Demonstrating a Personalized Secure-By-Default *Bring Your Own Device* Solution Based on Software Defined Networking

Steffen Gebert*, Thomas Zinner*, Nicholas Gray*, Raphael Durner†, Claas Lorenz‡, Stanislav Lange*
*University of Würzburg, Würzburg - {steffen.gebert,zinner,nicholas.gray,stanislav.lange}@informatik.uni-wuerzburg.de
†Technical University of Munich, Munich - r.durner@tum.de
‡genua GmbH, Kirchheim - claas.lorenz@genua.de

***Abstract*—Network virtualization is one classical use-case for Software Defined Networks (SDN). By programmatically instantiating virtual networks, traffic from one or more devices can be separated or connectivity can be established as needed. S-BYOD, which is presented in this demonstration, applies the SDN concept to Bring Your Own Device (BYOD) scenarios and offers *personalized* virtual networks that are set up and extended *on demand*. This is done once the user authenticates, activates access to additional applications, or as soon as applications scale out and involve more servers. The described proof-of-concept implementation explores, to what degree an agent-less BYOD solution, based only on SDN, can lower the attack surface by explicit user opt-ins for particular services. Further, an assessment of the number of required rules within the flow tables of switches completes this work.**

I. Introduction

While SDN's means to create virtualized networks can be helpful for BYOD scenarios, where potentially insecure devices should be included in organizational IT infrastructure, none of the currently available solutions applies SDN to offer a fine-grained restriction of network access. Instead, commercial solutions either group connect all devices into a special VLAN that can be further restricted, or - in the case of more sophisticated Mobile Device Management (MDM) solutions - an agent runs on the smartphones or tablets that checks its trustworthiness.

Sardine-BYOD (S-BYOD), the solution introduced in this work, aims at solving the problem only on the network level. No modification of the user-owned device or restriction of privileges is needed. Instead, S-BYOD restricts each device to its own virtual network. After authentication, the user can explicitly enable corporate services for this specific device following the *sudo* principle of operating systems. Based on an off-by-default approach, the number of services, to which a device running a potentially malicious app, can connect is reduced. To activate access to a particular service, an authenticated user can simply use the web portal of the BYOD solution.

II. Sardine-BYOD (S-BYOD)

The introduced BYOD solution is publicly available as open source[1] and implemented as plugin for the *ONOS* controller [1]. Its main innovation is the implementation of *personalized* virtual networks which are established and adjusted *on demand*. In contrast to other BYOD solutions, which usually group all devices together in a dedicated VLAN and thus do not allow to set up fine-grained policies, S-BYOD individually restricts network access for every device by the means of Sofware Defined Networking. Further, these virtual networks are automatically adjusted according to the user's current requirements, i.e., when additional services are requested or when the user roams between access points, as well as when changes within the IT services infrastructure are made.

A. Building Blocks

To achieve the outlined goal of SDN-based virtualized networks for BYOD users, the setup builds up on the following components and mechanisms, which are also shown in Figure 1:

OpenFlow-enabled switches: The wired network infrastructure is built using *OpenFlow*-enabled switches. OpenFlow [2] is a protocol that offers a standardized API for the communication between SDN controllers and OpenFlow-enabled switches. This allows to programmatically establish and modify individual virtual networks.

ONOS SDN controller: ONOS is a well-known controller for software defined networks and allows to modify the forwarding tables of switches using the OpenFlow protocol. By default, ONOS includes basic network services, including reactive forwarding and DHCP and offers APIs for SDN applications running inside the controller.

Wireless Access Points: Given the lack of OpenFlow-based wireless access points (APs), traditional APs are used, to which the BYOD users can connect. These APs form an

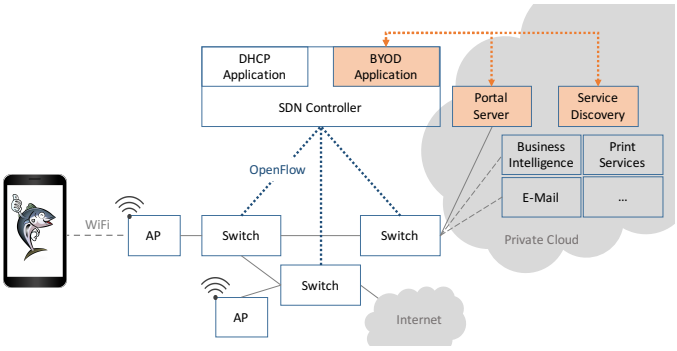

Fig. 1. Corporate network setup with BYOD client.

