ATLANTIS PRESS

Advances in Engineering Research, volume 149

2nd International Conference on Mechanical, Electronic, Control and Automation Engineering (MECAE 2018)

How to Protect Your Secret in Memory

Hongwei Zhou [1], Jinhui Yuan [2, *]

[1]Information Engineering University, Zhengzhou, China

[2]SIAS International University, Zhengzhou, China

Keywords: Hardware, operating system, privacy, secret.

Abstract. Due to the limitations of computer architecture, any sensitive data is in clear text in the memory. If we enter sensitive data, such as a card password, into a computer, the data is also plaintext. It leaves the window for the adversary to steal your secret. How to protect your secret in the memory? In this paper, we summarize the existing solutions as four kinds, analyzed their advantages and disadvantages. Without changing the hardware, we believe that the necessary modification to the operating system is a more appropriate solution to protect the secret in the memory.

1. Introduction

Computers have become an indispensable part of our life. We use the computer to accomplish a lot of things. As an example, we buy a train ticket on 12306 website. In order to pay for the ticket, we have to enter the bank card number and password. Even though computers have security mechanisms such as anti-virus software, we still worry about that our privacy data.

Is our privacy data secure? The answer is no. We often find the news that bank accounts are stolen. Existing security mechanisms, such as access control and antivirus software, do not completely prevent attackers from stealing privacy data from computers. Due to the limitations of computer architecture, our privacy data is always in clear text in the memory. It gives the attacker an entrance to steal the data. More seriously, the privacy data may be duplicated in the memory, resulting in multiple copies of privacy data that can be stolen by attackers.

How to protect our privacy data in the memory? Some solutions have been presented to alleviate the problem. For example, we are able to place sensitive privacy data in a virtual machine client and release the entire client when the data is processed. Because of the strong isolation of the virtual machine, it is difficult for the attacker to steal user privacy data. In this paper, we introduce the main kinds of related security mechanisms and analyze their advantages and disadvantages. Moreover, we also give our views on memory privacy data protection.

2. Challenge

The key challenge to protecting memory data is the data in the memory is in plain text. The existing mainstream computer architecture is von Neumann architecture. In this structure, all the instructions and data in the memory are plain text, and the processor can only handle instructions and data in the form of plain text. And in essence, the memory is also shared by OS and processes. It is possible to steal the secret in the memory by the adversary who is able to access the memory in the architecture.

More seriously, the user privacy data are copied all time in the computer. The computer system has a Pyramid like storage system. For example, a hard disk has a large capacity and a slow speed, while register has a small capacity and a fast speed. To ensure the overall performance of the computer system, all instructions and data need to be transferred from hard disk to register. It means that the data is duplicated in the process of transfer.

Some of the activities of the operating system have also led to the aggravation of this problem. For example, in order to improve the utilization of memory, the operating system replace the most recent pages that are not used to the disk on. If the content of the displaced page is not completely overwritten,

then the privacy data is copied again. Similar to this, system sleep can also cause data to be copied in the memory.

The challenge of protecting the security of privacy data in memory is that it is difficult for us to give up existing computer systems and operating systems. Most of existing computer is von Neumann architecture, and it is unrealistic to give up such a large number of computers in order to protect the security of private data in memory. Most of existing operating systems are also contribute to the above problem. To overcome it, many solutions have been designed. In the following section, we discuss them.

3. Existing solutions

In this section, we summarize the existing major types of solutions and analyze their advantages and disadvantages.

3.1 Change programming habits

Bad programming habits are also one of the factors that affect the security of sensitive data in the memory. For example, the programmers save the user password with variable A in the dialog A. However, the variable A is copied to another variable B in the dialog B. Ideally, there is only one variable to store user passwords in the process. But bad programming habits lead to two variables in the process that contain user passwords. It increases the risk of privacy data being stolen in the memory.

Another typical programming error is the incorrect use of the memory allocation function. In the C language, we often use the malloc() to allocate free the storage space. Noted that when we do not use the storage space, we have to use the free() to release the space. However, some programmers forget the storage space in time. It makes that the storage space are not released and used again. What's more serious is that the data in this storage space will not be erased. If the space holds your privacy data, the adversary may have enough time to steal them.

The storage space that has been released should also be erased. Not all storage space that has been released will be used immediately. Before being used, these storage spaces still keep the original data in most operating systems. It means that an attacker can take the time to steal the privacy data. Modern operating systems have provided a number of memory release functions that can be automatically erased the memory space. The programmers should use these functions to reduce the life cycle of privacy data in the memory.

