Blockchain based Secured Virtual Machine Image Monitor

Srijita Basu[1][a], Sandip Karmakar[2][b] and Debasish Bera[1][c]
[1]*Department of Computer Science and Engineering, Indian Institute of Information Technology, Kalyani, India*
[2]*Department of Computer Science and Engineering, National Institute of Technology, Durgapur, India*

Keywords: Virtual Machine Image, Blockchai, Smart Contract, Auditability, Cloud.

Abstract: Blockchain technology supports data immutability. Whereas, smart contracts are piece of self-executable codes running inside the blockchain network, responsible for the transformation or state change of these data. Furthermore, Cloud Computing is used in the application of data storage and usage. Several business enterprises use cloud for hosting their applications and data with a minimized effort, cost and hurdles of maintenance. However, ensuring security of client data and proper management of the Service Provider's infrastructure remains a crucial issue. In this article, an Ethereum based blockchain network has been proposed that monitors and assures the safety of the Virtual Machine Images (VMI) stored at the Cloud Service Provider (CSP) end. The proposed scheme tends to design a dedicated Smart Contract which handles each and every function, starting from request of a VMI by the Cloud Service Consumer (CSC) to the usage of the same by the later. The use of blockchain technology ensures that no single admin/third party can control/modify the system. This prevents unwanted modification of the VMIs by an intruder and guarantees the efficiency of the scheme to be higher than any other methodology designed for the same purpose till date.

1 INTRODUCTION

Cloud based services have become very popular among small and medium scale enterprises. It reduces not only the hardware and software maintenance costs, but also facilitates the CSC with various add-on features like on-demand scaling, pay per use pricing model, flexible and easy deployment options, etc. Cloud computing has drastically changed the sphere of renting services, by providing rents on softwares, operating environment, and even hardware resources. A substantial apprehension comes with the cloud based service usage in terms of security for both the CSC's data as well as CSP's infrastructure.

The various security concerns (Hashizume et al., 2013) include data locality, data segregation, Virtual Machine (VM) life cycle management, and public VM Image (VMI) repository regulation etc. In spite of the fact that several cloud specific schemes (Wan and Jiang, 2010) have been devised for each of these issues, there are still some areas which lacks significant work. One such vital security issue is lack of *management and tracking of the Virtual Machine Images (VMI)*, which results in the sustainability of various malicious VMIs in the CSP's data-centre.

Blockchain technology has been projected as one of the most secured solutions for data storage and monitoring. It has quite often been coined as a distributed and decentralized public ledger. Back in the year 2008, Satoshi Nakamoto had used Blockchain technology for constructing a crypto-currency platform known as Bitcoin (Nakamoto, 2008). Later in the year 2014, it was realized that blockchain could be used to permanently record any kind of transaction/change without a third party intervention. This was followed by the 2nd generation of Blockchain-Ethereum. Ethereum (Wood, 2014) introduced the tech-world to Smart Contracts (Macrinici et al., 2018). This new generation blockchain technology helps not only digital transactions but any service, asset, bond etc. can be exchanged between peers, depending on conditions that were depicted in the smart contracts. The blockchain architecture (Halpin. and Piekarska, 2017), with the help of mining algorithms (Wang et al., 2019), clearly portrays the difficulty for an intruder would face if he/she wishes to change the entries of the block. Thus, it is quite evident that blockchain with its smart contracts can be widely used to ensure secured data/service preservation.

[a] https://orcid.org/0000-0002-6835-947X
[b] https://orcid.org/0000-0002-8150-3026
[c] https://orcid.org/0000-0001-8888-6042

Basu, S., Karmakar, S. and Bera, D.
Blockchain based Secured Virtual Machine Image Monitor.
DOI: 10.5220/0010228804320439
In *Proceedings of the 7th International Conference on Information Systems Security and Privacy (ICISSP 2021)*, pages 432-439
ISBN: 978-989-758-491-6

In this paper an effort has been made to design an Ethereum (Blockchain) based Smart Contract that would ensure VMI security and tracking. The use of smart contract based blockchain technology in this system ensures that no malicious VMI exists at the CSP end and additionally tracks the entire life cycle of a VM (Schwarzkopf, 2015) by providing a series of authentic audit data (timestamp, violation action, etc) which helps to identify the intended activity. The rest of the paper is organized as follows. In Section II, a survey of related work is presented. Section III discusses the architecture of the proposed system. Section IV details the proposed methodology for VMI management and tracking. In Section V result analysis for the proposed scheme has been performed. Finally, Section VI concludes the article.

