LISBON | STARKNET CC

Thoth

Cairo/Starknet bytecode analyzer

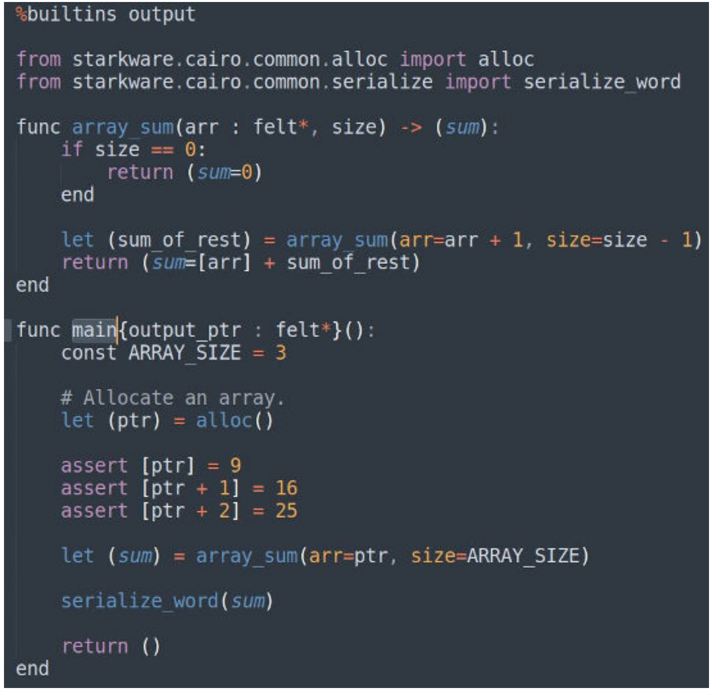
Patrick Ventuzelo (@Pat_Ventuzelo)

- Founder & CEO of **FuzzingLabs** | Senior **Security Researcher**
  - Fuzzing and vulnerability research
  - Development of security tools

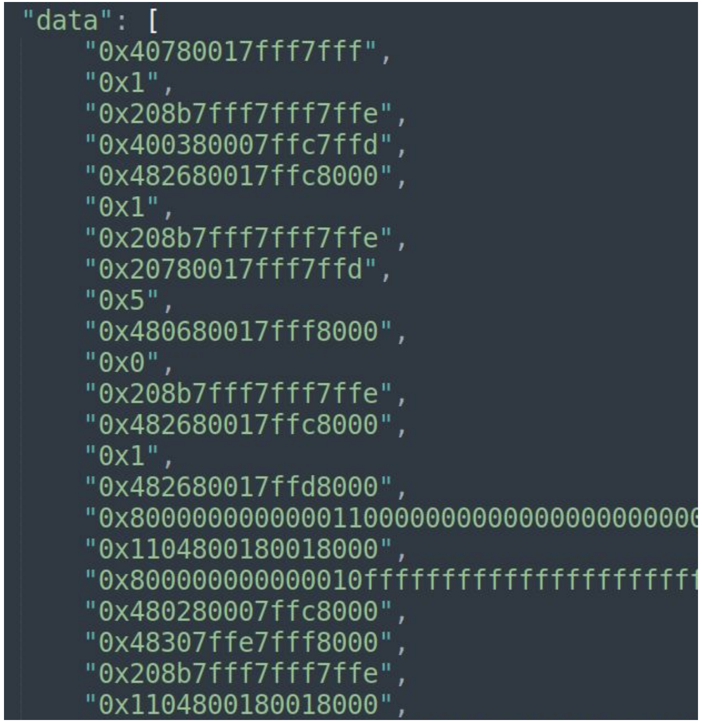
- Training/Online courses
  - **Rust** Security Audit & Fuzzing
  - **Go** Security Audit & Fuzzing
  - **WebAssembly** Reversing & Analysis
  - **Ethereum/Solidity** Security (WIP)
  - **Cairo** Security (WIP)

- Blockchain security since 2016
  - EthCC speaker (x3), Devcon speaker
  - Creator of **Octopus**
  - Public research about **EVM reversing & Tx analysis**
  - Lead developer of **Beaconfuzz**, eth2 differential fuzzer
  - **Fuzzing and audits** of dozen of L1/L2 implementations

Compilation - Cairo code into JSON artifact

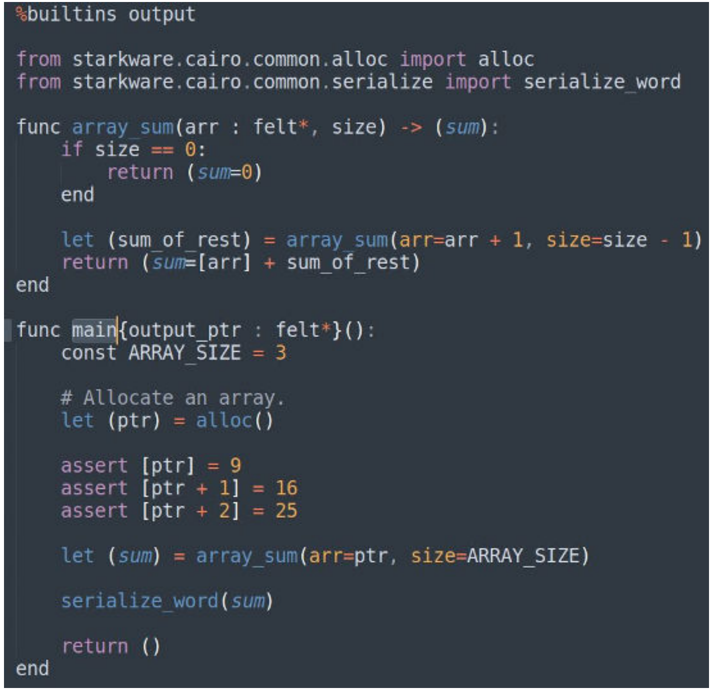


```
%builtins output

from starkware.cairo.common.alloc import alloc
from starkware.cairo.common.serialize import serialize_word

func array_sum(arr : felt*, size) -> (sum):
    if size == 0:
        return (sum=0)
    end

    let (sum_of_rest) = array_sum(arr=arr + 1, size=size - 1)
    return (sum=[arr] + sum_of_rest)
end

func main{output_ptr : felt*}():
    const ARRAY_SIZE = 3

    # Allocate an array.
    let (ptr) = alloc()

    assert [ptr] = 9
    assert [ptr + 1] = 16
    assert [ptr + 2] = 25

    let (sum) = array_sum(arr=ptr, size=ARRAY_SIZE)

    serialize_word(sum)

    return ()
end
```

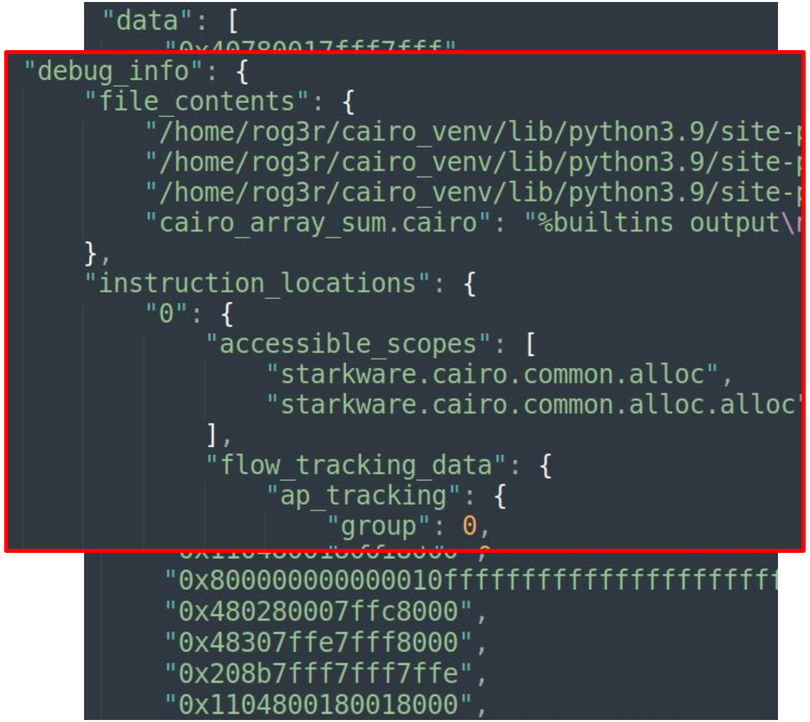


```
"data": [
    "0x40780017fff7fff",
    "0x1",
    "0x208b7fff7fff7ffe",
    "0x400380007ffc7ffd",
    "0x482680017ffc8000",
    "0x1",
    "0x208b7fff7fff7ffe",
    "0x20780017fff7ffd",
    "0x5",
    "0x480680017fff8000",
    "0x0",
    "0x208b7fff7fff7ffe",
    "0x482680017ffc8000",
    "0x1",
    "0x482680017ffd8000",
    "0x8000000000000110000000000000000000000
    "0x1104800180018000",
    "0x800000000000010ffffffffffffffffffffff
    "0x480280007ffc8000",
    "0x48307ffe7fff8000",
    "0x208b7fff7fff7ffe",
    "0x1104800180018000",
```

Compilation - Cairo code into JSON artifact

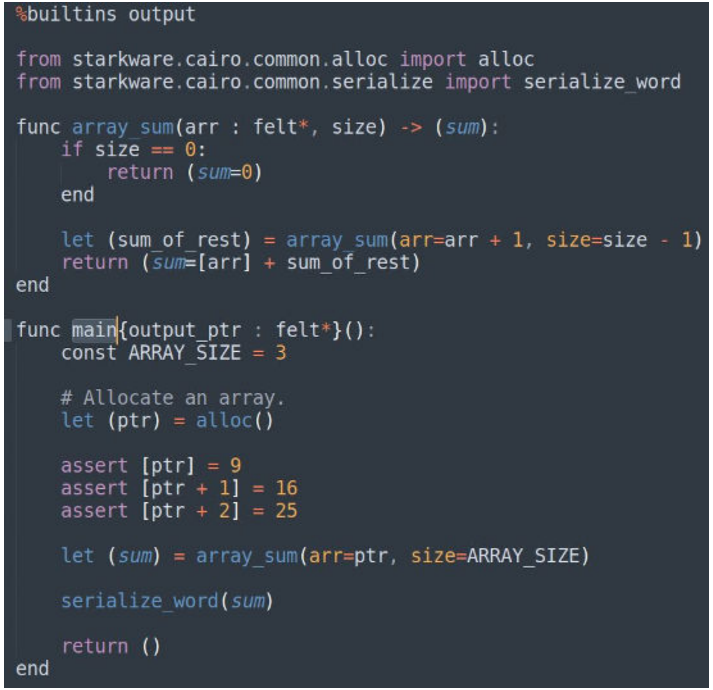


```
%builtins output

from starkware.cairo.common.alloc import alloc
from starkware.cairo.common.serialize import serialize_word

func array_sum(arr : felt*, size) -> (sum):
    if size == 0:
        return (sum=0)
    end

    let (sum_of_rest) = array_sum(arr=arr + 1, size=size - 1)
    return (sum=[arr] + sum_of_rest)
end

func main{output_ptr : felt*}():
    const ARRAY_SIZE = 3

    # Allocate an array.
    let (ptr) = alloc()

    assert [ptr] = 9
    assert [ptr + 1] = 16
    assert [ptr + 2] = 25

    let (sum) = array_sum(arr=ptr, size=ARRAY_SIZE)

    serialize_word(sum)

    return ()
end
```

