DARUD: Detecting and Arresting Rogue USB Devices in the V2X Ecosystem

Eyasu Getahun Chekole
Singapore University of Technology and Design
Singapore, Singapore
eyasu_chekole@sutd.edu.sg

Huaqun Guo
Cybersecurity Research Center
Singapore, Singapore
guohuaqun@u.nus.edu

***Abstract*—Vehicle-to-Everything (V2X) is a cutting-edge technology in intelligent transportation systems (ITS). In V2X, various entities communicate and cooperate each other to ensure road safety and efficiency. However, such communications and cooperation also pose various security risks to the transportation system. In particular, due to the involvement of several unattended roadside devices, such as roadside units (RSUs), sensors and controllers, the V2X ecosystem is highly vulnerable to malware attacks. Attackers may use rogue USB devices to inject various types of malware to the V2X system via the USB ports of the roadside devices. Such attacks may result in a debilitating impact on the safety and efficiency of road traffic. Although a wide-range of approaches have been proposed against USB-based attacks, most of them have several limitations, especially when applied in the V2X environment. For example, the widely adopted approaches against USB-based attacks are scanning USB devices using anti-malware/antivirus tools (which is often not effective against zero-day malware), disabling USB ports (security-through-obscurity has already several drawbacks), whitelisting USB devices using certain attributes of the USB devices (which is often bypassed by brute-force attacks and not effective against dishonest USB users). Furthermore, most of the existing solutions are implemented only for general-purpose computing devices (e.g., computers and servers), hence might not be suitable for sensors and tiny IoT devices involved in the V2X environment. Moreover, the configuration and update processes of most existing solutions requires physical access to the devices, which might not be feasible in V2X where devices and sensors are dispersed across various roadside locations. In this work, we propose and implement DARUD – a lightweight and automated toolkit that dynamically detects and prevents rogue USB devices in V2X. This is achieved by constructing a USB authorization policy based on kernel-level USB rules and fingerprints. The proposed solution can also be configured and updated to the roadside devices over-the-air via a secured VPN tunneling. This avoids the hassle of physically configuring or updating the USB-security solutions on each roadside device. The effectiveness of our proposed approach is also tested using a realistic V2X infrastructure.**

***Index Terms*—USB Security, USB Authorization, USB Whitelisting, Port-Security, V2X Security, Malware**

I. INTRODUCTION

Nowadays, Universal Serial Bus (USB) storage devices have been widely adopted in the computing industry, by advancing the traditional serial and parallel ports. USB offers several advantages, such as hot swapping, high data processing speed, and plug-and-play (PnP) capabilities, among others. However, due to the open standards and insecure designs of USB technologies, a wide-range of attackers have successfully exploited various vulnerabilities in USB protocols [1], [2], [3], [4], [5]. The most widely known USB-based attacks are injecting various types of malwares (e.g., computer viruses, spyware, ransomware and computer worms) to the host machines using rogue USB devices [6], [7]. By doing so, an attacker may hijack or subvert operations of the host machines, corrupt their memory system [8], [9] or takeover the machine in general.

The V2X ecosystem is more vulnerable to USB-based attacks due to the involvement of various unattended devices and sensors in V2X that are widely dispersed across various roadside locations. Since the V2X services are also highly interconnected, a malware attack even on a single V2X device may result in a devastating impact to the whole V2X ecosystem. Therefore, USB-based attacks are critical concerns in V2X [10].

Since USB-based attacks are also common problems even in general IT systems, a wide-range of countermeasures have been proposed against these attacks. However, they have several limitations, especially when applied in the V2X context. For example, the most commonly approach to prevent USB-related attacks is locking or disabling USB ports of the devices so that an attacker cannot inject a malware via the disabled ports [11], [12]. Such approach of *security-through-obscurity* has already several drawbacks, including restricting service availability and maintainability. Another widely adopted approaches against USB-based attacks is scanning USB devices using anti-malware/antivirus tools [13], [14]. This approach is often effective only against known malware, i.e., whose signature or behaviour is already known. It is not effective against unknown malware or zero-day attacks.

