École Polytechnique Fédérale de Lausanne

Information leakage in blockchain protocols

by Valentyna Pavliv

Master Thesis

Approved by the Examining Committee:

Prof. Dr. sc. ETH Mathias Payer
Thesis Advisor

Uros Tesic
Thesis Supervisor

EPFL IC IINFCOM HEXHIVE
BC 160 (Bâtiment BC)
Station 14
CH-1015 Lausanne

June 15, 2020

"That is what learning is.
You suddenly understand something you've understood all your life,
but in a new way."
— Doris Lessing

Dedicated to my little brother, Taras Pavliv.

Acknowledgments

I want to thank Mathias Payer to propose this project, which allowed me to deepen so many subjects.

I want to thank Uros Tesic for having accompanied and helped me during this journey.

And of course, I want to thank my family and my friends for their support.

Lausanne, June 15, 2020 Valentyna Pavliv

Abstract

Information leakage in blockchain protocols could be possible if serialization of transactions isn't done carefully. In that case, information on the stack could be sent with the transaction over the network to miners and even put into a block.

Works treating blockchain's security mostly concern theoretical aspects. We focus on the implementation part.

In this project we check if Bitcoin, Ethereum, Ripple, Monero, or Stellar leak computer information. The examination of the code didn't show any serialization problem in transactions.

Contents

Chapter 1

Introduction

The adoption of different blockchain protocols grows exponentially. They are not only a good way to decentralize information for various and varied uses but also known and fashionable since the Bitcoin explosion. Their principle is simple: blocks form a virtual chain by hashing the previous block and storing this information inside. In other words, each block (containing a hash of the previous block) is linked to the previous one. Hence, that forms an oriented chain of permanent information. This schema is shown in figure 1.1. It is permanent for a simple reason: it would require breaking a hash-function. More precisely, if we change something inside a block, the hash of the next one will not correspond anymore. The chain is now damaged and ineffective.

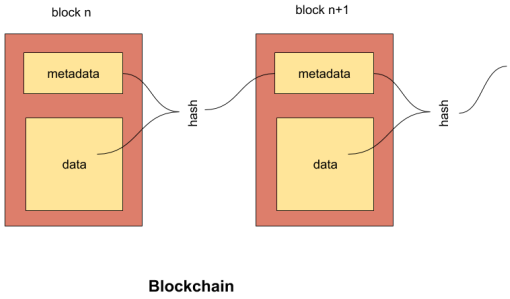

Figure 1.1: schema of a blockchain

This principle has a big advantage: it is decentralized, which means it can be stored over a community. As the storage doesn't depend on a particular place, it doesn't depend on the entity that possesses the disks. Blockchains give people the possibility to store data securely and prove that everything inside is accurate, without a third trusted party (like a bank in the case of cryptocurrencies).

If we look for papers about the blockchain's security problems, there were many searches

about security against enemies union or about side channels possible attacks [19] [20] [21]. Our project is about exploring more low-level security problem: the implementation. Even with a perfect protocol, whose security is proved, a small detail in the implementation could be forgotten. Thus, there would be a security breach. As it is a good time to build some blockchains due to their popularity, a lot of people try to build a new one. This means that not only the professionals write some code, and are not necessarily aware of this problematic. The goal of this project is to find these small mistakes that could lead to information leakage.

During the blockchain protocol, different types of data are transmitted. All these messages could potentially leak information: in the worst case, it could be a transaction that will end up in blockchains. As said previously, the data inside the blockchains are generally public and always permanent. It will be sufficient to download a blockchain to have available the leaked information. But other cases of information leakage are serious as well, as it will be sent over the network.

This is a very possible scenario if the serialization is badly implemented: while preparing a block to send it, some padding is added. This padding could be some information the computer put on the stack to use it previously, like internal information. Thus, it can be used to hack the computer.

We had some criteria for the chosen blockchains: the main one is that that it has to be coded in C/C++. The reason for that is that that is the only very used language where the memory is handled manually, and so serialization can be messed up. We also tried to find blockchains based on how often they are used. While the idea was to check the little-used ones, we saw that they are forked versions from most known blockchains. We thus decided to check the original versions (the very often used blockchains) as well.

The blockchains we had analyzed are the Bitcoin, Ethereum, Monero, Stellar, and Ripple. We looked at their structure and their serialization code. We saw that they use different approaches, that we described as well. Even if each of them had different ways to deal with serialization before sending data to a block, we didn't find any serialization problems.