2 RELATED WORK

This section presents a brief survey of different schemes on VM image tracking and security in a cloud environment. Virtual machine images require full-proof integrity schemes, as they initialize a VM from scratch including its configuration as well as security parameters or policies which if changed can pose threat to the client applications or data.

In (Wei et al., 2009), an image management scheme was proposed for secured cloud environment. The various sub-operations /sub-modules in this scheme include access control framework, Image filters, provenance tracking mechanism and a set of repository maintenance functions. The image provenance tracking system was used by the CSP to address two major issues viz. tracing the introduction of illegal or malicious content in a VMI and delegation of patching information. The proposed scheme achieved 4.9 times speedup and addressed VMI security aspects from both the image publisher as well as retriever sides. Though the scheme offered a proper tracking mechanism, it did not alleviate the risk of uploading a VMI with malicious content.

In (Kazim et al., 2013), Kazim et. al proposed an Encrypted Virtual Disk Images in Cloud (EVDIC) based on OpenStack Platform. The system ensured the protection of the stored virtual machine disk images at the Cloud Service Provider's end. The design consisted of a Virtual Machine Image encryption module, Virtual Machine Image disk decryption module, and Key management Module. The scheme aimed at protecting the disk images from getting compromised by some malicious internal user or administrator by unauthorized access. Though the work tends to provide security to the Virtual Machine Images from the malicious CSP users, it is unable to protect the same from the external intruders/ malicious Cloud Service Consumers.

A security framework that tried to protect the Virtual Machine Images (VMI) in a cloud platform was devised by Hussein, Alenezi, Wills and Walters in (Hussein et al., 2016). The paper formulated a research path following which VMI security could be strengthened. A model has was suggested where the industry based security controls have been taken in parallel with the academic security controls. At the later stage, the redundant security controls have been eliminated by semantically comparing each of the industry and academic controls. Finally, a combined security control model has been produced that is expected to meet the VMI security requirements in a better way.

It is evident from the above discussion that the existing VM/VMI management models are either incomplete or depend on some central or third party authority/CSP admin. Most of them have certain overheads and are unable to present a fool-proof solution to CSP and/or CSC. Therefore, there is a need to design a comprehensive model constructed on the core of blockchain network, that can not only identify the stakeholders, processes and properties of the cloud system involved, but also provide the required tracking and management to ensure a secured VMI repository.

3 SYSTEM MODEL

In this section, the basic system design is outlined with an overall insight into the entities of the proposed model. The well-established definition of cloud (Buyya et al., 2009) portrays it as a distributed system consisting of a collection of interconnected and virtualized computers that are dynamically provisioned and presented as one or more unified computing resource(s) (Buyya et al., 2009). The problem that is being dealt with in this article requires the system model to be designed accordingly.

The proposed system has been modelled to contain the following components. It might be noted that this composition is done according to the proposed work. However, there may be some additional entities which have been ignored here due to its lack of relevance with the present work. Fig. 1 describes the architectural design of the proposed model, describing every component as follows.

- *Data-centre:* A data-centre is a physical repository meant for gathering cloud computing resources and other existing components (Sahli et al., 2014)

- *Hosts/Server:* These are the real/physical infrastructures hosted inside a data-centre, used to serve the computation and execution requests coming from the client end (Sahli et al., 2014).
- *Virtual Machine (VM):* It is the virtual abstraction of the underlying infrastructure (Sahli et al., 2014) that exists inside a host. Multiple VMs can be instantiated as well as powered off or even killed i.e. deleted on-demand inside single host as per the user requests.
- *Virtual Machine Image (VMI):* These are the various flavours of images (customized Operating System images, sometimes pre-loaded with certain applications) that are used to boot a VM on requirement.
- *Cloud Storage:* Different kinds of essential data are stored in logical pools. The physical storage is generally sketched in two ways (Abbadi, 2011). i) It is distributed across different dedicated storage servers that may be located in a replicated form in geographically distant data-centres.

