J Inf Process Syst, Vol.17, No.4, pp.834~850, August 2021
https://doi.org/10.3745/JIPS.03.0165

ISSN 1976-913X (Print)
ISSN 2092-805X (Electronic)

JOURNAL OF INFORMATION PROCESSING SYSTEMS JIPS

Security Improvement of File System Filter Driver in Windows Embedded OS

Yeon Sang Seong*, Chaeho Cho*, Young Pyo Jun**, and Yoojae Won*

Abstract

IT security companies have been releasing file system filter driver security solutions based on the whitelist, which are being used by several enterprises in the relevant industries. However, in February 2019, a whitelist vulnerability was discovered in Microsoft Edge browser, which allows malicious code to be executed unknown to users. If a hacker had inserted a program that executed malicious code into the whitelist, it would have resulted in considerable damage. File system filter driver security solutions based on the whitelist are discretionary access control (DAC) models. Hence, the whitelist is vulnerable because it only considers the target subject to be accessed, without taking into account the access rights of the file target object. In this study, we propose an industrial device security system for Windows to address this vulnerability, which improves the security of the security policy by determining not only the access rights of the subject but also those of the object through the application of the mandatory access control (MAC) policy in the Windows industrial operating system. The access control method does not base the security policy on the whitelist; instead, by investigating the setting of the security policy not only for the subject but also the object, we propose a method that provides improved stability, compared to the conventional whitelist method.

Keywords

Access Control, File System Filter Driver, Mandatory Access Control, Whitelist, Windows Embedded OS

1. Introduction

The advancements in IT technology have enabled all the industrial sectors to establish smart industrial environments. Recently, several industrial operating system (OS) functions such as task processing, user interface, network and communication, and services have been improved, providing numerous opportunities in various fields. Among the industrial OSs, Microsoft Windows embedded OS is used in a considerable proportion of industrial devices [1]. Thus, the Windows embedded OS has been targeted by hackers. In an incident at Target Corporation in the United States in December 2013, a hacker inserted malicious code in the network where the point-of-sale (POS) system was installed through the internal business network and executed it in the POS system to capture crucial information [2,3]. Although anti-virus software has been used to address these threats, it is unsuitable for tackling new types of malicious code, and there is a possibility of computer resource-related issues in low-end systems or environments where the network speed is limited. Therefore, domestic and foreign IT security companies have been

Manuscript received March 20, 2020; first revision September 15, 2020; accepted October 11, 2020.

Corresponding Author: Yoojae Won (yjwon@cnu.ac.kr)
* Dept. of Computer Science Engineering, Chungnam National University, Daejeon, Korea (dustkdwkd@naver.com, greatopen@cnu.ac.kr, yjwon@cnu.ac.kr)
** Software Division, Yonsei University MIRAE Campus, Wonju, Korea (ypjun@yonsei.ac.kr)

releasing file system filter driver security solutions based on the whitelist, which are being used by several enterprises. However, in February 2019, a whitelist vulnerability was discovered in Microsoft Edge browser, which allows malicious code to be executed unknown to users. For this vulnerability, CVE 2019-0641, the Microsoft Security Response Center did not respond to questions on the whitelist but issued a statement that the problematic part would be removed at earliest [4]. If a hacker had inserted a program that executed malicious code into the whitelist, it would have resulted in considerable damage. File system filter driver security solutions based on the whitelist can cause serious problems, if the access rights of the whitelist are captured by hackers. These security solutions are based on discretionary access control (DAC) models, which are vulnerable because they only consider the target subject to be accessed, without considering the access rights of the file target object [5].

Therefore, there is a need for a security system with enhanced confidentiality to compensate for the vulnerability of security solutions based on the whitelist. The whitelist method is efficient because conventional file system filter driver security systems based on the whitelist execute only limited programs due to the nature of the industrial environment. In addition, this method is useful because the industrial environment generally does not require high computer performance, resulting in high memory efficiency. Furthermore, a security policy using the whitelist is effective from the administrator's point of view because the security administrator manages only the whitelist policy. However, if ownership of the whitelist is compromised, a fatal flaw can arise in the security system. As a result, crucial information in the system can be captured. Therefore, in order to prevent this issue, it is necessary to strengthen the security system of the local internal scope of an industrial system. In this study, we propose an industrial device security system for Windows to address this vulnerability, which improves the security of the security policy by identifying not only the access rights of the subject but also those of the object through the application of the mandatory access control (MAC) policy in the Windows industrial operating system. We do not base the security policy on the whitelist but set it for both the subject and object. The security policy sets a security label for the subject and object, and allows and denies access by comparing their security labels. Such security label comparison involves three steps. In step 1, the security label is extracted from the policy of the subject and object, respectively. In step 2, with the extracted security label, access permission or denial is determined through a security label comparison algorithm. In step 3, the messages for which access is allowed or denied are sent to the file system driver. Even if the attacker captures the security policy present in the whitelist, this MAC-based policy prevents him/her from running the malicious code required for attacks, unless the security policy of the object is captured. As there is a security policy for both the subject and object, it is possible to improve the security with this dual security, compared to the conventional whitelist method.

2. Related Work

2.1 Windows Embedded File System Filter Driver

2.1.1 File system filter driver overview

The Windows OS file system filter driver refers to a driver that processes input/output (I/O) requests between the highest-level driver and lower-level drivers in the driver hierarchy. This is also applicable to Windows embedded OS; as this OS is inherited from the standard OS, the filter driver structure is also inherited. In addition, the filter driver sends and receives I/O requests in the middle; hence it is possible to complement the functions provided by the file system filter driver or disk driver or add new functions. For example, if we create a file access control driver, we should write a file system filter driver because the OS provides the file system driver. Most drivers used for file access control or Internet protocol (IP) addresses and port control and monitoring are created using filter drivers. The file system filter driver, as its name indicates, is a filter driver used for the file system. Precisely, it is a driver that is located over the file system driver, which is a high-level driver, and is used to implement I/O monitoring and file access control by receiving I/O requests from user mode applications [6,7].