Another widely adopted approach against USB-based attacks is whitelisting USB devices using certain attributes of the devices [15], [16]. However, since various USB devices have many attributes in common, brute-force attackers [17] may easily make a matching with the whitelisting policies. Moreover, this approach does not work against dishonest users. For example, a dishonest user may share the attribute values of his whitelisted USB device to an attacker. The attacker can then spoof attributes of his own USB device and change it with the attribute values provided by the dishonest user. This will whitelist the attackers USB device to get access to the targeted host machines.

Another common limitation of most of the approaches discussed above is that the security configurations and updates are made physically, which requires physical access to the host device. Such configuration and updating mechanisms might not be feasible and convenient in the V2X environment as the devices and sensors in V2X are widely scattered over various roadside locations. Moreover, the most commonly used USB protection mechanism in the V2X environment is still physical security, e.g., physically locking the devices in a cabinet. Such approaches are not sufficient as an attacker can force to open cabinets either by unlocking physical keys or breaking the cabinet itself.

Furthermore, most of the existing solutions are implemented for general-purpose computing devices (e.g., computers and servers), hence might not be suitable for sensors and tiny IoT devices involved in the V2X environment. So, porting such solutions in the V2X environment could be challenging. Some solutions might also introduce high runtime overheads, which may affect the real-time and service availability requirements in delay-sensitive systems such as V2X [18], [19].

To overcome most of the limitations discussed above, we propose and implement a lightweight solution that effectively and dynamically **D**etects and **A**rrests **R**ogue **U**SB **D**evices (**DARUD**) from getting access to the V2X host devices. In particular, DARUD is lightweight toolkit that combines kernel-level USB rules and fingerprints to authorize USB devices in V2X, beyond the general-purpose computing devices.

The USB rules are constructed and generated using certain attributes (ATTRs) and environment variables (ENVs) of the USB device. The choice of ATTRs and ENVs is also determined by certain criteria. For example, the ATTRs and ENVs must be unique identifiers of the USB device and they must not change in certain conditions or through time. To generate a USB rule, DARUD first scans values of the ATTRs and ENVs specified in the rule construction. The generated USB rule will be deployed to the host device, and triggered at runtime by a USB kernel event (i.e., when the USB device is connected to the host device) and get executed to authorize the device.

However, only the USB rule is not enough to effectively authorize the USB device. This is because, such USB rules can be bypassed by brute-force attacks and, as discussed above, a dishonest user may share the ATTR and ENV values of his already authorized USB device to an attacker. To circumvent this problem, we also compute a fingerprint of the USB device and use it alongside the USB rule to effectively authorize the USB device. The fingerprint is computed using a cryptographic hash function over certain attributes of the USB device (typically the ATTRs and ENVs used to construct the USB rule) and a nonce (e.g., timestamp). Since it is computationally infeasible to reverse the attributes and the nonce from the hash value, the fingerprint is not susceptible to reverse-engineering or other types of attacks.

Finally, DARUD combines the generated USB rule and the fingerprint to form the USB authorization policy. This policy will be deployed at the host devices (i.e., devices deployed at roadsides) and gets executed at runtime (when triggered by a USB kernel event) and dynamically authorize USB devices (i.e., to deny or grant access to the host devices). The enforcement (i.e., configuration and update) of the authorization policy to the roadside devices in V2X is carried out over-the-air via a secured VPN tunneling. This avoids the hassle of physically configuring or updating the USB-security solutions on each roadside device in V2X. The policy will also be stored only in a specified root directory (e.g., "/lib/udev/rules.d/" for Linux systems) of the host device to prevent illegitimate manipulations of it by unauthorized users.

