electronics

MDPI

Article

Mandatory Access Control Method for Windows Embedded OS Security

Chaeho Cho [1], Yeonsang Seong [2] and Yoojae Won [1,*]

[1] Convergence Security Research Center, Department of Computer Engineering, Chungnam National University, Yuseong-gu, Daejeon 34134, Korea; greatopen@cnu.ac.kr
[2] Windows Driver Development Team, Hauri Inc., Gajeongbuk-ro, Yuseong-gu, Daejeon 34134, Korea; dustkdwkd@naver.com
* Correspondence: yjwon@cnu.ac.kr; Tel.: +82-042-821-6294

Abstract: The Windows Embedded operating system (OS) adopts a discretionary access control (DAC)-based policy, but underlying vulnerabilities exist because of external hacker attacks and other factors. In this study, we propose a system that improves the security of the Windows Embedded OS by applying a mandatory access control (MAC) policy in which the access rights of objects, such as files and folders, and subjects' privileges, such as processes, are compared. We conducted access control tests to verify whether the proposed system could avoid the vulnerabilities of DAC-based systems. Our results indicate that the existing DAC-based security systems could be neutralized if a principal's security policy is removed. However, in the proposed MAC-based Windows Embedded OS, even if the clearance and category values of a subject's files are given the highest rating, all accesses are automatically denied. Therefore, the execution of all files that were not previously registered on the whitelist was denied, proving that security was improved relative to DAC-based systems.

Keywords: Windows Embedded OS; file system filter driver; access control; discretionary access control; mandatory access control

Citation: Cho, C.; Seong, Y.; Won, Y. Mandatory Access Control Method for Windows Embedded OS Security. *Electronics* **2021**, *10*, 2478. https://doi.org/10.3390/electronics10202478

Academic Editor: George Angelos Papadopoulos

Received: 9 September 2021
Accepted: 8 October 2021
Published: 12 October 2021

1. Introduction

An embedded operating system (OS) is a specialized OS designed for specific purposes, and it is installed as a built-in component of a system, for example, a point of sales (POS), automatic teller machine (ATM), or a KIOSK. As embedded OSes are designed to serve specific purposes, they operate in a low-memory capacity, under low-power, and in low-CPU environments and have limited support capabilities.

For IoT devices based on the Windows Embedded OS, the safety of the OS, which is the infrastructure of the service, is very important, and additional security mechanisms are required. Recent research on detecting or blocking the execution of malware in advance when the OS is polluted by an external or internal hacker attack is focused on deep learning methods [1,2]. However, because Windows Embedded OSes use the discretionary access control (DAC) model for access control and require computer resources that are limited, it is inappropriate for applications to use systems requiring high resource allocation, such as traditional anti-virus programs, for security purposes.

In this study, we propose a method for improving the security of the Windows Embedded OS by implementing mandatory access control (MAC)-based policies to determine the privileges of subjects and objects. The proposed MAC-based security system was implemented as a file system filter driver, and the security policy was managed by the filter driver kernel memory and the file system's alternate data stream (ADS) to reduce resource usage and enhance policy security in limited system environments.

2. Background

2.1. File System Filter Driver

The file system filter driver, a component of the Windows kernel mode, is a driver that is called when a file I/O operation (creating, reading, writing, etc.) occurs in the OS. A file system filter driver that runs on top of the file system driver can intercept requests that target a file system or other file system filter drivers (Figure 1). Microsoft provides an official method for using the file system filter driver to expand or replace the file system function [3,4]. If the request is intercepted before the intended target is reached, the filter driver can extend or replace the functionality provided by the original target of the request. Most antivirus programs, file system encryption programs, file system backup programs, and snapshot programs use file system filter drivers [5].

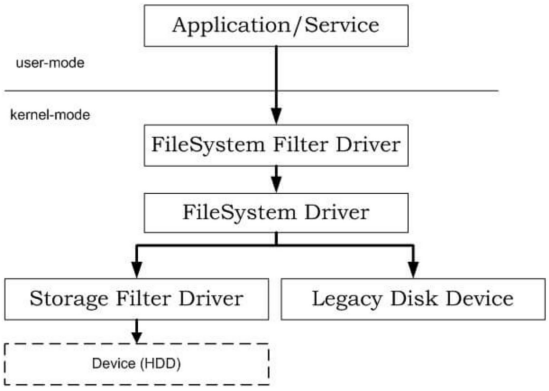

Figure 1. Structure of a file system filter driver.

