MalCon: A blockchain-based malware containment framework for Internet of Things

Ahmed Lekssays, Barbara Carminati, Elena Ferrari

Computer Networks 233 (2023) 109853

Contents lists available at ScienceDirect

Computer Networks

journal homepage: www.elsevier.com/locate/comnet

MalCon: A blockchain-based malware containment framework for Internet of Things☆

Ahmed Lekssays [a,*], Barbara Carminati [b], Elena Ferrari [b]

[a] Qatar Computing Research Institute, Doha, Qatar
[b] DiSTA, University of Insubria, Varese, Italy

ARTICLE INFO

Keywords:
Malware containment
Blockchain
Internet of Things
Security

ABSTRACT

IoT devices have become a primary medium for malware (e.g., botnets) to launch Distributed Denial of Service (DDoS) attacks. Such malware exploit low-security measures in IoT devices to spread in networks and recruit new victims. Thus, there is a need for malware countermeasures that consider both the security and operability of the network. Indeed, some IoT devices might run critical processes that do not tolerate interruptions.

This paper proposes MalCon, a blockchain-based malware containment framework for IoT. It aims to stop malware from spreading in a network by a set of containment strategies encoded into smart contracts to be executed by the infected devices. Moreover, MalCon provides a monitoring service that ensures trustworthy behavior in the network and reports to the system administrator any fraudulent activity of the monitored devices. MalCon was tested extensively with real-life malware and use cases. It quickly and drastically reduces the number of infected devices in a network, even in an extreme case of a fully connected network.

1. Introduction

IoT devices have been emerging drastically in the last few years. Due to their low computational power, they cannot run sophisticated security solutions, leading to weaker security guarantees. In addition, they usually adopt weak passwords (e.g., default vendors' passwords) and operate with unencrypted traffic.[1] As a result, attackers may inject different type of malware (e.g., ransomware, trojans, botnets, spyware, viruses) easily to compromise internal networks or attack external targets [1].

According to the NIST SP 800-83 Malware Incident Response guidelines [2], one of the leading guidelines in malware incident response, there are four main steps to contrast malware: preparation, detection and analysis, containment and eradication, and recovery. While many works have addressed the issue of IoT malware detection and analysis, a few have focused on malware containment. Malware containment aims at limiting the spread of malware in networks. To this end, the need for effective and efficient containment strategies arises. Given the spreading speed and the criticality of the potential damages, containment strategies need to: (i) be tailored to the attacking malware on the basis of its characteristics, (ii) ensure the operability of the network even under a malware attack, and (iii) be autonomously deployed to reduce the delay that the human interaction can cause.

Additionally, since IoT networks are heterogeneous and can involve several organizations, the containment process must be collaborative and based on threat information sharing.[2] Exchanging malware information increases organizations' resistance to such attacks since they can implement proactive strategies against the malware before they get attacked. However, organizations do not necessarily trust each other. So, an effective containment solution should ensure the traceability and integrity of the implemented containment strategies.

In this paper, to cope with the abovementioned requirements, we propose MalCon, a blockchain-based malware containment solution for IoT devices. MalCon is based on containment strategies deployed through smart contracts. These strategies are tailored for each specific malware based on its characteristics while considering the system's operability. In addition, MalCon keeps an honest behavior in the system by verifying the execution of suggested strategies after any malware infection. We leverage the blockchain to ensure traceability and integrity.

☆ This work has received funding from the Marie Sklodowska-Curie Innovative Training Network Real-time Analytics for Internet of Sports (RAIS) supported by the European Union's Horizon 2020 research and innovation programme under grant agreement No 813162. The content of this paper reflects only the authors' view and the Agency and the Commission are not responsible for any use that may be made of the information it contains.

* Work done at University of Insubria (Corresponding author)
E-mail address: alekssays@hbku.edu.qa (A. Lekssays).

[1] https://www.enisa.europa.eu/publications/baseline-security-recommendations-for-iot

[2] See for instance the EU Concordia DDos clearing house project https://www.concordia-h2020.eu/news/press-release-ddos-clearing-house-designated-high-potential-innovation-by-european-commission/.

https://doi.org/10.1016/j.comnet.2023.109853
Received 18 May 2022; Received in revised form 12 April 2023; Accepted 30 May 2023
Available online 5 June 2023

Containment actions done by a device are recorded in the blockchain to ensure accountability. In addition, since information about malware and adopted containment strategies are shared in the blockchain, the proposed solution ensures that they are tamper-resistant. Moreover, blockchain provides the tools to run smart contracts as autonomous programs. So, the proposed containment actions are guaranteed to follow the procedure encoded in the smart contract without the need for human interaction. Containment strategies are selected by taking into account malware characteristics (e.g., malicious actions, propagation scheme, etc.) and processes' characteristics, such as their tolerance to rebooting and their replicas' availability.

MalCon is designed for a trustless setting, where the trust of involved parties is not needed for the normal operation of the system. As a result, MalCon relies on smart contracts to propose containment actions. The key advantage of using smart contracts in MalCon is that they enable trustless collaboration between consortium members (i.e., organizations). Unlike a centralized solution that requires a central entity to be trusted, a blockchain-based solution with smart contracts can operate in a decentralized and transparent manner, with no need for trusted intermediaries. This is because smart contracts execute automatically and transparently, with the terms and conditions of the contract encoded on the blockchain, and enforced by the underlying consensus mechanism. Overall, MalCon provides a secure and efficient way for consortium members to collaborate on threat intelligence sharing to automate the mitigation actions, without the need for intermediaries or a central entity to be trusted.

MalCon is, to the best of our knowledge, the first approach that addresses malware containment in IoT leveraging blockchain. The issue of malware containment in IoT was not addressed extensively in the literature, even if IoT malware are having exponential growth.

Indeed, most previous proposals targeting IoT deal with malware detection rather than containment. Malware containment is addressed by [3], but with a straightforward approach of always disconnecting a device from the network if the proposed malware detection model predicts that the device is infected. The only approach we are aware of leveraging on blockchain for malware containment is [4], but it targets malware containment in the cloud. They used a graph analytics approach to predict possible infections, and leverage smart contacts that involve different parties to decide if a possible infected virtual machine should be disconnected from the network or not. The involved parties, namely the cloud provider, security officer, compliance officer, auditor, network admin, and solution admin, give their opinion regarding shutting down a virtual machine, depending on many factors, such as service-level agreements, the danger of the attack, the pricing that the action will cost, etc. MalCon differs from these solutions at many levels. First, MalCon targets a heterogeneous IoT setting where different organizations collaborate to defend against malware threats. The work in [4] focuses on containing malware affecting virtual machines in the cloud. Second, MalCon takes into consideration both the operability and security of infected devices, instead of only shutting down a device once it is infected like [3,4]. MalCon smart contracts encode the containment actions that should be taken without the need for human intervention. Moreover, MalCon is available as open source[3] and was tested extensively in different IoT settings, whereas the work in [4] provided only the theoretical background without real-life experiments.

The remainder of this paper is organized as follows. Section 2 presents MalCon building blocks, whereas Section 3 discusses the details of MalCon implementation. Section 4 analyzes the security of the proposed approach. Section 5 presents experimental results, whereas Section 6 concludes the paper.

