SOLIDIFIED

Audit Report for Violet Protocol - July 05, 2022

Summary

Audit Report prepared by Solidified covering the Violet Protocol smart contracts.

Process and Delivery

Independent Solidified experts performed an unbiased and isolated audit of the code. The debrief was on 2 June 2022.

Audited Files

The source code has been supplied in the form of a source code repositories:

https://github.com/violetprotocol/core-contracts

Final Commit hash: `6ae95c018519c6c016b6fab556fd7cd5f84ba2b9c`

Files audited:

```
contracts
├── factory
│   ├── VioletFactory.sol
│   ├── extensions
│   │   ├── deploy
│   │   │   ├── DeployLogic.sol
│   │   │   └── IDeployLogic.sol
│   │   ├── getter
│   │   │   ├── IVioletFactoryGetter.sol
│   │   │   └── VioletFactoryGetterLogic.sol
│   │   └── setter
│   │       ├── IVioletFactorySetter.sol
│   │       └── VioletFactorySetterLogic.sol
│   └── storage
│       └── VioletFactoryStorage.sol
├── library
│   ├── ExtensionLibrary.sol
│   ├── extensions
│   │   ├── delist
│   │   │   ├── DelistLogic.sol
│   │   │   └── IDelistLogic.sol
│   │   ├── enlist
│   │   │   ├── EnlistLogic.sol
```

```
│   │   │   └── IEnlistLogic.sol
│   │   └── getter
│   │       ├── ExtensionLibraryGetterLogic.sol
│   │       └── IExtensionLibraryGetterLogic.sol
│   └── storage
│       └── ExtensionLibraryStorage.sol
├── registry
│   ├── CredentialRegistry.sol
│   ├── extensions
│   │   ├── deregister
│   │   │   ├── DeregisterLogic.sol
│   │   │   └── IDeregisterLogic.sol
│   │   ├── getter
│   │   │   ├── CredentialRegistryGetterLogic.sol
│   │   │   └── ICredentialRegistryGetter.sol
│   │   ├── register
│   │   │   ├── IRegisterLogic.sol
│   │   │   └── RegisterLogic.sol
│   │   └── setFactory
│   │       ├── ISetFactory.sol
│   │       └── SetFactoryLogic.sol
│   └── storage
│       └── CredentialRegistryStorage.sol
└── shared
    ├── extensions
    │   ├── hooks
    │   │   ├── BeforeSensitiveOperationLogic.sol
    │   │   ├── IBeforeSensitiveOperationLogic.sol
    │   │   └── PausableBeforeSensitiveOperationLogic.sol
    │   ├── pausable
    │   │   ├── IPausableLogic.sol
    │   │   └── PausableLogic.sol
    │   └── restrictedExtend
    │       └── RestrictedExtendLogic.sol
    └── storage
        └── PausableStorage.sol
```

https://github.com/violetprotocol/extendable

Final Commit hash: `c2c36f76307515df7d5fbca7244af7546a1a4ae6`

Files audited:

SOLIDIFIED

Audit Report for Violet Protocol - July 05, 2022

```
contracts
├── errors
│   └── Errors.sol
├── extendable
│   └── Extendable.sol
├── extensions
│   ├── Extension.sol
│   ├── IExtension.sol
│   ├── InternalExtension.sol
│   ├── extend
│   │   ├── ExtendLogic.sol
│   │   └── IExtendLogic.sol
│   ├── permissioning
│   │   ├── IPermissioningLogic.sol
│   │   └── PermissioningLogic.sol
│   ├── replace
│   │   ├── IReplaceLogic.sol
│   │   ├── ReplaceLogic.sol
│   │   └── StrictReplaceLogic.sol
│   └── retract
│       ├── IRetractLogic.sol
│       └── RetractLogic.sol
├── storage
│   ├── CallerContextStorage.sol
│   ├── ExtendableStorage.sol
│   ├── PermissionStorage.sol
│   ├── ReentrancyStorage.sol
│   └── StorageTemplate.sol
└── utils
    ├── CallerContext.sol
    ├── Internal.sol
    ├── README.md
    └── ReentrancyGuard.sol
```

