 2026

18 November 2026

Zealynx

contact@zealynx.io

Contents

1. About Zealynx

2. Disclaimer

3. Overview

3.1 Project Summary

3.2 About Plunid

3.3 Audit Scope

4. Audit Methodology

5. Severity Classification

6. Executive Summary

6.1 The Key Findings

6.2 Architectural Security Observations

6.3 Security Strengths Observed

7. Audit Findings

7.1 Medium Severity Findings

7.2 Low Severity Findings

7.3 Informational Severity Findings

1. About Zealynx

Zealynx, founded in January 2024 by Carlos (Bloqarl), specializes in smart contract audits, and development. Our services include comprehensive smart contract audits, application security audits, such as pentesting, and AI Audits. We are trusted by clients such as Badger DAO, Ample protocol, Lido, Inverter, Matchain, and Golden Grid.

Our team believes in transparency and actively contributes to the community by creating educational content. This includes challenges for Foundry and Rust, security tips for Solidity developers, and fuzzing materials like Foundry and Echidna/Medusa. We also publish articles on topics such as preparing for an audit and battle testing smart contracts on our website Zealynx.io and our blogs.

Zealynx has achieved public recognition, including winning 1st prize in the Uniswap Hook Hackathon and a Top 5 position in the Beanstalk Audit public contest. Our ongoing commitment is to provide value through our expertise and innovative security solutions for smart contracts.

2. Disclaimer

A security review of a smart contract can't guarantee there are no vulnerabilities. It's a limited effort in terms of time, resources, and expertise to find as many vulnerabilities as possible. We can not assure 100% security post-review or guarantee the discovery of any issues with your smart contracts.

3. Overview

3.1 Project Summary

Plunid combines permissioned EURP transfers with on-chain profiles, privileged administration and off-chain identity and reward processes. The initial audit covered the frozen smart contracts and listed deployment scripts from 9 to 13 November 2026. This final private report records the original findings and the source-level mitigation review completed through 18 November 2026.

3.2 About Plunid

Crypto Plunid d.o.o. Niš operates the Plunid platform and Plunid EUR. PlunidAccounts defines administration and recovery roles. FieldManagement defines profile fields; ProfileManagement records user status, currency permissions and payout checkpoints. Eurp applies those policies to token issuance, transfers, transaction fees and reward-related events. Identity checks, private records, fiat backing and reward disbursement depend on off-chain processes outside this contract audit.

3.3 Audit Scope

Reviewed source: `dusan-markovic/plunid-audit@4c37ce69136bc89c57b7a39195d2e5c0ed0a156e`, tag `audit-ready`. Initial audit: 9-13 November 2026. The web portal, external databases, infrastructure, fiat reserves, actual KYC operations and legal/regulatory compliance are excluded. The mitigation review examined the separate fixes branch at commit 469d396819481af3a589908675cd7c53e4e2c9fa. Original finding references remain pinned to the frozen baseline.

```
contracts/PlunidAccounts.sol
contracts/FieldManagement.sol
contracts/ProfileManagement.sol
contracts/Eurp.sol
contracts/ERC20Interface.sol
contracts/Migrations.sol
migrations/1_initial_migration.js
migrations/2_deploy_contracts.js
migrations/3_deploy_eurp.js
```

4. Audit Methodology

Approach

The assessment combined source-code review, analysis of the supplied architecture and rights documentation, client clarification, and focused local scenario and regression tests recorded in the audit evidence. Findings were evaluated against the frozen revision and the stated administrative and off-chain trust assumptions. Submitted remediation was reviewed through 18 November 2026 with source comparison, local compilation, and focused EVM tests. No production interaction, deployed-bytecode comparison, portal verification, or verification of fiat reserves is implied.

Throughout the smart contract audit process, we prioritize the following aspects to uphold excellence:

> **1. Code Quality:** We diligently evaluate the overall quality of the code, aiming to identify any potential vulnerabilities or weaknesses.
> **2. Best Practices:** Our assessments emphasize adherence to established best practices, ensuring that the smart contract follows industry-accepted guidelines and standards.
> **3. Documentation and Comments:** We meticulously review code documentation and comments to ensure they accurately reflect the underlying logic and expected behavior of the contract.

