An Architecture for Distributing and Enforcing IoT Security at the Network Edge

Luca Deri
IIT/CNR, ntop
Via Moruzzi 1
56124 Pisa
luca.deri@iit.cnr.it, deri@ntop.org

Arianna Del Soldato
IIT/CNR
Via Moruzzi 1
56124 Pisa
arianna.delsoldato@iit.cnr.it

***Abstract*—Distributed networks are now a reality, and enforcing security on a single place is no longer possible. This requires multiple devices to apply security policies at the network edge in order to limit unwanted traffic to leave the designated area, as well as implementing fine-grained policies similar to what micro-segmentation is offering. In order to achieve this goal, it is required to distribute device-specified security policies across the network in a secure and resilient way.**

This paper describes the design and implementation of a novel architecture for both distributing and enforcing security policies designed to protect simple IoT devices as well servers and workstations. The validation step on a real network, confirmed that it could be successfully used to improve the overall security by moving protection from the center towards the network edge.

Keywords*— *Network security, traffic monitoring, Internet of Things, Domain Name Server, Digital Object Architecture.

I. Introduction and Motivation

As stated by the principle of least privilege [1], the idea is that every user, process or computer has to access only the information that is necessary in order to carry on its activity. For this reason, for years computer networks have been segmented in three zones: trusted, untrusted and semi-trusted also known as DMZ (demilitarized zone). The trusted network corresponds to the internal network (LAN) that is accessible only by the organisation staff and thus that it is supposed to be trusted. Here internal users often access the Internet by means of proxy services that can analyse data being exchanged, this to avoid direct Internet communications. Modern networks are much more complex to protect with respect to the past, as there is no sharp separation between roles. In particular, in the internal network not all devices can be fully trusted as it used to be in the past. With the advent of the IoT (Internet of Things) [2], BYOD (Bring Your Own Device), remote surveillance/assistance applications, and cloud-based connected devices such as fridges and thermometers, the above security model cannot be used anymore. In the past, in fact, the solution for securely accessing a private network was through a VPN (Virtual Private Network) that created a secure, authenticated, encrypted channel on which the internal network was accessible. Today, instead, users prefer to access network devices using mobile applications, so device manufacturers have started to create one specific application per device model that needs to be accessed remotely. The result is that for accessing your surveillance camera there is mobile app A, for the NAS (Network Access Storage) app B, and so on. These mobile applications communicate with network devices using cloud-based services operated by the device manufacturer and no longer under the control of the network administrator as it used to be in the old VPN days. While users are typically happy of accessing their devices using a mobile application without paying attention to cumbersome remote access procedures, the result is that secure access to a remote network is no longer enforced by the local network administrators. Security is now delegated to the device manufacturer that often outsources the service to specialised companies that are more inclined to reduce costs rather than to secure the communications. The result is that these devices have potentially opened a security breach that is hard to control, hence the only option for really securing the private network is to prevent these devices to connect to the Internet at all, making some of them (e.g. a surveillance camera) not so useful anymore, and thus also making this solution unfit.

Another aspect to consider in this scenario, is that IPS/IDS (Intrusion Prevention/Detection Systems) and firewall devices are no longer enough to keep a network secure. This is because most traffic today is encrypted and thus impossible to inspect unless MITM-like (Man In The Middle) techniques are used, thing that is not always possible due to regulatory laws that in some countries prevent the use of these techniques. However, even if such MITM techniques could be used, they do not guarantee complete inspection as many popular applications (e.g. Skype or BitTorrent) do not use SSL but proprietary encryption techniques. This problem is even worse with low-cost cloud-connected devices, as many of them cannot be updated when a security flaw is detected, as manufactures do not issue periodic security fixes. This opens up the network to vulnerabilities and remote accessing from the Internet that can lead the use these devices to participate in distributed attacks as demonstrated by the attack to Dyn servers [4] when low cost devices create a major Internet service disruption.

In essence in the past years the Internet evolved from a mutual trust environment, where everyone could read, create and exchange information, into an environment where even devices and services we use for carrying on daily activities might be insecure [6]. Nowadays devices operating on the internal network cannot be longer trusted although they are installed on a privileged network [7, 8] and also because cloud-services keep open a privileged channel with the outside world. In corporate networks, a typical countermeasure to this problem is to implement micro-segmentation [9, 11], that is the act of splitting a computer network into multiple networks segments where data exchanged across segments it is not simply routed/switched at packed header, but instead payload inspected by security devices. While micro-segmentation can contribute to reduce security risks as it prevents unwanted traffic to spread outside of a segment, it does not solve the problem unless the segment size is set to one. Moreover, it is no longer possible to provide a high degree of security using only centralised security devices or ACL-based devices for routing/switching traffic as they are too far from the edge and thus Lan activities can still happen mostly unsupervised. In essence, it is time to somehow rethink security a bit and specialise it for the plethora of devices that are now populating modern networks. It is not possible to leave home or small-business networks unprotected simply because there is not a skilled system administrator able to design network growth and supervise the operations. Even the so-called “smart security devices” such as Cujo or FingBox, that are now popular on the consumer market, have been designed for supervising the Internet access from the local network but without implementing any fine grained security policy nor protecting the internal network from insider threats. We believe that it is not possible to reach our goal simply by adding a new security device, but rather by making sure that all the network devices collaborate together to enforce the overall security.

