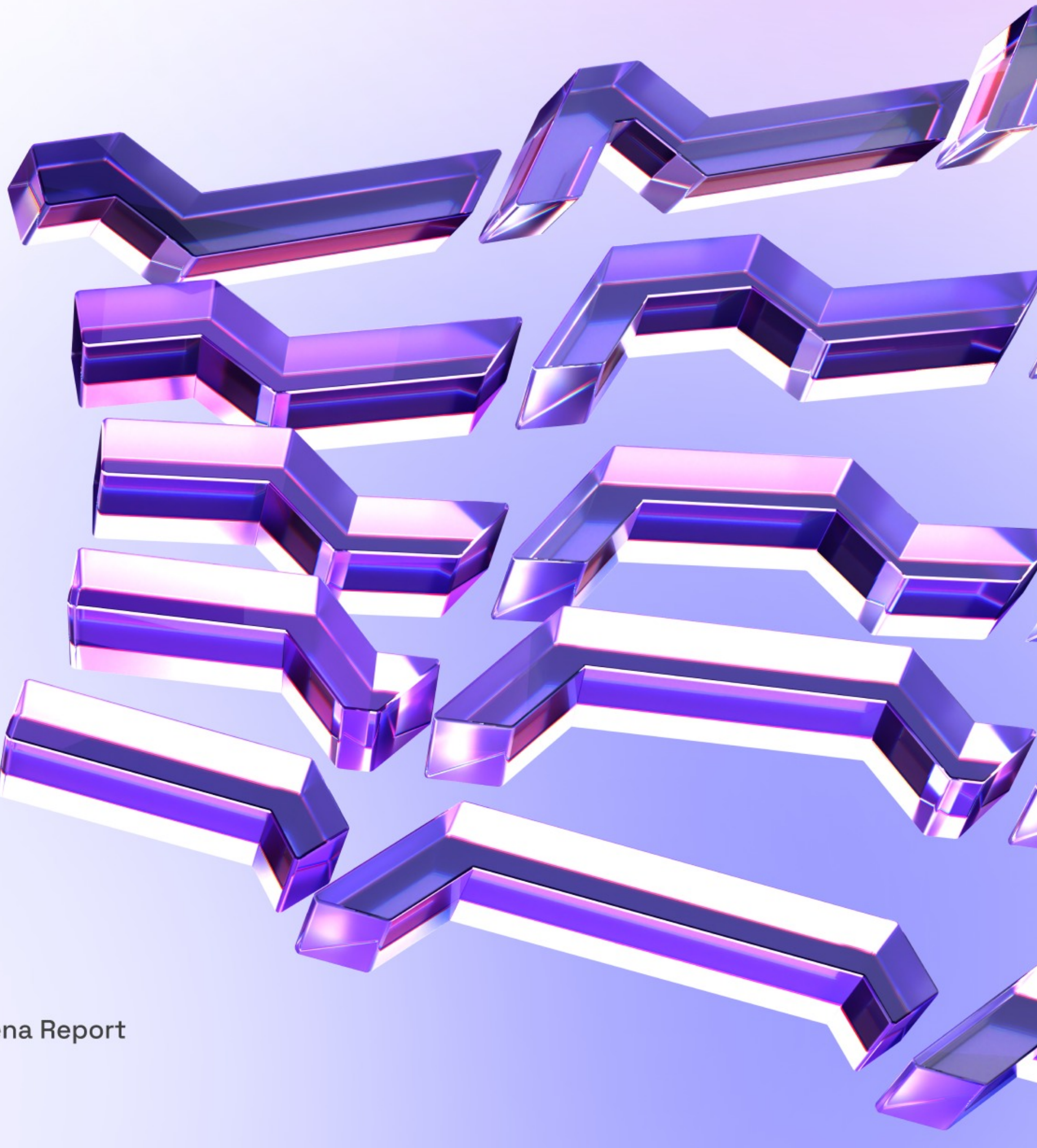
CJ42

potato-tipper-contract

October 24, 2025



Wake Arena Report

Contents

1. Document Revisions

1.0	Wake Arena Scan	October 24, 2025

2. Overview

This document presents findings identified by Wake Arena automated analysis.