[3] MalCon source code is available at https://github.com/lekssays/malcon

Table 1
Mitigation actions executed by the infected peers.

Symbol	Description	Impact on operability
FRMT	Formatting	High
RBT	Rebooting	Medium
DLF	Deleting malware files	Low
CCP	Closing and changing ports	Low

2. Containment process

MalCon containment consists of three sequential phases: emergency, healing, and strategies' execution verification, which are all performed on the blockchain (see Section 3 for more details). Before describing the containment phases, we need to introduce some preliminary concepts.

2.1. Basic concepts

Device. A device (or a peer) is a computer that runs processes and is part of a network. A device could either just execute a process related to the IoT environment, or it could also have the privilege to participate in the blockchain consensus process. We refer to this latter as a *privileged device*.

Process. A process is the execution of a program on a device. In the context of MalCon, we characterize process ρ by two main features. The first is the *replication availability*, which indicates the existence of execution of the same program on another device. The second is the *rebooting tolerance*, which indicates whether ρ can be rebooted without causing any interruption to the service it provides. Therefore, we model a process ρ running on a device D as a pair $\rho = (Rep, RBT)$, where $Rep \in \{yes, no\}$ and $RBT \in \{yes, no\}$ refer to replication availability and rebooting tolerance, respectively.

Each process has a *priority*, that is assigned by the MalCon smart contract based on process features, as follows:

- **Priority 1** (Highest priority): This priority is assigned to processes that do not have a replica and do not tolerate rebooting.
- **Priority 2**: This priority is assigned to processes that do not have a replica but tolerate rebooting.
- **Priority 3** (Lowest priority): This priority is assigned to processes that have a replica.

For each device, we consider the most critical process, that is, the process with the highest priority.[4] The characteristics of this process will play a crucial role in selecting the proper containment strategies (see Section 2.2 for more details).

Malicious actions. These are operations that malware could perform in an infected device. By reviewing different surveys on various malware families [5–7], we consider as possible malicious actions the following ones: encrypt files (EF), delete files (DLF), consume resources (CR), monitor systems (M), send traffic (ST), and open ports (OP).[5]

Malware. We model a malware m as a tuple $(m_a, PRP, ports)$, where m_a is the set of malicious actions that m can perform as defined above, $PRP \in \{yes, no\}$ refers to the ability of the malware to propagate in the network, and $ports$ refer to the ports that m uses for propagation. For example, we can represent Mirai [1], one of the major IoT botnets, as follows: $(\{ST, OP, CR\}, yes, \{23, 2323\})$, because it sends traffic, it opens ports, and it consumes resources (when attacking a target). In addition, it propagates through Telnet with ports 23 and 2323.

Strategy. A strategy defines a set of actions to be performed by IoT devices. In MalCon, we have two types of strategies: emergency and healing, corresponding to the first two stages of MalCon. For

[4] If multiple processes exist with the highest priority, we randomly select one of them.

[5] MalCon can be easily adapted to consider additional actions.

Table 2
Malware-based healing decision (PRP denotes a propagating malware while NPRP denotes a non-propagating malware).

	EF	DLF	CR	M	ST	OP
PRP	FRMT	CCP,DLF	CCP,DLF,RBT	CCP,DLF	CCP,DLF,RBT	CCP,DLF
NPRP	FRMT	DLF	DLF,RBT	CCP,DLF	CCP,DLF,RBT	CCP,DLF

Table 3
Process-based healing decision.

	Replication	Rebooting
Available	CCP,DLF,RBT	CCP,DLF,RBT
Unavailable	CCP,DLF,RBT	CCP,DLF

determining the actions to be considered in each phase, we have done an extensive literature review [8–10], as well as an analysis of leading industrial threat databases, such as TrendMicro[6] and Kaspersky[7]. According to our analysis, in this paper, we consider the actions shown in Table 1. These actions are encoded into a smart contract (cfr. Section 3.3 for more details). It is worth mentioning that the set of actions adopted in MalCon can be easily changed in case new types of malware arise. Also, a different set of actions can be easily supported, depending on the different settings (e.g, operating systems) of the involved IoT devices.

2.2. Emergency and healing strategies

Once a malware is detected on a device, the infected device has to submit its information to the blockchain. This information is analyzed by the blockchain (e.g., a smart contract) that, in case the detected malware has propagation capabilities, triggers the *emergency phase*. This aims to send to all devices in the network a command to execute the closing and changing ports (CCP) action with the aim of closing the ports that a malware uses. In case a service uses the same port, it changes it to another random port. For example, if a device D is infected with malware $Mirai$, the strategy will suggest closing and changing ports (e.g., 23, 2323) that $Mirai$ is using for propagation and stopping all connections with the infected device D.

The second phase is the *healing phase*, which aims at eradicating the malware's malicious actions with the lowest possible impact on the system's operability. To this end, we select a healing strategy based on both malware and process features. In the following, we describe the decision process.

Malware-based decision. The first dimension that MalCon considers to select the proper mitigation actions is the malicious actions the malware could perform, as well as its propagation capability. Based on our literature review, we identified a set of mitigation actions for each malware malicious action. Table 2 summarizes the healing actions that should be taken by considering only malware characteristics. These are defined in terms of malicious actions (see Section 2.1) the malware can perform (in columns) and its propagation capability (in rows). For instance, in the case of a malware that monitors a device and has propagation capabilities (i.e., cell (1,4)), the suggested strategy is $\{CCP, DLF\}$, that is, deleting malware files and closing and changing ports it uses.

In the case of malware performing multiple malicious actions, as healing actions, we select the union of mitigation actions corresponding to each malware malicious action. For example, let us consider a malware $m_1 = \{\{DLF, CR\}, no\}$, that deletes files, consumes resources, and does not have propagation capabilities. The strategy derived from Table 2 for m_1 is the union of cells (2,2) and (2,3), which results in $\{DLF, RBT\}$. This implies that the returned actions are: deleting the malicious file and rebooting the device.

Process-based decision. The key idea of the proposed healing strategy is to select the actions able to eradicate the malware (i.e., malware-based decision) while, at the same time, preserving as much as possible the system's operability. This mainly depends on the process features (aka, availability of a replica and rebooting tolerance). Therefore, among all processes in execution on the infected device, we consider the features of the process with the highest priority. The possible actions that can be taken based on process features are depicted in Table 3, where columns represent the process features (i.e., replication, rebooting), rows model whether the process has or not that feature, whereas cells denote possible actions to be implemented, respectively. The final set of actions is obtained as the intersection among the set of actions suggested for each process feature. We adopt the intersection to ensure that the process will not be interrupted if this impacts the operability. For example, suppose the process has a replica (i.e., cell(1,1)) and does not tolerate rebooting (i.e., cell(2,2)). In that case, the possible actions are {CCP,DLF}: closing and changing ports and deleting files, respectively, since rebooting the device will cause the interruption of a high-priority process.

