Solend Auditing Workshop

Smashing the lamports for fun and profit

twitter.com/solendprotocol
twitter.com/0xrooter

JUNE 30 2021

Introduction

- Not enough auditors in the space
- Program developers should have a deep understanding of security concepts
- Some best practices from Ethereum translate to Solana

Introduction

- Smart contracts more like rockets than web dev
    - You only get one shot to get it right
    - Finite time to write secure code, hackers have infinite time to hack
- Information in this presentation can be used dangerously
    - Be ethical

Agenda

- Types of issues
- Known attacks
- Recommendations
- Solana-specific notes
- Post-deployment
- Audit methodology

Types of issues

- Stealing funds
- Locking funds
  - Ransom
  - Griefing
  - **Usually happens when tx can’t complete due to always erroring**
- Skimming
  - Giving users slightly worse outcome
- Anything not intended by the developers

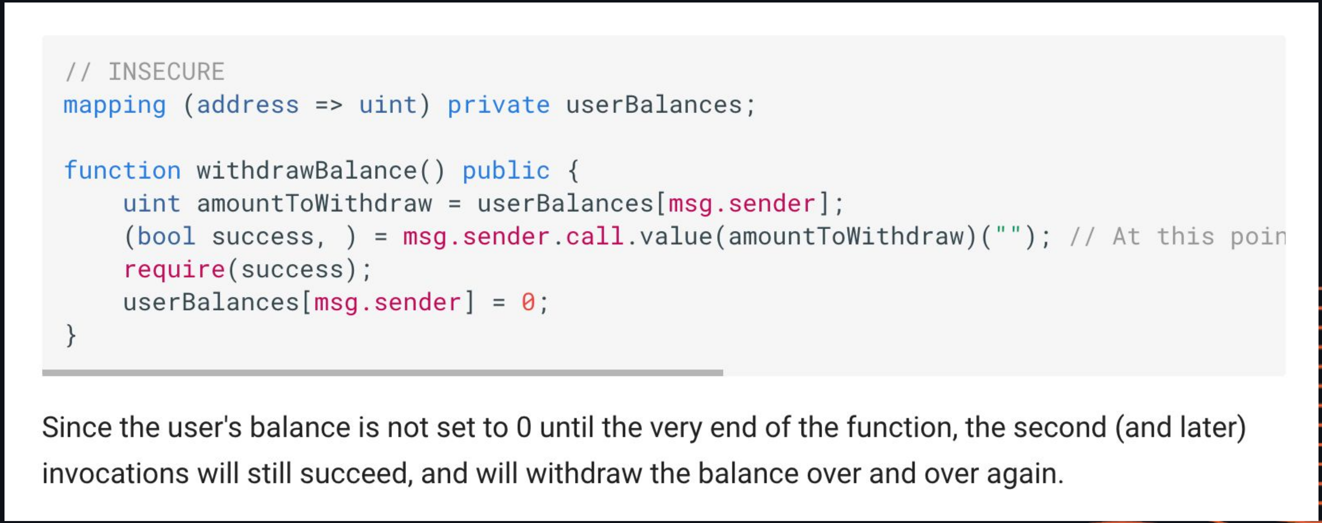
Known Attacks

Known attacks (not exhaustive)

- Reentrancy
- Frontrunning
- Economic
- Authorization
- Oracle manipulation
- Honeypots
- Block stuffing
- Safe math
- Timestamp spoofing

Known attacks: Reentrancy

- Calling external program which takes over control flow to call back into program, causing unintended behavior



```
// INSECURE
mapping (address => uint) private userBalances;

function withdrawBalance() public {
    uint amountToWithdraw = userBalances[msg.sender];
    (bool success, ) = msg.sender.call.value(amountToWithdraw)(""); // At this poin
    require(success);
    userBalances[msg.sender] = 0;
}
```

Since the user's balance is not set to 0 until the very end of the function, the second (and later) invocations will still succeed, and will withdraw the balance over and over again.

Known attacks: Reentrancy

- June 17 2016: The DAO hacked using first reentrancy attack
  - 3.6M ETH ($50M) stolen
- The DAO had 30 day withdrawal delay (speed bump)
- Critical update issued to rollback the hack
  - Resulted in Ethereum Classic fork

Known attacks: Frontrunning

- Calling program just before another user to influence outcome
- MEV

Known attacks: Frontrunning

- Frontrun
  - Buy before market buy on AMM
- Backrun
  - Sell right after market buy
  - Essentially an arb back to market
- Sandwich: frontrun then backrun
- Griefing
  - Cause user’s tx to fail

Known attacks: Economic

- Economic rules that can be exploited to steal funds
- Difficult to spot when auditing code
    - Code can be correct, but allow economic exploits

