Chapter 9
HONEYSCOPE: IoT Device Protection with Deceptive Network Views

Reham Mohamed, Terrence O'Connor, Markus Miettinen, William Enck, and Ahmad-Reza Sadeghi

Abstract The emergence of IoT has brought many new device manufacturers to the market providing novel products with network connectivity. Unfortunately, many of these new entrants to the market lack security engineering experience and focus heavily on time-to-market. As a result, many home and office networks contain IoT devices with security flaws and no clear path for security updates, making them attractive targets for attacks, e.g., recent IoT-centric malware such as Mirai. In this chapter, we discuss a network centric approach to protecting vulnerable IoT devices. We describe a system called HoneyScope, which seeks to achieve two goals. First, each IoT device has a different view of its local network, which limits the damage when a device is compromised. Second, virtual IoT devices are created to confuse and deceive attacker with sophisticated motivations (e.g., fake WiFi connected cameras). To achieve these goals, HoneyScope uses an SDN-based security gateway to create virtualized views of the network and nodes therein providing fine-grained control over the communications that individual devices may have.

9.1 Introduction

One of the big challenges facing IoT networks in homes and small offices—in comparison to traditional networks—is their relatively high susceptibility to security threats. Numerous heterogeneous IoT devices are being deployed in small office, home office (SOHO) networks, broadening the potential attack surface for

The original version of this chapter was revised: Chapter authors have been added. The correction to this chapter is available at https://doi.org/10.1007/978-3-030-02110-8_12

R. Mohamed · M. Miettinen · A.-R. Sadeghi
Technische Universität Darmstadt, Darmstadt, Germany

T. O'Connor · W. Enck (✉)
North Carolina State University, Raleigh, NC, USA
e-mail: whenck@ncsu.edu

E. Al-Shaer et al. (eds.), *Autonomous Cyber Deception*,
https://doi.org/10.1007/978-3-030-02110-8_9

adversaries, as many new IoT devices are affected by inherent security vulnerabilities. This is due to hundreds of new IoT manufacturers entering a market that is largely untapped and considered unsaturated, providing players that are first-to-market opportunities to gain considerable market share. As many manufacturers' focus is therefore on quickly shipping their products, this leaves little time and resources to focus on proper security design, implementation, and testing of new device models. Moreover, manufacturers are often producing low-cost devices (such as connected light bulbs or smart plugs) with hardly any or no budget at all for security. This results in many IoT device vendors shipping products that contain security vulnerabilities that are relatively easy to exploit by knowledgeable attackers.

Due to the newness of IoT, there is a lack of regulations and laws governing development and production of IoT devices. There exist also no dominant security standards that all device vendors would adhere to.

In most cases, the responsibility of securing devices is therefore left to end-users. Access to devices is typically controlled using default easy-to-guess credentials, but vendors do not force users to update them during the device on-boarding procedure. Many users will, however, not understand the risks (among others) associated with such default passwords, and many of them will not even care, leaving devices at the mercy of potential attackers. Other devices, on the other hand, are designed to work in a plug and play mode by default, leaving no room for users to modify security settings, even if they would like to.

Another major security threat is given by benign but intrusive functionality of devices that can possibly breach the privacy of users by, e.g., recording private conversations, taking photos, or recording videos and automatically uploading such information to the cloud without the user's consent. Already now smart voice assistants like Amazon Alexa and Google Assistant have been, intentionally and accidentally alike, triggered by viral ads like Burger King's ad that forced Google Assistant to recite the definition of the Whopper from Wikipedia [8], or other incidents like one affecting Amazon's Alexa, which—as explained by Amazon—misheard the wake word during the conversation of a wife with her husband before sending a recording of it to the wife's colleague [1].

9.1.1 Principle of "Need to See"

At the root of our approach to deception in SOHO networks is the principle of *need to see*, which is a variant of the traditional principle of *need to know* used in environments with sensitive information. A key observation is that while future SOHO networks may be filled with tens of IoT devices, most devices do not interact with one another. First, many devices communicate exclusively with the Internet. Second, for devices that communicate within the network, interaction is often with a small set of *controller* devices, such as smartphones and smart speakers (e.g., Alexa, Google Home). Therefore, there is no need for most IoT devices to *see* one another on the network.