2.1. Ackee Blockchain Security

Ackee Blockchain Security is an in-house team of security researchers performing security audits focusing on manual code reviews with extensive fuzz testing for Ethereum and Solana. Ackee is trusted by top-tier organizations in web3, securing protocols including Lido, Safe, and Axelar.

We develop open-source security and developer tooling Wake for Ethereum and Trident for Solana, supported by grants from Coinbase and the Solana Foundation. Wake and Trident help auditors in the manual review process to discover hardly recognizable edge-case vulnerabilities.

Our team teaches about blockchain security at the Czech Technical University in Prague, led by our co-founder and CEO, Josef Gattermayer, Ph.D. As the official educational partners of the Solana Foundation, we run the School of Solana and the Solana Auditors Bootcamp.

Ackee's mission is to build a stronger blockchain community by sharing our knowledge.

Ackee Blockchain a.s.

Rohanske nabrezi 717/4

186 00 Prague, Czech Republic

https://ackee.xyz

hello@ackee.xyz

2.2. Wake Arena

This report was generated using Wake Arena, an automated vulnerability analysis tool. Wake Arena utilizes Wake with additional detectors to perform comprehensive AI and static analysis.

To identify potential vulnerabilities and issues in smart contracts Wake framework utilizes:

- Code structure and patterns
- Control flow graph
- Data flow graph
- Common vulnerability patterns
- Contract interactions

The findings presented in this report are based on automated analysis optimized for precision, aiming for a low false-positive rate. The detection is not optimized for recall—it doesn't target finding all issues (which come at the cost of a high false-positive rate). This code review should be complemented with additional manual code review for a complete security assessment.

2.3. Disclaimer

We've put our best effort to find know vulnerabilities in the system, however automated findings shouldn't be considered as a complete list of all existing issues. The statements made in this document should not be interpreted as investment or legal advice, nor should its authors be held accountable for decisions made based on them.

2.4. Finding Classification

Each finding is classified by two independent ratings: `Impact` and `Confidence`.

Impact

Measuring the potential consequences of the issue on the system.

- **High** - Code that activates the issue will lead to undefined or catastrophic consequences for the system.
- **Medium** - Code that activates the issue will result in consequences of serious substance.
- **Low** - Code that activates the issue will have outcomes on the system that are either recoverable or don't jeopardize its regular functioning.
- **Warning** - The issue represents a potential security concern in the code structure or logic that could become problematic with code modifications.
- **Info** - The issue relates to code quality practices that may affect security. Examples include insufficient logging for critical operations or inconsistent error handling patterns.

Confidence

Indicating the probability that the identified issue is a valid security concern.

- **High** - The analysis has identified a pattern that strongly indicates the presence of the issue.
- **Medium** - Evidence suggests the issue exists, but manual verification is recommended.
- **Low** - Potential indicators of the issue have been detected, but there is a significant possibility of false positives.

3. Executive Summary

Revision 1.0

Audit Overview

This security audit evaluated the **CJ42/potato-tipper-contract** repository at commit `da1fddda23ad3647587cd75710aec5175cc2cb25`. The audit identified **10 AI-detected issues** and **0 static analysis issues** from Wake detections across the PotatoTipper protocol's smart contract implementation.

The PotatoTipper protocol implements an automated tipping mechanism on the LUKSO blockchain that rewards new followers with $POTATO tokens through an LSP1 Universal Receiver Delegate pattern. While the protocol demonstrates innovative use of LUKSO Standard Proposals (LSP1, LSP7, LSP26), the audit uncovered significant vulnerabilities affecting economic incentives, state management, and data parsing.