5. Severity Classification

Severity	Impact: High	Impact: Medium	Impact: Low
Likelihood: High	Critical	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

6. Executive Summary

The initial audit is complete. The delivered set contains **12 findings: 2 Medium, 7 Low and 3 Informational**, with no Critical or High in that set. The Medium findings concern owner-paid fees and restricted spenders using existing EURP allowances. Lower-severity findings address recovery authority, profile lifecycle, coded transfers, payout metadata and documentation.

At the reviewed fixes revision, eight findings are verified fixed in the supplied source (M-01, M-02, L-02, L-04, L-05, L-07, I-01 and I-02). L-01 and L-03 are superseded because the client removed recipient purpose-code lists. The client retained the self-transfer fee (L-06). The audit lead acknowledges I-03 after the documentation correction and new migration check, while actual deployment execution and Reserve signer custody remain unverified.

This is the **final private report for review** against frozen audit baseline 4c37ce69136bc89c57b7a39195d2e5c0ed0a156e and source-level mitigation revision 469d396819481af3a589908675cd7c53e4e2c9fa. The committed contract suite compiles but has three stale tests (12 passing of 15): two call a removed purpose-code setter and one assumes the old allowance rules. Focused mitigation checks passed. Integrations must account for the changed purpose-code policy and event ABI. These results do not certify deployed bytecode, off-chain operations, reserves, or launch readiness.

6.1 The Key Findings

Delegated transfers: principal-only allowances do not bound owner-funded fees (M-01); restricted spenders can retain use of an earlier allowance (M-02).

Administration and lifecycle: CST grants survive Foundation replacement (L-02); a funded profile can return to Registered (L-05); address migration omits the payout checkpoint (L-07).

Transfer compatibility and policy: partial purpose-code matching, empty optional lists, zero-value transfers and self-transfer fees require the specific dispositions recorded in L-01, L-03, L-04 and L-06.

Documentation and public data: independent KYC flags, public-field history and the Reserve signer identity need accurate descriptions (I-01 to I-03).

6.2 Architectural Security Observations

Payment authorization crosses contract boundaries. Current roles, profile status and currency permissions are stored separately from the EURP transfer implementation. A locally correct check does not establish that every payment route enforces the intended policy. M-01 and M-02 expose distinct fee-budget and delegated-spender gaps.

Recovery does not automatically revoke delegated administration. Matching recovery votes can replace privileged addresses, but CST membership has an independent lifetime. L-02 qualifies what Foundation replacement achieves. The reviewed Reserve-controlled pause blocks CST field writes, but recovery still depends on pausing before Foundation rotation and revoking unwanted CST grants before unpausing. I-03 concerns the documented identity of that recovery signer.

The trust boundary is deliberately hybrid. Contracts hold balances, permissions, public fields and reward-related markers. Identity decisions, confidential records, fiat backing and actual PURP payments are off-chain. The review does not validate those external operations. Profile migration copies selected metadata rather than sweeping token balances; L-07 addresses the separate payout-checkpoint omission.

A field flag cannot erase public history. I-02 concerns relabelling an already public field as confidential; neither a later flag change nor a new address removes historical disclosure. I-01 separately distinguishes nonempty-field validation from the KYC suspension flag.

6.3 Security Strengths Observed

Explicit privileged roles and recovery voting. Initial role addresses must be nonzero and distinct. Superadmin replacement requires matching suggestions from two current voters. CST persistence and the Reserve documentation findings qualify this recovery boundary.

Restricted initial profile state. Registration starts at Registered with a zeroed currency mask. CST must separately enable status and currency permission. Ordinary user transfers check profile status and the EURP bit; the delegated-spender and funded-downgrade findings identify the limits of those controls.

Narrow supply authority. The current Treasury alone can mint or burn. Those paths require an unpaused token, update supply and balances together, and emit transfer events. This does not establish off-chain fiat backing.