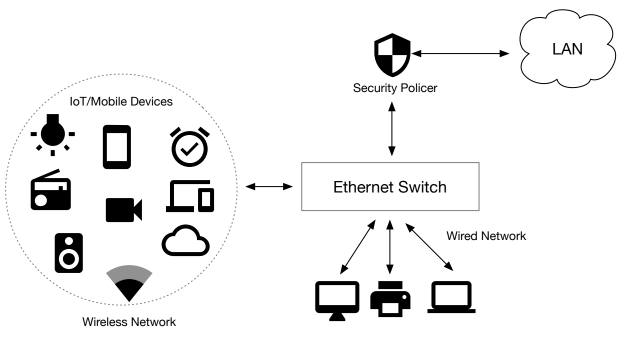
The motivation behind this paper is to move from coarse-grained security policies, implemented by a central firewall, towards edge-based fine-grained policy enforcement tailor-made for each device type/category [14]. Focusing on the edge, has become compulsory as [34] a great portion of all computing will happen at the network edge in the next couple of year. We propose moving security towards the edge of the network so that traffic policies enforcement can be applied as close as possible to the network devices [19]. This practice would stop unwanted device traffic from freely sending traffic, as well as reducing the load on central security devices that instead would have to be configured with thousands of policies. By leveraging on recent extensions to the DNS (Domain Name Server) protocol [12], this paper describes how these mechanisms could be profitably used for our needs as well how we could also exploit proposed extensions to other Internet protocols such as the DHCP. This means that the DNS in addition to being used for address resolution, it will also be used to distribute security policies as described later in this paper.

It is worth underlying that the scope of this work goes beyond network security. In fact, when an asset database is in place, this information could be used also for other purposes and be produced by multiple actors. For instance, passive network monitoring tools might access the database to complement packet-based traffic analysis with some extra information elements such as the operating system or the device type (e.g. printer, router, workstation or mobile device). This extra information not usually reported by probes, could be used by flow collectors for generating extra traffic metrics (e.g. what percentage of traffic is generated by mobile vs. static devices) and triggering new type of alerts not previously possible to generate (e.g. trigger an alert when a tablet receives an email via SMTP, or when a printer starts a SSH connection).

The rest of the paper is structured as follows. Section II describes the architecture design, section III evaluates related work, section IV describes the validation process and experiments, section V highlights some future work activities, and finally section VI concludes the paper.

II. Architecture Design

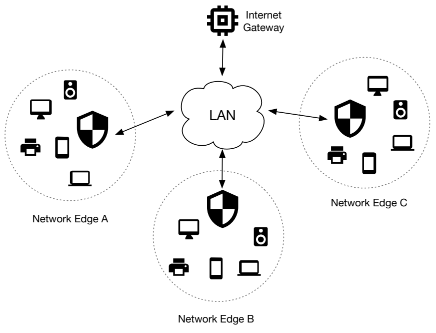
Most networks use devices such as firewalls and IPSs to enforce network security policies when connecting to the Internet. As the network edge is growing, this architecture needs to be complemented with additional mechanisms for enforcing local edge traffic that will not flow through the firewall. Policies can be natively implemented on existing devices or by means of additional devices to be deployed on the network. For instance, a Linux server can exploit the native firewall to implement such policies, whereas single-purpose devices such as printers or access points are not so versatile and thus need to be protected by deploying additional security devices. The proposed architecture is made of three main components: a *Traffic Policer* that is responsible for dropping/bridging network packets, an *Asset Policy Database* that contains information used by the policer in order to make decision and a *Network Discovery* the populates the database with information about the discovered network devices. Figure 1 highlights the *Traffic Security Policer* (or *Policer* in short) that is in essence a bump-in-the-wire that bridges the portion of the LAN to protect with the rest of the network.



1. Security Architecture

This device acts as a transparent bridge and drops selected packets not compliant with the specified security policies. On small/mid-size networks, the policer can run on a low-cost hardware device such as a RaspberryPI or a sub 50$ Linux OpenWRT-based device such as Ubiquity EdgeRouter X. The policer is deployed whenever we need to enforce edge traffic policies: it acts as a transparent traffic filtering device that can be placed on an existing network without changing the existing network configuration. Based on the network topology and security policies, one or more policiers can be deployed on a physical network: ideally, the more policers are deployed the better it is as it promotes micro-segmentation. All policers need to share the same security policies and make sure that whenever there is a change in policy, such modification is promptly distributed to all network edges.

The main difference between the policer and an IDS/IPS or a traditional firewall, relies on the nature of the filtering we perform that is not based on signature as on IPS/IDS. As described in section II.B, policer rules are fine-grained, feature per-device type configuration, and be layer-7 aware so we can filter both port-based (e.g. HTTP) and non-port-based protocols (e.g. BitTorrent or HTTP running on a non-standard port).