Critical findings include:

- **2 High-severity issues:** Incorrect memory parsing in `PotatoLib.getSettings()` (./src/PotatoLib.sol:29-30) that corrupts eligibility criteria, and a flash loan bypass vulnerability (./src/PotatoTipper.sol:287-290) that defeats anti-Sybil protections
- **6 Medium-severity issues:** State corruption from premature tip marking (./src/PotatoTipper.sol:292), griefing attacks via unfollow manipulation (./src/PotatoTipper.sol:239-241), front-running vulnerabilities, immutable external dependencies, memory overflow risks, and follower count manipulation
- **1 Low-severity issue:** Missing validation for zero-value tips
- **1 Warning:** Unsafe assembly memory access patterns

Overall Security Assessment:

The codebase exhibits a permissionless, autonomous design with no administrative controls, which aligns with decentralization principles but introduces challenges for incident response. The protocol's core architecture is sound, demonstrating proper integration with LUKSO standards and appropriate use of LSP1 delegate patterns. However, the identified vulnerabilities pose material risks to the protocol's economic model and operational reliability.

The most critical concern is the data parsing error in `PotatoLib.getSettings()` that reads from incorrect memory offsets, causing complete failure of the eligibility system. Combined with the flash loan bypass vulnerability, these issues fundamentally undermine the protocol's anti-Sybil mechanisms and tipping logic. The state management issues around premature tip marking create permanent exclusion scenarios where legitimate followers cannot receive tips even after transient failures are resolved.

The protocol's immutable external dependencies on the LSP26 Follower Registry and LSP7 $POTATO token contract create single points of failure with no recovery mechanisms. While the codebase includes defensive error handling through try-catch blocks for token transfers, several griefing vectors remain exploitable by malicious actors.

Positively, the contract demonstrates awareness of reentrancy risks through the checks-effects-interactions pattern attempt in `_sendTip()`, proper interface detection using ERC165, and appropriate event emissions for transparency. The three-mapping system for tracking follower states shows thoughtful design to prevent existing followers from gaming the system.

Recommendations: Address the critical data parsing bug immediately, implement time-locked or staking-based anti-Sybil measures to replace vulnerable balance checks, correct the state management flow to mark tips only after successful transfers, and consider adding emergency pause mechanisms despite the permissionless design philosophy.

Key Technical Findings

Critical Data Corruption in Settings Parser (HIGH)

The `PotatoLib.getSettings()` function at ./src/PotatoLib.sol:29-30 contains incorrect memory offset calculations that corrupt tipping eligibility criteria. The assembly code reads `minimumFollowers` from offset 34 instead of 64, extracting 32 bytes from the middle of `tipAmount`, and reads `minimumPotatoBalance` from offset 66, attempting to access memory beyond the 66-byte data boundary. This breaks the entire eligibility verification system, causing unpredictable behavior where followers may be incorrectly accepted or rejected regardless of their actual qualifications.

Flash Loan Bypass of Anti-Sybil Protection (HIGH)

The POTATO token balance check at ./src/PotatoTipper.sol:287-290 can be bypassed using flash loans, completely defeating the anti-Sybil mechanism. Attackers can temporarily borrow the required `minimumPotatoBalance` tokens, satisfy the eligibility check, receive tips, transfer the borrowed tokens to another Sybil account, and repeat the process within a single transaction. This removes the economic barrier intended to prevent Sybil attacks, allowing extraction of tips from multiple users with minimal capital cost.

State Corruption from Premature Tip Marking (MEDIUM)

The `_sendTip()` function at ./src/PotatoTipper.sol:292 sets `_tipped[msg.sender][follower] = true` before attempting the token transfer. When transfers fail due to insufficient balance, allowance exhaustion, or receiver rejections, followers remain permanently marked as tipped despite never receiving tokens. This creates irreversible state corruption with no retry mechanism, permanently excluding legitimate followers even after underlying issues are resolved.

Griefing Through Unfollow Manipulation (MEDIUM)

The logic at ./src/PotatoTipper.sol:239-241 incorrectly assumes that any unfollow notification without a prior follow must indicate an existing follower from before delegate installation. Attackers can trigger unfollow notifications for arbitrary addresses, causing them to be marked in `_wasFollowing`, which permanently prevents those addresses from receiving tips when they legitimately follow later.

Immutable External Dependency Risk (MEDIUM)

The hardcoded constant addresses for `_POTATO_TOKEN` and `_FOLLOWER_REGISTRY` at ./src/Constants.sol:26-29 create single points of failure. If either external contract is compromised, stops functioning, or undergoes breaking changes, all PotatoTipper instances become permanently unusable with no recovery path or migration mechanism.

