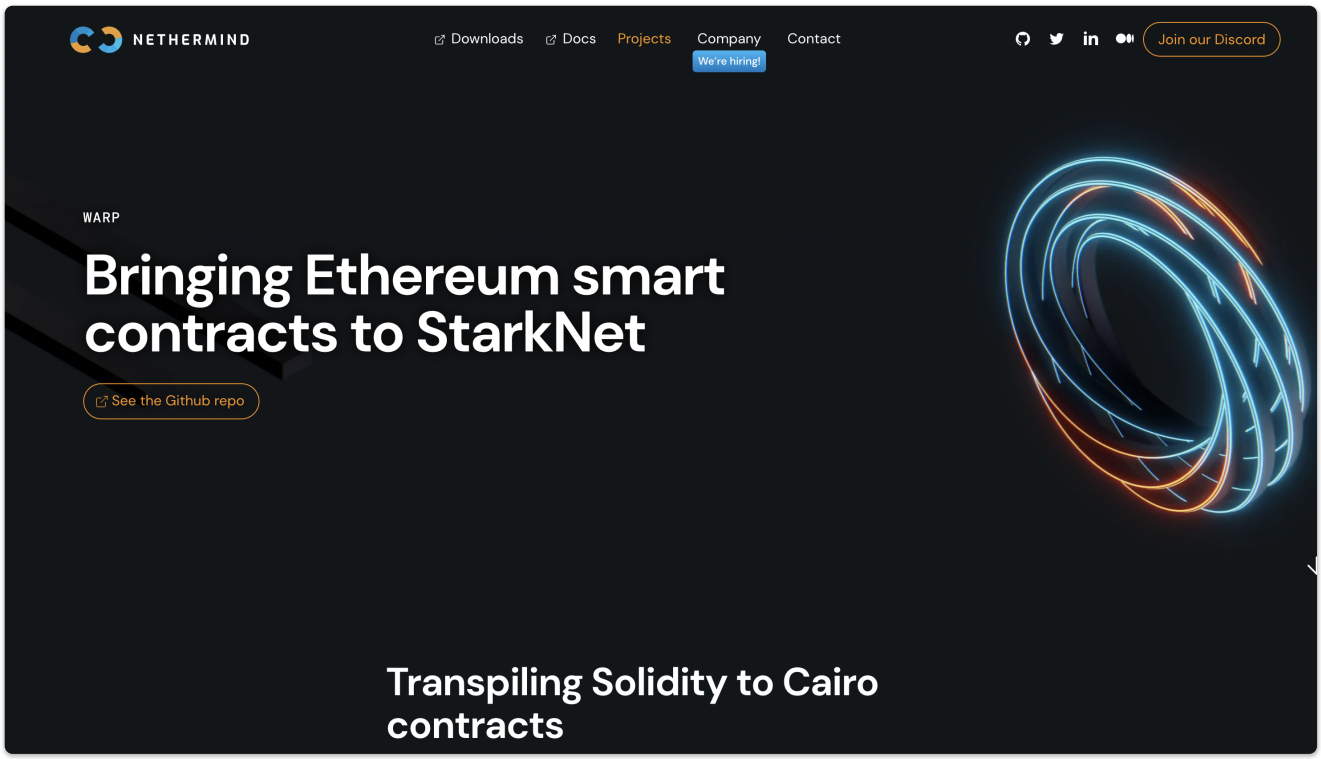
Lesson 14 - Warp

Warp



Warp allows you transpile Solidity contracts into Cairo

Installation Instructions

See Warp installation instructions

1. On macos:

```
brew install z3
```

2. On ubuntu:

```
sudo apt install libz3-dev
```

Make sure that you have the `venv` module for your python installation.

Installation

Without any virtual environment activated run the following in order:

```
yarn global add @nethermindeth/warp
```

Run the following to see the version and that it was installed:

`warp version`

Finally run the following to install the dependencies:

`warp install`

Test installation works by transpiling an example ERC20 contract:

`warp transpile example_contracts/ERC20.sol`

Using Docker

`docker build -t warp .`

`docker run --rm -v $PWD:/dapp --user $(id -u):$(id -g) warp transpile example_contracts/ERC20.sol`

Using Warp

`warp transpile example_contracts/ERC20.sol`

`warp transpile example_contracts/ERC20.sol --compile-cairo`

You can then deploy your cairo code to the network, with the following commands you need to specify the network, in our case alpha-goerli

```
warp deploy ERC20.json --network alpha-goerli
```

```
Deploy transaction was sent.
Contract address:
0x0403bd2f0abdd765398d6a50ff89cfe9ac48760f3b94ba2728bfbacdaff9f59a
Transaction hash:
0x32ca42d1341703cc957845ea53a71b3eb2e762ff148cb9dc522322eede94b65
```

You can invoke a transaction on your contract

```
warp invoke --program ERC20.json --address
0x0403bd2f0abdd765398d6a50ff89cfe9ac48760f3b94ba2728bfbacdaff9f59a  --
network
alpha-goerli --function store --inputs [13]
```

```
Invoke transaction was sent.
Contract address:
0x0403bd2f0abdd765398d6a50ff89cfe9ac48760f3b94ba2728bfbacdaff9f59a
Transaction hash:
0x1d1ec8278ccf41452737e80a54e7626299e598528363ced7a527d810f9d6881
```

