electronics

MDPI

Article

Kernel-Based Real-Time File Access Monitoring Structure for Detecting Malware Activity

Sung-Hwa Han [1] and Daesung Lee [2,*]

[1] Department of Information Security, Tongmyong University, Busan 48520, Korea; shhan@tu.ac.kr
[2] Department of Computer Engineering, Catholic University of Pusan, Busan 46252, Korea
* Correspondence: dslee@cup.ac.kr

Abstract: Obfuscation and cryptography technologies are applied to malware to make the detection of malware through intrusion prevention systems (IPSs), intrusion detection systems (IDSs), and antiviruses difficult. To address this problem, the security requirements for post-detection and proper response are presented, with emphasis on the real-time file access monitoring function. However, current operating systems provide only file access control techniques, such as SELinux(version 2.6, Red Hat, Raleigh, NC, USA) and AppArmor(version 2.5, Immunix, Portland, OR, USA), to protect system files and do not provide real-time file access monitoring. Thus, the service manager or data owner cannot determine real-time unauthorized modification and leakage of important files by malware. In this paper, a structure to monitor user access to important files in real time is proposed. The proposed structure has five components, with a kernel module interrelated to the application process. With this structural feature, real-time monitoring is possible for all file accesses, and malicious attackers cannot bypass this file access monitoring function. By verifying the positive and negative functions of the proposed structure, it was validated that the structure accurately provides real-time file access monitoring function, the monitoring function resource is sufficiently low, and the file access monitoring performance is high, further confirming the effectiveness of the proposed structure.

Keywords: real-time monitoring; hidden malware; file access monitoring; kernel-based structure; access control; endpoint detection and response; zero trust

Citation: Han, S.-H.; Lee, D. Kernel-Based Real-Time File Access Monitoring Structure for Detecting Malware Activity.. *Electronics* **2022**, *11*, 1871. https://doi.org/10.3390/electronics11121871

Academic Editor: Antonio Pescapè

Received: 12 April 2022
Accepted: 10 June 2022
Published: 14 June 2022

1. Introduction

Security threats emerge as IT technology evolves. A malicious attacker can directly access a service, leak important information, or slow down a service. However, this method may expose the identity of the attacker. Therefore, spear phishing malware that can hide the identity of the attacker is increasing used [1,2].

Malware can access an attack target (e.g., file, process, registry, or device) and modify or leak important information. In an enterprise environment, IPSs or IDSs or antivirus is applied to detect malware and prevent its execution [3]. A signature update service-type security technique can detect and prevent malware in networks or system layers by using a malware feature signature [4], which is periodically updated by the vendors who supply this security technique to detect the latest malware and prevent its execution [5].

However, malware has evolved. In a recent malicious attack, a security technique was applied to malware that is only used in information services [6]. A malicious attacker applies obfuscation or cryptography to malware to avoid detection by an IPS/IDS or antiviruses that are used in an enterprise environment [7,8]. This malware has complex logic, which is difficult for malware analysts to decipher. Script-based malware exists as text or saved files in a normal status and is executed by applications, such as web servers or web browsers. Because these applications are trusted, they are excluded from malware detection targets. Therefore, it is difficult to detect script-based malware [9]. In

addition, because malicious attackers apply detection avoidance tests, malware detection is becoming increasingly difficult [10]. In recent enterprise environments, white-list-based process management, end-point detection response (EDR) security management with AI-based IPS/IDS, and antivirus detection functions have been implemented [11].

The current operating systems (OS) provide event-monitoring functions, such as application execution, user login, and network status. An access control function is also provided to protect some of the important files of the system; however, a function for monitoring user file access is not provided. To provide safe service, it is necessary to identify all events for various information resources [12]. A file access monitoring function is also required to improve the accuracy of the EDR technique used in recent enterprise environments.

To overcome this limitation, we propose a real-time file access monitoring structure that can monitor user access to a file or directory in real time. The proposed structure monitors according to the policy set by the security manager, and notifies an event, only when it is confirmed that the file access corresponds to the absolute file path set by the policy. In addition, because the proposed structure operates at the kernel layer, malicious attackers cannot bypass the file access monitoring function enforced in the system.

Because the proposed real-time file access monitoring structure should be effective, its function and performance should be verified. Therefore, in this study, the positive and negative functions and performance of the proposed structure were verified.

Main contributions of the proposed file access monitoring structure are as follows:

- It is possible to monitor all user access events for important files in real time.
- The normal operation of other user applications is guaranteed by minimizing the resources used for file/directory access monitoring.
- As event data are generated using the file access monitoring function, EDR accuracy can be increased.
- If the structure proposed in this study is loaded into a security service or OS, the potential threat to information services can be prevented.

The remainder of this paper is organized as follows. In Section 1, the background, purpose, and contribution of this study are described. In Section 2, the security event monitoring technique provided by the current OS and the file access control technique protecting important files are analyzed. The technique applied to current malware is further verified, and an AI-based security technique is introduced to detect such malware. In Section 3, the security environment of the information service is analyzed and requirements are derived for the protection of the information service. In Section 4, a structure is proposed for a real-time file access monitoring function to satisfy the security requirements. In Section 5, the function and performance are verified to confirm the effectiveness of the proposed real-time file access monitoring structure. Finally, the limitations of this study are presented in Section 6.

2. Related Works

2.1. Security Event Monitoring Techniques

The application service manager should be able to check the status of as application and ensure that its performance reaches the target level [13] using the monitoring function provided by the OS, as presented in Table 1 [14–18].

These monitoring techniques do not determine whether the status of a system or user application is normal or abnormal. They list the current status of a system or application process. The system or service manager uses these monitoring functions to determine the status of the system and its applications, and if an abnormal status is detected, they can respond accordingly depending on the system or application status [19].

Table 1. Monitoring functions provided by OS.

Layer	Monitoring Function	Description
System	Process monitoring	• Lists the application processes running in the OS. • Identifies PID or PPIID, ownership, and process dependencies.
	Resource monitoring	• Checks the CPU and memory usage rates used by each application process. • Identifies storage capacity, usage, and remaining capacity for each storage device.
Network	Packet monitoring	• Inspects inbound/outbound packets. • Saves monitoring packet to file.
	TTY Session monitoring	• Monitors the TTY session of users connected to Telnet, rlogin, SSH protocol of Linux/Unix OS, or remote desktop protocol (RDP) of Windows OS.

2.2. Current Malware Trends

2.2.1. Security Technique Application

Malware is widely used by malicious attackers in information attacks. It enters the enterprise environment in various ways [20]; thus, an attacker uses malware to obtain in-formation or modifies it into invalid information to harm the owner [21,22]. Most enterprise environments apply IPS/IDS to the network or antivirus to the OS to detect and stop malware [23–25]. A malware analyst generates signatures by analyzing features such as logic, processing flow, keywords, and memory handling. The generated signature is deployed to an IPS/IDS or an antivirus, where it is compared to the inspection target packet, file, process, and memory with the signature to determine whether it is malware [26]. When these techniques are applied to an enterprise environment, enterprise damage can be reduced by preventing the execution of malware.

However, malware has evolved. Recently, malicious attackers have applied techniques such as cryptography, obfuscation, and memory injection to malware [27–30]. Figure 1 shows the types of obfuscations that can be applied to malware. Obfuscation is a security technique that protects the original software logic [31]. However, malicious attackers can apply this security technique to avoid malware detection [32] by changing the malware logic, processing flow, keywords, and memory handling procedures [33]. This modified malware cannot be detected because its features are different from the signature deployed in an IPS/IDS or antivirus. To detect malware applied with obfuscation, a new signature must be created. To achieve this, analysts must analyze obfuscated malware, which is a quite difficult process. In addition to applying the obfuscation technique, a malicious attacker can avoid the malware detection function by applying other techniques, such as garbage code injection or renaming the register to the malware [34].

