Audit of Smart Contracts

The rising deployment of *smart contracts* demands strong security assurances. Regrettably, it is challenging to create *smart contracts* that are free of security bugs. As a consequence, critical vulnerabilities in *smart contracts* are discovered and exploited periodically. Hence Auditors need to perform Smart Contract Audits making use of *data analytics*

www.manojmehta.com

Smart Contract

Smart contracts perform similar functions as paper-based agreements. The differentiating factor about smart contracts is that these are digital as well as self-executable in nature.

A smart contract is a self-executing computer program with the terms of the agreement between two entities being directly written into lines of code, intended to automatically execute, control or document the terms. The code and the agreements contained therein exist across a distributed, decentralized blockchain network

All smart contracts share some common features:

- Transparent – being on a distributed network, the status of smart contract is visible to all on the network.
- Immutable – a smart contract once programmed in the code, cannot be changed or cancelled.
- Tamper Proof – with immutability embedded in the code, the smart contract cannot be easily tampered.

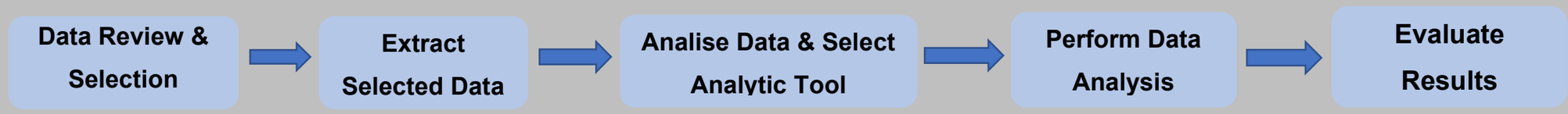
Having said this, there may be vulnerabilities in the code at the developer stage, intentional or otherwise, and bugs creeping into the code later. To ensure that such applications are safe, security audit of smart contracts is needed to check bugs and vulnerabilities.

Smart contract audit or security audit is a thorough analysis of the smart contract embedded code, in order to rectify design issues, errors in the code, or security vulnerabilities.

In Brief

Audit scenario is changing, technology has transformed business processes and created a wealth of data that can be leveraged by accountants and auditors with the requisite mindset. Data analysis can enable auditors to contribute towards a 'sustainable decentralised ecosystem'.

This briefing aims at basic data analytics and risk assessment concepts to enable accountants to understand this new environment, and does not focus on erstwhile auditing and accounting standards. A flowchart for applying data analytics, with emphasis on Audit by Exception (ABE), is depicted below:



Why Audit

Are all smart contracts essentially ‘smart’? Veracity is a critical issue, which is further escalated due to the recent hacks of smart contracts. An audit is vital to critically identify bugs, vulnerabilities and security flaws in smart contracts, which may have skipped detection at development stage. Due to its immutable nature, it is difficult (even impossible) to upgrade an already deployed smart contract. An Audit could be the final chance (before deployment) to save the project from being another victim of cyberattacks.

Smart Contract Vulnerabilities

Unfortunately, smart contracts have some vulnerabilities which has resulted in millions being ripped off.

Transaction Ordering Dependence (TOD)

One miner can outrun the transactions of other miner by paying more gas, and only the miner who closes the block decides the transaction order, which is the TOD vulnerability. In TOD, you may face unexpected malicious behaviour from miners.

Timestamp Dependency (the Verge Coin hack)

When a smart contract uses the timestamp in order to generate random numbers, the miner can post a timestamp within 30 seconds of the block being validated, enabling to predict a more preferred option to his chances in this lottery.

Re-Entrancy Attack (the bitcoin DAO Hack)

The re-entrancy vulnerability involved functions which could be repeatedly called before the first function call was ended. For this reason, different function external calls could interact in destructive ways.

Exception or Ownership Disorder (the Oyster Pearl or Parity attack)

The owner of smart contract can use the exception disorder vulnerability of the smart contract to avoid paying out the winning sum to its participants.

Integer Overflow – Underflow ($2^{256}+1 = 0$, $0-1 = 2^{256}$)

This error occurs when there is more data in a buffer than it can handle, causing data to overflow into adjacent storage. This can cause a system crash or worse, create an entry point for a cyberattack. Buffer underflow occurs when the temporary holding space during data transfer, is fed at a lower rate than it is being read from, resulting in stoppage of writing action, and the receiving device may get corrupted.

External Calls

One of the major dangers of calling external contracts is that they can take over the control flow, and make changes to your data that the calling function wasn't expecting.

SQL Injection (the NEO wallet hack)

Attackers are able to inject malicious instructions into benign ones, all of which are then sent to the database server through a web application. This allows anyone to change the token's total supply limit by transferring their own tokens to an unspecified address.

