



Syndicate

Smart Wallets

SMART CONTRACT AUDIT

13.11.2023

Made in Germany by Softstack.io



Table of contents

1. Disclaimer.....	3
2. About the Project and Company	4
2.1 Project Overview.....	5
3. Vulnerability & Risk Level	6
4. Auditing Strategy and Techniques Applied.....	7
4.1 Methodology	7
5. Metrics	8
5.1 Tested Contract Files	8
5.2 Source Lines & Risk.....	9
5.3 Capabilities	10
5.4 Dependencies / External imports	11
5.5 Source Unites in Scope	13
6. Scope of Work.....	14
6.1 Findings Overview	15
6.2 Manual and Automated Vulnerability Test.....	16
6.2.1 Unusable Smart Wallets Created By Factory	17
6.2.2 Unchecked Return Parameter In Validity Check.....	19
6.2.3 Invalid Token Receiver Address Due To Wrong Padded Calldata	20
6.3 SWC Attacks	21
6.4 Verify Claims	25
6.5 Unit Tests.....	26
7. Executive Summary.....	29



8. About the Auditor	30
----------------------------	----

1. Disclaimer

The audit makes no statements or warranties about utility of the code, safety of the code, suitability of the business model, investment advice, endorsement of the platform or its products, regulatory regime for the business model, or any other statements about fitness of the contracts to purpose, or their bug free status. The audit documentation is for discussion purposes only.

The information presented in this report is confidential and privileged. If you are reading this report, you agree to keep it confidential, not to copy, disclose or disseminate without the agreement of SYNDICATE INC. If you are not the intended receptor of this document, remember that any disclosure, copying or dissemination of it is forbidden.

Major Versions / Date	Description
0.1 (01.11.2023)	Layout
0.4 (02.11.2023)	Automated Security Testing Manual Security Testing
0.5 (06.11.2023)	Verify Claims and Test Deployment
0.6 (07.11.2023)	Testing SWC Checks
0.9 (08.11.2023)	Summary and Recommendation
1.0 (08.11.2023)	Final document
1.1 (13.11.2023)	Re-check 555b91b28c626d10ee2642b4b5d1d2932496df35



2. About the Project and Company

Company address:

SYNDICATE INC.
1049 El Monte Avenue Ste C #560
Mountain View, CA 94040
USA



Website: <https://syndicate.io>

Twitter: <https://twitter.com/syndicateio>

Substack: <https://syndicateprotocol.substack.com/>

2.1 Project Overview

Syndicate is a revolutionary platform that sits at the crossroads of finance and web3, focusing on decentralizing and democratizing the investing landscape. By seamlessly integrating both financial protocols and social networking, Syndicate is poised to radically reshape the way capital and communities interact and collaborate. A cornerstone of this platform is "Collectives," an innovative web3-based tool for social networking and community building. This allows users to craft on-chain social networks and has already seen integration from notable communities like Rug Radio, BFF, Kamp Kilmer, and FWB Fellowships.

Integral to the platform's functionality is the new ERC-721M standard, an evolution of the recognized ERC-721 NFT standard. This protocol has been designed with features that allow for greater modularity, including actions such as minting, burning, transferring, and evolving metadata. One of its standout attributes is its gas-optimized design, ensuring efficiency in minting and distribution.

For those less familiar with coding, Syndicate presents a no-code tool that makes the process of setting up Collectives straightforward. This user-friendly feature ensures that customization, from member limits to transferability, is accessible to a wider audience. Importantly, it has been designed with interoperability in mind, ensuring seamless interaction with various web3 tools and platforms.

What sets Syndicate's approach to social networks apart is its emphasis on user ownership, decentralized design, interoperability, and adaptability. In a Syndicate network, the users truly own the platform, a departure from traditional centralized entities. Moreover, these networks can evolve and modify based on the specific needs and desires of the community, ensuring they remain relevant and responsive.

Looking ahead, Syndicate has ambitious plans to intertwine unique social constructs with financial mechanisms. Such integration can potentially allow founders to raise capital directly from their social networks, investors to efficiently syndicate deals, and communities to trade work in exchange for ownership stakes. Notably, the Syndicate Genesis Collective, an on-chain social network, is crafted for the broader Syndicate community, offering a unique space for collaboration. The Collective is available for a limited period and is set to introduce a plethora of dynamic features for its early members.

In essence, Syndicate is not just a platform; it is a vision of the future. A future where the boundaries between finance and community blur, creating a more inclusive, dynamic, and transparent financial ecosystem.

3. Vulnerability & Risk Level

Risk represents the probability that a certain source-threat will exploit vulnerability, and the impact of that event on the organization or system. Risk Level is computed based on CVSS version 3.0.

Level	Value	Vulnerability	Risk (Required Action)
Critical	9 – 10	A vulnerability that can disrupt the contract functioning in a number of scenarios, or creates a risk that the contract may be broken.	Immediate action to reduce risk level.
High	7 – 8.9	A vulnerability that affects the desired outcome when using a contract, or provides the opportunity to use a contract in an unintended way.	Implementation of corrective actions as soon as possible.
Medium	4 – 6.9	A vulnerability that could affect the desired outcome of executing the contract in a specific scenario.	Implementation of corrective actions in a certain period.
Low	2 – 3.9	A vulnerability that does not have a significant impact on possible scenarios for the use of the contract and is probably subjective.	Implementation of certain corrective actions or accepting the risk.
Informational	0 – 1.9	A vulnerability that have informational character but is not effecting any of the code.	An observation that does not determine a level of risk

4. Auditing Strategy and Techniques Applied

Throughout the review process, care was taken to evaluate the repository for security-related issues, code quality, and adherence to specification and best practices. To do so, reviewed line-by-line by our team of expert pentesters and smart contract developers, documenting any issues as there were discovered.

