Analysis of transaction flooding attacks against Monero

João Otávio Chervinski*†‡, Diego Kreutz‡, Jiangshan Yu*
*Monash University, Australia
†CSIRO Data61, Australia
‡Federal University of Pampa, Brazil

2021 IEEE International Conference on Blockchain and Cryptocurrency (ICBC) | 978-1-6654-3578-9/20/$31.00 ©2021 IEEE | DOI: 10.1109/ICBC51069.2021.9461084

***Abstract*—Monero was one of the first cryptocurrencies to address the problem of providing privacy-preserving digital asset trading. Currently, it has a market capitalization of over 2.5 billion US dollars and is among the 15 most valuable cryptocurrencies. This digital currency aims to protect users' identities and hide transaction information by using obfuscation mechanisms such as stealth addresses and ring signatures. However, in spite of the efforts to protect Monero's users' privacy, researchers have found ways to identify true payment keys within a ring signature in the past, making attacks against transaction privacy feasible. Since then, the system has received updates and adopted improved measures to provide privacy. This work presents an analysis on how an attacker can take advantage of the system's current settings to conduct both a high-profile transaction flooding attack and a stealthier version. Our results show that after flooding the network for 12 months, the attacker can identify the true spend of 46.24% of newly created transaction inputs by conducting the strongest attack and 14.47% by using the low-profile strategy.**

***Index Terms*—Monero, Privacy, Blockchain, Transaction tracing**

I. Introduction

The buzz around Bitcoin [1] and cryptocurrencies led to fast paced innovation in payment systems around the world. One of the main reasons for the widespread interest in digital currencies is the potential carried by the technology that powers them, a distributed and decentralized data structure called blockchain. Blockchains enable the creation of decentralized trustless networks in which different parties can participate in transactions without having to rely on a trusted third-party as an intermediary. Transactions that take place between users are bundled into blocks and added to the end of a chain of blocks that contain previous transactions. When a new block is added, the state of the system is updated based on the information it contains. In the Bitcoin network, participants reach consensus about blocks that should be accepted by using an algorithm called *Proof-of-Work* (PoW), which requires solving a computational puzzle. Cryptocurrencies, in general, are not regulated by a central authority. Instead, they are maintained by world-wide contributors and both the source-code and history of transactions are open and accessible by anyone.

In the Bitcoin network, each user has a pair of public and private keys, which are used for receiving and sending payments, respectively. The user's public key is used as a pseudonym and payment receiving address and the associated private key is used to claim and spend the received money. As these addresses are pseudonyms and (supposedly) do not reveal any information about the owner, users believed that their Bitcoin transactions were anonymous. However, research has shown that attackers are able to associate the Bitcoin addresses of users with their IP addresses and user names found in forums and websites [2]–[4].

In order to provide a more secure and fungible alternative to other cryptocurrencies, Monero was launched in 2014. It was one of the first cryptocurrencies designed to address privacy issues (e.g. avoid linking user data with their pseudonym addresses) of previous digital coins. Monero is based on the CryptoNote protocol [5], which provides two main features, transaction untraceability and address unlinkability. Untraceability means that given a transaction input with multiple keys, it should not be possible to reveal which one of the keys was used to perform the payment, preventing the transaction from being traced. Address unlinkability ensures that when given two different addresses and no additional data, it should not be possible to link them to the same user. Monero also provides mechanisms to hide from blockchain observers the amount of coins sent in a transaction.

Unlike the immutable pseudonym keys used in other cryptocurrencies, Monero's users rely on the use of two pairs of keys, one pair of viewing keys and one of spending keys. An user's address is derived from the two public keys and some additional information. Everytime someone wants to send a payment to an address, a new, unique key is generated to receive that payment. This unique key is called a stealth address and it is used to provide unlinkability in the blockchain. The sender's information is protected through decoy keys called mixins. Those keys are outputs generated in previous transactions, which are used together with the payment key to create a signature that prevents observers from guessing the real input to a transaction.

Despite providing stronger security mechanisms, Monero has been a target of attacks that reduce transaction privacy [6]–[8]. Those attacks try to exploit weaknesses in the system to reveal transaction payment keys. For instance, in earlier versions of the system, design choices allowed the execution of attacks capable of tracing a large amount of input keys [9], [10]. In particular, those attacks exploited the fact that the

978-1-6654-3578-9/21/$31.00 ©2021 IEEE

system did not enforce a minimum number of decoys in a given input (allowing an input to be composed only by the key being spent) and the decoy selection algorithm, which followed a triangular distribution that leaked some information about the age of keys.

