Distributed Middlebox Architecture for IoT Protection

Lionel Metongnon*†, Ramin Sadre*, Eugène C. Ezin†
*ICTEAM, Université catholique de Louvain, Belgium †IFRI, Université d'Abomey-Calavi, Benin
Email: lionel.metongnon@uclouvain.be, ramin.sadre@uclouvain.be, eugene.ezin@imsp-uac.org

***Abstract*—The Internet of Things (IoT) is not one single entity, but a collection of different devices, communication technologies, protocols and services. IoT systems can span a large number of individually managed networks that are interconnected through the Internet and host the different components of an IoT application, such as sensor devices, storage servers and data processing services. Protecting such a complex multi-party system from abuse becomes a very challenging task. New difficulties arise everyday when policies are updated or new collaborations and federations appear between entities. Moreover, hacked IoT devices can also become the source of powerful attacks, as the Mirai malware has demonstrated, and therefore a danger for the other involved parties.**

In this paper, we propose an approach to improve the management and protection of collaborating IoT systems using distributed intrusion detection and permission-based access control. Our approach is based on interconnected middleboxes that monitor the communication between the various IoT networks and are able to stop incoming as well as outgoing attacks. We evaluate our approach through experiments with different types of attacks.

***Index Terms*—IoT, Federation Network, Security Network, Distributed mitigation**

1. Introduction

The rise of the Internet of Things (IoT) has resulted in a world-wide deployment of billions of small connected devices. Security-wise, these devices pose a major challenge. Often, they are too resource constrained to run sophisticated intrusion prevention software. In addition, by their nature, these devices are easily "'forgotten"'; once deployed, their firmware is not updated regularly and they do not receive the same attention by network managers as, for example, a desktop PC. Their vulnerability and their number make them an attractive target for attackers since, when hacked and infected with botnet software, they can be misused for powerful large DDoS attacks against other Internet participants [1], [2], [3], [4], [5], as shown by recent DDoS attacks achieving 600 Gbps [6] and 1.2 Tbps [7]. The state of the art in defending against such DDoS attacks is attack mitigation at the target. Companies like Arbor Networks, Renesys, Cloudflare, or Akamai use CDN networks to filter and absorb malicious traffic. However, although this approach is effective from the viewpoint of the target, the attack traffic still consumes resources on the path from the source to the mitigation point and the mitigation does not prevent a botnet from attacking other, potentially less protected hosts.

This situation is further complicated by the increasing complexity of the IoT ecosystem, which does not only consist of sensor devices but also encompasses intermediate data handling nodes, servers to store and process data, and devices hosting the end-user interfaces (e.g. smartphones). All these components are managed independently and by different operators and can be combined to build new applications. For example, an application might need real-time sensor data from the IoT networks of two Smart Cities A and B, historical data stored in a datacenter C, and the data mining capacities of a cloud D. Protecting such a multi-party environment from abuse becomes a challenging task. New difficulties arise everyday when policies are updated or new collaborations and federations appear between entities. On the other hand, a pragmatic approach where all devices and services are allowed to mutually access each other without restrictions can have dramatic consequences in the case an attacker manages to gain control over a sufficient number of hosts, as demonstrated by the recent DDoS attacks.

To address the above challenges, in this paper we propose a permission-based system of cooperating middleboxes that monitor and control the communication between IoT networks. The middleboxes are interconnected through a management layer that allows to implement different access policies and to propagate information on detected attacks. Since the middleboxes operate on network level they can be easily deployed in existing IoT networks without requiring modifications on the end hosts. Traffic monitoring and filtering by the middleboxes is bidirectional and flow based. We validate our approach by experiments in a virtual environment where we demonstrate how different types of attacks can be stopped at the source and effectively mitigated.

2. Related Work

Access control in IoT networks. By default, many resource-constrained IoT devices only implement a minimum level of access control, for example in the form of a password request, because complex access control systems are resource consuming. In addition, the typical activity of an IoT device, i.e. providing sensor data, is short and sparse and discourages the use of protocols with a large overhead.