4.1 Methodology

The auditing process follows a routine series of steps:

1. Code review that includes the following:
 - i. Review of the specifications, sources, and instructions provided to softstack to make sure we understand the size, scope, and functionality of the smart contract.
 - ii. Manual review of code, which is the process of reading source code line-by-line in an attempt to identify potential vulnerabilities.
 - iii. Comparison to specification, which is the process of checking whether the code does what the specifications, sources, and instructions provided to softstack describe.
2. Testing and automated analysis that includes the following:
 - i. Test coverage analysis, which is the process of determining whether the test cases are actually covering the code and how much code is exercised when we run those test cases.
 - ii. Symbolic execution, which is analysing a program to determine what inputs causes each part of a program to execute.
3. Best practices review, which is a review of the smart contracts to improve efficiency, effectiveness, clarify, maintainability, security, and control based on the established industry and academic practices, recommendations, and research.
4. Specific, itemized, actionable recommendations to help you take steps to secure your smart contracts.

5. Metrics

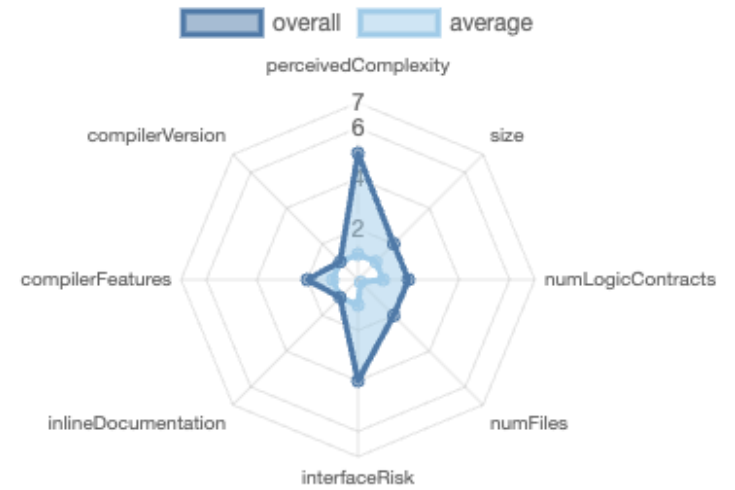
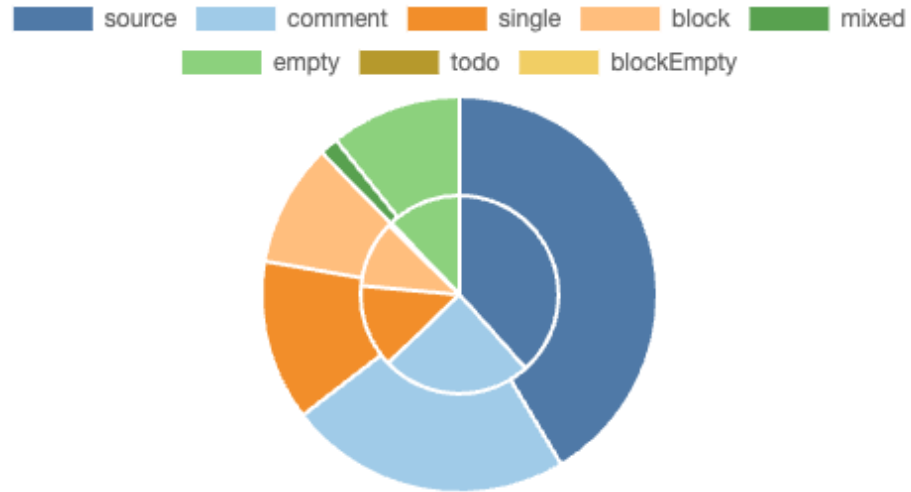
The metrics section should give the reader an overview on the size, quality, flows and capabilities of the codebase, without the knowledge to understand the actual code.

5.1 Tested Contract Files











The following are the MD5 hashes of the reviewed files. A file with a different MD5 hash has been modified, intentionally or otherwise, after the security review. You are cautioned that a different MD5 hash could be (but is not necessarily) an indication of a changed condition or potential vulnerability that was not within the scope of the review

