

HACKEN

PHANTASMA SECURITY ANALYSIS REPORT

Intro

This report may contain confidential information about IT systems and the intellectual property of the Customer, as well as information about potential vulnerabilities and methods of their exploitation.

The report can be disclosed publicly after prior consent by another party. Any subsequent publication of this report shall be without mandatory consent.

Name	Phantasma-NG
Website	https://phantasma.io
Repository	https://github.com/phantasma-io/phantasma-ng
Commit	5e5e105e267c2bc9cd7544b63fe333be8d5a553e
Platform	Layer-1 Protocol
Network	Phantasma (testnet)
Languages	C#
Methods	Automated Code analysis, Manual review, Fuzz testing
Auditor	Ahmad Pouladzade(a.pouladzade@hacken.io)
Approver	Andriy Kashcheyev (a.kashcheyev@hacken.io), Luciano Ciattaglia (l.ciattaglia@hacken.io)
Timeline	05.06.2022 - 22.12.2022
Changelog	06.10.2022 (Updates)
Changelog	27.12.2022 (Updated Report)
Changelog	09.01.2023 (Updated Report)
Changelog	31.01.2023 (Final Report)

Table of contents

- **Summary**
 - Documentation quality
 - Code quality
 - Architecture quality
 - Security score
- **Issues**
 - Transaction Validation (Signature)
 - No Gas calculation for Contract Opcodes
 - Tomb Compiler Issue
 - Low Code Test Coverage
 - VirtualMachine Unit Tests
 - Hardcoded gas limit
 - Ignoring Gas Target
 - Lack Of Documentation
 - Null Pointer Exceptions
 - Runtime Random Number Generator
 - No Compatible Wallet (Ecto)
 - Always Null Variables
 - Execution Frame Unit Tests
 - IDisposableObject May Not Disposed
 - Missing Dispose Call On IDisposable Object
 - Operation On Floating Point Values
 - Possible Deadlock
 - Runtime Values ranges limits assert
 - Unsafe Random Number Generation
 - Plain Text Private Key in keystore file
 - Build Flow for Release artifacts
 - Configuration Settings
 - Deployment
 - Disassembler Unit Tests
 - Ed25519 Private Key ambiguity
 - Empty Interfaces
 - Incorrect Namespace convention
 - Multisig Verification
 - Noncompliant Conditional Statement
 - Null Pointer Exceptions (Interop)
 - Random Crash in node initialization
 - Result Of Call To Method Is Ignored
 - Return In Loop
 - Unreachable Dispose Statement On Exception
 - Incomplete Smart Contracts SDK And Toolchain
- **Fuzz Tests**
 - Block Storage Fuzz Tests
- **Disclaimers**
 - Hacken disclaimer
 - Technical disclaimer

Summary

Phantasma and Phantasma-NG are Layer-1 blockchain protocol implementations. [Phantasma website](#) describes the projects as "custom built blockchain and ecosystem focused on enabling gamers, artists and grandmas alike to take advantage of the immense disruptive power that blockchain technology offers"

The [Whitepaper](#) lists the following core features:

- Side-chains (unlimited number)
- x10 Transactions per Block
- Governance Token (SOUL) also used for Staking
- Reward in SOUL to Block producers
- KCAL token as a utility token (network resource) to be used for transactions fees instead of SOUL
- PoS Consensus mechanism
- Active and Standby Block Producers
- On-chain Governance (via Stake Voting)
- Decentralized Storage for dApps with encryption and erasure coding scheme
- Cosmic Swap Framework (on-chain App)
- Oracles (on-chain App)
- Integrated Name System
- NFT

[Phantasma-NG](#) is a partial refactor, rewrite and change of [Phantasma codebase](#)

The Audit was performed against commit hash: [5e5e105e267c2bc9cd7544b63fe333be8d5a553e](#)

The codebase CLOC statistics:

Language	files	blank	comment	code
C#	367	14958	8970	73577
JSON	43	59	0	45975
Go	30	1623	379	32171
Protocol Buffers	25	261	191	1068
TOML	4	428	1228	452
MSBuild script	13	20	18	278
Bourne Shell	9	58	11	234
YAML	3	10	0	95
Markdown	2	33	0	91
XML	3	0	0	20
Dockerfile	1	5	1	9
SUM:	500	17455	10798	153970

Protocol consists of the following additional building blocks and subsystems:

- P2P and Consensus Engine based on [Tendermint](#)
- PoS for Sybil resistance (as native SmartContract concept)
- Custom Virtual Machine for external Smart Contracts
- Custom programming language for external Smart Contracts: [TOMB](#)
- Native Smart Contracts precompiles (integral part of the Node) written in node implementation native language C#
- EDDSA (Ed25519 signature schema) for Block Producers and User wallets

We have to highlight that at the time of Audit, Phantasma-NG implementation does not include (or partial) the following Whitepaper specified features which we consider part of core implementation (not dApps):

- Active and Standby Block Producers (partial implementation)

Phantasma-ng mainnet initially will have 5 Block producers, which will be bootstrapped and managed by Phantasma organization, but will be gradually expanded with additional Block producers outside of Phantasma organization to improve decentralization and censorship resistance. From the Block Trilemma perspective Phantasma Networks will be centralized with censorship risks at initial stage. We did not test scalability aspect but we see no serious architectural limitation (VM, P2P) for a validators set to achieve a relatively large transactions count. At the time of Audit, Block Producers (Validators) are selected based on time slot and voting power (Consensus)

Phantasma has two type of executable dApps (SmartContracts):

1. Native Contracts
2. External Contracts

Native Contracts are written in C# and embedded within a node executable. They are not simple precompile primitives but rather large and non-trivial logic (e.g. ExchangeContract, MarketContract). Some Native Contracts have intrinsically linked functionality for the Protocol layer.

The following Native Contracts are out of scope for the this audit report:

- Account - name registration, script attached to Account
- Consensus - PollContract, used on polls for new validators and changes on everything that's part of the governance values (fee amounts, etc.)
- Gas - Used in every single transaction, responsible for paying gas for each tx.
- Governance - holds all governance values, which can be changed through votes.
- Market - Sale and Auction contract for NFTs, can be used by anybody to create an auction or sell an NFT.
- Stake - Responsible for minting KCAL for staked SOUL tokens

Documentation quality

The total Documentation Quality score is **7.0** out of **10**. The project documentation was either outdated or referred to the previous version of Phantasma implementation. External Contract development documentation was difficult to be found and not up to date. Documentation on how to build the project was limited.

At the time of the initial investigation there was limited specifications for Blockchain design and no Architectural documentation. Phantasma development team had to be contacted for most of the questions (build, dockerization, contract development, contract deployment). Phantasma team made improvements to the documention and together with deprecation of custom VM language the Documentaion quality was improved. There is a new documentation available on <https://phantasma.gitbook.io/phantasmachain> including information how to build and run the node.