The effectiveness of our proposed approach is tested using a smart traffic light control system deployed in a realistic V2X infrastructure. In general, our paper makes the following main contributions:

1) We propose and develop DARUD – a lightweight toolkit that effectively and dynamically authorizes USB devices at runtime. DARUD systematically constructs the authorization policy (using kernel-level USB rules and fingerprints) to effectively and dynamically detect and arrest rogue USB devices. It is also effective against USB-based brute-force and zero-day malware attacks.
2) The proposed approach is effective even in the presence of a dishonest user who intends to share attributes of his whitelisted USB device to an attacker.
3) DARUD enforces the proposed authorization policy to the roadside V2X devices using an over-the-air approach via a secure VPN tunneling. Hence, the configuration is hassle-free (not require physical access to roadside devices) and secured against man-in-the-middle attacks.
4) The effectiveness of the proposed approach is evaluated using a realistic V2X infrastructure.
5) The proposed approach is efficient and does not introduce any significant delay or service degradation in the V2X operations.

II. DARUD: The Proposed Approach

As discussed in the introduction, DARUD is a lightweight toolkit we developed to dynamically detect and arrest rogue USB devices from getting access to the V2X devices. This is achieved by authorizing USB devices via systematically constructed kernel-level USB rules and fingerprints. DARUD comprises several scripts that perform different operations in the authorization process. It is deployed on a dedicated machine hosted at the management station where the attributes of USB devices get scanned for authorization by an administrator. In the following sections, we discuss a detailed account of DARUD, including its operations, policy construction, policy enforcement strategies, and workflows.

A. Scanning attributes

To generate the kernel-level USB rules, we need to first scan and retrieve values of the relevant USB attributes (ATTRs) and environment variables (ENVs). When the USB device is connected to the management machine (where DARUD is running on), DARUD automatically scans the USB device and retrieve the required attribute values of the USB device, e.g., serial number, device ID, manufacturer ID, etc. These values will be used to generate the USB rules (cf. Section II-B).

B. Constructing USB rules

The USB rules, which will be used to authorize the USB devices, are constructed using systematically selected ATTRs/ENVs and their value assignments. In particular, we construct the rules based on the specifications and naming conventions of udev (a userspace handler for kernel events in Linux). In the proposed approach, DARUD constructs the USB rule as follows.

$$R_{USB} \leftarrow (ACTION == ACVAL) \wedge (ATTR\{1\} == AVAL1) \wedge (ATTR\{2\} == AVAL2) \wedge (ATTR\{n\} == AVALn) \wedge ... \wedge (ENV\{1\} == EVAL1) \wedge (ENV\{2\} == EVAL2) \wedge (ENV\{n\} == EVALn) \quad (1)$$

Where R stands for rule, ACVAL is the action value which is either "ADD" (if the rule is to be executed when the USB device is connected to the host device) or REMOVE (if the rule is to be executed when the USB device is disconnected from the host device), AVAL is value of the attribute and EVAL is value of the environment variable.

The USB rule is then generated using the the retrieved USB ATTR values (e.g., ATTR{serial} == "370E0051E723", ATTR{idVendor} == "0451", ATTR{idProduct} == "8142") and ENV values (e.g., ENV{ID_MODEL_ID} == "0x24fd", ENV{ID_NET_NAME_MAC} == "wlx1c1bb5774f98"). The generated USB rule will be then saved in the udev file format and specification.

As discussed in the introduction, the choice of ATTRs and ENVs is a critical and an important aspect when constructing a USB rule. Some of the criteria to choose the ATTRs and ENVs are,

- That uniquely identifies USB devices.
- Non-volatile (not changeable) in certain conditions or through time. For example, attributes or environment variables whose values vary when the USB device is unplugged and plugged again should not be chosen.

C. Computing fingerprint

As discussed in the preceding sections, only the USB rules are not effective to detect and mitigate rogue USB devices. Because, there is a high chance that a brute-force attacker can reproduce the USB rules. Furthermore, a dishonest user may share attribute values of his authorized USB device to an attacker in exchange of some incentives.

