BAD: Blockchain Anomaly Detection

4 authors, including:

Matteo Signorini
Nokia Bell Labs, Paris, France
13 PUBLICATIONS   **28** CITATIONS

Roberto Di Pietro
Hamad bin Khalifa University
224 PUBLICATIONS   **4,463** CITATIONS

Waël Kanoun
19 PUBLICATIONS   **108** CITATIONS

Some of the authors of this publication are also working on these related projects:

Project   Blockchain View project

Project   Offline MicroPayments View project

BAD: a Blockchain Anomaly Detection solution

Matteo Signorini and Matteo Pontecorvi
Nokia Bell Labs
Paris-Saclay, France
firstname.lastname@nokia-bell-labs.com

Wael Kanoun
Thales
Abu Dhabi, UAE
wael.kanoun@thalesgroup.com

Roberto Di Pietro
Hamad Bin Khalifa University
Doha, Qatar
rdipietro@hbku.edu.qa

Abstract—**Anomaly detection tools play a role of paramount importance in protecting networks and systems from unforeseen attacks, usually by automatically recognizing and filtering out anomalous activities. Over the years, different approaches have been designed, all focused on lowering the false positive rate. However, no proposal has addressed attacks targeting blockchain-based systems. In this paper we present BAD: the first Blockchain Anomaly Detection solution. BAD leverages blockchain meta-data, named forks, in order to collect potentially malicious activities in the network/system.**

BAD enjoys the following features: (i) it is distributed (thus avoiding any central point of failure), (ii) it is tamper-proof (making not possible for a malicious software to remove or to alter its own traces), (iii) it is trusted (any behavioral data is collected and verified by the majority of the network) and (iv) it is private (avoiding any third party to collect/analyze/store sensitive information). Our proposal is validated via both experimental results and theoretical complexity analysis, that highlight the quality and viability of our Blockchain Anomaly Detection solution.

Index Terms—**Blockchain technology, Intrusion Detection Systems, Distributed Systems**

I. INTRODUCTION

The *Internet of Things* (IoT) digital revolution has brought a wide range of smart devices in the global market that are remotely accessible via Internet and able to communicate and cooperate with each other. This opens great opportunities from an application and service point of view, but it also creates new security challenges as devices are easily accessible from Internet[1].

To solve this situation, *intrusion detection systems* (IDS) have been developed in the past as tools aimed at strengthening the security of complex networks and systems via capturing, monitoring, and analyzing the peers' traffic or, more in general, their behavior [1]. These approaches, usually based on log analysis and data correlation, aim at building attack models and mitigation strategies on top of them. Existing IDS can be classified based on their approach to two classes: signature recognition or anomaly behavior [2]. On one hand, the first class leverages databases where signatures of well-known attacks are matched. This database is then used as a reference model to detect future occurrences of such attacks. Hence, this approach is not able to recognize new attacks whose signatures are still unknown. On the other hand, anomaly detection approaches build models of normal behavior and rise alerts for deviations from such baselines.

[1] https://en.wikipedia.org/wiki/2016_Dyn_cyberattack

Thus, the goal of an *anomaly detection system* (ADS) is to build the normal behavior model and then to challenge it with new/unknown behaviors in order to analyze how much they are close to the reference model.

IDS and *ADS* proved their functionalities so far, especially when based on trusted third parties that are responsible to build reference model and to alert end-users or end-devices if an unexpected behavior has been detected. We can consider the classic case of anti-virus companies that build and manage threat databases, which are then used to identify known threats or to predict zero days attack. However, this approach does not work for truly distributed peer-to-peer communities that lack trusted anchors or centralized management as in blockchain-based applications. Firstly designed as a support tool for Bitcoin [3], the blockchain technology allows untrusted peers within open (i.e. permission-less) communities to agree on the status of a shared database, without the necessity to access trusted third parties. The only assumption is that, the majority of involved peers is honest and willing to keep the application up and running against malicious users. However, has shown in real life applications, attackers can *eclipse* their victims (i.e. manipulate honest nodes access to the mainstream global blockchain) thus reducing the number of honest peers participating in the overall blockchain network. Eclipsing a node allows or simplifies several types of attacks as shown in [4], [5].

In this paper we propose BAD: the first Blockchain Anomaly Detection solution that allows peers in a blockchain network to be protected against eclipse attacks by sharing information on previous attacks (i.e. by re-distributing malicious forks to the whole peer-to-peer community).

To the best of our knowledge, our approach is the first one that leverages forks on a global scale to detect and prevent local threats. The idea behind BAD is to collect local attack logs in the form of (hashed) malicious transactions. These logs are generated by BAD because of an attack sequence injected by an attacker on isolated victims, and they are later reused to prevent similar attacks on uncorrupted nodes.

