UNIVERSIDAD CARLOS III DE MADRID

Master in Cybersecurity
Universidad Carlos III de Madrid

Advanced persistent threats and information leakage

Information Leakage Stego tools

100363772: Mario Palomares Gallego
100363615: Miguel Peidro Paredes
100363603: Sergio Bernardez Molina
100363695: David Silveira Amaro

April 2021

INDEX

1. Introduction

In the following document we are going to explain the steganography tool that has been developed. Its functioning mainly relies on blockchain technology, specifically on the Bitcoin implementation. The idea is to exfiltrate information hidden in transactions. Taking advantage of a bitcoin-like ledger (more on this later) as a steganographic cover channel provides many advantages, such as an incredible load of traffic, and high availability, hence hardening the detection and improving the availability.

Additionally, we make use of a public service like “Pastebin” where the users upload small texts. Typically, it is used as a workaround on IRC channels max entry limit. In our case the usage of this service is to allow the steganography tool to generate a common secret between the victim and the server. This shared secret will allow the use of symmetric cryptography to encrypt the message before the exfiltration process. As we will see later, this encryption is justified and does not break any steganographic principle.

The rest of the document is divided as follows. Section 2 gives some technical background on the different means to insert data into the blockchain and some ideas that have been proposed that apply steganography on blockchain. The structure of this section is highly inspired from sections of other papers (Omid Torki, 2021) (Giron & Martina, 2020), but although structure is similar, explanations are self-made from bottom-up. Section 3 explains the implementation of our steganographic tool highlighting the improvements it makes over the existing ones. Section 4 analyses the performance based on: robustness, capacity and stealthiness. And Section 5 makes a reasoning on how the functioning of this tool could be detected.

2. Background

2.1. Insertion techniques

One may imagine that inserting information inside a blockchain like bitcoin isn't possible mainly because looking at the transaction object it does not have many attributes and all of them are highly constrained.

The reality is that the most important attributes are the ones that indeed can be used to insert data into the blockchain. These attributes are in the inputs and outputs of a transaction (Bitcoin, 2021), specifically the addresses and the signatures. Let’s make a brief explanation on what this means.

As a starting point it must be clarified that an address on any cryptocurrency is the minimum unit for making transactions; analogous to a credit card, however, users can aggregate multiple addresses under a single name, this is called a wallet and is analogous to a bank account. Each address is a hash (more on this later) of a public key from a previously created key pair using ECDSA algorithm. So, as it can be seen Bitcoin makes use of asymmetric cryptography and the reason for hashing the public key is to add a layer of security on the users keys. As one may guess, preserving both keys is fundamental to perform transactions.

Unlike bank accounts, on all bitcoin-like cryptocurrencies there is no one database containing the balance of each wallet/address/user (note that cryptocurrencies like Etherium implement this) instead coin ownership is passed from one address to another. The next owner address is referenced in the output of a transaction and the previous owner address is specified in the input. In order for the new owner to use these new coins he will have to reference the output of this previous transaction on the input of the new transaction, proving at the same its identity in order to unlock these coins (this is done by signatures). As we can see inputs reference outputs of previous transactions hence making a chain. Note though this is not the explanation of the name blockchain, it just an implementation coincidence. Taking this into account we can imagine that as time goes by a whole tree with ramifications will be created due to the change of ownership of the bitcoins. The leaves of this tree are called the UTXO, and the name is self-explanatory: Unspent Transaction Outputs (Bitcoin, 2020).

Summarizing, outputs specify the destination address of the transaction (**address**), the value (**value**) to transfer and also the unlocking condition (**ScriptPubKey**). Usually, the unlocking condition is proving by a signature the possession of the address to which the coins are being sent, this scheme is known as Pay-to-pubkey-hash (P2PKH). However more complex schemes are possible through the Pay-to-script-hash (P2SH), this adds a **Script** field which contains the unlocking logic.

On the other hand, inputs reference the output of the previous transactions whose coins are going to be used and add a proof of the ownership (**ScriptSig**) to actually unlock and use those coins.

Having this in mind we can move forward and explain the different techniques of inserting data on Bitcoin blockchain.