The simplest security policy in any setting is that of strict isolation. It is simple to express and enforce. WiFi isolation is built into all commodity routers and access points. In fact, most hotels use WiFi isolation to ensure that guests can access the Internet, but not interact with one another directly. However, WiFi isolation is not suitable for SOHO networks. First, SOHO networks contain traditional devices where isolation impedes functionality, e.g., desktops, laptops, printers, and network attached storage (NAS) devices. Second, some intra-network communication is needed for controller devices to coordinate actions with IoT devices. Third, some IoT devices may be designed to work directly with one another, such as those from the same manufacturer (e.g., WeMo, D-Link) or using the same standardized protocols (e.g., HomeKit). Therefore, the *need to see* in SOHO environments is more complex than strict isolation.

9.1.2 Deception Through Network Views

WiFi isolation is a degenerative type of network view. That is, each device can only see itself and the network gateway. Consider the more general model, where there exists a policy for each network device that defines which other network devices it can see. More formally, let N be the set of devices on the network. Each $n \in N$ has a policy $P_n \in \mathscr{P}(N)$, where $\mathscr{P}(N)$ is the power set of N.[1] The policy P_n defines a specific network view for n.

The network view presented to an adversary, whether it be a compromised or misbehaving device, influences its perspective of the attack surface of the network. There is also no requirement that each $n \in N$ is a physical device. For example, N can include a virtual WiFi baby monitor camera that simply plays a feed on a loop. Such virtual deception devices can be used in several ways. Consider the scenario where one of the users' IoT devices is compromised and is used as an attack pivot, which proceeds to scan for other vulnerable network devices. First, the virtual deception device can act as a honeypot. Under normal scenarios, the virtual deception device should not receive any network connections from real devices. Second, the virtual deception device may make the adversary believe she has control of a real device, e.g., watching a live feed of a baby. Gaining access to the baby camera may be the adversary's goal, and the simulated feed may keep the adversary from burglarizing the home.

Network views provide usability in addition to deception. For example, the network view policy can exclude the virtual deception devices from the view of controller devices (e.g., smartphones). In this way, users will not be confused by the potentially many virtual deception devices.

[1] The power set of S is the set of all subsets of S.

9.1.3 HONEYSCOPE

In the remainder of this chapter, we present HONEYSCOPE, a security framework for small office and home office (SOHO) networks built on top of the concept of network views. HONEYSCOPE is a protection layer built on top of the local network and provides a fine-grained control over the communications of individual IoT devices in the network. HONEYSCOPE uses software defined networking (SDN) technologies [9] to realize *device-* and *device-group*-specific views of the network that *reduce the attack surface* against vulnerable devices in the network, *contain effects of device infections* in case of successful device compromise, and enforce effective measures for blocking unwanted release of contextual data from within the network to the outside. At the same time HONEYSCOPE acts as a deceptive obfuscation layer that decouples the network appearance of devices from their actual physical interfaces, providing the network owner fine-grained control over how devices and the network topology are presented to other devices and to the outside.

9.2 Design of HONEYSCOPE

The core idea on which HONEYSCOPE builds is to provide *device-group-specific* views on the local IoT network in order to be able to control the exposure of devices and contain potential security incidents, without adversely affecting the benign functionality of devices. HONEYSCOPE seeks to realize this through following design principles.

Grouping of Devices According to Their Vendor Due to the lack of proper interoperability standards for IoT devices, many device manufacturers offer vendor-specific interoperability solutions, often supported by vendor-specific cloud-based back-end services. Due to this approach, many IoT devices seldom—if ever—have the need to communicate with devices that do not fall within their vendor-specific device category. HONEYSCOPE uses this property to compartmentalize the local IoT network by placing individual IoT devices into vendor-specific groups. By limiting communication to happen only within the vendor group (and potentially the vendor cloud service), the attack surface of IoT devices inside the group can be effectively reduced and security attacks across vendor groups effectively mitigated.

Grouping of Devices According to Their Functionality The differentiation between device groups can also happen based on properties other than the device's manufacturer. For a number of specific applications like smart lighting there already exist to some degree protocols that enable interoperability between devices of different vendors (e.g., ZigBee Light Link). To support such (future) functionality, HONEYSCOPE supports orthogonally also groupings that are based on the declared functionality of devices, if this functionality requires interoperability across vendor-specific groups. As mentioned, this grouping is orthogonal to the vendor-specific groups, meaning that devices can be in parallel member of a vendor-specific group and one or more functionality-specific groups.

Vulnerable Device Isolation HONEYSCOPE isolates devices that are known to have vulnerabilities in a specific group with a very constrained view on other devices in the network. The vulnerable device group shall not have access to any other devices in the local IoT network, nor shall external devices be able to communicate with devices in this group. The only exception is access to vendor cloud services that are necessary to maintain the benign functionality of the device. This has twofold goals: (1) to protect vulnerable devices from being compromised by malware or active attacks originating from outside the network, and (2) to protect the rest of the IoT network's devices in case an adversary manages to compromise a vulnerable device.