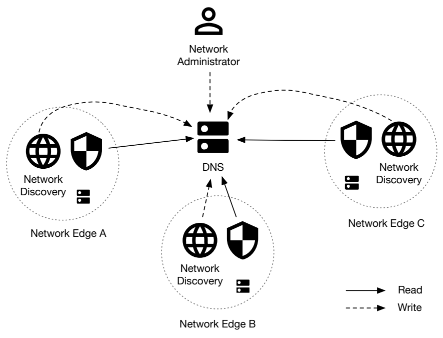
2. Security Policers Deployment

This means that we implement different security policies based on the device type. For instance, we define a default policy for tablets, another one for printers, and another for security cameras. The policer will apply the specified default policy unless for a given device a more specialised policy has been defined. Moreover this design allows human network administrators to define fine-grained policies while allowing coarse-grained policies to be applied on unknown devices types. In order to avoid humans to manually classify network devices according to categories (e.g. tablets, workstations, printers etc.), the policer takes advantage of a *Network Device Discovery* component that periodically performs an active network device discovery as described in section II.A. This allows new discovered devices to be bound to a generic device type policy (e.g. tablets cannot print, IoT devices cannot connect to the Internet) that could be further refined by human administrators as described later in this section. In addition, as networks can have multiple active policers at various network sites and be topologically distributed as depicted in figure 2, it is necessary to have a central location where policies and device assets information is securely stored and used to distribute it across the network. In particular, the proposed architecture is able to:

- dynamically define/remove security policies as devices are connected/disconnected from the network;
- be resilient to attacks and network faults in a way that security could still be enforced even in case of partial network disconnection or fault;
- prevent unauthorised changes to the asset policy database in order to disable security;
- protect heterogeneous devices ranging from single-task devices such as a temperature thermometer or a printer, to more general-purpose devices including tablets, and personal computers.

In order to satisfy the above requirements, our architecture has been based on the DNS (Domain Name System) [34], one of the core Internet protocols used to translate symbolic host names to numeric IP addresses and vice-versa.

By design, the DNS implements native mechanisms for distributing information, enforcing secure information exchange, and be resilient to attacks and faults. In the past few years, the DNS has been extended in scope, and it is now used for additional purposes not related to just IP address resolution such as SPF (Sender Policy Framework) and DKIM (Domain Keys Identified Mail) used to limit email spoofing. Recently the DNS has been extended with DOA [21], later renamed OX [22], that essentially is a new record type that allows people to store digital object information in the DNS.



3. Using the DNS as Asset Policy Database

As described in section II.B, being the OX record quite general, it allows us to store into the DNS security policies and use the DNS architecture to distribute them across the network. The advantages are manifold as we can exploit existing DNS servers for distributing our policies, as well use the native DNS security mechanisms named DNSSEC [24] to perform policy updates on a secure fashion. As depicted in figure 3, a central DNS server (authoritative server in the DNS parlance) stores the policies on a local DNS zone, that contains all DNS records delegated to a single manager. In our case, a single zone is usually enough to contain all OX records, even though the DNS provides mechanisms to further partition it. The zone(s) containing OX records is updated by the network discovery component by adding new discovered assets, and by human administrators whenever they want to update/refine existing policies. All DNS zone updates (i.e. add/remove/modify DNS records) are performed using native DNS mechanisms such as the nsupdate tool, part of the popular BIND DNS distribution. As the DNS is a distributed architecture, on large networks it is possible to replicate the central DNS server by deploying several secondary DNS servers at each network edge. Secondary servers are kept in sync with the primary server automatically by means of the native DNS mechanisms that automatically propagate changes from primary DNS server whenever the zone is updated. This way the asset policy database is in essence the DNS server, hence we do not need to use SQL databases such as MySQL or PostgreSQL to store policy information.

The discovery process adds the OX records in the DNS zone using the following format: <MAC ADDRESS>.<device type> and a <MAC ADDRESS> CNAME record, an alias in the DNS terminology, for identifying the device regardless of its type. Example IoT device whose MAC address is 8D: 30:62:56:00:1C will be added in the DNS zone as 8D306256001C.iot. When the discovery process is unable to identify the device type, the “unknown” suffix will be used. The default security policy specified for each device type is named <device type> without the MAC address. This naming schema allows devices to be grouped according to their type and it allows human administrators to further refine the device policies by adding new OX records. For instance, suppose that we want to split printers devices in two categories: one that is Internet accessible, and another one that is not. Administrators could define a category printers_no_internet_access with a policy of no internet access, and then update the zone by moving discovered printers that are not supposed to access the internet under the new node on the same DNS zone (e.g. AABBCCDDEEFF.printers_no_internet_access).

A. Network Device Discovery

The network discovery process actively discovers network devices and adds new devices into the DNS-based database [16, 17]. This activity is implemented by both sending probe packets to discover active devices, and dissecting traffic that traverses the *Policer*. Active probing is used to identify devices even if silent (e.g. a printer does not always need to communicate with the Internet), while passive traffic dissection can complement discovered assets with additional metadata (e.g. by decoding HTTP traffic it is possible to use the User Agent field to learn more about the device operating system). We can envisage deploying in the network both a centralised security policer responsible for inspecting traffic to/from the Internet, and several host-based policers limited to enforce traffic from/to workstations. This to complement centralised with on-device traffic enforcement. Simple devices such as a surveillance camera or a smart TV do not usually access the device policy database to enforce traffic, whereas a more advanced device such as a server or a workstation can access the database to validate network connections they receive. A side effect of this practice is the implementation of a fine-grained, distributed, software-based micro-segmentation by relying on the policy database.