- *Coinbase Transaction*: This is the name the first transaction each block contains (Bitcoin, 2020), and these transactions are set by miners to collect the transaction fees and the block mining reward. Since block mining rewards are actually new bitcoins and fees are implicitly of the miner's ownership, **ScriptSig** field is empty and this leaves room for 100 bytes. Though appealing for making use of this method one must be a miner and what is worse, successfully mine a block.
- *OP_RETURN*: This is the de-facto way of inserting arbitrary data in the blockchain (Bitcoin, 2020). The **Script** field contains the logic for unlocking a transaction (omitted in the case of P2PKH since it is always the same) and it is programmed on a Fortran-like stack-based programming language. Basically, the logic is made up of op-codes and values. OP_RETURN is just an op-code designed to mark a transaction as unspendable, hence being used to insert information. Even though this is the perfect method for inserting data in the blockchain is has nothing to do with steganography because the text is clearly visible.
- Output **address**: Since there is no way to check whether an address is being used or not, one can insert arbitrary data on this field. As it will be seen later this is not a trivial task and a knowledge on how bitcoin addresses are created is needed. An important remark here is that the use of the input address is not an option since the ownership of its associated private key is required to unlock the UTXO destined to that address.
- Non-standard transactions: One interesting field for inserting information could be the **Script** one, since P2SH enable bitcoin users to specify a complex logic to lock and unlock the bitcoins of a transaction one may think this is an opportunity for inserting

random data, but actually if the P2SH **script** is not among a set of proposed standard scripts (e.g. multisign) the transaction is considered as non-standard and many miners will refuse them.

2.2. Previous Proposals

While only in theory, some proposals have come up with the idea of using Bitcoin infrastructure to exfiltrate information. These ideas mainly use the output address field explained before. Following, we are going to explain the methodologies proposed by Xu et al. (Xu, Wu, Feng, & Zhang, 2019), Zhang et al. (Zhang, Wang, Zhang, & Waqas, 2020), Partala et al. (Partala, 2018) which are aligned with the proposal of this work. Other proposals more complex based on mathematical rationale such as Alsalami (Alsalami & Zhang, 2018) are not covered.

In the Partala scheme codification of the information is done by applying the LSB technique on the destination address of the transactions. To guarantee the order of the sent information only one transaction per block is broadcasted. Additionally, the receiver needs to know the address of the sender beforehand to detect the presence of a message in the network. As it can be seen, the capacity in this method is extremely low.

Zhang's scheme follows the same idea of using the destination address but in a completely different way. In this case several bytes of the destination address are used to exfiltrate information. Exfiltrating information this way is not a trivial task and the reason for it is the way in which an address is constructed (Bitcoin, 2021). We will discuss this issue in depth in the next section. To make a valid address one cannot just input arbitrary data to the address generator algorithm and hope the network to accept it, actually inputting the data at step 1 will make it unrecoverable since hashing functions are used, instead, additional software must be used. They made use of the vanitygen software (Bitcoin, 2020) a tool which enables you to specify a fixed n byte prefix the address must have, and by **brute forcing** the software finds the corresponding public and private key that satisfy this.

Taking this into account the tool would encrypt the information and later on codify it in base58 hence making the fixed prefix homogeneous with the rest of the address. After that it will generate all addresses necessary to send the whole message. To send all transactions at once an index would be generated, encrypted and sent into an OP_RETURN command signaling at the same time the start of a communication. The main problem with this algorithm is adding encrypted information on OP_RETURN which is always suspicious. Nonetheless this is the most similar proposal.

Xu scheme from the base is completely different from the previous one because the sender plays the role of a miner. The exfiltration information is done by the permutations of the transactions included in a block. This idea faces a lot of challenges in a cryptocurrency like bitcoin such as the mere achievement of mining a block which is almost impossible with such powerful mining pools existing already.

3. Implementation

3.1. Infrastructure

As it has been seen in the previous section all data insertion methods require making transactions which later get published hence making the information visible to the other endpoint. With exception of the OP_RETURN method the rest imply the burn of coins, this is the reason why using the mainnet is not an option, instead testnets comes as the way to go.

