Real-Time Self-defense Approach Based on Customized Netlink Connection for Industrial Linux-Based Devices

Ming Wan[1], Jiawei Li[1], and Jiangyuan Yao[2(✉)]

[1] School of Information, Liaoning University, Shenyang, China
wanming@lnu.edu.cn, lijiawei_lnu@hotmail.com
[2] School of Computer Science and Cyperspace Security, Hainan University, Hainan, China
yaojy@hainanu.edu.cn

Abstract. With the deep integration of IT (Information Technology) and OT (Operational Technology), various Linux operating systems have been successfully applied in critical industrial devices, such as Linux-based IIoT (Industrial Internet of Things) controllers or gateways, and the vulnerabilities of these systems may become a new breakthrough for the organized and high-intensity attacks. In order to prevent malwares from corrupting or disabling industrial Linux-based devices, this paper proposes a novel real-time self-defense approach, which can be easily developed without redesigning the basic software and hardware platform. By establishing the customized Netlink connection between kernel mode and user mode, this approach can monitor all application processes, and block each new malicious application process, which cannot conform to the trusted white-listing rules. All experimental results show that the proposed approach has a comparative advantage to effectively detect and prevent the malware-related attacks, and provides a self-defense function for industrial Linux-based devices, which meets their availability due to the millisecond resolution.

Keywords: Self-defense · Customized Netlink · Application process · Industrial Linux-based devices

1 Introduction

In recent years, the development of information-enabled industry has promoted the large-scale integration and interconnection of current industrial control systems, and the original self-imposed isolation of various industrial components has been broken [1]. As a result, the trend of increasing vulnerabilities appears as the nonlinear superposition, and today's industrial control systems are suffering from serious security threats due to the organized and high-intensity attacks [2–4]. Especially, advanced persistent threats have become the most popular and dangerous attack pattern, which infiltrates and destroys industrial infrastructures by exploiting various hidden zero-day vulnerabilities [5]. Although the researchers are giving full concentrations and efforts to excavate the

H. Gao et al. (Eds.): CollaborateCom 2020, LNICST 349, pp. 406–420, 2021.
https://doi.org/10.1007/978-3-030-67537-0_25

shortcomings of industrial hardware and software platforms, some hidden vulnerabilities still exist and are difficult to perform the patch updating [6, 7]. The main reasons can be summarized as follows: firstly, most of industrial vulnerabilities derive from the specialized function features which can be fully utilized by a professional adversary, and the traditional IT vulnerability mining technique is difficult to apply in wide range; secondly, although the number of industrial vulnerabilities is limited, their hiding ability becomes more disconcerting as their number decreasing; thirdly, the patch updating presents an obvious hysteretic nature, and it is a daunting task to repair these vulnerabilities because industrial control systems must ensure the uninterrupted running in a long term.

In order to achieve the deep integration of IT (Information Technology) and OT (Operational Technology), various Linux operating systems have been widely applied in critical industrial devices, for example, an open-source industrial IoT gateway is designed by exploiting the micro-service of Docker, whose software solutions are mainly based on Linux [8]; a software CNC (Computerized Numerical Control) controller which runs the Linux real-time operating system is developed to implement the EPL (Ethernet Powerlink) stack as a real-time kernel module [9]; SIMATIC WinCC which is widely used in today's industrial control systems can also provide the cross-platform support for Linux [10]. As a result, the vulnerabilities of Linux operating system become a subject worthy of special attention, and some potential security threats which aim at damaging or infecting the Linux operating system are coming to the surface. Typically, Mirai and Brickerbot malwares may pose serious downtime risks to Linux-based IIoT (Industrial Internet of Things) devices, because they can make an electronic device completely useless in a DDoS (Distributed Denial-of-Service) or PDoS (Permanent Denial-of-Service) attack [11, 12]. Although the physical layer authentication or secure transmission can provide an important opportunity to enhance IoT security [13, 14], the operating system-level security threats may still become a controversial issue. In order to deal with these security threats and challenges, some security defense technologies have been designed to ensure the availability of critical industrial devices, and these technologies can be roughly divided into two categories: indirect defense and direct defense. Furthermore, the indirect defense employs some external defense approaches to safeguard critical industrial devices, and one representative example is industrial firewall [15]. However, the indirect defense belongs to the traditional passive defense technologies, which cannot reply the targeted and multi-variability behaviors of industrial attacks. Differently, the direct defense strengthens the self-protection ability of critical industrial devices, and its defense mechanism is similar to the autoimmunity. As an example, trusted computing is a feasible approach which can provide the system integrity verification and identity authentication [16–18], but its main drawback is that it need redesign the software and hardware platform to add the trusted computing module.

In this paper, we propose a novel real-time self-defense approach for the Linux operating system, and this approach can be conveniently embedded without requiring the redesign of basic software and hardware platform. More specifically, this approach can monitor all application processes in kernel mode by establishing the customized Netlink connection, and check each new application process in user mode by comparing with the trusted white-listing rules. If one incoming application process is inconsistent with all white-listing rules, this process will be killed and an accompanying alarm will be

generated. In order to verify its defense effect, we perform the experimental verification and compared discussion by simulating the Modus/TCP runtime environment. Moreover, the experimental results show that our self-defense approach has an obvious advantage to effectively detect and prevent the malware-related attacks, and obtains the millisecond resolution which meets the availability of industrial Linux-based devices.

