Virtual IoT HoneyNets to mitigate cyberattacks in SDN/NFV-enabled IoT networks

Alejandro Molina Zarca*, Jorge Bernal Bernabe*, Antonio Skarmeta*, Jose M. Alcaraz Calero†
*Department of Information and Communications Engineering, University of Murcia, Murcia, Spain;
{alejandro.mzarca, jorgebernal, skarmeta}@um.es
†School of Engineering and Computing, University of the West of Scotland, Glasgow, Scotland;
jose.alcaraz-calero@uws.ac.uk

***Abstract*—As the IoT adoption is growing in several fields, cybersecurity attacks involving low-cost end-user devices are increasing accordingly, undermining the expected deployment of IoT solutions in a broad range of scenarios. To address this challenge, emerging Network Function Virtualization (NFV) and Software Defined Networking (SDN) technologies can introduce new security enablers, thereby endowing IoT systems and networks with higher degree of scalability and flexibility required to cope with the security of massive IoT deployments. In this sense, honeynets can be enhanced with SDN and NFV support, to be applied into IoT scenarios thereby strengthening the overall security. IoT honeynets are virtualized services simulating real IoT networks deployments, so that attackers can be distracted from the real target. In this paper, we present a novel mechanism leveraging SDN and NFV aimed to autonomously deploy and enforce IoT honeynets. The system follows a security policy-based approach that facilitates management, enforcement and orchestration of the honeynets and it has been successfully implemented and tested in the scope of H2020 EU project ANASTACIA, showing its feasibility to mitigate cyber-attacks.**

***Index Terms*—Cybersecurity, Security models, NFV, IoT, Security Policies, SDN**

I. Introduction

The Internet of Things comprises billions of heterogeneous devices interacting each other and generating enormous assorted data traffic. Constrained IoT devices strive to perform their tasks in potential hostile environments, whereby security and privacy aspects are even more difficult to address with hardware limitations in terms of computational power, memory and battery. Besides, the low-power wireless connectivity, Machine-to-Machine (M2M) interaction models, low-cost and unattended deployments, as well as the pervasive and dynamic environments sparks new security and privacy threats. Diverse types of evolved cyber-attacks such as for instance, distributed deny of service (DDoS) attacks, exploiting infected IoT bots (e.g. Mirai), are growing in IoT [1].

To address this new kind of IoT-based cyber-attacks, diverse scalable security mechanisms and strategies are emerging to mitigate them [2]. In this regard, honeynets can be used to help countering cyber-attacks in IoT. Honeynets are comprised of network(s) of honeypots, which aim to be used as baseline to detect cyber-attacks (e.g. botnets) and to learn about attackers' behaviors, making them believe that they are accessing to the real network, when they are in fact, accessing to a simulated, controlled and isolated network.

A highly-interaction honeypot (HIH) honeynet can imitate the activities of the production network, by hosting a variety of honeypots and services, to which an attacker is redirected to access to make him waste his time. This can give some extra time for the real system to take proper reaction and countermeasures in order to mitigate cyber-attacks and learn for future attack attempts. HIHs can capture not only the network activity, but also the system activity. However, HIHs can increase resource consumption in large-scale deployments. Honeynets, and in turn, honeypots, are not subject to deploy and implement all the services and functionalities of the production system. Indeed, honeypots usually provide simulations of only certain services to reduce maintenance cost. Unlike HIH, low-interaction honeypot (LIH) honeynet simulates only part of the network and services. So far honeynets have been studied and applied successfully mainly for protecting traditional distributed networks and systems, such as Cloud and Grid infrastructures and services. However, their application to the Internet of things has not yet paid enough attention.

Moreover, the process of configuring, setting-up and enforcing virtual honeynet is a complex and tedious task, that becomes even more complex, when the honeynet needs to be adapted on demand to the pervasive network environment. Therefore, it is necessary to devise an automated framework to configure, deploy and manage a flexible honeynet. In this paper, we present an autonomic virtual IoT honeynet deployment mechanism that relies on Network Function Virtualization (NFV) (NFV) for orchestration. The use of NFV for honeynets management enables an efficient way of deploying virtual HIH, empowering dynamic configuration and reconfiguration while maintaining the HIH configuration equivalent to the real IoT environment deployed in production.

Since an IoT network is composed of different sensor nodes, commonly equipped with a special kind of Operating Systems, such as Contiki OS [3], specially customized for constrained IoT devices, our proposal is able to set up virtual IoT honeynets on demand, not only as a proactive measure but also as a reactive countermeasure to mitigate cyber-attacks, emulating a real IoT sensor network which the attacker is redirected to. In our proposal, data and control plane of the honeynet are managed in a centralized way through an Software Defined Networks (SDN) architecture. The SDN controller is in charge of providing traffic filtering and forwarding capabilities, as well as a redirecting and diverting traffic connections between the real IoT environment and the virtual one. To the best of

our knowledge, this is the first attempt to define a mechanism to dynamically deploy virtual IoT honeynets through SDN and NFV, that can emulate a real physical IoT network of devices.

The proposed system has been designed as part of a holistic cyber-security framework that has been developed in the scope of ANASTACIA EU research project [4]. The ANASTACIA framework follows a context-awareness approach that can provide support for decision-making, thanks to the continuous analysis of the network situation provided by the monitoring module. It endows the system with on-demand dynamic deployment and update of honeypot services for the virtual IoT honeynet.

The rest of the paper is organized as follows: Section II discusses the state of the art. Section III introduces the general ANASTACIA security framework. Section IV delves into the proposed solution for virtual IoT honeynets dynamic deployment through SDN-NFV. Section V describes the implementation performed for the autonomic deployment of the virtual IoT honeynets. Section VI reports the experimental results to enforce virtual IoT honeynets through SDN-NFV. Finally, Section VII concludes the paper with a set of final remarks and presenting future research lines.

II. Related work

IoT security issues and challenges [5] [6] are unceasingly evolving, mainly due to the intrinsic nature of the IoT paradigm in terms of constraint features (power, memory and computing) as well as the huge amount of heterogeneity targets connecting each other. In this sense, [7] overviews the main issues and challenges, at different levels, to be tackled, such as confidentiality, heterogeneity, integrity, authentication or availability. The paper highlights that the solutions provided must be as much lightweight as possible due to the constrained nature of the IoT devices. In this regard, authors in [8] assess the lightweight virtualization for deploying Virtual Network Security Functions at the Edge of the network. The IoT security issues are exposed more in detail in [9] by providing specific types of attacks per layer like buffer overflow, DoS Attack or Jamming. That work provides several IoT security requirements in order to deal with the main identified IoT security challenges which are differentiated according to the IoT generation in [10].

Due to properties like dynamic deployment and reconfiguration, the combination of technologies like SDN and NFV is a suitable approach to manage several IoT security challenges at different levels. In fact, to achieve confidentiality, integrity and availability with self-healing, and self-repair capabilities in wireless environments, new context-aware and autonomic softwarized frameworks, such as SELFNET [11] focused on 5G networks and ANASTACIA [12] targeting IoT networks have emerged. Farris et al. [13] provided a survey of SDN/NFV based security mechanisms of which part of them are currently being deployed and analysed in deep for the research community. In [14] authors combine SDN and NFV by deploying and configuring dynamically the infrastructure in order to provide authentication, authorisation and channel protection on demand in IoT environments and in [15] authors address SDN/NFV for network filtering in IoT. [16] and [17] also take the advantages of SDN and NFV for providing multi-tenant network slicing in shared IoT infrastructures. Caraguay et. al [18] take the advantage of the dynamic nature of SDN/NFV for ensuring quality of service, and [19] follows the same approach in order to provide horizontal scalability for massive IoT environments.

By applying and combining SDN/NFV based dynamic reconfiguration and deployment, other relevant security functions like honeynets for IoT environments can be leveraged. Current research efforts for IoT honeynets like [20] shows the relevance of honeypots and honeynets techniques by generations. They highlight the advantages of these approaches to deal with DoS attack mitigation, or the detection of unknown vulnerabilities. In order to ease the honeynet deployments, Fan et al. [21], [22] present a high-level model to represent honeynets and they validate it through a framework for converting the model in specific configurations and apply them by deploying tools like Dionea and Honeyd honeypots in LXC-based and KVM-based virtual machines. Guerra et al. [23] provide multi-purpose implementation of a low-interaction IoT honeypot (HoneyIo4) for capturing attacks on four different IoT devices; camera, printer, video game console and cash registering machine. In [24] authors deploy well known honeypots like Dionaea or Kippo in order to detect IoT botnet attacks (e.g. Mirai) by analyzing logs activity. Wang et al. [25] propose a specific IoT honeypot called ThingPot, which implements XMPP/MQTT as high-interaction honeypot module and REST API as low-interaction honeypot module. Since honeypots and honeynets aim to emulate elements from the real environment, it is important to take into account that they should be continuously reconfigured, according to the status of the real one. In this regard, Narik et al. [26] present an intelligent and dynamic low-interaction honeypot based on Dynamic fuzzy rule interpolation approach with dynamic reconfiguration according to its rule base. Dowling et al. [27] also provides an adaptive low-interaction honeypot, this time based on the reinforcement of learned results. Apart from the configuration, it is worth highlighting the reachability of honeypot or honeynet. To this aim, there exit static approaches like HIoTPOT [28] which uses a proxy in order to determine whether the user is allowed or not to access the real IoT environment based on the source address, whereas works such as Fan et al [29] and Lin [30] propose reroutes the traffic from suspicious nodes dynamically against the honeypot by using SDN techniques. These kinds of techniques, combined with NFV, allow us to provide reactive capabilities that takes the advantage of a dynamic configuration and deployment. Specifically, our solution extends Fan et al concepts and models by allowing to deploy proactive/reactive IoT high-interaction honeynets from IoT honeynet security policies which model the current status of the IoT infrastructure. Then they can be instantiated and configured or directly reconfigured by NFV Management and Network Orchestration (NFV-MANO) and dynamic SDN network reconfiguration.

Table I shows a comparison among the analyzed solutions focused on the level of interaction, the target (honeypot or honeynet), whether the solution is related to IoT, whether it

Solution	Int. level	Target	IoT	Policy-b	SDN-b	NFV-b	Dynamic	Impl./Val.
Fan et al. [21]	High/Low	Honeynet	NO	Model-b	YES	NO	Conf+Dep	YES
HoneyIo4 [23]	Low	Honeypot	YES	NO	NO	NO	NO	YES
Banerjee [24]	High/Low	Honeynet	YES	NO	NO	NO	NO	YES
ThingPot [25]	High	Honeypot	YES	NO	NO	NO	NO	YES
Naik et. al [26]	Low	Honeypot	NO	NO	NO	NO	Conf.	YES
Dowling et. al [27]	Low	Honeypot	NO	NO	NO	NO	Conf.	YES
HIoTPOT [28]	Low	Honeypot	YES	NO	NO	NO	NO	YES
Lin et.al [30]	Low	Honeypot	YES	NO	YES	NO	NO	YES
Proposed model	**High**	**Honeynet**	**YES**	**YES**	**YES**	**YES**	**Conf+Dep.**	**YES**

TABLE I: IoT Honeynet SOTA Comparison

follows a policy-based, SDN-based or NFV-based approaches, as well as whether it provides dynamic configuration or deployment and whether the solution has been properly implemented and validated. The comparison shows that there is still no available solution like the proposed herein, able to deploy dynamically policy-based IoT specific high-interaction honeynets through SDN and NFV, which in addition, has been properly validated.

