PROTOFIRE

May 2025

Smart contract audit
Punk Domains

Content

Project description

Punk Domains allow anyone to create their own top-level domain (TLD), such as .wagmi, or a regular domain like techie.wagmi. Users can also customize their domains by adding a description, a redirect URL that works well with the Punk Domains browser extension, and a profile picture defined by the address and token ID of an NFT.

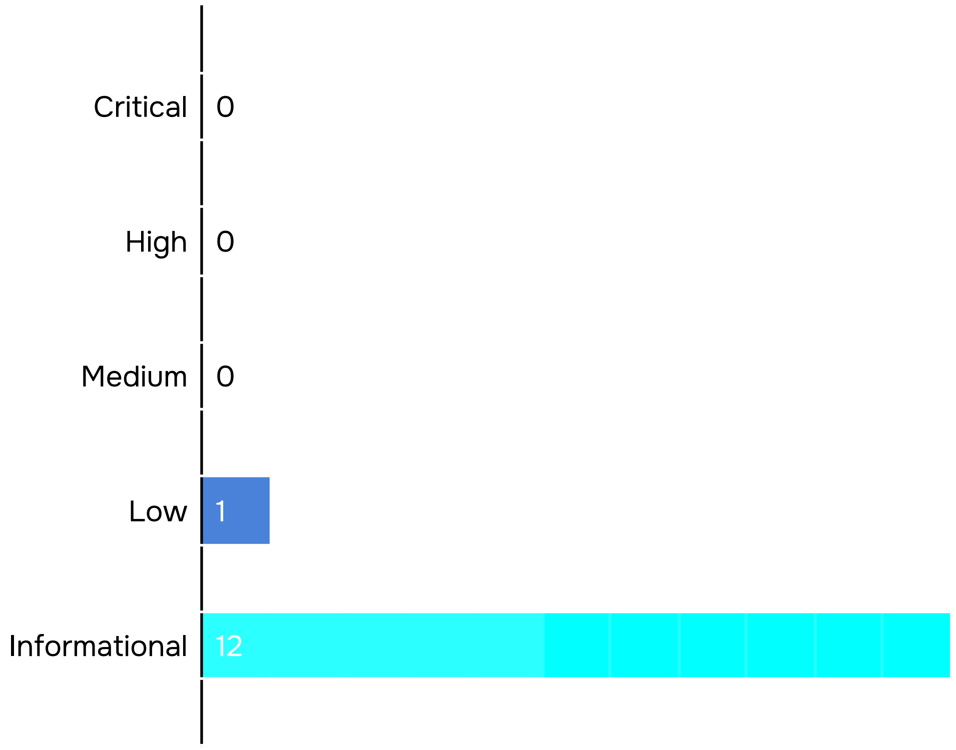
Executive summary

Type	ERC721
Languages	Solidity
Methods	Architecture Review, Manual Review, Unit Testing, Functional Testing, Automated Review
Documentation	README.md
Repository	https://github.com/punk-domains-2/punk-contracts

Reviews

Date	**Commit**
22/05/25	f26bb7ee78daff8ac8df76184dc1d0482c0a388f

Technical analysis and findings



Security findings

**** Critical

No critical severity issue was identified.

*** High

No High severity issue was identified.

** Medium

No medium severity issue was identified.

* Low

L01 - Inconsistent msg.sender usage In Metadata Retrieval

Impact	Low
Likelihood	High

The `getMetadata()` function in the `FlexiPunkMetadata` contract uses `msg.sender` directly. When this function is called by other contracts, such as `FlexiPunkTLD` through its `tokenURI()` function, `msg.sender` represents the calling contract instead of the original user. This can cause incorrect assumptions about the caller, especially if user-specific logic or permissions rely on `msg.sender`.

Recommendation: To avoid inconsistencies and potential vulnerabilities, pass the msg.sender manually to the getMetada() function call.

Path: contracts/factories/flexi/FlexiPunkTLD.sol

Found in: f26bb7e

Status: New

Informational

I01 - Events Not Indexed

The events declared in the PunkResolverNonUpgradable contract, specifically FactoryAddressAdded, DeprecatedTldAdded, DeprecatedTldRemoved, and CustomDefaultDomainSet, do not use indexed parameters. This omission makes it inefficient to filter and search for specific events in the transaction logs, as the blockchain's event filtering mechanism relies on indexed parameters to optimize queries. For example, the FactoryAddressAdded event includes the user and fAddr parameters, but neither is indexed, which prevents efficient querying for events related to a specific user or factory address. Similarly, the DeprecatedTldAdded and DeprecatedTldRemoved events lack indexed parameters for user and tAddr, and the CustomDefaultDomainSet event does not index user, dName, or dTld. This can lead to performance issues and increased complexity when analyzing logs or debugging, especially in large-scale deployments where numerous events are emitted.

