ETHBerlin @ETHBerlin · Jun 7

We are super excited that tomorrow's not only Friday but also @sohkai will be presenting @AragonProject's Aragon 0.5 for the #Berlin developer audience in @blueyard, so excited we are already imagining how it will all look like. Sign up here: eventbrite.com/e/aragon-dev-m…

Playing 🐈 + 🐀 with the EVM

A tale of two ScriptExecutors

🐾🐾🐾🐾🐾🐾🐾

Brett Sun | @sohkai | Aragon One

Setting

The background to our story

Delegate Proxies

0xf4	DELEGATECALL	6	1	Message-call into this account with an alternative account's code, but persisting the current values for *sender* and *value*. Compared with CALL, DELEGATECALL takes one fewer arguments. The omitted argument is $\boldsymbol{\mu}_{\mathbf{s}}[2]$. As a result, $\boldsymbol{\mu}_{\mathbf{s}}[3]$, $\boldsymbol{\mu}_{\mathbf{s}}[4]$, $\boldsymbol{\mu}_{\mathbf{s}}[5]$ and $\boldsymbol{\mu}_{\mathbf{s}}[6]$ in the definition of CALL should respectively be replaced with $\boldsymbol{\mu}_{\mathbf{s}}[2]$, $\boldsymbol{\mu}_{\mathbf{s}}[3]$, $\boldsymbol{\mu}_{\mathbf{s}}[4]$ and $\boldsymbol{\mu}_{\mathbf{s}}[5]$. Otherwise exactly equivalent to CALL except: $(\boldsymbol{\sigma}', g', A^+, \mathbf{o}) \equiv \begin{cases} \Theta(\boldsymbol{\sigma}^*, I_s, I_o, I_a, t, \\ \quad \boldsymbol{\mu}_{\mathbf{s}}[0], I_p, 0, I_v, \mathbf{i}, I_e + 1) & \text{if} \quad I_v \leqslant \boldsymbol{\sigma}[I_a]_b \ \wedge \ I_e < 1024 \\ (\boldsymbol{\sigma}, g, \varnothing, \mathbf{o}) & \text{otherwise} \end{cases}$ Note the changes (in addition to that of the fourth parameter) to the second and ninth parameters to the call Θ. This means that the recipient is in fact the same account as at present, simply that the code is overwritten *and* the context is almost entirely identical.

Delegatecall / Callcode and Libraries

There exists a special variant of a message call, named **delegatecall** which is identical to a message call apart from the fact that the code at the target address is executed in the context of the calling contract and `msg.sender` and `msg.value` do not change their values.

Forwarders

APPS

- Home
- Token Manager
- Voting
- **Finance**
- Permissions SOON
- Apps SOON
- Settings

Finance

Token Balances

BCC	ETH	ZRX	DNT	ANT
26	3.14	453	137	0
$16,938.11	$1,481.15	$557.47	$5.37	$0

Transfers

DATE	SOURCE / RECIPIENT	REFER
13/04/18	0x61D463f2613F04053bc875C72aA9dA22348Da2b5	Food aid
13/04/18	0x39a4D265db942361D92e2B0039cae73Ea72a2ff9	Request
13/04/18	0x772080b0D48De808dbE0F4050083B0f9205F6D29	Ether tra
13/04/18	0x39a4D265db942361D92e2B0039cae73Ea72a2ff9	Request

New Transfer

RECIPIENT (MUST BE A VALID ETHEREUM ADDRESS) *

thai-cave-rescue.donations.eth

AMOUNT

150000 DAI

REFERENCE

Disaster relief (Thai cave rescue)

Submit Transfer

APPS

- Home
- Token Manager
- Voting
- Finance
- Permissions SOON
- Apps SOON
- Settings

Finance

Token Balances

BCC	ETH	ZRX	DNT	ANT
26	3.14	453	137	0
$16,938.11	$1,481.15	$557.47	$5.37	$0

Transfers

DATE	SOURCE / RECIPIENT	REFER
13/04/18	0x61D463f2613F04053bc875C72aA9dA22348Da2b5	Food aid
13/04/18	0x39a4D265db942361D92e2B0039cae73Ea72a2ff9	Request
13/04/18	0x772080b0D48De808dbE0F4050083B0f9205F6D29	Ether tra
13/04/18	0x39a4D265db942361D92e2B0039cae73Ea72a2ff9	Request

Sign Transaction

Permission note:

You cannot directly perform this action. You do not have the necessary permissions.

Action Requirement

Here are some options that you can use to perform it:

Voting (ANT)

The *Voting (ANT)* app will create a new voting for ANT holders to decide wether to perform this action or not.

Tokens (XVT) → Voting (ANT)

1. The *Tokens (XVT)* app will forward actions requested by XVT token holders.
2. The *Voting (XVT)* app will create a new voting for ANT holders to decide wether to perform this action or not.

Action to be triggered

This transaction would eventually perform a payment to address 0x52b...1ka11 Estimated payment cost ?

Sign Transaction

Escalating requests to other modules:

```
interface IForwarder {
    function isForwarder() public pure returns (bool);
    function canForward(address sender, bytes evmCallScript) public view returns (bool);
    function forward(bytes evmCallScript) public;
}
```

Interface: https://github.com/aragon/aragonOS/blob/dev/contracts/common/IForwarder.sol

Example forwarder:

```
contract Voting is IForwarder, AragonApp {
    function isForwarder() public pure returns (bool) {
        return true;
    }

    function canForward(address _sender, bytes _evmCallScript) public view returns (bool) {
        return auth(_sender, CREATE_VOTES_ROLE);
    }

    /**
    * @notice Creates a vote to execute the desired action, and casts a support vote
    * @dev IForwarder interface conformance
    * @param evmScript Start vote with script
    */
    function forward(bytes evmScript) public {
        require(canForward(msg.sender, _evmScript));

        // Boom 💥
        newVote(evmScript);
    }
}
```

Implementation: https://github.com/aragon/aragon-apps/blob/master/apps/voting/contracts/Voting.sol

EVM Scripts

Oh boy

Interface:

```
interface IEVMScriptExecutor {
    function execScript(bytes script, bytes input, address[] blacklist)
        external
        returns (bytes);
}
```

Interface: https://github.com/aragon/aragonOS/blob/dev/contracts/evmscript/IEVMScriptExecutor.sol

CallsScript:

```
contract CallsScript is BaseEVMScriptExecutor {

    /**
    * @notice Executes a number of call scripts
    * @param _script [ specId (uint32) ] many calls with this structure ->
    *    [ to (address: 20 bytes) ] [ calldataLength (uint32: 4 bytes) ] [ calldata (calldataLength bytes) ]
    * @param _blacklist Addresses the script cannot call to, or will revert.
    * @return always returns empty byte array
    */
    function execScript(bytes _script, bytes, address[] _blacklist)
        external
        returns (bytes)
    {
        uint256 location = 4; // first 32 bits are spec id

        while (location < _script.length) {
            address contractAddress = _script.addressAt(location);

            // Check address being called is not blacklist
            for (uint i = 0; i < _blacklist.length; i++) {
```

Implementation: https://github.com/aragon/aragonOS/blob/dev/contracts/evmscript/IEVMScriptExecutor.sol