https://github.com/violetprotocol/erc1238-extendable

Final Commit hash: `7b4b1c6e719854cc0d79737ff646536600fb0b9c`

Files audited:

```
contracts
├── examples
│   └── Badge.sol
```

SOLIDIFIED

Audit Report for Violet Protocol - July 05, 2022

```
├── extensions
│   ├── URI
│   │   ├── ITokenURIGetLogic.sol
│   │   ├── ITokenURISetLogic.sol
│   │   ├── TokenURIGetLogic.sol
│   │   └── TokenURISetLogic.sol
│   ├── baseURI
│   │   ├── BadgeBaseURILogic.sol
│   │   ├── BaseURILogic.sol
│   │   ├── IBadgeBaseURILogic.sol
│   │   └── IBaseURILogic.sol
│   ├── burn
│   │   ├── BurnBaseLogic.sol
│   │   ├── BurnLogic.sol
│   │   ├── IBurnBaseLogic.sol
│   │   └── IBurnLogic.sol
│   ├── collection
│   │   ├── CollectionLogic.sol
│   │   └── ICollectionLogic.sol
│   ├── getters
│   │   ├── BalanceGettersLogic.sol
│   │   └── IBalanceGettersLogic.sol
│   ├── hooks
│   │   ├── badge
│   │   │   ├── BadgeBeforeBurnLogic.sol
│   │   │   └── BadgeBeforeMintLogic.sol
│   │   └── generic
│   │       ├── BeforeBurnLogic.sol
│   │       ├── BeforeMintLogic.sol
│   │       ├── IBeforeBurnLogic.sol
│   │       └── IBeforeMintLogic.sol
│   ├── mint
│   │   ├── BadgeMintLogic.sol
│   │   ├── ERC1238Approval.sol
│   │   ├── IBadgeMintLogic.sol
│   │   ├── IMintBaseLogic.sol
│   │   └── MintBaseLogic.sol
│   └── permission
│       ├── IPermissionLogic.sol
│       └── PermissionLogic.sol
├── interfaces
│   ├── IERC1155MetadataURI.sol
│   ├── IERC1238.sol
│   ├── IERC1238Receiver.sol
```

```
│   └── IERC165.sol
├── storage
│   ├── ERC1238ApprovalStorage.sol
│   ├── ERC1238CollectionStorage.sol
│   ├── ERC1238Storage.sol
│   ├── ERC1238URIStorage.sol
│   └── PermissionStorage.sol
└── utils
    ├── AddressMinimal.sol
    └── ERC165.sol
```

https://github.com/violetprotocol/erc721-extendable

Final Commit hash: `6460c9a4ad266577a30184722c1261c035137ada`

Files audited:

```
contracts
├── extensions
│   ├── base
│   │   ├── ERC721.sol
│   │   ├── Events.sol
│   │   ├── approve
│   │   │   ├── ApproveLogic.sol
│   │   │   └── IApproveLogic.sol
│   │   ├── burn
│   │   │   ├── BasicBurnLogic.sol
│   │   │   ├── BurnLogic.sol
│   │   │   ├── IBasicBurnLogic.sol
│   │   │   └── PermissionedBurnLogic.sol
│   │   ├── getter
│   │   │   ├── GetterLogic.sol
│   │   │   └── IGetterLogic.sol
│   │   ├── hooks
│   │   │   ├── BeforeTransferLogic.sol
│   │   │   └── IBeforeTransferLogic.sol
│   │   ├── mint
│   │   │   ├── BasicMintLogic.sol
│   │   │   ├── IBasicMintLogic.sol
│   │   │   ├── MintLogic.sol
│   │   │   └── PermissionedMintLogic.sol
│   │   ├── receiver
│   │   │   ├── IOnReceiveLogic.sol
│   │   │   └── OnReceiveLogic.sol
```

```
│   │   └── transfer
│   │       ├── ITransferLogic.sol
│   │       └── TransferLogic.sol
│   ├── enumerable
│   │   ├── ERC721Enumerable.sol
│   │   ├── getter
│   │   │   ├── EnumerableGetterLogic.sol
│   │   │   └── IEnumerableGetterLogic.sol
│   │   └── hooks
│   │       └── EnumerableBeforeTransferLogic.sol
│   └── metadata
│       ├── ERC721Metadata.sol
│       ├── burn
│       │   ├── IMetadataBurnLogic.sol
│       │   ├── MetadataBurnLogic.sol
│       │   └── PermissionedMetadataBurnLogic.sol
│       ├── getter
│       │   ├── IMetadataGetterLogic.sol
│       │   └── MetadataGetterLogic.sol
│       └── setTokenURI
│           ├── BasicSetTokenURILogic.sol
│           ├── IBasicSetTokenURILogic.sol
│           ├── ISetTokenURILogic.sol
│           ├── PermissionedSetTokenURILogic.sol
│           └── SetTokenURILogic.sol
└── storage
    ├── ERC721EnumerableStorage.sol
    ├── ERC721Storage.sol
    └── ERC721TokenURIStorage.sol
```