A behavior-based IPS/IDS or antivirus can detect obfuscated malware by applying a process-tracing technique [35]. However, if a malicious attacker applies a packing or protector technique to malware, it stops the antivirus; thus, a malicious attacker can bypass the behavior-based detection technique. The more techniques are applied to malware, the more difficult it is for malware analysts to define signatures that can detect malware, making malware detection increasingly difficult [36].

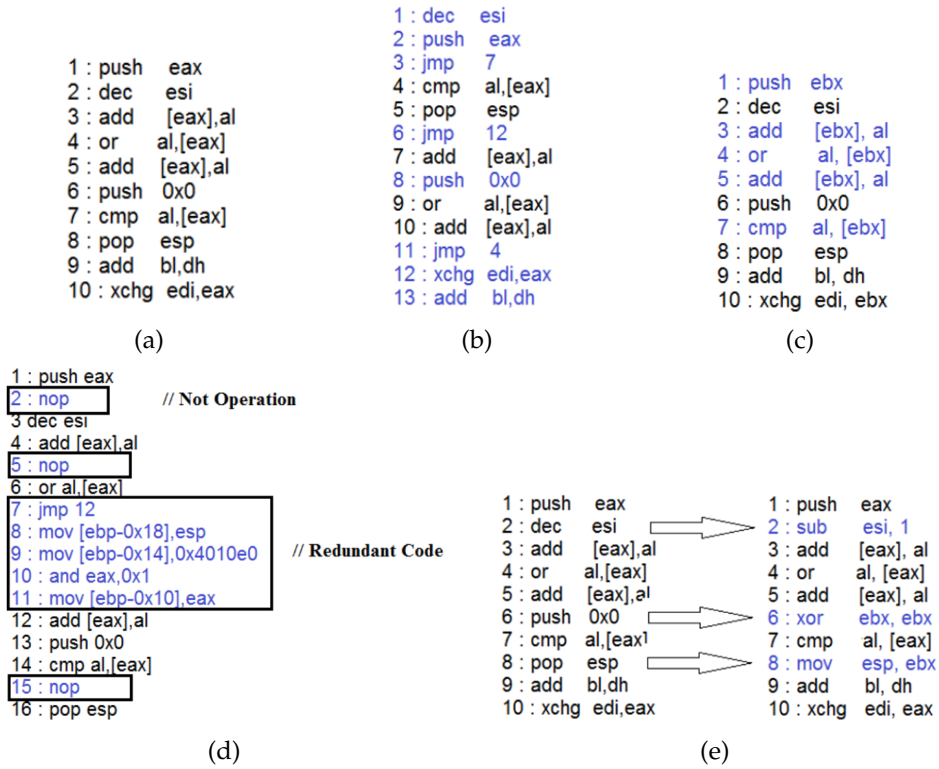

Figure 1. Malware obfuscation techniques: (**a**) original malware assembly code; (**b**) after applying jumping obfuscation; (**c**) after applying renaming obfuscation; (**d**) after applying redundant code obfuscation; and (**e**) after applying replacement obfuscation.

2.2.2. Conducting the Malware Detection Avoid Test

Malicious attackers may want malware to serve their purpose. However, IPS/IDS or antiviruses are not only applied to the target environment. Malware detection techniques have been applied to several devices and network switches. After a malicious attacker distributes malware through the Internet, it can be removed by an IPS or antivirus in the process of being delivered to the target system. To prevent this situation, a malware attacker performs a detection-avoidance test [37].

The detection avoidance test is a process in which malicious attackers check whether malware can avoid the detection functions of IPS/IDS and antiviruses [38]. A malicious attacker selects a malware detection technique that is available on the Internet and checks whether malware is detected [39]. If a malware detection technique detects the malware, a malicious attacker upgrades the malware until it is not detected. By applying this method to malware development, malicious attackers can safely distribute malware to a target system [40].

2.3. EDR

The number of techniques applied to malware as well as the amount of malware are increasing rapidly. In particular, with the kill-chain-based advanced persistence threat (APT) attack rising in popularity, the limitations of the security techniques applied to the enterprise are being identified [41].

A kill-chain-based APT attack comprises seven steps. The attacker collects basic information about the target (reconnaissance), creates a tool to collect detailed information (weapons), and penetrates the target (delivery). After generating malware to attack a target identified as a weapon (exploitation), it is installed and attacks according to the attacker's command (command and control) to achieve the final goal (act on the objective) [42]. The detailed attack preparation and execution processes have increased the quality of the

malware used in APT attacks. Therefore, it is difficult to detect or prevent malware used in APT attacks using legacy security techniques such as IPS/IDS or antiviruses [43].

To address these problems, Gartner proposed EDR in 2013 [44]. EDR is a new paradigm that identifies and responds to abnormal behavior based on the stable state of a system. It responds to threats after recognizing security threats to PC, servers, applications, processes, and files/directories [45]. Thus, an EDR threat is not defined in advance. EDR considers any unauthorized access to an endpoint as a threat. Therefore, it first defines the stable status of the system, and a threat is detected with respect to this stable status [46]. To define the stable status, an EDR-based security technique learns various operational data generated in an enterprise environment [47]. Because considerable data have to be learned for determining the stable status, the EDR-based security technique applies a machine-learning technique. The higher the volume, variety, and velocity of the learning data, the higher is the accuracy [48]. System, application, and network logs can be used as operational data. Therefore, when different types of data are available and sufficient learning data are guaranteed, the stable state can be defined more accurately using the machine-learning techniques [49].

3. Analyzing Current Security Environments

3.1. Analyzing Security Environment

The security event monitoring technologies listed in Table 1 are used for the system, information services, and security monitoring [50]. However, there is a limit to the detection of security threats that combine access layers. Figure 2 presents examples of typical complex security threats.

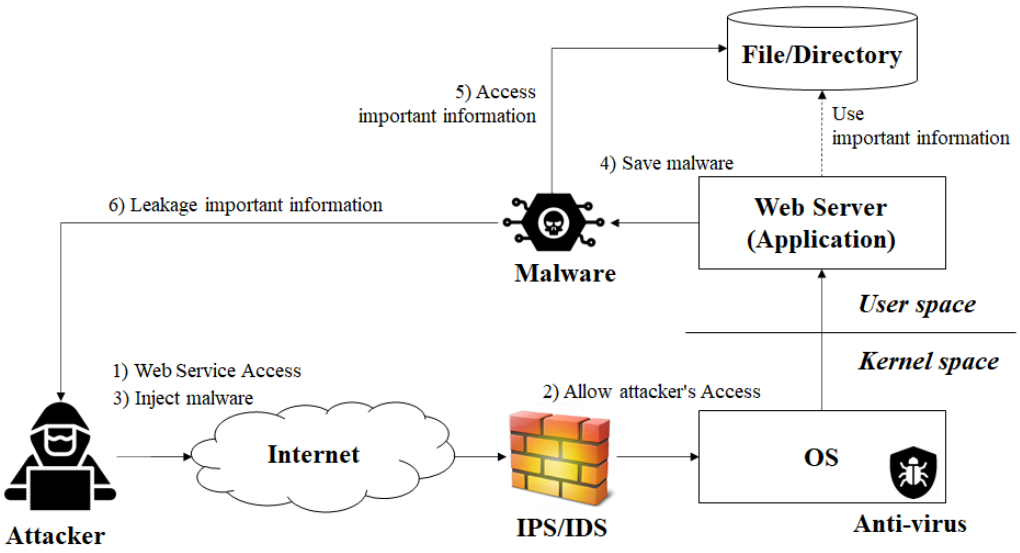

Figure 2. Multi-layered security threat.

A malicious attacker can upload script-based malware to a web server using a known channel (e.g., HTTP or HTTPS). When malware uploaded to the web server executes, it can access important files and leak or modify important information. An IPS/IDS cannot detect zero-day attacks or malware to which obfuscation techniques have been applied at the network layer. The script malware is executed on a web server. Because the web server is a trusted subject, the antivirus does not detect abnormal behavior on the web server. The resources used to access or modify important files of this malware are also at a very low level; consequently, they cannot be identified as an abnormal state of the system. Thus, legacy security techniques cannot detect or prevent combined access layer malware attacks.

