Digital Security

БЕЗОПАСНОСТЬ КАК ИСКУССТВО

EXPERT ANALYSIS

based on the source code audit of the OTN Token smart contract

Evaluation dates:
25.09.2017-29.09.2017

1. Introduction

1.1.Overview

This expert analysis presents the results of the security evaluation of the source code for the smart contract (the System) owned by OTN Foundation (the Company), as well as recommendations for eliminating the weaknesses (vulnerabilities) that were discovered and for increasing the level of security.

1.2.Abbreviations

Table 1.2-1. Abbreviations

Abbreviation	Meaning
DS	Data Security
IS	Information System
EVM	Ethereum virtual machine
ERC20	The adopted token standard in the Ethereum ecosystem
Ether	Cryptocurrency in the Ethereum network

1.3.Summary

Experts of the company Digital Security analyzed the security of the source code for the System's smart contract during the period from 25.09.17 to 29.09.17.

The following attacker models were considered:

- Outsider attack from the Ethereum network.
- Insider attack (one of the chosen owners).

Analysis showed that:

1. The security level of the smart contract can be classified as high.
2. Most of the weaknesses discovered have a low level of severity.

Main recommendations:

1. Eliminate the technical weaknesses that were identified.
2. Establish regulations for working with a smart contract.

The following describes the identified shortcomings and associated data security risks, as well as detailed recommendations for their elimination.

2. How the evaluation was performed

2.1.Threats to data security

The following three DS threats may affect the Company's information resources: breach of confidentiality, violation of integrity, and loss of availability.

The threat to confidentiality is aimed at disclosing information that is confidential in the Company. When the threat is carried out, information becomes known to persons who should not have access to it – a number of Company employees, customers, partners, competitors, or third parties.

The threat to integrity is aimed at modifying or distorting information, leading to a change in its structure or meaning, or complete or partial destruction.

The threat to availability means that users of the information system cannot access informational resources (denial of service).

The main purpose of the DS audit is to check whether these threats could be carried out using the given attacker model to affect the System's informational resources.

2.2. Attacker model

A potential attacker of the Company's System is viewed as a person or group of people who may or may not be in collusion, who either intentionally or unintentionally may potentially threaten data security, intrude on the System's informational resources, and harm the interests of the Company.

DS threats are viewed as basic threats to breach confidentiality and violate data integrity, as well as the threat that the System will fail to provide services to the Company's customers.

A deliberate attacker may pursue the following objectives (and also their various combinations):

- Malicious disruption of services (denial-of-service attack).
- Increasing their privileges.
- Unauthorized changes to business-critical information.

The evaluation used outsider and insider attacker models.

2.2.1.Outsider attack

The following outsider attacker model was used in the penetration test:

- An external intruder from the Internet who has access to the Ethereum network and has knowledge of the System being tested (since the contract is open source), but does not have rights in it.

2.2.2.Insider attack

The following insider attacker models was used in the penetration test:

- An internal intruder from the Ethereum network who has knowledge of the System being tested and has rights in it.

2.3.Scope of testing

During the security evaluation, the following source code files were analyzed:

Table 2.3 – 1. Scope of testing

File	Smart Contract
SafeMath.sol	SafeMath – implementation of secure calculations
Shareable.sol	Shareable – implementation of distributed management
BasicToken.sol	BasicToken – basic implementation of the ERC20 token
ERC20.sol	ERC20 – abstract class
OTNToken.sol	OTNToken – implementation of the OTN token
MintableToken.sol	MintableToken – implementation of the "issuable" token
StandartToken.sol	StandartToken – an extension of the ERC20 standard

3. Analysis of the smart contract source code

3.1. Vulnerabilities detected

3.1.1. Function is susceptible to Short Address Attack

Severity: high
Exploitability: low
Overall risk: medium

Description:
A compatible token is susceptible to a Short Address Attack, which allows the contract logic to be changed due to features of EMV.

Risk:
An attacker can transfer many more tokens than expected from the victim's account.

