Building Secure Cairo

StarkNetCC Lisbon, 31.10.2022,
Filipe Casal & Simone Monica

$ whoarewe

- **Filipe Casal**
- **Simone Monica**

- **Trail of Bits: trailofbits.com**
  - We help developers build safer software
  - R&D focused
  - Slither, Echidna, Amarna, ZKDocs, ...

Today's plan

- **Cairo Security & (Not So) Smart Contracts**
  - Common vulnerability patterns in Cairo & how to fix them
- **Amarna, static analysis for Cairo programs**
  - Features, usage & rules
- **VS Code StarkNet contract explorer**
  - Features & usage
- **Circomspect, static analysis for Circom programs**
  - Circom & current tooling
  - Rules & usage
- **Tooling Demo**

Zero-knowledge programming languages

- **New programming paradigm**
- **Languages are young and have design quirks**
- **Very few developer tools available (basically only syntax highlighting)**
    - Even harder to program and test software
- **As auditors, we also need tools**
    - To highlight potentially vulnerable code patterns
    - To perform variant analysis

Zero-knowledge programming languages

But used to power services handling millions of dollars

e.g., dYdX, Tornado Cash

```diff
2 circuits/mimcsponge.circom
@@ -21,7 +21,7 @@ template MiMCSponge(nInputs, nRounds, nOutputs) {
         }
     }

-    outs[0] = S[nInputs - 1].xL_out;
+    outs[0] <== S[nInputs - 1].xL_out;

     for (var i = 0; i < nOutputs - 1; i++) {
         S[nInputs + i] = MiMCFeistel(nRounds);
```

Cairo Security &
(Not So) Smart Contracts

A bit of history - Previous vulnerabilities

- Storage variable collision
- Implicit function import
- Direct function call

Storage variable collision

```
from starkware.cairo.common.cairo_builtins import HashBuiltin

// Suppose both have a balance storage variable
from a import a_get_balance, a_increase_balance
from b import b_get_balance, b_increase_balance

@external
func increase_balance_a{syscall_ptr: felt*, pedersen_ptr: HashBuiltin*, range_check_ptr}(
   amount: felt
) {
   a_increase_balance(amount);
   return ();
}

@external
func increase_balance_b{syscall_ptr: felt*, pedersen_ptr: HashBuiltin*, range_check_ptr}(
   amount: felt
) {
   b_increase_balance(amount);
   return ();
}
```

Implicit function import

```
// library.cairo
func assert_owner() {
   let (caller) = get_caller_address();
   let (owner) = owner_storage.read();
   assert caller = owner;
   return ();
}

func mint_internal(to: felt, amount: felt) {
   let (balance) = balance_of.read(to);
   balance_of.write(to, balance + amount);
   return ();
}

@external
func test_mint(to, amount) {
   mint_internal(to, amount);
   return ();
}
```

```
// main.cairo
from library import mint_internal, assert_owner

@external
func mint(to: felt, amount: felt) {
   assert_owner();
   mint_internal(to, amount);
   return ();
}
```

Direct function call

```
func ERC721_transferFrom{...}(
        _from: felt, to: felt, token_id: Uint256
    ):
        let (caller) = get_caller_address()
        let (is_approved) = _is_approved_or_owner(caller, token_id)
        assert is_approved = 1
        _transfer(_from, to, token_id)
        return ()
end

func _is_approved_or_owner{...}(
        spender: felt, token_id: Uint256
    ) -> (res: felt):
        // ...
        let (approved_addr) = ERC721_getApproved(token_id)
        if approved_addr == spender:
            return (1)
        end
        // ...
```

[ERC721_transferFrom and ERC721_safeTransferFrom allow improper transfer of tokens]()