Example 1. Let us consider a process without a replica that does not tolerate rebooting. According to Table 3, the returned actions are deleting files and closing ports (i.e., $\{CCP, DLF\}$), which are the result of the intersection of $\{CCP, DLF, RBT\}$ (for replication unavailability) and $\{CCP, DLF\}$ (for rebooting intolerance). In case the process tolerates rebooting, but it does not have a replica, the available actions are rebooting, deleting files, and closing ports (i.e., $\{CCP, DLF, RBT\}$). Indeed, in this case, the device can tolerate interruption for a short time (i.e, time of rebooting) without compromising the system's operability. Finally, if a process has a replica, the available actions are: rebooting, deleting files, and closing ports $\{CCP, DLF, RBT\}$, since the system's operability will not be compromised regardless of the chosen actions because there is always a backup replica.

Healing strategy selection. The final healing strategy, hereafter S_h, is obtained by considering the actions identified by both the malware-based decision process, denoted in what follows as S_m, and the process-based decision process, denoted in what follows as S_p. S_h is the intersection between actions in S_m and in S_p. The intersection ensures that if an action in S_m affects the system operability (i.e., rebooting), and the process does not tolerate it, this action is removed from the final healing strategy S_h. As such, we combine S_m and S_p to have the lowest impact on the operability of the system, by mitigating, at the same time, the different malware malicious actions.

Example 2. Let us consider a malware $m = \{\{DLF, ST\}, yes\}$, that propagates, deletes files, and sends traffic to an external entity. Then, let us consider a process $\rho = \{no, yes\}$ running on a device infected by m, which means that there are no replicas for this process, and it tolerates rebooting.

To determine the healing strategy S_h, we first derive the actions S_m. These are defined as the union of all mitigation actions corresponding to m malicious actions (i.e., the union of cells (1,2) and (1,5) in Table 2). Thus, $S_m = \{CCP, DLF\} \cup \{CCP, DLF, RBT\} = \{CCP, DLF, RBT\}$. Actions in S_p are the intersection of all the actions corresponding to ρ's features (i.e., the intersection of cells (2,1) and (1,2) in Table 3). Thus, $S_p = \{CCP, DLF, RBT\} \cap \{CCP, DLF, RBT\} = \{CCP, DLF, RBT\}$. Therefore, $S_h = \{CCP, DLF, RBT\} \cap \{CCP, DLF, RBT\} = \{CCP, DLF, RBT\}$, meaning that to stop the propagation of m we close ports, delete the malware file, and reboot the device.

2.3. Strategies' execution verification

In order to secure the network, we need to make sure that the suggested actions in the emergency and healing phases are indeed executed

[6] https://www. trendmicro.com/vinfo/us/threat-encyclopedia/
[7] https://threats.kaspersky.com

Table 4
Strategies and their corresponding checks.

	Checks
FRMT	Ping the device w_f seconds after submitting the strategy.
RBT	Ping the device w_r seconds after submitting the strategy.
DLF	–
CCP	Ping the device in the corresponding port.

Table 5
MalCon's transactions.

Transaction name	Definition
Peer identity	$pi_{tx} = \langle p_{id}, e, pk, addr, env \rangle$
Peer profile	$pp_{tx} = \langle p_{id}, p_{cr}, t \rangle$
Action	$a_{tx} = \langle a_{id}, cmd, env, t \rangle$
Strategy	$s_{tx} = \langle s_{id}, acts, t \rangle$
Malware	$m_{tx} = \langle m, dt, p_{inf} \rangle$
Election request	$e_{tx} = \langle e_{id}, s, t, p_{inf} \rangle$
Voting	$v_{tx} = \langle v_{id}, c_{id}, e_{id}, er, t \rangle$
Execution confirmation	$ec_{tx} = \langle p_{id}, e_{id}, v, s, t \rangle$

by the IoT devices. For this purpose, in the strategies' execution verification phase, we perform a couple of checks for each suggested action. Table 4 shows the steps followed by the privileged peers to check if an unprivileged peer executed the suggested strategy (see Section 3.4 for more details). The only action that cannot be directly checked is DLF, since privileged peers do not have access to unprivileged devices, they cannot deterministically check that a device deleted a file. However, for the purpose of malware containment, not deleting a malicious file is dangerous for the network only if the malware can propagate. To this end, the propagation ability of the malware is mitigated in the emergency strategy execution phase by closing and changing ports. Thus, the effect of not being able to check if an infected device actually deleted a file is only affecting the device itself and not its neighbors or the whole network. It is worth noting that the waiting periods (i.e., w_f and w_r) specified in the table are configurable by the deployer of the system.

3. Blockchain-based containment

In this section, we discuss the implementation of the proposed containment procedures over the blockchain. We recall that our containment phases are emergency, healing, and strategies' execution verification. First, the emergency phase aims to proactively stop the propagation of malware in the network. Second, the healing phase aims to eradicate the malware from the infected device. Finally, the strategies' execution verification aims at keeping the system safe and operating by making sure that the execution of strategies is done properly.

MalCon leverages two "block-chains" that complement each other. Hyperledger Fabric[8] hosts the smart contracts needed for strategy selection. Here, only privileged devices can execute these smart contracts. The unprivileged peers can read from Hyperledger Fabric's ledger and submit transactions, but they cannot participate in the consensus. Once selected, the strategy shall be executed only by one privileged peer. For this purpose, we implement a voting process to choose the privileged peer that will execute the strategy. The voting process is run on IOTA Tangle,[9] which is used to initiate elections and share votes. We exploit the real-time capabilities of IOTA Tangle with *WebSockets* to ensure a fast voting process.

We recall that MalCon is a containment solution, so it is designed to be integrated with a detection system able to provide the malware information (e.g., ability to propagate, malicious actions, ports used for propagation). We assume that unprivileged peers have a detection system in place to report malware. We assume that the detection system is installed either in the IoT device (e.g., a Raspberry PI) or in an edge device that monitors a set of computationally-limited IoT devices. MalCon can be run at the edge since it is not a heavy process, as shown in Section 5.3.2. Unprivileged peers interact with the blockchain through HTTP requests that are lightweight and do not require special software/hardware. They do not run the blockchain client, and they do not perform any blockchain-related activity (e.g., participating in consensus, ordering transactions, storing a copy of the ledger, etc.). Moreover, we assume that privileged peers are secured with strong passwords, and intrusion detection systems are in place. The privileged peers in MalCon can be workstations or servers. Table 5 summarizes MalCon's transactions that will be explained in the remainder of this section, whereas Table 6 explains the used notations.

Table 6
MalCon's notations.

Notation	Definition
w_r	Waiting time for checking rebooting actions
w_f	Waiting time for checking formatting actions
p_{id}	Peer's identifier
e	Peer's endpoint
pk	Peer's public key
$addr$	Peer's IOTA address
p_{cr}	Most critical process
t	Timestamp
s_{id}	Strategy's identifier
$acts$	Strategy's actions
m	Malware
dt	Malware's detection time
p_{inf}	Infected peer's identifier
e_{id}	Election's identifier
s	Election's suggested strategy
v_{id}	Voter's identifier
c_{id}	Candidate's identifier
er	Election's round
v	Total votes that a candidate got in an election
cmd	Command to be executed in the peer)
env	Peer's environment (i.e., OS and architecture)

3.1. System initialization and update

In this section, we introduce the basic transactions used to initialize and update the system. These transactions are peer identity, peer profile, and strategy transactions (see Table 5 for their format).

