Web3 Security

Adventure to Safer Web3 World

Brian Pak / Juno Im

Web3@KAIST

Introduction

Brian Pak
박세준

70+ wins in international hacking competitions
Including 6-time wins on DEFCON CTF
Winners of Paradigm CTF & Numen CTF (Web3)

Multiple vulnerabilities reported
Various global vendors and open-source projects
Ethereum vulnerability bounty leaderboard

Juno Im
임준오

Theori
Empowering Innovation with Security

ChainLight

Agenda

1. Cyber Security
2. Blockchain x Security
3. Security Threats in Web3
4. Solidity Security
5. Real World Examples
6. Future-proof Security

Cybersecurity

Security in Cyberspace

Theori

Cyberspace

Virtual environment with computer systems

Cyberspace

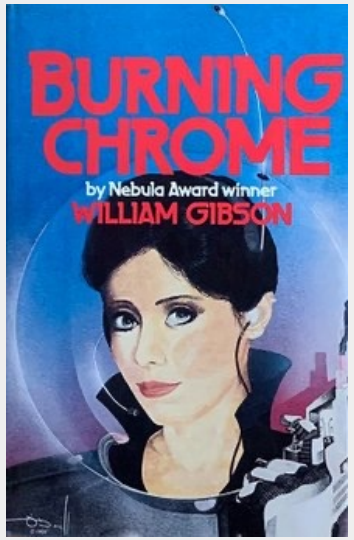
Cyberspace

COVID-19 accelerated DX

Evolution of Cyberspace

1982

The Origin



First appeared in cyberpunk fiction, authored by William Gibson

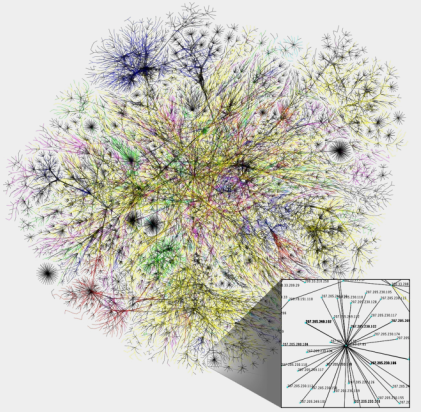
Gibson described it as an online computer network

Early 90s

The Internet

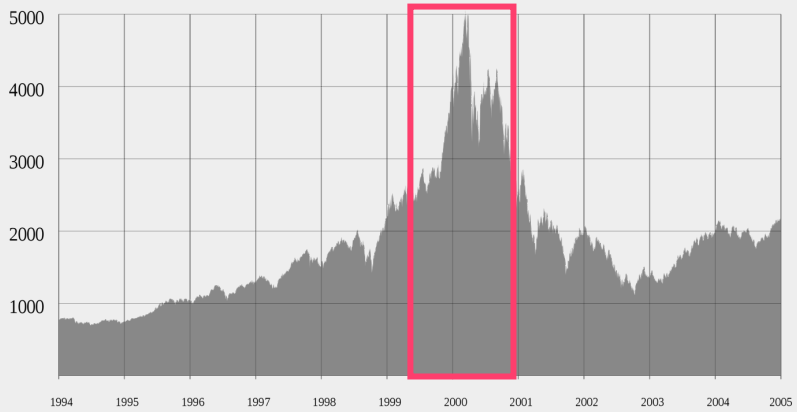
Initially developed in 1960s by the US DoD for military purpose

Later expanded into the commercial networks and enterprises market



Dot-com Era

Late 90s; Early 2000



Massive growth in Internet adoption with lots of money (VCs) and start-ups

E-commerce, communications, finance, ads

“Bubble” pops..

Current

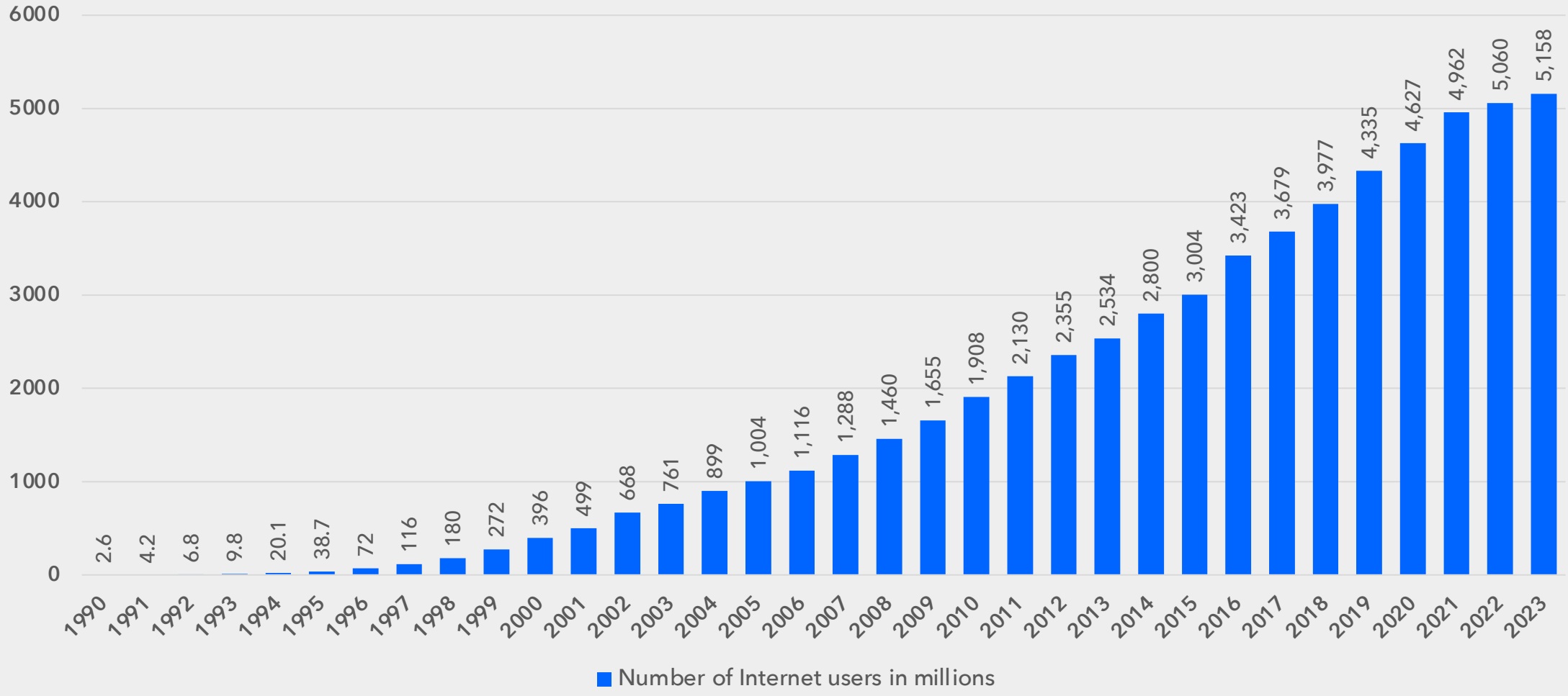
2023

Cloud infrastructure

Microservice Architecture

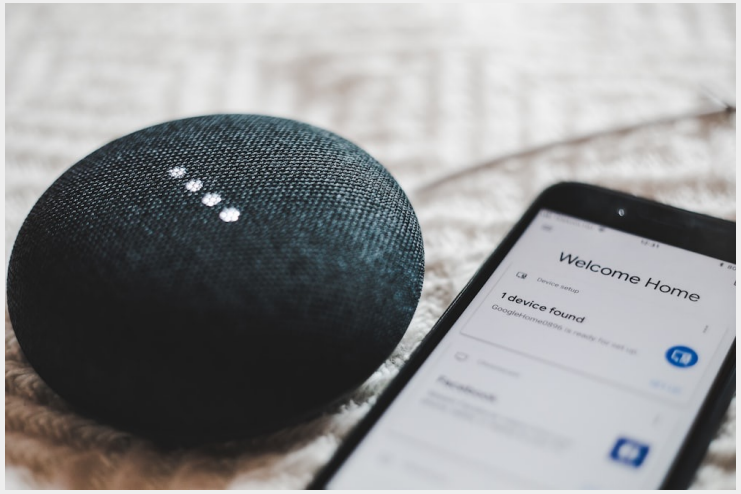
Blockchain (Web3) popularized

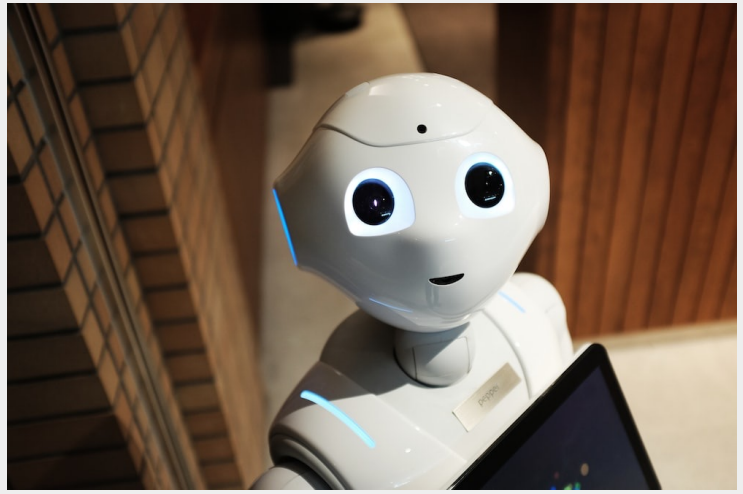
Evolution of Cyberspace



https://datareportal.com/global-digital-overview

Current State of Cyberspace





Current State of Cyberspace

Threats in Industry

IT / Tech

- Web & Mobile applications
- Cloud infrastructure
- CI/CD pipeline (DevOps)