Fixed module bindings with current role lookups. Immutable contract references constrain module replacement, while guards read current administrative roles. The separate CST mapping needs its own rotation treatment.

Concrete field-write guards. Field definitions are Foundation-controlled. Writes check eligible profile states and enabled fields; required public values cannot be empty, and a KYC-required write suspends an Active profile. These controls do not certify off-chain KYC or historical confidentiality.

Summary of Findings:

Vulnerability	Severity	Status
[M-01] Bound owner-paid fees on delegated EURP transfers	Medium	Fixed
[M-02] Recheck EURP eligibility of delegated spenders	Medium	Fixed
[L-01] Reject partial purpose-code matches in coded transfers	Low	Superseded
[L-02] Expire CST grants when Foundation changes	Low	Fixed
[L-03] Allow coded transfers when the recipient has no purpose-code list	Low	Superseded
[L-04] Process zero-value EURP transfers for eligible profiles	Low	Fixed
[L-05] Prevent funded profiles from returning to Registered	Low	Fixed
[L-06] Avoid charging the EURP fee on self-transfers	Low	Acknowledged
[L-07] Carry the PURP payout checkpoint across profile migration	Low	Fixed
[I-01] Document the independent field KYC suspension flag	Info	Fixed
[I-02] Prevent public fields from being relabeled confidential	Info	Fixed
[I-03] Correct the documented Reserve recovery address	Info	Acknowledged

7. Audit Findings

7.1 Medium Severity Findings

[M-01] Bound owner-paid fees on delegated EURP transfers

> **Fixed.** Source-level fix verified at b8213ee: delegated allowance now covers the owner's actual debit, including the platform fee. Callers must approve principal plus fee.
> Original finding and client discussion

Affected files

contracts/Eurp.sol:63-81

contracts/Eurp.sol:124-175

Description

An owner can approve a spender for a fixed number of EURP base units. On an eligible delegated transfer between users, `transferFrom` checks and reduces the allowance by the principal alone, while `_transfer` also charges the fixed transaction fee to the owner. The spender controls how many positive transfers consume that allowance. Each small transfer creates another owner-funded fee.

Vulnerable Scenario:

1. An Active owner with an eligible recipient approves a spender for 10 base units.
2. The spender calls `transferFrom(owner, recipient, 1)` ten times.
3. The recipient receives 10 units and the allowance reaches zero, but the owner also pays ten fees to Foundation.
4. Sending the same 10 units in one delegated call would charge one fee. The documented fee per transaction and principal-only allowance behavior both hold for each individual call; their combination leaves the owner without a limit on aggregate fees imposed by the spender.

Impact

An approved spender can drain the owner's EURP balance through repeated fees that exceed the approved principal. The fees remain with Foundation after the owner revokes the allowance.

Recommendations

In `Eurp.transferFrom`, retain the initial `allowances[from][msg.sender] >= tokens` check. Replace the successful-transfer allowance update with:

```
uint256 balanceBefore = balances[from];
bool success = _transfer(from, to, tokens);

if (success) {
    uint256 actualDebit = balanceBefore - balances[from];
    uint256 allowanceDebit = actualDebit > tokens ?
actualDebit : tokens;
    require(
        allowances[from][msg.sender] >= allowanceDebit,
        "Insufficient allowance for debit"
    );
    allowances[from][msg.sender] -= allowanceDebit;
}
```

This consumes principal plus fee for an ordinary fee-bearing transfer and principal for a fee-free transfer. The `max` also preserves nominal principal consumption on a self-transfer when its net balance change is smaller than `tokens`; if the self-transfer fee is larger, the fee is covered. If `_transfer` returns `false`, the allowance stays unchanged. If the allowance cannot cover the final debit, the transaction reverts, rolling back `_transfer` and its events.

This changes the existing principal-only approval semantics on fee-bearing transfers. Callers must approve enough to cover fees, and integrations and tests that calculate remaining allowance must be updated.