The *peer identity transaction* is submitted to Hyperledger Fabric by all peers in the network when joining. It encodes information about a peer, i.e., its identifier, the endpoint where it will receive all system communications, its public key, its IOTA address, and information about its environment (i.e., operating system and architecture). In contrast, the *peer profile transaction* is submitted to Hyperledger Fabric by all peers of the network to communicate information about their most critical process (that is, its tolerance to rebooting and replica availability), and the timestamp of the last update. A new transaction of this type is sent by a peer whenever there is a change in the most critical process it runs.

Action transactions are submitted to Hyperledger Fabric by one privileged peer upon agreement among all privileged ones. They contain information about each supported mitigation action, namely the action's identifier (i.e., symbols such as the ones defined in Table 1), the command to be executed, the environment (e.g., operating system), and the timestamp when it was published. This allows MalCon to be flexible and customizable since the deployer of the system can add actions depending on the devices that she has in her network, making MalCon suitable to be run in heterogeneous settings.

Finally, a *strategy transaction* is submitted to Hyperledger Fabric by one privileged peer upon agreement among all privileged ones. It holds

[8] https://www.hyperledger.org/use/fabric
[9] https://www.iota.org

information about a specific strategy, namely its identifier, its actions (among those defined in action transactions), and the timestamp when it was published. This allows the system to cope with new malware when they appear.

3.2. Voting process

We recall that a privileged peer is elected to execute the suggested healing and emergency strategies. We assume that the number of privileged peers is known to all of them. In addition, we assume that an election's round is valid if and only if all the privileged peers voted.

In order to initiate an election to elect the privileged peer who will execute the suggested strategies, an *election request transaction* is submitted to IOTA Tangle by the *Containment Smart Contract*, explained in detail in the next section. Each election request transaction (cfr. Table 5) has an identifier, a set of actions that were chosen by the *Containment Smart Contract*, a timestamp denoting when it is submitted, and the infected unprivileged peer's identifier.

When an *election request transaction* is submitted, all privileged peers vote randomly on a candidate excluding themselves, by submitting to IOTA Tangle a *voting transaction* v_{tx} (cfr. Table 5). The *voting transaction* consists of the voter's identifier, the candidate's identifier, the election's identifier, the election's round, and the timestamp when the *voting transaction* is submitted. Each privileged peer then counts the votes shared in the blockchain for a specific round and compares it with the number of privileged peers. If all privileged peers have voted, they download a copy of all voting transactions for that specific election's round. Then, the votes are counted locally by all peers for each election round. They locally sort the candidates based on their votes to choose the privileged peer who win the election (i.e., the executor). If a peer discovered that he/she is the winner through counting votes, it submits an *execution confirmation transaction* ec_{tx}. In case of a tie between two or more candidates, other rounds are automatically initiated where privileged peers vote on the candidates who got advanced to the next rounds (i.e., candidates who had a tie in the previous round). This process is repeated until there is no tie, and only one candidate wins the election. When the privileged peer who won the election submits an *execution confirmation transaction*, the other privileged peers check the result of the election to see if it matches the claim. If it does, they generate and sign one-time usage tokens with their private keys and send them to the executor via its endpoint. The latter gathers strictly more than $\frac{1}{2}$ of the privileged peers' tokens (including its own token). Then, it sends a request containing the strategies to be executed with the gathered tokens to the infected unprivileged peer via its communication endpoint. The latter verifies the signatures of the tokens with the privileged peers' public keys. If they are valid, it executes the strategies.

The above-described election procedure is adopted in all MalCon phases, so the selection of the privileged peer who will coordinate the execution of strategies is the same for emergency, healing, and strategies' execution verification.

3.3. Emergency and healing

Regarding the emergency phase, when an unprivileged peer is attacked by malware, its detection system provides the malware information which is then sent as a malware transaction m_{tx} (see Table 5) to Hyperledger Fabric. The transaction holds the malware characteristics, its detection time, and the identifier of the peer that detected it (i.e., infected peer). When a malware transaction is submitted, the *Containment Smart Contract*, described in Algorithm 1, is executed. The *Containment Smart Contract* is used for handling both the emergency and healing phase. Fig. 1 shows the sequence diagram for selecting strategies in MalCon.

We recall that, in the emergency phase, there is only one action to be executed: closing and changing the ports specified in the malware transaction, in case the malware has propagation capabilities. Therefore, this step only requires selecting a privileged peer to execute the action. This selection is made by a voting process (cfr. Section 3.2) triggered by the *Containment Smart Contract* (lines 2–4 of Algorithm 1). The smart contract checks if the malware can propagate (line 2); if so, it submits to IOTA Tangle an election request (i.e, transaction e_{tx}, defined in Table 5), with closing and changing ports as action and the corresponding ports to be closed (line 3). It is worth noting that `submitElectionReqTx()` function takes as input the strategy and the ports needed to be closed if CCP is suggested, otherwise the ports are an empty list. This function takes these inputs and forms the election request transaction e_{tx} defined in Table 5. Then, it submits e_{tx} to IOTA.

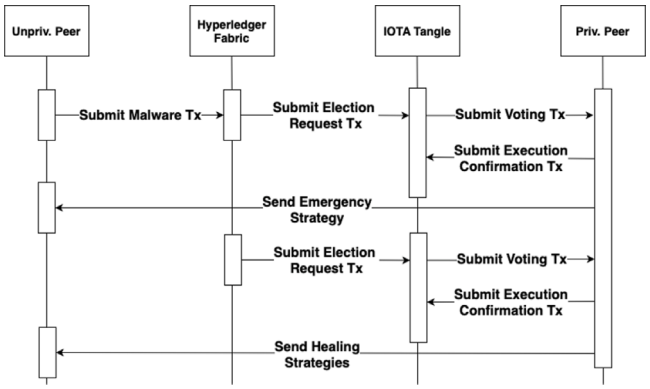


Fig. 1. Strategies execution sequence diagram.

Algorithm 1 Containment smart contract run by privileged peers

Input: Malware Transaction m_{tx}
1: $malware \leftarrow getMalware(m_{tx})$
2: **if** $malware.PRP == yes$ **then**
3: $\quad submitElectionReqTx([CCP], malware.ports)$
4: **end if**
5: $p_{cr} \leftarrow getCriticalProcess(p_{inf})$
6: $M_m \leftarrow Malwarebased Healing Decision(malware)$
7: $M_p \leftarrow Processbased Healing Decision(p_{cr})$
8: $Healing Strategy \leftarrow M_m \cap M_p$
9: $submitElectionReqTx(Healing Strategy, [\,])$

Once the execution of the emergency phase (lines 2–4) is done, the smart contract continues with the execution of the healing phase (lines 5–9). First, it gets the most critical process of the infected peer (line 5). Then, it determines the mitigation actions to be executed on the basis of the malware features, following the approach described in Section 2.2 (cfr. Table 2) (line 6). Similarly, it determines additional mitigation actions on the basis of the characteristics of the most critical process, (see Section 2.2 and Table 3) (line 7). Finally, it intersects the malware-based and process-based suggested actions to get the final healing strategy (line 8), as explained in Section 2.2. Then, it submits an election request transaction containing the strategy (line 9).