  ii) Again in some instances of lesser storage requirements, the storage server may be a part of computation server itself (the one that hosts the VM for running various applications).
- *Block:* A Block (Conti et al., 2018) is developed from VM Request and Response documents. A single block spans an entire VM request from its initiation to completion (details will be discussed in the upcoming sections). In addition to the main block there exists side block for every main block which contains the smart contract reports (explained in later section).
- *Blockchain:* As already mentioned, each block represents each VM access request. A sequence of such blocks forming a chain represents a series of requests from a particular user on different VMs.
- *Cloud Network:* This can be physical as well as virtual network. Since, the blockchain technology is being used in this model the network as a whole represents a blockchain P2P(peer-to-peer) model where each cloud host/server and consumer contains/shares the same copy of blockchain and communicates accordingly as and when required at the time of each blockchain consensus (Wang et al., 2019).
- *Smart Contract:* Smart Contracts (Macrinici et al., 2018) add enhanced feature to normal blockchain network. They consist of a set of self-executable code snippets embedded as a part of the blockchain itself. In this model, how the Smart Contracts control the entire process of VMI request and allocation, followed by the post allocation actions has been portrayed.

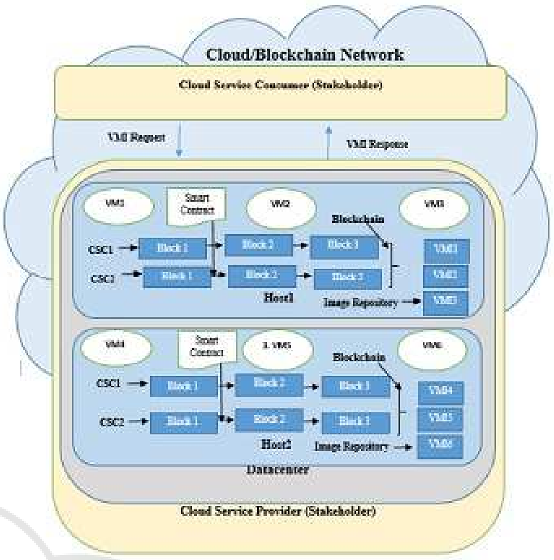

Figure 1: System Mode.

4 WORKING OF THE BLOCKCHAIN BASED VMI MONITORING SYSTEM

In this section, VMI Monitoring scheme is presented with its detailed working methodology. The specific issues that can be addressed by this scheme have been identified as follows.

4.1 Methodology

- *Authentication Phase*: Here the user registers itself with the CSP and gets his/her name enlisted as the CSC. In this Registration process, the user needs to supply some of the basic information like Name, Email, Contact, Organization Name, Organization location and some additional information if required, such as period of registration, specific purpose/project for which the user is registering etc. After this, the CSC logs in and sends its request in the form of specific VM instance requirement. E.g. the request contains the particular Storage, RAM, CPU cycle, OS, etc. requirements. In addition to that, the CSC might also add some specific security requirements with respect to this VM request. These requirements mainly depends on the purpose for which the VM was requested.

The kind of data to be deployed or applications to be executed on the VM directly influences this security requirements. After framing the request packet, it is send to the CSP.

If the system notices that the user is not a registered CSC then the login process fails. Additionally the system can also revert back/deny any abnormal request packet by analysing the requested parameters (Storage, CPU cycles). The request packet passing through all these process is forwarded to the next phase after appending the received request timestamp and the CSC ID with it.

