2018 15th International ISC (Iranian Society of Cryptology) Conference on Information Security and Cryptology (ISCISC)
August 28-29, 2018; Shahid Rajaee Teacher Training University - Tehran, Iran

BotcoinTrap: Detection of Bitcoin Miner Botnet Using Host Based Approach

Atefeh Zareh
Department of Computer Engineering and Information Technology
Amirkabir University of Technology
Tehran, Iran
zareh@aut.ac.ir

Hamid Reza Shahriari
Department of Computer Engineering and Information Technology
Amirkabir University of Technology
Tehran, Iran
shahriari@aut.ac.ir

***Abstract*—Bitcoin is one of the most successful cryptocurrencies. Many people invest money on creating new Bitcoins because of Bitcoin's market increase. They actually buy hardware and power to participate in Bitcoin mining. The market value of Bitcoin has also absorbed cybercriminals. They steal the process cycles from victims' machines and use them in mining activities by malware programs. There have been several security reports about these types of malicious activities. Although there are methods to detect botnets, to the best of our knowledge, none of non-commercial and published papers present detection method for these types. In this paper, we present Botcointrap, a novel approach to identify Bitcoin miner botnets (called Botcoin) based on dynamic analysis of executable binary files. This method benefits from a parameter value that all Botcoins must use across their computations and detect them in the lowest level of execution; therefore, our method can be used to overcome weaknesses of many other approaches. Our evaluation shows that the proposed approach efficiently identifies all simulated Botcoins.**

Keywords—Bitcoin, Blockchain, Bitcoin Mining, Miner, Malware, Botnet, Botcoin, Dynamic analysis, BotcoinTrap

I. INTRODUCTION

Botnet is a tool to commit online crimes and is the greatest threat to internet infrastructure. Botnet is a network of compromised end-user PCs (bots) infected with a malware (malicious software), and are controlled by a Botmaster. This network of victim's machines with a C&C (command and control) channel provides a complete toolset for different illegal activities. The goal of this network is financial gain for its botmaster with a low risk of being caught. These activities are very diverse, containing DDoS attack, pay-per-install, fake anti-virus, spam, click fraud, harvesting sensitive information (such as account login information and credit card data), and Bitcoin mining [1].

Botmaster decides which activity set will execute on the victim's machine based on which time table. Botmaster then enforces all plans to bots by command and control channel. Botmaster decisions obviously are based on maximizing the outcome and minimizing the risk of botnet detection.

Bitcoin mining has so many appealing features for botmasters to involve. We call these types of botnets that mine Bitcoin as Botcoin [2]. Despite of some other activities (for example, sending spam), mining requires little botmaster investment. Bitcoin's market increase is another very strong motivation for botmasters to make use of Bitcoin mining. Because of state-space search essential of mining, this activity is a marvelous candidate to distribute between bots. On the other hand, Bitcoin mining is a fully computational process and using mining solely on the victim's machine increases the risk of detection. Considering all these things, botmaster plans the bots.

Numerous technical reports and tools have emerged for botnet detection until now. As far as we know, none of these methods unveil all details of the botnet at once. All detection methods address a part of the big jigsaw puzzle. Botnet detection is classified into different aspects; bot detection, C&C detection and botmaster detection [3].

In this paper, we present a novel approach to detect Bitcoin miner bots. We show that several botnet detection methods can be bypassed by miner malware. We introduce a different behavior of Botcoins that is used to detect this type of malware. This method detects malicious samples effectively in our designed experiments.

The rest of this paper is organized as follows. Section II surveys related work. Section III provides the technical background on Bitcoin, Bitcoin mining and blockchain necessary of the remainder of the paper. In Section IV, we describe our approach, called BotcoinTrap. We present our results in Section V, while Section VI concludes this paper.

II. RELATED WORK

Related researches to our approach are considered in two main areas. The first investigates some botnet detection techniques that can be used in general for all types of botnets and is not for a specific type of botnet [3], [4], [5], [6]. We study the limitations and issues of all these methods to cover the majority of them in our approach. We present the advantage of our method comparing with these general methods in the remaining of the paper.