[M-02] Recheck EURP eligibility of delegated spenders

> **Fixed.** Source-level fix verified at bbab26a: positive approvals check owner and spender eligibility, and transferFrom rechecks the spender. The submitted allowance test still needs updating.
> Original finding and client discussion

Affected files

contracts/Eurp.sol:63-81

contracts/Eurp.sol:84-157

Description

`approve` accepts any nonzero spender address. `transferFrom` checks that address's allowance but passes only the owner and recipient to `_transfer`. The status and EURP-permission checks in `_transfer` therefore never inspect the caller using the allowance. Restricting or disabling a spender's profile does not stop that spender from using an existing approval.

Vulnerable Scenario:

An Active, EURP-permitted owner approves an Active, EURP-permitted spender for 1 EURP. The owner holds at least 1.1 EURP, and the recipient is another Active, EURP-permitted profile.

CST changes the spender to Blocked. The spender's direct `transfer(recipient, 1 EURP)` returns `false`.

The same spender calls `transferFrom(owner, recipient, 1 EURP)`. It succeeds: the recipient gains 1 EURP and the owner loses 1.1 EURP at the default fee. The same route remains open after the spender becomes Disabled or loses EURP permission while the allowance remains.

Impact

A restricted spender can continue moving an approving owner's EURP to eligible recipients, up to the remaining allowance, after the restriction was meant to stop delegated spending. The owner can revoke future use but cannot reverse transfers already made.

Recommendations

Check both parties when creating a **positive** allowance, then recheck the spender when it uses that allowance. Use the existing `isCurrencyAllowed` helper and leave `approve(spender, 0)` available so an owner can revoke an allowance even after either profile is restricted:

```
@@ function approve(address spender, uint tokens)
     require(spender != address(0), "Invalid spender address");
+    if (tokens > 0) {
+      require(
+        profileManagement.profileStatus(msg.sender) ==
ProfileManagement.ProfileStatus.Active &&
+        isCurrencyAllowed(msg.sender),
+        "Owner not eligible for EURP approval"
+      );
+      ProfileManagement.ProfileStatus spenderStatus =
profileManagement.profileStatus(spender);
+      require(
+        (spenderStatus == ProfileManagement.ProfileStatus.Active ||
+         spenderStatus == ProfileManagement.ProfileStatus.Suspended) &&
+        isCurrencyAllowed(spender),
+        "Spender not eligible for EURP approval"
+      );
+    }
     allowances[msg.sender][spender] = tokens;
@@ function transferFrom(address from, address to, uint256 tokens)
     require(allowances[from][msg.sender] >= tokens, "Insufficient
allowance");
+    ProfileManagement.ProfileStatus spenderStatus =
profileManagement.profileStatus(msg.sender);
+    require(
+      (spenderStatus == ProfileManagement.ProfileStatus.Active ||
+       spenderStatus == ProfileManagement.ProfileStatus.Suspended) &&
+      isCurrencyAllowed(msg.sender),
+      "Spender not eligible for EURP transfer"
+    );

     bool success = _transfer(from, to, tokens);
```

`_transfer` continues to enforce the owner and recipient restrictions. This changes delegated-transfer eligibility: integrations and the existing allowance test must stop using an unregistered spender. It also prevents unprofiled Foundation or Treasury addresses from using positive allowances; retain an administrative exception only if that behavior is explicitly required.

7.2 Low Severity Findings

[L-01] Reject partial purpose-code matches in coded transfers

> **Superseded.** Recipient purpose-code lists and their substring lookup were removed at fe9733c. The original partial-match scenario is obsolete; any valid four-uppercase-letter code is now accepted for an eligible recipient.
> Original finding and client discussion

Affected files

contracts/ProfileManagement.sol:245-261

contracts/Eurp.sol:190-196

Description

A recipient's comma-separated purpose-code list is intended to admit only listed codes on transferWithData. With the valid list GOOD, OTHR, hasProfilePurposeCode accepts the unlisted fragment GOO because it searches for a substring. A sender can use GOO to deliver EURP and emit TransferWithPurposeCodeSucceeded as though GOO were an authorized code. This misclassifies the coded payment. Ordinary transfer is explicitly allowed to bypass purpose-code checks.