2.1.2 Structure of file system filter driver

When a filter driver is newly placed on the stack of the file system driver, it receives all the I/O request packets (IRPs) that the I/O manager sends to the file system drive, after which it performs the desired processing and delivers them to a low-level filter driver or the file system driver (Fig. 1). Here, the file system driver cannot be aware of the operation performed by the high-level filter driver, due to its structure. Therefore, although the file system driver receives the IRP, whose information has been changed by the high-level filter driver, it performs the IRP as delivered directly by the I/O manager. Because the filter driver can control the status value of the low-level driver or the results of the processing requests, it can perform additional functions and file access control [8].

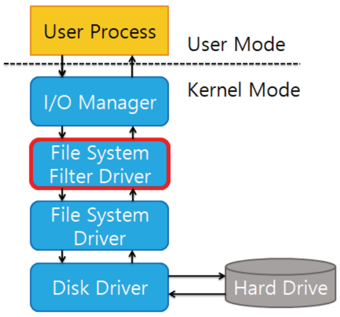


Fig. 1. File system filter driver structure.

2.1.3 IRP structure

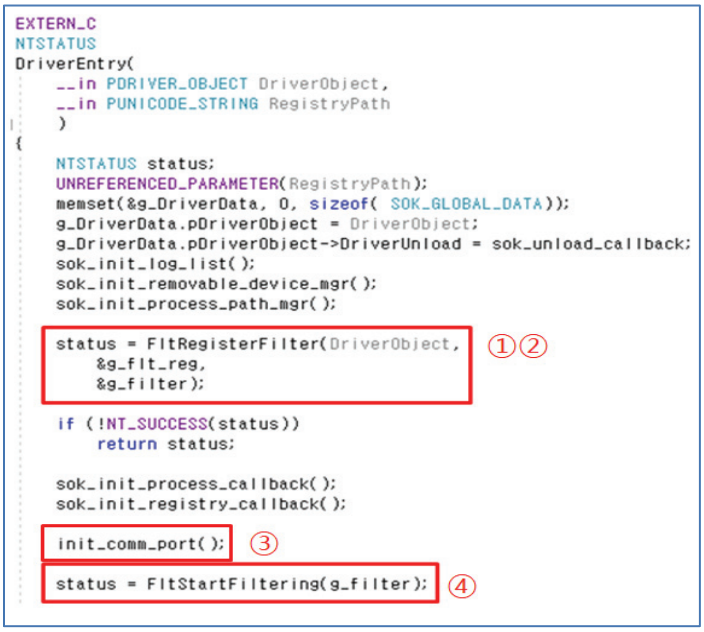
IRP is a packet that contains all the information on the I/O requests that the driver should process and is the name of a structure used in driver programming. All the drivers running in the kernel have a dispatch routine to communicate with the hardware or other drivers. This dispatch routine is called when there is an I/O request, and it returns the result after appropriate processing according to the requested content. When an I/O request sender fills the IRP structure with the request content to send the IRP, the IRP structure is sent as an argument, when the dispatch routine is called. The IRPs managed by the I/O manager form an I/O stack for each layered driver, to deliver the information suitable for each process [9]. The following is a description of the IRP structure used in the proposed security system.

2.1.4 Development of file system filter driver

Prior to the minifilter method, the legacy method of developing a filter driver utilized the IRP hooking method. This hooking method had been phased out because Microsoft supported the minifilter method, and driver development has been easier with the provision of a kernel application programming interface (API). Consequently, it provided security against Rootkit, which is a method of hacking kernels using hooks. The development of the filter driver involves four steps.

As seen in Fig. 2, the driver is registered, and the altitude is set in step 1. An IRP routine is registered in the FltRegisterFilter structure for the part required for the driver function, and the stack memory to store the security policy is configured in step 2. The kernel port is set to communicate with the user mode in step 3. Finally, the filter driver is loaded in step 4.

In step 3, it is developed to control the file input and output, and registered in the required IRP structure.



```
EXTERN_C
NTSTATUS
DriverEntry(
    __in PDRIVER_OBJECT DriverObject,
    __in PUNICODE_STRING RegistryPath
    )
{
    NTSTATUS status;
    UNREFERENCED_PARAMETER(RegistryPath);
    memset(&g_DriverData, 0, sizeof( SOK_GLOBAL_DATA));
    g_DriverData.pDriverObject = DriverObject;
    g_DriverData.pDriverObject->DriverUnload = sok_unload_callback;
    sok_init_log_list( );
    sok_init_removable_device_mgr( );
    sok_init_process_path_mgr( );

    status = FltRegisterFilter(DriverObject,        ①②
        &g_flt_reg,
        &g_filter);

    if (!NT_SUCCESS(status))
        return status;

    sok_init_process_callback( );
    sok_init_registry_callback( );

    init_comm_port( );   ③

    status = FltStartFiltering(g_filter);   ④
```

Fig. 2. Filter driver, DriverEntry.

2.2 File Access Control Model

In file access control, the target subject executes the object, and the non-target object is executed by the subject. In the Windows OS environment, the parent process becomes the subject and the file requested to be executed becomes the object. Further, we examine the file access control model related to this study.

2.2.1 Discretionary access control

DAC is a method of restricting access to the object based on the identifiers of the subject or groups to which it belongs [10]; hence, access control is discretionarily performed by the owner of the object. Therefore, as long as the subject has certain access permissions, i.e., unless the MAC model is maintained, it can hand over its permission to any other subject. The general attributes of the DAC model are summarized below. The drawbacks inherently inherited from the DAC model are as follows: First, control is based entirely on the identity of the subject because of the nature of the DAC [11]. Next, the identity of the subject is crucial; therefore, information in the system is exposed, if an action is performed using the identity of another person [12,13].

In Fig. 3, the subject has read, execute, and write rights to the object in the security policy. When the subject has access to the object, access control is determined based on the rights of the subject. However, if the whitelist security policy set with the rights of the subject alone allows attackers to obtain access to the system, the entire security system may be compromised. Domestic and foreign industrial system security products with this vulnerability manage the security policy using a whitelist-based DAC model. Unless this drawback is addressed, there is a concern that system information may be exposed.

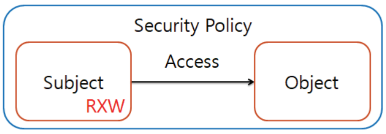


Fig. 3. Implementation of the DAC model.