Code quality

The total Code Quality score is **7.6** out of **10**. The code base is well modularized, follows comprehensive naming convention and easy to navigate. Codebase lacks comments and requires "reverse-engineering" or help and clarification from Phantasma technical team. Codebase contains outdated and obsolete code which slightly affects the score but complicates the analysis. Continuous Integration process and integration with code analysis tools was addressed and integrated by Phantasma team.

The original unit tests low coverage (26%) was addressed and improved by Phantasma team in particular in two major modules:

- Phantasma.Business: **77.0%**
- Phantasma.Core: **85.0%**

Architecture quality

The architecture quality score is **7.0** out of **10**. The architecture follows the traditional blockchain paradigm with internal modularization and functional separation. The selection of Tendermint for low-level consensus engine and p2p substem is a good choice. We advised Phantasma team to follow a similar approach with external contracts VM (using industry-tested VM execution environment, non-custom language and toolchain) The lack of Architectural specification and documentation is a minor issue but could be an obstacle for external contributions.

Security score

The security score is **8.5** out of **10**. As a result of the analysis, the code contained **1** Critical, **1** High, **3** Medium severity security-related issues. All raised issues had been acknowledged and fixed by the team, and at the time of the report, Phantasma team had the following resolutions:

- 1 Critical Issue fixed
- 1 High Issue fixed
- 1 Medium Issue fixed
- 1 High and 1 Medium issues rejected

The overall project Total score is **8.0** out of **10**.

Issues

Transaction Validation (Signature)

ID	PHA-1011
Severity	CRITICAL
Scope	Security, Consensus Rules
Status	Fixed (57b966e72994108762cfc43aeb4de53b2971c89d)
Reference	https://github.com/hknio/phantasma-ng/blob/2bdc7fa861f9ed20214dc6d60e8ee066f00f04aa/Phantasma.Business/src/Blockchain/Chain.cs

Details

Transaction signature validation was not correct

<https://github.com/hknio/phantasma-ng/blob/2bdc7fa861f9ed20214dc6d60e8ee066f00f04aa/Phantasma.Business/src/Blockchain/Chain.cs#L574>

The code was commented, thus there was **NO** validation and transactions with forged Address will be executed

If this code was uncommented,

```
//if (!transaction.HasSignatures)
//{
//    throw new ChainException("Cannot execute unsigned transaction");
//}
```

the signature validation was still **NOT** correct:

<https://github.com/hknio/phantasma-ng/blob/2bdc7fa861f9ed20214dc6d60e8ee066f00f04aa/Phantasma.Core/src/Domain/Transaction.cs>:

```
public bool HasSignatures => Signatures != null && Signatures.Length > 0;
```

Recommendation

Use mandatory signature verification on every transaction

No Gas calculation for Contract Opcodes

ID	PHA-410
Severity	HIGH
Scope	VM
Status	Fixed (6173e4f26c8aa535fa0e8d1f2e0e9abf8d53f2c6)
Reference	Phantasma-ng

Details

There was no execution fee for running smart contracts on Phantasma node. It caused the Phantasma node to be highly vulnerable to the DoS attack by running a function of the smart contract which contained an infinite while loop.

To reproduce the bug you have to deploy the below contract and call the test function with very big integer number as an input. You may see the node gets crashed or frozen.

```
contract while_test {
  public test(a:number):number
  {
    local counter :number := 0;
    while (counter != a){
      counter += 1;
    }
    return counter;
  }
}
```

All the nodes which has the open RPC ports could be affected by this bug.

Recommendation

You may either deduct the execution fee from the user who is calling the smart contract or define a maximum opcodes or gas inside the VM for running the opcodes.

Tomb Compiler Issue

ID	PHA-418
Severity	HIGH
Scope	VM, Toolchain
Status	Fixed (f95d8ee335f8580d3a3582bc827fe20ff168b558)
Reference	Phantasma-ng

TOMB

Details

TOMB compiler had severe issues, for example we can define variables and functions with compiler reserved keywords.

```
struct else {
  switch: number;
  throw: string;
}
struct throw {
  case: else;
}
contract contract {
  constructor(owner:address)
  {
  }
  public public(private :number, constructor:number):number
  {
    return private + constructor;
  }
}
```


Compiling of the above contract would be successful but after calling the method of the contract node become quiet unstable and unpredictable.

Recommendation

Redesign parser of `TOMB` compiler to throw compile error when user wants to use reserved keywords in invalid statements.

Comment

TOMB compiler is to be deprecated and replaced.

Low Code Test Coverage

ID	PHA-409
Severity	HIGH
Scope	Code Quality
Status	Fixed (bcb4189d246a65157cfbdfb831a8a545d8202f33)
Reference	Phantasma-ng

Details

There was a lack of unit tests and low tests code coverage.

Coverage	0.9%
Lines to Cover	45,806
Uncovered Lines	45,400
Line Coverage	0.9%

Recommendation

Add more unit tests and integration tests to the project to increase code coverage. Code coverage should be at least 85%.

Comments

Test coverage has been significantly improved in core modules <https://github.com/phantasma-io/phantasma-ng/pull/156>

VirtualMachine Unit Tests

ID	PHA-108
Severity	HIGH
Scope	Code Quality
Status	Fixed (AssemblerTests.cs)
Reference	https://github.com/hknio/phantasma-ng/blob/master/Phantasma.Business/tests/VM/VirtualMachi

Details

There was a lack of unit tests.

Non-exhaustive unit tests not covering ranges and corner cases to prevent resource DoS (e.g. stack overflow, aggressive GC)

VM is the most complex and critical part of the Blockchain.

Taking into account current documentation, proprietary nature and novelty (e.g. no existing documentation and test suites from other projects could be quickly applied), at the current state it is impossible to properly audit and verify VM without significant RnD effort (unit test and integration tests development)

Recommendation

Implement Unit tests covering edge cases.

For example:

1. large number of push
2. Same frame push
3. Invalid push/pop parameters
4. Uncorrelated Push/pop
5. Check frames expected values
6. Different simple scripts inputs

Hardcoded gas limit

ID	PHA-402
Severity	MEDIUM
Scope	VM
Status	Fixed (53c9f078e2d2d1a193a5c59c5b71ae637b4b0108)
Reference	DeliverTx

Details

The gas limit was hardcoded for executing transaction. Which could cause an undesirable runtime termination of a smart contract.