The second area that is related to our investigation contains some researches about Bitcoin miner malware in several aspects. Although to the best of our knowledge, none of non-commercial and published researches address this new malicious behavior to detect botnets, some studies have been done around this issue. Huang et al. Focus on dynamics of Bitcoin mining malware and conclude the count of Bitcoins that a number of mining botnets have made [1]. Technical details of ZeroAccess botnet are explored and one of plugins that this botnet download to perform on the victim's machine is Bitcoin mining plugin [7]. Also, Plohmann and Gerhards-Padilla characterize Miner Botnet in technical level [8]. Güring and Grigg have

978-1-5386-7582-3/18/$31.00 ©2018 IEEE

an investigation to show a threat of Botcoins in the economic perspective, and it shows the importance of detecting and stopping these types of defective malware [9].

III. Background

Nakamoto (a pseudonym) unveiled a system for an electronic money (named Bitcoin) and its electronic transactions without relying on trust [10]. Bitcoin is a crypto currency based on p2p network. There is no need for a central authority to create money, validate and perform transactions, recording balances and any other related activities.

Creating new Bitcoins, validating new transactions and persisting valid transactions in the system are done with Bitcoin miners. A miner is any person or group of people that distribute in a specific activity in Bitcoin network, called Bitcoin mining. A miner records transactions in a global ledger named the Bitcoin blockchain. The blockchain is a distributed data structure that allows all peers to access all transactions in the Bitcoin network and consequently, know everybody's Bitcoin balance.

Everybody can have several accounts on Bitcoin network. Each account has a pair of public and private key. Each person is known with their public key and could sign with his or her private key and hence, the privacy of senders and receivers in a transaction is protected. All transactions are signed with the private key of the sender of Bitcoin and then are broadcasted on Bitcoin network.

Miners gather some recent transactions and validate them according to the Bitcoin standards and sender's balance. A Miner puts valid transactions in a block structure and also puts the previous block hash in the block structure to chain this block to the latest previous block. Then the miner has to perform a hard-mathematical computation in this block to find a proper 32-bit data (called nonce) that is matched for this block.

This computation is based on the SHA-256 hashing algorithm. In fact, each miner must find a proper random number of nonce such that putting this number in its place in working block, results a proper SHA-256 hash value of that block. The condition of the hash's suitability is a certain count of zero bits at the beginning of it. This count is the difficulty factor of block mining and is determined by Bitcoin network dynamically.

Each miner broadcasts the whole block to everyone over the p2p Bitcoin network after finding the proper nonce. Other miners check the validity of this received block as soon as receiving the last mined block. Validating a block means that the hash of it must be less than a threshold. If the received block is valid, the miners stop working with their current block because the previous block hash is not valid anymore and a newer one exists. They update their working block and start to find proper nonce again in the new underwork block.

Only the blocks in the longest blockchain of Bitcoin are valid. Each mined block contains some new created Bitcoins that are owned with its miner. This prize and also transaction's fee is the miner's incentive to participate in the Bitcoin network to support this network by creating new Bitcoins, validating and recording transactions in the blockchain.

TABLE I. Fields of a Bitcoin Block Header

Field	Purpose	Update time	Size (Byte)
Version	The version of this block	When the miner software will be updated	4
Previous block hash	Hash of previous block	When a new block is broadcasted on Bitcoin network	32
Merkel Root hash	Hash of block transactions	Adding one accepted transaction in a block	32
Time	Current timestamp (in second) after 1970-01-01T00:00 UTC	After one second	4
Bits	Amount of current proof of work hardship	Adjusting hardship of proof of work	4
Nonce	32-bit random number such that meet the hash criteria of the block	Can be different for every block	4

Because of high difficulty of mining, people participate in some special groups (mining pools) and share their process power and then the resulted income. This is where a botnet comes in. A botnet is a mining pool and the bots are the members of this pool. Botmaster distributes the calculation between bots and expects each bot to send back the result after finding it. Table I shows all fields of a Bitcoin block header.