Vulnerable resource:

- Mintable.sol

Technical details:
This attack is possible because of the combination of two factors. First, EVM inserts zero bytes where transaction data is missing. Second, if the address used as an argument to the function on the client side is not fully validated, an attacker can send a specially created address to the victim for sending tokens. The special address is shorter than a normal address by several bytes (zero bytes). On the client side, a transaction will be formed that is shorter than usual.
Since EVM inserts zero bytes to replace the missing ones at the end of the data field, the value of the number of transmitted tokens will be multiplied by $2^{(X*8)}$, where X is the number of omitted null bytes at the end of the attacker's address.
The 'mint' function (address _to, uint256 _amount) is susceptible to this attack. An attacker can provide an address with zero bytes omitted at the end, and, if it is not correctly processed on the client side, a much larger number of tokens than expected will be generated at the attacker's address.

Recommended:

- Since the 'mint' function is not part of the ERC20 standard, we recommend switching places for the _to and _amount arguments.

3.1.2.Desyncing object data

Severity: medium
Exploitability: low
Overall risk: medium

Description:

Information about an object is stored in multiple variables, but synchronization is not provided.

Risk:

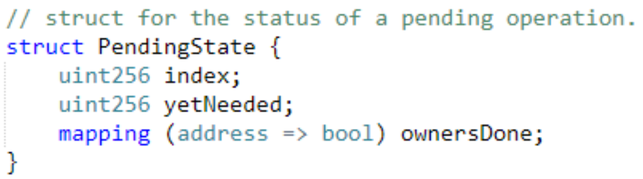
Legitimate use of the contract could lead to it being blocked.

Vulnerable resource:

- Shareable.sol

Technical details:

The Shareable contract uses mapping pendings and the pendingsIndex array to store pending operations.
The structure



```
// struct for the status of a pending operation.
struct PendingState {
    uint256 index;
    uint256 yetNeeded;
    mapping (address => bool) ownersDone;
}
```

stores the index of the pending operation, but **PendingState.index** is not synced with the position of 'pending' in pendingsIndex. This can be seen in the confirmAndCheck function (lines 256-258): the last pending operation is mixed inside pendingsIndex, but its index in the structure stored in 'mapping pendings' does not change. Considering the behavior described in section 3.2.4 (transaction processing), this could result in the contract being blocked or its logic being violated.

Recommended:

- Synchronize the specified parameters.

3.2.List of weaknesses found

3.2.1.Functionality inaccessible

Description:

The 'approve' function in the StandardToken contract is not available to the Truster if they have already used it and the Trustee has not yet transferred the entire amount of allowed tokens.

Risk:

The Truster might change their mind regarding the number of trusted tokens to transmit, but the contract will not allow them to.

Vulnerable resource:

- StandardToken.sol

Technical details:

The 'approve' function in the StandardToken contract has a check (line 50) that does not allow the Truster to change their decision regarding the amount of trusted tokens.

Recommended:

- Add a separate function to decrease or increase the number of trusted tokens.

2. The onlyPayloadSize modifier does not prevent a Short Address Attack

Description:

To prevent the possibility of a Short Address Attack, the developer added a special check for the length of the transmitted data – the onlyPayloadSize modifier.

Risk:

This check is not sufficient and may compromise the logic of the contract in the future.

Vulnerable resource:

- StandardToken.sol and BasicToken.sol

Technical details:

The onlyPayloadSize modifier (lines 20-23 in the BasicToken.sol file) is not a strong enough test and may compromise the contract's logic in the future (for example, when forking Metropolis).

Recommended:

- Since there is no clear solution to this issue at the moment, we recommend removing the onlyPayloadSize modifier. In order to prevent theft of tokens, users should be instructed to only use trusted exchanges and wallets.

3. Redundant code

Description:

The smart contract contains code that does not affect the logic, but consumes gas.

Risk:

Unnecessary complication of the contract and excessive gas consumption.

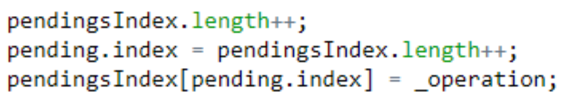
Vulnerable resource:

- Shareable.sol

Technical details and recommendations:

- Line 197. The "<=" sign could be replaced with "==", since the number of owners cannot be negative.
- Line 198. Calling the clearOwnersDone function does not change the state of the contract.
- Lines 241-243. The following code adds the next operation to the array of operations, and also stores the operation by its index. However, the code is not transparent and uses only odd elements of the array. Our recommendation is to consider reducing the number of lines and using functions that "speak for themselves".

