Enforcing Security in IoT and Home Networks

Luca Deri[1], Arianna Del Soldato[2]
IIT/CNR[1,2], ntop[1]
{luca.deri, arianna.delsoldato}@iit.cnr.it, deri@ntop.org

Abstract

Modern home and corporate networks are interconnecting many different devices types other than personal computers and printers. It is pretty common to have surveillance cameras or thermometers and control them through cloud-based services. Security-wise this practice can create potential threats when connected devices are not kept updated or if they can freely access the network.

This paper describes a novel approach to monitoring and enforcing network policies that takes advantage of techniques such as network discovery and device behaviour fingerprinting, to define per-device/user network policies and enforcing them at the network edge before unwanted traffic enters or leaves the monitored network perimeter.

1. Introduction and Motivation

The principle of least privilege as formulated by P. Denning and J. Saltzer*, has been used as cornerstone of information security for a long time. 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. In order to implement this principle, physical, logical and procedural security has to be enforced. Due to this, networks have often been divided 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's staff and thus that it is supposed to be trusted as internal users can access the Internet usually by means of proxy services that can analyse data being exchanged. Computers that must be accessible from the Internet are placed in the DMZ and enforced by firewalls and other security devices that restrict the access only to the required services.

With the advent of the *IoT* (Internet of Things), *BYOD* (Bring Your Own Device), remote surveillance/assistance applications, cloud-based connected devices such as fridges and thermometers, the above security model cannot be used anymore. This is because devices operating in the internal network cannot longer be trusted although they are installed on a trusted zone. In corporate networks, a typical countermeasure to this problem is the implementation of micro-segmentation that is the act of splitting a computer network into multiple small network segments where data exchanged across segments it is not simply routed but instead inspected by security devices. While micro-segmentation does not fully solve the problem, albeit it reduces security risks. Unfortunately, it cannot be easily used in small or home networks due to inability of network equipment to implement it, or lack (of skills) of network administrators to permanently supervise the network. 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

* Every program and every privileged user of the system should operate using the least amount of privilege necessary to complete the job.

many popular applications (e.g. *Skype* or *BitTorrent*) do not use *SSL* but proprietary encryption algorithms. This problem is even worse when low-cost cloud-connected devices (e.g. surveillance cameras) are used: often they often do not use encryption while periodically communicating with the cloud in order to allow mobile applications to remotely connect to the camera. Furthermore, many low-cost devices cannot be updated when a security flaw is detected, as manufactures do not issue periodic security fixes opening up the trusted network to vulnerabilities and remote access to the internal network from the Internet. Firewalls are unable to effectively implement application-level security as they often limit their scope to packet headers, and also because they are only configurable devices, hence pretty static. This contrary to most small/home-networks that are dynamic in terms of Internet addresses thus making impossible to restrict devices to specific network protocols when their address dynamically changes via *DHCP*. Another relevant aspect to consider, is that in small or IoT networks there is not a permanent network administrator that continuously supervises network activities, thus leaving to network users the burden of preserving network security as new devices and services are deployed.

These facts have been the motivation for the work described in this paper. Namely the need to create a simple security application, small enough in terms of computing resources, yet able to increase network protection of IoT/home networks by restricting Internet access based on the device type, and able to detect and insulate network threats caused by malware or compromised devices running on the internal network. The novel contribution of this paper is the idea that we can use dynamic network discovery not just to better map network devices by labelling them with a type/ category, but applying to each device a comprehensive network profile based on its type. This allows network security to be dynamically applied to networked devices regardless of them being configured with a fixed IP address. The architecture described in this paper has been validated on Linux and uses open-source software to make it suitable to be implemented in a plethora of different devices and thus serve as a tool for increasing security in Internet communications.

2. Architecture Design

A network policer device (Figure 1) is a low-cost device designed to be interconnected between the network to be protected and the Internet router that on most small networks often acts as firewall. Ideally the device should be a bump-in-the-wire and thus be transparent to users that should not require to modify their Internet address plan, even though it could also act as a *NAT*-router that masks the internal network addresses to the gateway. Being it placed close to the network egress, it can observe all the network traffic and thus effectively apply the network policies to Internet traffic, even though in this location it cannot enforce any policy on the internal network when two local devices communicate. In order to implement the latter, it must deployed as a security router in a micro-segmented network.

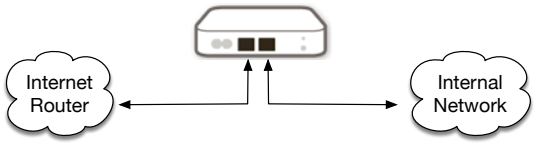

Figure 1: Network Traffic Policer Device

In firewalls and IDSs/IPSs, security policies are not differentiated based on the device type/user or operating system and thus they are coarse grained by nature. In essence it is not taken into account at all what type of device is generating or receiving traffic. This practice has many disadvantages:

• As we are not aware of the device type and operating system, it is not possible to implement specific checks, nor stop unexpected traffic. For instance, if device X is an iPad, whenever we

observe traffic that cannot have been generated by the device (e.g. a software download from the Google Play Android application store), it is required to trigger an alert and stop the suspicious activity. However, if you look at this traffic from a larger perspective, the traffic looks legitimate and security devices won't complain about it.

