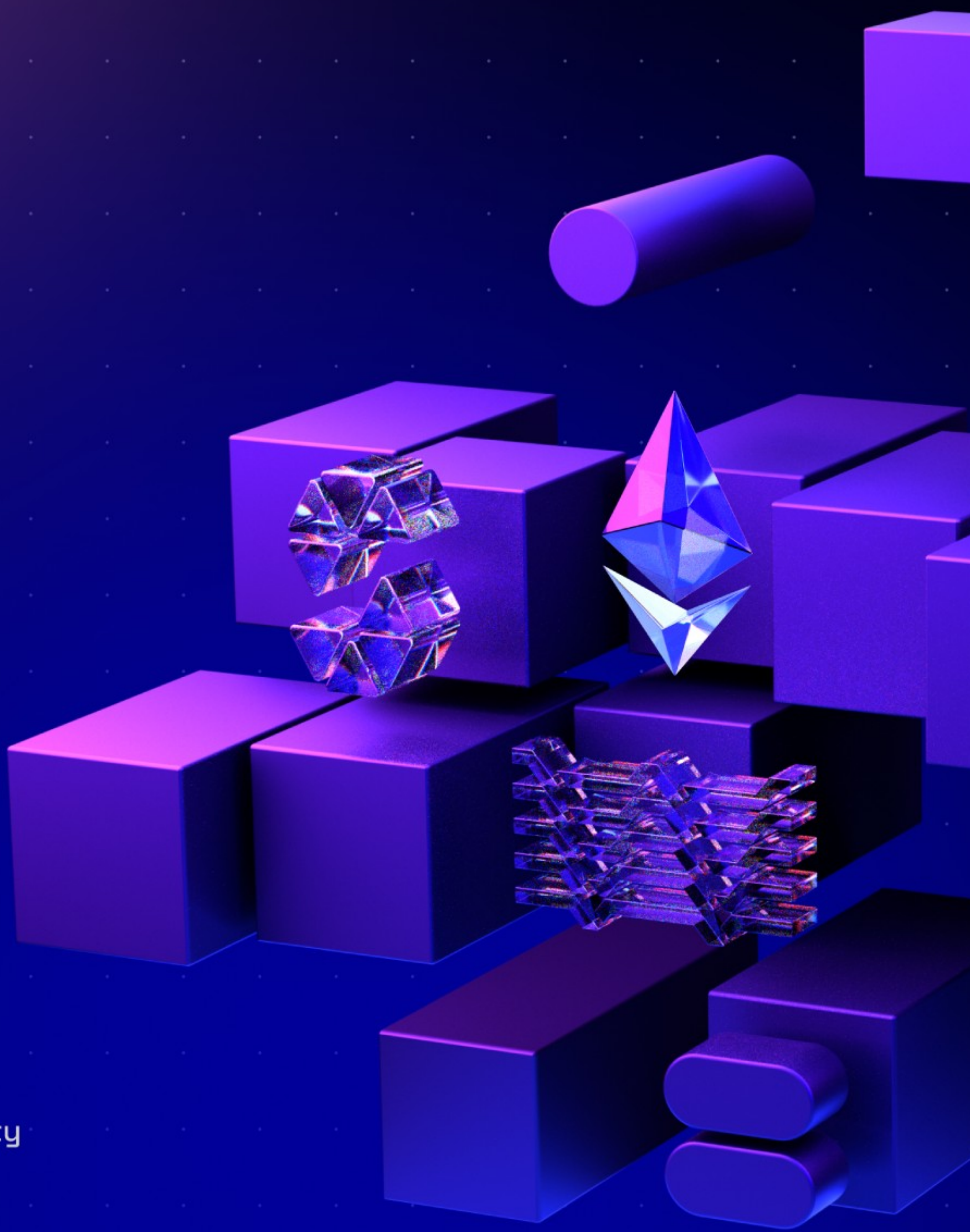
Safe

Smart Account

28.5.2025



Ackee Blockchain Security

Contents

1. Document Revisions

1.0-draft	Draft Report	20.01.2025
1.0	Final Report	27.05.2025
1.1	Final Report	28.05.2025

2. Overview

This document presents our findings in reviewed contracts.

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. Audit Methodology

1. **Verification of technical specification**

   The audit scope is confirmed with the client, and auditors are onboarded to the project. Provided documentation is reviewed and compared to the audited system.

2. **Tool-based analysis**

   A deep check with Solidity static analysis tool Wake in companion with Solidity (Wake) extension is performed, flagging potential vulnerabilities for further analysis early in the process.

3. **Manual code review**

   Auditors manually check the code line by line, identifying vulnerabilities and code quality issues. The main focus is on recognizing potential edge cases and project-specific risks.

4. **Local deployment and hacking**

   Contracts are deployed in a local Wake environment, where targeted attempts to exploit vulnerabilities are made. The contracts' resilience against various attack vectors is evaluated.

5. **Unit and fuzz testing**

   Unit tests are run to verify expected system behavior. Additional unit or fuzz tests may be written using Wake framework if any coverage gaps are identified. The goal is to verify the system's stability under real-world conditions and ensure robustness against both expected and unexpected inputs.

2.3. Finding Classification

A *Severity* rating of each finding is determined as a synthesis of two sub-ratings: *Impact* and *Likelihood*. It ranges from *Informational* to *Critical*.

If we have found a scenario in which an issue is exploitable, it will be assigned an impact rating of *High*, *Medium*, or *Low*, based on the direness of the consequences it has on the system. If we haven't found a way, or the issue is only exploitable given a change in *configuration* (system settings or parameters, such as deployment scripts, compiler configurations, using multi-signature wallets for owners, etc.) or given a change in the codebase, then it will be assigned an impact rating of *Warning* or *Info*.

Low to *High* impact issues also have a *Likelihood*, which measures the probability of exploitability during runtime.

