Least Authority
PRIVACY MATTERS

BNS-v2

Security Audit Report

Trust Machines

Updated Final Audit Report: 30 August 2024

Table of Contents

Overview

Background

Trust Machines has requested that Least Authority perform a security audit of their BNS-v2.

Project Dates

- **July 8, 2024 - July 24, 2024:** Initial Code Review *(Completed)*
- **July 26, 2024:** Delivery of Initial Audit Report *(Completed)*
- **August 16, 2024:** Verification Review *(Completed)*
- **August 20, 2024:** Delivery of Final Audit Report *(Completed)*
- **August 30, 2024:** Delivery of Updated Final Audit Report *(Completed)*

Review Team

- Will Sklenars, Security Researcher and Engineer
- Dominic Tarr, Security Researcher and Engineer

Coverage

Target Code and Revision

For this audit, we performed research, investigation, and review of the BNS-v2 followed by issue reporting, along with mitigation and remediation instructions as outlined in this report.

The following code repository is considered in scope for the review:

- BNS-v2:
  https://github.com/Trust-Machines/BNS-V2

Specifically, we examined the Git revision for our initial review:

- `061b23a9440c61463cd063323591415db40833a4`

For the verification, we examined the Git revision:

- `67ab6fe23664c54e0647ca941116e0bc8aec5d82`

For the review, this repository was cloned for use during the audit and for reference in this report:

- BNS-v2:
  https://github.com/LeastAuthority/trust-machines-bns-v2

All file references in this document use Unix-style paths relative to the project's root directory.

In addition, any dependency and third-party code, unless specifically mentioned as in scope, were considered out of scope for this review.

Supporting Documentation

The following documentation was available to the review team:

- Stacks Documentation:
  https://docs.stacks.co/stacks-101/bitcoin-name-system

- Website:
  https://trustmachines.co

Areas of Concern

Our investigation focused on the following areas:

- Correctness of the implementation;
- Vulnerabilities within each component and whether the interaction between the components is secure;
- Whether requests are passed correctly to the network core;
- Key management, including secure private key storage and management of encryption and signing keys;
- Denial of Service (DoS) and other security exploits that would impact the intended use or disrupt the execution;
- Protection against malicious attacks and other ways to exploit;
- Inappropriate permissions and excess authority;
- Data privacy, data leaking, and information integrity; and
- Anything else as identified during the initial analysis phase.

Findings

General Comments

BNS-v2 implements a structured name system with two levels implemented in the contract (namespace and name) and a third (subdomain) implemented off chain, which is out of the scope of this audit. There are several roles for managing namespaces, such as the (optional) manager and the (mandatory) `namespace-import` role that is responsible for (optionally) setting up a new namespace with names (importing) and then launching the namespace. The `namespace-import` is also authorized to perform several manager tasks in the event that the namespace does not have a manager, such as changing the namespace price function or revoking a name.

System Design

The BNS-v2 protocol is simple and well-defined, consisting of only 1,112 lines of Clarity. The simplicity of the codebase results in a relatively minimal attack surface. We found that the Trust Machines team has implemented generally comprehensive input validation and checks; however, our team did identify one Issue relating to administrator roles (Issue A), which we recommend be resolved.

Name owners can also create subdomains under their names, which would be stored off-chain in a `zonefile`. The contract stores the hash of the `zonefile`, which can be updated by the name owner when subdomains are updated. The contract does not have visibility of subdomains, as the `zonefile` itself is not persisted to the blockchain, only the hash.

Code Quality

We performed a manual review of the repository and found the codebases to be generally organized and well-written. Certain classes of vulnerabilities that occur in Solidity projects (such as uninitialized storage pointers) are, by default, eliminated or at least largely reduced by the use of the Clarity programming language. However, we found that the function authorization does not follow a consistent pattern. Although the `manager` role is always checked against `contract-caller`, the `owner` and `namespace-import` are sometimes checked only against `tx-sender` but, at other times checked against `tx-sender` or `contract-caller` (see Issue A). During our review, we also found some inconsistencies in variable and function naming that could be improved. For example, `revoked-at` could be renamed `revoked`, as it currently represents a Boolean and not a date. However, our team noted that this naming has been updated since the delivery of our initial audit report. Additionally, `namespace-reveal` and `name-register` seem to represent similar actions for both the namespace and name, but have different naming conventions (`reveal`, `register`). While these could be standardized to use either `real`, or `register`, our team acknowledges that there could be valid reasons for this naming, such as legacy APIs depending on specific function/property names.

Tests

The repository includes sufficient test coverage.

Documentation and Code Comments

The BNS-v2 public documentation is well-written and offers a clear overview of the BNS-v2 system and its usage patterns. Additionally, we found that code commenting is very thorough. The intended functionality of each function is clearly documented in the codebase, and most lines of code have descriptive comments.

