SECURING INTERNET OF THINGS DEVICES USING CONTAINERS

Senthil Kumaran. S¹, Balakannan. S. P. ²

¹Research Scholar, Department of Information Technology, Kalasalingam Academy of Research and Education, Krishnankoil, Tamil Nadu, INDIA.
²Associate Professor, Department of Information Technology, Kalasalingam Academy of Research and Education, Krishnankoil, Tamil Nadu, INDIA¹sk@stylesen.org, ²balakannansp@gmail.com

ABSTRACT

Internet of Things (IoT) is enhancing the way in which humans co-exist with machines and stresses each field of computer science due to its prolific nature. There are billions of devices that are getting on boarded and its exponentially growing year on year [1], which demands the best Quality of Service (QoS) and tight security requirements. Security takes prime importance when it comes to IoT since huge amount of personal data is being exposed via each device that gets on boarded to the internet and it is an important topic of research [2]. Security in the era of IoT should come in simple form but without compromising on highest possible quality. Edge computing or fog computing [3] is getting popularity because of IoT and the traditional or legacy technologies should be enhanced to cope up with the devices that get deployed on the edge. Containerization is the new light weight virtualization [4] which could be applied to any computing device. This paper discusses the use of containerization on edge computing devices to enhance security of the IoT devices. The edge devices provide more than one service as part of the IoT ecosystem and they are an important piece to enable QoS guarantees in the demanding IoT scenario. We will explore an architecture that can be used by Original Equipment Manufacturers (OEM) and other device vendors in order to enhance security of these IoT devices using containerization.

Key words-internet of things, iot, containers, security, edge computing, fog computing, device

I. INTRODUCTION

Internet of Things (IoT) is getting adopted at scale and increases the number of devices that participate in this ecosystem [1]. Due to the proliferation of devices there is a need for better and innovative way to administer and secure the participating devices. There is fast paced development in the field of computer science and associated research to cater this increasing demand posed by IoT.

Security is a prime concern when it comes to IoT since there is huge data that is being consumed and floated as part of the IoT ecosystem [2]. OEMs are considering new ways in order to secure the devices with light weight solutions considering the form factor and other resource constraints that exists in the IoT devices at large. In this paper we will discuss architecture to secure IoT devices based on containers, focusing specially on the edge devices that exist on the fog.

II. LITERATURE SURVEY

Fog computing [3] is a concept that emerged in the recent past in order to deliver the Quality of Service that is required for IoT. Fog computing is mainly the localization of services and associated content that is required by the terminal devices that exists in the IoT. The traditional devices such as switches, routers and other networking equipment are the candidates for fog computing also known as edge computing devices or simply edge devices. This paradigm is driving the Original Equipment Manufacturers (OEM) to add innovative methods to deliver their devices in the fog that is deployment ready for IoT.

Containerization is a concept in which that same host operating system deployed in a device can create isolated environments ranging from a single process to a complete full-fledged operating system. It came into existence right from the main frame days and was carried away by heavy weight virtualization technologies [6].
are considered as light weight virtualization technique [4] which is supported by the operating system kernel. In the Linux kernel [5] containers where first possible with the advent of cgroups [7] (control groups) implementation that was introduced in version 2.6.24 [8]. The first container implementation in Linux kernel was known as LXC [9] or Linux containers. This containerization became very popular in the year 2012 when the same LXC implementation was re-mastered as Docker [10]. Docker pioneered the concept of Linux containers and made it simple to use and deploy with the masses. Containers can work from small devices to large servers deployed in the data centers. Different operating system versions or distributions can be supported within the same host operating running on a hardware or device. In most of the cases the containers use the same underlying kernel as the host operating system but provide strict isolation between the environments of the containers.

2.1. PURPOSE

IoT devices are mainly resource constrained with less processing power, low memory, stringent power requirements and network capabilities [11]. Due to its constrained resources we cannot employ the same technologies which are used in full-fledged computing devices such as desktops, servers, etc. Each implementation of a service in an IoT device should take into consideration the available resources in the device and impose stringent techniques that will ensure maximum performance from these resource constrained devices. Traditional security algorithms cannot be used directly in order to secure IoT[12] devices due to the huge computing demands imposed by these algorithms which makes the device power hungry, which is a luxury as far as IoT devices are concerned.

In the same sense, traditional virtualization methodologies cannot be used in IoT devices due to their heavy weight nature. Containers are a perfect match for this situation which can provide isolation within these resource constrained IoT devices. When there are billions of IoT devices deployed in future, it always becomes important to keep them secured and updated in order to protect the privacy of users who are part of the IoT ecosystem. In traditional deployments we use a pull model [13] in order to deliver software updates to the hardware, which may not be the right thing to do in case of IoT devices, since these devices may not be monitored by humans or we cannot as the users to keep their devices up-to-date with the security fixes. With a fast-paced environment, updates to software should happen in a really quick way and should be propagated to all the devices in an efficient manner. Thus, push model [13] for software updates are the way to go in order to deliver updates and keep the proliferated IoT devices in good and secured condition.