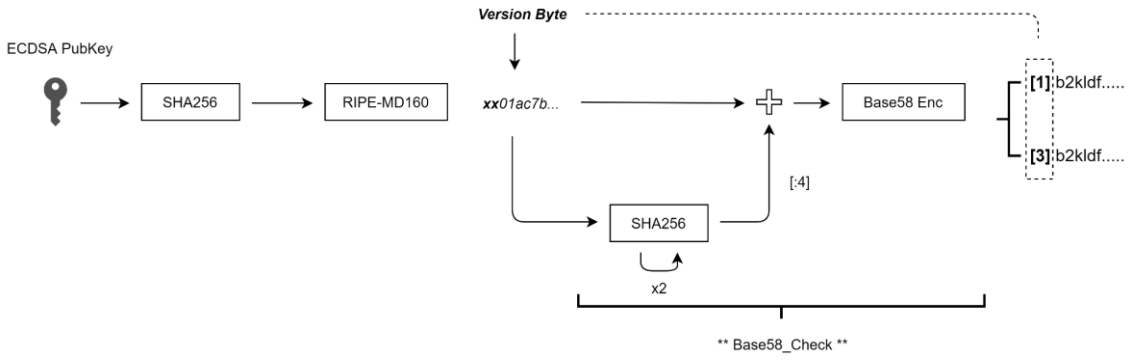
Independently of the network used, the main problem is to have access to make transactions on these networks, and option would be to make a full stack software which implements the bitcoin P2P protocol but that was too time consuming for a proof of concept. Instead, a much easier approach is to use an API service. In our case we make use of BlockCypher (BlockCypher, n.d.) which is a service that offers an API for making transactions over a wide range of cryptocurrencies: bitcoin, etherium, dogecoin… The most important part is that among these networks they included testing networks such as the Testnet3 (Bitcoin) and the BlockCypher one (self-proprietary). The key point is that it´s own testing network has the same implementation as the Bitcoin one with the only difference that blocks are published every minute instead of every 10 minutes, which actually is an advantage for the developing phase. This is the reason why we decided to use that network. The only inconvenience is that the API is limited to 200 requests per hour meaning that large messages cannot be tried but still for a proof of concept is more than enough.

An extremely important service the API offers is a WebSocket communication (2 way **connection oriented** communication channel) which enables the subscription to events. Subscribing to the “new-block” event makes it easy to be synchronized with the blockchain and is fundamental for the receiving endpoint.

3.2. Algorithm

On the proposed scheme information is going to be exfiltrated using the destination address of transactions. To avoid statistical analysis on the address that could detect the presence of non-random data (note that usually this value is the result of an encoding of a hash), information is ciphered before being sent. At this point we have the main novelty contribution this work proposes, and it concerns the election of the type of address.

On bitcoin historically there has only been one type of address. That algorithm to create these addresses followed these steps:



The critical point is the use of the base58 encoding (Nakamoto, 2019) on the last step. This codification is not as simple as a base64 one, instead its algorithm is based on mathematical operations which makes the encoded string output length incalculable. Bitcoin addresses accept a range from 26-35 bytes on these addresses but this variability must not be confused with the one produced by base58, this variability is due to the fact that starting 0 values are not encoded by base58 and bitcoin tolerates up to 9 consecutive zeros at the input of the base58 encoding. Anyway, the base58 variability means that making a valid base58 address by injecting arbitrary data to the input of this encoder content is almost impossible. By the way, how the output of HASH160, guarantees all these restrictions seems to be magic.

As it can be seen, using this type of address isn´t an option if you want to use the full length of it. Using only a prefix is possible as we explained previously in the Zhang (Zhang, Wang, Zhang, & Waqas, 2020) proposal, but you are limited to a short prefix since address is obtained by bruteforce.

Luckily since 2015 Bitcoin added a new type of address called the segwit address (Bitcoin, 2019). As its name hints it was developed to support a new feature called Segregated Witness (Bitcoin, 2021). This feature was developed to address two problems: transaction malleability (Klitzke, 2017) and low block size limit this last one impacting on transaction rates. The main takeaway from this address is that its generation algorithm changes significantly from the one presented before. This change added two important features: firstly, instead of having a bare integrity checksum, the new one would have **error detection codes** (Bitcoin, 2019), and secondly, key to follow our implementation, the encoding algorithm was changed to another one called bech32 which produced a **fixed m output** for each n length input. Talking about sizes, the input for the bech32 encoder must be 20 or 32 bytes length which is fair enough. As it can be seen these types of addresses contain the perfect properties for the tool and therefore these are the ones used.

As we said at the beginning of the explanation prior to encoding the information, as we now know in bech32 format, we encrypt the information to avoid detection due to statistical analysis. The encryption is made using AES and the key is different for each message transmitted. To share the key Pastebin service is used, hence both the server and the victim will have hardcoded a Pastebin user. When the victim wants to exfiltrate information it will scrape the Pastebin site to harvest existing posts aiming to repost with the specified user one paste selected randomly. The content of this Pastebin will be used to generate the password using a KDF function. The idea of reposting and existing paste is to make the traffic as common as possible. On the other end the server once it collects all the data (explained below) from the blockchain it just asks the Pastebin API for the pastes made by the hardcoded user and retrieves the last one made.