CallsScript:

```
function execScript(bytes _script, bytes, address[] _blacklist)
    external
    returns (bytes)
{
    uint256 location = 4; // first 32 bits are spec id

    while (location < _script.length) {
        address contractAddress = _script.addressAt(location);

        // Check address being called is not blacklist
        for (uint i = 0; i < _blacklist.length; i++) {
            require(contractAddress != _blacklist[i]);
        }

        uint256 calldataLength = uint256(_script.uint32At(location + 0x14));
        uint256 startOffset = location + 0x14 + 0x04;
        uint256 calldataStart = _script.locationOf(startOffset);

        // compute end of script / next location
        location = startOffset + calldataLength;
```

Implementation: https://github.com/aragon/aragonOS/blob/dev/contracts/evmscript/IEVMScriptExecutor.sol

CallsScript:

```
            require(contractAddress != _blacklist[i]);
        }

        uint256 calldataLength = uint256(_script.uint32At(location + 0x14));
        uint256 startOffset = location + 0x14 + 0x04;
        uint256 calldataStart = _script.locationOf(startOffset);

        // compute end of script / next location
        location = startOffset + calldataLength;
        require(location <= _script.length);

        assembly {
            let success := call(sub(gas, 5000),contractAddress, 0, calldataStart, calldataLength, 0, 0)

            switch success case 0 { revert(0, 0) }
        }
    }
}
}
```

Implementation: https://github.com/aragon/aragonOS/blob/dev/contracts/evmscript/IEVMScriptExecutor.sol

Running a script: runScript()

```
contract EVMScriptRunner {

    function runScript(bytes _script, bytes _input, address[] _blacklist)
        internal
        returns (bytes)
    {
        // Get the specified executor in the script
        IEVMScriptExecutor executor = getExecutor(_script);
        require(address(executor) != address(0));

        bytes4 sig = executor.execScript.selector;

        require(address(executor).delegatecall(abi.encodeWithSelector(sig, _script, _input, _blacklist)));

        // Get return data
        bytes memory output = returnedDataDecoded();

        return output;
    }
}
```

Example: executing votes

```
contract Voting is AragonApp {

    function _executeVote(uint256 _voteId) internal {
        Vote storage vote_ = votes[_voteId];

        vote_.executed = true;

        // Boom 💥 - Run script without input or blacklist
        runScript(vote_.executionScript, new bytes(0);, new address[](0));
    }

}
```

Implementation: https://github.com/aragon/aragon-apps/blob/master/apps/voting/contracts/Voting.sol

Here be dragons…

It’s a big day!

EVERDRAGONS

Delegate Script Executors

Add new EVMScript types (DelegateScript and DeployDelegateScript) #182

Edit

Merged as c76fac8 **sohkai** merged 38 commits into `dev` from `scripts` on Jan 24 · Jump to bottom

Conversation 34 | Commits 38 | Checks 0 | Files changed 50 | +1,596 −610

izqui (Jorge Izquierdo) on Jan 11 • edited Member

Close #166 (as it allows turing complete EVMScripts, by using the EVM by delegatecall'ing)

This PR introduces the notion of EVMScript specs, which will allow us to have more than just one type of EVM scripts when forwarding actions. This will also allow for gracefully upgrading script types. There is a limit of 4,294,967,296 script types, which I think will be fine for a couple centuries.

Pipeline

Closed

Reviewers

sohkai

onbjerg

Add new EVMScript types (DelegateScript and DeployDelegateScript) #182

Edit

Merged as c76fac8 **sohkai** merged 38 commits into `dev` from `scripts` on Jan 24 · Jump to bottom

Conversation 34 | Commits 38 | Checks 0 | Files changed 50 | +1,596 −610

izqui (Jorge Izquierdo) on Jan 11 • edited — Member

Close #166 (as it allows turing complete EVMScripts, by using the EVM by delegatecall'ing)

This PR introduces the notion of EVMScript specs, which will allow us to have more than just one type of EVM scripts when forwarding actions. This will also allow for gracefully upgrading script types. There is a limit of 4,294,967,296 script types, which I think will be fine for a couple centuries.

Pipeline

Closed

Reviewers

sohkai

onbjerg

sohkai on Jan 16 • edited — Member

🎉 I think this has the potential to do lots of crazy stuff (in a good way)!

DelegateScript:

```
contract DelegateScript is IEVMScriptExecutor {
    uint256 constant internal SCRIPT_START_LOCATION = 4;

    /**
    * @notice Executes script by delegatecall into a contract
    * @param _script [ specId (uint32) ][ contract address (20 bytes) ]
    * @param _input ABI encoded call to be made to contract (if empty executes default exec() function)
    * @param _blacklist If any address is passed, will revert.
    * @return Call return data
    */
    function execScript(bytes _script, bytes _input, address[] _blacklist) external returns (bytes) {
        require(_blacklist.length == 0); // dont have ability to control bans, so fail.

        // Script should be spec id + address (20 bytes)
        require(_script.length == SCRIPT_START_LOCATION + 20);
        return delegate(_script.addressAt(SCRIPT_START_LOCATION), _input);
    }

    /**
    * @dev Delegatecall to contract with input data
    */
```

DelegateScript:

```
    /**
    * @dev Delegatecall to contract with input data
    */
    function delegate(address _addr, bytes memory _input) internal returns (bytes memory output) {
        require(isContract(_addr));
        require(_addr.delegatecall(_input.length > 0 ? _input : defaultInput()));

        // Get returned data
        bytes memory ret = returnedData();

        require(ret.length > 0);

        return ret;
    }

    function defaultInput() internal pure returns (bytes) {
        return DelegateScriptTarget(0).exec.selector.toBytes();
    }
}
```

DeployDelegateScript:

```
contract DeployDelegateScript is DelegateScript {
    uint256 constant internal SCRIPT_START_LOCATION = 4;

    uint256[2**254] private cacheStorageOffset;
    mapping (bytes32 => address) cache;

    /**
    * @notice Executes script by delegatecall into a deployed contract (exec() function)
    * @param _script [ specId (uint32) ][ contractInitcode (bytecode) ]
    * @param _input ABI encoded call to be made to contract (if empty executes default exec() function)
    * @param _blacklist If any address is passed, will revert.
    * @return Call return data
    */
    function execScript(bytes _script, bytes _input, address[] _blacklist) external returns (bytes) {
        require(_blacklist.length == 0); // dont have ability to control bans, so fail.

        bytes32 id = keccak256(_script);
        address deployed = cache[id];
        if (deployed == address(0)) {
            deployed = deploy(_script);
            cache[id] = deployed;
```

