Getting Started in Blockchain Security and Smart Contract Auditing

Beau Bullock
@dafthack

- Pentester / Red Team at Black Hills Information Security
- Author / Instructor of Breaching the Cloud Training
- Host / CoinSec Podcast
- Certs: OSCP, OSWP, GXPN, GPEN, GWAPT, GCIH, GCIA, GCFA, GSEC
- Speaker: WWHF, DerbyCon, Black Hat Arsenal, BSides, Hack Miami, RVASec
- Tool Developer: MailSniper, PowerMeta, DomainPasswordSpray, MSOLSpray, HostRecon, Check-LocalAdminHash, MFASweep
- Cyberpunk Synthwave Metal Producer (NOBANDWIDTH)

@BHInfoSecurity

Roadmap

- Why is blockchain security important?
- Introduction to smart contracts
- Blockchain hack case studies
- Smart contract exploit demo
- Additional resources to get started

Why Blockchain Security?

Top DeFi Hacks Since September 2020

- Poly Network - $611,000,000 | 10 Aug 2021
- EasyFi - $59,000,000 | 19 Apr 2021
- Uranium Finance - $57,200,000 | 28 Apr 2021
- PancakeBunny - $45,000,000 | 19 May 2021
- Kucoin - $45,000,000 | 29 Sep 2020
- Alpha Finance - $37,500,000 | 13 Feb 2021
- Meerkat Finance - BSC - $32,000,000 | 04 Mar 2021
- Spartan Protocol - $30,500,000 | 02 May 2021
- StableMagnet - $27,000,000 | 23 Jun 2021
- Paid Network - $27,000,000 | 05 Mar 2021
- Harvest Finance - $25,000,000 | 26 Oct 2020
- XToken - $24,000,000 | 12 May 2021
- Popsicle Finance - $20,000,000 | 03 Aug 2021
- Pickle Finance - $19,700,000 | 22 Nov 2020
- bEarn - $18,000,000 | 17 May 2021
- Furucombo - $14,000,000 | 27 Feb 2021
- Compounder Finance - $12,000,000 | 02 Dec 2020
- Value DeFi 3 - $11,000,000 | 7 May 2021
- Yearn - $11,000,000 | 05 Feb 2021
- Rari Capital - $10,000,000 | 8 May 2021
- Value DeFi 2 - $10,000,000 | 5 May 2021
- Cover - $9,400,000 | 29 Dec 2020
- Punk Protocol - $8,950,000 | 10 Aug 2021
- THORChain 2 - $8,000,000 | 22 Jul 2021
- Hack Epidemic (Origin Protocol) - $8,000,000 | 17 Nov 2020
- Anyswap - $7,900,000 | 10 Jul 2021
- Warp Finance - $7,800,000 | 18 Dec 2020
- BurgerSwap - Arcadia Group - $7,200,000 | 28 May 2021
- Value DeFi - $7,000,000 | 14 Nov 2020
- Alchemix - $6,500,000 | 16 Jun 2021
- Belt - $6,300,000 | 29 May 2021
- Bondly - $5,900,000 | 15 Jul 2021
- Roll - $5,700,000 | 14 Mar 2021
- THORChain - $5,000,000 | 15 Jul 2021
- Eleven Finance - $4,500,000 | 22 Jun 2021
- ChainSwap - $4,400,000 | 11 Jul 2021
- PancakeBunny 2 - $2,400,000 | 16 May 2021
- DODO - $2,000,000 | 09 Mar 2021
- Akropolis - $2,000,000 | 12 Nov 2020
- Levyathan - $1,500,000 | 30 Jul 2021
- The Big Combo (Growth DeFi) - $1,300,000 | 09 Feb 2021
- Autoshark - $745,000 | 24 May 2021
- Merlin Labs - $680,000 | 26 May 2021
- Merlin Labs 2 - $550,000 | 26 May 2021
- Merlin Labs - $330,000 | 29 Jun 2021
- Saddle Finance - $275,735 | 20 Jan 2021
- SafeDollar - $248,000 | 28 Jun 2021

More Than $1.2 Billion Stolen

Source: https://www.rekt.news

Growing Use Cases for Blockchain

- Currency
- Decentralized Finance (lending, token swaps, crowdfunding)
- Digital Identity
- Arts and collectibles (NFTs)
- Supply chain (sourcing transparency, medical supply tracking)
- Media (anti-piracy)
- Real Estate
- Gaming (play-to-earn, metaverse property ownership)
- …and more

A New Frontier

- Birth of a new industry
- Many projects racing to be "first to market"
- Established, well-known companies (Microsoft, IBM, Wal-Mart, Disney, etc.) are implementing different use cases
- More than just smart contract security…

Blockchain Elements That Need Securing

- Layer 1
  - The underlying blockchain protocol itself (Bitcoin Core, Geth)
- Layer 2
  - An overlaying network on top of layer 1 typically focused on scalability (Bitcoin Lightning network)
- Smart Contracts
  - Automatically executing programs deployed to the blockchain (tokens, dApps, NFTs, etc.)
- Software Wallets
  - Custodial vs. non-custodial
- Hardware Wallets
  - Physical devices for storing private keys that are then used to send and receive funds
- Mining Software
  - Programs used to run specialized hardware used to perform mining
- Centralized Exchanges
  - Typically require KYC (Coinbase, Binance, etc.)
- Decentralized Exchanges
  - "DeFi" exchanges typically built via Smart Contract w/ web3 front-end
- People
  - Social engineering, rug pulls, asset protection

Introduction to Smart Contracts

"Smart contract code is unforgiving. Every bug can lead to monetary loss. You should not treat smart contract programming the same way as general-purpose programming. Writing DApps in Solidity is not like creating a web widget in JavaScript. Rather, you should apply rigorous engineering and software development methodologies, as you would in aerospace engineering or any similarly unforgiving discipline. Once you "launch" your code, there's little you can do to fix any problems."

