Crypton (&Studio)

PV01 – Smart Contract Audit

Date of Engagement: 09.09.2024

Contents

Version history

Version	Name	Date
0.1	First draft	11.09.2024
0.2	Internal review	12.09.2024
1.0	Recommendation plan	13.09.2024
1.1	Recommendation execution review	16.09.2024

Executive overview

The security assessment was scoped for the smart contracts of **PV01.** At the time of the audit, all source files were located at the link.

The team at CryptonStudio was provided 3 days for the engagement and assigned one full time security engineer to audit the security of the smart contracts. The security engineers are blockchain and smart contract security experts, with experience in advanced penetration testing, smart contract hacking, and have a deep knowledge in multiple blockchain protocols.

The purpose of this audit to achieve the following:

- Ensure the smart contracts' functions are intended.

- Identify potential security issues with the smart contracts.

In summary, CryptonStudio identified few security risks, and recommends performing further testing to validate extended safety and correctness in context to the whole set of contracts.

Vulnerabilities or issues observed by CryptonStudio 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. For every vulnerability, a risk level will be calculated on a scale of 1 to 5 with 5 being the highest likelihood or impact.

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL

Scope

CONTRACTS

- VF - PV01VaultFactory.sol
- BPV - PV01BondPerpetualVault.sol
- SPB - PV01SinglePaymentBondV2.sol

Codebase

https://github.com/pv01-org/blockchain

Commit

B5686730B87025FD09865707FE428EABA0ECCB48

Description	Solidity Source	Address Deployed Ethereum Mainnet
Bond Implementation V2	PV01SinglePaymentBondV2.sol	0x72FcAF8A019563ce92652D817e10eEd9Da419c05
Vault Factory	PV01VaultFactory.sol	0x7eB37F9326E2474D5178Fd5224bc35E30A5398B5
Vault Implementation	PV01BondPerpetualVault.sol	0xD418EE080ceaC1cef0dD597423FD950dB5207f78
Vault Deployed	As above	0x526Be1c610616be0e8e69893fC6766FddfBaDA61

SYSTEM OVERVIEW

Privileged Functions

In the contracts:

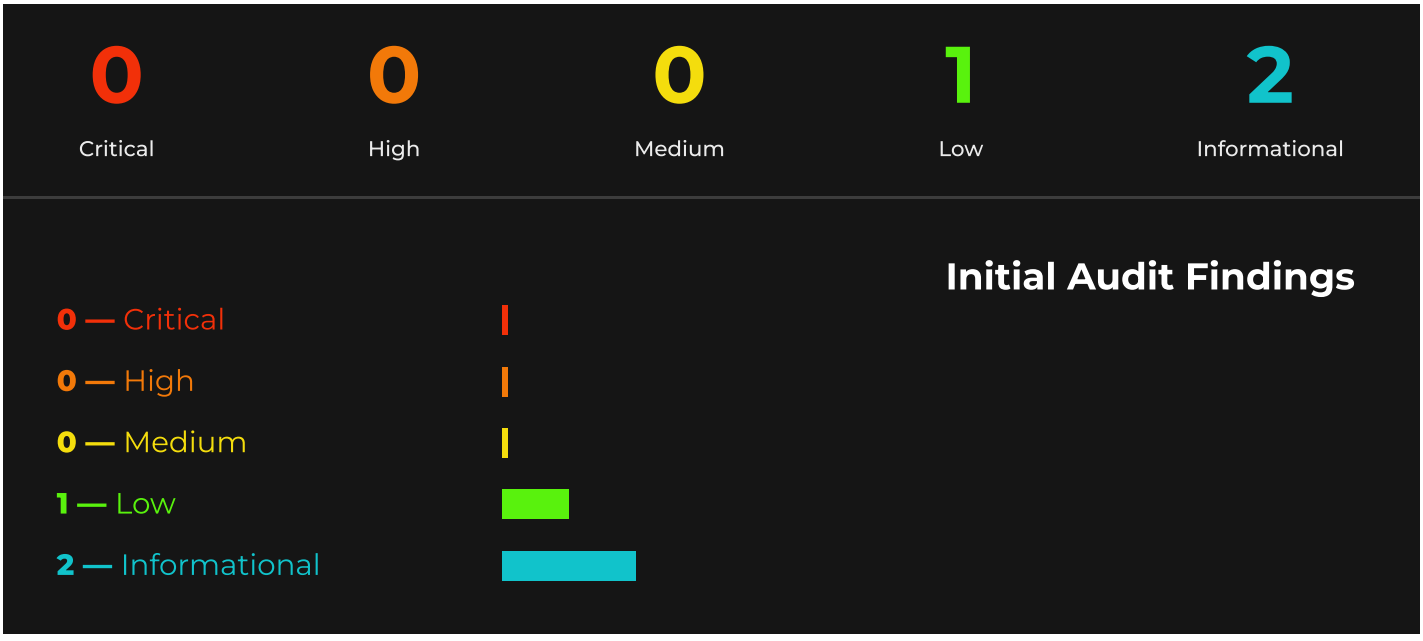
- PV01VaultFactory.sol
- PV01BondPerpetualVault.sol
- PV01SinglePaymentBondV2.sol