Chapter 2

Background

Before getting into details and subtleties of each blockchain, it is important to define some global terms and figure out how everything is working.

2.1 Blockchain

Blockchain is a technology to store data. There are many sorts of blockchains. We call them public (or permissionless) if everyone can contribute to write into them. If it is not the case, i.e. the participants are predefined, the blockchain is called private (or permissioned). Similarly, if everyone can read the data, the blockchain is called open, and closed otherwise. This is summarized in table 2.1. In our project, we are interested in open public blockchains, so all blockchains we analyzed are of this type.

Who is able to	read	write
everyone	public (permissionless)	open
predefined group	private (permissioned)	closed

Table 2.1: Different types of blockchains based on their permissions

Blockchains are also used for many applications, as smart contracts, cryptocurrencies, etc. But despite this diversity, blockchains share a lot of similarities that we are going to develop.

Blockchain draws its strength from its construction: it is a distributed ledger. The architecture of such ledger is shown in figure 2.1. The nodes represent users, and we see that there is no central authority. Instead, users are interconnected.

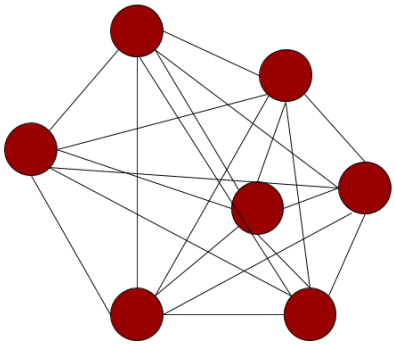

Figure 2.1: Distributed ledger schema

Thus, there is no trusted third party that stores and synchronize data, users do it. To construct such a secure and trustworthy system, some rules have to be established. We will start by explaining how the blockchains are built up, and how this allows us to manage data securely.

A blockchain is a chain of blocks. Every block is made from metadata and data. Data can be everything, but often it is made of something we refer to as transactions. Transactions are interactions between users (peers): it can be "A sends this amount to B" in the case of cryptocurrencies, or a contract in the case of smart contracts. A transaction is thus often made of:

- Input: The identifier of who did/signed the transaction
- Output: The identifier of the receiver of the transaction
- Other possible components: the amount (in case of a cryptocurrency), etc

Afterward, transactions are put into a block, and this block is added to the blockchain. This is illustrated in the figure 2.2.

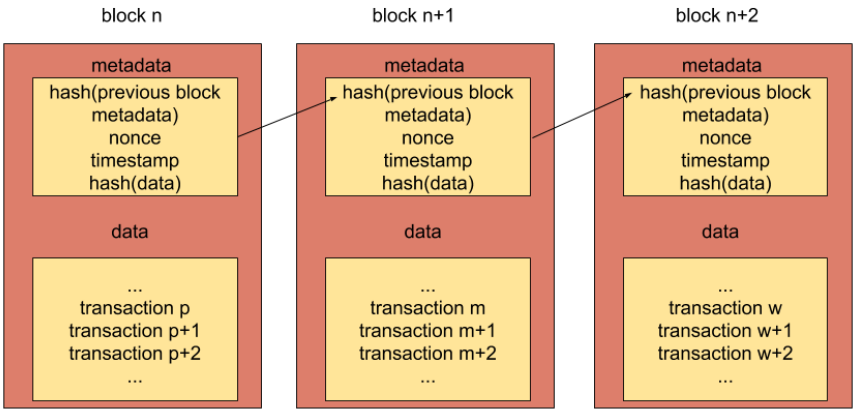

Figure 2.2: Basic construction of a blockchain

Metadata is the other component of a block and it is the one that chain blocks together. It is done with the help of hashes: each time a new block is added to the blockchain, we can get the hash of the previous block's metadata. It not only allows us to have an order in the blockchain but also it permits to seal the information: every time someone tries to change data (or metadata) in one of the previous blocks, the hashes of the following block will not correspond anymore. Metadata is also done of some other components, so we can usually find in it:

- hash(previous block's metadata)
- nonce: it is a random number, different in each block, to randomize the hash of the metadata
- timestamp: the time when the block was added to the blockchain
- hash(data): in order to facilitate the computing of the hash of the next block, we can already compute the hash of the data and put it in the metadata. This way, we secure the data from any unwanted changes.