Deceptive Views of the Network HONEYSCOPE also provides the possibility to create various deceptive views of the real, physical network. Each physical device in the IoT network has a virtual representation in the HONEYSCOPE virtualization layer. This allows the network owner to define how devices are perceived by other devices and the outside network. The deceptive views may also include virtual representations of non-existent devices to provide *honey views* of the network that allow to completely obfuscate the true topology of the actual physical network setup.

9.2.1 HONEYSCOPE *Implementation Approaches*

There are multiple options for HONEYSCOPE to implement network views.

Option 1: Multiple (V)LANs One way to realize the group separation of HONEYSCOPE would be to use a number of LANs or VLANs to represent the different groups of the local network. However, this would be incompatible with existing discovery protocols, as devices on separate LANs would be unable to discover each other. For example, mDNS, which is used by HomeKit, assumes that all devices are on the same LAN.

Option 2: Multiple SSIDs A second option is to use separate SSIDs of a WiFi access point. The main drawback of using this approach lies in the increased complexity of the bootstrapping process, as each device would need to be provisioned on the correct SSID. This would in general be a too complex task to be handled correctly by regular users. For example, many IoT devices have a bootstrap process that includes a smartphone app that automatically copies the SSID and WPA2 password from the smartphone. Therefore, the user would need to navigate multiple SSIDs when setting up devices.

Option 3: Use of SDN Technologies for Group Separation Software defined network (SDN) technology such as OpenFlow provides a unified network abstraction in which all devices in the network are controlled by the SDN controller, allowing fine-grained control over network connectivity of individual nodes. SDN provides the most versatility to programmatically implement network views. It also allows the architecture to extend to an arbitrary number of access points, which is increasingly

common as newer WiFi protocols such as 802.11ac provide faster performance when devices are close to the access point.

HONEYSCOPE uses SDN technology to control the topology of the IoT network in order to provide maximal flexibility to manage nodes in the local network and to realize a virtualization layer implementing the security design of HONEYSCOPE and enforce its security policies. In effect, the local WiFi access point acts as a security gateway controlled by a HONEYSCOPE SDN controller. The task of the security gateway is to realize the HONEYSCOPE virtualization layer by enforcing network connectivity policies defining the device group topology and create required virtual representations of nodes belonging to the honey views comprising the deception aspect of the HONEYSCOPE framework.

Erickson et al. [10] have developed a mechanism for identifying vulnerable devices and blocking them from accessing and attacking other devices in the network. Using this mechanism, they eliminate Man-In-the-Middle attacks at the link and service discovery layers. To realize this mechanism, they use a different SSID and WPA password for each device. However, this approach does not work properly with IoT networks as the IoT devices get the network information from their companion app on the mobile device of the user. HONEYSCOPE would overcome this drawback by using one SSID, and one LAN for all devices in the network using the SDN technology.

9.2.2 HONEYSCOPE *Network Structure*

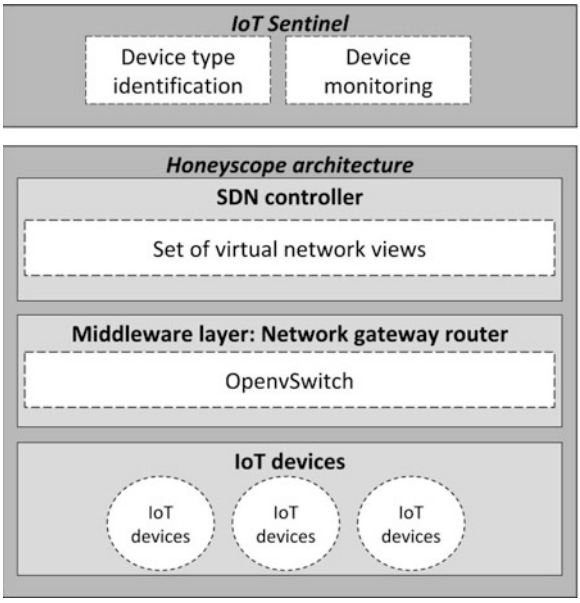

Fig. 9.1 HONEYSCOPE layered architecture

The HONEYSCOPE architecture consists of three main layers as shown in Fig. 9.1. The top level is the HONEYSCOPE controller: an SDN controller that controls and manages the HONEYSCOPE router controlling the local IoT network. This Open vSwitch-based router uses the OpenFlow protocol to communicate with the HONEYSCOPE SDN controller to manage the IoT devices in the local network.