Code : [DeliverTx](#)

```
public TransactionResult DeliverTx(ByteString serializedTx)
{
    TransactionResult result = new();
    var txString = serializedTx.ToStringUtf8();
    var tx = Transaction.Unserialize(Base16.Decode(txString));
    Log.Information($"Deliver tx {tx}");
    try
    {
        CurrentTransactions.Add(tx);
        var txIndex = CurrentTransactions.Count - 1;
        var oracle = Nexus.GetOracleReader();
        using (var m = new ProfileMarker("ExecuteTransaction"))
```

```

    {
      result = ExecuteTransaction(txIndex, tx, tx.Script, this.CurrentBlock.Validator,
        this.CurrentBlock.Timestamp, this.CurrentChangeSet, this.CurrentBlock.Notify, oracle,
        ChainTask.Null, 100000); //TODO: hardcoded gas limit
      if (result.Code == 0)
      {
        if (result.Result != null)
        {
          var resultBytes = Serialization.Serialize(result.Result);
          this.CurrentBlock.SetResultForHash(tx.Hash, resultBytes);
        }
      }
    }
  }
}
catch (Exception e)
{
  Log.Error("exception " + e);
  // log original exception, throwing it again kills the call stack!
  Log.Error("Exception was thrown while processing {0} error: {1}", result.Hash, e.Message);
  this.CurrentTransactions.Remove(tx);
  this.CurrentChangeSet.Clear();
  result.Code = 1;
  result.Codespace = e.Message;
}
return result;
}

```

Recommendation

The gas limit should be retrieved from the transaction input and Block gas limit must be specified in the VM specification

Ignoring Gas Target

ID	PHA-404
Severity	MEDIUM
Scope	Code Quality, VM
Status	Fixed: (6f6c441be66be559dfb91fb53b5f6ba870f0ebb9)
Reference	Phantasma-ng

Details

Null gas target was ignored. The `VMException` was initialized but never thrown. This could cause executing the rest of the method and ignoring `null GasTarget`.

Code:

[ExtCalls::Runtime_GasTarget](#)

```

private static ExecutionState Runtime_GasTarget(RuntimeVM vm)
{
  if (vm.GasTarget.IsNull)
  {
    new VMException(vm, "Gas target is now available yet");
  }
  var result = new VMObject();
  result.SetValue(vm.GasTarget);
  vm.Stack.Push(result);
}

```

```
    return ExecutionState.Running;  
}
```

Recommendation

Throw `VMException` to stop execution of the method because of `null GasTarget`.

Lack Of Documentation

ID	PHA-407
Severity	MEDIUM
Scope	Documentation
Status	Fixed (https://phantasma.gitbook.io/phantasmachain)
Reference	Phantasma-ng

Details

[\[Phantasma-Spook\]](#) [\[Phantasma-TOMB\]](#) [\[Phantasma-SDK\]](#) [\[Phantasma-Chain\]](#) [\[Phantasma-UnitySDK\]](#)

During initial stages of an audit Phantasma repositories lacked technical documentation about participating in the network, launching a node, implementing/deploying and calling smart contracts.

Phantasma team made improvements to documentation and created up to date resources: <https://phantasma.gitbook.io/phantasmachain> API Portal has been added: <https://apidocs.phantasma.io/>. The functionality can be derived from endpoints, but we expect proper description and business context.

Recommendation

Phantasma may need to enrich the technical documents as well as github repository's readme files.

Technical documentation around Node specification, VM and SmartContract development still requires improvements.

Null Pointer Exceptions

ID	PHA-411
Severity	MEDIUM
Scope	Consensus
Status	Fixed (Chain.cs)
Reference	Phantasma-ng

Details

Unhandled `NullPointerException` could occur due to some statements. If any exceptions happened by calling `AddBlock` method, the `lastBlock` could remained as `null` causing `NullReferenceException` in return statement.

```
public byte[] Commit()
{
    Block lastBlock = null;
    try
    {
        AddBlock(this.CurrentBlock, this.CurrentTransactions, 0, this.CurrentChangeSet);
        lastBlock = this.CurrentBlock;
        this.CurrentBlock = null;
        this.CurrentTransactions.Clear();
        Log.Information($"Committed block {lastBlock.Height}");
    }
    catch (Exception e)
    {
        Log.Error("Error during commit: " + e);
    }
    return lastBlock.Hash.ToArray();
}
```

Recommendation

Follow the safe code style to help prevent the occurrence of `NullReferenceException`.

Handle the exception in the way that make sure `lastBlock` never would be null.

Runtime Random Number Generator

ID	PHA-1002
Severity	MEDIUM
Scope	Cryptography, Security, VM
Status	Fixed (removed)
Reference	https://github.com/hknio/phantasma-ng/blob/2bdc7fa861f9ed20214dc6d60e8ee066f00f04aa/Phantasma.Business/src/Blockchain/Runtime.cs#L495

Details

Random generator does not have enough entropy to generate secure randomness:

```
public BigInteger GenerateRandomNumber()
{
    if (_randomSeed == 0 && this.Transaction != null)
    {
        SetRandomSeed(this.Transaction.Hash);
    }
    _randomSeed = ((RND_A * _randomSeed) % RND_M);
    return _randomSeed;
}
```

All parameters (Timestamp, RND_A, RND_M, transaction hash) are either completely visible or easily predictable

`GenerateRandomNumber()` is used as randomness source for VM exposed functionality: [ExtCalls.cs](#)

Recommendation

Use `SecureRandom` library from .NET: <https://docs.microsoft.com/en-us/dotnet/api/system.security.cryptography.randomnumbergenerator?view=net-6.0>

Comment

Phantasma team insisted that this randomness generation is `by design` :

The runtime number generator has to generate deterministic values for the transactions to be reproducible. The actual seed comes from several things including the block generation time which is mostly out of control of attackers, so they cant force a specific random number to be generated

We highlight that since it is used in VM in external calls there is a possible attack vector with predicting randomness, thus we consider it a security scope vulnerability.

The function `GenerateRandomNumber` was later removed, fixing this issue.

No Compatible Wallet (Ecto)

ID	PHA-423
Severity	MEDIUM
Scope	RPC, Documentation
Status	Fixed
Reference	Ecto Wallet