Most of the IoT devices are also connected via wireless communication channels, rather than with wired network communication. Over the Air (OTA) updates [14] are a way in which all the software and firmware updates to the devices are sent over a wireless mechanism. The OTA provisioning follows a push model where all the participating devices will be subscribed to the OEMs infrastructure to receive these updates. Updates can be sent on a hourly, daily, weekly or monthly basis which is quite different from how big server hardware used to get updated for software. When the updates are sent so frequently, caution must be taken in order to not break the devices to which these updates are being sent. There could be updates that are sent in small chunks which does not necessarily change all the services but fixes a small bug in one of the services deployed in the device. In such a scenario instead of disturbing other services running on the devices during an update, we should have a mechanism to update that specific service to which the bug fix applies without interrupting other services. These are the type of scenarios where our proposed architecture in this paper will help the IoT devices to stay updated and secured by employing containers within these device environments.

2.2. SCOPE

The traditional networking equipment such as switches, routers, firewalls; proxies, etc. are taking up extra services in the fog computing era in order to provide more services within the IoT environment. For example, a switch can also provide additional service such as content caching within the network in order to make localized content readily available without the need to reach the cloud servers. The main objective of this paper is to secure these edge devices that are overloaded with unusual service. In order to make such an edge device secure we will propose an architecture based on containers. The container technology could be anything that the device manufacturer or OEM is comfortable with and see a value in adopting it for the needs of the device. Some popular containerization technologies that are popular are listed below:

- LXC
- LXD

www.turkjphysiotherrehabil.org
Any container technology could be used for this purpose, provided the underlying host operating system supports it. Similarly, this paper proposes the architecture without making any explicit preference to host operating system, any operating system could be used within the host which is supported by the IoT device based on its hardware configuration.

The architecture proposed in this paper will be a generic design choice that could be adopted in order to create a secured edge computing device that will be used in the IoT ecosystem, with the following parameters in consideration:

- Service Isolation
- Host operating system isolation
- Easy updates without breaking operation of the device
- Overloading services within the same edge device

III. ARCHITECTURE

The overall architecture of securing IoT device with containerization is shown in Figure 1. Containers are lightweight virtualization technology that provides performance equal to the one observed in bare metal hardware with minimal or no overheads [15] [16]. On top of the device hardware we have the host operating system which is the base operating system on top of which we deploy the containers. The host operating system is hardened and will poses only the basic functionality required to bring up the device and the associated hardware. The host operating system will not run any other services apart from running the basics to bring up containers within its environment.

As seen in Figure 1, each service that must be run within the edge device can be run within a container. A container will have its own isolated operating system environment that is not related to the host operating system, except for sharing the host kernel depending on the container technology. Each container will have its own system libraries and binary utilities in order to run the service. Let us discuss the advantages and methodology of using this architecture on edge devices.

![Figure 1: Architecture for securing IoT / Edge devices](image)

- Isolation

The major advantage of using containers is isolation [17], where each process or service run within the container is independent and can be started and stopped without affecting other services running on the host operating system. Containers also provide process and data isolation which helps to secure the
service running on the container [17] from being affected by a bug that exists in other service that runs on a different container within the same device.

The service isolation within containers also brings in a huge security advantage of not sharing process space and memory. This helps in accidental overwriting and loss of data across services that are isolated within containers.

- **Host Operating System**

The host operating system is a minimal operating system image, which most of the time will not directly get involved in a network communication but acts as a base for bringing up the containers. The host operating system could be considered as a stringent firmware without much functionality that will be updated very rarely and runs stable at all time in order to keep the device up and running. In modern edge devices these host operating system updates are delivered on a separate partition. Once the update is received it gets loaded to a special staging partition. The boot loader then switches to boot from this staging partition and the host operating system is booted. Once the boot process is complete and there is no errors observed, the current staging partition is interchanged as the actual partition which the host operating system is loaded to. In case of any errors, the old partition which holds the old host operating system image is booted by the boot loader, thereby bringing up the device without affecting the operation of the device.

Thus, having two partitions that could be interchanged based on the arrival of a new image for host partition helps in bringing up the device or reverts back without breaking the device, due to an error in the update. This is similar to the OTA update mechanism seen in Google Pixel [18] and other android based devices that helps in not breaking the device itself [19]

- **Updates**

Over the Air (OTA) [14] is the common update mechanism that could be used in this architecture, where updates could be delivered for each container that is running in the device. There are the following possibilities that could be achieved with such a mechanism:

- Update sizes are very small
- Any number of updates could be sent in any frequency
- Targeted updates for specific services
- Targeted updates for security fixes
- Updates for changing dependent packages for specific services
- Transactional updates with a database to monitor
- Updates that does not require restart of the entire device
- Quick restarts after updates are installed since they are deployed in specific containers

Updates to the device services could be delivered at any point of time without affecting the operation of other services. There could be checks to ensure that the updates are not deployed when the service is doing some critical operation which could be programatically detected and ensure the updates are deployed only after the critical operation is completed by the service. By maintaining a transactional log of each of the updates and the corresponding patches, we can ensure that if an update is not working as intended or is breaking a service, we can revert to the previous patch or image of the container, so that the service is not down or getting affected.