Finance

- Web & Mobile applications
- Financial information
- Security "solutions"

Game

- Cheats / Anti-cheat
- IP Theft
- Web applications

Automotive

- Embedded hardware
- Firmware
- Physical security

Web3

- Centralized Exchanges (CEX)
- Decentralized Finance (DeFi)
- Non-Fungible Tokens (NFTs)
- Blockchain / Smart contracts

Blockchain x Security

Building Trust and Integrity in Blockchain

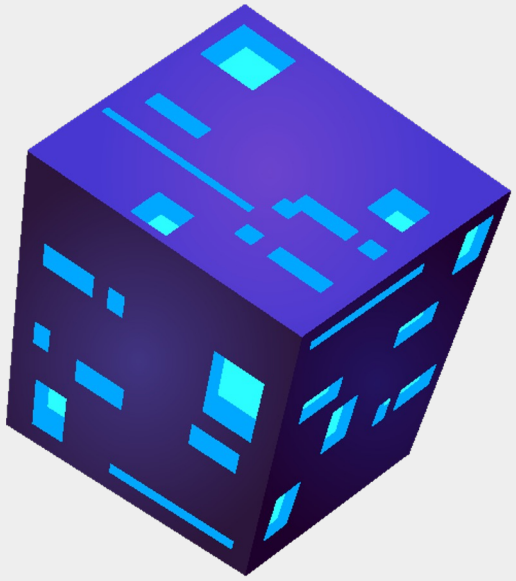
Theori

Blockchain x Security

Blockchain provides some **strong** guarantees

Immutable

Distributed / Decentralized



Transparent

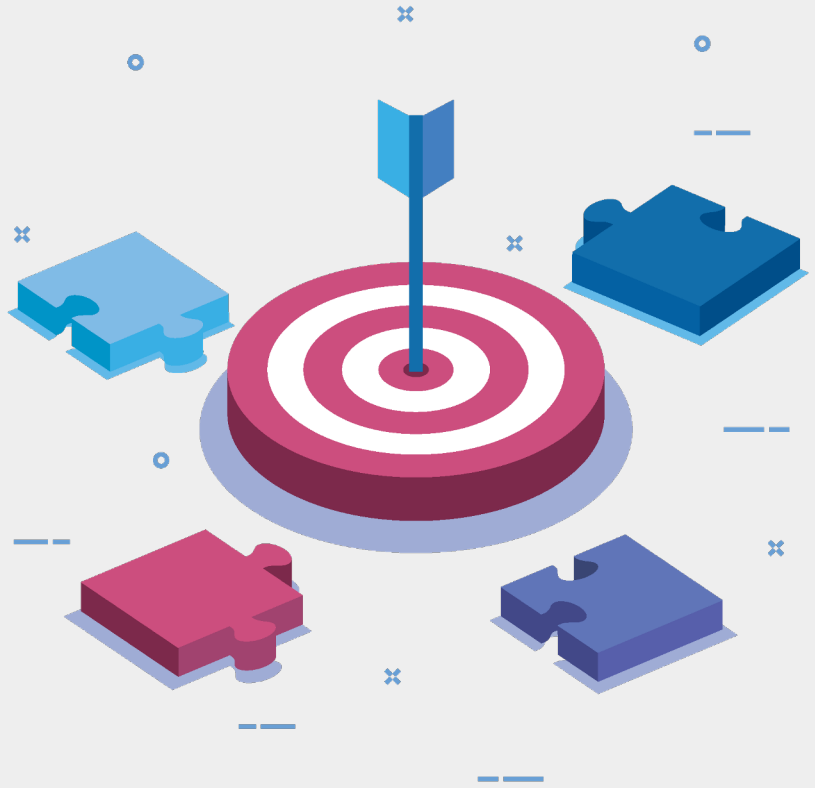
Secure

Blockchain x Security

New **paradigm**, new **frameworks** appeared

Blockchain x Security

New **attack surfaces** and **threat models** arise

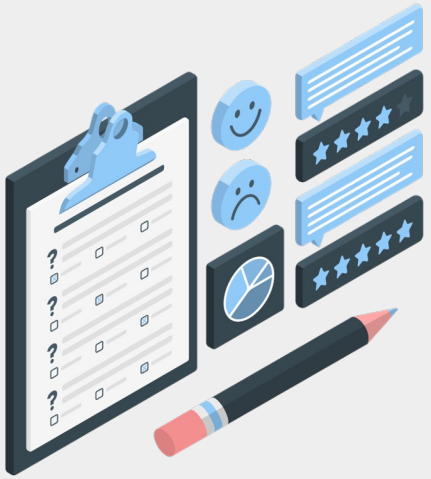


Blockchain x Security

One **tiny mistake** can cost a **fortune**

But, there are ways to make things more secure

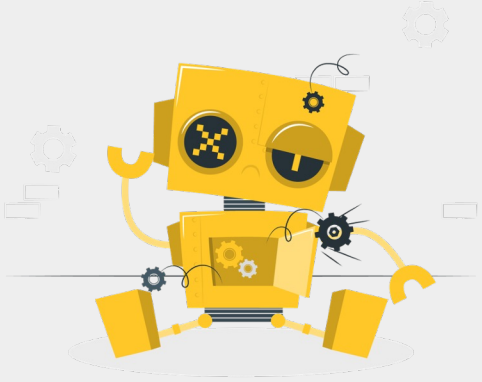
Bug bounties



Security audits

Project teams may **not** be well-funded

(Even though they may have large TVL)



Security Threats in Web3

Potential Threats and Challenges

Theori

Security Threats in Web3

Smart Contracts

Blockchain Network

Infrastructure & Off-chain

Centralization Risks

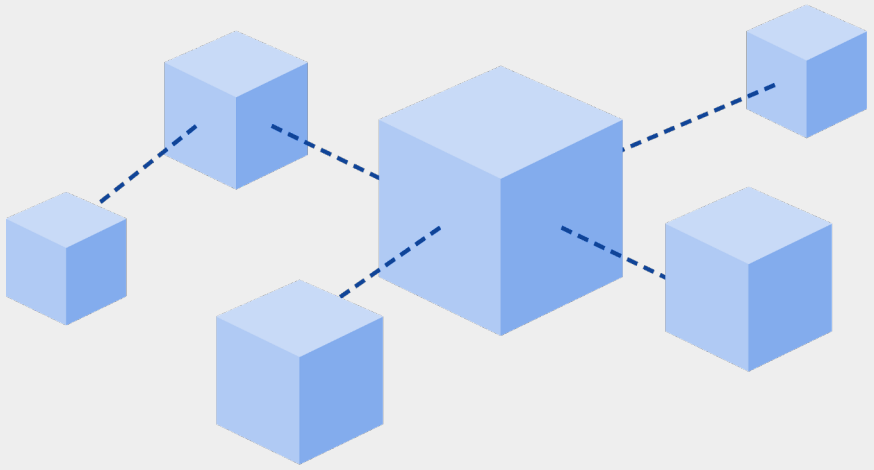
Security Threats in Web3

Smart Contracts

The Brain

- Reentrancy
- Insufficient ACL
- Integer overflow / underflow
- Financial engineering attacks
- Insecure governance model
- Logic bugs

Security Threats in Web3



Blockchain Network

Nodes

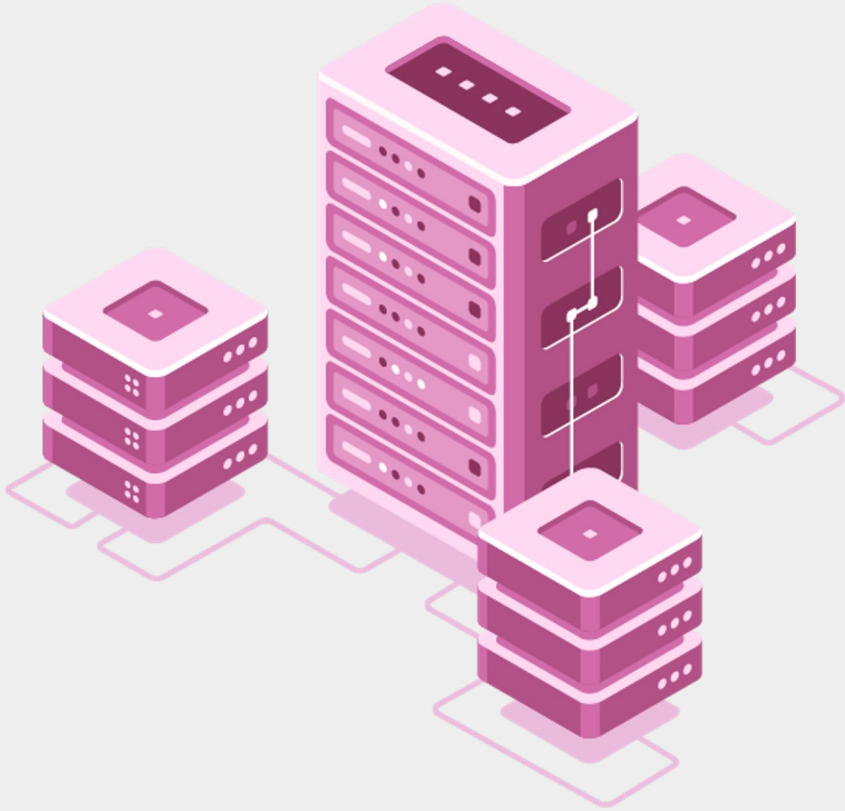
- Denial of Service
- Precompiled Smart Contracts bugs
- Remote Code Execution
- P2P Eclipse attack
- Consensus issues (Chain splits)
- Maximum (Miner) Extractable Value

Security Threats in Web3

Infrastructure & Off-chain

Web2 / Legacy

- Front-end web vulnerabilities
- Key management
- Lack of user input validation
- Events and log parsing
- Phishing
- State-sponsored cyber attacks



