DIVISION OF COMPUTER SCIENCE

Is Your Computing Environment Secure? - Security Problems with Interrupt Handling Mechanisms

Ping Hu
Bruce Christianson

Technical Report No.222

February 1995

Is Your Computing Environment Secure?
— Security Problems With Interrupt Handling Mechanisms

Ping Hu Bruce Christianson
School of Information Sciences, Hatfield Campus
University of Hertfordshire, England
{B.Christianson, P.Hu}@herts.ac.uk

February 16, 1995

Abstract

In an open distributed system, resources must be shared among various users. Security is one of the major issues in designing such a system. When a computer system is connected to a network, it is very important to ensure that the computer has the ability to manage its local resources securely. In this position paper, we will demonstrate that current computer architectures do give malicious users ways to penetrate computer systems and hence access the system or other user's secrets which are supposed to be well protected. We also propose some possible solutions to counterattack such security threats either at hardware or software level. The impacts of such mechanisms to system hardware and software are also discussed.

1 Introduction

Computers that are connected to computer networks enable users to utilise various facilities offered by others. A user on a computer can request services provided by other entities or share resources with other users in a distributed system. But security requirements must be enforced in a truly open distributed system. Generally each individual computer system is assumed to be responsible for maintaining its own local resources. This means that user information in a computer system must be protected from any unauthorised access. Because the network connections between individual computer systems are usually insecure, i.e. communications over the network are liable to suffer from passive or active attacks as well as data transmission errors and failures, various security mechanisms, such as cryptographic algorithms, are used to provide secure communication links between computer systems [1].

In an open distributed system, all messages passed between the peer communication entities must be authentic and secure, which includes not only message transmission over the computer network but also message processing inside both local and remote computer systems. From a user's point of view, the damage to him caused by security breaches is the same no matter where attacks take place, either during the message transmission or the message processing inside a computer system. To realise the resource sharing, users should have some degree of assurance that their assets shall not be abused while they navigate in the open distributed system. Some security models for open distributed systems do exist [3, 6, 7] whereas each computer system is usually assumed to be competent for managing and protecting its resources and computing environment within its boundary provided that the computer hardware and system software, e.g. the operating system, are in right place and functioning correctly.

But in this paper we will only discuss issues related to how securely a conventional computer system, which is connected into an open distributed system, preserves the resources that are under its control. In fact we will demonstrate that current computer architectures do not exclude malicious users from unauthorised access to resources managed by a computer system. This will make secret information like user passwords and cryptographic keys most vulnerable to attackers. Some counterattack mechanisms are also proposed. In order to implement these mechanisms, it is likely that some extra hardware components are required to avoid such security threats.

2 Secure computing environment

In an open distributed system, computer systems are usually available to public, i.e. users have unsupervised access to the equipment. The computers are assumed to be tamperproof, which means that their integrated hardware components such as *CPU*, *RAM* and *ROM* can hardly be changed or subverted without anybody else notice or suspicion. At any time each computer system can be rebooted and loaded with new software code from an archive which could be either locally available or over the network. A protocol is proposed in [4] for securely booting a computer in an open distributed environment. As long as the booting process succeeds, the computer is believed to be solely under the control or surveillance of the correctly loaded software code, e.g. an operating system kernel. For a single-user computer system such as a *PC*, a computing environment for the user who initiates the booting is then established. In the case of a multi-user computer system which ranges from a *UNIX* workstation to a very large time-sharing system, each subsequent user of the computer system is allocated a subsystem, e.g. a virtual machine, as the user's computing environment. We assume that users, when they log into the multi-user computer system, reasonably believe that the superuser or system administrator who initiates the booting process is honest and the loaded operating system functions correctly. The problem of how to login to a remote computer system securely is discussed in [2].

Each user of a computer system now and then requires part or whole of his computing environment to be "closed" to other entities. A "closed" sub-environment, which can exist for very short time or last a whole session, is used to store its user's secret information such as his password and to carry out sensitive activities such a key generation. The resources inside a closed environment should not be accessed from other entities either deliberately or accidently. How to realise such a closed secure environment in a computer system which is connected to an open distributed system? Provided that the system hardware[1] functions correctly, it can be done by setting the program counter to a trusted program address and disabling any other interrupts or untrusted hardware components such as *DMA* communications. If the computer system such as a *UNIX* workstation is not solely used by a single user, these interrupt handling activities may not be disabled in order to favour one user. In these cases the untrusted activities must be under the control or surveillance of some trusted entities such as the interrupt handlers so that the user's closed environments will not be compromised. Generally, it is assumed that in a conventional single computer environment computer security can be effectively enforced purely through the operating system running on the computer system concerned. High levels of security can be achieved purely through careful construction of the operating system software [5]