More precisely, the attack logs (usually discarded in standard blockchain applications) populate a threat database that allows other potential victims to be resilient against zero-day attacks already discovered.

This paper is organized as follows: in Section II the blockchain background technology is introduced as well as previous works on anomaly detection systems. Section III

describes our threat model. Sections IV and V introduce respectively our solution and the related experimental results. In Section VI we discuss the empirical overhead analysis of BAD as well as its theoretical complexity, while Section VII addresses issues and limitations of our system. Finally, Section VIII concludes the paper and introduces future work.

II. Technology Background and Related Work

In the rest of this paper we adopt the same notation used in [6] to describe blockchain and, in general, state replication distributed protocols. We will only consider permissionless blockchain technologies, where a race among peers is established for mining blocks and rising potential forks (see NPCcoin[2], Litecoin[3] and Dash[4] which are the cryptocurrencies with higher market capitalization[5]). We give some definitions from [6] below.

An *output* is a tuple consisting of an amount of bitcoins and a spending condition. The latter is usually a valid signature associated with the private key of the spender address, however it can be generally a script which could be exploited by an attacker.

An *input* is a tuple consisting of a reference to a previously created output and arguments for the spending condition. This allows the transaction creator has the permission to spend the referenced output. We call *UTXO* the set of unspent transaction outputs.

Definition 1. *[6] A transaction is a data structure that describes the transfer of bitcoins from spender to recipients. The transaction consists of a number of inputs and new outputs. The former result in the referenced output spent (removed from the UTXO), and the latter being added to the UTXO.*

Definition 2. *[6] A block consists of a transactions' list, a reference to the previous block and a nonce. Each block contains those transactions that the block creator (called the miner) has accepted in its memory-pool since the previous block.*

In the remaining of this section we give a brief review of standard ADS and of how a blockchain system works. We refer the reader to [6] for a formal treatment of the blockchain topic.

A. Blockchain Technology

A blockchain based solution can be generally seen as a distributed ledger which is maintained by all the peers within a peer-to-peer network. The blockchain stores all the transactions that have ever occurred in the network by grouping them together within blocks and by distributing them via a broadcast protocol. Transactions contain digital assets (such as coins in Bitcoin) which are sent directly from a sender to a receiver without the need of any trusted third party (i.e. banks in cryptocurrencies).

From a data structure perspective, the blockchain can be represented by a public and chronologically ordered list of blocks. Each block contains a list requests known as transactions. Peers willing to participate in the creation of new blocks (commonly known as miners in Bitcoin) compete to add a new block to the blockchain thus imposing their view of the ordered list. In order to prove that their block is the correct one, miners are required to provide a solution to a cryptographic problem (nonce), which makes it unlikely to be solved simultaneously by different peers. However, contemporary block creations can happen, thus causing chains' forks. The result is then to have different peers which are mining on distinct chains. Indeed, due to the longest chain rule, miners can only work on one single blockchain and, in order to avoid waste of computational power, the blockchain to be selected is always the longest one [7]. Blocks on shorter chains are named *orphans* and are eventually discarded in the standard implementation.

Fig. 1. Example of malicious code embedded within orphaned blocks

B. Anomaly Detection Systems

By recognizing and then discarding, sanitizing, or otherwise nullifying outliers input that might exploit security vulnerabilities, ADS often play a central role in many computer security systems [8]. Formally, an ADS can be defined as a couple (M, D), where M is the reference model describing the expected behavior while D is a similarity measure which specifies the actual behavior's deviation from M. Over the years, different ADS approaches have been proposed.

In statistical methods for anomaly detection, the system observes subjects' activities and generates different profile baselines to represent their behavior. Haystack was one of the earliest examples of statistical based ADS [9] which used a range of values that were considered normal and used to detect intrusions. Machine learning based prediction tools can be used to guess the next expected values; thus, they can be used in ADS to build the reference model by predicting normal incoming events, given the current ones. It is then possible to detect anomalies by selecting those next events which are not the ones anticipated by the prediction tools [10], [11], [12]. Machine learning approaches study

algorithms that allow systems to derive general behaviors from data, and which can be either supervised or unsupervised. The first model is created from known clean data while the second is constantly analyzing data and modifying the behavior model without owning a previous model. As an example, Spectrogram is a machine learning based statistical ADS for defense against web-layer code-injection attacks as a network situated sensor that dynamically assembles packets to reconstruct content flows and learns to recognize legitimate web-layer script input [13]. Taint-based techniques have been analyzed in ADS to avoid the false positives common issue. However, their applicability is limited by the need for accurate policies on the use of tainted data. Cavallaro et al. developed a solution which proved to be capable of detecting attack types that have been problematic for taint-based techniques, while significantly cutting down the false positive rate [14].