Prior to the advent of the minifilter method, the legacy method of developing a filter driver utilized the I/O request packet (IRP) hooking method. This hooking method was phased out when Microsoft supported the minifilter method, and driver development has become easier with the provision of a kernel application programming interface (API). Consequently, it provides security against rootkit, which is a method for hacking kernels using hooks.

2.2. Input/Output Request Packets

Windows OS uses data structures called IRPs, which communicate with each other and with the OS. These data structures describe I/O requests and can be equally well thought of as "I/O request descriptors". The IRP structure is self-contained in the sense that it holds all the information required for a driver to process an I/O request. Some parts of the IRP structure contain information that is common to all the participating drivers in the stack, while other parts hold information that is specific to a particular driver in the stack [4].

2.3. Access Control Model

Access control refers to controlling access by determining the access rights and security settings of each subject and object when a subject of access, such as a user or process, accesses an object, such as a file or directory. The existing Windows OSes adopt an access management policy called DAC (Figure 2). DAC is easy to implement because of its low share of resources and easy security policy management. However, the owner of a file can arbitrarily grant their rights to other users, which allows access to objects to be delegated or removed [6,7].

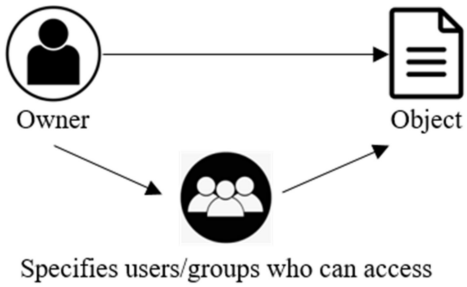

Figure 2. Implementation of the DAC model.

However, MAC is an access management policy that checks and compares the permissions of both the subject and the object. In other words, security levels are set for each user, as well as for objects (such as files); access can be controlled by comparing security labels on both sides when the main user accesses the object file, so that information set to a high security level is not exposed to low-security-level subjects (Figure 3). MAC demonstrates better security and confidentiality than DAC, but has the disadvantages of a complicated setup and difficult management.

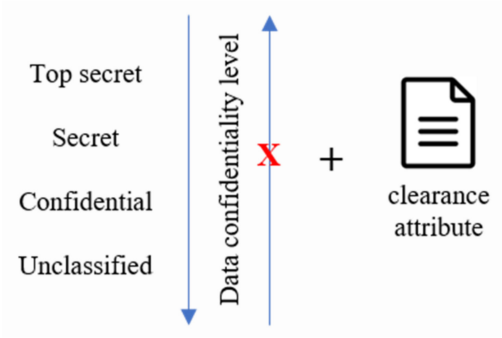

Figure 3. Implementation of the MAC model.

Role-based access control (RBAC) is an access management policy that groups each user by role, without setting access rights for a principal user. It contains structures similar to those in DAC, which allow consistent access control; however, there is no separate security policy for objects, and poor role assignment can expose objects to unauthorized subjects [8]. Table 1 lists the differences between the three above-mentioned models.

Table 1. Comparison of different access management policy models.

Model	DAC	MAC	RBAC
Method	gives rights to users	provides security clearances	works by segregating roles
Integrity	no	yes	yes
Prone to malicious attacks	yes	yes	no
Authorization management	no	no	yes
Dynamic Environment	no	no	no
Flexibility	high	low	high

2.4. Security-Enhanced Linux (SELinux)

SELinux is a Linux security module that applies a MAC-based security policy by improving the DAC of the existing Linux OS. SELinux is a kernel-level security module in which a secure kernel called Flask, developed by NAS in the United States, is ported

to the Linux OS and provides access control by combining information such as the user identity and the user's role, type, and level. The default policy in SELinux does not allow all processes to access files and folders. Each process, file, and folder is assigned a unique security label and security level, and read and write are allowed only if the security level of the subject and object match. The clearance of the security label ranges from 0 to 15, and the security label ranges from A to F.

When a file or folder is accessed during a process, all access decisions are first consulted upon with the DAC and then with the MAC (SELinux). If an action is denied in the DAC, SELinux (MAC) is not consulted, and the action is denied. The clearance security rule is shown in Figure 4. If the object has a higher clearance than the subject, read is denied because no read-up is applied, and if the object has a lower clearance than the subject, write is denied because no write-down is applied. If the subject and the object have the same clearance, read/write is allowed [9].

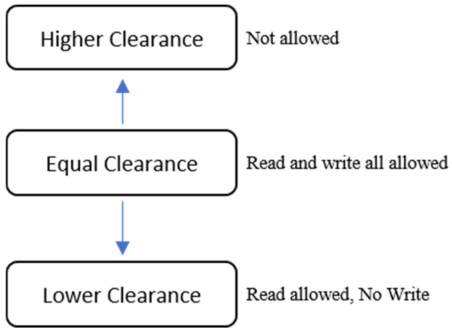

Figure 4. Clearance security rule.

