Abstract. At present, embedded devices have been widely used in people’s daily life, which makes more convenience for the public. However, embedded devices still have security problems, such as automatic driving accidents that may cause casualties. In the field of embedded device security, there are many studies, for instance, OPENTEE for ARM handheld devices, providing a secure execution environment for payment devices, and SGX for Intel desk top devices, for security-critical applications, such as bank teller systems, build a safe operating environment. But it is a lack of correlation between these new and existing technologies. In our opinion, through the combination of mature technology accumulation and embedded devices, the antivirus industry can create a more secure user environment. In this paper, we propose a source isolation method to make the resources used by key processes exclusive. This method uses antivirus software and builds a more comprehensive embedded security system in critical security applications. The experimental results show that the proposed method is effective and safe.

Keywords: Embedded system · Security-Related · Resource isolation · Virus

1 Introduction

Embedded devices are becoming more and more popular in people’s daily life [11], from aerospace, submarine missiles, to TV phones, watches and earphones, etc. They are everywhere, and their roles are becoming more and more important. For example, to reduce vehicles Automatic braking system for security risks.

An embedded device is usually composed of one or more hardware main bodies, and there is usually an embedded system specially developed for hardware running in the embedded device, and the system runs specifically for the device. These programs are divided into critical tasks and non-critical tasks according to different functions. For example, in the on-board embedded system, the automatic braking system is a critical task [14], and the on-board imaging system is a non-critical task. The execution of the critical task must be guaranteed, and it must be completed within the specified time from response to execution and completion. Otherwise serious consequences will occur, and non-critical tasks will be delayed for a few cycles without serious consequences. This article mainly focuses on the security issues of embedded systems with very high penetration rate-on-board systems [13].

© The Author(s) 2020
W. Lu et al. (Eds.): CNCERT 2020, CCIS 1299, pp. 222–234, 2020.
https://doi.org/10.1007/978-981-33-4922-3_17
The original in-vehicle systems were not connected to the Internet. They usually included imaging systems, air conditioning and ventilation systems, etc. In the disconnected era, people did not have an intuitive understanding of the consequences of such devices being controlled by hackers. However, with the development of in-vehicle systems, for example, Tesla and other electric vehicle manufacturers have taken the in-vehicle systems as their main selling point. They are more humane and smarter, and even allow the driver to let go of his hands and let the in-vehicle system replace people to realize autonomous driving. Such systems are usually networked, and they are connected to non-specially constructed general-purpose networks, so the vulnerabilities in the system will be easily enlarged. According to the description in [2], such systems are more resistant to network security. Poor, and the particularity of the system makes it impossible to use common system security defense measures [3], what will happen after being controlled by hackers? At the Def Con hacking conference in August 2015, hackers claimed that they had 6 ways to control Tesla’s vehicle and make it stop. Just imagine if the vehicle happened to be driving on a highway and suddenly stopped, it is likely that there will be more cars connected. Collision, causing immeasurable losses.

Do we could simply install anti-virus software in the vehicle system to solve these problems [9]? the answer is negative. Because embedded devices have relatively large limitations in storage capacity and computing performance [1], and modern on-board systems are a very complex hybrid system, they have as many as 80 control units on average [10]. Distributed in the management of multiple key tasks in the system, the operation of anti-virus software will inevitably consume storage, and may also lock up the resources occupied by key tasks. If the execution of key tasks is affected, such as an automatic braking system, it will be slightly delayed by 1 s. Zhong, stop the car, this may have caused the car crash.

In many hardware platforms, key tasks can be placed in a specific execution environment. For example, on the ARM platform, the key tasks are executed in the security domain of OPENTEE, and on the X86 platform, the key tasks are placed in the security domain of SGX. To execute, in this article, we pass Put the antivirus software in an isolated client similar to OpenTEE or SGX to run, and allocate independent resources, such as memory, cache, etc., for this isolated client. The isolated client is built through hardware virtualization features. Hardware virtualization has a very broad foundation. It is implemented on both ARM and X86 platforms, so the cost of implementation and promotion and use are of practical significance.

2 Related Work

Virtualization technology is the foundation of resource isolation technology. This technology guarantees from the bottom layer that the upper layer virtual machines can run their own processes and perform various tasks without interfering with each other. Resource isolation mainly isolates resources are CPU, memory, network and other resources. There are two types of resource isolation according to the implementation scheme: software isolation and hardware isolation.
2.1 Resource Software Isolation

Software resource isolation technology, also known as the first type of virtualization technology, mainly isolates the resources of multiple virtual machines through software simulation. This technology has the following problems and the current research status of solving these problems:

1) The issue of the authority of privileged instructions. Before the concept of hardware virtualization, the execution of all instructions was simulated by software, including privileged instructions. The way of simulation execution greatly reduced the performance of the system.