- Andreas M. Antonopoulos & Dr. Gavin Wood

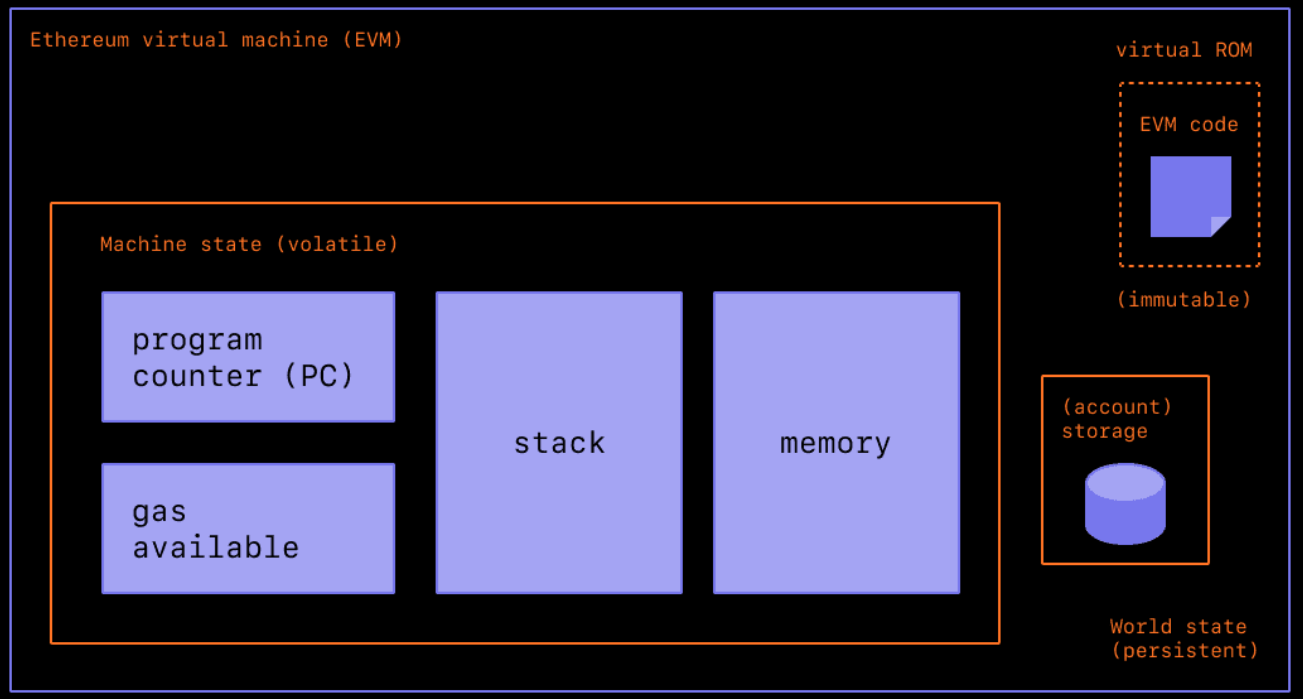
@BHInfoSecurity

What are Smart Contracts?

- Immutable computer programs
- Decentralized execution
- Run on virtual machines via node software
- Typically written in high-level languages
- Get compiled into bytecode prior to deployment to the blockchain
- Only run if they are called by a transaction

Ethereum Virtual Machine (EVM)

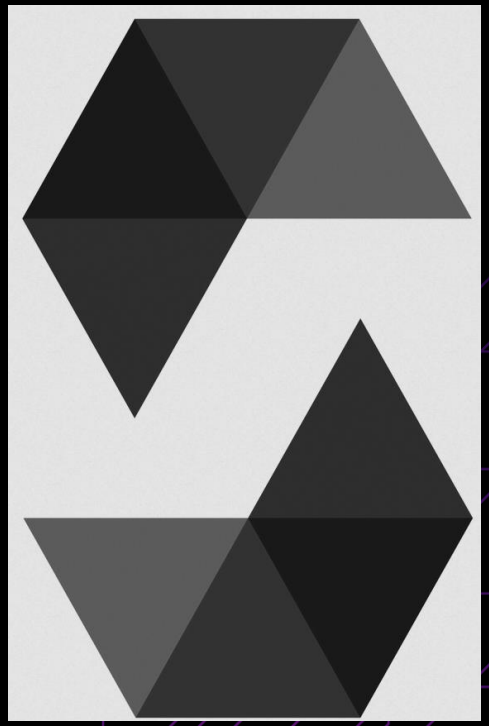
- Virtual machine on the blockchain
- Uses opcodes to execute tasks
  - Each opcode has a base gas cost
- All contracts are executed on all nodes
- Rules for changing the machine state are defined by the EVM
- Storage can be permanent or volatile (stack & memory)
- Accounts can be externally owned (EOA) or contract accounts



Source: https://ethereum.org/en/developers/docs/evm/

Solidity

- High-level language for writing smart contracts
- By far the most dominant language currently
- Syntax similar to JavaScript or C++
- Remix IDE for dev: https://remix.ethereum.org
- Code tends to be posted publicly & “verified” on sites like Etherscan



Solidity Example

```
// SPDX-License-Identifier: CC-BY-SA-4.0

// Version of Solidity compiler this program was written for
pragma solidity 0.6.4;


contract Faucet {
    // Accept any incoming amount
    receive() external payable {}

    // Give out ether to anyone who asks
    function withdraw(uint withdraw_amount) public {
        // Limit withdrawal amount
        require(withdraw_amount <= 10000000000000000);

        // Send the amount to the address that requested it
        msg.sender.transfer(withdraw_amount);
    }
}
```

Source: https://github.com/ethereumbook/ethereumbook/blob/develop/code/Solidity/Faucet.sol

