Security Audit Report

XET Asset Deployment

Delivered: July 12th, 2021

Prepared for Xfinite by

runtime verification

Table of Contents

Summary

Xfinite engaged Runtime Verification Inc to conduct a security audit of the XET token and its deployment script. The objective was to check the consistency of the specification of the XET token against its description in the XET white paper, and review the token's deployment script written in Python.

The audit was conducted by Musab Alturki in two rounds over the course of two calendar weeks:

1. Round 1 (from June 1, 2021 through June 15, 2021), which was based on the XET github repository (commit ID **ea100ce7d2457229adf04be5db3ff7fcc6940fde**)
2. Round 2 (on July 12, 2021) , which was based on the XET github repository (commit ID **98bfe40e9e486b355c16a90556d489bb5d5372e5**), which included changes based on the issues highlighted in the first round.

The audit referred to the white paper provided and the script and configuration files available in the aforementioned commits above.

The audit led to identifying two potentially critical issues, in addition to four informative findings and recommendations. The two issues are: a potential privilege escalation vulnerability through revealed secrets (A01) and a potential token hijacking attack through a misconfigured asset (A02). The remaining four informative findings are about improving the XET asset specification infrastructure (B01), clarifying the XET token trust model (B02), refactoring the Python code for increased readability and maintainability (B03) and fixing small typos in the README.md documentation (B04).

All the critical issues have been addressed, and most of the informative findings and general recommendations have been incorporated as well.

Scope

The audit is limited to the following artifacts:

1. The XET white paper v2.3.1, dated October 2020, Section 7.0
2. For the first round of the audit, the contents of the XET Github repository (commit ID **ea100ce7d2457229adf04be5db3ff7fcc6940fde**), containing the following files:
    a. config.cfg
    b. main.py
    c. README.md
    d. requirements.txt
3. For the second round of the audit, the contents of the XET Github repository (commit ID **98bfe40e9e486b355c16a90556d489bb5d5372e5**), containing the following files:
    a. config.cfg
    b. main.py
    c. README.md
    d. requirements.txt

Methodology

To ensure our audit is as thorough as possible, we followed the following steps:

1. Review the specification of the XET token as specified in the configuration file and the deployment script and cross-check them against the description in the white paper.
2. Review README.md and attempt to follow the instructions to compile and execute the main script while looking out for inconsistencies or issues.
3. Review the configuration and deployment code for correctness and consistency.

We also reviewed in a second round the changes that were made based on the issues raised in the first round of the audit to ensure that no problems/issues were introduced by these changes.

Disclaimer

This report does not constitute legal or investment advice. The preparers of this report present it as an informational exercise documenting the due diligence involved in the secure development of the target code only, and make no material claims or guarantees concerning the code’s operation post-deployment. The preparers of this report assume no liability for any and all potential consequences of the deployment or use of this code.

Blockchain and smart contract technologies are still a nascent software arena, and their deployment and public offering carries substantial risk. This report makes no claims that its analysis is fully comprehensive, and recommends always seeking multiple opinions and audits.

This report is also not comprehensive in scope, excluding other components critical to the correct operation of this system.

The possibility of human error in the manual review process is very real, and we recommend seeking multiple independent opinions on any claims which impact a large quantity of funds.

Findings

Findings presented in this section are issues that can cause the system to fail, malfunction, and/or be exploited, and should be properly addressed.

A01: Potential Privilege Escalation Vulnerability

The credentials of the Algorand account that is responsible for creating and managing the asset, and the algod security token were both hardcoded in plain text in the configuration file “config.cfg”. Hardcoding the account’s private key and the algod security token in the configuration file is not at all advised in a production environment. Anyone with the private key can recover the account and act on its behalf, including destroying the asset and creating new assets, among other actions. Furthermore, the algod security token authenticates the user to the server/node providing access to the network.

Making the repository public or migrating it as is to a publicly accessible repository puts the XET asset and the account creating it (along with its access to the network) at a high risk of being compromised.

Scenario

1. The attacker observes the account credentials and the algod security token.
2. The attacker uses this information to submit a transaction signed by the account to change the asset’s manager, effectively taking control of the asset entirely.
3. The attacker may even choose to submit a transaction signed by the account to rekey the account itself, taking full control of the account.

Recommendation

Remove all secrets from configuration files and everywhere in the repository. More specifically, separate out secrets, namely the owner account private key and the algod security token, from public configuration information. Only the public configuration file containing the asset parameters and account’s public key can be included as part of the repository. Required private data can be supplied by users of the repository. To facilitate this process, the repository may maintain _sample_ configuration file for private data, and include documentation explaining how this sample configuration file should be populated with user-supplied secrets.

Status

The client acknowledged that inclusion of these secrets was temporary. The repository has now been updated so that all secrets were removed. Secrets are now read from externally set environment variables.

A02: Possibility to Hijack the XET Token

Definitions of the “UnitName” and “AssetName” parameters of the XET token were incorrectly given so that the “UnitName” has the full name of the asset, which is Mzaalo, while the “AssetName” has the ticker symbol of the asset, which is XET.

As per the Algorand Developer Documentation:

- The “UnitName” is the name of a unit of the asset, i.e. the asset’s symbol, e.g. USD. In this case, it’s XET.
- The “AssetName” is the user-friendly name given to the asset, e.g. US Dollar. In this case it’s Mzaalo.

This misconfiguration can create confusion when the asset is used (e.g. when looking up the asset by name vs. unit name). It can also enable an attacker to hijack the XET token (with a bit of marketing and social engineering), which can cause systems integrating the asset (e.g. the rewards subsystem of Xfinite) to malfunction or get compromised.