IV. BotcoinTrap

In this section, we propose a method for detecting Botcoin. This method is based on dynamic analysis of the instruction trace and can be applied to any suspicious executable binary files. In this method, we found a constant parameter value that all of Botcoins must use across their malicious computations. We detect this malware in the lowest level of execution (assembly language level).

Botnets have a life cycle of seven steps: spread and infection, secondary injection, hiding and securing, rallying/bootstrapping, command and control, attack, remove and release. This kind of malware that we study in this research does a straightforward and repetitive functionality on a victim's machine in the attack phase. This functionality despite of other malware activities in other phases has limited variety; hence Botcoin detection by this feature has stronger accuracy.

The purpose of Bitcoin Miner is to find a proper Nonce for the Bitcoin block header such that the result of SHA-256 in the block header starts with a predefined count of zeros. The mining process is essentially a state-space search, and there are no tricky methods to find a proper nonce by less computation.

In the whole process of Bitcoin Mining, the current Bitcoin block header is the most important data for detecting Botcoins. This data is too big to locate in CPU registers and malware has to read this data from memory repeatedly to calculate the hash of the block. This repeated reading from memory helps us to trap the malware. In the next section, we describe block header in more details

A. *Forecasting specific part of the next Bitcoin Block header*

We want to forecast maximum data content that Botcoins provide in a victim's machine and use them in digesting function. Table I shows all fields of a Bitcoin block header. We use first two fields of a block header for this purpose, and shows in the remaining of this section that both of them have acceptable change rate.

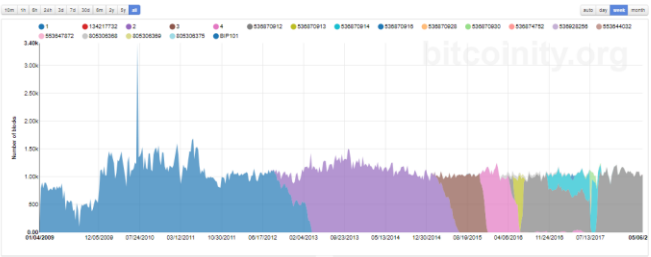


Fig. 1. Changes in the block version of the Bitcoin blockchain [11]

Fig. 1 is a stacked area graph, illustrating the number of blocks over time. Each color represents a Bitcoin blockchain version (First field of Table I) and the current block version value is 536870912. Based on Fig. 1, all miners rapidly update their block version to the most recent released version of block version.

The motivation of miners to this fast update is that they want their mined blocks to be accepted by other miners and can achieve their financial income from their mining activity as a result. As mentioned in Bitcoin Improvement Proposals, Miners are strongly recommended to upgrade to the newest version of blocks. When 95% of the past 1000 blocks have the newest version, blocks with older versions become orphaned and invalid entirely [12] [13, p. 0062].

Second field in Table I is 'previous block hash' that changes almost for every new released block. Bitcoin network adjusts the difficulty of creating new Bitcoins to guarantee the average ten minutes time for each new block release. All new blocks are announced on Bitcoin network, and thus we access to the previous block hash by listening to Bitcoin network. Because of the low change rate of previous block hash, we utilize this value in our detection approach.

Other fields in Table I are not the focus of this paper because their values are specified by each miner and are not predictable at all and we cannot exploit them for detecting the Botcoins. For this reason, we use only the first two fields of the block header in remaining of research (version and previous block hash), first one from Bitcoin community and the second one from the Bitcoin P2P network and they are overall 36 bytes. In the remaining of the paper, we called these 36 bytes data as common header data.

One advantage of this novel approach for malware detection is that these 36 bytes are common for all Blocks in any time that is mined either by legal or illegal miners (Botcoins). This data is completely predictable and can be a tricky key to reveal this malicious behavior.

B. *The Architecture of BotcoinTrap*

In the previous section, we showed that it is possible to predict 36 bytes of Botcoin data. In this section, we use this data to detect Botcoin. 36 bytes of data are too large to store in CPU data registers in regular common CPU architectures, and CPU cannot operate SHA-256 function by remaining CPU registers. For this reason, Botcoin has to read this data from memory repeatedly to calculate the hash of a block. This repeated reading from memory is the basis of our method.