Ethereum & Solidity Are Not Alone

Bitcoin + Stacks, RSK

Solana + Rust C, C++

Cardano + Plutus

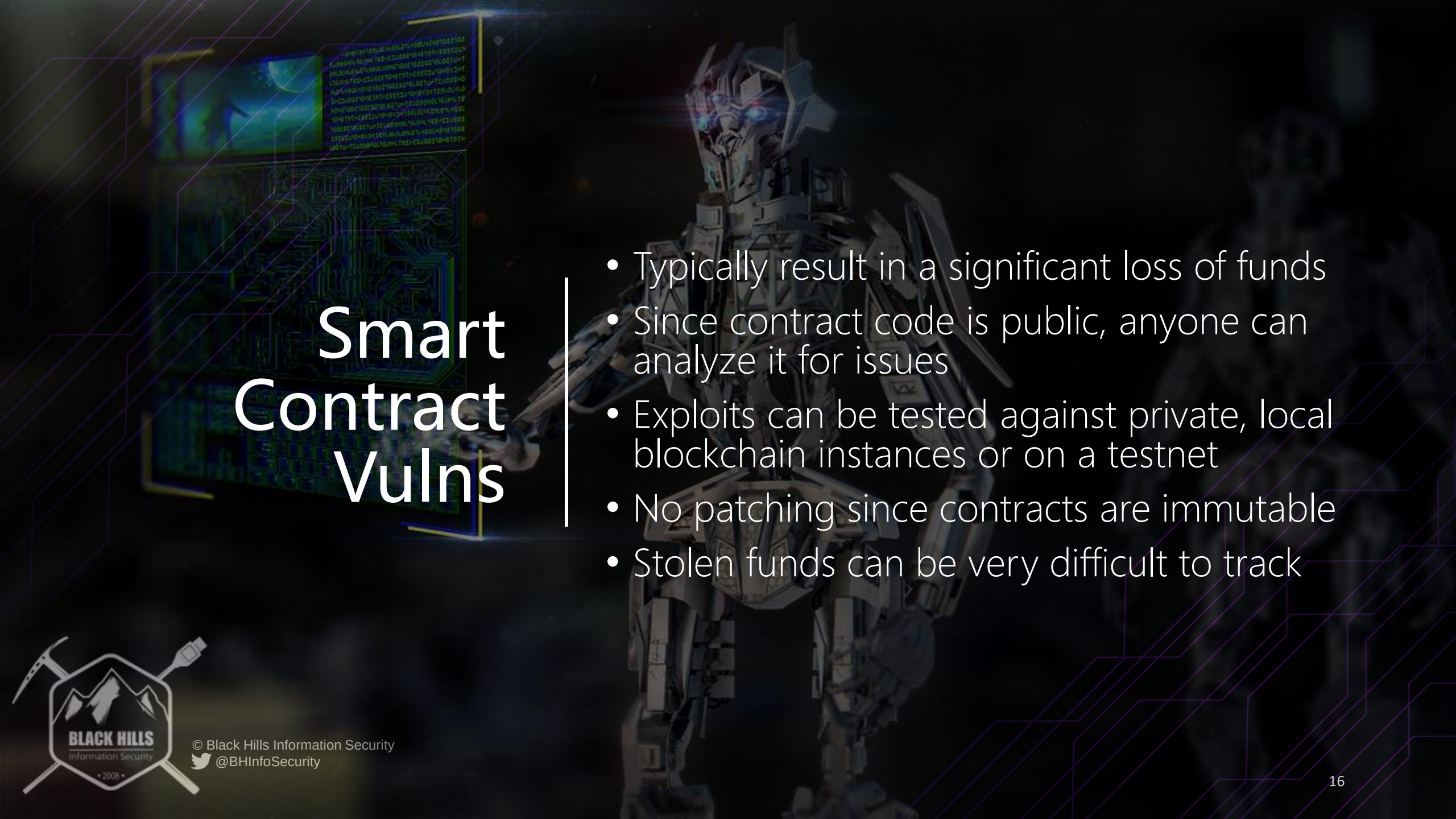
Avalanche + Solidity

Terra + Rust

Chia + Chialisp

Hyperledger Fabric + Go, JavaScript

Nem + Java

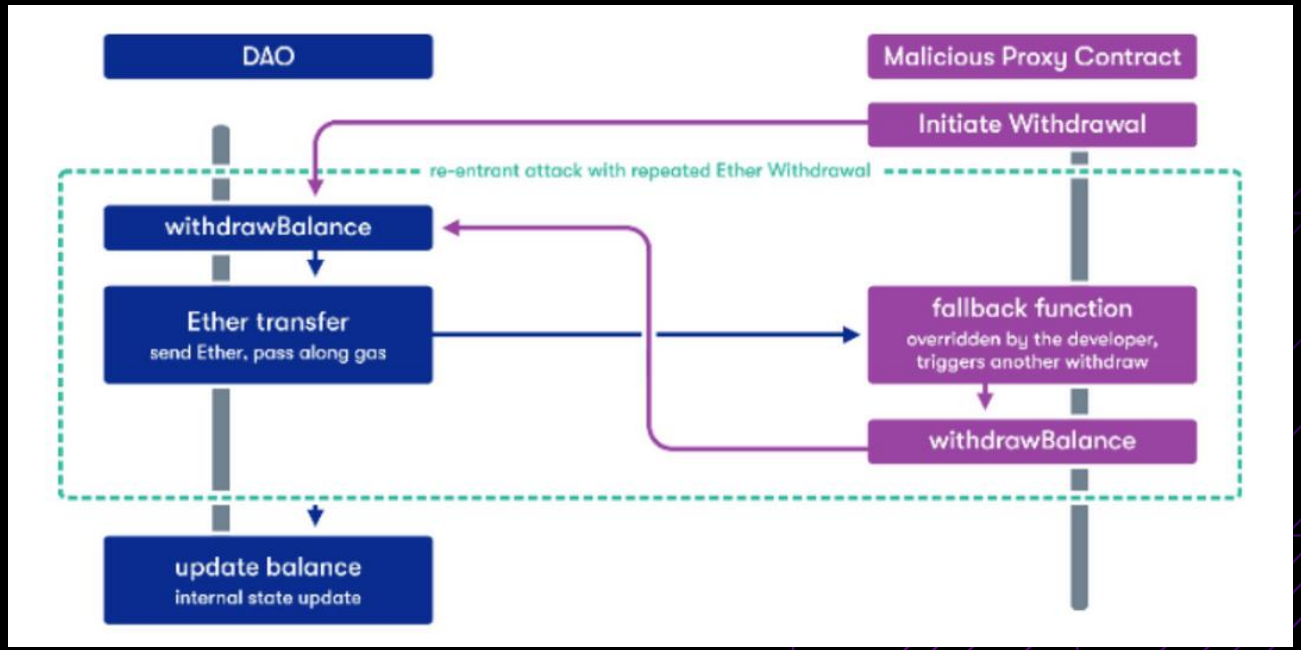


Smart Contract VuIns

- Typically result in a significant loss of funds
- Since contract code is public, anyone can analyze it for issues
- Exploits can be tested against private, local blockchain instances or on a testnet
- No patching since contracts are immutable
- Stolen funds can be very difficult to track

