sevenhash

JetSet

Security Audit

Disclaimer

Security audits cannot uncover all existing vulnerabilities; even an audit in which no vulnerabilities are found is not a guarantee of a secure system. However, code assessments enable the discovery of vulnerabilities that were overlooked during development and areas where additional security measures are necessary. In most cases, applications are either fully protected against a certain type of attack, or they are completely unprotected against it.

Some of the issues may affect the entire application, while some lack protection only in certain areas. This is why we carry out a source code audits aimed at determining all locations that need to be fixed. Within the customer-determined time frame, we performed an audit in order to discover as many vulnerabilities as possible.

The focus of our audit was limited to the code parts defined in the engagement letter. We assessed whether the project follows the provided specifications. These assessments are based on the provided threat model and trust assumptions. We draw attention to the fact that due to inherent limitations in any software development process and software product, an inherent risk exists that even major failures or malfunctions can remain undetected. Further uncertainties exist in any software product or application used during the development, which itself cannot be free from any error or failures.

These preconditions can have an impact on the system's code and/or functions and/or operation. We did not assess the underlying third-party infrastructure which adds further inherent risks as we rely on the correct execution of the included third-party technology stack itself.

Report readers should also take into account that over the life cycle of any software, changes to the product itself or to the environment in which it is operated can have an impact leading to operational behaviors other than those initially determined in the business specification.

Terminology & Risk Classification

For the purpose of this audit, we adopt the following terminology: To classify the severity of our findings, we determine the likelihood and impact (according to the CVSS risk rating methodology).

Likelihood: The likelihood of a finding to be triggered or exploited in practice.
Impact: The technical and business-related consequences of a finding.
Severity: An additive consideration based on the Likelihood and the Impact (as referenced below).

We use the table below to determine the severity of any and all findings. Please note that none of these risk classifications constitute endorsement or opposition to a particular project or contract.

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

Summary

Contracts In-scope:
Jetset.sol

BSCscan:
https://bscscan.com/address/0xed9f7e6bc5a85fa352be7335b092024832000e29#code

In the period 2 Jan. 2024 - 4 Jan. 2024 we audited Jetset smart contract.

In this period of time a total of 4 of issues were found.

Severity	Issues
Critical Risk	0
High Risk	1
Medium Risk	0
Low Risk	1
Gas Optimization	0
Informational	2

Issues

[Informational - 001]: Absence of Address Validation for Router in Constructor

Description: The constructor does not include a verification step to ascertain whether the router's address is either the zero address or the dead address.

Remediation: Remediation: Implement a validation check to confirm that the router address is not set to the zero address or the dead address. This recommendation is categorized as informational because it represents a best practice in parameter validation to ensure robust contract behavior.

[Low - 001]: Lack of Validation for Marketing Address

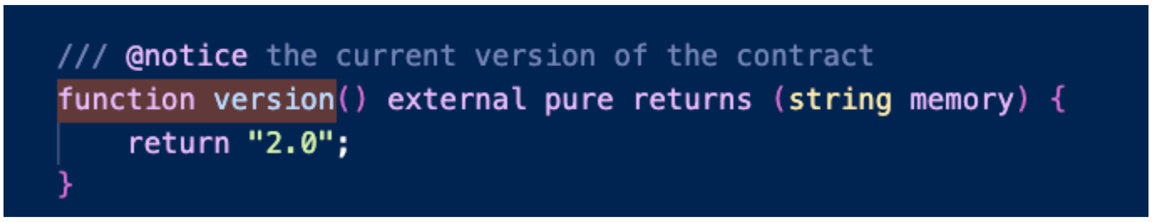
Description: The smart contract allows deployment without verifying the status of the marketing address, which could be set to either the zero address or the dead address. Should the marketing address be zero or dead, all functions invoking executeMarketingSwap() will erroneously direct the marketing fees to these non-functional addresses.

