Chapter 9: Smart Contract Security.

Smart contracts implement defensive programming which includes:
1. Minimalism and simplicity.
2. Code reuse.
3. Code quality.
4. Readability and Audit-ability.
5. Test coverage.

Examples of vulnerabilities:

1. ReEntrancy attack.
The DAO learned a lot from this attack.

2. Arithmetic Over/Under-flows.

3. Stupid uses of this.balance and address(this).balance.

4. Delegate-call.

5. Default visibilities.

6. Entropy Illusion.
In February 2018 Arseny Reutov blogged about his analysis of 3,649 live smart contracts that were using some sort of pseudo-random number generator (P.R.N.G.); he found 43 contracts that could be exploited.

Now this man has some god-tier endurance.

7. External Contract Referencing.
It is advised to read thoroughly the contract you want to reference before referencing to it.

8. Short address and parameter attack.
This attack is generally carried out on the front end of the blockchain program, it is as a result of no verification of the validity of the address.

9. Unchecked CALL return values.
Do not use call or send when transferring ether, as they do not revert the transaction when they fail.

10. Race conditions and Front running.
The vulnerability of a smart contract where an attacker observes the transactions in a contract and remakes the valuable ones with a higher gas price so that his own is included in the block to be mined.

This can also be done by the miners of the blockchain, this is very insecure.

11. Denial of Service (D.O.S).
This attack renders a contract unusable for a period of time or forever, depending on how evil the attacker is.

12. Block Timestamp Manipulation.
According to instructions, it is necessary to not use block.timestamp for important contract changes because they can easily be manipulated by evil miners. Rather, it is advised that block numbers are used as they cannot be easily manipulated by the miners.

13. Constructors with care.
Avoiding the use of the same contract name functions to set constructors as any slight mistake would make it change to a normal callable function, and can easily be hacked.

14. Uninitialized storage pointers.
The use of poorly initialized storage and memory variables.

Structs default to storage if "memory" is not specified. But do they overwrite the existing slots??

15. Floating points and precision.
Extreme care should be taken when working with multiplications and divisions in

Solidity.

16. Tx.Origin Authentication.
tx.origin is no longer in phase, it is much better to use the msg.sender to observe.

tx.origin is used to spot the wallet or address that made the first call to a series of transactions. If address A calls contract B which calls another contract C, the tx.origin at contract C is A.

17. Contract Libraries.
Use standard, safe and secure libraries like OpenZeppelin.

The most fundamental software security principle is to maximize the reuse of trusted code. “Don’t roll your own crypto”.