Known attacks: Economic

- Augur example: https://www.youtube.com/watch?v=8zJD1zsTfQs&feature=youtu.be&t=120
- Yearn example: https://github.com/yearn/yearn-security/blob/master/disclosures/2021-02-04.md

Known attacks: Authorization

- Gaining access to authorized-only functions
- Developers forget to lock down only-owner functions
  - You’d be surprised
- Hacker stealing authorized role private key or disgruntled employee with access

Known attacks: Authorization

- Parity didn’t have auth on selfdestruct(), causing 500k ETH ($300M) to be locked
    - Community rejected rollback proposals
- Icon (ICX) freeze() checked msg.sender != owner, instead of msg.sender == owner
    - Mitigated by running a bot to unfreeze anytime it was frozen (lol)

Known attacks: Oracle manipulation

- Exploits that arise from manipulation of an oracle
- Usually incorrect prices being used for swaps or liquidations
- Must trust oracle publisher **and** their opsec
  - Hacker
  - Disgruntled employee
- Consider value secured by oracle
  - This is the bounty for exploiting the oracle
  - Derivative consumers (aka parasitic) can quietly add to this value

Known attacks: Oracle manipulation

- Synthetix: https://messari.io/article/synthetix-suffers-an-oracle-attack-that-lost-roughly-37-million
- bZx: https://www.palkeo.com/en/projets/ethereum/bzx.html
- Compound: https://www.comp.xyz/t/fix-the-compound-oracle-problem/723
    - Pumped DAI/USDC on Coinbase, liquidated accounts
- (Theory): Steal oracle private key, publish bogus prices

Known attacks: Honeypots

- Programs that appear innocuous but trap users
- Fake tokens on Uniswap
  - Allows buying, but tx always fails on sell
  - Note, this specific example not as applicable to Solana due to SPL tokens
    - Can’t specify custom logic into token itself
- Watch out for similar-looking characters “iIlL”

Known attacks: Block stuffing

- Suppressing other txs by filling blocks
- Less relevant in Solana due to throughput
- Fomo3D: https://medium.com/coinmonks/how-the-winner-got-fomo3d-prize-a-detailed-explanation-b30a69b7813f

Known attacks: Safe math

- Careful manipulation of inputs to cause an overflow
  - “Unsafe math”
  - Solidity uints wrap around, enabling funky manipulation
- Less relevant in Solana since should be using checked_op
- Still a footgun
  - Avoid situations causing ix to fail, can lock funds
- Avoid floats
  - 10,000,000 + 1 = 10,000,000 (even hundreds of times)
  - https://floating-point-gui.de/

Known attacks: Timestamp spoofing

- Timestamp can be manipulated slightly by miner
- In Ethereum, can be manipulated up to 15m
- Less relevant in Solana due to block speed

Known attacks recap

- Reentrancy
- Frontrunning
- Economic
- Authorization
- Oracle manipulation
- Honeypots
- Block stuffing
- Safe math
- Timestamp spoofing

Recommendations

Recommendations

- Favor pull over push
  - Withdraw funds example:
    - Pull: each user calls program to withdraw their own
    - Push: admin calls function to loop through users to send out funds
    - Push can’t complete when #users too high due to computation limit
  - for loops always have to be scrutinized
  - Could mitigate with pagination
- Generalize: use caution around any >O(1) routine
  - Threshold where function always fails => DOS
  - Worst case causes locked funds
    - Happened to Solend devnet (user entered too many reserves, could no longer act)
- Use caution when making external calls
  - Gives execution flow to external program, could be anything
  - Mark untrusted calls with comments

Recommendations

- Caution with publicly callable functions
  - Mark with comments
- Math overflow
  - Overflows in Solidity would wrap around silently, resulting in nasty exploits
  - Now overflows cause tx to fail, which can still be an issue for locking funds
- Rounding
  - Common with interest calculations
  - If numbers are slightly off, could cause overflow
- Neatly organize assertions

Recommendations

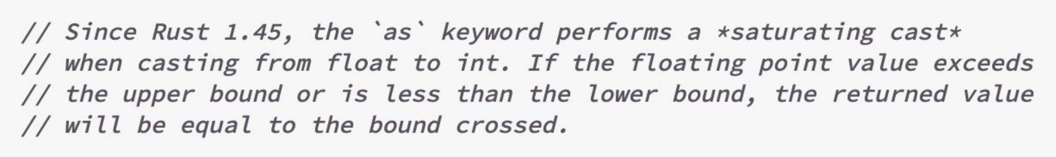
- Upgradability
  - Tradeoffs
- Circuit breakers
  - e.g. Freeze
  - Always leave withdrawals open
- Speed bumps
  - Delays
    - e.g. 24h delay before any parameter change can take effect
- Slow rollout
  - Gradually increasing cap
- Bug bounty