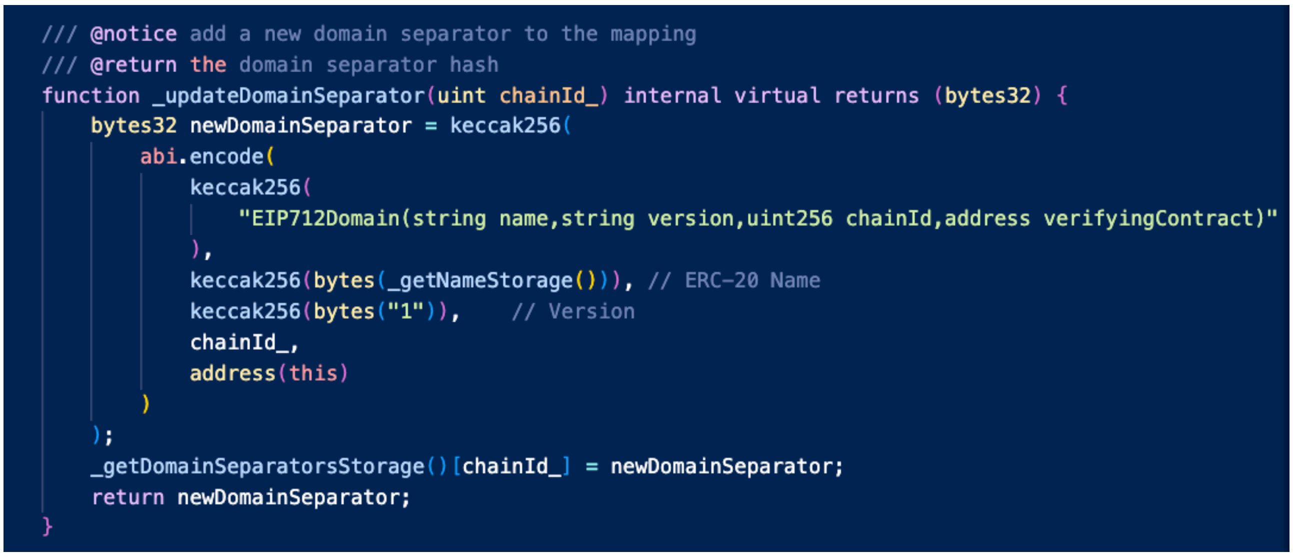
Remediation: Introduce a validation step to ensure the marketing address is neither the zero address nor the dead address. This check should be implemented both in the contract's constructor and within the setMarketingAddress() function.

[Informational - 002]: Inconsistency in Version Usage for Domain Separator Calculation

Description: There is a mismatch in the versioning used within the smart contract. Specifically, while computing the domain separator, the contract employs version 1, despite the smart contract itself being at version 2.

Remediation: Ensure consistency by using the same version number for the domain separator calculation as the current version of the smart contract.


```
/// @notice the current version of the contract
function version() external pure returns (string memory) {
    return "2.0";
}
```

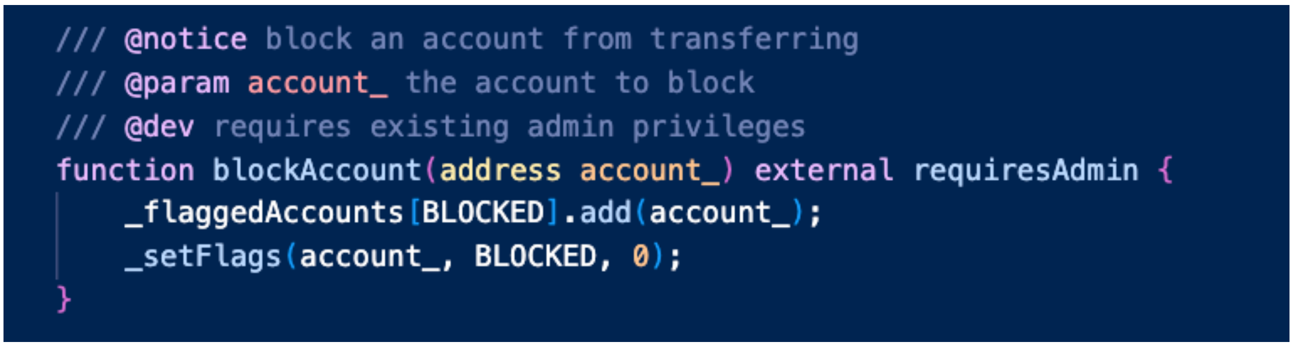


```
/// @notice add a new domain separator to the mapping
/// @return the domain separator hash
function _updateDomainSeparator(uint chainId_) internal virtual returns (bytes32) {
    bytes32 newDomainSeparator = keccak256(
        abi.encode(
            keccak256(
                "EIP712Domain(string name,string version,uint256 chainId,address verifyingContract)"
            ),
            keccak256(bytes(_getNameStorage())), // ERC-20 Name
            keccak256(bytes("1")),    // Version
            chainId_,
            address(this)
        )
    );
    _getDomainSeparatorsStorage()[chainId_] = newDomainSeparator;
    return newDomainSeparator;
}
```

[High - 001]: Owner's Ability to Block Addresses

Description: The smart contract grants the owner the capability to prevent a specific address from transferring or selling tokens. This functionality introduces a significant centralization concern, as it allows the owner to exert undue control over the token's transferability.

Remediation: Eliminate the functions that enable the owner to blacklist an address. This action will help decentralize control and ensure equitable token transfer capabilities for all users.



```
/// @notice block an account from transferring
/// @param account_ the account to block
/// @dev requires existing admin privileges
function blockAccount(address account_) external requiresAdmin {
    _flaggedAccounts[BLOCKED].add(account_);
    _setFlags(account_, BLOCKED, 0);
}
```