Security Threats in Web3

Centralization Risks Dependency

- ❖ Backdoors
- ❖ Rug pull
- ❖ Scams
- ❖ Majority attacks
- ❖ Upgradeability

Smart Contracts Security

Smart ≠ Secure

Theori

"Smart" Contracts

Smart Contracts

A digital contract executed via computerized transactions

Concept proposed by Nick Szabo in 1994

Plays a "brain" role and enables application development

Most blockchains support smart contracts (e.g. Ethereum, Aptos, Solana)

"Smart" Contracts

Smart Contracts

A digital contract executed via computerized transactions

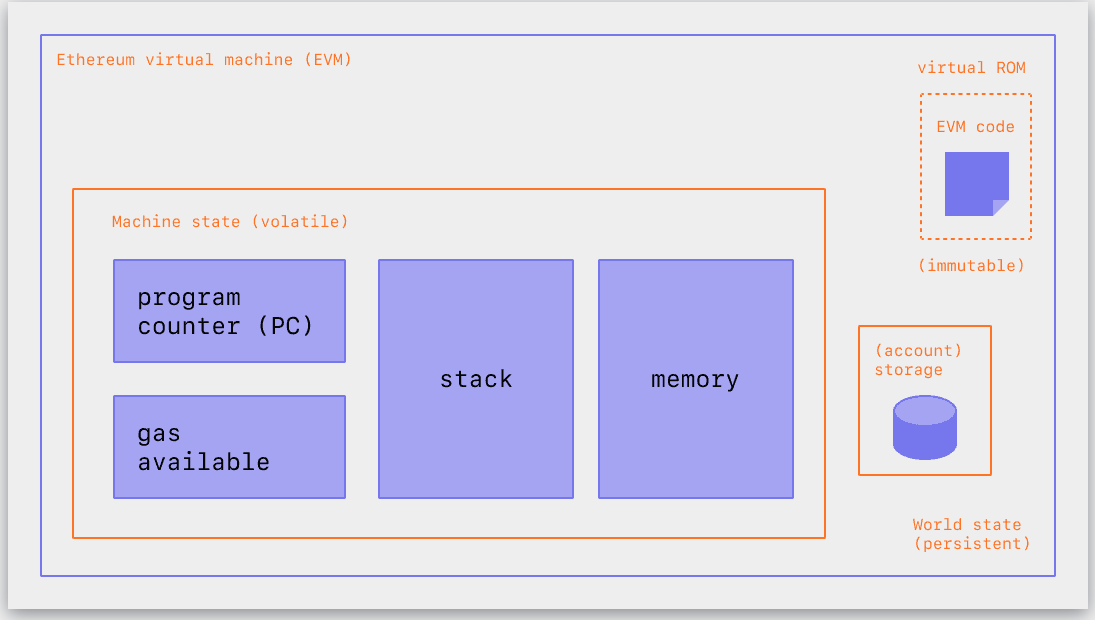
Observability

Verifiability

Privacy

Enforceability

Ethereum Virtual Machine (EVM)



The World's Computer

- Programmable, decentralized state-machine
- Turing-complete smart contracts can be executed
  - Decentralized computing platform!
- EVM Architecture
  - Stack-based VM
  - Gas as "fee"

Why Gas?

Ethereum blockchain uses fees as fuel for executing smart contracts
Gas usage is limited, and prices are adjusted according to market economics to ensure network stability

Solidity

Decentralized app (DApp) development language in EVM-based blockchain

- Similar syntax as JavaScript, Java, Go
- Basic programming structure
  - Arithmetic operations, types, constants and variables, control statements, function calls, memory, basic data structures, error handling, etc.
  - Reserved keywords and global variables to access blockchain info

```
// SPDX-License-Identifier: UNLICENSED
pragma solidity 0.8.16;

contract TheoriRules {
    function add(uint a, uint b) external pure returns(uint) {
        return a+b;
    }
}
```

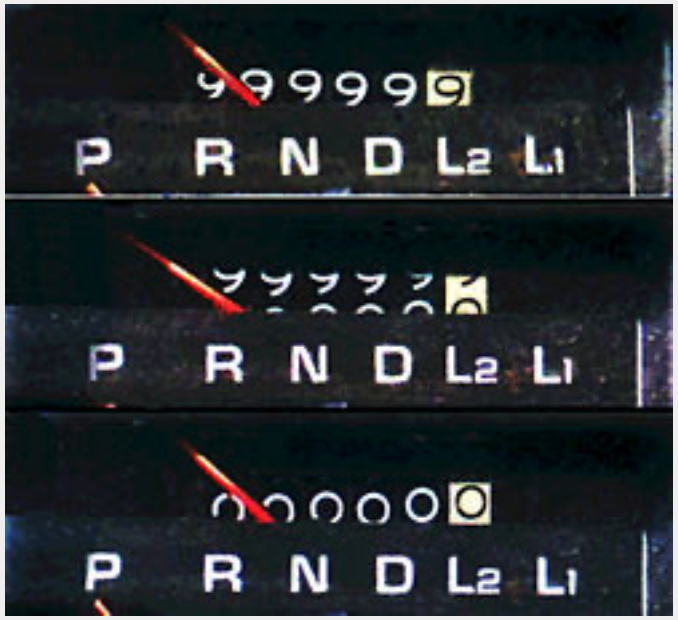
Compiled to Bytecode

```
0x608060405234801561001057600080fd5b506101b480610020
6000396000f3fe608060405234801561001057600080fd5b5060
04361061002b5760003560e01c8063771602f714610030575b60
0080fd5b61004a600480360381019061004591906100b1565b61
0060565b604051610057919061010056 5b60405180910390f35b
6000818361006e919061014a565b905092915050565b600080fd
5b60008190509190505 65b61008e8161007b565b811461009957
600080fd5b50565b600081359050610 0ab8161008565b929150
50565b60008060408385031215 6100c8576100c7610076565b5b
60006100d68582860161009c565b92505060206100e785828601
61009c565b9150509250929050565b6100fa8161007b565b8252
5050565b60006020820190506101156000830184 6100f1565b92
915050565b7f4e487b710000000000000...
```

Smart Contracts Security

- We will be focusing on Solidity code
  - Most of the smart contracts are deployed on EVM compatible chain and written in Solidity
- Smart Contract Weaknesses
  - SWC-101: Integer overflow / underflow
  - SWC-107: Reentrancy
  - SWC-136: Unencrypted Private Data On-Chain
  - SWC-128: DoS With Block Gas Limit
  - SWC-122: Lack of Proper Signature Verification
  - SWC-113: DoS with Failed Call

Integer overflow / underflow



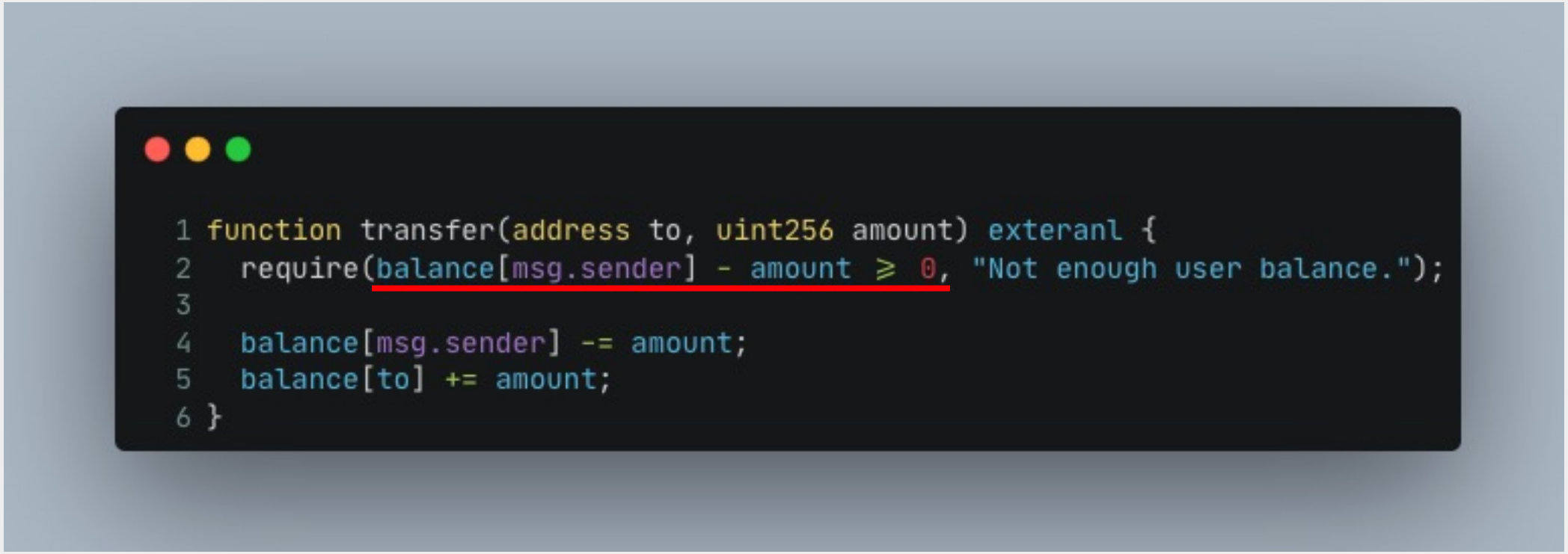
Integer overflow / underflow

Type	Storage size	Value range
char	1 byte	-128 to 127 or 0 to 255
unsigned char	1 byte	0 to 255
signed char	1 byte	-128 to 127
int	2 or 4 bytes	-32,768 to 32,767 or -2,147,483,648 to 2,147,483,647
unsigned int	2 or 4 bytes	0 to 65,535 or 0 to 4,294,967,295
short	2 bytes	-32,768 to 32,767
unsigned short	2 bytes	0 to 65,535
long	4 bytes	-2,147,483,648 to 2,147,483,647
unsigned long	4 bytes	0 to 4,294,967,295

