SALUS SECURITY

JUL 2024

CODE SECURITY ASSESSMENT

NUBIT

Overview

Project Summary

- Name: Nubit - kzg-rsmt2d
- Language: Go
- Repository:
  - https://github.com/RiemaLabs/nubit-kzg
  - https://github.com/RiemaLabs/rsmt2d
- Audit Range: See Appendix - 1

Project Dashboard

Application Summary

Name	Nubit - kzg-rsmt2d
Version	v1
Type	Go
Dates	Jul 05 2024
Logs	Jul 05 2024

Vulnerability Summary

Total High-Severity issues	1
Total Medium-Severity issues	0
Total Low-Severity issues	0
Total informational issues	5
Total	6

Contact

E-mail: support@salusec.io

Risk Level Description

High Risk	The issue puts a large number of users' sensitive information at risk, or is reasonably likely to lead to catastrophic impact for clients' reputations or serious financial implications for clients and users.
Medium Risk	The issue puts a subset of users' sensitive information at risk, would be detrimental to the client's reputation if exploited, or is reasonably likely to lead to a moderate financial impact.
Low Risk	The risk is relatively small and could not be exploited on a recurring basis, or is a risk that the client has indicated is low impact in view of the client's business circumstances.
Informational	The issue does not pose an immediate risk, but is relevant to security best practices or defense in depth.

Content

Introduction

1.1 About SALUS

At Salus Security, we are in the business of trust.

We are dedicated to tackling the toughest security challenges facing the industry today. By building foundational trust in technology and infrastructure through security, we help clients to lead their respective industries and unlock their full Web3 potential.

Our team of security experts employ industry-leading proof-of-concept (PoC) methodology for demonstrating smart contract vulnerabilities, coupled with advanced red teaming capabilities and a stereoscopic vulnerability detection service, to deliver comprehensive security assessments that allow clients to stay ahead of the curve.

In addition to smart contract audits and red teaming, our Rapid Detection Service for smart contracts aims to make security accessible to all. This high caliber, yet cost-efficient, security tool has been designed to support a wide range of business needs including investment due diligence, security and code quality assessments, and code optimisation.

We are reachable on Telegram (https://t.me/salusec), Twitter (https://twitter.com/salus_sec), or Email (support@salusec.io).

1.2 Audit Breakdown

The objective was to evaluate the repository for security-related issues, code quality, and adherence to specifications and best practices. Possible issues we looked for included (but are not limited to):

- Architectural Design
- Business Logic
- Information Leakage
- Access Control
- Data Validation
- Overflow/Underflow
- Bad Randomness
- Denial of Service
- Redundancy
- Best Practice
- Improving readability

1.3 Disclaimer

Note that this security audit is not designed to replace functional tests required before any software release and does not give any warranties on finding all possible security issues with the given smart contract(s) or blockchain software, i.e., the evaluation result does not guarantee the nonexistence of any further findings of security issues.

Findings

2.1 Summary of Findings

ID	Title	Severity	Category	Status
1	The BuildRangeProof function improperly handles corner cases	High	Business Logic	Pending
2	VerifyInclusion lacks code consistency	Informational	Improving readability	Pending
3	The branch that checks \`err\` will never be executed	Informational	Redundancy	Pending
4	Lack of boundary condition checks	Informational	Business Logic	Pending
5	Use a more complex implementation to obtain the min and max namespaces	Informational	Improving readability	Pending
6	The Equal function is missing a comparison for the \`InclusionOrAbsence\` field	Informational	Best Practice	Pending

2.2 Notable Findings

Significant flaws that impact system confidentiality, integrity, or availability are listed below.

1. The BuildRangeProof function improperly handles corner cases	
Severity: High	Category: Business Logic
Target: - nubit-kzg/nmt.go	

Description

The `BuildRangeProof` function is responsible for constructing inclusion proofs, populating various field values, and constructing the corresponding `KzgOpen`. Additionally, there are corner cases in inclusion proofs, such as when "start=0" or "end=w-1". For example, the following code segment attempts to handle the boundary condition when "end=w-1", constructing postIndex and the corresponding `KzgOpen` only if `proofEnd < w-1`.

nmt.go:L404-L412

```
func BuildRangeProof(proofStart, proofEnd int, leaves, leafHashes [][]byte, hasher
Hasher) (*NamespaceRangeProof, error) {
...
	if proofEnd < size {
		postIndex := proofEnd + 1
		postProof, _, err := hasher.Open(leafHashes, postIndex)
		if err != nil {
			return res, err
		}
		res.postIndex = postIndex
		res.openPostIndex = KzgOpen{postIndex, leaves[postIndex], postProof}
	}
...
}
```

However, the code uses `size` instead of `size-1` for the conditional check, which causes `postIndex == size` when `proofEnd == size-1`. This results in an out-of-bound index and will subsequently trigger a panic.