978-3-903176-24-9

Researchers have proposed more advanced techniques for access control. Liu *et al.* [8] implement the idea of a register authority (RA) to authenticate users and then to control device accesses inside the IoT network in order to reduce the impact of vulnerability exploits. Their solution requires that nodes that are contacted by other hosts first communicate with the RA to check whether access should be granted. Cruz-Piris *et al.* [9] propose to extend the MQTT protocol by user authentication and access rules. Since MQTT is push based, the access control is implemented on the servers, which are, in general, more powerful than the sensor nodes. No solution is proposed for connection attempts toward the sensor nodes, as done, for example in pull-based protocols like CoAP or during management activities. Novo [10] proposes an architecture using a blockchain to create a distributed access solution where IoT nodes send their traffic through management hubs providing access control. In the opposite direction, i.e. when receiving traffic, a node has to ask its hub to know if it should answer. In Xu *et al.* [11], a host has to request a capability token from a Local Coordinator before it can access other hosts. Policies are stored in the cloud where the coordinators can access them.

The above approaches require an active participation of the sending or receiving device in the access control. This makes them unsuitable for scenarios where the existing software on the IoT nodes cannot be updated or where nodes do not have enough resources to implement sophisticated protocols. For this reason, we propose to enforce access control in the network without relying on the end nodes.

Another aspect is the management of the access control itself which can become complex in multi-party and federated IoT environments. With its architecture of local coordinators and higher-level management implemented in the cloud, the previously mentioned publication by Xu *et al.* [11] addresses federated systems. We follow a similar hierarchical design. However, in our system, access control is implemented in middleboxes that monitor and filter the network communication between the collaborating systems. In contrast to [11], the end nodes are not directly involved.

DDoS mitigation. DDoS attacks are currently one of the most powerful attacks to bring down networked services. In a DDoS attack, traffic generated by a large number of hosts and amplified through different techniques floods the target. In commercially available DDoS protection services, the mitigation is done at the target side and consists in redirecting, filtering and absorbing the attack traffic by proxies and CDNs. The authors of [12] propose the usage of network function virtualization (NFV) to allocate dynamically resources to deal with DDoS traffic. Using a dispatcher and on-demand agents, they distribute the traffic to avoid congestion. Mitigation at the target is not a perfect solution, though, since the attack traffic still traverses the Internet and the attacker, even if identified, can still attack other, less protected, hosts.

Although the best position to detect an attack is at the victim side [13], DDoS attacks should be ideally stopped at the attacker side, as concluded in surveys [14], [15]. However, this is difficult to achieve due to the highly distributed nature of the attacks. In [16], the authors propose to mitigate DDoS attacks on Internet Service Provider (ISP) level using moving target defense (MTD) and Software Defined Networking (SDN) techniques. Their approach requires ISP to share their networks and to allow modifications of their routing rules. This level of trusts between the ISP can be difficult to reach and the effectiveness of the solution will depend on the number of ISP participating. In our solution, the mitigation is done at the network border, so the traffic is directly dropped there when an attack is detected.