The SDN controller is responsible for creating the network views for each IoT device connected to the local LAN. This is done by grouping devices into vendor- and function-specific groups as discussed above in Sect. 9.2 and applying a specified *network view* on each group in the network. The created virtual network views are unidirectional views. For example, if group x can view and send information to group y, this does not necessarily mean that group y can access and view group x.

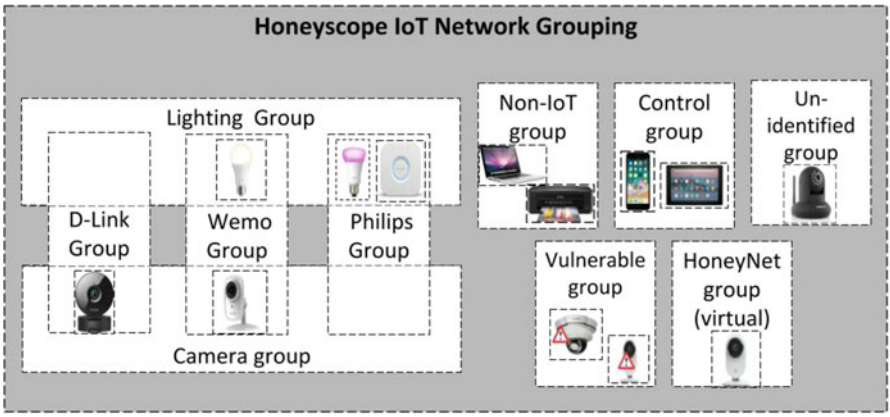

Fig. 9.2 HONEYSCOPE grouped IoT network

9.2.2.1 HONEYSCOPE Groups

There are five default groups that are created by the HONEYSCOPE controller in addition to the device vendor-specific groups and the function-specific groups as shown in Fig. 9.2:

- **Control group:** This group consists of controlling devices such as smartphones or tablets that should be able to control and therefore have access to any of the IoT devices in the network. Devices in this group can view and have access to any device in the IoT network. It is the only group in which devices have a virtual network view that looks like the real, physical network. All devices from any other groups can communicate with any device in this group as well, except for devices in the vulnerable devices group. Also, devices in this group don't see the deceptive virtual devices in the HoneyNet group. Typically, the smartphone of the home owner is the main device mapped to this group.

- **Vulnerable devices:** This group contains any detected vulnerable IoT devices in the local network. Devices in this group cannot view and do not have access to any devices in other groups in the network except the HoneyNet group. They are thus isolated from other IoT devices. The virtual network view for this group only contains a unidirectional connection to it from the control group and a bidirectional communication possibility with the HoneyNet group. Only the control group can send updates, messages, or information to this group.
- **Unidentified devices:** Devices that cannot be identified by the controller, such as devices that have just been released onto the market and are therefore yet unknown to the system providing device identification, are added to this group.
- **HoneyNet group:** This group's main task is to create virtual devices to deceive the intruder, like the virtual baby camera discussed in Sect. 9.1.2 providing a fake instance of a device to confuse the adversary in case it is able to intrude into the network.
- **Vendor-specific groups:** When a device is joining the network, the SDN controller identifies its type based on its communication behavior, and adds it to the corresponding manufacturer group. Devices in each group can communicate with each other but don't have access to devices from other groups they are not members of. An example of a vendor specific could be, e.g., the smart home automation gadgets provided by D-Link. These include smart power plugs, motion sensors, water sensors, IP cameras, door and window sensors, and alarm sirens. All of these devices can be configured to work together using rules defined in a cloud-assisted vendor-specific smartphone app. It makes therefore sense to place all such devices from this vendor to the same group.
- **Function-specific groups:** Devices from different manufacturers may need to be placed in function-specific groups when functional interoperability is needed. For example, devices providing smart lighting (e.g., smart light bulbs, switches) would be placed in a lighting-specific group in order to enable the light switch of one manufacturer to control smart light bulbs of another vendor.
- **Non-IoT devices group:** In the local network there are also other non-IoT devices like desktops, laptops, NAS devices, etc. that should be treated separately from IoT devices, as their functionality is much richer than that of typical IoT devices. Such devices are therefore added to a dedicated group that contains non-IoT and legacy devices. Devices in this group have a flexible virtual network view configurable by the user allowing the devices to view and interact with other IoT or non-IoT devices. The view for each device is defined according to the need of this device to communicate with other devices (e.g. interacting with printers, or specific IoT devices in the network) and can be derived, e.g., based on the set of applications installed on the device. For example, a non-IoT device like a laptop should be able to view other laptops in the network, in addition to printers, VoIP phones, cameras, smartphones, and tablets. Smartphones and tablets are located in the control group, while printers and VoIP phones are located in the non-IoT devices group. The laptop may also need access to IP cameras located in the function-specific "camera" group. This way, the laptop will have a virtual

network view that allows it to have access to the non-IoT group, the control group, and the camera group.