Impact: When attempting to construct inclusion proofs for the maximum namespace ID, an "index out of range" error will be triggered, preventing the proof from being generated in this case.

Proof of Concept

```
func TestBuildRangeProof(t *testing.T) {
	data := [][]byte{
		append(namespace.ID{0}, []byte("leaf_0")...),
		append(namespace.ID{1}, []byte("leaf_1")...),
		append(namespace.ID{3}, []byte("leaf_3")...),
	}
	nidSize := 1
	tree := New(sha256.New(), NamespaceIDSize(nidSize))

	for _, d := range data {
		err := tree.Push(d)
		assert.Equal(t, nil, err)
	}

	root, err := tree.Root()
	assert.Equal(t, nil, err)

	proof, err := tree.ProveNamespace(namespace.ID{3})
	assert.Equal(t, nil, err)

	got := proof.VerifyNamespace(sha256.New(), namespace.ID{3}, root)
	assert.False(t, got)
}
```

Recommendation

Change `` `size` `` to `` `size-1` `` to comply with the description in the document.

2.3 Informational Findings

2. VerifyInclusion lacks code consistency	
Severity: Informational	Category: Improving readability
Target: - nubit-kzg/kzgproof.go	

Description

The `VerifyInclusion` function is used to verify the inclusion proof in the namespace proof and validate whether the `openPostIndex` is valid using the passed commitment.

In this function, the code uses two different methods to call fields. Most of the code directly accesses different `KzgOpen` fields in the proof struct using field names such as `proof.openStart`, `proof.openEnd`, and `proof.openPreIndex`. Additionally, the code employs the method `proof.OpenPostIndex()` for access. This approach can also be observed in the `VerifyAbsence` function. Therefore, the code access to fields should be unified to comply with the design philosophy of Go language, which emphasizes code readability and conciseness.

Impact: No security impact with the code base, but it can indeed reduce code readability and potentially confuse other developers when reading and understanding the code.

Recommendation

Replace `proof.openStart`, `proof.openEnd`, and `proof.openPreIndex` with `proof.OpenStart()`, `proof.OpenEnd()`, and `proof.OpenPreIndex()` in the `VerifyInclusion` method to maintain consistency with other functions and improve code consistency and maintainability.

3. The branch that checks `err` will never be executed

Severity: Informational	Category: Redundancy
Target: - rsmt2d/extendeddatacrossword.go	

Description

The `solveCrossword` function repetitively calls `solveCrosswordRow` and `solveCrosswordCol` functions within a `for` loop to obtain copies of specific rows or columns from `eds` and passes them into the core function `rebuildShares` for attempted repair.

To ensure uninterrupted continuous repair of each row and column in the `solveCrossword` function's `for` loop, the `rebuildShares` function is designed to always return `nil` for its error return value.

extendeddatacrossword.go:L147-L153

```go
func (eds *ExtendedDataSquare) solveCrosswordRow(
	r int,
	rowRoots [][]byte,
	colRoots [][]byte,
) (bool, bool, error) {
...
	rebuiltShares, isDecoded, err := eds.rebuildShares(shares)
	if err != nil {
		return false, false, err
	}
	if !isDecoded {
		return false, false, nil
	}
...
}
```

extendeddatacrossword.go:L220-L226

```go
func (eds *ExtendedDataSquare) solveCrosswordCol(
	c int,
	rowRoots [][]byte,
	colRoots [][]byte,
) (bool, bool, error) {
...
	rebuiltShares, isDecoded, err := eds.rebuildShares(shares)
	if err != nil {
		return false, false, err
	}
	if !isDecoded {
		return false, false, nil
	}
...
}
```

This means that the handling of the `err` variable in the yellow-highlighted portion of the code is meaningless, because this part will never be executed.

Impact: No security impact with the code base, just keep the code clear.

Recommendation

Do not accept the third parameter returned by the `` `rebuildShares` `` function and remove the handling of the `` `err` `` variable in the above code. Alternatively, modify the `` `rebuildShares` `` function to only return the first two return values.

4. Lack of boundary condition checks	
Severity: Informational	Category: Business Logic
Target: - rsmt2d/extendeddatacrossword.go - rsmt2d/extendeddatasquare.go - rsmt2d/datasquare.go	

Description

Currently, many functions in these three files do not check for the condition `eds.width == 0` and `ds.width == 0`, allowing empty `dataSquare` or `ExtendedDataSquare` structures to successfully execute the function process without returning any warnings or errors. For example, it is possible to create an empty `dataSquare` using the `newDataSquare` function, compute an `eds` from an empty slice using the `ComputeExtendedDataSquare` function, and repair the `eds` using the `Repair` function, etc.