III. Security framework overview

The proposed security framework aims at exploiting the features of SDN/NFV-based security enablers to ensure self-protection, self-healing, and self-repair capabilities in IoT systems, complementing conventional security approaches. To this aim, security policies are defined according to different level of abstraction, to ensure higher flexibility and manage security control over heterogeneous networks. The required security actions can be enforced in different kinds of physical and virtual appliances, including both IoT networks and softwarized networks.

Figure 1 shows the proposed architecture which is composed by three main planes.

1) User Plane: The user plane provides interfaces and tools allowing administrators to specify the desired security policy definitions. Its policy editor provides an intuitive and user-friendly tool to configure security policies in a high-level security language, governing the configuration of the system and network, such as authentication, authorization, filtering, channel protection, and forwarding.

2) Security Orchestration plane: The Security Orchestration plane enforces policy-based security mechanisms and provides run-time reconfiguration and adaptation of security enablers, thereby providing the framework with intelligent and dynamic behavior. It is an innovative layer of our architecture that provides dynamic reconfiguration and adaptation in case of deviation from the expected behaviour.

The *Policy Interpreter* module plays a key role in the refinement of security policies. The high-level policies are first refined into medium-level security policy language, which allows to specify workflows related to security procedures in a technology-agnostic way. Then, these policies are translated into specific low-level configurations according to the selected enablers. The policy refinements process are further detailed in Section IV-D.

The *Security Enablers Provider* identifies the available security enablers according to the required capabilities and their relevant resource requirements. It also manages the security enabler plugins which implements the translation to low-level configurations.

The *Monitoring* component collects security-focused real-time information related to the system behavior from physical/virtual appliances. Its main objective is to provide alerts for the reaction module in case something is misbehaving. Security probes such as Intrusion Detection Systems (IDS) as well as flow and resource monitoring probes are deployed into the SDN, NFV and IoT infrastructure domains to support the monitoring services. In our framework, these IDS components are implemented as virtual security enablers that can be configured on demand either proactively and reactively according to the dynamic monitoring security policies. The security framework follows hybrid detection approach, thereby supporting both, signature-based intrusion detection using rule-based approach, as well as machine learning-based anomaly detection to discover unknown attacks (0-days attacks). Thus, the monitoring and intrusion detection policies includes the list of signatures, the detection filters, as well as the associated actions (e.g. rise alerts, stats, logs..).

The *Reaction* component is in charge of providing appropriate countermeasures, e.g. by selecting policies stored in the relevant repository and by requiring reconfiguration of the security enablers to cope with the detected threat.

The *Security Orchestrator* supervises the orchestration of the security enablers to be deployed into the Security Enforcement Plane according to the policy requirements. It has been designed to achieve interoperability by allowing the on-boarding of different kind of drivers in order to orchestrate the enforcement plane through IoT Controllers, SDN Controllers and NFV-MANO. This approach allows using different controller's implementations by adding new orchestration drivers. The connection with IoT controllers is done through REST interfaces and network protocols such as CoAP, LWM2M, whereas interactions with SDN Controllers (e.g. ONOS) and MANO orchestrators (e.g. OpenMANO) are accomplished through Northbound APIs exposed by specific vendors (usually REST interfaces). In addition, at run-time, it analyses the reaction outcomes and orchestrates the corresponding countermeasures. In this way, the overall framework aims to achieve self-healing and resilience capabilities, by constantly ensuring the satisfaction of the security requirements defined in the end-user policies.

3) Security Enforcement Plane: The security enforcement plane is divided in different domains. The *Control and Management domain* supervises the usage of resources and run-time operations of security enablers deployed over software-based and IoT networks. A set of distributed SDN controllers takes charge of communicating with the SDN-enabled network elements to manage connectivity in the underneath virtual and physical infrastructure. NFV ETSI MANO-compliant modules provide support for the secure placement and management of virtual security functions over the virtualized infrastructure. As the envisioned framework aims to cover legacy IoT scenarios, different IoT controllers can be used to manage IoT devices as well as low power and lossy networks (LoWPANs). These IoT controllers are usually deployed at the network edge to enforce security functions in heterogeneous IoT domains.

The *Infrastructure and Virtualization domain* comprises all the physical machines capable of providing computing, stor-

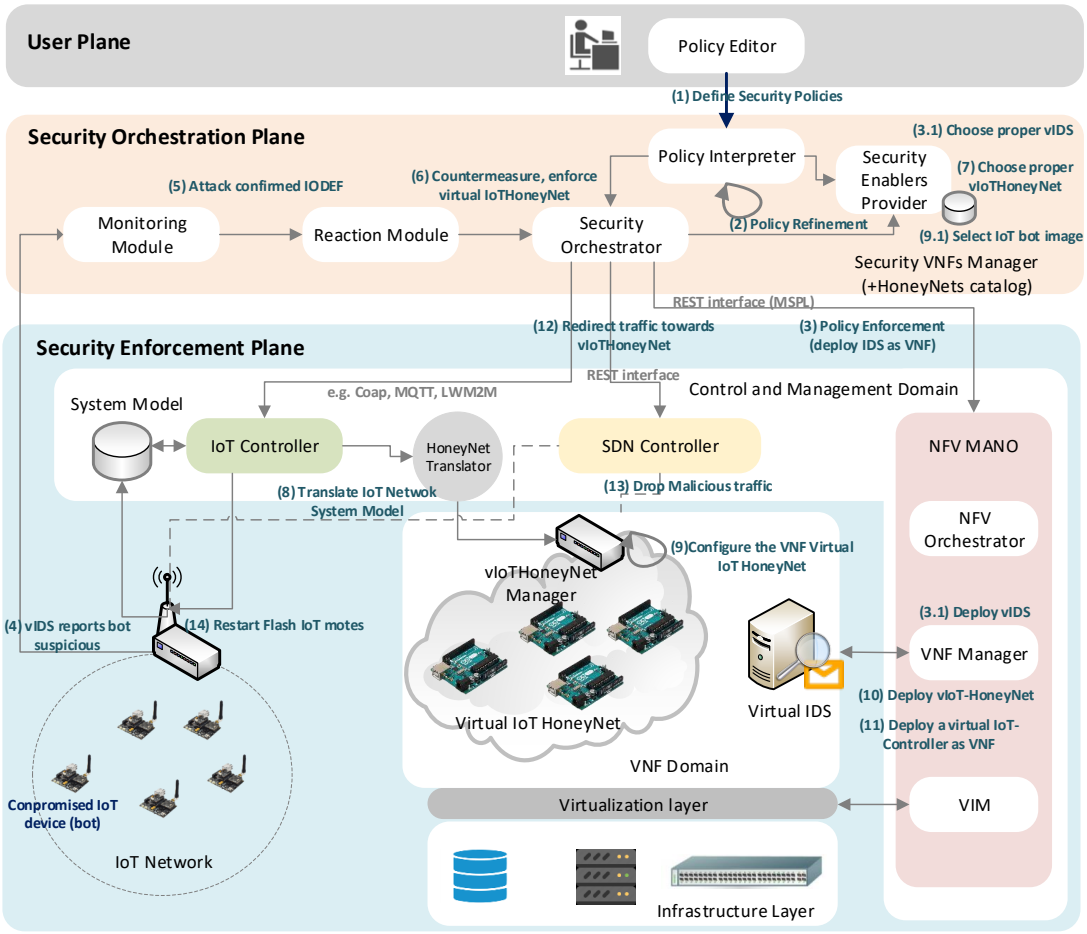


Fig. 1: High level overview of the proposed architecture

age, and networking capabilities, as well as the virtualization technologies, to provide an Infrastructures as a Service (IaaS) layer. This domain also includes the network elements responsible for traffic forwarding, according to the SDN controller rules, and a distributed set of security probes for data collection to support the monitoring services.

VNF domain accounts for the VNFs deployed over the virtualization infrastructure to enforce security within network services. Specific mechanisms will be developed to verify the trustworthiness of VNFs and to continuously monitor their key parameters, as well as specific attentions will be addressed to the provisioning of advanced security VNFs (such as virtual firewall, IDS/IPS, channel protection, etc.), capable to provide the defense mechanisms and threat countermeasures requested by security policies (e.g. the virtual IoT honeynet (vIoTHoneynet) Manager in charge of controlling the vIoTHoneynets, will be deployed as VNF).

IoT domain comprises the IoT devices to be managed. This includes the security enablers, actuators or software agents needed to enforce the security directives coming from the orchestration plane and managed at the enforcement plane by the IoT Controller. For instance, a special kind of local security agent can be deployed in IoT devices to protect the communications between two devices.

IV. Policy-based deployment of IoT honeynets based on SDN-NFV

A. Virtual IoT honeynets

Our proposal simulates a real IoT sensor's network to which the attacker is redirected to be distracted and investigated, countering the damage of his attack. Unlike traditional solutions, in our proposal, honeynets can be deployed not only pro-actively, but also reactively as a countermeasure to mitigate cyber-attacks, according to the reaction provided by reaction module. Besides, connectivity and data control in the honeynet is managed centralized through SDN. The SDN controller is in charge of providing traffic filtering and forwarding, as well as a redirection mechanism to divert and transfer the traffic connections from the real IoT network to the virtual IoT honeynet, according to the necessities of intrusion investigations.

The honeynets definition in terms of network and service topologies evolve dynamically according to the context and the actual IoT physical deployment. To this aim, the system model is kept up to date, fed through monitoring services by probe modules such as IoT crawlers, IDS, FlowMon, notifications, etc., in charge of generating automatically the description of the honeynet by scanning the target production network, whereby obtaining the necessary system and network data such as, IP address, operating system, services, resources and

open ports. Once the information has been properly gathered, honeynets are modeled using an extended version of the Technology Independent Honeynet Description Language (TIHDL) [21] which, in turn, inherits from the Common Information Model (CIM) [31]. Our honeynet description model improves TIHDL by including additional concepts needed for modeling IoT virtual honeynets, such as, physical location of the devices and resources provided by the IoT devices as it is defined in section IV-C.

By using a well-defined model as the TIHDL extended one, it can be analyzed and translated in order to transform the model information in configurations for different virtual environments. For instance, the Cooja [32] simulator for wireless sensor networks enables a holistic high interaction simulation for Contiki [3] operating system of IoT devices. Cooja allows simulations at network level, operating system level, and machine code instruction set level, enabling the deployment of virtualized HIH honeynets. When the virtual environment configuration is ready to be deployed, a vIoTHoneyNet Manager is installed in an Virtual Network Function (VNF), and it is deployed dynamically by the NFV-MANO. In this sense, the Honeynets deployment can be performed at the proper network location to protect users against the cyber-attack.