3.2. OS File Protection Techniques

The OS, which provides a safe operating environment for information services, consists of various files and directories. It maintains a safe state and provides an execution environment for user applications. It must also be able to allocate the resources required by the user application, and the resources allocated to the application must be protected until the application returns [51]. If the OS file/directory is deleted or modified, the functions that the OS should provide cannot be provided. Therefore, to provide a secure environment, the important OS files/directories must be protected [52]. AppArmor(version 2.5, Immunix, Portland, OR, USA) and SELinux(version 2.6, Red Hat, Raleigh, NC, USA) are provided by the Linux OS, and the group policy object is provided by the Windows OS. AppArmor(version 2.5, Immunix, Portland, OR, USA) and SELinux(version 2.6, Red Hat, Raleigh, NC, USA) operate as a structure in which the application process and kernel module interact, as shown in Figure 3 [53,54].

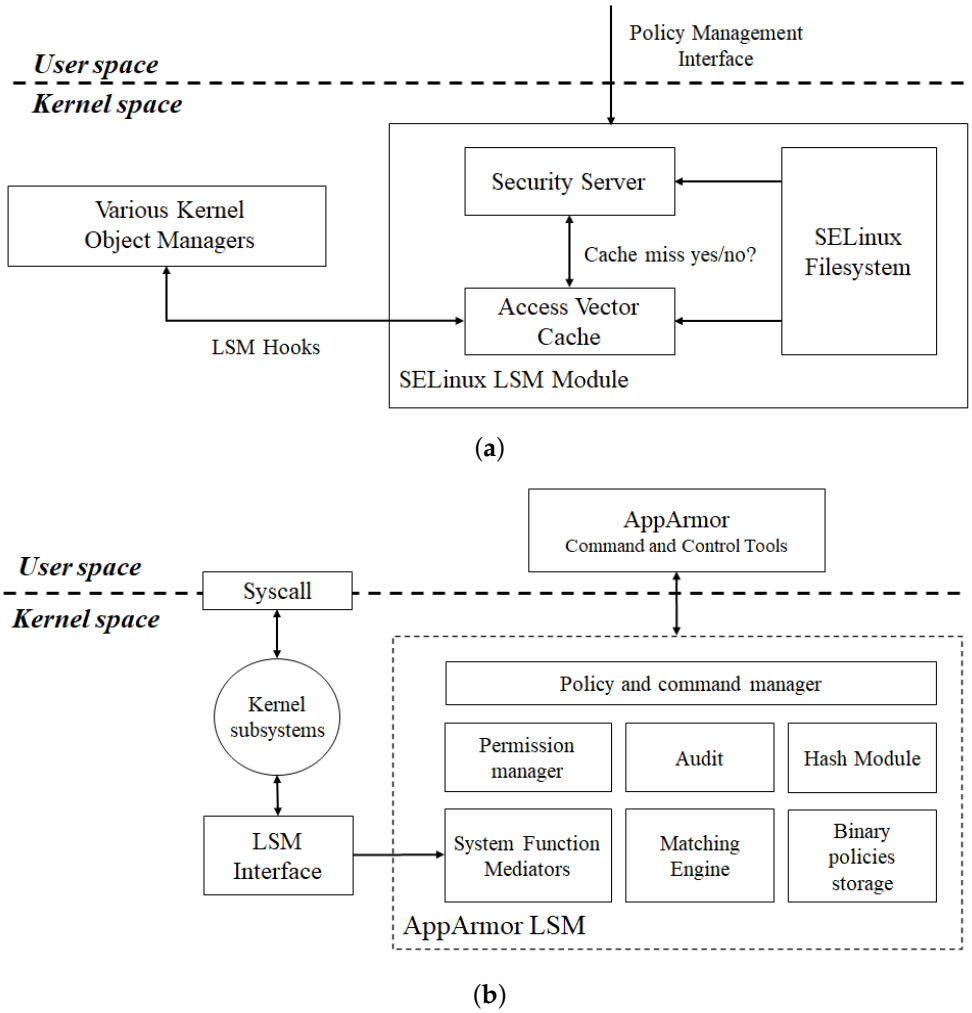

Figure 3. File access control structures: (**a**) SELinux registers the access control policy at the *security server* in the kernel and enforces it with the access vector cache. (**b**) AppArmor registers the access control policy using *command and control tools* and enforces it with *system function mediators*.

To protect the file access control policy, the security officer transfers objects representing files/directories to the SELinux(version 2.6, Red Hat, Raleigh, NC, USA) policy management interface or AppArmor's (version 2.5, Immunix, Portland, OR, USA) command and control tools, which interpret the received object and list the file/directory for protection, creating an access control policy for the file/directory list prior to transferring it to the kernel. The kernel module builds a policy table after receiving the access control

policy from the application layer. Subsequently, the kernel module checks the file access of the user to determine if it matches the policy registered in the policy table. If the path of the file accessed by the user is not registered in the policy table, SELinux(version 2.6, Red Hat, Raleigh, NC, USA) or AppArmor(version 2.5, Immunix, Portland, OR, USA) allows access to all users. If the file accessed by the user is the same as that registered in the policy, it denies user file access [55,56]. Because both AppArmor(version 2.5, Immunix, Portland, OR, USA) and SELinux(version 2.6, Red Hat, Raleigh, NC, USA) enforce kernel-based access control functions, users cannot bypass these file access functions [57,58].

3.3. Requirement for Application Service Security

The OS protects the system file/directory by applying SELinux(version 2.6, Red Hat, Raleigh, NC, USA) or AppArmor(version 2.5, Immunix, Portland, OR, USA). It also prevents unauthorized access to the file/directory used to provide system service. Thus, the OS implements an environment that can perform its original role. However, both SELinux(version 2.6, Red Hat, Raleigh, NC, USA) and AppArmor(version 2.5, Immunix, Portland, OR, USA) focus only on system file/directory protection and do not protect user applications.

Malware attacks enterprise systems and mainly user applications or user data. Therefore, not only personal devices but also malware should be precluded from entering the enterprise environment. IPS/IDS and antivirus legacy security solutions are effective for detecting, deleting, and preventing normal malware. However, it is difficult to detect malware to which obfuscation has been applied. In particular, in the current Internet environment where malware is rapidly increasing, detection methods using signature deployment are insufficient. Security managers must prevent malware from entering the enterprise environment. However, if malware is deactivated, the security manager finds it difficult to detect.

To solve this problem, a file access monitoring function is required. The requirements for file access monitoring are emphasized in both the perimeter-based security model and zero-trust model to protect user applications and data [59,60]. If a file access monitoring function is provided, the security manager can check access to important application files or user data. The security manager allows access to authenticated users and denies access to unauthorized users through this function. If the unauthorized access is for an unidentified application, the possibility of malware is high. Therefore, using this function, the security manager can check for the existence of malware.

To satisfy a variety of EDR-based security techniques, status data are required. The file access event data generated by the application are stable status data. Therefore, file access event data do not include unstable status data. If unstable status data are provided, the accuracy of the machine learning increases. Therefore, a file access monitoring function is required to create stable and unstable status data.

4. Kernel-Based Real-Time File Access Monitoring

4.1. File Access Monitoring Structure

In this paper, we propose a structure that can monitor the access to files required in many security environments. In the proposed file access monitoring structure, shown in Figure 4, the module operating at the kernel layer and the process operating at the application layer collaborate to monitor file access to the user application under the assumption that important files are identifiable. The proposed structure compares the absolute path of the monitoring file with that of the user file access event.

The proposed structure has five components. The *policy register* is an application interface executed by a security manager. The security manager executes the *policy register* and delivers an absolute file path to the kernel *policy manager*.