Recommendations

In `ProfileManagement.hasProfilePurposeCode`, replace the call to `containsSubstring(profilePurposeCode[user], code)` with a comparison against complete comma-delimited entries. Keep `containsSubstring` itself if other callers use its public selector. A direct replacement for the return statement is:

```
bytes memory list = bytes(profilePurposeCode[user]);
bytes memory sought = bytes(code);
if (sought.length == 0) return false;
uint256 start = 0;
for (uint256 i = 0; i <= list.length; i++) {
    if (i == list.length || list[i] == 0x2c) {
        uint256 first = start;
        uint256 last = i;
        while (first < last && list[first] == 0x20) first++;
        while (last > first && list[last - 1] == 0x20) last--;
        if (last - first == sought.length) {
            bool equal = true;
            for (uint256 j = 0; j < sought.length; j++) {
                if (list[first + j] != sought[j]) {
                    equal = false;
                    break;
                }
            }
            if (equal) return true;
        }
        start = i + 1;
    }
}
return false;
```

This keeps the existing case-sensitive byte comparison and permits spaces around list entries. It rejects an empty code, a prefix such as G00, and a code spanning a comma. It leaves ordinary `transfer` unchanged. The extra scan is proportional to the stored list length.

[L-02] Expire CST grants when Foundation changes

> **Fixed.** Source-level pause fix verified at 4746346: setProfileField now respects the Reserve pause. The recovery procedure still requires timely pausing and manual CST revocation.
> Original finding and client discussion

Affected files

contracts/PlunidAccounts.sol:149-159

contracts/ProfileManagement.sol:54-60

contracts/ProfileManagement.sol:137-155

Description

Foundation replacement changes the role address, but grants in the separate CSTAddresses mapping survive. A compromised Foundation can add its own address or an alias as a CST member before the founders replace it. After the rotation, that address still passes the CST checks for profile status, currency permissions, and field edits.

The retained authority can disable a funded Active profile after the replacement. The user keeps its EURP balance but cannot make an ordinary transfer. Revoking the former CST member stops further changes but does not restore a Disabled profile through changeProfileStatus. Moving the affected balance requires exceptional action by both founders to temporarily make that user's address Foundation.

Recommendations

In `PlunidAccounts`, add `uint256 public foundationEpoch` and increment it in the Founder2 confirmation branch of `changeFoundation`, immediately after assigning `Foundation = newAddress`. Do not increment it when Founder1 only suggests an address or when another admin role changes.

In `ProfileManagement`, replace the public `CSTAddresses` boolean mapping with a private boolean grant flag and a private grant-epoch mapping. Use one effective-membership check for every CST authority path:

```
mapping(address => bool) private cstMemberFlags;
mapping(address => uint256) private cstGrantEpoch;

function isCurrentCST(address member) internal view returns
(bool) {
    return cstMemberFlags[member] &&
        cstGrantEpoch[member] ==
plunidAccounts.foundationEpoch();
}

function CSTAddresses(address member) external view returns
(bool) {
    return isCurrentCST(member);
}
```

Replace `CSTAddresses[msg.sender]` with `isCurrentCST(msg.sender)` in both `onlyCSTeam` and `setProfileField`. In `addCSTMember`, set the member's flag to `true` and record `plunidAccounts.foundationEpoch()`; in `removeCSTMember`, set its flag to `false`. The explicit `CSTAddresses(address)` function preserves the existing getter selector while reporting effective membership.

After each confirmed Foundation change, all earlier grants expire, even if a former Foundation address later becomes Foundation again. The new Foundation must regrant authorized members, so CST operations pause until that happens.