Once the VNF has been deployed, the virtual IoT devices in the honeynet start simulation sensor values, randomly generated in the scope of some margins, given by the standard deviation over the mean values that the real IoT system was reporting so far. Besides, the system is designed to offer capabilities for reconfiguring the scenario. The probe module collects in real time updates from the real production network, adjusting the network structure of the virtual honeynet accordingly. Furthermore, the framework is designed to continuously monitor the real IoT devices in order to extract information to be used to provide a HIH of the IoT honeynets, so that in case of an infected device is detected an alert and reaction is raised generating then a dynamic up-to-date virtual IoT honeynet deployment with the updated IoT honeypots.

B. Virtual IoT honeynets data model

The context information maintained and managed by the System Model database of the *Control and Management Domain* layer of the framework provides meta-model data needed to interpret the concepts expressed in the high-level security policy language. The Context Information encompasses the environmental information retrieved from the security enforcement plane.

The monitored information, along with the real time instantiated system model is defined in a common language such as CIM (Common Information Model) [31] from DMTF. CIM provides a modelling mechanism to represent concepts available in IT domains. Indeed, some operating systems and platforms already support retrieving the current and instantiated status of the system in CIM model, providing detailed information about the managed system. This description can be used to retrieve information about which capabilities are provided by different system components, as well as particular network configurations, in order to perform the policy refinement from high-level security policy to medium-level policy language. The DTMF also provides other standard models to represent specific components of the underlying virtual environment, such as OVF, VMAN, CIMI and hardware infrastructure (e.g. SMASH, Redfish). For example, a packet filtering policy should be applied by a firewall element which has network traffic filtering capabilities, and the needed of extra information such as network IP addresses associated to a user or a device identifier can be obtained from the instantiated system model.

Nonetheless, CIM does not directly provide concepts for representing honeynets. To fill this gap, the Technology Independent Honeynet Description Language (TIHDL) [21] extends CIM allowing to represent flexible virtual honeynet models, which can comprise a combination of heterogeneous platforms for deploying honeynet in virtual networks and also deploy hybrid honeynet for both, low-interaction honeypots and high-interaction honeypots. It was already validated by authors in order to model and deploy different number of virtual honeypots (NVHs) for diverse platforms [22]. However, TIHDL was not designed having Internet of Things in mind. Several concepts need to be added, and some others redesigned, in order to model properly vIoTHoneynets. To this aim, our proposal extends TIHDL to represent several concepts needed to deploy vIoTHoneynets. Fig. 2 represents the root concepts in a Honeynet class.

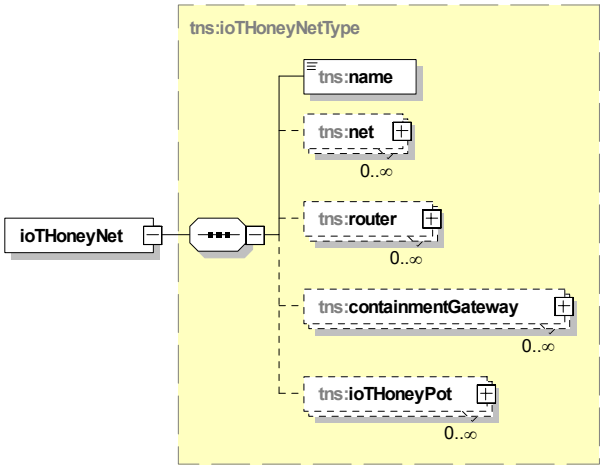


Fig. 2: IoT HoneyNets system data model schema.

Every *IoTHoneyNet* is comprised of several *Nets, Routers, Honeypots* and *ContainmentGatways*. The *ContainmentGatway* class aims to represent honeywalls and their interfaces. Notice that, although it has omitted in the model for the sake of simplicity, the classes inherit from already existing classes in CIM. For instance, *Router* and *HoneyPot* extends from *ComputerSystem*, and *Interface* inherits from *NetworkPort*.

The *IoTHoneyPot* class is represented in Fig. 3. A honeypot in IoT needs special attributes that are not needed in traditional honeypots. For instance, IoT honeypots can be placed in wireless devices which are deployed in a particular location. To represent this notion, the *IoTHoneyPot* class features the attribute *Location*. Besides, an IoT device usually is endowed with several supplied sensors and *resources* (e.g. tempera-

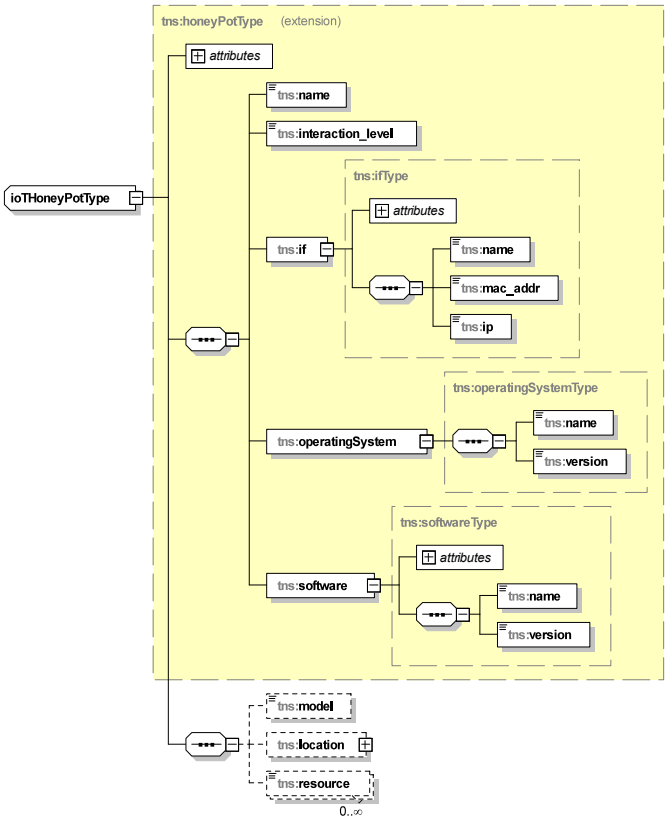


Fig. 3: Virtual IoT HoneyNets system model. IoT HoneyPot

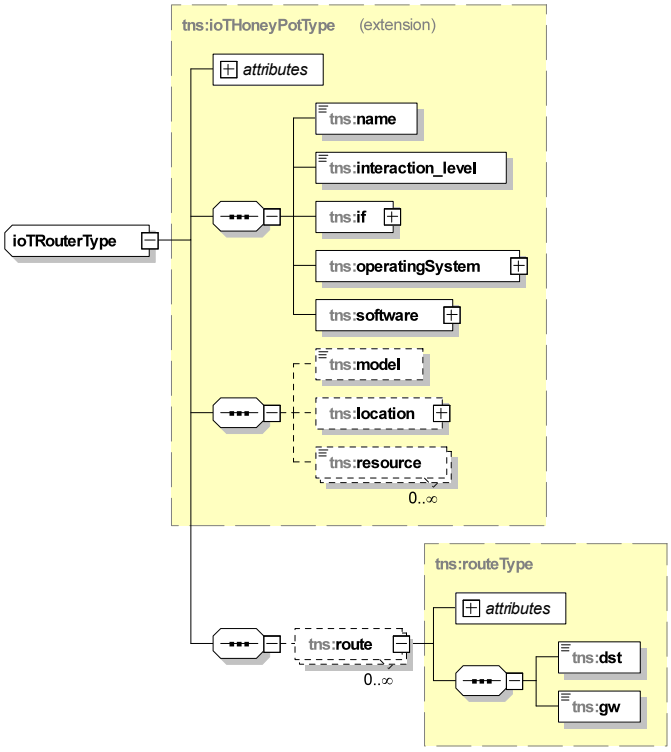


Fig. 4: Virtual IoT HoneyNets system model. Router class

ture). In addition, the particular *model* of the IoT device (e.g. Sky mote) is also important to be able to instantiate highly-interactive virtual honeypots. Thanks to this *model* and *resources* properties in the policy, the virtualized system enforcing the virtual IoT honeyNet will be able to adjust and configure the virtual IoT device to represent exactly the same constraints of the real one, including hardware constrains such as battery, memory and CPU, and other features such as network protocols supported.

The rest of the classes related to the *IoTHoneyPot* encompasses: *Interfaces* (Mac, IP, net), the *InteractionLevel* (low or high-interactive honeypot), *OperatingSystem* (e.g. Contiki) and *Software* deployed in the honeypot (e.g., Erbium).

Finally, the *IoTRouterType* is represented in the diagram of Fig. 4. IoT networks can be comprised of multi-hop wireless networks in which the IoT devices can act as routers to deliver the packet to the next hop. As in the *IoTHoneyPot* class, the routers also need to represent concepts such as *model*, *location* and *resource*. Besides, the *Router* class holds the routing table with the set of entries (destination-gateway) needed to represent accurately in the vIoTHoneyNet the same network topology than in the real physical IoT network. Moreover, the *Router* class is associated to a particular software of routing protocol (e.g. RPL). By modeling these concepts will allow to configure the vIoTHoneyNet exactly as it is the real IoT deployment.

C. Virtual IoT honeynets behaviour modelling

Slow DoS attacks, such as Slowloris or slowPost aims to overwhelm a target victim by sending incomplete or malformed requests at slow date rate, thereby occupying available connections of a target victim server. If an attacker(s) manage to take up enough connections, the victim server will get overwhelmed. Since these kind of DDoS attacks do not require high-performance systems to be accomplished, IoT ecosystems can be especially subject to these kind of attacks.

The IoT Honeynet behaviour model needed to mitigate these kind of DDoS attacks can be described as follows. Let us assume a network N defined as a set of interconnected computer nodes $N = \{N_1, N_2, N_3, ..., N_n\}$ where each of the computer nodes will play one and only one role R, defined as R =[ATTACKER, SERVICE, HONEYPOT]. Attackers $A = (A_1, A_2, ...A_a) \subset N \Rightarrow A_a.role = ATTACKER$ are a large number of IoT nodes that are performing the Low-Rate DDoS Attack. $SERVICES = (S_1, S_2, ...S_s) \subset N \Rightarrow S_s.role = SERVICE$ are the nodes victim of the attack. Notice that the victims of the attack can also be the own IoT devices. And $HONEYPOT = (H_1, H_2, ...H_s) \subset N \Rightarrow H_s.role = HONEYNET$ are the nodes offering fake services that are receiving the attack as a result of the mitigation. This scenario assumes that the behaviour of a node in terms of the services its offers is modeled as b function. And, that for each service node, there is an associated node in the honeypot providing the same behaviour with fake data. Formally, this is the defined as $\forall i \in s, b(S_i) = b(H_i)$. Also, the modeling of the behaviour of the attackers is defined by the function s indicating the delivery of malicious packet at a particular packet rate to form a Slow Rate DDoS attack,

defined the packet rate as r. Thus, formally, $b(A_i) = s(r)$. The implications of sending such packets have been already explained.