We can assume at least two architectures for our Botcoin detector method according to time of detection. One of them is called synchronous and the other is asynchronous architecture. In synchronous architecture, we instrument the executable file and monitor its memory read accesses and simultaneously calculate the common header data and compare these two data and then notify the client of detecting Botcoin by meaningful repeated equal results.

The biggest disadvantage of synchronous architecture in this case is that the execution of suspicious executable file become too slow and it can effect on the origin behavior of the Botcoin.

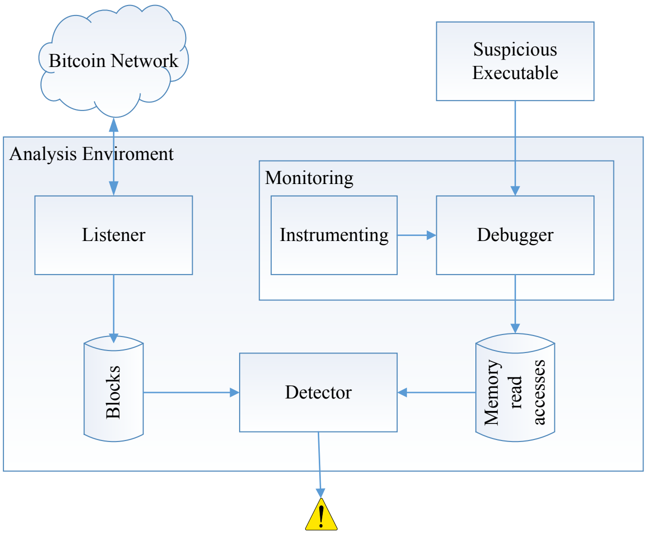


Fig. 2. Asynchronous architecture of Botcoin detector

As a consequence of synchronous architecture drawbacks, we immediately propose the asynchronous architecture. The asynchronous architecture is illustrated in Fig. 2. The main difference between asynchronous architecture and previous one is that the listener component logs the hashes of the new blocks instead of notifying them to instrumented debugger. Instrumented debugger also logs all memory read accesses without any extra process on them. Third component named detector receives these two logs and decides based on them about malicious behavior of mining. The cause of this architecture naming as asynchronous is according to executing the suspicious executable and detecting process asynchronously. In the remaining of this article we only describe asynchronous architecture in detail.

C. Sequence of process

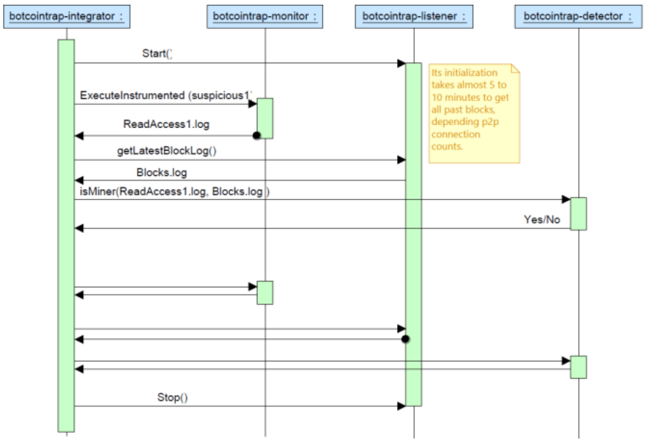

botcointrap-integrator :
botcointrap-monitor :
botcointrap-listener :
botcointrap-detector :
Start(
ExecuteInstrumented (suspicious1
ReadAccess1.log
getLatestBlockLog()
Blocks.log
isMiner(ReadAccess1.log, Blocks.log)
Yes/No
Stop()
Its initialization takes almost 5 to 10 minutes to get all past blocks, depending p2p connection counts.

