Virtualization Construction of Security Components of Edge IoT Agent Based on Security Requirements

Yuanyuan Ma¹, ², Mu Chen¹, ², *, Tao Zhang¹, ², *

¹Institute of Information and Communication, Global Energy Interconnection Research Institute, Nanjing Jiangsu, 210003, China
²State Grid Key Laboratory of Information & Network Security, Nanjing Jiangsu, 210003

mayuanyuan@geiri.sgcc.com.cn, henmu@geiri.sgcc.com.cn, zhangtao@geiri.sgcc.com.cn

Abstract—The IoT edge agent is located in the perception layer of the electric power Internet of Things, and uses the edge computing power to realize the data collection, processing and protection management of the Internet of Things terminal equipment. The traditional edge IoT proxy security solution relies heavily on security equipment and a closed service delivery method. The software-defined security-based virtualization technology can separate the hardware platform from the security function, and can separate the required security components according to changes in the field environment. The image is quickly deployed on the edge IoT agent access platform. However, once the security requirements or operating environment change, new security component images need to be regenerated and redeployed to the edge IoT agent hardware platform. The distribution of the image takes a lot of time, which makes the startup process of the container may take a few seconds or a few minutes, so that it cannot respond to change quickly in a changing application environment. This paper proposes a method for virtualizing the construction of security components of edge IoT agents based on security requirements. This method first creates a core container image with the minimum operating environment of the security components. On this basis, the operating environment requirements and corresponding security requirements are formally modeled Generate a build strategy, use multi-layer container images to build corresponding security components and complete deployment according to the build strategy. Tests show that this method can shorten the deployment time to milliseconds.

1. INTRODUCTION

The power Internet of Things refers to the wide deployment of smart devices with certain perception capabilities, computing capabilities and execution capabilities in power production, transmission, consumption and management of various links to achieve reliable information collection, secure transmission, collaborative processing, unified service and application integration, to promote the panoramic holographic perception of the entire process of power grid production and operation and enterprise management, information fusion and intelligent management and decision-making in the Internet of Things [1]. Its overall architecture includes perception layer, network layer, platform layer and application layer. Security protection runs through four layers: the perception layer, the network layer, the platform layer, and the application layer. The perception layer completes data collection, local communication, data collection, edge computing, data storage and other functions; the network layer...
completes network data communication functions; the platform layer completes connection management, device management, edge computing management, application management, and message processing. The application layer completes data service and business service functions.

The edge IoT agent is located in the perception layer of the power Internet of Things, which realizes the communication convergence of the perception layer, edge computing, regional autonomy, and can receive the control commands and configuration information issued by the platform layer [2]. The edge IoT agent uses the device's local communication interface to access and manage all kinds of sensors, terminals and other devices, extracts the business data through the protocol analysis, aggregates and stores, and performs standardized modeling according to the requirements of the object model. The business data is sent to the platform layer after processing. According to the "Basic Requirements for National Network Security Level Protection" requirements, it is required to provide security protection at the edge borders of the Internet of Things to ensure the security of the network structure, reasonably divide the network security domain, and strengthen the security boundary isolation [3]. In response to the above requirements, software-defined security is introduced at the edge IoT agent layer of the existing ubiquitous power IoT architecture system. By separating the core concept of the control plane and the data plane, and using the network protection capabilities provided by software-defined security to Make up for the hidden dangers caused by the limitations of equipment resources and insufficient capacity.

The widespread application of software-defined networks, its core concepts of "functional separation at the architectural level" and "software-based flexible deployment" have been extended to other areas [4]. Gartner first proposed "software-defined security" in 2012. Its core concept is to separate the security data plane from the control plane, decouple physical and virtual network security devices from their access modes, deployment methods, and implementation functions. The physical and virtual network security equipment is abstracted as a security resource pool, and the top layer is unified through software programming to intelligentize and automate business orchestration and management to complete the corresponding security functions, thereby achieving a flexible security protection mechanism [5]. Early research mainly relied on software-defined network architecture to achieve security protection. For example, AMQ relied on SDN controllers to actively isolate and protect abnormal devices and automatically update patches [6]. However, this solution is only applicable to static data flows and requires real-time monitoring of ingress traffic. It cannot be applied to the application scenarios of edge IoT proxy load virtualization and high data flow dynamics. FRESCO is also a software-defined security solution applied to SDN controllers [7]. Security functions are deployed on the SDN controller in the form of modules. These modules mainly implement functions such as packet analysis and data flow redirection. Its architecture design It makes the security application overly dependent on the controller environment, it is difficult to migrate and reuse, and may cause conflicts between upper and lower security policies. Avant-guard is a solution that uses data plane expansion to implement security functions [8]. The solution extends the OpenFlow protocol to solve the inherent communication bottleneck between the data plane and the control plane. It also enables dynamic detection and response to IP dynamic data flows, but it does not consider virtualizing security components and has not implemented it. The real hardware platform is separated from the security function.