Integer overflow / underflow - Example

```
function transfer(address to, uint256 amount) exteranl {
  require(balance[msg.sender] - amount >= 0, "Not enough user balance.");

  balance[msg.sender] -= amount;
  balance[to] += amount;
}
```

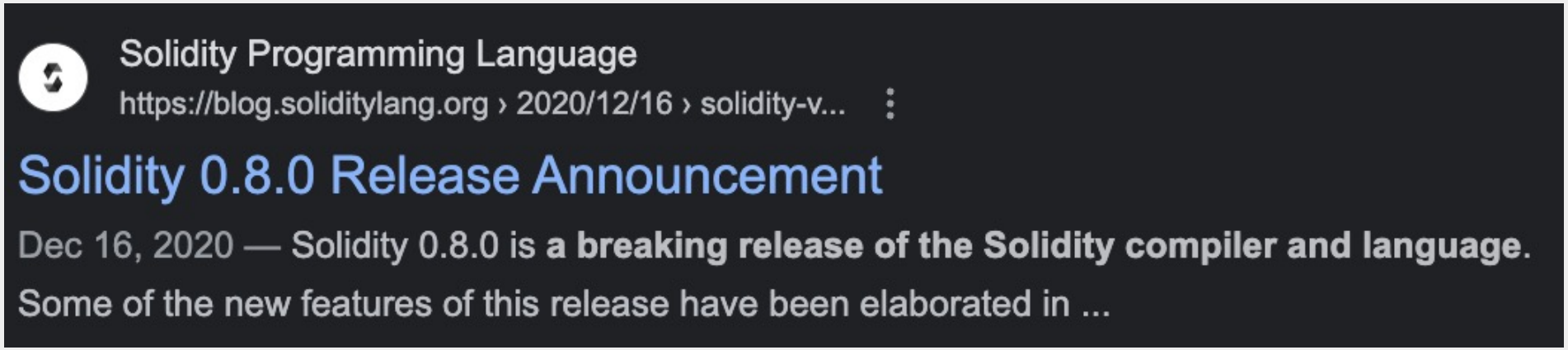
Integer overflow / underflow - Example



```
1 function transfer(address to, uint256 amount) exteranl {
2   require(balance[msg.sender] - amount >= 0, "Not enough user balance.");
3
4   balance[msg.sender] -= amount;
5   balance[to] += amount;
6 }
```

Integer overflow / underflow - Remediation

- From solidity 0.8.0, compiler add safeguards on the entire of arithmetic calculations **ON Dec 16, 2020.**



Reentrancy

- Any interaction from a contract (A) with another contract (B) and any transfer of Ether hands over control to that contract (B).

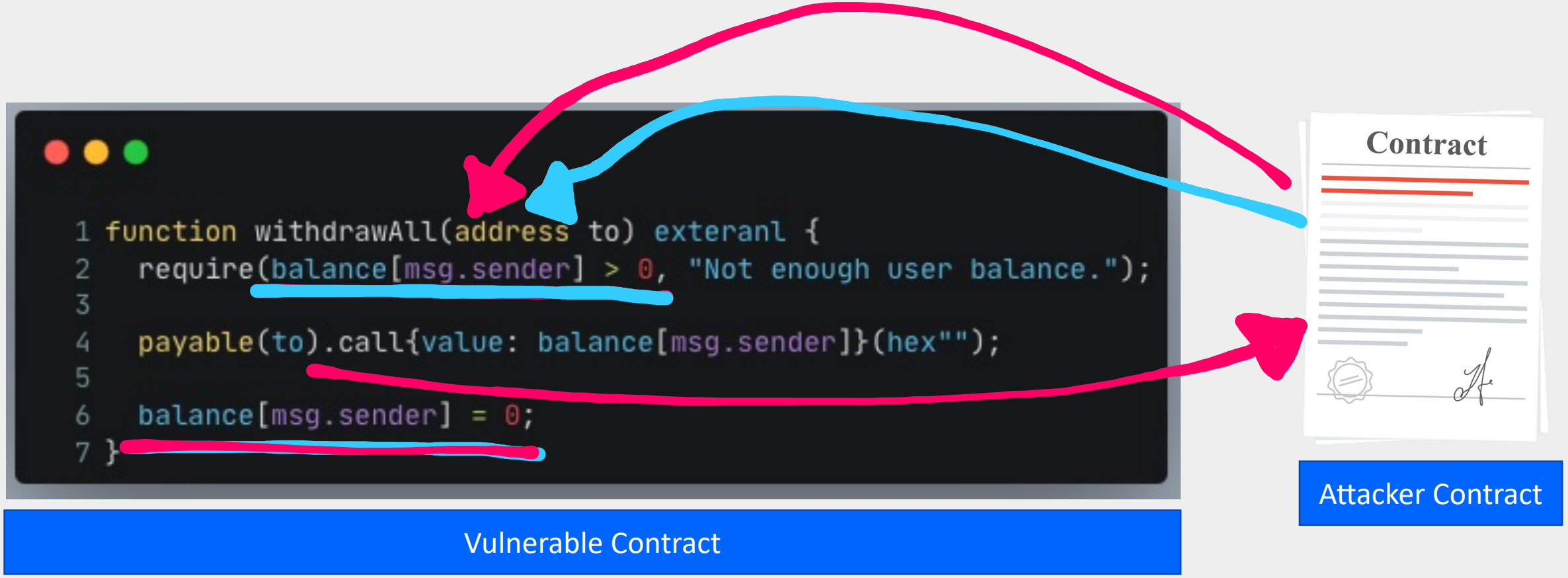
- This makes it possible for B to call back into A before this interaction is completed.

- To give an example, the following code contains a bug (it is just a snippet and not a complete contract):

Reentrancy - Example

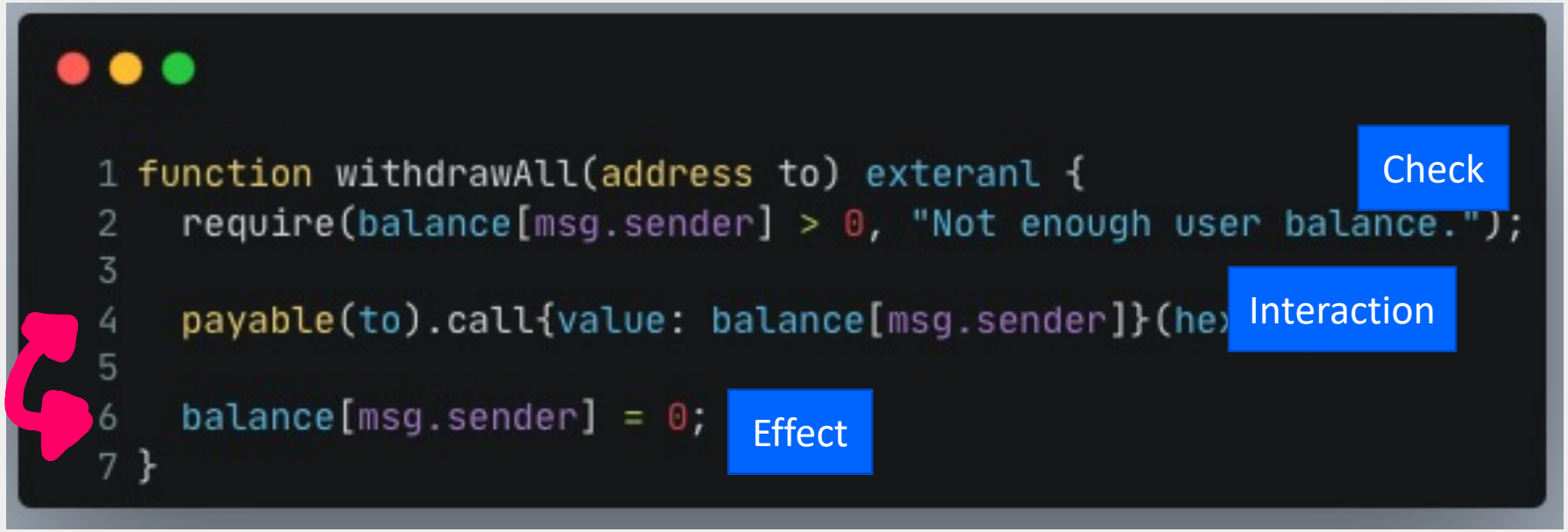
```
function withdrawAll(address to) exteranl {
  require(balance[msg.sender] > 0, "Not enough user balance.");

  payable(to).call{value: balance[msg.sender]}(hex"");

  balance[msg.sender] = 0;
}
```