https://github.com/violetprotocol/ethereum-access-token

Final Commit hash: `8fd71dbd2ae75f450e1432a45aab2fac103b0dc8`

Files audited:

```
contracts
├── AuthCompatible.sol
├── AuthVerifier.sol
├── DummyDapp.sol
├── EtherMail.sol
├── IAuthVerifier.sol
└── KeyInfrastructure.sol
```

Intended Behavior

The smart contracts implement the credentials registry contracts for the violet protocol, based on an extendable smart contract pattern that allows modularized smrt contracts to be constructed from extension plugins. The source also includes ERC-721 and ERC-1238 implementations based on the extendable framework and a non-extendable “access token” that requires each call to be authorized by an off-chain signature.

Code Complexity and Test Coverage

Smart contract audits are an important step to improve the security of smart contracts and can find many issues. However, auditing complex codebases have their limits and a remaining risk is present (see disclaimer).

Users of a smart contract system should exercise caution. In order to help with the evaluation of the remaining risk, we provide a measure of the following key indicators: **code complexity**, **code readability**, **level of documentation**, and **test coverage**.

Note that high complexity or lower test coverage does equate to a higher risk. Certain bugs are more easily detected in unit testing than a security audit and vice versa. It is, therefore, more likely that undetected issues remain if the test coverage is low or non-existent.

Criteria	Status	Comment
Code complexity	Medium-high	The nature of the extendable framework and the low-level operations required within the framework itself results in a relatively complex architecture.
Code readability and clarity	High	-
Level of Documentation	High	-
Test Coverage	High	-

SOLIDIFIED

Audit Report for Violet Protocol - July 05, 2022

Issues Found

Solidified found that the Violet Protocol contracts contain 2 critical issues, 4 major issues and 5 minor issues, 10 informational notes complete the report.

We recommend all issues are amended, while the notes are up to the team's discretion, as they refer to best practices.

Issue #	Description	Severity	Status
1	Extendable: PermissioningLogic.sol: Anyone can take full control of an extendable contract if ownership is renounced **Resolved in:** **https://github.com/violetprotocol/extendable/pull/7**	Critical	Resolved
2	core-protocol: PausableLogic.sol: Anyone can pause the contract if pauser role is renounced **Resolved in:** **https://github.com/violetprotocol/core-contracts/pull/11**	Critical	Resolved
3	erc1238: Mint approval signatures can be replayed **Resolved in** **https://github.com/violetprotocol/erc1238-extendable/pull/11**	Major	Resolved
4	TransferLogic.sol: Wrong from address in _safeTransfer() **Resolved in** **https://github.com/violetprotocol/erc721-extendable/pull/8**	Major	Resolved
5	DeregisterLogic.sol: The last credential address is always removed **Resolved in** **https://github.com/violetprotocol/core-contracts/pull/12**	Major	Resolved
6	Various usages of msg.sender instead of _lastExternalCaller()	Major	Resolved
7	Extendable: Extendable.sol: Post-fallback hook	Minor	Resolved

	by-passed in most cases **Resolved in** **https://github.com/violetprotocol/extendable/pull/9**		
8	Extendable: Extendable contracts might fail with very large numbers of extensions	Minor	Acknowledged
9	erc1238: ERC1238Approval.sol: Invalid signatures are not detected and malleable signatures are accepted for mint approvals **Resolved in** **https://github.com/violetprotocol/erc1238-extendable/pull/13** **https://github.com/violetprotocol/ethereum-access-token/pull/8**	Minor	Resolved
10	RegisterLogic.sol: The same credential address can be registered multiple times **Resolved in** **https://github.com/violetprotocol/core-contracts/pull/15**	Minor	Resolved
11	EnlistLogic.sol: Enlisting a contract with an empty name breaks different invariants **Resolved in** **https://github.com/violetprotocol/core-contracts/pull/13**	Minor	Resolved
12	Extendable: Extension storage layout needs to be maintained between versions	Note	Acknowledged
13	erc721-extendable: Inconsistent event declaration	Note	Acknowledged
14	erc721-extendable: Contract owner has full control over all assets	Note	Acknowledged
15	Some functions can be marked as a view	Note	Acknowledged
16	Some functions can be marked as a external	Note	Acknowledged
17	PermissionLogic.sol: Wrong error messages **Resolved in** **https://github.com/violetprotocol/erc1238-extendable/pull/14**	Note	Resolved