For example, Lomas and Christianson [4] discuss the issue of securely booting a relatively stateless computer system connected to a network in a potentially hostile environment. Since the computer is used only by the booting procedure initiator, the closed secure computing environment means that the program counter is set to a fixed address in *ROM* from which the booting procedure starts and all interrupt handling mechanisms are disabled. Then the user password can be input and used to check the correctness of booting code which is loaded over the network. If the booting code is error-free, the password will be erased and the control will be transferred to the secure booting code.

It is easy to think of other examples in which such closed secure environments are required in order for users to carry out sensitive activities.

1. A user is asked to sign a document using public key cryptographic algorithm. The user's private key must be kept secret at all time, and the document integrity must be preserved while the signature is calculated.

2. When a user sets up a connection to a server, a closed environment must be set up in order to generate a new session key and to maintain the key confidentially during the whole session.

3. When a user intends to send a message, a closed environment is required from the time when the message starts to be created until the message is encrypted.

Our question is like this: is such a closed computing environment set up in a conventional computer system really secure, i.e. can the environment really enforce confidentiality or integrity to the information inside it? From the following discussion we will see that in most cases the answer to the question is no unless some special designed mechanism is employed in the computer system.

[1]Here we mean the basic hardware components inside a computer "box", such as *CPU*, *ROM* and *RAM*. It does not include removable disk drivers or other external peripheral devices.

3 Interrupt and handling mechanisms

The problem comes from interrupts and their handling mechanisms in a computer system. In the real world, not all interrupt signals to a computer system are processed by software. In fact some interrupt handling mechanisms are implemented at hardware level, which cannot be disabled under any circumstances, e.g. a power failure interrupt handler. Some interrupt events like power failures are so fatal to a computer system that they must be responded to immediately. Actually in almost all computer systems those interrupt handling mechanisms are implemented at hardware level to speed up the processing time because for example after a power crash signal is detected, the time available to a computer system to process the event is very short (probably a few instruction cycles) before the computer system is shut down. Under current computer architectures, when those fatal and urgent interrupt events occur, the whole current computer state is automatically saved with the help of some hardware components to somewhere permanent such as on a hard disk. Of course the state includes all the system and user's running process information so that the computer state can be recovered and all the process in the computer system can continue from the exact points where they were interrupted after power supply to the computer system is restored. In some cases users of a computer system may even not be aware that the power supply to the computer system was just shut down and restored again but all information in every user's environments has "safely" been dumped to somewhere on a hard disk.

Even if every computer system is tamper-proof, it should relatively be easy for someone to cut the power supply to a computer system and force the system to download its state information on a hard disk. There must exist some ways that other authorised users are able to access the downloaded information since all computer systems are assumed to be accessible to public. For example, the hard disk might be connected to an open network or can be replaced by someone who pretends to be a repairman. As a result, the information inside a user's closed environment in a computer system is not "closed" and secure any more if the environment happens to be downloaded. Beware that the mechanisms by which a computer system handles these interrupt events are determined by its architecture, so usually there is no way to change it through reconstruction of the operating system software.

4 Security threats

Now we see that a closed environment for a user is not so closed and secure as it is supposed to be. Obviously no advanced technology is required for an attacker to access the downloaded information on a hard disk, i.e. to penetrate a user's closed environment. The consequences of the security breaches can be classified into two categories, i.e. information confidentiality violation and integrity violation.

An attacker can copy the downloaded computer system state information and search some user's environment for secrets, which violates the information integrity policy. For example, by this means an attacker may steal a user's plain text password if he can successfully get the user's login process information. Encryption keys, even user private keys, can also be comprised in the same way if they are stored in such closed environments.

Alternatively, an attacker can subvert a user's closed environment by changing the downloaded system state information before the information is used to restore the computer system state. Examples of this kind of integrity violation are easy to list. Malicious users can change other user's program, document and even cryptographic keys without their owner's notice.

We assumed that computer systems in an open distributed system can be accessed by users without supervision. With current computer architectures a user can hardly prevent other untrusted entities from accessing his secret information stored in a computer system which is connected to a network. It will be extremely dangerous if, in the event of a fatal and urgent interrupt like a power failure to a computer system which implements its handling mechanisms at hardware level, the current computer system state is automatically dumped on a hard disk and restored from the information on the hard disk later without anyone's awareness.