There is one last rule to set up in the blockchain construction: who of all users can put the next block into the blockchain. This problem is rather hard due to the architecture of distributed ledger. As the peers are randomly connected between themselves, they don't have access to the same pending transactions. If everyone could randomly add a new block, each block would be different by the order of transactions or transactions themselves. This also would allow malicious users to add transactions that spend the same cryptocurrency (double spending problem). To counter this, we put in place a consensus model: it's a set of rules that permits us to decide who will add the next block. Each blockchain has its consensus model, or combine some already existing. The two most used consensus models are Proof of Work (PoW) and Proof of Stake (PoS).

Using the PoW model, everyone can add the next block, but at one condition: he has to solve a puzzle. It is a difficult and time-consuming exercise, with a lot of competition. For each block, the puzzle is independent of the previous one, so the chances to solve it are equal for each new block (for the same computational power). Those who are trying to solve these puzzles are called miners. The miner who finds an answer to the puzzle shows it to the others. Afterward, the provided solution is verified (which is really easy) and the block is added to the blockchain. To motivate the miners to waste energy and find the answer, each time they solve the puzzle they got rewarded.

On the other hand, the PoS model doesn't need a lot of energy either a lot of time. The idea is that more you have money for a cryptocurrency using this consensus model, more you will want it to work properly. Thus, richer a user is, more authority he has to decide if a block deserves to be added to a blockchain. In practice, more coins you own, more you are allowed to validate a new block. Thus, users who help to add blocks are called validators. To add a new block, validators either propose a new block or attest existing ones if they are valid. If a block reaches some number of attestations, it is added to the blockchain.

Cryptocurrencies using this consensus model also put some rules in order to punish bad behavior. Before becoming a validator, a user has to stake a certain amount of coins on the network: for misbehavior, this amount is reduced. On the contrary, good behavior is rewarded: by helping validate blocks or by the time they are available. [1][2]

It should still happen that two blocks are added at the same time. We call this situation a conflict: we have now two versions of the same blockchain. But this is generally solved by the longer blockchain principle. As soon as a miner adds a new block to one of the versions, all other versions would be shorter and thus discarded. [23]

2.2 Stack

A stack in the programming language is a data structure: the only two things you can do are put something on the top of the stack, or to take the last objects (that are thus on the top of the stack). Computer's stack memory is used for stock variables that don't require names and are used only once. We can put on it addresses (where the program wants to go after computing) or local variables (specific to a function but not used outside it). The most important condition of the stack is to be quick: the values on it will be reused really soon.

This particularity force a certain implementation. In order to keep track of where is the top of the stack, we stock this place, by his address, apart, in a register. This register is called the stack pointer (SP). It's not the only information about the stack that is stored: there are other registers, like frame pointer, etc.

The fact is, it is slow to always delete information put on the stack, and not necessary. As we

keep track of what we need to read and where (with the registers), we only overwrite values that are in place.

A stack is thus a block of memory, where each line has an address and the same size. If an object too big is put on the stack, it will be cut and fill many lines. If the object is on the contrary too small to fill one line, just the first values of the corresponding line are overwritten, with some padding the rest of the line. We can see in figure 2.3 a schema of the stack with padding after an example structure is put on it. The stack pointer moved to the top of the stack (just after the structure), and there is some padding done inside the storage of the structure. The reason why the computer works with padding (and not putting everything one after another) is about the speed execution. It deals with memory faster when everything is aligned. The alignment rule is that a variable put on the stack can exceed on other lines only if it is also started at the beginning of a line. Hence, the padding is unavoidable.

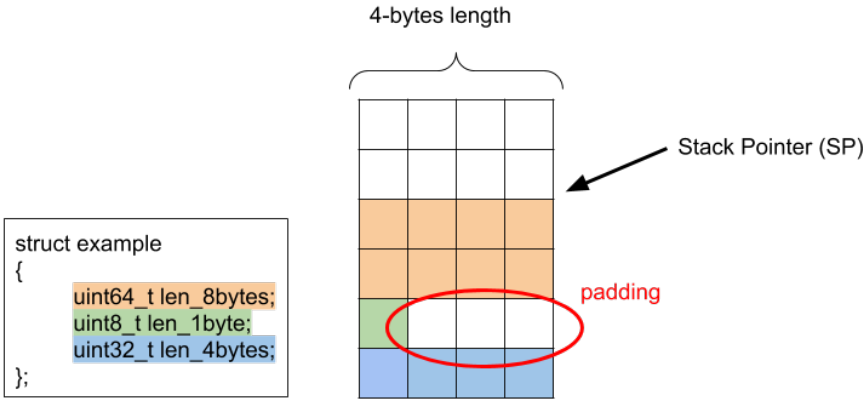

Figure 2.3: Stack schema after putting a struct on it

This is done by default, and C/C++ languages don't allow to change the order of the data put on the stack in order to fit the best the lines of the stack.

