

Smart Contract Security / Solidity Security


Peter Robinson
August 9, 2022



Thank you for your help!

Jordan Coppard, Franck Cassez, Ermyas Abebe, Horacio Mijail Anton Quiles, Adrian Sutton

and Anton Atanasov for the starting point of the forest photo: <https://www.pexels.com/photo/landscape-photo-of-forest-1655901/>

A dark, atmospheric photograph of a forest. The scene is dimly lit, with a misty or foggy atmosphere. Bare, dark tree trunks and branches are visible, some with a few small, yellowing leaves. A path or clearing leads from the foreground into the distance, where a faint light source, possibly the sun or moon, is visible through the trees. The overall mood is mysterious and somber.

Context

Ethereum Client Software, EVM, ABI, and Solidity

High Level Things to Consider

Diving into Solidity

Complex Stuff

Tools

Context

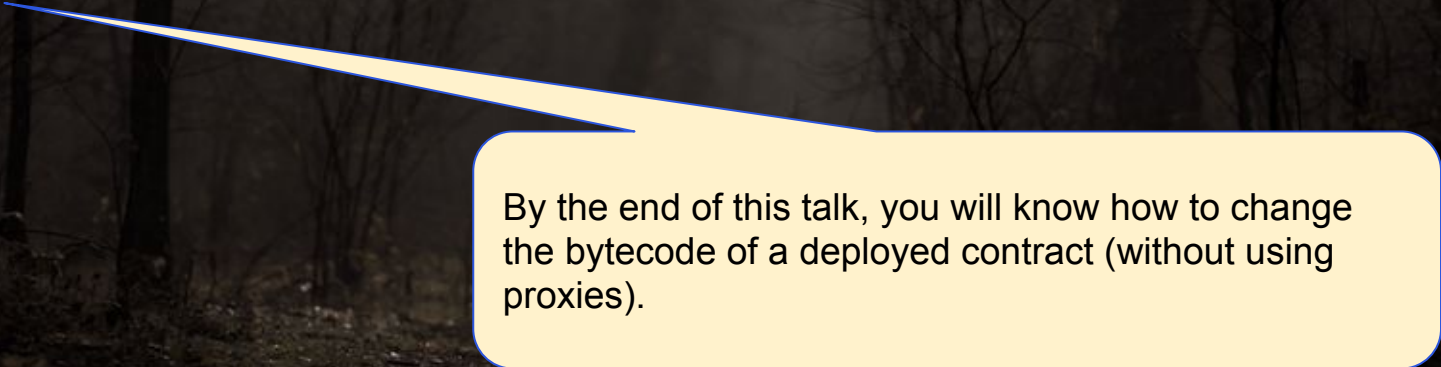
Ethereum Client Software, EVM, ABI, and Solidity

High Level Things to Consider

Diving into Solidity

Complex Stuff

Tools



By the end of this talk, you will know how to change the bytecode of a deployed contract (without using proxies).

Dark Forest

- Dark Forest is a term introduced author Liu Cixin in a science fiction book where identifying someone else's location means certain death at the hand of aliens.
 - https://en.wikipedia.org/wiki/The_Dark_Forest
- Dan Robinson, Georgios Konstantopoulos brought the term to the crypto space to describe the adversarial environment of the Ethereum mempool.
 - Post a transaction to the mempool, and Generalised MEV Bot will determine if there is any value that can be extracted by front running the transaction, or any inter-contract calls resulting from the transaction.
 - <https://www.paradigm.xyz/2020/08/ethereum-is-a-dark-forest>
- I am reusing the term in this talk in a slightly different way.
 - If there is a weakness in your contracts or applications, then expect people / bots to find the weakness and exploit it.
- Ethereum is a Dark Forest.



Context

Smart Contract Security vs Solidity Security

- Smart Contract Security:
 - Security that is defined by the properties of the EVM.
- Solidity Security:
 - Security implications of the writing Smart Contracts using Solidity.

What does a contract have?

Contracts:

- Code.
- Storage.
- Balance.
- Nonce.

What does a contract have?

Contracts:

- Code.
- Storage.
- Balance.
- Nonce.



Smart Contract Security is about preventing *unexpected* changes to these three, in particular the top two!

What does a contract have?

Contracts:

- Code.
- Storage.
- Balance.
- Nonce.



Smart Contract
unexpected changes to the
top two!

and that includes
selfdestruct(addr)

in particular the

What does a contract have?

Contracts:

- Code.
- Storage.
- Balance.
- Nonce.

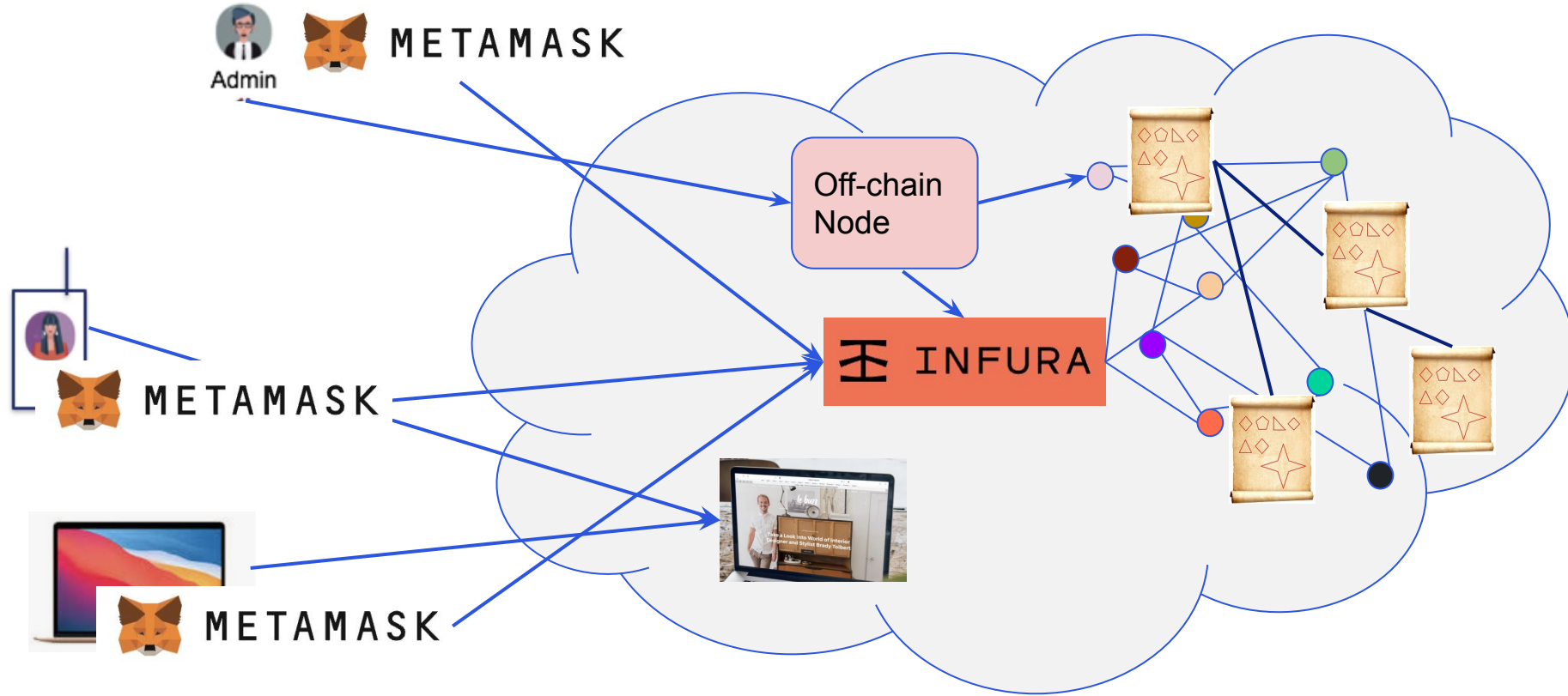


Smart Contracts
unexpected changes
top two!

and that includes
getting stuck

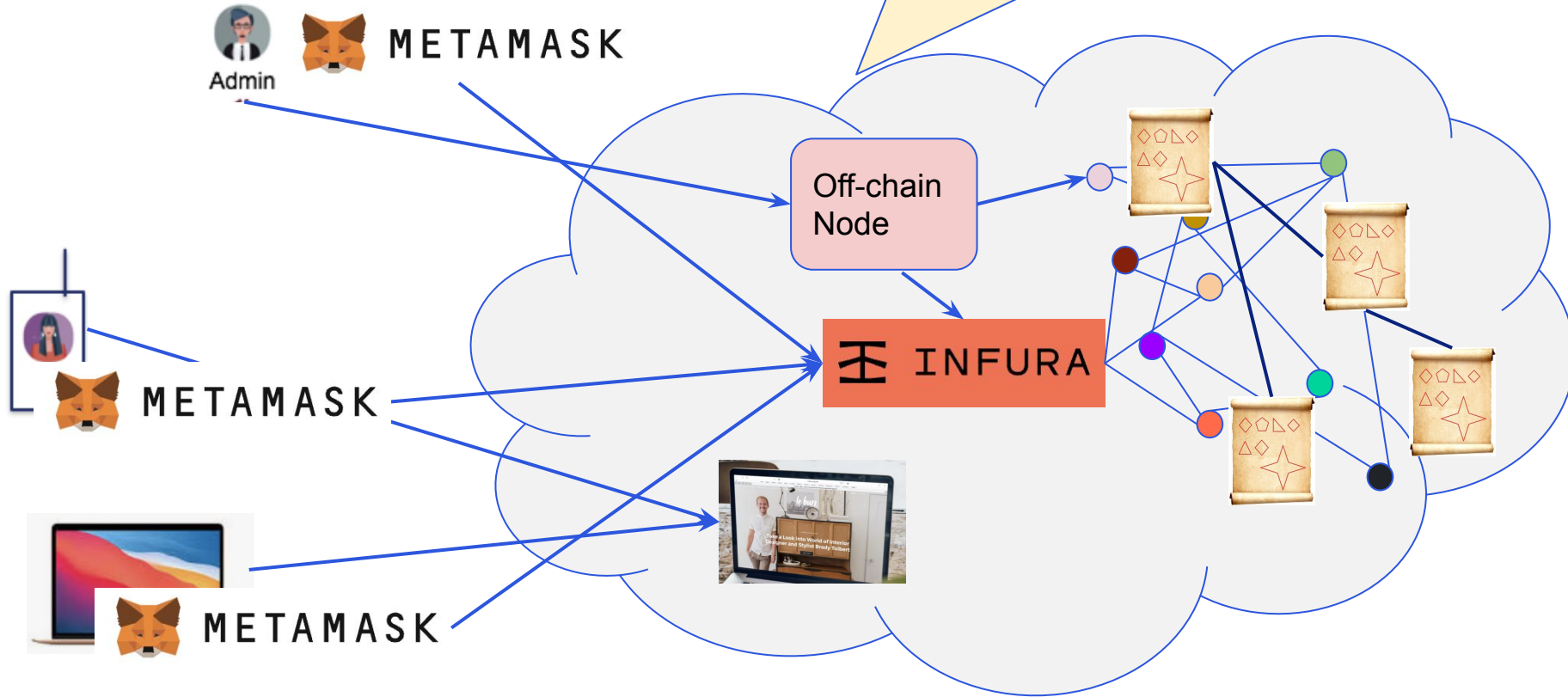
particular the

Applications



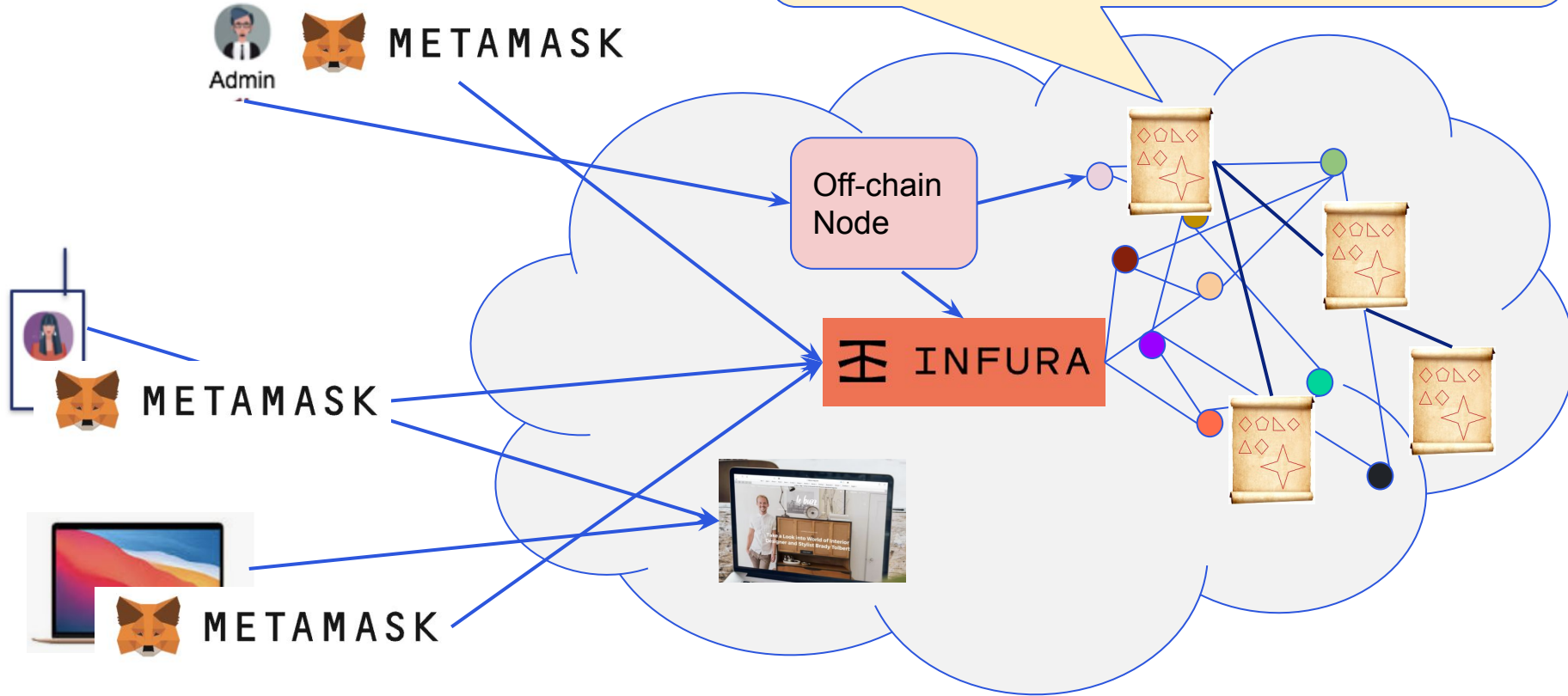
Applications

Smart Contracts are part of larger applications.
Attackers often target other parts of the application.