Recommendation: To address this issue, add the indexed keyword to the relevant parameters in the event declarations

Path: src/contracts/resolver/non-upgradeable/PunkResolverNonUpgradable.sol

Found in: f26bb7e

Status: New

I02 - Floating Pragma

The PunkResolverNonUpgradable contract uses a floating pragma statement (pragma solidity ^0.8.4;), which allows the contract to be compiled with any version of Solidity starting from 0.8.4 up to, but not including, 0.9.0. While this provides flexibility, it introduces potential security risks. Future versions of Solidity may introduce changes or bugs that could affect the behavior of the contract. Additionally, using a floating pragma can lead to inconsistencies if different developers or environments compile the contract with different compiler versions, potentially resulting in unexpected behavior or vulnerabilities.

Recommendation: To mitigate this issue, specify a fixed Solidity version in the pragma statement. For example, instead of using ^0.8.4, explicitly set the version to 0.8.4. This ensures that the contract is always compiled with the intended compiler version, reducing the risk of unexpected behavior due to compiler changes. Fixing the pragma version also improves reproducibility and consistency across different development and deployment environments.

Path: src/contracts/*

Found in: f26bb7e

Status: New

I03 - External Functions Declared as Public

In the PunkResolverNonUpgradable contract, most of the functions, such as getDefaultDomain, getDefaultDomains, getDomainHolder, getDomainData, getDomainTokenUri, getFactoriesArray, getFirstDefaultDomain, getTldAddress, getTldFactoryAddress, and getTlds, are declared as public instead of external. Declaring these functions as public allows them to be called both internally and externally, but in this contract, they are primarily intended for external use. Using public instead of external increases gas costs because public functions copy the calldata into memory when called externally, whereas external functions can directly access calldata without copying. This inefficiency can lead to unnecessary gas consumption, especially in contracts with high usage or frequent calls to these functions.

Recommendation: To optimize gas usage and align with the intended usage of the functions, change the visibility of functions that are meant to be called externally to external.

Path: src/contracts/resolver/non-upgradeable/PunkResolverNonUpgradable.sol

Found in: f26bb7e

Status: New

I04 - Poor Design In Name Max Length Modification

The changeNameMaxLength function in the FlexiPunkTLD contract allows the owner to arbitrarily modify the nameMaxLength variable, including reducing its value. While this does not cause functional issues, it is considered poor design because it introduces inconsistencies in the system. Domain names that already exceed the new lower limit would remain valid, creating a mismatch between the maximum length for new names and the length of existing ones. Additionally, the absence of an upper bound for the nameMaxLength variable could result in impractically large values, which may not align with the intended use of the contract and could lead to inefficiencies.

Recommendation: To improve the design, restrict the changeNameMaxLength function to only allow increasing the nameMaxLength value. This ensures that the maximum length can only be adjusted upwards, maintaining consistency with existing domain names. Furthermore, implement an upper bound limit for the nameMaxLength variable to prevent excessively large values that could lead to inefficiencies or impractical use cases. These changes will enhance the overall design and usability of the contract while maintaining alignment with its intended functionality.

Path: src/contracts/factories/flexi/FlexiPunkTLD.sol

Found in: f26bb7e

Status: New

I05 - Lack Of Functionality To Set Default Name After Deletion

In the FlexiPunkTLD contract, the burn function and the _beforeTokenTransfer hook delete the defaultNames mapping entry for an address when a domain is burned or transferred. This leaves the address without a default name, and there is no function available for the user to manually set a new default name unless they purchase or acquire another TLD. This limitation is a design flaw, as it restricts the user's ability to manage their default domain name in scenarios where their previous default name is no longer valid. It also reduces the flexibility and usability of the contract for users who may want to update their default name without minting or buying a new domain.

Recommendation: Introduce a dedicated function that allows users to set a new default name for their address. This function should validate that the user owns the domain they wish to set as their default name and update the defaultNames mapping accordingly. By providing this functionality, the contract will offer a more user-friendly experience and ensure that users can manage their default names effectively, even after burning or transferring their previous default domain.