Reentrancy - Example



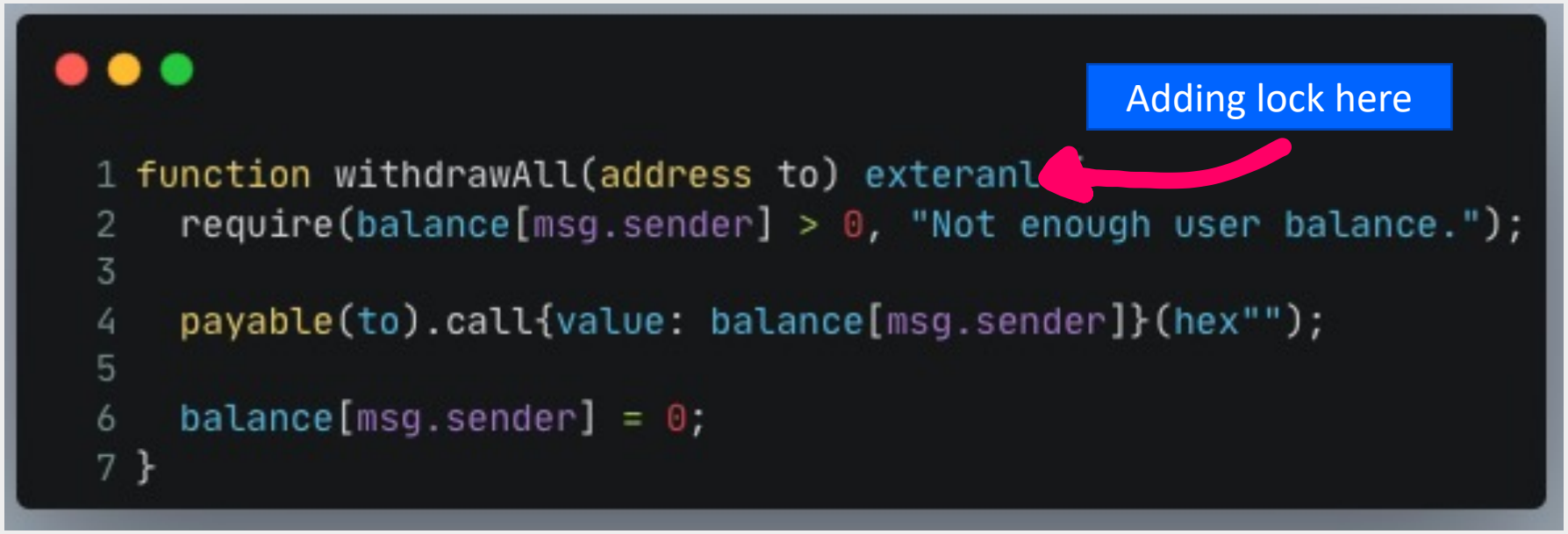
Reentrancy - Remediation

- Checks-Effects-Interactions pattern
  - https://docs.soliditylang.org/en/v0.6.11/security-considerations.html



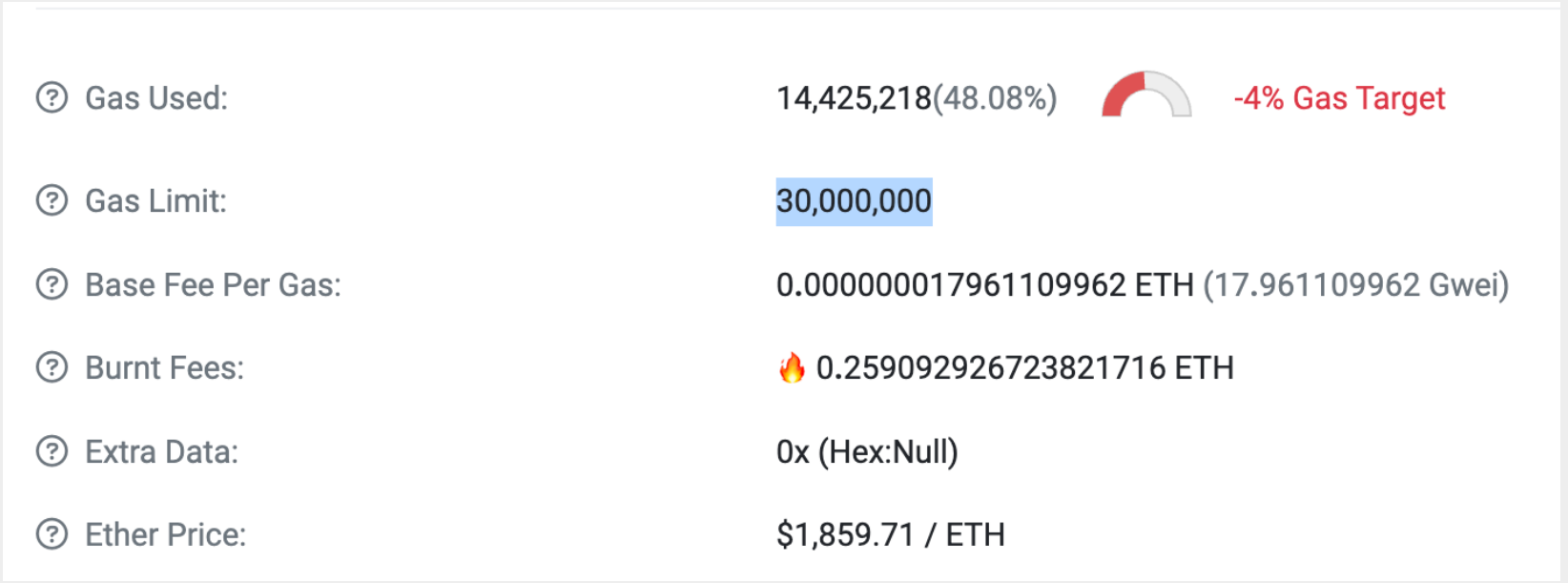
Reentrancy - Remediation

- Non-reentrant modifier (mutex)
  - Enforce limits on call to the same function among the same call stack.



DoS With Block Gas Limit

- User pays “Gas” as a transaction fee.
- Block has a limitation of maximum gas, Gas Limit.



DoS With Block Gas Limit

```
function updateUserInfoByName(string calldata userName, uint256 age) external {
    for (uint256 i=0; i<users.length; i++) {
      if (keccak256(users[i].name) == keccak256(abi.encodePacked(userName))) {
        users[i].age = age;
        break;
      }
    }
}
```

DoS With Block Gas Limit - Remediation

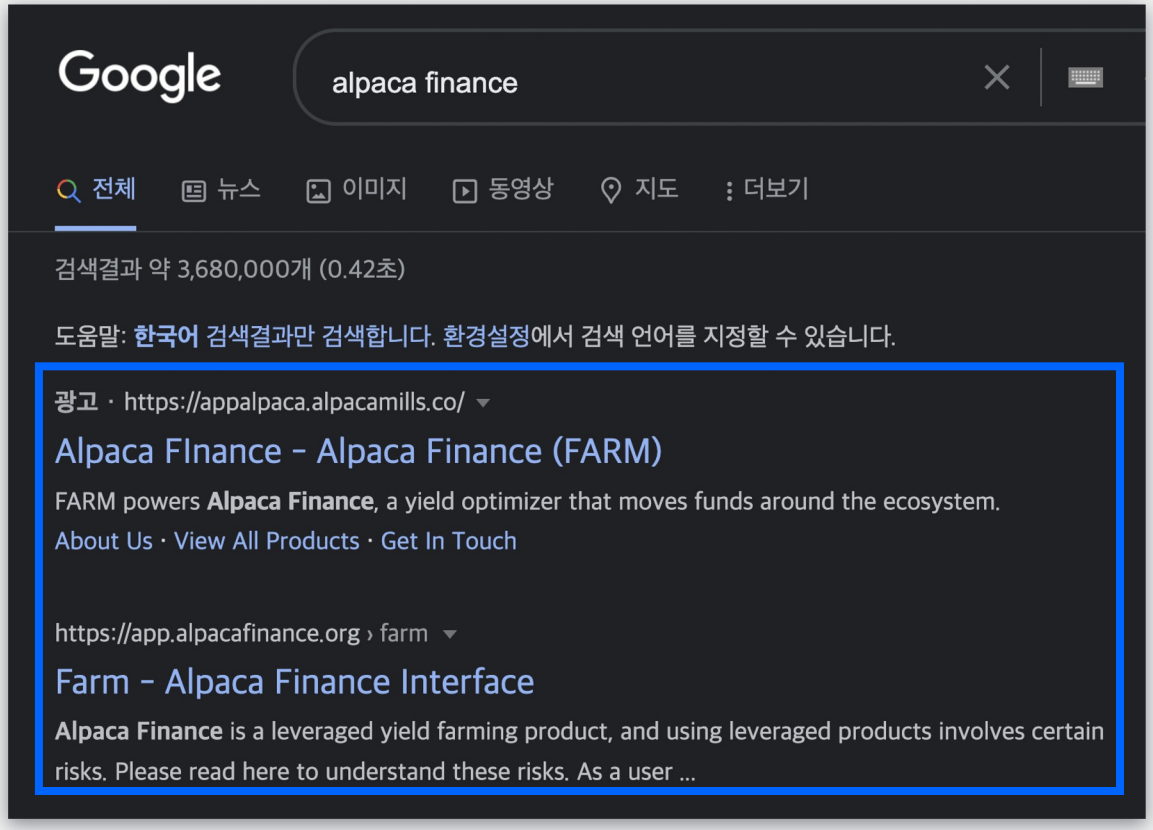
DoS With Block Gas Limit - Remediation

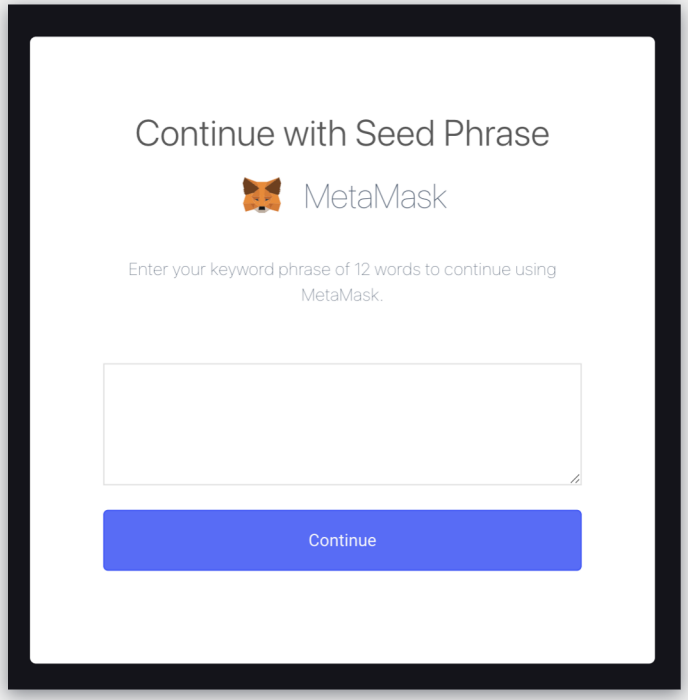
```
1 mapping(bytes32⇒uint256) userNameToIndex;
2 function updateUserInfoByName(string calldata userName, uint256 age) external {
3     users[userNameToIndex(keccak256(userName))].age = age;
4 }
```