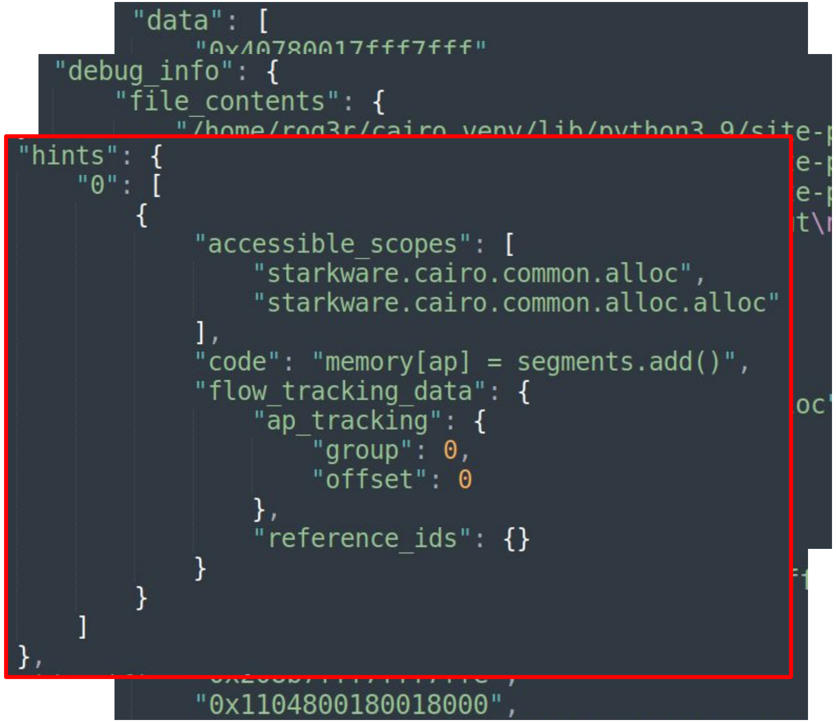


```
"data": [
    "0x40780017fff7fff"
"debug_info": {
    "file_contents": {
        "/home/rog3r/cairo_venv/lib/python3.9/site-
        "/home/rog3r/cairo_venv/lib/python3.9/site-
        "/home/rog3r/cairo_venv/lib/python3.9/site-
        "cairo_array_sum.cairo": "%builtins output\
    },
    "instruction_locations": {
        "0": {
            "accessible_scopes": [
                "starkware.cairo.common.alloc",
                "starkware.cairo.common.alloc.alloc"
            ],
            "flow_tracking_data": {
                "ap_tracking": {
                    "group": 0,
    "0x800000000000010fffffffffffffffff
    "0x480280007ffc8000",
    "0x48307ffe7fff8000",
    "0x208b7fff7fff7ffe",
    "0x1104800180018000",
```

Compilation - Cairo code into JSON artifact

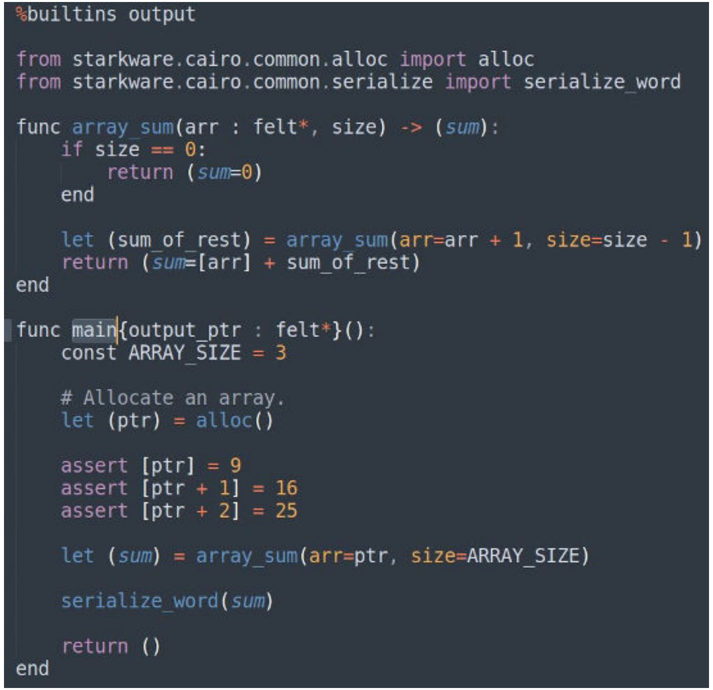


```
%builtins output

from starkware.cairo.common.alloc import alloc
from starkware.cairo.common.serialize import serialize_word

func array_sum(arr : felt*, size) -> (sum):
    if size == 0:
        return (sum=0)
    end

    let (sum_of_rest) = array_sum(arr=arr + 1, size=size - 1)
    return (sum=[arr] + sum_of_rest)
end

func main{output_ptr : felt*}():
    const ARRAY_SIZE = 3

    # Allocate an array.
    let (ptr) = alloc()

    assert [ptr] = 9
    assert [ptr + 1] = 16
    assert [ptr + 2] = 25

    let (sum) = array_sum(arr=ptr, size=ARRAY_SIZE)

    serialize_word(sum)

    return ()
end
```

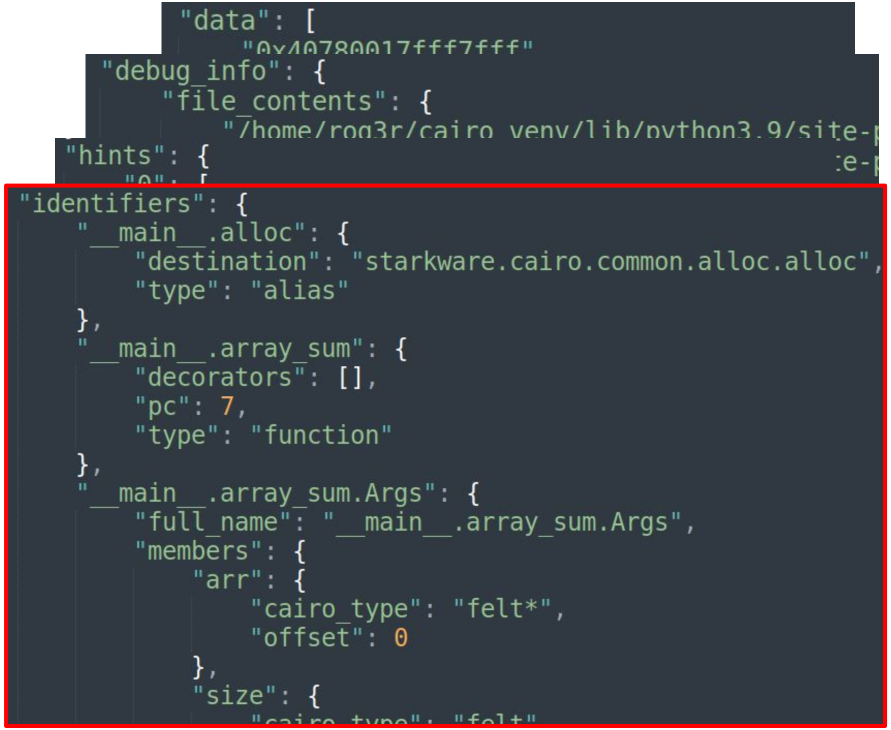


```
"data": [
    "0x40780017fff7fff"
"debug_info": {
    "file_contents": {
        "/home/rog3r/cairo_venv/lib/python3.9/site-p
"hints": {
    "0": [
        {
            "accessible_scopes": [
                "starkware.cairo.common.alloc",
                "starkware.cairo.common.alloc.alloc"
            ],
            "code": "memory[ap] = segments.add()",
            "flow_tracking_data": {
                "ap_tracking": {
                    "group": 0,
                    "offset": 0
                },
                "reference_ids": {}
            }
        }
    ]
},
        "0x1104800180018000",
```

Compilation - Cairo code into JSON artifact



```
%builtins output

from starkware.cairo.common.alloc import alloc
from starkware.cairo.common.serialize import serialize_word

func array_sum(arr : felt*, size) -> (sum):
    if size == 0:
        return (sum=0)
    end

    let (sum_of_rest) = array_sum(arr=arr + 1, size=size - 1)
    return (sum=[arr] + sum_of_rest)
end

func main{output_ptr : felt*}():
    const ARRAY_SIZE = 3

    # Allocate an array.
    let (ptr) = alloc()

    assert [ptr] = 9
    assert [ptr + 1] = 16
    assert [ptr + 2] = 25

    let (sum) = array_sum(arr=ptr, size=ARRAY_SIZE)

    serialize_word(sum)

    return ()
end
```



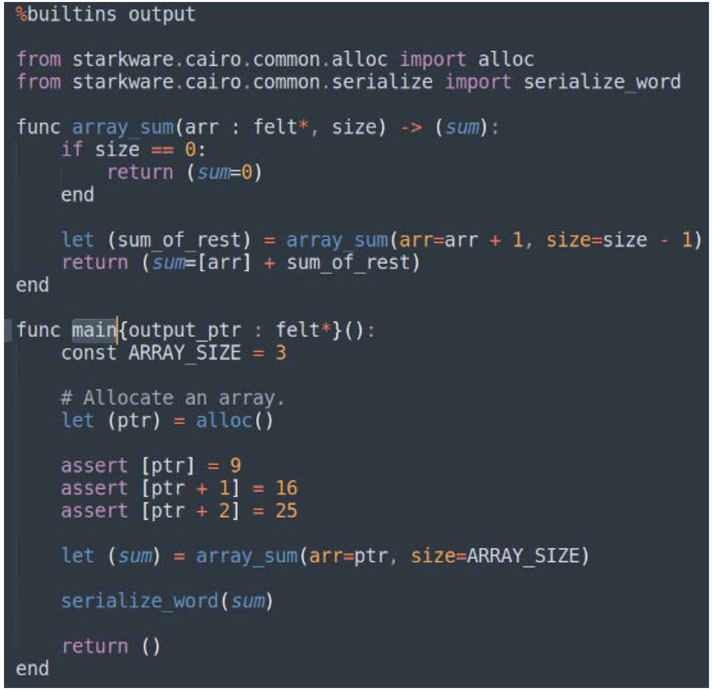
```
"data": [
    "0x40780017fff7fff"
```

```
"debug_info": {
    "file_contents": {
        "/home/rog3r/cairo_venv/lib/python3.9/site-p
                                                :e-p
```