And check the status

```
warp status
0x1d1ec8278ccf41452737e80a54e7626299e598528363ced7a527d810f9d6881 --
network alpha-goerli
```

which will give a answer similar to

```
        {
                "block_hash":
"0x1c55254f16d087f0bf7776183c4d38549680e68600394167f304f1afe5a035e",
                "tx_status": "ACCEPTED_ON_L1"
        }
```

You should be able to see the details on the block explorer
Voyager Block Explorer

Unsupported Features

Support Status	Symbol
Will likely never be supported	❌
Being developed/investigated	⚒️
Currently Unknown/If added in Cairo	❓

Solidity	Support Status
fallback functions with args	⚒️
delegate calls	⚒️
low level calls	❌
indexed parameters	❓
nested tuple expressions	❓
typeName expressions	❓
gasleft()	❓
msg.value	❓
msg.sig	❓
msg.data	❓
tx.gasprice	❓
tx.origin	❓
try/catch	❓
block.coinbase	❓
block.gaslimit	❓
block.basefee	❓
block.chainid	❓
block.difficulty	❌
precompiles (apart from ecrecover)	❓
selfdestruct	❓
blockhash	❓

functions pointers in storage	❓
sha256 (use keccak256 instead)	❌
ternary operator	⚒️
receive	❓
Inline Yul Assembly - arithmetic (add, sub ...)	⚒️
Inline Yul Assembly - (memory, calldata, storage)	❓
user defined errors	❓
function call options e.g x.f{gas: 10000}(arg1)	❓
member access of address object e.g address.balance	❓
nested tuple assignments	❓

There is also now a vyper transpiler

UniStark

See Announcement
See repo

Nethermind has used to transpile Uniswap V3 Solidity contracts to Cairo.

Starknet JVM

SDK for JVM languages:

- Java
- Kotlin
- Scala
- Clojure
- Groovy

Example Java code

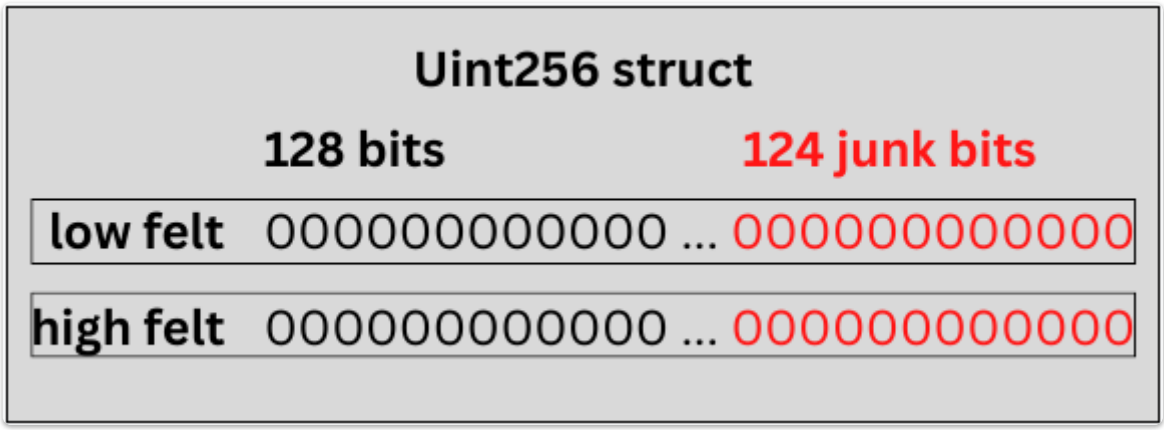
```
import com.swmansion.starknet.account.Account;
import com.swmansion.starknet.account.StandardAccount;
import com.swmansion.starknet.data.types.BlockTag;
import com.swmansion.starknet.data.types.Felt;
import com.swmansion.starknet.provider.Provider;
import com.swmansion.starknet.provider.Request;
import com.swmansion.starknet.provider.gateway.GatewayProvider;

public class Main {
    public static void main(String[] args) {
        // Create a provider for interacting with StarkNet
        Provider provider = GatewayProvider.makeTestnetClient();

        // Create an account interface
        Felt accountAddress = Felt.fromHex("0x13241455");
        Felt privateKey = Felt.fromHex("0x425125");
        Account account = new StandardAccount(provider, accountAddress,
privateKey);

        // Make a request
        Felt contractAddress = Felt.fromHex("0x42362362436");
        Felt storageKey = Felt.fromHex("0x13241253414");
        Request<Felt> request = account.getStorageAt(contractAddress,
storageKey, BlockTag.LATEST);
        Felt response = request.send();

        System.out.println(response);
    }
}
```

Security

Useful article
We have covered most of the issues in the article, but it is a useful checklist and also has information about preparing for audits.
We will cover Oracles in more detail in Lesson 15

Article -how to hack any cairo contract

The potential problem revolves around the Uint256 library.
This has some 'junk' bits.



The problem is that some of the comparison operation is_le() also includes the 'junk' bits in the comparison.
This also applies to other functions

- uint256_add()
- uint256_mul()
- uint256_sub()
- uint256_lt()
- uint256_eq()

This vulnerability happens if the junk bits are set, for example someone inputs a Uint256 with junk bits set.
The solution is to check Uint256 values as arguments, by using the uint256_check()

In Cairo 1.0 the Uint256 will be a native type.

Patterns to avoid

This article lists some patterns to avoid when writing programs / contracts.

Some points we have not explicitly covered

1. The view annotation does not prevent state changes.
2. Signature schemes need to check for replay attacks
3. L1 to L2 messaging should check the validity of L2 addresses
4. Non imported functions can still be called from a contract.
   The Amarna static analyser can detect this problem