Scope

The scope of this audit included the BNS-v2 contract, but not the `subdomain/zonefile` system. `Subdomains/zonefiles` were excluded from the scope, as that part of the system is still under active development by the Trust Machines team. However, our team noted that `subdomains/zonefiles` are supported by the current version of the protocol, and recommends a follow-up audit, which focuses on this part of the system.

Dependencies

Our team did not identify any security concerns resulting from the unsafe use of dependencies, as the BNS-v2 does not depend on any outside contracts.

Specific Issues & Suggestions

We list the issues and suggestions found during the review, in the order we reported them. In most cases, remediation of an issue is preferable, but mitigation is suggested as another option for cases where a trade-off could be required.

ISSUE / SUGGESTION	STATUS
Issue A: Malicious Contract Can Authorize as tx-sender	Resolved

Issue A: Malicious Contract Can Authorize as tx-sender

Location

Examples (non-exhaustive):

`BNS-V2.clar#L293`

`BNS-V2.clar#L397`

[`BNS-V2.clar#L427`](BNS-V2.clar#L427)

Synopsis

If an authenticated role (such as a name owner) calls a third contract, that contract could call a method on the BNS-v2 contract, be authenticated as representing `tx-sender`, and transfer names or change settings.

Impact

There are many functions in the contract that contain a check for a `tx-sender`, including, for example, the owner of a name, and also the `namespace-import` role for a namespace. All manager roles are checked against `contract-caller` and are, hence, not vulnerable. Additionally, unless a user calls the malicious contract again, it will not be possible to perform any action that requires multiple blocks, such as registering a name and then transferring it. The most damaging action would be to steal (transfer) names that the `tx-sender` owns, but the easiest attack would be to steal `stx` by buying names that the attacker has listed. The attacker could also perform malicious actions using almost any method available in the contract.

Preconditions

To steal a name, the malicious contract would need to know what names a potential victim owns. This could be recorded in a data structure that the attacker updates. The malicious contract must be able to perform any action within the transaction; therefore, the information it needs would have to be preconfigured.

Feasibility

The attack is technically straightforward, but it does require the victim to interact with a malicious contract. The exploit requires a phishing attack, and an 'airdrop' is a likely bait.

Technical Details

When running a function on a Clarity contract, two variables representing the caller are set. The `tx-sender` is the account that initiates the transaction, and `contract-caller` is the entity that calls this particular contract. When an account P calls a contract C directly, then `tx-sender` will be P, and `contract-caller` will also be P. But if account P calls a contract B, which then calls C, then C will have `tx-sender` P and `contract-caller` B. When checking for the manager role, BNS-v2 consistently checks whether the `manager` is equal to the `contract-caller`. But when checking the `namespace-import` or name owner, BNS-v2 generally checks if the owner is equal to the `tx-sender` *or* the `contract-caller`. This means that if the owner calls a malicious contract, and that contract subsequently calls BNS-v2, the call will be authorized as the initiator of the call, the `tx-sender`, enabling them to perform any action that can be done by a name owner or a `namespace-import` role (the `manager` role is not vulnerable to this, as it is checked against `contract-caller` consistently).

Users can set post-conditions that abort the transaction if unexpected transfers occur (see the Mitigation section below). However, some authorized methods do not involve token transfers but may enable future transfers. For example, if the user has a post-condition preventing any of their owned tokens from changing, a malicious contract that gets authorized via a `tx-sender` check could still call `list-in-ustx` and list an owned name for a very low price. The attacker would then be able to purchase that name quickly and list it for sale again at a much higher price. The user would then be obliged to either buy their own name back or abandon it.

Even if the user mitigates this Issue by correctly using post-conditions, the attacker still has some options. For example, in an attack on an owner, the attacker could still access `list-in-ustx` and `update-zonefile-hash`. If the attack was on the `namespace-import` rule, the attacker could perform `name-revoke`, `namespace-update-price`, as well as `namespace-freeze-price`.

However, our team noted that there is only a single `namespace-import` role per namespace, so there will be fewer potential victims. Moreover, since a user who creates a namespace is more invested in the system, they are also likely to be more cautious with regards to general security practices.

Mitigation

Stacks users can set post-conditions that will abort a transaction if an unexpected transfer occurred (or if an expected one did not occur). This can be used to prevent the most damaging attacks (such as an attacker transferring a user's name/token); however, they cannot protect against methods that do not actually transfer tokens (as explained in the Technical Details section above).