[L-03] Allow coded transfers when the recipient has no purpose-code list

> **Superseded.** The optional recipient purpose-code list was removed at fe9733c. Coded transfers with a valid four-uppercase-letter code no longer depend on a recipient list.
> Original finding and client discussion

Affected files

contracts/Eurp.sol:190-196

contracts/ProfileManagement.sol:245-260

Description

The architecture makes a recipient's purpose-code list optional: its restriction applies only when the list is defined. Registration leaves the list empty, but `transferWithData` unconditionally requires `hasProfilePurposeCode` to return true. With an empty list, that check rejects every nonempty code before an otherwise eligible transfer occurs. Active and Suspended recipients therefore cannot receive a coded EURP payment until CST configures a list. The call returns false, moves no tokens, and emits no coded-success event. Ordinary `transfer` still works but does not carry the coded metadata.

Recommendations

In `Eurp.transferWithData`, keep the existing `require(bytes(pCode).length > 0)` and replace the condition at line 192 with:

```
if (
    (bytes(profileManagement.profilePurposeCode(to)).length ==
0 ||
        profileManagement.hasProfilePurposeCode(to, pCode)) &&
    transfer(to, tokens)
) {
```

An empty recipient list now admits any nonempty code. A configured list still requires `hasProfilePurposeCode` to approve the code. Keep the existing `transfer(to, tokens)` call inside the condition so profile status, currency permission, balance, and pause rules continue to govern whether funds move and the success event is emitted. The additional profile getter call costs gas on coded transfers.

[L-04] Process zero-value EURP transfers for eligible profiles

> **Fixed.** Source-level fix verified at b8436ec: eligible zero-value ERC-20 transfers emit Transfer(0) without a platform fee. Coded transfers separately reject zero amounts at fe9733c.
> Original finding and client discussion

Affected files

contracts/Eurp.sol:72-86

contracts/Eurp.sol:124-175

Description

EURP exposes the ERC-20 `transfer` and `transferFrom` methods, but `_transfer` reverts whenever `tokens == 0`, before emitting a `Transfer` event. ERC-20 requires zero-value transfers to be processed and logged. For two Active profiles allowed to use EURP, a positive transfer succeeds while `transfer(recipient, 0)` reverts. An approved spender's `transferFrom(owner, recipient, 0)` also reverts.

An atomic batch that includes a zero allocation consequently fails in full. The caller pays gas and must filter zero allocations before retrying; no EURP is lost.

Recommendations

Remove the zero-amount rejection at `Eurp.sol:86`. Keep the existing sender and recipient eligibility checks. In the Active user-to-user branch, require the fee balance and set `chargeFee` only when `tokens > 0`, so an eligible zero transfer emits `Transfer(from, to, 0)` without charging a fee. In `transferFrom`, require a positive existing allowance when `tokens == 0` before calling `_transfer`; otherwise an unapproved caller could emit zero-value transfers on another user's behalf:

```
require(
    allowances[from][msg.sender] >= tokens &&
    (tokens > 0 || allowances[from][msg.sender] > 0),
    "Insufficient allowance"
);
```

The proposed change makes eligible direct and approved delegated zero transfers succeed, while unapproved zero transfers, paused transfers, and transfers involving disallowed profiles remain rejected.

[L-05] Prevent funded profiles from returning to Registered

> **Fixed.** Source-level fix verified at 7f9ea80: changeProfileStatus rejects a return to Registered while registration and address migration remain available.
> Original finding and client discussion

Affected files

contracts/ProfileManagement.sol:83-118

contracts/Eurp.sol:124-157

docs/architecture-EN.md:50-56

docs/Platform user rights EN.png

Description

Registered is described as the initial, pre-KYC status; the rights matrix says a Registered profile cannot hold or send EURP. Yet `changeProfileStatus` permits CST to select Registered for an already funded Active, Suspended, or Blocked profile. The transition does not change its EURP balance.

A user becomes Active and receives 5 EURP.

CST changes the profile from Active to Registered. The address still holds 5 EURP.

The owner's transfers to another user and to Foundation both return `false`. An approved spender's `transferFrom` also returns `false`.

The owner cannot change their own status. Access to the balance resumes if CST or Foundation moves the profile to an eligible status. With EURP permission enabled, Suspended permits a transfer to Foundation. An accidental downgrade or use of Registered for re-verification therefore prevents the holder from using their funds until an administrator intervenes. The balance is not destroyed.