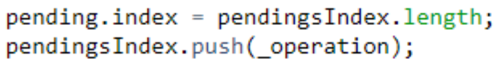
Figure 3.2.3 - 1. Current version of transaction processing



```
pendingsIndex.length++;
pending.index = pendingsIndex.length++;
pendingsIndex[pending.index] = _operation;
```

- Line 219. It is redundant to use the onlyOwner modifier for the functions for receiving smart contract data, because any member of the Ethereum network can bypass the modifier and get the data stored in the contract.

Figure 3.2.3 - 1. Recommended version of transaction processing



```
pending.index = pendingsIndex.length;
pendingsIndex.push(_operation);
```

4. The fallback function isn't used

Description:

If a participant invokes a smart contract and doesn't specify the function, or uses an invalid signature, control will be passed to the fallback function.

Risk:

The smart contract may have an unexpected behavior for the user that could result in participants losing ether.

Vulnerable resource:

- OTNToken.sol

Technical details:

The OTNToken contract and its descendants do not have a fallback function.

Recommended:

- Since interaction with the contract must follow established rules, our recommendation is to declare a fallback function that generates an exception.

Appendix 1. Analysis of the security level. Reference information

Analysis of the security level

The security analysis evaluates the severity and exploitability of the vulnerabilities identified during the audit. Exploitability is determined by how easy it is to access and take advantage of the vulnerability.

Severity of threats

The severity of a vulnerability is determined from the possible consequences of exploiting this vulnerability in terms of threats to the confidentiality, integrity, and availability of information that is processed on a vulnerable resource. Table A–1 describes the severity levels.

Table A–1. Severity levels

Value	Breach of confidentiality	Violation of integrity	Loss of availability
None	Does not occur	Does not occur	Does not occur
Low	An attacker gains access to non-critical information as a result of privilege escalation	Compromised integrity of non-critical information with normal user rights	Short-term denial-of-service failure in a critical application
Medium	Breach of confidentiality for critical information with normal user rights	Compromised integrity of critical information with normal user rights	Denial of service in a critical application or short-term denial of service
High	Breach of confidentiality for critical information with admin rights	Compromised integrity of critical information with admin rights	Denial of service

Ease of exploitation

The ease of exploitation is determined by the hardware and software, professional skills, time, and computing resources needed for a potential attacker to exploit the vulnerability (Table A–2).

Table A-2. Ease of exploitation

Value	Description
Low	Exploitation of the vulnerability requires the development of new software, analysis of the attacked system's configuration, identification and testing of possible approaches to exploiting the vulnerability, and the availability of processing power or time. The attacker must have considerable professional skills and knowledge in specific areas.
Medium	Exploitation of the vulnerability requires special programs or hardware, analysis of the attacked system's configuration, and the availability of processing power or time. The attacker only needs to have a basic level of professional skills and knowledge to carry out an attack.
High	Exploitation of the vulnerability does not require special hardware or software, significant investments of time or processing power, or detailed knowledge of the attacked system's configuration. The attacker does not need specific professional skills or knowledge to carry out an attack.

Accessibility of the vulnerability

The "accessibility" of a vulnerability reflects which types of users have access to a vulnerable resource (Table A-3).

Table A-3. Accessibility

Value	Description
Low	Superusers
Medium	Registered users
High	All users

Exploitability

The exploitability of the vulnerability is based on the ease of exploitation and the accessibility of the vulnerability as shown in Table A-4.

Table A-4. Exploitability

Exploitability		Ease of exploitation		
		Low	Medium	High
Availability	Low	Low	Low	Medium
	Medium	Low	Medium	High
	High	Medium	High	High

Vulnerability risk

The vulnerability risk (for one of the threats) is based on the severity (for one of the threats) and the exploitability, as shown in Table A-5.

Table A-5. Risk level

Vulnerability risk		Exploitability		
		Low	Medium	High
Severity	Low	Low	Low	Medium
	Medium	Low	Medium	High
	High	Medium	High	High

Appendix 2. Analysis of the security level. Reference information

Table B-1 shows the system's vulnerabilities and weaknesses that were discovered during the evaluation.

Table B-1. Vulnerabilities and weaknesses discovered in the system

Vulnerabilities detected		
Vulnerability	Overall risk	Details in section
Function is susceptible to Short Address Attack	Medium	3.1.1.
Desyncing object data	Medium	3.1.2.
List of weaknesses found		
Functionality inaccessible	3.2.1.	
The onlyPayloadSize modifier does not prevent a Short Address Attack	3.2.2.	
Redundant code	3.2.3.	
The fallback function isn't used	3.4.2.	