Guard

Smart contracts security assessment

Final report

Tariff: Standard

SatoshiDEX Token

March 2024

0xguard.com

hello@0xguard.com

Contents

Introduction

The report has been prepared for **SatoshiDEX Token**.

$SATX (SATX) is an ERC-20 standard token with ERC20Burnable and ERC20Permit extensions made by OpenZeppelin. The token has no mint functionality, no taxes.

The contract is available at 0x9F9bb3D5Af7cC774F9b6ADF66E32859B5a998952 in the BNB Smart Chain.

Name	SatoshiDEX Token
Audit date	2024-03-14 - 2024-03-15
Language	Solidity
Platform	Binance Smart Chain

Contracts checked

Name	Address
SatxToken	0x9F9bb3D5Af7cC774F9b6ADF66E32859B5a998952

Procedure

We perform our audit according to the following procedure:

Automated analysis

- Scanning the project's smart contracts with several publicly available automated Solidity analysis tools
- Manual verification (reject or confirm) all the issues found by the tools

Manual audit

- Manually analyze smart contracts for security vulnerabilities
- Smart contracts' logic check

Known vulnerabilities checked

Title	Check result
Unencrypted Private Data On-Chain	passed
Code With No Effects	passed
Message call with hardcoded gas amount	passed
Typographical Error	passed
DoS With Block Gas Limit	passed
Presence of unused variables	passed
Incorrect Inheritance Order	passed
Requirement Violation	passed
Weak Sources of Randomness from Chain Attributes	passed
Shadowing State Variables	passed
Incorrect Constructor Name	passed
Block values as a proxy for time	passed
Authorization through tx.origin	passed
DoS with Failed Call	passed
Delegatecall to Untrusted Callee	passed
Use of Deprecated Solidity Functions	passed
Assert Violation	passed
State Variable Default Visibility	passed
Reentrancy	passed

Unprotected SELFDESTRUCT Instruction	passed
Unprotected Ether Withdrawal	passed
Unchecked Call Return Value	passed
Floating Pragma	passed
Outdated Compiler Version	passed
Integer Overflow and Underflow	passed
Function Default Visibility	passed

Classification of issue severity

High severity	High severity issues can cause a significant or full loss of funds, change of contract ownership, major interference with contract logic. Such issues require immediate attention.
Medium severity	Medium severity issues do not pose an immediate risk, but can be detrimental to the client's reputation if exploited. Medium severity issues may lead to a contract failure and can be fixed by modifying the contract state or redeployment. Such issues require attention.
Low severity	Low severity issues do not cause significant destruction to the contract's functionality. Such issues are recommended to be taken into consideration.

Issues

High severity issues

No issues were found

Medium severity issues

No issues were found

Low severity issues

No issues were found

Conclusion

SatoshiDEX Token SatxToken contract was audited. No severity issues were found.

Disclaimer

This report is subject to the terms and conditions (including without limitation, description of services, confidentiality, disclaimer and limitation of liability)set forth in the Services Agreement, or the scope of services, and terms and conditions provided to the Company in connection with the Agreement. This report provided in connection with the Services set forth in the Agreement shall be used by the Company only to the extent permitted under the terms and conditions set forth in the Agreement. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes without 0xGuard prior written consent.

This report is not, nor should be considered, an “endorsement” or “disapproval” of any particular project or team. This report is not, nor should be considered, an indication of the economics or value of any “product” or “asset” created by any team or project that contracts 0xGuard to perform a security assessment. This report does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors, business, business model or legal compliance.