Path: src/contracts/factories/flexi/FlexiPunkTLD.sol

Found in: f26bb7e

Status: New

I06 - Missing Upper Bound Check For Royalty

The changeRoyalty function in the FlexiPunkTLDFactory contract allows the owner to modify the royalty variable without enforcing an upper bound. This is problematic because the FlexiPunkTLD contract assumes that the royalty value will not exceed

5000 basis points (50%) and includes checks to enforce this limit. However, if the royalty value in the factory contract exceeds 5000, it could lead to inconsistencies or failures when the value is used in the FlexiPunkTLD contract. This lack of validation in the factory contract creates a design flaw, as it relies on downstream contracts to enforce constraints that should be handled at the source.

Recommendation: Add an upper bound check in the changeRoyalty function to ensure that the royalty value cannot exceed 5000 basis points. This will align the factory contract's behavior with the expectations of the FlexiPunkTLD contract and prevent potential inconsistencies. By enforcing this limit at the factory level, the system will maintain consistency and reduce the risk of misconfiguration.

Path: src/contracts/factories/flexi/FlexiPunkTLDFactory.sol

Found in: f26bb7e

Status: New

I07 - Lack Of Input Validation Allows Storage Bloat And Unicode Abuse

Description: Contracts allow users to store arbitrary strings without limits on the number of entries or constraints on the content. The contract accepts any Unicode characters, including emojis and lookalike (homoglyph) letters. This makes the system vulnerable to phishing and spoofing attacks, for example by using visually similar characters from different alphabets (such as Cyrillic "a" instead of Latin "a"). Such issues can also disrupt off-chain integrations that do not properly handle or distinguish these characters.

Recommendation: It is recommended to enforce strict limits on the number and length of strings that users can register, and to restrict the character set to prevent the use of Unicode homoglyphs and emojis. Implementing input validation will help protect against storage bloat, phishing, and compatibility issues with external systems.

Path: contracts/*

Found in: f26bb7e

Status: New

I08 - No Event for Royalty/Referral Changes

Description: There are no events for:
- changeRoyaltyFeeReceiver()

- changeRoyaltyFeeUpdater()
- changeForbiddenTldsAddress()
- changeMetadataAddress()
- changeNameMaxLength()
- changeRoyalty()
- toggleBuyingTlds()
- removeFactoryAddress()

All those operations are changing the storage and it is a good practice to emit events. Becomes handy when using subgraphs.

Recommendation: Emit event on each function.

Path: contracts/factories/flexi/FlexiPunkTLD.sol; contracts/factories/flexi/FlexiPunkTLDFactory.sol; contracts/resolver/non-upgradable/PunkResolverNonUpgradable.sol

Found in: f26bb7e

Status: New

I09 - No Pausing Mechanism For Transfers Or Burn Functions

Description: The contract currently allows only the minting functionality to be paused or disabled, while transfers and burn operations remain always available. This means that if a bug or exploit affecting these actions is discovered, there is no way to temporarily freeze or limit contract activity to prevent further damage. As a result, tokens could continue to be transferred or burned even during emergencies.

Recommendation: Consider implementing an emergency pause mechanism that can temporarily disable all critical functions, including transfers and burn operations. This would allow the contract owner or authorized role to respond quickly and mitigate risks in case of unforeseen issues or exploits.

Path: contracts/factories/flexi/FlexiPunkTLD.sol

Found in: f26bb7e

Status: New

I10 - No Supply Cap

Description: There is no maximum limit (“cap”) on the number of domains that can be minted per TLD. Unless the owner manually disables minting, an unlimited number of domains can be created, which may not align with scarcity requirements or the intended tokenomics. Can cause DoS if not controlled.

Recommendation: Consider implementing an optional supply cap per TLD to enforce scarcity if that is an important business or community feature.

Path: contracts/factories/flexi/FlexiPunkTLD.sol

Found in: f26bb7e

Status: New

I11 - Gas Optimization: Unbounded Array Operations and Custom Errors

Description: Some functions perform operations that loop over the entire factories array, including nested loops over TLDs. If the arrays grow too large, these on-chain operations may run out of gas, potentially causing denial of service for users. While most interactions are off-chain, the risk still exists as the system scales. Additionally, throughout the contracts, standard require() statements are used, which can be more expensive in terms of gas compared to Custom Errors.