To overcome these issues, we create a cryptographic fingerprint (FP) of the USB device as an additional layer of security. More specifically, we use the USB rule created above and a nonce (e.g., timestamp) as parameters to compute the USB fingerprint. Formally, the USB fingerprint is constructed as follows.

$$FP_{USB} = \mathcal{H}(R_{USB}, timestamp) \quad (2)$$

where $\mathcal{H}(\cdot)$ is a cryptographic hash function, R_{USB} is the whitelisting rule constructed using Eq. (1), and $timestamp$ is the system time at which the fingerprint is generated and it serves against replay attacks.

Since reversing a cryptographic hash function is computationally infeasible, an attacker cannot reconstruct the attributes and nonce from the generated USB fingerprint. The generated fingerprint will also be hardcoded to the USB device hardware in a read-only and immutable format, and can be queried via USB standard-defined ways. The fingerprint will be then used alongside the USB rule to form the USB authorization policy (cf. Section II-D).

D. The authorization policy

The authorization (Authz) policy is the one that decides whether a given USB device can get access to the host machine (the roadside V2X device in this case) or not. It is a Boolean-valued function over the verification of both the USB rule, i.e., $verify(R_{USB})$, and the fingerprint, i.e., $verify(FP_{USB})$. Formally,

$$Authz_{USB} \leftarrow verify(R_{USB}) \wedge verify(FP_{USB}) \quad (3)$$

Where $verify(R_{USB})$ and $verify(FP_{USB})$ are to be executed at runtime when the USB device is connected to the V2X device. Details are provided below.

Authorization of the USB device takes place when it is connected to the host machine (or the V2X device in this case). At the time of plugging the USB device to the V2X device, a kernel event will automatically trigger (via the udev daemon) the execution of the USB rule verification, i.e., $verify(R_{USB})$ and the USB fingerprint verification, i.e., $verify(FP_{USB})$. In brief, $verify(R_{USB})$ checks the matching between the ATTR and ENV values of the USB device retrieved at runtime with that of the values stored in the generated R_{USB}. Any mismatch of the ATTR and ENV values means the USB device is not the authorized one, hence it will be automatically blocked from accessing the V2X device. Similarly, the $verify(FP_{USB})$ checks whether the fingerprints match at runtime by comparing the FP_{USB} value stored in the USB device and the one previously stored in the V2X device (or by computing a new FP_{USB} from the stored R_{USB} and its respective timestamp). A mismatch between the two fingerprints means the USB device is not the authorized one, hence it will be automatically denied access to the V2X device. Therefore, the USB device will be authorized and get access to the V2X devices only when both the kernel-rules and fingerprint checks are successful.

E. Enforcing the policy

One of the limitations with the existing USB-based solutions is that their inability to easily configure the solutions on remotely stationed devices. DARUD solved this problem by implementing an over-the-air security configuration option. As such, DARUD can remotely enforce the generated authorization policy (involving the USB rules and fingerprints) to all