As shown in Fig. 9.2, there are intersection points between vendor-specific and function-specific groups. For example, in this network, there is a Philips Hue smart device in both the lighting group (function-specific) and the Philips group (vendor-specific). The function-specific groups contain devices from different vendors but they share the same function and usage, so they need to communicate together in a separate group.

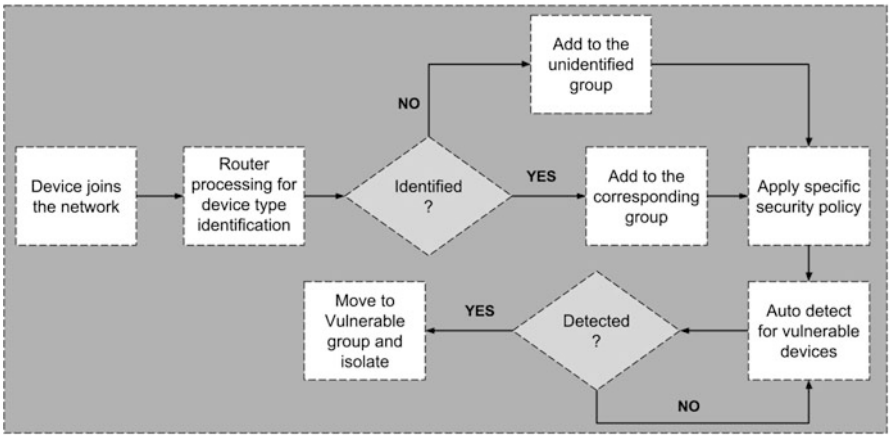

Fig. 9.3 Grouping and isolation processes

9.2.3 Device Type Identification

Most devices joining the network do not explicitly advertise what type of device they are. They need therefore to be identified before assigning them to HONEYSCOPE groups. When a device joins the local IoT network, the SDN controller identifies its type by using the developed IoT sentinel model by Miettinen et al. [12] for unknown device type identification. IoT sentinel is based on identifying the device type by profiling the communication behavior of the device and using machine learning classifiers to identify its device type. It uses a database that contains pre-captured behaviors for most of the current IoT devices in the market, and from this database, the device type is determined and defined. The main idea of IoT sentinel is to create device fingerprints by monitoring the communication behavior of the new added device during the setup phase. From this generated fingerprint, IoT sentinel is able to map the device to its corresponding device type using machine learning-based classifiers.

If the device is identified successfully, it is added to its corresponding group in the local network: a vendor-specific group and possibly one or more function-specific groups.

For example, if the device was detected to be a Philips Hue light, it will be added to the Philips Hue group as well as the lighting group. When the device is added to the corresponding group, the SDN controller creates a virtual network view that matches the groups the device is added to. If IoT sentinel fails to successfully identify the joining device, the device will be added to the unidentified group which contains all unidentified devices in the network. The joining device will be able to view the control group, the HoneyNet group, and any other devices in the unidentified group.

The HONEYSCOPE IoT network will be defined by different customized network views for each device added to it according to this device type identification process.

IoT sentinel will notify the SDN controller if any vulnerabilities become known for any of the IoT devices registered with the SDN controller. For obtaining this information, IoT sentinel utilizes an IoT security service, which aggregates network-wide information about known vulnerabilities associated with particular IoT device types. If a vulnerability is detected that affects a particular device, it will be moved from its group, regardless whatever this group is, to the vulnerable group. This way, the device will be isolated from other devices in the network not to affect any of them until the device vulnerability has been removed, e.g., by applying an appropriate firmware patch. The device will not have any access to any devices in the local network. However, devices in the control group will have a unidirectional connection to it, e.g., in order to check device status or send updates and other messages.

9.3 HONEYSCOPE Components

9.3.1 HONEYSCOPE *Controller*

There are many SDN controllers that can be used with HONEYSCOPE. However, we selected RYU controller [6] which is based on Python, because of its support for the higher and newest versions of the OpenFlow protocol [11], it supports the 1.5 OpenFlow standard. In addition to that, it has the Nicira extensions for OpenFlow matching [5]. Although Nicira is a vendor specific implementation, it is luckily implemented in OpenvSwitch. It provides some additional criteria to match on. We have developed a Ryu application to implement the management and control of the whole network.