To prevent further attacks, Monero has received updates to increase the number of decoy keys required in transaction inputs and adopted more effective strategies for decoy key selection. Currently, as of Monero version 0.16.0.3 (September 2020), transaction inputs are composed by 11 input keys, where 10 of those are decoys. For the mixin selection, the system uses a gamma distribution to select decoys based on their position in the blockchain [11].

We propose and evaluate a new attack based on flooding the network with transactions to trace the payment keys of honest users in Monero's blockchain. While the idea is rather simple, i.e., to flood Monero's network with transactions whose input and output keys are owned by the attacker, we need to address some challenges and pay attention to details that impact the results. First, the transaction fee is proportional to its size in bytes, which means that the attacker should carefully chose a cost-effective size. Second, we try to maximize the number of output keys per transaction. Ideally, the attacker should create cost-effective transactions with as many output keys as possible. Third, we discuss how to explore chain reactions on the transaction tracing attack and how it can potentially increase the number of traced transaction inputs.

Our main contribution is as follows: (a) a new transaction flooding attack to trace payment keys of other users of Monero's system; and (b) an evaluation of the effectiveness of the attack.

This paper is organized as follows. In Section II we introduce the transaction flooding attack. We evaluate the attack using data generated with basis on empirical observations of Monero's blockchain in Section III. A discussion on related work is provided in Section IV Finally, we provide final remarks in Section V.

II. The Transaction Flooding Attack

Assuming a Monero transaction tx with one input $tx.in$ containing four input keys ($tx.in = \{pk_1, pk_2, pk_3, pk_4\}$), while one of the keys (e.g. pk_4) represent the real coin being spent, the remaining three keys are being used as decoys to hide the real output key being spent in the transaction. However, if three of the public keys (e.g. pk_1, pk_2 and pk_3) are owned by the attacker, it becomes straightforward to find out which one is the payment key. This is one of the most basic principles of the transaction flooding attack, i.e., the attacker has to own a set of valid output keys and make it as big as possible.

The attacker can remove or mark as known a key pk_x from the input of a transaction created by another user when the key belongs to him/her. If the attacker has not yet spent the key on a payment, s/he knows that pk_x is being used as a decoy. Second, if the attacker knows that the key has already been spent in a previous transaction (e.g. pk_4), then s/he knows that the key is a mixin as well. In both cases, the key pk_x can be safely removed from the input keys $tx.in$ of a transaction tx.

The previous example can be exploited in practice through a transaction flooding attack. This attack explores Monero's ring signature scheme, which hides the real input keys by mixing them with different output keys (used as decoy keys) generated by previous transactions. The core idea of the transaction flooding attack is simple. The attacker has to create transactions to build up a big knowledge base (i.e. set of output keys) from which the system might select keys to be used as mixins in future transactions. As discussed before, if the attacker knows all keys except one of the transaction's input $tx.in$, s/he can easily find out which key is being spent in that input of the transaction.

In Monero, every time a new transaction is created, 10 mixins must be included in each input together with the true spend key to form an input ring of size 11. The system selects the mixins from the output keys of previous transactions with a decoy picker that uses a gamma distribution [11] and adds them to the transaction's input, as shown in Figure 1. Each input $tx.in$ of transaction tx will have its own set of mixins. Finally, a digital signature is created to allow the payee to get the payment without requiring the payer to reveal his key.

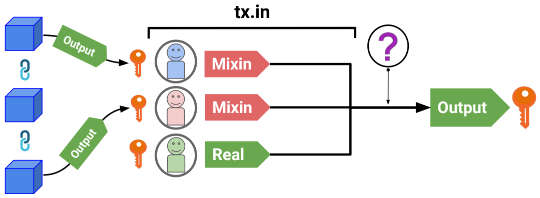

Fig. 1. Moneros' ring signature.

The main challenge for a successful transaction flooding attack is to have enough keys so that the system selects all mixins of an input $tx.in$ from the attacker's set of keys. In order to own output keys, the attacker has to flood the network with valid transactions (ideally low-cost ones to make the attack feasible). The outputs of those transactions will be sent to addresses owned by the attacker.

The number of output keys of a transaction (used as decoys in future transactions) is given by the number of addresses receiving a share of the payment. Each receiving address will get an output key containing a number of coins (XMR). Those output keys can be later selected by the system to be used in the mixin set of future transaction inputs.

It is worth emphasizing that each transaction has a fee. This fee is used for paying off the miners that perform computational work to validate the transaction.