The security label rule is shown in Figure 5. If the subject belongs to the same category as the object, the subject complies with no write-down. Furthermore, it is designed to configure various security labels by combining the clearance and category of the security label.

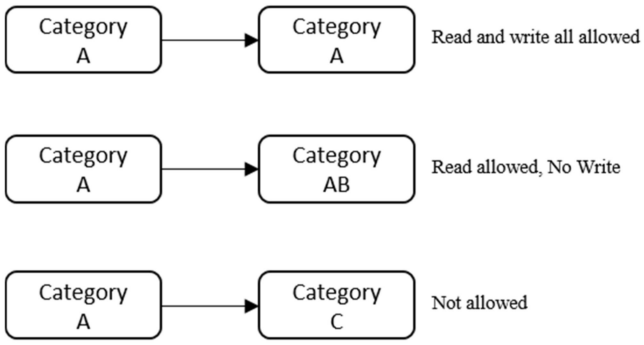

Figure 5. Security label rule.

If the DAC allows an action, the decision is sent to SELinux for a MAC to check the security label and level of the subject and the object. If the subject has a higher security level than the object, only read is allowed and write is denied, thereby reinforcing the security of the Linux OS.

2.5. ADSes

ADSes are not well-known features and were included primarily to provide compatibility with files in the Macintosh file system. Each file has at least one data stream; an ADS allows files to contain more than one stream of data. In Windows, this default data stream is called $DATA. Windows uses file streams for some of its record-keeping purposes, but ADSes can also be used to hide data. A file held in the ADS of another file has no icon of its

own and is not displayed to the user by Windows. Nonetheless, a user can still access a file placed in an ADS and even run it directly from the ADS without having to extract it from its hidden location. The ADS has been given a bad reputation because its capability to hide data from a user on his/her own computer has been abused by the authors of malware in the past. Adversaries may store malicious data or binaries in the ADS area rather than directly in files. This may be done to evade defenses such as static indicator scanning tools and antiviruses [10,11].

3. Design and Implementation

This section describes the design and implementation of the MAC-based file system filter drivers. The components of the proposed system and the features of each component, as well as the manner in which MAC-based systems can behave and enhance security, are described.

3.1. Overview

In the security system proposed in this study, the MAC-based file system filter driver comprises three main components. Figure 6 shows the overall design and implementation of the MAC-based file system filter driver. The "policy setting" component establishes and manages security policies and stores security labels in the stack of filter drivers in the form of files and processes. "Access control" components are responsible for determining whether access control is based on security labels when the principal approaches an object. We determine access control using policy comparison algorithms by identifying the policies of both the principal and the object, which are the access request file policies. The "log management" component stores the events generated by the file access control process in a text file format.

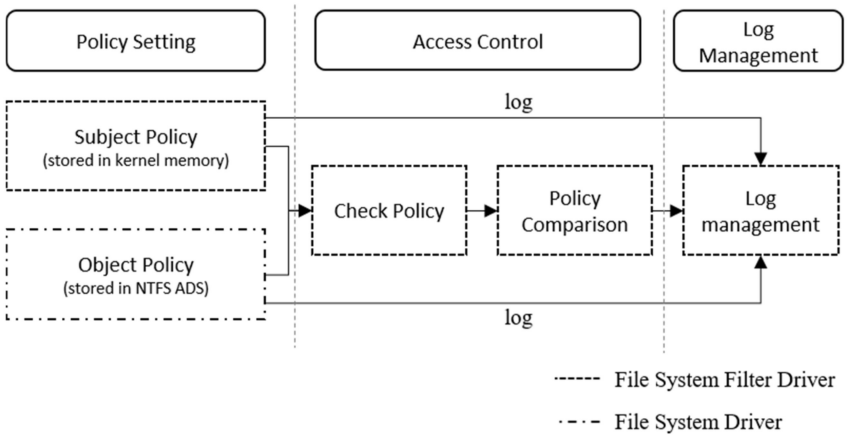

Figure 6. Design and implementation of the MAC-based file system filter driver.