B. DNS-Base Network Policy Database Specification

As previously described, the asset policy database is implemented on top of the DNS, hence the stored information must be specified in a format that is compatible with the domain name server specification. This is not an architectural design limitation but rather a nice feature to have, because it is possible to rely on existing software libraries as well as to leverage on a protocol that operating systems can natively handle, and thus without having to define a new custom protocol. As this work should be used both in small (where a MAC address is meaningful) as well as large networks with routed traffic (where non-local MAC addresses are not visible as they have been replaced with the router's MAC), all device DNS records are uniquely identified with a MAC address and an optional CNAME record (i.e. an alias in the DNS world) to be used to for gluing the device IP address to its MAC address. In this way both local and non-local devices can be identified into the DNS database: both MAC and IP information are stored with no information duplication. Moreover, inside the DNS protocol specification, it is also possible to define multiple different record types for the same key. This means, for example, that for MAC address 4A:00:06:A1:7A:51 it is possible to define a CNAME to IP address 192.168.1.1 (i.e. the IP currently used by the above MAC address) as well as a TXT record where we can store textual information about the MAC. By leveraging on this property of DNS records, it is possible to specify several attributes for the same MAC such as the ingress and egress traffic security policies. The format of DNS records we have used to specify asset information is an extension of the OX record., implemented over DNS as specified in [22]. DNS records contain an object type that might be opaque and private to the producer and the consumer of the data. Each record is identified by the DNS OX Enterprise PEN (Private Enterprise Number) [23] already used in other Internet protocols to identify enterprise-specific datatypes and specified in the OX-ENTERPRISE field. With this mechanism it is possible to use both OX-specified types (in the value range 0..100 and currently limited to asset contact information such as email, website, telephone and public key) as well as define additional ones (in range 101…99999) that will be used for the *Policy Database*. The OX-LOCATION field specifics the information scope (e.g. local), as well as the media type is usually set to base-64 encoding used to encode the data field. The following table specifies the additional values of the OX-TYPE field we have defined for the *Asset Database*.

1. POLICY DATABASE: OX TYPES

Id	**Name**	**Data Source (D=Discovery, H=Human)**	
101	Device Operating System	DHCP, HTTP User Agent	D/H
102	Device Type	SSDP, SNMP, MDNS	D/H
103	Device Name	DNS, MDNS, NetBIOS	D
104	Device Description	SSDP	D
105	Services URL	SSDP	D
106	Hardware Manufacturer	MAC Address (OUI)	D
107	Last Known IPv4 Address	DHCP, ARP	D
108	Last Known IPv6 Address	DHCPv6, Traffic Analysis	D
109	Provided Services (Server)	MUD	D/H
110	Device User	802.1X, Radius	D/H
111	Permitted Ports (Server)	MDNS/DNS-SD, MUD	D/H
112	Permitted Services (Client)		H
113	Permitted Ports (Client)		H
114	Permitted MAC (Client)		H
115	Permitted MAC (Server)		H

Table 1 contains the minimum information necessary for characterising a device in terms of name, provided/used services and capabilities. With the term service we identify what is often called layer seven (or application) protocol and thus we envisage the use of DPI (Deep Packet Inspection) techniques. In this way we can identify protocols also on non-standard ports (e.g. HTTP traffic running on port 1234), as well as proprietary protocols such as Skype in a simple and concise format, thing that would not be possible using IP addresses and ports. The first column contains the OX type that we have defined, and that we will try to push into a new revision of [22] that is still mostly unspecified and open to extensions. The last table column contains a non-exhaustive list of data sources from which the OX fields could be populated, both during the device discovery phase as well at runtime listening to periodic multi/broad-cast messages that the devices emit. Fields marked with D are populated uniquely by the discovery process, while H means are set by human network administrators and D/H by both of them. In particular, active/passive network discovery relies on the following protocols:

- DHCP/DHCPv6 (Dynamic Host Configuration Protocol)
  Used for retrieving the initial host configuration, these protocols can also be used to learn more about the device requesting the configuration. In particular DHCP field id 55 contains the ordered parameters list requested via DHCP (e.g. DNS server, gateway, NetBIOS [30] name) by clients. DHCP fingerprinting [31] is a technique that exploits this id to identify the operating system (and in some cases even more about the device model) of the client that will be used to populate OX id 101.

- SDP (Session Description Protocol)
  This protocol [29] was initially created for announcing multimedia capabilities on the network. It is currently used on a broader fashion by many network devices to announce theirs services. Most modern network devices including smart TV, Internet routers and mobile devices use this protocol to gather the list of available network services.

- SNMP (Simple Network Management Protocol)
  It is a protocol used for managing network devices. It can be used to learn more about the device type by using the MIB-II system and interfaces groups. Furthermore, in authenticated networks, it could also be used to retrieve the device user by inspecting the 802.1X MIBs.

- MAC (Media Access Control) Address
  The MAC address is important for many simple IoT devices usually communicate only with devices of the same manufacturer, or with other devices often identified with a MAC address rather than an IP. For this reason, OX fields 114 and 115, if present, contain the list of MAC addresses (note that they can be partially specified as they can contain just the initial MAC bytes) that these devices can connect, or be contacted.