The last piece of information to be explained is the exchange protocol carried by the server and the victim on the blockchain. To guarantee a high capacity, multiple consecutive transactions are sent to the blockchain so that one block may contain more than one crafted transaction. On the other hand, to keep the communication stealthy, we generate a new input address for each transaction except for the first and the last one which are hardcoded and serve as a starting and ending event. Since addresses are generated randomly there is no way to track which is going to be the following address used. One may think this way to proceed is a waste of addresses but actually this is the recommended method to perform, **more precisely to receive**, transactions on the blockchain (Bitcoin, 2021), basically because the anonymity on the network is a double-edged sword, making commerce to struggle with people stating that

the same transaction belongs to each of them. If we look at the number of possible addresses worrying about this issue becomes nonsense:

1,461,501,637,330,902,918,203,684,832,716,283,019,655,932,542,976

So, if input addresses are generated randomly, how does the server know which transaction are the "exfiltrating" ones? Wallets come handy for this task. The sender will associate all the addresses being created to a hardcoded wallet name. On the other end the server stores all the transactions it detects after the starting transaction is seen and once the end transaction is published it will request the BlockCypher API the addresses created on the hardcoded wallet which by the way **are presented in the order of creation**, giving us an implicit sequence number. Once it has the addresses it just retrieves the destination address from the transactions whose input addresses are present in the result of the API.

4. Analysis

4.1. Capacity

For addressing this feature, we must take into account the architecture and the implementation of the tool.

Regarding the architecture we have used the BlockCypher testnet blockchain which although its implementation is almost the same to Bitcoin there are several considerations that make the analysis completely different for each network such as different block proposal rates and different traffic density. Moreover, the usage of the BlockCypher API to access these blockchains has a rate limit per hour so the capacity analysis in that case is trivial. Therefore, we are also going to consider a scenario without the free plan API limit.

On the implementation side, the tool enables you to send multiple transactions at the same time but that doesn't mean we have infinite throughput, we have to take into account that we are working with blockchain technology and nothing is published until it is mined. This means that the throughput of the tool will depend on how long the transactions take to be mined in blocks. This is actually something unpredictable and the only way of making your transaction to be mined earlier is to offer higher fees to miners but still that doesn't give you an exact metric, that's the reason why we are going to assume a probability percentage of having our transactions on a block adjusted to each blockchain activity.

Taking all of these into account we obtain the following values:

				Bytes/H	
	Mining %	Block/H	TX/Block	Without API	With API
Bitcoin	30%	6	1800	64,8 M	240
Blockcypher	100%	60	1800	2160 M	800

Bytes/Tx	20
[API] Req/H	200
Req/Chunk	5

These are the formulas used to obtain the final values:

$$Whitout\ API = Mining\% * \frac{Block}{H} * \frac{Tx}{Block} * \frac{Bytes}{Tx}$$

$$With\ API = \ Mining\% * \frac{Req}{H} * \frac{Req}{Chunk} * \frac{Bytes}{Tx}$$

Note that due to the high restrictions with the API, the formula for the API ignores “Block/min” and “TX/block”. Instead “Req/H” and “Req/Chunk” are used.

As it can be seen from the table above BlockCypher is the clear winner mainly because of its high mining probability and its high mining rate. However, as we are going to see in the next section this shouldn´t be the only fact to consider in order to choose the blockchain to use.

4.2. Robustness

Robustness in this tool has been achieved for free, it has been accomplished due to a side effect on a design decision, and that is the usage of bech32 encoded addresses to exfiltrate information. The mere encoding itself appends to the input an checksum with correction capabilities. This is indeed kind of natural since on any cryptocurrency having the money been moved precisely to the account specified is critical.

Thanks to this type of address, we don't need to worry about bit corruption since that not only would be detected but corrected.

4.3. Stealthiness

Stealthiness in this work implies that the transaction must be as normal as any other one, this implies that all its fields contain data undifferentiable from normal transactions. Therefore we are going to analyse stealthiness in terms of the most important fields of a transaction.

Addresses: The tool is exfiltrating data by hiding the information in the output address of the transaction, usually this field contains a high entropy stream result of applying an encoding to a hash function result. Since we are encrypting the data before sending it the entropy of the address is almost the same as the one of a hash function.

Talking about the input address the tool generates a new random address for each new transaction so there is no direct way to correlate transactions between them.

Value: Amounts have been set to be generated randomly in a broad range so that there is no correlation.

Script && Script Pub Key: The most common payment scheme used in bitcoin is P2PKH so this is the one that we use in our transactions.

