CD SECURITY

AUDIT REPORT

XNS

January 2026

Introduction

A time-boxed security review of the **XNS** protocol was done by **CD Security**, with a focus on the security aspects of the application's implementation.

Disclaimer

A smart contract security review can never verify the complete absence of vulnerabilities. This is a time, resource, and expertise-bound effort where we try to find as many vulnerabilities as possible. We can not guarantee 100% security after the review or even if the review will find any problems with your smart contracts. Subsequent security reviews, bug bounty programs, and on-chain monitoring are strongly recommended.

About **XNS**

XNS is an Ethereum-native naming system that maps human-readable names to Ethereum addresses and smart contracts. Names are permanent, non-transferable, and globally unique, and both EOAs and smart contracts can be named under the same registry.

Names follow a strict lowercase format and are registered under either public or private namespaces. Public namespaces open to everyone after a 30-day exclusivity period for the creator, while private namespaces remain fully controlled by the creator.

Bare names without a namespace are premium and cost 10 ETH. Internally, they resolve through a special “x” namespace, meaning `name` and `name.x` point to the same address.

Eighty percent of all registration fees are permanently burned, contributing to Ethereum’s deflationary mechanics. The remaining 20 percent is distributed to namespace creators and the XNS contract owner, depending on namespace type, with the “eth” namespace explicitly disallowed to avoid ENS confusion.

Severity classification

Severity	Impact: High	Impact: Medium	Impact: Low
Likelihood: High	Critical	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

Impact - the technical, economic, and reputation damage of a successful attack

Likelihood - the chance that a particular vulnerability gets discovered and exploited

Severity - the overall criticality of the risk

Security Assessment Summary

***review commit hash* - e3435b39bd3e23659ab4a544a5b5df7e095eaa34**

Scope

The following smart contracts were in scope of the audit:

- `xns/contracts/src/XNS.sol`
- `xns/contracts/src/interfaces/IXNS.sol`

The following number of issues were found, categorized by their severity:

- Critical & High: 0 issues
- Medium: 1 issues
- Low: 2 issues

Findings Summary

ID	Title	Severity	Status
[M-01]	Creator address cannot be updated	Medium	Fixed
[L-01]	Owner address is immutable	Low	Fixed
[L-02]	Signature revoke not possible	Low	Acknowledged
[I-01]	Possible Gas optimisation within the `_registerNamespace`	Informational	Fixed
[I-02]	The `keccak256(bytes("eth")` can be precalculated	Informational	Fixed

Detailed Findings

[M-01] Creator address cannot be updated

Severity

Impact:

High

Likelihood:

Low

Description

The current protocol's design assumes that namespace creator cannot be updated. This can be problematic in the event of creator's address compromise, especially that accumulated fees can be hijacked by the adversary. Additionally, the protocol cannot rely on assumption that every creator will use multi-signature wallets at all.

Recommendations

It is recommended to add functionality allowing the creator updating its address. The suggested implementation should follow the design of OpenZeppelin's `Owner2Step` library.

[L-01] Owner address is immutable

Description

The current protocol's design assumes that protocol's owner is immutable and cannot be updated. This can be problematic in the event of owner's address compromise or accidental private keys disclosure, especially that accumulated fees can be hijacked by the adversary. This finding is reported despite the design choice statement within the `DEV_NOTES.md` file.

Recommendations

It is recommended to add functionality allowing the owner updating its address. The suggested implementation should follow the design of OpenZepplin's `Owner2Step` library.

[L-02] Signature revoke not possible

Description

Currently, the protocol's design assumes that signatures prepared by EOA have no deadline. Additionally, there is no possibility to revoke any signature. Thus, in the certain scenarios, single EOA might be forced to be registered to certain namespace, despite the fact the EOA may change the mind or get interested with different, sponsored namespace in the meantime.

Recommendations

It is recommended to consider adding deadline to the signature, or to implement mechanism that revokes or prevents certain signature usage.

Comment from client

" I considered the recommendation to support revocation of authorization signatures and decided to intentionally leave the design unchanged. The reasoning is as follows:

- A deadline parameter does not meaningfully improve safety. A user who is confident about receiving a specific name via the authorization flow may choose a long or far-future deadline. If they later change their mind, the deadline provides no practical way to revoke consent. As a result, adding a

deadline parameter does not materially reduce risk and adds protocol complexity without clear benefit.

- I explored adding an explicit on-chain invalidation function that would accept the RegisterNameAuth struct and signature, and invalidate the authorization if the recovered signer matches the recipient. However, this approach does not work for contract recipients (e.g. ERC20 token contracts) that rely on the EIP-1271 authorization flow (e.g., name registrations within private namespaces) but do not expose an execution mechanism to call the invalidation function. In such cases, revocation would be impossible, leading to inconsistent behavior across recipient types. I prefer to avoid solutions that work only for some recipients and require extensive documentation.
- By signing an authorization to register an XNS name, the user expresses his intent to have an XNS name. If a user later prefers a different name than the one originally authorized, they can simply register a new name themselves before the signed message is executed. As each address can own only one name, the registration via authorization would fail. As a result, revocation is not considered unnecessary. "

[I-01] Possible Gas optimisation within the `_registerNamespace`

Description

Within the `_registerNamespace` is calculated twice, while it could be done once and reused as `nsHash` variable.

```
        require(keccak256(bytes(namespace)) != keccak256(bytes("eth")),
"XNS: 'eth' namespace forbidden");

        bytes32 nsHash = keccak256(bytes(namespace));
        require(_namespaces[nsHash].creator == address(0), "XNS: namespace
already exists");
```

Recommendations

It is recommended to use `nsHash` variable twice to save some Gas.

[I-02] The `keccak256(bytes("eth")` can be precalculated

Description

Within the `_registerNamespace` the `keccak256(bytes("eth")` is recalculated each time the function is called. This value could be a precalculated hash constant instead.

```
        require(keccak256(bytes(namespace)) != keccak256(bytes("eth")),
"XNS: 'eth' namespace forbidden");

## Recommendations

It is recommended to use precalculated value to save some Gas.
```