Impact: No security impact with the code base. However, due to the lack of checks for extreme cases such as `eds.width` and `ds.width`, developers might be misled, leading to incorrect program execution that appears successful without errors, thus creating other potential issues in subsequent development.

Recommendation

Check the boundary conditions of the input parameters mentioned above, and promptly store error or log messages to provide alerts.

5. Use a more complex implementation to obtain the min and max namespaces

Severity: Informational	Category: Improving readability
Target: - nubit-kzg/nmt.go	

Description

The `ProveNamespace` function returns a range proof for a given namespace ID in a `NamespacedMerkleTree`. If the namespace ID is outside the tree's range, it returns an empty proof. However, since a more concise method for obtaining the minimum and maximum namespaces has already been implemented in the code, this section should be modified to enhance readability.

nmt.go:L291-L297

```go
func (n *NamespacedMerkleTree) ProveNamespace(nID namespace.ID) (*NamespaceRangeProof,
error) {
...
	root, err := n.Root()
	if err != nil {
		return &NamespaceRangeProof{}, fmt.Errorf("failed to get root: %w", err)
	}
	// extract the min and max namespace of the tree from the root
	treeMinNs := namespace.ID(MinNamespace(root, n.NamespaceSize()))
	treeMaxNs := namespace.ID(MaxNamespace(root, n.NamespaceSize()))
...
}
```

Impact: No security impact with the code base, solely to improve readability.

Recommendation

Revise the code to the following content.

nmt.go

```go
func (n *NamespacedMerkleTree) ProveNamespace(nID namespace.ID) (*NamespaceRangeProof,
error) {
    ...
    treeMinNs, err := n.MinNamespace()
    if err != nil {
        return &NamespaceRangeProof{}, fmt.Errorf("failed to get root: %w", err)
    }
    treeMaxNs, err := n.MaxNamespace()
    if err != nil {
    return &NamespaceRangeProof{}, fmt.Errorf("failed to get root: %w", err)
    ...
}
```

6. The Equal function is missing a comparison for the `InclusionOrAbsence` field	
Severity: Informational	Category: Best Practice
Target: - nubit-kzg/kzgproof.go	

Description

The `Equal` function is used to compare two `NamespaceRangeProof` instances and returns a boolean indicating whether the two proofs are equal.

kzgproof.go:L158-L165

```go
func (proof NamespaceRangeProof) Equal(input *NamespaceRangeProof) bool {
	return proof.start == input.start && proof.end == input.end &&
		proof.preIndex == input.preIndex && proof.postIndex == input.postIndex &&
		proof.openStart.Equal(&input.openStart) &&
		proof.openEnd.Equal(&input.openEnd) &&
		proof.openPreIndex.Equal(&input.openPreIndex) &&
		proof.openPostIndex.Equal(&input.openPostIndex)
}
```

However, this code snippet lacks the comparison of the `InclusionOrAbsence` field within the two `NamespaceRangeProof` structures.

Impact: No security impact with the code base, just to make the comparison of two proofs more comprehensive.

Recommendation

Add a comparison for the `InclusionOrAbsence` field in the `Equal` function.

Appendix

Appendix 1 - Files in Scope

This audit covered the following files:

File	SHA-1 hash
nubit-kzg/commit.go	4bd1ac0c861888ac040416503eb9edbd62f1c7ec
nubit-kzg/hasher.go	727082ae06be4a690443b8c216d981c94e1d88dd
nubit-kzg/kzgproof.go	2b040abfe76702033d4ff9c52fcfde1e0c5128a4
nubit-kzg/nmt.go	197344f169be73cf41436e87af5367b9bf5dab22
nubit-kzg/proof.go	3278e788252fbc3fb6ec3f24501135a94ccb21ed
nubit-kzg/subrootpaths.go	fd33326c0f7557b812a946f63aac030d2fbe2980
rsmt2d/codecs.go	f1acd9f2672e98fb38dbd684dcebc310223b1ee8
rsmt2d/datasquare.go	78a744f009403fbfbd8fd64472ac2c05970a18a5
rsmt2d/extendeddatacrossword.go	a033defca576da01ace95f3fce269c08f03f81fb
rsmt2d/extendeddatasquare.go	1e08145a3056fcbed8352ce66d358027015f0418
rsmt2d/kzg.go	9e89609d75017766e3e2d7d49f3d1e640feff36a
rsmt2d/leopard.go	6cde064159237a9dccf46596b225ff808d37a4c1
rsmt2d/tree.go	ed46749526a5607c954f1c4b97ab7b37c8b680cd
rsmt2d/utils.go	c1abaa66927b99e9162560d8421fe5dd0e882f02