The existing software-defined security scheme separates the hardware platform from the security function through virtualization technology, and can quickly deploy the required security component images on the edge IoT agent access platform according to the changes in the field environment. However, once the security requirements or operating environment change, new security component images need to be regenerated and redeployed to the edge IoT agent hardware platform. The distribution of the image takes a lot of time, which makes the startup process of the container may take a few seconds or a few minutes, which makes it impossible to respond to and change quickly in changing application environments [9]. This paper proposes a method for virtualizing the construction of security components of edge IoT agents based on security requirements. This method first creates a core container image with the minimum operating environment of the security components. On this basis, the operating environment requirements and corresponding security requirements are modeled to generate security component construction strategies to generate security component instantiations for different application scenarios.
2. OVERALL ARCHITECTURE

The overall architecture of IoT terminal security layer protection based on software-defined security includes security application, security control platform and security hardware resource pool, as shown in Figure 1. The three layers can communicate with each other through a set of interfaces, southbound interface and northbound interface. The security control platform interacts with the security hardware resource pool through the southbound interface and interacts with the security application layer through the northbound interface to complete security management and scheduling based on software-defined security.

![Diagram of IoT terminal security layer protection architecture based on software-defined security](image)

The security control platform is its core component, which is mainly responsible for resource pooling management of security equipment, collection and analysis of various security information sources, interfacing with customer business systems, and policy analysis and execution of corresponding security applications.

The security application layer is composed of programs developed according to specific security requirements. It uses the open API of the security control platform to implement the corresponding security functions.

The security hardware resource pool is deployed at the edge IoT agent layer. It consists of a security component resource pool and network function virtual equipment. Various resource pools are formed through the management and scheduling of the security control platform to provide external security capabilities.

The security component resource pool is composed of virtualized security components. Use container technology to virtualize the security services on the original security devices and isolate them. Virtualization uses container-based lightweight virtualization technology. The lightweight virtual machine can directly load and run applications on the host platform to share the core of the host machine, and the performance loss is relatively small. At the same time, during the operation process, the complexity caused by the alternation of system calls is avoided. Lightweight isolation method can achieve resource sharing while providing isolation mechanism. Virtualized security functions mainly include security access, security access and security monitoring:

2.1 Container technology

Docker is a lightweight virtualization technology that implements virtualization at the operating system level and provides a resource pooling solution that is completely different from traditional virtualization technologies. In the early days, Docker used LXC, the Linux Container technology, to isolate various resources in the system, thereby building a closed sandbox within the system, and different sandboxes
were isolated from each other. The sandbox is also called a container. Many containers in the host machine share a system kernel. Due to the existence of the isolation mechanism, the container thinks that it is in an independent host. The container in Docker corresponds to the virtual machine in the traditional virtualization technology. There are many common points between them, that is, the redistribution and isolation of host resources, but there are essential differences in the way of implementation. In the Docker virtualization mode, all containers share a system core, and the Docker container technology provides a certain degree of isolation for the containers.

2.2 Joint File System

The Joint File System is a lightweight, high-performance, hierarchical file system [50], which is the technical foundation for implementing Docker images. The joint file system supports the submission and layer-by-layer stacking of changes in the file system, and supports mounting different directories under the same virtual file system. This feature enables imaging to be inherited through layers.

The joint file system uses the copy-on-write technology COW (copy-on-write). When a user modifies a file with read-only permissions, the user does not modify the original file, but the modified content is saved to a user-readable and writable file system. The joint file system mounts read-write files and read-only files in a unified virtual directory, and each file modification of the user is submitted as a single submission, which is superimposed layer by layer to shield the user from the underlying implementation of the file system. The user experience is still in a separate file system.

The joint file system generally includes a read-write layer and a read-only layer. Each layer may be composed of multiple layers. The read-only layer is a file that needs to be protected or does not want to be modified, such as a key file related to the operation of the system or a file containing important data. The read-only layer can be shared by multiple containers, and the read-write layer is unique to the container.