3.2. Policy Setting

Policy settings are responsible for establishing, storing, and managing the security policies required for access control. To store and manage the security policies of the subjects and objects, the principal's security policy is stored in the kernel memory area of the file system filter driver using the kernel port, and the object's security policy is stored in the ADS area of the NTFS file system. The object's security policy settings first record the policy in a text file. After the security policy is completed, the file information is recorded through the kernel port to the filter driver memory and stored in the ADS area of the NTFS file system, which ultimately completes the security policy setting. As the security policy of the object is stored in the ADS area, the existence of the file cannot be known through a general file-explorer program; thus, the security of the policy file is ensured. The ADS area is available because it can be reliably accessed by the file system driver; therefore, a

separate speed can be used (Figure 7). In this study, an object's policy was given a value from 1 to 9 for clearance and a value from A to E for category.

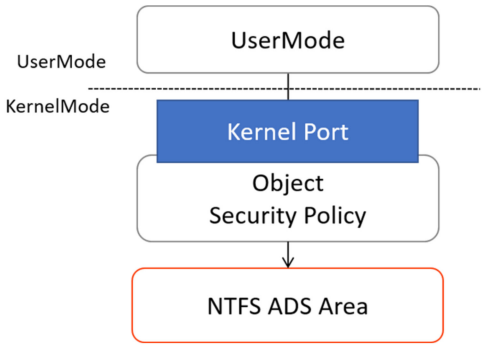

Figure 7. Object's policy setting through a kernel port.

A subject's security policy setting is implemented in the form of a process because the subject accesses the object, and the process accesses the file. Therefore, the file corresponding to the subject is first executed and the security policy is set in the state of being a process. The configured policy is sent to the file system filter driver through the kernel port. The most important parts of the subject's security policy are the security clearance and category (Figure 8). In this study, the subject's policy was given a value from one to nine for clearance and from A to E for category.

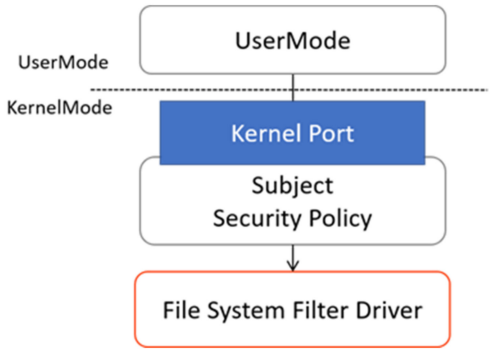

Figure 8. Subject's policy setting through a kernel port.

To use the kernel port, we used the Kernel API provided by the Windows Driver Kit. The name of the port was specified using the POBJECT_ATTRIBUTES structure. This port allows security policies entered in the user mode to be stored in the stack memory within the filter driver. In the filter driver, the stored information is passed back to the file system driver and mapped to the ADS region of each file. The subject's security policy consists of the process ID, level of access (clearance), and category. The object's security policy consists of the path where the file is stored, password for accessing the file, level of access (clearance), category, and the owner of the file.

3.3. Access Control

The main access control process compares the stored security policies when the object file is accessed to determine whether access is allowed. The file system filter driver operates at a higher level than the file system driver, and it performs access control by extracting the security policy of the principal stored in the kernel memory area and that of the object stored in the ADS area (Figure 9).

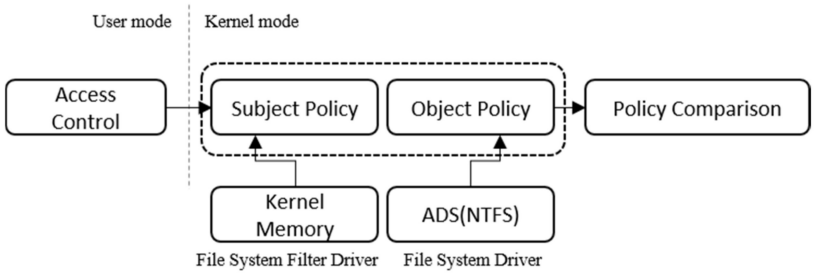

Figure 9. Fetching and comparing the security policies.

In access control, the parent process becomes the subject, and the file accessed by the parent process for execution becomes an object. For a subject to access an object, the security labels of each are compared, and access control is determined according to the Bell–LaPadula (BLP) security model. The security policy in this study comprises a combination of security level (clearance) and protection category (category), and it is composed according to two rules applied in the BLP model, namely, No Read-Up and No Write-Down. No Read-Up means that the subject cannot read or write to objects with a higher security level, and No Write-Down means that a subject cannot write to an object with a security level lower than that of the subject. If the subject and the object have the same security level, both read and write are possible.