[1] https://github.com/lsinfo3/2016-itc-sbyod-onos-app

Extended Service Set (ESS), meaning they all advertise the same ESSID and are connected to the fixed, OpenFlow-based network.

Corporate Services: Users of the wireless network need to access business applications that are running in the corporate infrastructure. Such applications can include browser-based access via HTTPS, email services using IMAP/SMTP, or other IP-based protocols, e.g., for printing. Furthermore, Internet access might need to be available to a private device in the corporate environment. It is assumed that most of these applications are running within a private cloud environment in the corporate IT infrastructure.

Service Discovery: As cloud-based applications scale accordingly to their usage, the virtual machines, on which an application is running, change over time, i.e., when using modern deployment techniques [3]. Hence, the IP addresses of these applications change as well. In order to allow different instances of cloud applications to connect to each other resp. to connect to their microservice instances [4], service discovery systems [5] are an essential part of modern cloud-based application architectures. This work makes use of *Consul* [6] as discovery service.

Captive Portal: Finally, a web-based captive portal is used to authenticate and authorize users. This portal has access to the corporation's authentication services, i.e., LDAP/AD, and further uses *two-factor authentication* (2FA), based on the *Time-Based One-Time Password Algorithm* (TOTP, [7]) to prevent an attacker from accessing corporate services with stolen credentials. The captive portal communicates with the BYOD controller module using RESTful APIs. In the demonstration, the captive portal is implemented using *Meteor* [8] as client- and server-side web framework. Figure 2 shows the portal, in which the authenticated user can explicitly enable particular applications, while every action that elevates privileges has to be authenticated using a 2FA verification.

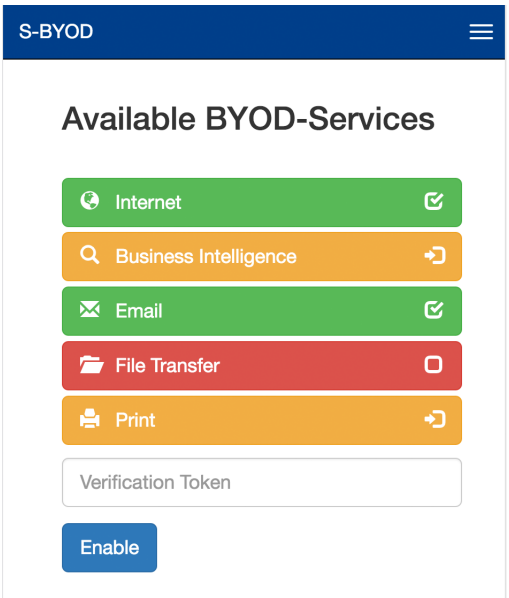

Fig. 2. Screen shot of the captive portal to activate access to corporate services. Green: activated services, red: deactivated services, yellow: services to be activated after 2FA.

B. OpenFlow Rule Setup

In order to allow basic network connectivity, redirect unauthenticated users to the captive portal, as well as to drop all unwanted traffic, the following rules are defined by the ONOS controller in order of increasing priority:

Table-miss action drop all r_{miss}**:** This *deny all* default rule drops all unmatched packets by defining an all-wildcard rule without any actions.

ARP handling r_{ARP}**:** All ARP packets are forwarded to the SDN controller and processed by ONOS' proxyarp app.

DHCP service r_{DHCP}**:** All DHCP packets are forwarded to the SDN controller and processed by ONOS' DHCP app.

HTTP to controller $r_{HTTP\_to\_ctrl}$**:** For intercepting outgoing HTTP connections of any unauthenticated client, this low-priority rule redirects any TCP traffic to port 80 to the controller. This allows S-BYOD to intercept the HTTP connection and redirect the client to the portal web site for authentication.