Applications

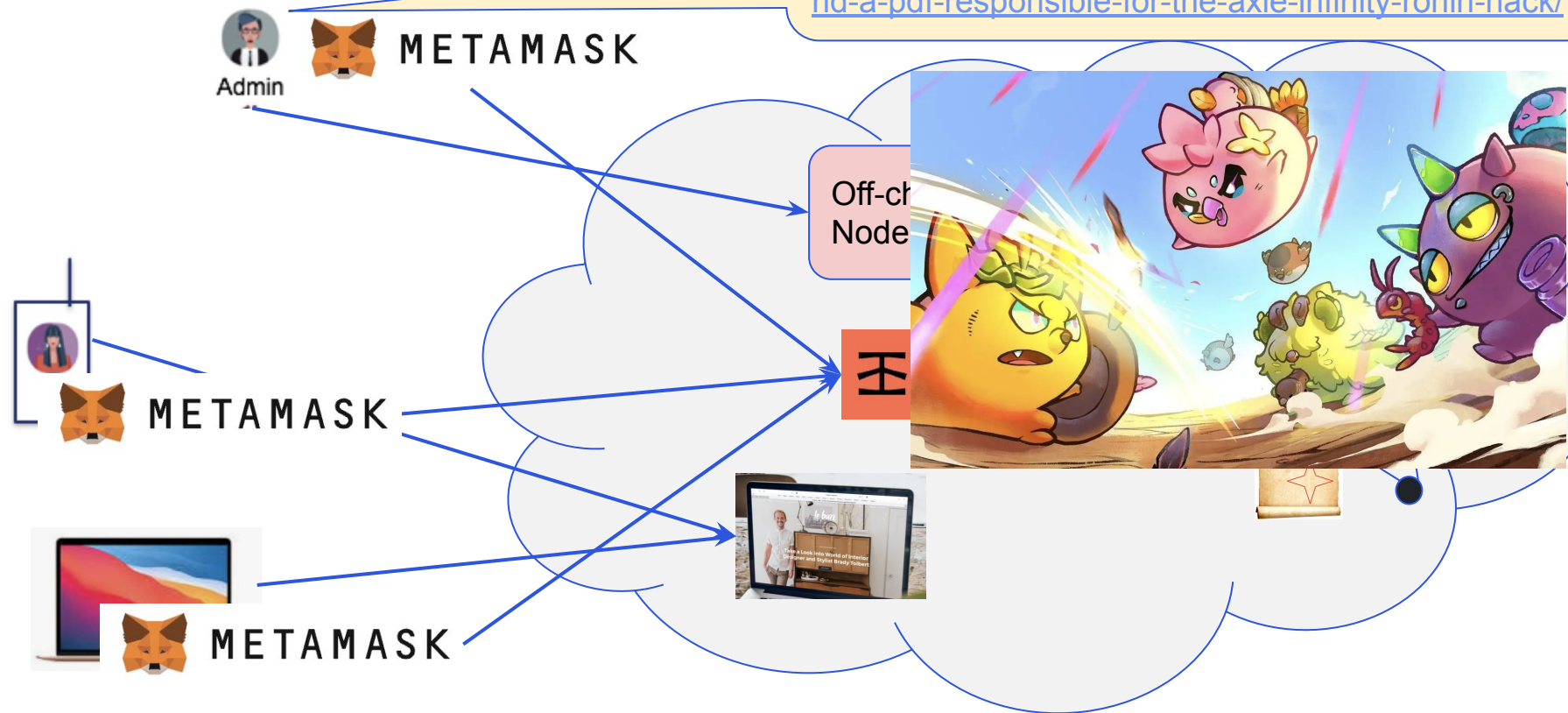
Contracts are often used together with multiple other contracts. They need to be analysed separately and as a whole.



Applications

Complex social engineering can be used to attack an application: for example the Ronin hack.

<https://www.newsbtc.com/axs/are-a-fake-job-offer-and-a-pdf-responsible-for-the-axie-infinity-ronin-hack/>

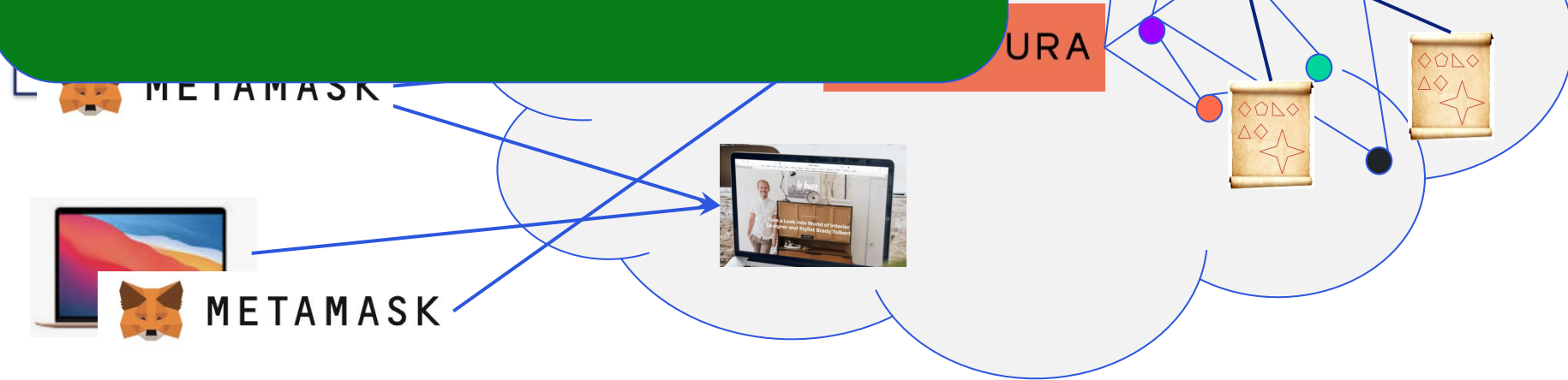


Applications

When you do Threat Modelling, you need to identify all of your Attack Surfaces, not just the contracts.

https://owasp.org/www-community/Threat_Modeling

<https://www.fortinet.com/resources/cyberglossary/attack-surface>



Ethereum Client Software, EVM, ABI, and Solidity

Ethereum Client Software, EVM, ABI, and Solidity

Understanding what the

Ethereum Client Software,

Ethereum Virtual Machine (EVM),

Application Binary Interface (ABI),

and the Solidity programming language

each provide will help you on your Smart Contract security journey.

Ethereum Client Software

Software that runs the Ethereum node, including executing contract code in the EVM.

Important thing to understand:

- **eth_call RPC method:**
 - Typically used for view calls, though can be used for any function.
 - Executes on the local node.
 - Is not included in a block, and hence does not update the state of the blockchain.
- **eth_sendRawTransaction RPC method:**
 - Submits a transaction, that can be value transfer and a function call.
 - Executed when included in a block.
 - Incorporate state updates if last EVM opcode is STOP, RETURN or SELFDESTRUCT;
 - Discard state updates if the last EVM opcode is REVERT or INVALID, or an exceptional situation occurs; for example: out of gas, stack overflow / underflow, invalid jump destination.

<https://github.com/ethereum/yellowpaper>

EVM executes
opcodes defined in the
Yellow Paper

ETHEREUM: A SECURE DECENTRALISED GENERALISED TRANSACTION LEDGER
BERLIN VERSION 7251f10 – 2022-07-31

DR. GAVIN WOOD
FOUNDER, ETHEREUM & PARITY
GAVIN@PARITY.IO

ABSTRACT. The blockchain paradigm when coupled with cryptographically-secured transactions has demonstrated its utility through a number of projects, with Bitcoin being one of the most notable ones. Each such project can be seen as a simple application on a decentralised, but singleton, compute resource. We can call this paradigm a transactional singleton machine with shared-state.

Ethereum implements this paradigm in a generalised manner. Furthermore it provides a plurality of such resources, each with a distinct state and operating code but able to interact through a message-passing framework with others. We discuss its design, implementation issues, the opportunities it provides and the future hurdles we envisage.

1. INTRODUCTION

With ubiquitous internet connections in most places of the world, global information transmission has become incredibly cheap. Technology-rooted movements like Bitcoin have demonstrated through the power of the default, consensus mechanisms, and voluntary respect of the social contract, that it is possible to use the internet to make a decentralised value-transfer system that can be shared across the world and virtually free to use. This system can be said to be a very specialised version of a cryptographically secure, transaction-based state machine. Follow-up systems such as Namecoin adapted this original “currency application” of the technology into other applications albeit rather simplistic ones.

Ethereum is a project which attempts to build the generalised technology; technology on which all transaction-based state machine concepts may be built. Moreover it aims to provide to the end-developer a tightly integrated end-to-end system for building software on a hitherto un-

is often lacking, and plain old prejudices are difficult to shake.

Overall, we wish to provide a system such that users can be guaranteed that no matter with which other individuals, systems or organisations they interact, they can do so with absolute confidence in the possible outcomes and how those outcomes might come about.

1.2. Previous Work. Buterin [2013a] first proposed the kernel of this work in late November, 2013. Though now evolved in many ways, the key functionality of a blockchain with a Turing-complete language and an effectively unlimited inter-transaction storage capability remains unchanged.

Dwork and Naor [1992] provided the first work into the usage of a cryptographic proof of computational expenditure (“proof-of-work”) as a means of transmitting a value signal over the Internet. The value-signal was utilised here as a spam deterrence mechanism rather than any kind of currency, but critically demonstrated the potential for

PC: 0x0, opcode: PUSH1 0x80
PC: 0x2, opcode: PUSH1 0x40
PC: 0x4, opcode: MSTORE
PC: 0x5, opcode: CALLVALUE
PC: 0x6, opcode: DUP1
PC: 0x7, opcode: ISZERO
PC: 0x8, opcode: PUSH2 0x0010
PC: 0xb, opcode: JUMPI
PC: 0xc, opcode: PUSH1 0x00
PC: 0xe, opcode: DUP1
PC: 0xf, opcode: REVERT
PC: 0x10, opcode: JUMPDEST
....

Execution **always** starts at offset 0.

This includes for:

- init code fragments,
- Transactions,
- View calls,
- and cross-contract calls.