Back to our days

Arithmetic

- Division
- Comparison
- Uint256

L1 <-> L2 messages quirks

Division

```
@view
func normalize_tokens{...}() -> (normalized_balance : felt) {
   let (user) = get_caller_address();

   let (user_current_balance) = user_balances.read(user);
   let (normalized_balance) = user_current_balance / 10**18;

   return (normalized_balance);
}
```

```
user_current_balance = 10.5 * (10 ** 18)
normalized_balance = -18092513943330656068486613915475…
```

Division - Correct

```
from starkware.cairo.common.math import unsigned_div_rem

@view
func normalize_tokens{...}() -> (normalized_balance : felt) {
   let (user) = get_caller_address();

   let (user_current_balance) = user_balances.read(user);
   let (normalized_balance, _) = unsigned_div_rem(user_current_balance, 10**18);

   return (normalized_balance);
}
```

Use `unsigned_div_rem` from the standard library

How to do comparisons?

```
from starkware.cairo.common.math import assert_le
from starkware.starknet.common.syscalls import get_caller_address

@storage_var
func balance(account: felt) -> (res: felt) {
}

@external
func transfer{...}(recipient: felt, amount: felt) {
   let (sender) = get_caller_address();
   let (balance: felt) = balance.read(sender);
   // Check that the user has enough tokens
   assert_le(amount, balance);
   // ...
   return ();
}
```

How to do comparisons? Correct

```
from starkware.cairo.common.math import assert_nn_le
from starkware.starknet.common.syscalls import get_caller_address

@storage_var
func balance(account: felt) -> (res: felt) {
}

@external
func transfer{...}(recipient: felt, amount: felt) {
   let (sender) = get_caller_address();
   let (balance: felt) = balance.read(sender);
   // Check the user has enough tokens
   assert_nn_le(amount, balance);
   // ...
   return ();
}
```

Use `assert_nn_le` to check the amount is not negative.

Uint256

Uint256 elements are made of two felts.

```
struct Uint256 {
   // The low 128 bits of the value.
   low: felt,
   // The high 128 bits of the value.
   high: felt,
}
```

```
from starkware.cairo.common.uint256 import Uint256, uint256_le
from starkware.starknet.common.syscalls import get_caller_address

@storage_var
func balance(account: felt) -> (res: Uint256) {
}

@external
func transfer{...}(recipient: felt, amount: Uint256) {
   let (sender) = get_caller_address();
   let (balance: Uint256) = balance.read(sender);
   // Check the user has enough tokens
   let (res) = uint256_le(amount, balance);
   assert res = TRUE;
   // ...
   return ();
}
```

Uint256 - correct

```
from starkware.cairo.common.uint256 import Uint256, uint256_le,
uint256_check
from starkware.starknet.common.syscalls import get_caller_address

@storage_var
func balance(account: felt) -> (res: Uint256) {
}

@external
func transfer{...}(recipient: felt, amount: Uint256) {
   uint256_check(amount);
   let (sender) = get_caller_address();
   let (balance: Uint256) = balance.read(sender);
   // Check the user has enough tokens
   let (res) = uint256_le(amount, balance);
   assert res = TRUE;
   // ...
   return ();
}
```

Use `uint256_check` to ensure the element is a valid Uint256.
Use [`SafeUint256`]() for operations.

l1 -> l2 message

- l1 contract calls `sendMessageToL2(uint256 toAddress, uint256 selector, uint256[] calldata payload)` on the StarkNet core contract.

```
function deposit(uint256 receiver, uint256 amount) public {
    require(receiver != 0 && receiver < FIELD_PRIME);

    token.safeTransferFrom(msg.sender, address(this), amount);

    uint256 memory payload = new uint256[](3);
    payload[0] = receiver;
    payload[1] = amount & ((1 << 128) - 1);
    payload[2] = amount >> 128;

    starknetContract.sendMessageToL2(
        l2Contract,
        DEPOSIT_SELECTOR,
        payload
    );
}
```