- *VM Allocation Phase*: The request is processed for the first time in this phase. The requested parameters are examined and accordingly a VMI is selected from the public VMI repository. The VMI should be such that it fulfils all the CSC requirements along with any specific security requirement the CSC has included. For executing this allocation, the existing "virtual_machine_basic mapping" (explained in the later section) is checked to find the possible VMIs with permissible hardware and software configurations. A list of VMIs is returned, out of which one is selected, maintaining the load balancing (Ferris, 2014) criteria of the CSP. After selecting the VMI, system creates a VMI request document. This document includes the hashed timestamp of the received request, hash of the VM ID (VM ID of any VM can be retrieved from the System Database), hash of the host ID that deploys the concerned VM, hash of the CSC ID initiating the request, and finally the hashed data containing the purpose of request and any specific security requirements of the CSC if any. The necessary changes according to this request document is made in the existing "virtual_machine_allocation" mapping (explained in the later section). The timestamp, CSC ID, VM ID, Host ID and the additional security data if any, are inserted into this mapping. The basic details for the particular VMI is also updated in the "virtua_machine_basic" mapping, as the free space, RAM, CPU cycles and any other resource would now be changed with the new allocation being done. Once all the changes have been made, the time stamp for the same is also noted as the response time stamp.
- *VM Response Phase*: In this step, the system constructs the VMI response document. This document contains the hash of the timestamp of the received response i.e. the timestamp when the allocation was reflected in the blockchain, the hash of the allocated VM login details (VM IP address and password) and the hashed VM metadata. The system logs the VM Request and Response documents in the blockchain (for the particular user) as a new block. The block is appended in a chronological way, i.e. this block appears after the last block which had been formed with respect to the previous request from the same user. The change of state of each VMI can now be easily tracked from the blockchain network.
- *VM Action Triggering Phase*: The system now executes necessary function using which the CSC can log in to the VMI, allocated to him/her using the valid VM IP and password. On successful login, the system logs every action as an event into the blockchain. For every action it consults the Blockchain Database to return the policy list mapped against this particular VMI. After fetching the list, it is compared with the most recent CSC action. If this action is found to be a valid one then no step is taken otherwise for every illegitimate action the same is logged as an event with the particular time stamp and CSC ID in the side block. This phenomenon ensures auditability and a rigorous monitoring of the VMI. Moreover, the CSC is forcefully logged out by ending its session and depending upon the sensitivity of the VMI, the password for the same might also be modified so that the CSC is unable to login again. In case of lower VMI sensitivity the CSC is given a chance to re-login in the VMI with the same credentials, but this time the particular object for which access was denied during the previous check is locked.

  The above scenario mainly concentrates on the immediate action taken against any illegitimate access request. Along with this, the normal modifications conducted on the VM should also be noted. E.g. the user might install a new software in the allocated VM or he/she might update the version of some pre-installed software. Every such action must be logged into the blockchain. At the same time the overhead of this event logging activities in the blockchain should be kept as minimal as possible (as each transaction involves consumption of Ether Gas (Wood, 2014)). Keeping this in mind, the proposed system runs a periodic blockchain write, which logs all the legitimate VM modification events for that particular period into the blockchain and modifies its policy list accordingly for that VM. This minimizes the overhead of saving every change immediately as it occurs in a particular VM and reflecting the same in the policy list. Though the periodic updates save transaction charges that would have been incurred if continuous logging would have taken place, it induces another risk. Imagine a scenario where User A is working with VM X and some valid modifications have been made by him/her on VM X. The periodic update is yet to occur and in the meantime another user, User B is also allocated VM X. It should be noted that the actual version of VM X is different from what the CSP checks before allocating the same to User B. Moreover, the policy list with respect to VM X has not been updated. This might be counted as a vulnerability of the system. There might be certain unlogged changes in VM X and the Policy list that needs to addressed against this new allocation. A solution to this issue has been devised where the system in the VM Allocation Phase checks the following set of data viz. i) Whether the allocated VM is already shared by some other user at present. If the answer to this query is yes then ii) Check the last update timestamp in the blockchain, for the concerned VM (VM X). If this timestamp is found to be right after the allocation request was made, then no further action needs to be taken and VM X is allocated to User B. But if this timestamp is found to be before the allocation request then the allocation is halted for the time being. A forceful blockchain write is made against the action User A-> VM X. After this the new allocation of User B-> VM X is made.