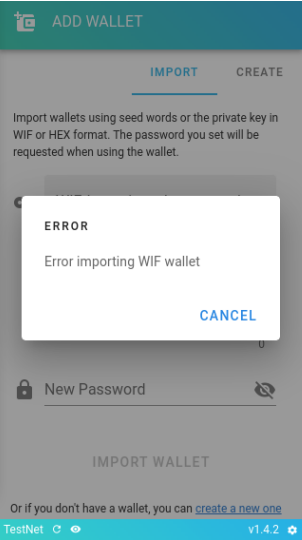
Details

Ecto Wallet advertized on Phantasma site (<https://phantasma.io/wallets/>) was not compatible with the Phantasma node.

```

2022-08-11 12:02:49Z 1246 [INF] <=> HTTP POST /rpc responded 415 in 5.6888 ms
2022-08-11 12:02:49Z 1340 [INF] <=> HTTP POST /rpc responded 415 in 0.6143 ms
2022-08-11 12:03:10Z 9668 [INF] <=> HTTP POST /rpc responded 415 in 0.6188 ms
2022-08-11 12:03:10Z 9794 [INF] <=> HTTP POST /rpc responded 415 in 0.4584 ms
2022-08-11 12:05:40Z 1691 [INF] <=> HTTP POST /rpc responded 415 in 0.4041 ms
2022-08-11 12:05:40Z 1867 [INF] <=> HTTP POST /rpc responded 415 in 0.4622 ms
2022-08-11 12:05:48Z 6912 [INF] <=> HTTP POST /rpc responded 415 in 0.5963 ms
2022-08-11 12:06:15Z 9467 [INF] <=> HTTP POST /rpc responded 415 in 0.4233 ms
2022-08-11 12:06:43Z 0043 [INF] <=> HTTP POST /rpc responded 415 in 0.5457 ms
2022-08-11 12:06:43Z 0180 [INF] <=> HTTP POST /rpc responded 415 in 0.3435 ms
2022-08-11 12:07:02Z 8984 [INF] <=> HTTP POST /rpc responded 415 in 1.1934 ms
2022-08-11 12:07:02Z 9285 [INF] <=> HTTP POST /rpc responded 415 in 0.3074 ms
2022-08-11 12:07:11Z 9606 [INF] <=> HTTP POST /rpc responded 415 in 0.4824 ms
2022-08-11 12:07:28Z 9485 [INF] <=> HTTP POST /rpc responded 415 in 0.3614 ms
2022-08-11 12:07:28Z 9619 [INF] <=> HTTP POST /rpc responded 415 in 0.4052 ms
2022-08-11 12:07:38Z 8581 [INF] <=> HTTP POST /rpc responded 415 in 0.4879 ms
2022-08-11 12:09:58Z 7275 [INF] <=> HTTP POST /rpc responded 415 in 0.4328 ms
2022-08-11 12:09:58Z 7436 [INF] <=> HTTP POST /rpc responded 415 in 0.4709 ms
2022-08-11 12:10:07Z 7505 [INF] <=> HTTP POST /rpc responded 415 in 0.7623 ms
2022-08-11 12:10:07Z 7746 [INF] <=> HTTP POST /rpc responded 415 in 0.6309 ms
2022-08-11 12:10:16Z 0068 [INF] <=> HTTP POST /rpc responded 415 in 0.6235 ms
2022-08-11 12:15:42Z 0278 [INF] <=> HTTP POST /rpc responded 415 in 0.4991 ms
2022-08-11 12:15:42Z 0422 [INF] <=> HTTP POST /rpc responded 415 in 0.4877 ms
2022-08-11 12:15:57Z 7250 [INF] <=> HTTP POST /rpc responded 415 in 0.4814 ms
2022-08-11 12:15:57Z 7465 [INF] <=> HTTP POST /rpc responded 415 in 0.3126 ms
2022-08-11 12:16:04Z 9762 [INF] <=> HTTP POST /rpc responded 415 in 0.7333 ms
2022-08-11 12:16:48Z 0251 [INF] <=> HTTP POST /rpc responded 415 in 0.4851 ms
2022-08-11 12:16:48Z 0434 [INF] <=> HTTP POST /rpc responded 415 in 0.4741 ms
2022-08-11 12:16:54Z 2673 [INF] <=> HTTP POST /rpc responded 415 in 0.5635 ms

```



InvokeScript or InvokeRawScript are used to call the chain without changing it - it fetches values from the chain without changing its state

Recommendation

Fix the incompatibility of the wallet with current version of node rpc methods.

Always Null Variables

ID	PHA-400
Scope	Code Quality
Severity	MEDIUM
Status	Fixed (04590e7ad8564caa7ae8f83a22c0cd7874580b78)

Details

Some variables were always `null`.

`TokenUtils::ExecuteScript()`
`Node::PromptGenerator()`

A reference to null should never be dereferenced/accessed. Doing so will cause a `NullReferenceException` to be thrown. Uncaught exception will cause abrupt node termination.

```
private static VMObject ExecuteScript(StorageContext storage, IChain chain, byte[] script, ContractInterface abi, string methodName)
{
    var method = abi.FindMethod(methodName);
    if (method == null)
    {
        throw new Exception("ABI is missing: " + method.name);
    }
    ...
}
```

```
private string PromptGenerator()
{
    var height = this.ExecuteAPIR("getBlockHeight", new string[] { "main" });
    return string.Format(prompt, height.Trim(new char[] { ' ' }));
}
public string ExecuteAPIR(string name, string[] args)
{
    // TODO fix
    /*var result = _nexusApi.Execute(name, args);
    if (result == null)
    {
        return "";
    }
    return result;*/
    return null;
}
```

Recommendation

Follow the safe code style (e.g. Rigorous Null checks, [Option pattern](#)) to help prevent the occurrence of `NullReferenceException`.

Alternative solution is enabling [Null Safety](#) feature for the project and use various null-related C# operators:

- null-forgiving (!) operator
- null-coalescing (??) operator
- null-conditional (?.) operator

Execution Frame Unit Tests

ID	PHA-106
Severity	MEDIUM
Scope	Code Quality, VM
Status	Fixed: (e6f675425f1ac554496ce0735dc0a104d5a5225d)
Reference	https://github.com/hknio/phantasma-ng/blob/master/Phantasma.Business/tests/VM/ExecutionFrameTest.cs

Details

There was a lack of unit tests.

ExecutionFrame is used in critical functionality path: <https://github.com/hknio/phantasma-ng/blob/2bdc7fa861f9ed20214dc6d60e8ee066f00f04aa/Phantasma.Business/src/VM/VirtualMachine.cs#L91>

Recommendation

Implement Unit tests covering Frames variations

IDisposableObject May Not Disposed

ID	PHA-403
Severity	MEDIUM
Scope	Code Quality
Status	Fixed: (d34bd3482991acd756b6ecf616134d5a42135a93)
Reference	StreamWriter.Dispose()

Details

Exception during execution of `stop` method could lead to resources leak.

```
void IDisposable.Dispose()
{
    new Thread(() =>
    {
        Stop();
    }).Start();
}
```

Recommendation

You can use `try-catch-finally` statement in `stop` method.

Missing Dispose Call On IDisposable Object

ID	PHA-408
Severity	MEDIUM
Scope	Code Quality
Status	Fixed (6d82eb1c5e4b61d996710eedac3642595839cdf9)
Reference	Phantasma-ng

Details

IDisposable objects should dispose their resources.

Code :

```
ECDsa::FromDER()  
CompressionUtils::Decompress()  
EndpointCacheManager::Add()  
EndpointCacheManager::Get()  
CacheMiddleware::Invoke()  
RequestUtils::Request()  
RequestUtils::RequestAsync()  
Utils::ToJsonString()
```

```
public static byte[] FromDER(byte[] signature)  
{  
    var decoder = new Asn1InputStream(signature);  
    var seq = decoder.ReadObject() as DerSequence;  
    if (seq == null || seq.Count != 2)  
        throw new FormatException("Invalid DER Signature");  
    var R = ((DerInteger)seq[0]).Value.ToByteArrayUnsigned();  
    var S = ((DerInteger)seq[1]).Value.ToByteArrayUnsigned();  
    byte[] concatenated = new byte[R.Length + S.Length];  
    Buffer.BlockCopy(R, 0, concatenated, 0, R.Length);  
    Buffer.BlockCopy(S, 0, concatenated, R.Length, S.Length);  
    return concatenated;  
}
```

Recommendation

Either calling `Dispose()` method or using `using` statement will dispose the resources and fix these issues.