PC: 0x10, opcode: JUMPDEST
PC: 0x11, opcode: POP
PC: 0x12, opcode: PUSH1 0x03
PC: 0x14, opcode: PUSH1 0x01
PC: 0x16, opcode: SSTORE

PC: 0x17, opcode: PUSH1 0xc1
PC: 0x19, opcode: DUP1
PC: 0x1a, opcode: PUSH2 0x0024
PC: 0x1d, opcode: PUSH1 0x00
PC: 0x1f, opcode: CODECOPY
PC: 0x20, opcode: PUSH1 0x00
PC: 0x22, opcode: RETURN

PC: 0x23, opcode: INVALID
PC: 0x24, opcode: PUSH1 0x80

....

init code fragments initialise storage and specifies the code to be used as the contract runtime code

Application Binary Interface (ABI)

<https://docs.soliditylang.org/en/v0.8.13/abi-spec.html>

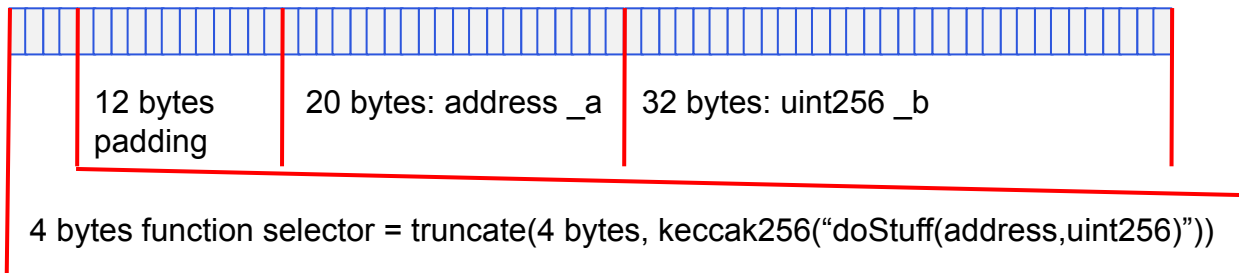
“The Contract Application Binary Interface (ABI) is the **standard way to interact with contracts in the Ethereum ecosystem**, both from **outside the blockchain** and for **contract-to-contract** interaction. Data is encoded according to its type, as described in this specification. The encoding is not self describing and thus requires a schema in order to decode.

We assume the **interface functions of a contract are strongly typed, known at compilation time and static**. We assume that all contracts will have the interface definitions of any contracts they call available at compile-time.

This specification does not address contracts whose interface is dynamic or otherwise known only at run-time.”

Application Binary Interface (ABI)

```
doStuff(address _a, uint256 _b)
```



NOTE:

- All parameters are padded to 32 bytes.
- Encoding rules for arrays, bytes, and structs are more complex.
For details see: <https://docs.soliditylang.org/en/v0.8.15/abi-spec.html>

Application Binary Interface (ABI)

ABI encoding used in the following situations:

- Functions and parameters.
- Function return values.
 - Note: In case of error or panic reverts, functions return:
 - `Panic(uint256 _errorCode)`
 - `Error(string _errorMessage)`
 - Note that with recent Solidity versions, reverts can also return custom errors.
- Event parameters.

Solidity

A high level programming language that:

- Compiles to EVM bytecode that can execute successfully in the EVM.
- Follows the ABI encoding rules.

Many “Smart Contract” concepts are really just creations of Solidity:

- payable
- Storage data structures: mapping, arrays, structs.
- Memory management.
- enum
- Function visibility (external, public, internal & private).
- Functions.
- Revert chaining across contract calls.
- Solidity style “inheritance”.
- Modifiers.
- Checked.
- fallback / receive.
- Constructors and initialising contract code.
- Contract types.
- Function types.
- Structs as parameters.
- Using

Learn about the EVM before /
in-parallel with learning about Solidity.

Your ability to create secure Smart
Contracts will be improved by
understanding what the EVM provides
and what are creations of Solidity.

https://www.youtube.com/watch?v=RxL_1AfV7N4

Many “Smart Contract” concepts are really just creations of Solidity:

- payable
- Storage data structures: mapping, arrays, structs.
- Memory management.
- enum
- Function visibility (external, public, internal & private).
- Functions.
- Revert chaining across contract calls.
- Solidity style “inheritance”.
- Modifiers.
- Checked.
- fallback / receive.
- Constructors and initialising contract code.
- Contract types.
- Function types.
- Structs as parameters.
- Using

All of these (security) features of Solidity need to be encoded by the compiler to bytecode.

This assumes that the Solidity compiler is trusted and does not introduce compilation bugs.

How realistic is this assumption?

Solidity

Note that Solidity code can dive into assembler to execute specific assembler / EVM opcodes.

```
function bytesToAddress1(bytes memory _b, uint256 _startOffset) internal
    pure returns (address addr) {

    assembly {
        addr := mload(add(_b, add(32, _startOffset)))
    }
}
```

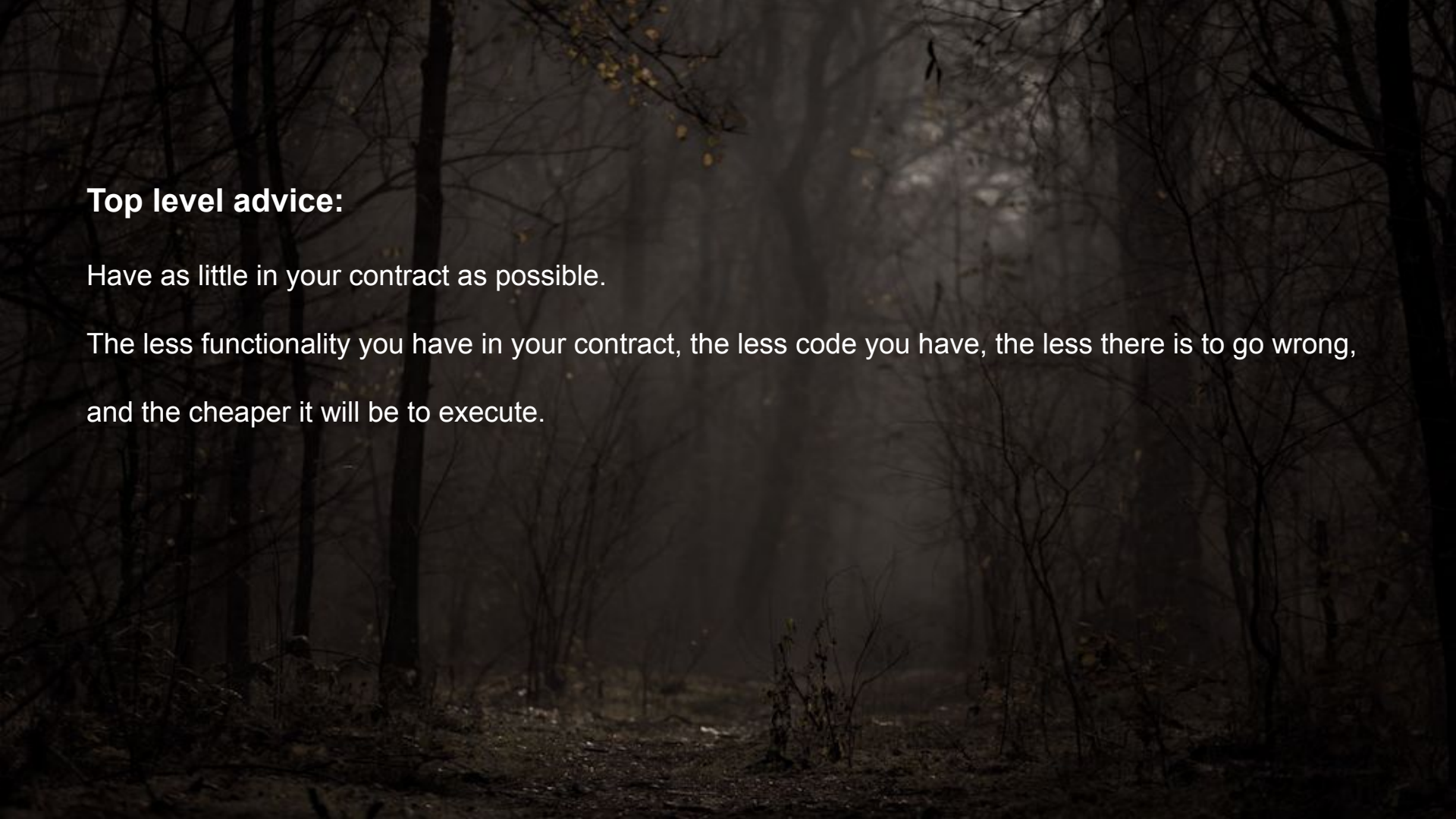
Smart Contract Security vs Solidity Security

- Smart Contract Security:
 - Security that is defined by the properties of the EVM.
- Solidity Security:
 - Security implications of the writing Smart Contracts using Solidity.

Given a Solidity smart contract can dive into assembler and do anything....

Is Smart Contract Security and Solidity Security the same thing?

But what about SECURITY?

A dark, atmospheric photograph of a forest. The scene is dimly lit, with a mist or fog hanging between the trees. Bare, thin tree trunks and branches are visible, some with a few yellowing leaves. A path or clearing leads from the foreground into the distance, where a faint light source, possibly the sun or moon, is visible through the haze. The overall mood is mysterious and somber.

Top level advice:

Have as little in your contract as possible.

The less functionality you have in your contract, the less code you have, the less there is to go wrong, and the cheaper it will be to execute.

High Level Things to Consider

Upgrade

Upgrade

Good	Bad
All code has bugs. Upgrading allows bugs to be resolved.	Bugs can be inadvertently introduced. The issues could be in the new code, some new data introduced with the upgrade, or how the new code works with the existing data.
New features can be added.	Contract is less trustworthy because of project owners could add malicious code to the project. For example adding a function: <code>withdrawAllCoins()</code> .
	Who / which group of people can upgrade the contract? What security procedures do they use? If they are hacked, then... see above.

Upgrade

Good

Bad

All code has bugs. Upgrading allows bugs

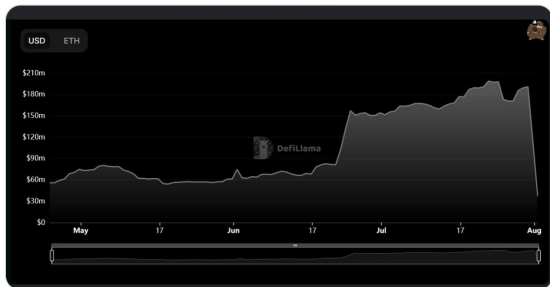
Bugs can be inadvertently introduced. The issues could be in the new code, some new data introduced with the upgrade, or how the new code works with the existing data.

Contract is less trustworthy because of project owners could add malicious code to the project. For example adding a function: **`withdrawAllCoins()`**.

Who / which group of people can upgrade the contract? What security procedures do they use? If they are hacked, then... see above.



1/ Nomad just got drained for over \$150M in one of the most chaotic hacks that Web3 has ever seen. How exactly did this happen, and what was the root cause? Allow me to take you behind the scenes 📌



9:45 AM · Aug 2, 2022 · Twitter Web App

Pausing, Freezing, and Banning

Pausing, Freezing, and Banning


Pausable: Ability for operations of a contract to be stopped in case of emergency.

Modifiers

```
contract Stuff {  
    modifier aMod(uint256 _amount) {  
        // Some code here  
        _;  
        // Some more code here  
    }  
  
    function doStuff(uint256 _amount) aMod(_amount) public {  
        // Code  
    }  
}
```

Modifiers

```
contract Stuff {  
    modifier aMod(uint256 _amount  
        // Some code here  
        _;  
        // Some more code here  
    }  
  
    function doStuff(uint256 _amc  
        // Code  
    }  
}  
  
contract Stuff {  
    function doStuff(uint256 _amount) public {  
        // Some code here  
        // Code  
        // Some more code here  
    }  
}
```



Pausing, Freezing, and Banning

Pausable: Ability for operations of a contract to be stopped in case of emergency.

```
modifier whenNotPaused() {  
    _requireNotPaused();  
    _;  
}  
function _requireNotPaused() internal view virtual {  
    require(!paused(), "Pausable: paused");  
}  
function paused() public view virtual returns (bool) {  
    return _paused;  
}
```

2100 gas on
first read

Freezing and Banning

Banning: Disallow deposits and operations for an address.

Freezing: Banning + disallow withdrawals.

Assuming each are implemented as a mapping:

```
mapping(address => bool) public banned;
```

```
mapping(address => bool) public frozen;
```

2100 gas on
first read per
address

For a transfer with a to and
from address: 4200 gas to
check if banned

Who chooses which accounts are
banned or frozen?

How quickly can freezing of accounts
happen given the governance model?

Front Running

Dark Forest and Front Running

The following is based on an example by Rodrigo Herrera Itie, which is based on an article by Dan Robinson & Georgios Konstantopoulos.

```
// SPDX-License-Identifier: LGPL-3.0-only
pragma solidity ^0.8.9;

contract Pray {
    receive() external payable {}

    function _transfer() external {
        uint256 bal = address(this).balance;
        (bool success, ) = payable(msg.sender).call{value: bal}("");
        require(success);
    }
}
```

Front Running Prevention: Commit - Reveal

The best remediation is to **remove the benefit of front-running in your application**, mainly by removing the importance of transaction ordering or time.

Commit - Reveal:

1. Indicate intent to do some action (claim a bounty, propose a purchase or sale), submit a commitment:
 - Submit: `keccak256(encoded action, msg.sender)`
 - Contract stores the commitment (message digest)
2. To execute the action, reveal the proposed action:
 - Submit: encoded action
 - Contract checks that the commitment `== keccak256(encoded action, msg.sender)`

Dark Forest and Front Running

```
// SPDX-License-Identifier: LGPL-3.0-only
pragma solidity ^0.8.9;

contract CommitReveal {
    bytes32 commitment;
    uint256 timeout;
    uint256 constant ONE_DAY = 24 * 60 * 60;

    receive() external payable {}

    function register(bytes32 _commitment) external {
        if (commitment != bytes32(0)) {
            require(block.timestamp > timeout, "Wait");
        }
        commitment = _commitment;
        timeout = block.timestamp + ONE_DAY;
    }

    function transfer(bytes32 _secret) external {
        require(commitment == keccak256(abi.encodePacked(_secret, msg.sender)), "Mismatch");

        uint256 bal = address(this).balance;
        (bool success, ) = payable(msg.sender).call{value: bal}("");
        require(success, "Transfer failed");
    }
}
```

NOTE: This code compiles, however I haven't run it.