The full definitions are as follows:

Severity

		Likelihood			
		High	**Medium**	**Low**	**N/A**
Impact	**High**	Critical	High	Medium	-
	Medium	High	Medium	Low	-
	Low	Medium	Low	Low	-
	Warning	-	-	-	Warning
	Info	-	-	-	Info

Table 1. Severity of findings

Impact

- **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 cannot be exploited given the current code and/or *configuration*, but could be a security vulnerability if these were to change slightly. If we haven't found a way to exploit the issue given the time constraints, it might be marked as a "Warning" or higher, based on our best estimate of whether it is currently exploitable.
- **Info** - The issue is on the borderline between code quality and security. Examples include insufficient logging for critical operations. Another example is that the issue would be security-related if code or *configuration* was to change.

Likelihood

- **High** - The issue is exploitable by virtually anyone under virtually any circumstance.
- **Medium** - Exploiting the issue currently requires non-trivial preconditions.
- **Low** - Exploiting the issue requires strict preconditions.

2.4. Review Team

The following table lists all contributors to this report. For authors of the specific revision, see the “Revision team” section in the respective “Report revision” chapter.

Member’s Name	Position
Michal Převrátil	Lead Auditor
Dmytro Khimchenko	Auditor
Naoki Yoshida	Auditor
Josef Gattermayer, Ph.D.	Audit Supervisor

2.5. Disclaimer

We’ve put our best effort to find all vulnerabilities in the system, however our 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.

3. Executive Summary

Safe is a multi-signature smart contract wallet designed for collective management of digital assets. The wallet requires a predefined threshold of owner signatures before any transaction can be executed. To enhance functionality, Safe supports extensions through modules and fallback handlers.

Revision 1.0

Safe engaged Ackee Blockchain Security to perform a security review of Safe Smart Account with a total time donation of 20 engineering days in a period between April 14 and May 12, 2025, with Michal Převrátil as the lead auditor.

6 engineering days were dedicated to manually-guided differential fuzzing using the Wake testing framework.

The audit was performed on the commit `b115c4c`[1] in the safe-smart-account repository. The scope of the audit included all Solidity files in the `contracts` directory, excluding `contracts/examples` and `contracts/test`.

A kick-off meeting with Safe was conducted at the beginning of the engagement to discuss technical details and main goals for the audit, which enhanced the efficiency and effectiveness of the review process.

`d89d156`[2] was initially used as the target commit, but it was later updated to include changes in the `CompatibilityFallbackHandler` contract. An additional 0.5 engineering day was used to review the changes.

We began our audit with a manual review of the codebase in parallel with manually-guided fuzzing using the Wake testing framework. For static analysis, we utilized Wake vulnerability and code quality detectors.

During the review, we focused on ensuring:

- basic concepts of Safe (such as owners management and signature checking) remain implemented correctly;
- refactored assembly blocks labeled as memory-safe are indeed memory-safe;
- reentrancy and front-running attacks are not possible;
- standards such as ERC-165, ERC-1271 and EIP-712 are implemented correctly;
- integer overflows and underflows do not lead to security vulnerabilities;
- the contract is compatible with ERC-4337 smart accounts;
- backward compatibility is fully achieved through the `CompatibilityFallbackHandler` contract; and
- common issues such as data validation are not present.

Our review resulted in 19 findings, ranging from Info to Medium severity. The most severe finding M1 was discovered through manually-guided fuzzing. The issue revealed a possibility of a front-running attack where an attacker could deploy a new Safe on behalf of a user without executing the intended callback.

The M1 issue was found to be present in already deployed contracts across all supported chains. Ackee Blockchain Security initiated an immediate responsible disclosure over a secure channel as soon as the finding was identified to mitigate any possible risk. The validity of the finding was promptly acknowledged by the Safe team.

The code is well documented, and possible caveats and security considerations are explained. There is room for improvements in terms of user experience (W1, W7, I4, I5). The reviewed version of Safe is incompatible with EIP-7702 smart accounts.

Ackee Blockchain Security recommends Safe:

- document that the Safe account is not fully compatible with EIP-7702;
- clearly mark files under `contracts/examples` as non-production code;
- document functionalities that are not supported by `CompatibilityFallbackHandler`; and
- address all identified issues.

See Report Revision 1.0 for the system overview and trust model.

Revision 1.1

Safe engaged Ackee Blockchain Security to perform a fix review of the findings from the previous revision.

The review was performed between May 20 and May 27, 2025 on the commit `5d26505`[3] in the safe-smart-account repository.

2 findings were acknowledged and the remaining 17 findings were fixed.

No new findings were discovered.

No changes were introduced except for the finding fixes.

[1] full commit hash: `b115c4c5fe23dca6aefeeccc73d312ddd23322c2`, link to commit

[2] full commit hash: `d89d1569fdacdd1d8a2788c2d5db50e873d30560`, link to commit

[3] full commit hash: `5d26505388e9ee014ad9ac497aa48e3a13426eb1`, link to commit

4. Findings Summary

The following section summarizes findings we identified during our review. Unless overridden for purposes of readability, each finding contains:

- *Description*
- *Exploit scenario* (if severity is low or higher)
- *Recommendation*
- *Fix* (if applicable).

Summary of findings:

Critical	High	Medium	Low	Warning	Info	Total
0	0	1	2	7	9	19

Table 2. Findings Count by Severity

Findings in detail:

Finding title	Severity	Reported	Status
M1: Front-running attack can bypass callback execution during Safe deployment	Medium	1.0	Fixed
L1: `CompatibilityFallbackHandler` does not provide full compatibility	Low	1.0	Fixed
L2: Strict calldata check on `masterCopy` call	Low	1.0	Fixed
W1: Misleading event emissions	Warning	1.0	Fixed

Finding title	Severity	Reported	Status
W2: Use of precomputed `msg.data`	Warning	1.0	Fixed
W3: Scratch space assumed zeroed out	Warning	1.0	Fixed
W4: Safe `setup` may emit outdated information	Warning	1.0	Fixed
W5: `onlyNonceZero` check can be bypassed	Warning	1.0	Fixed
W6: Locked tokens possibility	Warning	1.0	Fixed
W7: `ProxyCreationL2` nonce value is not user given argument	Warning	1.0	Fixed
I1: Documentation issues	Info	1.0	Fixed
I2: Unnecessary typecasts to `payable`	Info	1.0	Fixed
I3: Code optimizations	Info	1.0	Fixed
I4: Factory `initializer` error not propagated	Info	1.0	Fixed
I5: No view function for `FallbackManager` handler address	Info	1.0	Acknowledged
I6: `SafeStorage` can be defined as abstract	Info	1.0	Fixed