Fig. 3. The sequence diagram of the asynchronous detector

The sequence diagram of asynchronous approach is illustrated in Fig. 3. The three main components of monitor, listener and detector in Fig. 3 are shown as three lifelines in Fig. 3. One more lifeline named integrator is exists in sequence diagram. This component has responsibility for integrating all other components and achieves the desired result. Integrator may be an operator or a software component.

Integrator component starts the listener component. Listener component listens to the Bitcoin network and records each new broadcasted block with timestamp in a file named Blocks.log until integrator stop this component. Then integrator sends command to monitor component to instrument a given suspicious executable file or instrument a given process. After a while that is defined by the integrator, the monitor stops its monitoring and then returns the result file called ReadAccess.log to the integrator. This file contains all memory read accesses that suspicious application has in this period of time. Integrator sends these mentioned files to detector component and receives the detection result. Integrator can repeat these processes as wish.

D. Detection Algorithm

In this section, we describe the internal logic of detector component. Our detection algorithm has inspired from The Venus Flytrap. It is a plant that catches its prey with a special trapping leaf. Each of these special leaves consists of a pair of fatal lobes hinged at the midrib. Each lobe has tree hairlike trichomes on the upper surface of it. This plant snap lobes shut when is stimulated with a prey. It is vital for the plant to detect the prey correctly. Closing the lobes without any real prey is costly and on the other hand, if the plant doesn't detect the existence of prey on the trap structure, it loses the potential food. This plant uses the most heuristic method to survive. When an insect or spider crawling along the leaf, contacts a hair, the trap prepares to close, snapping shut only if another contact occurs within approximately twenty seconds of the first strike.

```
/*
Data structures:

blocksFile (time, hash)
ReadFile (time, count, content)
*/

BlackList = contains 36 first parts of the current
Bitcoin block

main (){
  ReadFile = getMonitorLog()
  BlockFile = getListenerLog()
  Update BlackList with getNextLine of readLine
  detect()
}

detect(){
  latestHashOccurence = null
  while(is not end of readFile){
    l1 = getNextHashLine(readFile)
    if (l1 is near enough to latestHashOccurence
(M)){
      Alarm mining activity and do any necessary
action and exit
    }else{
      latestHashOccurence = l1 // forgot previous
latestHashOccurence
    }
  }
  Notify end of detection without mining activity
detection
}
lineNumber getNextHashLine(readFile){
  /*
  * current Bitcoin Block as black content,
  * and any line of ReadFile that is equal
  * to each of these 36, as BlackLine.
  */
  latestBlackLine = null
  while(it is not end of ReadFile){
    content = getNextLine (ReadFile)
    if (isInBlackList(content)){
      read N-1 next lines
      if (these N bytes cover black list){
        return content line as hash occurrence.
      }
    }
  }
}
```

Fig. 4. Detector algorithm

As we see in the previous section, detector receives two series of logs from listener and monitoring component and has the responsibility of detecting mining activity. These two types of logs have a common timestamp data, that is used to join these two files. The pseudocode of detector is presented in Fig. 4.

The detector component receives one file from the monitoring component (named readFile) and another file from the listener component (named blockLog). ReadFile contains every memory read access of suspicious file with the format of (time, count, content) and blockLog containing headers hashes of broadcasted blocks from the Bitcoin network after executing the detector and has a simple structure of (time, hash).

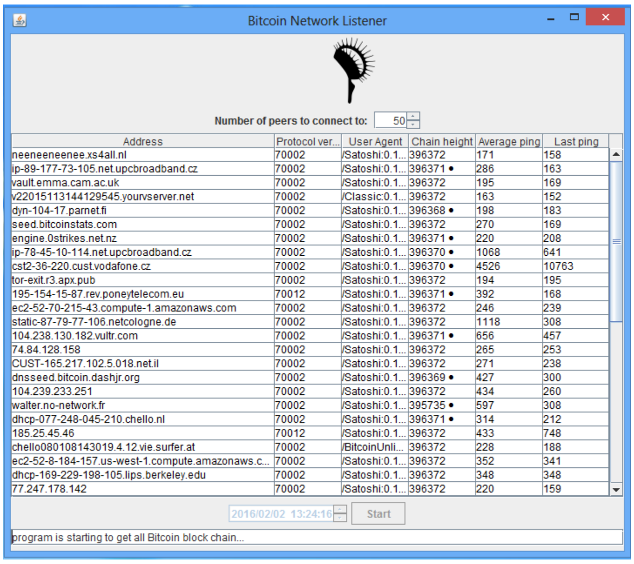
36 bytes of data that all miners containing Botcoins work with repeatedly is known data. We investigate the content of memory read access file and if these 36 bytes of data are in an N-sized data of this file, it is announced as a hash occurrence. Hash occurrence is like hitting one of hairlike trichomes in the case of Venus Flytrap.