The behaviour of the SDN controller is twofold. Nodes are interconnected by mean of switches that are controlled by an SDN controller. The behaviour of the SDN Controller is to forward traffic to the nodes. At the initial state, before of the detection of the attack, moment defined as tn, the forwarding will be done to the service nodes. So that, $b(SDN_tn) = f(SERVICE)$. Then, when the attack is detected, moment defined as td, the forward will be updated to redirect the traffic to the honeypot nodes. So that, $b(SDN_td) = f(HONEYPOT)$. The network has also an special node running an IDS software to perform the detection of the attack. Such detection is based on the matching of a well-known signature available in the packets to perform the detection of the attack against a database of signatures. Formally, let us define the set of attack signatures as $R = (R_1, R_2, ...R_r)$ and thus let us defined the function to detect an attach as a match m function that will be true if there is an occurrence of the rule R_i matching against the signature of every packet, defined as $P = (P1, ..., P_k)$ so that $\forall R_i \forall p_k \Rightarrow \exists m(R_i, p_k) = true$. Thus, the behaviour of the IDS is defined as $b(IDS) = m(R, P)$. Finally, the modeling of the mitigation of the attack through the honeynet can be formalized as the change in behaviour from $b(SDN_tn)$ to $b(SDN_td)$ when $m(R, P) = true$.

D. Policy-based security management in SDN/NFV-enabled networks

IoT deployments are comprised of disparate kind of devices which might differ a lot in the available resources, implemented protocols, or connectivity technology employed. Besides, some domains are prone to receive more attacks than others and the level of criticality varies among deployments, thereby changing the economic or strategic interest from attackers. Consequently, different security requirements might apply to different IoT deployments. The security refinement process defined herein is based upon a policy based strategy for the enforcement and management of security requirements over an IoT platform providing inter-operability and avoiding vendor lock-in. Security policies are a flexible way to tailor the security requirements needed by an IoT platform to the specific domain where the IoT platform is deployed. Them also ease the security management activities required to control the fulfillment of the claims included in the policy, allowing to set up monitoring activities, measurements, thresholds, alerts, reaction activities, etc.

In this way, security policy operations are divided into three main tasks. Namely, the process that parses the high level policy into a machine readable format (policy refinement), the transformation of that policy format into low level configuration rules (policy translation) and the process for configuring the system (policy enforcement). The sequence diagram of Fig. 5 shows the main workflow for a proactive policy definition and its refinement from High-level Security Policy Language (HSPL) to Medium-level Security Policy Language (MSPL) based on [33], as well as the policy translation process from MSPL to low-level configurations.

First, the security administrator defines the security policy in HSPL (Fig. 5-step 1). Then, the Policy Interpreter receives a policy enforcement request, starting the refinement process by the identification of the capabilities (Fig. 5-step 3), understanding a capability as a main purpose of the policy (e.g. filtering capability will be identified when a security policy is defined in order to drop traffic).

Once the Policy Interpreter has identified the capabilities, it performs a request to the Security Enabler Provider (Fig. 5-step 4), with the aim to get a list of Security Enablers capable to enforce the mentioned capabilities. In this context, a Security Enabler corresponds to a piece of software or hardware capable to implement some specific security properties (e.g. filtering, forwarding...). The Security Enabler Provider then returns the aforementioned list of Security Enablers for each capability. Afterwards, the Policy Interpreter verifies whether each security policy could be enforced using at least one of the Security Enabler received (Fig. 5-step 8).

If the security policy could not be enforced, the Policy Interpreter returns a non-enforzable analysis to the user, indicating the issue (Fig. 5-step 9). Otherwise, the Policy Interpreter retrieves system model information (e.g. technical information of an IoT device) and it performs the policy refinement taking into account the capability of the HSPL policy as well as the specific system model information in order to generate one or several (e.g. bi-directional behavior) MSPL policies (Fig. 5-step 12). Like in the previous case, if there is some issue during the refinement, the user is notified with a non-enforzable analysis but, in other case, the MSPL policy and the list of available security enablers are uploaded to the Security Policy Repository and they are also sent to the Security Orchestrator in order to proceed with the best security enabler selection for the medium-level security policy translation process (Fig. 5-steps 14,15).

When the Security Orchestrator receives the MSPL policy and the list of security enablers, it retrieves information about the underlying technologies in order to select the best security enabler for the policy enforcement by using the current context and system information to make the decision (Fig. 5-steps 16,17,18). Then, the Security Orchestrator requests the MSPL translations to the Policy Interpreter for each pair of MSPL and selected security enabler. The Policy Interpreter retrieves from the Security Enabler Provider the plugin which implements the MSPL translation for the selected security enabler, and executes it, generating the expected low-level configurations (Fig. 5-steps 21-27), which are sent to the Security Orchestrator (Fig. 5-step 23) who proceeds with the enforcement process.

Fig. 6 shows the policy enforcement process. The Security Orchestrator receives the enabler's configuration and it triggers the enforcement process through the SDN Controller, the IoT Controller, the NFV MANO (Fig. 6-steps 1,2,3) or even by a direct communication with the security enabler (e.g. legacy physical router), depending on the requirements of the security policy. On the one hand, if a specific VNF is required, and it is not already deployed, the NFV-MANO creates, configures and deploys a new one (Fig. 6-steps 6,7). Otherwise the

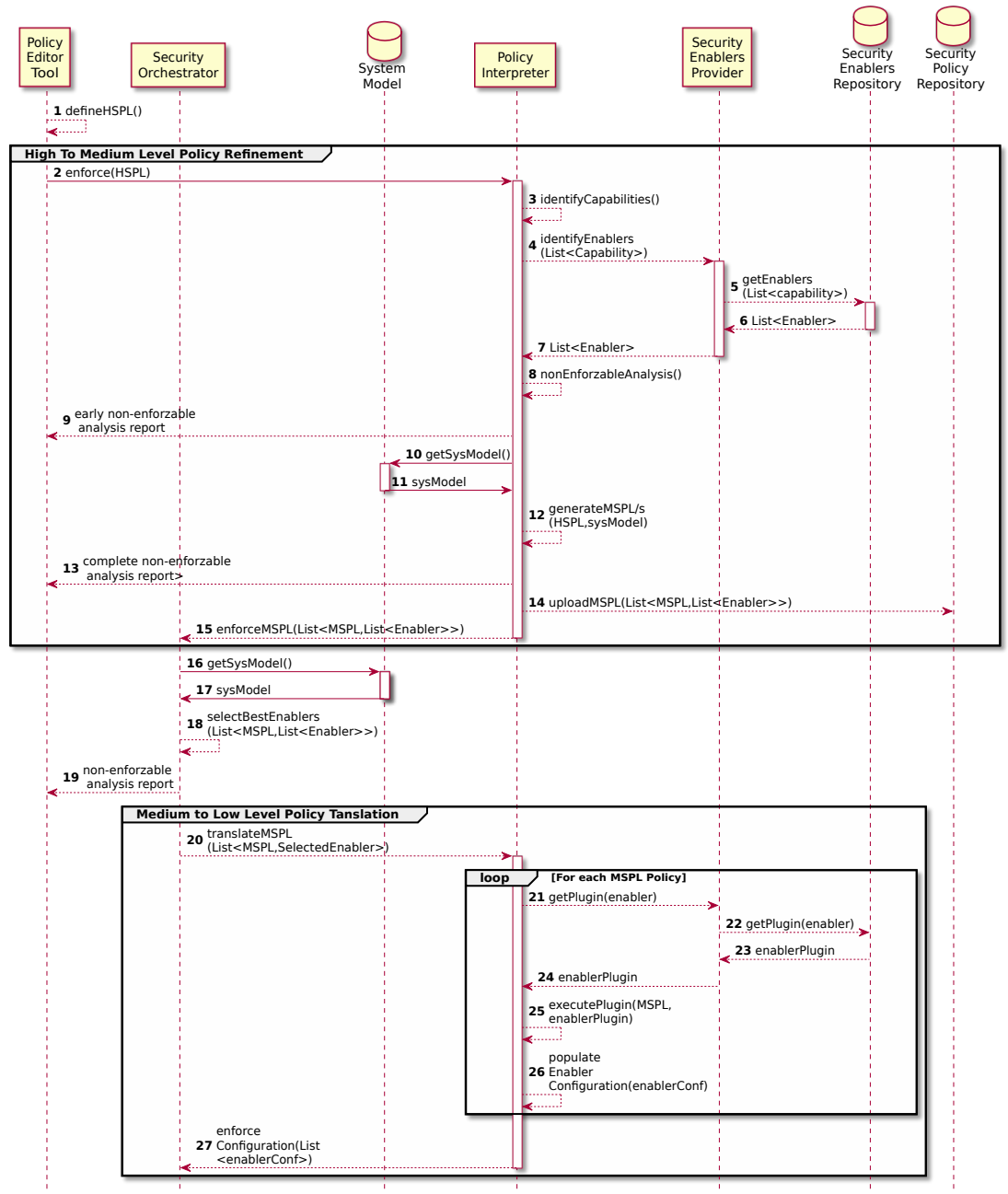


Fig. 5: Policy Refinement and Policy Translation processes

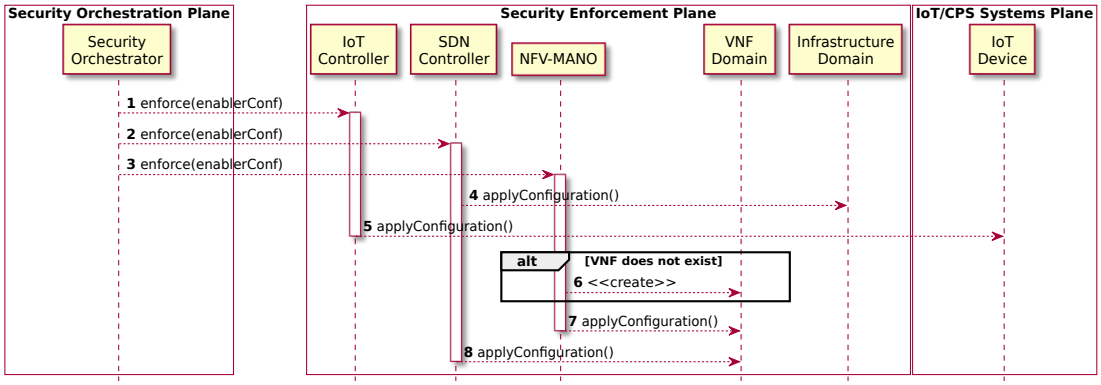


Fig. 6: Policy Enforcement process

NFV-MANO just enforce the received configuration over the VNF. Besides, if the security policy is SDN related, the

SDN Controller is in charge of enforcing the policy over the managed SDN network (Fig. 6-steps 4,8). Finally, if the security policy is IoT related, the IoT Controller enforces the IoT configurations over the managed IoT devices by using specific IoT constrained protocols (Fig. 6-steps 5).