The problem of the padding is that it can be anything that the computer put there before. It is always undefined. For that reason, it is for example impossible to compare structures by memcmp - it will also compare the padding.

On the other hand, the packed structure's mode (data without padding, everything is put one after another) can be set in C/C++ languages. This is used in mainly two cases: when we want to save memory or when we want to send data in the network. [3]

2.3 Serialization

Serialization is a way to store and/or send data over a network. In our project, we are interested in the last one. The reason why we don't send data structures as it is stored in memory is that there are different types of architecture in a computer, and thus different ways of storage. To deal with this problem, a common protocol has to be put in place. This protocol requires the data to be sent in packed mode (serialized), with a predefined sens.

The order of bytes in the memory is called the endianness. There is two sorts of systems:

- a big endian system: the most significant byte is stored at the lowest address (thus the least significant byte is stored at the higher address)
- little endian system: the most significant byte is stored on the higher address (thus the least significant byte is stored at the lowest address)

This is showed in figure 2.4.

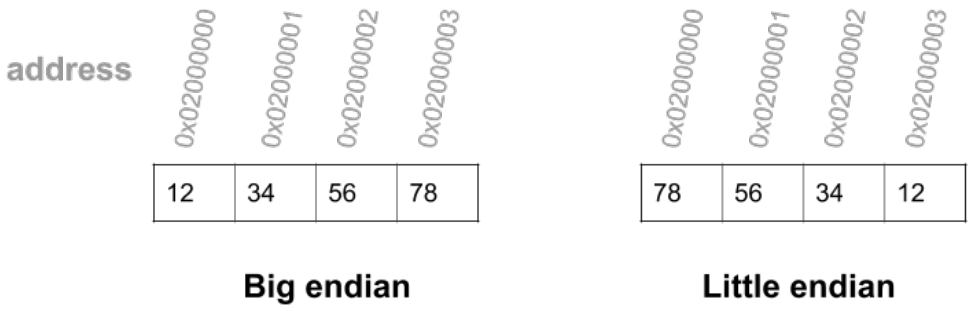

Figure 2.4: Endianness demonstrated with storage of 0x12345678

It is important to notice that only on data bigger than one byte we can see the endianness. If we want to store "abcd", where each letter is one byte, it would be stored the same way on both systems.

The order mostly used over the internet is the big endian, also called the network order. [4]

Chapter 3

Cryptocurrencies

In this project, different blockchain protocols were examined, under some criteria. The global criterion is that a blockchain has to be written in C/C++ languages. The reason for this is that with this language a bad serialization is possible. When the transaction is sent into the network in order to be added into a blockchain, if the programmer forgets to serialize the data, the transaction will contain the padding from the stack as well.

Another criterion was based on the popularity of the blockchain: more a blockchain is known, more any serialization problem would be impacting the users. In addition to that, known blockchains are often forked to create new ones, so the impact would also spread on those. On the other hand, more a blockchain is popular, less it is probable that it has this implementation error. So it is important to check blockchains that are on both ends.

Based on those criteria, we examined and present you the following blockchains. For each cryptocurrency, there is a more precise explanation of why it is pertinent to our project.

3.1 Bitcoin

Bitcoin (BTC) is probably the most known and popular cryptocurrency, making it the most used blockchain. Bitcoins transactions are monetary transactions: if you send bitcoins to someone, the transaction will be written in a block. To be more precise, you have a wallet (that has an address) with a number of bitcoins. At each transaction, you have to send all the bitcoins you have inside your wallet. If you want to spend only a part of the total amount, you have to send the rest to your wallet address. For example, if Alice has 5 bitcoins and wants to send 3 bitcoins to Bob, she will create two transactions:

- Alice sends 3 bitcoins to Bob

- Alice sends 2 bitcoins to Alice

This way, all bitcoins moved from Alice's wallet, but she got the unspent coins back.