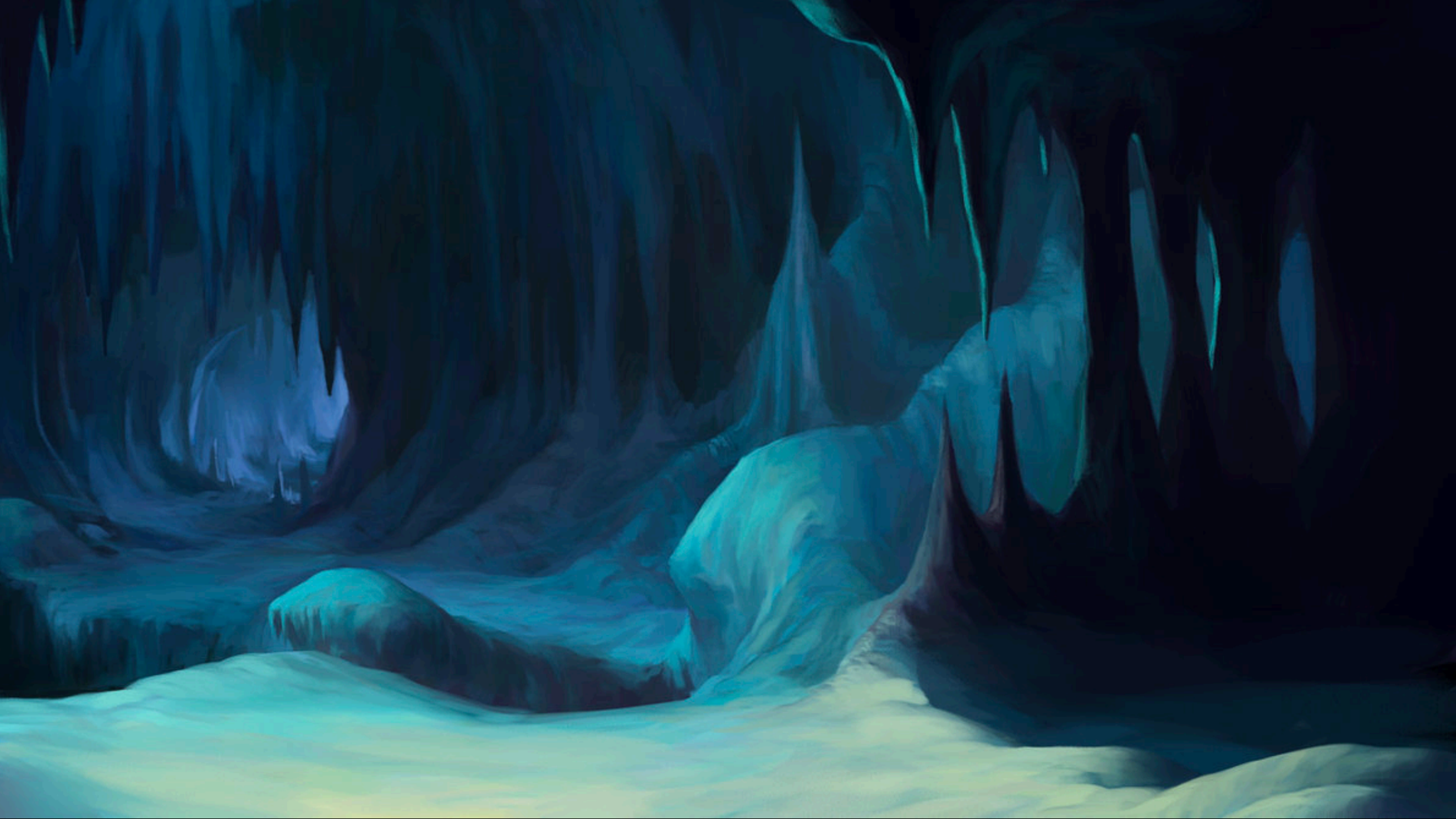
DeployDelegateScript:

```
/**
* @notice Executes script by delegatecall into a deployed contract (exec() function)
* @param _script [ specId (uint32) ][ contractInitcode (bytecode) ]
* @param _input ABI encoded call to be made to contract (if empty executes default exec() function)
* @param _blacklist If any address is passed, will revert.
* @return Call return data
*/
function execScript(bytes _script, bytes _input, address[] _blacklist) external returns (bytes) {
    require(_blacklist.length == 0); // dont have ability to control bans, so fail.

    bytes32 id = keccak256(_script);
    address deployed = cache[id];
    if (deployed == address(0)) {
        deployed = deploy(_script);
        cache[id] = deployed;
    }

    return DelegateScript.delegate(deployed, _input);
}

/**
```

DeployDelegateScript:

```
    address deployed = cache[id];
    if (deployed == address(0)) {
        deployed = deploy(_script);
        cache[id] = deployed;
    }

    return DelegateScript.delegate(deployed, _input);
}

/**
* @dev Deploys contract byte code to network
*/
function deploy(bytes _script) internal returns (address addr) {
    assembly {
        // 0x24 = 0x20 (length) + 0x04 (spec id uint32)
        // Length of code is 4 bytes less than total script size
        addr := create(0, add(_script, 0x24), sub(mload(_script), 0x04))
        switch iszero(extcodesize(addr))
        case 1 { revert(0, 0) } // throw if contract failed to deploy
    }
}
}
```

Let’s explore!



LIVE
bitconnect

LIVE
bitconnect

aragon / aragonOS

Unwatch 48 | Unstar 326 | Fork 118

Code | Issues 37 | Pull requests 8 | ZenHub | Wiki | Releases 35 | More

<Deleted> #218

Edit | New issue

Closed **sohkai** opened this issue on Feb 5 · 0 comments · Jump to bottom

sohkai on Feb 5 • edited

Member

No description provided.

<Deleted> is **closed**

+ add dependency

izqui (Jorge Izquierdo) added the **audit: whg** label on Feb 5

sohkai changed the title from **DelegateScript can be selfdestructed** to **<Deleted>** on Feb 5

sohkai closed this on Feb 5

Pipeline

Closed

Assignees

No one—assign yourself

Labels

audit: whg

Projects

None yet

Milestone

No milestone

Write Preview

From **@adriamb**:

> Seems that there's only one DelegateScript instance globally created in EVMScriptRegistryFactory. This DelegateScript has the public everybody callable method execScript(bytes _script, bytes _input, address[] _blacklist) . If the delegatecalled contract method contains a selfdestruct, the destructed contract will be the EVMScriptRegistryFactory baseDel instance , that cannot be redeployed since it's not upgreadable.

```
pragma solidity ^0.4.18;

interface DelegateScriptTarget {
    function exec() public;
}

contract DelegateScript{
    function execScript(address _addr) external{
        delegate(_addr);
    }
    function delegate(address _addr) internal {
        require(_addr.delegatecall(DelegateScriptTarget(0).exec.selector));
    }
}

contract DelegateScriptTargetImpl is DelegateScriptTarget {
    event DelegateScriptTargetImplCall();
    function exec() public {
        DelegateScriptTargetImplCall();
        selfdestruct(0x0);
    }
}

contract AATest {
```

```
interface DelegateScriptTarget {
    function exec() public;
}

contract DelegateScript{
    function execScript(address _addr) external{
        delegate(_addr);
    }
    function delegate(address _addr) internal {
        require(_addr.delegatecall(DelegateScriptTarget(0).exec.selector));
    }
}

contract DelegateScriptTargetImpl is DelegateScriptTarget {
    event DelegateScriptTargetImplCall();
    function exec() public {
        DelegateScriptTargetImplCall();
        selfdestruct(0x0);
    }
}

contract AATest {

    event LogNotAContract(address _addr);
    DelegateScriptTargetImpl impl;
    DelegateScript ds;

    function AATest() public {
        impl = new DelegateScriptTargetImpl();
        ds = new DelegateScript();
    }

    function isContract(address _target) internal view returns (bool) {
        uint256 size;
        assembly { size := extcodesize(_target) }
        return size > 0;
    }

    function t() public {
        if (!isContract(ds)) {
```

```
contract DelegateScriptTargetImpl is DelegateScriptTarget {
    event DelegateScriptTargetImplCall();
    function exec() public {
        DelegateScriptTargetImplCall();
        selfdestruct(0x0);
    }
}

contract AATest {

    event LogNotAContract(address _addr);
    DelegateScriptTargetImpl impl;
    DelegateScript ds;

    function AATest() public {
        impl = new DelegateScriptTargetImpl();
        ds = new DelegateScript();
    }

    function isContract(address _target) internal view returns (bool) {
        uint256 size;
        assembly { size := extcodesize(_target) }
        return size > 0;
    }

    function t() public {
        if (!isContract(ds)) {
            LogNotAContract(ds);
            return;
        }
        ds.execScript(impl);

    }
}
```

Calling `AATest.t()` twice will log `LogNotAContract`.

