D2: Access Control

Portland State
Computer Science

#2: Access Control

- Similar to OWASP Top 10
- Insufficient access control and authentication checks
- Insecure access control methods
- Private, internal functions and data are accessible through a contract's public/external functions
- Results in unauthorized access
- **Loss**: estimated at 150,000 ETH (~$30M USD at the time)

Walkthrough scenario

- A **smart contract** designates the address which initializes it as the contract's owner in an initialization function
  - Grants special privileges such as the ability to withdraw the contract's funds.
- Initialization function not protected and can be called by anyone — even after it has already been called
- Allows anyone to become the owner of the contract and take its funds.

Example

- Owning a wallet contract (7/19/2017)
  - https://blog.zeppelin.solutions/on-the-parity-wallet-multisig-hack-405a8c12e8f7

Santiago Palladino Follow
Developer and Security Researcher at Zeppelin Solutions.
Jul 19, 2017 · 3 min read

The Parity Wallet Hack Explained

TL;DR

- A vulnerability was found on the Parity Multisig Wallet version 1.5+, that allowed an attacker to steal over 150,000 ETH (~30M USD).

It was possible to turn the Parity Wallet library contract into a regular multi-sig wallet and become an owner of it by calling the initWallet function. -- Parity

  - Could have been up to ~$180M, but white hat hackers "stole" the rest and returned it to rightful owners
    - https://medium.freecodecamp.org/a-hacker-stole-31m-of-ether-how-it-happened-and-what-it-means-for-ethereum-9e5dc29e33ce

Code vulnerability example #1

- Contract's initialization function sets the caller of the function as its owner.

```
function initContract() public {
  owner = msg.sender;
}
```

  - Logic is detached from the contract's constructor and does not keep track of the fact that it has already been called.
  - Anyone can call `initContract` after contract creation to become owner

Code vulnerability example #2

- Parity WalletLibrary in example
- Library used to implement common wallet functions
  - Initializer allows one to specify withdraw limit and owners

```
function initWallet(address[] _owners, uint _required, uint _daylimit) {
    initDaylimit(_daylimit);
    initMultiowned(_owners, _required);
}
```

- Library implemented as an external contract call to reduce costs
  - Rather than have each contract deploy a copy of the exact same library code, wallets do this…

```
address constant _walletLibrary =
0xa657491c1e7f16adb39b9b60e87bbb8d93988bc3;
```

  - Then, use `delegatecall()` to invoke its functions
    - `DELEGATECALL` instruction in EVM takes call and invokes the exact same one on the contract you're using it on

```
function isOwner(address _addr) constant returns (bool) {
  return _walletLibrary.delegatecall(msg.data);
}
```

- Issue within fallback function
  - Fallback receives payment if someone sends you $
  - Otherwise, `msg.data` has unknown function call that should be handled by library since no function in contract matches
    - `delegatecall` dispatches unknown calls to library

```
function() payable {
    if (msg.value > 0)
        Deposit(msg.sender, msg.value);
    else if (msg.data.length > 0)
        _walletLibrary.delegatecall(msg.data);
}
```

  - Issue: ALL public calls in library can now be called (including `initWallet` again!)
  - Leads to..
    - Unintended call to `initWallet` https://etherscan.io/tx/0x707aabc2f24d756480330b75fb4890ef6b8a26ce0554ec80e3d8ab105e63db07
    - Followed by transfer out of wallet WHG https://etherscan.io/tx/0x9654a93939e98ce84f09038b9855b099da38863b3c2e0e04fd59a540de1cb1e5

Code vulnerability example #3

- MetaCoin contract for purchasing and exchanging coins
  - `sendCoin` call to `doTransfer` from `msg.sender` to `receiver`

```
contract MetaCoin {
  mapping (address => uint) public balances;

  function sendCoin(address receiver, uint amount) public returns(bool sufficient) {
    if (balances[msg.sender] < amount) return false;
    doTransfer(msg.sender, receiver, amount);
    return true;
  }

  function doTransfer(address from, address to, uint amount) {
    balances[from] -= amount;
    balances[to] += amount;
  }
```

  - What errors are there?
    - `doTransfer` not set to `internal` (can be called externally)
    - No check on `from` being `msg.sender` in `doTransfer`
    - Bonus vulnerability: Underflow and overflow on balances update not checked

Code vulnerability example #4

- Same contract

```
contract MetaCoin {
  string private secretPassword;
  mapping (address => uint) public balances;

  function mintNewCoins(uint amount, string password) public {
    require(password == secretPassword);
    balances[msg.sender] += amount;
  }
```

  - What is the error?
    - Contract's password set to "`private`", but appears in clear on blockchain
    - Find `secretPassword` and mint coins
  - Everything is public by design
    - Contract code & storage
    - Transaction contents
    - Private modifier does nothing for secrecy!

Code vulnerability example #5

```
def input(choice):
        if self.storage["player1"] == msg.sender:
                self.storage["p1value"] = choice
                return(1)
        elif self.storage["player2"] ==  msg.sender:
                self.storage["p2value"] = choice
                return(2)
        else:
                return(0)
```

- Adversary can see everything!
  - Must use a bit-commitment protocol
  - Two players commit to a keyed cryptographic hash of choice
  - Both reveal choice to determine winner

Remediation

- Remove all catch-all function dispatchers (specify exact calls allowed)
- Ensure calls are `internal`, unless intended to be `external`
- Validate identity before execution using modifiers and via `require`

```
contract Unprotected{
  address private owner;

  modifier onlyOwner {
    require(msg.sender==owner);
    _;
  }

  function constructor() public {
    owner = msg.sender;
  }

  // This function should be protected
  function changeOwner_broken(address _newOwner) public {
    owner = _newOwner;
  }

  function changeOwner_fixed(address _newOwner) public onlyOwner {
    owner = _newOwner;
  }
}
```

SI CTF Lab 3.4, 3.5