DNS server connectivity r_{DNS}**:** As soon as a new client is detected, rules are provisioned to allow DNS traffic to the corporate DNS server. Therefore, persistent rules for each direction between a client device and the DNS server are installed throughout the path. Connectivity to the DNS server configured through DHCP is essential also for unauthenticated users, as otherwise no attempts to establish outgoing HTTP connections will be made, which is necessary for the portal redirection.

Portal connectivity r_{portal}**:** To enable connectivity between the client device and the captive portal via a secure HTTPS connection, the complete path is provisioned by one rule per direction between client and portal server.

Service Connectivity $r_{service}$**:** As soon as a user has successfully authenticated at the portal and enabled a particular service, these rules permit network-side access to the IP addresses of the servers offering this service, e.g., email, web-based service, or any other protocol. Given the higher priority of rules, the previously applied rules to drop packets are overridden to successfully forward packets between the client and the destination server.

Internet Connectivity $r_{external}$**:** All traffic that is routed outside of the Layer 2 network, in which clients and the accessed applications reside, require connectivity to the default gateway. Therefore, once the user enables Internet connectivity in the portal, rules that allow traffic between the MAC addresses of the client and the default gateway are installed on switches along the path. Given the gateway's configuration to not route traffic inside the OpenFlow-managed network, access to this MAC address does not allow the user to bypass security means which prevent accessing other internal devices.

All client-specific rules are installed once the new client is connected. All service-specific rules are installed once a user enables or disables application access in the captive portal. Once the BYOD service gets notified by the controller that a host disappeared from the network, all these rules are removed from the switches.

Once the controller receives packets originating from an unauthorized user accessing a web page via the $r_{HTTP\_to\_ctrl}$ redirection, it instructs the switch to rewrite the source and

destination IP and MAC addresses to the portal using a `packet_out` message. As the packets on the way back from the portal web server to the user also need to match the intercepted connection's addresses, further `packet_out` messages also rewrite the endpoint addresses of this intercepted connection. As the portal only responds to the HTTP request with a HTTP *Location* redirect header to its resolvable address, no flow entries are set up in the switches for this short-lived connection.

C. Wireless Client Mobility

To ensure high user satisfaction, wireless clients need to be able to roam between wireless access points, i.e., when moving from one room or building to another. The security mechanisms and rules installed by the BYOD solution therefore have to support this use case.

As the introduced S-BYOD relies mostly on static rules for performance and scalability reasons, a roaming client needs to be detected quickly and accordant rules need to be updated as well. RFC 5227 [9] introduces procedures for mobile clients to detect the correct setup of wireless networks spanning multiple access points. This is achieved by sending an ARP packet to the previous gateway, once the client switches to the new AP. As this mechanism is implemented in most of the current operating systems, this also offers a safe way to detect the roaming device. In particular, the controller immediately notices this through the received ARP packets which originate from a different switch (port) and thus allows S-BYOD to update all flow rules and maintain connectivity for the roaming device.

III. Discussion

While ease of use is one important factor for user satisfaction, security of a BYOD implementation is an important aspect as well. Therefore, different mechanisms of S-BYOD that mitigate or still allow certain known attacks will be discussed in the following. Further, an estimation of the resource requirement in number of flows in OpenFlow switches will be provided.

A. IP Address Spoofing

To allow connectivity to the portal, every device connected to the WiFi has to receive an IP address from the DHCP service. By spoofing the IP address of another, already authenticated client, an attacker could gain access to internal services. Guessing such IP address is easy, as the used IP address range is known once connected to the network. Therefore, all client-specific flow rules $r_{service}$ and $r_{external}$ have to also match against the client's MAC address and thus lead to all traffic using a self-assigned IP address being dropped.