On October of 2018, the Monero community announced the launch of the Bulletproof protocol[1] to replace the *Ring Confidential Transaction* (RingCT) on the task of generating range proofs. This new protocol generates cryptographic proofs up to 97% smaller in size when compared to those

[1] https://www.getmonero.org/2018/10/11/monero-0.13.0-released.html

of RingCT. Smaller proofs lead to smaller transaction size and, as a result, lower fees. In short, the introduction of the Bulletproof protocol made the transaction flooding attack even more interesting and worth investigating.

With the introduction of the Bulletproof update, Monero transactions now pay a fee based on their weight. The weight of a transaction is calculated based on the number of outputs it contains. Initially, for transactions with only two outputs, their weight is equal to their size in bytes. As the number of outputs increases, the transaction weight also increases although it increases more than the transaction size does. This happens because bulletproofs increase in size sub-linearly when more outputs are added to a transaction, but the time required to verify those proofs is still linear to the amount of outputs. The increase in transaction weight takes that into account and increases the fees to make up for the time required for proof verification [11].

In order to accommodate changes in transaction volume, Monero supports dynamic block sizes. This means that whenever the volume of transactions starts to increase, the block size will adapt over time so that users can get their transactions into blocks without having to compete for space by paying high fees to miners. However, this also means that if the amount of transactions increases rapidly beyond the size limit, the block size won't grow accordingly. If miners wish to increase the block size past the current limit, they will suffer a block reward penalty. This makes the creation of oversized blocks only profitable if the fees paid by the transactions included in the block are higher than the block reward penalty incurred. This prevents malicious users from bloating the system with large blocks while paying low transaction fees.

To execute a transaction flooding attack, a malicious party needs to take the system's measures against flooding into account to avoid spending large amounts of money due to security measures. With that in mind, an optimal strategy would be to take advantage of the free space inside blocks to create flooding transactions. Currently, Monero's minimum maximum block size, i.e, the maximum block size for when the transaction volume is low, is 300 kB. By keeping the block size within the limit, the attacker does not have to pay higher fees in order to accommodate his/her transactions, but only try to take advantage of the unused space inside each block. With that in mind, we would like to evaluate how can an attacker benefit from a transaction flooding attack that simply takes advantage of the free space inside Monero's blocks to perform transaction input tracing attacks.

A. The Attacker Model

As Monero's blockchain data is public and accessible to anyone, the attacker is able to access blockchain data. We assume that the attacker is willing to pay transaction fees in exchange for the ability to trace transaction inputs.

We also assume that the attacker possesses at least two different Monero addresses to use in the process of flooding the network. One of those addresses must contain the amount of XMR needed to cover the fees paid for the attack transactions. The other addresses will be used to receive transactions and store output keys. Note that creating a new Monero wallet is easy and has no extra cost.

Finally, we assume that the attacker is able to create as many transactions as he wants at any given time t as long as he pays the transaction fees. It is up to the miners to select and validate the transactions, i.e., there is no timing guarantee, and if the network's transaction pool has transactions that pay higher fees waiting to be confirmed, those will be selected by the miners first.

B. Flooding Monero's Network

To trace input keys, the attacker must create a large number of output keys before he/she tries to trace transaction inputs. Only transactions created after the beginning of the attack are subject to tracing. As we show in Section III, the attack gets more effective as the attacking timeframe stretches.

Monero selects decoy keys using a gamma distribution to select a block of transactions and then selects an output key randomly from within that block [2]. This process is repeated until 10 decoy keys are obtained for each transaction input ring. The decoy selection algorithm is more likely to select output keys from blocks generated in the last few days and as time goes on and outputs become older, their likelihood of being selected decrease.

The strategy adopted by the attacker is to generate new output keys by sending payments to his own Monero wallet addresses. Each transaction output spends 1 piconero (10^{-12} XMR), the minimum amount allowed in a transfer, leading the cost of creating transactions to be mostly composed by transaction fees.

To maximize the efficiency of the attack, the output keys must be evenly distributed across the blockchain. Due to that, the attacker must continually create transaction outputs in order for the transaction input tracing to be as effective as possible. Based on empirical observations of Monero's main net data from blocks 2,154,590 (1 Aug, 2020) to 2,176,789 (31 Aug, 2020), we devised a strategy that can be followed to take advantage of unused block space by an attacker. Table I summarizes the data obtained in our observations.

In our analysis of blockchain data from the month of August 2020, we observed that the average block space taken up by transactions is 50.984 kB of data and each block contains around 18 transactions in average. Considering the minimum maximum block size of 300 kB, this leaves around 250 kB of free space that can be used by an attacker. Additionally, we found out that 5.941% of the blocks mined in the network contain no user generated transactions, only the miner reward transaction, and therefore are empty.