The *policy manager* receives the absolute file path that passes through the *policy register*. The *policy manager* is a system-call handler. The *policy database* stores absolute file paths. Because the *policy database* is composed of a linked list, multiple absolute file paths can

be registered. The *file access event parser* inspects all file access events for the user. The *file access event parser* compares the access target file path of the file access event with the absolute file path registered in the *policy database*. If the access target file path of the file access event matches the absolute file path registered in the *policy database*, the file access event information is registered as a *file access event node*. The *event monitor* is a daemon application. The *event monitor* uses a system call to check whether there is a *file access event node*, and if so, it notifies the security manager. The key function of the proposed structure is to monitor unauthorized file/directory access. The structure differs from that of SELinux or AppArmor, as presented in Table 2.

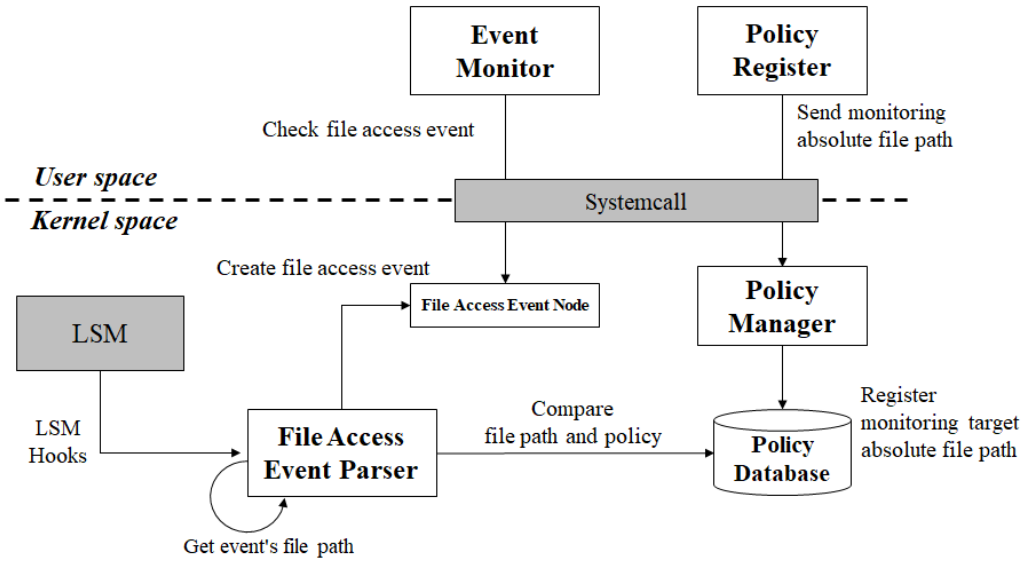

Figure 4. Real-time file access monitoring structure.

Table 2. Difference between the proposed structure and legacy file access control security techniques (SELinux/AppArmor).

Difference	SELinux/AppArmor	Proposed Structure
Key Function	• Provide access control function	• All file/directory access monitoring
Protect mechanism	• Pre-protect method • Deny all unauthorized access • Pre-defined accesses are allowed	• Post-protect method • All file accesses are allowed • All file accesses are the target of security analysis
Policy managing method	• Object-based policy distribution • A security officer should know all policy object	• User generates monitoring policy • Users only need to know the absolute file path for monitoring
Security managing subject	• Security officer or system administrator can manage security policy	• All users • Data owner or application service manager, system manager manage security

4.2. File Access Monitoring Sequence

The existing OS can be accessed by multiple users and has a multi-processing function. Therefore, multiple file access events can occur through multiple processing steps. The security manager monitors all file access events, whereas the file access monitoring structure proposed in this study operates in the order shown in Figure 5.

After determining the important file, the security manager obtains the absolute file path of the file and registers the policy in the *policy register*. The *policy register* creates a policy structure that includes the absolute file path in the kernel *policy manager* and delivers it using a system call. The *policy manager* receives this information and stores it in a *policy database*.

When a user accesses a file, first the OS generates a user file access event node and then a task node that includes the user file access event node, along with a handler to process it and register it in an interrupt requeue (IRQ) [61]. The task nodes registered in the IRQ are processed sequentially using the OS process module. Linux provides a Linux security module (LSM), an interface that enforces user-defined functions for task node processing [62]. If a user-defined handler is registered in the LSM, the user function can be enforced before task node processing. In this study, a file access event-gathering handler was registered in the LSM to gather file access event information [63].

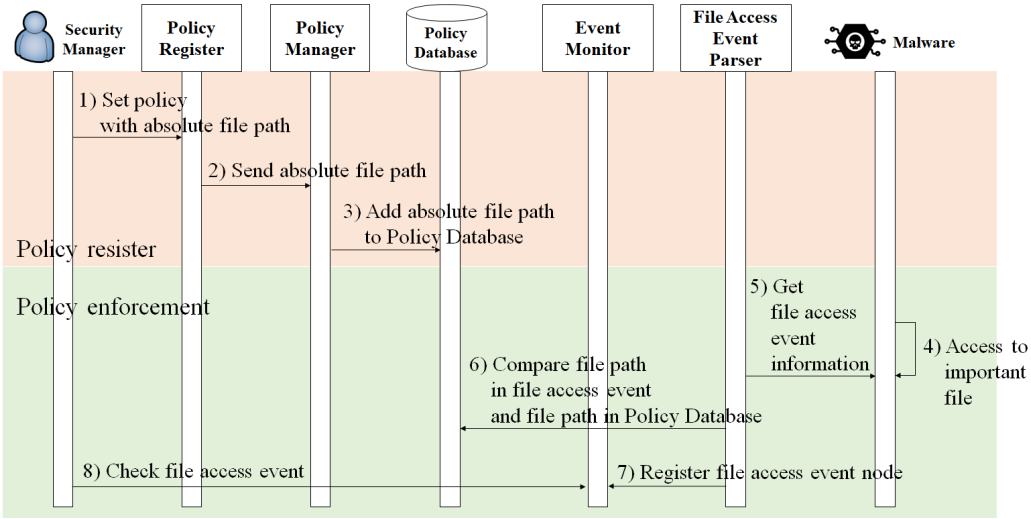

Figure 5. Real-time file access monitoring function sequence.

When a user-defined handler is registered in the LSM, the user file access event is delivered to the *file access event parser*. The *file access event parser* analyzes the file access event and identifies the subject session ID, process ID, and the absolute file path accessing the file. The *file access event parser* then compares the absolute file path identified in the user file access event with the entire absolute file path registered in the *policy database*. The *policy database* comprises a linked list that can be searched sequentially. If an absolute file path matches a path in the policy database, the *file access event parser* creates a *file access event node.*

The *event monitor* is a daemon that continuously checks for a *file access event node*. If there is a *file access event node*, the *event monitor* transfers it to the application layer using a system call, and outputs it on the screen.

4.3. Mechanism of the File Access Monitoring Function

The proposed real-time file access monitoring function generates a *file access event node* only for user file access to the absolute file path. Therefore, the monitoring policy uses the absolute file path (*FP*) as a parameter.

The real-time access monitoring mechanism includes the following definitions:

$$U = \{x | x\ is\ all\ monitoring\ function\ \}$$
$$P = \{x \in U | Access\ monitoring\ policy\ enforced\ to\ information\ system\}$$
$$AR = \{x | x\ is\ set\ of\ all\ file\ access\ request\}$$

In the aforementioned state, the absolute file path (*FP*) from the user file access request r is expressed as follows:

$$r_{FP}, \text{ where } r \in AR \tag{1}$$

The decision result (*file access event node* generation, *ENG*) of the *file access event parser*s decision function (generating decision, *GD*) that compares r_{FP} with the absolute file path in the monitoring policy P_{FP} is defined as follows:

$$ENG = GD(r_{FP}, P_{FP}), \text{ for } r \in AR \tag{2}$$

When the absolute file path in the user file access event r_{FP} matches the absolute file path in the monitoring policy P_{FP} (ENG is true, ENG_T), a *file access event node* is generated. Therefore, in such a case, there is a file access event notification occurrence (FEN_O), expressed as follows:

$$FEN_O \supset ENG_T \tag{3}$$

When the monitoring policy is enforced, the *file access event node* that does not generate (FEN_I) is the inverse function of Equation (3).

$$FEN_I = \overline{ENG_T} = \overline{GD(r_{FP}, P_{FP})}, \text{ but } r \in AR \tag{4}$$

Equation (1) denotes the absolute file path extraction from a file access event. Equation (2) is a monitoring function that enforces the process. When the absolute file path is registered in the monitoring policy, and the user file access event path matches, the *file access event parser* generates a *file access event node*. In the process denoted by Equation (3), if there is a *file access event node*, the *event monitor* checks it and notifies the security manager. Equation (4) indicates that, when the absolute file path registered in the monitoring policy and the path of the user file access event do not match, the *file access event parser* ignores the user file access event. These are the file access-monitoring mechanisms proposed in this study.