2.2.2 Mandatory access control

MAC is a method of restricting access by checking not only the rights of the subject, but also those of the object. Access control cannot be managed discretionarily by the owner of the object and is enforced because the access rights for the object are checked [14].

As observed in Fig. 4, MAC is a method of controlling access according to the relevant level or category by setting the security label between the subject and object in the security policy, and assigning the level and category to each user in the label. It is implemented in a kernel based on the Bell-LaPadula (BLP) model, and there are five levels and 64 categories in a label. When comparing the MAC mechanism to real life, it is to implement a secret handling authority. By dividing the user levels, it solves the Trojan horse problem in the levels. The basic principle is that the subject of the same level can read/write the object of the same level, according to the user level. It is a method of controlling access according to the rule that the subject of the relevant level cannot read (no read-up) the object of a higher level, and the subject of the relevant level cannot write (no write-down) to the object of a lower level. This complies with the BLP model, which can control hierarchical relationship access. However, because access control thorough the mutual relationships between different groups within the same level is required, it performs access control by category such that only the subject belonging to the relevant category can get access to the relevant object. In the BLP model, the levels are organized hierarchically, and the categories are organized within each level. The MAC information in the security database comprises the identifiers corresponding to the object, level, and category. After finding the security label using the identifier information of the object that a process wants to access, it checks the access rights by comparing the level and category content of the processor [15].

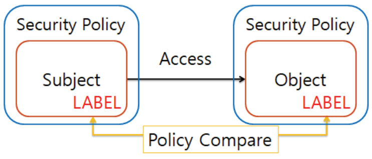


Fig. 4. Implementation of the MAC model.

2.2.3 Policy-based access control

Policy-based access control was proposed to control the access of task computing in a ubiquitous computing environment. It is a method of handling access to resources through a network and sets the security policy to evaluate requests for network resources [16,17]. It is not based on the subject but on the data evaluated according to the resource requests for forming the security policy in a ubiquitous computing environment. However, as the security method is a single security type, the security system is vulnerable, if a hacker captures the security policy.

2.3 Related Security Systems

2.3.1 SELinux (Security-Enhanced Linux)

In Linux OS, a Linux kernel security module provides a mechanism for supporting the access control security policy including the MAC model to strengthen the security of a file system that adopts the DAC model. SELinux is a security system that adopts the MAC model applied with the BLP model, as depicted in Fig. 5. The concept of no read-up and no write-down is applied to SELinux, and read/write rights are allowed only when the label is the same [18].

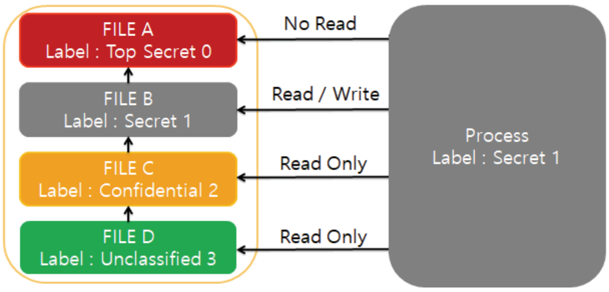


Fig. 5. SELinux security label.

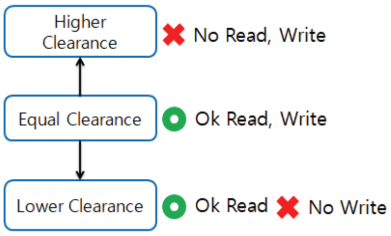


Fig. 6. Multi-level security.

The SELinux security label is divided into multi-level security (MLS) and multi-category security (MCS). MLS is a security level (clearance), which allows the subject to access objects with the same security level as that of the subject, restricting access to different security levels. In the proposed security system, the security label is composed of a combination of the security level and category. The clearance of the security label ranges from 0–15; the lower the level, the higher is the security. The clearance security rule is shown in Fig. 6. If the object has higher clearance than the subject, read is denied because no read-up is applied. If the object has lower clearance than the subject, write is denied because no write-down is applied. If the subject and object have the same clearance, read/write is allowed.

The MCS category security rule is illustrated in Fig. 7. It comprises categories A–F and is set to restrict access of unequal categories. If the categories are the same, access is allowed. However, if they are not, access is restricted according to no read-up. If the subject belongs to the category of the object, it complies with no write-down [19]. Furthermore, it is designed to configure various security labels by combining the clearance and category of the security label.

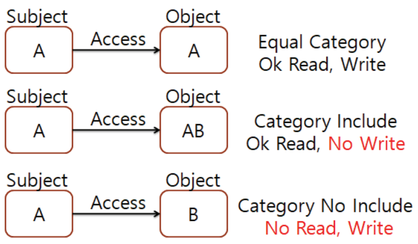


Fig. 7. Multi-category security.

2.3.2 Ahnlab, TrusLine

Ahnlab “TrusLine” is a whitelist-based security solution optimized for industrial control systems that use only designated programs because of their high sensitivity to stable operation. It includes product features such as Windows embedded OS support, unauthorized application installation control, and IP/Port blocking external storage. Furthermore, there are features such as blocking the path of infection by preventing automatic device execution, providing independence from OS patch updates, minimizing the memory/CPU utilization of control systems, and managing policies through the DAC security policy model [20].

2.3.3 McAfee, Application Control

It is a whitelist-based control system security solution that supports Windows embedded OS and Linux OS. It generates the whitelist dynamically using a trust model, provides DAC policy-based ease of use, blocks the execution of illegal script drivers (memory monitoring), and prevents changes to the configuration of the registry key requests [20].

2.3.4 Industrial Defender, HIPS

The CoreTrace Bouncer 6-based host intrusion prevention system (HIPS) is an intrusion prevention system for whitelist-based control systems, which has been provided to global smart grid power equipment companies such as ABB, GE Energy, and Itron. It supports Windows embedded OS, Linux, Unix, controls whitelist-based application installation operation, minimizes network traffic impact, and adds the whitelist dynamically when updating software. It also adopts the DAC model [20].