Operation On Floating Point Values

ID	PHA-412
Severity	MEDIUM
Scope	Code Quality
Status	Fixed (86ced4d14973ea29f7061d67443bcb3cd27e21af)
Reference	Phantasma-ng

Details

Operations on floating point values could yield unexpected result.

Code	Utils::WeightedFilter()
Code	Utils::WeightedFilter()

```
internal static IEnumerable<TResult> WeightedFilter<T, TResult>(this IList<T> source, double start, double end, Func<T, TResult> func)
{
    ...
    if (source.Count == 0 || start == end) yield break;
    ...
}
```

Recommendation

Its better to use `epsilon` when trying to compare floating point values.

Possible Deadlock

ID	PHA-413
Severity	MEDIUM
Scope	Code Quality
Status	Fixed (EthereumInterop.cs#L67 , NeoInterop.cs#L74)
Reference	Phantasma-ng

Details

Locking with `String.Intern` could cause deadlocks.

Locking on strings is discouraged. The main reason is that (because of string-interning) some other code could lock on the same string instance and create a potential deadlock.

Code:

[EthereumInterop::Update](#)

```
public override IEnumerable<PendingSwap> Update()
{
    // wait another 10s to execute eth interop
    //Task.Delay(10000).Wait();
    try
    {
        lock (String.Intern("PendingSetCurrentHeight_" + EthereumWallet.EthereumPlatform))
        {
            ...
        }
    }
}
```

[NeoInterop::Update](#)

```
public override IEnumerable<PendingSwap> Update()
{
    lock (String.Intern("PendingSetCurrentHeight_" + "neo"))
    {
        ...
    }
}
```

```
...  
}
```

Recommendation

Suggested to use `ConcurrentDictionary` instead.

Runtime Values ranges limits assert

ID	PHA-1009
Severity	MEDIUM
Scope	Code Quality
Status	Fixed (cb592ec97a04fa8dca9da6832945dd7ee5192d2c)
Reference	https://github.com/hknio/phantasma-ng/blob/2bdc7fa861f9ed20214dc6d60e8ee066f00f04aa/Phantasma.Business/src/Blockchain/Runtime.cs

Details

All external input values should have been defensively validated against ranges / lengths / sizes

Example: <https://github.com/hknio/phantasma-ng/blob/2bdc7fa861f9ed20214dc6d60e8ee066f00f04aa/Phantasma.Business/src/Blockchain/Runtime.cs#L1036>

```
public BigInteger GetBalance(string symbol, Address address)  
{  
    Expect(TokenExists(symbol), $"Token does not exist ({symbol})");  
    var token = GetToken(symbol);  
    return Chain.GetTokenBalance(this.Storage, token, address);  
}
```

symbol and **address** with excessively large length values could be used to overwhelm VM resources to cause GC and potential node downtime.

Recommendation

Use very defensive asserts in all code paths which accept external inputs.

All external inputs should be treated as malicious and harmful to node operation and must be validated as early as possible

Unsafe Random Number Generation

ID	PHA-421
Severity	MEDIUM
Scope	Cryptography, Security
Status	Fixed (e6b72bd9ffe48ba8172cafaf800468517a57c8e0)
Reference	Phantasma-ng

Details

Cryptographically weak random generator used.

Code:

[SwapContract::CreatePool](#)

```
private Random rnd = new Random();
private Dictionary<string, Connection> _connections = new Dictionary<string, Connection>();
protected abstract WalletStatus ### Status { get; }
...
var bytes = new byte[32];
rnd.NextBytes(bytes);
token = Base16.Encode(bytes);
```

Recommendation

As the System.Random class relies on a pseudorandom number generator, it should not be used for security-critical application. In such context, the System.Cryptography.RandomNumberGenerator class which relies on a cryptographically strong random number generator (RNG) should be used in place.

Recommended Secure Coding Practices

```
using System.Security.Cryptography;
...
var randomGenerator = RandomNumberGenerator.Create(); // Compliant for security-sensitive use cases
byte[] data = new byte[32];
randomGenerator.GetBytes(data);
return BitConverter.ToString(data);
```

Plain Text Private Key in keystore file

ID	PHA-422
Severity	LOW
Scope	Keystore, Security
Status	Workaround (bcb4189d246a65157cfbdfb831a8a545d8202f33)
Reference	Phantasma-ng

Details

The private key of a validator was stored in a json file as plain text (used by Tendermint)

priv_validator_key.json :

```
{
  "address": "7CCF8D79D6B7ED4B0FACFD4C3C1FA547DEA0E5CD",
  "pub_key": {
    "type": "tendermint/PubKeyEd25519",
    "value": "IiiA7W0pkTGnIFi/zqIHjfgMt/1gLZUwo/Bt6L8F6Ms="
  },
  "priv_key": {
    "type": "tendermint/PrivKeyEd25519",
```

```
"value": "qJv1X0IF1M1jq5Nd0zeTBdd2nw0ajnNMhYHQDcpoPvoikIDtbSmRMacgWL/OogeN+Ay3/WAt1TCj8G3ovwXoyw=="  
}  
}
```

Using OS file permissions (e.g. 400) is insufficient.

Recommendation

In decentralized permissioned systems Blockchain node is run and can join by anyone. Access to the system can be compromised at different levels (weak password for ssh login, rootkit execution via malicious downloaded software, exploits in software on open ports...). Having validators keys in plaintext gives an intruder nearly immediate financial and technical attack vectors without absolutely any effort. Swiss cheese model for security is applicable here. Tendermint is a generic framework. Systems, which utilize its functionality must not rely on tendermint defaults. We recommend to follow standard keystore encryption methods (e.g. <https://github.com/ethereum/wiki/wiki/Web3-Secret-Storage-Definition>)

Comments

Description and limitations of the currently implemented workaround:

- The mainnet will be running with trusted validating (block producing) nodes only in a centralized network topology with hardened keys management policies. Decentralized nodes will be added shortly after the recommendation below is implemented. The Phantasma team is rapidly working towards this.
- Tendermint signing key will be stored on disk encrypted with AES-256. During node startup it should be decrypted (temporary file deleted immediately) and passed as environment variable. This approach partially reduces keys leakage. We still recommend to implement decryption internally and use keystore file format where IV, encrypted key and passphrase key derivation scheme. We strongly recommend to use encryption key as a derivation from passphrase with KBDF2 and a large number of iterations.