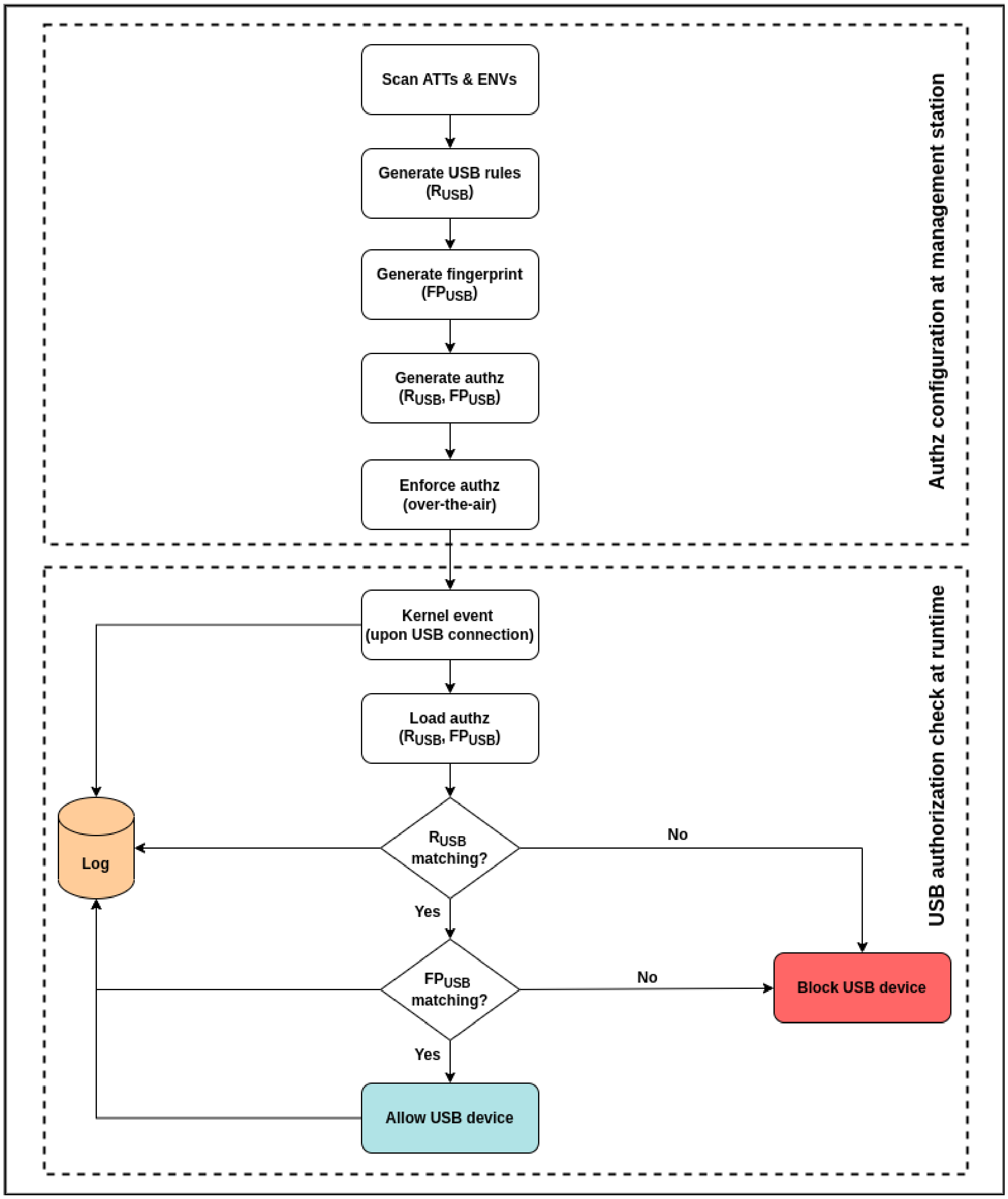
Authz configuration at management station
Scan ATTs & ENVs
Generate USB rules (R_{USB})
Generate fingerprint (FP_{USB})
Generate authz (R_{USB}, FP_{USB})
Enforce authz (over-the-air)
USB authorization check at runtime
Kernel event (upon USB connection)
Load authz (R_{USB}, FP_{USB})
Log
R_{USB} matching?
No
Yes
FP_{USB} matching?
No
Yes
Block USB device
Allow USB device

Fig. 1. A high-level architecture of DARUD

roadside devices involved in the V2X ecosystem. The over-the-air policy enforcement is carried out via a secured VPN tunneling to ensure that the policy and related information in transit is not altered by a man-in-the-middle attack. Furthermore, since the policy will be stored only in the root directory (e.g., “/lib/udev/rules.d/” for Linux systems), the user must have root access to the remote V2X devices to be able to enforce the authorization policy. This further prevents illegitimate users from enforcing an arbitrary authorization policy.

F. Logging port activities

As an additional layer of security, DARUD also includes a logging functionality. Any USB port activities, e.g., any attempt of plugging or unplugging USB devices to the V2X devices, will be logged and reported back to the management station. The administrator can then monitor any illegal activities done on the USB ports of the roadside V2X devices.