- As most security devices such as IPSs rely on filtering rules for detecting threats, the more rules are loaded into the system, the slower it gets. Differentiating rules based on the device type would allow analysis to be speed up by reducing the number of rules applied per device, and also complement them with more specialised per-device rules.

In order to implement per-device network policy that also overcomes firewall limitation, it is compulsory to implement a reliable network discovery technique able to identify devices with a high degree of confidence. Similar to tools like *fing* or *nmap* that actively scan the network for detecting devices along with their main features (e.g. operating system or device type), it is necessary to setup a network discovery service both active and passive. The active network discovery is a periodic task that performs local subnet scan by polling devices by means of various network protocols including *NetBIOS* and *SNMP*. The passive service constantly monitors broadcast/multicast-based protocols used to advertise services including *MDNS* (Multicast DNS)/*DNS-SD* (DNS Service Discovery) used for implementing DNS-based service discovery, *SDP* (Session Description Protocol) used by many network devices to announce theirs services, and *ARP* (Address Resolution Protocol) that can be used to identify active yet silent network devices. Based on device information reported by the network discovery process, the application enforces the network policies by relying on the underline firewall system similar to what the MUD Internet draft proposes. Devices are divided in categories each with a set of protocol capabilities and allowed up/download throughput. For instance, core protocols such as DNS traffic are monitored, and the protocol traffic is passing through a traffic policer that prevents devices to send/receive too many requests and thus create issues such as those created the during the Dyn cyberattack. Each network device family (e.g. printer, mobile device, computer, TV) is bound to a network traffic profile that specifies what application protocols can be used, both as client and server. As most modern protocol use dynamic ports, in order to identify the real application protocol, deep packet inspection (DPI) techniques are compulsory. Contrary to the common believe that DPI is a computationally intensive technology, the lessons learnt while developing and maintaining *nDPI*, an open-source DPI framework, allowed us to successfully run it on devices with limited computing resources.

3. Implementation and Validation

The architecture above described has been implemented using Linux on two types of low cost devices: a Raspberry *PI3* and a PC Engines *APU2C0* with hardware prices ranging from 40 to 140 Euro. Protocol detection has been implemented using nDPI, an open-source deep packet inspection developed by the author and able to detect over 200 protocols.

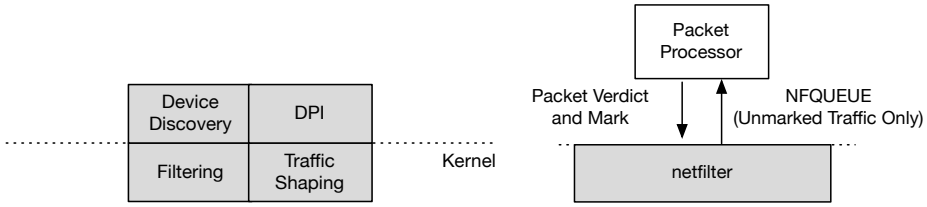

Figure 2: Network Traffic Policer Device

The application is implemented using Linux on top of the *netfilter* firewall component leveraging on the lessons learnt while developing an open source traffic monitoring application named *ntopng*.

The discovery component, implemented inside the packet processor application, listens on the network for service announcements and performs periodic network scans. Packet filtering and enforcement happens in the kernel. In netfilter it is possible to bundle hosts (both using IP and MAC addresses) in so-called *ipset*'s. Devices are divided in categories based on their type (e.g. printers). As the discovery component classifies new devices, it places them on the corresponding ipset that groups them based on their category. Each ipset has associated a list of application protocols that are allowed/ forbidden for the specified category. It is also possible to define further ipset's that identify the list of IP/MAC addresses that are allowed to contact specific device types. For instance, through this mechanism it is possible to restrict access to video surveillance cameras from specific hosts, or prevent non local hosts to access the NAS server. The user space component initialises the DPI engine, and listens for packets sent to user-space through a netfilter *NFQUEUE* socket previously configured. Only the initial connection packets are sent to user space as once the application protocol has been detected, the connection is marked through the *CONNTRACK* marker and a verdict (pass/ drop/shape) is bound to the connection. Thanks to this, the netfilter component bridges/routes future connection packets in kernel without sending them to user-space as the connection verdict is kept for the connection lifetime. This allows very few packets to be processed in user space and thus obtain packet processing performance close to the same box without packet inspection in place. Packet shaping is implemented in the Linux kernel and configured from user space. Shaping is performed per application protocol so that it is possible for instance to drop Facebook but allow BitTorrent at 1 Mbit/ s. In order to differentiate among protocols and shapers, the packet processor application marks connections according to a configuration file where the list of allowed protocols with their associated marker are specified per ipset/queue as depicted in Figure 3.

```
#nf:X       Application Protocols with associated marker
0           Facebook=4,Apple=2,BitTorrent=2
1           SMTP=3,SIP=1

# Read CONNMARK and set it in mark
iptables -A PREROUTING -t mangle -j CONNMARK --restore-mark
# Set default actions for markers
iptables -A PREROUTING -t mangle -m mark --mark 1 -j ACCEPT
iptables -A PREROUTING -t mangle -m mark --mark 2 -j DROP
# Shape traffic for markers 3 and 4
iptables -A REROUTING -t mangle -m mark --mark 3 -j CLASSIFY --set-class 1:3
iptables -A PREROUTING -t mangle -m mark --mark 4 -j CLASSIFY --set-class 1:4
# Define ipset and add a sample iPad device to it
ipset create macdevices hash:mac
ipset create ipdevices iphash
ipset add macdevices 04:69:f2:58:49:21
# Send unmarked traffic to NFQUEUE in case it matches the ipset
iptables -A PREROUTING -t mangle -m mark --mark 0 -m set --match-set macdevices src -
j NFQUEUE --queue-num 0
iptables -A PREROUTING -t mangle -m mark --mark 0 -m set --match-set ipdevices src -j
NFQUEUE --queue-num 1
# Save mark into CONNMARK
iptables -A POSTROUTING -t mangle -j CONNMARK --save-mark
```