18	ERC721 implementation: No _afterTokenTransfer hook **Resolved in** **https://github.com/violetprotocol/erc721-extendable/pull/9**	Note	Resolved
19	Extendable.sol: Unused import **Resolved in** **https://github.com/violetprotocol/extendable/pull/8**	Note	Resolved
20	Cache array lengths when iterating a for loop **Resolved in** **https://github.com/violetprotocol/erc1238-extendable/pull/16** **https://github.com/violetprotocol/extendable/pull/10** **https://github.com/violetprotocol/core-contracts/pull/14**	Note	Resolved
21	BurnLogic.sol: Consider re-using BaseBurnLogic logic	Note	Acknowledged

Critical Issues

1. Extendable: PermissioningLogic.sol: Anyone can take full control of an extendable contract if ownership is renounced

Extendable contracts allow a contract owner to add logic or replace logic. The `updateOwner()` function allows ownership to be renounced in the common way by setting ownership to `address(0)`. However, this state is also used to mark an extendable contract as uninitialized, allowing anyone to initialize the contract again and claim ownership and full control over the contract.

Recommendation
Consider adding a check for `address(0)` to `updateOwner()` and implementing a separate renounceOwnership() function that uses different null address to indicate renounced ownership, for example `address(0x000000000000000000000000000000000000dEaD)`.

2. core-protocol: PausableLogic.sol: Anyone can pause the contract if pauser role is renounced

The `setPauser()` function allows the pausable role to be renounced in the common way by setting it to `address(0)`. However, this state is also used to mark the role as uninitialized, allowing anyone to call `init()` again and claim the pauser role.

Recommendation
In contrast to issue 1, in this case, allowing the role to be renounced is probably undesired behavior. We, therefore, recommend adding a check for `address(0)` to `setPauser()`.

Major Issues

3. erc1238: Mint approval signatures can be replayed

Minting a non-transferable token requires the receiver's approval in the form of a signed message. However, the signed message does not contain a unique identifier that prevents reuse of such a signature, such as a nonce per signer. This means that the minter can use an approval signature many times until expiry time. Since the mint process involves an `amount` parameter, this is likely to be undesired behavior.

Recommendation
Consider adding a nonce per receiver address to the signed approval message and rejecting already used signatures.

4. TransferLogic.sol: Wrong from address in _safeTransfer()

The from address in the _checkOnERC721Received call is set to 0, which means that the onERC721Received call is done with the zero address as the from address. The implementation therefore does not conform to the ERC721 standard, which can lead to transfers that should not be executed and vice versa (a receiver might want to only accept minted tokens or tokens from a certain address, which would not work with this implementation).

Recommendation
Call _checkOnERC721Received with from set to the from address of the _safeTransfer call.

5. DeregisterLogic.sol: The last credential address is always removed

In deregister, a comparison is made instead of an assignment:

`state.credentialAddresses[i] == state.credentialAddresses[state.credentialAddresses.length -1];` Because of this, the last credential address is always removed instead of the one that is passed to the function

Recommendation

Change the comparison to an assignment.

6. Various usages of `msg.sender` instead of `_lastExternalCaller()`

Various extensions use msg.sender to determine the caller:

- `DeployLogic.sol`: `deploy`
- `BurnBaseLogic.sol`: `_burn` and `_burnBatch`
- `BurnLogic.sol`: `_burnBatchAndDeleteURIs`
- `MintBaseLogic.sol`: `_mintToContract`, `_mintBatchToContract`, `_mint`, and `_mintBatch`
- `PermissionLogic.sol`: `setRootController`, `setIntermediateController`, and `setController`
- `ApproveLogic.sol`: `approve`, `setApprovalForAll`
- `OnReceiveLogic.sol`: `_checkOnERC721Received`
- `TransferLogic.sol`: `transferFrom`, `safeTransferFrom`

As noted in CallerContext.sol, this can lead to wrong behavior for extensions and `_lastExternalCaller()` should be used instead.