2) Compression of the process address space. Although the process can be run effectively through software simulation, the run is based on the premise that the address space is compressed, because the process-specific address is allocated to the kernel for use. If you want to fully control the process execution, you must compress the address space. The consequence of compressing the address space is that the virtual machine’s own management program needs to reserve part of the address space to store important data structures, such as IDT, GDT, etc., through these structures to manage the running image better and easier. However, if a large number of programs are compiled before that, these compressed addresses must be accessed according to the execution mode of the operation. In order to be able to process the request, performance will inevitably decrease, and if not processed, the system will crash.

3) Interrupt virtualization. The current research on this aspect has not proposed a good method to improve the way of software simulation to improve the efficiency of the mechanism.

The first type of virtualization technology mentioned above has insurmountable performance defects due to its own architecture. As shown in Fig. 1, VMM running in the user mode is the key to this problem and cannot be improved. This architecture cannot meet the real-time characteristics of the vehicle system.

![Fig. 1. Basic architecture of resource software isolation](image-url)
2.2 Resource Hardware Isolation

At present, there are not many studies on resource hardware isolation technology for embedded devices, and most of them are concentrated in some research universities.

![Multi-core multi-operating system model](image)

**Fig. 2.** Multi-core multi-operating system model

Based on the above-mentioned embedded multi-core operating system resource isolation model, this paper proposes a client solution based on a single multi-core, supporting ARM virtualization extension features, running a complete operating system per core, and running a specific strategy for each operating system.

2.3 Current Status of Research on Resource Isolation of Vehicle System

At present, most vehicle systems use SoC systems [13], in which the processor architecture uses ARM architecture [12], which provides conditions for us to implement resource isolation and sharing. By isolating the ARM processors of the SoC system according to different usage requirements For example, the driving control system is assigned to run in an independent large core, so that non-critical tasks will not interfere with him, and the control system can respond to user control in a very short time as a key task. Isolating the multimedia system into a dedicated multimedia core can achieve a better experience without affecting key tasks such as the control system.

At present, there are very few studies on resource isolation of vehicle systems that use ARM hardware virtualization extension technology. In the paper of [16], ARM’s TrustZone is used to track and protect the system, but TrustZone cannot provide a complete system simulation. Need to make a lot of changes to the original system, it is more difficult to reform the system. In [17] the paper also uses TrustZone to provide a security zone for the system. The TrustZone method is more suitable for use in the context of processes, such as payment systems, fingerprint recognition systems, etc.

2.4 Current Status of Anti-virus Software Running on Vehicle Systems

Due to the particularity of the on-board system itself, it requires extremely high security. Few manufacturers consider security. By hardware isolation of resources, we can provide
a completely isolated environment for the control system and install antivirus the software provides an operating environment.

The client computer installed with anti-virus software is a low-priority client. It runs only when the vehicle is stopped or is being maintained or repaired. Through such a symbiosis method, it can not only provide strong security for the vehicle, but also requests for exclusive resources that would destroy critical mission.

3 Modeling of Vehicle Resource Isolation System

3.1 Isolation Model Design

Based on the hardware characteristics described in the ARM platform architecture analysis above, the following hardware isolation model of vehicle system resources is designed.

![Resource hardware isolation model](image)

The entire general model is divided into three layers, namely: 1. Hardware isolation layer; 2. VMM management layer; 3. Application program operation layer.

All operating systems on this vehicle-mounted system platform run on the user layer, and the CPU isolation problem of the operating system is completed through the ARM virtual extension mechanism, so that all user-mode operating systems have their own CPU, memory and hardware devices, which are isolated from each other. Do not interfere.

3.2 Implementation and Verification of on-Board System

In the system design, we divide the system into three layers: 1. Hardware abstraction layer. 2. The virtual machine management layer is the VMM layer. 3. The application layer is the system where the control system, multimedia system and anti-virus software are installed. This chapter mainly designs the virtual machine management layer, namely the VMM layer, which provides a complete virtual environment for various applications through the VMM layer, and assigns different permissions to different applications.
3.2.1 CPU Isolation Design

The isolation of the physical processor is the most critical design [15], because if the processor resource isolation design is not perfect, then there may be the possibility of non-critical tasks affecting critical tasks, which will greatly affect the stability and reliability of the system. A big threat, especially when used in vehicle systems, has unimaginable consequences (Fig. 4).

Most modern ARM processors are multi-core architectures, such as ARM-v7 or ARM-v8 processors that can provide virtualization extensions, most of which are multi-core processors, and each processor can run in parallel and used in isolation, So that the applications in each client are isolated from each other without interference.

The processor isolation architecture is as shown in the figure above. The VMM corresponding to each physical processor is described by a virtual machine management structure, which includes the following parameters: 1. Client status area. 2. Host status area. 3. The VM executes the control domain. 4. The VM exits the control domain. 5. The VM enters the control domain. 6. The VM exits the information domain.