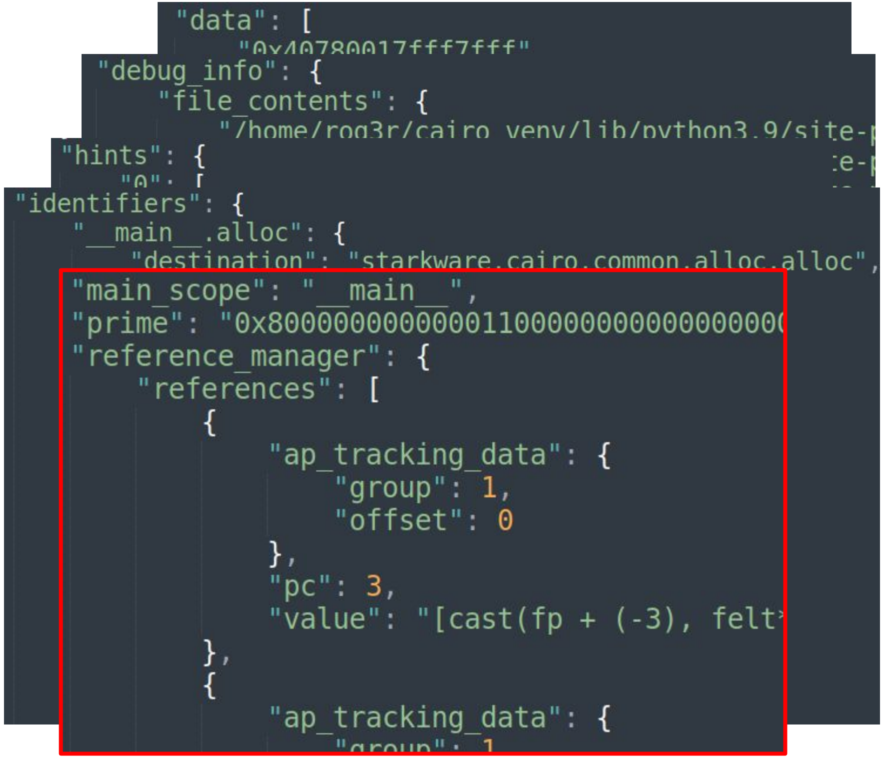
```
"hints": {
    "0": [
```

```
"identifiers": {
    "__main__.alloc": {
        "destination": "starkware.cairo.common.alloc.alloc",
        "type": "alias"
    },
    "__main__.array_sum": {
        "decorators": [],
        "pc": 7,
        "type": "function"
    },
    "__main__.array_sum.Args": {
        "full_name": "__main__.array_sum.Args",
        "members": {
            "arr": {
                "cairo_type": "felt*",
                "offset": 0
            },
            "size": {
                "cairo_type": "felt"
```

Compilation - Cairo code into JSON artifact



```
%builtins output

from starkware.cairo.common.alloc import alloc
from starkware.cairo.common.serialize import serialize_word

func array_sum(arr : felt*, size) -> (sum):
    if size == 0:
        return (sum=0)
    end

    let (sum_of_rest) = array_sum(arr=arr + 1, size=size - 1)
    return (sum=[arr] + sum_of_rest)
end

func main{output_ptr : felt*}():
    const ARRAY_SIZE = 3

    # Allocate an array.
    let (ptr) = alloc()

    assert [ptr] = 9
    assert [ptr + 1] = 16
    assert [ptr + 2] = 25

    let (sum) = array_sum(arr=ptr, size=ARRAY_SIZE)

    serialize_word(sum)

    return ()
end
```



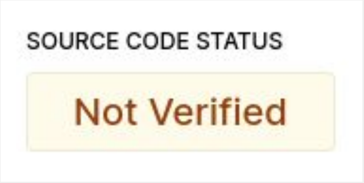
```
"data": [
    "0x40780017fff7fff"
"debug_info": {
    "file_contents": {
        "/home/rog3r/cairo_venv/lib/python3.9/site-p
                                                      :e-p
"hints": {
    "0": [
"identifiers": {
    "__main__.alloc": {
        "destination": "starkware.cairo.common.alloc.alloc",
"main_scope": "__main__",
"prime": "0x80000000000001100000000000000000(
"reference_manager": {
    "references": [
        {
            "ap_tracking_data": {
                "group": 1,
                "offset": 0
            },
            "pc": 3,
            "value": "[cast(fp + (-3), felt*
        },
        {
            "ap_tracking_data": {
                "group": 1
```

© 2022 FuzzingLabs - Thoth - StarkNetCC 2022 Lisbon

Why Thoth has been created?

- Problematic
  - Most contracts on the mainnet/testnet are **not verified**
  - **Only** the full **JSON artifact** is mandatory and stored online

- Goal
  - Analysis of closed source contract for due diligence
  - Help developers to understand compiler operations
  - Thoth is intended to be a complete tool

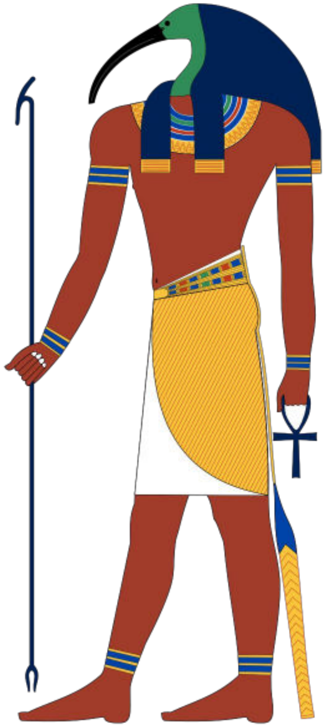


SOURCE CODE STATUS

Not Verified

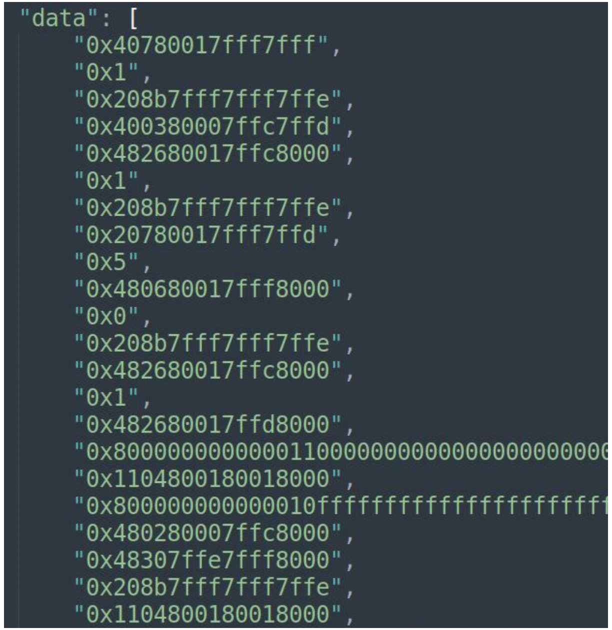
- Name: **Thoth**
  - God of the **moon**, **sacred texts**, mathematics, sciences, magic, messenger and recorder of the deities, **master of knowledge**, and patron of scribes.
  - Pronounced **toss** or tot
    - I know the naming pronunciation s*cks a bit…
  - Inspired by other amazing tools
    - Octopus, Slither, Mythril, etc.