The way a transaction is written inside a block is simple: there are 3 parts for each transaction that are written in a block. These parts are

- the header - containing some metadata, like a hash of the whole transaction to check its integrity
- the inputs - containing information about input, like your bitcoin amount
- the outputs - the value you will spend in this transaction

Once the transaction is done and signed, it has to wait to be added into a block: this is done by miners (those who add new blocks at the end of the blockchain). Miners have to solve a mathematically complicated problem called proof of work. More precisely, their computer has to test a really big number of values to find the one that solves the problem. This hard computation implies that only times to times someone finds an answer. If a miner solves the problem, he can add pending and verified transactions in a block and put the block at the end of the blockchain. To motivate miners to use their computer for this purpose, they are paid with bitcoins. [23]

This whole system is designed to provide a secure payment system: it is for example impossible to double-spend money. To understand why, let's imagine a scenario where a malicious user wants to double-spend the same bitcoin. Hence, he will create two transactions with it: these transactions are sent in a place of pending transactions (called pool). Once a block is created, if the miner verified both of the transactions, one of them is automatically discarded: the double-spending try failed. If a transaction is verified after the other one (already being in the blockchain), it will be discarded as well. But another possibility is that two miners add blocks simultaneously, each containing one of these transactions. In this case, the coin is double spent until one of the versions of the blockchain eliminated, by the longer blockchain principle. Thus, one of the double-spending transaction will be discarded with the shorter blockchain. This system makes double-spending impossible. To be sure that no one will try to double-spend money, it is important to wait that the transaction is some blocks behind the last block.

On the other hand, the way Bitcoin deals with transactions, it took extremely long to confirm a transaction (10 minutes). It is computationally considered impossible to compute 6 proof of work. So, it is recommended to wait that at least 6 blocks are added to the block with the transaction you are interested in, before considering that this transaction is valid. This is one of the biggest disadvantages of Bitcoin: to have a confirmed transaction a user has to wait around one hour.

The main reason we selected Bitcoin is that it is the most used and popular cryptocurrency. This implies that a big part of smaller blockchains are forked from the Bitcoin code (they use

bitcoin code as a basis). So, verifying the Bitcoin code will have an impact on a whole family of blockchains (like FreiCoin, CureCoin, AuroraCoin, etc.).

3.2 Ethereum

The creator of Ethereum (ETH), Vitalik Buterin, wanted to create a more performing blockchain that can be used as a cryptocurrency but also has other functionalities. So, Ethereum blockchain is widely used as money, but for smart contracts as well. Smart contracts is a way to follow contract terms automatically, without any trusted intermediate or judge.

The way Ethereum is built has some differences with Bitcoin. It has three types of transactions: one type for the cryptocurrency transaction and two other for smart contracts. The values put in one transaction are Similar to Bitcoin (i.e. the address of the receiver, a signature, etc). The transaction is checked, the amount is verified as well as the signature. If all checks pass the transaction waits that miners put it in a new block (with the same consensus as Bitcoin, even if Ethereum is switching to Proof of Stake).

Ethereum was selected not only because it satisfies our prerequisites, but also because it is a really complex coin (due to smart contracts). So, Ethereum makes transaction serialization at many levels, which is interesting to check. Also, it is one of the most used blockchains as well, so if something is wrong with it the impact would be huge.

3.3 Ripple

Ripple (XRP) is a cryptocurrency. While it is less known as Ethereum or Bitcoin, it is still used, in particular by banks, that consider it more advantageous. The reasons are of course that it is secure and transparent, but also that the banks don't have to pay any fee for the transactions.

As for other cryptocurrencies, Ripple uses the same idea of distributed ledger, but it has also a main difference with other cryptocurrencies: Ripple doesn't maintain a blockchain. Instead, it has a consensus model for the right work of the cryptocurrency, called the Ripple Protocol consensus algorithm. The protocol consists of a voting system. More exactly, to avoid any sort of problem, every server should vote if a transaction is valid. It is enough that only one considers that it is not the case for not counting the transaction as valid. This system has a big advantage over Bitcoin: the transaction took only a few seconds (3 to 5 seconds) to be validated.

Another difference with Bitcoin is that this cryptocurrency wants its user to register with their true identity: there is no anonymity, a user has to be registered. But their identity is not written on the transaction, so the privacy is preserved.

Even if Ripple doesn't maintain a blockchain, it still sends transactions over the network. So, as it presents a lot of similarities with other blockchains and fulfills our prerequisites, it is selected in our project. [5] [6]

3.4 Stellar

Stellar (XLM) is the biggest competitor of Ripple, in the sense that it promotes the same type of protocol. It allows us to make transactions over different currencies, with an open-source code. The difference with Ripple is that Stellar maintains a blockchain. Stellar also has known some problems with the protocol of the consensus algorithm, giving it a bad reputation.