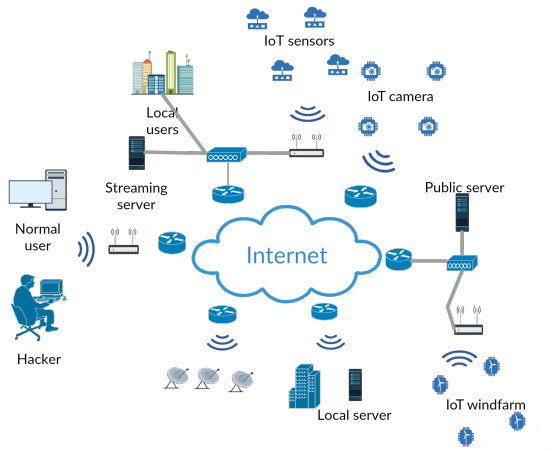


Figure 1: Collaboration between different IoT networks

3. Distributed Middlebox Architecture

3.1. Motivation

The Internet of Things is a not a single network of homogeneous devices and services. Its various components, that is sensors, servers, end-user devices, etc., are hosted in smaller and larger networks owned and managed by different organizations. Building complex IoT applications means to aggregate and process data from different sources and at different places in the Internet. Thanks to Machine-to-Machine communication this can happen without human intervention. However, it requires that the different entities have agreements on who is allowed to communicate with whom. An example is depicted in Figure 1, showing various IoT networks providing different types of data and services and users wanting to access them. In order to collaborate in a safe way, security policies have to be carefully defined and implemented. For example, IoT cameras should send video streams only to the streaming server.

Implementing such policies on all sites is cumbersome and error prone. Small mistakes can take months to be detected and will expose the affected subsystems to attacks. Although similar problems exist in other types of networked systems, they are particularly serious in the context of IoT with its billions of poorly managed devices. Sometimes, network administrators or end users might not be even aware

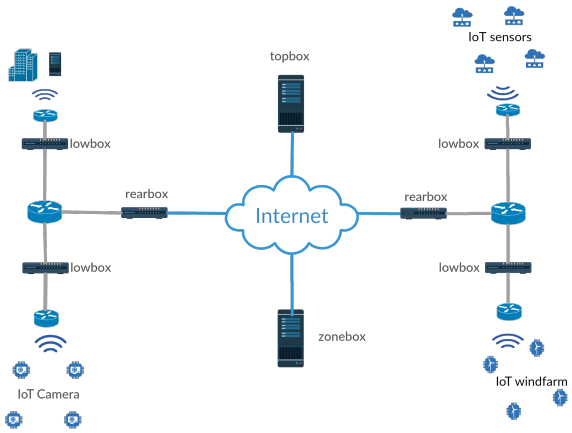


Figure 2: System overview

that vulnerable devices exist in their networks. It is realistic to assume that sooner or later an attacker will gain control over them and use them to perform attacks against other parts of the system.

The goal of this paper is to provide a security system that helps collaborating IoT networks to communicate and exchange information without exposing them to attacks. Based on the above observations, we define the following requirements to such a security system:

1) It should allow to define and enforce access control between all participating networks.
2) It should operate on network level since we cannot expect that all vulnerable devices are known in advance and that their vulnerabilities can be fixed.
3) It should be independent of the concrete communication protocols used between IoT components.
4) It should allow to detect DDoS and similar attacks.
5) Once an attack is detected, the system should stop it at the source in order to prevent that the attack traffic strains the Internet and that other collaborating networks are attacked.

3.2. System overview

Our security system consists of middleboxes deployed inside or at the border of networks hosting IoT components (Figure 2). These boxes are placed such that they can monitor and filter the traffic between the network and the Internet. In larger IoT networks consisting of several subnetworks, more middleboxes can be also deployed in front of the individual subnetworks. A middlebox has two tasks:

1) When a host in a network tries to access a host in another network, the middlebox of the target's network checks whether the source host is authorized to access the target host. If not, the traffic is blocked. Similarly, other policies, such as limitations on the number of messages, can be enforced.
2) The middlebox looks for signs of attack traffic entering or leaving the network. Since the box only monitors network traffic, our focus in this paper is on attacks that can be detected by flow-based intrusion detection, such as DoS and DDoS attacks, etc.

To achieve both tasks, middleboxes are forming the lower layer of a hierarchical structure of security devices. This structure allows to easily implement security policies between all collaborating (or federating) networks. General security policies are defined in form of rules by the network administrators. They determine which and how much communication is allowed between the IoT components hosted by the different networks, in this way guaranteeing on one hand a smooth operation of legitimate IoT application and preventing, on the other hand, unauthorized accesses attempted by hacked devices.

The hierarchical structure also allows middleboxes to exchange information on identified threats. Concretely, if a middlebox detects attack traffic directed to the network it is responsible for, it will block the traffic and then check if the source of the attack is located in one of the collaborating networks. If this is the case, it will notify the middlebox of the source, which will then filter the attack traffic right in front of the attack source before it can leave the source network. We explain how the middleboxes work in Section 3.3. The higher part of the hierarchical security structure, which we call management layer, is described in Section 3.4.