Each running client is described by this abstract structure, stored in memory, and pointed to by a special register.

3.2.2 Device Isolation Design

The vehicle system has about 80 devices with an average value that need to be controlled, so the design of device isolation is also particularly important, especially the allocation of different device controls for different clients, and the static setting of priorities.

The isolation design of the entire in-vehicle equipment is less difficult, but the workload is huge, because the consistency of the peripheral equipment is poor, and various bus standards are used, such as AHB (high-performance bus), ASB (system bus) and APB (peripheral) Bus), etc., so our current isolation scheme only isolates the AHB standard, and other standards will be considered and supported later.
1. AHB Device principle (Fig. 5)

![Fig. 5. ARM’s typical AMBA bus architecture](image)

From the bus architecture in Fig. 6, AHB is in the high-speed device link. It is used by most SoC systems and is mainly used in high-performance, high-clock frequency system structures, such as high-speed RAM, NAND FLASH, etc. In the connection between DMA and Bridge, this bus is also a device bus used for critical tasks, so this isolation prioritizes the completion of device isolation on the AHB bus to ensure the reliability and availability of key tasks for vehicle equipment.

![Fig. 6. AHB Device Isolation](image)

A research statement on software reliability mentioned that errors in device drivers account for 70% or more of the entire system. The isolation of the above device drivers can also effectively reduce the system error rate and improve system reliability.
2. High-speed memory isolation

ARM has added the feature of Second-stage of translation (secondary translation) to the hardware, and supports the conversion of the physical addresses of all clients into actual physical addresses, instead of using shadow page tables (Fig. 7).

![High-speed memory isolation](image)

**Fig. 7.** High-speed memory isolation

In this paper, in the process of initializing the client by VMM, each client is allocated a completely isolated memory area. Each client maintains its own page table. When the client’s GPA (Guest Physical Address) to HPA (Host Physical Address) after the query of Address) hits, with the assistance of ARM virtualization technology, there is no need to switch to the host machine. In this way, performance can be improved and the interference between different clients can be isolated.

3.3 Test and Analysis

In this vehicle-mounted resource hardware isolation system, it mainly involves the involvement and realization of CPU isolation, peripheral equipment and memory isolation, etc. This article mainly verifies the correctness of several aspects by writing a set of test cases: 1. CPU isolation characteristic test. 2. Peripheral equipment isolation characteristic test. 3. Memory resource isolation characteristic test. 4. Performance test. The tests are in two Linux clients. The test case judges whether the expected function is completed according to the result output by the terminal.
3.3.1 Test Environment

At present, the vehicle-mounted platform can already run on the NVIDIA Jetson TK1 kit, but in order to obtain the visualization parameters for testing, this article runs the platform in the Qemu environment. In order to maintain consistency, all parameters are simulated NVIDIA Jetson TK1 parameters (Table 1).

| Type            | Version                      |
|-----------------|------------------------------|
| Kernel          | Linux 4.2.0-16-generic       |
| CPU             | NVIDIA Jetson TK1            |
| AHB Bus Frequency | 1000 MHz                    |
| Disk            | Samsung SSD 850 EVO 120 GB   |
| Memory          | DDR3 4096 MB*2               |
| QEMU            | qemu-2.3.0                   |

3.3.2 CPU Isolation Testing

The CPU isolation of VMM allocates a unique CPU representation structure in the memory for each client when the client is started. Each structure is maintained by VMM. In this way, the isolation is completed and the resource usage of each client will be mapped to this structure, and the allocated resources are represented by the following structure (Fig. 8):

```c
struct cell {
    struct kobject kobj;
    struct list_head entry;
    unsigned int id;
    cpumask_t cpus_assigned;
    u32 num_memory_regions;
    struct jailhouse_memory *memory_regions;
    #ifdef CONFIG_AHB
    u32 num_AHB_devices;
    struct jailhouse_AHB_device *ahb_devices;
    #endif /* CONFIG_AHB */
};
```

Fig. 8. Cell Structure

In the configuration of the client linux-arm-demo, there are a total of 4 cores, and the client occupies a total of 3 CPU. After enabling the client, 3 CPU outputs are obtained through lscpu output (Fig. 9).

There are 4 CPUs in total. After the client is started, 3 CPUs 1–3 are occupied.
3.3.3 PCI Device Isolation Testing

After enabling QEMU-VM, all AHB devices in all systems are traversed and output to the console, as shown in Fig. 1. Enable the client PCI-demo. The configuration file of this demo only applies for a PCI device from VMM, as shown in Fig. 2. According to the expected idea, the client successfully applies for a PCI device from QEMI-VM: 00:1b.0. After applying, use the device, and after using it, release the device, as shown in the figure below, the device is successfully returned to QEMU-VM (Figs. 10 and 11).