Smart Contract Vulns: Reentrancy

- Vulnerability where a function can be re-entered into prior to completion
- Vuln contract has a function that allows for withdrawing Ether prior to the balance update
- Attacker initiates a call to the vulnerable function from a malicious contract
- Target contract sends Ether to Attacker contract triggering fallback function
- A `fallback function` re-enters into the withdraw function causing a loop until all funds are drained

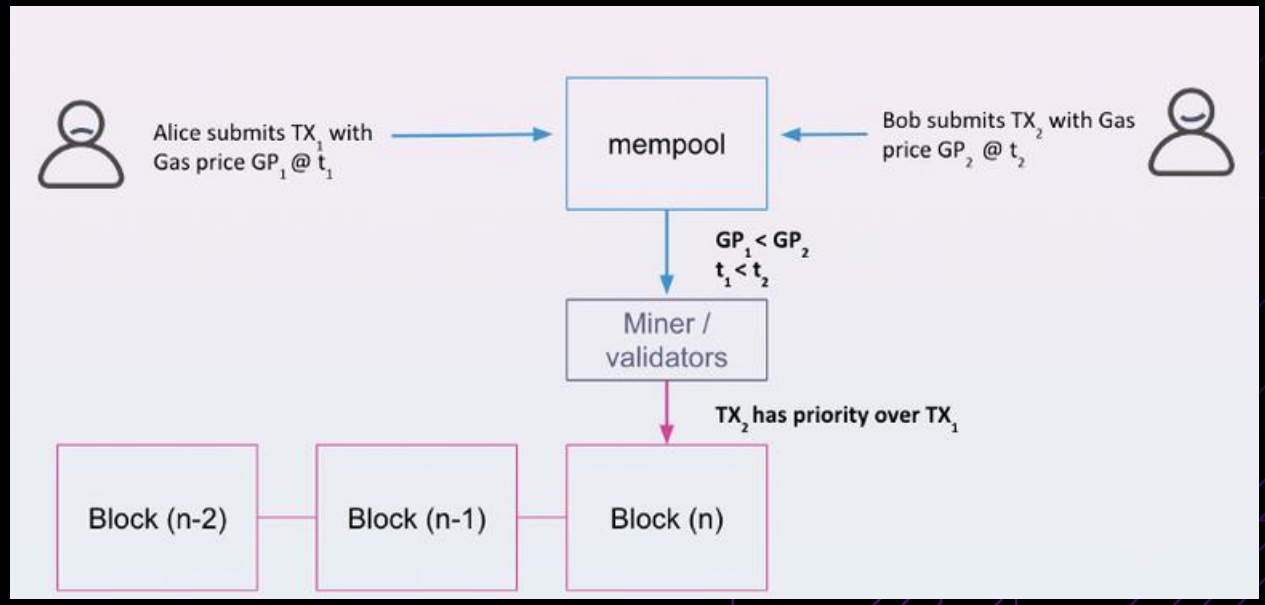


Source: https://quantstamp.com/blog/what-is-a-re-entrancy-attack

Real World Example: The DAO Hack ($150 Million Stolen)
https://hackingdistributed.com/2016/06/18/analysis-of-the-dao-exploit/

Smart Contract Vulns: Front-Running

- Prior to a block being mined all transactions are visible in the **mempool**
- Observers can see these transactions and react
- Miners give priority to higher gas prices so an attacker can have their transaction mined before the original sender by paying more



Source: https://blog.enigma.co/preventing-dex-front-running-with-enigma-df3f0b5b9e78

Real World Example: Bancor
https://hackernoon.com/front-running-bancor-in-150-lines-of-python-with-ethereum-api-d5e2bfd0d798

Smart Contract Vulns: Integer Overflow and Underflow

- Overflows and underflows can occur by performing arithmetic operations that exceed maximum or minimum integer values
- Unsigned Integers (`uint`)
  - Range is: 0 to $(2^{256} - 1)$
- Signed Integers (`int`)
  - Range is: -2^{255} to $(2^{256} - 1)$
- Example uint overflow: $1 + (2^{256} - 1) = 0$
- Can be exploited to bypass certain security checks

Real World Example: BEC Token
https://techcrunch.com/2018/04/25/overflow-error-shuts-down-token-trading/

Smart Contract Vulns: Denial-of-Service

- Block Gas Limit
  - The Ethereum network specifies a block gas limit
  - When contracts are called each action performed requires a certain amount of gas
  - If the block limit is exceeded the transaction will revert
  - Can lead to issues that are even non-malicious
- Unexpected Revert
  - Contract attempts to send funds but malicious contract reverts any payments

Real World Example: GovernMental
https://hackernoon.com/smart-contract-attacks-part-2-ponzi-games-gone-wrong-d5a8b1a98dd8

Smart Contract Vulns: Access Control

- Governs who can transfer assets, mint tokens, vote on proposals, freeze transfers, etc.
- Ownership over a contract allows administrative tasks
- By default the **`owner`** is the account that deployed it
- Ownership can potentially be transferred via exploit, lack of privileged action checks, or even abused by owners themselves
- Can also delay actions using a timelock

```
contract Fallout {

  using SafeMath for uint256;
  mapping (address => uint) allocations;
  address payable public owner;


  /* constructor */
  function Fal1out() public payable {
    owner = msg.sender;
    allocations[owner] = msg.value;
  }

  modifier onlyOwner {
            require(
                msg.sender == owner
```

Source: https://ethernaut.openzeppelin.com/

Real World Example: Parity Wallet
https://blog.openzeppelin.com/on-the-parity-wallet-multisig-hack-405a8c12e8f7/

Smart Contract Vulns: Timestamp Dependence

- Occurs from a misunderstanding of time keeping in a smart contract
- `block.timestamp` and `block.number` can be used to trigger time-dependent events
- The block timestamp can be modified by miners to a degree
- Block time is typically 14 seconds but it's not constant
- Modification of a few seconds may result in exploitation

More Smart Contract Vulns

- New issues are still being found all the time
- Functions that appear secure on their own may yield critical issues when combined with others
- Smart Contract Weakness Classification Registry
  - https://swcregistry.io/
- Consensys - Smart Contracts Known Attacks
  - https://consensys.github.io/smart-contract-best-practices/known_attacks/