Figure 3: Packet Processor Configuration and Traffic Policy Configuration

In the above example two ipset's, each with its associate NFQUEUE, are created to demonstrate that it is possible to identify devices both using their MAC and/or IP. Devices can be added to ipset's both statically (e.g. for devices that are static such as an access point or a surveillance camera) as in the above example, or dynamically based on the result of the periodic device discovery. Once the packet application is configured, runtime modifications are necessary only to add/remove/move devices across ipset's. By registering with the CONNTRACK component, it is also possible to log network events when a connection ends, and thus produce useful information for specialised security application and network forensics investigation.

Validation of this application has been performed on small home/business networks with Internet downlink of up to 200 Mbit/sec and about 50 physical devices being monitored by the device. The device performance depends on the device type. The PI3 when equipped with a second Ethernet port via USB was able to handle ~60 Mbit with the speed mostly limited by the hardware networking part that in this device is not very efficient and limited to 100 Mbit. The APU2C0 instead was able to handle up to ~400 Mbit with a traffic generator and thus on this experiment successfully enforced the traffic with no noticeable slowdown whatsoever. In terms of processing overhead, being DPI limited to the first few connection packets, when comparing the processing speed with/without DPI (and thus when routing/bridging packets in the kernel), the device does not add any perceivable latency and the load on the CPU is basically the same. The traffic classification process is quite reliable and we have not reported any file positive result. Instead, due to the large plethora of devices, the device discovery needs to be improved, in particular when low cost devices are used. This is because they do not fully honour the protocol specification and export thought network protocols very limited device capabilities information that make difficult to reliably identify all the devices. For this reason unidentified devices are places on a specific ipset where limited traffic activities are allowed.

4. Final Remarks and Future Work

This work has demonstrated that the use of low cost devices and open source software can be effectively used to create solutions for complementing security offered by firewalls and networks devices. The validation process has confirmed that implementation is quite efficient as it mostly lives into the Linux kernel, and it can effectively be used on real networks.

Future work activities include improvements to the network discovery service and further refinement of traffic policies as well see if it's possible to embed some lightweight traffic enforcement policies usually offered by IDS/IPS that would be too heavy and complex to both integrate and maintain into a low cost home/small business device.

References

Saltzer, Jerome H. (1974). *Protection and the control of information sharing in multics*. Communications of the ACM. 17 (7): 389. ISSN 0001-0782

Peter J. Denning. (1976). *Fault Tolerant Operating Systems*. ACM Comput. Surv. 8, 4, 359-389.

Nimmy Reichenberg (2014). I*mproving Security via Proper Network Segmentation*. Security Week, Mar. 2014.

J. Michael Butler (2013). *Finding Hidden Threats by Decrypting SSL*, SANS Institute, Nov. 2013.

Lear, E. and Romanascu D., *Manufacturer Usage Description Specification* (MUD), draft-ietf-opsawg-mud-05, March 2017.

Gordon Fyodor Lyon (2009). *Nmap Network Scanning: The Official Nmap Project Guide to Network Discovery and Security Scanning. Insecure*, 2009.

S. Cheshire and M. Krochmal (2013). *Multicast DNS.* RFC 6762, Feb. 2013.

M. Handley, V. Jacobson, C. Perkins (2006). *SDP: Session Description Protocol*, RFC 4566.

S. Cheshire, M. Krochmal (2013). *DNS-Based Service Discovery.* RFC 6763, Feb. 2013.

J. Roberts (2016), *Who to Blame for the Attack on the Internet.* Fortune Magazine, Oct. 13, 2016.

L. Deri, M. Martinelli, A. Cardigliano (2014). *nDPI: Open-Source High-Speed Deep Packet Inspection.* Proc. of TRAC 2014 workshop, Aug. 2014.

L. Deri, M. Martinelli, A. Cardigliano (2014), *Realtime High-Speed Network Traffic Monitoring Using ntopng*, Proceedings of LISA 2014 workshop, November 2014.

D. Napier (2001), *Security: IP Taxables/NetFilter -- Linux's next-generation stateful packet filter*. Sys Admin 10, Dec. 2001.