Finding title	Severity	Reported	Status
I7: Missing L2-specific variant of `createChainSpecificProxyWithNonce`	Info	1.0	Fixed
I8: Interface type used for parameter that accepts zero address	Info	1.0	Fixed
I9: `ChangedThreshold` event is emitted unconditionally	Info	1.0	Acknowledged

Table 3. Table of Findings

Report Revision 1.0

Revision Team

Member's Name	Position
Michal Převrátil	Lead Auditor
Dmytro Khimchenko	Auditor
Naoki Yoshida	Auditor
Josef Gattermayer, Ph.D.	Audit Supervisor

System Overview

Safe is a multi-signature smart account that allows users to manage their assets and execute transactions on Ethereum. A transaction guard can be attached to the Safe to restrict the types of transactions that can be executed. Any number of modules can be attached to the Safe. Modules allow execution of arbitrary transactions from the Safe. Module guards can be used to limit the types of transactions that can be executed through a module.

A Safe may be extended with a fallback handler. The fallback handler is a contract that can be used to handle calls to the Safe that do not match any of the signatures of the implemented functions. A compatibility fallback handler exists to provide compatibility with older versions of the Safe contract. The extensible fallback handler is a fallback handler that handles receiving tokens, validating ERC-1271 signatures, and can be extended with additional handlers based on their ERC-165 interface.

Trust Model

Owners of a Safe have full control over the Safe. Any attached modules must be trusted, as they can execute arbitrary transactions from the Safe.

Attached fallback handler must be trusted as it can verify ERC-1271 signatures on behalf of the Safe.

Safe proxy factory may be trusted to provide a front-running protection when used correctly. I.e. it is guaranteed that a precomputed Safe address will belong to the intended owners as long as the Safe setup is performed as a initialization step of the proxy deployment.

Fuzzing

A manually-guided differential stateful fuzz test was developed during the review to test the correctness and robustness of the system. The fuzz test employs fork testing technique to test the system with external contracts exactly as they are deployed in the deployment environment. This is crucial to detect any potential integration issues. The differential fuzz test keeps its own Python state according to the system's specification. Assertions are used to verify the Python state against the on-chain state in contracts.

The list of all implemented execution flows and invariants are available in Appendix B.

The fuzz test simulates the whole system and makes strict assertions about the behaviour of the contracts. The fuzz test focuses on:

- Default operation on `Safe` smart account
- `GuardManager` contract behaviour
- `FallbackManager` contract behaviour and `ExtensibleFallbackHandler` and signature operations

The most severe findings M1, W6, and W7 were discovered using fuzz testing in Wake testing framework.

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

M1: Front-running attack can bypass callback execution during Safe deployment

Medium severity issue

Impact:	Medium	Likelihood:	Medium
Target:	SafeProxyFactory.sol	Type:	Front-running

Description

The `createProxyWithCallback` function can be front-run by calling `createProxyWithNonce`, resulting in a successful Safe account creation but bypassing the callback function execution. This vulnerability exists because both functions can generate the same deployment address using the same nonce value.

Listing 1. Excerpt from [SafeProxyFactory.createProxyWithCallback]()

```
uint256 saltNonceWithCallback = uint256(
keccak256(abi.encodePacked(saltNonce, callback)));
proxy = createProxyWithNonce(_singleton, initializer,
saltNonceWithCallback);
```

The front-running attacker can calculate the nonce using the following formula:

```
uint256 nonce = uint256(keccak256(abi.encodePacked(saltNonce, callback)));
```

This issue was found during a manually-guided fuzzing session.

Exploit scenario

1. Alice, a legitimate user, submits a transaction to `createProxyWithCallback` to set up a Safe account with initialization logic in the callback function.

2. Bob, an attacker, observes Alice's transaction in the mempool and calculates the same nonce value that will be used.
3. Bob front-runs Alice's transaction by calling `createProxyWithNonce` with the calculated nonce value.
4. Bob's transaction succeeds in deploying the Safe account, causing Alice's subsequent transaction to fail.
5. As a result, while the Safe account is created, the callback function containing critical initialization logic is never executed. This can be particularly severe if the callback was meant to set up security parameters.

Recommendation

Modify the `createProxyWithCallback` function to use a unique nonce derivation method that cannot be reproduced by `createProxyWithNonce`.

Fix 1.1

The issue was fixed by removing the `createProxyWithNonce` function as it was not actively used.

[Go back to Findings Summary]

L1: `CompatibilityFallbackHandler` does not provide full compatibility

Low severity issue

Impact:	Low	Likelihood:	Low
Target:	CompatibilityFallbackHandler.sol	Type:	N/A

Description

The `CompatibilityFallbackHandler` contract provides compatibility between the latest Safe contract and older versions. Since a fallback handler cannot execute transactions on behalf of the Safe, the compatibility is limited to read-only (non-state-changing) calls.

However, the `CompatibilityFallbackHandler` does not implement the `encodeTransactionData(address, uint256, bytes, Enum.Operation, uint256, uint256, uint256, address, address, uint256)` function.

Exploit scenario

Alice, a developer, creates a contract that integrates with a Safe contract and relies on the `encodeTransactionData` function. Bob, the Safe contract owner, upgrades the Safe contract to the latest version with the `CompatibilityFallbackHandler` contract attached.

The integration fails because the `CompatibilityFallbackHandler` contract does not implement the required `encodeTransactionData` function.

Recommendation

Implement the `encodeTransactionData` function in the `CompatibilityFallbackHandler` contract to maintain compatibility with older

Safe contract implementations.

Fix 1.1

The `encodeTransactionData` function was implemented in the `CompatibilityFallbackHandler` contract.

Go back to Findings Summary

L2: Strict calldata check on `masterCopy` call

Low severity issue

Impact:	Low	Likelihood:	Low
Target:	SafeProxy.sol	Type:	N/A

Description

The `SafeProxy` contract implements a `fallback` function that forwards most function calls to the singleton contract. The `masterCopy` function call is handled directly in the `fallback` implementation.