Then we define an M-sized window of read access data, and expect to see second hash occurrence event in the same window. If two hash occurrences are done on an M-sized window, it is interpreted as a Bitcoin mining occurrence and otherwise we reset the state of the detector as waiting for the first hash occurrence. M-sized window is like the timer in case of Venus Flytrap.

E. BotcoinTrap Implementation

We implement the asynchronous architecture of Botcoin detector .In this section, we describe the details of the implementation containing the tools that are utilized in implementing, programming languages that are used in each part of project, user interfaces of the program and the simple manual of it and the necessary command lines.

Monitoring component is utilized PIN for debugging the suspicious executable file or process. PIN is a dynamic binary instrumentation framework and hence, there is no need to recompile the application [14]. We implement the specific instrument in C++ programming language for recording all memory read accesses in a file and providing the output of Monitoring component.



Number of peers to connect to: 50

Address	Protocol ver...	User Agent	Chain height	Average ping	Last ping
neeneeneenee.xs4all.nl	70002	/Satoshi:0.1...	396372	171	158
ip-89-177-73-105.net.upcbroadband.cz	70002	/Satoshi:0.1...	396371 •	286	163
vault.emma.cam.ac.uk	70002	/Satoshi:0.1...	396372	195	169
v220151113144129545.yourvserver.net	70002	/Classic:0.1...	396372	163	152
dyn-104-17.parnet.fi	70002	/Satoshi:0.1...	396368 •	198	183
seed.bitcoinstats.com	70002	/Satoshi:0.1...	396372	270	169
engine.0strikes.net.nz	70002	/Satoshi:0.1...	396371 •	220	208
ip-78-45-10-114.net.upcbroadband.cz	70002	/Satoshi:0.1...	396370 •	1068	641
cst2-36-220.cust.vodafone.cz	70002	/Satoshi:0.1...	396370 •	4526	10763
tor-exit.r3.apx.pub	70002	/Satoshi:0.1...	396372	194	195
195-154-15-87.rev.poneytelecom.eu	70012	/Satoshi:0.1...	396371 •	392	168
ec2-52-70-215-43.compute-1.amazonaws.com	70002	/Satoshi:0.1...	396372	246	239
static-87-79-77-106.netcologne.de	70002	/Satoshi:0.1...	396372	1118	308
104.238.130.182.vultr.com	70002	/Satoshi:0.1...	396371 •	656	457
74.84.128.158	70002	/Satoshi:0.1...	396372	265	253
CUST-165.217.102.5.018.net.il	70002	/Satoshi:0.1...	396372	271	238
dnsseed.bitcoin.dashjr.org	70002	/Satoshi:0.1...	396369 •	427	300
104.239.233.251	70002	/Satoshi:0.1...	396372	434	260
walter.no-network.fr	70002	/Satoshi:0.1...	395735 •	597	308
dhcp-077-248-045-210.chello.nl	70002	/Satoshi:0.1...	396371 •	314	212
185.25.45.46	70012	/Satoshi:0.1...	396372	433	748
chello080108143019.4.12.vie.surfer.at	70002	/BitcoinUnli...	396372	228	188
ec2-52-8-184-157.us-west-1.compute.amazonaws.c...	70002	/Satoshi:0.1...	396372	352	341
dhcp-169-229-198-105.lips.berkeley.edu	70002	/Satoshi:0.1...	396372	348	348
77.247.178.142	70002	/Satoshi:0.1...	396372	220	159

2016/02/02 13:24:16

program is starting to get all Bitcoin block chain...