This report should not be used in any way to make decisions around investment or involvement with any particular project. This report in no way provides investment advice, nor should be leveraged as investment advice of any sort. This report represents an extensive assessing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Static code analysis

```
INFO:Detectors:
Math.mulDiv(uint256,uint256,uint256) (@openzeppelin/contracts/utils/math/
Math.sol#123-202) performs a multiplication on the result of a division:
- denominator = denominator / twos (@openzeppelin/contracts/utils/math/Math.sol#169)
- inverse = (3 * denominator) ^ 2 (@openzeppelin/contracts/utils/math/Math.sol#184)
Math.mulDiv(uint256,uint256,uint256) (@openzeppelin/contracts/utils/math/
Math.sol#123-202) performs a multiplication on the result of a division:
- denominator = denominator / twos (@openzeppelin/contracts/utils/math/Math.sol#169)
- inverse *= 2 - denominator * inverse (@openzeppelin/contracts/utils/math/
Math.sol#188)
Math.mulDiv(uint256,uint256,uint256) (@openzeppelin/contracts/utils/math/
Math.sol#123-202) performs a multiplication on the result of a division:
- denominator = denominator / twos (@openzeppelin/contracts/utils/math/Math.sol#169)
- inverse *= 2 - denominator * inverse (@openzeppelin/contracts/utils/math/
Math.sol#189)
Math.mulDiv(uint256,uint256,uint256) (@openzeppelin/contracts/utils/math/
Math.sol#123-202) performs a multiplication on the result of a division:
- denominator = denominator / twos (@openzeppelin/contracts/utils/math/Math.sol#169)
- inverse *= 2 - denominator * inverse (@openzeppelin/contracts/utils/math/
Math.sol#190)
Math.mulDiv(uint256,uint256,uint256) (@openzeppelin/contracts/utils/math/
Math.sol#123-202) performs a multiplication on the result of a division:
- denominator = denominator / twos (@openzeppelin/contracts/utils/math/Math.sol#169)
- inverse *= 2 - denominator * inverse (@openzeppelin/contracts/utils/math/
Math.sol#191)
Math.mulDiv(uint256,uint256,uint256) (@openzeppelin/contracts/utils/math/
Math.sol#123-202) performs a multiplication on the result of a division:
- denominator = denominator / twos (@openzeppelin/contracts/utils/math/Math.sol#169)
- inverse *= 2 - denominator * inverse (@openzeppelin/contracts/utils/math/
Math.sol#192)
Math.mulDiv(uint256,uint256,uint256) (@openzeppelin/contracts/utils/math/
Math.sol#123-202) performs a multiplication on the result of a division:
- denominator = denominator / twos (@openzeppelin/contracts/utils/math/Math.sol#169)
- inverse *= 2 - denominator * inverse (@openzeppelin/contracts/utils/math/
Math.sol#193)
Math.mulDiv(uint256,uint256,uint256) (@openzeppelin/contracts/utils/math/
Math.sol#123-202) performs a multiplication on the result of a division:
```

```
- prod0 = prod0 / twos (@openzeppelin/contracts/utils/math/Math.sol#172)
- result = prod0 * inverse (@openzeppelin/contracts/utils/math/Math.sol#199)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#divide-before-multiply
INFO:Detectors:
ERC20Permit.constructor(string).name (@openzeppelin/contracts/token/ERC20/extensions/ERC20Permit.sol#39) shadows:
- ERC20.name() (@openzeppelin/contracts/token/ERC20/ERC20.sol#58-60) (function)
- IERC20Metadata.name() (@openzeppelin/contracts/token/ERC20/extensions/IERC20Metadata.sol#15) (function)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#local-variable-shadowing
INFO:Detectors:
ERC20Permit.permit(address,address,uint256,uint256,uint8,bytes32,bytes32) (@openzeppelin/contracts/token/ERC20/extensions/ERC20Permit.sol#44-67) uses timestamp for comparisons
Dangerous comparisons:
- block.timestamp > deadline (@openzeppelin/contracts/token/ERC20/extensions/ERC20Permit.sol#53)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp
INFO:Detectors:
ShortStrings.toString(ShortString) (@openzeppelin/contracts/utils/ShortStrings.sol#63-73) uses assembly
- INLINE ASM (@openzeppelin/contracts/utils/ShortStrings.sol#68-71)
StorageSlot.getAddressSlot(bytes32) (@openzeppelin/contracts/utils/StorageSlot.sol#59-64) uses assembly
- INLINE ASM (@openzeppelin/contracts/utils/StorageSlot.sol#61-63)
StorageSlot.getBooleanSlot(bytes32) (@openzeppelin/contracts/utils/StorageSlot.sol#69-74) uses assembly
- INLINE ASM (@openzeppelin/contracts/utils/StorageSlot.sol#71-73)
StorageSlot.getBytes32Slot(bytes32) (@openzeppelin/contracts/utils/StorageSlot.sol#79-84) uses assembly
- INLINE ASM (@openzeppelin/contracts/utils/StorageSlot.sol#81-83)
StorageSlot.getUint256Slot(bytes32) (@openzeppelin/contracts/utils/StorageSlot.sol#89-94) uses assembly
- INLINE ASM (@openzeppelin/contracts/utils/StorageSlot.sol#91-93)
StorageSlot.getStringSlot(bytes32) (@openzeppelin/contracts/utils/StorageSlot.sol#99-104) uses assembly
- INLINE ASM (@openzeppelin/contracts/utils/StorageSlot.sol#101-103)
StorageSlot.getStringSlot(string) (@openzeppelin/contracts/utils/
```

```
StorageSlot.sol#109-114) uses assembly
 - INLINE ASM (@openzeppelin/contracts/utils/StorageSlot.sol#111-113)
StorageSlot.getBytesSlot(bytes32) (@openzeppelin/contracts/utils/
StorageSlot.sol#119-124) uses assembly
 - INLINE ASM (@openzeppelin/contracts/utils/StorageSlot.sol#121-123)
StorageSlot.getBytesSlot(bytes) (@openzeppelin/contracts/utils/StorageSlot.sol#129-134)
uses assembly
 - INLINE ASM (@openzeppelin/contracts/utils/StorageSlot.sol#131-133)
Strings.toString(uint256) (@openzeppelin/contracts/utils/Strings.sol#24-44) uses
assembly
 - INLINE ASM (@openzeppelin/contracts/utils/Strings.sol#30-32)
 - INLINE ASM (@openzeppelin/contracts/utils/Strings.sol#36-38)
ECDSA.tryRecover(bytes32,bytes) (@openzeppelin/contracts/utils/cryptography/
ECDSA.sol#56-73) uses assembly
 - INLINE ASM (@openzeppelin/contracts/utils/cryptography/ECDSA.sol#64-68)
MessageHashUtils.toEthSignedMessageHash(bytes32) (@openzeppelin/contracts/utils/
cryptography/MessageHashUtils.sol#30-37) uses assembly
 - INLINE ASM (@openzeppelin/contracts/utils/cryptography/MessageHashUtils.sol#32-36)
MessageHashUtils.toTypedDataHash(bytes32,bytes32) (@openzeppelin/contracts/utils/
cryptography/MessageHashUtils.sol#76-85) uses assembly
 - INLINE ASM (@openzeppelin/contracts/utils/cryptography/MessageHashUtils.sol#78-84)
Math.mulDiv(uint256,uint256,uint256) (@openzeppelin/contracts/utils/math/
Math.sol#123-202) uses assembly
 - INLINE ASM (@openzeppelin/contracts/utils/math/Math.sol#130-133)
 - INLINE ASM (@openzeppelin/contracts/utils/math/Math.sol#154-161)
 - INLINE ASM (@openzeppelin/contracts/utils/math/Math.sol#167-176)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage
INFO:Detectors:
Different versions of Solidity are used:
 - Version used: ['0.8.24', '^0.8.20']
 - 0.8.24 (contracts/SatxToken.sol#3)
 - ^0.8.20 (@openzeppelin/contracts/interfaces/IERC5267.sol#4)
 - ^0.8.20 (@openzeppelin/contracts/interfaces/draft-IERC6093.sol#3)
 - ^0.8.20 (@openzeppelin/contracts/token/ERC20/ERC20.sol#4)
 - ^0.8.20 (@openzeppelin/contracts/token/ERC20/IERC20.sol#4)
 - ^0.8.20 (@openzeppelin/contracts/token/ERC20/extensions/ERC20Burnable.sol#4)
 - ^0.8.20 (@openzeppelin/contracts/token/ERC20/extensions/ERC20Permit.sol#4)
 - ^0.8.20 (@openzeppelin/contracts/token/ERC20/extensions/IERC20Metadata.sol#4)
 - ^0.8.20 (@openzeppelin/contracts/token/ERC20/extensions/IERC20Permit.sol#4)
 - ^0.8.20 (@openzeppelin/contracts/utils/Context.sol#4)
 - ^0.8.20 (@openzeppelin/contracts/utils/Nonces.sol#3)