- ARP (Address Resolution Protocol)
  Designed for discovering Ethernet addresses associated to IPv4 addresses, it can be used to identify network devices and detect if they are connected to the network.

- MUD (Manufacturer Usage Description)
  The MUD [33] is in essence a new DHCP field id that contains a URL to a MUD description file provided by the device manufacturer. Such file specifies the “intended device usage” that is basically what a device is about, including what are the protocols and ports used by the device for providing the expected services. The MUD extension is still being standardised and thus devices on the market are not supporting it yet, even though its adoption will be probably very quick because of cybersecurity issues caused by networked devices that could be limited by knowing the intended use of devices.

We do not expect all devices to specify all the OX items listed in Table 1, as some information might be missing for specified devices (e.g. the device operating system might be unknown for many proprietary devices). However, the goal of this work is to define a comprehensive specification for networked devices that is accessible through the DNS, and that can be used as single source of information for various purposes including network security and monitoring.

C. Network Policy Database: Security Guidelines

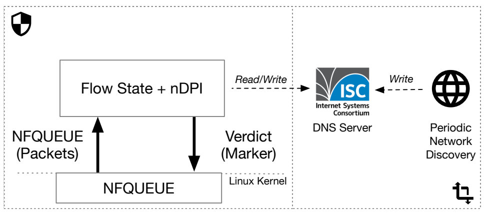
In order to avoid disclosing information about network assets, DNS OX records queries should be disabled for clients sitting outside of the protected network. The DNS system allows a bulk record record transfer named zone transfer (AXFR): this operation should be enabled only for those hosts running the traffic policer and disabled for any other host. The aim is to avoid that a compromised internal host creates a DNS database mirror. In all cases, hosts allowed to perform queries need to be restricted only to selected OX IDs that could be used to enforce traffic such as OX ids' 111-114. DNS information should be modified in a secure way and only by authorised clients, feature that is a standard in all modern DNS implementations. Moreover, the use of DNSSEC is recommended (but not mandatory) in our architecture, as the DNS is used to drop unwanted communications, and thus it is compulsory to make sure that the information on which decisions are made is reliable and untampered. Finally, as DNS records are natively cached using a TTL (Time To Live) value specified in the DNS zone, it is important to use a relatively low TTL (i.e. no longer than 60 seconds) to make sure that record changes are immediately effective, while still using caching to avoid querying the DNS too often. This is usually not a problem in terms of decision latency, as explained later in this paper. Nevertheless, caching speeds up operations as it prevents the *Policer* to access the database too often by saving CPU cycles and thus reducing load.

III. RELATED WORK

The proposed work is a novelty with respect to what we can find in literature [10] when considering all aspects such as distributed security for both conventional and IoT devices, network discovery as seed of security information, and the use of the DNS as distributed and secure policy database. Platforms such as *Microsoft Azure* IoT and *Amazon AWS IoT* are tailored for business processes, as they allow IoT devices to communicate in a secure and distributed fashion. Other solutions such as Siemens *MindSphere* are designed for proprietary IoT devices and thus limited in scope. [35] Uses the DNS for IoT limited to device tracking with no security support. In [36] DOA/OX record is used to keep IoT devices firmware updated using a decentralised architecture. [37] Seems to be similar to this work in many aspects, but it seems to use discovery to locate assets to be later scanned for vulnerabilities. The concept of distributed security policer is well known in literature, however all works [37, 13, 3] are focusing only on configuration automation of firewall rules and DDoS (Distributed Denial of Service) mitigation systems.

IV. IMPLEMENTATION AND VALIDATION

We have validated the proposed architecture (see figure 4) leveraging on open source software so that we could both spread this work, and ease our task of integrating it on working Internet drafts such as those defining OX RR. The developed software has been deployed on various network families ranging from a home/small business network hosting IoT devices, to a regional ISP (Internet Service Provider). This has been important not just to better understand the architecture scalability and flexibility, but also for evaluating network discovery on heterogeneous network infrastructures. While the discovery component has been ported to various operating systems including MacOS, Linux and Windows, the policer is operating system dependent and thus we have developed it on Linux that allowed us to run it both on x86 workstations but also on embedded MIPS/ARM devices as earlier discussed. The DNS server we have used in our experiments is the open source ISC BIND 9.12.1 that natively supports DOA/OX. The network discovery component is based on *ntopng* [15], a homegrown open source network monitoring application, whereas the policer has been implemented on top of the open source deep packet inspection library named *nDPI* [5], we have also developed, and the Linux firewall framework named netfilter.