3.3. The middleboxes

To protect an IoT network, there is at least one box (the rearbox) needed at the network border. Depending on the size of the network, its topology and the security policies the owner of the network wants to enforce there could be one or more lowboxes deployed inside the network. Only the rearbox communicates with the outside world and it is in charge of configuring the lowboxes.

The middleboxes implement permission-based access policies on network flow level. When a source host S sends a packet to a destination host D, the middlebox of the source network first checks its list of permissions if it has a permission matching the flow identifier[1] of the packet. If there is no such permission and there exists a middlebox which is responsible for D, the middlebox of S requests a permission from D's middlebox. If the destination middlebox declines the request the traffic is blocked at the source. Similarly, the destination middlebox will block flows for which no permission has been requested beforehand. Note that permissions have a (configurable) time-to-live.

This two-party authorization allows implementing dynamic access control. For example, a destination network that is experiencing heavy load can decide to temporarily decline new permission requests or issue permissions with a short time-to-live. On the other hand, permissions with a long duration reduce the communication overhead and the latency introduced by the verification process when a new permission is requested from a remote middlebox. Network administrators might also decide to permanently

1. We define the flow identifier as the classic five-touple (source address, destination address, source port, destination port, protocol).

install permissions on a middlebox to make it accept traffic from networks that do not have a middlebox.

A middlebox has four different lists updated in real time:

1) The *permission list* contains all permissions granted or denied. A permission contains the flow identifier (the five-tuple), the duration of this permission and finally the time when the permission was issued.
2) The *policy list* represents the policies that the owner of the network protected by middlebox wants to enforce for the individual devices inside the network.
3) The *flow list* contains information on all flows currently active, such as size in bytes, the number of packets, and the duration. This list is scanned by the detection methods (see below) for attack patterns.
4) The *alert list* contains the list of flows that have been identified as suspicious. Each entry contains the flow identifier of the affected flow and the type of offense detected for that flow.

Entries in the permission list, the flow list and the alert list have a duration after which they expire and are removed.

The policies in the policy list contain the following information for the individual devices:

- Information that is used when the middlebox has to decide whether a permission request should be accepted. For example, the network owner could define that only incoming connections towards port 80 are allowed for device 1.2.3.4 or that device 1.2.3.5 does not accept more than 10 simultaneous connections.
- Information used when sending a permission request to another middlebox. For example, the policy could specify that a device 1.2.3.4 needs permissions of 15 seconds when communicating with other hosts.
- What to do with flows from/to the device that are on the alert list, i.e. identified as suspicious. For example, the policy could define that suspicious flows should be blocked or rate limited.

There is also a *default policy* that is used when no specific policy exists for a device or when devices are communicating with a network without a middlebox. Policies are installed by the network administrator on the rearbox which duplicates it on the lowbox it manages.

For every outgoing packet, the box of the source network first checks if it is part of a new flow, i.e. no permission exists yet. If this is the case, a request is sent to the middlebox responsible for the destination host (if no box is present at the destination, the default policy is applied). The obtained permission is then recorded. If the flow already has a permission we check whether it is still valid to decide whether the packet can be forwarded or must be dropped.

For incoming packets, the box first checks whether the flow the packet belongs to was already granted permission. If not, we raise an alert and drop the packet. In addition, an entry is added to the permission list to block further packets from that flow for a specific duration configured in the policies and the source middlebox is notified. Packets with a permission are treated as usual. The default policy is applied if the packet comes from an unmanaged network.

In addition to managing and enforcing permissions, a middlebox performs two other tasks on every flow:

1) *Intrusion detection:* Traffic is analyzed for attack patterns. If a flow with suspicious behavior is identified by the target middlebox, it is added to the alert list and the source middlebox is notified, so it can add the flow to its alert list, too.
2) *Attack mitigation:* Flows on the alert list are subject to mitigation actions. This can mean to complete block the flow or to follow a softer approach, such as a rate limitation. The latter prevents that false positives completely block the normal functioning of an IoT application.