3.4. Strategies' execution verification

When the winner of an election submits an execution confirmation transaction, the strategies' execution verification procedure in Algorithm 2 is triggered. This procedure is deployed at the level of each privileged peer. It starts by retrieving information about the infected peer's detection endpoint (line 1) and initializes the checks (line 2). Then, it checks if the infected unprivileged peer has executed the strategies, following the procedure described in Algorithm 2 (lines 3–20). For closing and changing ports action, it checks if the specified ports are closed or not (lines 4–8). For formatting and rebooting, it

waits for different periods (i.e., w_f and w_r) depending on the action (in our implementation, w_f = 120 s and w_r = 30 s), then it pings the unprivileged peer to see if it is alive (lines 9–20). If the device does not respond within the given time period, the ping check is considered a failed check. If some checks fail (including a failed ping request), it disables the infected peer from any interaction with the blockchain and sends a notification to the system admin to take the necessary mitigation actions (e.g., physically formatting it or removing it from the network) (lines 21–24). One elected privileged peer per period φ (specified by the deployer) is responsible for sending a notification to the admin and disabling the infected peer from submitting any transaction to the blockchain. All privileged peers send the results of their checks to the elected privileged peer at the end of each period φ.

Algorithm 2 Strategies' execution verification procedure run by privileged peers

Input: Execution Confirmation Transaction ec_{tx}

```
1: detectionEndpointPort ← getDetectionEndpointPort(ec_tx.infectedPeer)
2: checks ← 0
3: for MitigationAction in ec_tx.strategy do
4:   if MitigationAction == CCP then
5:     port ← ec_tx.strategy.CCP.port
6:     if ping(infectedPeer, ec_tx.strategy.CCP.ports) == True then
7:       checks ← checks + 1
8:     end if
9:   else if MitigationAction == RBT then
10:    wait(w_r)
11:    if ping(ec_tx.infectedPeer, detectionEndpointPort) == True then
12:      checks ← checks + 1
13:    end if
14:  else if MitigationAction == FRMT then
15:    wait(w_f)
16:    if ping(ec_tx.infectedPeer, detectionEndpointPort) == True then
17:      checks ← checks + 1
18:    end if
19:  end if
20: end for
21: if checks < ec_tx.strategy.size() then
22:   SendAdminNotification()
23:   DisablePeer(infectedPeer)
24: end if
```

4. Security analysis

In this section, we discuss our assumptions and possible attacks that unprivileged peers could perform. In addition, we show how we mitigate them to keep the system secure.

We recall that in MalCon we focus on containing malware in IoT environments where organizations exchange threat information for effective containment. We assume each organization has an equal number of privileged peers (e.g., admins) that represent them in all MalCon operations, and different numbers of unprivileged peers, that is, IoT devices. The privileged peers participate in consensus under the Byzantine Fault Tolerant model [11], so we assume that $\frac{2}{3}$ of them are honest. We assume that they are secured with malware detection and intrusion detection mechanisms. Thus, we focus on attacks on unprivileged peers. We note that the decisions taken by privileged peers are a result of running smart contracts. So, a privileged peer cannot make a wrong decision because a consensus will not be reached on it. However, we can consider the case where a privileged peer decides to manipulate the output of the smart contracts (it is from the $\frac{1}{3}$ that is not honest). In other words, it does not follow the protocol and acts as a malicious peer. Although this is against the assumptions of our protocol, we show in what follows that the effect of such malicious activities is limited. First, a privileged peer cannot target a specific device since it is randomly selected, and it knows ahead of time neither the peer that is infected nor the strategies it will execute. Second, the best it can do is a denial of service to an already infected device. For example, the infected device reported that malware is using port 5555, and the privileged peer asked to close port 5555 and port 7878 which is used for a legitimate service. This action can trigger the system administrator since the device will not be operational. Thus, the attack itself cannot be done in a stealthy way.

We assume that the majority of unprivileged peers is honest, but strictly less than the majority of unprivileged peers could be compromised. Such assumptions were adopted in other work involving blockchain and IoT, such as [12,13]. In MalCon, possible attacks that unprivileged peers could perform are: (1) failing to execute strategies, (2) repetitively submitting malware transactions, (3) submitting fake malware transactions, and (4) not submitting malware transactions.

Failing to execute strategies. Unprivileged peers receiving a strategy to be executed could choose not to execute it. We detect this attack for all supported actions, except DLF (delete a file) through the strategies' execution verification phase, described in Section 3.4. After each execution, a set of checks are performed to determine if the infected device has executed the suggested strategies (this phase is implemented by Algorithm 2 – lines 3–20). If the checks fail, we disable the device to stop any interaction with the blockchain and notify the system admin to take the necessary actions (Algorithm 2 – lines 21–24). It is worth noting that for DLF, we cannot check if the malware executable is deleted or not. However, the malicious file is dangerous for the network only if the malware can propagate. If the malware propagates, closing and changing ports CCP actions will be suggested (Algorithm 1 – lines 2–5). The execution of the latter can be verified, so the malware will not propagate to other devices.

Repetitively submitting malware transactions. Unprivileged peers could repetitively submit transactions in order to flood the system and cause a denial of service. In order to detect this attack, we introduce a threshold t_d that represents the maximum number of allowed submissions in a period φ. So, after each submission, we check if the count of submissions in a period φ exceeds t_d. If so, we disable the device and notify the system admin to take the necessary actions.

Submitting fake malware transactions. Unprivileged peers could submit fake malware transactions to cause a denial of service for specific services that the honest unprivileged peers are running. However, for this attack, the honest unprivileged peers will only execute one action (i.e., closing and changing ports) because this action will allow the device to provide the service without interruption (as discussed in Section 2.2 and done by Algorithm 1 – lines 2–4). So, if there is a service running in that specific port, they will just forward it to another random port, so they will not be affected. On the other hand, the infected unprivileged peer will have to execute the emergency and healing strategy that can be checked as discussed earlier. In addition, in case it submits several fake malware transactions, it will be detected as discussed in the repetitive submissions attack.

Not submitting malware transactions (i.e., free riding). Unprivileged peers could refrain from submitting malware transactions. This attack could result in malware spreading in the network. However, since the majority of the unprivileged peers are honest, any malware circulating in the network will be detected by an honest unprivileged peer. We show the simulation and analysis of free-riding attacks in Section 5.3.3.

5. Experiments

We test our solution's effectiveness in stopping malware from propagation and its performance in terms of containment time. We adopt two metrics: (i) the difference between the number of infected and uninfected devices with and without our solution, and (ii) the average time from the malware detection to the moment of strategies execution by the infected peers. In addition, we compared our solution with the proposal described in [14], since, to our knowledge, it is the only paper that does a similar experiment in a real-life setting.