Listing 2. Excerpt from [SafeProxy.fallback]()

```
// 0xa619486e == bytes4(keccak256("masterCopy()")). The value is right padded to 32-bytes with 0s
if eq(calldataload(0), 0xa619486e00000000000000000000000000000000000000000000000000000000) {
```

The code compares the first 32 bytes of calldata with the `masterCopy` function selector extended by 28 bytes of zero.

The implemented behavior is inconsistent with the Solidity function delegation mechanism. Solidity delegates functions based on the selector bytes only. The remaining bytes are decoded as arguments, and any additional bytes are ignored.

Exploit scenario

In Solidity, the `masterCopy()` function may also be called with the `0xa619486e0a` bytes payload. However, the `SafeProxy` contract does not handle this case and delegates the call to the singleton contract.

The call may then be handled by a fallback handler, potentially returning an incorrect value for the `masterCopy` address.

Recommendation

Mask the `calldataload(0)` value to extract only the first 4 bytes and use the masked value for comparison.

Fix 1.1

The issue was fixed by logically shifting the `calldataload(0)` value right by 224 bits before comparison and adjusting the compared constant accordingly.

[Go back to Findings Summary]

W1: Misleading event emissions

Impact:	Warning	Likelihood:	N/A
Target:	ExtensibleBase.sol, SignatureVerifierMuxer.sol	Type:	Logic error

Description

The codebase contains two instances where updating a storage value with an identical value results in misleading event emissions.

Listing 3. Excerpt from [ExtensibleBase._setSafeMethod]()

```
if (address(newHandler) == address(0) && address(oldHandler) != address(0)) {
    delete safeMethod[selector];
    emit RemovedSafeMethod(safe, selector);
} else {
    safeMethod[selector] = newMethod;
    if (address(oldHandler) == address(0)) {
        emit AddedSafeMethod(safe, selector, newMethod);
    } else {
        emit ChangedSafeMethod(safe, selector, oldMethod, newMethod);
    }
}
```

The `_setSafeMethod` function emits an `AddedSafeMethod` event when both `oldHandler` and `newHandler` are the zero address. Additionally, when both addresses are identical but non-zero, a `ChangedSafeMethod` event is emitted despite no actual change occurring.

Listing 4. Excerpt from [SignatureVerifierMuxer.setDomainVerifier]()

```
if (address(newVerifier) == address(0) && address(oldVerifier) != address(0)) {
    delete domainVerifiers[safe][domainSeparator];
    emit RemovedDomainVerifier(safe, domainSeparator);
} else {
    domainVerifiers[safe][domainSeparator] = newVerifier;
    if (address(oldVerifier) == address(0)) {
```

```
        emit AddedDomainVerifier(safe, domainSeparator, newVerifier);
    } else {
        emit ChangedDomainVerifier(safe, domainSeparator, oldVerifier,
newVerifier);
    }
}
```

Similarly, in the `setDomainVerifier` function:

- an `AddedDomainVerifier` event is emitted when both `oldVerifier` and `newVerifier` are the zero address; and
- a `ChangedDomainVerifier` event is emitted when both verifiers are identical but non-zero.

Recommendation

Do not emit the `AddedSafeMethod` and `AddedDomainVerifier` events when both addresses are zero. Consider not emitting any events when the new value matches the existing value.

Fix 1.1

The code was simplified to emit only the `Changed*` events regardless of whether the state variable was actually changed. This approach maintains consistency across the codebase.

[Go back to Findings Summary]

W2: Use of precomputed `msg.data`

Impact:	Warning	Likelihood:	N/A
Target:	SafeToL2Migration.sol	Type:	N/A

Description

The `SafeToL2Migration` contract handles the upgrade process from an older version of the `Safe` contract to an L2 version.

The contract emits a `SafeMultiSigTransaction` event to maintain complete consistency with the `SafeL2` contract during the migration process.

The transaction data is precomputed based on the assumption that the migration function is called without additional data.

Listing 5. Excerpt from SafeToL2Migration.migrateToL2

```
// 0xef2624ae - bytes4(keccak256("migrateToL2(address)"))
bytes memory functionData = abi.encodeWithSelector(0xef2624ae, l2Singleton);
migrate(l2Singleton, functionData);
```

Listing 6. Excerpt from SafeToL2Migration.migrateFromV111

```
// 0xd9a20812 - bytes4(keccak256("migrateFromV111(address,address)"))
bytes memory functionData = abi.encodeWithSelector(0xd9a20812, l2Singleton,
    fallbackHandler);
migrate(l2Singleton, functionData);
```

Exploit scenario

Safe owners call one of the migration functions with extra calldata bytes. As a result, the transaction data signed by the Safe owner differ from the data emitted in the `migrate` function.

Listing 7. Excerpt from [SafeToL2Migration]

```
function migrate(address l2Singleton, bytes memory functionData) private {
    singleton = l2Singleton;

    // Encode nonce, sender, threshold
    bytes memory additionalInfo = abi.encode(0, msg.sender, threshold);

    // Simulate a L2 transaction so Safe Tx Service indexer picks up the Safe
    emit SafeMultiSigTransaction(
        MIGRATION_SINGLETON,
        0,
        functionData,
        Enum.Operation.DelegateCall,
        0,
        0,
        0,
        address(0),
        payable(address(0)),
        "", // We cannot detect signatures
        additionalInfo
    );
```

Recommendation

Use the `msg.data` parameter instead of the precomputed values to ensure the correctness of emitted events under all circumstances.

Note that this approach will still not achieve 100% accuracy when the migration function is called through an intermediary contract.

Fix 1.1

The precomputed values were removed in favor of `msg.data`. Additionally, it was documented that `msg.data` may still be inaccurate and the current implementation is a best effort.

[Go back to Findings Summary]

W3: Scratch space assumed zeroed out

Impact:	Warning	Likelihood:	N/A
Target:	SafeProxy.sol	Type:	N/A

Description

The `SafeProxy` contract implements a `fallback` function that forwards most function calls to the singleton contract. The `masterCopy` function call is handled directly in the `fallback` implementation.