E. Attack detection and virtual IoT Honeynet dynamic configuration and deployment

By following the previous policy-based approach it is possible to define a set of pro-active security policies in order to configure and deploy monitoring agents which feeds the reaction part of the framework. Thus, the monitored data is filtered, processed and analyzed, issuing verdicts about anomalies occurring in the monitored platform (potential threats or ongoing attacks). The identified events are notified to the reaction module of the framework which creates countermeasures (i.e. a set of security policies) reacting to threats or attacks, and triggering the countermeasure's enforcement. Besides, the security reaction process is also able to notify the administrator, who might provide feedback and trigger critical countermeasures that require explicit consent or to override the security policy.

When the reaction countermeasure is performed automatically, the reaction module generates and provides a set of countermeasures as new security policies to the Security Orchestrator, which acts similar to when it receives the MSPLs from the Policy Interpreter in the pro-active scenario. Fig. 7 shows the main workflow regarding a botnet DDoS attack detection, instantiated for an IoT honeynet deployment countermeasure. First, a previously deployed IDS VNF detects the signature of the botnet attack (e.g. Mirai) [34] (Fig. 7-steps 1,2). The infected zombie (bot) in the IoT domain can be detected based on flow-based metrics, which are sent to ANASTACIA monitoring module for further analysis. The Monitoring module analyzes the thread and communicates the alert (e.g. IODEF) to the reaction module (Fig. 7-step 3,4), which is in charge of making the decision regarding the particular kind of countermeasure to be taken. In this case, the reaction module indicates that IoTHoneynet and networking operations that must be enforced, so it requests the enforcement of the countermeasures to the Security Orchestrator (Fig. 7-step 6).

The Security Orchestrator receives the attack information as well as countermeasures (which could be security policies pending to be fulfilled) and it executes the orchestrator algorithm (Fig. 7-step 7). Algorithm 1 shows the pseudo-code of orchestration algorithm.

The algorithm iterates over the countermeasures verifying whether there is any unsatisfied dependency. If so, the countermeasure is queued until the dependency is solved (filtering and forwarding will depend on the vIoTHoneynet deployment). Otherwise, the algorithm retrieves system model information of the underlying technologies related to the elements involved in the countermeasure. If the countermeasure is related with IoT, the algorithm also includes in the system model the IoT infrastructure information available in the IoT Controller. The

Data: $AI = attackInfo$,
$C = \{countermeasuresList\}$
Result: $EnforcedC' \subset C$
for c *in* C **do**
  if c *has unsatisfied dependencies* **then**
    $queue(c)$;
    $continue$;
  end
  $sm \leftarrow getSystemModel(c)$;
  if $c \subset \{IoTContermeasures\}$ **then**
    $ioTsm \leftarrow getIoTSystemModel(c)$;
    $sm \leftarrow sm \cup ioTsm$;
  end
  $mspl \leftarrow fillMSPL(c, sm)$;
  $candidates \leftarrow getEnablerCandidates(mspl)$;
  $se \leftarrow selectEnabler(mspl, candidates, sm)$;
  $conf \leftarrow translate(mspl, se)$;
  if $mspl \subset \{HoneynetMSPLs\}$ **then**
    $cf \leftarrow genCustomFirmware(conf, AI)$;
  end
  $enforce(conf, se, cf)$;
end

Algorithm 1: Security Orchestrator Algorithm for countermeasures enforcement

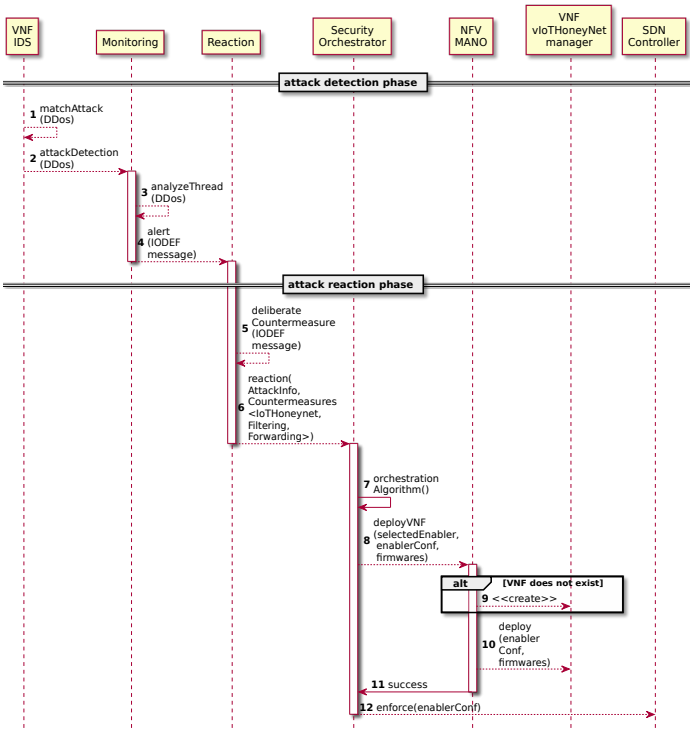


Fig. 7: IoTHoneyNet deployment process

information of the infrastructure and the countermeasure are then used in order to fulfill the MSPL policy. The algorithm then retrieves a list of security enabler candidates capable of enforcing the MSPL policy, and it selects the best candidate by considering the current status of the infrastructure (system model), the candidates and the security policy requirements.

Once the best security enabler is selected, the algorithm obtains the enabler configuration by a MSPL policy translation. If the security policy is IoT Honeynet related, a custom firmware generation could be required in order to emulate not only the current infrastructure, even the specific attack behaviour (e.g.

emulate a Mirai botnet). Finally, the enabler configuration is enforced through the selected enabler (Fig. 7-steps 8,12). For the vIoTHoneynet, the NFV-MANO verifies if there is already deployed a suitable VNF which implements the vIoTHoneynet manager. If so, the vIoTHoneynet configuration is sent to the vIoTHoneynet manager who starts the vIoTHoneynet according on the received parameters. Otherwise, the process is barely the same, but the NFV-MANO first creates a suitable VNF instance for the vIoTHoneynet manager.

When the vIoTHoneynet is running, the Security Orchestrator receives a notification from the vIoTHoneynet manager, and it verifies whether the event satisfies any queued countermeasure. If so, it processes and enforces the filtering and forwarding policies through the SDN controller in a transparent way for the attacker. On the one hand, the traffic from the real IoT domain to the attacker as well as the traffic from the real IoT domain which is generating the DDoS attack are filtered. On the other hand, the traffic from the attacker to the real IoT infrastructure is redirected to the vIoTHoneynet and the simulated DDoS generated by the vIoTHoneynet is also filtered. In this way the attacker believes he is still controlling the affected bot device, but the attack is unsuccessful and the process allows security administrators take advantage of the situation (e.g. learning the attacker methodology).

V. Implementation

Figure 8 shows the deployment that has been instantiated, based on the developments carried out in the scope of this research in order to perform the testing of the proposed architecture. The monitoring module is instantiated using the XL-SIEM tool, which is a Security Information and Event Manager able to detect issues along the architecture by monitoring network resources, provided by Atos. XL-SIEM is able to infer cyber-atacks by analyzing the detected anomalies. Those issues are notified to the Reaction module, providing the information in the Incident Object Description Exchange Format IODEF [35] standard.

The Reaction module analyzes the threats and it generates a reaction which is sent to the Security Orchestrator in an adequate format, able to represent the coordination and execution of command and control for cyber-defense components, i.e. the Open Command and Control (OpenC2) [36] language. Once the Security Orchestrator knows the reaction, it performs the requested modifications over the architecture by using security policies. To provide policy definition, refinement, and translation features, a Policy Editor Tool and a Policy Interpreter have been implemented in Python, based on the outcomes of the SECURED project policy models [33].

Specifically, different plugins have been implemented for the policy translation from MSPL to low-level configurations. Namely, a plugin that translates MSPL IoT HoneyNet policies into Cooja emulator configurations which can be applied by the Northbound API of our vIoTHoneyNet manager. Listing 1 provides an MSPL policy example which embeds the IoT honeynet system model. The *virtualIoTHoneyNetAction* specifies the action to be performed over the honeynet (other actions such as reconfigure, stop or restart the

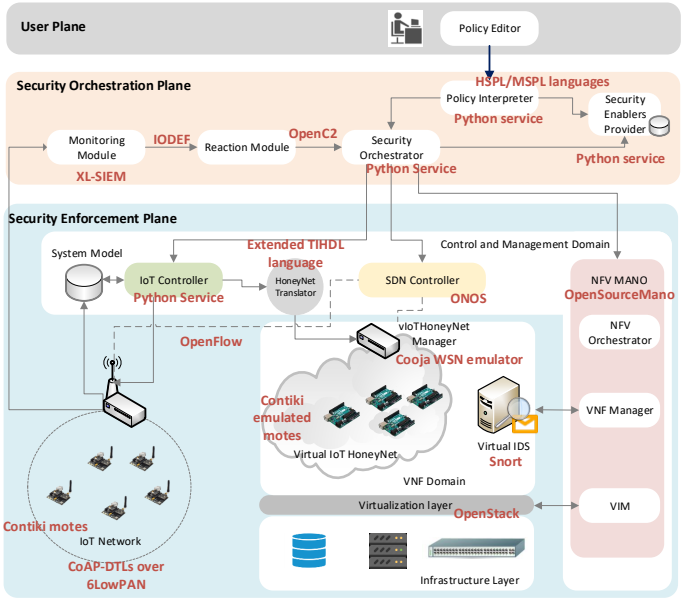


Fig. 8: Architecture instantiation and implementation

environment can be specified), whereas the *IoTHoneyNet* model describes general information regarding the network and the honeynet itself, like identifiers or descriptions. More concrete information is provided about the main elements such as *routers* and *IoTHoneyPots*, like their interaction levels (LOW, HIGH), defined as the degree of replication between the real and virtual environments, the operating system, the installed software and its version, the available resources of the IoT honeypot (e.g. temperature sensor, humidity sensor...) and even its physical location expressed in Cartesian coordinates.

```
<configurationRule>
 <configurationRuleAction xsi:type='VIoTHoneyNetAction'>
  <VIoTHoneyNetActionType>DEPLOY</VIoTHoneyNetActionType>
    <ioTHoneyNet>
      <name>REST with RPL router</name>
      <net id="1"> <name>net</name></net>
      <router id="1">
        <name>Wismote RPL Root</name>
        <interaction_level>LOW
        </interaction_level>
        <if id="1" net="str1234">
          <name>i1</name>
          <mac_addr>ROUTER_MAC</mac_addr>
          <ip>ROUTER_IP</ip>
        </if>
        <operatingSystem>
          <name>contiki</name><version>2.7</version>
        </operatingSystem>
        <software id="1">
          <name>RPL</name><version>3.14</version>
        </software>
        <model>Wismote</model>
        <location><x>33.2601</x><y>30.6432</y></location>
        <resource>TEMPERATURE</resource>
      </router>

      <ioTHoneyPot id="2">
        <name>Erbium Server</name>
        <interaction_level>LOW</interaction_level>
        <if>...</if>
        <operatingSystem>
            <name>contiki</name>
        </operatingSystem>
        <software id="1">
          <name>Erbium Server</name><version>3.14159</version>
        </software>
        <model>Sky</model>
        <location>...</location>
        <resource>Temperature</resource>
      </ioTHoneyPot>
       ...
    </ioTHoneyNet>
   </configurationRuleAction>
</configurationRule>
```