5. Implementation

5.1. Function Verification

5.1.1. Function Verification Items

The proposed real-time file access monitoring method is effective. Because the scope of functional verification should encompass the entire scope of this study, functional verification items were derived based on the file access monitoring function mechanism, as presented in Table 3.

Func_VF_1, Func_VF_2, and Func_VF_3 are positive verifications of the proposed real-time file access monitoring function, and Func_VF_4 is negative verification. If the four functions are confirmed as correct, it can be verified that all functions of the structure proposed in this study are correct.

5.1.2. Function Verification Result

The hardware environment used to verify the proposed structure comprises a i3-4150 CPU, 8 GB memory, and 500 GB HDD. The proposed real-time file access monitoring structure was implemented in the CentOS 7.9 OS environment, and the verification items defined in Table 3 were executed. To verify the functional operation of the real-time file access monitoring function, the start and end of the debug messages were checked.

When a user file access event occurs, func_VF_1 checks whether the kernel *file access event parser* has obtained the absolute file path from the file access event.

Therefore, as shown in Figure 6, not only the access target absolute file path but also the user session-id, user id, and subject information of the user process were correctly obtained.

Table 3. Function verification items.

Function ID	Verification object

Func_VF_1	• Verification Equation (1)
	• When a file access event occurs, verify that the *file access event parser* correctly obtains the absolute file path from the file access event.
	• Code sequence:
	`VOID register_file_access_event_hook_func(){`
	`....`
	`GET_FILE_ACCESS_EVENT_INFO get_file_access_event_info(void);`
	`...`
	`}`
	`void get_file_access_event_info(void){`
	`....`
	`while(1){`
	`irq_file_access_event(IRQ *irq, DEV *device, SESSION *user_session);`
	`...`
	`}`
	`}`

Func_VF_2	• Verification Equation (2)
	• Check *file access event node*'s monitoring policy (if there is a monitoring policy and whether absolute file path in file access event and monitoring policy are the same).
	• Code sequence:
	`void scan_policy_database(string *absolute_file_path, DATABASE *policy_db){`
	`struct *police_node = policy_db->policy_node_head;`
	`...`
	`while(policy_node) {`
	`if(strcmp(policy_node->file_path, absolute_file_path)==0)`
	`register_file_access_event_node(absolute_file_path, policy_node->policy_id);`
	`break;`
	`} else`
	`policy_node = policy_node-> node_next;`
	`}`
	`}`

Func_VF_3	• Verification Equation (3)
	• Check file access event notification when there is *file access event node*.
	• Code sequence:
	`int main(void) {`
	`struct *file_access_event_node;`
	`...`
	`while (1) {`
	`file_access_event_node = check_file_access_event_node();`
	`if(file_access_event_node)`
	`printf("File access event occur : %d : %s\n",`
	`file_access_event_node->policy_id, file_access_event_node->file_path);`
	`}`
	`return 0;`
	`}`

Func_VF_4	• Verification Equation (4) • Check if file access event's absolute file path is different from that in monitoring policy to determine if the event is ignored correctly. • Code sequence: Same that of Func_VF_2.

- **Func_VF_1**

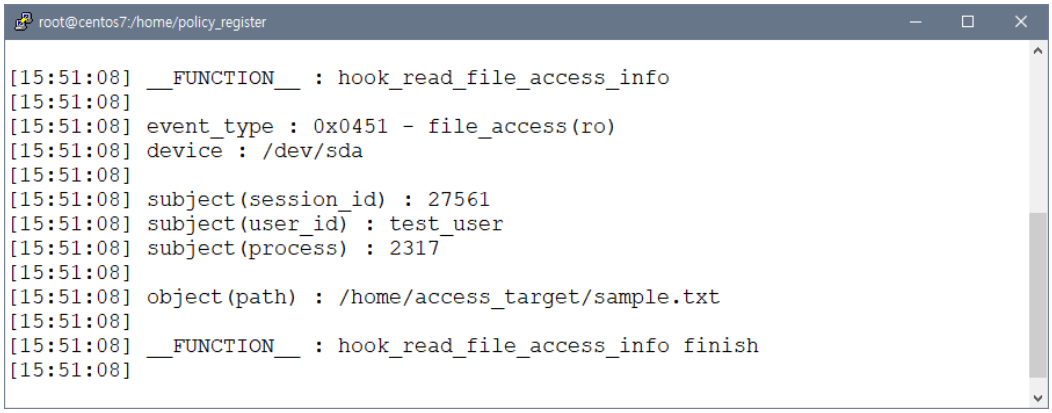

Figure 6. File access event check verification.

- **Func_VF_2**

After Func_VF_1, the monitoring policy of the *file access event parser* is checked. If there is a matching monitoring policy, the *file access event parser* stops scanning the monitoring policy.

As a result of the verification shown in Figure 7, the file access event parser begins by comparing the absolute file path in the file access event with that of the monitoring policy. When a matching monitoring policy is found, the file access event parser stops scanning.

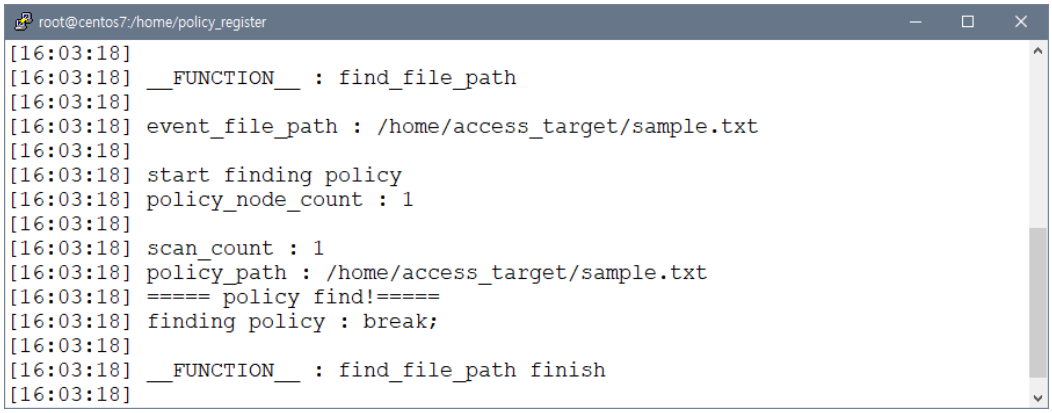

Figure 7. Monitoring policy scanning verification.

- **Func_VF_3**

Once the *file access event node* is generated, the *event monitor* confirms this and checks whether the file access event has been notified. If there are no file access events, the event monitor is not notified.

Consequently, as shown in Figure 8, when the *file access event parser* generates a file access event node, the *event monitor* checks and notifies the file access event.