Listing 8. Excerpt from [SafeProxy.fallback]()

```
// We mask the singleton address when handling the `masterCopy()` call to ensure that it is correctly
// ABI-encoded. We do this by shifting the address left by 96 bits (or 12 bytes) and then storing it in
// memory with a 12 byte offset from where the return data starts. Note that we **intentionally** only
// do this for the `masterCopy()` call, since the EVM `DELEGATECALL` opcode ignores the most-significant
// 12 bytes from the address, so we do not need to make sure the top bytes are cleared when proxying
// calls to the `singleton`. This saves us a tiny amount of gas per proxied call.
mstore(0x0c, shl(96, _singleton))
return(0, 0x20)
```

The implementation assumes that the first 12 bytes of memory (scratch space) are zeroed out. However, the Solidity compiler does not guarantee this behavior, even when no code precedes the inline assembly block.

Recommendation

Use the zero slot in memory (0x60 - 0x7f) instead of the scratch space to avoid possible issues with new versions of the Solidity compiler.

Fix 1.1

The finding was fixed by using the first 12 bytes of the return data from the zero slot instead of the scratch space.

Go back to Findings Summary

W4: Safe `setup` may emit outdated information

Impact:	Warning	Likelihood:	N/A
Target:	Safe.sol	Type:	Logic error

Description

The `Safe.setup` function initializes the contract with owners, fallback handler, and modules.

The function emits a `SafeSetup` event as its final operation:

Listing 9. Excerpt from [Safe.setup]()

```
setupModules(to, data);

if (payment > 0) {
    // To avoid running into issues with EIP-170 we reuse the handlePayment function (to avoid adjusting code that has been verified we do not adjust the method itself)
    // baseGas = 0, gasPrice = 1 and gas = payment => amount = (payment + 0) * 1 = payment
    handlePayment(payment, 0, 1, paymentToken, paymentReceiver);
}
emit SafeSetup(msg.sender, _owners, _threshold, to, fallbackHandler);
```

The event data may be inaccurate because the `setupModules` function can execute an arbitrary delegatecall. This delegatecall may modify the contract's state, but the event uses local variables that do not reflect these potential changes.

Recommendation

Read all relevant event parameters directly from the contract storage when emitting the `SafeSetup` event to ensure accurate data emission.

Fix 1.1

The `SafeSetup` event was moved to the beginning of the `setup` function before the delegatecall. This ensures correct event emission order relative to state changes. The documentation now clarifies that the event data reflects input parameters rather than potential state changes.

[Go back to Findings Summary]

W5: `onlyNonceZero` check can be bypassed

Impact:	Warning	Likelihood:	N/A
Target:	SafeToL2Migration.sol	Type:	Logic error

Description

The `SafeToL2Migration` contract enables migration of a Safe to a corresponding L2-specific implementation. Due to backend constraints, the Safe must not have any executed transactions (the `nonce` value must be 0 before the migration).

Listing 10. Excerpt from SafeToL2Migration

```
modifier onlyNonceZero() {
    // Nonce is increased before executing a tx, so first executed tx will have nonce=1
    require(nonce == 1, "Safe must have not executed any tx");
    _;
}
```

The `onlyNonceZero` check can be bypassed through the following steps:

1. A Safe is created with a module;
2. An arbitrary transaction is executed on the Safe, increasing the `nonce` value to 1; and
3. The migration is executed from the module, which does not increase the `nonce` value.

Recommendation

Ensure that the described scenario does not introduce security issues on the backend.

Fix 1.1

The finding was resolved by adding a comment to the `onlyNonceZero` modifier documenting that the described bypass scenario is possible.

Go back to Findings Summary

W6: Locked tokens possibility

Impact:	Warning	Likelihood:	N/A
Target:	TokenCallbackHandler.sol	Type:	Configuration

Description

The `TokenCallbackHandler` contract lacks validation to ensure its functions are called exclusively as fallbacks from a Safe contract.

When users directly send ERC-721 or ERC-1155 tokens to the contract address, these tokens become permanently locked in the contract.

This issue was identified during a manually-guided fuzzing session.

Recommendation

Implement validation to prevent users from sending tokens to this contract directly. Alternatively, acknowledge and document the behavior.

Fix 1.1

The issue was fixed by fetching the `FALLBACK_HANDLER_STORAGE_SLOT` storage value and comparing the decoded address with `msg.sender`. The execution reverts if the address does not match, preventing accidental token transfers.

Go back to Findings Summary

W7: `ProxyCreationL2` nonce value is not user given argument

Impact:	Warning	Likelihood:	N/A
Target:	SafeProxyFactory.sol	Type:	Logic error

Description

The `SafeProxyFactory.createProxyWithCallbackL2` function call emits the `SafeProxyFactory.ProxyCreationL2` event. However, the `saltNonce` value in `SafeProxyFactory.ProxyCreationL2` returns an internal salt value, and it is not a user-provided value.

This issue was found during manually-guided fuzzing session.

Recommendation

Either

- emit the event in the `createProxyWithCallbackL2` function as well;
- perform the operation in `createProxyWithCallbackL2` without using the `createProxyWithNonceL2` function.

Fix 1.1

The finding was resolved as the affected function was removed as part of the M1 fix.

Go back to Findings Summary

I1: Documentation issues

Impact:	Info	Likelihood:	N/A
Target:	FallbackManager.sol, SignatureVerifierMuxer.sol, MultiSendCallOnly.sol	Type:	Code quality

Description

The codebase contains multiple instances of incorrect or misleading documentation.

Listing 11. Excerpt from [FallbackManager]()

```
// @notice Forwards all calls to the fallback handler if set. Returns 0 if no
handler is set.
```

The statement "Returns 0 if no handler is set" is ambiguous. The function returns empty bytes rather than `uint(0)` when no handler is set.

Listing 12. Excerpt from [SignatureVerifierMuxer.isValidSignature]()

```
// Signature is for an `ISafeSignatureVerifier` - decode the signature.
// Layout of the `signature`:
// 0x00 to 0x04: selector
// 0x04 to 0x36: domainSeparator
// 0x36 to 0x68: typeHash
// 0x68 to 0x88: encodeData length
// 0x88 to 0x88 + encodeData length: encodeData
// 0x88 + encodeData length to 0x88 + encodeData length + 0x20: payload
length
// 0x88 + encodeData length + 0x20 to end: payload
```

The `domainSeparator` and `typeHash` comments appear to document decimal values with a hexadecimal prefix, resulting in incorrect byte offsets for all positions except the first one.