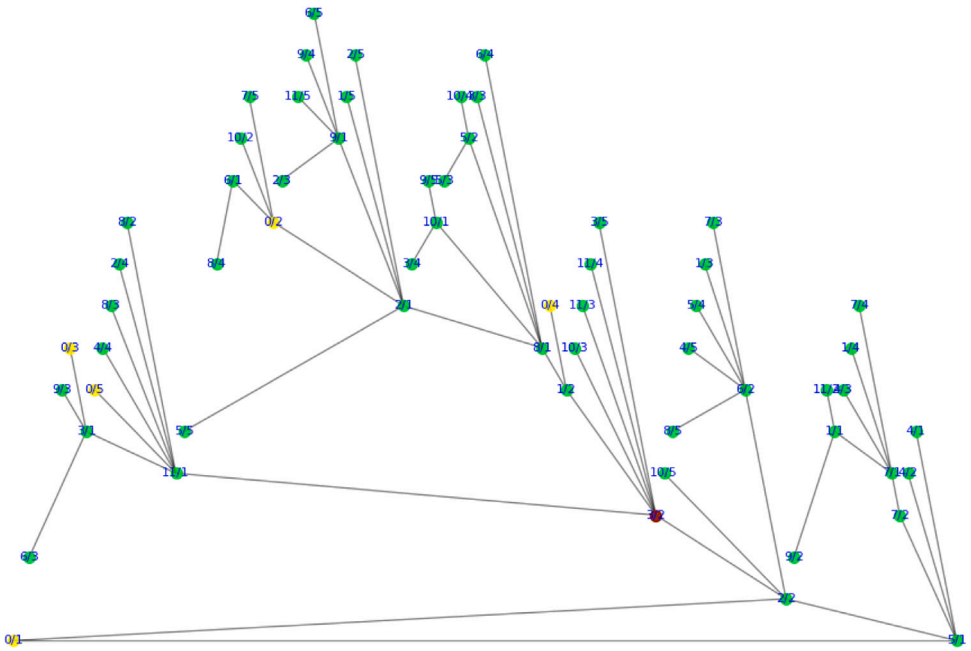


Fig. 2. The smart city network topology for Scenario #1 (The node IDs are in the format peerID/organizationID).

5.1. Environment

We have implemented our solution using Hyperledger Fabric 2.2 locally on an octa-core Intel Core i7 3.6 GHz CPU with 16 GB of RAM and with a Python client that connects to IOTA 1.0 DevNet through PyOTA library. In Fabric, we adopted the *OR Endorsement Policy* which implies the random selection of one privileged peer to run the smart contracts. All peers are simulated with Docker containers with 512 MB of RAM.

To test our approach, we used two malware: Mirai [1] and an in-house Mirai-like malware. Since we did not find a live Mirai executable to simulate the actual spreading, we built the source code extracted from a public repository[10] with our C&C server. On the other hand, the in-house Mirai-like malware infects devices by brute-forcing credentials from Mirai's words list, using Telnet protocol with multi-threading, similar to Mirai. The in-house Mirai-like malware was designed for quick infection, similar to the in-house malware built by [14]. We used Mirai for all the experiments, except for the comparison with [14]. To challenge our solution, all unprivileged devices are configured with a password randomly selected from the Mirai words list. This allows fast malware to spread in the network, which is considered a worst-case scenario. In contrast, the privileged devices were configured with strong passwords.

5.2. Experimental settings

To test our solution under different scenarios, we have considered the following factors:

Network's connectivity. The number of links among nodes impacts the spreading speed. For example, in the case of a fully connected network, an infected node quickly spreads the malware to the network. Thus, we consider a fully connected network as an extreme case for testing MalCon.

Security level of passwords. Using default passwords or common weak passwords makes the malware spread faster since they brute force devices' credentials using a hardcoded word list. Thus, adopting weak passwords from botnets' words list is an extreme case for testing MalCon.

Number of privileged peers. MalCon relies on a voting process among privileged peers, where a tie among candidates would imply additional voting rounds. Thus, the number of privileged peers impacts the containment time. Thus, a large number of privileged peers is an extreme case for MalCon performance.

5.2.1. Scenarios

We tested MalCon in three different scenarios, adapted from real-life settings.

Scenario #1: Smart City. This scenario is based on a realistic IoT setting involving multiple organizations (e.g., vendors, Internet Service Providers, etc.). Here, the organizations create a consortium that agrees to exchange information about malware threats in a decentralized fashion. This scenario's topology consists of 5 organizations and 60 devices, as shown in Fig. 2, where the infected device, privileged peers, and unprivileged peers are colored in red, yellow, and green, respectively. This network topology is adopted in various works about smart cities [15] and health care systems [16]. This scenario presents several challenges for MalCon performance. The network topology is connected (each device is connected to one or more devices). In addition, the initial device that we infected has the highest number of neighbors (i.e., 7 direct neighbors). Moreover, all the unprivileged peers have weak passwords.

Scenario #2: Fully Connected Network. This scenario is an extreme case where the network is fully connected. So, all devices have 59 adjacent peers.

Scenario #3: Small Network. In this scenario, we adopt the experiment settings in [14]. We contacted the authors of [14] to get the missing experiment's details (e.g., the in-house malware implementation, network topologies, etc.). Their network consists of 20 devices configured with random weak passwords hardcoded in their in-house malware. Regarding network topologies, they adopt random ones, where a device is at least connected to another device.

[10] https://github.com/jgamblin/Mirai-Source-Code

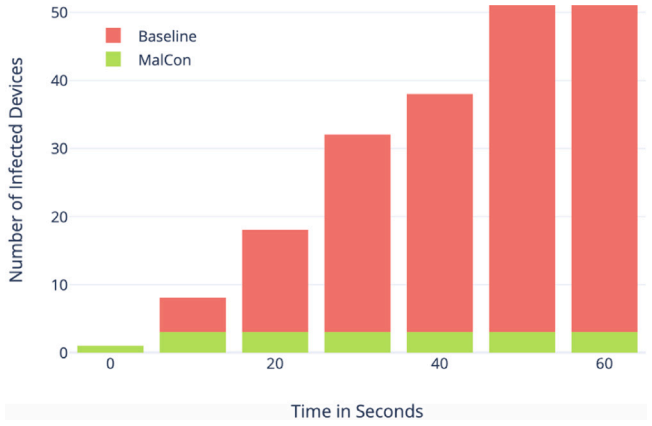


Fig. 3. Number of infected devices, by enabling and disabling MalCon for Scenario #1.

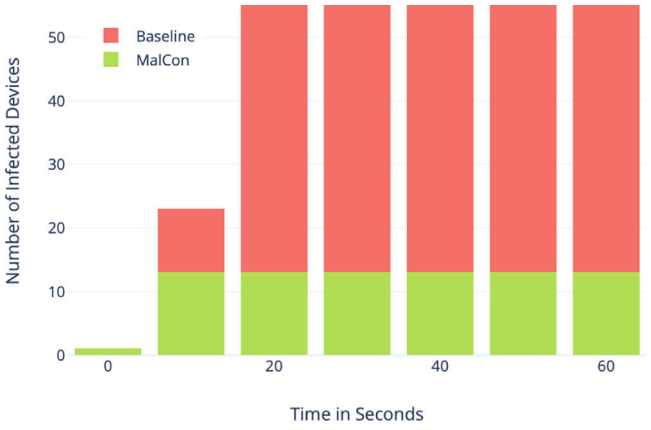


Fig. 4. Number of infected devices by enabling and disabling MalCon for Scenario #2.

5.3. Results

5.3.1. Effectiveness

Scenario #1: Smart City. The baseline for this scenario is the network without MalCon. The goal is to see the difference of infected devices between the baseline and MalCon after injecting Mirai botnet in a device. We infect the unprivileged device with the highest number of direct connections (i.e., 7 in the network topology shown in Fig. 2). The average time of brute-forcing credentials by Mirai is 7 s. After 50 s, 51 over 60 devices were infected. The 9 non-infected devices consist of 5 privileged peers that, by design, have strong passwords; and 4 unprivileged peers, that were connected only to a privileged peer, so the infection did not reach them because of the strong passwords. On the other hand, with MalCon enabled, only 3 devices were infected, including the device where we injected Mirai intentionally. The two infected devices had passwords that happened to be the first ones in the hardcoded passwords list used by Mirai. Fig. 3 shows the comparison between enabling and disabling MalCon (i.e., baseline).