B. Wireless Client Isolation

In order to prevent communication between mobile devices and especially to prevent an attacker gaining knowledge of other (potentially authenticated) users' MAC addresses, the access points operate in *client isolation* mode. By activating such a setting, no incoming traffic of wireless clients is sent directly back to the air interface, but only to the wired connection. There, the wired OpenFlow-based network takes care of separation of clients. Further implications of ARP/MAC spoofing will be discussed in Section III-D. OpenFlow-enabled WiFi-APs, if they would exist, are therefore not even needed, as long as wireless clients never need to communicate directly.

C. Estimation of OpenFlow Rules

One concern with large-scale OpenFlow deployments is the number of required forwarding rules. Therefore, the usage of flow table entries is discussed in the following. As all rules defined by S-BYOD use exact matches, i.e., MAC and IP addresses, these rules can be stored in content-addressable switch memory. In contrast, using wildcards, e.g., matching for `nw_dst= '192.168.0.*'`, would require expensive and very limited ternary content-addressable memory (TCAM).

In real-world scenarios this is mitigated by utilizing multiple switches and thus, the client-specific rules are distributed over the switches. Then, only the switches on the path between a client and its connection endpoints receive the accordant rules. For the extreme case that all users are connected to only a single switch, the number of required rules is calculated as follows and provides a worst-case estimation:

$$n_{total} = n_{base} + n_{client} + n_{service} \tag{1}$$

With the following rules to provide the basic network setup:

$$\begin{aligned} n_{base} = &\underbrace{|r_{miss}|}_{1} + \underbrace{|r_{ARP}|}_{1} + \underbrace{|r_{DHCP}|}_{2} \\ &+ \underbrace{|r_{discovery}|}_{2} + \underbrace{|r_{HTTP\_to\_ctrl}|}_{1} \end{aligned} \tag{2}$$

$r_{discovery}$ contains rules which are automatically added by the controller for discovery of links and broadcast domains.

For each of the c connected clients, 4 additional rules are added to enable basic connectivity, regardless of the client's authentication status:

$$n_{client} = c \cdot (\underbrace{|r_{DNS}|}_{2} + \underbrace{|r_{portal}|}_{2}) \tag{3}$$

Finally, to estimate the number of rules for enabled services, $s_{i,j} = 1$ denotes service j being enabled for client i and $|r_{service,j}|$ denotes the number of rules required for this service:

$$n_{services} = \sum_{i=1}^{c} \sum_{j=1}^{j_{max}} s_{i,j} \cdot |r_{service,j}| \tag{4}$$

Figure 3 depicts this relationship for different numbers of active BYOD clients and enabled applications per user, while assuming that every service is reachable through two IP addresses.

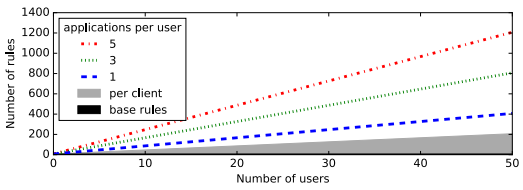

Fig. 3. Amount of required rules in a single-switch setup, 2 IPs per application.

D. Open Issues and Possible Extensions

While the presented S-BYOD system already provides a solid state of implementation that can compete with other existing BYOD solutions, different aspects offer room for further improvements:

Captive Portal Detection: Modern operating systems implement a portal detection (aka. *Walled Garden Detection*), which expects a certain return code or content for well-known URLs[2]. As such detection is not always working flawlessly, several approaches are currently in discussion, including *Wireless Internet Service Provider roaming* (WISPr), which automates logins, as well as [10], which suggests a new DHCP option (for IPv4) and a new IPv6 Router Advertisement option for restricted network conditions. These problems, however, are not specific to the proposed solution.

ARP spoofing: The authentication using the captive portal is one critical factor. As soon as an attacker spoofs the MAC address of an authenticated user, the corresponding session can be taken over. Wireless client isolation already hides devices from each other [11]. Further, concepts similar to Cisco's *port security* [12] can be implemented in the controller to detect ARP spoofing attacks. Finally, 802.1X-based network access control would offer the most secure way for authentication and can be implemented on top of the current setup. The portal using 2FA would then only be used for authorization of particular services.