2.4 CVE-2019-0641 Vulnerability

A Google researcher discovered security vulnerability in the handling of the whitelist by Microsoft Edge. This vulnerability allows site addresses to be loaded in the Edge browser through a secret whitelist that has no user interaction. An attacker can ignore the whitelist created by the user and randomly load the content. The same vulnerability is present in Windows Embedded 10, where Edge can be used in the embedded OS. Microsoft Security Response Center provided security updates to control this issue [21]. However, whitelist-related security issues are likely to continue, unless the structural weakness of the whitelist is addressed.

3. Design and Implementation of a File System Filter Driver Applied with MAC

3.1 Overview

The security system of the whitelist MAC model proposed in this study comprises three functions. Initially, a file system filter driver should be implemented to include these functions. For implementation, we refer to the file system filter driver described in Section 2. The components include policy setting, access control, and log management

Fig. 8 shows the overall structure of the design and implementation of the file system filter driver based on the MAC model.

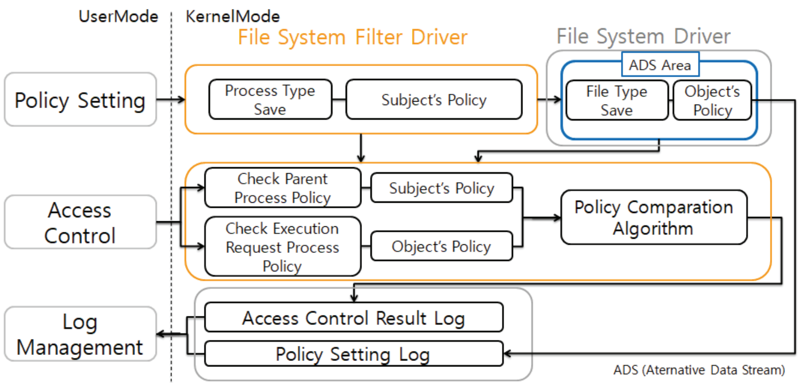


Fig. 8. Overall structural diagram.

3.2 Policy Setting

In order to set the policy, communication between the kernel mode and user mode should be established, for which a kernel port is required. Based on the implementation described in Section 2.1.4, a kernel port is configured to store the security policy in the file system filter driver and an alternative data stream (ADS) area. As shown in Fig. 9, the kernel port is required for policy setting. The policy of the subject is stored in the file system filter driver, and that of the object is stored in the ADS area.

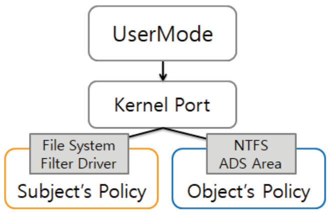
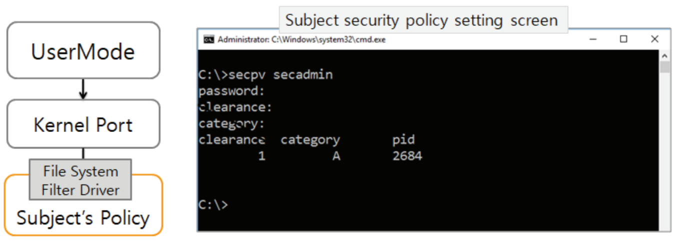


Fig. 9. Policy setting through a kernel port.

3.2.1 Policy setting – process type

Storing the policy in a process type is a function of the subject. Policy setting for the subject is implemented in a process type because the subject has access to the object, similar to file access by the process. The security policy is set by storing it in the filter driver memory through the kernel port. On the left of Fig. 10 is a flow chart that shows the sending of the security policy of the subject to the file system filter driver through the kernel port in the user mode. On the right is a screen, which shows that the security policy is set to the cmd.exe process for clearance_1 and category_A.


UserMode
Kernel Port
File System Filter Driver
Subject's Policy
Subject security policy setting screen
Administrator: C:\Windows\system32\cmd.exe

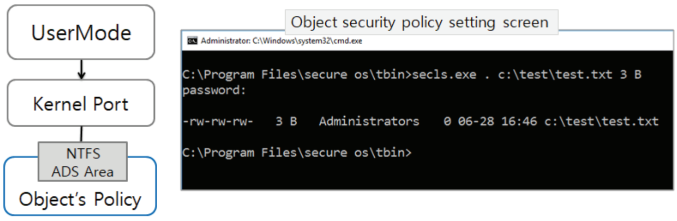
```
C:\>secpv secadmin
password:
clearance:
category:
clearance  category       pid
        1         A       2684

C:\>
```

Fig. 10. Security policy setting of the subject.

3.2.2 Policy setting – file type

Storing the policy in a file type is a function of the object, and the security policy is set in a text file by storing it in the ADS area of the object file through the kernel port. On the left of Fig. 11 is a flow chart that shows the sending of the security policy of the object to the ADS area through the kernel port in the user mode. On the right is a screen, which shows that the security policy is set to the c:\test\test.exe file for clearance_3 and category_B.


UserMode
Kernel Port
NTFS ADS Area
Object's Policy
Object security policy setting screen
Administrator: C:\Windows\system32\cmd.exe

```
C:\Program Files\secure os\tbin>secls.exe . c:\test\test.txt 3 B
password:

-rw-rw-rw-   3 B    Administrators    0 06-28 16:46 c:\test\test.txt

C:\Program Files\secure os\tbin>
```

Fig. 11. Security policy setting of the object.

3.3 Access Control

Access control is a function that determines whether access to the files requested by a Windows user is to be restricted, according to the security policy.

3.3.1 Security label extraction

In file access control, the parent process is the subject and the file requested for execution is the object. For a subject to access the object, the security labels of the subject and object are compared to determine access control according to the BLP security rules. For this, it is necessary to extract the security labels of the subject and object. The subject policy extraction function fetches the security policy from the ADS area set in the policy setting of the parent process, as shown in Fig. 12.

In this study, the security labels are extracted using the IRP_MJ_CREATE routine function described in Section 2.1.4. This function maps the memory to the file system driver to process the file requested for execution. Using this function, the handle value is obtained through the process identifier (PID) of the parent process in the IRP structure, and the security label is extracted by accessing the ADS area of the file with the obtained handle value.