Fig. 5. Listener User Interface

Listener component is developed in Java language. This component listens to Bitcoin network and records the hash of all broadcasted blocks in a file. We show the user interface of this component in Fig. 5. The client of this software must enter the count of desired peers in the box on top of the form and then click on the start button. After clicking this button, the current time date is written in the box next to the button and all boxes and the button are disabled. All rows in the form show the peers that are connected to listener component. The whole Bitcoin blocks after that time is recorded in listener file.

Detector component receives the output files of monitoring and listener components and detect that if the suspicious application is malicious or not based on the algorithm that is described in the previous section.

V. Evaluation

Because of our limitation in accessing real Botcoin malware, we develop some simulated Botcoins. We evaluate this proposed method experimentally by these simulated Botcoins. The Bitcoin miner malware utilizes the real Bitcoin mining software as a core component of mining process and then wrap the core, by other software to communicate with C&C and provide botmaster parameters [2]. As a consequent of mentioned fact, we simulate several Botcoins with different parameters of the core miner to cover a wide range of real Botcoins functionalities.

We benefit DiabloMiner [15] in Java language that works based on GPU processing and BfgMiner [16] in C language that mines in both CPU and GPU. We cover various source languages of miners (Java and C) in our test cases. Also various kinds of processor (CPU and GPU) are considered in our test. Mining standalone or working as a member of a pool is another variation that we considered in the experiment.

Considering all of these criteria for diversifying, we create a 6-sized data set of Botcoin executable binary files. Then we consider 10 benign executable files in our data set. We examine the developed software and consequently the proposed approach with the data set.

As we expect due to the strictness of the method, all Botcoins were found, and because of the unlikely observation of this memory access pattern, no other software was caught up. This experiment led to zero false positive and zero false negative in the test set.

As far as we know there are no published papers to represent a methodology to detect Botcoins. For this reason, we cannot compare our results with previous ones and cannot have a benchmark with them. In this reason we also present a comparison between our approach with some other general botnet detection methods in the context of detecting Botcoin:

Dynamic analysis of OS APIs: All Bitcoin miners containing Botcoins must use encryption functions to perform their mining functions. Some of them use OS APIs to calculate this encryption functionality, and in these cases, we can hook some proper OS APIs to help malicious behavior detection. Miners can easily implement their desired functions independent of the OS, and therefore these types of detections can easily be bypassed.

Static analysis of crypto functions: Another candidate solution for detecting Botcoins is a static analysis of cryptographic functions. In this method, we can find the targeted function in any binary executable file. All static analysis based detection approaches can be bypassed with some methods such as obfuscation [17].

Dynamic analysis of network traffic: Botcoins can receive commands in a wide variety of protocols. Some of them are well known and can be detected, but there is no force to use them and a bot can use any desired protocol. In the worst scenario, the malware uses a customized, secure encrypted channel to communicate information with C&C server. Communicating with well-known mining pools is also not helpful to detect Botcoin because Botcoins can use proxies to hide their real destination IPs. As a consequence of this variety, network traffic analysis based approaches have several problems in detecting Botcoins.

TABLE II. COMPARISON WITH OTHER METHODS

Approaches	Not using the OS APIs for hashing	Self-Modifying code (for example, Packer)	Obfuscation	Using Encrypted C&C Protocol
Dynamic analysis of OS APIs	No	No	No	Don not care
Static analysis of crypto functions	Yes	No	No	Don not care
Dynamic analysis of method calls	No	Yes	Yes	Don not care
Dynamic analysis of network traffic	Don not care	Yes	Yes	No
Botcoin Trap	Yes	Yes	Yes	Yes

VI. CONCLUSION

This paper is the first study on Bitcoin miner botnet detection techniques. Although there are some researches on Botcoin, none of them has the concern of giving some novel techniques to find them based on this malicious behavior. This technique is host based and is based on dynamic analysis of a binary executable file or its process.