Real-world Incidents

Real World Case I: Phishing

Low **technical difficulty**, but **highly effective** attack **#Pay2Hack**





Real World Case I: Phishing

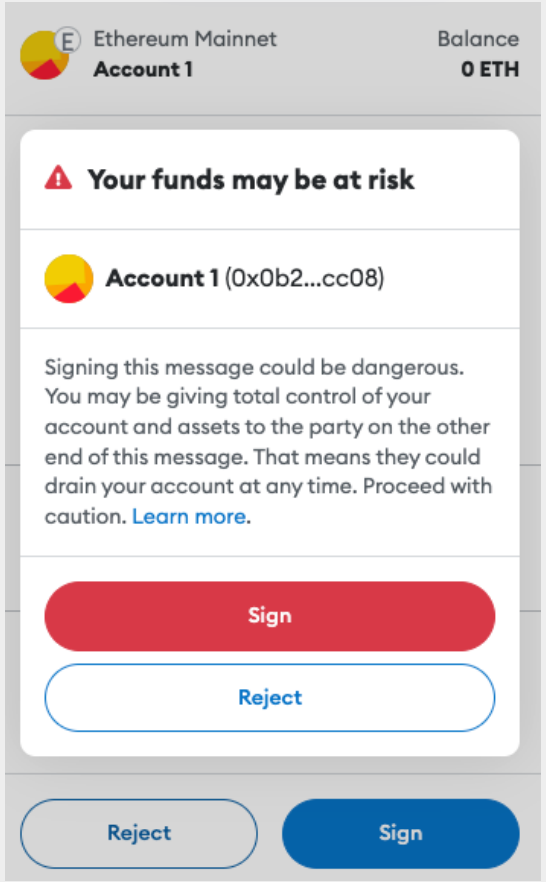
Low **technical difficulty**, but **highly effective** attack #Pay2Hack

People studied no one (even Metamask) asks for seed phrases

They started to ask you “sign” something. (= tx hash)

For more details:

https://blog.chainlight.io/si-vis-pacem-para-bellum-exploring-metamask-phishing-4605425d80a7



Real World Case II: Harvest Finance

- 💸 $33.8M of losses ($24M to attacker)
- 💥 Classic example of price oracle attack with a flash loan
- 🙀 The attacker successfully gained profit with 10 ETH
  - ❖ Swap to increase price of USDC token
    (USDT ⇒ USDC)
  - ❖ Deposit USDC into Vault
  - ❖ Swap to decrease price of USDC token
    (USDC ⇒ USDT)
  - ❖ Withdraw USDC from Vault
    (price is lower, so we get more USDC)
  - ❖ Repeat

0xc6028a9fa486f52efd2...	→	Harvest.Finance: Hacker 1	20 Ether
0xc6028a9fa486f52efd2...	→	Harvest.Finance: Hacker 1	20 Ether
0xc6028a9fa486f52efd2...	→	Harvest.Finance: Hacker 1	20 Ether
0xc6028a9fa486f52efd2...	→	Harvest.Finance: Hacker 1	20 Ether
0xc6028a9fa486f52efd2...	→	Harvest.Finance: Hacker 1	20 Ether
0xc6028a9fa486f52efd2...	→	Harvest.Finance: Hacker 1	20 Ether
0xc6028a9fa486f52efd2...	→	Harvest.Finance: Hacker 1	20 Ether
0xc6028a9fa486f52efd2...	→	Harvest.Finance: Hacker 1	20 Ether
0xc6028a9fa486f52efd2...	→	Harvest.Finance: Hacker 1	20 Ether
Tornado.Cash: 10 ETH	→	Harvest.Finance: Hacker 1	9.984 Ether

Flash borrow 18M USDT and 50M USDC

Swap 17M USDT to USDC

Deposit 50M USDC & Receive 52M pool tokens

Swap 17M USDC to USDT

Withdraw 50.8M USDC with 52M pool tokens

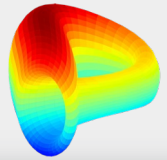
Repay flash loans
⇒ $500K profit

Increases price of USDC

Query USDC price

Decreases price of USDC

Query USDC price



Curve
yUSD pool

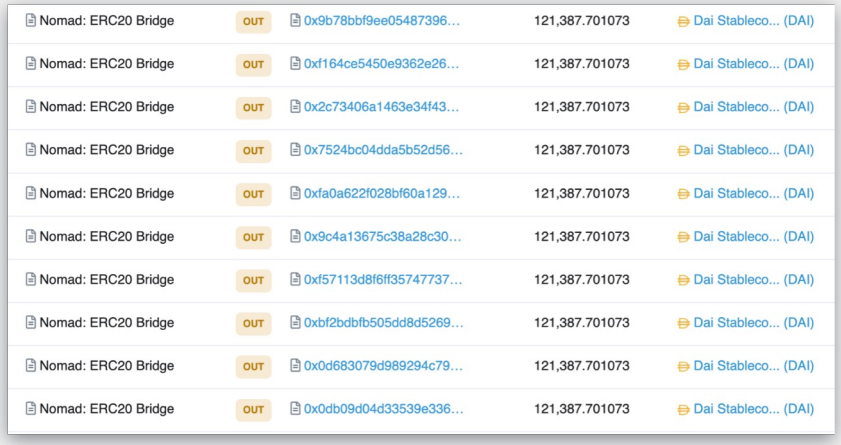
Real World Case III: Nomad Bridge

NOMAD

💸 $190M of losses

💥 “Every-man-for-himself” as everyone copied the attack

(First crowd hacking..?)



Nomad: ERC20 Bridge	OUT	0x9b78bbf9ee05487396…	121,387.701073	Dai Stableco… (DAI)
Nomad: ERC20 Bridge	OUT	0xf164ce5450e9362e26…	121,387.701073	Dai Stableco… (DAI)
Nomad: ERC20 Bridge	OUT	0x2c73406a1463e34f43…	121,387.701073	Dai Stableco… (DAI)
Nomad: ERC20 Bridge	OUT	0x7524bc04dda5b52d56…	121,387.701073	Dai Stableco… (DAI)
Nomad: ERC20 Bridge	OUT	0xfa0a622f028bf60a129…	121,387.701073	Dai Stableco… (DAI)
Nomad: ERC20 Bridge	OUT	0x9c4a13675c38a28c30…	121,387.701073	Dai Stableco… (DAI)
Nomad: ERC20 Bridge	OUT	0xf57113d8f6ff35747737…	121,387.701073	Dai Stableco… (DAI)
Nomad: ERC20 Bridge	OUT	0xbf2bdbfb505dd8d5269…	121,387.701073	Dai Stableco… (DAI)
Nomad: ERC20 Bridge	OUT	0x0d683079d989294c79…	121,387.701073	Dai Stableco… (DAI)
Nomad: ERC20 Bridge	OUT	0x0db09d04d33539e336…	121,387.701073	Dai Stableco… (DAI)

🙀 Code upgrade added a bug

- ❖ Special cases were added for “legacy” messages
- ❖ Failed to handle special case of None (0x0)

🤯 By itself, not exploitable, except…

- ❖ During initialization, 0x0 was accidentally set as a trusted Merkle root
- ❖ On Ethereum, uninitialized storage defaults to 0x0
- ❖ All messages with an uninitialized root are now valid!

Real World Case IV: Ronin Network

💸 $624M of losses

💥 State-sponsored attack (North Korea); Broke “multisig”

👰 Bridge contract used a 5 of 9 signature check

- ❖ 5 validators must sign a message
- ❖ 9 total validators

🙀 4 validators were run by **ONE** company

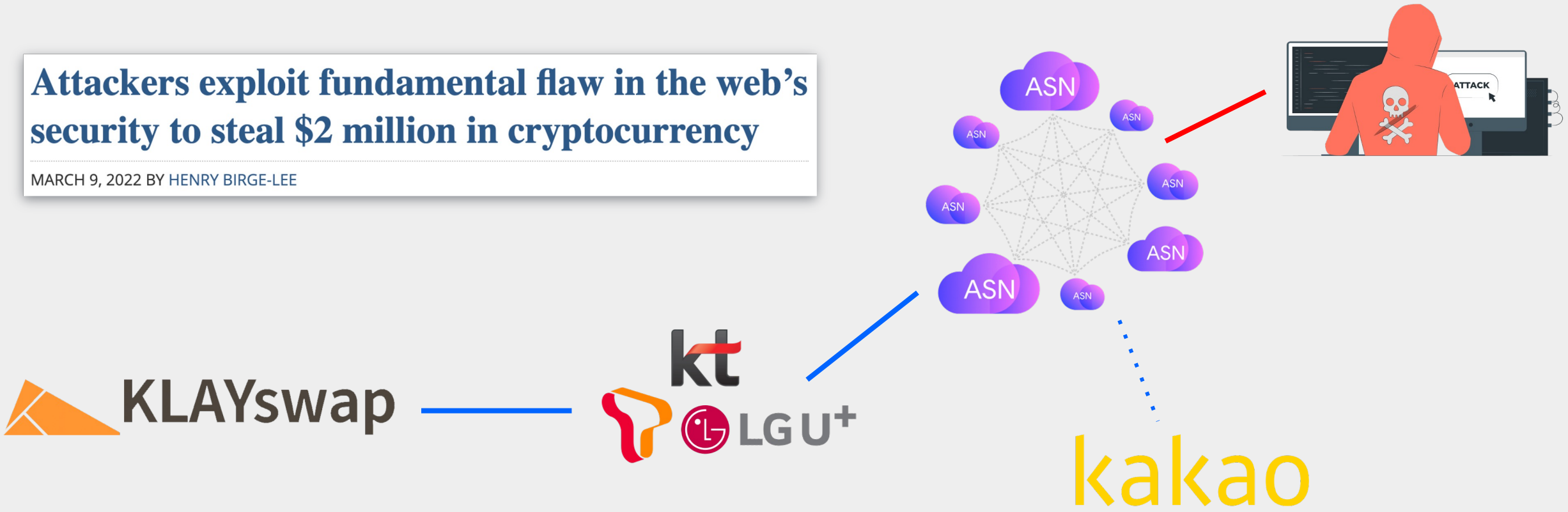
- ❖ 1 additional validator approved that company to sign on its behalf…🤦