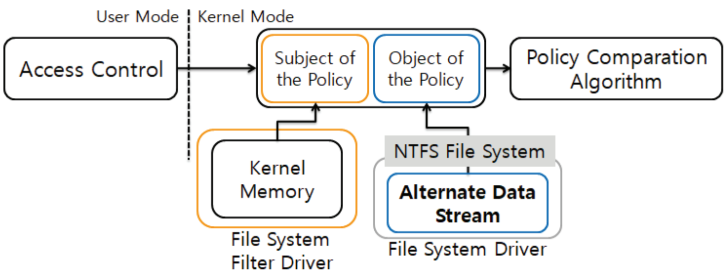
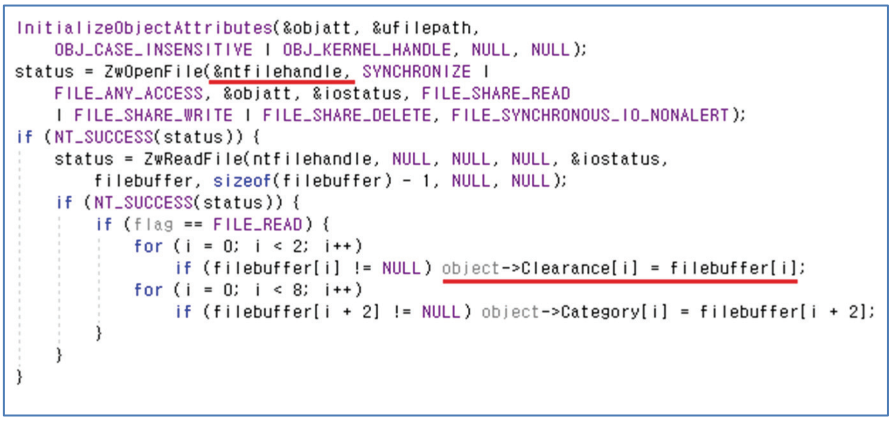


Fig. 12. Security label extraction.

In Fig. 13, by fetching the ntfilehandle structure through the ZwOpenFile function and extracting the security label with the ZwReadFile function, the clearance and category are mapped to the security label and stored in the ADS area. For object policy extraction, the security policy is extracted from the file requested for execution in the user mode, using the same routine function that extracts the subject policy.



```
InitializeObjectAttributes(&objatt, &ufilepath,
    OBJ_CASE_INSENSITIVE | OBJ_KERNEL_HANDLE, NULL, NULL);
status = ZwOpenFile(&ntfilehandle, SYNCHRONIZE |
    FILE_ANY_ACCESS, &objatt, &iostatus, FILE_SHARE_READ
    | FILE_SHARE_WRITE | FILE_SHARE_DELETE, FILE_SYNCHRONOUS_IO_NONALERT);
if (NT_SUCCESS(status)) {
    status = ZwReadFile(ntfilehandle, NULL, NULL, NULL, &iostatus,
        filebuffer, sizeof(filebuffer) - 1, NULL, NULL);
    if (NT_SUCCESS(status)) {
        if (flag == FILE_READ) {
            for (i = 0; i < 2; i++)
                if (filebuffer[i] != NULL) object->Clearance[i] = filebuffer[i];
            for (i = 0; i < 8; i++)
                if (filebuffer[i + 2] != NULL) object->Category[i] = filebuffer[i + 2];
        }
    }
}
```

Fig. 13. Policy extraction.

3.3.2 Security-label comparison algorithm

The security label is registered by adopting the BLP model. It is a combination of clearance and category and abides by the two principles applied in the BLP model. There are no read-up and no write-down, and the proposed security system complies with the same MAC-based security policy. No read-up indicates that the subject cannot read and write simultaneously to an object with higher clearance. No write-down indicates that the subject cannot write to an object with lower clearance. If the clearance of the subject and object are the same, reading and writing are allowed. We applied rules to prevent information flow from high to low clearance for maintaining the confidentiality of the information. No read-up and no write-down are applicable to a category, in the same manner. No read-up indicates that reading is not allowed, if the category of the subject belongs to that of the object, whereas no write-down indicates that writing is not allowed, if the category of the object belongs to that of the subject. If the category of the subject and object are the same, reading and writing are allowed. The flowchart of the security-label comparison algorithm to which these BLP security rules are applied is shown in Fig. 14. It compares the categories of the subject and object. If the category of the subject belongs to the category of the object or is the same as that of the object, it proceeds to clearance comparison. If the category of the subject does not belong to the category of the object, no read-up is applied. Next, it compares the clearance of the subject and object. If the clearance of the subject is higher than that of the object, no write-down is applied. If the clearance of the subject is lower than that of the object, no read-up is applied. If the clearance of the subject and object are the same, all access is allowed. However, if the clearance of the subject belongs to that of the object, no write-down is applied.

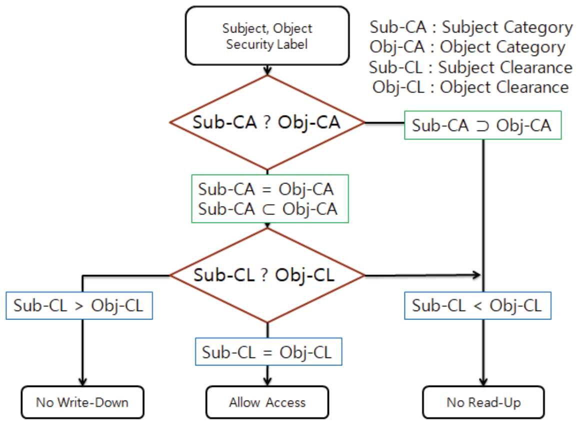
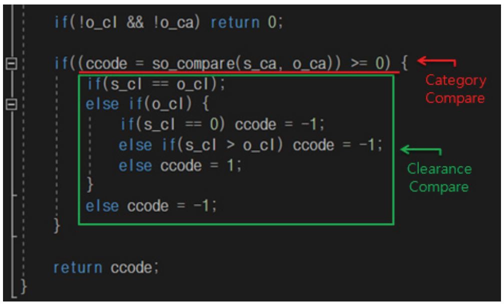


Fig. 14. Security label comparison algorithm.



```
    if(!o_cl && !o_ca) return 0;

    if((ccode = so_compare(s_ca, o_ca)) >= 0) {
        if(s_cl == o_cl);
        else if(o_cl) {
            if(s_cl == 0) ccode = -1;
            else if(s_cl > o_cl) ccode = -1;
            else ccode = 1;
        }
        else ccode = -1;
    }

    return ccode;
}
```

Fig. 15. Security label comparison.