We were also able to identify the number of outputs that are generated more often by user transactions. Table II presents the possible output set sizes for transactions and how frequently they appear in the blockchain.

[2] https://github.com/monero-project/monero/blob/master/src/wallet/wallet2.cpp#L1030

TABLE I
SUMMARY OF DATA FROM MONERO BLOCKS 2154590 TO 2176789 (AUGUST 2020).

Description	Value
No. of transactions	403,755
Avg. block size	50.984 kB
Avg. transactions per block	18.187
No. of transaction inputs	880,750
Avg. inputs per transaction	2.181
No. of transaction outputs	939,566
Avg. outputs per transaction	2.327
Percentage of empty blocks	5.941%

The two most frequent output set sizes are 2, which makes up 94.943% of transactions, and 16, which appears in 2.089% of the transactions in the blockchain. It is interesting to note that 16 output transactions are the most effective in terms of the ratio between the fees required to create the transaction and the number of outputs generated. With that in mind, since we desire to optimize the number of outputs we can generate given the free space available in each block, we conduct an analysis of the transaction flooding attack when the attacker generates 16 output transactions to flood the network.

We are aware that 16 output transactions only make up 2.089% of the total transactions in the blockchain. A sudden increase in this type of transaction would raise suspicion and might even lead developers to temporarily limit the number of outputs that can be generated by a transaction to thwart the attack. For that reason, we also analyze how an attacker can benefit from a transaction flooding attack by creating transactions that generate only 2 outputs.

Because the majority of honest transactions generate 2 outputs, even if the abnormal volume of transactions raises suspicion, taking action against the the attacker without harming honest network users becomes more difficult. That is the case because user addresses in Monero are protected by one-time addresses [11] and are unique for every transaction. Additionally, users' IP addresses are protected by Dandelion++, making it hard to take measures against the attacker's IP address [12], [13].

In order to evaluate the feasibility of conducting a transaction flooding attack in our two proposed scenarios, we performed a preliminary analysis in a private Monero testnet using version 0.16.0.3 of the system. We refer to the attacker's 2 inputs/2 outputs transactions and 2 inputs/16 outputs transactions as 2/2 and 2/16 transactions respectively. The objective of this analysis was to identify how many transactions an attacker can fit in a block while taking into account the data obtained from the main Monero chain. For our experiment we considered a network scenario in which every block contains 20 user generated transactions with 2 inputs and 2 outputs each and one miner reward transaction with 1 output. The average size of 2/2 transactions is 2.544 kB and this value is the same for both honest user's and attacker's transactions. The 2/16 transactions have an estimated size of 3.802 kB. The approximate sizes were obtained by analyzing Monero's blockchain data and through verification using a function that estimates transaction sizes, the latter obtained from the source code on Monero's Github repository[3].

The 20 user generated transactions considered in our experiment occupy 50.88 kB of block space, an amount that is similar to the average amount of data contained in a block on Monero's network. The data obtained in our empirical observation of Monero's blockchain is shown in Table I. Note that while transactions can have 1 input, 2 inputs are used if users have multiple output keys in their wallet when the transaction is created[4]. This strategy is used in order to make the transactions look similar to one another, with the majority being composed by 2 inputs and 2 outputs [14]. For that reason we chose to consider user and attacker created transactions containing 2 inputs.

After adding 20 user transactions to the transaction pool, we created attacker transactions to observe how many of those can be added in the remaining space inside the blocks. For 2/2 transactions, we found that a maximum of 96 can be added to a block in addition to the user transactions, leading to a total of 116 transactions and a block size of 294.43 kB according to the Onion Monero Blockchain Explorer tool[5]. When it comes to 2/16 transactions, 32 can be added on top of user transactions, increasing the block size to 293.95 kB and the number of transactions to 52. One interesting observation is that when adding up the size of the individual transactions (20 * 2.544 kB transactions and 32 * 3.802 kB transactions) we obtain 172.544 kB, but the blockchain explorer reports a block size of 293.95 kB. If instead of the size in kB, we perform the calculation using the transaction weight[6], we obtain a total of 293.96 kB, which is closer to the actual block size displayed in the block explorer.

Having obtained the number of attacker transactions that can be included in the free space inside each block, we need to execute the attack to evaluate its impact in the system.

C. The Tracing Algorithm

In order to trace transactions, a malicious actor can either target specific transactions and its transaction inputs or run a tracing algorithm over all blockchain transaction data generated after the attack began. In both cases, the attacker must keep track of the keys generated by attack transactions during the network flooding phase.