The documentation of `encodeData` and `payload` encoding is incorrect. While the documentation describes a tightly packed format `| length (32 B) | data |`, the code uses `abi.decode`. ABI encoding follows a different format where the offset to encoded data is stored first, followed by the length at that offset, and finally the data with right padding to 32 bytes.

Listing 13. Excerpt from [MultiSendCallOnly.multiSend]()

```
// We shift by 248 bits (256 - 8 [operation byte]) it right since mload will
   always load 32 bytes (a word).
```

The `MultiSendCallOnly.multiSend` documentation contains a typographical error. The corrected version of this documentation exists in `MultiSend.multiSend`.

Listing 14. Excerpt from [MultiSend.multiSend]()

```
// We shift by 248 bits (256 - 8 [operation byte]) right, since mload will
   always load 32 bytes (a word).
```

The `checkNSignatures(bytes32, bytes, bytes, uint256)` function is documented in the changelog as being implemented in the `CompatibilityFallbackHandler` contract, but it is implemented in the main `Safe` contract.

Recommendation

Fix the documentation issues.

Fix 1.1

All the documentation issues were fixed.

[Go back to Findings Summary]()

I2: Unnecessary typecasts to `payable`

Impact:	Info	Likelihood:	N/A
Target:	**/*.sol	Type:	Code quality

Description

The codebase contains multiple unnecessary typecasts to the `payable` type when casting to the `ISafe` interface.

The complete list of occurrences is available in Appendix B.

Recommendation

Remove the `payable` typecasts and cast directly to the `ISafe` interface.

Fix 1.1

The finding was fixed by introducing a new `INativeCurrencyPaymentFallback` interface defining the `receive` function. The `ISafe` interface now inherits from this interface, making the typecasts necessary.

Go back to Findings Summary

I3: Code optimizations

Impact:	Info	Likelihood:	N/A
Target:	ModuleManager.sol, CompatibilityFallbackHandler.sol, ERC165Handler.sol, SignatureVerifierMuxer.sol	Type:	Gas optimization

Description

The project code may be optimized for gas usage in multiple instances.

Listing 15. Excerpt from [ModuleManager.preModuleExecution]()

```
require(msg.sender != SENTINEL_MODULES && modules[msg.sender] != address(0),
"GS104");
```

The line can be optimized as follows:

```
if (msg.sender == SENTINEL_MODULES || modules[msg.sender] == address(0))
revertWithError("GS104");
```

Listing 16. Excerpt from [ERC165Handler.addSupportedInterfaceBatch]()

```
bytes4 interfaceId;
uint256 len = handlerWithSelectors.length;
for (uint256 i = 0; i < len; ++i) {
    (bool isStatic, bytes4 selector, address handlerAddress) =
MarshalLib.decodeWithSelector(handlerWithSelectors[i]);
    _setSafeMethod(safe, selector, MarshalLib.encode(isStatic,
handlerAddress));
    if (i > 0) {
        interfaceId ^= selector;
    } else {
        interfaceId = selector;
    }
}
```

Listing 17. Excerpt from [ERC165Handler.removeSupportedInterfaceBatch]()

```
bytes4 interfaceId;
uint256 len = selectors.length;
for (uint256 i = 0; i < len; ++i) {
    _setSafeMethod(safe, selectors[i], bytes32(0));
    if (i > 0) {
        interfaceId ^= selectors[i];
    } else {
        interfaceId = selectors[i];
    }
}
```

The `if (i > 0)` conditions can be removed because applying xor to zero has the same result as assigning the value. Note that removing the if conditions does not necessarily save more gas, but it makes the code more readable.

Recommendation

Consider the code optimizations.

Fix 1.1

The finding was resolved by applying the recommended optimizations.

[Go back to Findings Summary]()

I4: Factory `initializer` error not propagated

Impact:	Info	Likelihood:	N/A
Target:	SafeProxyFactory.sol	Type:	Code quality

Description

The `SafeProxyFactory` contract allows an arbitrary call on the created proxy contract. The `initializer` parameter defines the call data.

Listing 18. Excerpt from [SafeProxyFactory.deployProxy]()

```
assembly {
    if iszero(call(gas(), proxy, 0, add(initializer, 0x20),
mload(initializer), 0, 0)) {
        revert(0, 0)
    }
}
```

The revert error from the `initializer` call is not propagated to the caller.

Recommendation

Consider propagating the revert error from the `initializer` call to the caller to improve the user experience.

Fix 1.1

The finding was resolved by following the recommendation.

[Go back to Findings Summary]()

I5: No view function for `FallbackManager` handler address

Impact:	Info	Likelihood:	N/A
Target:	FallbackManager.sol	Type:	Code quality

Description

Each contract in `contracts/base` implements functionality for reading stored data in the `Safe` contract:

- `GuardManager` implements the `getGuard` view function;
- `ModuleManager` implements the `getModuleGuard` view function;
- `OwnerManager` implements the `getOwners` view function; and
- `Executor` does not use any storage.

The `FallbackManager` contract lacks a view function to access its stored data, unlike other contracts in the same directory.

This issue was found during manually-guided fuzzing.

Recommendation

Implement a `getFallbackHandler` view function to maintain consistency with other contracts and improve usability.

Acknowledgment 1.1

The finding was acknowledged by the client with the following comment:

> *The view function for reading the transaction and module guards are internal to the contract. As such, we will keep the fallback handler without a view function, as the refactor will add unnecessary complexity (as the function will be*

> *implemented outside of* `assembly`*, while the fallback method itself is in* `assembly`*) and its value can be accessed via the* `StorageAccessible` *interface. We will consider adding either new view functions to the* `CompatibilityFallbackHandler` *or an* `Accessor` *contract to more easily read these values (fallback handler, guards, etc.) from Safes in the future.*
>
> — Safe team

[Go back to Findings Summary]

I6: `SafeStorage` can be defined as abstract

Impact:	Info	Likelihood:	N/A
Target:	SafeStorage.sol	Type:	Code quality

Description

The `SafeStorage` contract is designed to be used as a base contract.