The main objectives and contributions of this paper are summarized as follows: firstly, we present the basic design principle of self-defense approach based on customized Netlink connection, and analyze its three significant aspects, which can directly relate to the benefits against the redesign of basic software and hardware platform; secondly, we depict the detailed execution procedures of self-defense approach based on customized Netlink connection, which can expound the simplicity and feasibility of self-defense process implementation; thirdly, we perform a lot of qualitative and quantitative experiments to verify the fine effectiveness of self-defense approach, and we hope that our approach can bring a feasible opportunity for the cyber security defense of industrial Linux-based devices, especially Linux-based IIoT devices.

The rest of this paper is organized as follows: Sect. 2 outlines the detailed design of self-defense approach based on customized Netlink connection, including basic design principle of self-defense approach and main execution procedures of white-listing monitoring and defense. Section 3 presents our experiments and discussions, including functional testing, comparative analysis and file protection verification. Finally, Sect. 4 provides some concluding remarks regarding this research.

2 Self-defense Approach Based on Customized Netlink Connection

2.1 Basic Design Principle

In general, the application process in the Linux operating system can be created by the following two ways: one is the indirect creation by generating an additional process copy, and the other is the direct creation by using the system call *execve()*. Actually, the additional process copy can also invoke the function *execve()* in its final phase. When a new user software boots, the kernel will be motivated to gain control of CPU, and the interface *system_call()* will be used to find the corresponding system call service routine according to the Linux system call table *sys_call_table*. In this case, we can further modify *sys_call_table*, and change the *execve()* address to the user-defined *hook_execve()* address, which is an important function installed in kernel mode. Before executing the native system call *execve()*, we can monitor and block the new application process. Figure 1 shows the basic design principle of our approach, which contains three significant aspects:

1) Netlink connection: the Netlink connection is an excellent method to achieve interprocess communications between kernel mode and user mode, and it can be established by some mature APIs (Application Program Interfaces) [19]. Moreover, the customized Netlink connection can support asynchronous communications from kernel mode to user mode, and send new process events to user mode without interfering with the stable runtime of other application processes.

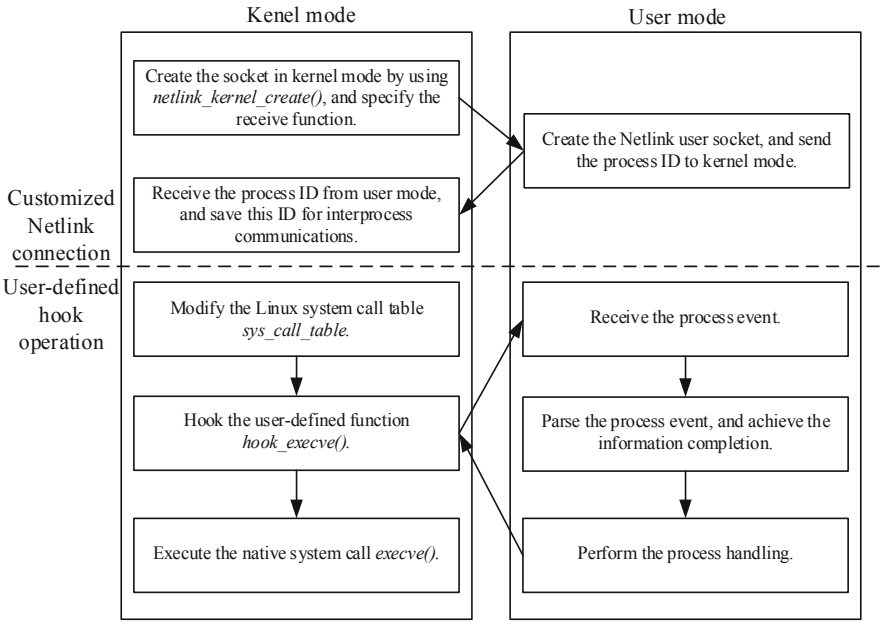


Fig. 1. Basic design principle of self-defense approach based on customized Netlink connection.

2) Hook operation: the hook operation can intercept the normal execution flow of one application, and insert some other operations or the terminated instruction. According to the Linux system call table *sys_call_table*, we can design a user-defined function *hook_execve()* before the system call *execve()*, and analyze the new process creation of executable file.

3) LKM (Loadable Kernel Module) mechanism: LKM mechanism [20], which improves the scalability and maintainability of Linux kernel, can provide an external interface for dynamic loading and unloading of modules. In our approach, the Netlink connection and hook operation are designed as LKMs, which smoothly run as two additional parts of Linux kernel.

The main advantages of our approach can be summarized as follows: firstly, the customized Netlink connection can establish a communication path between kernel mode and user mode, and it can enhance the real-time performance to receive new process events from kernel mode; secondly, the process handling is only performed in user mode, and the fewer operations in kernel mode cannot affect the system stability; finally, due to the frequent process creation, our approach can effectively use the simultaneous multi-threading technology in user mode to perform the process event parsing, information completion and process handling.