Scenario #2: Fully Connected Network. The Mirai botnet was injected on a random unprivileged device since the network is fully connected, so each device has 59 direct neighbors. As in the previous experiment, the baseline is the network without MalCon. The infected devices after 10 s are 23. After 20 s, 55 devices were infected. After enabling MalCon, only 13 devices were infected, and the infection happened in the first 10 s and then stopped. Fig. 4 shows the comparison between the baseline and MalCon enabled, in a fully connected network.

Scenario #3: Small Network. We compare our solution with the solution discussed in [14]. The authors in [14] infected at each try two random devices in the network with an in-house developed malware. They repeated the experiment 1000 times with random network topologies at each try with the condition that any device is connected at least to another device where each try lasts 40 s. Then, they calculate the average rate of infection by dividing the total number of infections counted in the 1000 tries by the total number of tries. We tested our solution with the same setting using our in-house Mirai-like malware and random weak passwords from the Mirai words list. While the average infection rate of [14] is 1.27, our average rate of infection under the same conditions was 1.05, which makes MalCon a suitable solution for malware containment.

5.3.2. Performance

We conducted an experiment to check the average time to contain malware with a different number of organizations. For this experiment, our average containment time results show that it takes 0.98 s and 5.19 s to contain malware in a network of 3 and 20 organizations, respectively. It is worth mentioning that the number of unprivileged nodes does not affect the average containment time.

In order to test the feasibility in a real-world setting, we further tested MalCon in a network of 7 simulated devices and 3 Raspberry PIs, to measure MalCon's resource consumption. We used Raspberry PI 2 Model B with a Quad Core Cortex-A53 CPU with ARMv7 Architecture (32-bit) and 1 GB of RAM for all the devices. We selected this specific model since it has limited computational power compared to its successors (e.g., Raspberry Pi 4 Model B). An instance of MalCon was deployed on the Raspberry PIs, while other IoT devices in the network were simulated using Docker containers on a server. To detect malware, we used ClamAV,[11] an open-source antivirus engine, on all devices. Our results show that MalCon's CPU consumption ranges between 2.47% and 25.0% with an average of 7.3%. In addition, RAM usage varies from 2.53% to 88.18% with an average of 13.9%. The peaks in resource usage were observed while performing a full scan of the disk. We selected a full scan since it is a heavy process that overloads the memory. It is worth noting that a full scan is not needed to keep MalCon operational. MalCon expects a detection system in place as a hidden process that monitors the files and scans new files. This task is performed by the devices with a reasonable resource consumption as shown earlier. For MalCon, the devices do not run the blockchain client (e.g., they do not participate in consensus and they do not have a copy of the ledger). They only interact with blockchain endpoints through HTTP requests (e.g., to report malware). Thus, MalCon as a standalone application is lightweight and can be used alongside an antivirus in computationally limited IoT devices.

We iterate that MalCon supports different computing paradigms. In case there is an edge or a fog node that controls low-power IoT devices, these nodes will do the detection on their behalf. Otherwise, the IoT device itself can do it. We supported this claim with a limited Raspberry Pi from the old generation with only 1 GB of RAM. In addition, the protocol itself is lightweight since it is only an HTTP server, so even the very limited IoT devices can support it. Regarding the commands, they can be tailored depending on the IoT devices

[11] http://www.clamav.net/

environment (i.e., operating system and architecture) as discussed in Section 3.

5.3.3. Free riding simulation and analysis

In order to evaluate the effect of devices that refrain from submitting malware transactions to MalCon, we have simulated a network with different percentages of malicious nodes, namely 10%, 30%, and 49%. We have simulated a network of 60 devices with 10 randomly generated topologies where a device is a least connected to another device. It is worth mentioning that this attack is only a danger for the network if the devices are infected and they refrain from submitting the detection to MalCon. So, in our simulation, we assume that they are infected, and are trying to infect other devices. For 10% of malicious nodes (6 malicious devices out of 60), the 6 devices were identified for not submitting malware transactions almost immediately in about an average of 0.06 s. For 30% of malicious nodes (18 malicious devices out of 60), the average identification time is 1.12 s. Finally, for 49% of malicious nodes (29 malicious devices out of 60), the average identification time is 1.89 s.

In order to better interpret and quantify these results, we have done a probability analysis for this simulation. Based on the randomly generated topologies, the average number of neighbors per node is 2. In our network, we have two types of nodes: honest and malicious. In order to avoid immediate identification, a malicious node needs to have two malicious nodes as neighbors (even if this information is not available for the nodes initially). Let M and N be the number of malicious devices and the total number of devices, respectively. The probability of having two malicious neighbors $P(m)$ is shown in Eq. (1).

$$P(m) = \frac{\binom{M-1}{2}}{\binom{N-1}{2}} \tag{1}$$

For 10% malicious nodes, $P(m_{10}) = 0.58\%$. For 30% malicious nodes, $P(m_{30}) = 7.94\%$. Finally, for 49% malicious nodes, $P(m_{49}) = 22.09\%$. So, the probability analysis confirms the simulation results. Thus, MalCon is able to operate where strictly less than $\frac{1}{2}$ of devices refrain from submitting malware transactions.

5.3.4. Discussion

The conducted experiments show that MalCon is suitable for malware containment in IoT networks since it significantly reduces the number of infected devices in the network. For instance, even in a fully connected network with weak passwords, only 13 devices out of 55 were infected. In addition, in a typical IoT setting, only three devices were infected. The average infection rate in our solution is 1.05 compared to [14] which has 1.27. On the other hand, the average containment time from the moment of the detection to the moment of executing strategies varies from 0.98 s to 5.19 s for 3 and 20 organizations, respectively. So, MalCon is suitable for large deployments because unprivileged peers do not affect its performance. It is worth noting that the containment time is highly affected by IOTA's latency and the geolocation of the nearest node. MalCon uses IOTA's 1.0 DevNet which has lower throughput compared to the mainnet.

6. Conclusion

In this paper, we have presented MalCon, a blockchain-based malware containment framework for IoT. It aims to limit the damages that malware can do in IoT networks by proactively stopping them from propagation while keeping the network operational. It exploits collaboration among different organizations to share malware information. It suggests tailored strategies for all devices to prevent propagation based on the malware's propagation scheme and the processes that they run. As a part of our future work, we aim to migrate MalCon to a permissionless blockchain and analyze the attack vectors that come with such migration. In addition, we plan to remove the human-in-the-loop when contacting the system administrator in some emergency cases.

CRediT authorship contribution statement

Ahmed Lekssays: Conceptualization, Methodology, Software, Writing – original draft. **Barbara Carminati:** Conceptualization, Methodology, Writing – review & editing, Supervision. **Elena Ferrari:** Conceptualization, Methodology, Writing – review & editing, Supervision.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

No data was used for the research described in the article.