Why is this possible?

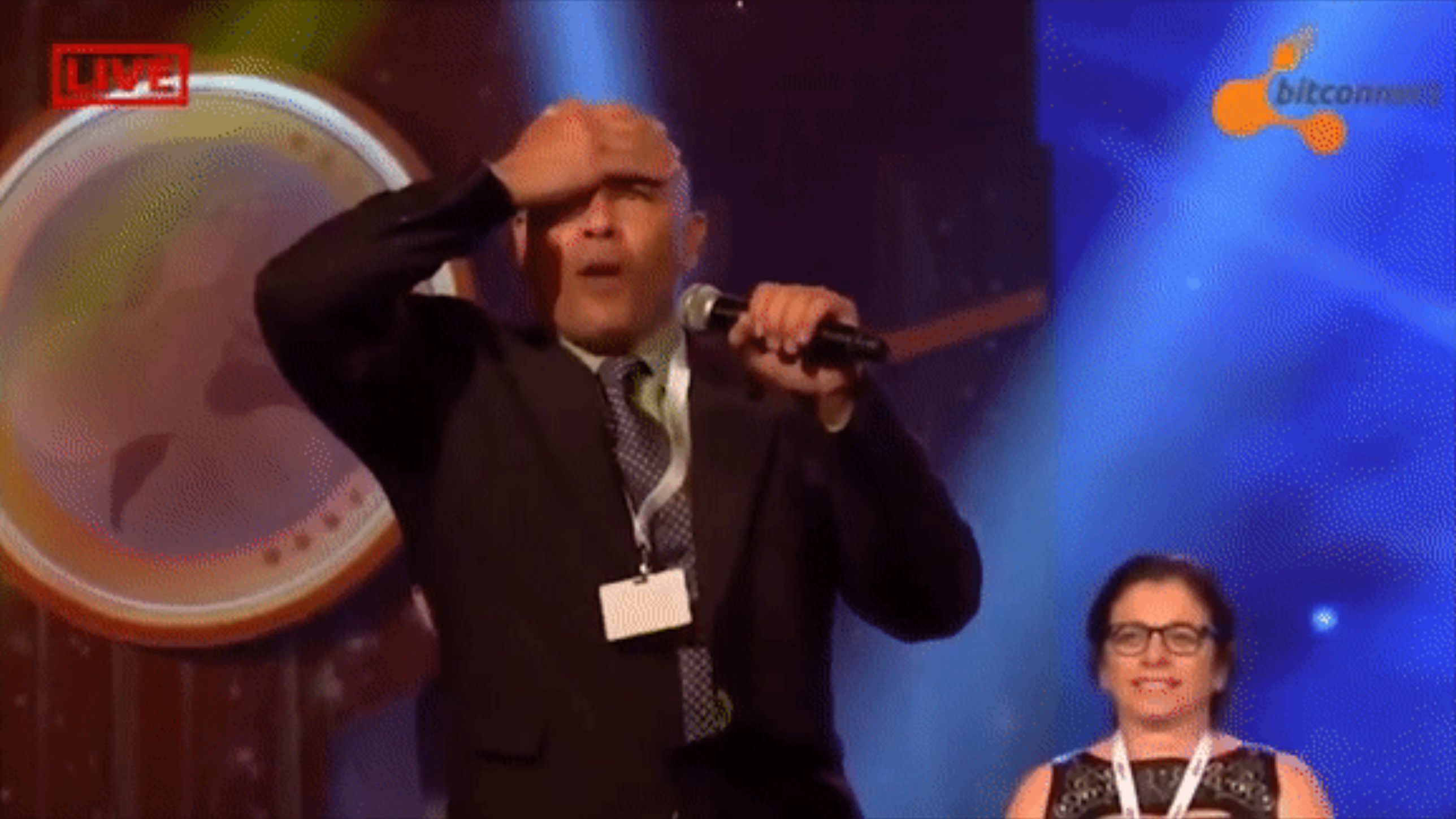
```
contract DelegateScript is IEVMScriptExecutor {
    uint256 constant internal SCRIPT_START_LOCATION = 4;

    /**
    * @notice Executes script by delegatecall into a contract
    * @param _script [ specId (uint32) ][ contract address (20 bytes) ]
    * @param _input ABI encoded call to be made to contract (if empty executes default exec() function)
    * @param _blacklist If any address is passed, will revert.
    * @return Call return data
    */
    function execScript(bytes _script, bytes _input, address[] _blacklist) external returns (bytes) {
        require(_blacklist.length == 0); // dont have ability to control bans, so fail.

        // Script should be spec id + address (20 bytes)
        require(_script.length == SCRIPT_START_LOCATION + 20);
        return delegate(_script.addressAt(SCRIPT_START_LOCATION), _input);
    }

    /**
    * @dev Delegatecall to contract with input data
    */
```