4. Validation: Traffic Policer and Network Discovery

The network discovery component is activated at startup and periodically (e.g. every 15 minutes) restarted to discover new network assets or poll existing one that have not made any network traffic yet (i.e. silent devices). This list includes for instance thermometers and smoke detectors that do not constantly transmit traffic, and thus that need to be probed in order to detect their presence. Network discovery works in a few consecutive steps:

- From the local network interface, it is possible to read the configured network/mask. The first action performed is to send an ARP request to all local hosts with the except of the network and broadcast address, as well the local host address. Packets are forged and sent on the local interface using the *libpcap* library part of every Linux distribution.
- A Multicast DNS service discovery "`M-SEARCH * HTTP/1.1`" is sent to 239.255.255.250:1900. Devices running a MDNS daemon will report the list of services they advertise. As not all hosts will reply to this service discovery, unresponsive hosts are also queried individually in unicast. Note that this step is not redundant as it allows us to discover even those devices connected to the network and running a MDNS daemon that does not have a local IP address.
- To all hosts that have sent back an ARP reply, and thus that are alive, the discovery component sends further probe packets:
    - A Multicast DNS DNS-SD PTR record query hat will return a list of service types being advertised on the local network. Note that modern operating system versions reply to these queries whereas old versions do not. For those hosts that reply back, we gather the list of services they advertise (e.g. file sharing or remote access) that is useful to characterise the hosts.
    - A SNMP query for the MIB-II system group and based on the response, if any, further requests to further characterise the device model.
    - A Multicast DNS request to query the symbolic host name for the IPv4 address that sent back the ARP reply. The name to characterise the device sub model: "Galaxy S7" or "Luca's iPad" are just some examples.
- Once all the responses have been collected, it is possible to associate a device type and a name to all local hosts that provided enough information. In order to do that, the MAC OUI is also used to enhance the discovery whenever there is too little information to make a decision about an asset. For instance, some manufacturers produce only a few similar products so once we identify a device manufactured by a printer manufacturer, such device is marked as a printer.

At the end of each discovery session, results are stored in the local DNS zone in OX format according to the device type. The zone is updated with all the discovery results using the DNS maintenance utility `nsupdate` that allows a DNS zone to be updated. Once the zone is updated, the primary DNS server informs the secondary DNS servers that will then perform a zone transfer to update their records. As the discovery component is not responsible for setting specific device policies, newly discovered devices will use the default device type policy unless human network administrators will define a specific policy by updating the corresponding OX record. The policer accesses network traffic through the NFQUEUE mechanism that is essentially a packet queue between the kernel and user space. Using the `iptables` Linux configuration tool, it is possible to instruct the system to forward to a specified NFQUEUE queue those packets traversing the device that have no marker specified. In fact, netfilter allows packets to be marked with a numeric identifier that can be used by the networking stack to drop, route or shape traffic. Using the Linux CONNTRACK netfilter mechanism, it is possible to specify a marker for a connection so that once a packet has been marked with a non-zero identifier, the kernel will honour the marker also for all the future packets belonging to the same connection. This means that the traffic policer does not have to process all connection packets, but only the first few packets (i.e. UDP can be limited to one packet and for TCP is no more than 8 packets) in order to make a verdict on the connection. For instance, allowed connections are marked with marker 1, those to be dropped with 2, and allowed ones with reduced bandwidth are marked with 3. Once a connection has been marked, via iptables it is possible to tell the Linux kernel not to send the Policer future packets belonging to the same connection as they have already a marker that the kernel will honour accordingly. In a nutshell the traffic policer:

- Uses *nDPI* to detect the connection application protocol. Note that in *netfilter* a connection is not limited to TCP but also to other protocols such as UDP.
- Utilises a hash table to keep the state of connections for which *nDPI* has not made a verdict yet (e.g. too few packets have been received for the connection).
- Once *nDPI* has detected the application protocol, it decides what marker associate to the processed packet, this using the security policy specified in OX. As the policer

has full payload visibility, it can passively extract from traffic further information that can be used to improve device characterisation such as the HTTP user agent that can disclose the device operating system and model. In case *nDPI* is unable to detect the application protocol, a default protocol named Unknown is used to this connection. At this point, the policer can free the memory for the hash bucket associated to the connection, as it will not receive further packets for the same connection via NFQUEUE.

By configuring netfilter to send the traffic policer only the initial connection packets using the CONNTRACK mechanism, it is possible to process most packets inside the kernel and thus avoid costly kernel-to-userspace communications. The advantage of this implementation design is that the policer makes decisions in user space while traffic is processed inside the kernel using the standard netfilter mechanisms without hacks or custom modules. This clean design allowed us to port this tool across computer architectures, and to both embed it on low-cost devices as well run the same code on powerful x86 servers. In order to avoid the traffic policer to talk with the DNS server too often, we have decided to read the whole DNS configuration at startup and periodically refresh it in case of changes detected using the zone serial number. This has the advantage not to add any latency in packet processing due to DNS access.