```

```
 - ^0.8.20 (@openzeppelin/contracts/utils/ShortStrings.sol#4)
 - ^0.8.20 (@openzeppelin/contracts/utils/StorageSlot.sol#5)
 - ^0.8.20 (@openzeppelin/contracts/utils/Strings.sol#4)
 - ^0.8.20 (@openzeppelin/contracts/utils/cryptography/ECDSA.sol#4)
 - ^0.8.20 (@openzeppelin/contracts/utils/cryptography/EIP712.sol#4)
 - ^0.8.20 (@openzeppelin/contracts/utils/cryptography/MessageHashUtils.sol#4)
 - ^0.8.20 (@openzeppelin/contracts/utils/math/Math.sol#4)
 - ^0.8.20 (@openzeppelin/contracts/utils/math/SignedMath.sol#4)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#different-
pragma-directives-are-used
INFO:Detectors:
Context._contextSuffixLength() (@openzeppelin/contracts/utils/Context.sol#25-27) is
never used and should be removed
Context._msgData() (@openzeppelin/contracts/utils/Context.sol#21-23) is never used and
should be removed
ECDSA.recover(bytes32,bytes) (@openzeppelin/contracts/utils/cryptography/
ECDSA.sol#89-93) is never used and should be removed
ECDSA.recover(bytes32,bytes32,bytes32) (@openzeppelin/contracts/utils/cryptography/
ECDSA.sol#112-116) is never used and should be removed
ECDSA.tryRecover(bytes32,bytes) (@openzeppelin/contracts/utils/cryptography/
ECDSA.sol#56-73) is never used and should be removed
ECDSA.tryRecover(bytes32,bytes32,bytes32) (@openzeppelin/contracts/utils/cryptography/
ECDSA.sol#100-107) is never used and should be removed
Math.average(uint256,uint256) (@openzeppelin/contracts/utils/math/Math.sol#96-99) is
never used and should be removed
Math.ceilDiv(uint256,uint256) (@openzeppelin/contracts/utils/math/Math.sol#107-115) is
never used and should be removed
Math.log10(uint256) (@openzeppelin/contracts/utils/math/Math.sol#321-353) is never used
and should be removed
Math.log10(uint256,Math.Rounding) (@openzeppelin/contracts/utils/math/Math.sol#359-364)
is never used and should be removed
Math.log2(uint256) (@openzeppelin/contracts/utils/math/Math.sol#268-304) is never used
and should be removed
Math.log2(uint256,Math.Rounding) (@openzeppelin/contracts/utils/math/Math.sol#310-315)
is never used and should be removed
Math.log256(uint256) (@openzeppelin/contracts/utils/math/Math.sol#372-396) is never
used and should be removed
Math.log256(uint256,Math.Rounding) (@openzeppelin/contracts/utils/math/
Math.sol#402-407) is never used and should be removed
Math.max(uint256,uint256) (@openzeppelin/contracts/utils/math/Math.sol#81-83) is never
used and should be removed
```

```
Math.min(uint256,uint256) (@openzeppelin/contracts/utils/math/Math.sol#88-90) is never
used and should be removed
Math.mulDiv(uint256,uint256,uint256) (@openzeppelin/contracts/utils/math/
Math.sol#123-202) is never used and should be removed
Math.mulDiv(uint256,uint256,uint256,Math.Rounding) (@openzeppelin/contracts/utils/math/
Math.sol#207-213) is never used and should be removed
Math.sqrt(uint256) (@openzeppelin/contracts/utils/math/Math.sol#221-252) is never used
and should be removed
Math.sqrt(uint256,Math.Rounding) (@openzeppelin/contracts/utils/math/Math.sol#257-262)
is never used and should be removed
Math.tryAdd(uint256,uint256) (@openzeppelin/contracts/utils/math/Math.sol#25-31) is
never used and should be removed
Math.tryDiv(uint256,uint256) (@openzeppelin/contracts/utils/math/Math.sol#61-66) is
never used and should be removed
Math.tryMod(uint256,uint256) (@openzeppelin/contracts/utils/math/Math.sol#71-76) is
never used and should be removed
Math.tryMul(uint256,uint256) (@openzeppelin/contracts/utils/math/Math.sol#46-56) is
never used and should be removed
Math.trySub(uint256,uint256) (@openzeppelin/contracts/utils/math/Math.sol#36-41) is
never used and should be removed