5 Counterattack mechanisms

In this section we will discuss various ways to counterattack above security threats to user's closed environments and investigate their advantages and disadvantages.

5.1 Rolling back

An easy and efficient way to avoid integrity breaches is for each user to roll back his computation to a point which he is willing to trust in the event of a power failure interrupts. In the worst case, a user can abort all his processes. For example, if the secure booting procedure proposed in [4] was interrupted, the booting initiator could always re-start the process from its very beginning. If a user process is suspended after a document and its checksum have been received but not been verified yet, the user can ask a new copy of the document and its checksum when his process is resumed.

A lot of literature has been published to discuss techniques for transaction rolling-back. Our problem is that how to let each current user of a computer know that the system is crashed and there exists a security threat to their processes. Current computer architectures and operating system design strategies tend to make interrupt handling mechanisms transparent to user and other unrelated processes. But if it is a power crash interrupt that causes a computer system to shut down, it might not be very difficult to let the current users know it when the system is recovered. Such as users can detect that the computer system is not in operation, a superuser or system administrator can tell all current users in person, or a piece of special code can be put into the booting code, which will be invoked as the system is recovered, to inform each current running processes. However the rolling back techniques can only be used to protect the information integrity.

It will be unacceptable for a process to roll back every time it is interrupted[2]. Since the operating system kernel is still running and is trusted in most cases, usually the kernel can do something to protect the interrupted process, e.g. to encrypt the process before it is saved on a hard disk. But the operating system kernel may not be able to encrypt the process which is subject to schedule out, e.g. the time between an urgent interrupt signal arrival and the response is not long enough for the kernel to encrypt the process. We will discuss this later as it is also related to maintaining the confidentiality policy.

5.2 Special dump location

If all users of a computer system accept the condition that their processes can be aborted abruptly at the system's discretion[3], both the confidentiality and integrity of user information can be easily preserved. We need just ban the computer system state dumping from those fatal and urgent interrupt handling mechanisms. It may require slight modification to the computer system hardware. Alternatively, we can always redirect the system state dumping to a dummy location or device, e.g. `/dev/null` in a *UNIX* workstation.

But for most computer users, the interrupt handling mechanisms are still required to make processes run continually even in the event of power crashes. A special storage device could be attached to a computer system to save the system state. The requirement to the device is that only the trusted authorities such as recovery process are able to access the system state information only for system recovery purpose. A fixed internal hard disk in a computer system plus a carefully designed operating system kernel might satisfy this requirement. This is also a solution to the problem in Section 5.1.

We should also bear in mind that it is impossible to have one such device for several computer systems because of untrusted network connections. This certainly complicates the system design. Furthermore, a generally accepted philosophy for security is that a user will only trust an entity while he is using its services. When a computer system is recovered from a power crash, the running operating system kernel or the superuser who is responsible for the recovery may not be trusted by some of the original users, i.e. it is hard to safeguard a device in an open environment for long time.

5.3 Extra hardware components

We can also preserve the information confidentiality and integrity stored in a computer system if we are able to modify the design of current computer architectures. An obvious strategy is to make sure that some *RAM* segment or registers in a computer system will never be dumped by the fatal and urgent interrupt handling hardware components. One way to implement this strategy is that two registers are added to a computer system

[2]In a time-sharing system, every process will be scheduled out after it occupies the *CPU* for a period of time.

[3]For example in most cases a user on a single-user system like a *PC* would rather have his processes aborted than take a risk that his secrets will be compromised.

and are used to tell the hardware components which area of the memory contains secret information and should not be downloaded to a hard disk in the event of emergency.

This will make the recovery process difficult since all secret information will be lost when a power failure event occurs. But it might be still acceptable for some user processes, e.g. a session key between a user and a remote entity would be discarded even if it could be recovered because the connection terminates for some protocols in the event of a power failure interrupt. Alternatively, each user can input his secrets such as his password again, or some rolling back techniques can be used to recover the affected processes.

Another disadvantage of this strategy is that the size of the segment used to keep secret information is dynamic and variable. For some processes maybe only a user password or cryptographic keys need to be put in the segment, but it is quite possible for a process to put a large document or even the process itself in the segment. The secret information of all processes must be kept in the segment at all times.