Front-Running Attack Vector (MEDIUM)

The first-come-first-served tip distribution mechanism lacks protection against transaction ordering manipulation. MEV bots can monitor the mempool for follow transactions and submit identical follows with higher gas fees to steal tips intended for legitimate followers.

Assembly Memory Overflow (MEDIUM)

The offset calculation error at ./src/PotatoLib.sol:30 attempts to read 32 bytes from offset 66 in a 66-byte array, accessing uninitialized memory beyond the data boundary. This reads unpredictable values into `minimumPotatoBalance`, causing erratic eligibility checks.

Follower Count Manipulation (MEDIUM)

The follower count check at ./src/PotatoTipper.sol:282-285 uses raw counts without validating follower legitimacy. Attackers can create Sybil networks where bot accounts follow each other to meet `minimumFollowers` thresholds, then systematically collect tips without genuine social engagement.

4. Findings Summary

Summary of findings:

High	Medium	Low	Warning	Info	Total
2	6	1	1	0	10

Table 1. Findings Count by Impact

Findings in detail:

Finding title	Impact	Reported	Status
H1: Incorrect memory offset parsing in PotatoLib getSettings function	High	1.0	Reported
H2: Sybil Attack Amplification Through Flash Loan Balance Bypass	High	1.0	Reported
M1: Follower Count Manipulation Through Sybil Networks	Medium	1.0	Reported
M2: Permanent Tip Marking Despite Transfer Failure	Medium	1.0	Reported
M3: Front-running Vulnerability in Follow/Tip Mechanism	Medium	1.0	Reported
M4: Griefing Attack via Unfollow Manipulation	Medium	1.0	Reported
M5: Assembly Memory Read Overflow in Settings Parser	Medium	1.0	Reported

Finding title	Impact	Reported	Status
M6: Cascading Failure Through Immutable External Contract Dependencies	Medium	1.0	Reported
L1: Zero Tip Amount Not Validated	Low	1.0	Reported
W1: Missing bounds checking in PotatoLib assembly memory access	Warning	1.0	Reported

Table 2. Table of Findings

Report Revision 1.0

Findings

The following section presents the list of findings discovered in this revision.

For the complete list of all findings, Go back to Findings Summary

H1: Incorrect memory offset parsing in PotatoLib getSettings function

High impact finding

Impact:	High	Confidence:	High
Target:	PotatoLib.sol	Detection:	Wake AI

Description

The `getSettings` function in PotatoLib contains critical memory offset errors when parsing packed data. The function attempts to parse a tuple of `(uint256, uint16, uint256)` from memory but uses incorrect offsets for reading the values.

When data is encoded using `abi.encodePacked(uint256, uint16, uint256)`, the memory layout is:

- Bytes 0-31: Length of the bytes array (66 in this case)
- Bytes 32-63: First uint256 (tipAmount)
- Bytes 64-65: uint16 (minimumFollowers)
- Bytes 66-97: Second uint256 (minimumPotatoBalance)

However, the current implementation incorrectly reads:

Listing 1. Code snippet from ./src/PotatoLib.sol

```
assembly {
    tipAmount := mload(add(rawValue, 32))
    minimumFollowers := mload(add(rawValue, 34))
    minimumPotatoBalance := mload(add(rawValue, 66))
}
```

The critical errors are:

1. `minimumFollowers` is read from offset 34 instead of 64, causing it to read 32 bytes from the middle of the first uint256
2. `minimumPotatoBalance` is read from offset 66, which attempts to read 32 bytes starting from byte 66, extending beyond the 66-byte data boundary

This results in completely incorrect values being parsed for both `minimumFollowers` and `minimumPotatoBalance`, breaking the entire tipping eligibility system.

Go back to Findings Summary

H2: Sybil Attack Amplification Through Flash Loan Balance Bypass

High impact finding

Impact:	High	Confidence:	High
Target:	PotatoTipper.sol	Detection:	Wake AI

Description

The PotatoTipper contract implements an anti-Sybil mechanism by requiring followers to hold a minimum balance of POTATO tokens before they can receive tips. However, this protection can be completely bypassed using flash loans, allowing attackers to extract tips from multiple users with minimal capital.