Targeting a specific input in another user's transaction is straightforward, the attacker only needs to check the keys included in the input ring and mark those owned by him as decoys. If able to identify 10 out of the 11 keys in the ring, the

[3] https://github.com/monero-project/monero/blob/8966ac314c01715bede8c2e8e4d0a896d3edc3d5/src/wallet/wallet2.cpp#L752

[4] https://github.com/monero-project/monero/blob/1bb4ae3b5e9c6e9599fbefb2fd311d7c850ff06c/src/wallet/wallet2.cpp#L9315

[5] https://github.com/moneroexamples/onion-monero-blockchain-explorer

[6] https://github.com/monero-project/monero/blob/8966ac314c01715bede8c2e8e4d0a896d3edc3d5/src/wallet/wallet2.cpp#L813

TABLE II
FREQUENCY OF OUTPUT SET SIZES IN THE BLOCKCHAIN.

No. of outputs	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Transactions (%)	0	94.943	0.989	0.647	0.256	0.191	0.156	0.126	0.106	0.097	0.104	0.073	0.091	0.071	0.061	2.089

remaining one is the true spend. If unable to identify 10 keys, the attacker may still be able to reduce the size of the decoy key set according to the number of keys he knows, allowing him to guess which one is the true spend with a probability of $\frac{1}{n}$, where n is the number of keys remaining in the ring.

The input tracing procedure can also be executed for every transaction added to the chain after the attack began. To do that, the attacker must obtain a copy of Monero's blockchain data and follow the steps described in Algorithm 1. This procedure is more effective as it allows traced inputs to be used towards reducing the privacy of other transactions as well. That is done by adding newly discovered true spend keys to the attacker's set of known transaction outputs, as now he possesses knowledge of when this particular key has been spent. The discovered true spend key can now be marked as spent in other transaction inputs, since it has already been spent somewhere else and can't possibly be the true spend in a different transaction input.

The procedure begins with two data inputs, namely the block data extracted from Monero's blockchain and the set of keys owned by the attacker (Line 1). On each iteration of Algorithm 1, the inputs of each transaction are extracted (Line 8). For each input (Lines 9 to 23) there's the need to check the output keys contained in the ring and mark those which are in the attacker's set of keys. If the attacker knows all but one key of the transaction input (Lines 11 to 18), then the remaining key is the true spend (traced key) and should be added to the list of keys known by the attacker (Lines 19 to 21). When the set of attacker's keys increases (Line 20), it means new true spend keys have become known. In that case the analysis will be run again on all blocks as more input keys can be potentially traced (Lines 5 to 24). The algorithm will stop only when zero inputs (Line 19) were able to be traced in the last iteration (Line 4 and 26), meaning that there no new keys and therefore new true spend keys cannot be identified. Once ended, the algorithm returns the list of identified true spend keys (Line 30).

III. EVALUATION AND RESULTS

In order to obtain our results we conducted simulations for 2 distinct scenarios using data generated according to the parameters obtained in our blockchain analysis, described in Section II-B. Scenario I consists of a network setting where the attacker executed a flooding attack using transactions with 2 inputs and 16 outputs, generating 32 transactions and 512 malicious outputs in each block in addition to 21 transactions and 41 outputs generated by honest users. Scenario II analyzes the impact of an attack where the malicious actor creates transactions with 2 inputs and 2 outputs, generating 96 transactions and 192 malicious outputs in each block for a total of 233 outputs per block when adding the 41 user generated outputs.

For each scenario we measured the attack's impact from a period of one month to twelve months of sustained flooding (filling up the free space in every block). For the evaluation, we used the same decoy key selection algorithm based on the gamma distribution that is used on Monero's source code.

Our results show, for a transaction input ring created after the attack, what is the likelihood of its decoy keys set to be composed only by attacker owned outputs. We also show how likely it is for the decoy set to be composed of a number of attacker's keys ranging from 0 to 9. In order to obtain our results, we generated the data containing the transactions according to each attack scenario and then executed the decoy selection algorithm on the data to generate 1 million input rings. Based on the results obtained, we extracted the probability for an input ring to be composed by a given number of attacker outputs. Table III shows the results for the flooding attack with 2/16 transactions.

For the attack with 16 output transactions, we observed that it is unlikely for an attacker to be unable to identify at least 3 decoy keys after flooding the network for one month. Also, after only one month of sustained flooding, an attacker is able to remove all of the decoy keys in 41.21% of newly created transaction inputs. This probability grows to 46.24% after twelve months of continuous attack. We argue that a one month attack is the most cost effective, given the 5% increase in traceability power from one to twelve months, but a twelve times increase in the time and number of transactions required.