This technique overcomes with a variety of limitations in network level, such as C&C channel encryption, and also static code solutions such as code obfuscation. Because this technique is done at assembly level of mining activity then can bypass some techniques that a usually malware utilize to hide its malicious activities. The competition between Botcointrap and other methods is presented in Table II. This method is evaluated by a set of executable binary files, containing simulated Botcoins and some other benign executable binary files. We prove that it has an excellent accuracy about in our experiment.

REFERENCES

[1] D. Y. Huang *et al.*, “Botcoin: Monetizing Stolen Cycles.,” in *NDSS*, 2014.

[2] B. Krebs, “Botcoin: Bitcoin Mining by Botnet.” [Online]. Available: http://krebsonsecurity.com/2013/07/botcoin-bitcoin-mining-by-botnet/. [Accessed: 18-May-2018].

[3] S. Khattak, N. R. Ramay, K. R. Khan, A. Syed, S. A. Khayam, and others, “A taxonomy of botnet behavior, detection, and defense,” *Commun. Surv. Tutor. IEEE*, vol. 16, no. 2, pp. 898–924, 2014.

[4] M. Egele, T. Scholte, E. Kirda, and C. Kruegel, “A survey on automated dynamic malware-analysis techniques and tools,” *ACM Comput. Surv. CSUR*, vol. 44, no. 2, p. 6, 2012.

[5] D. Dagon, G. Gu, C. P. Lee, and W. Lee, “A taxonomy of botnet structures,” in *Computer Security Applications Conference, 2007. ACSAC 2007. Twenty-Third Annual*, 2007, pp. 325–339.

[6] E. Khoshhalpour and H. R. Shahriari, “BotRevealer: Behavioral Detection of Botnets based on Botnet Life-cycle,” *ISC Int. J. Inf. Secur.*, vol. 10, no. 1, pp. 55–61, 2018.

[7] J. Wyke, “The ZeroAccess botnet–Mining and fraud for massive financial gain,” *Sophos Tech. Pap.*, 2012.

[8] D. Plohmann and E. Gerhards-Padilla, “Case study of the miner botnet,” in *Cyber Conflict (CYCON), 2012 4th International Conference on*, 2012, pp. 1–16.

[9] P. Güring and I. Grigg, “Bitcoin & Gresham’s Law-the economic inevitability of Collapse,” *October–December Httpiang OrgpapersBitcoinBreachesGreshamsLaw Pdf*, 2011.

[10] S. Nakamoto, “Bitcoin: A peer-to-peer electronic cash system,” *Consulted*, vol. 1, p. 2012, 2008.

[11] “Blockchain blocks version.” [Online]. Available: https://data.bitcoinity.org/bitcoin/block_version/all?c=block_version&r=week&t=a. [Accessed: 18-May-2018].

[12] “bips: Bitcoin Improvement Proposals-bip0034,” 12-Oct-2017. [Online]. Available: https://github.com/bitcoin/bips/blob/master/bip-0034.mediawiki. [Accessed: 12-Oct-2017].

[13] “bips: Bitcoin Improvement Proposals - bip0062,” 12-Oct-2017. [Online]. Available: https://github.com/bitcoin/bips/blob/master/bip-0062.mediawiki. [Accessed: 12-Oct-2017].

[14] C.-K. Luk *et al.*, “Pin: building customized program analysis tools with dynamic instrumentation,” in *ACM Sigplan Notices*, 2005, vol. 40, pp. 190–200.

[15] P. McFarland, *DiabloMiner: OpenCL miner for Bitcoin*. 2018.

[16] “BFGMiner - a modular ASIC/FPGA Bitcoin miner.” [Online]. Available: http://bfgminer.org/. [Accessed: 22-Mar-2018].

[17] A. Moser, C. Kruegel, and E. Kirda, “Limits of static analysis for malware detection,” in *Computer security applications conference, 2007. ACSAC 2007. Twenty-third annual*, 2007, pp. 421–430.