- **Only the bytecode is the Truth**

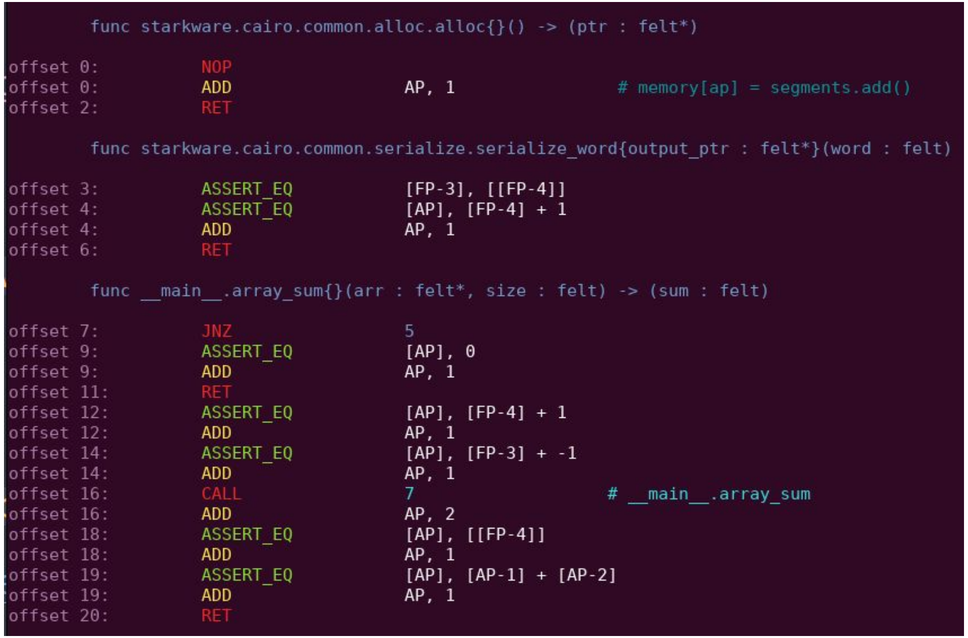


Disassembler

Disassembler



```
"data": [
    "0x40780017fff7fff",
    "0x1",
    "0x208b7fff7fff7ffe",
    "0x400380007ffc7ffd",
    "0x482680017ffc8000",
    "0x1",
    "0x208b7fff7fff7ffe",
    "0x20780017fff7ffd",
    "0x5",
    "0x480680017fff8000",
    "0x0",
    "0x208b7fff7fff7ffe",
    "0x482680017ffc8000",
    "0x1",
    "0x482680017ffd8000",
    "0x800000000000011000000000000000000000
    "0x1104800180018000",
    "0x800000000000010ffffffffffffffffffffff
    "0x480280007ffc8000",
    "0x48307ffe7fff8000",
    "0x208b7fff7fff7ffe",
    "0x1104800180018000",
```



```
        func starkware.cairo.common.alloc.alloc{}() -> (ptr : felt*)

offset 0:          NOP
offset 0:          ADD                AP, 1                # memory[ap] = segments.add()
offset 2:          RET

        func starkware.cairo.common.serialize.serialize_word{output_ptr : felt*}(word : felt)

offset 3:          ASSERT_EQ          [FP-3], [[FP-4]]
offset 4:          ASSERT_EQ          [AP], [FP-4] + 1
offset 4:          ADD                AP, 1
offset 6:          RET

        func __main__.array_sum{}(arr : felt*, size : felt) -> (sum : felt)

offset 7:          JNZ                5
offset 9:          ASSERT_EQ          [AP], 0
offset 9:          ADD                AP, 1
offset 11:         RET
offset 12:         ASSERT_EQ          [AP], [FP-4] + 1
offset 12:         ADD                AP, 1
offset 14:         ASSERT_EQ          [AP], [FP-3] + -1
offset 14:         ADD                AP, 1
offset 16:         CALL               7                   # __main__.array_sum
offset 16:         ADD                AP, 2
offset 18:         ASSERT_EQ          [AP], [[FP-4]]
offset 18:         ADD                AP, 1
offset 19:         ASSERT_EQ          [AP], [AP-1] + [AP-2]
offset 19:         ADD                AP, 1
offset 20:         RET
```

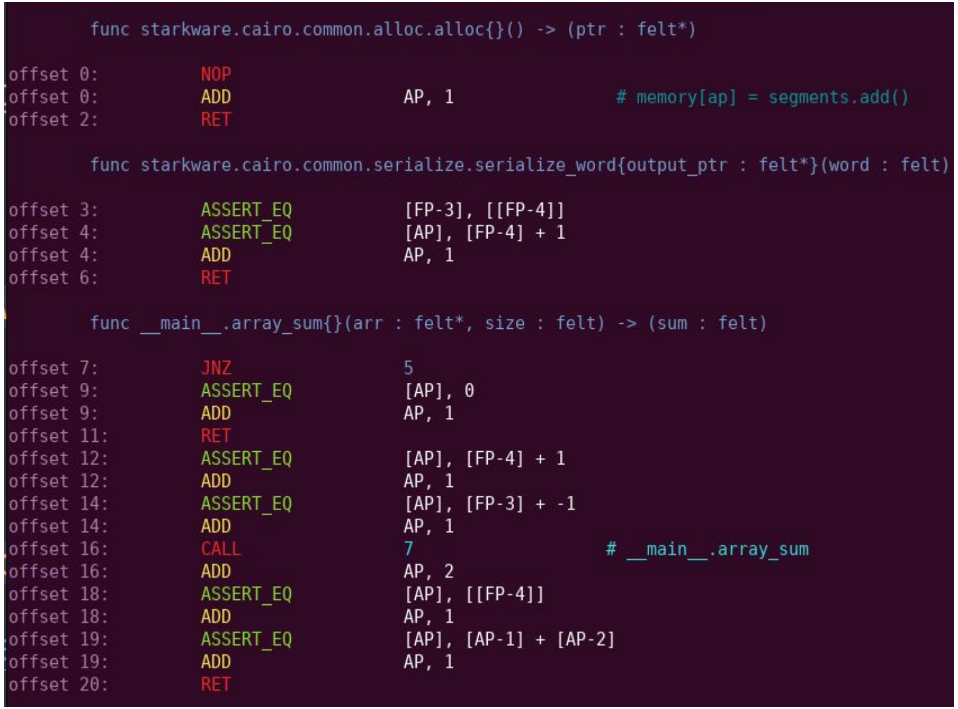
Disassembler

- “***Visual representation** of the bytecode as a linear **sequence of instructions**.*”

- Several data are in the **JSON**.

- **Interesting information**
  - Builtins
  - Structures
  - Events
  - Constants representation
  - Functions ID and names matching

- Example
  - `thoth local cairo_array_sum.json` **`-b`**



```
        func starkware.cairo.common.alloc.alloc{}() -> (ptr : felt*)

offset 0:          NOP
offset 0:          ADD                 AP, 1                # memory[ap] = segments.add()
offset 2:          RET

        func starkware.cairo.common.serialize.serialize_word{output_ptr : felt*}(word : felt)

offset 3:          ASSERT_EQ           [FP-3], [[FP-4]]
offset 4:          ASSERT_EQ           [AP], [FP-4] + 1
offset 4:          ADD                 AP, 1
offset 6:          RET

        func __main__.array_sum{}(arr : felt*, size : felt) -> (sum : felt)

offset 7:          JNZ                 5
offset 9:          ASSERT_EQ           [AP], 0
offset 9:          ADD                 AP, 1
offset 11:         RET
offset 12:         ASSERT_EQ           [AP], [FP-4] + 1
offset 12:         ADD                 AP, 1
offset 14:         ASSERT_EQ           [AP], [FP-3] + -1
offset 14:         ADD                 AP, 1
offset 16:         CALL                7                   # __main__.array_sum
offset 16:         ADD                 AP, 2
offset 18:         ASSERT_EQ           [AP], [[FP-4]]
offset 18:         ADD                 AP, 1
offset 19:         ASSERT_EQ           [AP], [AP-1] + [AP-2]
offset 19:         ADD                 AP, 1
offset 20:         RET
```

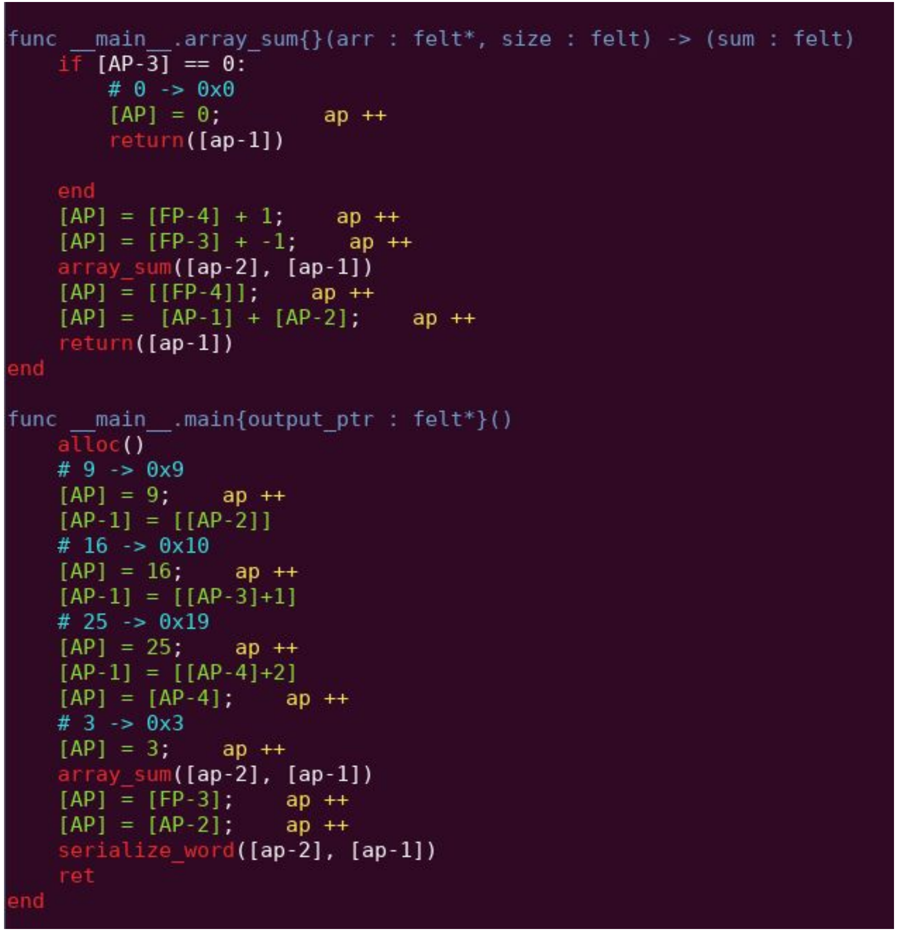
Decompiler version 0.1.0

Decompiler

- *"A decompiler is a computer program that **takes bytecode** as input, and **attempts to create a high-level source file** that (ideally) can be successfully compiled."*

- Features
  - Recovery of parameters from function calls.
  - Generation of imports.

- The first version of the decompiler
  - Similar to the disassembly output.
  - AP/FP is **complicated** to understand **for beginners.**
  - Limited support of **if/else** blocks.



```
func __main__.array_sum{}(arr : felt*, size : felt) -> (sum : felt)
    if [AP-3] == 0:
        # 0 -> 0x0
        [AP] = 0;          ap ++
        return([ap-1])

    end
    [AP] = [FP-4] + 1;     ap ++
    [AP] = [FP-3] + -1;     ap ++
    array_sum([ap-2], [ap-1])
    [AP] = [[FP-4]];     ap ++
    [AP] =  [AP-1] + [AP-2];     ap ++
    return([ap-1])
end

func __main__.main{output_ptr : felt*}()
    alloc()
    # 9 -> 0x9
    [AP] = 9;     ap ++
    [AP-1] = [[AP-2]]
    # 16 -> 0x10
    [AP] = 16;     ap ++
    [AP-1] = [[AP-3]+1]
    # 25 -> 0x19
    [AP] = 25;     ap ++
    [AP-1] = [[AP-4]+2]
    [AP] = [AP-4];     ap ++
    # 3 -> 0x3
    [AP] = 3;     ap ++
    array_sum([ap-2], [ap-1])
    [AP] = [FP-3];     ap ++
    [AP] = [AP-2];     ap ++
    serialize_word([ap-2], [ap-1])
    ret
end
```

Call Graph

Call Graph

- *"Call graph represents calling **relationships between subroutines** in a computer program."*

- **Node** represents a function.
- **Edge(a, b)** indicates that function **a** calls function **b**.