G. Workflow

As discussed in the preceding sections, DARUD performs pre-authorization (i.e., scanning the USb device and generating USB rues and fingerprints at the management station) and runtime USB authorization (i.e., authorizing the USB device when it is connected to the host machine at runtime). The overall USB device authorization process in DARUD is illustrated in Figure 1.

III. Experimental Design

The effectiveness of DARUD has been tested using a realistic V2X infrastructure. In particular, it is tested using an AI-based smart mobility management system (SmartMMS). SmartMMS is designed to enhance ***road safety*** (e.g., by broadcasting traffic accidents and road hazards to other vehicles) and ***traffic efficiency*** (e.g., by making smart decision when to turn “ON” or “OFF” traffic and pedestrian lights). This is achieved by correlating and aggregating various information obtained from different sensors (e.g., 3D laser scanners, line laser scanners, pedestrian sensors, video cameras, traffic light push-buttons and loop detectors) or beacon messages collected by RSUs from the on-board units (OBUs) of vehicles.

Architecturally, SmartMMS is a complex infrastructure consisting of various devices placed at remote sites (i.e., field devices) and backend stations. In addition to the sensors mentioned above, the remote sites consist of site computers (where SmartMMS runs on), RSUs and network switches. The backend stations also consists of management computers (where the proposed USB authorization policy generations and configurations take place), servers and terminal desktops. The communication between the remote site and backend computers is tunneled through a VPN connection (configured using industrial standard cryptographic protocols). The communication between the vehicle equipped with OBU and RSUs is carried out via a dedicated short-range communication (DSRC) wireless link. The overall architecture of SmartMMS is depicted in Figure 2. Most of the devices involved in SmartMMS have USB ports, which are susceptible for USB-based attacks. Therefore, we enforce the proposed policy in those devices to protect SmartMMS against such attacks.

IV. Results

A. Security guarantees

The effectiveness of the proposed approach is evaluated using the SmartMMS V2X infrastructure. DARUD effectively detects and mitigates rogue USB devices without any false positive or false negative errors. DARUD also ensures end-to-end security as the over-the-air USB configurations takes place via a secured VPN tunneling. Furthermore, DARUD effectively prevents brute-force and zero-day USB attacks as well as dishonest USB users.

B. Efficiency

The proposed USB authorization policy is very lightweight and it can only be triggered when a USB device is connected to the V2X device. As such, there is no any significant delay or service degradation when DARUD is tested on the SmartMMS testbed.

V. Related work

Most of the related works in USB security and their respective limitations are highlighted in the introduction. The widely adopted USB-security mechanism, especially in the V2X context, is locking the roadside V2X devices in a cabinet (some uses smart locking) [11], [12]. Such approach is far from effective as an attacker can forcibly open or break the cabinet. Scanning USB devices from malware/viruses is also another common approach [13], [14]. However, it is not effective against zero-day malwares. Although USB whitelisting-based approaches [15], [16] have been promising to prevent USB-based attacks, they are also often bypassed by hardware spoofing and brute-force attacks [17]. Therefore, the existing approaches against USB-based attacks are far from effective, especially when applied in the V2X environment. We hope DARUD would help to bridge the research gap in this regard.

VI. Conclusion

USB-based attacks have been a widely known security concerns for decades. Such attacks are even worse in the V2X environment where several roadside devices and sensors left unattended. In this work, we developed DARUD, a lightweight toolkit that systematically authorizes USB devices before they are in use. Unlike the existing solutions, the authorization policy in DARUD combines kernel-level USB rules and fingerprints to dynamically detect and block rogue USB devices in V2X. The proposed approach addressed several limitations of the existing solutions. These includes, effectively preventing USB-based brute-force attacks, zero-day attacks, and dishonest users. It also provides an over-the-air security configuration option, which is feasible for the V2X environment. In the future, we are intended to further enhance the processes in DARUD and evaluate it using different use-cases.