In the comparison algorithm described using the source code depicted in Fig. 15, the categories of the subject and object are first compared, before comparing the clearance. Clearance comparison complies with the BLP model. If the clearance of the subject (s_cl) and object (o_cl) are the same (i.e., s_cl == o_cl), all access is allowed. If the clearance of the subject (s_cl) is lower than that of the object (o_cl) (i.e., s_cl> o_cl), read access is denied by no read-up.If the clearance of the subject (s_cl) is higher than that of the object (o_cl) (i.e., s_cl < o_cl), read access is allowed but write access is denied by no write-down.

The overall flowchart of the proposed access control is shown in Fig. 16. In a user application, the user passes the requested information to Kernel32.dll. The request information is first delivered to the I/O manager. Subsequently, it is formed into an IRP structure and delivered to the file system filter driver, before going to the file system driver. After the security policy of the subject (parent process) and object (file requested for execution) are extracted, the access control result is determined through the policy comparison algorithm. The extracted access control result is delivered to the file system driver. At this point, the access control result and information are output to the log.

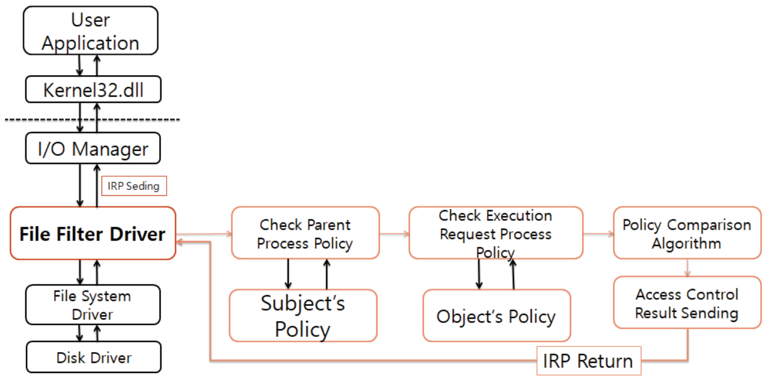


Fig. 16. Access control flowchart.

3.4 Log Management

This function collects the events generated through the security policy and access control, and stores them as logs. In addition, it generates these logs in a file format such that the user can view the logs at the user level.

4. Experimental

4.1 Test Environment

To verify whether the proposed system can prevent the vulnerability we aim to address, we built a test environment and conducted access control tests. The tests were conducted by comparing two systems, namely, the conventional system using HAURI Inc.'s WhiteSec and the proposed system. Test 1 was conducted on a conventional whitelist-based security system, whereas Test 2 was conducted on the proposed MAC model-based security system. The test environments built for Test 1 and Test 2 are specified in Table 1. The Windows embedded OS was commonly installed in both test environments.

Table 1. Test environments

	PC for whitelist-based filter driver test	PC for MAC-based filter driver test
OS	Windows Embedded OS POS Ready7 x64	Windows Embedded OS POS Ready7 x64
CPU	VMware Workstation 14 Core 2	VMware Workstation 14 Core 2
RAM	2 GB	2 GB
Installed security system	WhiteSec 1.3 of HAURI Inc.	MAC-based file system filter driver
Purpose	Whitelist-based test	MAC-based test

4.2 Test Process and Results

Test 1 intends to show the vulnerability of the conventional whitelist DAC-based security solution in the Windows embedded OS, whereas Test 2 intends to demonstrate how the proposed MAC-based security solution tackles the vulnerability exposed in Test 1. In the Test 1 scenario, by capturing the rights of the whitelist, an attacker includes amalware.exe file inserted with unregistered malware into the whitelist and executes the file.

The Test 1 scenario is illustrated in Fig. 17. malware.exe, which can be executed through the command prompt, is deliberately stored in the Windows embedded OS. The attacker of the OS includes malware.exe in the whitelist by capturing the access rights to the whitelist containing cmd.exe and executes malware.exe through the command prompt.

If malware.exe is a real malware sample, the entire system will be infected by the malware. Thus, there is vulnerability in the DAC model, where in the entire security system can be neutralized, if the whitelist is captured.

Test 2 is conducted on the Windows embedded OS, where the proposed MAC model-based whitelist security system is installed. The security label of the cmd.exe file is set to clearance "1" and category "A," as shown in Fig. 18.

The Test 2 scenario is as follows. malware.exe is deliberately stored in the system, and can be executed through cmd.exe. The initial security label of malware.exe is set to clearance "9" and category "C" by the proposed security system.

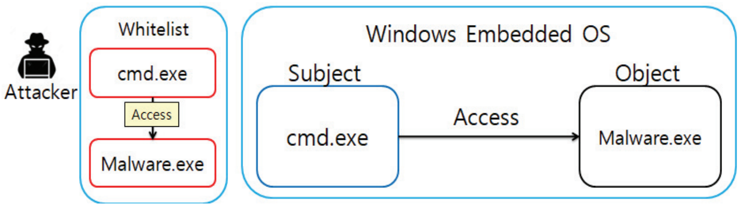
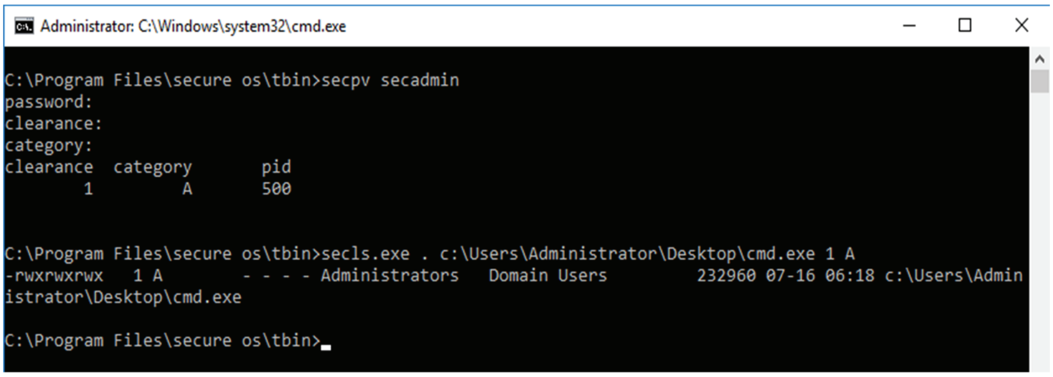