9.3.2 HONEYSCOPE *Security Gateway*

For the gateway router, our prototype uses a Linksys WRT1900AC [2] router running a modified OpenWRT firmware. OpenWRT gives developers flexibility in

creating the network design and flashing the desired operating system image to it [4]. The Linksys router is considered as one of the most stable hardware that can be used with OpenWRT. However, one of the main drawbacks of using it is its limited memory. OpenvSwitch [3] is installed on the Linksys router to enable using SDN and OpenFlow. Instead of the normal bridge in the router, an OVS bridge is added to work properly with SDN. OpenvSwitch is a multiplayer virtual switch that provides more automation and programmability. SDN needs such automation to be able to perform the separation between the control plane and the data plane smoothly.

As mentioned in the previous section, one of the main goals of HONEYSCOPE is to divide the local IoT network to groups in which devices are visible to each other while they are invisible to any other devices out of this specific group's range. Figure 9.3 describes the process of grouping and isolating the new added devices. When a device joins the local IoT network, the SDN controller examines the packets coming from and to this device and processes the device's behaviors to be able to identify the device type accordingly.

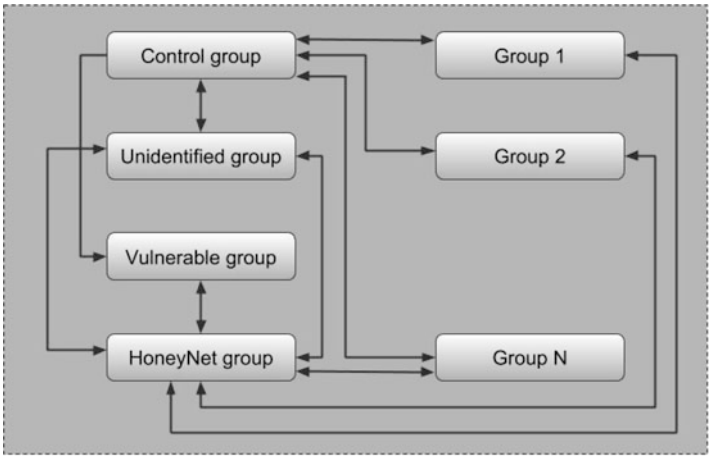

Fig. 9.4 Communication between network groups

9.3.3 Communication Between Network Groups

As shown in Fig. 9.4, the direction of the arrows represents how the communication goes between any two groups, for example, the communication between the control group and the vulnerable group is unidirectional from the control to the vulnerable group, as communication in the other direction is denied. As shown, there is no communication between different vendor-specific or function-specific groups, nor between any of these groups and the vulnerable group. Each of these groups cannot

view or access other groups. There is no need for these devices to communicate and by isolating them we reduce the attack surface of devices in the local IoT network.

The communication between the control group and the vulnerable group requires special arrangements. As mentioned, the control group has access to the vulnerable group through unidirectional communication. However, the vulnerable group doesn't have access to the control group and can't send any packets to devices in it. However, in some use cases like when devices in the vulnerable group need to be configured or software updates need to be installed on them, there needs to be a way for the controlling devices to discover devices in the vulnerable group.

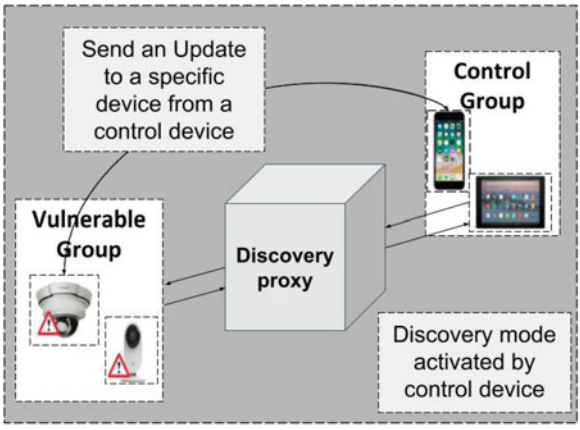

Fig. 9.5 Communication between control and vulnerable groups

Software updates for most IoT devices are facilitated using a specific smartphone application to get updates from the manufacturer servers and install them onto the device. The mobile application downloads the firmware update and uploads it to the corresponding IoT device. Note that in some cases the IoT device connects directly to the manufacturer's servers without the help of its mobile application. In such cases access to the control group may not be required.