Listing 19. Excerpt from [SafeStorage](#)

```
 * @title SafeStorage - Storage layout of the Safe Smart Account contracts to
be used in libraries.
 * @dev Should be always the first base contract of a library that is used
with a Safe.
 * @author Richard Meissner - @rmeissner
 */
contract SafeStorage {
```

However, it is not marked as `abstract`, which would better reflect its intended usage.

Recommendation

Mark the `SafeStorage` contract as `abstract` to indicate its intended usage as a base contract.

Fix 1.1

The `SafeStorage` contract was marked as `abstract` along with the `SignatureValidatorConstants` contract.

[Go back to Findings Summary](#)

I7: Missing L2-specific variant of `createChainSpecificProxyWithNonce`

Impact:	Info	Likelihood:	N/A
Target:	SafeProxyFactory.sol	Type:	Code quality

Description

The `SafeProxyFactory` contract implements L2-specific variants of deployment functions that include event emission for L2 networks. However, the `createChainSpecificProxyWithNonce` function lacks its corresponding L2-specific variant, unlike other deployment functions in the contract.

This issue was found during manually-guided fuzzing session.

Recommendation

Implement an L2-specific variant of `createChainSpecificProxyWithNonce` that includes event emission, following the pattern of other L2 deployment functions in the contract.

Fix 1.1

The finding was resolved by adding a new `createChainSpecificProxyWithNonceL2` function emitting an extra event with additional parameters.

[Go back to Findings Summary]

I8: Interface type used for parameter that accepts zero address

Impact:	Info	Likelihood:	N/A
Target:	SafeProxyFactory.sol	Type:	Code quality

Description

When a parameter accepts the zero address as a valid input with special handling, it is more appropriate to use the `address` type rather than an interface type. This is because interface types semantically indicate that the input should be a contract implementing that interface.

The following parameter uses an interface type but accepts the zero address:

Listing 20. Excerpt from [SafeProxyFactory.createProxyWithCallback]()

```
if (address(callback) != address(0)) callback.proxyCreated(proxy,
    _singleton, initializer, saltNonce);
```

This issue was found during manually-guided fuzzing session.

Recommendation

Change the parameter type to `address` to indicate that zero address is a valid parameter value.

Fix 1.1

The finding was resolved as the affected function was removed as part of the [M1]() fix.

[Go back to Findings Summary]()

I9: `ChangedThreshold` event is emitted unconditionally

Impact:	Info	Likelihood:	N/A
Target:	OwnerManager.sol	Type:	Code quality

Description

The `ChangedThreshold` event is emitted regardless of whether the threshold value has changed.

Listing 21. Excerpt from [OwnerManager.changeThreshold]()

```
if (_threshold > ownerCount) revertWithError("GS201");
// There has to be at least one Safe owner.
if (_threshold == 0) revertWithError("GS202");
threshold = _threshold;
emit ChangedThreshold(_threshold);
```

Recommendation

Add a condition to emit the `ChangedThreshold` event only when the threshold value changes.

Acknowledgment 1.1

The finding was acknowledged by the client with the following comment:

> *Emitting events even when values do not change (for example, when setting a module guard), is consistent with the rest of the Safe contracts. We no not wish to change them, as we believe there would be limited value in doing so and do not expect Safe owners to regularly execute transactions that would emit these events without changing values.*

> — Safe team

Go back to Findings Summary

Appendix A: How to cite

Please cite this document as:

Ackee Blockchain Security, Safe: Smart Account, 28.5.2025.

Appendix B: Wake Findings

This section lists the outputs from the Wake framework used for testing and static analysis during the audit.

B.1. Fuzzing

The following table lists all implemented execution flows in the Wake fuzzing framework.

ID	Flow	Added
F1	Creation of Safe account	1.0
F2	Perform random operation with Safe account	1.0
F3	Transfer ERC721 token to specific contract	1.0
F4	Transfer ERC1155 token to specific contract	1.0
F5	Install debug transaction guard to Safe account	1.0
F6	Install reentrancy guard to Safe account	1.0
F7	Install owners-only guard to Safe account	1.0
F8	Enable example module for Safe account	1.0
F9	Disable example module for Safe account	1.0
F10	Execute from example module	1.0
F11	Execute from example module with return value	1.0
F12	Change Safe account threshold	1.0
F13	Remove owner of Safe account	1.0
F14	Add owner and set threshold of Safe account	1.0
F15	Swap owner of Safe account	1.0
F16	Install `ExtensibleFallbackHandler` to Safe account	1.0
F17	Set Safe method with `ExtensibleFallbackHandler`	1.0

ID	Flow	Added
F18	Call with method installed at `ExtensibleFallbackHandler`	1.0
F19	Perform random user direct set method operation to `ExtensibleFallbackHandler` without Safe account	1.0
F20	Remove interface from `ExtensibleFallbackHandler` by Safe account	1.0
F21	Transfer random ERC20 token to deployed contract	1.0
F22	Transfer random ERC20 token from Safe account	1.0
F23	Add batch interface with `ExtensibleFallbackHandler`	1.0
F24	Remove batch interface with `ExtensibleFallbackHandler`	1.0
F25	Install module guard to Safe account	1.0
F26	Set domain verifier for Safe account	1.0
F27	Simulate Safe account operation	1.0
F28	Perform multi-send operation by Safe account	1.0
F29	Perform multi-send call-only operation by Safe account	1.0
F30	Perform `isValidSignature` check on Safe account	1.0

Table 4. Wake fuzzing flows

The following table lists the invariants checked after each flow.