l1 -> l2 message

- l2 `deposit` function which handles a message sent from l1.

```
@l1_handler
func deposit{...}(from_address: felt, user: felt, amount_low: felt, amount_high: felt) {
   // Check the message was sent by the expected l1 contract
   assert from_address = L1_CONTRACT_ADDRESS;

   let amount = Uint256(low=amount_low, high=amount_high);

   token.permissionedMint(user, amount);

   return ();
}
```

l1 -> l2 message cancellation

- startL1ToL2MessageCancellation(uint256 toAddress, uint256 selector,uint256[] calldata payload,uint256 nonce)
- cancelL1ToL2Message(uint256 toAddress,uint256 selector,uint256[] calldata payload,uint256 nonce)

```
function cancelDeposit(uint256 receiver, uint256 amount, uint256 nonce) public {
    require(receiver != 0 && receiver < FIELD_PRIME);

    uint256 low = amount & ((1 << 128) - 1);
    uint256 high = amount >> 128;

    uint256 memory payload = new uint256[](3);
    payload[0] = receiver;
    payload[1] = low;
    payload[2] = high;

    starknetContract.cancelL1toL2Message(
        l2Contract,
        DEPOSIT_SELECTOR,
        payload,
        nonce
    );

    token.transfer(receiver, amount);
}
```

l1 -> l2 message cancellation - correct

```
function cancelDeposit(uint256 receiver, uint256 amount, uint256 nonce) public
{
    require(receiver != 0 && receiver < FIELD_PRIME);

    uint256 low = amount & ((1 << 128) - 1);
    uint256 high = amount >> 128;

    uint256 memory payload = new uint256[](4);
    payload[0] = uint256(uint160(msg.sender));
    payload[1] = receiver;
    payload[2] = low;
    payload[3] = high;

    starknetContract.cancelL1toL2Message(
        l2Contract,
        DEPOSIT_SELECTOR,
        payload,
        nonce
    );

    token.safeTransfer(receiver, amount);
}
```

Use `msg.sender` in the `payload`. This way, only the address that started the deposit can cancel it.

l2 -> l1

- `send_message_to_l1(to_address: felt, payload_size: felt, payload: felt*)`

```
from starkware.starknet.common.messages import send_message_to_l1
from starkware.starknet.common.eth_utils import assert_eth_address_range

@external
func initiate_withdraw{...}(
      l1_recipient: felt,
      amount: Uint256) {
   uint256_check(amount);
   assert_eth_address_range(l1_recipient);

   let (sender) = get_caller_address();
   token.permissionedBurn(sender, amount);

   let (payload: felt*) = alloc();
   assert payload[0] = WITHDRAW_MESSAGE;
   assert payload[1] = l1_recipient;
   assert payload[2] = amount.low;
   assert payload[3] = amount.high;

   send_message_to_l1(to_address=l1_contract_address, payload_size=4, payload=payload);
}
```

l2 -> l1

- `consumeMessageFromL2(uint256 fromAddress, uint256[] calldata payload)`

```
function withdraw(address recipient, uint256 amount) external {
    // Users must withdraw at least 10 tokens
    require(amount >= 10 * 10**18);

    uint256 low = amount & ((1 << 128) - 1);
    uint256 high = amount >> 128;

    uint256[] memory payload = new uint256[](4);
    payload[0] = WITHDRAW_MESSAGE;
    payload[1] = recipient;
    payload[2] = low;
    payload[3] = high;

    starknetContract.consumeMessageFromL2(l2Contract, payload);

    token.safeTransfer(recipient, amount);
}
```

l2 -> l1 - Correct

```
from starkware.starknet.common.messages import send_message_to_l1
from starkware.starknet.common.eth_utils import assert_eth_address_range

@external
func initiate_withdraw{...}(l1_recipient: felt, amount: Uint256) {
   uint256_check(amount);
   assert_eth_address_range(l1_recipient);

   let ten_tokens = Uint256(low=10 * 10**18, high=0);
   let (is_lt) = uint256_lt(ten_tokens, amount);
   assert is_lt = TRUE;

   let (sender) = get_caller_address();
   token.permissionedBurn(sender, amount);

   let (payload: felt*) = alloc();
   assert payload[0] = WITHDRAW_MESSAGE;
   assert payload[1] = l1_recipient;
   assert payload[2] = amount.low;
   assert payload[3] = amount.high;

   send_message_to_l1(to_address=l1_contract_address, payload_size=4,
payload=payload);
}
```

We add the check on the l2 side to avoid users losing tokens.
l2 to l1 messages are not cancellable.