C. ADS Challenges

ADS usually need to protect the reference model used to detect known and unknown threats [15], [16]. In host-based ADS (H-ADS) this database is stored locally while in network-based ADS (N-ADS) it can be either centralized on a trusted third party or distributed among the peers.

The problem of having centralized data-storage and management systems which are susceptible to breaches becomes even worse in truly distributed networks such as the ones leveraging the blockchain technology [17]. Furthermore, although the blockchain technology prevents several types of unexpected behaviors from malicious or compromised peers on a global scale, it does not eliminate attacks on a local scale. Indeed, local malfunctioning of the blockchain (see Section III) are discarded and cannot be used by others to recognize attack sequences that get repeated over time.

As a result, ADS tools aimed at protecting blockchain-based systems cannot solely rely on information appearing within the mainstream chain but also need to take into account local contexts, and to share such information on a global scale.

III. THREAT MODEL

The solution that we propose in this paper has been designed to be resilient against any class of attacks where a malicious entity can append its own transactions within the blockchain in order to inject malicious code in the system. However, for the sake of simplicity and clarity, we will use the well-known *eclipse attack* [4], [5] to provide an example of these attacks, and how our solution counters them.

Definition 3. *A* fake transaction *is a blockchain transaction used as a side channel to deliver an unexpected message.*

Definition 4. *A* malicious transaction *is a special kind of fake transaction in which the hidden message has the main purpose of attacking on or more peers within the network.*

Definition 5. *A* fake block *is a blockchain block that contains one or more fake/malicious transactions. Fake blocks can be either eventually discarded or accepted as part of the mainstream chain.*

The standard blockchain network (used for Bitcoin) has been designed to be decentralized and independent of any public key infrastructure. Indeed, each node connects to 8 other nodes given in a list that is obtained by querying DNS seeders. In an eclipse attack, the attacker infects a node's list of IP addresses, thus forcing the victim's node in connecting to IP addresses controlled by the attacker. Furthermore, the attacker also aims at filtering and manipulating victim's incoming connections.

One way to execute an eclipse attack, is to repeatedly and rapidly forming unsolicited incoming connections to the victim by attacker's controlled IP addresses and then to wait until the victim restarts [18]. Hence, one challenge for the attacker is to control enough number of IP addresses in order to increase the probability that all the victim's outgoing connections will be directed to IP addresses controlled by him (see Section V). Once the attacker has monopolized all the victim's connections, he can filter incoming blocks and sends his own *fake blocks* containing either *malicious transactions* as it has been done in ZombieCoin [19] (see Fig. 1). For the above attack to succeed, we assume the following attack capabilities:

- **Network Control**: the attacker can manipulate victims' connections in order to control their inbound and outbound traffic thus being able to isolate them. This is a standard requirement for the eclipse attack;
- **Blockchain Control**: the attacker is capable of creating fake blocks which are sent to the victim. Their content is forged ad hoc by the attacker and usually contains a malicious payload.

a) Liveness of the system.: As described in Section V-A, in this paper we assume to have one or multiple powerful attackers who are able to perform eclipse attacks on one or several victims. However, they have to be run in finite time windows. This means that we always assume that the victim(s) will eventually: i) recognize a fork, ii) synchronize with the mainstream blockchain technology and iii) share all the information collected during the eclipse attack with other peers in the network.
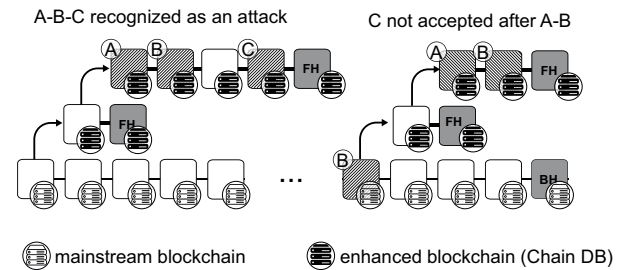
Fig. 2. An overview of BAD being used as a tool to avoid known blockchain-based attacks to be repeated over time

IV. BAD: A BLOCKCHAIN ANOMALY DETECTION SOLUTION

The core idea behind our Blockchain Anomaly Detection (BAD) consists in providing a new decentralized system

based on the blockchain technology which leverages all the information collected from past forks. In blockchain-based applications, forks become more important as the chances to create their evolution for malicious purposes get higher. The main reason behind this is that while attacks may happen only once within a single device, when they are repeated over time against other devices they usually keep behaving in the same way. Hence, by collecting information on previous attacks it could be possible to black list them and to prevent them even within peers that have not been attacked yet.