Table IV shows the results for the 2/2 transactions attack. This can be considered a stealthier version of the transaction flooding attack, due to the fact that the attacker's transactions can easily blend in with honest user's transactions, as they have the same number of inputs and outputs. After attacking the network for one month, the attacker is able to identify the true spend in 10.72% of newly created transaction inputs. After twelve months, this probability grows to 14.47%. In this scenario, in the same way as the first one, we can conclude that a shorter attack duration is more cost effective. Unlike in the first strategy however, the majority of transaction inputs have their anonymity set size reduced by 8 and 9, leaving the attacker with a 33.33% and 50% chance of guessing the real input, respectively.

IV. RELATED WORK

To our knowledge, no existing strategy has been capable of linking real-world information to user addresses on the Monero blockchain. Known attacks against Monero, such as tampering the wallet's code and personification of a remote

Algorithm 1 Tracing input keys

```
procedure TRACE_INPUTS($blocks, attackerKeys$)
    $tracedKeys \leftarrow \{\}$
    while $true$ do
        $knownKeys \leftarrow |attackerKeys|$
        for each $block \in blocks$ do                                   ▷ For each block
            $transactions \leftarrow getTransactions(block)$
            for each $transaction \in transactions$ do
                $inputs \leftarrow getInputs(transaction)$
                for each $input \in inputs$ do
                    $keys \leftarrow getKeys(input)$
                    $mixinSetSize \leftarrow |keys| - 1$         ▷ Currently fixed at 10 mixins + true spend key
                    $mixinsRemoved \leftarrow 0$
                    for each $key \in keys$ do
                        if $key \in attackerKeys$ then
                            $mixinsRemoved \leftarrow mixinsRemoved + 1$
                        end if
                    end for
                    if $mixinsRemoved == mixinSetSize$ then
                        $realKey \leftarrow keys - (attackerKeys \cap keys)$
                        $attackerKeys \leftarrow attackerKeys \cup realKey$
                        $tracedKeys \leftarrow tracedKeys \cup realKey$
                    end if
                end for
            end for
        end for
        if $knownKeys == |attackerKeys|$ then
            $break$
        end if
    end while
    return $tracedKeys$
end procedure
```

TABLE III
PROBABILITIES OF IDENTIFYING DECOY KEYS AFTER CONDUCTING A FLOODING ATTACK WITH 16 OUTPUT TRANSACTIONS.

Attack length (months)	Probability of having decoy keys identified by the attacker										
	0 keys	1 key	2 keys	3 keys	4 keys	5 keys	6 keys	7 keys	8 keys	9 keys	10 keys
1	0%	0%	0%	0.0002%	0.0066%	0.0688%	0.6532%	3.9523%	15.9134%	38.1883%	41.2172%
2	0%	0%	0%	0.0002%	0.0029%	0.0481%	0.4912%	3.2209%	14.3293%	37.5354%	44.372%
3	0%	0%	0%	0.0003%	0.0035%	0.0408%	0.4454%	3.0567%	13.8596%	37.3604%	45.2333%
4	0%	0%	0%	0.0001%	0.0031%	0.0441%	0.4193%	2.984%	13.6687%	37.1598%	45.7209%
5	0%	0%	0%	0.0002%	0.0035%	0.0361%	0.4114%	2.9119%	13.5591%	37.2122%	45.8656%
6	0%	0%	0%	0.0002%	0.0016%	0.0377%	0.4183%	2.8909%	13.3983%	37.137%	46.116%
7	0%	0%	0%	0%	0.0026%	0.04%	0.4119%	2.8625%	13.5346%	37.0752%	46.0732%
8	0%	0%	0%	0.0001%	0.0026%	0.038%	0.4007%	2.8797%	13.4402%	37.0942%	46.1445%
9	0%	0%	0%	0.0002%	0.0029%	0.0411%	0.3922%	2.8597%	13.354%	37.1057%	46.2442%
10	0%	0%	0%	0%	0.0024%	0.0405%	0.4041%	2.8443%	13.3535%	37.059%	46.2962%
11	0%	0%	0%	0%	0.0029%	0.034%	0.4051%	2.876%	13.3579%	37.0503%	46.2738%
12	0%	0%	0%	0.0003%	0.0022%	0.0408%	0.4035%	2.8446%	13.3959%	37.0663%	46.2464%

node, are limited to revealing the real keys spent in transaction inputs [9], [15]–[17].

