shieldify

Etherspot

Session Key Validator Module

SECURITY REVIEW

Date: 29 May 2024

CONTENTS

1. About Shieldify

Positioned as the first hybrid Web3 Security company, Shieldify shakes things up with a unique subscription-based auditing model that entitles the customer to unlimited audits within its duration, as well as top-notch service quality thanks to a disruptive 6-layered security approach. The company works with very well-established researchers in the space and have secured multiple millions in TVL across protocols, also can audit codebases written in Solidity, Vyper, Rust, Cairo, Move and Go.

Learn more about us at shieldify.org.

2. Disclaimer

This security review does not guarantee bulletproof protection against a hack or exploit. Smart contracts are a novel technological feat with many known and unknown risks. The protocol, which this report is intended for, indemnifies Shieldify Security against any responsibility for any misbehavior, bugs, or exploits affecting the audited code during any part of the project's life cycle. It is also pivotal to acknowledge that modifications made to the audited code, including fixes for the issues described in this report, may introduce new problems and necessitate additional auditing.

3. About Etherspot - Session Key Validator Module

The `ERC20SessionKeyValidator` is a Solidity smart contract that implements the `IValidator` interface from the `IERC7579Module` contract. It is designed to validate user operations (UserOps) for ERC20 token transfers and approvals based on pre-configured session keys.

Features

- Allows wallet owners to enable and disable session keys for specific ERC20 tokens.
- Supports setting spending limits, validity periods, and pausability for each session key.
- Validates UserOps against the configured session key parameters before execution.
- Provides functions to manage session keys, including rotation and pause/unpause.
- Implements EIP-712 for signature verification.

Usage

To use the `ERC20SessionKeyValidator` contract, wallet owners can enable session keys with specific configurations for ERC20 token transfers and approvals. The contract will validate UserOps against the configured session key parameters before execution, ensuring that the requested operation is within the allowed limits and validity period.

Learn more about ERC20SessionKeyValidator's concept and the technicalities behind it here

4. Risk Classification

4.1 Impact

- **High** - results in a significant risk for the protocol's overall well-being. Affects all or most users
- **Medium** - results in a non-critical risk for the protocol affects all or only a subset of users, but is still unacceptable
- **Low** - losses will be limited but bearable - and covers vectors similar to griefing attacks that can be easily repaired

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

4.2 Likelihood

- **High** – almost certain to happen and highly lucrative for execution by malicious actors
- **Medium** – still relatively likely, although only conditionally possible
- **Low** – requires a unique set of circumstances and poses non-lucrative cost-of-execution to rewards ratio for the actor

5. Security Review Summary

The security review lasted 5 days with a total of 160 hours dedicated to the audit by four researchers from the Shieldify team.

Overall, the code is well-written. The audit report contributed by identifying several Medium and Low severity issues, concerning compliance with EIP-7579, which the module is based on, as well as input validation, among other findings of lower severity.

The Etherspot team has done a great job with their test suite and provided exceptional support and responses to all of the questions that the Shieldify researchers had.

5.1 Protocol Summary

Project Name	**Etherspot – Session Key Validator**
Repository	etherspot-prime-contracts
Type of Project	ERC-7579, EIP-712, ERC20 Session Key Validator Module
Audit Timeline	5 days
Review Commit Hash	dfef1e483837b47a3ea24ab65c3e76a2fb2e0007
Fixes Review Commit Hash	7d6b951e7938ebabd736ae38fd850e212230072d

5.2 Scope

The following smart contracts were in the scope of the security review:

File	nSLOC
src/modular-etherspot-wallet/modules/validators/ERC20SessionKeyValidator.sol	179
src/modular-etherspot-wallet/interfaces/IERC20SessionKeyValidator.sol	17
Total	**196**

6. Findings Summary

The following number of issues have been identified, sorted by their severity:

- **Critical** and **High** issues: **0**
- **Medium** issues: **2**
- **Low** issues: **2**

ID	Title	Severity	Status
[M-01]	Non-compliance with EIP-7579	Medium	Fixed
[M-02]	Not Active Sessions Would Be Considered Valid	Medium	Fixed
[L-01]	Flawed Validation By Session Creation	Low	Fixed
[L-02]	Missing Zero Value Checks in `enableSessionKey()`	Low	Fixed

7. Findings

[M-01] Non-compliance with EIP-7579

Severity

Medium Risk

Description

1. The `onInstall()` function misses a revert statement if the module is already installed.
2. The `onUninstall()` function misses a revert statement if there is an error.
3. The `onUninstall()` function misses to delete from the `walletSessionKeys` mapping for the current user.
4. The `isValidSignatureWithSender()` and `isInitialized()` functions are left unimplemented.