Apart from this application-level analysis another interesting perspective is the protocol-based analysis. Regarding this the victim makes use only of HTTP calls for communicating with the Blockcypher and Pastebin API which is completely normal, so in this sense the application preserves stealthiness.

4.4. Underlying algorithm

The algorithm doesn't stick to any steganography model: cover selection, cover synthesis or cover modification, so in fact, we are talking about a coverless steganography tool.

The approach of selection and synthesis cover are discarded because the algorithm does not have any database with addresses. Neither is it a modification-based tool because we are creating our own addresses.

So, we can conclude that the algorithm designed is a coverless steganography tool.

5. Counterfeit

As it has been seen in the previous section the tool makes the most of the Bitcoin specification to hide itself in the mess of transactions but if some specific conditions are satisfied some level of awareness that something is happening can be achieved although this can neither enable to detect the information being sent, because of encryption, nor stopping the communication, because of the continuous address generation. Following we explain the level of detection that can be performed.

As any steganographic tool this one requires start and end notifications events between the sender and the receiver to properly send the message. In the case of this tool this is accomplished by hardcoding a bitcoin address on both ends which is precisely used on these two occasions. This by itself is not that much of a problem since it is totally common to reuse an address for paying (not for receiving). The problem comes when this is added to the fact that for increasing the capacity of the tool multiple transactions are sent non-stop. When sending large messages this can expose some information to a statistical analysis, such as a significant increase in the number of transactions in the network. Moreover, this statistical analysis can be extremely detectable on the BlockCypher blockchain due to its lack of activity.

Bibliography

Alsalami, N., & Zhang, B. (2018, December 4). Uncontrolled Randomness in Blockchains: Covert Bulletin Board for Illicit Activities.

Bitcoin. (2019, September 24). *en.bitcoin.it*. Retrieved from https://en.bitcoin.it/wiki/BIP_0142

Bitcoin. (2019, September 24). *en.bitcoin.it*. Retrieved from https://en.bitcoin.it/wiki/BIP_0173

Bitcoin. (2020, November 17). *en.bitcoin.it*. Retrieved from https://en.bitcoin.it/wiki/Vanitygen

Bitcoin. (2020, June 27). *en.bitcoin.it*. Retrieved from https://en.bitcoin.it/wiki/UTXO

Bitcoin. (2020, January 17). *en.bitcoin.it*. Retrieved from https://en.bitcoin.it/wiki/Coinbase

Bitcoin. (2020, June 27). *en.bitcoin.it*. Retrieved from https://en.bitcoin.it/wiki/OP_RETURN

Bitcoin. (2021, March 14). *en.bitcoin.it*. Retrieved from https://en.bitcoin.it/wiki/Segregated_Witness

Bitcoin. (2021, April 11). *en.bitcoin.it*. Retrieved from https://en.bitcoin.it/wiki/Address_reuse

Bitcoin. (2021, March 14). *en.bitcoin.it*. Retrieved from https://en.bitcoin.it/wiki/Transaction

Bitcoin. (2021, February 25). *en.bitcoin.it*. Retrieved from https://en.bitcoin.it/wiki/Technical_background_of_version_1_Bitcoin_addresses

BlockCypher. (n.d.). *www.blockcypher.com*. Retrieved from https://www.blockcypher.com/dev/bitcoin/?python#introduction

Giron, A. A., & Martina, J. E. (2020). Bitcoin Blockchain Steganographic Analysis. In A. A. Giron, *International Conference on Applied Cryptography and Network Security. Springer, Cham* (pp. 41-57). Rome: Springer.

Klitzke, E. (2017, July 20). *eklitzke.org*. Retrieved from https://eklitzke.org/bitcoin-transaction-malleability

Nakamoto, S. (2019, November 27). *tools.ietf.org*. Retrieved from https://tools.ietf.org/id/draft-msporny-base58-01.html

Omid Torki, M. A.-T. (2021, January 08). Blockchain for steganography: advantages, new algorithms and open challenges. Retrieved from https://arxiv.org/pdf/2101.03103.pdf

Partala, J. (2018, August 20). Provably Secure Covert Communication on Blockchain. Oulu, Finland. Retrieved from https://www.mdpi.com/2410-387X/2/3/18/htm

Xu, M., Wu, H., Feng, G., & Zhang, X. (2019). Broadcasting steganography in the blockchain. In *Internation Workshop on Digital Watermarking* (pp. 256-267). Springer.

Zhang, L., Wang, W., Zhang, Z., & Waqas, R. (2020, July). A Covert Communication Method Using Special Bitcoin Addresses Generated by Vanitygen. Hong Kong, Japan.

.