The vulnerability lies in the balance check mechanism that only verifies the current balance without considering the source of funds:

Listing 2. Code snippet from ./src/PotatoTipper.sol

```
287 // CHECK if the followers has the minimum amount of $POTATO tokens required
288 if (_POTATO_TOKEN.balanceOf(follower) < minimumPotatoBalance) {
289     return unicode"▯ Not eligible for tip: minimum ▯ balance required not
    met";
290 }
```

The protocol checks the follower's POTATO token balance at line 288, but this check can be satisfied with temporarily borrowed funds. Since the entire follow-tip-repay sequence can occur within a single transaction, the same POTATO tokens can be recycled across multiple Sybil accounts.

Go back to Findings Summary

M1: Follower Count Manipulation Through Sybil Networks

Medium impact finding

Impact:	Medium	Confidence:	High
Target:	PotatoTipper.sol	Detection:	Wake AI

Description

The protocol uses raw follower counts as an eligibility criterion without validating the legitimacy or quality of followers. This allows attackers to manipulate the system through Sybil attacks and artificial follower inflation.

Listing 3. Code snippet from ./src/PotatoTipper.sol

```
// CHECK the follower has the minimum number of followers required
if (_FOLLOWER_REGISTRY.followerCount(follower) < minimumFollowers) {
    return unicode"□ Not eligible for tip: minimum follower required not met";
}
```

The vulnerability stems from treating all followers equally without considering:

- Whether followers are genuine users or bot accounts.
- The age or activity of follower accounts.
- Circular following patterns between colluding accounts.
- Self-following if permitted by the LSP26 Registry.

Attackers can create networks of Universal Profiles that follow each other to meet any follower count requirement without genuine social engagement.

Go back to Findings Summary

M2: Permanent Tip Marking Despite Transfer Failure

Medium impact finding

Impact:	Medium	Confidence:	High
Target:	PotatoTipper.sol	Detection:	Wake AI

Description

The `_sendTip` function violates the checks-effects-interactions pattern by marking followers as tipped before attempting the actual token transfer. When transfers fail, followers remain permanently marked as tipped despite never receiving tokens, creating irreversible state corruption.

Listing 4. Code snippet from ./src/PotatoTipper.sol

```
292 _tipped[msg.sender][follower] = true;
293
294 // Transfer □ $POTATO □ tokens as tip to the new follower
295 // Return a success or error message that can be decoded from the
    `UniversalReceiver` event
296 try _POTATO_TOKEN.transfer({
297     // ... transfer parameters ...
298 }) {
299     emit TipSent({from: msg.sender, to: follower, amount: tipAmount});
300     return abi.encodePacked(unicode"□ Successfully tipped □ to new follower:
    ", follower.toHexString());
301 } catch (bytes memory errorData) {
302     emit TipFailed({from: msg.sender, to: follower, amount: tipAmount,
    errorData: errorData});
303     return unicode"□ Failed tipping □. LSP7 transfer reverted";
304 }
```

The state change at line 292 happens before the external call at line 296. If the transfer fails for any reason:

- Insufficient user balance;

- Insufficient operator allowance;
- Receiver’s universalReceiver reverting; or
- Gas issues or other EVM errors.

The follower is still marked as tipped and can never receive tips again, even after the issue is resolved.

Go back to Findings Summary

M3: Front-running Vulnerability in Follow/Tip Mechanism

Medium impact finding

Impact:	Medium	Confidence:	High
Target:	PotatoTipper.sol	Detection:	Wake AI

Description

The PotatoTipper protocol is vulnerable to front-running attacks where MEV bots can monitor the mempool for follow transactions and front-run legitimate followers to steal tips. The vulnerability stems from the first-come-first-served tip distribution mechanism without any protection against transaction ordering manipulation.