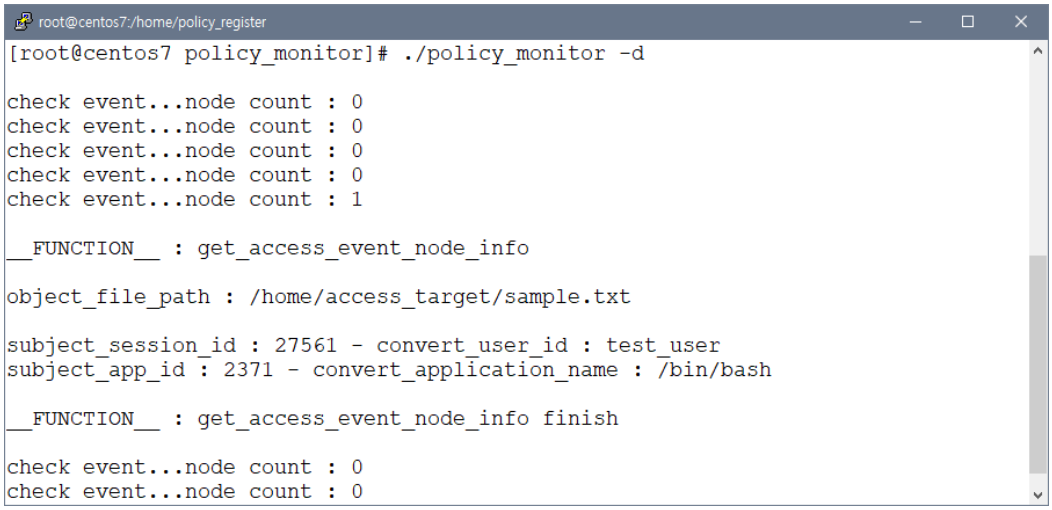

Figure 8. Event monitor's notification function verification. The event monitor checks the file access event node in the kernel periodically. When the event monitor confirms that the file access event node is generated, it notifies the file access event.

- **Func_VF_4**

In Func_VF_1, if user file access does not match the monitoring policy, a further check is conducted to determine whether *file access event parser* ignores this event.

As shown in Figure 9, the *file access event parser* compares the file access event with all the monitoring policies. The *file access event parser* compares the user access target file with the next monitoring policy to determine whether it differs from the absolute file path registered in the monitoring policy. Thus, the *file access event parser* does not generate a file access event node when there is no matching monitoring policy, even after comparing it to the previous monitoring policy.

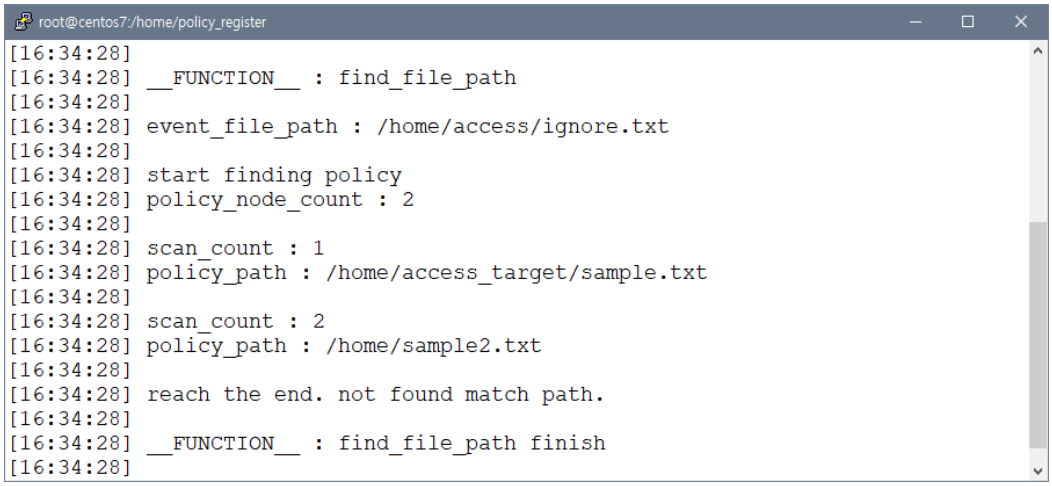

Figure 9. *File access event parser* ignoring function verification. When user file access does not match any monitoring policies, the *file access event parser* ignores the user file access event.

5.2. Performance Verification

5.2.1. Performance Verification Items and Methodology

Even if the proposed real-time security file access-monitoring function performs correctly, the resources used to provide the monitoring function should be small and fast. Only then does the monitoring function affect the operation of other processes. In addition, when multiple file accesses occur in the OS, the monitoring function can identify and notify all file access events. To verify the performance, verification items were selected, as presented in Table 4.

Table 4. Performance verification items.

Performance ID	Verification Object
Perf_VF_1	• Check resource requirements to notify file access events • Check the system resource (CPU occupancy) required for file access notification when 100, 250, 500, 1000, 2500, or 5000 file access events are generated after registering the monitoring policy • The average value was measured for 10 file access events
Perf_VF_2	• With multiple monitoring policies (100, 250, 500, 1000, 2500, 5000) registered, check the time to enforce the last policy • The time gap from the first policy check scan time to the last policy scan is used as the average value of 10 measurement values
Perf_VF_3	• When file access events occur at approximately 200 event/min with random time gaps, the monitoring function can catch all file access events, checking three times during 24 h. • If all file access events are monitored, the proposed monitoring function is strong enough to resist the stress of the actual service environment

The performance verification environment is the same as the function verification environment.

A comparison should be conducted to evaluate the performance of the proposed real-time security file access monitoring structure, wherein the application process is interrelated to the kernel module. The structure is the most similar to that proposed in this study, and the most widely used security technique is SELinux(version 2.6, Red Hat, Raleigh, USA). Therefore, the same verification was performed using SELinux, and the results were compared.

5.2.2. Performance Verification Results

- **Perf_VF_1**

Table 5 lists the resources used to notify the file access event in real time using the proposed security file access monitoring structure when generating the event in SELinux.

Table 5. CPU usage file access event notification using the *policy enforcement server* in SELinux.

Component	CPU Usage (%) File Access Event Count					
File Access Event Monitor	Policy Count					
	100	250	500	1000	2500	5000
	CPU usage(%)					
	0.24	0.31	0.46	0.85	1.43	1.94
Policy Enforcement Server (SELinux)	Policy Count					
	100	250	500	1000	2500	5000
	CPU usage(%)					
	0.29	0.42	0.71	1.14	1.88	4.13

It was confirmed that the CPU usage in notifying the proposed file access event was less than that of the SELinux event occurred.

- **Perf_VF_2**

To check whether the structure proposed in this study could provide rapid file access event notification, with multiple dummy policies registered, the time gap between the first and last monitoring policy scans was measured.

As shown in Figure 10, the policy scan time of the proposed real-time security file access monitoring structure was verified to be faster than that of SELinux under the same conditions.

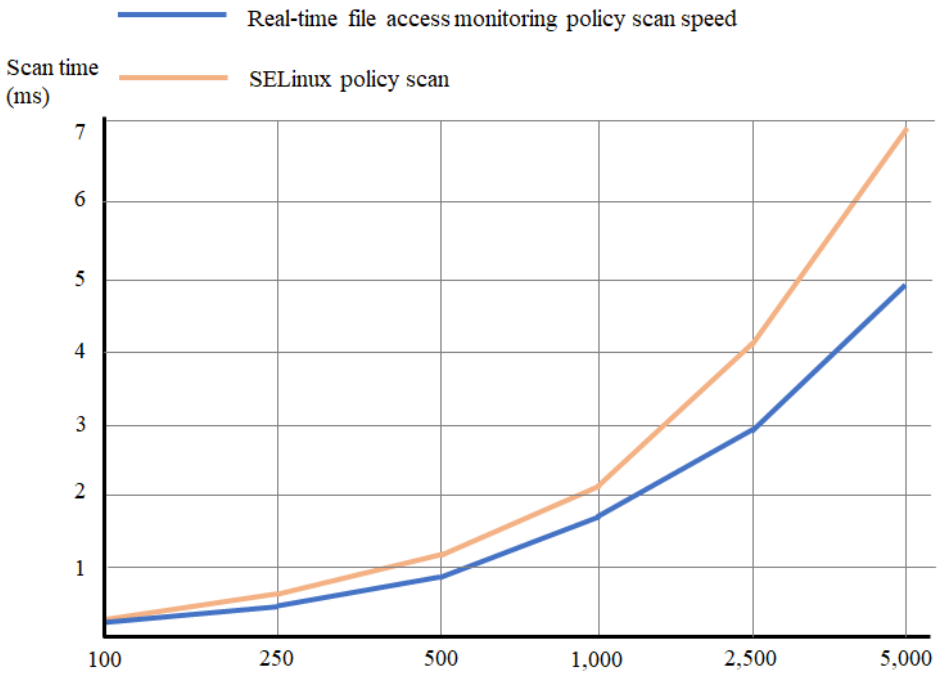

Figure 10. File access event detection time measurement.