Recommendations

Make Registered an initial-only status: add `require(newStatus != ProfileStatus.Registered, "Registered is an initial-only status!");` after the existing `newStatus != ProfileStatus.None` check in `changeProfileStatus`. Keep `registerProfile` and the new-address initialization in `changeProfileAddress` able to create Registered profiles. Use Suspended for post-activation review, or Blocked when the intended restriction is stricter; both retain the Foundation exit while EURP permission remains enabled. This prevents a funded profile from entering the inaccessible Registered state while preserving initial registration, Registered-profile address migration, and the existing post-activation review path.

[L-06] Avoid charging the EURP fee on self-transfers

> **Acknowledged.** The client explicitly elected to retain the fixed fee on self-transfers. This behavior remains in the reviewed source and is an accepted residual issue, not a fix.
> Original finding and client discussion

Affected files

contracts/Eurp.sol:124-175

docs/architecture-EN.md:98-100

Description

The architecture applies the fixed platform fee when an Active user sends EURP to another user. In `Eurp._transfer`, the Active user-to-user branch also matches `from == to`: the sender's own Active address is an eligible recipient. The function requires `tokens + txFee`, transfers the principal to the same balance, then charges `txFee` to Foundation and emits `TaxCalculated`.

For example, with a 5 EURP balance, `transfer(self, 1 EURP)` leaves the sender with 4.9 EURP and credits Foundation 0.1 EURP even though no principal reached another address. A self-transfer of the entire 5 EURP reverts for lack of an additional fee. An accidental self-transfer therefore costs the user the fixed fee.

Recommendations

In the Active user-to-user branch, set `chargeFee = from != to` before the balance check. Replace the unconditional `balances[from] >= tokens + txFee` condition with `balances[from] >= tokens + (chargeFee ? txFee : 0)`. Keep the sender/recipient status and currency checks and the existing `Transfer(from, to, tokens)` emission. A self-transfer then requires only the principal balance, moves no net principal, and emits neither the Foundation fee transfer nor `TaxCalculated`. Transfers between distinct eligible addresses continue to charge the same fee and calculate the same reward.

[L-07] Carry the PURP payout checkpoint across profile migration

> **Fixed.** Source-level fix verified at 1bec900: changeProfileAddress carries lastTaxReturnWithdraw to the replacement address.
> Original finding and client discussion

Affected files

contracts/ProfileManagement.sol:192-196

contracts/ProfileManagement.sol:199-235

Description

The platform uses `lastTaxReturnWithdraw` to record the last EURP transaction included in a user's paid PURP reward calculation. The marker is stored by profile address. `changeProfileAddress` copies the profile's type, status, public fields, purpose code and currency permissions to the new address, but leaves this marker at the old address.

CST records a paid-through transaction hash for a Personal profile.

CST migrates that profile to a new address.

The old address retains the hash, while `lastTaxReturnWithdraw(newAddress)` returns an empty string.

A payout request from the new address therefore lacks the checkpoint needed to delimit the next calculation. The off-chain tool or CST must reconstruct the old address's marker separately; otherwise eligible transactions may be skipped or counted again. The payout process requires the marker to follow the user's profile.

Recommendations

In `changeProfileAddress`, copy the marker alongside the other address-keyed profile data, before restoring the migrated status:

```
lastTaxReturnWithdraw[newAddress] =
lastTaxReturnWithdraw[oldAddress];
```

Keep the old address's marker as historical data. The copy makes the new profile expose the same paid-through hash immediately after migration, including after repeated migrations, without changing initial registration or the authorized `setLastTaxReturnWithdraw` update path.