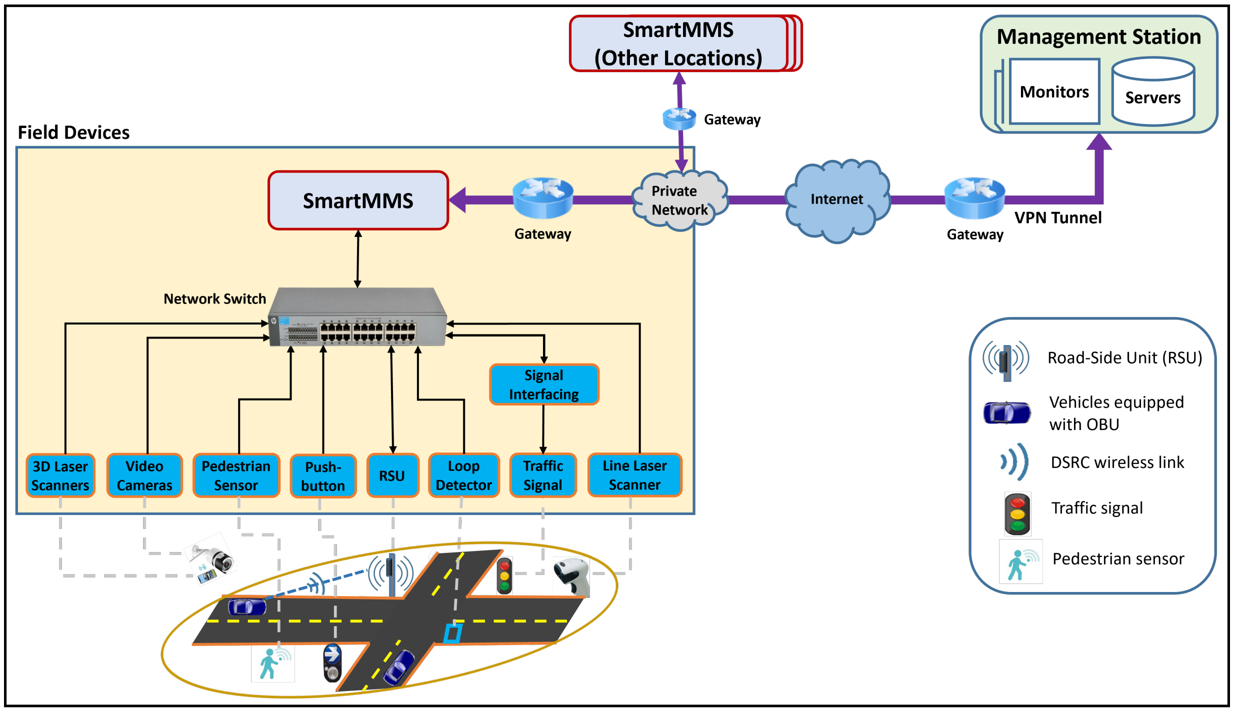

Fig. 2. A high-level architecture of the SmartMMS infrastructure

References

[1] N. Nissim, R. Yahalom, and Y. Elovici, "Usb-based attacks," *Computers & Security*, vol. 70, pp. 675–688, 2017. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167404817301578

[2] D. Pham, A. Syed, and M. Halgamuge, "Universal serial bus based software attacks and protection solutions," *Digital Investigation*, vol. 7, pp. 172–184, 04 2011.

[3] P. Bojović, I. Basicevic, M. Pilipovic, Z. Bojovic, and M. Bojovic, "The rising threat of hardware attacks: Usb keyboard attack case study," 2019.

[4] C. Cimpanu, "A list different types of usb attacks," 03 2018. [Online]. Available: https://www.bleepingcomputer.com/news/security/heres-a-list-of-29-different-types-of-usb-attacks/

[5] D. V. Pham, A. Syed, and M. N. Halgamuge, "Universal serial bus based software attacks and protection solutions," *Digital Investigation*, vol. 7, no. 3, pp. 172–184, 2011. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1742287611000041

[6] H. Liu, R. Spolaor, F. Turrin, R. Bonafede, and M. Conti, "Usb powered devices: A survey of side-channel threats and countermeasures," *High-Confidence Computing*, vol. 1, no. 1, p. 100007, 2021.