This principle is used to block the flow of information from a high level to a low level in advance to maintain the confidentiality of information and underlies the model that forms the basis of the forced access control method (Figure 10). The No Read-Up and No Write-Down rules are applied in the same way as the protection category rules. "No Read-Up" is impossible to read if the subject's category belongs to the object's category; "No Write-Down" cannot be written if the object's category belongs to the subject's category. Read and write are possible only when the categories of the subject and object are the same. The final access control determination is sent to the file system driver and recorded as a log. Figure 10 shows a flow chart of the security label comparison algorithm and the access control process using the BLP security rule. First, it compares the security categories of the subject and the object. If the subject's category belongs to or is the same as the object category, it proceeds to a security level comparison. Upon comparing the security levels of the subject and the object, if the subject's level is higher than the object's level, the No Write-Down rule is applied, and if the subject's level is lower than the object's level, the No Read-Up rule is applied. If the subject and object are of the same class, all access is allowed. Table 2 lists all the algorithm parameters that were varied in this study.

Table 2. Algorithm parameters.

Parameters	Description
Subject	Users or processes accessing system resources
Object	system resources
Bell–LaPadula model	A state machine model that enforces confidentiality to prevent unauthorized access to privileged secrets.
Read-Up	Only read access to objects whose security level is below the subject's current clearance level
Write-Down	Only write access to objects whose security level is higher than its current clearance level
Clearance	degree of security
Category	specific department of security
Subject-CA	subject category
Object-CA	object category
Subject-CL	subject clearance
Object-CL	object clearance

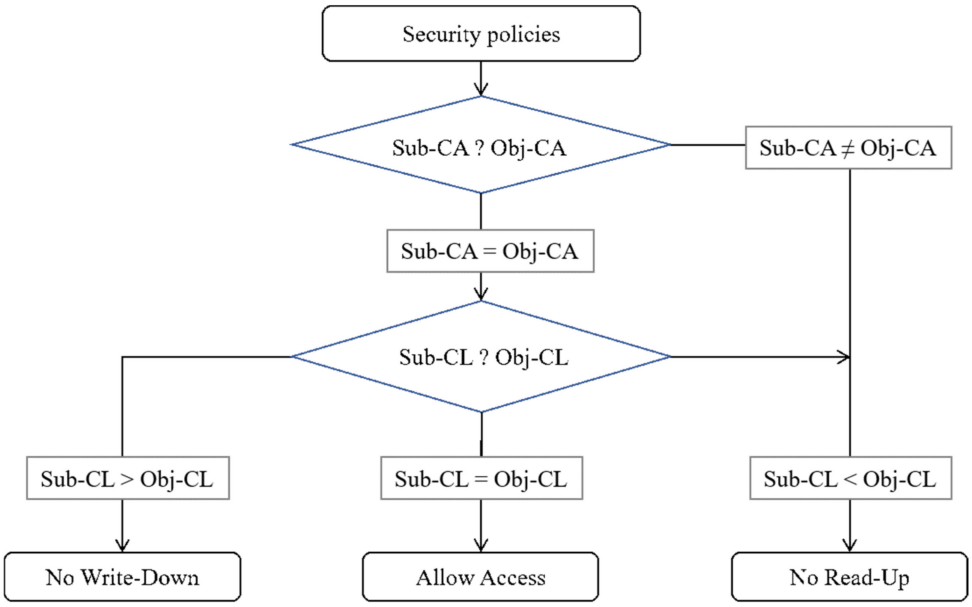

Figure 10. Security comparison algorithm.

To implement policy setting and access control, the information requested by the user application is first transferred to Kernel32.dll. After the requested information is delivered to the I/O manager, it is converted to an IRP structure and is delivered to the file system filter driver before being sent to the file system driver. The file system filter driver extracts the security policy of the subject (parent process) and the object (file requested to be executed), and then extracts the result of access control using a policy comparison algorithm. The extracted result is sent to the file system driver and recorded as a log. The entire process is shown in Figure 11.