3.3.4 Memory Isolation Testing

Enter the address belonging to the client and return the valid result. Enter a physical address that does not belong to the client, and invalid is returned. As shown in the figure, the output result of the vra program is the same as the expected result, indicating that the memory isolation is executed successfully, and different clients can only access their assigned physical address space, but cannot access the address space of other clients (Fig. 12).
3.3.5 Install a Simple Behavior Detection Engine Testing

First run the linux-arm-demo client in VMM, which is used to run the anti-virus software runtime environment. As shown in Fig. 1, next run a behavior detection engine in the client. As shown in Fig. 2. As shown in Fig. 3, the client runs a centos image, which contains basic shell tools that can perform read and write operations. This also provides a runtime basis for the behavior detection engine that needs to be run in this experiment (Fig. 13).

Fig. 12. Memory usage

Fig. 13. VMs engine running status

The behavior detection engine can start, run, and detect. After starting the mirror, you can read and write files in the mirror. It can be seen from Fig. 3 that the behavior detection engine can run the program normally, download the centos image and start it, and the image contains the engine that can read and write basic files, reaching expectations.

4 Conclusion

In this article, we propose a new mechanism to make the new security technologies works with the existing ones. The experimental results shows it. But the system is a rudimentary form. If the virus detection and killing mechanism want to perfectly operated in modern vehicle systems, many other measures are needed to involved, to improve the reliability of the system, such as the division of resources and priority. For the classification of processes’ levels, the system itself also needs to pass the certification of security standards, but this scheme effectively uses the current general virus detection and killing mechanism, which reduces the time cost of this research and makes it easier to use previous successful experience.

Acknowledgments. I would like to thank AsiaInfo Technologies (Chengdu), Inc for its support in this research, not only for the sufficient arrangement of time, but also for the support of hardware and personnel. At the same time, I would like to thank my chief Yeli Xu, my boss Liang Yu, and my colleague Shengzhe Kan assisted in the development.
References

1. Zhou, S.: On embedded network security technology. Network Security Technology and Application (in Chinese)
2. Wang, Z.: Research on network security technology based on embedded operating system. Network Security Technology and Application (in Chinese)
3. Du, G., Wang, L.: About the security analysis of embedded network firewall. Network Security Technology and Application (in Chinese)
4. Lutz, R.: Analyzing software requirements errors in safety-critical. Embedded Syst., 126–133 (1993). https://doi.org/10.1109/isre.1993.324825
5. Kane, A.: Runtime Monitoring for Safety-Critical Embedded Systems (2015)
6. Nejati, S., Alesio, S.D., Sabetzadeh, M., et al.: Modeling and analysis of CPU usage in safety-critical embedded systems to support stress testing. In: International Conference on Model Driven Engineering Languages and Systems. Springer, Heidelberg (2012)
7. Popek, G.J., Goldberg, R.P.: Formal requirements for virtualizable third generation architectures. Commun. ACM 17(7), 412–421 (1974)
8. Krammer, M., Martin, H., Karner, M., Watzenig, D., Fuchs, A.: System Modeling for Integration and Test of Safety-Critical Automotive Embedded Systems, 2 (2013). https://doi.org/10.4271/2013-01-0189
9. Ding, Y.: Thinking of virus killing mechanism of embedded devices. Information Security and Communication Secrecy (in Chinese)
10. Luo, J., Hubaux, J.-P.: Embedded Security in Cars. Embedded Security in Cars – Securing Current and Future Automotive IT Applications (2005). https://doi.org/10.1007/3-540-28428-1_7
11. Babar, S., Stango, A., Prasad, N., et al.: Proposed embedded security framework for internet of things (IoT). In: IEEE International Conference on Wireless Communication. IEEE (2011)
12. Othman, N.A., Aydin, I., Karakose, M.: An efficient embedded security system for reduce car accident to build safer world based on IoT. In: 2019 International Artificial Intelligence and Data Processing Symposium (IDAP) (2019)
13. Ali, S., Al Balushi, T., Nadir, Z.: Embedded Systems Security for Cyber-Physical Systems (2018)
14. Kevan, T.: Facing the embedded security crisis. Desktop Eng. 23(9), 16–19 (2018)
15. Nathi, R.A., Sutar, D.S.: Embedded payload security scheme using CoAP for IoT device. In: 2019 International Conference on Vision Towards Emerging Trends in Communication and Networking (ViTECoN) (2019)
16. Ye, H.: Security protection technology of cyber-physical systems. Int. J. Security Appl. 9, 159–168 (2015)
17. Dong, P., Han, Y., Guo, X., Xie, F.: A systematic review of studies on cyber physical system security. Int. J. Security Appl. 9, 155–164 (2015)
Open Access  This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (http://creativecommons.org/licenses/by/4.0/), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter’s Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter’s Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.