In earlier versions of Monero, the biased results of the mixin sampling algorithm and the creation of transactions without any decoy keys could be explored by an attacker to trace transaction inputs [9], [10]. The biased sampling algorithm favored older output keys when choosing mixins to be included in a transaction input. That led to the spent key being the most recent one, i.e, generated in the highest block among all the mixins, in more than 90% of inputs created before the algorithm was updated. The lack of decoy keys happened because there was no enforcement of rules to ensure a minimum amount of mixins for each input. Inputs without mixins are traceable as they contain only the spent key. If

TABLE IV
PROBABILITIES OF IDENTIFYING DECOY KEYS AFTER CONDUCTING A FLOODING ATTACK WITH 2 OUTPUT TRANSACTIONS.

Attack length (months)	**Probability of having decoy keys identified by the attacker**										
	0 keys	**1 key**	**2 keys**	**3 keys**	**4 keys**	**5 keys**	**6 keys**	**7 keys**	**8 keys**	**9 keys**	**10 keys**
1	0%	0.0005%	0.0069%	0.0841%	0.5484%	2.6789%	8.8621%	20.1555%	30.236%	26.7017%	10.7259%
2	0%	0.0004%	0.0032%	0.0495%	0.3862%	2.06%	7.3786%	18.3691%	30.0532%	29.0775%	12.6223%
3	0%	0.0002%	0.0026%	0.0436%	0.3491%	1.8562%	6.9399%	17.7506%	29.9131%	29.7777%	13.367%
4	0%	0%	0.0034%	0.0436%	0.3212%	1.792%	6.6444%	17.4056%	29.8314%	30.2132%	13.7452%
5	0%	0.0002%	0.0034%	0.0378%	0.3193%	1.7167%	6.5851%	17.1911%	29.7151%	30.3689%	14.0624%
6	0%	0.0001%	0.003%	0.0403%	0.304%	1.6866%	6.4669%	17.1548%	29.7657%	30.4862%	14.0924%
7	0%	0.0001%	0.0019%	0.0353%	0.2933%	1.6499%	6.4279%	17.015%	29.6069%	30.676%	14.2937%
8	0%	0.0002%	0.0029%	0.0357%	0.2937%	1.6397%	6.3376%	16.9802%	29.6957%	30.7021%	14.3122%
9	0%	0.0002%	0.0027%	0.0336%	0.2927%	1.6322%	6.3519%	16.8741%	29.6969%	30.7457%	14.37%
10	0%	0.0001%	0.0025%	0.0387%	0.2896%	1.6376%	6.3203%	16.9527%	29.6454%	30.7178%	14.3953%
11	0%	0.0002%	0.0032%	0.0385%	0.2891%	1.6215%	6.2682%	16.8732%	29.6624%	30.8374%	14.4063%
12	0%	0.0002%	0.0031%	0.0366%	0.2803%	1.6359%	6.315%	16.8408%	29.5862%	30.8318%	14.4701%

the spent key was chosen by the system to be included as a mixin in a future transaction it could be ignored because it was already spent before, thus it could not be the key being spent in the input. Those strategies allowed attackers to compromise Monero's privacy just by analyzing transaction data contained in the blockchain. The weakness of existing mixin selection algorithms in CryptoNote-style blockchains, including Monero, has been further studied and formalised [6]. Yu et al. have identified two new inference attacks on the transaction untraceability, modelled the mixin selection process as "The Sun-Tzu Survival Problem" and analysed it by using graph theory [6]. They provided theoretical foundations including the least upper bound of transaction untraceability guarantee and a general solution that achieves provably optimal transaction untraceability.

The Monero wallet code was also a target of recent attacks [15]. Findings have shown that an attacker can tamper a wallet code and, consequently, choose the mixins for a given input. As the system will not verify wether or not those keys are being spent in other inputs of the same transaction, this makes it possible to mark all the keys in the transaction as spent, regardless of which input each key is spent in. By setting up a fake cryptocurrency wallet service, the attacker takes control of the selection of mixins and is able to include the spent keys in transactions created by other users, reducing the Monero's essencial privacy guarantees.

Attackers can also use personified remote nodes to trace inputs in the Monero network [16]. To avoid the heavy work of making and keeping up to date a full copy of the blockchain, clients of the network commonly rely on remote nodes to request transaction information and send payments. After checking the possible mixins of the request, the node operator may abort the client transaction and wait for a retry. Upon receiving the retry request, the attacker might be able to identify the real key by searching for the one appearing in both requests.

A recent work proposed a strategy to trace transaction inputs by finding closed-sets of public keys included in transaction inputs [17]. A closed-set is defined as a set of transaction inputs in which the number of distinct keys across all the inputs is the same as the number of inputs in the set. In a closed-set each public key must have been spent in one of the inputs in the set and used as mixin in the other inputs. This allows an attacker to identify output keys that have already been spent and remove them from other inputs. As the number of decoys is reduced the privacy of the transaction inputs is weakened, allowing the execution of further attacks.