2.2 White-Listing Monitoring and Defense

Based on the above design principle, we further set the trusted white-listing rules which contain all regular application processes after the system boots, and compare these rules

with the new process to perform the process handling: if matched, the new process will be allowed to be created by executing the native system call; if unmatched, the new process will be killed and an accompanying alarm will be generated. Additionally, in order to ensure uniqueness, each white-listing rule must include three parts: the process name, its absolute path and MD5 signature. Figure 2 shows the main execution procedures of our approach, and the detailed descriptions are listed as follows:

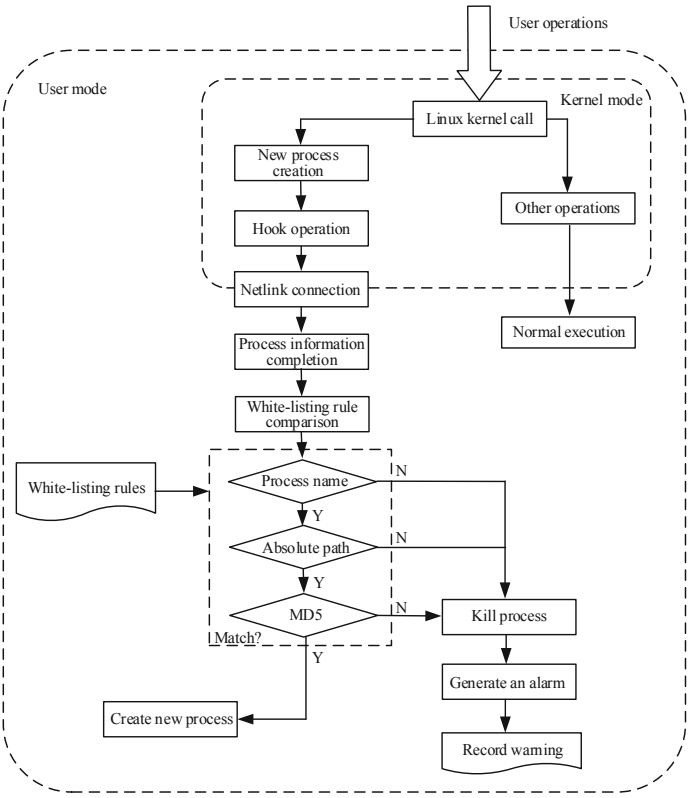


Fig. 2. Main execution procedures of white-listing monitoring and defense.

Step 1: Load LKMs, including the customized Netlink connection module and user-defined hook operation module. On the one hand, this step can establish the communication between kernel mode and user mode by inserting the customized Netlink connection module; on the other hand, this step can modify the process creation procedure, and add the user-defined hook operation module.

Step 2: Establish the customized Netlink connection. In user mode, the socket for the communication between kernel mode and user mode is created, and the corresponding information can be sent to kernel mode.

Step 3: Monitor each new process creation. In kernel mode, when the Linux system call for one new process creation is monitored, this process event (including the process

ID and its absolute path) is sent to user mode, and the information completion (including the process name and MD5 signature) is achieved in user mode.

Step 4: Check the new process. In user mode, the new process is checked by comparing with each white-listing rule, including the process name, its absolute path and MD5 signature.

Step 5: Perform the process handling. According the compared result, this step judges the legitimacy of new process: if matched, the new process will be allowed to be created by executing the native system call; if unmatched, the new process and its child processes will be killed.

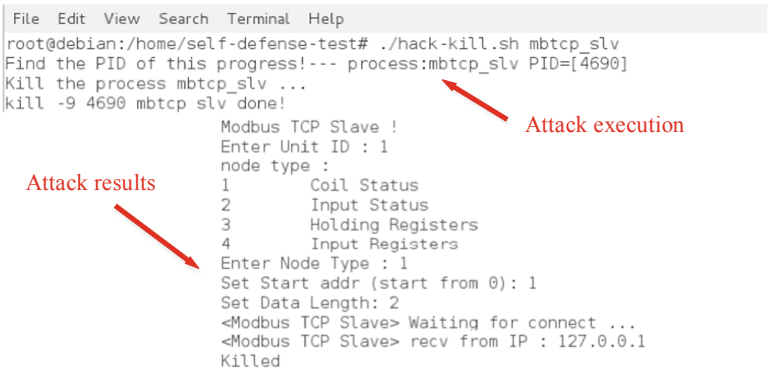
Step 6: Generate an alarm. For the abnormal process creation, an alarm is generated in user mode, and the warning information can be recorded to the log, such as the process name, its absolute path, MD5 signature and the alarm time.