[7] R. A. Efendy, A. Almaarif, A. Budiono, M. Saputra, W. Puspitasari, and E. Sutoyo, "Exploring the possibility of usb based fork bomb attack on windows environment," in *2019 International Conference on ICT for Smart Society (ICISS)*, vol. 7, 2019, pp. 1–4.

[8] E. G. Chekole, J. H. Castellanos, M. Ochoa, and D. K. Y. Yau, "Enforcing memory safety in cyber-physical systems," in *Katsikas S. et al. (eds) Computer Security. SECPRE 2017, CyberICPS 2017*, 2017. [Online]. Available: https://doi.org/10.1007/978-3-319-72817-9_9

[9] E. G. Chekole, S. Chattopadhyay, M. Ochoa, and G. Huaqun, "Enforcing full-stack memory safety in cyber-physical systems," in *Proceedings of the International Symposium on Engineering Secure Software and Systems (ESSoS'18)*, 2018. [Online]. Available: https://doi.org/10.1007/978-3-319-94496-8_2

[10] A. D. Kumar, K. N. R. Chebrolu, R. Vinayakumar, and S. K. P, "A brief survey on autonomous vehicle possible attacks, exploits and vulnerabilities," *CoRR*, vol. abs/1810.04144, 2018.

[11] J. Oliveira, P. Pinto, and H. Santos, "Distributed architecture to enhance systems protection against unauthorized activity via usb devices," *Journal of Sensor and Actuator Networks*, vol. 10, 2021.

[12] CurrentWare, "How to disable usb ports & block usb mass storage devices," 2021. [Online]. Available: https://www.currentware.com/how-to-disable-usb-ports/

[13] S. Neuner, A. G. Voyiatzis, S. Fotopoulos, C. Mulliner, and E. R. Weippl, "Usblock: Blocking usb-based keypress injection attacks," in *Data and Applications Security and Privacy XXXII*, F. Kerschbaum and S. Paraboschi, Eds. Cham: Springer International Publishing, 2018.

[14] A. Safarkhanlou, A. Souri, M. Norouzi, and S. E. H. Sardroud, "Formalizing and verification of an antivirus protection service using model checking," *Procedia Computer Science*, vol. 57, pp. 1324–1331, 2015, 3rd International Conference on Recent Trends in Computing 2015 (ICRTC-2015). [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1877050915019729

[15] H. Mohammadmoradi and O. Gnawali, "Making whitelisting-based defense work against badusb," in *Proceedings of the 2nd International Conference on Smart Digital Environment*, ser. ICSDE'18. New York, NY, USA: Association for Computing Machinery, 2018, p. 127–134.

[16] X. Shen, N. Li, C. Liu, and M. Cai, "Wmud: A whitelist method of usb devices redirection for virtual desktop," in *IEEE 21st International Conference on High Performance Computing and Communications; IEEE 17th International Conference on Smart City; IEEE 5th International Conference on Data Science and Systems (HPCC/SmartCity/DSS)*, 2019.

[17] E. Tirado, B. Turpin, C. Beltz, P. Roshon, R. Judge, and K. Gagneja, "A new distributed brute-force password cracking technique," in *Future Network Systems and Security*, R. Doss, S. Piramuthu, and W. Zhou, Eds., 2018.

[18] E. G. Chekole and G. Huaqun, "Ics-sea: Formally modeling the conflicting design constraints in ics," in *Proceedings of the Fifth Annual Industrial Control System Security (ICSS) Workshop*, 2019, p. 60–69.

[19] E. G. Chekole, S. Chattopadhyay, M. Ochoa, H. Guo, and U. Cheramangalath, "Cima: Compiler-enforced resilience against memory safety attacks in cyber-physical systems," *Computers & Security*, vol. 94, p. 101832, 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167404820301061