- **Perf_VF_3**

To confirm the strength of the proposed file access monitoring function, a script was created for a real IT service status. The file access event script generated approximately 200 file access events per minute. When a file access event occurred, a random time gap was applied. If the proposed file access monitoring function can monitor all file access events generated by the script, then the proposed real-time file access monitoring structure is deemed sufficiently strong. The verification result is presented in Table 6.

Table 6. File access event occurrence count and monitoring count.

Time	File Access Event Occurrence Count by Script	File Access Monitoring Count by the Proposed Structure
1	310,308	310,308
2	307,293	307,293
3	302,187	302,187

Table 6 shows that the file access event count generated by the script is the same as that obtained by file access monitoring. Therefore, the proposed real-time file access monitoring structure is strong enough.

5.3. Performance Verification Analysis

To verify the function of the proposed real-time security file access monitoring structure, both positive and negative functions were verified empirically by implementing a laboratory methodology. The verification confirmed that the proposed real-time security file access monitoring structure monitors all user file access events. The structure proposed in this study notifies the user when he/she accesses a file registered in the monitoring policy. In addition, it was confirmed that the proposed structure does not provide access to files that are different from the file path registered in the monitoring policy. Therefore, it was validated that all monitoring functions of the proposed real-time security file access monitoring structure perform correctly.

Further, it was verified that the monitoring function was sufficiently effective through performance measurement results and it required fewer resources for file access monitoring event notifications than SELinux. It was also verified that the proposed structure had little effect on the operation of other applications. Furthermore, enforcement of the monitoring policy proved to be faster than that of SELinux under the same conditions. Thus, the proposed structure was sufficiently strong and guaranteed real-time performance.

6. Conclusions

In recent security environments, malware reinforced through techniques such as cryptography, obfuscation, and cross-platforms can enter the target system through various paths. It is difficult to detect or prevent malware using IPS, IDS, or antivirus in enterprise environments. Thus, the next best option is that the security manager applies the post-response via real-time file access monitoring to reduce the damage caused by malware. However, the current OS does not provide a real-time file access monitoring function for an application service manager or data owner to detect unauthorized modifications and leakages in important files.

In this paper, a structure was proposed, from the perspective of post-response, to monitor file access in real time. Because the proposed structure enforces a monitoring policy in the kernel, the user cannot bypass the monitoring function. The performance of the proposed real-time file access structure was verified, confirming that target file access-monitoring was correctly provided. The result of the performance verification confirmed that fewer system resources than that in SELinux were utilized and monitoring of the policy enforcement time was sufficiently fast and strong to provide file access monitoring in a real IT service environment.

If the proposed monitoring structure was applied to the enterprise environment, the security manager is able to identify users who access files or directories containing important information on services. Alternatively, because it is possible to check for malware accessing important files, the damage caused by security accidents will be reduced. However, because this study focused only on monitoring files as targets, it is extremely difficult to detect malware that selects other targets as victims, such as application processes and devices. Therefore, we plan to conduct further studies to monitor these additional factors.

Author Contributions: S.-H.H. proposed the idea, conducted the experiments, and wrote the manuscript. D.L. provided advice on the research approach, guided the experiments, and checked and revised the manuscript. All authors have read and agreed to the published version of the manuscript.

Funding: This research was supported by Tongmyong University Research Grant no. 2021A017.

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

References

1. Dewan, P.; Kashyap, A.; Kumaraguru, P. Analyzing social and stylometric features to identify spear phishing emails. In Proceedings of the 2014 APWG Symposium on Electronic Crime Research (eCrime), Birmingham, AL, USA, 23–25 September 2014; Volume 11, pp. 1–13.
2. Allodi, L.; Chotza, T.; Panina, E.; Zannone, N. The need for new antiphishing measures against spear-phishing attacks. *IEEE Secur. Priv.* **2019**, *18*, 23–34. [CrossRef]