In our solution, information about orphaned blocks and forks in general are not discarded (as usually done by other blockchain-based applications. Such information is rather collected, enriched and then shared with the other peers in the network. Among all the other information, what is recorded is: i) the time at which the fork has started, ii) the time at which the fork has been detected, and iii) the number and type of malicious transactions (if any). Fig. 2 shows a snippet of what is detected. The longer chain in the middle is the mainstream blockchain (eventually agreed by all the peers). All the other shorter blockchain branches represent forks that happened in the past with *fork head* (FH) being the last blocks accepted before the detection of the fork. Last but not least, as a toy example, Fig. 2 also depicts some malicious transactions labeled as *A*, *B*, and *C* that, as explained in Section III, can be either fake transactions or valid transaction embedding malicious code.

The collection of all the fork-related information and the construction of an enhanced blockchain as the one depicted in Fig. 2 made us able to design BAD as an ADS for blockchain-based applications. In fact, if this enhanced blockchain is eventually agreed by all the peers as already done for the mainstream blockchain, to build our ADS we had just to redefine the couple (M,D) introduced in Section II-B. In BAD, M will be represented by the mainstream blockchain thus describing the expected behavior while all the D will be represented by the forks thus describing similarity measures and their deviation from M. It is then possible to learn that, as shown in Fig. 2, *A-B-C* have been previously labeled as an attack thus preventing them to be re-executed on different peers.

Note that our solution is particularly efficient when the attacker replicates the same malicious transactions at each attack. However, for a more general case we need an additional layer (for instance provided by machine learning or similarity measurement tools) where a sequence of suspicious transactions can be compared with a set of malicious sequences (collected overtime by BAD), in order to be identified as an attack and eventually prevented.

A. Application Stack

The standard blockchain application stack is structured in three layers: shared data, shared protocol and application.

Shared Data Layer: contains the core blockchain and its overlay network. It is still based on the core blockchain protocol but it is used to build networks (called sidechains

[20]) that work in parallel to the Bitcoin blockchain (or to other blockchain-based applications) to perform tasks that Bitcoin cannot solve while still relaying on the same blockchain data structures. Whatever forms this overlay networks take, they all share the connection to the Bitcoin blockchain (or any other blockchain). Such a connection is used to bootstrap their own alternative solution by leveraging the Bitcoin peer-to-peer network;

Shared Protocol Layer: thanks to the blockchain it is now possible to develop decentralized applications with built-in data (transaction payload), validation processes, and transactions that are not controlled by any single entity;

Application Layer: applications built on top of the shared data layer and the shared protocol layer work very similarly to the ones we have nowadays. However, they inherit security, privacy and decentralization properties from the underlying blockchain technology. Hence, peers using these applications will be able to talk with each other and finally reach an agreement which is trusted even though no central authority has been used.

BAD has been designed to be an ad hoc solution (i.e. a blockchain based application plug-in or a third party service) rather than to be embedded within Bitcoin or any other specific application. The reason for such approach is that BAD does not rely on a specific blockchain and can be set up to detect attacks on any blockchain application. Indeed, the core Bitcoin elements such as the *wallet* and the *miner* do not contain BAD elements but just interact with them. Here, we describe each BAD's module and how it interacts with standard blockchain application:

- Transaction Filter (Tx Filter): this module intercepts standard blockchain messages and forward them to both the *miner* and the *chain manager*, thus not interrupting the standard blockchain protocol. Furthermore, it allows the collection of transactions meta-data;
- Chain Manager: it is responsible to build our enhanced blockchain which, among the other elements, contains information on the forks that have been generated so far. It receives messages from the *transaction filter* and retrieves additional missing information from the *chain database* which finally stores our *enhanced blockchains* (i.e. blockchains built starting from the mainstream chain but with additional information on forks, see Fig. 2 for an example). Last but not least, the chain manager notifies the pattern inspector if the enhanced blockchain has been updated and some threat analysis has to be done;
- Pattern Inspector: leverages the *chain database* to detect unexpected behaviors. The inspection on the forks can be done with any approach ranging from signatures to heuristic static analysis and is aimed at finding sequences of transactions which were found to be dangerous in the past;
- Threat Detector: starting from the anomalies found by the *pattern inspector* this module performs root-cause analysis by exploiting past blockchain activities (past blocks and transactions within them) to roll back all the

operations done by the victim. Afterwards, all the attack information are collected within a *threat database* which contains the information on all the malicious patterns within the blockchain that have to be considered malicious (depending on the security policy being adopted).