3.4. The management layer

Middleboxes must be able to find and communicate with each other. We propose a hierarchical design inspired by DNS for scalability. Rearboxes connect to a *zonebox* which operates on country, ISP or AS level. The registration of a rearbox to its zonebox is mandatory. During registration, the rearbox provides a list of IP addresses it is responsible for. The communication between the boxes is encrypted, and the keys are exchanged during the registration. The zoneboxes are managed by the country or AS they protect. If needed, another level can be added to the hierarchy which interconnects the zoneboxes (indicated as *topbox*). The boxes in the management layer serve two purposes:

- To provide information about other middleboxes to a middlebox: A middlebox who needs to find the middlebox responsible for a target or source address contacts its zonebox which might, similar to DNS, forward the request to the topbox.
- To provide mitigation on large scale: If mitigation at a rearbox fails or cannot be done because the attack source is in a network without a middlebox, the zonebox can try to find another box that is on the path between the attacker and the victim and that could block the attack traffic. Such boxes could be deployed by the ISPs inside their backbone or at PoPs.

The above requires algorithms to find the optimal mitigation point, agreements between the operators of IoT networks and their ISPs, etc. Due to space restrictions, we will not discuss this aspect further in this paper. It should be noted that such a setup is only required for large scale treatment of attacks. If IoT networks want to collaborate on small scale, only middleboxes and a zonebox are needed. The latter is the scenario that we will evaluate in the next sections.

4. Experimental methodology

We validate our approach on a network topology emulated with mininet (http://www.mininet.org). The topology contains three IoT networks that are interconnected through a central router representing the Internet:

- $Network\ 1$ contains 20 nodes representing an IoT network consisting of two subnetworks;
- $Network\ 2$: like $Network\ 1$, but with only 10 nodes;

- $Network$ 3 hosts two nodes representing a data storage or processing server and a malicious node, respectively;

The hosts in $Network$ 1 and $Network$ 2 represent resource-constrained IoT devices with a wireless connection. To simulate the latter, we set a packet loss rate of 2%, a bandwidth of 100 Mb/s and 5 ms delay. For the server and the attacker in $Network$ 3, we choose the same bandwidth but a delay of 2 ms. We do not simulate a topbox since our main goal is to show the advantage of the mitigation at the attack source in the following. The rearboxes are emulated by programmable switches at the borders of each of the three networks controlled with the Pox controller [17].

Background traffic represents normal interactions between the different networks. The IoT devices in the two networks emulate IoT services that send respectively reply to messages sent on UDP. For simplicity the same policy is applied to all the nodes inside the system. The policy treats all the nodes as equal in term of priority, i.e. it does not impose limitations on the amount of data they are allowed to exchange nor on the number of connections. By default, permissions of a duration of 10 s are requested.

We perform experiments with different types of attacks. In the next section, we describe for each experiment the concrete background traffic used in that experiment, the nature of the attack, and the method used by the middleboxes to detect it. The controller scans the alert list, i.e. the list of flows marked as suspicious by the detection method, every 500 ms and executes for each alert the mitigation action depending on the offense type indicated in the alert. In our experiments, the default policy is to block the flow. It should be remembered that the detection and mitigation procedures run on all middleboxes, i.e. an attack flow can be potentially detected and mitigated by the middlebox of the source as well as by the middlebox of the destination according to their policies. If the destination middlebox detects the attack it will send an alert to the source box.

5. Experimental validation

Scenario 1: DoS attack. In our first experiment, the malicious host in $Network$ 3 sends UDP packets of 500 bytes with an inter-packet time of 3 s against a set of 15 random nodes of $Network$ 1. This simulates a generic DoS attack where an attacker tries, for example, different messages to exploit a vulnerability. For the background traffic, 10 nodes in $Network$ 1 send UDP packets of 20 bytes to random nodes in $Network$ 2 with an inter-packet time of 5 s. The nodes in $Network$ 1 reply with a UDP packet to each incoming packet. We record the traffic before the victims' middlebox, before the attacker's box and on the attacker node itself. The system is configured to detect a DoS attack when a node exchanges at least 5 packets with a destination and $\frac{size\ of\ flow}{flow\ duration} > 100$.