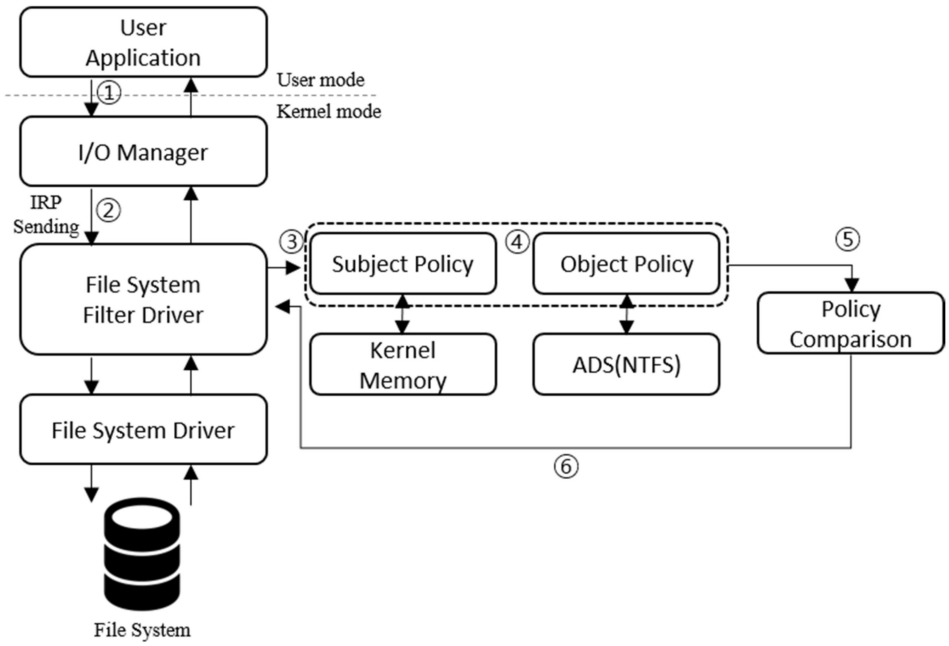

Figure 11. File access control and policy comparison process via the IRP. ① User requests file execution through process. ② File I/O manager passes IRP to filter driver. ③ Security policy of the subject chain process is checked in kernel memory. ④ Security policy of the object execution request file is checked in the ADS area. ⑤ Access permission and restriction are determined using the security policy comparison algorithm. ⑥ Value of file access status is returned to the file system.

3.4. Log Management

Log management stores the policy information written in the policy setting and the event information generated during access control in the file system filter driver kernel memory. The stored logs are passed to the user mode through the kernel port at the request of the user and are then stored in a file format. In the policy setting stage, when policy information is recorded in the kernel memory, a log related to the policy is recorded; when an "access denial" event occurs in the access control stage, the information about the subject and the object is recorded in the log.

The main components of the log are the time when the event occurred, the access control policy, a file containing the names of the subject and the object, and the reason for which access was denied. Figure 12 shows the log management configuration of the proposed system.

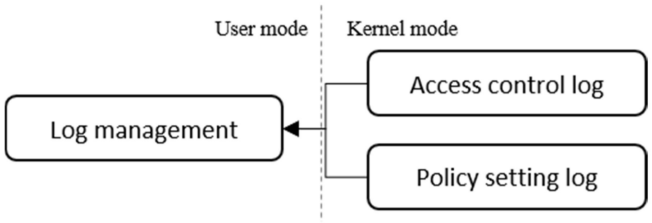

Figure 12. Log management configuration of the proposed system.

4. Experimentation

We built a test environment and conducted access control tests to verify whether the proposed system can prevent vulnerability. Two systems, namely a DAC-based system using the native Windows Embedded OS and the proposed MAC-based system, were compared in the tests. The evaluation method and goal of the experiment were as follows: assuming that the administrator's privileges have been stolen by a hacker, a random program called malware.exe is executed in the existing DAC-based system and the proposed MAC-based system. In the existing DAC-based system, only the clearance and category of cmd.exe corresponding to the subject are checked, so the malware.exe file will be executed. In the proposed system, the clearance and category of malware.exe corresponding to the object must be checked, and even a hacker with administrator rights cannot know the clearance and category of the object hidden in ADS, so execution will be rejected. Table 3 and Figure 13 present the complete test environment.

Table 3. Test environments.

	DAC-Based System	**Proposed System**
OS	Windows Embedded OS POS Ready7 x64	
CPU	Intel Core2 i5-3470 @3.6 GHz (VMWare)	
RAM	2 GB	
Installed security system	Antivirus system	MAC-based file system filter driver
Purpose	Whitelist-based test	MAC-model-based test

4.1. DAC-Based System

The first scenario is an experiment in an environment in which a DAC-based whitelist security system is installed on an existing Windows Embedded OS (Figure 14). First, the malware.exe file is stored in the Windows Embedded OS through the command prompt. The attacker seizes access to the existing security solution and includes the malware.exe file in the whitelist. The attacker then executes the malware.exe file through the command prompt, which infects the system with malware. This scenario shows that the existing security systems can be neutralized if the principal's security policy is removed.