Listing 1: IoT HoneyNet Model Example

```
<simconf>
 <motetype>
  se.sics.cooja.mspmote.WismoteMoteType
  <identifier>1</identifier>
  <description>Wismote RPL Root</description>
  <source>border-router.c</source>
  <commands>make border-router.wismote
      TARGET=wismote</commands>
  <firmware>border-router.wismote</firmware>
  <moteinterface>Position</moteinterface>
  <moteinterface>IPAddress</moteinterface>
 </motetype>

 <motetype>
  se.sics.cooja.mspmote.WismoteMoteType
  <identifier>2</identifier>
  <description>Erbium Server</description>
  <source>coap-server.c</source>
  <commands>make coap-server.wismote
      TARGET=wismote</commands>
  <firmware>coap-server.wismote</firmware>
  <moteinterface>Position</moteinterface>
  <moteinterface>IPAddress</moteinterface>
  <moteinterface>TEMPERATURE</moteinterface>
 </motetype>

 <mote>
  <breakpoints></breakpoints>
  <interface_config>
   org.contikios.cooja.interfaces.Position
   <x>33.2601</x><y>30.6432</y><z>0.0</z>
  </interface_config>
  <interface_config>
   org.contikios.cooja.mspmote.
       interfaces.MspMoteID
   <id>1</id>
  </interface_config>
  <motetype_identifier>1</motetype_identifier>
 </mote>
 ...
</simconf>
```

Listing 2: Corresponding Cooja CSC model

Listing 1 shows an example of IoT honeynet MSPL whereas Listing 2 shows its corresponding translation into Cooja Simulation Model configurations (CSC model). As it can be observed, it allows specifying the type of IoT device (*motetype*), even including the source code to be compiled, or directly the IoT firmware, as well as the platform and available resources.

Besides the vIoTHoneynet configuration, additional policy enforcements like traffic filtering and traffic forwarding must be accomplished in order to reconfigure the network for accommodating the new virtual appliance as well as for redirecting the attacker to the vIoTHoneyNet. In this sense, it has been developed a plugin to translate MSPL filtering and forwarding policies to specific SDN ONOS Controller Northbound API configurations.

```
<configurationRule>
 <configurationRuleAction xsi:type='FilteringAction'>
  <FilteringActionType>DENY</FilteringActionType>
 </configurationRuleAction>
 <configurationCondition xsi:type='FilteringConfCondition'>
  <packetFilterCondition>
   <SourceAddress>aaaa::/64</SourceAddress>
   <DestinationAddress>cccc:2/128<DestinationAddress>
   <Interface>2</Interface>
    ...
 <externalData xsi:type='Priority'><value>60000</value>
     ...
</configurationRule>

<configurationRule>
 <configurationRuleAction xsi:type='TrafficDivertAction'>
  <TDivertActionType>FORWARD</TDivertActionType>
   <packetDivertAction>
    <packetFilterCondition>
     <Interface>3</Interface>
      ...
  </configurationRuleAction>
  <configurationCondition xsi:type='TDivertConfCondition'>
   <packetFilterCondition>
    <SourceAddress>cccc::2/128</SourceAddress>
    <DestinationAddress>aaaa::/64</DestinationAddress>
    <Interface>1</Interface>
    ...
  <externalData xsi:type='Priority'><value>60000</value>
  ...
</configurationRule>
```

Listing 3: MSPL Filtering Example

```
[{"priority": 60000,
  "treatment": {
   "instructions": [{"type": "NOACTION"}]},
  "selector": {
   "criteria": [
     {"type": "IPV6_SRC","ip": "aaaa::/64"},
     {"type": "IPV6_DST","ip": "cccc::2/128"},
     {"type": "IN_PORT","port": "2"}]}},

{"priority": 60000,
 "treatment": {
  "instructions":[{"type": "OUTPUT","port": "3"}]}
 "selector": {
  "criteria": [
    {"type": "IPV6_SRC","ip": "cccc::2/128"},
    {"type": "IPV6_DST","ip": "aaaa::/64"},
    {"type": "IN_PORT","port": "1"}]}}]
```

Listing 4: ONOS Northbound Filtering Configuration Example

Listing 3 shows examples of filtering and forwarding security policies. On the one hand, the filtering policy indicates the traffic that goes from the real IoT deployment (*AAAA::/64*) to the attacker (*CCCC::2/128*) must be dropped. On the other hand, the forwarding policy indicates the traffic coming from the attacker to the real IoT deployment must be redirected to the interface where the vIoTHoneynet has been deployed. Listing 4 shows the configuration obtained after the policy translation process for filtering and forwarding policies by using ONOS as SDN security enabler. Specifically, it provides filtering and forwarding rules to be applied through the ONOS Northbound API.

Regarding the orchestration process, it has been developed a Python application which allows to enforce the aforementioned security policies by applying the configuration or tasks to the different policy enforcement points. In the case of the vIoTHoneyNet security policy, the Security Orchestrator implementation is also in charge of obtaining the IoT physical architecture model in a extended TIHDL format from the IoT Controller and include it in an IoTHoneynet MSPL security policy. Once the MSPL has been generated, the Security Orchestrator gets the final Cooja configuration through the Policy Interpreter, which executes the vIoTHoneyNet plugin in order to translate the IoT system network modeled in our extended TIHDL language into Cooja configurations. Cooja has been chosen, since it allows developing the IoT device functionality in C language by customizing, compiling and loading the IoT firmwares and network specifications into the platform. It even provides the real IoT device locations in coordinates, and it also allows to provide interesting parameters such as the transmission range, interference range and success ratio for LoWPANs.

Once the Cooja CSC model has been obtained, the Security Orchestrator selects the proper firmwares to replicate the real IoT behavior and it requests the deployment of the vIoTHoneyNet with the specific configuration and firmwares through the NFV-MANO. In order to deploy on demand the vIoTHoneyNet, our vIoTHoneyNet manager implementation provides an API capable to receive the Cooja CSC model and the specific firmwares to configure and execute the Cooja simulation in the VNF. When the simulation has been started and the network is ready to be reachable from outside (i.e. routing protocol algorithm has converged), the vIoTHoneyNet manager warns the Security Orchestrator, which enforces the filtering and traffic divert policies through ONOS SDN

controller for redirecting the traffic generated or received for the physical architecture to the virtual one. Besides, it drops the malicious traffic sent to the victim in order to mitigate the current attack.

VI. Performance Evaluation

The section aims to determine the feasibility of the deployment for the proposed virtual IoT honeynet mechanism. The goal is to apply an IoT honeynet security policy as reaction countermeasure to mitigate an attack in a reasonable time by deploying a virtual IoT honeynet as much realistic as possible to the real physical IoT deployment. The performance tests have been carried out by applying 100 times IoT Honeynet security policies over each of the two different sections on the Smart Building floors we are using in our premises. IoT sensors are heterogeneous in terms of sensoring capabilities and operating system version installed for each floor.

The full time of the IoT honeynet policy deployment has been split in different times to allow a fine-grain analysis, as is shown in figure 9. In the translation among the physical model to the virtual one, it is measured the time taken by translator plugin to provide the Cooja CSC model from the IoT HoneyNet physical model (i). For the compilation (ii) and load (iii) of the IoT devices code, it is measured the time taken to compile the code for all IoT devices and the time required to load the compiled code into the IoT devices. In addition, it is considered the overall time required by the simulation to be ready, i.e. all IoT devices are up and running (iv). In case the simulation uses a routing algorithm, the IoT network convergence time measures the time needed by the router to learn a route to all IoT devices (v). Finally, the Policy Enforcement time measures the time taken to apply the network policy configurations in order to filter and divert the traffic as described in previous section (vi). The tests have been supported by a virtual machine with 4 CPUs and 2 GB of RAM memory. This virtual machine has been hosted in an Intel Core i7-2600 at 3.40 GHz with 8 GB of RAM memory.

Regarding the use cases, Figure 10 shows the first use case corresponding to first floor of our building, which is comprised by 20 sky motes distributed along 30x15m, executing Contiki OS 2.7, empowered by 8 Mhz, 10KB RAM and 48KB Flash, measuring humidity, temperature, light and CO2. One of them is a RFID sensor as door keeper, and finally, there is one more as a router using RPL as routing algorithm which connects all sensors to the smart build network.

On the other hand, figure 11 shows the second-floor use case which is comprised by 50 wismote motes, distributed along 37.5x15m, executing contiki OS 3.1, empowered by 16 Mhz, 16KB RAM and 128KB Flash, measuring humidity, temperature, light, presence and CO2. In this case, all doors are equipped by RFID sensors, and finally, at the same way of the previous case, there is one more as a router using RPL as routing algorithm which connects all sensors to the smart build network.

Figures 12 and 13 shows the time taken for each step in the vIoTHoneyNet policy deployment process in order to virtualise up to 50 Sky and Wismote motes respectively, without taking into account the spatial location. In the one hand, in both cases all parameters increase as the number of IoT devices scales. The translation time is similar in both cases, since the cooja models are quite similar for the different contiki versions (less than 1 second). On the other hand, the compilation and load times are greater in the second case, since the contiki 3.1 operating system version is heavier, what lengthens the final start of the simulation. In both cases, the most expensive time is the compilation time.

According to the physical scenarios proposed, in the first case (20 sky motes), the IoT honeynet is ready in less than 15 seconds, and in the second case (50 wismote motes) it is up and running in less than 45 seconds.

These results can be compared with exiting related-works available in the literature [21], [22]. In this sense, these works, which measures similar magnitudes, requires up to 40 seconds in order to deploy only one non-LXC based honeypot, and therefore our solution outperforms those proposals.

Depending on the grade of the similitude needed, we can replicate not only the IoT devices with their configurations, but also the physical network topology. Indeed, we could generate a different topology in order to obtain some network benefits such as for instance, improving the convergence time in case we are not using static IP assignment. Since we are using RPL as routing protocol and the convergence time could be a handicap, the translator plugin allows to replicate the physical environment, and specify a concrete network topology.

Figure 14 shows the IoT honeynet virtualization by replicating the physical positions for each use case. The mote marked as 1 represents the router and each square of the grid is 10 meters large. The system is considered converged when the router learns through RPL routing protocol at least one route for each device in the virtual IoT honeynet, i.e. when all nodes are fully reachable from outside. Since the topology is determinant in this process, we compare the results of the physical topology deployment emulation with the results of a classical mesh topology deployment emulation where border router is located at the upper-left corner.