Build Flow for Release artifacts

ID	PHA-11
Severity	LOW
Scope	Code Quality
Status	Being Addressed
Reference	https://github.com/phantasma-io/phantasma-ng/releases

Details

Non verifiable download release artifacts

[Released packages](#) must be accompanied by digest and/or PGP Signature verifiable by external parties to reduce forged / malicious code fork or injection

Recommendation

For production (mainnet) releases:

1. Implement automated release workflow: <https://github.com/phantasma-io/phantasma-ng/tree/master/.github/workflows>
2. Sign with PGP and include signature as release artifact (most secure) or use SHA2 digest

Configuration Settings

ID	PHA-11
Severity	LOW
Scope	Documentation
Status	Fixed: (https://phantasma.gitbook.io/phantasmachain)
Reference	https://github.com/hknio/phantasma-ng

Details

There was no documentation describing configuration and settings for Node setup

Recommendation

Provide documentation with a list of possible configuration parameters (config file, location, arguments)

Deployment

ID	PHA-10
Severity	LOW
Scope	Documentation, Code Quality
Status	Fixed (https://phantasma.gitbook.io/phantasmachain)
Reference	https://github.com/hknio/phantasma-ng

Details

There was no production configuration deployment guidelines and instructions.

Recommendation

Provide documentation how to configure Node and run Node with production settings and guidelines

1. Resources required (CPU,Memory,Disk)
2. Networking (ports, NAT, firewall)
3. Security (keystore encryption, API/RPC credentials,)
4. Troubleshooting and maintenance (logging, log files, logs rotation)

Disassembler Unit Tests

ID	PHA-105
Severity	LOW
Scope	Code Quality

Status	Fixed: (e6f675425f1ac554496ce0735dc0a104d5a5225d)
Reference	https://github.com/hknio/phantasma-ng/blob/master/Phantasma.Business/tests/VM/DisassemblerTest.cs

Details

There was a lack of unit tests.

Disassembly is used in non-critical functionality path: <https://github.com/hknio/phantasma-ng/blob/2bdc7fa861f9ed20214dc6d60e8ee066f00f04aa/Phantasma.Business/src/VM/VirtualMachine.cs#L229>

Disassembler used for debugging RnD Contract Development purposes it could be indispensable tool

Test coverage should be more than **80%** overall across all project

Disassembler should have **100%** tests coverage for all individual OpCodes/Instructions

Recommendation

Implement Unit tests covering all Opcodes disassembly

Ed25519 Private Key ambiguity

ID	PHA-1001
Severity	LOW
Scope	Cryptography
Status	Fixed (6f6c441be66be559dfb91fb53b5f6ba870f0ebb9)
Reference	https://github.com/hknio/phantasma-ng/blob/master/Phantasma.Core/src/Cryptography/EdDSA/Ed25519.cs#L44

Details

method Ed25519PrivateKeyParameters is PrivateKey and not a seed although both are random:

```
var privateKeyParameters = new Ed25519PrivateKeyParameters(privateKeySeed, 0);
```

This was confusing from code maintenance since private key has higher security context than seed.

Recommendation

Rename `privateKeySeed` to `privateKey`

Empty Interfaces

ID	PHA-401
Severity	LOW
Scope	Code Quality

Status	Being Improved
Reference	Phantasma-ng

Details

[IChainSwap](#)

[IEndpointResponse](#)

Some interfaces do not declare any members. Defining and using these interfaces has no outcome in source code and project structure.

```
public interface IChainSwap
{
}
```

```
public interface IEndpointResponse
{
}
```

Recommendation

It's better to remove these interfaces and use attributes instead. Also adding some methods to these interface is a good idea.

Incorrect Namespace convention

ID	PHA-406
Severity	LOW
Scope	Code Quality
Status	Fixed (8ba0d90ce70d890d789dd4331b17c40f16b48749)
Reference	Phantasma-ng

Details

Some classes or interfaces have incorrect namespace that do not correspond to file location. Types are declared in namespaces in order to prevent name collisions and as a way to organize them into the object hierarchy. Types that are defined outside any named namespace are in a global namespace that cannot be referenced in code.

Note

Almost all files have an incorrect namespace. As an example consider this file path in project : `phantasma-ng/Phantasma.Business/src/Blockchain/Contracts/AccountContract.cs`

```
using System.Linq;
using System.Numerics;
using System.Collections.Generic;
using Phantasma.Core;
using Phantasma.Core.Context;
namespace Phantasma.Business.Contracts
{
}
```

Code:

There is no namespace for this code: [ContractNames](#)

Recommendation

Use sub-namespaces to organize/group related namespaces.

The correct name space should be like this:

```
namespace Phantasma.Business.Blockchain.Contracts;
```

Multisig Verification

ID	PHA-1012
Severity	LOW
Scope	Cryptography, Security
Status	Fixed (not used in normal operations)
Reference	https://github.com/hknio/phantasma-ng/blob/2bdc7fa861f9ed20214dc6d60e8ee066f00f04aa/Phantasma.Core/src/Domain/Transaction.cs

Details

Transaction signature validation for multisig could be very compute expensive and could be used for DoS attack because transaction is perfectly valid but very computationally *heavy*

<https://github.com/hknio/phantasma-ng/blob/2bdc7fa861f9ed20214dc6d60e8ee066f00f04aa/Phantasma.Core/src/Domain/Transaction.cs#L135>

Recommendation

1. Limit number signatures for transaction for ECDSA
2. Consider for the future aggregatable signatures algorithms like **BLS** or **Schnorr**

Client Comment

ECDSA signatures are not used in normal operation, we'll look into aggregatable signatures once it's necessary.

Noncompliant Conditional Statement

ID	PHA-420
Severity	LOW
Scope	Code Quality
Status	Fixed (71b87ccf1dbab2d7e4109bfa98d1d713b98cbf99)
Reference	Phantasma-ng

Details

Having all branches in a `if` chain with the same implementation was an error. Either a copy-paste error was made and something different should be executed, or there shouldn't be a `if` chain at all.

Code:

SwapContract::CreatePool

```
if ( tradeRatio == 0 )
{
    tradeRatio = tradeRatioAmount;
}
else
{
    // Ratio Base on the real price of token
    // TODO: Ask if this is gonna be implemented this way.
    //Runtime.Expect(tradeRatioAmount == tradeRatio, $"TradeRatio < 0 | {tradeRatio} != {tradeRatioAmount}");
    tradeRatio = tradeRatioAmount;
}
```

Serialization::Serialize

```
public static void Serialize(BinaryWriter writer, object obj, Type type)
{
    ...
    else
    if (type == typeof(ushort))
    {
        writer.Write((ushort)obj);
    }
    ...
    else
    if (type == typeof(ushort))
    {
        writer.Write((ushort)obj);
    }
    ...
}
```

Node::SetupNodeKeys

```
private PhantasmaKeys SetupNodeKeys()
{
    var keyStr = Environment.GetEnvironmentVariable("PHA_KEY");
    PhantasmaKeys nodeKeys = null;
    if (!string.IsNullOrEmpty(keyStr))
    {
        nodeKeys = new PhantasmaKeys(Convert.FromBase64String(keyStr));
    }
    if (nodeKeys is null)
    {
        nodeKeys = new PhantasmaKeys(Convert.FromBase64String(Settings.Default.Node.TendermintKey));
    }
    if (nodeKeys is null)
    {
        nodeKeys = PhantasmaKeys.FromWIF(Settings.Default.Node.NodeWif);
    }
    //TODO wallet module?
    return nodeKeys;
}
```

Recommendation

Make sure that all branches in a `if` chain have the different implementations.