3 Experimental Verification and Discussion

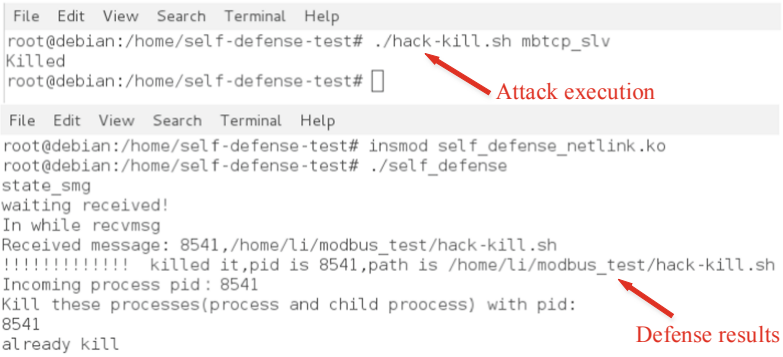
3.1 Functional Testing and Verification

Various soft PLCs (Programmable Logic Controllers) have been developed to complete industrial process control and automated production, for example, the OpenPLC can run on the Linux operating system to emulate a traditional PLC, which can use Modbus/TCP to implement the data acquisition and control management in industrial control communications [21]. In order to facilitate the functional verification, we install the Modbus/TCP master/slave software on the Linux operating system to simulate the Modus/TCP runtime environment. Under normal circumstances, we suppose the Modbus/TCP master executes a read operation whose purpose is to read the coil status of Modbus/TCP slave by using the function code "01" [22].

In order to verify the defense effect, we construct a malicious script "hack-kill.sh" to simulate a devastating attack, whose main purpose is to destroy the Modbus/TCP slave process. More specifically, this common attack can threaten the Modbus/TCP application, and the attack principle can be briefly described as follows: by searching through all running processes, this malicious script "hack-kill.sh" can locate the Modbus/TCP slave process, and forcibly kill it to destroy the normal industrial control activities. In this experiment, we respectively perform the same attack with and without our self-defense approach, and the experimental results are shown in Fig. 3. From this figure we can see that, when the system is not protected by our self-defense approach, Fig. 3(a) shows that the malicious script "hack-kill.sh" can be unknowingly executed, and the Modbus/TCP slave process is killed. More specifically, the malicious script finds the Modbus/TCP slave process ID 4690, and forcibly kills it to destroy proper functioning. Differently, when the system is zoned and protected by our self-defense approach, Fig. 3(b) shows that the malicious script "hack-kill.sh" can be successfully prevented without disturbing the availability of Modbus/TCP application. That is, before the malicious script performs destructive actions, our self-defense approach can check and kill the incoming process ID 8541 generated by the malicious script. From the compared results we can conclude that, our self-defense approach can effectively detect and restrict the script-related attacks, which use the native Linux script running mechanism to destroy various industrial applications.



(a) Attack execution and results without our self-defense approach

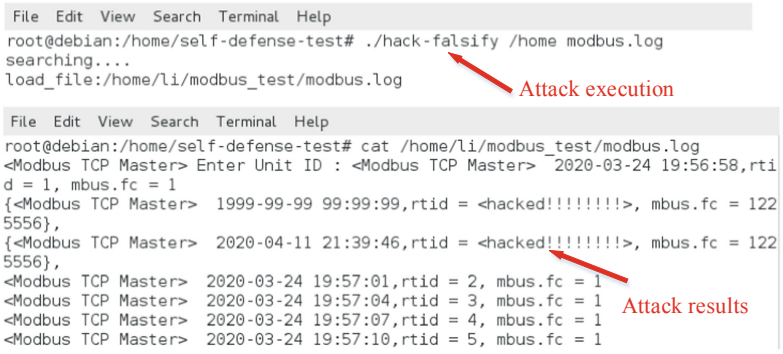


(b) Attack execution and defense results with our self-defense approach

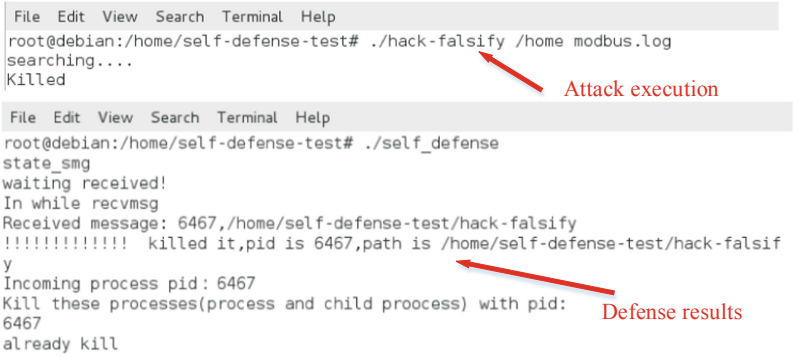
Fig. 3. Compared results by executing the malicious script "hack-kill.sh" with and without our self-defense approach.

Different from the malicious script, many malware-related attacks always execute the malicious codes which are designed by using some application development languages. In order to verify the corresponding defense effect, we develop one malware "hack-falsify" by using the classic Linux C program. Moreover, this malware can simulate a disturbing attack, whose fundamental purpose is to intentionally tamper with some critical file contents after searching for the specified target path and file. Similarly, we also perform the same attack with and without our self-defense approach, and the experimental results are shown in Fig. 4. In this experiment, the malware "hack-falsify" attempts to rewrite some critical parameter values in the Modbus/TCP log file "modbus.log", which may have a significant impact on the auditing of technological process. From this figure we can see that, when the system is not protected by our self-defense approach, Fig. 4(a) shows that the malware "hack-falsify" can be easily executed, and some critical parameter values in Lines 2 and 3 have been secretly modified, for example, the parameter *rtid* has been modified to "hacked". Conversely, when the system is zoned and protected by our self-defense approach, Fig. 4(b) shows that the malware "hack-falsify" can be successfully prevented, and the incoming process ID 6467 is killed

before the malware performs disturbing actions. To sum up, the compared results offer further proof that our self-defense approach can effectively detect and restrict some malware-related attacks, which develop some malicious applications to disturb various industrial activities.