ID	Invariant	Added	Status
IV1	Transactions do not revert except where explicitly expected	1.0	Success
IV2	Safe account owner's `isOwner` returns True	1.0	Success
IV3	Safe account threshold is at expected value	1.0	Success

ID	Invariant	Added	Status
IV4	Safe account deployment address is at expected value	1.0	**Fail** (M1)
IV5	Safe account nonce is at expected value	1.0	Success
IV6	Safe account fallback handler address is at expected value	1.0	**Fail** (I5)
IV7	Safe account modules are installed correctly and in expected order	1.0	Success
IV8	Guard's `supportInterface` returns expected value	1.0	Success
IV9	Safe account `supportInterface` returns expected value	1.0	Success
IV10	Safe account and deployed contract have expected token balances	1.0	Success
IV11	Events are emitted as expected	1.0	Success
IV12	`isValidSignature` for Safe account returns expected value	1.0	Success

Table 5. Wake fuzzing invariants

B.2. Detectors

wake detect unnecessary-typecast

```
[INFO][HIGH] Unnecessary typecast [unnecessary-typecast]
     * @return Message hash.
     */
    function getMessageHash(bytes memory message) public view returns (bytes32) {
        return getMessageHashForSafe(ISafe(payable(msg.sender)), message);
    }

    /**
contracts/handler/CompatibilityFallbackHandler.sol
```

```
[INFO][HIGH] Unnecessary typecast [unnecessary-typecast]
     */
    function isValidSignature(bytes32 _dataHash, bytes calldata _signature) public view override ret
        // Caller should be a Safe.
        ISafe safe = ISafe(payable(msg.sender));
        bytes memory messageData = encodeMessageDataForSafe(safe, abi.encode(_dataHash));
        bytes32 messageHash = keccak256(messageData);
        if (_signature.length == 0) {
contracts/handler/CompatibilityFallbackHandler.sol
```

```
[INFO][HIGH] Unnecessary typecast [unnecessary-typecast]
     */
    function getModules() external view returns (address[] memory) {
        // Caller should be a Safe.
        ISafe safe = ISafe(payable(msg.sender));
        (address[] memory array, ) = safe.getModulesPaginated(SENTINEL_MODULES, 10);
        return array;
    }
contracts/handler/CompatibilityFallbackHandler.sol
```

```
[INFO][HIGH] Unnecessary typecast [unnecessary-typecast]
     * @param supported True if the interface is supported, false otherwise
     */
    function setSupportedInterface(bytes4 interfaceId, bool supported) public override onlySelf {
        ISafe safe = ISafe(payable(_manager()));
        // invalid interface id per ERC165 spec
        require(interfaceId != 0xffffffff, "invalid interface id");
        mapping(bytes4 => bool) storage safeInterface = safeInterfaces[safe];
contracts/handler/extensible/ERC165Handler.sol
```

```
[INFO][HIGH] Unnecessary typecast [unnecessary-typecast]
     * @param handlerWithSelectors The handlers encoded with the 4-byte selectors of the methods
     */
    function addSupportedInterfaceBatch(bytes4 _interfaceId, bytes32[] calldata handlerWithSelectors
        ISafe safe = ISafe(payable(_msgSender()));
        bytes4 interfaceId;
        uint256 len = handlerWithSelectors.length;
        for (uint256 i = 0; i < len; ++i) {
contracts/handler/extensible/ERC165Handler.sol
```

```
[INFO][HIGH] Unnecessary typecast [unnecessary-typecast]
     * @param selectors The selectors of the methods to remove
     */
    function removeSupportedInterfaceBatch(bytes4 _interfaceId, bytes4[] calldata selectors) externa
        ISafe safe = ISafe(payable(_msgSender()));
        bytes4 interfaceId;
        uint256 len = selectors.length;
        for (uint256 i = 0; i < len; ++i) {
contracts/handler/extensible/ERC165Handler.sol
```

Figure 1. Unnecessary typecasts

```
wake detect unnecessary-typecast

[INFO][HIGH] Unnecessary typecast [unnecessary-typecast]
            interfaceId == type(IERC165).interfaceId ||
            interfaceId == type(IERC165Handler).interfaceId ||
            _supportsInterface(interfaceId) ||
❱           safeInterfaces[ISafe(payable(_manager()))][interfaceId];
    }

    // --- internal ---
contracts/handler/extensible/ERC165Handler.sol

[INFO][HIGH] Unnecessary typecast [unnecessary-typecast]
     * @return sender The original `msg.sender` (as received by the FallbackManager)
     */
    function _getContext() internal view returns (ISafe safe, address sender) {
❱       safe = ISafe(payable(_manager()));
        sender = _msgSender();
    }

contracts/handler/extensible/ExtensibleBase.sol

[INFO][HIGH] Unnecessary typecast [unnecessary-typecast]
     * @param newMethod A contract that implements the `IFallbackMethod` or `IStaticFallbackMethod`
     */
    function setSafeMethod(bytes4 selector, bytes32 newMethod) public override onlySelf {
❱       _setSafeMethod(ISafe(payable(_msgSender())), selector, newMethod);
    }

    // --- fallback ---
contracts/handler/extensible/FallbackHandler.sol

[INFO][HIGH] Unnecessary typecast [unnecessary-typecast]
     * @param newVerifier A contract that implements `ISafeSignatureVerifier`
     */
    function setDomainVerifier(bytes32 domainSeparator, ISafeSignatureVerifier newVerifier) public o
❱       ISafe safe = ISafe(payable(_msgSender()));
        ISafeSignatureVerifier oldVerifier = domainVerifiers[safe][domainSeparator];
        if (address(newVerifier) == address(0) && address(oldVerifier) != address(0)) {
            delete domainVerifiers[safe][domainSeparator];
contracts/handler/extensible/SignatureVerifierMuxer.sol

[INFO][HIGH] Unnecessary typecast [unnecessary-typecast]
     */
    function getMessageHash(bytes memory message) public view returns (bytes32) {
        bytes32 safeMessageHash = keccak256(abi.encode(SAFE_MSG_TYPEHASH, keccak256(message)));
❱       return keccak256(abi.encodePacked(bytes1(0x19), bytes1(0x01), ISafe(payable(address(this))).
    }
}
contracts/libraries/SignMessageLib.sol

[INFO][HIGH] Unnecessary typecast [unnecessary-typecast]
     */
    function getMessageHash(bytes memory message) public view returns (bytes32) {
        bytes32 safeMessageHash = keccak256(abi.encode(SAFE_MSG_TYPEHASH, keccak256(message)));
❱       return keccak256(abi.encodePacked(bytes1(0x19), bytes1(0x01), ISafe(payable(address(this))).
    }
}
contracts/libraries/SignMessageLib.sol
```

Figure 2. Unnecessary typecasts

Thank You

Ackee Blockchain a.s.

Rohanske nabrezi 717/4
186 00 Prague
Czech Republic

hello@ackee.xyz