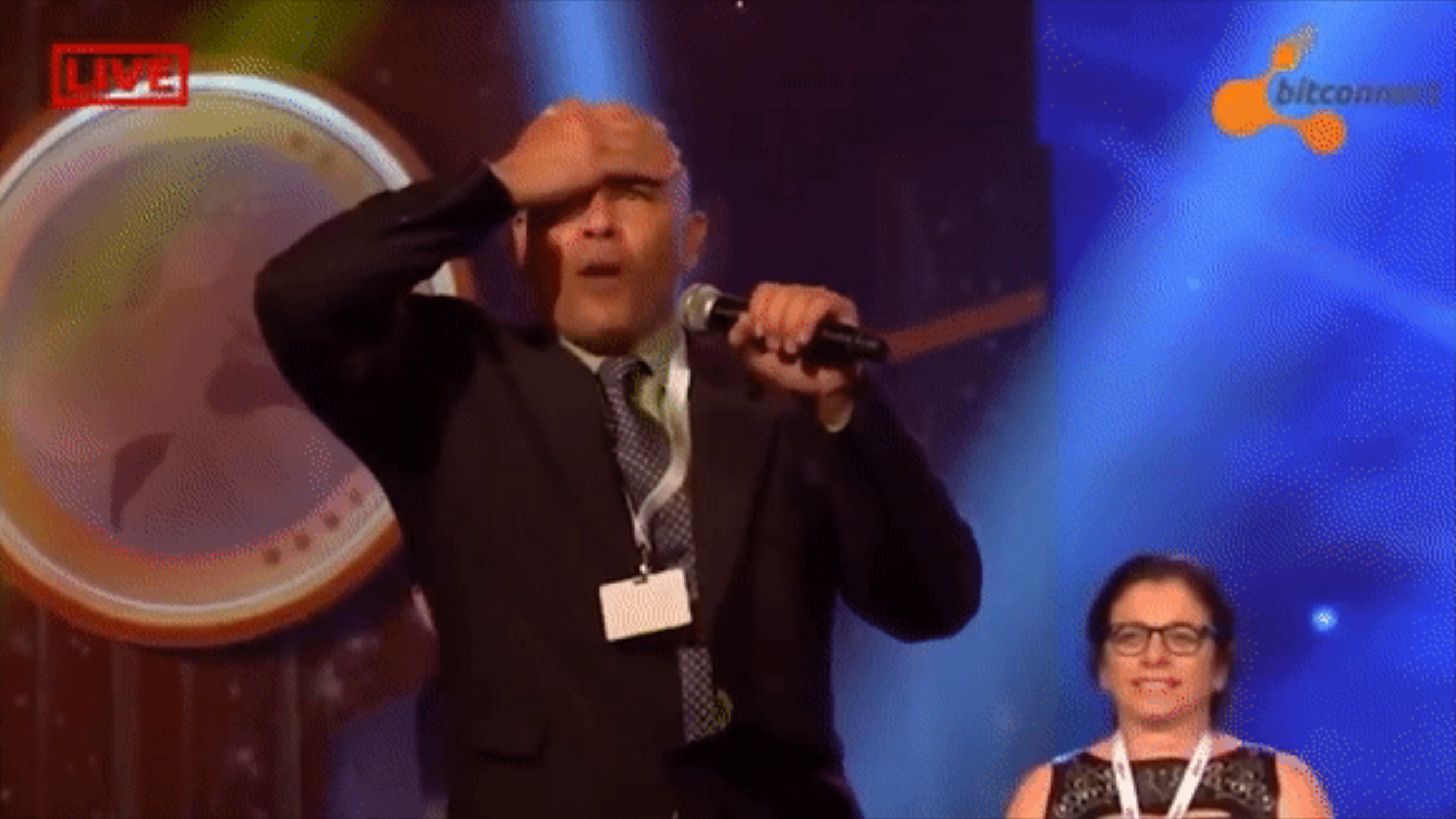
There’s another delegate call!

```
contract EVMScriptRunner {

    function runScript(bytes _script, bytes _input, address[] _blacklist)
        internal
        returns (bytes)
    {
        // Get the specified executor in the script
        IEVMScriptExecutor executor = getExecutor(_script);
        require(address(executor) != address(0));

        bytes4 sig = executor.execScript.selector;

        require(address(executor).delegatecall(abi.encodeWithSelector(sig, _script, _input, _blacklist)));

        // Get return data
        bytes memory output = returnedDataDecoded();

        return output;
    }
}
```

Bro, just tl;dr me

- Anyone could execute execScript()
- We allowed the caller to choose which contract to delegatecall into
- That delegated contract could do *anything*


LIVE
bitconnect


LIVE
bitconnect

Write Preview

From **@adriamb**:

> Seems that there's only one DelegateScript instance globally created in EVMScriptRegistryFactory. This DelegateScript has the public everybody callable method execScript(bytes _script, bytes _input, address[] _blacklist) . If the delegatecalled contract method contains a selfdestruct, the destructed contract will be the EVMScriptRegistryFactory baseDel instance , that cannot be redeployed since it's not upgreadable.

```
pragma solidity ^0.4.18;

interface DelegateScriptTarget {
    function exec() public;
}

contract DelegateScript{
    function execScript(address _addr) external{
        delegate(_addr);
    }
    function delegate(address _addr) internal {
        require(_addr.delegatecall(DelegateScriptTarget(0).exec.selector));
    }
}

contract DelegateScriptTargetImpl is DelegateScriptTarget {
    event DelegateScriptTargetImplCall();
    function exec() public {
        DelegateScriptTargetImplCall();
        selfdestruct(0x0);
    }
}

contract AATest {
```

Proposed solution:

Script executors that `delegatecall` must have their return values checked. Any script being called must conform to a standardized return data format of:

```
[ success (bool) ][ data ]
```

If the `returndatasize` is 0 (this is what happens in the case if the delegated contract selfdestructs) or if `success` is false, then we revert the transaction.

Example contracts showing this check:

```
contract SelfDestruct {
    function d() returns (uint) {
        selfdestruct(this);
        return 1;
    }
}

contract NoDestruct {
    function d() returns (uint) {
        return 1;
    }
}

contract Delegate {
    function delegate(address c) returns (uint) {
        bytes4 sig = NoDestruct(c).d.selector;
        c.delegatecall(sig, new bytes(0x0));

        uint size;
        assembly {
            size := returndatasize
            switch size case 0 { revert(0, 0) }
        }
        return size;
```

```
contract SelfDestruct {
    function d() returns (uint) {
        selfdestruct(this);
        return 1;
    }
}

contract NoDestruct {
    function d() returns (uint) {
        return 1;
    }
}

contract Delegate {
    function delegate(address c) returns (uint) {
        bytes4 sig = NoDestruct(c).d.selector;
        c.delegatecall(sig, new bytes(0x0));

        uint size;
        assembly {
            size := returndatasize
            switch size case 0 { revert(0, 0) }
        }
        return size;
    }
}

contract DelegateRunner {
    Delegate d;
    function DelegateRunner() {
        d = new Delegate();
    }
    function runDelegate(address c) returns (uint) {
        return d.delegate(c);
    }
    function checkDelegate() view returns (bool) {
        address delegate = d;
        uint256 size;
        assembly { size := extcodesize(delegate) }
        return size > 0;
```

```
        return 1;
    }
}

contract Delegate {
    function delegate(address c) returns (uint) {
        bytes4 sig = NoDestruct(c).d.selector;
        c.delegatecall(sig, new bytes(0x0));

        uint size;
        assembly {
            size := returndatasize
            switch size case 0 { revert(0, 0) }
        }
        return size;
    }
}

contract DelegateRunner {
    Delegate d;
    function DelegateRunner() {
        d = new Delegate();
    }
    function runDelegate(address c) returns (uint) {
        return d.delegate(c);
    }
    function checkDelegate() view returns (bool) {
        address delegate = d;
        uint256 size;
        assembly { size := extcodesize(delegate) }
        return size > 0;
    }
}
```

Running `DelegateRunner.runDelegate()` on an instance of `NoDestruct` will return 32 (`uint` size) while running it on `SelfDestruct` will revert.