The role "owner" has access to privileged functions. According to the documentation, multisignature smart contracts are used to mitigate risks.

In **PV01VaultFactory,** the owner is able to create Vaults and control their versions using createVault(), changeVaultImpl() and setImplAddress() functions

In **PV01BondPerpetualVault,** the owner is able to rollover the vault to a new asset, which can put user assets at risk if applied incorrectly. The owner can also pause and unpause the vault.

Assessment summary & findings overview



All issues were addressed either by being fixed or because they were by design. **There are no open issues.**

Issue	Severity	Status	Comment from developer
Global-01	Informational	Acknowledged	Accepted. This is by design, we accept that long strings cost slightly more gas for the benefit of clear messaging. For contracts where space is at a premium (bond implementation) we have moved to custom errors.
VF-01	Informational	Acknowledged	Accept the gas-saving suggested improvement, which will be applied in a future release.
VF-02	Low	Acknowledged	Accepted. This is by design, the vault contract is long-lived and upgradeability is desirable. The upgrade process is already tested and will be thoroughly trialled before it is performed in production.

FINDINGS & TECH DETAILS

Global-01

Severity — Informational

Status — Acknowledged

Description

Most of require statements contain revert strings longer than 32 bytes. It leads to extra gas usage. If message data can fit into 32 bytes, it is always a better idea to use BYTES32 datatype rather than bytes or strings as using BYTES32 datatype is much cheaper in Solidity. Thats a good practice to use custom errors and if->else statements since they always fit into 32 bytes.

Recommendation

Check all revert strings and consider using 32 bytes at maximum for revert messages or custom errors.

Alleviation

Accepted. This is by design, we accept that long strings cost slightly more gas for the benefit of clear messaging. For contracts where space is at a premium (bond implementation) we have moved to custom errors.

VF-01

Severity — Informational

Status — Acknowledged

Description

No reason to allocate data_ to memory in function createVault, in context of given contracts.

When a function parameter is a string that will not be changed, it should ideally be allocated in calldata rather than memory.

Calldata is a non-modifiable, non-persistent area where function arguments are stored, and it costs less gas to access. Parameters stored in calldata do not need to be copied to memory, thus saving gas when the function is called externally.

Recommendation

Replace memory to calldata for data_ variable in createVault() function.

Alleviation

Accepted. Accept the gas-saving suggested improvement, which will be applied in a future release.

VF-02

Severity — Low

Status — Acknowledged

Description

The changeVaultImpl() function allows the owner to change the logic of the smart contract at any time. Since the Vault smart contract holds users' assets, the update process must be clear and safe.

Recommendation

While multisignature smart contracts are already in use, it is recommended to add a delay for this action so that users can familiarize themselves with the update.

Alleviation

Accepted. This is by design, the vault contract is long-lived and upgradeability is desirable. The upgrade process is already tested and will be thoroughly trialled before it is performed in production.

Contacts

crypton.studio

info@crypton.studio

+371 261 19 169

OUR MEDIA

Scan the QR code to open our sources