(a) Attack execution and results without our self-defense approach



(b) Attack execution and defense results with our self-defense approach

Fig. 4. Compared results by executing the malware "hack-falsify" with and without our self-defense approach.

3.2 Performance Comparison

In order to illustrate the advantage of high efficiency, we select the consuming time to handle different process events as an important performance indicator, because an excellent defense approach can take short CPU time to achieve the noticeable protection. Moreover, we compare with a user-mode defense approach, whose main design principle can be summarized as follows: by using the native Netlink connector built in the Linux operating system, this approach can get the new process ID after triggering the system call *execve()*, and search the file system */proc* to obtain the absolute path of this new

process. Additionally, this process event can be checked and handled by comparing with its own white-listing rules. For a comprehensive analysis, we consider the following two situations: one is the consuming time comparison for one trusted process event, which can be regarded as a normal application conforming to the white-listing rules; the other is the consuming time comparison for one malicious process event, which can be treated as an abnormal application deviating from the white-listing rules.

The consuming time comparison for one trusted process event is shown in Fig. 5, and different curves depict the consuming time changes of two approaches under 25 experiments, respectively. From this figure we can see that, the consuming time of our self-defense approach is significantly lower than the one of user-mode defense approach. More precisely, the average consuming time of user-mode defense approach reaches about 0.83 ms, and the average consuming time of our self-defense approach is only about 0.69 ms. That is to say, our self-defense approach to handle one trusted process event can improve the efficiency by 17.45%.

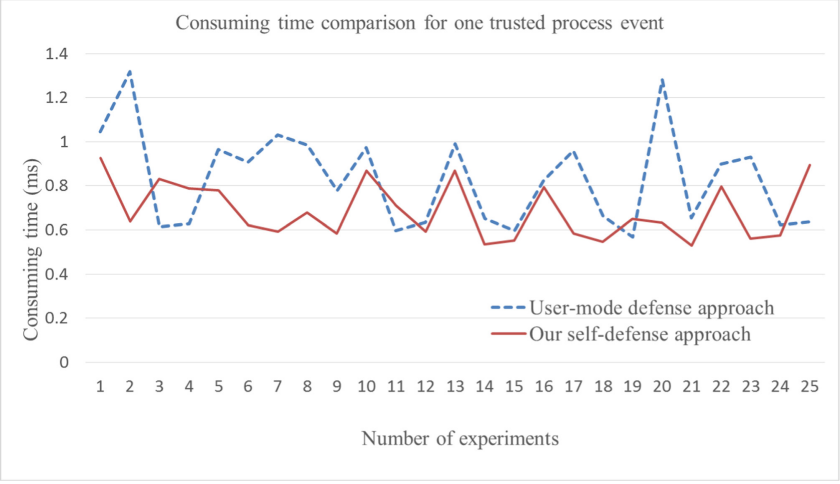


Fig. 5. Consuming time comparison for one trusted process event.

Similarly, Fig. 6 shows the consuming time comparison for one malicious process event under 25 experiments, and our self-defense approach also has a distinct advantage to prevent an abnormal application. Especially, the average consuming time of user-mode defense approach reaches about 1.77 ms, and the average consuming time of our self-defense approach is only about 1.63 ms. Namely, the increased efficiency of our self-defense approach to handle one malicious process event can reach 7.92%. From the above experimental results we can conclude that our self-defense approach can obtain the millisecond resolution which meets the availability of industrial Linux-based devices, and have better real-time performance to protect Linux operating systems.

In extreme cases, when monitoring each new process creation, our self-defense approach can obtain more accurate process event, whose application process may be created and released instantaneously. For example, Fig. 7 shows the different absolute

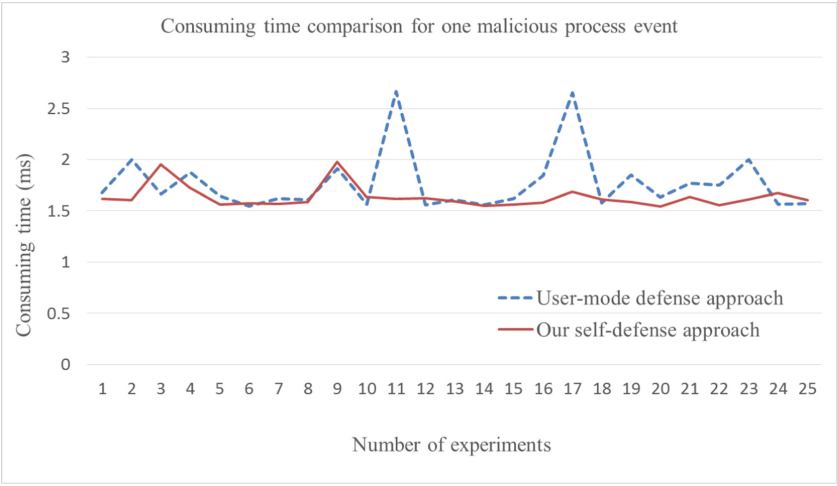


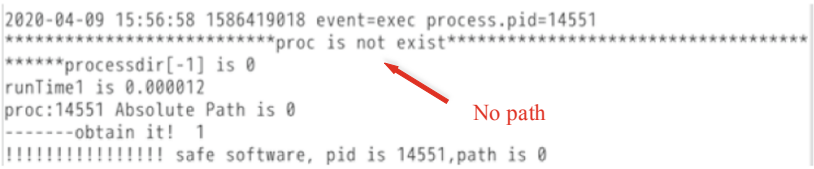
Fig. 6. Consuming time comparison for one malicious process event.