Fig. 3 shows a simple implementation of BAD's threat database. Here, recalling the toy example given in Fig. 2 in which three blocks (*A*,*B* and *C*) were found to contain malicious code, we show how this information is collected and later shared with other peers. BAD's threat database is basically a dynamic (i.e. not sized) array of array in which S_i represents the i-th attack sequence detected while T_i represents the hash of the i-th transaction which was found to contain part of the payload's attack sequence.

Information used to fill the threat database is provided by the pattern inspector and used by the transaction filter to avoid the repeating of known attacks. The filtering process is accomplished by the BAD's transaction filter module each time a new block is received and its overhead has been analyzed in Section VI-B.
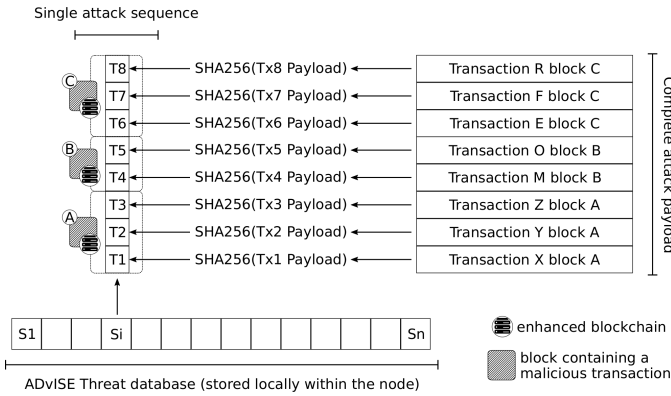

Fig. 3. Implementation of the threat database in BAD

V. EXPERIMENTAL TEST

In this section we show how BAD has been used in our experimental platform to prevent attacks across different networks thanks to the information collected from forks. The goal of this experiment has been to detect forks on a given peer that were caused by an eclipse attack and then to share this information with other peers in order to build a reference model aimed at detecting future occurrences of the same attack.

A. Test Bed

For the sake of simplicity and clarity, we describe here a toy environment which has been deployed and run over our real systems. Such environment is composed by two domains, A and B, that represent two separated private IP networks with a router between them. In domain A (*B respectively*), we have deployed two full nodes and one lightweight client. On the one hand, the two full nodes[6] *A1* (*B1* in B) and *A3* (*B3* in B)

[6]https://bitcoin.org/en/full-node#what-is-a-full-node

have been executed each on a Virtual Machine with 4 GB of RAM and Linux Ubuntu 16.04. Both have been executed in regtest experimental mode[7] (a mode in which local testing environment can be created with instantaneous on-demand block generations and digital assets creation without any real value). *A3* (*B3* in B) is controlled by a malicious user. On the other hand, for the wallet we have used a Bitcoin Java BitcoinJ client (version 0.14.3)[8] running on a 4GB RAM PC with Windows 8.1 installed. This wallet acts as the victim in the eclipse attack and is labeled as *A2* (*B2* in B). *A1* (*B1* in B) and *A3* (*B3* in B) are connected to each other, which means they can exchange blocks between them and agree on the longest chain (to do so we used on each node the following command: `bitcoin-cli -regtest addnode IPaddressoftheothernode add`). Nodes in domains A and B are initially synchronized on the same blockchain (this block chain is generated using the command `bitcoin-cli -regtest generate X`[9]).

B. Attack Detection and Prevention

Our attack consists of eclipsing *A2* and forcing it to only communicate with *A3* which is controlled by the attacker. Furthermore, *A3* does not exchange any block with *A1* to avoid being detected by other nodes in the same domain (to do so we used the following command on node *A3*: `bitcoin-cli -regtest addnode IPaddressofA1 remove`). Then *A3* sends to *A2* three new blocks which contain fake or malicious transactions (to do so, we have used the command: `bitcoin-cli -regtest generate 3`) and that get accepted by *A2* as connected to the previous blockchain header and as representing the longest chain. Assuming that the three new blocks created by *A3* contained a malicious payload, at this point we have *A2* that has been attacked. After the attack and once the eclipse is removed and all the connections re-established, *A2* receives a longer chain from peers within the domain A and accepts it but without discarding the information of those three malicious blocks. Furthermore, the information regarding those potentially malicious blocks are sent in broadcast thus reaching other peers (in our toy example represented by the node *B2*).