Learn More

- **Want to learn more about common Cairo vulnerabilities?**
  - Building secure contracts
    - Available at https://github.com/crytic/building-secure-contracts
    - Includes detailed information about the most common vulnerabilities

Not So Smart Contract	Description
Improper access controls	Broken access controls due to StarkNet account abstraction
Integer division errors	Unexpected results due to division in a finite field
View state modifications	View functions don't prevent state modifications
Arithmetic overflow	Arithmetic in Cairo is not safe by default
Signature replays	Account abstraction requires robust reuse protections
L1 to L2 Address Conversion	L1 to L2 messaging requires L2 address checks
Incorrect Felt Comparison	Unexpected results can occur during felt comparison
Namespace Storage Var Collision	Storage variables are not scoped by namespaces
Dangerous Public Imports in Libraries	Nonimported external functions can still be called

Amarna, static analysis for Cairo programs

Amarna, static analysis for Cairo programs

- **Finds 14 types of code-smells and vulnerabilities in Cairo code**
- **Compiler-identical parsing of Cairo code and StarkNet contracts**
  - Now supports Cairo v0.10
- **It allows us to easily write rules**
- **Available at github.com/crytic/amarna**

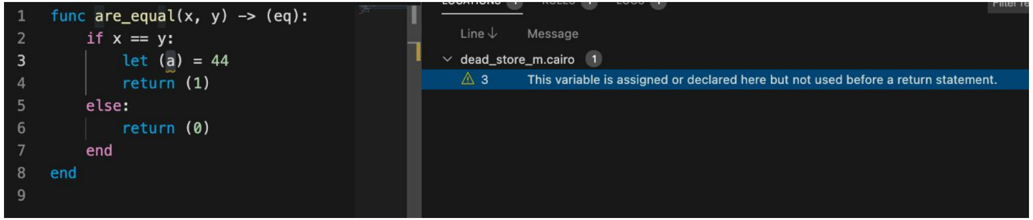
#	Rule	What it finds	Impact	Precision
1	Arithmetic operations	All uses of arithmetic operations +, -, *, and /	Info	High
2	Unused arguments	Function arguments that are not used in the functions in which they appear	Warning	High
3	Unused imports	Unused imports	Info	High
4	Mistyped decorators	Mistyped code decorators	Info	High
5	Unused functions	Functions that are never called	Info	Medium
6	Error codes	Function calls that have return values that must be checked	Info	High
7	Inconsistent assert usage	Asserts that use the same constant in different ways, e.g., `assert_le(amount, BOUND)` and `assert_le(amount, BOUND - 1)`	Warning	High
8	Dead stores	Variables that are assigned values but not used before a return statement	Info	Medium
9	Unchecked overflows	Function calls that ignore the returned overflow flags, e.g., `uint256_add`	Warning	High
10	Caller address return value	Function calls to the `get_caller_address` function.	Info	High
11	Storage variable collision	Multiple `@storage_var` with the same name.	Warning	High
12	Implicit function import	Function with decorator `@external, @view, @l1_handler` that is being implicitly imported.	Warning	High
13	Unenforced view function	State modification within a `@view` function	Error	High
14	Uninitialized variable	Local variables that are never initialized.	Info	High

Amarna, static analysis for Cairo programs

- CI/CD: GitHub action integration with amarna-action
- Simple to use:

```
$ pip install amarna
$ cd your_cairo_project
$ amarna . -o results.sarif
```

- Exports results as SARIF, and visualize them in VSCode:



How does Amarna find vulnerabilities?