It would be super interesting to deploy this and put some Ether into it, and see if this is complicated enough to fool Generic MEV Bots.

Another addition could be to have the time window when transfer can be called to start some time in the future.

Dark Forest and Front Running

```
// SPDX-License-Identifier: LGPL-3.0-only
pragma solidity ^0.8.9;

contract CommitReveal2 {
    mapping (address => bytes32) commitments;

    receive() external payable {}

    function register(bytes32 _commitment) external {
        commitments[msg.sender] = _commitment;
    }

    function transfer(bytes32 _secret) external {
        require(commitments[msg.sender] ==
            keccak256(abi.encodePacked(_secret, msg.sender)), "Mismatch");

        uint256 bal = address(this).balance;
        (bool success, ) = payable(msg.sender).call{value: bal}("");
        require(success, "Transfer failed");
    }
}
```

NOTE: This code compiles, however I haven't run it.

It would be super interesting to deploy this and put some Ether into it, and see if this is complicated enough to fool Generic MEV Bots.

Front Running Prevention: Batch Auctions

Batch Auctions:

- Individual orders are grouped together into a batch and executed at the same time.
- The same clearing price is assigned to all orders within one batch.

Check Deployed Configuration

Celo Optics Recovery Mode



Optics Recovery Mode

■ Announcements



tim

1 Nov '21

I just posted this [announcement](#) ²⁴⁵ in the [Optics Discord server](#) ³¹⁵, and cross-posting here:

We recently discovered an issue with Optics beta and our team is actively working to resolve it. Here is what we know: Despite no known vulnerability, and without alerting the community, someone unilaterally activated the Optics recovery mode on the Ethereum GovernanceRouter contract. While Optics continues to function, recovery mode gives the recovery manager account full control of Optics, overriding the governance multisig. This event occurred less than 15 minutes after cLabs terminated James Prestwich's employment for misconduct: a parting of ways that was not due to issues around Optics or James' technical work.

Configuration

“Recovery Mode” should have been controlled by a multisig: it was controlled by just one EOA.

“Recovery Mode Timelock” was supposed to be long enough for people to react and perhaps exit their funds. It was set to 1 second.

Take home:

- **Check the configuration of your deployed project.**
- **Where possible limit possible configuration values to reasonable values.**
 - For instance, if a “Recovery Mode Timelock” is supposed to be long enough for people to react, and perhaps exit their funds, then maybe a reasonable minimum is one day.

Diving into Solidity

Compiler Version

Compiler Version

Use the latest released version of the compiler.

- Defects are fixed in the solc compiler.
- Old versions of the compiler might generate code that contain issues that someone might be able to exploit.

Compiler Version

Solidity pragma

```
pragma solidity 0.8.15;
```

- Tells the solc compiler to fail if it isn't version 0.8.15.
- Useful if you only want to compile the contract to be deployed with a certain version of the compiler: the version that it has been tested with.

Compiler Version

Solidity pragma

```
pragma solidity ^0.8.15;
```

or equivalently:

```
pragma solidity >=0.8.15 <0.9.0;
```

- Tells the solc compiler to fail if it isn't version 0.8.15 or later, but before version 0.9.0.
- Useful to incorporate bug fix versions of the compiler, but to indicate that the code might break at version 0.9.0.

When compiling with version 0.9.x, the compilation will fail. This will trigger you to review your code, given changes in 0.9.0. This is a good thing!

Compiler Version

What about this advice?

“Locking the pragma helps ensure that contracts do not accidentally get deployed using, for example, the latest compiler which may have higher risks of undiscovered bugs.”

<https://consensys.net/blog/developers/solidity-best-practices-for-smart-contract-security/>

- My perspective is that new features might have undiscovered bugs, however, older features are likely to have bug fixes.
- Attackers could utilise bug fixes to identify how to attack your code.

Function Visibility

Function Visibility

	Contract ABI transaction, eth_call, cross-contract	Call from inheriting contract	Call from same contract
external*	✓		
public	✓	✓	✓
internal		✓	✓
private			✓

Functions

It is common for non-trivial contracts to include lots of other contracts:

```
contract GpactV2CrosschainControl is
    CrosschainFunctionCallReturnInterface,
    CbcDecVer,
    CallExecutionTreeV2,
    CallPath,
    CrosschainLockingInterface,
    AtomicHiddenAuthParameters,
    ResponseProcessUtil {
```

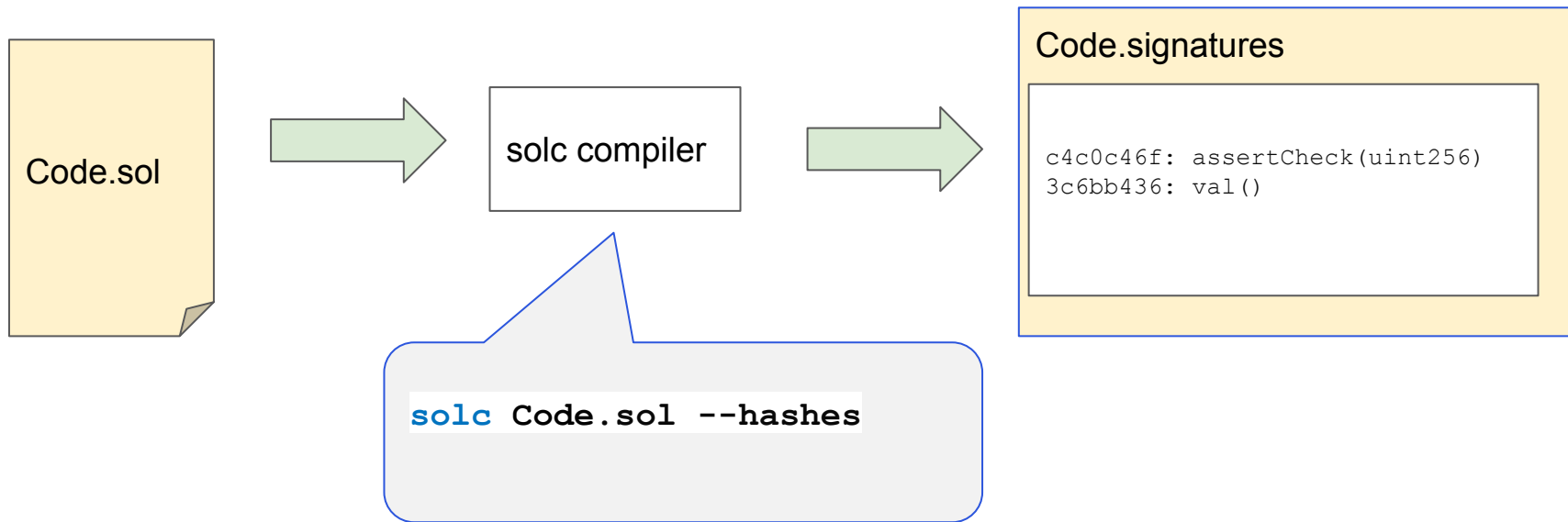
Functions

It is very easy to miss a function, especially if some of the code has been *acquired / copied*.

```
contract MyContract is B, F {  
    ...  
}  
contract B is C, D, E {  
    ...  
}  
contract C {  
    function iBetYouDidntKnowAboutMe() external {  
        ...  
    }  
}
```

Functions

Check the function selectors of the contract for functions you didn't know about / forgot.



Access Control

Access Control

- Limit who can call **external** and **public transaction** functions in your code.

```
function doStuff() external {  
    require(msg.sender == authorised, "Not authorised");  
}
```

```
function doStuff() external {  
    require(authorised[msg.sender], "Not authorised");  
}
```

Access Control

- Limit who can call **external** and **public transaction** functions in your code.

```
function doStuff() external {  
    require(msg.sender == authorised, "Not authorised")  
}
```

```
function doStuff() external {  
    require(authorised[msg.sender], "Not authorised")  
}
```

NOTE: There is no need to have access control on **view** functions, as they can't change the state of the ledger, and all state is public.

Access Control

```
contract WalletLibrary {  
    address public owner;  
  
    function initWallet(address _owner) external {  
        owner = _owner;  
    }  
}
```

This is a simplification.

2nd Parity Wallet hack (the selfdestruct one):

- Nov 6, 2017
- **513,774.16** Ether

fallback and receive

fallback and receive

Understand how fallbacks work.

```
receive() external payable { ... }
```

- Ether transfers.
- Transactions with no call data.

```
fallback() external { ... }
```

```
fallback() external payable { ... }
```

```
fallback(bytes _input) external { ... }
```

```
fallback(bytes _input) external payable { ... }
```

- Called for transactions when there is at least one byte of call data.
- Payable variants called for Ether transfers when there is no receive function.

fallback and receive

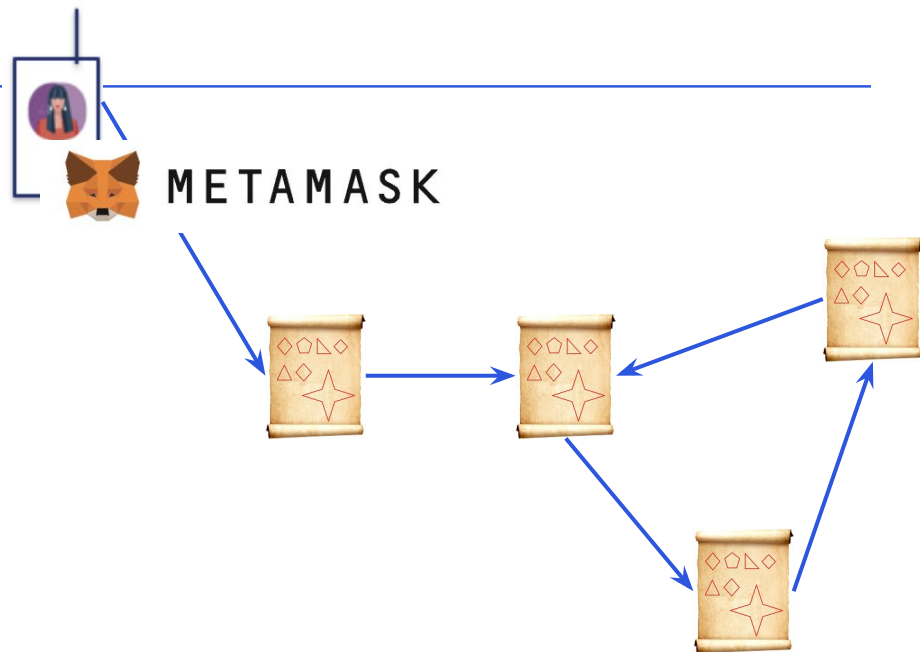
- An old ethos was, “If someone sends your contract Ether (possibly by mistake), accept it!”
- If you don’t plan to manage Ether in your contract, don’t have any payable functions.
 - That includes receive and fallback.

Reentrancy

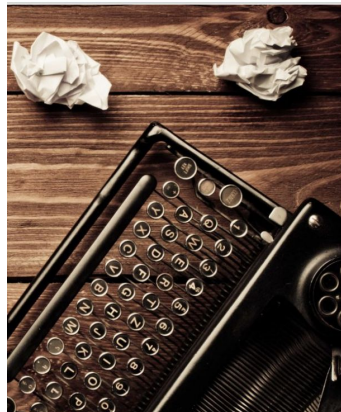
Reentrancy

Reentrancy issues can affect any cross-contract interaction:

- value transfer
- function call



› Ethereum › Understanding The DAO Attack



Ethereum

Understanding The DAO Attack

Blockchain strategist David Siegel gives a step overview of the attack on The DAO for journalists and media members.

By David Siegel

Reentrancy

```
// SPDX-License-Identifier: GPL-3.0
pragma solidity >=0.6.2 <0.9.0;

// THIS CONTRACT CONTAINS A BUG - DO NOT USE
contract Fund {
    /// @dev Mapping of ether shares of the contract.
    mapping(address => uint) shares;
    /// Withdraw your share.
    function withdraw() public {
        (bool success,) = msg.sender.call{value: shares[msg.sender]}("");
        if (success)
            shares[msg.sender] = 0;
    }
}
```

Reentrancy

```
// SPDX-License-Identifier: GPL-3.0
pragma solidity >=0.6.2 <0.9.0;

// THIS CONTRACT CONTAINS A BUG - DO NOT USE
contract Fund {
    /// @dev Mapping of ether shares of the contract
    mapping(address => uint) shares;
    /// Withdraw your share.
    function withdraw() public {
        (bool success,) = msg.sender.call{value: shares[msg.sender]}("");
        if (success)
            shares[msg.sender] = 0;
    }
}
```

call is a cross-contract call, providing all remaining gas.