Figure 3 shows the number of packets per second observed at the different vantage points. The background traffic is mostly constant, except for a few losses caused by the configured packet loss rate (the emulated IoT application does not resend lost packets). The attack starts at time 9 s and is mitigated when the detection conditions are reached at around time 27 s. To better illustrate the functioning of the system, we first let only the middlebox of $Network$ 1 run the detection and mitigation process as show in Figure 3a. As can be seen, the traffic observed at the attacker side drops from around 10 packets per second to 5 packets/s since the middlebox of $Network$ 1 stops the attack packets before they can reach the IoT nodes. The legitimate background traffic is not affected. This represents the normal behaviour of today solution using firewall and IDS to stop the malicious traffic when identify.

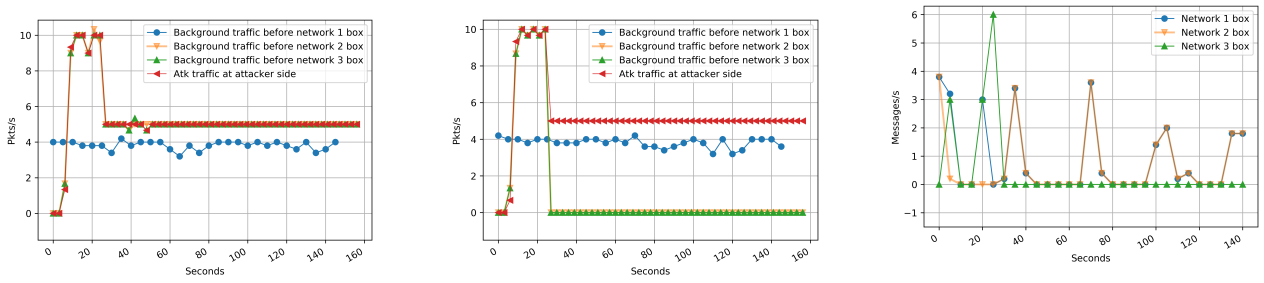
For Figure 3b, the middlebox on the attacker side is enabled, too. Once the attack is detected, the middlebox blocks the attack traffic at the source for an infinite duration, hence the drop to 0 packet/s after mitigation.

We have not included the management messages (permissions and alerts) in the packet statistics in Figure 3a and Figure 3b. Figure 3c shows the number of management messages exchanged. Note the periodic peeks caused by the limited life time of permissions and mitigation actions. In practice, the traffic overhead caused by the management messages is relatively low since multiple management messages can be aggregated into one packet.

Scenario 2: DDoS attack. In the second experiment, 10 nodes in $Network$ 1 and 10 nodes in $Network$ 2 send UDP packet packets of 50 bytes every seconds to the server in $Network$ 3. The nodes in $Network$ 2 start 10 s after the nodes in the first network. The goal is to simulate a small DDoS attack against the server. We are using the same background traffic as in the first experiment with the DoS attack. The middleboxes detect a DDoS attack when a host receives more than 5 kBytes and $4/5$ of the incoming flows exceed a duration of 15 s. Instead of completely blocking the attack source like in the DoS attack, we implement a less aggressive mitigation policy on the target middlebox that still allows to provide a basic service to legitimate requests coming from the source even when under attack: The mitigation consists in blocking one third of the flows every time the threshold is reached at the target box level. This is repeated until the attack traffic falls below the detection threshold.

Figure 4a and Figure 4b show the results when the traffic is only mitigated by the victim side (we do not show the background traffic separately in order to keep the figure readable). After mitigation, traffic is still sent from $Network$ 1 and 2, but it does not reach the target. In contrast, no attack traffic leaves $Network$ 1 and $Network$ 2 if the source middleboxes participate in the mitigation (Figure 4c). We omit the similarly looking plot for the management messages due to space restrictions.