Math.unsignedRoundsUp(Math.Rounding) (@openzeppelin/contracts/utils/math/
Math.sol#412-414) is never used and should be removed
MessageHashUtils.toDataWithIntendedValidatorHash(address,bytes) (@openzeppelin/
contracts/utils/cryptography/MessageHashUtils.sol#63-65) is never used and should be
removed
MessageHashUtils.toEthSignedMessageHash(bytes) (@openzeppelin/contracts/utils/
cryptography/MessageHashUtils.sol#49-52) is never used and should be removed
MessageHashUtils.toEthSignedMessageHash(bytes32) (@openzeppelin/contracts/utils/
cryptography/MessageHashUtils.sol#30-37) is never used and should be removed
Nonces._useCheckedNonce(address,uint256) (@openzeppelin/contracts/utils/
Nonces.sol#40-45) is never used and should be removed
ShortStrings.byteLengthWithFallback(ShortString,string) (@openzeppelin/contracts/utils/
ShortStrings.sol#116-122) is never used and should be removed
SignedMath.abs(int256) (@openzeppelin/contracts/utils/math/SignedMath.sol#37-42) is
never used and should be removed
SignedMath.average(int256,int256) (@openzeppelin/contracts/utils/math/
SignedMath.sol#28-32) is never used and should be removed
SignedMath.max(int256,int256) (@openzeppelin/contracts/utils/math/SignedMath.sol#13-15)
is never used and should be removed
SignedMath.min(int256,int256) (@openzeppelin/contracts/utils/math/SignedMath.sol#20-22)
is never used and should be removed
```

```
StorageSlot.getAddressSlot(bytes32) (@openzeppelin/contracts/utils/
StorageSlot.sol#59-64) is never used and should be removed
StorageSlot.getBooleanSlot(bytes32) (@openzeppelin/contracts/utils/
StorageSlot.sol#69-74) is never used and should be removed
StorageSlot.getBytes32Slot(bytes32) (@openzeppelin/contracts/utils/
StorageSlot.sol#79-84) is never used and should be removed
StorageSlot.getBytesSlot(bytes) (@openzeppelin/contracts/utils/StorageSlot.sol#129-134)
is never used and should be removed
StorageSlot.getBytesSlot(bytes32) (@openzeppelin/contracts/utils/
StorageSlot.sol#119-124) is never used and should be removed
StorageSlot.getStringSlot(bytes32) (@openzeppelin/contracts/utils/
StorageSlot.sol#99-104) is never used and should be removed
StorageSlot.getUint256Slot(bytes32) (@openzeppelin/contracts/utils/
StorageSlot.sol#89-94) is never used and should be removed
Strings.equal(string,string) (@openzeppelin/contracts/utils/Strings.sol#91-93) is never
used and should be removed
Strings.toHexString(address) (@openzeppelin/contracts/utils/Strings.sol#84-86) is never
used and should be removed
Strings.toHexString(uint256) (@openzeppelin/contracts/utils/Strings.sol#56-60) is never
used and should be removed
Strings.toHexString(uint256,uint256) (@openzeppelin/contracts/utils/Strings.sol#65-78)
is never used and should be removed
Strings.toString(uint256) (@openzeppelin/contracts/utils/Strings.sol#24-44) is never
used and should be removed
Strings.toStringSigned(int256) (@openzeppelin/contracts/utils/Strings.sol#49-51) is
never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code
INFO:Detectors:
Pragma version^0.8.20 (@openzeppelin/contracts/interfaces/IERC5267.sol#4) necessitates
a version too recent to be trusted. Consider deploying with 0.8.18.
Pragma version^0.8.20 (@openzeppelin/contracts/interfaces/draft-IERC6093.sol#3)
necessitates a version too recent to be trusted. Consider deploying with 0.8.18.
Pragma version^0.8.20 (@openzeppelin/contracts/token/ERC20/ERC20.sol#4) necessitates a
version too recent to be trusted. Consider deploying with 0.8.18.
Pragma version^0.8.20 (@openzeppelin/contracts/token/ERC20/IERC20.sol#4) necessitates a
version too recent to be trusted. Consider deploying with 0.8.18.