We wanted to check this blockchain because it reunites all conditions we want for our project: it is barely known, a C++ written cryptocurrency. It is thus more likely that it has serialization problems than for previously seen blockchains.

3.5 Monero

Monero (XMR) is a cryptocurrency forked from Bytecoin. It is often confusing, because there is two cryptocurrencies called Bytecoin: Bytecoin(BTE) forked from Bitcoin, and Bytecoin(BCN) that has nothing to do with Bitcoin. Monero is a fork from the last one (BCN). It is known to be more evolved than Bytecoin, with an active community: the code was completely rewritten and fixed many issues Bytecoin has. It is very interesting to analyze Monero: apart from the fact it is written in C++, making possible serialization error, Monero really focused on privacy.

While Bitcoin transactions are publicly available, Monero makes it computationally hard to see who paid which amount and what amount is received. Monero makes it computationally so hard to connect any transaction with a wallet address that it is considered unbreakable with the actual power we can put in it.

Monero uses three ticks to ensure privacy. First of all, each time a transaction is made the receiver creates a single-use address. This way, we could never associate beneficiary addresses between them along with the blockchain. To hide the sender's address, it is a little bit more complicated: Monero uses a cryptographic primitive called a ring signature. Ring signature enables sign a transaction, but this signature could belong to anyone from a ring of different addresses. It is thus impossible to know who of them send the transaction. To top it all, Monero also splits a transaction into smaller ones, which significantly increase the difficulty to find out the final amount that was sent.

To add the transaction into the blockchain, it has to be first checked by miners. As the input address is anonymous, to prevent double-spending, another cryptographic mechanism is used:

Monero uses key images. A unique key image is added to each transaction. This allows miners to check if a new transaction with a key image was already used and written into the blockchain. If it is the case, the transaction is discarded because it means that it was already spent in another transaction. After the transactions are checked, the miners use the proof of work consensus to build a new block and add it to the blockchain. [7] [8] [9]

Chapter 4

Results

We will inspect in this section how serialization is done in different blockchains. The problem could be in risky casts, unions etc.

4.1 Bitcoin

Bitcoin [10] organizes the code by grouping files by their thematic: we have a file for the consensus, a file for the crypto, a file for the wallet, etc. There are also some files external to all the groups. One of these external files is the serialize header file.

First of all, the serialize file includes the endian file. It is used to build a wrapper for integer type in order to have compatible serializers.

The serialize file also implemented compile-time polymorphism. This means that the serialization function takes a type as argument (it is thus called a template function) and based on it serialize correctly. So, each type has a dedicated serialization function, but it is managed automatically. When the serialize file is called, there are no casts of an entire object: each field is accessed separately and serialized individually.

The transaction is built inside the wallet file. When it is created, each field of the transaction is put in a stream, so there is also no padding in a sent transaction.

4.2 Ethereum

Ethereum [11] grouped its files by their thematic. When looking in the p2p (peer to peer) package, there are two types of sent data. Packets are either an enum (like the P2pPacket), which is thus

secure, either it is sent with RLP format.

Ethereum provides its own serialization method, called RLP. It is used to encode a list of items. An item itself is made either of a list of items or of a byte array. Each item has its own rule of encoding, you can find here [12].

RLP takes into account the endianness: it is using big-endian. With this format, all padding is removed.

The RLP code is stored in the libdevcore package. We are interested in the RLPStream class because it is what is sent over the network. This class contains different appending to the stream - one per each type. Compile-time polymorphism is used for sequences of data. If we look at functions taking a predefined type (lists, vectors, pairs, char, int, etc), the serialization is done correctly. The most important function in this class for our project is the appendRaw function - when it is called, it adds all the data regardless of what it is to the stream. So, the padding could be added as well: all places where this function is used have to be checked. Hence, appendRaw is called with moderation and doesn't represent any signs of misusing.

More precisely about the transaction: they are created in the libethereum package, in the Transaction file. There is different types of transaction: those concerning the cryptocurrencies, and those concerning smart contracts. But they are treated the same way for the serialization: they are put into a transaction queue, which is then sent in order to be added in a block.

The transaction queue is a class defined in the libethereum package. It handles the pending transactions (that the user wants to add to the blockchain). The methods of this class concern the actions that could be done over the queue: it can add new transactions (if it verified), or get transactions that are already in it. This file thus also categorize transactions by their priority or if they are verified. All transactions in the queue are already encoded with the RLP method, so there is no serialization problem with sent transactions.