The **receive()** function in the other contract could call **withdraw()** to drain all funds.

Reentrancy

```
// SPDX-License-Identifier: GPL-3.0
pragma solidity >=0.6.0 <0.9.0;

contract Fund {
    /// @dev Mapping of ether shares of the contract.
    mapping(address => uint) shares;
    /// Withdraw your share.
    function withdraw() public {
        uint share = shares[msg.sender];
        shares[msg.sender] = 0;
        payable(msg.sender).transfer(share);
    }
}
```

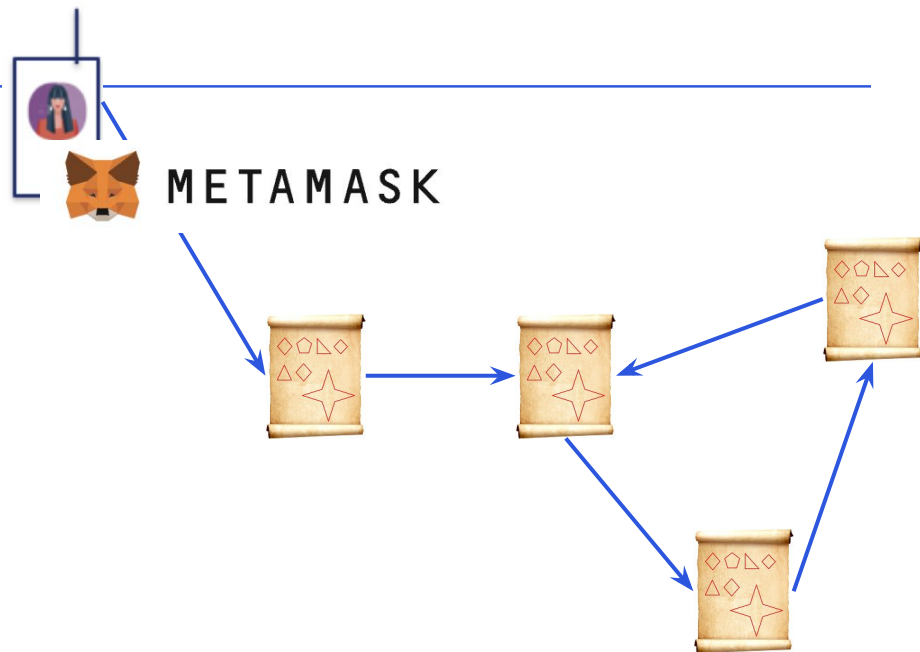
Update state,
then do cross-contract
interaction.

Reentrancy

Use the Check-Effects-Interactions pattern:

“Most functions will first perform some checks (who called the function, are the arguments in range, did they send enough Ether, does the person have tokens, etc.). These checks should be done first.

As the second step, if all checks passed, effects to the state variables of the current contract should be made. Interaction with other contracts should be the very last step in any function.”



Reentrancy Guard

```
abstract contract ReentrancyGuard {  
    uint256 private constant _NOT_ENTERED = 1;  
    uint256 private constant _ENTERED = 2;  
    uint256 private _status;  
  
    constructor() {  
        _status = _NOT_ENTERED;  
    }  
  
    modifier nonReentrant() {  
        require(_status != _ENTERED, "ReentrancyGuard: reentrant call");  
        _status = _ENTERED;  
        _;  
        _status = _NOT_ENTERED;  
    }  
}
```


Reentrancy Guard

```
abstract contract ReentrancyGuard {  
    uint256 private constant _NOT_ENTERED = 1;  
    uint256 private constant _ENTERED = 2;  
    uint256 private _status;
```

```
    constructor() {  
        _status = _NOT_ENTERED;  
    }
```

```
    modifier nonReentrant() {  
        require(_status != _ENTERED, "ReentrancyGuard: reentrant call");  
        _status = _ENTERED;  
        _;  
        _status = _NOT_ENTERED;  
    }
```

First write to
non-zero value:
22100 gas

Read of cold
storage slot:
2100 gas

Write to hot
storage slot:
2900 gas

Reset to original value
of hot storage slot:
100 gas + 2800 gas
refund

Total gas cost of nonReentrant():
 $2100 + 2900 + 100 - 2800 = 2300$ gas

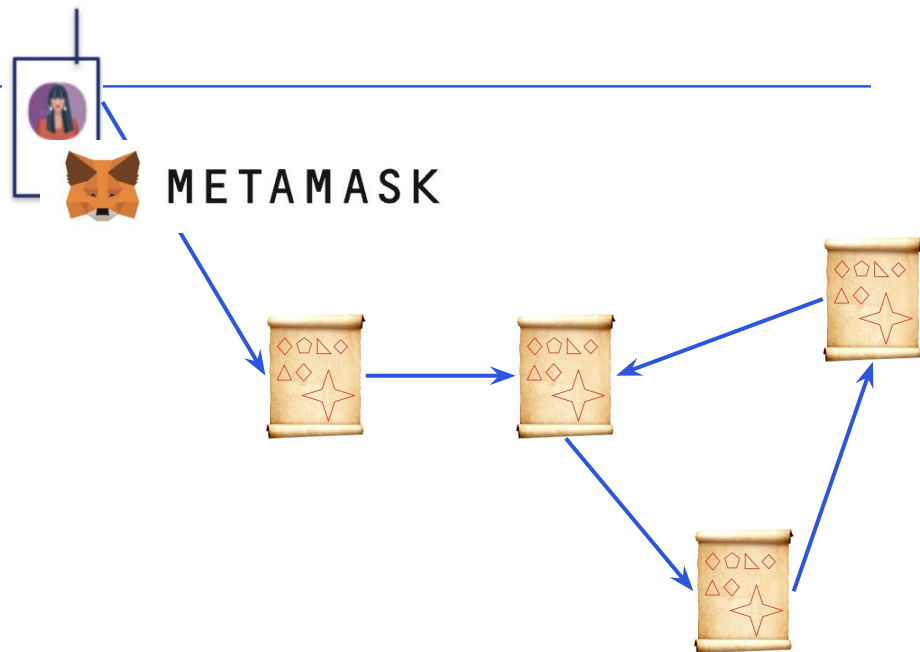
Limiting Gas

The maximum gas that can be passed to a called contract is 63/64ths of the remaining gas.

If the transaction has a gas limit of 10,000,000 gas, and you assume minimalistic functions that cost circa 3200 gas to call, this limits the call depth to around **500**.

Remaining gas can be further restricted by specifying the amount of gas available:

```
otherC.func{gas: 5000} ();
```



How much gas is something using???

```
uint256 public gas1;  
uint256 public gas2;  
uint256 public gas3;  
  
function measure() external {  
    gas1 = gasleft();  
    gas2 = gasleft();  
    otherC.func();  
    gas3 = gasleft();  
}
```

Storing to a cold storage slot whose current value is 0x00 costs 22100 gas.

$\text{gas2} - \text{gas1} - 22100 = 13$

$\text{gas3} - \text{gas2} - 22100 - 13 = \text{cost of otherC.func.}$

Limiting Gas

```
otherC.func{gas: 5000} ();
```

- Opcode prices change.
- If you limit the gas passed to a called function too much, and the gas price goes up, then your application might not work anymore.

	Before Berlin	Berlin to London	After London
SSTORE (val = 1)	20,000	22,100	22,100
SLOAD (return val)	800	100	100
SSTORE (val = 0)	800 +19,200 refund	100 +19,900 refund	100 +19,900 refund


Max refund: gas / 2

Max refund: gas / 5

Modifiers

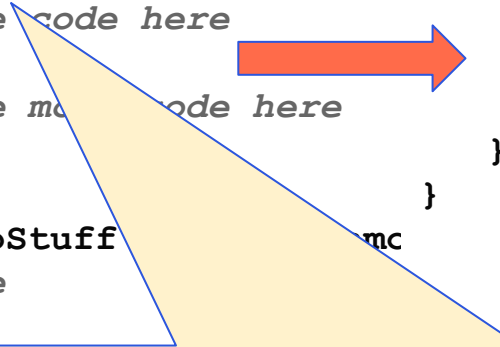
Modifiers

```
contract Stuff {  
    modifier aMod(uint256 _amount  
        // Some code here  
        _;  
        // Some more code here  
    }  
  
    function doStuff(uint256 _amc  
        // Code  
    }  
}  
  
contract Stuff {  
    function doStuff(uint256 _amount) public {  
        // Some code here  
        // Code  
        // Some more code here  
    }  
}
```



Modifiers

```
contract Stuff {  
    modifier aMod(uint256 _amount)  
        // Some code here  
        _;  
        // Some more code here  
    }  
  
    function doStuff(uint256 _amount) public {  
        // Some code here  
        // Code  
        // Some more code here  
    }  
}
```



The diagram illustrates the use of a modifier in Solidity. On the left, a contract named 'Stuff' is shown with a modifier 'aMod' that takes a 'uint256' parameter '_amount'. The modifier's body contains two lines of code: '// Some code here' followed by a semicolon and '_;', and then '// Some more code here'. Below the modifier is a function 'doStuff' that also takes a 'uint256' parameter '_amount' and is marked as 'public'. The function's body contains three lines of code: '// Some code here', '// Code', and '// Some more code here'. A red arrow points from the modifier definition to the function call site. A yellow callout box points to the modifier definition.

Modifiers are often defined in inherited contracts that are far, far away.

Modifiers

```
contract Stuff {  
    modifier aMod(uint256 _amount  
        // Some code here  
        ;  
    // Some more code here  
}  
  
function doStuff(uint256 _amc  
    // Code  
}  
}
```

contract Stuff {
 modifier aMod(uint256 _amount
 // Some code here
 ;
 // Some more code here
}

function doStuff(uint256 _amc
 // Code
}
}

Don't put code in here! No one is expecting it!

Modifiers

- Limit modifier code to checks.

```
contract Election {
    mapping(address => bool) public registered;
    mapping(uint256 => bool) public proposalActive;

    modifier isEligible() {
        require(registered[msg.sender], "Not registered to vote!");
        _;
    }
    modifier isValid(uint256 _proposal) {
        require(proposalActive[_proposal], "Proposal not active!");
        _;
    }

    function vote(uint256 _proposal) isEligible isValid(_proposal) public {
        // Code
    }
}
```

Modifiers

- Preferably no cross-contract calls!

```
contract Registry {
    function isVoter(address _addr) external returns(bool) {
        // Code that could do anything, including calling vote()!
    }
}

contract Election {
    Registry registry;

    modifier isEligible() {
        require(registry.isVoter(msg.sender));
        _;
    }
    function vote() isEligible public {
        // Code
    }
}
```

Value Transfer

```
contract Transfer {
    function sendValue1(address payable _to) external payable {
        _to.transfer(msg.value);
    }

    function sendValue2(address payable _to) external payable {
        require(_to.send(msg.value), "Send failed");
    }

    error AnError(string _msg, bytes _revertInfo);

    function sendValue3(address payable _to) external payable {
        (bool sent, bytes memory data) = _to.call{value: msg.value}("");
        if (!sent) {
            revert AnError("Send failed", data);
        }
    }
}
```

Supplies 2300
gas stipend

Supplies 2300
gas stipend

What about this?

```
// SPDX-License-Identifier: GPL-3.0

contract SendContract {
    address payable public richest;
    uint public mostSent;
    error NotEnoughEther();
    constructor() payable {
        richest = payable(msg.sender);
        mostSent = msg.value;
    }
    function becomeRichest() public payable {
        if (msg.value <= mostSent) revert NotEnoughEther();
        richest.transfer(msg.value);
        richest = payable(msg.sender);
        mostSent = msg.value;
    }
}
```

What about this?

```
// SPDX-License-Identifier: GPL-3.0
contract SendContract {
    address payable public richest;
    uint public mostSent;
    error NotEnoughEther();
    constructor() payable {
        richest = payable(msg.sender);
        mostSent = msg.value;
    }
    function becomeRichest() public payable {
        if (msg.value <= mostSent) revert NotEnoughEther();
        richest.transfer(msg.value);
        richest = payable(msg.sender);
        mostSent = msg.value;
    }
}
```

Attacker locks up contract
by ensuring this reverts

Withdrawal Pattern

```
// SPDX-License-Identifier: GPL-3.0
contract WithdrawalContract {
    address public richest;
    uint public mostSent;
    mapping (address => uint) pendingWithdrawals;
    error NotEnoughEther();

    constructor() payable {
        richest = msg.sender;
        mostSent = msg.value;
    }
    function becomeRichest() public payable {
        if (msg.value <= mostSent) revert NotEnoughEther();
        pendingWithdrawals[richest] += msg.value;
        richest = msg.sender;
        mostSent = msg.value;
    }
    function withdraw() public {
        uint amount = pendingWithdrawals[msg.sender];
        // Remember to zero the pending refund before
        // sending to prevent re-entrancy attacks
        pendingWithdrawals[msg.sender] = 0;
        payable(msg.sender).transfer(amount);
    }
}
```

Attacker can only hinder themselves if this reverts.