File	Fingerprint (MD5)
./smart-wallets/SmartWalletMixinManager.sol	e915556538faf823e44061ce74d98130
./smart-wallets/operators/ExecOperator.sol	4b03c2e8a6b4382a5af78bdbe847df0b
./smart-wallets/mixins/NoDelegateCallMixin.sol	b763b3c1b00738318d85b1cd1df783a0
./smart-wallets/mixins/RecurringPaymentMixin.sol	a6d11b978fd45c7f980aee1f2ec0937e
./smart-wallets/mixins/RolesMixin.sol	eca9d3d5f3869e0bfe4f3a3a09c4be18
./smart-wallets/mixins/NoSelfAuthorizedMixin.sol	b4e677bfda9fbd94324b484a9c57c7c9
./smart-wallets/utils/SafeFactoryWrapper.sol	2f63e07faeabd02d04dab3d5b59bd062
./smart-wallets/utils/SynSafeUtils.sol	38dadf54bf4cdb65e06fc1ebd2a60972
./smart-wallets/SmartWalletModule.sol	bc72e2d47f0e37badba2d9f9368099cc
./smart-wallets/factory/SmartWalletFactory.sol	ec3db620f14f4aea9b1ff20092740a1a
./smart-wallets/interfaces/ISmartWalletMixinManager.sol	1f4f335c364738583dbcc57c0c684376
./smart-wallets/interfaces/IGnosisSafe.sol	4e0d86c573ddcc0690e84a8e33878244
./smart-wallets/interfaces/ISmartWalletModule.sol	9623f8428885539d18f5ed2b82db9440
./smart-wallets/interfaces/ISmartWalletMixin.sol	83c815650ca8b9efb0d77c5aed8c00d4

5.2 Source Lines & Risk





5.3 Capabilities


Solidity Versions observed	 Experimental Features	 Can Receive Funds	 Uses Assembly	 Has Destroyable Contracts	
0.8.21			yes (3 asm blocks)		
 Transfers ETH	 Low-Level Calls	 DelegateCall	 Uses Hash Functions	 ECTrecover	 New/Create/Create2
			yes	yes	

Exposed Functions

This section lists functions that are explicitly declared public or payable. Please note that getter methods for public stateVars are not included.

 Public	 Payable				
48	0				
External	Internal	Private	Pure	View	
41	51	0	4	9	

StateVariables

Total	 Public
21	14

5.4 Dependencies / External imports

Dependency / Import Path	Source
@openzeppelin/contracts-upgradeable/interfaces/IERC2981Upgradeable.sol	https://github.com/OpenZeppelin/openzeppelin-contracts-upgradeable/tree/v4.6.0/contracts/interfaces/IERC2981Upgradeable.sol
@openzeppelin/contracts-upgradeable/proxy/utils/Initializable.sol	https://github.com/OpenZeppelin/openzeppelin-contracts-upgradeable/tree/v4.6.0/contracts/proxy/utils/Initializable.sol
@openzeppelin/contracts-upgradeable/security/ReentrancyGuardUpgradeable.sol	https://github.com/OpenZeppelin/openzeppelin-contracts-upgradeable/tree/v4.6.0/contracts/security/ReentrancyGuardUpgradeable.sol
@openzeppelin/contracts-upgradeable/token/ERC721/ERC721Upgradeable.sol	https://github.com/OpenZeppelin/openzeppelin-contracts-upgradeable/tree/v4.6.0/contracts/token/ERC721/ERC721Upgradeable.sol
@openzeppelin/contracts-upgradeable/token/ERC721/IERC721ReceiverUpgradeable.sol	https://github.com/OpenZeppelin/openzeppelin-contracts-upgradeable/tree/v4.6.0/contracts/token/ERC721/IERC721ReceiverUpgradeable.sol
@openzeppelin/contracts-upgradeable/token/ERC721/IERC721Upgradeable.sol	https://github.com/OpenZeppelin/openzeppelin-contracts-upgradeable/tree/v4.6.0/contracts/token/ERC721/IERC721Upgradeable.sol
@openzeppelin/contracts-upgradeable/token/ERC721/extensions/ERC721RoyaltyUpgradeable.sol	https://github.com/OpenZeppelin/openzeppelin-contracts-upgradeable/tree/v4.6.0/contracts/token/ERC721/extensions/ERC721RoyaltyUpgradeable.sol
@openzeppelin/contracts-upgradeable/token/ERC721/extensions/IERC721MetadataUpgradeable.sol	https://github.com/OpenZeppelin/openzeppelin-contracts-upgradeable/tree/v4.6.0/contracts/token/ERC721/extensions/IERC721MetadataUpgradeable.sol
@openzeppelin/contracts-upgradeable/token/common/ERC2981Upgradeable.sol	https://github.com/OpenZeppelin/openzeppelin-contracts-upgradeable/tree/v4.6.0/contracts/token/common/ERC2981Upgradeable.sol

Dependency / Import Path	Source
@openzeppelin/contracts-upgradeable/utils/AddressUpgradeable.sol	https://github.com/OpenZeppelin/openzeppelin-contracts-upgradeable/tree/v4.6.0/contracts/utils/AddressUpgradeable.sol
@openzeppelin/contracts-upgradeable/utils/ContextUpgradeable.sol	https://github.com/OpenZeppelin/openzeppelin-contracts-upgradeable/tree/v4.6.0/contracts/utils/ContextUpgradeable.sol
@openzeppelin/contracts-upgradeable/utils/StringsUpgradeable.sol	https://github.com/OpenZeppelin/openzeppelin-contracts-upgradeable/tree/v4.6.0/contracts/utils/StringsUpgradeable.sol
@openzeppelin/contracts-upgradeable/utils/introspection/ERC165Upgradeable.sol	https://github.com/OpenZeppelin/openzeppelin-contracts-upgradeable/tree/v4.6.0/contracts/utils/introspection/ERC165Upgradeable.sol
@openzeppelin/contracts-upgradeable/utils/introspection/IERC165Upgradeable.sol	https://github.com/OpenZeppelin/openzeppelin-contracts-upgradeable/tree/v4.6.0/contracts/utils/introspection/IERC165Upgradeable.sol
@openzeppelin/contracts/access/Ownable.sol	https://github.com/OpenZeppelin/openzeppelin-contracts/tree/v4.6.0/contracts/access/Ownable.sol
@openzeppelin/contracts/proxy/Clones.sol	https://github.com/OpenZeppelin/openzeppelin-contracts/tree/v4.6.0/contracts/proxy/Clones.sol
@openzeppelin/contracts/security/Pausable.sol	https://github.com/OpenZeppelin/openzeppelin-contracts/tree/v4.6.0/contracts/security/Pausable.sol
@openzeppelin/contracts/utils/Strings.sol	https://github.com/OpenZeppelin/openzeppelin-contracts/tree/v4.6.0/contracts/utils/Strings.sol
@openzeppelin/contracts/utils/introspection/ERC165Checker.sol	https://github.com/OpenZeppelin/openzeppelin-contracts/tree/v4.6.0/contracts/utils/introspection/ERC165Checker.sol