As a second step we execute the same eclipse attack in the domain B. However this time we have leveraged BAD and the information gathered so far (sent to *B2* by *A2*) to prevent the attack from succeed again. Hence, we first eclipse *B2* by forcing it to only connect to *B3* that is actually controlled by a malicious user. At this point, as previously done in domain A, *B3* generates three malicious blocks which contains, among the others, the same three malicious transactions used in the attack against *A3*. However, unlike *A3*, *B3* has now BAD which is running and checking for the upcoming transactions. As also shown in Fig. 2, BAD detects blocks that are different but containing the same transactions (or a subset) in the same order. The final result is then the prevention of the complete

[7]https://bitcoin.org/en/developer-examples#regtest-mode
[8]https://bitcoinj.github.io
[9]to create X blocks in the blockchain

attack as only a small subset of the malicious transactions is accepted (in our example by *A2*) before BAD recognizes them as malicious. As done by *A2*, also *B2* will share the information with other peers once it realizes that it was previously mining and elaborating on blocks that belong to a fork.

VI. OVERHEAD ANALYSIS

The core elements introduced by BAD on the classical Bitcoin protocol are the broadcast of brand new forks, and their orphaned blocks, as well as the detection of malicious transactions on new received blocks. In this section, we analyze the introduced bandwidth overhead to show that our solution is scalable and thus deployable within the standard Bitcoin network. In particular, the results of our analysis show that our system has minimal bandwidth consumption in comparison with the one consumed by standard nodes.

A. Bandwidth overhead

We have analyzed the overhead introduced by our solution in the worst-case scenario, i.e. the whole global Bitcoin fork activity to affect one single node named *NX*. Our overhead is then defined as the amount of bandwidth that *NX* consumes due to the fork broadcast introduced in BAD. To this aim, and to be rooted on real data, we have considered the maximum number of orphaned blocks discarded by the Bitcoin community during last year. We are interested in the total number of orphaned blocks because it includes those used to attack the victims (see Section III). Furthermore, we assume this number to have a small variance since a smart adversary, to stay hidden in the network, would not create an anomalous number of orphaned blocks. A more abstract, and less constrained, analysis is given in Section VI-B.

To analyze BAD's overhead, we have then designed the p2p network surrounding our *NX* node. By construction, nodes in the Bitcoin network create a random graph—randomness being due to the selection of outgoing connections. In the vanilla Bitcoin protocol, each node attempts to keep a minimum of 8 outgoing connections at all time. However, it has been observed that, on average, a Bitcoin node has 32 outgoing connections [21]. Furthermore, the total number of orphaned blocks discarded during last year (2016)[10] was 141 with a maximum block size of 0.993201 MB. As such, in our worst-case scenario, we consider all those 141 orphaned blocks (of the maximum size) to be collected and re-distributed in broadcast by *NX*. To broadcast all these blocks with their transactions, *NX* would send broadcast messages to its neighbors, which sum up to the global size of $32 \times 0.993201 \times 141 = 4.481$ GB per year. It is important to highlight that the total number of orphaned blocks is independent of the node's bandwidth. Hence, our worst-case scenario can be applied to any node: from lightweight SVP clients to relay nodes or miners. Furthermore, the total node/month upload bandwidth could vary according to nodes capabilities and ISP resources,

it could start with 150 GB/month (which is the minimum recommended upload bandwidth to run a Bitcoin core [11]) and reach values up to 300 GB/month and more.
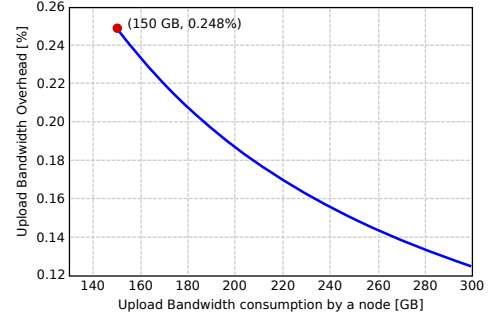

Fig. 4. Overhead introduced by the system as a function of the bandwidth consumption of a node

Fig. 4 plots the result of our BAD's overhead analysis which is approximated by the following formula:

$$Ovh = \frac{\text{BAD data broadcast (per year)}}{\text{total data exchanged (per year)}} = \frac{4.481}{m \times 12} \quad (1)$$

where m is the average bandwidth consumption of a node per month. Fig. 4 shows the maximum overhead introduced in the case of 150 GB of upload bandwidth consumption which is of 0.248%. The results is an overhead on the bandwidth of only 0.248%. This finally proves BAD to be smoothly deployable in the standard Bitcoin network.

B. Complexity

In the previous section we have analyzed BAD's overhead in the worst case, i.e. with an attacker using Bitcoin's forks to spread malicious code. However, statistics and real data used for such analysis refer to natural forks appeared over time in the network due to its delay.