The experiment demonstrates how network administrators can reach very specific goals in terms of quality of service by adapting the mitigation policy. For example, the system on the victim side could also apply a mitigation policy that is aggressive at the beginning (block everything when the attack is detected) and rolled back once the

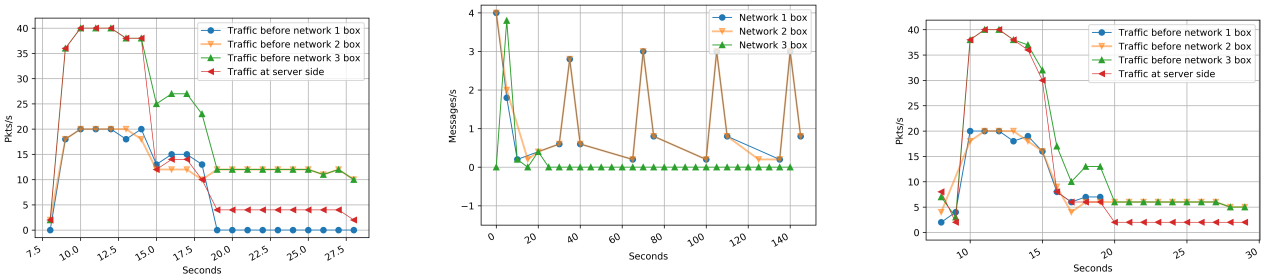


(a) Only target middlebox

(b) With source middlebox

(c) With source middlebox: management messages

Figure 3: DoS attack with different mitigation strategies



(a) Only target middlebox

(b) Only target middlebox: management messages

(c) With source middlebox

Figure 4: DDoS attack with different mitigation strategies

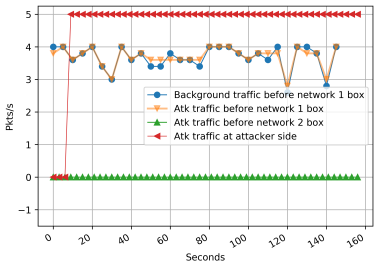


Figure 5: Reflection attack

source middleboxes confirm that they have activated their mitigation procedures.

Scenario 3: Reflection attack. In our last experiment, a malicious user in $Network$ 3 spoofs IP addresses of random nodes of $Network$ 2 and tries to attack them with an amplified reflection attack by sending messages to the devices in $Network$ 1 with an inter-packet time of 3 s. We also have background traffic from random nodes of $Network$ 2 that send packets to nodes in $Network$ 1.

Figure 5 shows that the reflection attack traffic doesn't pass the middlebox of $Network$ 3 because the latter did not receive permission requests from the middlebox of $Network$ 1 for that traffic. In fact, the attacker tried to request permissions on behalf of $Network$ 1 but was rejected by the authentication procedure. The legitimate background traffic is not affected. Although an attacker could guess the IP addresses and port numbers of an existing permitted UDP flow and successfully pass the middlebox in this way, the other rules specified by the network administrator in the policies (permission duration, limitation of packet rate etc.) would still apply.

6. Conclusion

We have presented a distributed approach for the protection of collaborating IoT networks. Our system consists of middleboxes that are deployed on network borders and that enforce security policies. Communication between the networks is monitored and controlled by a permission-based policy system implemented on flow level. In addition, the middleboxes detect attack patterns and automatically execute mitigation actions. Attacks originating from collaborating networks are stopped at the attack side or near it thanks to the ability of the middleboxes to exchange information on detected threats and access policies. Default policies regulate the communication with unmanaged networks. We show in our experiments that different types of attacks can be completely mitigated with our approach, often without disruption of regular traffic. A key characteristic of our approach is that it is purely network based and, therefore, does not require modifications on existing IoT end hosts nor their active participation in the protection scheme.

As future work, we plan to study alternatives to the tree-like organization of the boxes and their potential impact on the reliability and reaction time of the system.