Case Studies

Uranium Finance Hack

- April 28, 2021
- $57 million stolen
- Uniswap clone on Binance Smart Chain

Case Study – Uranium Finance Hack

- Uranium Finance copied code from Uniswap and modified portions of it
- Portions of the **`swap`** function were changed to a larger value, but the balance check was unaltered
- Attacker deposited minimum funds to each of the token pair contracts
- Exploiting the misplaced zero, the attacker was able to drain each liquidity pool

```
        uint balance0Adjusted = balance0.mul(10000).sub(amount0In.mul(16));
        uint balance1Adjusted = balance1.mul(10000).sub(amount1In.mul(16));
        require(balance0Adjusted.mul(balance1Adjusted) >=
uint(_reserve0).mul(_reserve1).mul(1000**2), 'UraniumSwap: K');
```

Modified Uranium Code

VS

Original Uniswap Code

```
        uint balance0Adjusted = balance0.mul(1000).sub(amount0In.mul(3));
        uint balance1Adjusted = balance1.mul(1000).sub(amount1In.mul(3));
        require(balance0Adjusted.mul(balance1Adjusted) >=
uint(_reserve0).mul(_reserve1).mul(1000**2), 'UniswapV2: K');
```

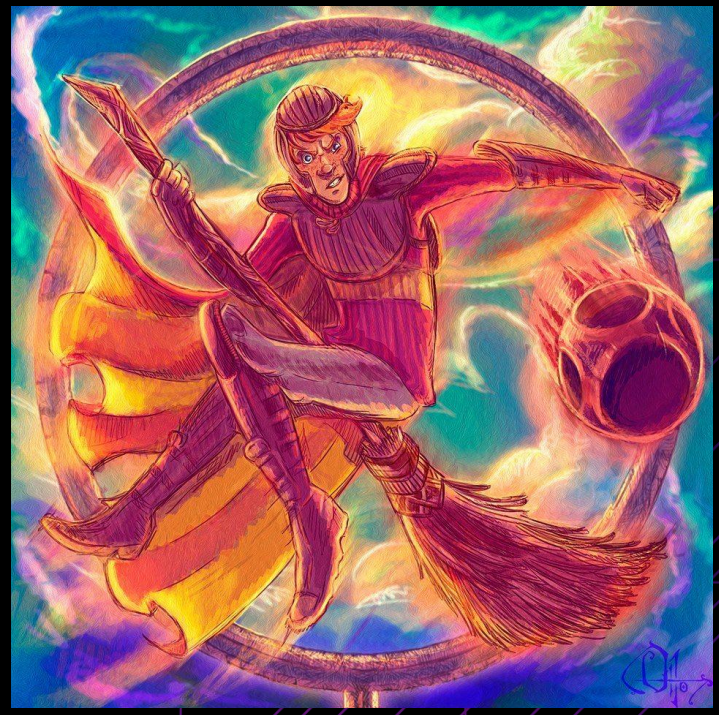
Source: https://www.rekt.news/uranium-rekt/

Poly Network Hack

- August 10, 2021
- $610 million stolen
- Enables cross-chain transactions
  - Ethereum
  - Binance Smart Chain
  - Polygon

Case Study – Poly Network Hack

- **`keepers`** are trusted entities for cross-chain transactions
- User sends a transaction on one blockchain and it gets repeated on a destination
- The **`keepers`** sign the block on the source blockchain
- User submits the signed block to **`EthCrossChainManager`** on the destination
- If sig is valid, the contract executes the transaction on the destination blockchain as the **`EthCrossChainManager`** contract, not the user



Case Study – Poly Network Hack

- **EthCrossChainManager** has the permission to change keepers via the **EthCrossChainData** contract
- Attacker exploited this to make themselves a **keeper**
- Attacker could now sign fake blocks with arbitrary transactions resulting in $610 million stolen

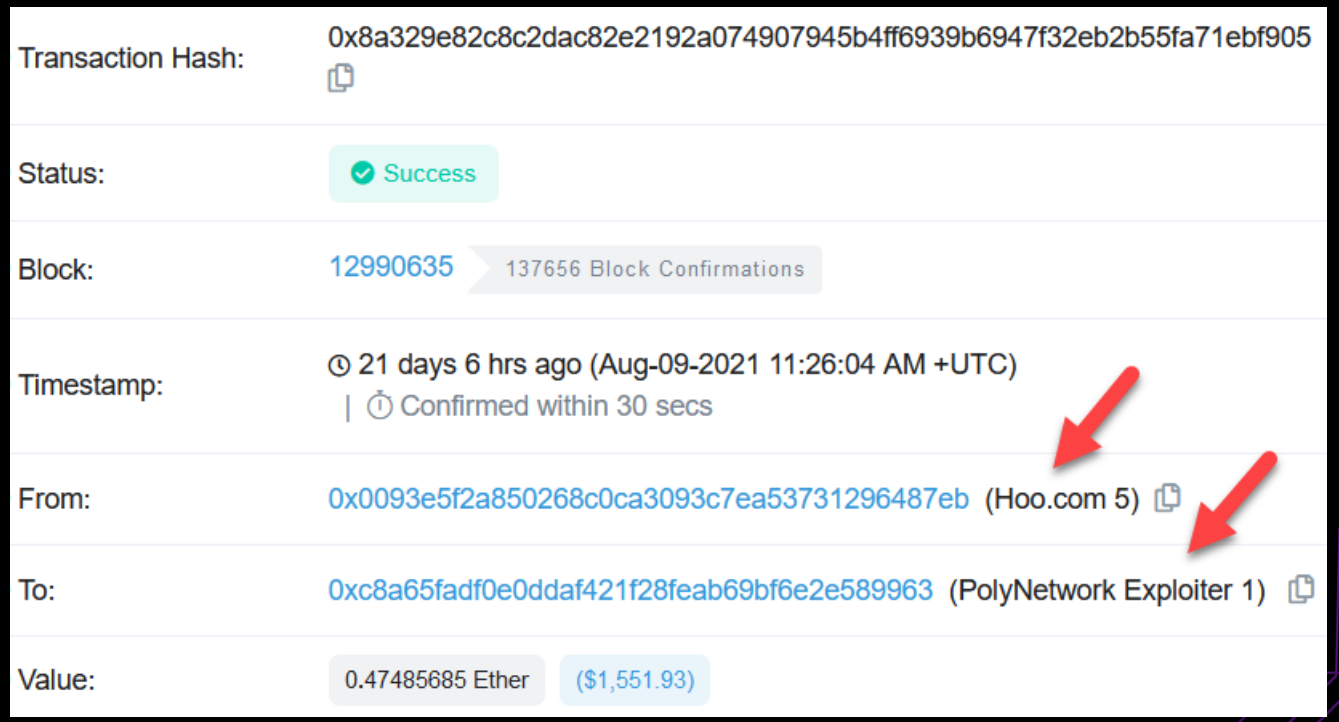
```
function _executeCrossChainTx(address _toContract, bytes memory _method, bytes memory _args,
        // Ensure the targeting contract gonna be invoked is indeed a contract rather than a
        require(Utils.isContract(_toContract), "The passed in address is not a contract!");
        bytes memory returnData;
        bool success;

        // The returnData will be bytes32, the last byte must be 01;
        (success, returnData) = _toContract.call(abi.encodePacked(bytes4(keccak256(abi.encode

        // Ensure the executation is successful
        require(success == true, "EthCrossChain call business contract failed");

        // Ensure the returned value is true
        require(returnData.length != 0, "No return value from business contract!");
        (bool res,) = ZeroCopySource.NextBool(returnData, 31);
        require(res == true, "EthCrossChain call business contract return is not true");

        return true;
    }
```