5.5 Source Unites in Scope

File	Logic Contracts	Interfaces	Lines	nLines	nSLOC	Comment Lines	Complex. Score
src/contracts/smart-wallets/SmartWalletMixinManager.sol	1		160	144	67	60	35
src/contracts/smart-wallets/SmartWalletModule.sol	1		123	111	57	36	45
src/contracts/smart-wallets/Utils/SynSafeUtils.sol			36	36	28	1	
src/contracts/smart-wallets/Utils/SafeFactoryWrapper.sol	1	1	77	64	42	19	42
src/contracts/smart-wallets/interfaces/IGnosisSafe.sol		1	28	14	4	14	5
src/contracts/smart-wallets/interfaces/ISmartWalletModule.sol		1	25	8	4	1	7
src/contracts/smart-wallets/mixins/RecurringPaymentMixin.sol	1		143	135	73	46	66
src/contracts/smart-wallets/mixins/RolesMixin.sol	1		272	228	118	82	75
src/contracts/smart-wallets/mixins/NoDelegateCallMixin.sol	1		47	39	14	26	10

File	Logic Contracts	Interfaces	Lines	nLines	nSLOC	Comment Lines	Complex. Score
src/contracts/smart-wallets/interfaces/ISmartWalletMixinManager.sol		1	16	8	4	1	3
src/contracts/smart-wallets/operators/ExecOperator.sol	1		165	139	81	49	67
src/contracts/smart-wallets/interfaces/ISmartWalletMixin.sol		1	19	10	5	1	3
src/contracts/smart-wallets/mixins/NoSelfAuthorizedMixin.sol	1		48	40	14	25	10
src/contracts/smart-wallets/factory/SmartWalletFactory.sol	1		105	97	42	36	78
Totals	9	5	1264	1073	553	397	446

- **Lines:** total lines of the source unit
- **nLines:** normalized lines of the source unit (e.g. normalizes functions spanning multiple lines)
- **nSLOC:** normalized source lines of code (only source-code lines; no comments, no blank lines)
- **Comment Lines:** lines containing single or block comments
- **Complexity Score:** a custom complexity score derived from code statements that are known to introduce code complexity (branches, loops, calls, external interfaces, ...)

6. Scope of Work



The Syndicate Team provided us with the files that needs to be tested. The scope of the audit are the smart wallet contracts.

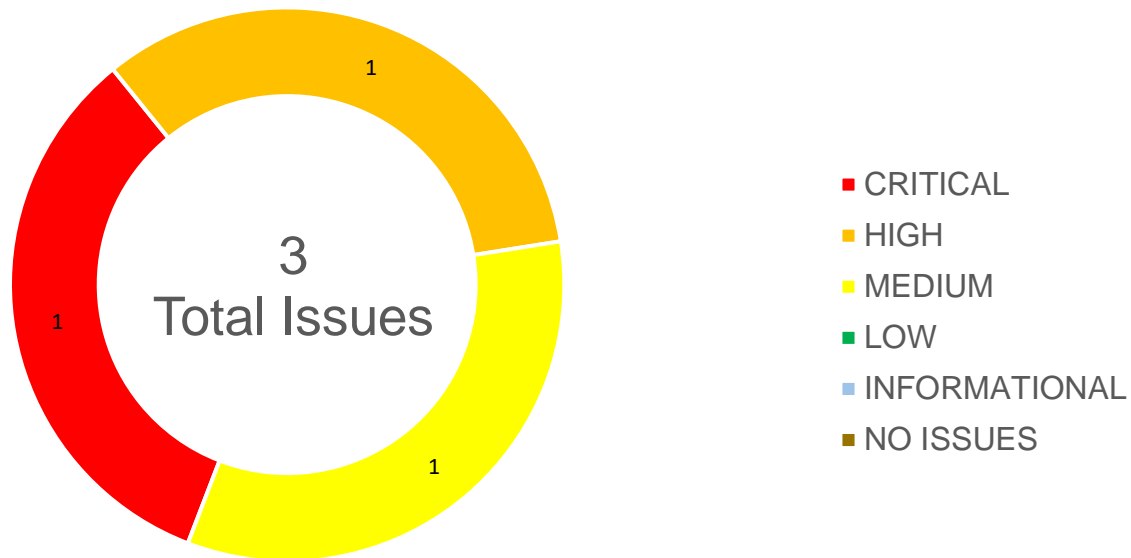
The team put forward the following assumptions regarding the security, usage of the contracts:

1. The audit must confirm that the SmartWalletModule operates securely and according to the intended logic, particularly verifying that the module strictly adheres to the mapping of allowed operators and that no unauthorized transactions bypass the MixinManager.
2. It is critical to verify the integrity of the MixinManager and associated Mixins to ensure that they enforce the cluster-wide conditions reliably, without any loopholes that could be exploited to perform unauthorized or malicious transactions.
3. The auditor should scrutinize the ManagingWallet, especially its role as a 1-of-1 owner of all wallets in a cluster, to validate that it has exclusive control over wallet management functions and that there are no vulnerabilities that could compromise its authority.
4. For the Operators, the audit must examine the modular contracts to ensure that they authenticate and authorize transactions accurately, and that the custom checks or combinations of checks do not introduce any risks or potential for unauthorized access.
5. The audit should evaluate the mechanisms that allow the Smart Wallets to interact with external contracts and services, ensuring that these interactions do not expose the wallets to risks such as front-running or unauthorized access from external contracts.

The main goal of this audit was to verify these claims. The auditors can provide additional feedback on the code upon the client's request.

6.1 Findings Overview





No	Title	Severity	Status
6.2.1	Unusable smart wallets created by factory	CRITICAL	FIXED
6.2.2	Unchecked Return Parameter In Validity Check	HIGH	FIXED
6.2.3	Invalid Token Receiver Address Due To Wrong Padded Calldata	MEDIUM	FIXED

6.2 Manual and Automated Vulnerability Test

CRITICAL ISSUES

During the audit, softstack's experts found **one Critical issue** in the code of the smart contract.

6.2.1 Unusable Smart Wallets Created By Factory

Severity: CRITICAL

Status: FIXED

Code: NA

File(s) affected: SmartWalletFactory.sol, SmartWalletModule.sol

Update: <https://github.com/SyndicateProtocol/protocol-v2/commit/555b91b28c626d10ee2642b4b5d1d2932496df35>

Attack / Description	The SmartWalletFactory is a contract to create smart wallets by creating an ExecOperator, SmartWalletModule and SmartWalletMixinManager. The intended functionality is to call the SmartWalletModule execTransactionFromModule and execTransactionFromModuleReturnData functions through the ExecOperator. These two functions are protected by the onlyOperator modifier, which allows access for registered operators only. The operators mapping is by default empty and can be managed by the owner of the contract. The SmartWalletModule contract inherits from OpenZeppelin's Ownable contract, which is setting the Owner to the contract deployer by default. While deploying the SmartWalletModule via the SmartWalletFactory, the factory contract is initialized as the owner of SmartWalletModule. Since the factory has no functions to call functions of the module, all owner functions including changing modules owner or adjusting operators can never be called. Subsequently, the ExecOperator will never be able to execute transactions with the deployed SmartWalletModule and thus, no transactions can be processed by smart wallets deployed by the SmartWalletFactory.
Code	SmartWalletFactory.sol line 29 – 46 function create(bytes32 salt, address[] calldata mixins) external whenNotPaused returns (ExecOperator execOp, SmartWalletModule smartWalletModule,

	<pre> SmartWalletMixinManager smartWalletMixinManager) { execOp = new ExecOperator{salt:salt}(); smartWalletModule = new SmartWalletModule{salt: salt}(); smartWalletMixinManager = new SmartWalletMixinManager{salt: salt}(address(smartWalletModule)); // add default mixins for the mixin manager smartWalletMixinManager.updateDefaultMixins(mixins); emit SmartWalletCreated(address(execOp), address(smartWalletModule), address(smartWalletMixinManager)); } SmartWalletModule.sol line 32: constructor() Ownable() {} </pre>
Result/Recommendation	<p>It is recommended to pass an owner into SmartWalletFactory's create function and setting this owner as the owner of SmartWalletModule during contract creation. This owner should be an external owned account to adjust the operators after deployment. Additionally, the ExecOperator's address could be passed into the the SmartWalletModules constructor to set the operator right away in the SmartWalletFactory's create function.</p>

HIGH ISSUES

During the audit, softstack's experts found **one High issue** in the code of the smart contract.



6.2.2 Unchecked Return Parameter In Validity Check

Severity: HIGH

Status: FIXED

Code: CWE-252

File(s) affected: SmartWalletModule.sol

Update: <https://github.com/SyndicateProtocol/protocol-v2/commit/555b91b28c626d10ee2642b4b5d1d2932496df35>

Attack / Description	<p>The SmartWalletModule has two functions to execute transactions: <code>execTransactionFromModule</code> and <code>execTransactionFromModuleReturnData</code>. Both functions are checking the validity of given call data by calling <code>SmartWalletMixinManager</code>'s <code>check</code> function. This function checks all registered mixins for transaction allowance and returns a boolean, indicating if the transaction is allowed. However, the returned boolean is not checked and the transaction will be executed even if the <code>SmartWalletMixinManager</code> returns false (indicating invalid transaction).</p> <p>Most of the implemented mixins are reverting on an invalid transaction calls, but there are still paths not reverting and allowing invalid transactions to pass (i.e. there are no default mixins listed). Additionally, mixins added in the future may be vulnerable due to relaying on a function revert by returning false, instead of reverting directly.</p>
Code	<p>SmartWalletModule.sol line 85 - 87 & line 106 - 108:</p> <pre>ISmartWalletMixinManager(smartWalletMixinManager).check(msg.sender, smartWallet, targetAddress, value, data, operation);</pre>
Result/Recommendation	<p>It is highly recommended to check the return value and revert transaction execution if the submitted data is invalid. This can be done by adding a <code>require</code> statement which requires the return value of <code>SmartWalletMixinManager</code>'s <code>check</code> function to be true.</p>

MEDIUM ISSUES



During the audit, softstack's experts found **one Medium issue** in the code of the smart contract.