Here, we analyze a more general use case, the one in which the attacker creates as many blocks as needed (thus also generating more forks in the system). The result, as shown in the remaining of this section, is that BAD's bandwidth overhead, in the worst case, can only by proportional (up to a constant factor in real cases) to the size k of our Threat Database $\mathcal{T}$. Let $S_1, \ldots, S_k$ be the malicious transaction sequences of k attacks detected and stored in $\mathcal{T}$. Each sequence S_i has a length of ℓ_i transactions injected by the attacker to complete attack i. We call *partial sequence* (PS_i, j) a subsequence of S_i starting from the first transaction and ending with the j-th transaction of S_i. Note that (PS_i, ℓ_i) represents the full attack i. For each i we can have at most $\ell_i - 1$ incomplete sequences. Each node in the network maintains a set U of partial transactions. Given that $H(t)$ is the hash of a transaction t, every time t is analyzed by a node, BAD performs two actions:

1) If there is a partial sequence $(PS_i, j) \in U$ such that $(PS_i, j)||H(t) = (PS_i, j+1)$, we replace (PS_i, j) with

---

[10]https://blockchain.info/charts/n-orphaned-blocks

[11] https://bitcoin.org/en/bitcoin-core/features/requirements

$(PS_i, j+1)$ in U. Here $||$ is the standard concatenation function.

2) If $H(t)$ represents the first block of a sequence S_i, then we insert $(PS_i, 1)$ into U.

Finally, BAD checks if there is a (PS_i, ℓ_i) in U and, in that case, discards the transaction t. While the correctness of this approach follows from the construction, the analysis requires some more effort. Note that in the worst case (which is when every transaction of every attack has the exact same hash), every transaction will create a new partial sequence $(PS_i, 1), \forall i$, plus it will increase at most $\ell_i - 1$ existing partial sequences in U for each i. This translates in the following number of steps:

$$W(t) = k + \sum_{i=1}^{k}(\ell_i - 1) = \sum_{i=1}^{k} \ell_i$$

Since (in a real scenario) each attack sequence is no longer than a constant c of transactions, the total work $W(t)$ for a given transaction will be at most $c \cdot k = O(k)$ where $k = |\mathcal{T}|$. In case the size of $\mathcal{T}$ grows very quickly, pruning techniques can be adopted to regulated its dimension. For example, hold or infrequent attacks could be discarded in favor of the newly discovered ones.

VII. Discussion

In the standard blockchain implementation (used in Bitcoin), each victim possesses two tables: *tried* and *new*. On the one hand, the *tried* table contains 64 buckets each of which stores 64 IP addresses. These IP addresses refer to known peers which have successfully established connections with the victim in the past. On the other hand, the *new table* contains 256 buckets each of which stores 64 IP addresses. These addresses refer to peers with whom the victim has not yet initiated a successful connection. In blockchain applications, each peer randomly selects 8 IP addresses (i.e. 8 *tried* neighbors) among all the addresses stored in the *tried* table. As such, in order to prevent (deterministically) the victim from sending in broadcast the fork information collected during the attack, the attacker has to eclipse either all the 4096 peers contained in the *tried* table or the 8 peers which are randomly selected. This is a very complex attack since, in order to eclipse the victim's *tried* neighbors, the attacker must have access to the victim's tried table. This access can be obtained either by compromising the victim's device or by monitoring the network and eavesdropping the victim's connections and behavior.

It is also important to highlight that the block propagation time is approximately in the interval $[0s \dots 60s]$ with the percentage of nodes not receiving this block decreasing exponentially when time increases [21]. Hence, once the *tried* neighbors IP addresses have been identified, the attacker has to succeed in the eclipse attack within 60 seconds. This demands strong processing power if the attacker is able to control 50% of the addresses in the victim's *tried* table with a probability to eclipse the victim in 1 hour time investment close to zero.

The above two considerations, together with the fact that the *tried* and *new* tables are dynamically updated over time, make a total eclipse of the victim's surrounding network very unlikely to happen. However, this only relates to Bitcoin. Indeed, other blockchain based applications might use different peer-to-peer protocols, mining algorithms and consensus schemes thus being more exposed to eclipse attacks. For all these blockchains BAD can play a crucial role in the mitigation of eclipse attacks.

VIII. Conclusion

In this paper we have proposed BAD: the first Blockchain Anomaly Detection solution. In particular, BAD allows to detect anomalous transactions and to prevent them from being further spread. We have provided a thorough description of BAD, as well as a preliminary validation of its performances over a toy network and a comprehensive analysis of BAD's complexity in the presence of an attacker who can create forks at will. Next steps are aiming at a fully fledged implementation of BAD, followed by a comprehensive testing on our internal corporate network.

An interesting open research question is related to which machine learning techniques should be included in the BAD architecture in order to build strong prediction tools for heterogeneous malicious transactions.