Recommendation

Consider replacing `msg.sender` calls with calls to `_lastExternalCaller()`.

Minor Issues

7. Extendable: Extendable.sol: Post-fallback hook by-passed in most cases

---

In function `_fallback()` the existence of a fallback extension is checked. However, in case an extension is found in the default case the delegate call performs a low-level assembly return,

meaning that the code block in line 138 and 139, including the call to `_afterFallback()`, is never reached.

Recommendation
Consider refactoring the delegate call logic so that all hooks are executed.

8. Extendable: Extendable contracts might fail with very large numbers of extensions

---

The data-structures used to keep track of extensions grow dynamically with each extension added. In some cases, iterations over these data structures are performed. Should these data-structures grow too large, these transactions might hit the block gas limit and fail. This is unlikely since it would require a large number of extensions, but is possibility in extreme cases.

Recommendation
Consider enforcing a maximum number of extensions per extendable contract.

Team Reply
"This is a valid issue but only in extreme scenarios where an Extendable is incredibly large. This will be solved in an upcoming update to the Extendable and will not be addressed now."

9. erc1238: ERC1238Approval.sol: Invalid signatures are not detected and malleable signatures are accepted for mint approvals

---

In function `_verifyMintingApproval` the return value of `ecrecover` is not checked for `address(0)`, which indicates an invalid signature. Whilst this is not a security issue in this particular case, it may lead to an inconsistent error message.
In addition, the implementation allows for malleable signatures and does not detect invalid `v` parameters. This is also not a security risk in this case, but goes against best practice guidelines.

Recommendation
Consider checking for invalid and malleable signatures. An examples of best practice for this can be found in the OpenZeppelin implementation:

https://github.com/OpenZeppelin/openzeppelin-contracts/blob/5e007871991e4f04e871bf5fb1200668ff16b35f/contracts/utils/cryptography/ECDSA.sol#L142.

10. RegisterLogic.sol: The same credential address can be registered multiple times

---

When register is called multiple times with the same address, state.credentialAddresses will contain this address multiple times. This also leads to problems for the deregister logic (as it will depend on the order of inserts if all addresses will be removed).

Recommendation

Consider adding a check if the address already exists.

11. EnlistLogic.sol: Enlisting a contract with an empty name breaks different invariants

---

enlist can be called with name set to the empty string and the contract is successfully enlisted. However, this will break different invariants. It will never be possible to delist this contract, RestrictedExtendLogic will not work, and it will be possible to enlist the contract a second time under a different name (which will break different invariants in DelistLogic).

Recommendation

Consider to require that the name is non-empty in enlist or change the logic for checking the existence (boolean flag) if empty names should be supported.

Notes

12. Extendable: Extension storage layout needs to be maintained between versions

Like in other external storage upgradability patterns, the correctness of the framework's operation relies on the storage layout to remain unchanged. However, it should be safe to add new variables at the end of the storage, meaning any new variables must be appended at the end of the contract. Whilst this behavior is to be expected, it should be well-documented.

Recommendation
Consider adding some notes on storage safety to the documentation.

Team Reply
"Upgrades to storage modules can only be done in certain methods:

- *For new variables, it must be appended to the existing struct and the structure of it cannot be modified other than by appending.*
- *For reconstruction, you must completely migrate existing storage data which can be very messy and involved, carries high risk of state corruption. It is better to avoid and to simply point to a new slot and copy the relevant data.*

Actionable: This will be added in documentation.."

13. erc721-extendable: Inconsistent event declaration

All events of this ERC-721 implementation are declared in the respective interface files of their extension. However, the `Transfer` event is defined in the global `Events.sol` file. This inconsistent behavior might confuse developers building on this implementation.

Recommendation
Consider unifying event declaration practice.