Figures 15 and 16 show the convergence time for both use cases by virtualising the physical topology and a mesh topology. For the first-floor use case, results are quite similar since the physical topology is close to a mesh topology. The best results are near to 17 seconds, while the worse results are close to 100 and 125 seconds respectively, being the vast majority comprised between 20 and 40 seconds. On the other hand, in the second-floor use case, since the physical distribution is more random, we can observe the mesh topology obtains significant better results, being the majority of the results close to 35 seconds with a maximum of 334 seconds and a minimum of 24 seconds. It should be noted that IoT deployments devices are sleeping as much as they can in order to save energy, thereby generating a great dispersion in the results. The average values are close to 40 and 191 seconds, respectively.

Regarding filtering and forwarding policies enforcement, the performance evaluation accomplished measures the time taken since the filtering/forwarding policies enforcement have been requested, until them have been enforced in the vSwitch. Since

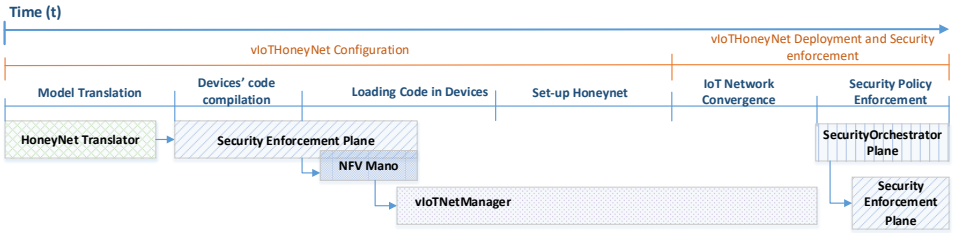


Fig. 9: Testbed measured times representation

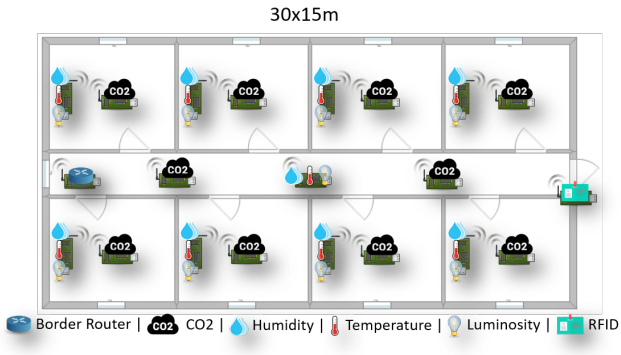


Fig. 10: Testbed - Sky 20 physical distribution

in the experiments the networking policies are applied for a whole IoT subnet, the time taken by policies is independent of the number of the IoT devices. In this regard, the translation filtering and forwarding processes are independent of the use case, and the lightest steps in terms of time consumption.

Table II shows the performance time in the IoT honeynet policy enforcement process. For the first use case the virtual IoT honeynet are deployed in less than 60s, whereas in the second use case the average time is close to 4 minutes. These times might be improved through the use of static IP addresses (without the convergence period, the average time of the second use case is also below 60 seconds) or by a more detailed study of the simulator parameters for the convergence times (out of the scope of this paper).

In addition, different tests were conducted to analyze the CPU and RAM memory usages of our vIoTHoneynet manager. Figures 17 and 18 show the resources consumption measured for the different use cases. The CPU metrics are almost use case independent and the simulation is using completely one CPU at a 100%, regardless of the number of devices. Finally, as it was predictable, RAM memory grows according to the incremental number of devices, being bigger in the second use case, since the nodes are more complex. Comparing these memory consumption results against the performance results achieved by current related works, it can be seen that, in our solution, an IoT honeynet of 10 Wismote IoT honeypots consumes around 102,4 MB of RAM, whereas previous works [21], [22] requires up to 726 MB of RAM for only one non-LXC based honeypot.

VII. Conclusions

This paper has exposed a novel solution to manage dynamically virtual IoT HoneyNets to mitigate cyberattacks in SDN/NFV-enabled IoT networks. The proposal allows administrators to deploy IoT Honeynets as a service through high level security policies over IoT infrastructures such as Smart Buildings. The approach adopted allows to replicate the physical IoT architecture on a virtual environment, by translating the physical architecture model to common interoperable IoTHoneyNet data model, and in turn, translating it to a virtualized environment deployed as VNFs. The whole process is driven by network security policies defined over the SDN controller and NFV MANO, whereby filtering, dropping and diverting the network traffic dynamically, and adapting the network behavior according to the new deployed vIoTHoneyNets needs.

The performance evaluation accomplished has demonstrated the feasibility of the proposed solution. Results has shown the successful deployment of IoT Honeynets with full connectivity. The deployment times behaves as expected, following a linear increasing trend as the number of nodes grows. Besides, the proposal has demonstrated that the virtual IoT Honeynets can be deployed on demand in a totally transparent way to the attacker, since the network behavior modification is performed fast, once the IoT Honeynet has been deployed.

As future work, we envisage to investigate on virtual IoTHoneynet for 5G-enabled IoT devices to reach broader and WAN scenarios. Finally, we also expect to design and implement, in the scope of ANASTACIA cognitive approaches (e.g. based on AI), in order to counter cyber-attacks in IoT.

Acknowledgments

This work has been partially funded by "Fundacion Seneca de la Region de Murcia", under the program "Jimenez de la Espada de Movilidad Investigadora, Cooperacion e Internacionalizacion" (20177/EE/17). The research has been also supported by EU projects H2020 ANASTACIA, G.A. 731558 and H2020 5G-PPP ICT-2016-2 SliceNet project GA 761913, as well as by the a postdoctoral INCIBE grant, with code INCIBEI-2015-27363.

References

[1] Y. Gao, Y. Peng, F. Xie, W. Zhao, D. Wang, X. Han, T. Lu, and Z. Li, "Analysis of security threats and vulnerability for cyber-physical systems," in *Proceedings of 2013 3rd International Conference on Computer Science and Network Technology*, Oct 2013, pp. 50–55.

[2] J. Granjal, E. Monteiro, and J. S. Silva, "Security for the internet of things: A survey of existing protocols and open research issues," *IEEE Communications Surveys Tutorials*, vol. 17, no. 3, pp. 1294–1312, thirdquarter 2015.

[3] A. Dunkels, B. Gronvall, and T. Voigt, "Contiki - a lightweight and flexible operating system for tiny networked sensors," in *29th Annual IEEE International Conference on Local Computer Networks*, Nov 2004, pp. 455–462.

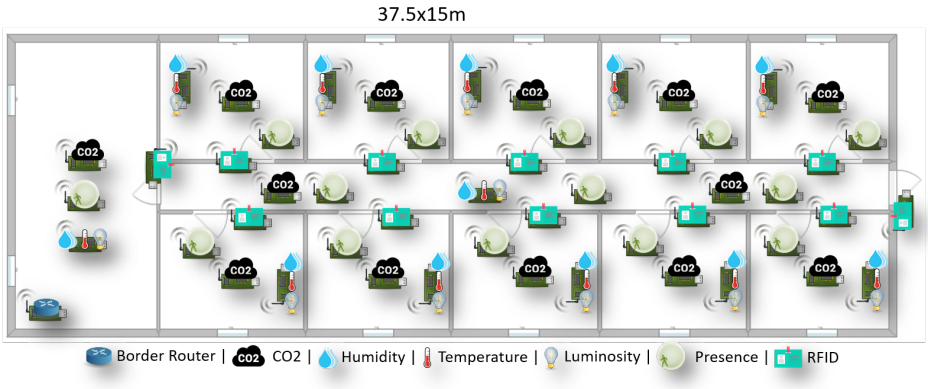


Fig. 11: Testbed - Wismote 50 physical distribution

Use case	Translation	Simulation start	Convergence	Filtering	Forwarding	Total (s)
20-mesh	0.288	13.26	36.48	0.37	0.36	50.758
20-phy	0.288	13.26	37.89	0.37	0.36	52.168
50-mesh	0.53	43.41	39.93	0.37	0.36	82.6
50-phy	0.53	43.41	191.57	0.37	0.36	236.24

TABLE II: Policy enforcement timing

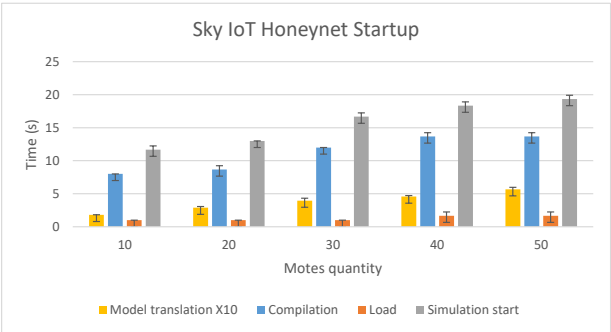


Fig. 12: IoT Honeynet Startup time. Sky Contiki

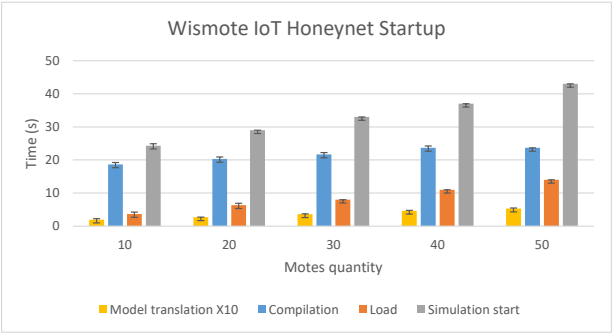


Fig. 13: IoT Honeynet Startup time. Wismote Contiki

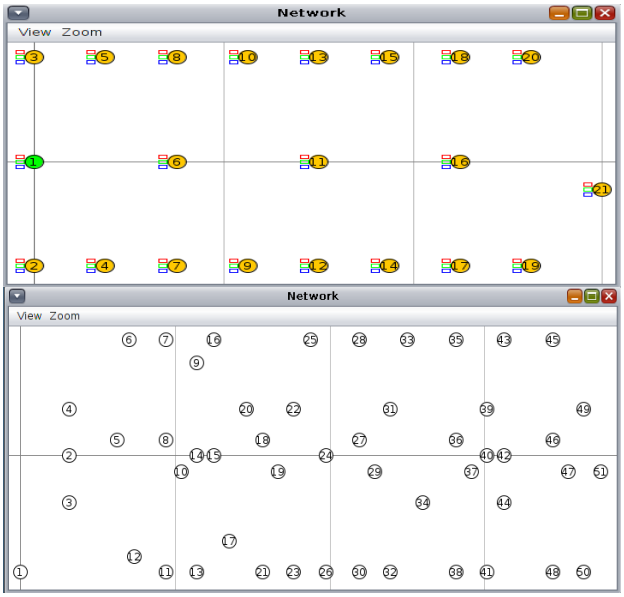


Fig. 14: Testbed Physical Topology

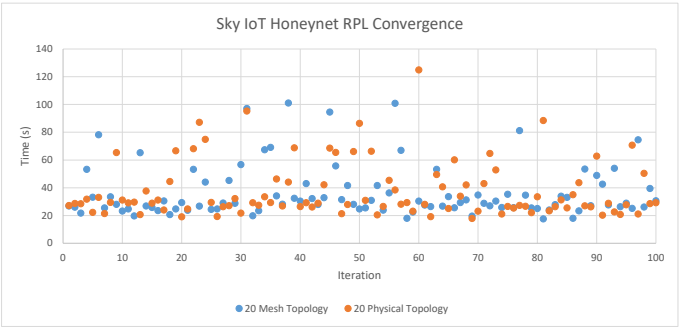


Fig. 15: IoT Honeynet RPL Convergence. Sky Contiki

[4] A. Molina Zarca, J. B. Bernabe, R. Trapero, D. Rivera, J. Villalobos, A. Skarmeta, S. Bianchi, A. Zafeiropoulos, and P. Gouvas, "Security management architecture for nfv/sdn-aware iot systems," *IEEE Internet of Things Journal*, vol. 6, no. 5, pp. 8005–8020, Oct 2019.