The discovery process has been tested in business networks where most devices are Windows/Linux/MacOS systems and other devices are pretty standard such as printers, routers, and access points. In addition to that, we have tested it on a few home networks featuring smart devices such as networked audio and video, IoT devices (i.e. personal health devices, smart lights, thermometers, alarm system) and device hubs (i.e. Logitech Harmony and Amazon Alexa). These environments are the most challenging for discovery: these devices are using proprietary protocols, do not answer to probe queries, and operate through cloud services for remote connecting to the controller application. In order to test the discovery accuracy, we have compared our results with state of the art discovery applications such as [18] and found our results pretty accurate. Testing it on about several networks ranging from 10 to 300 heterogeneous devices, about 12-15% of the discovered devices are not categorised. When devices are divided in sub-categories (e.g. classify a server according to its operating system), active discovery is able to identify correctly only hosts that support one advertising protocol such as MDNS or SDP. This means that over 80% of devices such as printers and NASs, as well Apple OSX/iOS computers are properly identified, whereas most Windows and Unix workstations cannot be further identified unless they run a service such as HTTP or SSH. The *nDPI* protocol detection accuracy is already covered in [20], as well, the performance on small embedded devices is satisfactory for networks with up to 300 Mbit uplinks, whereas for 1 Gbit line rate processing a dual-core x86 server is necessary. We have tested how the developed solution compares in terms of latency with respect to stock Linux kernel bridging on a low-end PC Engines APU2 computer. As reported in the Table 2, in average, the latency added by the policer is about 150 µsec, but in some conditions the latency is increased of an order of magnitude. This is an expected behaviour at the beginning of a connection when nDPI needs to be involved, even though limited only to the first few connection packets.

2. LATENCY MEASUREMENTS

	Policer Bridging	Linux Bridging
Max	1945 usec	193 usec
Min	202 usec	60 usec
Avg	283 usec	131 usec

The ability to use a central policer combined with a per-host policer on selected hosts (e.g. on Linux workstations), contributed to block unwanted traffic at the edge, and shown to be much more effective than the standard Linux firewall that has not layer 7 visibility. In our tests, we have not been able to fully evaluate the effectiveness of our implementation with proprietary devices. This is due to the nature of these devices that do not speak open protocols nor answer queries coming from devices whose network manufacturer is different from them. Handling mobile devices such as tablets and smartphones is working pretty well mostly because there are only two predominant mobile operating systems that makes our life easy.

V. FUTURE WORK ITEMS

The current policer implementation filters traffic based on layer 7 protocols, but it does not yet honours some OX records such as the list of permitted MAC addresses or provided services. Future policer implementations should support the whole set of OX records.

The general perception within the Internet community, is that today's users expect security and privacy even when deploying cheap and simple devices. Since many IoT devices are deployed in unprotected environments such as corridors and walls, these devices should have a way to protect both the device itself and the data they store. Following the principle that devices should be capable of protecting themselves, MUD [33] has been proposed with the goal to provide devices to inform the network what sort of access and network functionality they require/provide. As already mentioned, we plan to propose the adoption of MUD inside the DOA record specification, in order to glue these two draft specifications with the aim of enhancing the overall network and Internet security. While the discovery process we proposed is responsible for locating assets and generating default security policies validated and enhanced by human administrators, we have not taken into account intra-device communications. We plan to implement security policies based on the principles of opportunistic networking [32] that exploits the human social characteristics in order to perform the message routing and data sharing. For instance, if two hosts A and B communicate often over protocol X, and host C also communicates with B over X, then communications between A and C might be acceptable, while a communication pattern not previously observed might be suspicious. Currently we are trying to refine this idea, and afterwards to see how it could fit within the scope of this work.

Finally, we also plan to promote this work in the OX IETF community in order to extend the current specification draft as described in this paper. The IETF has recently proposed the use of the domain name 'home.arpa.' [25, 38] for naming within residential networks. According to this recommendation, names ending with this extension reference a locally served zone, whose contents are unique only to a particular home network. Such names refer to nodes and/or services such as printers, toaster, etc. Our work could be used as foundation for keeping home networks safe and secure.

VI. Final Remarks

As predicted by many analysts, the network edge will become increasingly important, with the IoT just being one of the major driving force for this change. For this reason, it is necessary to enforce traffic policies as close as possible to network devices, unlike most security architectures do today. In order to democratise security, we need all devices to be able to protect themselves and thus it is compulsory to design a distributed and open architecture for enforcing security at the edge. This paper has presented a DNS-based architecture that addresses these challenges using standard and open protocols with no proprietary or custom solutions. This combined with, the use of open source software we used to implement and validate it, can help to widespread the idea and promote it within standard bodies. Finally, this has been the first attempt to extend OX outside of its original scope, and use the DNS not just for name resolution but as a proven, distributed, and scalable solution for network security.

Acknowledgment

The authors would like to thank Andrea Passarella for countless discussions about opportunistic networking within the scope of this work, Daniele Corciulo for helping us to improve network discovery with IoT devices, Elia Porciani and Maurizio Martinelli for evaluating early ideas of this work, Antonio Palumbo and Simone Mainardi for having helped us to improve network discovery and enhanced OX integration in ntopng.