Thus for the entire procedure of sending a request by the CSC to fulfilment of the same, it has been observed how the blockchain network and the embedded smart contract not only helps to monitor the VMIs across its lifecycle but also prevents any malicious user to bring unintended changes to the VMI. On one hand it secures the CSP infrastructure by assuring that the VMIs stored in the CSP public repository are safe. On the other hand any legitimate CSC who is allocated a VM from this public repository can now be sure enough to have received an uninfected VM. Moreover, any unnatural change in a particular VMI is auditable from the information present in the side blocks. This information can be utilized by the CSP, even at a later stage to decide directly whether to accept a particular CSC request or to reject it as this CSC might have been involved in some improper actions pertaining to any previous request which could be easily tracked from the side blocks. Therefore, it is evident that this scheme is beneficial from both the CSC as well as the CSP end.

4.2 Smart Contracts

The structure and role of the smart contract has been discussed elaborately in this sections. As described earlier, a smart contract serves as the main driving agent behind a successful and secure VM allocation and monitoring process. The variables used in this smart contract have been depicted in Table I along with their name, purpose and nature (State (S)/Local(L) (Anonymous, 2019)). Some of the vital functions and events have also been elaborated as follows:

Table 1: Smart Contract Variables.

Name and Type of Variable	***Purpose and Nature***
address public csc_ID	Stores the address of CSC at the time of SLA formation(S)
uint public vm_ID	VM/VMI identifier(S)
mapping(uint-> CSC_request) publicvirtual_machine_request;	Stores the various request parameter against each CSC Identifier(S)
mapping(uint-> VMI_UI) publicvirtua_machine_allocation;	Stores the allocation mapping of the VM against each user(S)
uint public event_ts	Stores the timestamp of the most current event encountered by the CSC (L)
String public action_name	Stores the name of the most current event encountered by the CSC (L)
mapping(uint-> VMI_policy) publicvirtual_machine_policy;	Stores the Policy details against each VM(S)
String public VM_IP;	Stores the VM IP address (L)
String VM_password;	Stores the VM IP password (L)
enum VM_state { Idle; Busy; Dead; }	Stores the different states of a Virtual Machinee(State Variable)

- *Event Login:* It logs/stores the user id and the Time stamp at which a valid CSC logs in to his/her allocated VM.
- *Function New Event:* The function checks for the VM and user sensitivity. It also checks whether

the VM state is idle and the sender of this function is a valid CSP. If all the conditions are true then it stores the action that the user performs on the VM as an event in a periodic manner as mentioned in theVM Action triggering Phase in the previous subsection. The event contains the action name, timestamp, User ID and VM ID

- *Func get_ VMI_policy:* It returns the policy id and the read_deny_policy containing the read deny policies for the particular VM. The same would have been write_deny_policy and execute_deny_policy while checking the write and execute deny policy list. So for each VM 3 deny policy lists are maintained.
- *Func Check_policy:* The function matches the deny_policy list returned from the previous function with EName (action/event name) that the CSS had just performed. If no match occurs, then action can be allowed and the action is unblocked with respect to this user where as if a match occurs the function constructs a message "Uid + denied access to + EName". Now depending upon the sensitivity of the VM two set of actions might be possible, i) Event "Deny Access" takes place that logs the above message along the particular VM ID and the CSC is forcefully logged out of the system, and the login password for the same is changed so that the CSC might not re-login, or ii) Event "Deny Access" takes place that logs the above message along the particular VM ID and the CSC is forcefully logged out of the system. In the 2nd case the VM sensitivity is low and therefore the user is not permanently unallocated from the VM. He/she is just temporarily logged out and prevented from performing this particular action.

The functions associated with the Smart Contract that mainly controls the VM Action Triggering Phase have been separately depicted in Fig. 2. It acts as the core control of the Smart Contract.

5 RESULT ANALYSIS

Here an Ethereum (Wood, 2014) based Smart Contract with Solidity (Dannen, 2017) (Version 0.4.4) over a Remix (Yann300, 2020) platform has been used to design the intended system. In the private blockchain setup the average Gas price has been taken as 0.00000002 Ether, and 1 Ether =391.933 USD was observed at the time of experiment. Table III shows the cost associated with the functions that are mainly involved in ensuring the VMI security as well as monitoring.