paths obtained by two approaches under the same process ID. From this figure we can find that, the same process ID is 14551, and our self-defense approach can obtain its full absolute path. On the contrary, the user-mode defense approach only shows "proc is not exist", namely, this approach cannot obtain any path information of process ID 14551. The main reason for this case is that before the user-mode defense approach searches the file system */proc* to obtain the absolute path, the application process has been released and the corresponding process information in the file system */proc* has been deleted. Therefore, under the same defense function, our self-defense approach can have a great advantage in information acquisition of new process event, which may provide more detailed warning information for a significant post-audit.

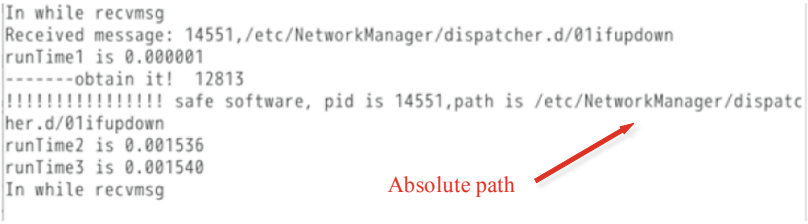
Compared with the user-mode defense approach, our self-defense approach not only has better real-time performance to satisfy industrial high availability requirements, but also forms a relatively complete process event to support security vulnerability analysis and auditing. In a word, our self-defense approach can provide more effective solutions to protect industrial Linux-based devices.

3.3 File Protection Design and Verification

Based on the hook operation mechanism, we can further develop a file protection function, which can further prevent an illegal access to some sensitive files. By using the hook operation, this file protection function can intercept the system calls to shared library functions, and insert some new executing codes which achieve the file access control. In the dynamic link libraries of Linux operating system, the environment variable *LD_PRELOAD* can affect the runtime linkers of various application programs, and preferentially allow to load one dynamic link library before running the application program. In other words, we can design a new library function to replace the original system call *read()*, and this new library function can check the protection status of one sensitive



(a) Absolute path obtained by user-mode defense approach



(b) Absolute path obtained by our self-defense approach

Fig. 7. Different absolute paths obtained by two approaches under the same process ID.

file to authorize or deny its access. More specifically, if one sensitive file is set to the protected file, this new library function will deny its access request, and generate an accompanying alarm. Actually, the most important advantage of file protection function is to prevent unauthorized disclosure of information when an adversary tries to penetrate and view some sensitive files by using some common Linux commands. Figure 8 shows the main execution procedures of file protection function, and the detailed descriptions are listed as follows:

Step 1: Load new library function in the dynamic link libraries. Based on the environment variable *LD_PRELOAD*, this new library function can own the highest priority. When one file is accessed, the hook operation of file protection can be executed before triggering the original system call *read()*.
Step 2: Monitor each new file access. If one new file access is monitored, the file protection function can be triggered.
Step 3: Check and handle the new file access. By comparing with the pre-determined absolute path which relates to the protected file, this step can judge the legitimacy of new file access: if matched, this file needs to be protected, and the corresponding file access can be denied; if unmatched, the file contents can be allowed to view.
Step 4: Generate an alarm. For the abnormal file access, an alarm is generated in user mode, and the warning information can be recorded to the log.

In order to verify the file protection effect, we suppose that one adversary has successfully penetrated some critical industrial Linux-based devices, and wants to steal confidential information by viewing some sensitive files. In general, two different methods can be easily carried out to achieve this goal: one is to view these sensitive files by using some common Linux commands, such as “cat” and “tail”, and the other is to view these sensitive files by applying some customized malwares, for example, we