Fig. 17. DAC model-based whitelist security system test.



```
C:\Program Files\secure os\tbin>secpv secadmin
password:
clearance:
category:
clearance  category       pid
        1         A       500


C:\Program Files\secure os\tbin>secls.exe . c:\Users\Administrator\Desktop\cmd.exe 1 A
-rwxrwxrwx   1 A        - - - - Administrators   Domain Users        232960 07-16 06:18 c:\Users\Admin
istrator\Desktop\cmd.exe

C:\Program Files\secure os\tbin>_
```

Fig. 18. Security label setting.

The attacker captures the access rights to the security policy of malware.exe and sets malware.exe to the highest clearance "1," as depicted in Fig. 19.

The attacker executes the malware.exe file through cmd.exe, but access is blocked by the proposed security solution, as shown in Fig. 20.

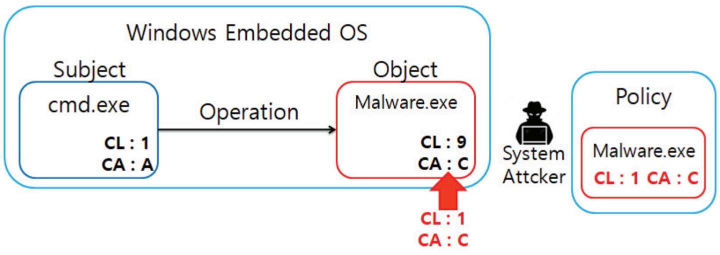
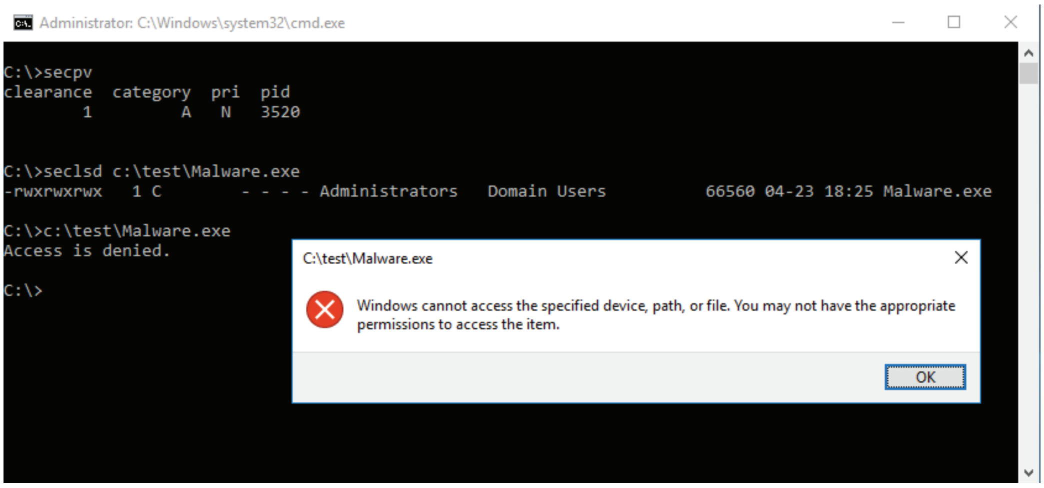


Fig. 19. MAC model-based security system test.



```
C:\>secpv
clearance  category  pri  pid
        1         A    N   3520


C:\>seclsd c:\test\Malware.exe
-rwxrwxrwx   1 C         - - - - Administrators   Domain Users         66560 04-23 18:25 Malware.exe

C:\>c:\test\Malware.exe
Access is denied.

C:\>
```

C:\test\Malware.exe

Windows cannot access the specified device, path, or file. You may not have the appropriate permissions to access the item.

Fig. 20. Access control results.

5. Conclusion

Currently, Windows embedded OS has the largest share of the industrial system market, and domestic and foreign IT security companies are releasing whitelist-based file system filter drivers for system security. However, as the products entering the market are based on the DAC model, they are vulnerable with respect to protecting the policy. Attackers can capture the access rights by taking advantage of this vulnerability, and can wreak havoc on the entire system. Therefore, there is a need to improve the security of the file system filter driver.

When the ownership or policy of the subject is captured, which is the vulnerability of the DAC model, there is a risk that the rights of all the objects owned by the subject may be captured. The duplicate whitelist method, which is one of the methods for preventing this vulnerability, includes a policy that the programs are registered on and a policy used when updating. The former policy is not exposed, but the latter policy is exposed. Thus, it is safer than single-policy management. However, because of the nature of the DAC model, only the subject has access rights to the object. Thus, if the ownership of the subject is captured, there is a high possibility that the entire object can be captured.

In this study, we designed a Windows embedded OS security system based on the MAC model and introduced a method for preventing vulnerability of the access rights set for the subject alone. The proposed system sets access rights for both the object and subject, and restricts access by comparing the policy of the subject and object. Further, the proposed system was designed to store the events generated during access control. In order to verify whether the proposed system could address the vulnerability of the conventional system, we setup a test environment and performed access control tests. The test results demonstrated that because the rights for both the subject and object were determined and the access rights were granted by comparing the security labels, external attackers were prevented from capturing the access rights.

Acknowledgement

This work was supported by Institute for Information & communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) (No. 2019-0-01343, Training Key Talents in Industrial Convergence Security).

References

[1] S. S. Park, "A study on the whitelist-based process control method for security of POS system," M.S. thesis, Department of Information Security, Sungkyunkwan University, Seoul, Korea, 2016.