Team Reply
"As you noticed, some events are defined in the interfaces whilst in the Transfer event case, it is being defined in a separate interface that is then being imported. This is because of the shared usage of the Transfer event across the Mint, Burn and Transfer extensions. Instead of redefining the same event (which could lead to inconsistencies if one is updated but not others), we have

them use the same event. In some cases this is the only way to cleanly define it e.g. an extension that inherits both Mint and Burn base logic, solidity forces us to redefine/override the event because there's a conflict (both contracts have the same event).”

14. erc721-extendable: Contract owner has full control over all assets

In extendable contracts the owner can replace any extension and has full control over the contract. This means that if a token is entirely implemented from extensions, all assets are essentially under the control of the contract owner. This may undesired behavior for the key functionalities of basic token contract, such as token transfers.

Recommendation
Consider implementing certain key functionality in a non-extendable way.

Team Reply
“We expect that contracts will manage ownership on their own terms. In the wild we expect implementors to opt for a more decentralised ownership approach (where necessary and subject to their product needs/concerns). We do not prescribe a single address ownership model for all use cases.”

15. Some functions can be marked as a view

The following functions do not modify the state and can the view modifier can be added to them:

- PausableLogic.sol: isNotPaused, isPaused, getPauser
- CollectionLogic.sol: balanceFromBaseId
- PermissionLogic.sol: getRootController, getIntermediateController, getController
- TokenURIGetLogic.sol: tokenURI
- GetterLogic.sol: _exists

Recommendation
Consider adding the view modifier to the functions.

Team Reply

“Smart contracts that use the Extendable framework should not use `view` *modifiers because it is then compiled to a* `STATICCALL` *which reverts when hitting the* `fallback()` *function, used for calling any function in an extension.*

Fallback functions are always payable which marks it as state-mutating, causing a conflict between the compiled `STATICCALL` *opcode. Since all function calls are routed through the* `__fallback` *in* `Extendable`*, regardless of if a function is marked with view or not, the entrypoint always expects the function to be state-mutating and reverts.”*

16. Some functions can be marked as a `external`

Public functions found throughout the repositories can be marked as external for gas savings.

Recommendation

Consider adding the `external` modifier to functions that aren’t called anywhere inside the contract.

17. `PermissionLogic.sol`: Wrong error messages

The error messages in `setIntermediateController` and `setController` both refer to the `newRootController` variable, which does not exist for these functions.

Recommendation

Consider changing the error message.

18. ERC721 implementation: No `_afterTokenTransfer` hook

While the ERC721 implementation closely follows the OpenZeppelin implementation, the `_afterTokenTransfer` hook is missing, which makes porting certain tokens (e.g., ERC721 that supports voting / delegation, which is easily implemented with this hook) more involved.

Recommendation

If easy support for tokens that make use of this hook is a design goal, consider adding the hook.

19. Extendable.sol: Unused import

The PermissioningLogic.sol import is not used and can be removed.

Recommendation

Remove the unused import.

20. Cache array lengths when iterating a for loop

Minor gas savings can be had by computing and storing the length of an array once before the for loop. See this gist for more information.

Recommendation

Consider caching the array length variable outside the for loop conditional expression to save gas.

21. BurnLogic.sol: Consider re-using BaseBurnLogic logic

The logic of _burnBatchAndDeleteURIs is also present in BaseBurnLogic's _burnBatch.

Recommendation

Consider re-using the logic of BaseBurnLogic's _burnBatch() in _burnBatchAndDeleteURIs() as is done in burn() for example.

Team Reply

"_burnBatch() is not called in _burnBatchAndDeleteURIs and the code duplicated in order to "inject" a call to _deleteTokenURI() in the for loop. This isn't as elegant but a call to _burnBatch() would mean iterating over the ids a second time to delete the token URIs which incurs a significant cost.
In our tests, calling burnBatch() with deleteURI as true costs 425,721 gas if the logic from _burnBatch() is duplicated in _burnBatchAndDeleteURIs and 430,191 gas if _burnBatch() is called."

Audit Report for Violet Protocol - July 05, 2022

Disclaimer

Solidified audit is not a security warranty, investment advice, or an endorsement of Violet Protocol or its products. This audit does not provide a security or correctness guarantee of the audited smart contract. Securing smart contracts is a multistep process, therefore running a bug bounty program as a complement to this audit is strongly recommended.

The individual audit reports are anonymized and combined during a debrief process, in order to provide an unbiased delivery and protect the auditors of Solidified from legal and financial liability.

Oak Security GmbH