Txn Hash	Age	From		To	Value	Token
0xc917838cc3d1edd871…	20 days 7 hrs ago	0x250e76987d838a7531…	IN	PolyNetwork Exploiter 1	43,023.751365396442021965	Uniswap (UNI)
0xa7c56561bbe9fbd48e…	20 days 7 hrs ago	0x250e76987d838a7531…	IN	PolyNetwork Exploiter 1	673,227.941533113298891801	Dai Stableco... (DAI)
0x3f55ff1fa4eb3437afe4…	20 days 7 hrs ago	0x250e76987d838a7531…	IN	PolyNetwork Exploiter 1	1,032.12483694	Wrapped BTC (WBTC)
0x5a8b2152ec7d553803…	20 days 8 hrs ago	0x250e76987d838a7531…	IN	PolyNetwork Exploiter 1	96,389,444.229984	USD Coin (USDC)

Example transactions of hacked funds being sent to attacker wallet

Case Study – Poly Network Hack

- But they returned it… all of it
- Why? Bad OPSEC? Are they trying to appear to be a whitehat?



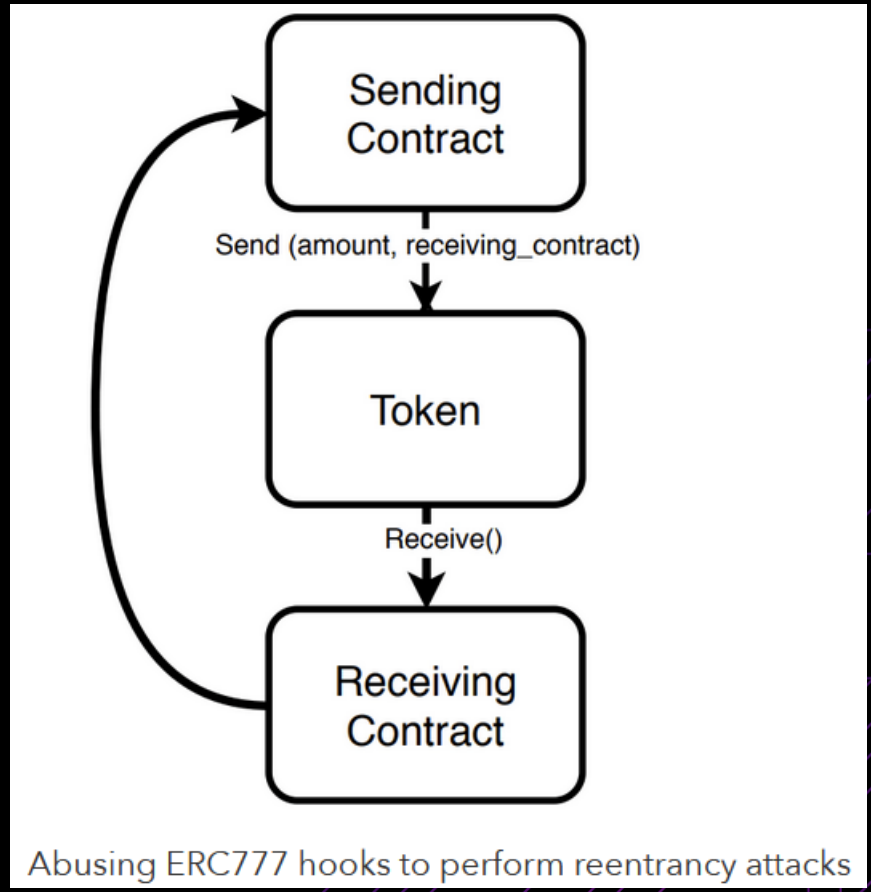
Attacker funded their account directly from an exchange (hoo.com)

Cream Finance Hack

- August 30, 2021
- $18.8 million stolen
- Decentralized Lending Protocol (flash loans)

Case Study – Cream Finance Hack

- Cream protocol implements the ERC777 AMP token contract
- ERC777 has "hooks" that call the sender and receiver contracts
- The `_callPreTransferHooks` function of the AMP token contract enabled Reentrancy on the Cream contract
- Allowed the attacker to nest a second `borrow()` function in the token `transfer()` function before the initial was updated