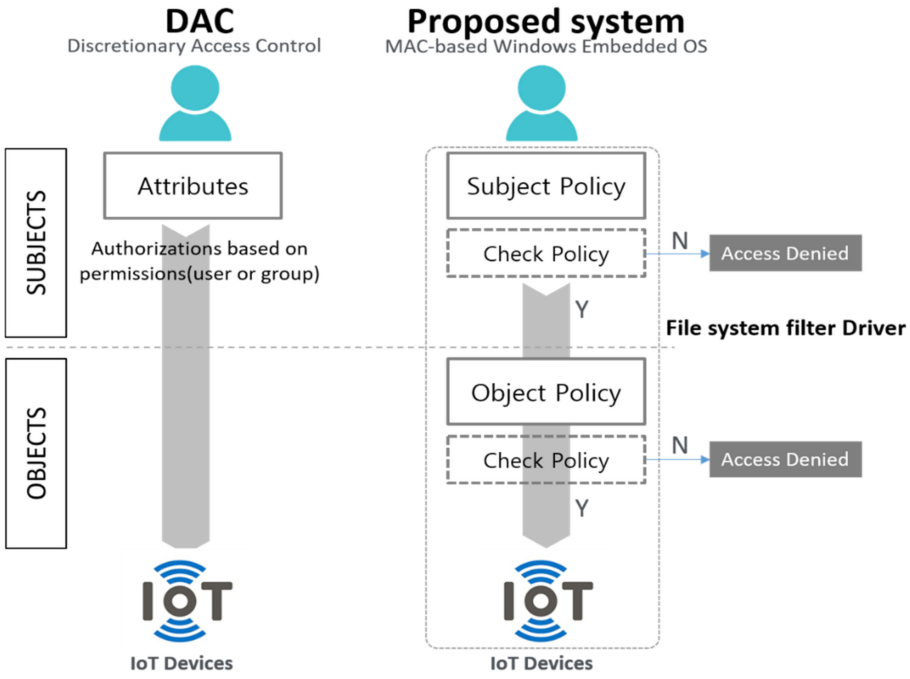

Figure 13. DAC-based whitelist security system.

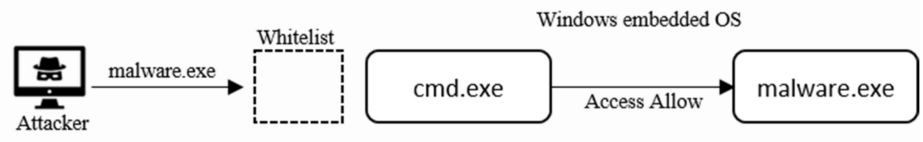

Figure 14. DAC-based whitelist security system.

4.2. Proposed MAC-Based System

The second scenario involves the installation of a MAC-based whitelist security system on the Windows Embedded OS proposed in this study (Figure 15). Similar to Scenario 1, the malware.exe file is stored by the hacker. As it is based on the MAC model, the malware.exe file is stored, and the clearance and category values are set by default. These two sets of setup information are stored in the ADS area of the NSTF file system and are automatically encrypted during the save phase. Even if the clearance and category values of the main cmd.exe file are set to the highest rating, the hacker does not know the object's, i.e., the malware.exe file's, clearance and category values, and the subject's security policy does not match; thus, access is automatically denied.

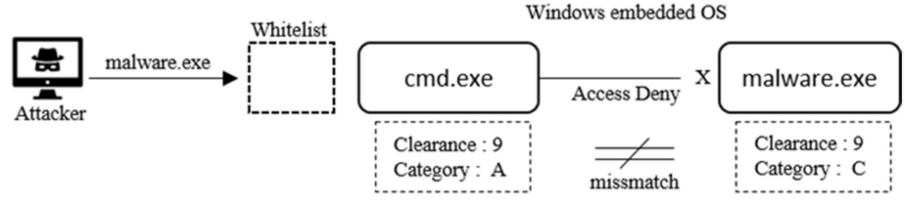

Figure 15. MAC-based whitelist security system.

As shown in Figure 16, the principal's security policy confirms that even if the clearance is set to 9 and category to A, it is rejected because it is different from the object's default security policy. For hackers to access the security policy of objects, both the system administrators' authentication and authentication using passwords that are automatically granted when stored in ADS are required.