References

1. J. H. Saltzer, “Protection and the control of information sharing in multics”. Communications of the ACM. 17 (7): 389, 1974. ISSN 0001-0782.
2. E. Borgia, “The Internet of Things vision: Key features, applications and open issues”, Computer Communications, Vol. 54, December 2014.
3. Y. Chen, K. Hwang and W. S. Ku., "Collaborative Detection of DDoS Attacks over Multiple Network Domains”, in IEEE Transactions on Parallel and Distributed Systems, vol. 18, no. 12, Dec. 2007.
4. K. York, “Dyn Statement on 10/21/2016 DDoS Attack”, https://dyn.com/blog/dyn-statement-on-10212016-ddos-attack/, October 2016.
5. L. Deri, M. Martinelli, T. Bujlow and A. Cardigliano, "nDPI: Open-source high-speed deep packet inspection," Proceedings of 2014 IWCMC Conference, 2014.
6. Ericsson, “IoT security - protecting the networked society", White Paper, 2017.
7. J. Kindervag, “Building Security into Your Networks DNA: The Zero Trust Network Architecture,” Forrester Research, Tech. Rep., 2010.
8. R. Sandhu, “Good-enough security,” IEEE Internet Computing, vol. 7, no. 1, pp. 66–68, Jan 2003.
9. B. Peterson, “Secure Network Design: Micro Segmentation”, Sans Institute, February 2016.
10. I. Bouij-Pasquier, A.A. El Kalam, A.A. Ouahman, M. De Montfort, “A Security Framework for Internet of Things”, In Lecture Notes in Computer Science, vol 9476, Springer, 2015.
11. VMware, “Data Center Micro-Segmentation: A Software Defined Data Center Approach for a Zero Trust Security Strategy,” Tech. Rep., 2014.
12. P. Mockapetris, “Domain Names - Implementation and Specification”, RFC 1035, November 1987.
13. T. Markham and C. Payne, "Security at the network edge: a distributed firewall architecture”, Proceedings of DISCEX '01, 2001.
14. L. Deri, A. Del Soldato, “Enforcing Security in IoT and Home Networks”, Proceedings of ITASEC 18 Conference, February 2018.
15. L. Deri, M. Martinelli, and A. Cardigliano, “Realtime high-speed network traffic monitoring using ntopng”, In Proceedings of LISA ’14, USENIX Association, 2014.
16. G. G. Richard, "Service advertisement and discovery: enabling universal device cooperation," in IEEE Internet Computing, vol. 4, no. 5, 2000.
17. Hwa-Chun Lin, Shou-Chuan Lai and Ping-Wen Chen, "An algorithm for automatic topology discovery of IP networks," Proc. of ICC 98, 1998.
18. ComputerWorld, “Fing is a great app to see who is on your network”, https://www.computerworld.com/article/2472460/networking/fing-is-a-great-app-to-see-who-is-on-your-network.html, 2017.
19. J E. Al-Shaer, "Managing firewall and network-edge security policies”, Proceedings of 2004 IEEE/IFIP Network Operations and Management Symposium, 2004.
20. L. Deri, M. Martinelli, T. Bujlow and A. Cardigliano, "nDPI: Open-source high-speed deep packet inspection," Proceedings of 2014 IWCMC Conference, 2014.
21. Internet Society, “Overview of the Digital Object Architecture (DOA)”, https://www.internetsociety.org/resources/doc/2016/overview-of-the-digital-object-architecture-doa/, October 2016.
22. A. Durand, R. Bellis, DNS Object Exchange, Internet Draft, https://tools.ietf.org/id/draft-durand-object-exchange-00.html, December 2017.
23. IANA, "SMI Network Management Private Enterprise Codes Registry", https://www.iana.org/assignments/enterprise-numbers/enterprise-numbers.
24. R. Arends and others, “Protocol Modifications for the DNS Security Extensions”, RFC 4035, March 2005..
25. P. Pfister, T. Lemon, “Special Use Domain 'home.arpa.”, draft-ietf-homenet-dot-14, September 2017.
26. IETF WG, “Zero Configuration Networking”, http://www.zeroconf.org.
27. S. Cheshire and M. Krochmal, “Multicast DNS”. RFC 6762, Feb. 2013.
28. S. Cheshire, M. Krochmal, “DNS-Based Service Discovery”. RFC 6763, Feb. 2013.
29. B. Peterson, “Secure Network Design: Micro Segmentation”, Sans Institute, February 2016.
30. IETF, “Protocol standard for a NetBios Service on a TCT/UDP Transport: Concepts and Methods/Detailed Specifications”, RFC 1001/1002, March 1987.
31. T. Matsunaka, A. Yamada and A. Kubota, "Passive OS Fingerprinting by DNS Traffic Analysis," Proceedings for 2013 IEEE 27th AINA Conference, Barcelona, 2013.
32. M. Conti, S. Giordano, M. May, A. Passarella, “From opportunistic networks to opportunistic computing”, IEEE Communications Magazine, Vol. 48, Issue 9, 2010.
33. E. Lear, R. Drops, D. Romascanu, Manufacturer Usage Description Specification, draft-ietf-opsawg-mud-20, April 2018.
34. D. Newman, “Top 10 Trends For Digital Transformation In 2018”, Forbes Magazine, September 2017.
35. B. Karakostas, “A DNS Architecture for the Internet of Things: A Case Study in Transport Logistics", In Procedia Computer Science, Volume 19, Pages 594-601, 2013.
36. P. Brisson, D. Vilches, F. López, F. Torre, E. Crudele, M. Banchoff, M. Ferrigno, A. Barbieri, “DOA over DNS Prototype”, ICANN 60, November 2017.
37. Pwnie Express, “Welcome to the IoT Security Gap”, https://www.pwnieexpress.com/resources/iot-security-gap-download, 2017.
38. J. Latour, ”Home Network Registry Idea”, Tech Day, ICANN 60, October 30, 2017.