Recommendation: Monitor the growth of arrays and consider implementing paging or indexing solutions to maintain efficiency as the system scales. Where possible, replace require() statements with Custom Errors to further reduce gas costs.

Path: contracts/*

Found in: f26bb7e

Status: New

Risk section

The project uses a floating pragma statement (pragma solidity ^0.8.4;), which allows the contract to be compiled with any version of Solidity starting from 0.8.4 up to, but not including, 0.9.0. This introduces a potential risk when using Solidity version 0.8.20 or higher, as these versions introduce the PUSH0 opcode. While the PUSH0 opcode can reduce gas costs, it is not supported by all EVM-compatible chains. Deploying the contract on a chain that does not support this opcode could result in runtime errors or the contract becoming unusable. This issue is particularly critical for contracts intended to be deployed across multiple chains, as it limits compatibility and could lead to unexpected behavior if the contract is compiled with a newer Solidity version.

Throughout the codebase, there is a lack of consistent zero address (0x0000000000000000000000000000000000000000) validation in functions that accept addresses as input. This omission can lead to significant security and functional risks. Allowing the zero address to be used in critical operations, such as adding factory addresses, setting ownership, or interacting with mappings, can result in unintended behavior. For example, if the zero address is mistakenly added to a list of authorized addresses or used as a key in mappings, it could create vulnerabilities, such as unauthorized access or the inability to properly manage or revoke permissions. Additionally, the zero address is often used as a placeholder or default value, and failing to validate it can lead to logic errors or unexpected contract behavior.

Throughout the codebase, functions that require specific permissions, such as changeRoyalty, changeRoyaltyFeeReceiver, changeRoyaltyFeeUpdater, and others, rely on custom logic to enforce access control. For example, the changeRoyalty function checks if the caller is the royaltyFeeUpdater, and similar checks are used for other functions. While this approach provides basic access control, it lacks the flexibility and robustness of a granular role-based access control system. Without a standardized and modular access control mechanism, there is an increased risk of errors, such as misconfigured permissions or unauthorized access. Additionally, if the owner account or other privileged roles are compromised, the lack of granular roles could lead to significant damage, as the owner has unrestricted access to all sensitive functions.

The last payment in `_sendPayment` in FlexiPunkTLD is implemented as `payable(owner()).call{value: address(this).balance}("");`.
If the contract receives any extra ETH, whether accidentally or via forced refunds, the entire contract balance is sent to the owner on each purchase. This could result in unintended funds being transferred to the owner and potentially disrupt expected payment logic. We would recommend to calculate the specific payment amount intended for the owner and transfer only that amount, instead of sending the entire

contract balance. This approach prevents accidental loss or sweeping of unrelated funds.

When a factory is removed from the factories array, any TLDs it previously created remain active and functional unless they are deprecated through a separate process. This can cause confusion, especially if factories are frequently added or removed, as there is no automatic link between factory removal and the status of the TLDs it deployed. Consider implementing a system to track which TLDs were created by each factory and either update their status or clean up associated data when a factory is removed. This would help maintain clarity and prevent potential operational issues.

Approach and methodology

To establish a uniform evaluation, we define the following terminology in accordance with the OWASP Risk Rating
Methodology:

	Likelihood Indicates the probability of a specific vulnerability being discovered and exploited in real-world scenarios
	Impact Measures the technical loss and business repercussions resulting from a successful attack
	Severity Reflects the comprehensive magnitude of the risk, combining both the probability of occurrence (likelihood) and the extent of potential consequences (impact)

Likelihood and impact are divided into three levels: High H, Medium M, and Low L. The severity of a risk is a blend of these two factors, leading to its classification into one of four tiers: Critical, High, Medium, or Low.

When we identify an issue, our approach may include deploying contracts on our private testnet for validation through testing. Where necessary, we might also create a Proof of Concept PoC to demonstrate potential exploitability. In particular, we perform the audit according to the following procedure:

	Advanced DeFi Scrutiny We further review business logics, examine system operations, and place DeFi-related aspects under scrutiny to uncover possible pitfalls and/or bugs
	Semantic Consistency Checks We then manually check the logic of implemented smart contracts and compare with the description in the white paper.
	Security Analysis The process begins with a comprehensive examination of the system to gain a deep understanding of its internal mechanisms, identifying any irregularities and potential weak spots.