V. CONCLUSION

This work presented an analysis on the traceability of Monero's transactions using a transaction flooding attack. The proposed attack consists in exploiting the Bulletproof protocol to create a large number of transactions, aiming to control a large portion of the keys that are used to provide privacy to Monero's transaction inputs. Simulation results show that by executing the proposed attack for one month, a malicious actor could trace 41.21% of all transaction inputs created after the month. The results show the existence of vulnerabilites on Monero's privacy mechanisms. The presented analyses emphasize the importance of detecting and patching vulnerabilities in privacy and security mechanisms provided by cryptocurrencies.

ACKNOWLEDGEMENT

This work was partially supported by the Australian Research Council (ARC) under project DE210100019.

REFERENCES

[1] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," 2008.

[2] A. Biryukov, D. Khovratovich, and I. Pustogarov, "Deanonymisation of clients in bitcoin p2p network," in *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2014, pp. 15–29.

[3] S. Meiklejohn, M. Pomarole, G. Jordan, K. Levchenko, D. McCoy, G. M. Voelker, and S. Savage, "A fistful of bitcoins: characterizing payments among men with no names," in *Proceedings of the 2013 conference on Internet measurement conference*. ACM, 2013, pp. 127–140.

[4] M. Fleder, M. S. Kester, and S. Pillai, "Bitcoin transaction graph analysis," *arXiv preprint arXiv:1502.01657*, 2015.

[5] N. Van Saberhagen, "Cryptonote v 2.0," 2013. [Online]. Available: https://static.coinpaprika.com/storage/cdn/whitepapers/1611.pdf

[6] J. Yu, M. H. A. Au, and P. J. E. Veríssimo, "Re-thinking untraceability in the cryptonote-style blockchain," in *32nd IEEE Computer Security Foundations Symposium, CSF 2019, Hoboken, NJ, USA, June 25-28, 2019*. IEEE, 2019, pp. 94–107.

[7] D. A. Wijaya, J. K. Liu, R. Steinfeld, D. Liu, and J. Yu, “On the unforkability of monero,” in *Proceedings of the 2019 ACM Asia Conference on Computer and Communications Security, AsiaCCS 2019, Auckland, New Zealand, July 09-12, 2019*, S. D. Galbraith, G. Russello, W. Susilo, D. Gollmann, E. Kirda, and Z. Liang, Eds. ACM, 2019, pp. 621–632.

[8] T. Cao, J. Yu, J. Decouchant, X. Luo, and P. Veríssimo, “Exploring the monero peer-to-peer network,” in *Financial Cryptography and Data Security - 24th International Conference, FC 2020, Kota Kinabalu, Malaysia, February 10-14, 2020 Revised Selected Papers*, ser. Lecture Notes in Computer Science, J. Bonneau and N. Heninger, Eds., vol. 12059. Springer, 2020, pp. 578–594.

[9] A. Kumar, C. Fischer, S. Tople, and P. Saxena, “A traceability analysis of monero’s blockchain,” in *European Symposium on Research in Computer Security*. Springer, 2017, pp. 153–173.

[10] A. Miller, M. Möser, K. Lee, and A. Narayanan, “An empirical analysis of traceability in the monero blockchain,” *arXiv preprint arXiv:1704.04299*, 2017.

[11] K. M. Alonso, “Zero to monero,” 2020.

[12] M. Outreach, “Dandelion++ for monero,” 2020. [Online]. Available: https://www.monerooutreach.org/stories/dandelion.html

[13] ErCiccione, “Another privacy-enhancing technology added to monero: Dandelion++,” 2020. [Online]. Available: https://web.getmonero.org/2020/04/18/dandelion-implemented.html

[14] “Why does monero create two or more inputs for transactions?” 2019. [Online]. Available: https://monero.stackexchange.com/questions/11530/why-does-monero-create-two-or-more-inputs-for-transactions

[15] D. A. Wijaya, J. Liu, R. Steinfeld, and D. Liu, “Monero ring attack: Recreating zero mixin transaction effect,” in *2018 17th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/12th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE)*. IEEE, 2018, pp. 1196–1201.

[16] K. Lee and A. Miller, “Authenticated data structures for privacy-preserving monero light clients,” in *IEEE EuroS&PW*. IEEE, 2018, pp. 20–28.

[17] Z. Yu, M. H. Au, J. Yu, R. Yang, Q. Xu, and W. F. Lau, “New empirical traceability analysis of cryptonote-style blockchains,” 2019.