4.3 Ripple

Ripple [13] takes care of the organization: everything is grouped by the thematic (for example there are crypto, ledger, or peerfinder files). In each file, there is either another group of files (so it is organized by the sub-thematic), either only header files and a package containing their implementation.

The protocol file contains a serializer implementation. Just like with Bitcoin, Ripple uses compile-time polymorphism: it defines add functions over all types. Also, there is an add raw method, but it is used conscientiously, without additional padding. The data will be added into a blob (binary large object), which will be sent over the network. The implementation treats each type carefully, adding only the length of the data, so there is no padding problem with this file.

The protocol package concern the messages that are sent between peers. As transactions are built somewhere else and don't constitute a message by themselves, we have to check how they are serialized as well.

The transaction class can be found in the misc package, from the app file. This class shows how a transaction is treated: for example, the status it has, or setter and getter methods over different fields of the transaction. Before sending a transaction, this class converts it into a JSON file. Hence, the transaction doesn't contain any serialization problem neither.

4.4 Stellar

Stellar [14] organization is less clear than for other cryptocurrencies, despite it is again grouped in different packages. It is the only blockchain that doesn't have any serialize file.

In order to establish a connection between peers, Stellar uses a TCP connection to send messages. There are different types of messages: like the hello message, or an authentication one. A message is defined in the xdr package as a union of different message types, where a message type is an enum of all the different types.

In the peer file, the message sending and receiving is implemented, with the help of virtual functions. Stellar use polymorphism in order to deal efficiently with different situations. To send a message in a communication (different from a hello or an error message), an authenticated message is created. Information concerning the TCP connection is added (like the sequence and the mac) and sent with the metrics corresponding to the message. In the same way, if the message is a Hello or Error message, only adding additional TCP information is avoided. The metrics are stored with the Medida library [15] and sent with JSON format. There is no padding sent with the messages.

The transactions are sent via a Transaction Queue, defined in the herder package. This class deals with pending transactions. If a new transaction is done, Stellar tries to add it to the queue (with the try add method). Some checks are done before adding it, for example, if the transaction is a duplicate one. If a transaction is not valid, only its status is returned. A status of a transaction is defined as an enum, and can be

- pending: for valid transactions
- duplicate, error, try again later: not valid transactions

If a transaction is valid, its pending status is added to the queue, along with information concerning this transaction. It is serialized correctly.

4.5 Monero

Monero [16] organizes its files into packages, with really clear thematic per package. The p2p package is about communication between peers. It is used to look up for the peers and build a network from online (called white in the project) peers. The information that needs to be sent in this package is put into a stream, called Archive.

The Archive is then serialized, using the serialization file from the serialization package. Monero uses the Boost library [17] for the serialization. This tool is used for network communication, if we put serialized object into an Archive, we can recover it with the same type. In this file, polymorphism is used to serialize basic types in an easier way: the type is checked with boost methods. These provided methods [18], like is integral, verifies if a type can store the asked type (if it can store an integral in our example). The serialize function is then adapted to the type. There is no padding problem with this implementation.

As for Bitcoin, transactions are processed and sent in the wallet file. In fact, Monero defines two wallet files: wallet2 and simple wallet. Wallet2 implements the core functionality, used by simple wallet.

In the wallet2 file, the transfer main method manage pending transactions: this is where all the checks concerning a transaction are made. For instance, one of the tests that the transaction has to pass is that there is more that one person in the ring, for the right work of the ring signature. If all parameters concerning the transaction are valid, the transaction is sent. A pending transaction is defined as a structure. To send it, it is wrapped in a JSON object. There is no padding with sent transactions.

Chapter 5

Related Work

The security of a blockchain is almost always focused on theoretical aspects of the blockchain. As this article [21] resumes, the main challenges are:

- Majority attack: if a group of persons get 51% of mining power, they could use it at their advantage. For example, they could decide to accept double spending.
- Forks: if a blockchain updates, not everyone would get the new version immediately. This opens a security breach if the rules are not compatible.
- Size of the blockchain: longer is the blockchain, more it would take time to verify if a transaction is valid based on those that are already in the blockchain.

The system security is not taken into account: there is no papers about the implementations of different blockchains.

On the other hand, information leaks from the stack are not new: we got a similar work proposed in the context of Linux kernels [22]. They searched for problems due to

- Not initializing memory, which implies padding done with information that was used previously (put on the stack)
- Bad bounding when reading data, which implies reading beyond the buffer

They did found and patched some problems.

We can notice how comparable their work is: they were searching for similar vulnerabilities. This problem is thus present, but not explored in the case of the blockchains. Hence, our project aims to develop this part.