DNS tunneling: A well-known attack vector of captive portal authentication services is DNS tunneling. Using such a mechanism, an unauthenticated user can communicate with the Internet using an own, external DNS relay server that receives the traffic encapsulated in the user's DNS requests forwarded by the corporate DNS server. In order to detect and prevent such tunneling, an extra DNS resolver can be installed that monitors the number of host names requested per device. As soon as a rate limit is exceeded, an administrator can be notified or the device can be automatically blocked.

IV. Related Work

Programmable BYOD Security (PBS [13]) applies SDN/OpenFlow to mobile devices using the *PBS-DROID* application. This lets the Android device run an OpenFlow-controlled switch with the smartphone apps connected to it. By running on the device itself, it allows per-app policies. Compared to S-BYOD, PBS is able to work on a more fine-grained level, while the network infrastructure itself does not have to be changed. In contrast, the motivation behind S-BYOD is to not require changes to the devices, but instead providing a high level of security through the support of the network infrastructure.

[2]Google OSs expect an HTTP status 204 from `http://clients3.google.com/generate_204` / Apple OSs access `http://captive.apple.com/hotspot-detect.html`

V. Conclusion

The presented approach for an SDN-based BYOD implementation can be used together with an existing OpenFlow-based infrastructure. It makes use of SDN's means to dynamically define virtual networks, which are set up and maintained on a per-user level. Using a portal web site and two-factor authentication, a safe yet comfortable way is provided to the user, to only selectively enable connectivity to the applications that are currently needed and thus lower the range of services that a potentially malicious device can access. By connecting to a service discovery system used by corporate applications, changes can be immediately reflected with updates to the flow rules in the switches and thus provide always up-to-date network access.

Acknowledgments

This work has been performed in the framework of the SARDINE project and is partly funded by the BMBF (Project ID 16KIS0261). The authors alone are responsible for the content of the paper. The authors want to thank Lorenz Reinhart and Benedikt Pfaff for their programming work.

References

[1] ONOS project - Open Network Operating System. [Online]. Available: http://onosproject.org/

[2] Open Networking Foundation, "OpenFlow switch specification, version 1.5.0," Dec 2014.

[3] J. Humble and D. Farley, *Continuous Delivery: Reliable Software Releases Through Build, Test, and Deployment Automation*, 1st ed. Addison-Wesley Professional, 2010.

[4] S. Newman, *Building Microservices*. O'Reilly Media, 2015.

[5] G. Kousiouris, D. Kyriazis, T. Varvarigou, E. Oliveros, and P. Mandic, *Taxonomy and State of the Art of Service Discovery Mechanisms and their relation to the Cloud Computing Stack Cloud Computing Stack*. Information Science Reference - Imprint of: IGI Publishing, 2012, ch. 11.

[6] HashiCorp. Consul. [Online]. Available: https://www.consul.io/

[7] D. M'Raihi, S. Machani, M. Pei, and J. Rydell, "TOTP: Time-Based One-Time Password Algorithm," RFC 6238 (Informational), Internet Engineering Task Force, May 2011.

[8] Meteor. [Online]. Available: https://www.meteor.com/

[9] S. Cheshire, "IPv4 Address Conflict Detection," RFC 5227 (Proposed Standard), Internet Engineering Task Force, Jul. 2008.

[10] W. Kumari, O. Gudmundsson, P. Ebersman, and S. Sheng, "Captive-portal identification in DHCP / RA," Working Draft, August 2015. [Online]. Available: http://www.ietf.org/internet-drafts/draft-wkumari-dhc-capport-16.txt

[11] T. Mirzoev and S. White, "The role of client isolation in protecting wi-fi users from ARP spoofing attacks," *CoRR*, vol. abs/1404.2172, 2014.

[12] National Security Agency (NSA), "Port security on cisco access switches," https://www.nsa.gov/ia/_files/factsheets/factsheet-cisco%20port%20security.pdf.

[13] S. Hong, R. Baykov, L. Xu, S. Nadimpalli, and G. Gu, "Towards sdn-defined programmable byod (bring your own device) security," in *Proceedings of the 2016 Network and Distributed System Security Symposium (NDSS'16)*, February 2016.