2.3 Container multilayer image

Multi-layer imaging is formed by superimposing multiple imaging layers, and each layer records a layer of imaging information through a certain type, similar to a linked list structure. Finally, through the joint file system, multiple read-only layers are merged, and finally a unified view is provided to the user. Multi-layer imaging can be realized based on the joint file system. Copy-on-write refers to "copying" only when "write" is needed, mainly for scenarios where files are modified. In the system image, if the same image is used to start multiple containers, you can not copy the files in the image when the container starts, but all the containers share the file system in the image. When a container needs to modify a file, it copies the file to be modified to its own separate file system, but only saves a copy. Write-in copying can reduce file copy operations, improve disk utilization, and ensure the security of the original image. Allocation on write refers to the allocation of space when a new file needs to be written. In the system imaging scenario, the container does not need to allocate space to it when it is started, but allocates space when the container needs to create a new file. Allocation on write can improve the utilization of storage resources.

Each layer of image can be an independent file system, and each layer only contains different files from the bottom layer, which are stacked in sequence. The bottom layer is the core image, which contains the minimum environment required for system operation. During instantiation, an additional read-write layer is added to the core image, commonly referred to as the "container layer". All file system changes (such as file writing, modification, and deletion) are recorded in the container layer. The upper container layer consists of the docker image and the docker container. If you delete the docker instance, the container will also be deleted, but the image itself is not affected. Different containers instantiated from the same image can access the same bottom layer. The containers have their own container layers and are responsible for all corresponding storage changes and states. The conversion relationship between the container and the image is: the container can be directly converted into an image, and the image cannot be directly converted into a container unless a new read-write layer is added.
In the power IoT edge IoT agent environment, the containers derived from the images of different security components under the software-defined security architecture are usually stored on different hardware platforms, and the security control platform stores and distributes the images through the security component resource pool. When a new security application needs to be deployed on the edge IoT agent platform, its corresponding security component image will be distributed to the corresponding hardware platform. To speed up the distribution process, hierarchical copying techniques are often used to avoid loading duplicate layers. According to the image distribution strategy, when loading different images, the unchanged layer does not need to be reloaded, and only the layer that needs to be changed can be completely reloaded. But even so, it is difficult for developers to achieve layered optimization processing, and small environmental differences will still cause developers to generate huge images. In response to this problem, this paper proposes a container image generation method based on security requirements. This method first formalizes the modeling of the operating environment and corresponding security requirements, and then obtains the image generation optimization strategy, thereby reducing the size of the image that must be generated. This method can reduce the size of a single Docker image to achieve rapid deployment.

3. FORMAL MODELING BASED ON SECURITY REQUIREMENTS

3.1 Security requirements and attributes of the operating environment

In order to adapt to different field application scenarios, you first need to define the attributes of the image generation strategy: 1) Image data size and repetition rate; 2) Field environment: security component resource pool and edge IoT agent connectivity, network speed, and overall network Put on; 3) Disk read performance: the read speed of the image in the security component resource pool and the write speed of the storage unit of the IoT edge agent hardware platform; 4) CPU performance: the operation of the CPU on the edge IoT agent gateway hardware platform Speed and workload. 6) Security function: the required security function and its corresponding code file size.

3.2 Formalization of security requirements and operating environment

In order to achieve fine-grained security component image generation, first of all, it is necessary to formally express according to the security requirements and operating environment of the deployment site. The purpose of the formal expression is to establish the relationship between user needs and the security component image through modeling and analysis. Mapping relations. This paper mainly logically abstracts factors such as security requirements and data flow in the operating environment, security application time, user identity and role, environment and scenario, and trust relationship.

Semantically defined, data flow refers to the communication data in the running scenario of the edge IoT agent, abstracting it as a point-to-point data communication process between the source and sink; the security application time refers to the specific security application specified by the security rules Operating time, duration, and time interval between applications; user identity and role refer to the access user’s own identity attributes and the security rights and responsibilities that the user may have in a specific network area, and the user entity The type of role and the level of security level; the environment and scene refers to the network location and operating environment where the edge IoT agent is located. Different locations and different scenarios may put forward different requirements for security strategies. The trust relationship refers to the trust relationship between users and users, between users and networks, and between users and edge IoT agents. This trust relationship will directly affect the formulation of security policies.