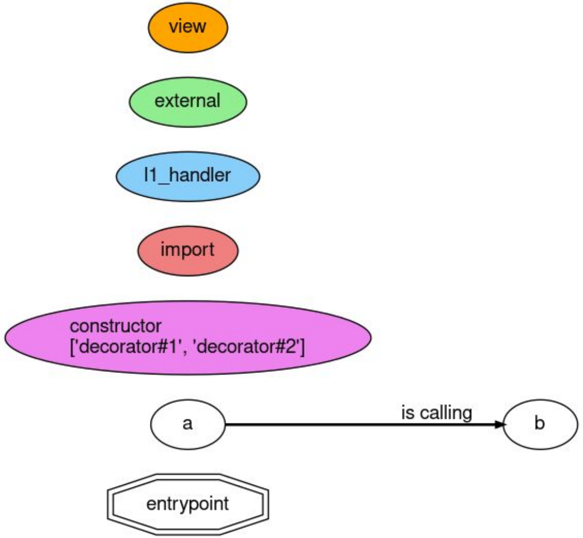
- Legend:
    - Colors for important functions (import, constructor, etc.)
    - Octagonal shape for **entry-point**.

You Retweeted

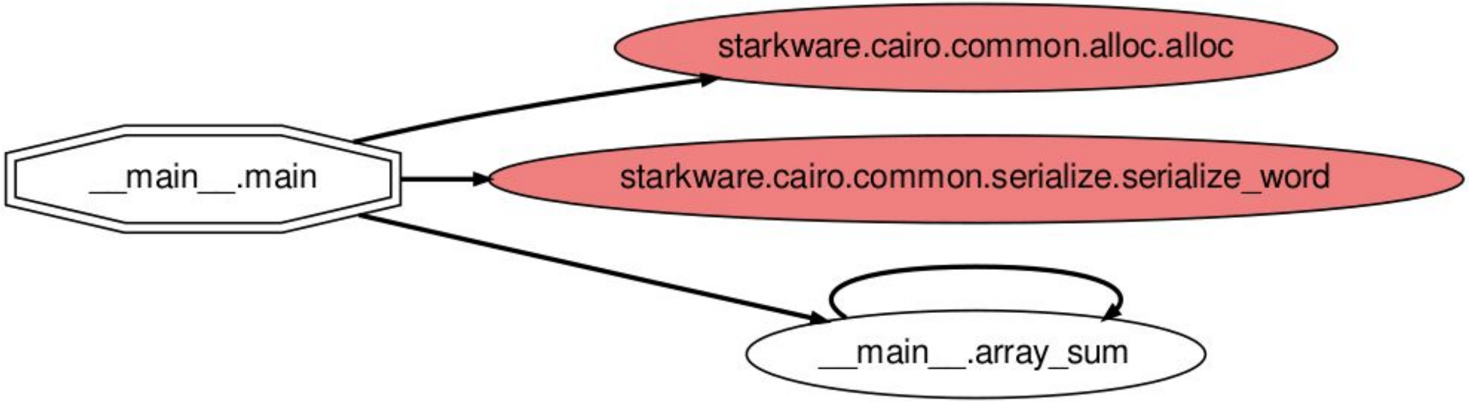
Carbonable.io @Carbonable_io · Oct 19

/3 To schematize the architecture of our contracts with their dependencies, we also used Thoth to generate the callgraphs of our functions proposed by @FuzzingLabs :

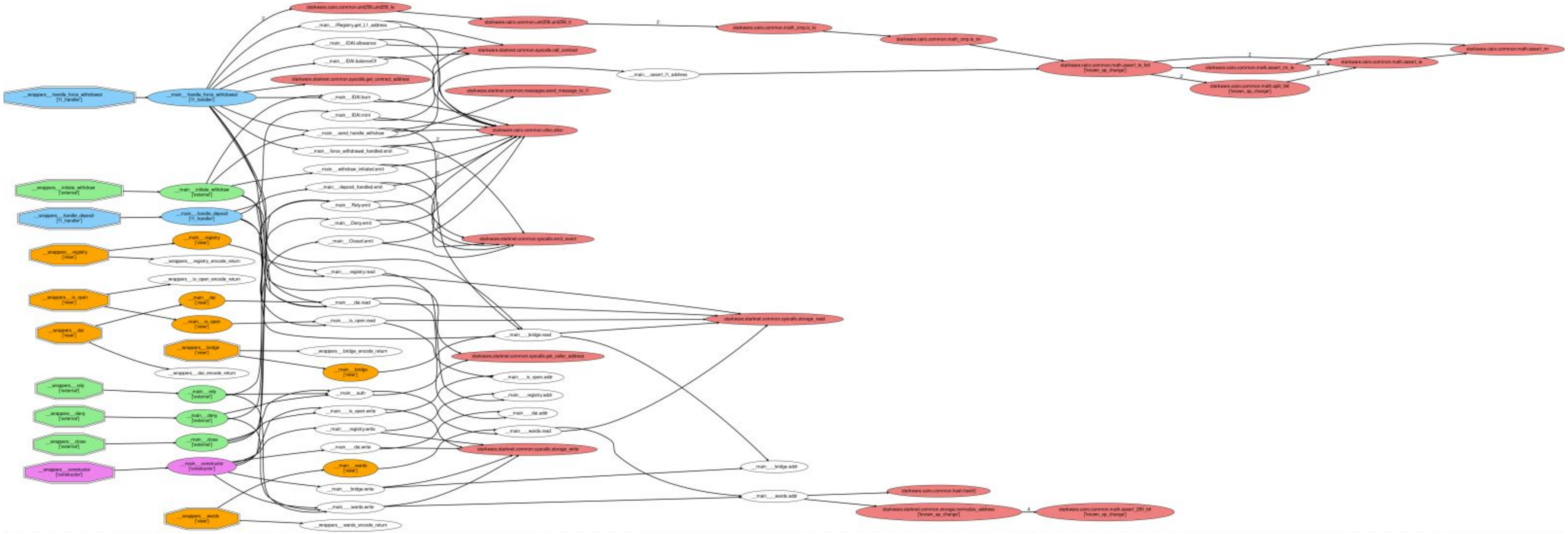
- Example
    - `thoth local cairo_array_sum.json -call -view=True`



Call Graph - Simple example (array_sum)


__main__.main
starkware.cairo.common.alloc.alloc
starkware.cairo.common.serialize.serialize_word
__main__.array_sum

Call Graph - Advanced example (dai bridge)


__wrappers__.handle_force_withdrawal ['l1_handler']
__main__.handle_force_withdrawal ['l1_handler']
__wrappers__.initiate_withdraw ['external']
__main__.initiate_withdraw ['external']
__wrappers__.handle_deposit ['l1_handler']
__main__.handle_deposit ['l1_handler']
__wrappers__.registry ['view']
__main__.registry ['view']
__wrappers__.registry_encode_return
__wrappers__.is_open_encode_return
__wrappers__.is_open ['view']
__main__.dai ['view']
__main__.is_open ['view']
__wrappers__.dai ['view']
__wrappers__.bridge ['view']
__wrappers__.dai_encode_return
__wrappers__.rely ['external']
__main__.rely ['external']
__wrappers__.deny ['external']
__main__.deny ['external']
__wrappers__.close ['external']
__main__.close ['external']
__wrappers__.constructor ['constructor']
__main__.constructor ['constructor']
__wrappers__.wards ['view']
starkware.cairo.common.uint256.uint256_le
__main__.IRegistry.get_L1_address
__main__.IDAI.allowance
__main__.IDAI.balanceOf
starkware.starknet.common.syscalls.get_contract_address
__main__.IDAI.burn
__main__.IDAI.mint
__main__.send_handle_withdraw
__main__.force_withdrawal_handled.emit
__main__.withdraw_initiated.emit
__main__.deposit_handled.emit
__main__.Rely.emit
__main__.Deny.emit
__main__.Closed.emit
__main__.registry.read
__main__.dai.read
__main__.is_open.read
__wrappers__.bridge_encode_return
__main__.bridge ['view']
__main__.auth
__main__.is_open.write
__main__.registry.write
__main__.dai.write
__main__.wards ['view']
__main__.bridge.write
__main__.wards.write
__wrappers__.wards_encode_return
starkware.cairo.common.uint256.uint256_lt
starkware.starknet.common.syscalls.call_contract
starkware.starknet.common.messages.send_message_to_l1
starkware.cairo.common.alloc.alloc
starkware.starknet.common.syscalls.emit_event
__main__.bridge.read
starkware.starknet.common.syscalls.get_caller_address
__main__.is_open.addr
__main__.registry.addr
__main__.dai.addr
__main__.wards.read
starkware.starknet.common.syscalls.storage_write
__main__.assert_l1_address
starkware.cairo.common.math_cmp.is_le
starkware.cairo.common.math_cmp.is_nn
starkware.cairo.common.math.assert_le_felt ['known_ap_change']
starkware.cairo.common.math.assert_nn_le
starkware.cairo.common.math.split_felt ['known_ap_change']
starkware.cairo.common.math.assert_le
starkware.cairo.common.math.assert_nn
starkware.starknet.common.syscalls.storage_read
__main__.bridge.addr
__main__.wards.addr
starkware.cairo.common.hash.hash2
starkware.starknet.common.storage.normalize_address ['known_ap_change']
starkware.cairo.common.math.assert_250_bit ['known_ap_change']
2
4

Control Flow Graph (CFG)

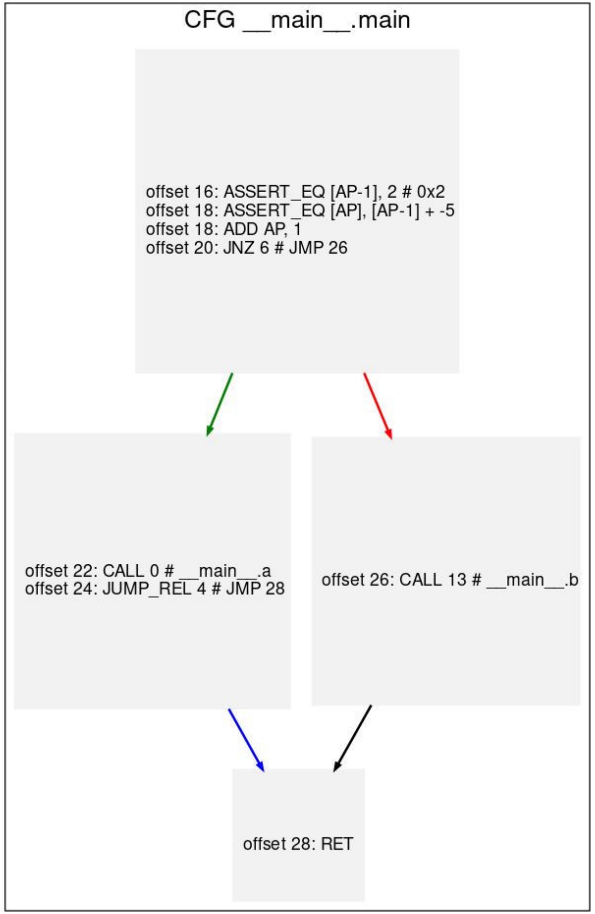
Control Flow Graph (CFG)

- *“Control-flow graph (CFG) is a **representation**, using graph notation, **of all paths that might be traversed** through a program **during its execution**.”*

- Representation
  - Basic block
    - Each node represents a **basic block.**
    - **Straight-line piece of code** without any jumps or jump offsets.
    - Jump offsets **start a block** and jump opcodes **end a block**.
  - Edges
    - Conditional True/False jump, Direct jump, Fallthrough.