Null Pointer Exceptions (Interop)

ID	PHA-423
Severity	LOW
Scope	Code Quality
Status	Fixed (Functionality deprecated)
Reference	Phantasma-ng

Details

Unhandled `NullReferenceException` could occur due to some statements. Initializing variables inside a `try-catch` statement without having a good logic in `catch` statement could leave a variable uninitialized.

Code: [EthereumInterop::GetInteropTransfers\(\)](#)

```
private static Dictionary<string, List<InteropTransfer>> GetInteropTransfers(Nexus nexus,
    TransactionReceipt txr, EthAPI api, string[] swapAddresses)
{
    Log.Debug($"get interop transfers for tx {txr.TransactionHash}");
    var interopTransfers = new Dictionary<string, List<InteropTransfer>>();
    Nethereum.RPC.Eth.DTOs.Transaction tx = null;
    try
    {
        // tx to get the eth transfer if any
        tx = api.GetTransaction(txr.TransactionHash);
    }
    catch (Exception e)
    {
        Log.Error("Getting eth tx failed: " + e.Message);
    }
    ...
    var interopAddress = ExtractInteropAddress(tx);
    ...
    if (tx.Value != null && tx.Value.Value > 0)
    ...
}

public static Address ExtractInteropAddress(Nethereum.RPC.Eth.DTOs.Transaction tx)
{
    //Using the transaction from RPC to build a txn for signing / signed
    var transaction = Nethereum.Signer.TransactionFactory.CreateTransaction(tx.To, tx.Gas, tx.GasPrice, tx.Value, tx.Inj
        tx.R, tx.S, tx.V);
    ...
}
```

Recommendation

Consider throwing an exception in `catch` statement to stop execution of the method or handling uninitialized variables in `finally` block.

Client Comment

This code will be removed entirely. It had been used in [TokenSwapper](#) which is removed as well and therefore all interop classes in `Phantasma.Node` will be removed.

Random Crash in node initialization

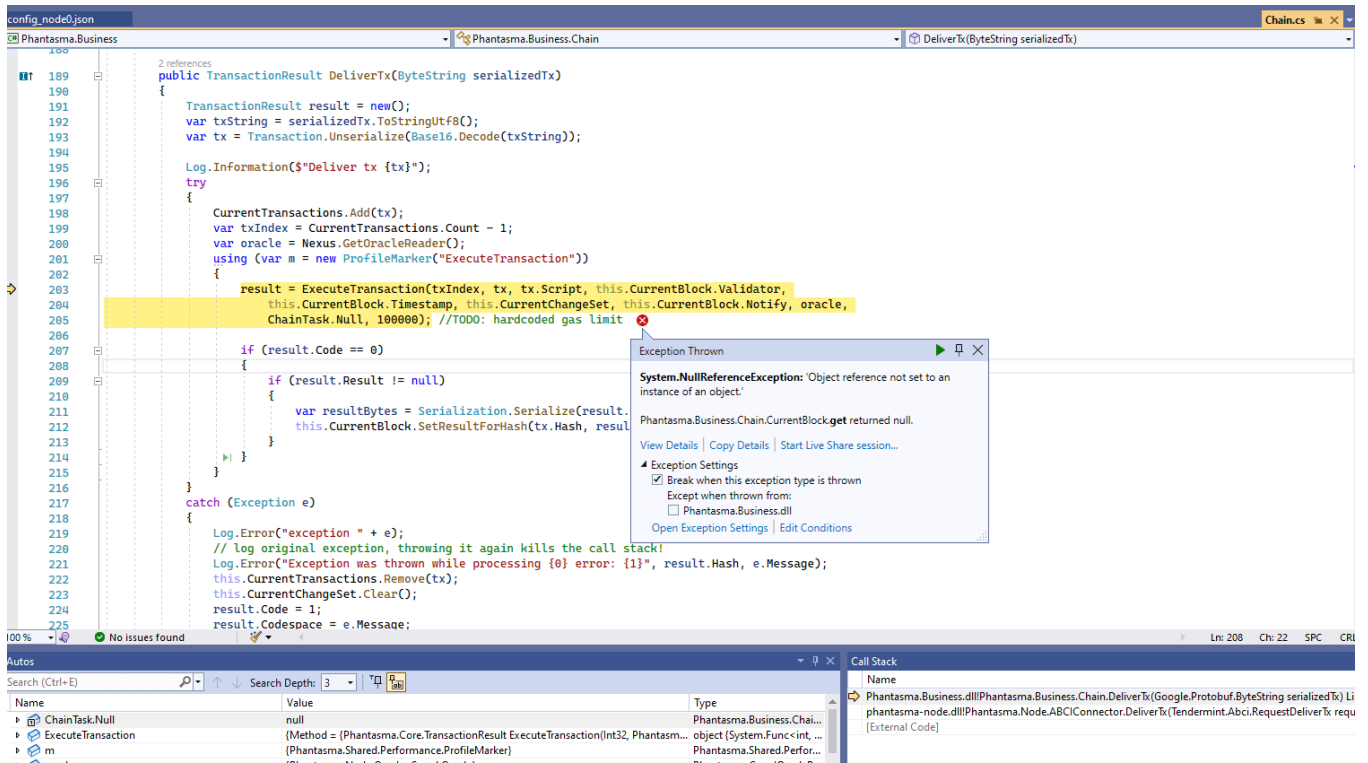
ID	PHA-414
Severity	LOW
Scope	Code Quality, VM
Status	Fixed
Reference	Phantasma-ng
Code	DeliverTx

Details

In some occasions when node was started there would be andom crash due to null pointer exception.

```
public TransactionResult DeliverTx(ByteString serializedTx)
{
    TransactionResult result = new();
    var txString = serializedTx.ToStringUtf8();
    var tx = Transaction.Unserialize(Base16.Decode(txString));
    Log.Information($"Deliver tx {tx}");
    try
    {
        CurrentTransactions.Add(tx);
        var txIndex = CurrentTransactions.Count - 1;
        var oracle = Nexus.GetOracleReader();
        using (var m = new ProfileMarker("ExecuteTransaction"))
        {
            result = ExecuteTransaction(txIndex, tx, tx.Script, this.CurrentBlock.Validator,
                this.CurrentBlock.Timestamp, this.CurrentChangeSet, this.CurrentBlock.Notify, oracle,
                ChainTask.Null, 100000); //TODO: hardcoded gas limit
            if (result.Code == 0)
            {
                if (result.Result != null)
                {
                    var resultBytes = Serialization.Serialize(result.Result);
                    this.CurrentBlock.SetResultForHash(tx.Hash, resultBytes);
                }
            }
        }
    }
    catch (Exception e)
    {
        Log.Error("exception " + e);
        // log original exception, throwing it again kills the call stack!
        Log.Error("Exception was thrown while processing {0} error: {1}", result.Hash, e.Message);
        this.CurrentTransactions.Remove(tx);
        this.CurrentChangeSet.Clear();
        result.Code = 1;
        result.Codespace = e.Message;
    }
    return result;
}
```

The crash was seen on Windows Environment only



Recommendation

Adding more unit tests and integration tests that cover all edge cases will minimize random unhandled crashes.