Bro, just tl;dr me

- Require all callees to return at least a byte of information, to make sure their execution context wasn’t destroyed

aragon / aragonOS

Unwatch 48 | Unstar 326 | Fork 118

Code | Issues 37 | Pull requests 8 | ZenHub | Wiki | Insights

Require minimum return data from EVMScript executors to prevent selfdestructs #255

Edit

Merged izqui merged 2 commits into `dev` from `evmscript-min-return` on Mar 27

Conversation 7 | Commits 2 | Checks 0 | Files changed 6

+75 −11

izqui commented on Mar 25 Member

#250, Prevent EVMScript runners from self destructing apps

Close #250

Require minimum return data from EVMScript executors to prevent self-… ... dede447

Reviewers

sohkai

Assignees

No one—assign yourself

Require minimum return data from EVMScript executors to prevent selfdestructs #255

Edit

Merged izqui merged 2 commits into `dev` from `evmscript-min-return` on Mar 27

Conversation 7 | Commits 2 | Checks 0 | Files changed 6

all commits | Jump to... | +75 −11

Diff settings | No Whitespace | Review changes

8 contracts/evmscript/EVMScriptRunner.sol

Copy path | View

```
@@ -20,7 +20,11 @@ contract EVMScriptRunner is AppStorage, EVMScriptRegistryConstants {
20  20
21  21          require(executorAddr.delegatecall(sig, calldataArgs));
22  22
23      -       return returnedDataDecoded();
    23  +       bytes memory ret = returnedDataDecoded();
    24  +
    25  +       require(ret.length > 0);
    26  +
    27  +       return ret;
24  28      }
25  29
26  30      function getExecutor(bytes _script) public view returns (IEVMScriptExecutor) {
@@ -57,4 +61,4 @@ contract EVMScriptRunner is AppStorage, EVMScriptRegistryConstants {
57  61          require(kernel == preKernel);
58  62          require(appId == preAppId);
59  63      }
60      - }
    64  + }
```

5 contracts/evmscript/executors/CallsScript.sol

Copy path | View

```diff
@@ -47,5 +47,10 @@ contract CallsScript is IEVMScriptExecutor {
                    switch success case 0 { revert(0, 0) }
                }
            }
+
+           bytes memory ret = new bytes(1);
+           ret[0] = 1;
+
+           return ret;
        }
    }
```

10 contracts/evmscript/executors/DelegateScript.sol

Copy path | View

```diff
@@ -5,7 +5,7 @@ import "../IEVMScriptExecutor.sol";

 interface DelegateScriptTarget {
-    function exec() public;
+    function exec() public returns (bool);
 }


@@ -35,7 +35,11 @@ contract DelegateScript is IEVMScriptExecutor {
     function delegate(address _addr, bytes memory _input) internal returns (bytes memory output) {
         require(isContract(_addr));
         require(_addr.delegatecall(_input.length > 0 ? _input : defaultInput()));
-        return returnedData();
+        bytes memory ret = returnedData();
+
+        require(ret.length > 0);
+
+        return ret;
     }

     function isContract(address _target) internal view returns (bool) {
@@ -61,4 +65,4 @@ contract DelegateScript is IEVMScriptExecutor {
         }
         return ret;
     }
- }
+ }
```

izqui / fucked.sol Secret

Created 4 months ago • Report gist

Star 0 Fork 0

Code Revisions 1

Embed <script src="https://gi Download ZIP

fucked.sol

Copy Raw

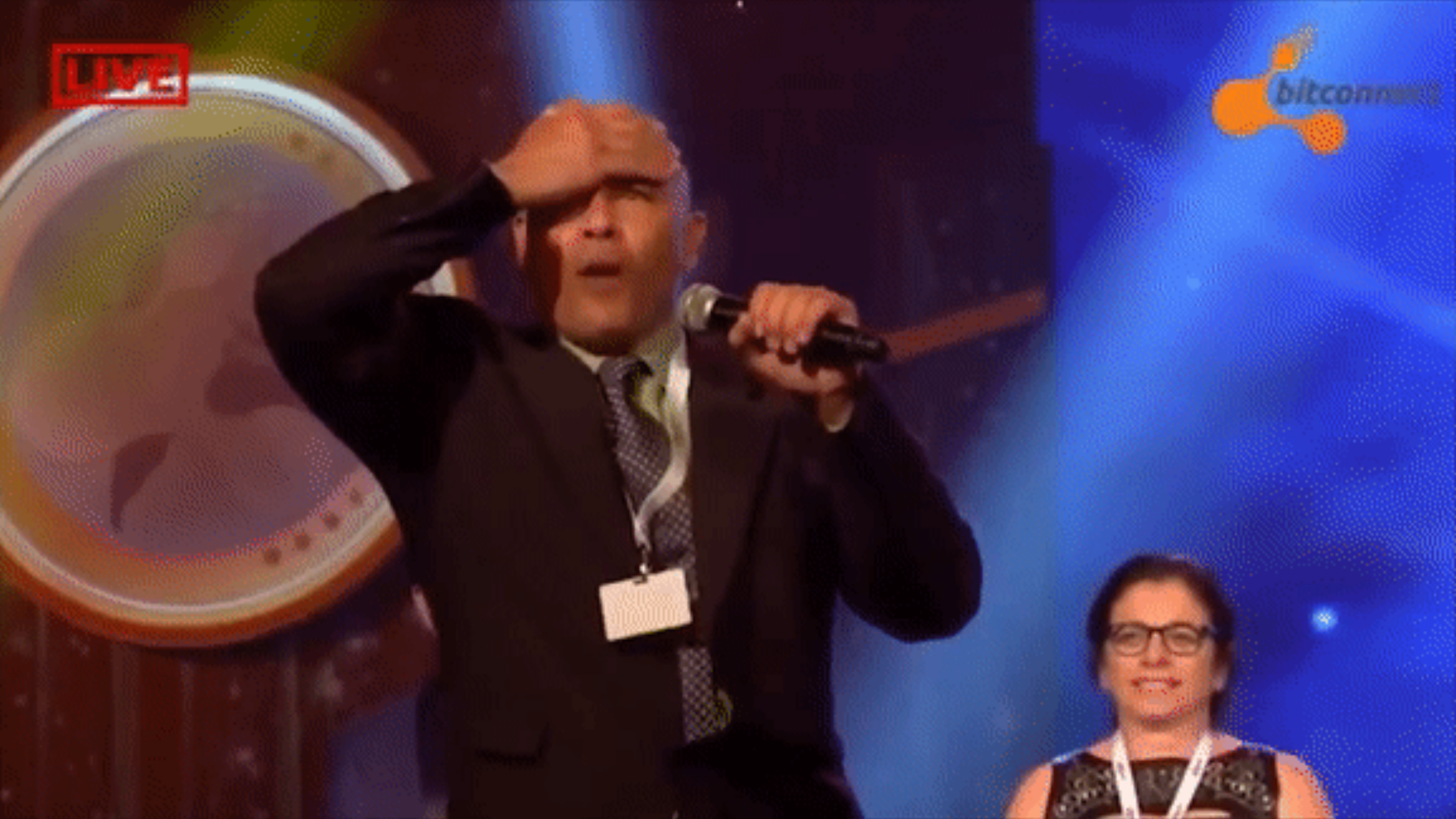
```
contract AragonApp {
    ScriptRunner runner = new ScriptRunner();
    MaliciousScript script = new MaliciousScript();

    function runMaliciousScript() {
        runScript(script);
    }

    function runScript(address script) {
        require(address(runner).delegatecall(runner.run.selector, script));
    }
}

contract ScriptRunner {
    function run(address script) {
        require(script.delegatecall(IScript(0).run.selector));

        uint256 size;
        assembly {
            size := returndatasize
        }
```