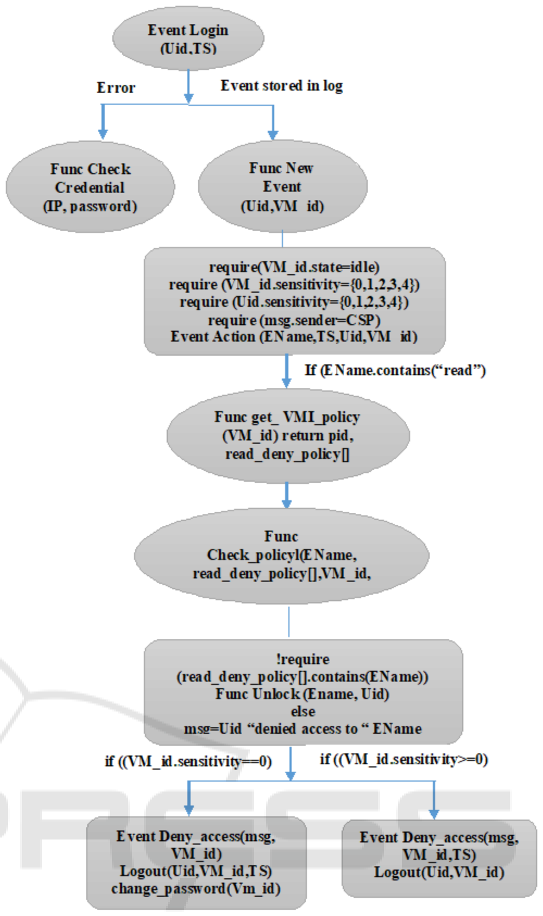
Event Login (Uid,TS)
Error
Event stored in log
Func Check Credential (IP, password)
Func New Event (Uid,VM id)
require(VM_id.state=idle) require (VM_id.sensitivity={0,1,2,3,4}) require (Uid.sensitivity={0,1,2,3,4}) require (msg.sender=CSP) Event Action (EName,TS,Uid,VM id)
If (EName.contains("read")
Func get_ VMI_policy (VM_id) return pid, read_deny_policy[]
Func Check_policyl(EName, read_deny_policy[],VM_id,
!require (read_deny_policy[].contains(EName)) Func Unlock (Ename, Uid) else msg=Uid "denied access to " EName
if ((VM_id.sensitivity==0)
if ((VM_id.sensitivity>=0)
Event Deny_access(msg, VM_id) Logout(Uid,VM_id,TS) change_password(Vm_id)
Event Deny_access(msg, VM_id,TS) Logout(Uid,VM_id)

Figure 2: Flow Chart for VMI Monitor Smart Contract.

Table 2: Associated Cost.

Function	***Gas Used***	***Cost (Ether)***	***USD***
Event Login	13256	0.00079536	.31
Function New Event	41567	0.00207835	0.82
Func get VMI policy	12754	0.00038262	0.15
Func Check policy	50153	0.00250765	0.98

The effect of the number of VMIs at the CSP end on the Gas cost has been plotted in Fig 3. It has been observed that the Gas cost increases linearly with the number of VMIs deployed in the system. Another observation has also been made in Fig 4., that shows a linear relationship again, between the total no. of policy statements in the Blockchain database and the total Gas cost incurred.

In (Kazim et al., 2013), managing security of Virtual Machine Images in a cloud environment has been observed that there are mainly three facets of Virtual

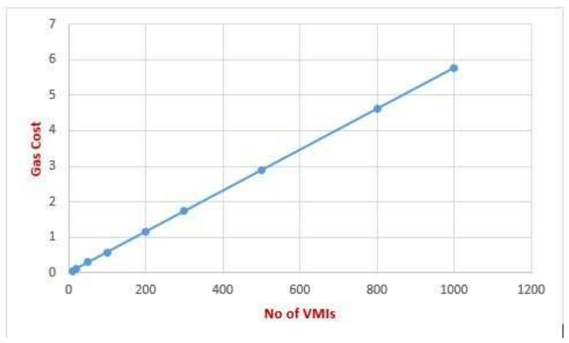

Figure 3: No. of VMIs v/s Gas Cost.