[2] K. Srinivasan, C. Y. Chang, C. H. Huang, M. H. Chang, A. Sharma, and A. Ankur, "An efficient implementation of mobile raspberry Pi Hadoop clusters for robust and augmented computing performance," *Journal of Information Processing Systems*, vol. 14, no. 4, pp. 989-1009, 2018.

[3] Telecommunications Technology Association, "Security requirements of the POS system (TTAK.KO-12.0181)," 2011 [Online]. Available: http://www.tta.or.kr/data/ttas_view.jsp?totalSu=643&by=desc&order=publish_date&rn=1&pk_num=TTAK.KO-12.0181&nowSu=251.

[4] AhnLab, "POS threat, flaw attack," 2015 [Online]. Available: https://www.ahnlab.com/kr/site/securityinfo/secunews/secuNewsView.do?menu_dist=2&curPage=1&seq=23403.

[5] I. Fratric, "Microsoft Edge: Default Flash click2play whitelist is insecure," 2019 [Online]. Available: https://bugs.chromium.org/p/project-zero/issues/detail?id=1722.

[6] S. G. Hong, "Study on strengthening document security using file system driver," PhD dissertation, Chung-Ang University, Seoul, Korea, 2011.

[7] S. J. Kim, "A study of effective rootkit-detection based on Windows system," Master's thesis, Konkuk University, Seoul, Korea, 2008

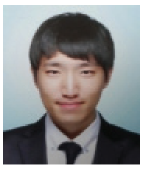
[8] Microsoft," File Systems driver design guide," 2020 [Online]. Available: http://msdn.microsoft.com/ko-kr/windows/hardware/gg462968.

[9] G. S. Mahmood, D. J. Huang, and B. A. Jaleel, "A secure cloud computing system by using encryption and access control model," *Journal of Information Processing Systems*, vol. 15, no. 3, pp. 538-549, 2019.

[10] K. Fan, X. Yao, X. Fan, Y. Wang, and M. Chen, "A new usage control protocol for data protection of cloud environment," *EURASIP Journal on Information Security*, vol. 2016, article no. 7, 2016. https://doi.org/10.1186/s13635-016-0031-6

[11] S. P. Hong, "Design and implementation of mandatory access control based on Linux kernel," Master's thesis, Hanseo University, Seosan, Korea, 2001.

[12] B. S. Choi, “Design and implementation of secure Linux kernel based on RBAC mechanism,” Master’s thesis, Hannam University, Daejeon, Korea, 2004.
[13] J. N. Kim, S. W. Sohn, and C. H. Lee, “Test on the security and performance on the basis of the access control policy implemented by secure OS,” *The KIPS Transactions: Part D*, vol. 10, no. 5, pp. 773-780, 2003.
[14] D. E. Bell and L. J. La Padula, “Secure computer system: Unified exposition and multics interpretation,” MITRE Corp., Bedford, MA, Technical Report No. 2997, 1976.
[15] Y. Jing, J. H. Kim, and D. W. Jeong, “a universal model for policy-based access control-enabled ubiquitous computing,” *Journal of Information Processing Systems*, vol. 2, no. 1, pp. 28-33, 2006.
[16] A. Rafique, D. Van Landuyt, E. Truyen, V. Reniers, and W. Joosen, “SCOPE: self-adaptive and policy-based data management middleware for federated clouds,” *Journal of Internet Services and Applications*, vol. 10, article no. 2, 2019. https://doi.org/10.1186/s13174-018-0101-8
[17] Red Hat Enterprise Linux 7, “SELinux User’s and Administrator’s Guide,” 2021 [Online]. Available: https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/7/html/selinux_users_and_administrators_guide/index.
[18] C. Bodei, P. Degano, F. Nielson, and H. R. Nielson, “Static analysis of processes for no read-up and no write-down,” in *Foundations of Software Science and Computation Structure*. Heidelberg, Germany: Springer, 1999, pp. 120-134.
[19] H. Yoo, J. H. Yun, and T. Shon, “Whitelist-based anomaly detection for industrial control system security,” *The Journal of Korean Institute of Communications and Information Sciences*, vol. 38, no. 8, pp. 641-653, 2013.
[20] Microsoft, “CVE-2019-0641: Microsoft Edge Security Feature Bypass Vulnerability,” 2019 [Online]. Available: https://msrc.microsoft.com/update-guide/en-us/vulnerability/CVE-2019-0641.
[21] S. Parkinson, S. Khan, J. Bray, and D. Shreef, “Creeper: a tool for detecting permission creep in file system access controls,” *Cybersecurity*, vol. 2, article no. 14, 2019. https://doi.org/10.1186/s42400-019-0031-1



Yeon Sang Seong **https://orcid.org/0000-0003-4966-7482**

He is on the Windows kernel security team at HAURI Inc. Currently, he is pursuing a master's degree in computer engineering from Chungnam National University.

Chaeho Cho **https://orcid.org/0000-0003-2285-8553**

He is a Ph.D. student in the Department of Computer Science and Engineering at Chungnam University, Korea. His main research interests are network security using machine learning and digital forensics.

Young Pyo Jun **https://orcid.org/0000-0001-6923-8031**

He received the B.S. degree in Computer Science from Kwangwoon University, Korea in 1985 and received M.S. and Ph.D. degrees in Computer Science from KAIST, Korea, in 1987 and 1994, respectively. Dr. Jun joined the faculty of the Division of Computer and Telecommunications Engineering at Yonsei University, Wonju, Korea, in 2020. He is currently an Associate Professor in the Software Division, Yonsei University. He is interested in artificial intelligence, application software, and system software.

Yoojae Won **https://orcid.org/0000-0002-7706-5983**

He received the B.S. and M.S. degrees from the Department of Computational Statistics at Chungnam National University, Korea, in 1985 and 1987, respectively. He received his Ph.D. from the Department of Computer Science Engineering at Chungnam National University, Korea, in 1998. In 1987-2001, he worked as a principle researcher at the Electronics and Telecommunications Research Institute (ETRI). In 2001-2004, he was at AhnLab Inc. as a CTO. Before joining Chungnam National University, he was at the Korea Internet & Security Agency (KISA) as a Executive Vice President from 2004 to 2014. Currently, he is a professor at the Department of Computer Science Engineering, Chungnam National University, Daejeon, Korea, and the director of Convergence Security Research Center.