To be able to perform software updates with the help of the controlling device, IoT devices in the vulnerable group use multicast and broadcast-based discovery protocols to allow the controlling device (e.g., smartphone, or tablet of the user) to be aware of them.

To enable the controlling device to discover vulnerable devices, a virtual discovery proxy is used for facilitating limited communication for this discovery purpose from the vulnerable group towards the control group as shown in Fig. 9.5, demonstrating how the communication between these groups is handled. The discovery proxy is activated by the controlling device (e.g., by activating a special configuration/update mode of the system). When activated, the proxy will forward broadcast and multicast messages from devices in the vulnerable group to the specific controlling device that activated the discovery mode. Thus, the controlling device can discover the presence of IoT devices while limiting their communication

to any other devices in the local IoT network. We envisage that the user can activate the discovery mode of HONEYSCOPE with the help of a smartphone application on the controlling device, thereby activating the discovery proxy. The communication between both groups can be summarized as follows:

- Communication from any control device to the vulnerable devices: this is granted by direct communication from the control device to the vulnerable group.
- Communication from vulnerable devices group to the control group: This communication is denied. It can only temporarily enabled through the discovery proxy when the controlling device initiates a special discovery mode. It acts like a "virtual VPN" between the controlling device and vulnerable group, strictly limiting the delivery of multicast and broadcast packets of devices in the vulnerable group to the controlling group only.

9.3.4 Case Study

Here we will discuss an exemplary scenario to demonstrate how HONEYSCOPE adds a level of security to the local IoT network. Assume there is a D-Link IP camera in the local IoT network that is susceptible to be infected by an IoT malware like Mirai [7]. At first, when this IP camera is joining the network, it will be identified and added to the D-Link and camera groups. Only D-Link devices, in addition to all cameras in the network, can view and have access to this D-Link camera.

Once the IoT security service notifies HONEYSCOPE that the camera is vulnerable to, e.g., Mirai, the SDN controller will move the camera to the vulnerable devices group and remove it from both the D-Link and camera groups.

When it is moved to the vulnerable group, the infected camera will have a new limited virtual network view in which it can't view any devices in the local network except the virtual devices in the HoneyNet group, and can be viewed by the controlling devices as well. This way, other devices in the D-Link and camera groups will be protected from the vulnerable device should it be infected by the IoT malware.

The communication between the controlling devices and the vulnerable devices is unidirectional: this is to enable the controlling device, e.g., to send software updates for fixing the vulnerability of the camera. After that, the camera can again be added back to the D-Link and camera groups.

9.4 Conclusion

This chapter has provided an overview of the architecture, design, and the deception function of HONEYSCOPE. HONEYSCOPE is based on creating virtual deceptive network views for each IoT device in the local IoT network. SDN technology is used to manage the network by using an SDN controller that is responsible for creating these deceptive network views for each device according to its identified

device type. For device type identification, the IoT sentinel system is used. Through the use of deceptive network views, HONEYSCOPE is able to provide a higher layer of security to the local IoT network against both external and internal attacks.

9.5 Hands-on Exercises

1. **(Intermediate)** In this exercise, you will use the Mininet SDN emulation environment (http://mininet.org/) to implement the core network views concept behind HONEYSCOPE. The best way to become familiar with Mininet is to go through the tutorial. Once you are familiar with Mininet, construct the topology shown in the left half of Fig. 9.6. The goal of this exercise is to construct the network views policy shown in the right half of Fig. 9.6. To accomplish this, you may wish to modify Mininet's "Learning Switch" tutorial to enforce access control based on hard-coded MAC addresses. Use the OpenFlow protocol to program a switch to perform custom packet forwarding. Implement a software defined network controller that can read frame and packet source and destination fields in order to implement the HONEYSCOPE network views policy. Test network visibility using the `ping` command. Note that for this exercise, you only need to worry about unicast traffic. Finally, you may choose to use the Pox controller, which is the default with Mininet, or choose another controller such as Ryu, ONOS, or OpenDaylight. Mininet can be configured to use a *remote controller* (e.g., a controller outside the control of Mininet).