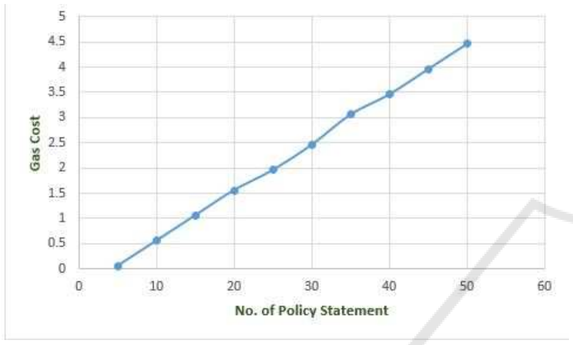

Figure 4: No. of Policy Statement v/s Gas Cost.

Machine Image that has been taken care of. The system proposes and combines three different systems each for access control, image filtering, and provenance tracking. In a real life scenario, the minimal size of the VMI is approximately 10 GB. In general, on an average, more than 5000 files run on each VMI (when deployed as VM). A CSP (private) with a medium sized data centre running at least 500 such VM instances (Marr and Kowalski, 2014). Therefore, running and maintaining the above model at the CSP end would result in a high percentage of over-head. The access control policies have inevitable dependencies on their owner as proposed by the system. This creates a bottle neck and makes it vulnerable at the administrator/owner level. Once the central authority is compromised the entire system loses and its viability. Finally, the provenance tracking system is designed to track the derivation history of 500 VMIs separately. Each new VMI can experience a huge number of state transitions in its entire lifespan. Any centralized system, maintaining this record, would encounter a high number of system entries triggered by each such transition.

On a contrary, the Blockchain based VMI Monitor, proposed in this article, assures that no change in any VMI occurs without the consent of the CSP or all the concerned hosts that are part of the blockchain network. The Smart Contract, as already seen, is a set of auto executable codes that run when some specific criteria/conditions are met. Here, the blockchain network with its embedded smart contract, i) Maintains the access control policies for the VMIs, ii) Takes the post violation decision depending on its in-built code, and iii) Monitors or tracks the VMI state changes. The violation reports are stored in the side blocks of the block chain, which provides a full-proof means of monitoring/auditing the VMIs. The VMIs can undergo innumerable state changes, all of which are well reflected in the blockchain that stores the entire VMI metadata. Moreover, due to the inherent property of immutability in blockchain,it can be assumed that an adversary can not control a majority of the computing power in the network, as a result avoiding 51 percent attack (Conti et al., 2018). The administrator/system admin level compromise also becomes quite difficult in this environment due to the presence of miners. Impersonification by the intruder also becomes a tough job here, as the private cryptographic keys (Conti et al., 2018) represents each uniqueparticipant of the network.

6 CONCLUSION

In this article, a formal model of blockchain based VMI Monitor has been proposed that runs on an Ethereum platform. Different components of the proposed model have been described in detail. The Cloud or the VMI operations are main concern in this work. They have been elaborated and the different phases of the entire procedure have been detailed. Furthermore, the smart contract for this particular problem has also been designed along with the utility and role of the various variables and functions. It has also been established that the proposed system is better than the previous scheme(Kazim et al., 2013) that has been implemented to solve the same set of issues.

It has been observed that the smart contract based system not only provides a proper VMI access control mechanism but also provides the scope of VMI auditability and secured operations from the perspective of CSC as well (as observed that the side blocks are checked when any CSC with high sensitivity is dealt with). This methodology turns to be beneficial not only for the CSP, who can now maintain a secured Cloud infrastructure but also for the Cloud Service Consumer (CSC) who can now deploy his application/data in a VMI without the fear of getting his/her data compromised.

The main overhead lies in the scalability of the system with the number of VM transactions i.e. number of CSC request. The more the number of request, more the size of the blockchain and with the increased number of blocks the time required to finalize the same and add it to the main chain using the PoS (Baliga, 2017) consensus method increases. An effort has been made here (as mentioned in the previous sections) to balance the number of transactions by following a user defined periodic update method. This avoids unnecessary immediate transactions involving blockchain write.

Future work is geared towards the development of an automated tool based on the proposed methodology. Another aspect that can be considered as future work is enactment of the security issues related to virtual machine instances for multi-provider federated clouds. The model, in its current form, considers only single CSP set-ups.