6.2.3 Invalid Token Receiver Address Due To Wrong Padded Calldata

Severity: MEDIUM

Status: FIXED

Code: NA

File(s) affected: RecurringPaymentMixin.sol

Update: <https://github.com/SyndicateProtocol/protocol-v2/commit/555b91b28c626d10ee2642b4b5d1d2932496df35>

Attack / Description	<p>The RecurringPaymentMixin checks given calldata for validity. The data is expected to be the hex representation of a token transfer function call, which requires a recipient and a token amount. The calldata is expected to have the following format:</p> <ul style="list-style-type: none">4 bytes function selector (first 4 bytes of the hashed function signature)32 bytes for the first parameter (address of the transfer receiver)32 bytes for the second parameter (amount of the token transfer) <p>The <code>isAllowed</code> function checks each part of the data for validity by checking if the received data bytes are matching the expected parameters. It checks the first 4 bytes for the function selector successfully. Afterwards, it checks the first 32 bytes for the token receiver without respecting the function selector at the beginning, which results in a right padded address value. The resulting address of correctly encoded calldata will never match the expected recipient address and every call will be reverted.</p>
Code	<p>RecurringPaymentMixin.sol line 85 - 88:</p> <pre>address recipientTakenFromCalldata; assembly { recipientTakenFromCalldata := mload(add(add(data, 0x20), 0x0)) }</pre>
Result/Recommendation	<p>It is recommended to change the starting position (0x0) to 0x4 to start reading the 32 bytes for the receiver address after the 4th byte of the call data. This ensures, that first 32 bytes after the function selector is read and the address can be fetched correctly.</p>



--	--

LOW ISSUES

During the audit, softstack's experts found **no Low issues** in the code of the smart contract

INFORMATIONAL ISSUES

During the audit, softstack's experts found **no Informational issues** in the code of the smart contract.

6.3 SWC Attacks

ID	Title	Relationships	Test Result
SWC-131	Presence of unused variables	CWE-1164: Irrelevant Code	✓
SWC-130	Right-To-Left-Override control character (U+202E)	CWE-451: User Interface (UI) Misrepresentation of Critical Information	✓
SWC-129	Typographical Error	CWE-480: Use of Incorrect Operator	✓
SWC-128	DoS With Block Gas Limit	CWE-400: Uncontrolled Resource Consumption	✓

ID	Title	Relationships	Test Result
SWC-127	Arbitrary Jump with Function Type Variable	CWE-695: Use of Low-Level Functionality	✓
SWC-125	Incorrect Inheritance Order	CWE-696: Incorrect Behavior Order	✓
SWC-124	Write to Arbitrary Storage Location	CWE-123: Write-what-where Condition	✓
SWC-123	Requirement Violation	CWE-573: Improper Following of Specification by Caller	✓
SWC-122	Lack of Proper Signature Verification	CWE-345: Insufficient Verification of Data Authenticity	✓
SWC-121	Missing Protection against Signature Replay Attacks	CWE-347: Improper Verification of Cryptographic Signature	✓
SWC-120	Weak Sources of Randomness from Chain Attributes	CWE-330: Use of Insufficiently Random Values	✓
SWC-119	Shadowing State Variables	CWE-710: Improper Adherence to Coding Standards	✓
SWC-118	Incorrect Constructor Name	CWE-665: Improper Initialization	✓
SWC-117	Signature Malleability	CWE-347: Improper Verification of Cryptographic Signature	✓

ID	Title	Relationships	Test Result
SWC-116	Timestamp Dependence	CWE-829: Inclusion of Functionality from Untrusted Control Sphere	✓
SWC-115	Authorization through tx.origin	CWE-477: Use of Obsolete Function	✓
SWC-114	Transaction Order Dependence	CWE-362: Concurrent Execution using Shared Resource with Improper Synchronization ('Race Condition')	✓
SWC-113	DoS with Failed Call	CWE-703: Improper Check or Handling of Exceptional Conditions	✓
SWC-112	Delegatecall to Untrusted Callee	CWE-829: Inclusion of Functionality from Untrusted Control Sphere	✓
SWC-111	Use of Deprecated Solidity Functions	CWE-477: Use of Obsolete Function	✓
SWC-110	Assert Violation	CWE-670: Always-Incorrect Control Flow Implementation	✓
SWC-109	Uninitialized Storage Pointer	CWE-824: Access of Uninitialized Pointer	✓
SWC-108	State Variable Default Visibility	CWE-710: Improper Adherence to Coding Standards	✓
SWC-107	Reentrancy	CWE-841: Improper Enforcement of Behavioral Workflow	✓