[5] J. B. Bernabe and A. Skarmeta, "Introducing the Challenges in Cybersecurity and Privacy - The European Research Landscape," in *Challenges in Cybersecurity and Privacy - the European Research Landscape*, ser. RIVER PUBLISHERS SERIES IN SECURITY AND DIGITAL FORENSICS, J. B. Bernabe and A. Skarmeta, Eds. River Publishers, 7 2019, pp. 1–21. [Online]. Available: https://doi.org/10.13052/rp-9788770220873

[6] S. Ziegler, C. Crettaz, E. Kim, A. Skarmeta, J. B. Bernabe, R. Trapero, and S. Bianchi, *Privacy and Security Threats on the Internet of Things*. Cham: Springer International Publishing, 2019, pp. 9–43.

[7] M. Ahmad, T. Younis, M. A. Habib, R. Ashraf, and S. H. Ahmed, "A review of current security issues in internet of things," *Recent Trends and Advances in Wireless and IoT-enabled Networks*, p. 11, 2019.

[8] A. BOUDI, I. FARRIS, M. BAGAA, and T. TALEB, "Assessing lightweight virtualization for security-as-a-service at the network edge,"

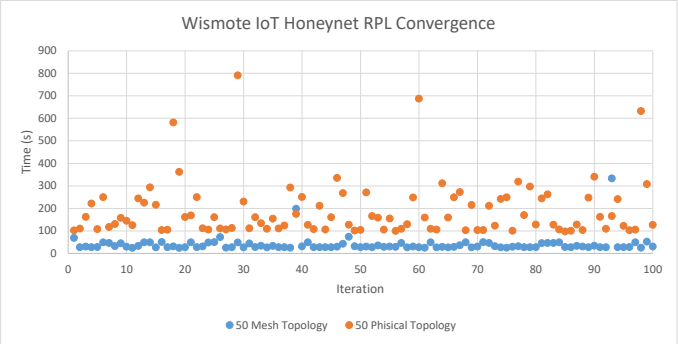


Fig. 16: IoT Honeynet RPL Convergence. Wismote Contiki

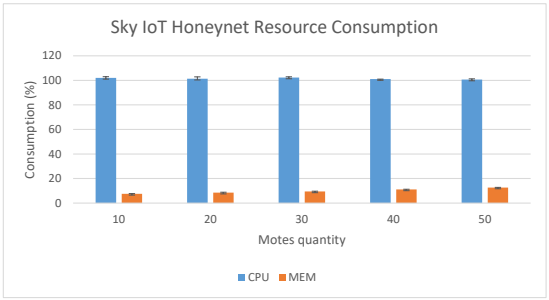


Fig. 17: Resource usage consumuption. Sky Contiki.

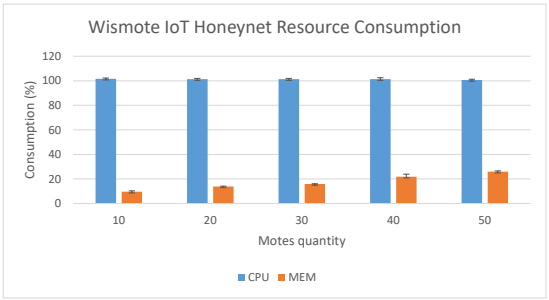


Fig. 18: Resource usage consumuption.Wismote Contiki

IEICE Transactions on Communications, vol. E102.B, no. 5, pp. 970–977, 2019.

[9] K. Chen, S. Zhang, Z. Li, Y. Zhang, Q. Deng, S. Ray, and Y. Jin, "Internet-of-things security and vulnerabilities: taxonomy, challenges, and practice," *Journal of Hardware and Systems Security*, vol. 2, no. 2, pp. 97–110, 2018.

[10] C. Vorakulpipat, E. Rattanalerdnusorn, P. Thaenkaew, and H. D. Hai, "Recent challenges, trends, and concerns related to iot security: An evolutionary study," in *2018 20th International Conference on Advanced Communication Technology (ICACT)*. IEEE, 2018, pp. 405–410.

[11] J. P. Santos, R. Alheiro, L. Andrade, V. Caraguay, A. Leonardo, L. I. Barona López, M. A. Sotelo Monge, L. J. Garcia Villalba, W. Jiang, H. Schotten *et al.*, "Selfnet framework self-healing capabilities for 5g mobile networks," *Transactions on Emerging Telecommunications Technologies*, vol. 27, no. 9, pp. 1225–1232, 2016.

[12] S. Ziegler, A. Skarmeta, J. Bernal, E. Kim, and S. Bianchi, "Anastacia: Advanced networked agents for security and trust assessment in cps iot architectures," in *2017 Global Internet of Things Summit (GIoTS)*, June 2017, pp. 1–6.

[13] I. Farris, T. Taleb, Y. Khettab, and J. Song, "A survey on emerging sdn and nfv security mechanisms for iot systems," *IEEE Communications Surveys & Tutorials*, vol. 21, no. 1, pp. 812–837, 2018.

[14] A. M. Zarca, D. Garcia-Carrillo, J. B. Bernabe, J. Ortiz, R. Marin-Perez, and A. Skarmeta, "Enabling virtual aaa management in sdn-based iot networks," *Sensors*, vol. 19, no. 2, p. 295, 2019.

[15] A. Molina Zarca, J. Bernal Bernabe, I. Farris, Y. Khettab, T. Taleb, and A. Skarmeta, "Enhancing iot security through network softwarization and virtual security appliances," *International Journal of Network Management*, vol. 28, no. 5, p. e2038, 2018, e2038 nem.2038. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/nem.2038

[16] S. Do, L. V. Le, B. S. P. Lin, and L.-P. Tung, "Sdn/nfv based internet of things for multi-tenant networks," *Transactions on Networks and Communications*, vol. 6, no. 6, pp. 40–40, 2018.

[17] D. Sinh, L.-V. Le, B.-S. P. Lin, and L.-P. Tung, "Sdn/nfv—a new approach of deploying network infrastructure for iot," in *2018 27th Wireless and Optical Communication Conference (WOCC)*. IEEE, 2018, pp. 1–5.

[18] Á. L. V. Caraguay, P. L. González, R. T. Tandazo, and L. I. B. López, "Sdn/nfv architecture for iot networks." in *WEBIST*, 2018, pp. 425–429.

[19] S. Do, L.-V. Le, B.-S. P. Lin, and L.-P. Tung, "Sdn/nfv-based network infrastructure for enhancing iot gateways," in *2019 International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*. IEEE, 2019, pp. 1135–1142.

[20] A. D. Oza, G. N. Kumar, and M. Khorajiya, "Survey of snaring cyber attacks on iot devices with honeypots and honeynets," in *2018 3rd International Conference for Convergence in Technology (I2CT)*. IEEE, 2018, pp. 1–6.

[21] W. Fan, D. Fernández, and V. A. Villagrá, "Technology independent honeynet description language," in *Model-Driven Engineering and Software Development (MODELSWARD), 2015 3rd International Conference on*. IEEE, 2015, pp. 303–311.

[22] W. Fan, D. Fernández, and Z. Du, "Versatile virtual honeynet management framework," *IET Information Security*, vol. 11, no. 1, pp. 38–45, 2016.

[23] A. Guerra Manzanares, "Honeyio4: the construction of a virtual, low-interaction iot honeypot," B.S. thesis, Universitat Politècnica de Catalunya, 2017.

[24] M. Banerjee and S. Samantaray, "Network traffic analysis based iot botnet detection using honeynet data applying classification techniques," *International Journal of Computer Science and Information Security (IJCSIS)*, vol. 17, no. 8, 2019.

[25] M. Wang, J. Santillan, and F. Kuipers, "Thingpot: an interactive internet-of-things honeypot," *arXiv preprint arXiv:1807.04114*, 2018.

[26] N. Naik, C. Shang, Q. Shen, and P. Jenkins, "Intelligent dynamic honeypot enabled by dynamic fuzzy rule interpolation," in *2018 IEEE 20th International Conference on High Performance Computing and Communications; IEEE 16th International Conference on Smart City; IEEE 4th International Conference on Data Science and Systems (HPCC/SmartCity/DSS)*. IEEE, 2018, pp. 1520–1527.

[27] S. Dowling, M. Schukat, and E. Barrett, "Improving adaptive honeypot functionality with efficient reinforcement learning parameters for automated malware," *Journal of Cyber Security Technology*, vol. 2, no. 2, pp. 75–91, 2018.

[28] U. D. Gandhi, P. M. Kumar, R. Varatharajan, G. Manogaran, R. Sundarasekar, and S. Kadu, "Hiotpot: surveillance on iot devices against recent threats," *Wireless personal communications*, vol. 103, no. 2, pp. 1179–1194, 2018.

[29] W. Fan and D. Fernández, "A novel sdn based stealthy tcp connection handover mechanism for hybrid honeypot systems," in *2017 IEEE Conference on Network Softwarization (NetSoft)*. IEEE, 2017, pp. 1–9.

[30] H. Lin, "Sdn-based in-network honeypot: Preemptively disrupt and mislead attacks in iot networks," *arXiv preprint arXiv:1905.13254*, 2019.

[31] "Common Information Model (CIM), DMTF." http://www.dmtf.org/standards/cim.

[32] F. Osterlind, A. Dunkels, J. Eriksson, N. Finne, and T. Voigt, "Cross-level sensor network simulation with cooja," in *Proceedings. 2006 31st IEEE Conference on Local Computer Networks*, Nov 2006, pp. 641–648.

[33] C. Basile, "D4.2 Policy transformation and optimization techniques, Secured EU project."

[34] S. Mamoru, N. Masafumi, K. Tadashi, K. Minoru, and S. Yuji, "Mirai botnet detection and countermeasures," *Internet Infrastructure Review (IIR) Vol.33*, 2016.

[35] R. Danyliw, J. Meijer, and Y. Demchenko, "The incident object description exchange format (iodef)," *Internet Engineering Task Force (IETF), RFC-5070*, 2007.

[36] O. Forum, "Open command and control (openc2)," https://openc2.org/members.html.