🤯 Hack 1 company ⇒ Control 5 of 9 validators ⇒ Profit

Real World Case V: KLAYswap

Attackers exploit fundamental flaw in the web's security to steal $2 million in cryptocurrency

MARCH 9, 2022 BY HENRY BIRGE-LEE



Real World Case V: KLAYswap

💸 $2M of losses

💥 Infrastructure & Web2 compromise ⇒ Damage in Web3

(Web3 Smart Contract was SAFU 🤯)

🙀 BGP hijack resulting in front-end loading attacker's code

🙀 SSL/TLS bypass possible with Let's Encrypt and ZeroSSL

🤔 KLAYswap used CloudFlare

- ❖ More difficult to hijack as CloudFlare is widely announced
- ❖ Instead, attacker targeted a library hosted on a third-party server

Real World Case V: KLAYswap

Celer Network hacked with BGP hijack 7 months later ☠️

- ❖ Hosted on Amazon AWS, but still vulnerable to BGP hijack

Celer

Celer Network cBridge Users Lose $240k in DNS Hijack, CELR Lists on Coinbase

Jamie McNeill

Last updated: 19 August 2022

BGP hijacks are NOT going away.
Protocols must take precautions!

Real World Incidents – Hands on exercise

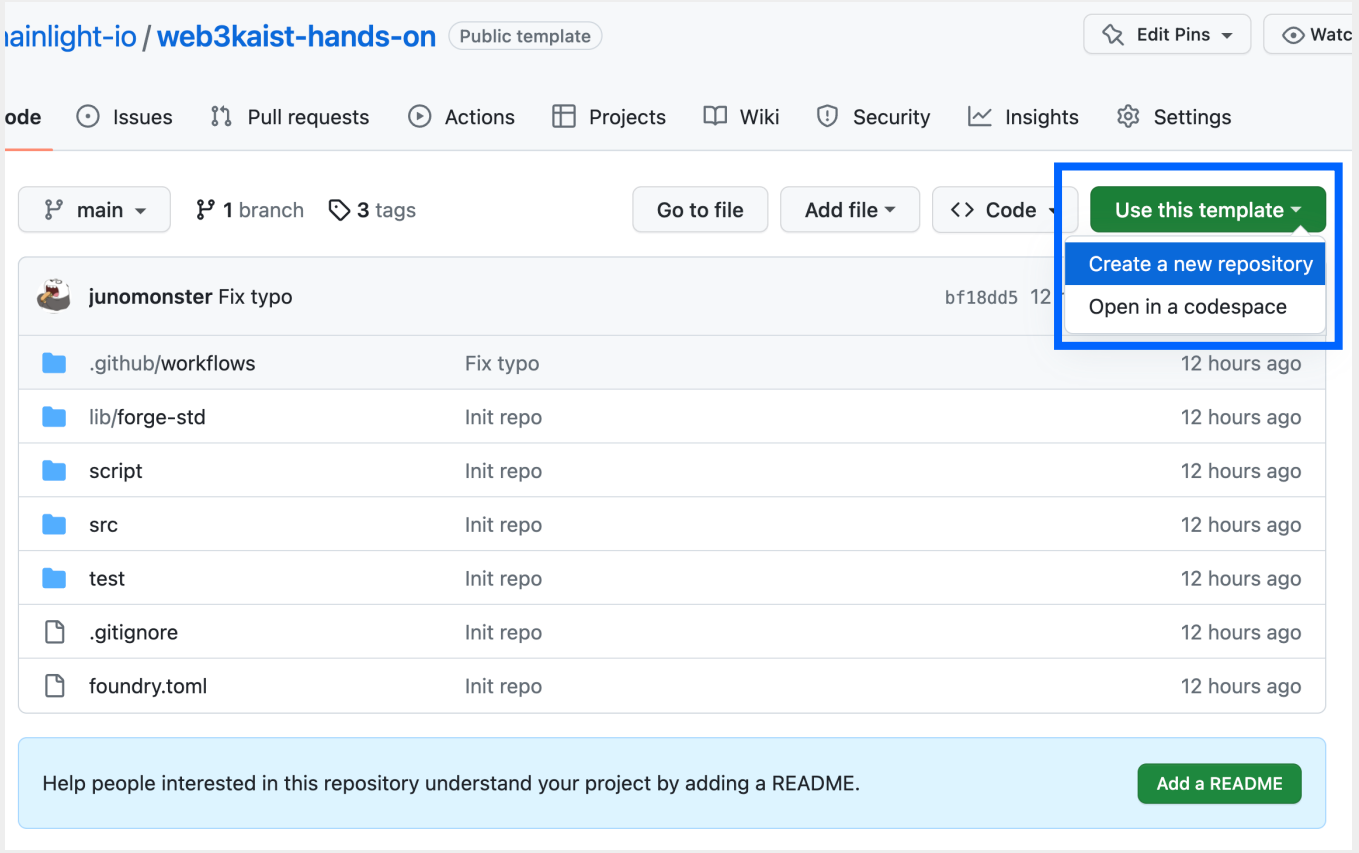
- Testing environment setup: GitHub Codespace + foundry-rs
- Hands on exercise: Code with me
  - Reentrancy Bug Easy
  - Reentrancy Bug Hard
  - Integer Over/Underflow

Real World Incidents – Hands on exercise

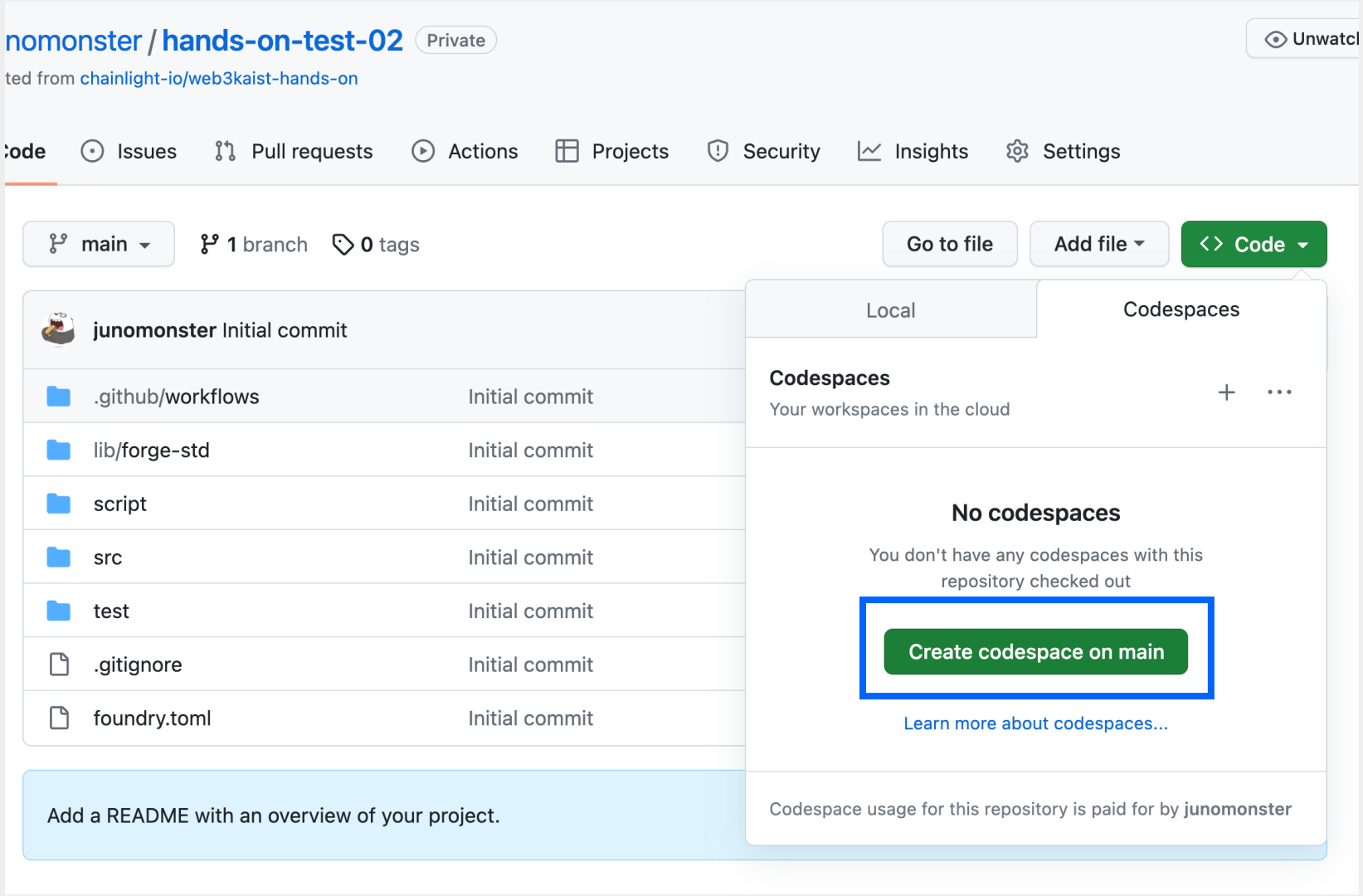
- ”Foundry”: blazing fast, portable and modular toolkit for Ethereum application development written in Rust 🦀
  - Fast & flexible compilation pipeline
  - Tests are written in Solidity
  - Fast fuzz testing
  - Fast remote RPC forking mode
  - Flexible debug logging
  - Portable (5-10MB) & easy to install
  - Fast CI
- Test like a pro (KR): https://www.youtube.com/watch?v=C8V8mlxwgXI&t=1731s