Location of Affected Code

File: src/modular-etherspot-wallet/modules/validators/ERC20SessionKeyValidator.sol

Recommendation

Refactor the code to comply with the EIP. Consider implementing `isValidSignatureWithSender()` and `isInitialized()`, even though the EIP does not enforce the latter, its implementation is straightforward:

1.

```
+ error ERC20SKV_ModuleAlreadyInstalled();

function onInstall(bytes calldata data) external override {
+ if (initialized[msg.sender] == true) revert
    ERC20SKV_ModuleAlreadyInstalled();
  initialized[msg.sender] = true;
}
```

2, 3.

```
+ error ERC20SKV_ModuleNotInstalled();

function onUninstall(bytes calldata data) external override {
+ if (initialized[msg.sender] == false) revert
    ERC20SKV_ModuleNotInstalled();
  address[] memory sessionKeys = getAssociatedSessionKeys();
  for (uint256 i; i < sessionKeys.length; i++) {
    delete sessionData[sessionKeys[i]][msg.sender];
  }

+ delete walletSessionKeys[msg.sender];
  initialized[msg.sender] = false;
}
```

4.

```
function isInitialized(address smartAccount) external view returns (bool)
    {
- revert NotImplemented();
+ return initialized[smartAccount];
}
```

Team Response

Fixed as proposed.

[M-02] Not Active Sessions Would Be Considered Valid

Severity

Medium Risk

Description

The `validateSessionKeyParams()` function doesn't fully ensure the session's validity. If a session is meant to be valid only after a specific period, the function will still validate it successfully even if this period hasn't elapsed yet.

Location of Affected Code

File: src/modular-etherspot-wallet/modules/validators/ERC20SessionKeyValidator.sol#L127

```
function validateSessionKeyParams(
  address _sessionKey,
  PackedUserOperation calldata userOp
) public returns (bool valid) {
// code

@> if (sd.validUntil == 0 || sd.validUntil < block.timestamp)  revert
   ERC20SKV_InvalidSessionKey();

// code
}
```

Recommendation

Consider implementing the following check for the `validAfter` property:

```
- if (sd.validUntil == 0 || sd.validUntil < block.timestamp) revert
   ERC20SKV_InvalidSessionKey();
+ if (sd.validUntil == 0 || sd.validUntil < block.timestamp || sd.
   validAfter == 0 ||
+ sd.validAfter > block.timestamp) revert ERC20SKV_InvalidSessionKey();
```

Team Response

Fixed as proposed.

[L-01] Flawed Validation By Session Creation

Severity

Low Risk

Description

In the `enableSessionKey()` function, an invalid session can be created because the `validAfter` property can be larger than `validUntil`. Moreover, the `sessionKey` should be enforced to be unique, and the execution should revert if the `sessionKey` already exists. Otherwise, the `sessionKey` might be unintentionally overridden.

Location of Affected Code

File: src/modular-etherspot-wallet/modules/validators/ERC20SessionKeyValidator.sol#L61

```
function enableSessionKey(bytes calldata _sessionData) public {
@> address sessionKey = address(bytes20(_sessionData[0:20]));
@> uint48 validAfter = uint48(bytes6(_sessionData[80:86]));
@> uint48 validUntil = uint48(bytes6(_sessionData[86:92]));

// code
}
```

Recommendation

Consider implementing the following validation checks:

```
+ error ERC20SKV_AlreadyExsistingSessionKey(address sessionKey);
+ error ERC20SKV_InvalidDuration(uint48 validUntil, uint48 validAfter);

function enableSessionKey(bytes calldata _sessionData) public {
+ if (sessionData[sessionKey][msg.sender])
+   revert ERC20SKV_AlreadyExsistingSessionKey(sessionKey);

+ if (validUntil <= validAfter)
+   revert ERC20SKV_InvalidDuration(validUntil, validAfter);

// code
}
```

Team Response

Fixed as proposed.

[L-02] Missing Zero Value Checks in `enableSessionKey()`

Severity

Low Risk

Description

In the `enableSessionKey()` function, the `sessionKey`, `token`, `interfaceId`, `funcSelector` and `spendingLimit` parameters lack checks for zero values / addresses.

Location of Affected Code

File: src/modular-etherspot-wallet/modules/validators/ERC20SessionKeyValidator.sol#L61

```
function enableSessionKey(bytes calldata _sessionData) public {
@> address sessionKey = address(bytes20(_sessionData[0:20]));
```

```
@> address token = address(bytes20(_sessionData[20:40]));
@> bytes4 interfaceId = bytes4(_sessionData[40:44]);
@> bytes4 funcSelector = bytes4(_sessionData[44:48]);
@> uint256 spendingLimit = uint256(bytes32(_sessionData[48:80]));
// code
}
```

Recommendation

Consider implementing the following validation checks:

```
+ error ERC20SKV_InvalidSessionKey(address sessionKey);
+ error ERC20SKV_InvalidToken(address token);
+ error ERC20SKV_InvalidInterfaceId(bytes4 interfaceId);
+ error ERC20SKV_InvalidFuncSelector(bytes4 funcSelector);
+ error ERC20SKV_InvalidSpendingLimit(uint256 spendingLimit);

+ if (sessionKey == address(0))
+   revert ERC20SKV_InvalidSessionKey(sessionKey);

+ if (token == address(0))
+   revert ERC20SKV_InvalidToken(token);

+ if (interfaceId == bytes4(0))
+   revert ERC20SKV_InvalidInterfaceId(token);

+ if (funcSelector == bytes4(0))
+   revert ERC20SKV_InvalidFuncSelector(funcSelector);

+ if (spendingLimit == 0)
+   revert ERC20SKV_InvalidSpendingLimit(spendingLimit);
```

Team Response

Fixed as proposed.

shieldify

Thank you!