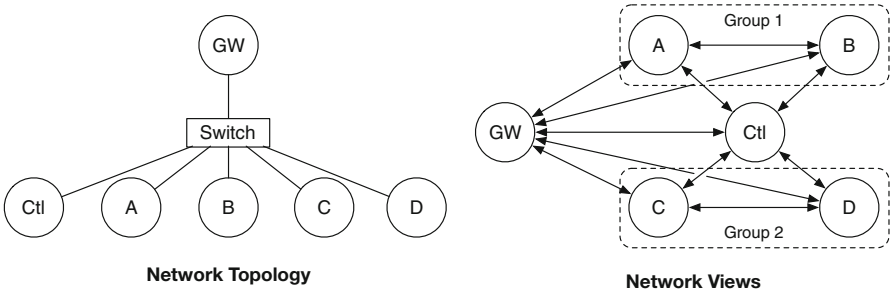

Fig. 9.6 Network topology and views for exercises. GW is the gateway and Ctl is a controlling device such as a smartphone. Arrowheads indicate network visibility. For example, GW and Ctl can see A, B, C, and D; however, A and B cannot see C and D

2. **(Advanced)** In this exercise, you will extend Exercise 1 to real hardware. To complete this exercise, you will need a router/access point capable of running OpenWRT/LEDE (https://openwrt.org/) and a Raspberry Pi. Start by setting up OpenWRT with OpenVSwitch (OVS) and your controller running on the Raspberry Pi. The WiFi SDN project at Helsinki is a good starting place (https://wiki.helsinki.fi/display/WiFiSDN/Software-Defined+Wi-Fi+Networks+with+Wireless+Isolation). Once the data plane and control plane is setup, port your

solution to Exercise 1 to this environment. Note the controller will only see one port: `wlan0`. However, by enabling WiFi isolation mode, all of the network traffic will be forced through the soft-switch, allowing the controller to define flow-mod rules that restrict which devices can receive packets from one another. Again, for this exercise, only worry about unicast traffic.

3. **(Advanced)** In the previous two exercises, you only considered unicast traffic. However, many IoT devices depend on multicast protocols (e.g., HomeKit uses mDNS), which may leak information between network groups. Further, IoT discovery protocols (e.g., SSDP) rely on multicast protocols for advertisement and discovery of network services, providing information about IoT device applications and services outside the scope of HONEYSCOPE policies. Extend Exercise 2 to also mediate multicast traffic. To test your solution, explore the use of Avahi (https://www.avahi.org/) and nss-mdns (https://github.com/lathiat/nss-mdns) from Linux. Alternatively, macOS devices advertise services via mDNS (aka Bonjour). Hint: consider making copies of multicast packets and sending the copies to hosts allowed by the policy.

References

1. Amazon's Alexa recorded private conversation and sent it to random contact. https://www.theguardian.com/technology/2018/may/24/amazon-alexa-recorded-conversation. Accessed: 2018-06-20.
2. Linksys WRT 1900AC. https://www.linksys.com/us/p/P-WRT1900AC/. Accessed: 2018-06-03.
3. OpenvSwitch. https://www.openvswitch.org/. Accessed: 2018-06-03.
4. OpenWRT. https://openwrt.org/. Accessed: 2018-06-03.
5. RYU Nicira extensions. http://ryu.readthedocs.io/en/latest/nicira_ext_ref.html. Accessed: 2018-06-03.
6. RYU SDN controller. https://osrg.github.io/ryu/. Accessed: 2018-06-03.
7. Manos Antonakakis, Tim April, Michael Bailey, Matt Bernhard, Elie Bursztein, Jaime Cochran, Zakir Durumeric, J Alex Halderman, Luca Invernizzi, Michalis Kallitsis, et al. Understanding the Mirai botnet. In *USENIX Security Symposium*, 2017.
8. H. Chung, M. Iorga, J. Voas, and S. Lee. Alexa, can I trust you? *Computer*, 50(9):100–104, 2017.
9. ONF Market Education Committee et al. Software-defined networking: The new norm for networks. *ONF White Paper*, 2012.
10. Jeremy Erickson, Qi Alfred Chen, Xiaochen Yu, Erinjen Lin, Robert Levy, and Z. Morley Mao. No one in the middle: Enabling network access control via transparent attribution. In *Proceedings of the 2018 on Asia Conference on Computer and Communications Security*, ASIACCS '18, pages 651–658, New York, NY, USA, 2018. ACM.
11. Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker, and Jonathan Turner. OpenFlow: Enabling innovation in campus networks. *SIGCOMM Comput. Commun. Rev.*, 38(2):69–74, March 2008.
12. Markus Miettinen, Samuel Marchal, Ibbad Hafeez, N. Asokan, Ahmad-Reza Sadeghi, and Sasu Tarkoma. IoT Sentinel: Automated device-type identification for security enforcement in IoT. In *Proc. 37th IEEE International Conference on Distributed Computing Systems (ICDCS 2017)*, June 2017.