Listing 5. Code snippet from ./src/PotatoTipper.sol

```
function _onFollow(address follower) internal returns (bytes memory message) {
    bool isFollowing = _FOLLOWER_REGISTRY.isFollowing(follower, msg.sender);

    // CHECK to ensure this came from a legitimate notification callback from the LSP26 Registry
    if (!isFollowing) return unicode"□ Not a legitimate follow";

    // ... eligibility checks ...

    return _sendTip(follower);
}
```

The protocol processes follow notifications in the order they are received from the LSP26 Registry. Since blockchain transaction ordering can be manipulated through gas price bidding, attackers can:

1. Monitor the mempool for follow transactions to users with tipping enabled.

2. Submit the same follow transaction with higher gas fees.
3. Have their transaction processed first by miners/validators.
4. Receive the tip intended for the legitimate follower.

The lack of any commitment scheme, time delays, or proof of genuine intent makes this attack highly profitable for MEV bots.

Go back to Findings Summary

M4: Griefing Attack via Unfollow Manipulation

Medium impact finding

Impact:	Medium	Confidence:	High
Target:	PotatoTipper.sol	Detection:	Wake AI

Description

The `_onUnfollow` function contains flawed logic that allows attackers to prevent legitimate users from receiving tips by manipulating the unfollow mechanism. The vulnerability stems from incorrect assumptions about follower state tracking.

Listing 6. Code snippet from ./src/PotatoTipper.sol

```
234 // If `address_` never followed the user after it connected the Potato Tipper,
235 // this proves that `address_` was an existing follower at install time BPT.
236 //
237 // Handle cases of existing followers unfollowing -> then re-following to try to get a tip
238 // Lock them out and prevent from tipping them if they try to re-follow.
239 if (!_hasFollowedSinceDelegate[msg.sender][address_]) {
240     _wasFollowing[msg.sender][address_] = true;
241     return "";
242 }
```

The logic at lines 239-240 assumes that if an address unfollows without having followed after delegate installation, it must have been an existing follower. However, this assumption is flawed because:

1. An address can trigger an unfollow notification without ever having followed.
2. The check only verifies `!_hasFollowedSinceDelegate`, not actual follow status.

3. This incorrectly marks addresses as ”existing followers” in `_wasFollowing`.

When these addresses later legitimately follow, the check at line 212 in `_onFollow` prevents them from receiving tips:

Listing 7. Code snippet from ./src/PotatoTipper.sol

```
212 if (_wasFollowing[msg.sender][follower]) {
213     return unicode"⍰⍰ Follower followed before. Not eligible for a tip";
214 }
```

Go back to Findings Summary

M5: Assembly Memory Read Overflow in Settings Parser

Medium impact finding

Impact:	Medium	Confidence:	High
Target:	PotatoLib.sol	Detection:	Wake AI

Description

The `getSettings` function in PotatoLib contains a critical memory overflow bug due to incorrect offset calculation in assembly code. The function reads beyond the boundaries of the input array, accessing uninitialized memory.

Listing 8. Code snippet from ./src/PotatoLib.sol

```
27 assembly {
28     tipAmount := mload(add(rawValue, 32))
29     minimumFollowers := mload(add(rawValue, 34))
30     minimumPotatoBalance := mload(add(rawValue, 66))
31 }
```

The function expects a 66-byte input with the following layout:

- Bytes 0-31: `tipAmount` (uint256).
- Bytes 32-33: `minimumFollowers` (uint16).
- Bytes 34-65: `minimumPotatoBalance` (uint256).

However, the assembly code at line 30 uses offset 66, which attempts to read bytes 66-97. Since the input is only 66 bytes long (validated in PotatoTipper.sol at line 266), this reads 32 bytes beyond the array boundary into uninitialized memory.

The `mload` operation always reads 32 bytes, so:

- `mload(add(rawValue, 32))` correctly reads bytes 0-31 for `tipAmount`.
- `mload(add(rawValue, 34))` reads bytes 2-33, extracting the uint16 from the packed data.
- `mload(add(rawValue, 66))` incorrectly reads bytes 34-65 and 32 bytes of garbage data.

The correct offset for `minimumPotatoBalance` should be 50 to read bytes 18-49 (34-65 in the array).

Go back to Findings Summary

M6: Cascading Failure Through Immutable External Contract Dependencies

Medium impact finding

Impact:	Medium	Confidence:	Medium
Target:	Constants.sol	Detection:	Wake AI

Description

The PotatoTipper contract has hardcoded, immutable dependencies on external contracts that create a single point of failure for the entire tipping ecosystem. These dependencies cannot be updated if the external contracts fail, are compromised, or undergo breaking changes.