The programmer should select the appropriate type of variable to store the privacy data. Different types of variables have different life cycles. The life cycle of a static variable is consistent with the life cycle of the process. Using static variables to store privacy data does lead to a long life cycle of privacy data, and it gives attackers more time to attack. If programmers have good programming habits, we recommend that programmers use heaps to store privacy data. By using malloc() and free(), we have capability to control the life cycle of private data, and it reduces the time window that they are stolen. Otherwise, we suggest that programmers use local variables to store private data. In this way, the system will automatically recycling storage space, and is larger may be used soon again, thereby erasing the privacy data.

Changing programming habits is an immediate and effective solution to protect the privacy data in the memory. This approach does not require change of the hardware and operating system, so it is easy for users to accept this solution. This method does not apply to existing software without open source. Rewriting software is a time-consuming task. One possible approach is to build an automated upgrade tool that automatically scans the existing software for inappropriate implementations and updates it to new software. As far as we know, there is no such tool yet.

3.2 Hardware-based solution

Since system structure is the core cause of memory data security, and people want to make the some changes to overcome this problem. Of course, people don't want to subvert the von Neumann

architecture. People want to overcome this problem with a much smaller cost. So people try to change the hardware of the system from multiple angles and evaluate the cost of this modification.

AEGIS[1] and XOM[2] are the hardware-based solution. Since the processor can not process the ciphertext format of instructions and data, can we design a processor that can handle ciphertext? Under the guidance of this idea, they presented the new processor design solution. In short, this new type of processor added the encryption unit and the integrity verification unit. After the instruction or data enter the processor, they are first decrypted and validated. Before the data leaves the processor, the processor encrypts them. Thus the data in the memory are encrypted, and it is difficult to steal the privacy data in the memory as long as the attacker has no control over the processor.

Histar[3] is another solution with tagged memory. Strictly speaking, Histar is an information flow integrity solution. But it is able to prevent illegal access by users by controlling the flow of information. Histar is developed on Loki which is tagged memory architecture. Loki is built on the FPGA board, and Histar is a Unix-like operating system which is porting to run on Loki. With the support of Loki, Histar is capable of enforcing the security policy for the application by mapping application policies to the hardware. In this way, Histar can strictly control the flow of information in the computer system, so that the privacy data can be avoided by an attacker.

In our opinion, the main problem with hardware-based solutions is specific requirements for hardware. Currently, the processors, which is similar to that are designed by AEGIS and XOM, are not mass-produced and used. Loki is also just a laboratory experiment platform. Although such methods can solve the problem well, such methods are difficult to use because of the limitations of hardware requirements.

3.3 VMM-based solutions

As virtualization technologies mature and apply, people naturally want to use virtualization to protect secret data in memory. In essence, virtualization technology is presented for the virtualization of the hardware. Then we can build a virtual hardware platform for the virtual machine client to meet the special needs with the support of virtualization technology. In this way, we do not need to make special modifications to the hardware, but to build a specific security mechanism in the virtual machine client.

The direct idea is to run multiple virtual machine clients to run different applications[4]. For example, we run two virtual client machines on a physical system. The processes, which may hold privacy data, run in the secure client, such as online banking service. Other processes such as the games run in other client. Thus, the privacy data are isolated to a single client and is not disturbed by the process in other client. This method is easy to deploy, but it can not effectively avoid the attacker's theft of privacy data. An attacker can still use the vulnerability in the operating system to enter the client and carry out the theft.

SP^3 is presented to protect privacy data in the memory page with the support of the hypervisor. In virtualization, the client can only access the hardware through a virtual machine monitor. This request is received by the virtual machine monitor when the client accesses the memory page. At this time, the virtual machine monitor decrypts the physical page and provides the decrypted memory page to the client. When the client no longer accesses the page in the near future, the virtual machine monitor encrypts the physical pages. In this way, only when the page is normally accessed, the content of the page will exist in clear text. In other times, they are in the form of ciphertext. It effectively reduces the time for attackers.

In our opinion, the key challenge of this approach is the performance overhead of the virtual machine. Because of the existence of the virtual machine monitor, the simple operation may result in more performance overhead. Although there are many ways to improve performance, virtualization can still lead to a significant decline in performance. People don't sacrifice the overall performance of the system to protect their memory privacy data.

3.4 OS-based solutions

If we can't modify the hardware too much, optimizing the operating system is also a viable way to protect memory data. The basic positions of operating system makes the modification of operating

system may effectively protect the memory data. When page replacement can lead to increased risk of memory privacy data exposure, it would be a good idea to ban pages that contain privacy data. The Linux operating system provides the API that allows users to pin the specified page in the memory, which is the instantiation of this idea.