Containers are light weight virtualization and hence booting or starting a container and the associated service within the container is going to take few seconds or even milliseconds depending on the service.
that is getting started. This ensures quick restarts to services neglecting long delays and keeping the service down for a long time.

IV. DEVICE BENCHMARK AND RESULTS DISCUSSION

The basis of this paper comes from the idea that containers provide light weight virtualization within the device. In the Architecture section above, we discussed the various methodology to enhance security using containers within the device. In this section we will conduct an experiment to prove that containers provide light weight virtualization which does not hinder the performance of the device, with the help of Dragonboard 410c [22] which is a popular ARM [20] based development board that is widely used to develop edge and fog computing solutions.

<table>
<thead>
<tr>
<th>TABLE – 1DRAGONBOARD 410C CONFIGURATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>SoC</td>
</tr>
<tr>
<td>CPU</td>
</tr>
<tr>
<td>Cores</td>
</tr>
<tr>
<td>Clock Speed</td>
</tr>
<tr>
<td>Instruction Set</td>
</tr>
<tr>
<td>RAM</td>
</tr>
<tr>
<td>eMMC</td>
</tr>
<tr>
<td>Storage</td>
</tr>
<tr>
<td>Ethernet</td>
</tr>
<tr>
<td>Wireless</td>
</tr>
<tr>
<td>Bluetooth</td>
</tr>
<tr>
<td>GPIO</td>
</tr>
<tr>
<td>Price</td>
</tr>
</tbody>
</table>

The configuration of Dragonboard 410c is shown in Table 1. The Dragonboard 410c is installed with Debian [21] GNU/Linux operating system with Linux kernel [5] version 4.13. LXC was used as the container technology in order to run the benchmarking. The benchmarking tool sysbench [23], is a standard benchmark which evaluates the performance of Linux machines. Measurements were taken using sysbench on bare metal and within the LXC container, for the following parameters. Each benchmark was run four times and the averages of the four runs were taken as the result, which are presented and discussed in this section.

1. Memory
2. CPU
3. Input Output (I/O)

<table>
<thead>
<tr>
<th>TABLE – 2DRAGONBOARD 410C BENCHMARK COMPLETION TIME IN SECONDS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parameter</td>
</tr>
<tr>
<td>Memory</td>
</tr>
<tr>
<td>CPU</td>
</tr>
<tr>
<td>I/O</td>
</tr>
</tbody>
</table>

Table 2 shows the benchmark completion time in seconds. Based on the results shown in Table II we can see the difference in time (in seconds) for the execution of sysbench on parameters such as Memory performance, CPU performance and I/O performance are negligible or in some cases similar. Thus, from this experiment we can clearly see there is no performance overhead in using containers within a device and the results are like running a bare metal device without any containers involved. This helps in understanding that by deploying the architecture shown in Figure 1, within a device will help in enhancing security of the device with minimal or no performance overheads.

V. FUTURE DIRECTIONS

An edge computing device that runs more than one service such as IPv6 assignment, switching, content caching, wireless configuration and communication, Zigbee / Bluetooth protocol communications, etc. should be

www.turkjphysiotherrehabil.org 7964
implemented on a IoT development board such as Dragonboard, RaspberryPi, HiKey, etc. to demonstrate the effectiveness of this proposed architecture with containers. The implementation on this development board should also show OTA updates that could be done for each of these services independently and make use of all the advantages listed in the above sections. A performance analysis should be done to benchmark the overall operation of the edge device, thus created.

VI. CONCLUSION

The principal aim of this paper was to showcase an architecture that can take advantage of latest containerization technologies in order to deliver the required Quality of Service (QoS) on the edge computing devices that are part of the IoT ecosystem, thus by ensuring service level isolation and security. The host operating system will always be stable and always kept small with basic functionality in order to ensure availability of the edge device, even though some of the services may get affected due to a faulty or erroneous update. This paper will also stand as reference model for researchers and OEMs to deliver IoT edge devices that can be overloaded with as many services they can depending on the resources available on the device with minimal overhead for isolation and security implementation.

REFERENCES

10. Docker. https://docs.docker.com/
19. A/B (Seamless) System Updates https://source.android.com/devices/tech/ota/ab
22. Dragonboard 410c https://www.qualcomm.com/products/dragonboard-410c