ID	Title	Relationships	Test Result
SWC-106	Unprotected SELFDESTRUCT Instruction	CWE-284: Improper Access Control	✓
SWC-105	Unprotected Ether Withdrawal	CWE-284: Improper Access Control	✓
SWC-104	Unchecked Call Return Value	CWE-252: Unchecked Return Value	✓
SWC-103	Floating Pragma	CWE-664: Improper Control of a Resource Through its Lifetime	✓
SWC-102	Outdated Compiler Version	CWE-937: Using Components with Known Vulnerabilities	✓
SWC-101	Integer Overflow and Underflow	CWE-682: Incorrect Calculation	✓
SWC-100	Function Default Visibility	CWE-710: Improper Adherence to Coding Standards	✓

6.4 Verify Claims

6.4.1 The audit must confirm that the SmartWalletModule operates securely and according to the intended logic, particularly verifying that the module strictly adheres to the mapping of allowed operators and that no unauthorized transactions bypass the MixinManager.

Status: tested and verified ✗

6.4.2 It is critical to verify the integrity of the MixinManager and associated Mixins to ensure that they enforce the cluster-wide conditions reliably, without any loopholes that could be exploited to perform unauthorized or malicious transactions.

Status: tested and verified ✓

6.4.3 The auditor should scrutinize the ManagingWallet, especially its role as a 1-of-1 owner of all wallets in a cluster, to validate that it has exclusive control over wallet management functions and that there are no vulnerabilities that could compromise its authority.

Status: tested and verified ✓

6.4.4 For the Operators, the audit must examine the modular contracts to ensure that they authenticate and authorize transactions accurately, and that the custom checks or combinations of checks do not introduce any risks or potential for unauthorized access.

Status: tested and verified ✓

6.4.5 The audit should evaluate the mechanisms that allow the Smart Wallets to interact with external contracts and services, ensuring that these interactions do not expose the wallets to risks such as front-running or unauthorized access from external contracts.

Status: tested and verified ✓

6.5 Unit Tests

Running 3 tests for src/test/smart-wallets/factory/SmartWalletFactoryTest.t.sol:SmartWalletFactoryTest

[PASS] test_createSmartWalletFromFactory() (gas: 1777024)

[PASS] test_ownerCanPause() (gas: 18562)

[PASS] test_ownerCanUnpause() (gas: 13510)

Test result: ok. 3 passed; 0 failed; 0 skipped; finished in 1.29ms

Running 13 tests for src/test/smart-wallets/mixins/RecurringPaymentMixinTest.t.sol:RecurringPaymentMixinTest

[PASS] test_IsAllowedReturnTrue() (gas: 37504)

[PASS] test_RevertWhenAndIncorrectFunctionCallDataIsSent() (gas: 26654)

[PASS] test_RevertWhenIncorrectRecurringPaymentAmount() (gas: 59370)

[PASS] test_RevertWhenIncorrectTargetAddress() (gas: 25153)

[PASS] test_RevertWhenNonOwnerCallsSetPaymentToken() (gas: 12447)

[PASS] test_RevertWhenNonOwnerCallsSetRecipient() (gas: 12394)

[PASS] test_RevertWhenPaymentIntervalHasNotPassed() (gas: 24921)

[PASS] test_RevertWhenTheRecipientOfTheDataIsNotCorrect() (gas: 58742)

[PASS] test_SetNextPaymentTimestamp() (gas: 18503)

[PASS] test_SetPaymentInterval() (gas: 18527)

[PASS] test_SetPaymentToken() (gas: 20505)

[PASS] test_SetRecipient() (gas: 20395)

[PASS] test_SetRecurringPaymentAmount() (gas: 18547)

Test result: ok. 13 passed; 0 failed; 0 skipped; finished in 2.77ms

Running 1 test for src/test/smart-wallets/SmartWalletForkTest.t.sol:SmartWalletForkTest

[PASS] testWrapper() (gas: 286181)

Test result: ok. 1 passed; 0 failed; 0 skipped; finished in 7.02s

Running 12 tests for src/test/smart-wallets/SmartWalletMixinManagerTest.t.sol:SmartWalletMixinManagerTest

[PASS] testCustomMixinsSetButNotDefault() (gas: 116149)
[PASS] testDefaultAndCustomMixinsSet() (gas: 171064)
[PASS] testDefaultAndCustomMixinsSetRevertCustom() (gas: 176733)
[PASS] testDefaultAndCustomMixinsSetRevertCustomDifferentOperator() (gas: 175720)
[PASS] testDefaultAndCustomMixinsSetRevertDefault() (gas: 171525)
[PASS] testDefaultMixinsSet() (gas: 120599)
[PASS] testNoDefaultMixinsSet() (gas: 10751)
[PASS] testRevertDefaultMixinsSet() (gas: 118358)
[PASS] testRevertSetCustomMixins() (gas: 65732)
[PASS] testRevertSetDefaultMixins() (gas: 63433)
[PASS] testSetCustomMixins() (gas: 113745)
[PASS] testSetDefaultMixins() (gas: 110697)

Test result: ok. 12 passed; 0 failed; 0 skipped; finished in 1.70ms

Running 7 tests for src/test/smart-wallets/SmartWalletModuleTest.t.sol:SmartWalletModuleTest