Mappings

Mappings

Values for (`mapping(uint256 => uint256) private map;`) are stored at locations:

```
storage[keccak256(key . storage slot number)] = value
```

Note:

- “.” is concatenation.
- Nothing is stored at `storage[storage slot number]`

Mappings

```
// SPDX-License-Identifier: GPL-3.0
pragma solidity >=0.6.0 <0.9.0;
contract Map {
    mapping (uint256 => uint256)[] private array;
    function allocate(uint newMaps) public {
        for (uint256 i = 0; i < newMaps; i++)
            array.push();
    }
    function writeMap(uint map, uint key, uint value) public {
        array[map][key] = value;
    }
    function readMap(uint map, uint key) public view returns (uint) {
        return array[map][key];
    }
    function eraseMaps() public {
        delete array;
    }
}
```

Mappings

```
// SPDX-License-Identifier: GPL-3.0
pragma solidity >=0.6.0 <0.9.0;
contract Map {
    mapping (uint256 => uint256)[] private array;
    function allocate(uint newMaps) public {
        for (uint256 i = 0; i < newMaps; i++)
            array.push();
    }
    function writeMap(uint map, uint key, uint value) public {
        array[map][key] = value;
    }
    function readMap(uint map, uint key) public view returns (uint) {
        return array[map][key];
    }
    function eraseMaps() public {
        delete array;
    }
}
```

Calling delete array does not delete any of the elements of the maps.

Mappings

```
// SPDX-License-Identifier: GPL-3.0
pragma solidity >=0.6.0 <0.9.0;
contract Map {
    mapping (uint256 => uint256)[] private array;
    function allocate(uint newMaps) public {
        for (uint256 i = 0; i < newMaps; i++)
            array.push();
    }
    function writeMap(uint map, uint key, uint value) public {
        array[map][key] = value;
    }
    function readMap(uint map, uint key) public view returns (uint) {
        return array[map][key];
    }
    function eraseMaps() public {
        delete array;
    }
}
```

Imagine:

- allocate(1)
- writeMap(0, 2, 5)
- eraseMaps()
- allocate(1)
- readMap(0, 2)

Integers

Integers

Integer arithmetic:

- Numbers in Solidity are integers.
- Hence, $3 / 2 = 1$.

Integers

Integer arithmetic:

- Numbers in Solidity are integers.
- Hence, $3 / 2 = 1$.

```
contract MyContract {
    address[] public owners;
    uint256 holdings;

    function payoutEqually() external {
        uint256 numOwners = owners.length;
        uint256 payoutForEachOwner = holdings / numOwners;
        ...
    }
}
```

Integers

Integer arithmetic:

- Numbers in Solidity are integers.
- Hence, $3 / 2 = 1$.

```
contract MyContract {  
    address[] public owners;  
    uint256 holdings;  
  
    function payoutEqually() external {  
        uint256 numOwners = owners.length;  
        uint256 payoutForEachOwner = holdings / numOwners;  
        ...  
    }  
}
```

If holdings isn't divisible by numOwners, then there will be some value left in the contract.

Integers and Voting

```
contract MyContract {
    address[] public voters;
    mapping(address => bool) votes;

    function countVotes() external view returns (bool) {
        uint256 votesFor = 0;
        for (uint256 i = 0; i < voters.length; i++) {
            if (votes[voters[i]]) {
                votesFor++;
            }
        }
        return votesFor >= voters.length / 2;
    }
}
```

Integers and Voting

```
contract MyContract {
    address[] public voters;
    mapping(address => bool) votes;

    function countVotes() external view returns (bool)
    {
        uint256 votesFor = 0;
        for (uint256 i = 0; i < voters.length; i++) {
            if (votes[voters[i]]) {
                votesFor++;
            }
        }
        return votesFor >= voters.length / 2;
    }
}
```

Minimum votesFor to return true	voters.length
0	1
1	2
1	3
2	4
2	5
3	6

Integers and Voting

```
contract MyContract {
    address[] public voters;
    mapping(address => bool) votes;

    function countVotes() external view returns (bool)
    {
        uint256 votesFor = 0;
        for (uint256 i = 0; i < voters.length; i++) {
            if (votes[voters[i]]) {
                votesFor++;
            }
        }
        return votesFor > voters.length / 2;
    }
}
```

Minimum votesFor to return true	voters.length
1	1
2	2
2	3
3	4
3	5
4	6

Integers and Voting

```
contract MyContract {
    address[] public voters;
    mapping(address => bool) votes;

    function countVotes() external view returns (bool)
    {
        uint256 votesFor = 0;
        for (uint256 i = 0; i < voters.length; i++) {
            if (votes[voters[i]]) {
                votesFor++;
            }
        }
        return votesFor >= voters.length / 2 + 1;
    }
}
```

Minimum votesFor to return true	voters.length
1	1
1	2
2	3
2	4
3	5
3	6

Integers Underflows and Overflows

```
uint8 x = 255;  
uint8 y = 1;  
return x + y;
```

Integers Underflows and Overflows

```
uint8 x = 255;  
uint8 y = 1;  
return x + y;
```

By default, this overflow is checked and a revert is thrown as `panic(0x11)`

This could leave your contract in an unusable state.

Integers Underflows and Overflows

```
unchecked {  
    uint8 x = 255;  
    uint8 y = 1;  
    return x + y;  
}
```

unchecked stops the revert from being thrown BUT returns 0, which could cause other problems...

<https://docs.soliditylang.org/en/v0.8.15/security-considerations.html#two-s-complement-underflows-overflows>

tx.origin

tx.origin

What is this?

```
require(tx.origin == msg.sender);
```

tx.origin

What is this?

```
require(tx.origin == msg.sender, "Must be called from an EOA");
```

tx.origin

Imagine you have this contract

```
// SPDX-License-Identifier: GPL-3.0
pragma solidity >=0.7.0 <0.9.0;
// THIS CONTRACT CONTAINS A BUG - DO NOT USE
contract TxUserWallet {
    address owner;

    constructor() {
        owner = msg.sender;
    }

    function transferTo(address payable dest, uint amount) public {
        require(tx.origin == owner);
        dest.transfer(amount);
    }
}
```

tx.origin

and you are tricked into
sending Ether to this contract

```
// SPDX-License-Identifier: GPL-3.0
pragma solidity >=0.7.0 <0.9.0;
interface TxUserWallet {
    function transferTo(address payable dest, uint amount) external;
}
contract TxAttackWallet {
    address payable owner;
    constructor() {
        owner = payable(msg.sender);
    }
    receive() external payable {
        TxUserWallet(msg.sender).transferTo(owner, msg.sender.balance);
    }
}
```

Understanding errors and error propagation

Reverting

Solidity	EVM Opcode	Return data
revert(string)	REVERT	Error(string)
revert CustomErr(params);	REVERT	CustomErr(params)
require(condition)	REVERT	Error("")
require(condition, string)	REVERT	Error(string)
assert(condition)	REVERT	Panic(0x01)

Reverting

Solidity	EVM Opcode	Return data
revert(string)	REVERT	Error(string)
revert CustomErr(params);	REVERT	CustomErr(params)
require(condition)	REVERT	Error("")
require(condition, string)	REVERT	Error(string)
assert(condition)	REVERT	Panic(0x01)

Asserts are for checking things that should always be true.
Use in conjunction with code analysis tools.

Panic Errors

Panic Code	Reason
0x00	Used for generic compiler inserted panics.
0x01	If you call assert with an argument that evaluates to false.
0x11	If an arithmetic operation results in underflow or overflow outside of an unchecked { ... } block.
0x12	If you divide or modulo by zero.
0x21	If you convert a value that is too big or negative into an enum type.
0x22	If you access a storage byte array that is incorrectly encoded.
0x31	If you call .pop() on an empty array.
0x32	If you access an array, bytesN or an array slice at an out-of-bounds or negative index
0x41	If you allocate too much memory or create an array that is too large.
0x51	If you call a zero-initialized variable of internal function type.

Other Circumstances of Error()

Other situations in which REVERTS occurs with no message:

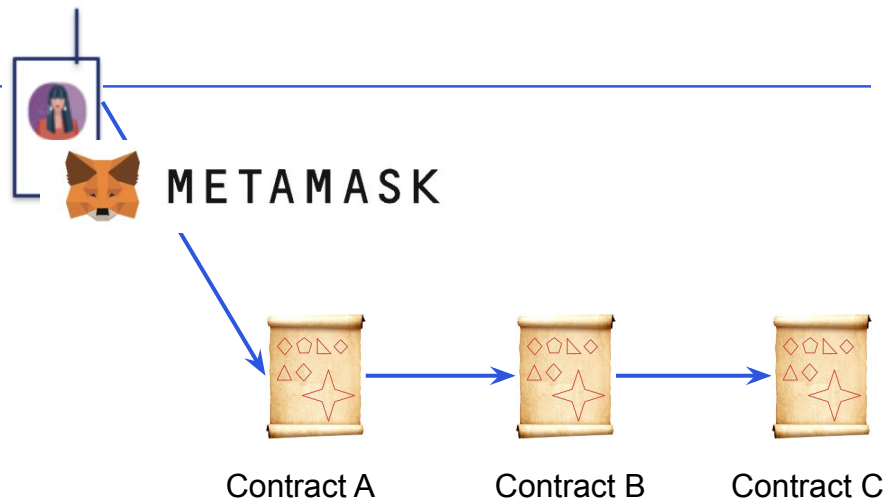
- If you perform an external function call targeting a contract that contains no code.
- If your contract receives Ether via a public function without payable modifier (including the constructor and the fallback function).
- If your contract receives Ether via a public getter function.
- If not enough calldata is supplied for a function call.

Error Propagation

REVERTS in a cross-contract call **can** cascade, failing the entire transaction.

They typically do.

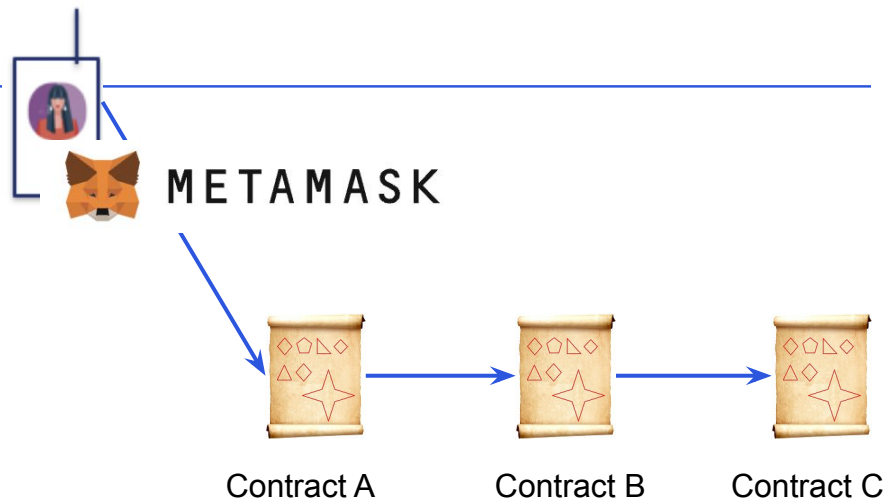
But they don't have to.



Error Propagation

Failures always cascade in these situations:

- `.transfer()` fails.
- Contract create via **new** fails.
- A typical function call fails.
 - `contractB.func(param1)`

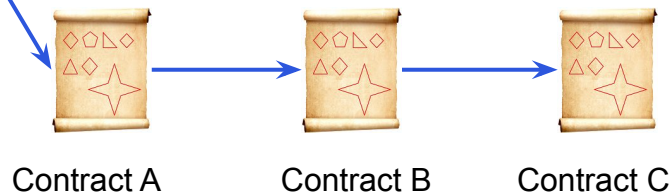


Error Propagation



Failures can be caught like this:

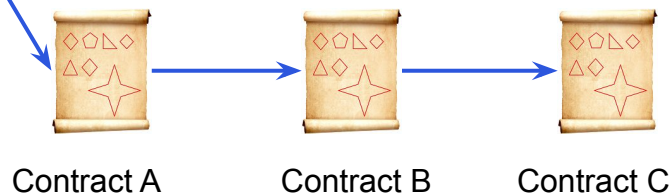
```
bool success;  
bytes memory returnEncoded;  
(success, returnEncoded) = address(this).call(abi.encodeWithSelector(this.func.selector));
```



Error Propagation

What about this?

```
contract TryCatch {  
    uint256 val;  Other otherContract;  
  
    error Panic1(uint256 _code);  
  
    function stuff(uint256 _param) external {  
        try otherContract.func(_param) returns (uint256 v) {  
            val = v;  
        } catch Error(string memory reason) {  
            revert(reason);  
        } catch Panic(uint256 errorCode) {  
            revert Panic1(errorCode);  
        } catch (bytes memory /* lowLevelData */) {  
        }  
    }  
}
```



Timestamp Manipulation

PoW (pre-merge)

PoW block creators set the block creation time (valid as long as within 15 second window).

This is changes the value returned in Solidity code by `block.timestamp / now()`

PoS (post-merge)

Block timestamp is set to the start time of the slot.