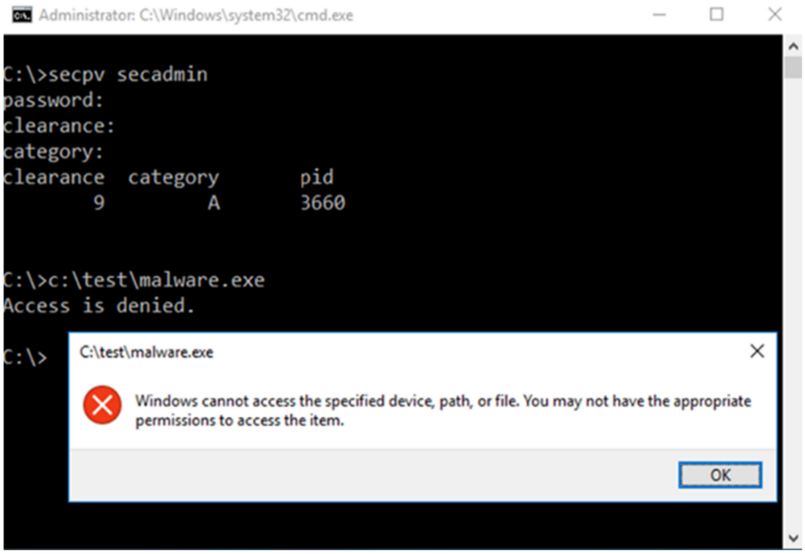

Figure 16. MAC-based whitelist security system execution results.

5. Conclusions

In this study, we designed and implemented a Windows Embedded OS security system based on the MAC model. The proposed system establishes permissions for both the subject and the object, and limits access by comparing the policies of the subject and the object. To verify that the proposed system can avoid the vulnerabilities of DAC-based access control, a test environment was built and access control tests were conducted in which the authority levels of both the subject and the object were judged and security labels were compared to confirm access control. The proposed access control method is similar to the existing SELinux method. However, SELinux is a security module added to the Linux OS. Therefore, in the access control stage, the DAC model installed by default on the Linux OS is first applied to SELinux, and the SELinux security module is applied secondarily. Unlike the SELinux security module, the proposed MAC-based file system filter driver applies MAC-based access control first in the file system filter driver, and its security policy management is designed to be simpler than that of SELinux. Using the methods proposed in this study can improve the security of Windows Embedded OSes.

Author Contributions: Research paper for access control methods, Y.S.; design and experiment, C.C.; study supervision, Y.W. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported by the Institute for Information and Communications Technology Planning and Evaluation (IITP) grant funded by the Korean Government (MSIT) (No.2019-0-01343, Training Key Talents in Industrial Convergence Security).

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Catak, F.O.; Mustacoglu, A.F. Distributed denial of service attack detection using autoencoder and deep neural networks. *J. Intell. Fuzzy Syst.* **2019**, *37*, 3969–3979. [CrossRef]
2. Catak, F.O.; Ahmed, J.; Sahinbas, K.; Khand, Z.H. Data augmentation based malware detection using convolutional neural networks. *PeerJ Comput. Sci.* **2021**, *7*, e346. [CrossRef] [PubMed]
3. MSDN. File System Filter Driver Design Guide. Available online: https://docs.microsoft.com/en-us/windows-hardware/drivers/ifs/ (accessed on 8 September 2021).
4. MSDN. Overview in the Windows I/O Model. Available online: https://docs.microsoft.com/en-us/windows-hardware/drivers/kernel/overview-of-the-windows-i-o-model (accessed on 8 September 2021).
5. Zhang, C.; Wu, Y.; Yu, Z.; Li, Z. Research and implementation of file security mechanisms based on file system filter driver. In Proceedings of the 2017 Annual Reliability and Maintainability Symposium (RAMS), Orlando, FL, USA, 23–26 January 2017; pp. 1–6. [CrossRef]

6. Ausanka-Crues, R. Methods for Access Control: Advances and Limitations. Available online: https://www.cs.hmc.edu/~mike/public_html/courses/security/s06/projects/ryan.pdf (accessed on 8 September 2021).
7. Bai, Q.H.; Zheng, Y. Study on the access control model. In Proceedings of the 2011 Cross Strait Quad-Regional Radio Science and Wireless Technology Conference, Harbin, China, 26–30 July 2011; Volume 1, pp. 830–834.
8. Bell, D.E.; LaPadula, L.J. *Secure Computer Systems: Mathematical Foundations*; Mitre Corp: Bedford, MA, USA, 1973; pp. 74–244. Available online: http://www-personal.umich.edu/~{}cja/LPS12b/refs/belllapadula1.pdf (accessed on 8 September 2021).
9. Red Hat Enterprise. Linux 7 SELinux User's and Administrator's Guide. Available online: https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/7/pdf/system_administrators_guide/red_hat_enterprise_linux-7-system_administrators_guide-en-us.pdf (accessed on 8 September 2021).
10. Meshram, B.B.; Patil, D.N. Digital forensic analysis of hard disk for evidence collection. *Int. J. Cyber Criminol.* **2018**, *7*, 100–110. [CrossRef]
11. Pittman, R.D.; Shaver, D. Windows forensic analysis. In *Handbook of Digital Forensics and Investigation*; Academic Press: Cambridge, MA, USA, 2010; pp. 209–300.