Listing 9. Code snippet from ./src/Constants.sol

```solidity
// Address of the $POTATO Token contract deployed on LUKSO Mainnet.
ILSP7 constant _POTATO_TOKEN =
ILSP7(0x80D898C5A3A0B118a0c8C8aDcdBB260FC687F1ce);

// Address of the Follower Registry deployed on LUKSO Mainnet based on the
LSP26 standard.
ILSP26 constant _FOLLOWER_REGISTRY =
ILSP26(0xf01103E5a9909Fc0DBe8166dA7085e0285daDDcA);
```

The contract critically depends on these external contracts for core functionality:

- **_FOLLOWER_REGISTRY**: Used to verify follow relationships and follower counts.
- **_POTATO_TOKEN**: Used for balance checks, allowances, and token transfers.

Using `constant` declarations makes these addresses permanently immutable at the bytecode level. If either external contract:

- Becomes compromised or malicious;
- Stops functioning due to bugs or attacks;
- Undergoes breaking API changes; or
- Is paused or self-destructs.

Then all deployed PotatoTipper instances become permanently unusable with no recovery path. Users' operator allowances could remain locked with no way to interact with the contract.

Go back to Findings Summary

L1: Zero Tip Amount Not Validated

Low impact finding

Impact:	Low	Confidence:	High
Target:	PotatoTipper.sol	Detection:	Wake AI

Description

The _sendTip function does not validate that tipAmount is greater than zero, allowing users to configure zero-value tips that mark followers as tipped without transferring tokens.

Listing 10. Code snippet from ./src/PotatoTipper.sol

```solidity
(uint256 tipAmount, uint16 minimumFollowers, uint256 minimumPotatoBalance) =
    settingsValue.getSettings();
```

After retrieving tipAmount from user settings, the contract proceeds to mark followers as tipped and attempts transfers without checking if tipAmount > 0. This creates misleading state and events.

[Go back to Findings Summary]

W1: Missing bounds checking in PotatoLib assembly memory access

Impact:	Warning	Confidence:	High
Target:	PotatoLib.sol	Detection:	Wake AI

Description

The `getSettings` function in PotatoLib uses assembly to read memory at fixed offsets without internally validating that the input `rawValue` has sufficient length. This creates a potential for out-of-bounds memory access if the function is called with incorrectly sized data.

Listing 11. Code snippet from ./src/PotatoLib.sol

```
function getSettings(bytes memory rawValue) internal pure returns (uint256,
uint16, uint256) {
    uint256 tipAmount;
    uint16 minimumFollowers;
    uint256 minimumPotatoBalance;

    assembly {
        tipAmount := mload(add(rawValue, 32))
        minimumFollowers := mload(add(rawValue, 34))
        minimumPotatoBalance := mload(add(rawValue, 66))
    }

    return (tipAmount, minimumFollowers, minimumPotatoBalance);
}
```

The function attempts to read:

- 32 bytes from offset 32 (for tipAmount).
- 32 bytes from offset 34 (for minimumFollowers).
- 32 bytes from offset 66 (for minimumPotatoBalance).

If `rawValue` is shorter than 98 bytes (66 bytes of data + 32 bytes for length prefix), the assembly code will read beyond the allocated memory, potentially accessing unrelated data or causing undefined behavior.

Currently, the only caller (PotatoTipper._sendTip) does validate the length:

Listing 12. Code snippet from ./src/PotatoTipper.sol

```
if (settingsValue.length != 66) {
    return unicode"□ Invalid settings. Must be encoded as (uint256,uint16,uint256)";
}
```

However, this creates tight coupling and violates defensive programming principles. The library should be self-contained and safe to use regardless of caller validation.

Go back to Findings Summary

Appendix A: How to cite

Please cite this document as:

Ackee Blockchain Security, Wake Arena Report | CJ42: potato-tipper-contract, October 24, 2025.

Thank You

Ackee Blockchain a.s.

Rohanske nabrezi 717/4
186 00 Prague
Czech Republic

hello@ackee.xyz