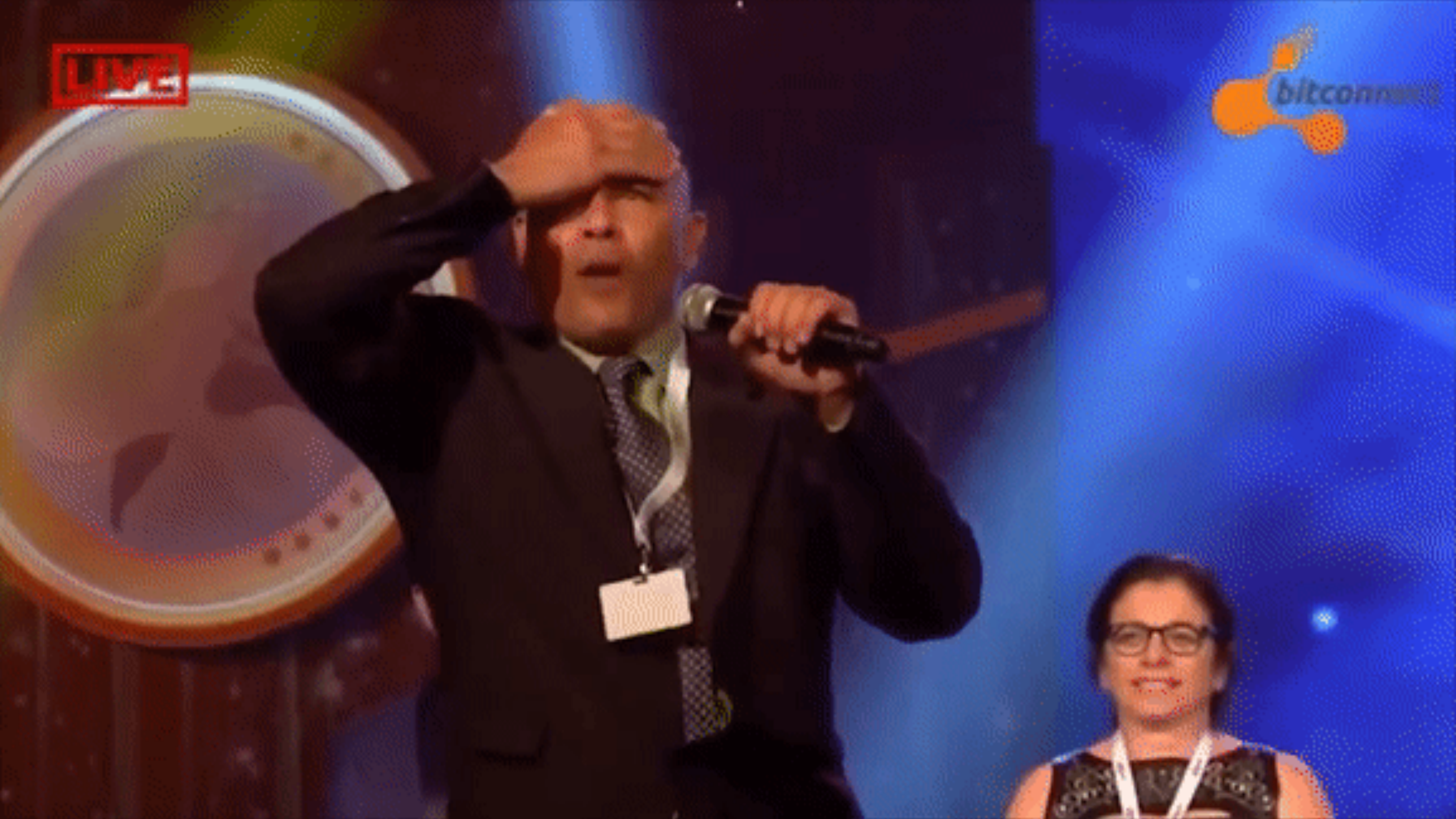
```
contract AragonApp {
    ScriptRunner runner = new ScriptRunner();
    MaliciousScript script = new MaliciousScript();

    function runMaliciousScript() {
        runScript(script);
    }

    function runScript(address script) {
        require(address(runner).delegatecall(runner.run.selector, script));
    }
}

contract ScriptRunner {
    function run(address script) {
        require(script.delegatecall(IScript(0).run.selector));

        uint256 size;
        assembly {
            size := returndatasize
        }

        require(size > 0);
    }
}

contract IScript {
    function run() returns (bool);
}

contract MaliciousScript is IScript {
    function run() returns (bool) {
      address(this).delegatecall(this.secondLayerKill.selector);
      return true;
```

```
    function runScript(address script) {
        require(address(runner).delegatecall(runner.run.selector, script));
    }
}

contract ScriptRunner {
    function run(address script) {
        require(script.delegatecall(IScript(0).run.selector));

        uint256 size;
        assembly {
            size := returndatasize
        }

        require(size > 0);
    }
}

contract IScript {
    function run() returns (bool);
}

contract MaliciousScript is IScript {
    function run() returns (bool) {
      address(this).delegatecall(this.secondLayerKill.selector);
      return true;
    }

    function secondLayerKill() {
        selfdestruct(this);
    }
}
```

Bro, just tl;dr me

- We can’t catch cases if a contract recursively delegates (delegatecall into a second delegatecall)