Chapter 6

Conclusion

Blockchains are used over many domains and became very popular since the Bitcoin apparition in the different media. Although they are tried by many, their principle is often unknown. The trust comes from the fact that searches were done about their algorithms: they are admitted as safe. Our project aims to check the implementation part: the messages sent over the network may contain padding, due to bad serialization.

Our project was designed to test if there is information leakage in the following blockchains: Bitcoin, Ripple, Monero, Curecoin, and Stellar.

The criteria for those blockchains were

- Implemented in C/C++: the information leakage we are looking for is depended on how the language treats memory and, more precisely, how it deals with the stack. With C/C++, errors like bad casts imply unsafe padding.
- Original code: many blockchains are forked from already existing ones. We can think about DigitalCoin, Freecoin, Curecoin, Tierion, Stratis (and a lot more) that are done on Bitcoin's base. Their disparity often consists of different consensus model: the serialization code is left unchanged. In order to have a maximal impact, we check each family only by the original implementation.

It is clear that all low-value coins are forks (often from bitcoin). So, in our project, the chosen blockchains are rather popular and used ones. Programmers that wrote them were aware of the information leakage problematic. The different solutions are provided: some of them use specialized libraries (like Boost in the case of Monero). Otherwise, appropriate algorithms are set up: we can think of Ethereum and the RLP solution. There are no stack leaks in the coins we have checked.

Bibliography

[1] https://consensys.net/blog/blockchain-explained/what-is-proof-of-stake/.

[2] https://academy.binance.com/fr/blockchain/proof-of-stake-explained.

[3] https://www.nccgroup.com/us/about-us/newsroom-and-events/blog/2019/october/padding-the-struct-how-a-compiler-optimization-can-disclose-stack-memory/.

[4] https://searchnetworking.techtarget.com/definition/big-endian-and-little-endian.

[5] https://cointelegraph.com/ripple-101/what-is-ripple.

[6] https://www.digitaltrends.com/computing/what-is-ripple/.

[7] https://www.investopedia.com/terms/m/monero.asp.

[8] https://monero.stackexchange.com/questions/2158/what-is-moneros-mechanism-for-defending-against-a-double-spend-attack.

[9] https://blockgeeks.com/guides/monero/.

[10] https://github.com/bitcoin/bitcoin. (accessed: 10.03.2020).

[11] https://github.com/ethereum. (accessed: 27.02.2020).

[12] https://eth.wiki/en/fundamentals/rlp.

[13] https://github.com/ripple/rippled. (accessed: 03.03.2020).

[14] https://github.com/stellar/stellar-core. (accessed: 05.05.2020).

[15] http://dln.github.io/medida/.

[16] https://github.com/monero-project/monero. (accessed: 18.05.2020).

[17] https://theboostcpplibraries.com/.

[18] https://theboostcpplibraries.com/boost.typetraits#ex.typetraits_01.

[19] Rune Tevasvold Aune, Adam Krellenstein, Maureen O'Hara, and Ouziel Slama. "Footprints on a Blockchain: Trading and Information Leakage in Distributed Ledgers". In: *The Journal of Trading* 12.3 (2017), pp. 5–13. ISSN: 1559-3967. DOI: 10.3905/jot.2017.12.3.005. eprint: https://jot.pm-research.com/content/12/3/5.pdf. URL: https://jot.pm-research.com/content/12/3/5.

[20] Xiaoqi Li, Peng Jiang, Ting Chen, Xiapu Luo, and Qiaoyan Wen. “A survey on the security of blockchain systems”. In: *Future Generation Computer Systems* 107 (2020), pp. 841–853. ISSN: 0167-739X. DOI: https://doi.org/10.1016/j.future.2017.08.020. URL: http://www.sciencedirect.com/science/article/pii/S0167739X17318332.

[21] Iuon-Chang Lin and Tzu-Chun Liao. “A survey of blockchain security issues and challenges.” In: *IJ Network Security* 19.5 (2017), pp. 653–659.

[22] Salva Peiró, M Muñoz, Miguel Masmano, and Alfons Crespo. “Detecting stack based kernel information leaks”. In: *International Joint Conference SOCO’14-CISIS’14-ICEUTE’14*. Springer. 2014, pp. 321–331.

[23] Dylan Yaga, Peter Mell, Nik Roby, and Karen Scarfone. “Blockchain technology overview”. In: *arXiv preprint arXiv:1906.11078* (2019).