Acknowledgments

We thank the anonymous reviewers for the useful feedback and insights that helped us improve the quality of this manuscript.

Funding

This work has received funding from the Marie Sklodowska-Curie Innovative Training Network Real-time Analytics for Internet of Sports (RAIS) supported by the European Union's Horizon 2020 research and innovation programme under grant agreement No 813162. The content of this paper reflects only the authors' view and the Agency and the Commission are not responsible for any use that may be made of the information it contains.

Code availability

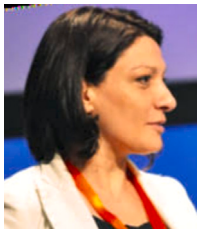
The authors declare that they have open-sourced all the parts of MalCon on https://github.com/Lekssays/malcon.

References

[1] M. Antonakakis, T. April, M. Bailey, M. Bernhard, E. Bursztein, J. Cochran, Z. Durumeric, J.A. Halderman, L. Invernizzi, M. Kallitsis, et al., Understanding the mirai botnet, in: 26th {USENIX} Security Symposium, {USENIX} Security 17, 2017, pp. 1093–1110.
[2] S. NIST, 800-83, in: Guide to Malware Incident Prevention and Handling, 2013.
[3] S.M.P. Dinakarrao, H. Sayadi, H.M. Makrani, C. Nowzari, S. Rafatirad, H. Homayoun, Lightweight node-level malware detection and network-level malware confinement in IoT networks, in: 2019 Design, Automation & Test in Europe Conference & Exhibition, DATE, IEEE, 2019, pp. 776–781.
[4] A. Malvankar, J. Payne, K.K. Budhraja, A. Kundu, S. Chari, M. Mohania, Malware containment in cloud, in: 2019 First IEEE International Conference on Trust, Privacy and Security in Intelligent Systems and Applications, TPS-ISA, IEEE, 2019, pp. 221–227.
[5] N. Idika, A.P. Mathur, A Survey of Malware Detection Techniques, Vol. 48, No. 2, Purdue University, 2007, Citeseer.
[6] I.A. Saeed, A. Selamat, A.M. Abuagoub, A survey on malware and malware detection systems, Int. J. Comput. Appl. 67 (16) (2013).
[7] M. Damshenas, A. Dehghantanha, R. Mahmoud, A survey on malware propagation, analysis, and detection, Int. J. Cyber-Secur. Digit. Forensics 2 (4) (2013) 10–30.
[8] K. Angrishi, Turning internet of things (IoT) into internet of vulnerabilities (IoV): Iot botnets, 2017, arXiv preprint arXiv:1702.03681.
[9] M. Humayun, N. Jhanjhi, A. Alsayat, V. Ponnusamy, Internet of things and ransomware: evolution, mitigation and prevention, Egypt. Inform. J. (2020).
[10] L. Jaramillo, Malware detection and mitigation techniques: lessons learned from Mirai DDOS attack, J. Inform. Syst. Eng. Manag. 3 (3) (2018) 19.
[11] E. Androulaki, A. Barger, V. Bortnikov, C. Cachin, K. Christidis, A. De Caro, D. Enyeart, C. Ferris, G. Laventman, Y. Manevich, et al., Hyperledger fabric: A distributed operating system for permissioned blockchains, in: Proceedings of the Thirteenth EuroSys Conference, 2018, pp. 1–15.
[12] H.B. Desai, M.S. Ozdayi, M. Kantarcioglu, Blockfla: Accountable federated learning via hybrid blockchain architecture, in: Proceedings of the Eleventh ACM Conference on Data and Application Security and Privacy, 2021, pp. 101–112.

[13] V. Mugunthan, R. Rahman, L. Kagal, BlockFLow: Decentralized, privacy-preserving, and accountable federated machine learning, in: International Congress on Blockchain and Applications, Springer, 2021, pp. 233–242.
[14] S.M.P. Dinakarrao, X. Guo, H. Sayadi, C. Nowzari, A. Sasan, S. Rafatirad, L. Zhao, H. Homayoun, Cognitive and scalable technique for securing IoT networks against malware epidemics, IEEE Access 8 (2020) 138508–138528.
[15] C.-L. Zhong, Z. Zhu, R.-G. Huang, Study on the IOT architecture and gateway technology, in: 2015 14th International Symposium on Distributed Computing and Applications for Business Engineering and Science, DCABES, IEEE, 2015, pp. 196–199.
[16] A. Elsts, X. Fafoutis, P. Woznowski, E. Tonkin, G. Oikonomou, R. Piechocki, I. Craddock, Enabling healthcare in smart homes: the SPHERE IoT network infrastructure, IEEE Commun. Mag. 56 (12) (2018) 164–170.

Ahmed Lekssays is a researcher at Qatar Computing Research Institute in Doha, Qatar. He received his Ph.D. in Computer Science from the University of Insubria in Varese, Italy. He received his master's degree in Software Engineering with high honors from Al Akhawayn University in Ifrane, Morocco, where he mainly focused on computer security and malware detection on mobile devices. His research interests are related to computer security and privacy, blockchain, distributed systems, and malware analysis.



Barbara Carminati is full professor Computer Science at the University of Insubria, Italy. Her main research interests are related to security and privacy for innovative applications, like online social networks, cloud computing, semantic web, data outsourcing, XML data sources, Web service, and data streams. On these topics she has published 100+ articles with peer reviews (more than 30 in international journals) in prestigious magazines. From January 2009 to March 2015, Barbara Carminati has been the editor in chief of the Computer Standards & Interfaces journal, Elsevier press. From 2015 she is an associate editor for IEEE Transactions on Dependable and Secure Computing journal, and since 2017, associate editor for IEEE Trans. on Services Computing journal. She is a ACM and IEEE senior member.

Elena Ferrari is a full professor of Computer Science at the University of Insubria (Italy), where she leads the STRICT SociaLab. She received her Ph.D. and M.Sc. degree in Computer Science from the University of Milano (Italy). Her research interests are in the broad area of cybersecurity, privacy, and trust. Current research includes security and privacy for Big Data and Iot, access control, machine learning for cybersecurity, risk analysis, blockchain, and secure social media. Prof. Ferrari has published several books and more than 240 papers in international journals and conference proceedings. She has received several awards, including the 2009 IEEE Technical Achievement Award, the ACM CODASPY Research Award, the ACM SACMAT 10 Year Test of Time Award, an IBM Faculty Award, and a Google Research Award. She is an ACM and IEEE Fellow. Prof. Ferrari currently serves as associate editor in chief of IEEE Internet Computing. She is associate editor for the ACM Transactions on Data Science, IEEE Transactions on Services Computing, Data Science and Engineering, and the International Journal of Cooperative Information Systems. She is the chair of the steering committee of the ACM Symposium on Access Control Models and Technologies and has been the PC chair of several major conferences in the data management and security/privacy fields. She has also repeatedly served on the program committees of all major data management conferences.

Prof. Ferrari's has led several research projects in the field of cybersecurity and privacy, which have been funded by several companies and institutions (e.g., EU Commission, Google, IBM, EOARD/AFOSR, and the Italian Ministry for University and Research). In 2018, she has been named one of the 50 most influential Italian women in tech. More details can be found at http://dawsec.dicom.uninsubria.it/elena.ferrari/.