There is no ability for PoS validators to manipulate it.

block.number & Block Stuffing Attacks

Fomo3D is a gambling game where players buyetc etc etc
and the winner is the last account that puts a bid in.

```
if (block.number < WIN_PERIOD + lastBidTime) {  
    // you win  
}
```

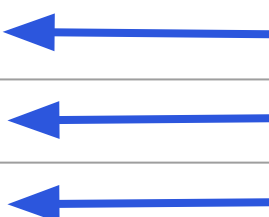
Attacker did a DoS attack, stuffing blocks for WIN_PERIOD blocks.

Complex Stuff

Storage Collisions

Storage Layout: Delegate Call

Memory Slot	Contract A	Contract B
0	val1	bVal
1	val2	addr
2		balances



For full talk see:

<https://www.youtube.com/watch?v=FnE9tpcMY88>

For Delegate Call and Call Code, Contract B executes in the context of Contract A.

Open Zeppelin Upgrade Proxy

Storage Slot	Proxy	Contract E v1	Contract E v2
0		val1	val1
1		val2	val2
2		val4	val3
3		val5	val4
4		val7	val5
5		val8	val7
6			val8
0x360894a13b...	Implementation		

AudiusAdminUpgradeabilityProxy



Spreek
@spreekaway

...

Audius governance attacked for \$6m (attackers profit \$1m)

Audius's upgradable proxy used storage slot 0.

<https://rekt.news/audius-rekt/>



etherscan.io

Ethereum Transaction Hash (Txhash) Details | Etherscan

Ethereum (ETH) detailed transaction info for txhash

0x4227bca8ed4b8915c7eec0e14ad3748a88c4371d4176e7...

2022 · Twitter Web App

```
12  */
13  contract AudiusAdminUpgradeabilityProxy is UpgradeabilityProxy {
14      address private proxyAdmin;
15      string private constant ERROR_ONLY_ADMIN = "
```

Attacker changed **proxyAdmin** by writing to contract being proxied, storage slot 0.

Storage Collisions Type 2

```

contract Storage {
    mapping (uint256 => uint256) public map;

    function setVal(uint256 _key, uint256 _val) external {
        map[_key] = _val;
    }

    function setVal1(uint256 _key, uint256 _val) external {
        uint256 mapSlot = 0;
        bytes32 slot = keccak256(abi.encode(_key, mapSlot));
        getUint256Slot(slot).value = _val;
    }

    function getVal(uint256 _key) external view returns (uint256) {
        uint256 mapSlot = 0;
        bytes32 slot = keccak256(abi.encode(_key, mapSlot));
        return getUint256Slot(slot).value;
    }

    struct Uint256Slot {
        uint256 value;
    }

    function getUint256Slot(bytes32 slot) internal pure returns (Uint256Slot storage r) {
        /// @solidity memory-safe-assembly
        assembly {
            r.slot := slot
        }
    }
}

```



```
contract Storage {  
    mapping (uint256 => uint256) public map;
```

```
    function setVal(uint256 _key, uint256 _val) external {  
        map[_key] = _val;  
    }
```

```
    function setVal1(uint256 _key, uint256 _val) external {  
        uint256 mapSlot = 0;  
        bytes32 slot = keccak256(abi.encode(_key, mapSlot));  
        getUint256Slot(slot).value = _val;  
    }
```

```
    function getVal(uint256 _key) external view returns (uint256)  
    {  
        uint256 mapSlot = 0;  
        bytes32 slot = keccak256(abi.encode(_key, mapSlot));  
        return getUint256Slot(slot).value;  
    }
```

```
    struct Uint256Slot {  
        uint256 value;  
    }
```

```
    function getUint256Slot(bytes32 slot) internal pure returns (Uint256Slot storage r) {  
        /// @solidity memory-safe-assembly  
        assembly {  
            r.slot := slot  
        }  
    }
```

map can be accessed by

- map(uint256)
- setVal(uint256, uint256)

map can also be accessed by

- getVal(uint256)
- setVal1(uint256, uint256)

```
contract Storage {  
    mapping (uint256 => uint256) public map;
```

```
    function setVal(uint256 _key, uint256 _val) external {  
        map[_key] = _val;  
    }
```

```
    function setVal1(uint256 _key, uint256 _val) external {  
        uint256 mapSlot = 0;  
        bytes32 slot = keccak256(abi.encode(_key, mapSlot));  
        getUint256Slot(slot).value = _val;  
    }
```

```
    function getVal(uint256 _key) external view returns (uint256) {  
        uint256 mapSlot = 0;  
        bytes32 slot = keccak256(abi.encode(_key, mapSlot));  
        return getUint256Slot(slot).value;  
    }
```

```
    struct Uint256Slot {  
        uint256 value;  
    }
```

```
    function getUint256Slot(bytes32 slot) internal pure returns (Uint256Slot storage r) {  
        /// @solidity memory-safe-assembly  
        assembly {  
            r.slot := slot  
        }  
    }
```

An audit of the code searching for code that touches **map** will not find setVal1 or getVal.

Changing Bytecode of a Contract

Create2

- Create a contract at a known address.
- Contract address depends on:
 - Address of creator.
 - Bytecode of contract.
 - Parameters of constructor.
 - Salt value supplied by creator.
- Use-case:
 - Contracts that act as judges for off-chain interactions, which only need to be created if there is a dispute.

Example of using create2

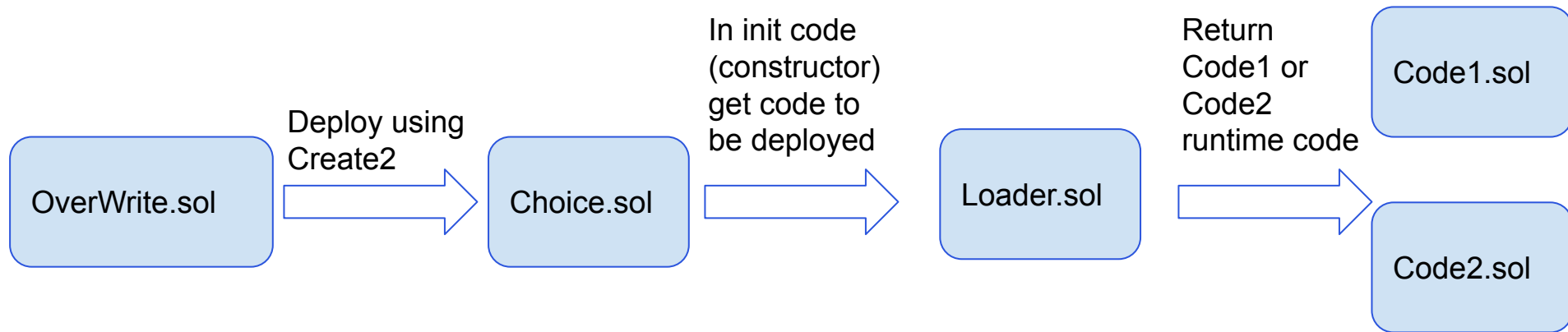
```
contract D {
    address public owner;
    constructor() {
        owner = msg.sender;
    }

    function withdrawal() external {
        selfdestruct payable(owner);
    }
}
```

```
contract CreateMe {
    function create() public {
        bytes32 salt = bytes32(0x01);
        D d = new D(salt: salt)(arg);
    }

    function predictAddr() pure returns (address) {
        bytes32 salt = bytes32(0x01);
        address predictedAddress =
            address(uint160(uint(keccak256(
                abi.encodePacked(
                    bytes1(0xff),
                    address(this),
                    salt,
                    keccak256(abi.encodePacked(
                        type(D).creationCode))
                )))));
        return predictedAddress;
    }
}
```

Changing Bytecode of a Contract



Get.sol, Code1.sol and Code2.sol

```
interface Get {  
    function get() external pure  
        returns (uint256);  
  
    function withdrawal() external;  
}
```

```
contract Code1 is Get {  
    function get() external pure  
        returns (uint256) {  
        return 1;  
    }  
    function withdrawal() external {  
        selfdestruct(payable(address(this)));  
    }  
}  
  
contract Code2 is Get {  
    function get() external pure  
        returns (uint256) {  
        return 2;  
    }  
    function withdrawal() external {  
        selfdestruct(payable(address(this)));  
    }  
}
```


Choice.sol

```
contract Choice {
    constructor() {
        address _loader = 0x9d83e140330758a8fFD07F8Bd73e86ebcA8a5692;
        Loader loader = Loader(_loader);
        bytes memory code = loader.getCode();

        uint256 memOfs = dataPtr(code);
        uint256 len = code.length;
        copy(memOfs, 0, len);

        assembly {
            return(0, len)
        }
    }
}
```

```
// From
https://github.com/ethereum/solidity-examples/blob/master/src/unsafeMemory.sol

// Size of a word, in bytes.
uint internal constant WORD_SIZE = 32;

// Copy 'len' bytes from memory address 'src', to address 'dest'.
// This function does not check the or destination, it only copies
// the bytes.
function copy(uint src, uint dest, uint len) internal pure {
    // Copy word-length chunks while possible
    for (; len >= WORD_SIZE; len -= WORD_SIZE) {
        assembly{
            mstore(dest, mload(src))
        }
        dest += WORD_SIZE
        src += WORD_SIZE
    }

    // Copy remaining bytes
    uint mask = 256 - ((WORD_SIZE - len) - 1);
    assembly{
        let srcpart := and(mload(src), not(mask))
        let destpart := and(mload(dest), mask)
        mstore(dest, or(destpart, srcpart))
    }
}

function dataPtr(bytes memory bts) internal pure returns (uint addr)
{
    assembly{
        addr := add(bts, /*BYTES_HEADER_SIZE*/32)
    }
}
}
```

Overwrite.sol

Could be improved by supplying salt value.

```
contract OverWrite {
    address public choiceContract;

    function create2() external {
        bytes32 salt = bytes32(uint256(0x01));
        Choice d = new Choice(salt: salt)();
        choiceContract = address(d);
    }

    function c() public view returns (address) {
        bytes32 salt = bytes32(uint256(0x01));
        address predictedAddress = address(uint160(uint(keccak256(abi.encodePacked(
            bytes1(0xff),
            address(this),
            salt,
            keccak256(abi.encodePacked(
                type(Choice).creationCode
            ))
        )))));
        return predictedAddress;
    }
}
```

Thoughts

- Loader could have had a function `setCode(bytes _code)` to set-up the code to be used, rather than hard code it.
- The constructor in `Choice` returned before the whole constructor was called. If there were other constructors in other inherited contracts, then those other constructors might not have been called.
- Block explorers would detect that the source code and the deployed contract code don't match. An attacker would need to have Loader initially return code that matched the source code users expect.

Is this a problem???

- An audit of the code would identify the “interesting” constructor code in Choice.sol.
- However, the constructor code in Choice.sol could have been buried very deep in the contract hierarchy.
- If the attacker could sneak this malicious code in, then surely they could sneak other malicious code in, and not worry about going to such lengths as these to switch the code.
- All storage related to the contract would be lost when selfdestruct is called.

ⓘ Warning

There are some peculiarities in relation to salted creation. A contract can be re-created at the same address after having been destroyed. Yet, it is possible for that newly created contract to have a different deployed bytecode even though the creation bytecode has been the same (which is a requirement because otherwise the address would change). This is due to the fact that the compiler can query external state that might have changed between the two creations and incorporate that into the deployed bytecode before it is stored.

Final word on Create2: Upgrade

Don't use this as a way of upgrading your contracts!

All contract storage would be lost in the upgrade (that is, the contract data)

Value Transfer when payable is blocked

Value Transfer when Payable is Blocked

Value transfer to contract:

- Is there a way of sending Eth to a contract without calling the receive() function?

That is, getting around:

```
contract MyContract {  
}
```

Value Transfer when Payable is Blocked

Value transfer to contract:

- Is there a way of sending Eth to a contract without calling the receive() function?

That is, getting around:

```
contract MyContract {  
    receive() external payable {  
        revert("I don't accept Eth!");  
    }  
}
```

which is the same as:

```
contract MyContract {  
}
```


Value Transfer when Payable is Blocked

Value transfer to contract:

- Is there a way of sending Eth to a contract without calling the receive() function?

That is, getting around:

```
contract MyContract {  
    receive() external payable {  
        revert("I don't accept Eth!");  
    }  
}
```

yes

```
address payable addr = payable(address(contractAddr));  
selfdestruct(addr);
```

Create2

Create2 allows the address of a contract to be determined prior to it being deployed.

Eth could be sent to the address before the contract is deployed.