The SWC Registry offers a complete and up-dated index of known smart contract vulnerabilities and anti-patterns along with real world examples. Browsing the registry is a good way of keeping up-to-date with the latest attacks

Smart Audit Checks

To ensure your code is easily trackable by auditors, team members, automated audit tools and the community, it is advisable to follow the style guide based on 'Solium's Standards'. Having a set and automatically enforced style guide will additionally make it easier to spot erroneous code.

(1) **Correct Functions Visibility**

Functions in solidity can have four visibility specifiers: public, external, internal or private, with public being the default. State variables can be public, internal or private, with internal being the default. Explicitly specifying the visibility of functions and state variables is a security best practice. Absent specifiers can be dangerous especially in the case of functions where the default is public accessibility. If such a function has critical logic then it can be triggered from any external address to potentially misuse the contract. The first hack on the Parity Multisign Wallet exploited such missing function visibility specifiers leading to the attacker stealing $31M worth of Ether.

(2) **Data Storage**

In solidity, data can be stored in memory which is non-persistent and less expensive or in storage which is persistent and very expensive, and such parameters should be optimised while writing *smart contracts*. By default, state and local variables are stored in storage and function parameters are stored in memory.

(3) **Prevent overflow or underflow**

Underflow or overflow can be prevented by using 'SafeMath' library to perform math operations in smart contracts.

(4) **External Contract**

As such, every external call should be treated as a potential security risk. When it is not possible, or undesirable to remove external calls, use the recommendations.

(5) **Check for Re-Enterancy**

Re-entrancy attack, probably the most famous Ethereum vulnerability, surprised everyone when discovered for the first time, during a multi-million heist which led to a hard fork of Ethereum.

(6) **Delegate Call Check**

The 'delegatecall' function is used to call functions from other contracts as if they belong to the caller contract. Thus, the caller may change the state of the calling address. This may be insecure.

(7) **Save Gas on *Smart Contracts***

Saving gas is necessary to build an efficient smart contract. It is a tricky issue faced by the developers. Auditors need to understand which instructions consume more gas and how it can avoid or minimize them.

(8) **Timestamp Dependence**

If the contract function can tolerate a 15-second drift in time, it is safe to use block timestamp

(9) **Compiler warnings**

All the compiler warnings are serious issues, ignored sometimes by developer and contracts are deployed, making them vulnerable to threats. Audit recommends necessary action to remove all the warnings.

(10) **Ownership of the deployed contract**

It is very important to provide ownership to a contract at the time of deployment or a restriction to function calls else attacker may call those function or transfer ownership. The 'Oyster Pearl Hack' is famous case here, as in this case ownership of smart contract was open, the attacker transferred ownership to himself and was able to mint tokens of worth $ 300,000.

(11) **Oracle calls**

Oracles are third party services which are not part of the blockchain consensus mechanism. The main challenge with oracles is the need to trust these sources of information and verify.

(12) **Pragma Locks to specific compiler version**

Since the locking is done through the filesystem, any database that does not have an associated file will return unknown. This includes both in-memory databases and some temporary databases.

(13) **Automated Audit Tools**

After manual and unit testing, your smart contract undergoes automation testing that is done using several open source security tools, specified below.

Visualization tools: Surya, Solgraph, Evm-Labs, Ethereum-graph-debuger

Static & Dynamic Analysis: Mythril, Oyente, Securify, Smartcheck

Test Coverage: Solidity-coverage

Linters (linters improve the code quality): Solcheck, Solhint, Solium

Audit Checklist

Once the above discussion is clear, let's take a look at the structure of a Smart Contract Audit:

(a) Disclaimer: Here the Auditor says that the audit is just a report of findings and not a legally binding document and that it doesn't guarantee the integrity of smart contract.

(b) Overview: A quick view of the Smart Contract, proposed to be audited and good practices found.

(c) Hacks or Attacks: In this section you'll talk about the attacks performed to the contract and the results. Just to verify that it is, in fact secure.

(d) Critical Vulnerabilities: Determine critical issues that could damage substantially the integrity of the contract. Certain bug that would potentially allow attackers is a critical issue.

(e) Medium vulnerabilities: Determine those vulnerabilities that could cause limited damage to the contract. Like a bug allowing people to modify a random variable.

(f) Low Vulnerabilities: Determine those issues that don't really damage the contract and could exists in the deployed version of the contract.

(g) Line by line Observations: In this section the Auditors analyses the most important code lines where there is scope for potential improvements.

(h) Concluding: Auditors note on the audit process, opinion about the contract security, operational integrity and remarks.

The Bottom Line

There are still many unknowns with respect to how blockchain will impact the audit and assurance profession, including the speed with which it will do so. Blockchain will likely do away with the need to have “cloud” accounting software centrally hosted at some data centres. Permissioned blockchains shall require higher efforts then the Permissionless ones. While we cannot change the fact that technology is disrupting our profession, we can choose to embrace it and find new, value-added ways to serve our clients.