References

[1] Michele De Donno, Nicola Dragoni, Alberto Giaretta, and Angelo Spognardi. Analysis of ddos-capable iot malwares. In *2017 Federated Conference on Computer Science and Information Systems (FedCSIS)*, pages 807–816. IEEE, 2017.

[2] Georgios Kambourakis, Constantinos Kolias, and Angelos Stavrou. The mirai botnet and the iot zombie armies. In *MILCOM 2017-2017 IEEE Military Communications Conference (MILCOM)*, pages 267–272. IEEE, 2017.

[3] O. Gayer, O. Wilder, and I. Zeifman. Cctv ddos botnet in our own back yard. https://www.incapsula.com/blog/cctv-ddos-botnet-back-yard.html. Accessed: 2018-02-11.

[4] L. Constantin. Thousands of hacked cctv devices used in ddos attacks. http://www.pcworld.com/article/3089346/security/thousands-of-hacked-cctv-devices-used-in-ddos-attacks.html, 2016. Accessed: 2018-02-11.

[5] D. Cid. Large cctv botnet leveraged in ddos attacks. https://blog.sucuri.net/2016/06/large-cctv-botnet-leveraged-ddos-attacks.html, 2016. Accessed: 2018-02-11.

[6] Elisa Bertino and Nayeem Islam. Botnets and internet of things security. *Computer*, 50(2):76–79, 2017.

[7] Scott Hilton. Dyn analysis summary of friday october 21 attack. https://dyn.com/blog/dyn-analysis-summary-of-friday-october-21-attack/, Accessed: 2018-02-11.

[8] Jing Liu, Yang Xiao, and CL Philip Chen. Authentication and access control in the internet of things. In *2012 32nd International Conference on Distributed Computing Systems Workshops*, pages 588–592. IEEE, 2012.

[9] Luis Cruz-Piris, Diego Rivera, Ivan Marsa-Maestre, Enrique De La Hoz, and Juan Velasco. Access control mechanism for iot environments based on modelling communication procedures as resources. *Sensors*, 18(3):917, 2018.

[10] Oscar Novo. Blockchain meets iot: An architecture for scalable access management in iot. *IEEE Internet of Things Journal*, 5(2):1184–1195, 2018.

[11] Ronghua Xu, Yu Chen, Erik Blasch, and Genshe Chen. A federated capability-based access control mechanism for internet of things (iots). In *Sensors and Systems for Space Applications XI*, volume 10641, page 106410U. International Society for Optics and Photonics, 2018.

[12] AHM Jakaria, Bahman Rashidi, M Ashiqur Rahman, Carol Fung, and Wei Yang. Dynamic ddos defense resource allocation using network function virtualization. In *Proceedings of the ACM International Workshop on Security in Software Defined Networks & Network Function Virtualization*, pages 37–42. ACM, 2017.

[13] Saman Taghavi Zargar, James Joshi, and David Tipper. A survey of defense mechanisms against distributed denial of service (ddos) flooding attacks. *IEEE communications surveys & tutorials*, 15(4):2046–2069, 2013.

[14] Christos Douligeris and Aikaterini Mitrokotsa. Ddos attacks and defense mechanisms: classification and state-of-the-art. *Computer Networks*, 44(5):643–666, 2004.

[15] Tao Peng, Christopher Leckie, and Kotagiri Ramamohanarao. Survey of network-based defense mechanisms countering the dos and ddos problems. *ACM Computing Surveys (CSUR)*, 39(1):3, 2007.

[16] Jessica Steinberger, Benjamin Kuhnert, Christian Dietz, Lisa Ball, Anna Sperotto, Harald Baier, Aiko Pras, and Gabi Dreo. Ddos defense using mtd and sdn. In *NOMS 2018-2018 IEEE/IFIP Network Operations and Management Symposium*, pages 1–9. IEEE, 2018.

[17] Pox controller. https://openflow.stanford.edu/display/ONL/POX+Wiki. Accessed: 2018-12-07.