Force Feeding Attacks

Force Feeding Attacks use the mechanisms described in previous slides to target code that uses the contract balance to determine actions.

```
if (address(this).balance > 10) {  
    // do something  
}
```

Obscure Functionality

What is going on here?

```
function doStuff(bool _checker, uint256 _val) external {  
    _checker.doSomethingComplex(_val);  
}
```

Using adds capabilities / functions to builtin types

```
library SomeLibrary {
  function doSomethingComplex(bool _self, uint256 _value) external {
    // do stuff
  }
}

contract BuiltIn {
  using SomeOtherLibrary for *;

  function doStuff(bool _checker, uint256 _val) external {
    _checker.doSomethingComplex(_val);
  }
}
```

Function Types

```
// SPDX-License-Identifier: BSD
pragma solidity ^0.8.0;

contract FunctionType {
    function(uint256) internal pure returns(uint256) private funcToUse;

    function change(uint256 _choice) external {
        funcToUse = _choice == 1 ? func1 : func2;
    }

    function callFunc(uint256 _val) external view returns(uint256) {
        return funcToUse(_val);
    }

    function func1(uint256 _val) internal pure returns(uint256) {
        return _val + 1;
    }

    function func2(uint256 _val) internal pure returns(uint256) {
        return _val + 2;
    }
}
```

```
pragma solidity ^0.5.0;
```

```
contract GuessTheNumber {
```

```
    uint _secretNumber;  
    address payable _owner;  
    event success(string);  
    event wrongNumber(string);
```

```
    constructor(uint secretNumber) payable public {  
        _secretNumber = secretNumber;  
        _owner = msg.sender;  
    }
```

```
    function getBalance() view public returns (uint) {  
        return address(this).balance;  
    }
```

```
    function guess(uint n) payable public {  
        require(msg.value == 1 ether);  
        uint p = address(this).balance;  
        checkAndTransferPrize(/*The prize*/p , n/*guessed number*/  
                               /*The user who should benefit */ ,msg.sender);  
    }
```

```
    function checkAndTransferPrize(uint p, uint n, address payable guesser) internal returns(bool){  
        if(n == _secretNumber) {  
            guesser.transfer(p);  
            emit success("You guessed the correct number!");  
        }  
        else {  
            emit wrongNumber("You've made an incorrect guess!");  
        }  
    }
```

```
    function kill() public {  
        require(msg.sender == _owner);  
        selfdestruct(_owner);  
    }
```

Read through this code...

How does it work?

Any issues?


```
function guess(uint n) payable public
```

```
{
```

```
    require(msg.value == 1 ether);
```

```
    uint p = address(this).balance;
```

```
    checkAndTransferPrize(/*The prize [U+202E]/*rebmun desseug*/n , p/*[U+202D]
```

```
    |      /*The user who should benefit */ ,msg.sender);
```

```
}
```

```
function checkAndTransferPrize(uint p, uint n, address payable guesser) internal returns(bool)
```

```
{
```

```
    if(n == _secretNumber)
```

```
    {
```

```
        guesser.transfer(p);
```

```
        emit success("You guessed the correct number!");
```

```
    }
```

```
    else
```

```
    {
```

```
        emit wrongNumber("You've made an incorrect guess!");
```

```
    }
```

```
}
```

```
function kill() public
```

```
{
```

```
    require(msg.sender == _owner);
```

```
    selfdestruct(_owner);
```

```
}
```

Right to Left Override

Left to Right Override

Obscure Functionality

There is some really obscure, little used functionality in Solidity.

I think using obscure functionality is problematic for these reasons:

- It obfuscates the code.
- Most Solidity devs may not fully understand the full intention of the code.
- This may lead to bugs / security issues not being detected.

Security Tools

solc Compiler

solc Compiler

- Understand and fix compilation warnings.
- Lots of historic security issues (<https://swcregistry.io/>) have been **resolved** in newer versions of the compiler by adding compiler fatal errors and warnings.

Satisfiability Modulo Theories (SMT) Checker

Checker for some security issues can be enabled in the solc compiler.

- Whether requires or asserts will or might fail.
- Arithmetic underflow and overflow.
- Division by zero.
- Trivial conditions and unreachable code.
- Popping an empty array.
- Out of bounds index access.
- Insufficient funds for a transfer.

Other Tools

Lots of tools.

- **MythX** - MythX is a professional-grade cloud service that finds **common security bugs** and **verify the correctness of smart contracts** from mythx.io.
- **Mythril** - The Swiss army knife for smart contract security.
- **Slither** - Static analysis framework with detectors for many common vulnerabilities, tracking capabilities and is written in Python.
- **Contract-Library** - Decompiler and security analysis tool for all deployed contracts.
- **MadMax** - Static analysis tool for gas DoS vulnerabilities.
- **Gigahorse** - Fast binary lifter and program analysis framework written in Datalog.
- **Echidna** - The only available fuzzer for Ethereum software. Uses property testing to generate malicious inputs that break smart contracts.
- **Manticore** - Dynamic binary analysis tool with **EVM support**.
- **Oyente** - Analyze Ethereum code to find common vulnerabilities, based on this [paper](#).
- **Securify** - Fully automated online static analyzer for smart contracts, providing a security report based on vulnerability patterns.
- **SmartCheck** - Static analysis of Solidity source code for security vulnerabilities and best practices.
- **Octopus** - Security Analysis tool for Blockchain Smart Contracts with support of EVM and (e)WASM.
- **sFuzz** - Efficient fuzzer inspired from AFL to find common vulnerabilities.
- **Vertigo** - Mutation Testing for Ethereum Smart Contracts.

I haven't tried any of these tools.

I don't know:

- * which are free.
- * which support recent Solidity versions.
- * which work the best.

Formally Verified Smart Contracts

TrustWorthy Smart Contracts Team at ConsenSys R&D

Formal Verification of Smart Contracts

Deductive Verification of Smart Contracts with Dafny

Franck Cassez (✉)^{1,10}, Joanne Fuller¹, and Horacio Mijail Antón Quiles¹

ConsenSys, New York, USA

franck.cassez@consensys.net joanne.fuller@consensys.net

horacio.mijail@consensys.net

Abstract. We present a methodology to develop verified smart contracts. We write smart contracts, their specifications and implementations in the verification-friendly language DAFNY. In our methodology the ability to write specifications, implementations and to reason about correctness is a primary concern. We propose a simple, concise yet powerful solution to reasoning about contracts that have *external calls*. This includes arbitrary re-entrancy which is a major source of bugs and attacks in smart contracts. Although we do not yet have a compiler from DAFNY to EVM bytecode, the results we obtain on the DAFNY code can



Joanne Fuller



Horacio Mijail
Anton Quiles



Milad Ketabi



Franck
Cassez




David Pearce



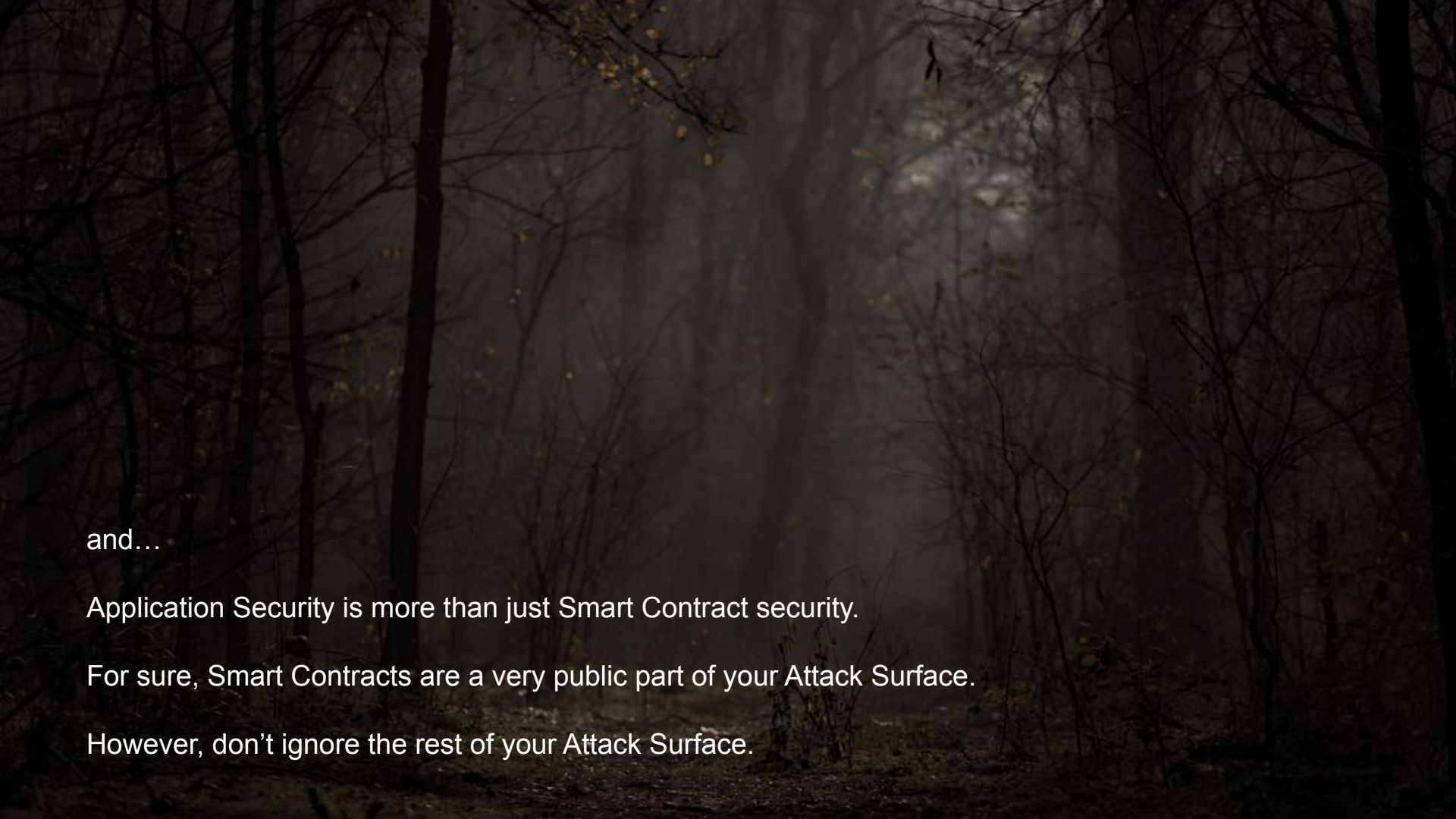
<https://arxiv.org/abs/2208.02920>

Summary

A dark, atmospheric photograph of a forest. The scene is dimly lit, with a misty or foggy atmosphere. Tall, slender tree trunks are visible, some with sparse, dark leaves. A path or clearing leads into the distance, where a faint light source, possibly the sun or moon, is visible through the trees. The overall mood is mysterious and somewhat ominous.

Smart Contract Security is very complex

This talk is just the start of your journey through the **Dark Forest**.

A dark, atmospheric photograph of a forest. The scene is dimly lit, with a misty or foggy atmosphere. Bare, dark tree trunks and branches are visible, some with a few small, yellowish leaves. A path or clearing leads into the distance, where a faint light source, possibly the sun or moon, is visible through the trees. The overall mood is mysterious and somewhat ominous.

and...

Application Security is more than just Smart Contract security.

For sure, Smart Contracts are a very public part of your Attack Surface.

However, don't ignore the rest of your Attack Surface.

Links

<https://consensys.net/blog/developers/solidity-best-practices-for-smart-contract-security/>

<https://docs.soliditylang.org/en/v0.8.15/security-considerations.html>

<https://consensys.github.io/smart-contract-best-practices/>

<https://medium.com/loom-network/how-to-secure-your-smart-contracts-6-solidity-vulnerabilities-and-how-to-avoid-them-part-1-c33048d4d17d>

<https://github.com/sigp/solidity-security-blog>

<https://101blockchains.com/smart-contract-best-practices/>

<https://betterprogramming.pub/solidity-smart-contract-security-preventing-reentrancy-attacks-fc729339a3ff>

<https://stackoverflow.com/questions/73011320/solidity-security-fallback-function-priority>

<https://swcregistry.io/>

Future Talks

August 17: Ethereum Reference Tests: what, how, why

- Horacio Mijail: 12:30pm Brisbane (GMT +10)

August 24: On the Security of Crosschain Transactions

- Ermyas Abebe: 12:30 pm: Brisbane (GMT+10)

Sept 7: Progress and Pitfalls on the Path to Decentralized Identity

- Paul Ashley: 12:30 pm: Brisbane (GMT+10)

Sept 21: Chainalysis

- Jacob Illum: 12:30 pm: Brisbane (GMT+10)

Oct 5: Socio-Legal aspects of Blockchain

- Lachlan Robb: 12:30pm: Brisbane (GMT+10)

Nov 30: DeDa: A Defi-enabled Data Marketplace for Personal Data Management

- Minfeng Qi: 12:30 pm: Brisbane (GMT+10)

You Tube, Slack, Meet-up, Example Code

YouTube: <https://www.youtube.com/c/ethereumengineeringgroup>

Slack invitation link:

https://join.slack.com/t/eth-eng-group/shared_invite/zt-48ggg3kk-bUT3PWRn16hCpFclbcZrvQ

Meet-up: <https://www.meetup.com/ethereum-engineering/>

Example code: <https://github.com/drinkcoffee/EthEngGroupSolidityExamples>

Formal Methods Reading Group: Join the Slack and the go to fm-reading-group to learn more.