Solana-specific notes

- Understand limitations
  - Compute limit (200k)
  - Tx size limit (1232 bytes)
  - Accounts limit (35)
- Assert conditions about accounts
  - Assert account is rent exempt to avoid losing crucial data
  - Assert owner is expected
  - Check that data is deserialized correctly (assert version)
  - Assert global programs are correct (e.g. spl token program)
- Use caution when closing accounts
  - Closing an account releases lamports and is good hygiene
  - Account data remains until end of tx
    - Subsequent ixs referencing the soon-to-be-deleted account can have undefined behavior
    - Could even credit account with more lamports, cancelling the deletion
  - Check both that data is non-zero and lamports is non-zero

Post-deployment

- Just because a program is safe today, doesn’t mean it’s safe tomorrow
  - Platform update could introduce vulnerability
  - e.g.



```
// Since Rust 1.45, the `as` keyword performs a *saturating cast*
// when casting from float to int. If the floating point value exceeds
// the upper bound or is less than the lower bound, the returned value
// will be equal to the bound crossed.
```

  - Modifying parameters
- Monitoring
  - Initially page team
  - Once confident in accuracy and runbooks, auto-respond
- Runbooks for reacting to emergencies

Auditing Methodology

Auditing methodology

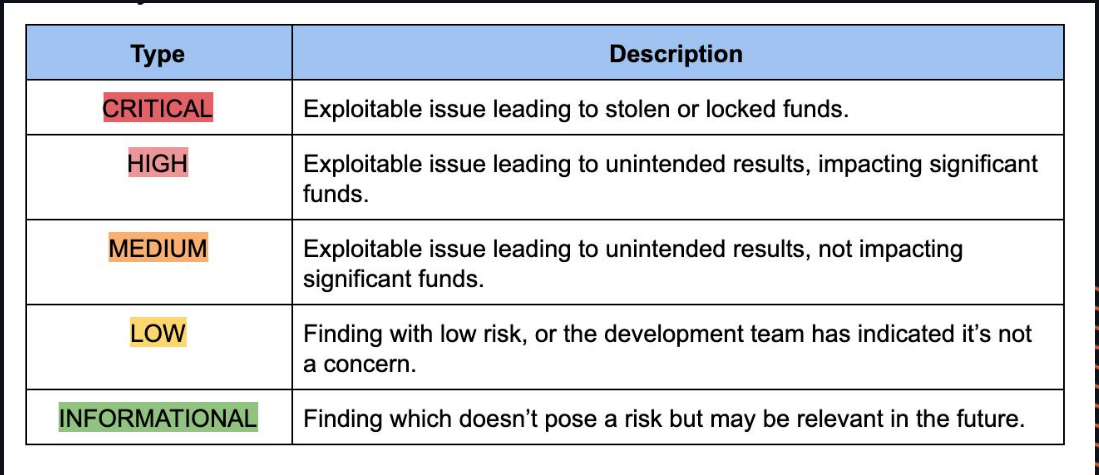
- Comes down to hours of reading and walking through potential exploits
- Build a mental library of known exploits
  - Read other audit reports: Compound, Aave
- Should be audited by fresh eyes
  - Author tends to have tunnel vision after so long
  - If developer audits, take a good break
- Independent auditors to decrease groupthink

Auditing methodology

- Split into squads
  - Detailed readthrough of code
  - Leave no stone unturned
- Start with a flow
  - e.g. Withdraw
- Create a checklist of invariants
  - e.g. Only user’s balance should change
  - e.g. Should only be able to withdraw up to withdrawable amount
  - Note, often the result of auditing is “it works as intended.” Checklist is a way to “show your work”
- Consider edge cases
- Consider args supplied by caller (especially if public function)
- Generally while reading, something inspires a known attack vector
  - Follow the thread (rubber ducking), looking for ways to cause unintended behavior

Auditing methodology

- Take notes
- Categorize into severity
  - CRITICAL, HIGH, MEDIUM, LOW, NIT



Type	Description
CRITICAL	Exploitable issue leading to stolen or locked funds.
HIGH	Exploitable issue leading to unintended results, impacting significant funds.
MEDIUM	Exploitable issue leading to unintended results, not impacting significant funds.
LOW	Finding with low risk, or the development team has indicated it’s not a concern.
INFORMATIONAL	Finding which doesn’t pose a risk but may be relevant in the future.

Thank you

Appendix

Resources

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

Other example exploits

- [Add here]