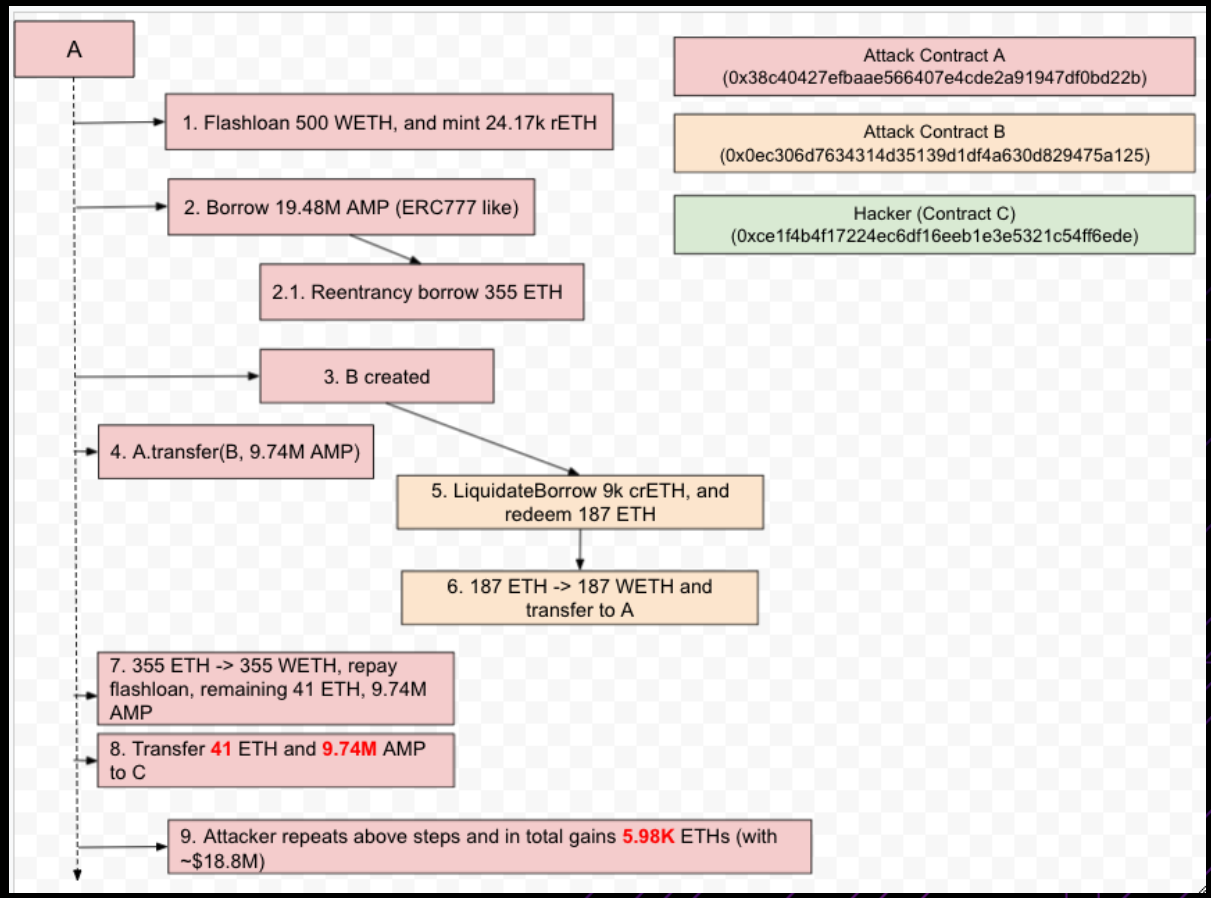


Abusing ERC777 hooks to perform reentrancy attacks

Source: https://www.zengo.com/imbtc-defi-hack-explained/

Case Study – Cream Finance Hack

- Attacker flash loan borrows 500 ETH, uses it as collateral to borrow 19 million AMP
- Exploits reentrancy to borrow 355 more ETH before initial `borrow()` is updated
- Creates new contract to liquidate part of the borrowed AMP, uses it to repay flash loan
- 41 ETH and 9.74 Million AMP leftover sent to attacker
- Rinse & Repeat X 17 times



Source: https://twitter.com/peckshield/status/1432250680799027204/photo/1

Live Exploit Demo

The Setup

- Ganache – Personal Ethereum blockchain hosted on VM
  - https://www.trufflesuite.com/ganache
- Truffle – Development framework for Ethereum
  - https://www.trufflesuite.com/truffle
- Remix IDE – Web-based IDE for writing and deploying smart contracts
  - https://remix.ethereum.org
- Metamask – Crypto wallet
  - https://metamask.io/
- Damn Vulnerable DeFi
  - https://www.damnvulnerabledefi.xyz/

Blockchain Hacking QuickStart Guide:
https://start.blockchainhax.com

Exploit Steps Recap

- Identify vulnerability in smart contract
- Deploy malicious contract that calls function in the target contract
- Call the exploit function in the malicious contract specifying the target function
- Exploit sets attacker contract as approved sender of ERC20 token
- Attacker contract initiates transfer of all tokens from pool to attacker wallet

Security Tools

VS Code + Solidity Visual Developer

https://github.com/ConsenSys/vscode-solidity-auditor

- Reading through the code thoroughly is a must
- Security centric syntax and semantic highlighting
- Find external calls, developer notes in comments, storage modifiers, access modifiers,
- Uses Sūrya for generating call graphs
- Hover over keywords to show basic security notes

VS Code + Solidity Visual Developer

TrusterLenderPool.sol

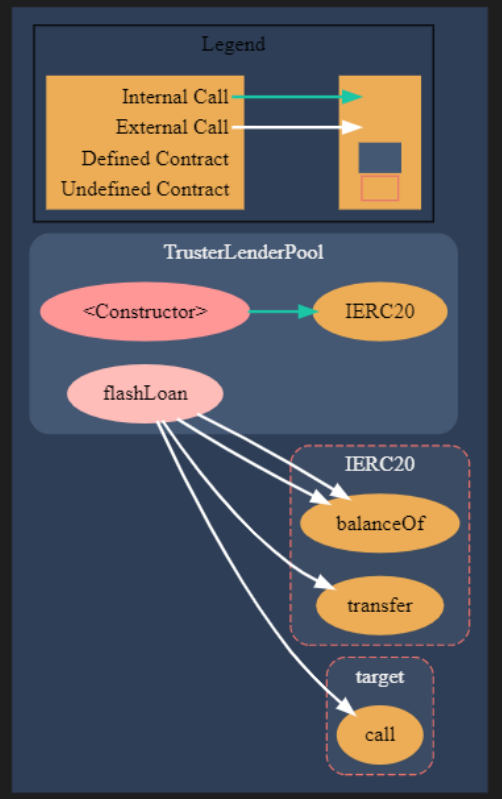
TrusterLenderPool.sol > {} pragma > {} solidity → ^0.6.0