The most immediate idea for protecting memory data at the operating system level is to encrypt private data in the memory, and the method is called as memory encryption. Before the processor access to the page, the operating system decrypts the page. After the processor stops accessing the page, the operating system encrypts the page again. We present our solution which is lightweight memory encryption[6]. We identify the state of the page that contains the sensitive information by modifying the logo of the page table entry, and provide users with interfaces that identify sensitive data, allocate storage space, and recycle storage space.

We believe that this kind of method will be acceptable to users without changing hardware and applications. Because of the basic position of the operating system, the problem of memory data protection can be alleviated at the bottom of the system. However, we can't completely avoid memory data in the form of plaintext. In other words, this kind of method only reduces the risk of privacy data.

4. Discussion

On our analysis, we found that the hardware-based solution, although can effectively protect the privacy of data memory, but due to the special requirements for hardware, makes this method difficult to be used .in wide range. But it does not mean that it is useless. The desire for new hardware is one of the driving forces for hardware development. The full mining of existing hardware expands the application range of existing hardware. It makes it is possible to support more interesting solutions based on hardware. Trustvisor[7] build a certified execution environment with TPM's dynamic trust root which can be used to protect privacy data. We believe that more hardware-based methods will be proposed.

In our opinion, VMM-based solutions are interesting and effective methods. The operating system is a huge software system, and it's not a pleasant thing to modify it. We can not make big changes to the hardware due to the cost. With the support of VMM, some interesting solutions are presented. These methods are very clever using virtualization to bridge the operating systems to the hardware. The biggest drawback of VMM-based solutions is performance overhead. To save the performance overhead, people are trying to miniaturize the virtual machine. For example, I/O virtualization is not necessary to protect memory data, and it is cut off. There are now some small virtualization products that are dedicated to a certain security mechanism, providing security support such as strong isolation.

OS-based solutions are the incomplete solution. As long as the processor cannot accept ciphertext instructions and data, privacy data in the memory is always available in clear text. The most fundamental starting point for these methods is to shorten the life cycle of memory data. The ideal situation is to decrypt the processor before access and encrypt it immediately after access. But this undoubtedly leads to significant performance overhead. The local principle states that the instructions and data that have just been accessed are often accessed again. So the system repeatedly performs encryption and decryption to deal with local principles. Choosing the right time to implement encryption will be the key to this approach.

In our view, the most suitable solution at this stage is OS-based solutions. No special requirements for hardware and user software, OS-based solutions is easily accepted and applied. The biggest challenge of OS-based solutions is the size of the operating system. In view of this, we cannot rely entirely on the operating system to provide a security basis because of its large size.

5. Conclusion

Aiming at the fact that the memory data exists in the form of plaintext and lacks the necessary protection, we summarize and study the related work of memory data protection, and divide it into four aspects. In the discussion with experts, we further understand the advantages and disadvantages of

different types of work. In our view, it is the most feasible method to optimize the operating system and shorten the life cycle of security sensitive data to protect them in the memory.

Acknowledgements

Corresponding author Jinhui Yuan and her coauthors would like to thank the anonymous reviewers for their insightful comments that helped improve the presentation of this paper. The work is supported in part by the National Natural Science Foundation of China (61303074).

References

[1] Suh, G. E., O'Donnell, C. W., & Devadas, S. (2005). Aegis: a single-chip secure processor. *IEEE Design & Test of Computers, 10*(2), 63-73.

[2] Lie, D., Thekkath, C. A., & Horowitz, M. (2003). Implementing an untrusted operating system on trusted hardware. *Nineteenth Acm Symposium on Operating Systems Principles* (Vol.37, pp.178-192). ACM.

[3] Zeldovich, N., Boyd-Wickizer, S., & Kohler, E. (2006). Making information flow explicit in HiStar. *Usenix Symposium on Operating Systems Design and Implementation* (Vol.54, pp.19-19). USENIX Association.

[4] Garfinkel, Tal, Pfaff, Ben, Chow, Jim, Rosenblum, Mendel, & Boneh, Dan. (2003). Terra: a virtual machine-based platform for trusted computing. *Acm Sigops Operating Systems Review, 37*(5), 193-206.

[5] Yang, J., & Kang, G. S. (2008). Using hypervisor to provide data secrecy for user applications on a per-page basis. *ACM Sigplan/sigops International Conference on Virtual Execution Environments* (Vol.1, pp.71-80). ACM.

[6] Zhou H. W., Yuan J. H., (2017). Protect Sensitive Data with Lightweight Memory Encryption. International Conference on Information Technology, Computer, and Electrical Engineering

[7] Mccune, J. M., Li, Y., Qu, N., & Zhou, Z. (2010). Trustvisor: efficient tcb reduction and attestation. *Cylab, 41*(3), 143-158.