Stacks users can also protect themselves by using a proxy contract that was the owner of their names, and that called BNS-v2 using the `as-contract` function. In this case, `as-contract` sets the contract address as `tx-sender` for the call. The proxy contract would therefore be hard coded to only call BNS-v2; hence, a third contract attack would be impossible.

Ideally, stacks users should never call an untrusted contract with the same account that owns their names, or should check whether any contract they wish to interact with calls other contracts. However, our team notes that it is not realistic to expect all users to devote the time necessary to understand all the nuances of a contract they wish to use. If a security issue can be remediated by a technical solution, doing so is highly recommended (see the following Remediation section).

Remediation

In order to authorize a role, we recommend that `contract-caller` be checked instead of `tx-sender`.

Status

The BNS-v2 contract no longer uses `tx-sender`.

Verification

Resolved.

About Least Authority

We believe that people have a fundamental right to privacy and that the use of secure solutions enables people to more freely use the Internet and other connected technologies. We provide security consulting services to help others make their solutions more resistant to unauthorized access to data and unintended manipulation of the system. We support teams from the design phase through the production launch and after.

The Least Authority team has skills for reviewing code in multiple Languages, such as C, C++, Python, Haskell, Rust, Node.js, Solidity, Go, JavaScript, ZoKrates, and circom, for common security vulnerabilities and specific attack vectors. The team has reviewed implementations of cryptographic protocols and distributed system architecture in cryptocurrency, blockchains, payments, smart contracts, zero-knowledge protocols, and consensus protocols. Additionally, the team can utilize various tools to scan code and networks and build custom tools as necessary.

Least Authority was formed in 2011 to create and further empower freedom-compatible technologies. We moved the company to Berlin in 2016 and continue to expand our efforts. We are an international team that believes we can have a significant impact on the world by being transparent and open about the work we do.

For more information about our security consulting, please visit https://leastauthority.com/security-consulting/.

Our Methodology

We like to work with a transparent process and make our reviews a collaborative effort. The goals of our security audits are to improve the quality of systems we review and aim for sufficient remediation to help protect users. The following is the methodology we use in our security audit process.

Manual Code Review

In manually reviewing all of the code, we look for any potential issues with code logic, error handling, protocol and header parsing, cryptographic errors, and random number generators. We also watch for areas where more defensive programming could reduce the risk of future mistakes and speed up future audits. Although our primary focus is on the in-scope code, we examine dependency code and behavior when it is relevant to a particular line of investigation.

Vulnerability Analysis

Our audit techniques include manual code analysis, user interface interaction, and whitebox penetration testing. We look at the project's website to get a high level understanding of what functionality the software under review provides. We then meet with the developers to gain an appreciation of their vision of the software. We install and use the relevant software, exploring the user interactions and roles. As we do this, we brainstorm threat models and attack surfaces. We read design documentation, review other audit results, search for similar projects, examine source code dependencies, skim open issue tickets, and generally investigate details other than the implementation. We hypothesize what vulnerabilities may be present and possibly resulting in Issue entries, then for each, we follow the following Issue Investigation and Remediation process.

Documenting Results

We follow a conservative and transparent process for analyzing potential security vulnerabilities and seeing them through successful remediation. Whenever a potential issue is discovered, we immediately create an Issue entry for it in this document, even before having verified the feasibility and impact of the issue. This process is conservative because we document our suspicions early even if they are later shown to not represent exploitable vulnerabilities. We generally follow a process of first documenting the suspicion with unresolved questions, then confirming the issue through code analysis, live experimentation, or automated tests. Code analysis is the most tentative, and we strive to provide test code, log captures, or screenshots demonstrating our confirmation. After this, we analyze the feasibility of an attack in a live system.

Suggested Solutions

We search for immediate and comprehensive mitigations that live deployments can take, and finally, we suggest the requirements for remediation engineering for future releases. The mitigation and remediation recommendations should be scrutinized by the developers and deployment engineers, and successful mitigation and remediation is an ongoing collaborative process after we deliver our Initial Audit Report, and before we perform a verification review.

Before our report, including any details about our findings and the solutions are shared, we like to work with your team to find reasonable outcomes that can be addressed as soon as possible without an overly negative impact on pre-existing plans. Although the handling of issues must be done on a case-by-case basis, we always like to agree on a timeline for a resolution that balances the impact on the users and the needs of your project team.

Resolutions & Publishing

Once the findings are comprehensively addressed, we complete a verification review to assess that the issues and suggestions are sufficiently addressed. When this analysis is completed, we update the report and provide a Final Audit Report that can be published in whole. If there are critical unaddressed issues, we suggest the report not be published and the users and other stakeholders be alerted of the impact. We encourage that all findings be dealt with and the Final Audit Report be shared publicly for the transparency of efforts and the advancement of security learnings within the industry.