7.3 Informational Severity Findings

[I-01] Document the independent field KYC suspension flag

> **Fixed.** English and Serbian architecture documents now distinguish isKYCRequired from isRequired and describe the actual suspension rule.
> Original finding and client discussion

Affected files

docs/architecture-EN.md:79-91

docs/architecture-SR.md:74-86

contracts/FieldManagement.sol:10-14

contracts/ProfileManagement.sol:165-188

Description

The architecture lists `isPublic`, `isEnabled`, and `isRequired` as all field attributes and says an Active profile is suspended when a required field changes. The frozen contracts define a fourth, independent flag, `isKYCRequired`. `setProfileField` suspends an Active profile based on `isKYCRequired`, while `isRequired` only rejects an empty value for a public field.

Both mixed configurations are accepted. An Active owner can change a public field with `isRequired=true, isKYCRequired=false` and remain Active; changing one with `isRequired=false, isKYCRequired=true` suspends the profile. Administrators or integrations following the architecture can therefore apply the wrong KYC-review rule.

Recommendations

Update both language versions of the architecture to list `isKYCRequired` as a fourth field attribute. Replace the required-field suspension description with the implemented rule: any successful `setProfileField` call for an Active profile suspends it when `isKYCRequired=true`, regardless of `isRequired`, whether the field is public or private, or whether a supplied public value differs from its current value. State that `isRequired` enforces a nonempty value on-chain only for public fields; private values are handled off-chain. Add both mixed-flag examples above so field configuration and client code use the intended flag.

[I-02] Prevent public fields from being relabeled confidential

> **Fixed.** Source-level guard prevents a public field from being relabeled private; both architecture documents explain the irreversible public history.
> Original finding and client discussion

Affected files

contracts/FieldManagement.sol:46-50

contracts/ProfileManagement.sol:165-184

contracts/ProfileManagement.sol:218-227

docs/architecture-EN.md:79-89

docs/architecture-SR.md:74-84

Description

The architecture says a private field's value is confidential and stored only in the external database. Foundation can nevertheless change an already populated public field to `isPublic=false`. That flag change does not remove its existing value from `profileFieldsCollection`. A private-field update accepts an empty on-chain marker but also leaves the old public value in place; profile-address migration copies it to the new address. If Foundation later marks the field public again, the old value appears as its current on-chain value.

A field labeled private can therefore still expose a previously written value through the public getter, and a later publicity change can surface stale data. The original public transaction history is permanently visible regardless of any flag change.

Recommendations

Reject public-to-private changes in `changeFieldPublicity` after the ID check, for example with `require(!fields[id].isPublic || value, "Public field cannot become private!");`. To make a public field private, create a new field ID with `isPublic=false`, disable the old public field, and switch the web form and database mapping to the new ID. Update the English and Serbian architecture to state that publicity changes cannot retroactively hide on-chain values and that an existing public field cannot be relabeled private. The guard prevents new private labels from being attached to fields with public history; it does not erase values already published on-chain.

[I-03] Correct the documented Reserve recovery address

> **Acknowledged.** Reserve is documented as the deployer address, and migration 2 now checks Reserve() against the deployment sender. The audit lead acknowledges residual deployment and custody verification without an executed deployment record.
> Original finding and client discussion

Affected files

docs/contracts/permissions-and-roles-SR.md:3-16

contracts/PlunidAccounts.sol:30-51

migrations/2_deploy_contracts.js:1-23

Description

The Serbian roles document identifies Reserve as the PlunidAccounts contract address. The constructor instead sets `Reserve = msg.sender`, and the supplied migration deploys the contract from the deployment account. Reserve is therefore the deployer's address, whose signer can cast the third vote in the two-of-three superadmin recovery process; the PlunidAccounts contract address is not that signer.

The incorrect identity in the custody documentation could lead operators to preserve or monitor the contract address instead of the deployer key, reducing the recovery mechanism to the two founder keys if the deployer key is lost.

Recommendations

In `permissions-and-roles-SR.md`, replace “Reserve (ovo je ujedno i adresa ugovora)” with “Reserve (EOA adresa naloga koji je deployovao ugovor PlunidAccounts.sol; nije adresa ugovora)”. State that the deployer key is held under the intended joint-custody arrangement, and require the deployment checklist to record and verify the live `Reserve()` getter against the deployer address. No constructor change is needed: `Reserve = msg.sender` implements the intended authority.