In this paper, time conditional operation rules (ECA) are used to extract abstract generation strategies and generate detailed steps. Algorithm 1 generates operation rules based on the abstract security requirements and operating environment. The security policy generator continuously monitors the ECA rules instantiated from the template. Whenever an event of a specific pattern is observed so that all conditions of a specific combination are met, it will execute the corresponding Authorized operation.
Algorithm 1 Event condition operation rules

Input: E: Event pattern collection
Input: C: User-specified conditions
Input: D: User-specified delay
Input: T: Trust type
Input: n: Number of images corresponding to security components
Output: R: Rule template

\[
T \leftarrow T_{\text{Indirect}} \cup T_{\text{Direct}}
\]

\[
D \leftarrow D_{\text{Amount}} \times D_{\text{Unit}}
\]

for \( i \in \{1, 2, \ldots, n\} \) do

if \( Ti \) == true then

if \( Ci \in \text{PE} \) then

\( Ri \leftarrow Ci \dashv Di \)

end if

else

\( Ri \leftarrow \text{deny} \)

end if

end for

return R

3.3 Image generation algorithm

For an abstract security component container set \( A \), define the imaging instance set associated with the set as \( D_A \), define the security requirements and the operating environment constraints on the container set \( A \) as \( R_A \), \( R \) is the interaction rule generated by the algorithm. The final result of such an interaction model enables the declared image instance data set to be used for all security requirements and operating environments applicable to container set \( A \).

For an abstract security component container set \( A \), it can be further decomposed into a series of specific security functions \( I_k \).

The file system instance defined for this specific interaction is \( D_k^I \). Then, there is the following relationship between the specific file system instance \( D_k^I \) and the imaged instance set \( D_A \):

\[
D_A \in \bigcup_{k=1}^{n} D_k^I
\]  

Equation 1 shows that a specific set of imaging instances must establish at least the same set of imaging instances as the abstract set of security component containers. At the same time, all collections of image instances contain core container images corresponding to the minimum operating environment. Therefore, it is necessary to meet:

\[
I_{\text{core}} \in I_k
\]  

In the security policy rules, the event patterns in the ECA rules represent the security requirements and operating environment. In the ECA rules, the event part \( E \) can only contain 1 security function, while the condition part \( C \) may be composed of several security functions using policy operators. If the operation part \( A \) contains a certain safety requirement or operating environment requirement, the corresponding safety function is defined in the event part \( E \).

4. MULTI-LEVEL CONTAINER IMAGE CONSTRUCTION

Designing and implementing multi-layer container images in Docker’s existing templates can implement security component image management. The core container image is a read-only template that contains the file system. The user creates a container based on the image, and a running container is a running instance created from the container image. In order to facilitate storage and update of the image, the implementation of the image is based on a layered structure of the joint file system, that is, the image is actually composed of layers of file views. as shown in figure 2
Multi-layer image mountain is formed by stacking multiple image layers, and each layer records a layer of image information through a certain type, similar to a linked list structure. Finally, through the union file system (Union File System), multiple read-only layers are merged, and finally a unified view is provided to the user. The joint file system supports Copy on Write (COW) and Allocate-on-Demand (Aod). The joint file system is a hierarchical file system. Simply put, it is a file system that provides different users with a unified view by combining different directories [6]. The joint file system is implemented on the basis of the existing file system. It does not need to be formatted and other processing, just need to mount it to use. It can realize layer-by-layer stacking of the file system, and use it in the image to achieve layering.

Copy-on-write refers to "copying" only when "writing" is needed, mainly for scenarios where files need to be modified. In the system image, if the same image is used to start multiple containers, you cannot copy the files in the image when the container starts, but all the containers share the file system in the image. When a container needs to modify a file, it copies the file to be modified to its own separate file system, but only saves a copy. Write-in copying can reduce file copy operations, improve disk utilization, and ensure the security of the original image. Allocation on write refers to the allocation of space when a new file needs to be written. In the system imaging scenario, the container does not need to allocate space to it when it is started, but allocates space when the container needs to create a new file. Allocation on write can improve the utilization of storage resources.

The multi-layer container image architecture of the edge IoT agent is shown in Figure 3. The file system required by the bottom layer of the system is made into a bottom layer core container image layer, and then the security components are made another layer of the security component image layer. Together form the image layer. Then use the writable folder as the container layer, and the container layer is writable. Finally, after the joint file operation, it is the container layer file system that is finally presented to the user.