1. Amarna parses the Cairo code with the compiler grammar
2. Runs three types of rules:
    - **local rules** analyse each file independently
    - **gatherer rules** analyse each file independently and gather data to be used in post-process rules
    - **post-process rules** run after all files were analyzed and use the data gathered with the gatherer rules

How does Amarna find vulnerabilities?

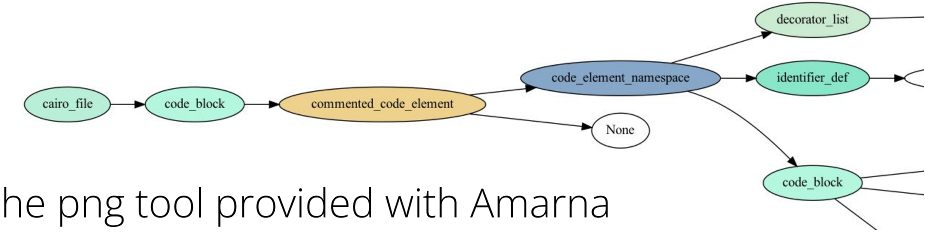
Examples of different rules:

- **local rules:** find all arithmetic operations in a file
- **gatherer rules:** gather all declared functions, and called functions
- **post-process rules:** find unused functions using the gathered data, i.e., functions that were declared but never called.

Extending Amarna with new rules

Knowing what to look for is usually the hard part

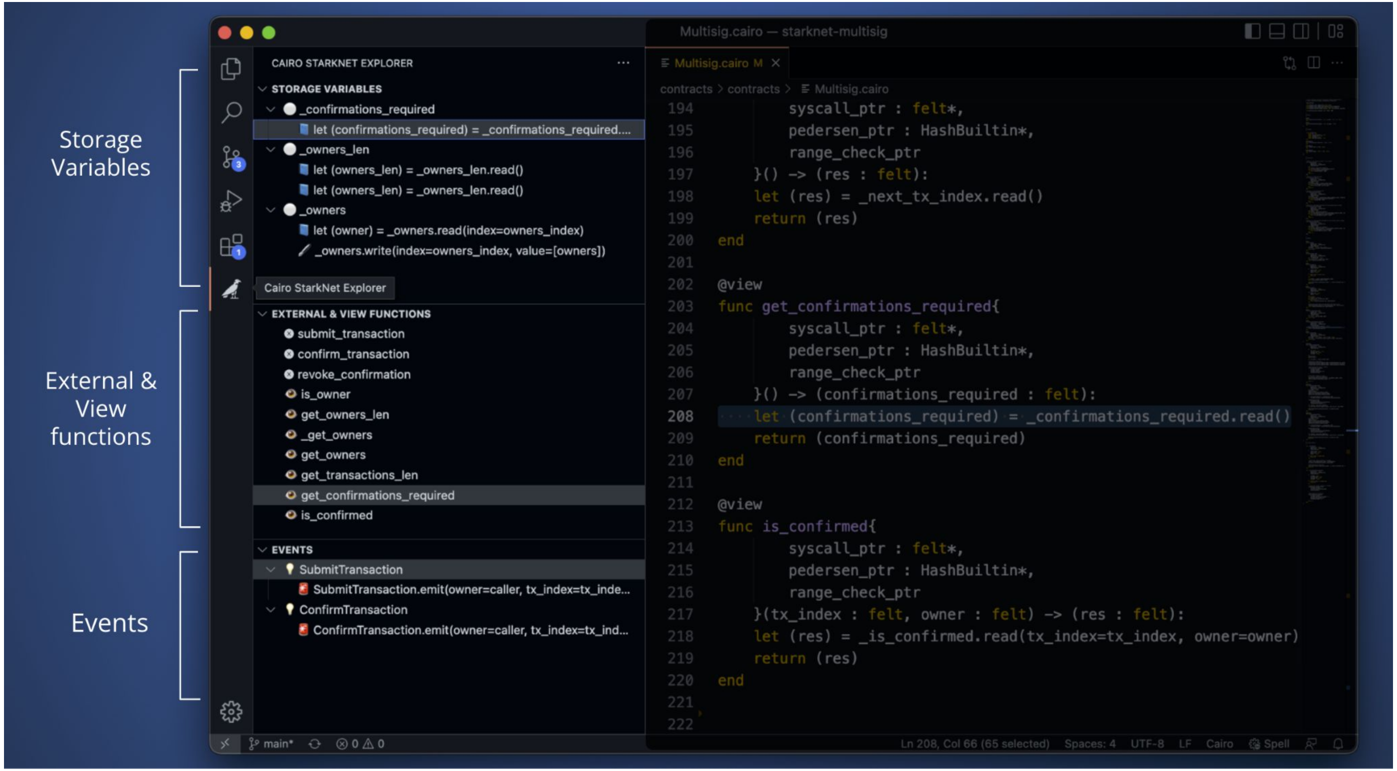
Creating new rules 101:



- Create a small test program
- Visualize the test program tree with the png tool provided with Amarna
- Determine what type of information the rule needs:
  - Local information: write a local rule
  - Global information: write a post-process rule.
    - Several gatherers are already implemented (e.g., import gatherer, function call gatherer), but a more specific one might be needed.

VS Code StarkNet explorer

VS Code StarkNet explorer

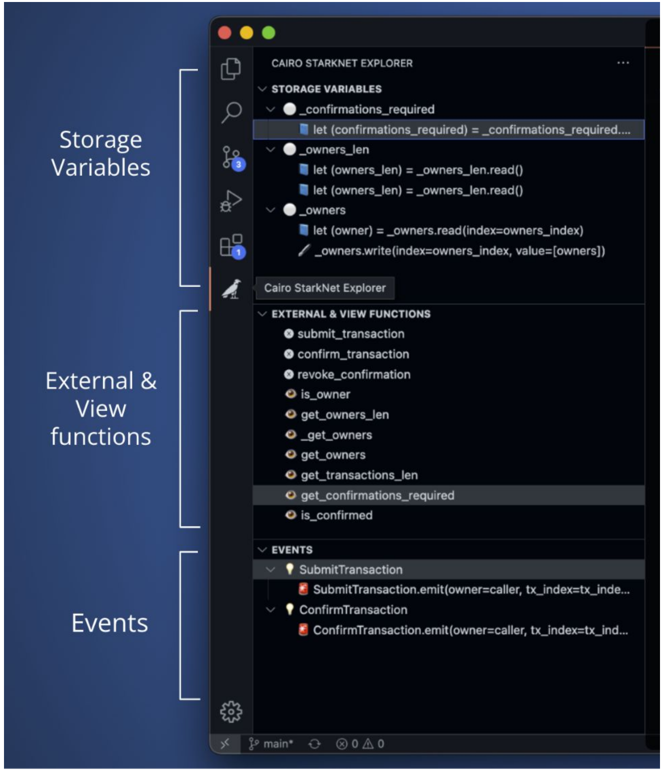


VS Code StarkNet Explorer

- **Storage variables:** where they are read and where they are written
- **External & View functions:** quickly navigate to all external and view functions
- **Events:** shows event declaration and where each event is emitted
- The view is automatically updated while the code is written
- Available at

  github.com/crytic/vscode-starknet-explorer

  marketplace.visualstudio.com/items?itemName=trailofbits.starknet-explorer



Circomspect, the Circom static-analyzer

Circom - a circuit compiler

- **Circuit DSL and compiler**
- **Outputs R1CS constraints which can be passed to Snarkjs**
  - Snarkjs currently supports Groth16 and Plonk
- **Few tools exist besides the compiler**

```
pragma circom 2.0.0;

// This circuit template checks that c is
// the product of the two inputs a and b.

template Multiplier(){
   // Declaration of signals a, b, and c.
   signal input a;
   signal input b;
   signal output c;

   // Constraint ensuring that c = a * b.
   c <== a * b;
}

component main {public [a, b]} = Multiplier();
```