LIVE
bitconnect


LIVE
bitconnect

Arachnid / **OpcodeChecker.sol**

Last active a month ago • Report gist

Star 7 Fork 1

Code Revisions 3 Stars 7 Forks 1

Embed <script src="https://gi Download ZIP

OpcodeChecker.sol

Copy Raw

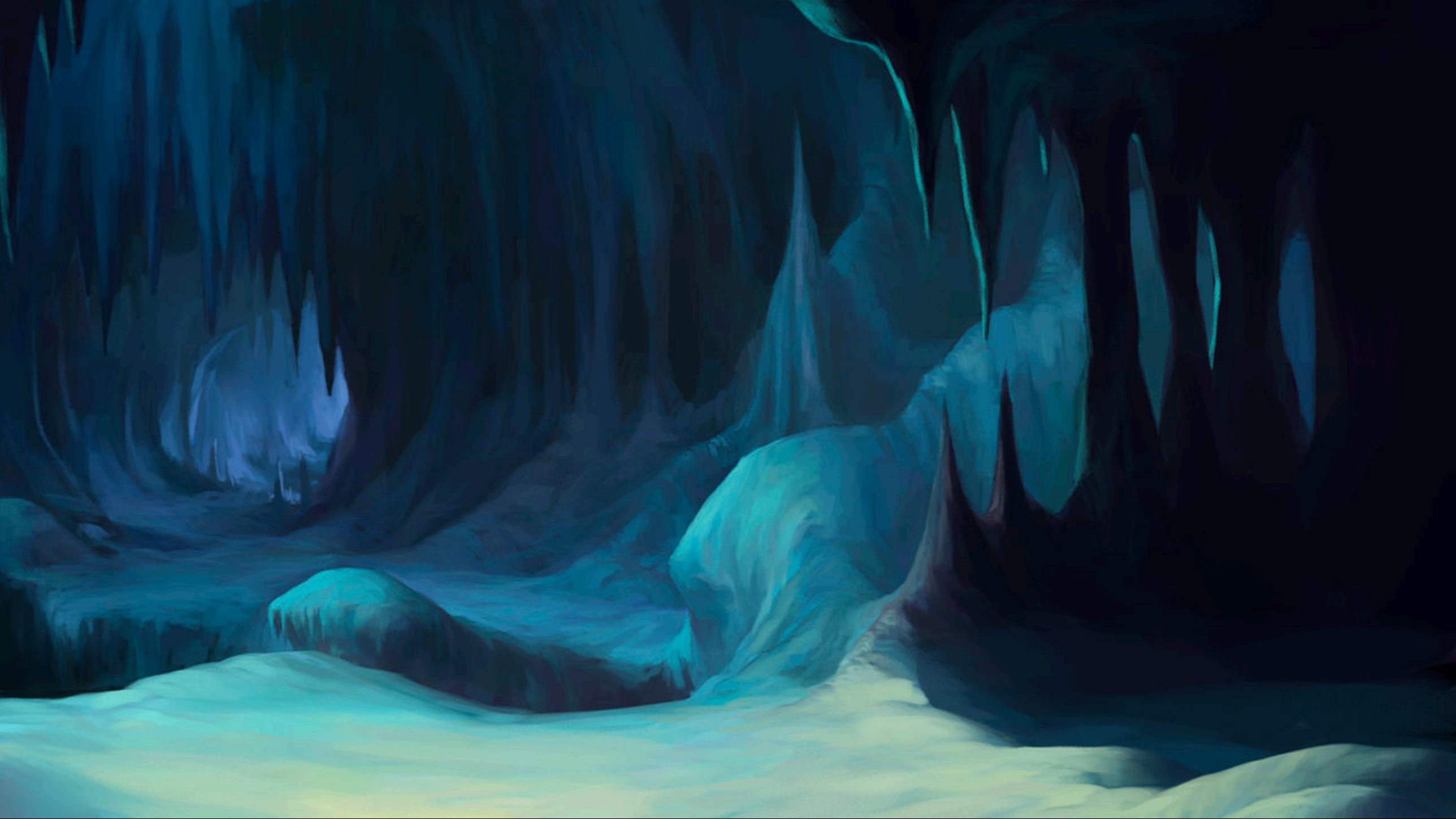
```
/**
 * @dev OpcodeChecker processes contract code to generate a bitmap of used opcodes.
 *
 * DO NOT USE: See the vulnerability identified by Recmo below. A patch will be provided soon.
 *
 * The generated bitmap can be used to enforce whitelisting and blacklisting on contract code.
 * Bit n of the bitmap is set iff opcode n is used. For instance, the presence of the STOP opcode
 * will result in bit 0 of the bitmap being set.
 *
 * A best-effort attempt is made to skip over unreachable data, but there may be false positives.
 * To the extent the checker is written correctly, there are no false negatives.
 *
 * To check a contract against a blacklist, construct a mask by ORing the values of the opcodes you
 * want to prohibit - for instance, if you want to prohibit SELFDESTRUCT (0xFF) CALLCODE (0xF2) and
 * DELEGATECALL (0xF4), your mask should be
 > `(1 << OP_SELFDESTRUCT) | (1 << OP_CALLCODE) | (1 << OP_DELEGATECALL)`.
 */
 contract OpcodeChecker {
    uint8 constant OP_STOP = 0x00;
    uint8 constant OP_PUSH1 = 0x60;
    uint8 constant OP_PUSH32 = 0x7F;
    uint8 constant OP_JUMP = 0x56;
    uint8 constant OP_JUMPDEST = 0x5B;
    uint8 constant OP_RETURN = 0xF3;
    uint8 constant OP_SELFDESTRUCT = 0xFF;
```

OpcodeChecker.sol

Copy Raw

```
/**
 * @dev OpcodeChecker processes contract code to generate a bitmap of used opcodes.
 *
 * DO NOT USE: See the vulnerability identified by Recmo below. A patch will be provided soon.
 *
 * The generated bitmap can be used to enforce whitelisting and blacklisting on contract code.
 * Bit n of the bitmap is set iff opcode n is used. For instance, the presence of the STOP opcode
 * will result in bit 0 of the bitmap being set.
 *
 * A best-effort attempt is made to skip over unreachable data, but there may be false positives.
 * To the extent the checker is written correctly, there are no false negatives.
 *
 * To check a contract against a blacklist, construct a mask by ORing the values of the opcodes you
 * want to prohibit - for instance, if you want to prohibit SELFDESTRUCT (0xFF) CALLCODE (0xF2) and
 * DELEGATECALL (0xF4), your mask should be
 > `(1 << OP_SELFDESTRUCT) | (1 << OP_CALLCODE) | (1 << OP_DELEGATECALL)`.
 */
 contract OpcodeChecker {
    uint8 constant OP_STOP = 0x00;
    uint8 constant OP_PUSH1 = 0x60;
    uint8 constant OP_PUSH32 = 0x7F;
    uint8 constant OP_JUMP = 0x56;
    uint8 constant OP_JUMPDEST = 0x5B;
    uint8 constant OP_RETURN = 0xF3;
    uint8 constant OP_SELFDESTRUCT = 0xFF;

    bytes32 constant UNREACHABLE_MASK =
        (bytes32(1) << OP_STOP) |
        (bytes32(1) << OP_JUMP) |
        (bytes32(1) << OP_RETURN) |
        (bytes32(1) << OP_SELFDESTRUCT);

    mapping(bytes32=>bytes32) public opcodesUsed;
    mapping(address=>bytes32) public bytecodeHash;

    function getCode(address target) internal view returns(bytes memory) {
```

```
    function getCode(address target) internal view returns(bytes memory) {
        uint codelen;
        assembly { codelen := extcodesize(target) }
        bytes memory ret = new bytes(codelen);
        assembly {
            extcodecopy(target, add(ret, 32), 0, codelen)
        }
        return ret;
    }

    function process(address target) public returns (bytes32 bitmap) {
        if(bytecodeHash[target] != 0) return opcodesUsed[bytecodeHash[target]];

        bytes memory code = getCode(target);
        bytes32 codeHash = keccak256(code);
        bytecodeHash[target] = codeHash;

        bitmap = analyse(code);
        opcodesUsed[codeHash] = bitmap;
    }

    function analyse(bytes code) internal pure returns (bytes32 bitmap) {
        bool reachable = true;
        for(uint pc = 0; pc < code.length; pc++) {
            uint8 opcode = uint8(code[pc]);
            if(reachable) {
                bytes32 bit = bytes32(1) << opcode;
                bitmap |= bit;

                if(opcode >= 0x60 && opcode <= 0x7F) {
                    pc += opcode - 0x5F;
                    continue;
                } else if(reachable && ((bit & UNREACHABLE_MASK) != 0)) {
                    reachable = false;
                }
            } else if(opcode == OP_JUMPDEST) {
                reachable = true;
            }
```



aragon / aragonOS

Unwatch 48 | Unstar 326 | Fork 118

Code | Issues 37 | Pull requests 8 | ZenHub | Wiki | Insights

Remove delegate executors #324

Edit

Merged **sohkai** merged 3 commits into `dev` from `remove-delegate-executors` on Jun 6

Conversation 3 | Commits 3 | Checks 0 | Files changed 13 | +27 −454

sohkai commented on May 25 • edited

Member

Also removes related tests and size checks we added to the proxies in an attempt to make these executors safer.

Reviewers

izqui ✓

bingen ✓

Lesson time!

- Don’t use delegatecalls
- Protect your delegatecalls
- Don’t allow a caller determine the delegation location
- If you really want the BFG, you can probably find it

hack.aragon.org

Aragon Developer Portal

Get started | Tutorial | aragonOS | aragon.js | Aragon UI | Help | Search

Aragon Developer Portal

What you need to get started building with Aragon

Get started

Take the tutorial >

Thank you!

@sohkai