Pragma version^0.8.20 (@openzeppelin/contracts/token/ERC20/extensions/
ERC20Burnable.sol#4) necessitates a version too recent to be trusted. Consider
deploying with 0.8.18.
Pragma version^0.8.20 (@openzeppelin/contracts/token/ERC20/extensions/
ERC20Permit.sol#4) necessitates a version too recent to be trusted. Consider deploying
```

```
with 0.8.18.
Pragma version^0.8.20 (@openzeppelin/contracts/token/ERC20/extensions/
IERC20Metadata.sol#4) necessitates a version too recent to be trusted. Consider
deploying with 0.8.18.
Pragma version^0.8.20 (@openzeppelin/contracts/token/ERC20/extensions/
IERC20Permit.sol#4) necessitates a version too recent to be trusted. Consider deploying
with 0.8.18.
Pragma version^0.8.20 (@openzeppelin/contracts/utils/Context.sol#4) necessitates a
version too recent to be trusted. Consider deploying with 0.8.18.
Pragma version^0.8.20 (@openzeppelin/contracts/utils/Nonces.sol#3) necessitates a
version too recent to be trusted. Consider deploying with 0.8.18.
Pragma version^0.8.20 (@openzeppelin/contracts/utils/ShortStrings.sol#4) necessitates a
version too recent to be trusted. Consider deploying with 0.8.18.
Pragma version^0.8.20 (@openzeppelin/contracts/utils/StorageSlot.sol#5) necessitates a
version too recent to be trusted. Consider deploying with 0.8.18.
Pragma version^0.8.20 (@openzeppelin/contracts/utils/Strings.sol#4) necessitates a
version too recent to be trusted. Consider deploying with 0.8.18.
Pragma version^0.8.20 (@openzeppelin/contracts/utils/cryptography/ECDSA.sol#4)
necessitates a version too recent to be trusted. Consider deploying with 0.8.18.
Pragma version^0.8.20 (@openzeppelin/contracts/utils/cryptography/EIP712.sol#4)
necessitates a version too recent to be trusted. Consider deploying with 0.8.18.
Pragma version^0.8.20 (@openzeppelin/contracts/utils/cryptography/
MessageHashUtils.sol#4) necessitates a version too recent to be trusted. Consider
deploying with 0.8.18.
Pragma version^0.8.20 (@openzeppelin/contracts/utils/math/Math.sol#4) necessitates a
version too recent to be trusted. Consider deploying with 0.8.18.
Pragma version^0.8.20 (@openzeppelin/contracts/utils/math/SignedMath.sol#4)
necessitates a version too recent to be trusted. Consider deploying with 0.8.18.
Pragma version0.8.24 (contracts/SatxToken.sol#3) necessitates a version too recent to
be trusted. Consider deploying with 0.8.18.
solc-0.8.24 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-
versions-of-solidity
INFO:Detectors:
Function ERC20Permit.DOMAIN_SEPARATOR() (@openzeppelin/contracts/token/ERC20/extensions/
ERC20Permit.sol#80-82) is not in mixedCase
Function IERC20Permit.DOMAIN_SEPARATOR() (@openzeppelin/contracts/token/ERC20/
extensions/IERC20Permit.sol#89) is not in mixedCase
Function EIP712._EIP712Name() (@openzeppelin/contracts/utils/cryptography/
EIP712.sol#146-148) is not in mixedCase
Function EIP712._EIP712Version() (@openzeppelin/contracts/utils/cryptography/
```

```
EIP712.sol#157-159) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-
solidity-naming-conventions
INFO:Detectors:
ShortStrings.slitherConstructorConstantVariables() (@openzeppelin/contracts/utils/
ShortStrings.sol#40-123) uses literals with too many digits:
	- FALLBACK_SENTINEL =
0x00000000000000000000000000000000000000000000000000000000000000FF (@openzeppelin/
contracts/utils/ShortStrings.sol#42)
SatxToken.constructor() (contracts/SatxToken.sol#10-12) uses literals with too many
digits:
	- _mint(msg.sender,10000000000 * 10 ** decimals()) (contracts/SatxToken.sol#11)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#too-many-
digits
INFO:Slither:. analyzed (21 contracts with 85 detectors), 99 result(s) found
```

0x Guard