[PASS] testExec() (gas: 26337)
[PASS] testExecMixinManagerNotSet() (gas: 19711)
[PASS] testExecNotOperator() (gas: 18202)
[PASS] testExecWithReturn() (gas: 27941)
[PASS] testExecWithReturnNotOperator() (gas: 20267)
[PASS] testSetMixinManagerNotOwner() (gas: 13245)
[PASS] testSetOperatorsNotOwner() (gas: 54997)

Test result: ok. 7 passed; 0 failed; 0 skipped; finished in 1.22ms

Running 2 tests for src/test/smart-wallets/mixins/NoDelegateCallMixinTest.t.sol:NoDelegateCallMixinTest

[PASS] testCall() (gas: 6600)
[PASS] testRevertDelegateCall() (gas: 9370)

Test result: ok. 2 passed; 0 failed; 0 skipped; finished in 497.82µs

Running 2 tests for src/test/smart-wallets/mixins/NoSelfAuthorizedMixinTest.t.sol:NoSelfAuthorizedMixinTest

[PASS] testCall() (gas: 8722)

[PASS] testRevertCallSelf() (gas: 11551)

Test result: ok. 2 passed; 0 failed; 0 skipped; finished in 632.14µs

Running 13 tests for src/test/smart-wallets/mixins/RolesMixinTest.t.sol:RolesMixinTest

[PASS] test_IsAllowedReturnsTrueWhenAllConditionsAreMet() (gas: 32500)

[PASS] test_RevertWhenCalldataLengthIsTooShort() (gas: 26046)

[PASS] test_RevertWhenFunctionIsNotAllowed() (gas: 44181)

[PASS] test_RevertWhenModuleDoesntHaveARole() (gas: 26347)

[PASS] test_RevertWhenOperatorCannotOperateOnModule() (gas: 31930)

[PASS] test_RevertWhenSendingValueIsNotPermitted() (gas: 33171)

[PASS] test_RevertWhenTargetIsNotAllowed() (gas: 34300)

[PASS] test_RevokeRoleTarget() (gas: 27959)

[PASS] test_SetModuleRole() (gas: 26025)

[PASS] test_SetRoleTarget() (gas: 44313)

[PASS] test_SetScopeFunction() (gas: 50270)

[PASS] test_UpdateScopeTarget() (gas: 45988)

[PASS] test_revokeScopeFunction() (gas: 53023)

Test result: ok. 13 passed; 0 failed; 0 skipped; finished in 1.93ms

Running 5 tests for src/test/smart-wallets/operators/ExecOperatorTest.t.sol:ExecOperatorTest

[PASS] testExec() (gas: 62077)

[PASS] testExecWithReturn() (gas: 63632)

[PASS] testRevertExec() (gas: 60919)

[PASS] testRevertExecWithReturn() (gas: 62922)

[PASS] testRevertWhenRequestIdHasBeenUsed() (gas: 107496)

Test result: ok. 5 passed; 0 failed; 0 skipped; finished in 2.56ms

Ran 9 test suites: 58 tests passed, 0 failed, 0 skipped (58 total tests)

7. Executive Summary

Two independent softstack experts performed an unbiased and isolated audit of the smart contract codebase provided by the Syndicate Team. The main objective of the audit was to verify the security and functionality claims of the smart contract. The audit process involved a thorough manual code review and automated security testing.

Overall, the audit identified a total of 3 issues, classified as follows:

- One critical issues were found.
- One high severity issues were found.
- One medium severity issues were found.
- No low severity issues were discovered
- No informational issues were identified

The audit report provides detailed descriptions of each identified issue, including severity levels, CWE classifications, and recommendations for mitigation. It also includes code snippets, where applicable, to demonstrate the issues and suggest possible fixes. We advise the Syndicate team to implement the recommendations to further enhance the code's security and readability.

Update (13.11.2023): We confirm that all previously identified issues in the Smart Wallet contracts have been effectively addressed and resolved. The updates and fixes implemented by the Syndicate Team have been thoroughly reviewed and meet the required security standards.



8. About the Auditor

Established in 2017 under the name Chainsulting, and rebranded as softstack GmbH in 2023, softstack has been a trusted name in Web3 Security space. Within the rapidly growing Web3 industry, softstack provides a comprehensive range of offerings that include software development, security, and consulting services. Softstack's competency extends across the security landscape of prominent blockchains like Solana, Tezos, Ethereum and Polygon. The company is widely recognized for conducting thorough code audits aimed at mitigating risk and promoting transparency.

The firm's proficiency lies particularly in assessing and fortifying smart contracts of leading DeFi projects, a testament to their commitment to maintaining the integrity of these innovative financial platforms. To date, softstack plays a crucial role in safeguarding over \$100 billion worth of user funds in various DeFi protocols.

Underpinned by a team of industry veterans possessing robust technical knowledge in the Web3 domain, softstack offers industry-leading smart contract audit services. Committed to evolving with their clients' ever-changing business needs, softstack's approach is as dynamic and innovative as the industry it serves.

Check our website for further information: <https://chainsulting.de>

How We Work



1 -----

PREPARATION

Supply our team with audit ready code and additional materials



2 -----

COMMUNICATION

We setup a real-time communication tool of your choice or communicate via e-mails.



3 -----

AUDIT

We conduct the audit, suggesting fixes to all vulnerabilities and help you to improve.



4 -----

FIXES

Your development team applies fixes while consulting with our auditors on their safety.



5 -----

REPORT

We check the applied fixes and deliver a full report on all steps done.