5.4 Software solution

A software solution to ensure the information confidentiality and integrity might be also possible, but it probably still needs some hardware support. When a fatal and urgent interrupt event such as a power failure to a computer system occurs, each process in the computer system will have all its secret information erased or encrypted before the control is transferred to a interrupt handler. It could be done by either the process itself or some entity in the operating system, but the latter seems more appropriate. Since it is the nature of an interrupt event that determines how quickly the computer system must respond, the total processing time for secret information deletion or encryption must be kept to a minimum. A feasible implementation method of this software solution might be for each process's secret information to be encrypted well in advance of any fatal and urgent interrupts, i.e. to be encrypted whenever the process is suspended.

The real difficulty to this solution is that in the event of a fatal and urgent interrupt how can the current running process have its secrets erased or encrypted. First, is the time available for the operations long enough? Second, if the answer to the first question is yes, how can the entity which does the job be invoked? As the interrupt handling mechanism is implemented at hardware level of a computer system, the entity must know the interrupt event in advance. It can hardly be realised unless the entity is a part of the interrupt handling mechanism.

We must also solve another problem: who encrypts the entity's encryption key? Well, it can be encrypted with another (either secret or public) key in the operating system kernel when this entity is created. As the recovery process is in progress, the superuser or system administrator can then input the key again. However the entity's key must be destroyed while the current running process's secret information is erased or encrypted[4].

[4]To encrypt a key with the key itself is not a good idea and it is against the key management policy.

6 Discussion

We could also design a hybrid strategy from what has been discussed in Section 5. For example, an entity is created in the operating system to encrypt all secret information of suspended processes and a special segment keeps the entity's encryption key and the secret information of the current running process. When a power failure interrupt occurs, the interrupt handling hardware components will dump all the current system state except the secret information in the special segment. To recover the system state, the entity has to regain its key from input or other secure channels, and the current running process can always be rolled back to the last time it is suspended.

It depends on the user and system requirements and the computing environment to choose which mechanism is used to make user's closed environment really secure. For a small computer system, dumping the computer state to a dummy file sounds a simple and efficient way to maintain user's information confidentiality and integrity. To realise other strategies, some extra hardware components are required to be added to a computer system, and at the same time the operating system and other system software may need some modification. However for a large time-sharing system, recoverability seems essential to its users. From our discussion, the modification is likely to affect a large part of the computer system, including both hardware and software. Transaction rolling back techniques are old topics in both database systems and distributed systems. We have not discussed how these techniques can be applied here yet although they are very important.

Another aspect that we have not included in this paper is the impact of these strategies to a user programming environment. A user interface should be created for each user to specify which information has to be put in his closed environment, i.e. the special segment, and to let the operating system kernel know where this environment is. As a result, probably current programming languages, compilers or operating systems have to be modified to cope with it.

7 Summary

Contrary to most people's belief a computer system which is accessible to public is not secure even if the operating system and other system software that are running in the system are honest and capable of managing resources within the system, i.e. the system software is trusted by its users. Current computer system architectures to handle some exception events enable malicious users to access secret information which they should not be able to. Actually no advanced technology is required to do it. Some mechanisms have to be devised to make sure that the system environment is really secure. In this paper, we have discussed various approaches to eliminate such security threats to computer systems. Various facts influence the decision on which one is chosen to suit a computer system. In depth analysis is planned as our future work.

References

[1] ISO 7498-2. *Information Processing Systems – Open Systems Interconnection – Basic Reference Model, Part 2 Security Architecture.* International Standards Organization, 1988.

[2] R. J. Anderson and T. M. A. Lomas. On fortifying key negotiation schemes with poorly chosen passwords. *Electronics Letters*, 30(13):1040–1041, 1994.

[3] M. Gasser, A. Goldstein, C. Kaufman, and B. Lampson. The Digital Distributed System Security Architecture. In *Proceedings of the 12th National Computer Security Conference*, pages 305–319, Baltimore, Md., October 1989.

[4] M. Lomas and B. Christianson. To whom am I speaking? *IEEE Computer Magazine*, 28(1):50–54, 1994.

[5] C. J. Mitchell and P. Hu. Distributed system security model. Technical Report CSD-TR-93-6, Department of Computer Science, Royal Holloway, University of London, Surrey, U.K., June 1993.

[6] R. Molva, G. Tsudik, E. Van Herreweghen, and S. Zatti. *KryptoKnight* authentication and key distribution system. In Y. Deswarte et al., editors, *Proceedings of the Second European Symposium on Research in Computer Security, ESORICS92*, pages 155–174, Toulouse, France, November 1992. Computer Society, Spring-Verlag.

[7] J. Steiner, C. Neuman, and J. I. Schiller. Kerberos: An authentication service for open network systems. In *Proceedings of the USENIX Winter Conference*, pages 191–202, February 1988.