- Usage
  - Useless for most Cairo developers.
  - Interesting but situational for auditors.
  - **Critical** for decompiler and analysis tools to get better results.

- Example:
  - `thoth local cairo_array_sum.json `**`-cfg`**` -view=True`



CFG __main__.main

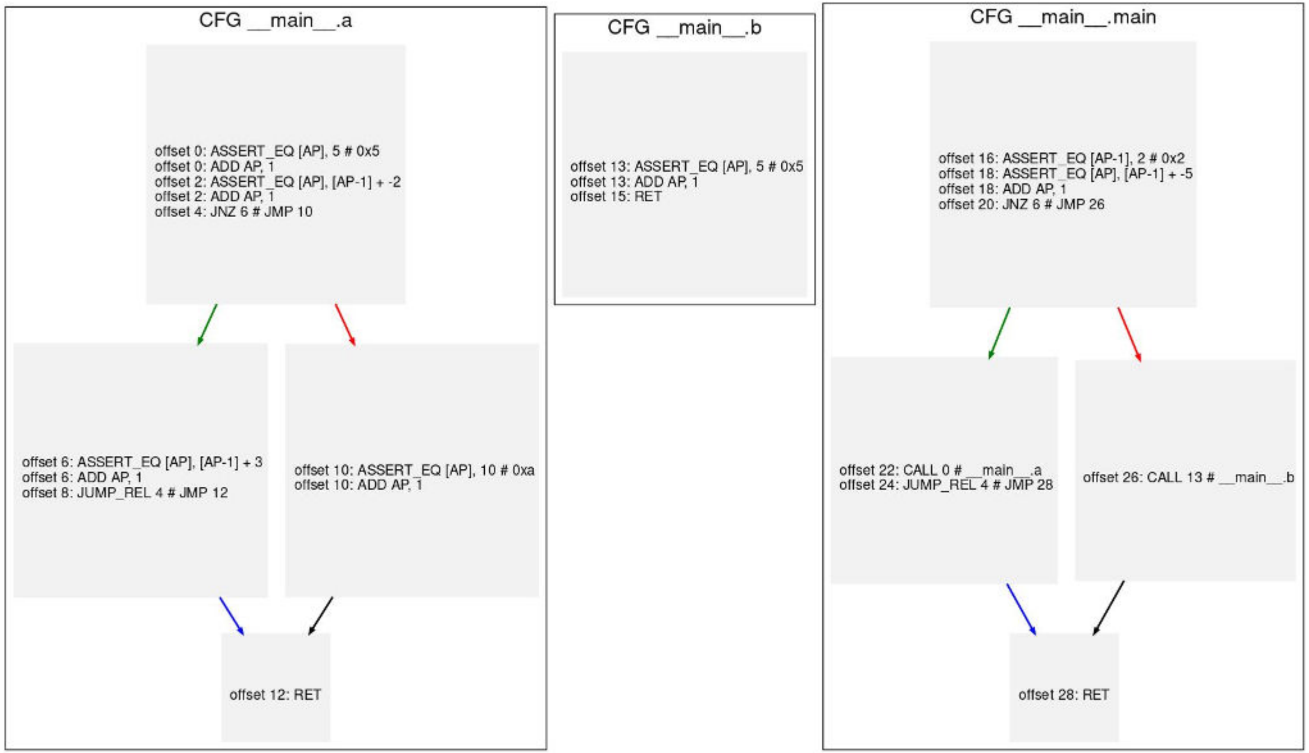
```
offset 16: ASSERT_EQ [AP-1], 2 # 0x2
offset 18: ASSERT_EQ [AP], [AP-1] + -5
offset 18: ADD AP, 1
offset 20: JNZ 6 # JMP 26
```

```
offset 22: CALL 0 # __main__.a
offset 24: JUMP_REL 4 # JMP 28
```

```
offset 26: CALL 13 # __main__.b
```

```
offset 28: RET
```

Control Flow Graph (CFG)



CFG __main__.a

```
offset 0: ASSERT_EQ [AP], 5 # 0x5
offset 0: ADD AP, 1
offset 2: ASSERT_EQ [AP], [AP-1] + -2
offset 2: ADD AP, 1
offset 4: JNZ 6 # JMP 10
```

```
offset 6: ASSERT_EQ [AP], [AP-1] + 3
offset 6: ADD AP, 1
offset 8: JUMP_REL 4 # JMP 12
```

```
offset 10: ASSERT_EQ [AP], 10 # 0xa
offset 10: ADD AP, 1
```

```
offset 12: RET
```

CFG __main__.b

```
offset 13: ASSERT_EQ [AP], 5 # 0x5
offset 13: ADD AP, 1
offset 15: RET
```

CFG __main__.main

```
offset 16: ASSERT_EQ [AP-1], 2 # 0x2
offset 18: ASSERT_EQ [AP], [AP-1] + -5
offset 18: ADD AP, 1
offset 20: JNZ 6 # JMP 26
```

```
offset 22: CALL 0 # __main__.a
offset 24: JUMP_REL 4 # JMP 28
```

```
offset 26: CALL 13 # __main__.b
```

```
offset 28: RET
```

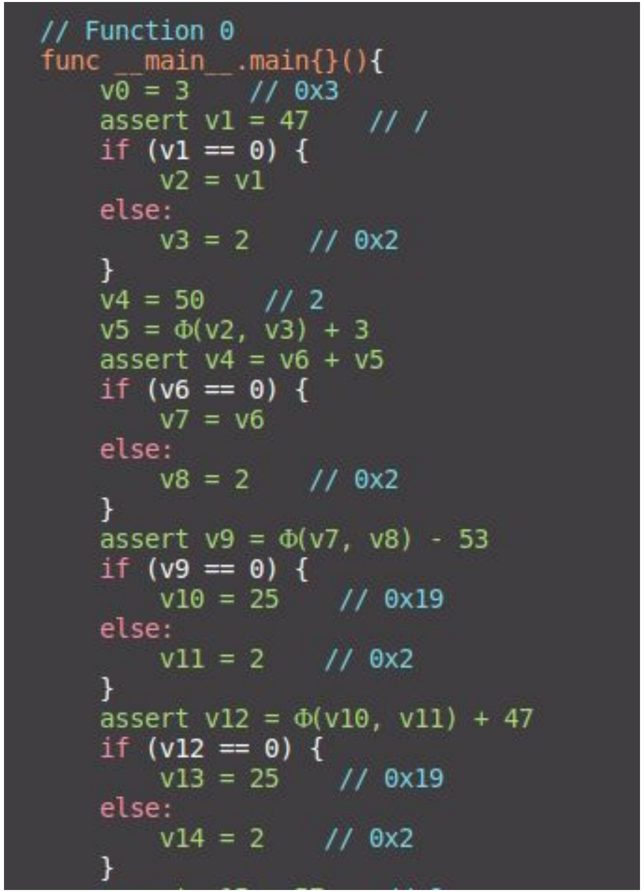
Decompiler version 0.3.0

Decompiler version 0.3.0

- Major decompilation improvement
  - By leveraging on the **CFG**.
  - Introduction of **Single Static Assignment (SSA)**.
  - Creation of a **virtual stack of variable** per basic block.

- Single Static Assignment (SSA)
  - *“**Static single assignment** form (abbreviated **SSA** form/SSA) is a property of an intermediate representation (IR), which requires that **each variable is assigned exactly once**, and every variable is defined before it is used.”*
  - Each variable is assigned once.
  - Each variable is defined before being used.
  - **phi node (Φ)** represents multiple potential value for a same variable chosen depending on the predecessor of the current block.

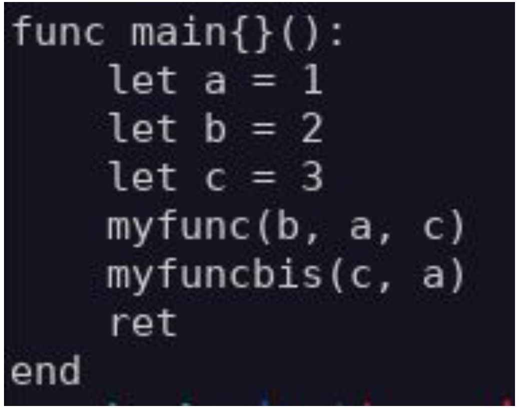
- Example
  - `thoth local cairo_nested_if_phi_node.json -d --color`



```
// Function 0
func __main__.main{}(){
    v0 = 3    // 0x3
    assert v1 = 47    // /
    if (v1 == 0) {
        v2 = v1
    else:
        v3 = 2    // 0x2
    }
    v4 = 50    // 2
    v5 = Φ(v2, v3) + 3
    assert v4 = v6 + v5
    if (v6 == 0) {
        v7 = v6
    else:
        v8 = 2    // 0x2
    }
    assert v9 = Φ(v7, v8) - 53
    if (v9 == 0) {
        v10 = 25    // 0x19
    else:
        v11 = 2    // 0x2
    }
    assert v12 = Φ(v10, v11) + 47
    if (v12 == 0) {
        v13 = 25    // 0x19
    else:
        v14 = 2    // 0x2
    }
```

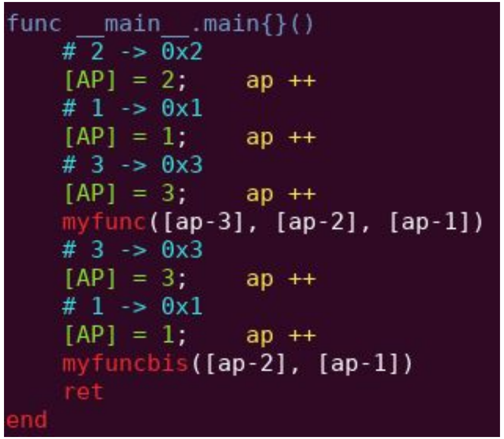
Decompiler evolution

- Original Source code



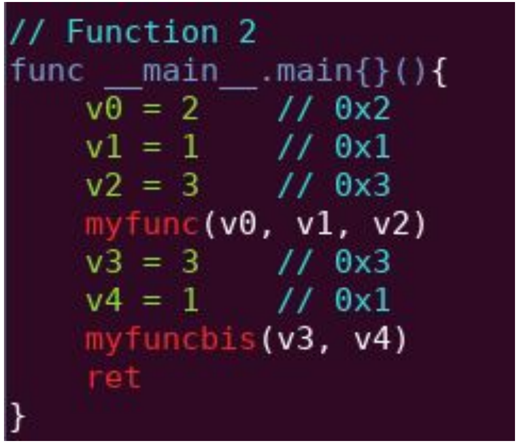
```
func main{}():
    let a = 1
    let b = 2
    let c = 3
    myfunc(b, a, c)
    myfuncbis(c, a)
    ret
end
```