References

[1] R. Di Pietro and L. V. Mancini, *Intrusion Detection Systems*, 1st ed. Springer Publishing Company, Incorporated, 2008.

[2] H. Javitz and A. Valdes, "The SRI IDES statistical anomaly detector," in *Proceedings. 1991 IEEE Computer Society Symposium on Research in Security and Privacy*. Institute of Electrical & Electronics Engineers (IEEE), 1991.

[3] A. Zohar, "Bitcoin," *Communications of the ACM*, vol. 58, no. 9, pp. 104–113, aug 2015.

[4] A. Singh, T. wan johnny Ngan, P. Druschel, and D. S. Wallach, "Eclipse attacks on overlay networks: Threats and defenses," in *In IEEE INFOCOM*, 2006.

[5] K. Nayak, S. Kumar, A. Miller, and E. Shi, "Stubborn mining: Generalizing selfish mining and combining with an eclipse attack," in *2016 IEEE European Symposium on Security and Privacy (EuroS P)*, March 2016, pp. 305–320.

[6] R. Wattenhofer, *The Science of the Blockchain*. CreateSpace Independent Publishing Platform, 2016.

[7] I. Eyal and E. G. Sirer, *Majority Is Not Enough: Bitcoin Mining Is Vulnerable*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, pp. 436–454.

[8] B. Juba, C. Musco, F. Long, S. Sidiroglou-Douskos, and M. C. Rinard, "Principled sampling for anomaly detection." in *NDSS*. The Internet Society, 2015.

[9] S. Smaha, "Haystack: an intrusion detection system," in *[Proceedings 1988] Fourth Aerospace Computer Security Applications*. Institute of Electrical & Electronics Engineers (IEEE), 1988.

[10] R. Halavati, S. B. Shouraki, P. Esfandiar, and S. Lotfi, "Rule based classifier generation using symbiotic evolutionary algorithm," in *19th IEEE International Conference on Tools with Artificial Intelligence(ICTAI 2007)*. Institute of Electrical & Electronics Engineers (IEEE), oct 2007.

[11] M. E. Cintra, H. de Camargo, E. R. Hruschka, and M. do Carmo Nicoletti, "Fuzzy rule base generation through genetic algorithms and bayesian classifiers a comparative approach," in *Seventh International Conference on Intelligent Systems Design and Applications (ISDA 2007)*. Institute of Electrical & Electronics Engineers (IEEE), oct 2007.

[12] E. H. Cardenas and H. A. Camargo, "Multiobjective genetic generation of fuzzy classifiers using the iterative rule learning," in *2012 IEEE International Conference on Fuzzy Systems*. Institute of Electrical & Electronics Engineers (IEEE), jun 2012.

[13] Y. Song, A. D. Keromytis, and S. J. Stolfo, "Spectrogram: A mixture-of-markov-chains model for anomaly detection in web traffic." in *NDSS*. The Internet Society, 2009.

[14] L. Cavallaro and R. Sekar, *Taint-Enhanced Anomaly Detection*. Springer Berlin Heidelberg, 2011, pp. 160–174.

[15] S. S. Murtaza, A. Hamou-Lhadj, W. Khreich, and M. Couture, "Total ADS: Automated software anomaly detection system," in *2014 IEEE 14th International Working Conference on Source Code Analysis and Manipulation*. Institute of Electrical & Electronics Engineers (IEEE), sep 2014.

[16] J. Kim and H. S. Kim, "PBAD: Perception-based anomaly detection system for cloud datacenters," in *2015 IEEE 8th International Conference on Cloud Computing*. Institute of Electrical & Electronics Engineers (IEEE), jun 2015.

[17] J. J. Xu, "Are blockchains immune to all malicious attacks?" *Financial Innovation*, vol. 2, no. 1, dec 2016.

[18] E. Heilman, A. Kendler, A. Zohar, and S. Goldberg, "Eclipse attacks on bitcoin's peer-to-peer network," in *Proceedings of the 24th USENIX Conference on Security Symposium*, ser. SEC'15. Berkeley, CA, USA: USENIX Association, 2015, pp. 129–144.

[19] S. T. Ali, P. McCorry, P. H.-J. Lee, and F. Hao, *ZombieCoin: Powering Next-Generation Botnets with Bitcoin*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2015, pp. 34–48.

[20] A. Back, M. Corallo, L. Dashjr, M. Friedenbach, G. Maxwell, A. Miller, A. Poelstra, J. Timón, and P. Wuille, "Enabling blockchain innovations with pegged sidechains," *White Paper - Semantic Scholar*, 2014.

[21] C. Decker and R. Wattenhofer, "Information propagation in the bitcoin network," in *IEEE P2P 2013 Proceedings*, Sept 2013, pp. 1–10.