Result Of Call To Method Is Ignored

ID	PHA-415
Severity	LOW
Scope	Code Quality
Status	Fixed (6f6c441be66be559dfb91fb53b5f6ba870f0ebb9)
Reference	Phantasma-ng
Code	Nexus::LoadNexus()

Details

Possible `null` return on result of call to `GetChainByName` method. The `GetChainByName` method could load a chain to `_chainCache` property, but on the `LoadNexus` method this implementation could cause unexpected results.

```
public void LoadNexus(StorageContext storage)
{
    var chainList = this.GetChains(storage);
    foreach (var chainName in chainList)
    {
        GetChainByName(chainName);
    }
}
```

Recommendation

Check result of `GetChainByName` method.

Return In Loop

ID	PHA-416
Severity	LOW
Scope	Code Quality
Status	Fixed (bfeb381e29668e330af1d144231e947c1895ea4f)
Reference	Phantasma-ng

Details

Some loop statements had `return` statement.

Code: [NeoAPI::ParseStack](#)

```
protected static StackItem ParseStack(JsonElement stack)
{
    //var items = new List<StackItem>();
    var stackArray = stack.EnumerateArray();
    if (stackArray.Count() > 0)
    {
        foreach (var child in stackArray)
        {
            var item = ParseStackItems(child);
            return item;
            //items.Add(item);
        }
    }
    return null;
    //return items.ToArray();
}
```

Recommendation

Check all values in the collection and return the result or use `[0]` statement to remove unnecessary loop statement.

Unreachable Dispose Statement On Exception

ID	PHA-419
Severity	LOW
Scope	Code Quality
Status	Fixed (d34bd3482991acd756b6ecf616134d5a42135a93)
Reference	Phantasma-ng
Code	ProfileMaker.cs

Details

If any exception occurred during execution of the method, dispose was not called.

```
public void Stop()
{
    if (CurrentSession == this)
        CurrentSession = null;
    if (output == null)
        return;
    writer = new System.IO.StreamWriter(output);
    bool first = true;
    //convert to microseconds
    double scale = 1000000.0 / System.Diagnostics.Stopwatch.Frequency;
    BeginDocument();
    var stack = new List<int>();
    for (int i = 0, end = events.Count; i != end; ++i)
    {
        for (int j = stack.Count; j-- > 0;)
        {
            var p = events[stack[j]];
            if (p.parentIdx == i)
                break;
            EndEvent(p.name, p.end * scale);
        }
        var e = events[i];
        BeginEvent(e.name, e.start * scale, first);
        first = false;
        if (i + 1 < end && events[i + 1].parentIdx == i)
        {
            stack.Add(i);
        }
        else
        {
            EndEvent(e.name, e.end * scale);
        }
    }
    for (int j = stack.Count; j-- > 0;)
    {
        var p = events[stack[j]];
        EndEvent(p.name, p.end * scale);
    }
    EndDocument();
    writer.Flush();
    writer.Dispose(); // <--- Unreachable statement on exception
    output.Close();
    output.Dispose();
    output = null;
}
```

Recommendation

You can use `try-catch-finally` statement to `dispose` the writer.

Incomplete Smart Contracts SDK And Toolchain

ID	PHA-405
Severity	LOW
Scope	Smart Contracts Documentation, Smart Contracts SDK
Status	Being Improved
Reference	Phantasma-ng



Details

[\[Phantasma-SDK\]](#) [\[Phantasma-Chain\]](#) [\[Phantasma-UnitySDK\]](#)

There is a lack of toolchain and SDK for external Smart Contracts Development.

There was recently added a tool for Smart Contract testing: <https://teknopt.github.io/Phantasma-Contract-Tester>

Recommendation

Enhance toolchain/SDK for External Smart Contracts developers.

Enhance documentation with examples of different external contracts and step-by-step tutorials for different stages of development cycle (compilation, running testnet, deploying on testnet, calling the contract, checking state changes, updating, etc.)

Enhance VM specification documentation.

Fuzz Tests

Fuzz Tests have been executed against the following subsystems:

- Block Storage: <https://github.com/phantsma-io/phantsma-ng/tree/master/Phantasma.Core/src/Storage>

Block Storage Fuzz Tests

Unit tests were introduced first to prepare for fuzz entry points

Corpus: Bytes sequence permutations for input parameters (Key,Value)

Methods tested: Serialized, Unserialized, Put,Get,Has,Delete,Any,Count,Add,IndexOf,Contains,All,Remove,AddRaw,SetRaw,GetRaw

Store: MemoryStore

Number of fuzz executions: 4780591930

There were **no** issues or crashes detected.

Detailed logs can be found here: [fuzz/logs](#)

Disclaimers

Hacken disclaimer

The code base provided for audit has been analyzed according to the latest industry code quality, software processes and cybersecurity practices at the date of this report, with discovered security vulnerabilities and issues the details of which are disclosed in this report (Source Code); the Source Code compilation, deployment, and functionality (performing the intended functional specifications). The report contains no statements or warranties on the identification of all vulnerabilities and security of the code. The report covers the code (branch/tag/commit hash) submitted to and reviewed, so it may not be relevant to any other branch. Do not consider this report as a final and sufficient assessment regarding the utility and safety of the code, bug-free status, or any other contract statements. While we have done our best in conducting the analysis and producing this report, it is important to note that you should not rely on this report only — we recommend proceeding with several independent audits, public bug bounty program and CI/CD process to ensure security and code quality. English is the original language of the report. The Consultant is not responsible for the correctness of the translated versions.

Technical disclaimer

Protocol Level Systems are deployed and executed on hardware and software underlying platforms and platform dependencies (Operating System, System Libraries, Runtime Virtual Machines, linked libraries, etc.). The platform, programming languages, and other software related to the Protocol Level System may have vulnerabilities that can lead to security issues and exploits. Thus, Consultant cannot guarantee the explicit security of the Protocol system in full execution environment stack (hardware, OS, libraries, etc.)