- Thoth version 0.1.0



```
func __main__.main{}()
    # 2 -> 0x2
    [AP] = 2;    ap ++
    # 1 -> 0x1
    [AP] = 1;    ap ++
    # 3 -> 0x3
    [AP] = 3;    ap ++
    myfunc([ap-3], [ap-2], [ap-1])
    # 3 -> 0x3
    [AP] = 3;    ap ++
    # 1 -> 0x1
    [AP] = 1;    ap ++
    myfuncbis([ap-2], [ap-1])
    ret
end
```

- Thoth version 0.3.0



```
// Function 2
func __main__.main{}(){
    v0 = 2    // 0x2
    v1 = 1    // 0x1
    v2 = 3    // 0x3
    myfunc(v0, v1, v2)
    v3 = 3    // 0x3
    v4 = 1    // 0x1
    myfuncbis(v3, v4)
    ret
}
```

Analyzer

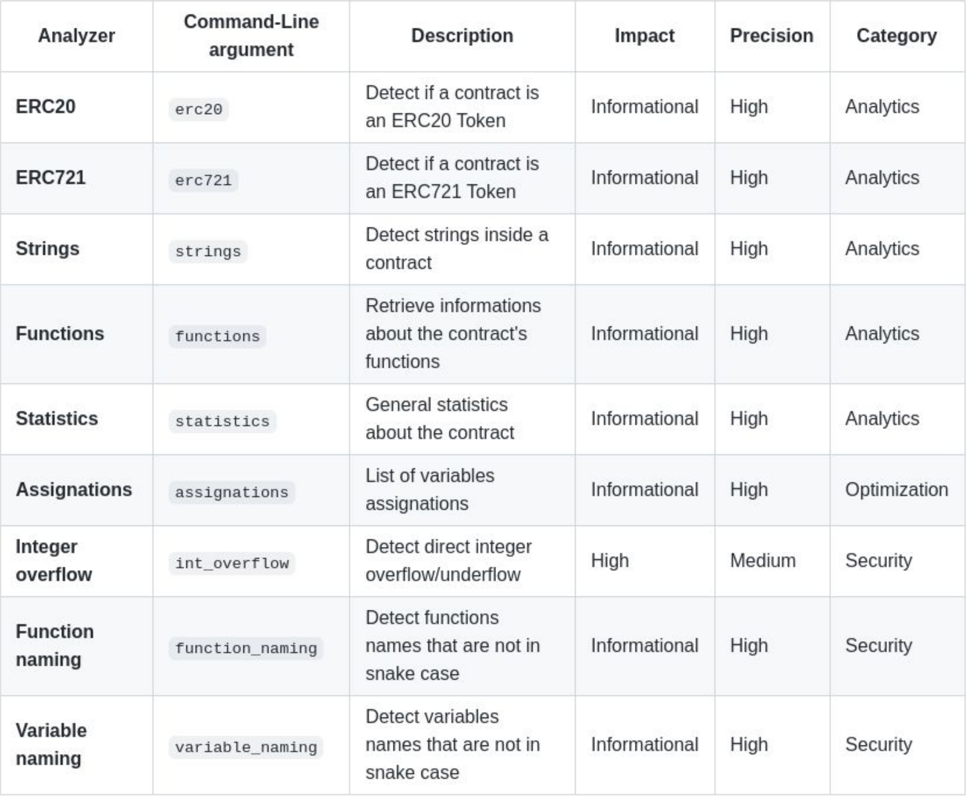
Analyzer

- The analyzer allows to detect and analyze particular behaviors in smart contracts.
  - Using the previously extracted information.

- **Analytics**
  - Interesting facts about the contract.
  - ERC detections, strings, etc.

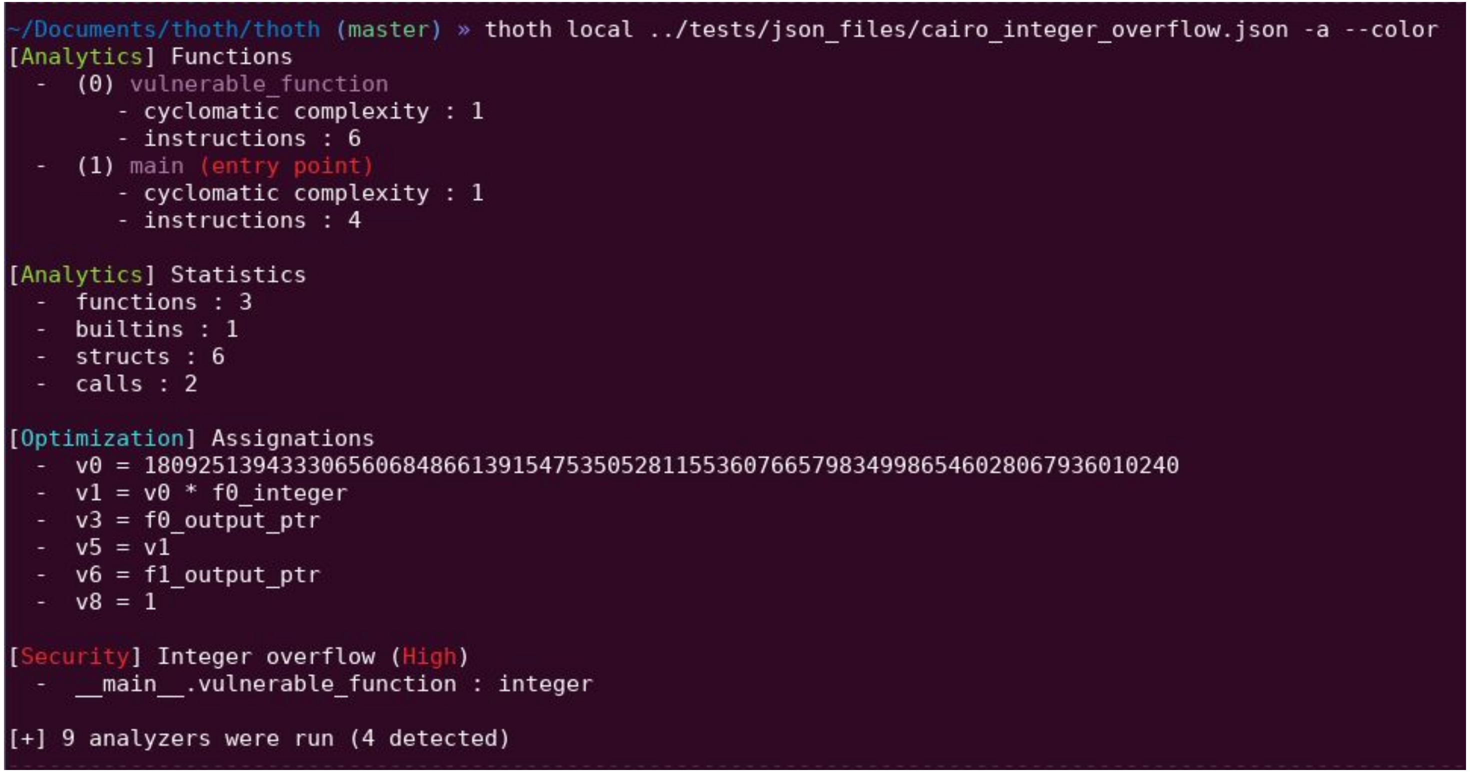
- **Optimization**
  - Detection of potential bytecode optimization.
  - Constants propagation, unused assignment, unused imports, etc.

- **Security**
  - Detection of security vulnerabilities & flaws.
  - Integer overflow, Reentrancy, etc.



Analyzer	Command-Line argument	Description	Impact	Precision	Category
ERC20	`erc20`	Detect if a contract is an ERC20 Token	Informational	High	Analytics
ERC721	`erc721`	Detect if a contract is an ERC721 Token	Informational	High	Analytics
Strings	`strings`	Detect strings inside a contract	Informational	High	Analytics
Functions	`functions`	Retrieve informations about the contract's functions	Informational	High	Analytics
Statistics	`statistics`	General statistics about the contract	Informational	High	Analytics
Assignations	`assignations`	List of variables assignations	Informational	High	Optimization
Integer overflow	`int_overflow`	Detect direct integer overflow/underflow	High	Medium	Security
Function naming	`function_naming`	Detect functions names that are not in snake case	Informational	High	Security
Variable naming	`variable_naming`	Detect variables names that are not in snake case	Informational	High	Security