Hands on exercise – Create your own testbed

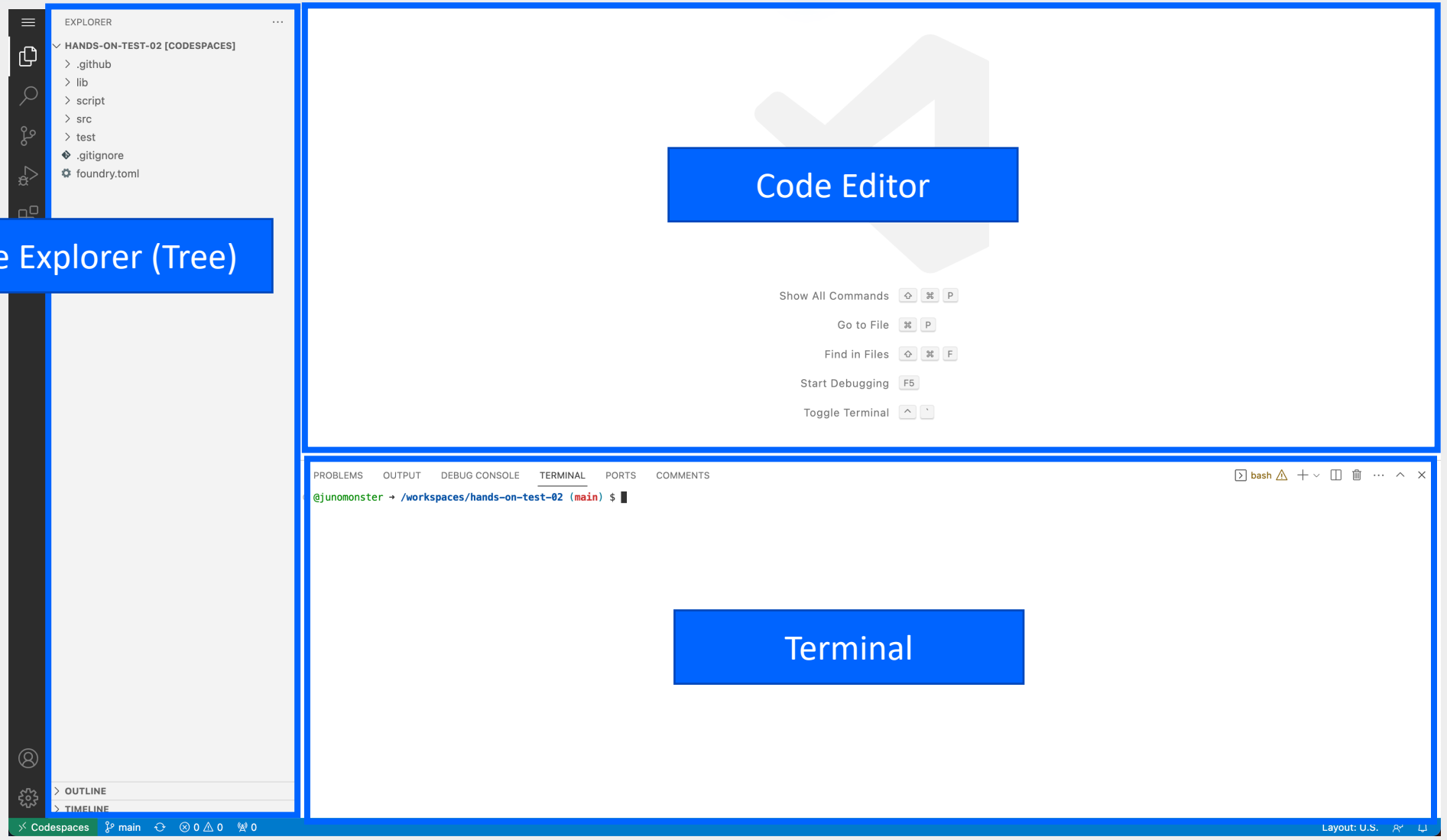
- https://github.com/chainlight-io/web3kaist-hands-on



Hands on exercise - Create your own testbed



Hands on exercise – Create your own testbed



Hands on exercise – Create your own testbed

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS COMMENTS

Download Installer

```
● @junomonster → /workspaces/hands-on-test-02 (main) $ curl -L https://foundry.paradigm.xyz | bash
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
  0     0    0     0    0     0      0      0 --:--:-- --:--:-- --:--:--     0
100  1887  100  1887    0     0   5770      0 --:--:-- --:--:-- --:--:--  5770
Installing foundryup...
################################################################################ 100.0%

Detected your preferred shell is bash and added foundryup to PATH. Run 'source /home/codespace/.bashrc' or start a new terminal session to use foundryup.
Then, simply run 'foundryup' to install Foundry.
```

Load Installer

```
○ @junomonster → /workspaces/hands-on-test-02 (main) $
● @junomonster → /workspaces/hands-on-test-02 (main) $ . ~/.bashrc
○ @junomonster → /workspaces/hands-on-test-02 (main) $ foundryup
```

Execute Installer

```
.x0x.x0x.x0x.x0x.x0x.x0x.x0x.x0x.x0x.x0x.x0x.x0x.x0x.x0x.x0x.x0x.x0x.x0x.x0x.x0x

 ╔═╗╔═╗╦ ╦╔╗╔╔╦╗╦═╗╦ ╦
 ╠╣ ║ ║║ ║║║║ ║║╠╦╝╚╦╝    Portable and modular toolkit
 ╚  ╚═╝╚═╝╝╚╝═╩╝╩╚═ ╩     for Ethereum Application Development
                            written in Rust.

.x0x.x0x.x0x.x0x.x0x.x0x.x0x.x0x.x0x.x0x.x0x.x0x.x0x.x0x.x0x.x0x.x0x.x0x.x0x.x0x

Repo       : https://github.com/foundry-rs/
Book       : https://book.getfoundry.sh/
Chat       : https://t.me/foundry_rs/
Support    : https://t.me/foundry_support/
Contribute : https://github.com/orgs/foundry-rs/projects/2/

.x0x.x0x.x0x.x0x.x0x.x0x.x0x.x0x.x0x.x0x.x0x.x0x.x0x.x0x.x0x.x0x.x0x.x0x.x0x.x0x

foundryup: installing foundry (version nightly, tag nightly-388c3c0a528cdee61498372d52e605f993674570)
foundryup: downloading latest forge, cast, anvil, and chisel
########################################
```

Hands on exercise – Create your own testbed

❖Lecture goal: Pass the three test cases below

```
⊗ @junomonster → /workspaces/hands-on-test-02 (main) $ forge test -v
  [⠒] Compiling...
  No files changed, compilation skipped

  Running 3 tests for test/SafeVaultExploit.t.sol:SafeVaultTest
  [FAIL. Reason: Assertion failed.] testIntegerOverUnderflow() (gas: 40188)
  [FAIL. Reason: Assertion failed.] testReentrancySuccessEasy() (gas: 46061)
  [FAIL. Reason: Assertion failed.] testReentrancySuccessHard() (gas: 46026)
  Test result: FAILED. 0 passed; 3 failed; finished in 10.20ms

  Failing tests:
  Encountered 3 failing tests in test/SafeVaultExploit.t.sol:SafeVaultTest
  [FAIL. Reason: Assertion failed.] testIntegerOverUnderflow() (gas: 40188)
  [FAIL. Reason: Assertion failed.] testReentrancySuccessEasy() (gas: 46061)
  [FAIL. Reason: Assertion failed.] testReentrancySuccessHard() (gas: 46026)

  Encountered a total of 3 failing tests, 0 tests succeeded
○ @junomonster → /workspaces/hands-on-test-02 (main) $ 
```

Hands on exercise – Code with me (Live Coding)

- The final answers are available on the main repo's tags:
  - https://github.com/chainlight-io/web3kaist-hands-on/tree/ReentrancyEasyAnswer
  - https://github.com/chainlight-io/web3kaist-hands-on/tree/ReentrancyHardAnswer
  - https://github.com/chainlight-io/web3kaist-hands-on/tree/IntegerOverUnderflowAttackHandlerAnswer

Future-proof your Security

Preparing for Safe Web3 Ecosystem

Theori

The way to more secure Web3 ecosystem

Secure Coding

Test Driven Development

Security Audits

Bug Bounty

If anything changes, do the above steps again!

Wrap Up

Cyber / Web3 Security

- Introduce ability to implement any application logics in a decentralized environment
- Immutability, transparency, distributed, and decentralized are exciting features, but security is important
- It is a relatively new field and expected to mature over the next few years
- Smart contracts are still human-implemented programs and are not immune to mistakes
- **However, Web3 security is not just about smart contract security**
- Requires not only traditional security skills, but also blockchain-specific and financial engineering knowledge

Thank You

Web3 Security
Adventure to Safer Web3 World

Brian Pak, CEO, Theori
brian@theori.io

Juno Im, Lead, ChainLight
juno@theori.io

Theori