REFERENCES

Abbadi, M. I. (2011). Clouds' infrastructure taxonomy, properties, and management services. In Advances in Computing and Communications, Part IV, edited by A. Abraham, J. L. Mauri, J. Buford, J. Suzuki, and S. M. Thampi, Heidelberg: Springer-Verlag, 406-420.

Anonymous (2019). Local and state solidity variables and use of parameters. BitDegree Learn. https://www.bitdegree.org/learn/solidity-variables, Accessed 30 September 2019.

Baliga, A. (2017). Understanding blockchain consensus models. In Persistent. https://www.persistent.com/wp-content/uploads/2018/02/wp-understanding-blockchain-consensus-models.pdf, Accessed April 2017.

Buyya, R., Yeo, C. S., Venugopal, S., Broberg, J., and Brandic, I. (2009). Cloud computing and emerging it platforms: Vision, hype, and reality for delivering computing as the 5th utility. Future Generation computer systems, 25(6), 599-616.

Conti, M., Kuma, E. S., Lal, C., and Ruj, S. (2018). A survey on security and privacy issues of bitcoin. IEEE Communications Surveys and Tutorials. 20(4), 3416-3452.

Dannen, C. (2017). Introducing ethereum and solidity. In: Dannen, C. (eds.) (Vol. 1), Berkeley: Apress, pp. 1–185. Springer Nature Switzerland.

Ferris, J. M. (2014). Red hat inc.: Load balancing in cloud-based networks. U.S. Patent 8,849,971.

Halpin., H. and Piekarska, M. (2017). Introduction to security and privacy on the blockchain. In 2017 IEEE European Symposium on Security and Privacy Workshops (EuroS and PW), pp. 1-3, IEEE.

Hashizume, K., Rosado, D. G., Fernández-Medina, E., and Fernandez, E. B. (2013). An analysis of security issues for cloud computing. In *Journal of internet services and applications*. SPRINGER.

Hussein, R. K., Alenezi, A., Wills, G. B., and Walters, R. J. (2016). A framework to secure the virtual machine image in cloud computing. In 2016 IEEE international conference on smart cloud (SmartCloud), pp. 35-40, IEEE.

Kazim, M., Masood, R., and Shibli, M. A. (2013). Securing the virtual machine images in cloud computing. In Proceedings of the 6th International Conference on Security of Information and Networks, pp. 425-428.

Macrinici, D., Cartofeanu, C., and Gao, S. (2018). Smart contract applications within blockchain technology: A systematic mapping study. Telematics and Informatics, 35(8), 2337-2354.

Marr, M. D. and Kowalski, M. P. (2014). Amazon technologies inc. scaling a virtual machine instance. U.S. Patent 8,825,550.

Nakamoto, S. (2008). Bitcoin: A peer-to-peer electronic cash system. https://bitcoin.org/bitcoin.pdf.

Sahli, H., Belala, F., and Bouanaka, C. (2014). Model-checking cloud systems using bigmc. In 8th International Workshop on Verification and Evaluation of Computer and Communication Systems, pp. 25-33.

Schwarzkopf, R. (2015). Virtual machine lifecycle management in grid and cloud computing. https://doi.org/10.17192/z2015.0407.

Wan, Z. and Jiang, X. (2010). Hypersafe: a lightweight approach to provide lifetime hypervisor control-flow integrity. In IEEE Symposium on Security and Privacy, pp. 380-395, IEE.

Wang, W., Hoang, D., Hu, P., Xiong, Z., Niyato, D., Wang, P., Wen, Y., and Kim, D. I. (2019). A survey on consensus mechanisms and mining strategy management in blockchain networks. IEEE Access. 7, 22328-22370.

Wei, J., Zhang, X., Ammons, G., Bala, V., and Ning, P. (2009). Managing security of virtual machine images in a cloud environment. In Proceedings of the 2009 ACM workshop on Cloud computing security, pp. 91-96.

Wood, G. (2014). Ethereum: A secure decentralised generalised transaction ledger. Ethereum project yellow paper, 151, 1-32.

Yann300 (2020). Remix documentation. Release 1, https://readthedocs.org/projects/remix-ide/downloads/pdf/latest/.