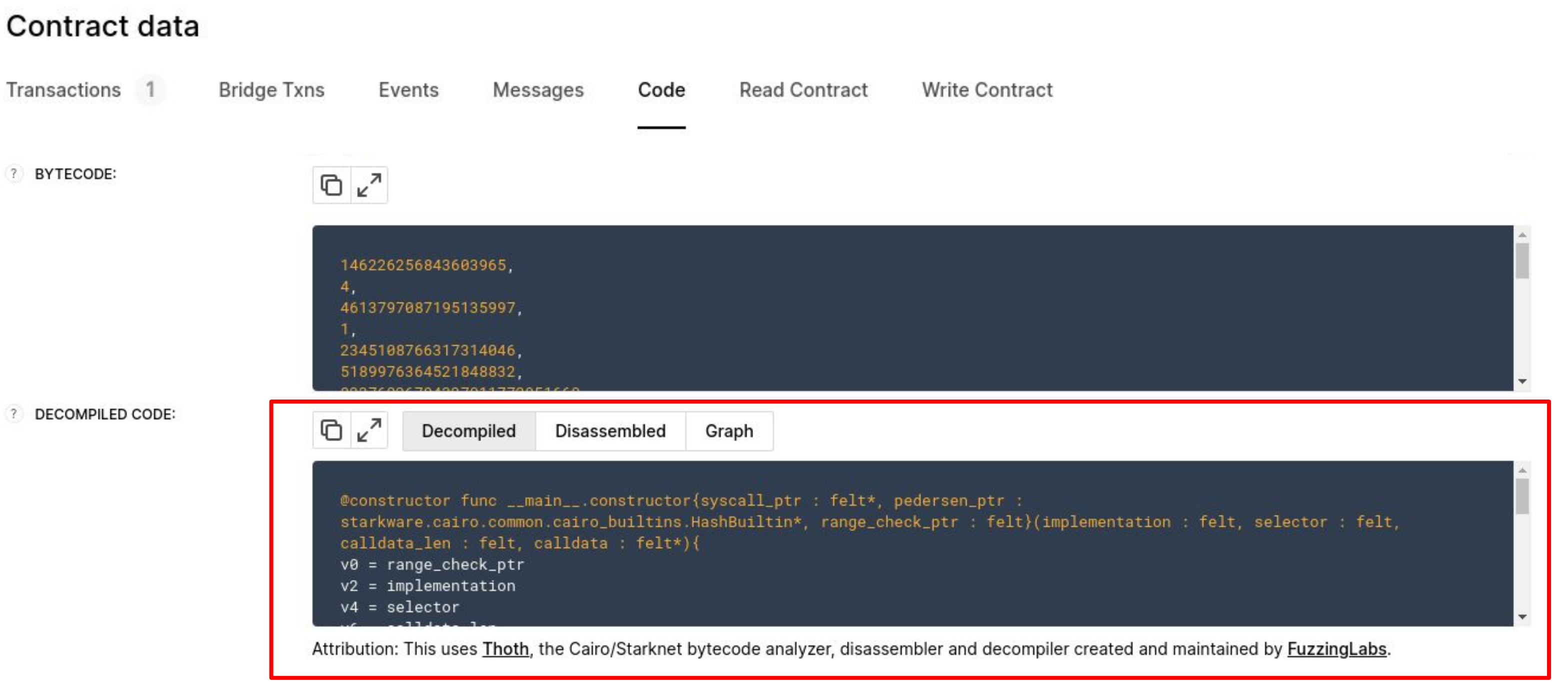
Analyzer - example (integer_overflow)



```
~/Documents/thoth/thoth (master) » thoth local ../tests/json_files/cairo_integer_overflow.json -a --color
[Analytics] Functions
  -  (0) vulnerable_function
        - cyclomatic complexity : 1
        - instructions : 6
  -  (1) main (entry point)
        - cyclomatic complexity : 1
        - instructions : 4

[Analytics] Statistics
  -  functions : 3
  -  builtins : 1
  -  structs : 6
  -  calls : 2

[Optimization] Assignations
  -  v0 = 1809251394333065606848661391547535052811553607665798349986546028067936010240
  -  v1 = v0 * f0_integer
  -  v3 = f0_output_ptr
  -  v5 = v1
  -  v6 = f1_output_ptr
  -  v8 = 1

[Security] Integer overflow (High)
  -  __main__.vulnerable_function : integer

[+] 9 analyzers were run (4 detected)
```

Upcoming features

Integration inside Voyager



Contract data

BYTECODE:

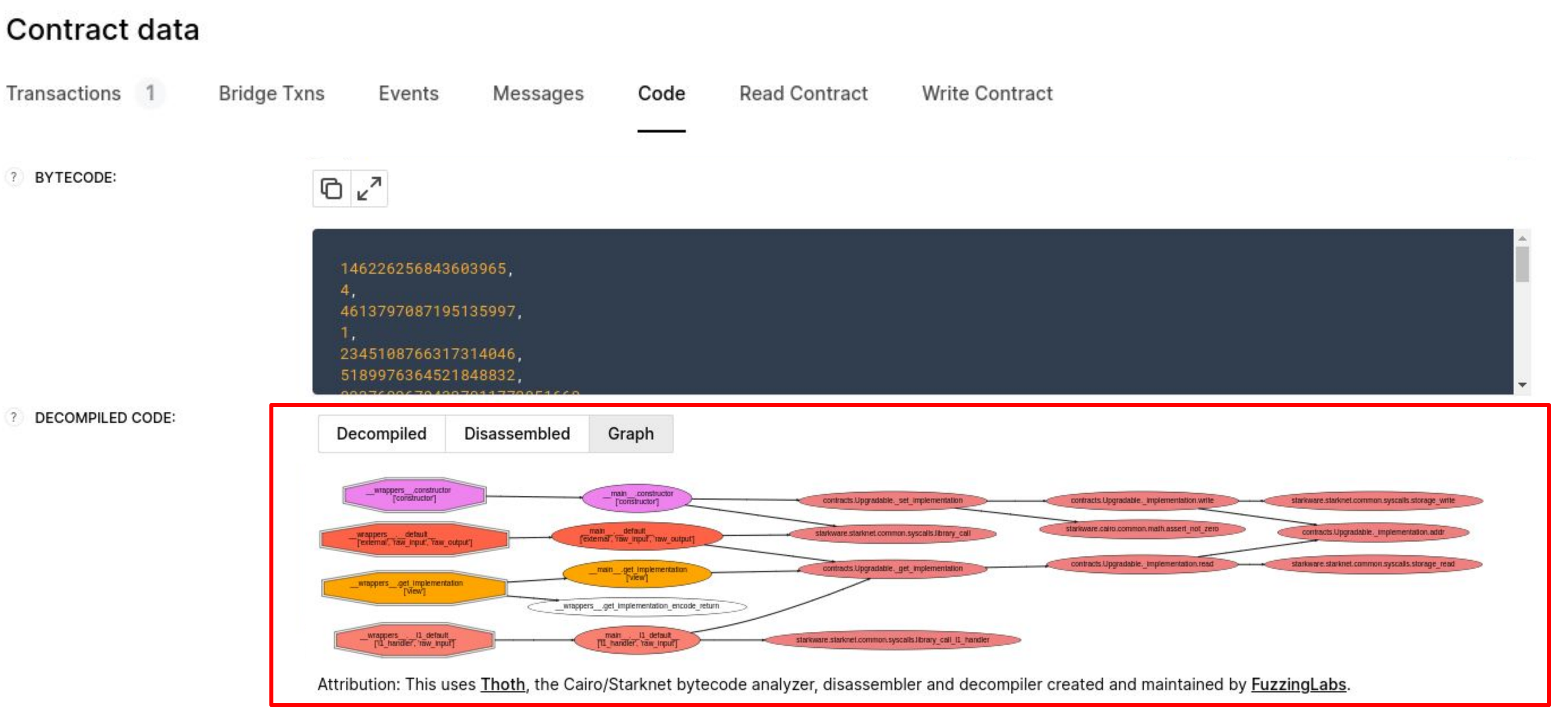
```
146226256843603965,
4,
4613797087195135997,
1,
2345108766317314046,
5189976364521848832,
```

DECOMPILED CODE:

```
@constructor func __main__.constructor{syscall_ptr : felt*, pedersen_ptr :
starkware.cairo.common.cairo_builtins.HashBuiltin*, range_check_ptr : felt}(implementation : felt, selector : felt,
calldata_len : felt, calldata : felt*){
v0 = range_check_ptr
v2 = implementation
v4 = selector
```

Attribution: This uses Thoth, the Cairo/Starknet bytecode analyzer, disassembler and decompiler created and maintained by FuzzingLabs.

Integration inside Voyager



Contract data

Transactions 1 | Bridge Txns | Events | Messages | Code | Read Contract | Write Contract

? BYTECODE:

```
1462262568436039650,
4,
4613797087195135997,
1,
2345108766317314046,
5189976364521848832,
```

? DECOMPILED CODE:

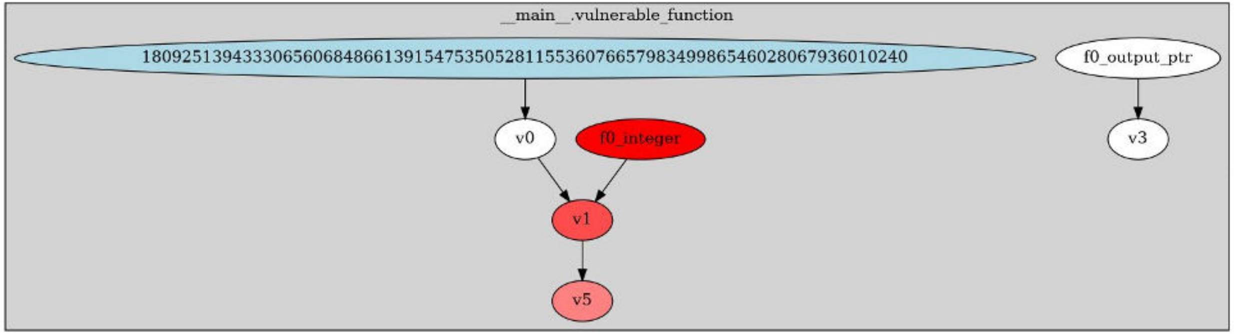
Decompiled | Disassembled | Graph

__wrappers__.constructor
['constructor']
__main__.constructor
['constructor']
contracts.Upgradable._set_implementation
contracts.Upgradable._implementation.write
starkware.starknet.common.syscalls.storage_write
__wrappers__.__default__
['external', 'raw_input', 'raw_output']
__main__.__default__
['external', 'raw_input', 'raw_output']
starkware.starknet.common.syscalls.library_call
starkware.cairo.common.math.assert_not_zero
contracts.Upgradable._implementation.addr
__wrappers__.get_implementation
['view']
__main__.get_implementation
['view']
contracts.Upgradable._get_implementation
contracts.Upgradable._implementation.read
starkware.starknet.common.syscalls.storage_read
__wrappers__.get_implementation_encode_return
__wrappers__.__l1_default__
['l1_handler', 'raw_input']
__main__.__l1_default__
['l1_handler', 'raw_input']
starkware.starknet.common.syscalls.library_call_l1_handler

Attribution: This uses Thoth, the Cairo/Starknet bytecode analyzer, disassembler and decompiler created and maintained by FuzzingLabs.

Upcoming features

- Create a representative **logo**
- Create **VS code plugin**
- Improve the **decompiler**
  - Debug info, refs, etc.
- Add more **analysis** scripts
  - Mainly security related.
  - ERC detections.



- Implement **Data Flow Graph (DFG)**
  - For variables and constants dependencies representation.
- Implement **Tainting**
  - Allows identifying supplied arguments propagation impact.
- Implement **Symbolic execution**
  - To mathematically solve the constraints to reach certain paths and detect potential optimizations of the bytecode.

Thanks for your time! Any questions?

- Contact me!
  - Twitter: **@Pat_Ventuzelo**
  - Mail: patrick@fuzzinglabs.com