Bitcoin architecture research by analysing the github code and the

My main aim : to check the hypothesis of james that its possible to have the attack on the blockchain using meterpreter and the trove tools to real the real transaction . whether to check whether the code provides some fuzziness and other safety features.

implementation :-

**Some benchmarks :**

1. Chacha\_poly protocol :- in this , we used the modified version of standard MAC to ensure the optimised version on memory constraint hardware ( on constraint wallet devices ) could be run .
2. Mempool eviction and bloom filter test and other steps of PoW implementations that will encode the transaction in the wallet , create the UTXO and also do the validation.

Consensus :

1. Merkle tree construction :- there are issues in their implementation , if , due to generation of odd keys in the tree , the last leaf node will be duplicated to make it even , thus making theroot signature compromised. The solution is to keep as validity Boolean check and then stopping when the check is false.

**Security checks on the elf :-**  interesting python file going on the readable . so they do the following :- position independent execution test of the bitcoin transaction script so that the address stored position is randomized which is the another layer of security on the encrypted communication. This include the pasing of headers with checking the flags and headers are notchanged due to the changes. Also checking the type of file implementation which includes the

Checking for the read only relocations to check the binding condition of the flags to the code as it changes the addresses and also checks for the flags

Also the use of stack canary to avoid buf overflow attacks.

Also for the main linking files of the bitcoin , we will be checking the following codebase :

ALLOWED\_LIBRARIES = {

# bitcoind and bitcoin-qt

'libgcc\_s.so.1', # GCC base support

'libc.so.6', # C library

'libpthread.so.0', # threading

'libanl.so.1', # DNS resolve

'libm.so.6', # math library

'librt.so.1', # real-time (clock)

'libatomic.so.1',

'ld-linux-x86-64.so.2', # 64-bit dynamic linker

'ld-linux.so.2', # 32-bit dynamic linker

'ld-linux-aarch64.so.1', # 64-bit ARM dynamic linker

'ld-linux-armhf.so.3', # 32-bit ARM dynamic linker

'ld-linux-riscv64-lp64d.so.1', # 64-bit RISC-V dynamic linker

# bitcoin-qt only

'libxcb.so.1', # part of X11

'libfontconfig.so.1', # font support

'libfreetype.so.6', # font parsing

'libdl.so.2' # programming interface to dynamic linker

}

And afterwards use the library of this consortium to use it as the stepping stone to check the integrated testing and dependency libraries.

Some alternative hash functions have also come out to make the PoW not the terrible monstor of energy guzzler , but even being able to verify and encode the transaction on embedded hardware.

Indexing :- the index DB consist of the disk loc on filter info corresponding to the block , sha256 hash and the header. These are the serialised being added into the informationDB and then being encrypted with the DBhashof the ey which will thn will be used to authenticate. Each block can have filtered types mapped by the filter\_type). Also getting the invalid argument will make the complete transaction block invalid.

A gold mines , check only the .h files for the very smooth explanation of the API.

Implementation of the Transaction Index which is inherited of all the blockchain index , we can check the transaction by hash , creating the index to be constructed for being used by the other values.

**Chain.h :- to be continued**