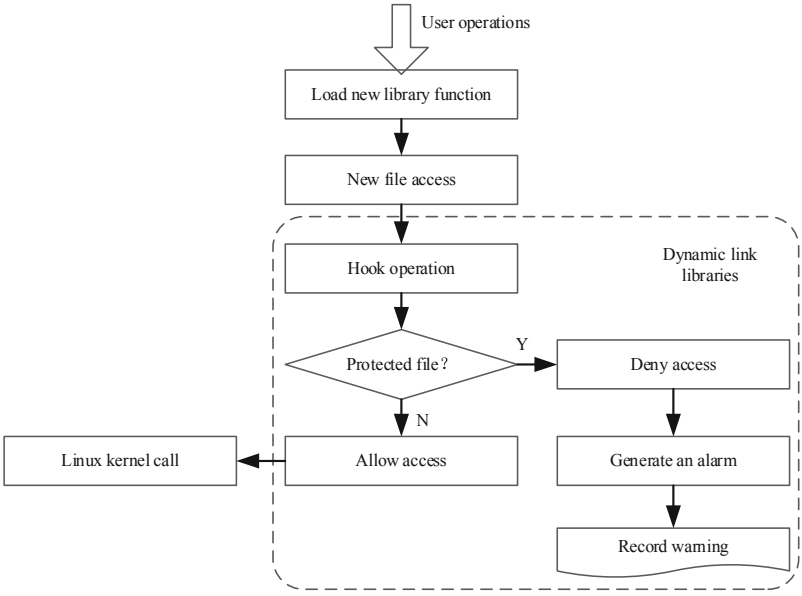
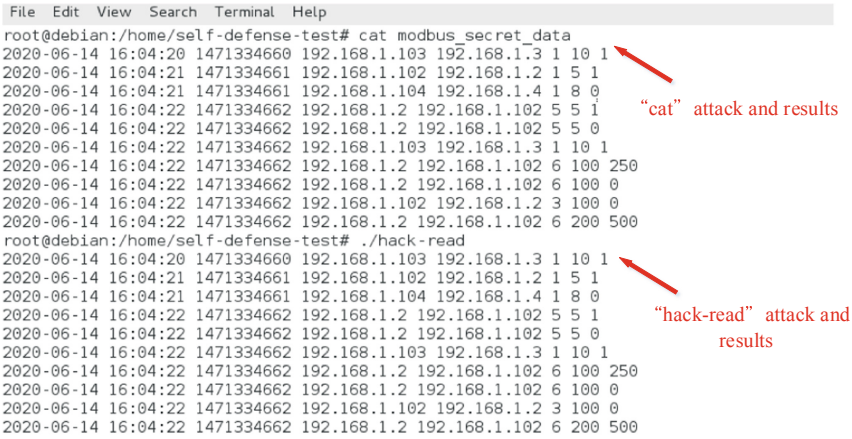
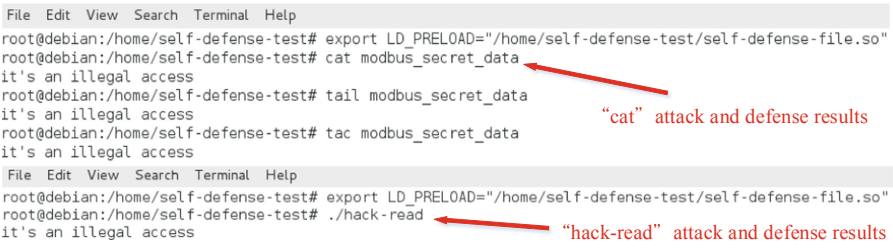


Fig. 8. Main execution procedures of file protection function.

develop a malware "hack-read" to realize this function, and this malware is designed by using the classic Linux C program. In our experiment, we also perform the same attack with and without our file protection function, and the experimental results are shown in Fig. 9. Moreover, we suppose that the file "modbus_secret_data" is a sensitive log which stores some key Modbus/TCP communication data, and the malware "hack-read" can directly search to view this file. From this figure we can see that, when the file "modbus_secret_data" is not protected by our file protection function, Fig. 9(a) shows that both the common Linux command "cat" and the malware "hack-read" can be easily executed to view all sensitive log records. Differently, when the file "modbus_secret_data" is protected by our file protection function, Fig. 9(b) shows that both the common Linux commands and the malware "hack-read" cannot be executed to view any sensitive log record, and the terminal can generate an alarm "it's an illegal access". Additionally, it is worth mentioning that three different Linux commands ("cat", "tail" and "tac") can be successfully prevented by our file protection function in Fig. 9(b). Therefore, the above compared results convincingly demonstrate that our file protection function can effectively protect sensitive information from leaving the confines of normal industrial production.



(a) Attack execution and results without file protection function



(b) Attack execution and defense results with file protection function

Fig. 9. Compared results with and without our file protection function.

4 Conclusion

Various Linux operating systems have been widely applied in critical industrial devices (especially Linux-based IIoT controllers or gateways), whose vulnerabilities can be viciously exploited by some potential security threats. In order to protect industrial Linux-based devices, this paper proposes a real-time self-defense approach based on the customized Netlink connection, which can be easily embedded without requiring the redesign of basic software and hardware platform. On the one hand, this approach can establish the interprocess communication between kernel mode and user mode to enhance the real-time performance; on the other hand, this approach can handle new process events in user mode without affecting the system stability. Additionally, this approach can be further developed to realize the file protection function. All experimental results show that the proposed self-defense approach can effectively prevent the malware-related attacks, and have better efficiency due to the millisecond handling latency. In a word, the proposed self-defense approach can successfully protect industrial Linux-based devices without disturbing their availability.

Acknowledgements. This work is supported by the Program of Hainan Association for Science and Technology Plans to Youth R & D Innovation (Grant No. QCXM201910), the Natural Science Foundation of Liaoning Province (Grant No. 2019-MS-149), the National Natural Science Foundation of China (Grant No. 61802092), and the Scientific Research Setup Fund of Hainan University (Grant No. KYQD (ZR) 1837).