Circomspect, static analysis for Circom

- Written in Rust, based on the Circom compiler
- Detects code-smells and potential vulnerabilities in Circom code
- Compiles to an SSA intermediate representation, which allows for basic data-flow analysis
- Available at github.com/trailofbits/circomspect

```
template BinSum(n, ops) {
    signal input in[ops][n];
    signal output out[nout];

    var lin = 0;
    var lout = 0;
    var nout = nbits((2 ** n - 1) * ops);

    var e2 = 1;
    for (var k = 0; k < n; k++) {
        for (var j = 0; j < ops; j++) {
            lin += in[j][k] * e2;
        }
        e2 = e2 + e2;
    }

    e2 = 1;
    for (var k = 0; k < nout; k++) {
        out[k] <-- (lin >> k) & 1;
        out[k] * (out[k] - 1) === 0;

        lout += out[k] * e2;  // The value assigned here is not used.
        e2 = e2 + e2;
    }

    lin === nout;  // Should use `lout`, but uses `nout` by mistake.
}
```

Circomspect, static analysis for Circom

- **Focus on finding issues not flagged by the compiler**
  - Always run the compiler with `circom --inspect`
- **Results can be written to stdout, or as SARIF**

```
warning: Using the signal assignment operator `<--` does not constrain the assigned signal.
    ┌─ examples/dead-assignment.circom:21:9
    │
21  │         out[k] <-- (lin >> k) & 1;
    │         ^^^^^^^^^^^^^^^^^^^^^^^^^ The assigned signal `out[k]` is not constrained here.
    │
    = Consider using the constraint assignment operator `<==` instead.

warning: The value assigned to `lout` is not used in either witness or constraint generation.
   ┌─ examples/dead-assignment.circom:8:5
   │
8  │     var lout = 0;
   │     ^^^^^^^^^^^^ This assignment to `lout` could be removed.

warning: The value assigned to `lout` is not used in either witness or constraint generation.
    ┌─ examples/dead-assignment.circom:24:9
    │
24  │         lout += out[k] * e2;  // The value assigned here is never read.
    │         ^^^^^^^^^^^^^^^^^^^ This assignment to `lout` could be removed.

3 issues found.
```

Your mission: Try them out!

Amarna

Available at github.com/crytic/amarna

```
$ pip install amarna
$ cd your/cairo/project
# Print results summary
$ amarna . -s
# Export results as SARIF
$ amarna . -o results.sarif
```

Circomspect

Available at github.com/trailofbits/circomspect

```
$ cargo install circomspect
$ cd your/circom/project
# Print results to stdout
$ circomspect circuits
# Export results as SARIF
$ circomspect circuits -s results.sarif
```

VSCode Cairo StarkNet explorer

Available at github.com/crytic/vscode-starknet-explorer

or the VSCode extension Marketplace

```
After installing the extension
 - open a Cairo contract in VSCode
 - open the extension tab
```

Demos

Amarna

Write a rule to find:
- calls to get_caller_address
- in a @l1_handler

Use the skeleton at
https://gist.github.com/fcasal/a3b160322395b4399ba917a759e35151

VSCode Cairo StarkNet explorer

After installing the extension
- open a Cairo contract in VSCode
- open the extension tab

Thanks for listening

Filipe Casal
Senior Security Consultant

filipe.casal@trailofbits.com
www.trailofbits.com

Simone Monica
Security Consultant

simone.monica@trailofbits.com
www.trailofbits.com

TRAIL OF BITS