report | graph (this) | graph | inheritance | parse | flatten | funcSigs | uml | draw.io

```
1  pragma solidity ^0.6.0;
2
3  import "@openzeppelin/contracts/token/ERC20/IERC20.sol";
4  import "@openzeppelin/contracts/utils/ReentrancyGuard.sol";
5
   UnitTest stub | dependencies | uml | draw.io
6  contract TrusterLenderPool is ReentrancyGuard {
7
8      IERC20 public damnValuableToken;
9
       ftrace
10     constructor (address tokenAddress) public {
11         damnValuableToken = IERC20(tokenAddress);
12     }
13
       ftrace | funcSig
14     function flashLoan(
15         uint256 borrowAmount,
16         address borrower,
17         address target,
18         bytes calldata data
19     )
20         external
21         nonReentrant
22     {
23         uint256 balanceBefore = damnValuableToken.balanceOf(address(this));
24         require(balanceBefore >= borrowAmount, "Not enough tokens in pool");
25
26         damnValuableToken.transfer(borrower, borrowAmount);
27         (bool success, ) = target.call(data);
28         require(success, "External call failed");
29
30         uint256 balanceAfter = damnValuableToken.balanceOf(address(this));
31         require(balanceAfter >= balanceBefore, "Flash loan hasn't been paid back");
32     }
33
34 }
35
```

Call Graph | Extension: Solidity Visual Developer

search...



Slither

https://github.com/crytic/slither

- Solidity source code analyzer
- Detects many common issues such as reentrancy, functions that allow users to self destruct the contract, uninitialized variables, and more
- Low false positives
- Can create graphic representation of function calls

Slither

```
$ slither trust-flat.sol

TrusterLenderPool.flashLoan(uint256,address,address,bytes) (trust-flat
.sol#160-178) ignores return value by damnValuableToken.transfer(borro
wer,borrowAmount) (trust-flat.sol#172)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentati
on#unchecked-transfer

TrusterLenderPool.flashLoan(uint256,address,address,bytes).target (tru
st-flat.sol#163) lacks a zero-check on :
                - (success) = target.call(data) (trust-flat.sol#173)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentati
on#missing-zero-address-validation

Different versions of Solidity is used:
        - Version used: ['>=0.6.0<0.8.0', '^0.6.0']
        - >=0.6.0<0.8.0 (trust-flat.sol#5)
        - >=0.6.0<0.8.0 (trust-flat.sol#85)
        - ^0.6.0 (trust-flat.sol#148)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentati
on#different-pragma-directives-are-used

Pragma version>=0.6.0<0.8.0 (trust-flat.sol#5) is too complex
Pragma version>=0.6.0<0.8.0 (trust-flat.sol#85) is too complex
Pragma version^0.6.0 (trust-flat.sol#148) allows old versions
solc-0.6.0 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentati
on#incorrect-versions-of-solidity

Low level call in TrusterLenderPool.flashLoan(uint256,address,address,
bytes) (trust-flat.sol#160-178):
        - (success) = target.call(data) (trust-flat.sol#173)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentati
on#low-level-calls
trust-flat.sol analyzed (3 contracts with 75 detectors), 8 result(s) f
ound
```

Mythril & MythX

https://github.com/ConsenSys/mythril
https://mythx.io/

- Mythril
  - Symbolic execution vulnerability scanner
  - Can scan bytecode directly
  - Free and open source
- MythX
  - Static analysis, symbolic analysis & fuzzing
  - Has an API you can submit scan jobs to
  - Integrates into dev frameworks
  - Not free

Mythril

```
$ myth analyze trust-flat.sol
==== External Call To User-Supplied Address ====
SWC ID: 107
Severity: Low
Contract: TrusterLenderPool
Function name: flashLoan(uint256,address,address,bytes)
PC address: 1007
Estimated Gas Usage: 19854 - 201493
A call to a user-supplied address is executed.
An external message call to an address specified by the caller is exec
uted. Note that the callee account might contain arbitrary code and co
uld re-enter any function within this contract. Reentering the contrac
t in an intermediate state may lead to unexpected behaviour. Make sure
 that no state modifications are executed after this call and/or reent
rancy guards are in place.
--------------------
In file: trust-flat.sol:172

damnValuableToken.transfer(borrower, borrowAmount)

--------------------
Initial State:

Account: [CREATOR], balance: 0x10000406, nonce:0, storage:{}
Account: [ATTACKER], balance: 0x1, nonce:0, storage:{}
Account: [SOMEGUY], balance: 0x0, nonce:0, storage:{}
```

@BHInfoSecurity

Resources to Get Started

@BHInfoSecurity

Resources to Get Started

- Books
  - Mastering Ethereum - https://github.com/ethereumbook/ethereumbook
  - Hands-On Smart Contract Development
- Learn Solidity
  - https://cryptozombies.io/ - Free dApp building game
  - https://solidity-by-example.org
- CTF
  - Ethernaut
    - https://ethernaut.openzeppelin.com/
  - Damn Vulnerable DeFi
    - https://www.damnvulnerabledefi.xyz/

Bug Bounties

- Immunefi
  - https://immunefi.com/
- Consensys Bug Bounty List
  - https://consensys.github.io/smart-contract-best-practices/bug_bounty_list/
- Code 423n4
  - https://code423n4.com/
- Hacken
  - https://hackenproof.com/programs
- Chainlink
  - https://hackerone.com/chainlink?type=team
- The Graph
  - https://thegraph.com/security/

Key Takeaways

1. Use cases for blockchain are growing significantly
2. Hacks typically result in a significant loss of funds
3. As this is an emerging technology, new smart contract vulnerabilities are still being discovered
4. Security analyzers are good, but understanding the code is a must
5. Smart contracts are only one piece of the growing blockchain ecosystem

The End

- Follow me on Twitter
  - Beau Bullock - @dafthack
- BlockchainHAX QuickStart Guide
  - https://start.blockchainhax.com
- CoinSec Podcast – Weekly show about blockchain security
  - coinsecpodcast.com
  - @coinsecpodcast
- Black Hills Information Security
  - https://www.blackhillsinfosec.com
  - @BHInfoSecurity