Scenario

1. The attacker creates a new fake asset using the proper values for the “UnitName” and “AssetName” parameters of the real (misconfigured) XET token, and sets himself as the manager of this fake asset.
2. The attacker takes advantage of Xfinite’s reputation and marketing efforts to lure users into buying fake assets or acquiring them for goods and services within the Xfinite ecosystem.

Recommendation

Interchange the values used for the “AssetName” and “UnitName” parameters in “config.cfg” so that:

```
assetName = Mzaalo

unitName = XET
```

Furthermore, the documentation in the README.md file needs to be updated as well.

Status

The client acknowledged the issue. The “AssetName” and “UnitName” parameter definitions have now been fixed.

Informative Findings & Recommendations

Findings presented in this section are not known to cause system failures, and do not necessarily represent any flaw in the system. However, they may indicate areas where the code deviates from best practices.

B01: Inconsistent XET Asset Specification Structure

The XET asset parameters were specified in two different places:

1. Some (required and optional) parameters are specified in the configuration file “config.cfg”
2. Some other (required and optional) parameters are specified by the deployment script, “main.py”, inside the function “Create_ASA_TXN(...)”

Not having these parameters defined in one place hinders readability and maintainability of code. Furthermore, this approach can generally cause confusion and may affect how the asset is perceived:

1. It may seem at a first glance that the asset parameters specification is missing a “defaultFrozen” attribute, which as per the developer documentation, is a required attribute upon creation of the asset.
2. It gives the impression that the asset does not specify any configurable attributes (namely, the Address, Reserve, Freeze and Clawback addresses), while in fact these are set to the creator address in the script. The implication of having these attributes set is significant for any asset as users will now need to trust the creator address with their holdings of the asset (more on this below in B02). Of course, once the asset is created, these attributes are publicly available to users and can be explored in a blockchain viewer (like algoexplorer.io). But for someone relying on config.cfg to read the asset parameters, it can create confusion, and may be suspected as an attempt to hide the fact that these attributes are actually set.

Recommendation

1. Add “defaultFrozen = False” to “config.cfg” under “[XET_params]”, and then update “main.py” to read this value from “config.cfg”.

2. Add the configurable attributes as fields in the “config.cfg” file, and then update “main.py” to read this information from “config.cfg”. The creator address is already there in the file (as the value of the field “primary_address”) and can be used to populate these asset parameters.

Status

The “DefaultFrozen” attribute is now read from the configuration file as a parameter to the main script. The configurable parameters have been changed based on the recommendation for B02 below.

B02: Unclear XET Token Authorization Model

The main script set the mutable parameters of the asset, namely the Manager, Reserve, Clawback and Freeze addresses, to be the address of a particular account that belongs to admins of Xfinite. It’s very important to have a clear understanding of why these addresses need to be set in the first place to anything other than the ZeroAddress. They provide significant authority to the addresses to which they are set (in this case the asset creator). Users therefore would need to trust the creator for their holdings of the asset, since the creator has now full control over what happens to these assets even if they are held by users. Some users may be discouraged to participate in the system if they realize they have to have this level of trust put into it. As per the developer documentation, these attributes are normally set to null, unless there are strong reasons to set them to a non-ZeroAddress value.

Recommendation

1. Have a clear understanding of the role and authority the asset creator should have over holdings of the XET token
2. If there is a need for this authority to be given the creator account:
    a. Make the reasons why these attributes are set clear in the documentation of the system (probably the white paper) and explain how this authority will and will not be used.
    b. Have the creator account be a multisig account (so transactions signed by the account would require a minimum number of signatures > 1). This can give users confidence that the central control of these assets is at least in the hands of multiple accounts (presumably belonging to different people

or roles) who are less likely to collude in case some of these accounts are compromised.

Status

The Reserve, Clawback and Freeze Addresses are now set to None. The Manager address is still set to the creator address as the system needs to authorize the creator with admin rights over the token.

B03: Python Code Refactoring Suggestions

1. A deviation from python code style conventions

The main script “main.py” largely followed Python’s naming style conventions, except for asset-creating function “Create_ASA_TXN()”, whose name contains uppercase letters.

Recommendation: To have a consistent naming style and to fully adhere to Python’s coding style conventions, the method “Create_ASA_TXN()” should be renamed so that the name is all lower-case (with words separated by underscores). For example, the name `create_asa_txn()` would be preferred.

2. Entangled Configuration Parsing logic

The main script “main.py” mixed the logic that reads input from a parser object with other parts of the logic (e.g. constructing an algod client object, creating the assets, … etc).

Recommendation: It would be preferable to isolate this logic that reads input from configuration file(s) to populate variables in the code. These variables may then be passed as parameters to functions. For example, instead of reading the Algorand server parameters inside `network_client()`, we can change the function’s signature so that these parameters are passed as actual arguments to `network_client(address, token)`, where values of `address` and `token` are the responsibility of the function that reads from configuration files. Localizing the logic dealing with file I/O improves readability and maintainability of the code.

3. **Unnecessary Type Conversion Statement**

Lines 69 and 70 read integer values from “config.cfg” as strings then converted them into the int type.

Recommendation: This indirection is unnecessary since the these values can be directly read as values of the int type using “getint” of the “parser” object, so something like:

```
total = parser.getint("XET_params","total_supply")

fractions = parser.getint("XET_params","decimals")
```

Then line 72 becomes redundant and can be removed.

Status

All these comments have been addressed as suggested.

B04: README.md Typos

Some small corrections/suggestions to improve the installation instructions in README.md:

1. A typo: “3. pip install -r requirements” should be “3. pip install -r requirements.txt” (or the file should be renamed to “requirements”)
2. It could be useful to highlight that by `python` and `pip` the instructions refer to `python3` and `pip3`, or they could just be used explicitly in the instructions.

Status

All these comments have been addressed as suggested.