![Diagram of multi-level container image architecture](image_url)
5. TESTING AND ANALYSIS
In the test, the front-end hardware edge IoT agency deployed an embedded development platform Raspberry Pi 4 based on the Arm processor, the processor is 1.5GHz Broadcom BCM2711 (quad-core Cortex-A72), and the onboard memory capacity is 4GB. 64G storage space. The security control platform uses 2 blade servers, the CPU is Intel Xeno E5645 CPU, 4TB hard disk and 128GB memory. One of them is used as a controller and the other is used as a security component resource pool. The security component image storage is a disk array with 4TB storage space. All devices are connected via Gigabit LAN.

We selected 5 different types of security component images as the test object, each component size is between 100MB and 500MB. The security component images required for different application scenarios and security requirements are independently expanded by means of simulation upgrade. For each component, we simulate 5 newer versions. We consider the 5 types of original images and all their updated versions as the same family series. The data change rates of different images in the same series are 10%, 20%, 30%, 40% and 50%. In the experiment, we first tested the loading time of the original image and the updated image. The statistical results are shown in Figure 4.

Fig 4. Load time of different images

The horizontal axis of Figure 4 represents the updated versions of the original images of the five security components and the corresponding updated versions, and the vertical axis represents the time of deployment. It can be seen from the way that when the security requirements and operating environment change, the deployment time of the updated image is greatly reduced compared to the original image deployment time. When the probability of updating the data is only 10%, the deployment time can be shortened to a few tens of milliseconds greatly speeds up its ability to respond quickly.

6. CONCLUSIONS
This paper aims at the problem that the existing virtualization technology cannot quickly generate an updated image and dynamically load it when the security requirements or operating environment changes, and proposes a method for virtualizing the construction of edge IoT proxy security components based on security requirements. The minimum operating environment of the security component creates a core container image, on this basis, the operating environment requirements and corresponding security requirements are formally modeled to generate a construction strategy, and the multi-layer container image is used to construct the corresponding security component according to the construction strategy and complete the deployment. Tests have proved that the new method can shorten the deployment time to milliseconds compared to the time spent in the original image distribution deployment.

ACKNOWLEDGMENT
This work was supported by the Science and Technology Project of State Grid Corporation of China: "Research on Key Technologies of Fundamental Security Protection and Control for IoT Smart Terminals" (Grants No. SGGR0000XTJS1900905).
REFERENCES
[1] Yousefpour, Ashkan, et al. "All one needs to know about fog computing and related edge computing paradigms: A complete survey." Journal of Systems Architecture 98 (2019): 289-330. J. Clerk Maxwell, A Treatise on Electricity and Magnetism, 3rd ed., vol. 2. Oxford: Clarendon, 1892, pp.68-73.
[2] Wang, Xiaofei, et al. "Convergence of edge computing and deep learning: A comprehensive survey." IEEE Communications Surveys & Tutorials 22.2 (2020): 869-904. K. Elissa, “Title of paper if known,” unpublished.
[3] Tao, Zeyi, et al. "A survey of virtual machine management in edge computing." Proceedings of the IEEE 107.8 (2019): 1482-1499. Y. Yorozu, M. Hirano, K. Oka, and Y. Tagawa, “Electron spectroscopy studies on magneto-optical media and plastic substrate interface,” IEEE Transl. J. Magn. Japan, vol. 2, pp. 740-741, August 1987 [Digests 9th Annual Conf. Magnetics Japan, p. 301, 1982].
[4] Puliafito, Carlo, et al. "Container migration in the fog: A performance evaluation." Sensors 19.7 (2019): 1488.
[5] Gartner. Build Adaptive Security Architecture You’re your Organization. https://www.gartner.com/smarterwithgartner/build-adaptive-securi-architecture-into-your-organization.
[6] M. McBride, M. Cohn, S. Deshpande, M. Kaushik, M. Mathews and S. Nathan, SDN security considerations in the data center, Menlo Park, CA, USA, 2018.
[7] Shin, Seung Won, et al. "Fresco: Modular composable security services for software-defined networks." 20th Annual Network & Distributed System Security Symposium. Ndss, 2018.
[8] Shin, Seungwon, et al. "Avant-guard: Scalable and vigilant switch flow management in software-defined networks." Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security. 2019.
[9] Z. Lu, Y. Wu, J. Xu and T. Wang, "An Acceleration Method for Docker Image Update," 2019 IEEE International Conference on Fog Computing (ICFC), Prague, Czech Republic, 2019, pp. 15-23.