References

1. Cheminod, M., Durante, L., Valenzano, A.: Review of security issues in industrial networks. IEEE Trans. Ind. Inform. **9**(1), 277–293 (2013)
2. Lyu, X., Ding, Y., Yang, S.: Safety and security risk assessment in cyber-physical systems. IET Cyber-Phys. Syst.: Theory Appl. **4**(3), 221–232 (2019)
3. Wu, G., Sun, J.: Optimal switching integrity attacks on sensors in industrial control systems. J. Syst. Sci. Complex. **32**(1), 1290–1305 (2019)
4. Adepu, S., Kandasamy, N.K., Zhou, J., Mathur, A.: Attacks on smart grid: power supply interruption and malicious power generation. Int. J. Inf. Secur. **19**(2), 189–211 (2019). https://doi.org/10.1007/s10207-019-00452-z
5. Yuan, H., Xia, Y., Zhang, J., Yang, H., Mahmoud, M.: Stackelberg-game-based defense analysis against advanced persistent threats on cloud control system. IEEE Trans. Ind. Inform. **6**(3), 1571–1580 (2020)
6. Pogliani, M., Quarta, D., Polino, M., Vittone, M., Maggi, F., Zanero, S.: Security of controlled manufacturing systems in the connected factory: the case of industrial robots. J. Comput. Virol. Hacking Tech. **15**(3), 161–175 (2019). https://doi.org/10.1007/s11416-019-00329-8
7. Wan, M., Shang, W., Zeng, P.: Double behavior characteristics for one-class classification anomaly detection in networked control systems. IEEE Trans. Inf. Forensics Secur. **12**(12), 3011–3023 (2017)
8. Nguyen-Hoang, P., Vo-Tan, P.: Development an open-source industrial IoT gateway. In: 2019 19th International Symposium on Communications and Information Technologies (ISCIT), pp. 201–204. IEEE, Ho Chi Minh City (2019)
9. Erwinski, K., Paprocki, M., Grzesiak, M., Karwowski, K., Wawrzak, A.: Application of ethernet powerlink for communication in a Linux RTAI open CNC system. IEEE Trans. Ind. Electron. **60**(2), 628–636 (2013)
10. Tufail, H., Anwar, M., Qasim, I., Azam, F.: Towards the selection of optimum alarms system in leading industry automation software. In: 2019 8th International Conference on Industrial Technology and Management (ICITM), pp. 241–246. IEEE, Cambridge (2019)
11. Su, J., Vasconcellos, D., Prasad, S., Sgandurra, D., Feng, Y., Sakurai, K.: Lightweight classification of IoT malware based on image recognition. In: 2018 IEEE 42nd Annual Computer Software and Applications Conference (COMPSAC), pp. 664–669. IEEE, Tokyo (2018)
12. Kolias, C., Kambourakis, G., Stavrou, A., Voas, J.: DDoS in the IoT: Mirai and other Botnets. Computer **50**(7), 80–84 (2017)
13. Zhang, N., et al.: Physical layer authentication for internet of things via WFRFT-based Gaussian gag embedding. IEEE Internet Things J. **7**, 9001–9010 (2020)
14. Zhang, N., Wu, R., Yuan, S., Yuan, C., Chen, D.: RAV: relay aided vectorized secure transmission in physical layer security for internet of things under active attacks. IEEE Internet Things J. **6**(5), 8496–8506 (2019)
15. Lee, S., Lee, S., Yoo, H., Kwon, S., Shon, T.: Design and implementation of cybersecurity testbed for industrial IoT systems. J. Supercomput. **74**(9), 4506–4520 (2017). https://doi.org/10.1007/s11227-017-2219-z

16. Yuan, J., Li, X.: A reliable and lightweight trust computing mechanism for IoT edge devices based on multi-source feedback information fusion. IEEE Access **6**, 23626–23638 (2018)
17. Maene, P., Götzfried, J., Clercq, R., Müller, T., Freiling, F., Verbauwhede, I.: Hardware-based trusted computing architectures for isolation and attestation. IEEE Trans. Comput. **67**(3), 361–374 (2018)
18. Ashraf, N., Masood, A., Abbas, H., Latif, R., Shafqat, N.: Analytical study of hardware-rooted security standards and their implementation techniques in mobile. Telecommun. Syst. **74**(3), 379–403 (2020). https://doi.org/10.1007/s11235-020-00656-y
19. Jia, J., Liu, G., Han, D., Wang, J.: A novel packets transmission scheme based on software defined open wireless platform. IEEE Access **6**, 17093–17118 (2018)
20. Zarrabi, A., Samsudin, K., Adnan, W.A.W.: Linux support for fast transparent general purpose checkpoint/restart of multithreaded processes in loadable kernel module. J. Grid Comput. **11**, 187–210 (2013)
21. Alves, T., Buratto, M., Souza, F., Rodrigues, T.: OpenPLC: an open source alternative to automation. In: IEEE Global Humanitarian Technology Conference (GHTC), pp. 585–589. IEEE, San Jose (2014)
22. Wan, M., Shang, W., Kong, L., Zeng, P.: Content-based deep communication control for networked control system. Telecommun. Syst. **65**(1), 155–168 (2016). https://doi.org/10.1007/s11235-016-0223-x