3. Huh, J.H. Implementation of lightweight intrusion detection model for security of smart green house and vertical farm. *Int. J. Distrib. Sens. Netw.* **2018**, *14*, 1550147718767630. [CrossRef]
4. Sarıkaya, A. Anomaly-Based Cyber Intrusion Detection System with Ensemble Classifier. Master's Thesis, Middle East Technical University, Ankara, Turkey, 2018.
5. Mohaisen, A.; Alrawi, O. Av-meter: An evaluation of antivirus scans and labels. In *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment;* Springer: Cham, Switzerland, 2014; pp. 112–131.
6. Roseline, S.A.; Geetha, S. A comprehensive survey of tools and techniques mitigating computer and mobile malware attacks. *Comput. Electr. Eng.* **2021**, *92*, 107143. [CrossRef]
7. Abraham, S.; Chengalur–Smith, I. An overview of social engineering malware: Trends, tactics, and implications. *Technol. Soc.* **2010**, *32*, 183–196. [CrossRef]
8. Schrittwieser, S.; Katzenbeisser, S.; Kinder, J.; Merzdovnik, G.; Weippl, E. Protecting software through obfuscation: Can it keep pace with progress in code analysis? *ACM Comput. Surv. CSUR* **2016**, *49*, 1–37. [CrossRef]
9. Preda, M.D.; Maggi, F. Testing android malware detectors against code obfuscation: A systematization of knowledge and unified methodology. *J. Comput. Virol. Hacking Tech.* **2017**, *13*, 209–232. [CrossRef]
10. Barabosch, T.; Gerhards-Padilla, E. Host-based code injection attacks: A popular technique used by malware. In Proceedings of the 9th International Conference on Malicious and Unwanted Software: The Americas (MALWARE), Fajardo, PR, USA 28–30 October 2014 ; pp. 8–17.
11. Najafi, P.; Koehler, D.; Cheng, F.; Meinel, C. NLP-based Entity Behavior Analytics for Malware Detection. In Proceedings of the IEEE International Performance, Computing, and Communications Conference (IPCCC), Austin, TX, USA, 29–31 October 2021; pp. 1–5.
12. Seo, K.K. Development of Certification Program for Application Service Provider: Application Certification. *J. Korea Saf. Manag. Sci.* **2005**, *7*, 97–108.
13. Boniface, M.; Phillips, S.C.; Sanchez-Macian, A.; Surridge, M. Dynamic service provisioning using GRIA SLAs. In *International Conference on Service-Oriented Computing;* Springer: Berlin/Heidelberg, Germany, 2007; pp. 56–67.
14. Buyya, R. Parmon: A portable and scalable monitoring system for clusters. *Softw. Pract. Exp.* **2000**, *30*, 723–739. [CrossRef]
15. Yamiun, M.M.; Katt, B.; Gkioulos, V. Detecting windows-based exploit chains by means of event correlation and process monitoring. In *Future of Information and Communication Conference;* Springer: Cham, Switzerland, 2019; pp. 1079–1094.
16. Mehnaz, S.; Mudgerikar, A.; Bertino, E. Rwguard: A real-time detection system against cryptographic ransomware. In *International Symposium on Research in Attacks, Intrusions, and Defenses;* Springer: Cham, Switzerland, 2018; pp. 114–136.
17. Kazienko, P.; Kiewra, M. Rosa—Multi-agent system for web services personalization. In *International Atlantic Web Intelligence Conference;* Springer: Berlin/Heidelberg, Germany, 2003; pp. 297–306.
18. Anagnostakis, K.G.; Ioannidis, S.; Miltchev, S.; Greenwald, M.; Smith, J.M.; Ioannidis, J. Efficient packet monitoring for network management. In Proceedings of the NOMS 2002. IEEE/IFIP Network Operations and Management Symposium.' Management Solutions for the New Communications World' (Cat. No. 02CH37327), Florence, Italy, 19 April 2002; pp. 423–436.
19. Saez, J.C.; Casas, J.; Serrano, A.; Rodríguez-Rodríguez, R.; Castro, F.; Chaver, D.; Prieto-Matías, M. An OS-oriented performance monitoring tool for multicore systems. In *European Conference on Parallel Processing;* Springer: Cham, Switzerland, 2015; pp. 697–709.
20. Gu, G.; Porras, P.A.; Yegneswaran, V.; Fong, M.W.; Lee, W. Bothunter: Detecting malware infection through IDS-driven dialog correlation. In Proceedings of the USENIX Security Symposium, Vancouver, BC, Canada, 11–13 August 2007; pp. 1–16.
21. Ikegami, Y.; Yamauchi, T. Attacker investigation system triggered by information leakage. In Proceedings of the IIAI 4th International Congress on Advanced Applied Informatics, Okayama, Japan, 12–16 July 2015; pp. 24–27.
22. Hsu, F.; Chen, H.; Ristenpart, T.; Li, J.; Su, Z. Back to the future: A framework for automatic malware removal and system repair. In Proceedings of the 22nd Annual Computer Security Applications Conference (ACSAC'06), Miami Beach, FL, USA, 11–15 December 2006; pp. 257–268.
23. Claffey, G.F.; Regan, H.J. InnovatEDU a collaboration to reduce higher ed security risk. In Proceedings of the 39th Annual ACM SIGUCCS Conference on User Services, San Diego, CA, USA, 6–9 November 2011; pp. 161–164.
24. Hu, X.; Wang, T.; Stoecklin, M.P.; Schales, D.L.; Jang, J.; Sailer, R. Asset risk scoring in enterprise network with mutually reinforced reputation propagation. In Proceedings of the IEEE Security and Privacy Workshops, San Jose, CA, USA, 17–18 May 2014; pp. 61–64.
25. Huh, J.H.; Hwang, S. Development of Java Capstone Design of Network Security Curriculum: Focusing on DDoS Intrusion Detection System. International Information Institute (Tokyo). *Information* **2017**, *20*, 8057–8066.
26. Daryabar, F.; Dehghantanha, A.; Udzir, N.I. Investigation of bypassing malware defences and malware detections. In Proceedings of the 7th International Conference on Information Assurance and Security (IAS), Melacca, Malaysia, 5–8 December 2011; pp. 173–178.
27. Tuscano, A.; Koshy, T.S. Types of Keyloggers Technologies–Survey. In *ICCCE 2020;* Springer: Singapore, 2021; pp. 11–22.
28. Baysa, D.; Low, R.M.; Stamp, M. Structural entropy and metamorphic malware. *J. Comput. Virol. Hacking Tech.* **2013**, *9*, 179–192. [CrossRef]
29. Holt, T.J.; Dupont, B. Exploring the factors associated with rejection from a closed cybercrime community. *Int. J. Offender Ther. Comp. Criminol.* **2019**, *63*, 1127–1147. [CrossRef]

30. Apvrille, A. Cryptography for mobile malware obfuscation. In Proceedings of the RSA Conference Europe, London, UK, 12–13 October 2011.
31. Suk, J.H.; Lee, J.Y.; Jin, H.; Kim, I.S.; Lee, D.H. UnThemida: Commercial obfuscation technique analysis with a fully obfuscated program. *Softw. Pract. Exp.* **2018,** *48*, 2331–2349. [CrossRef]
32. Vu, D.L.; Nguyen, T.K.; Nguyen, T.V.; Nguyen, T.N.; Massacci, F.; Phung, P.H. HIT4Mal: Hybrid image transformation for malware classification. *Trans. Emerg. Telecommun. Technol.* **2020**, *31*, e3789. [CrossRef]
33. Trajanovski, T.; Zhang, N. An automated behaviour-based clustering of IoT botnets. *Future Internet* **2021**, *14*, 6. [CrossRef]
34. Singh, J.; Singh, J. Challenge of malware analysis: Malware obfuscation techniques. *Int. J. Inf. Secur. Sci.* **2018**, *7*, 100–110.
35. Pham, D.P.; Vu, D.L.; Massacci, F. Mac-A-Mal: MacOS malware analysis framework resistant to anti evasion techniques. *J. Comput. Virol. Hacking Tech.* **2019**, *15*, 249–257. [CrossRef]
36. Yan, W.; Zhang, Z.; Ansari, N. Revealing packed malware. *IEEE Secur. Priv.* **2008**, *6*, 65–69. [CrossRef]
37. Kang, B.; Yerima, S.Y.; Sezer, S.; McLaughlin, K. N-gram opcode analysis for android malware detection. *arXiv* **2016**, arXiv:1612.01445. Available online: https://arxiv.org/abs/1612.01445 (accessed on 11 April 2022).
38. Bukac, V. IDS System Evasion Techniques. Master's Thesis, Masarykova Univerzita, Brno, Czech Republic, 2010.
39. Webster, M.P. Formal Models of Reproduction: From Computer Viruses to Artificial Life. Ph.D. Thesis, University of Liverpool, Liverpool, UK, 2008.
40. Payer, U.; Teufl, P.; Lamberger, M. Hybrid engine for polymorphic shellcode detection. In *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment;* Springer: Berlin/Heidelberg, Germany, 2005; pp. 19–31.
41. Yadav, T.; Rao, A.M. Technical aspects of cyber kill chain. In *International Symposium on Security in Computing and Communication;* Springer: Cham, Switzerland, 2015; pp. 438–452.
42. Zhong, M.; Zhou, Y.; Chen, G. Sequential model-based intrusion detection system for IoT servers using deep learning methods. *Sensors* **2021**, *21*, 1113. [CrossRef]
43. Lee, S.; Huh, J.H. An effective security measures for nuclear power plant using big data analysis approach. *J. Supercomput.* **2019**, *75*, 4267–4294. [CrossRef]
44. Park, S.H.; Yun, S.W.; Jeon, S.E.; Park, N.E.; Shim, H.Y.; Lee, Y.R.; Lee, S.J.; Park, T.R.; Shin, N.Y.; Kang, M.J.; et al. Performance evaluation of open-source endpoint detection and response combining google rapid response and osquery for threat detection. *IEEE Access* **2022**, *11*, 1523. [CrossRef]
45. Möller, D.P. Threat Intelligence. In *Cybersecurity in Digital Transformation;* Springer: Cham, Switzerland, 2020; pp. 29–45.
46. Tselios, C.; Tsolis, G.; Athanatos, M. A comprehensive technical survey of contemporary cybersecurity products and solutions. In *Computer Security;* Springer: Cham, Switzerland, 2019; pp. 3–18.
47. Chandel, S.; Yu, S.; Yitian, T.; Zhili, Z.; Yusheng, H. Endpoint protection: Measuring the effectiveness of remediation technologies and methodologies for insider threat. In Proceedings of the International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery (CyberC), Guilin, China, 17–19 October 2019; pp. 81–89.
48. Argaw, S.T.; Troncoso-Pastoriza, J.R.; Lacey, D.; Florin, M.V.; Calcavecchia, F.; Anderson, D.; Burleson, W.; Vogel, J.M.; O'Leary, C.; Eshaya-Chauvin, B.; et al. Cybersecurity of Hospitals: Discussing the challenges and working towards mitigating the risks. *BMC Med. Inform. Decis. Mak.* **2020**, *20*, 146. [CrossRef]
49. WWeissman, D.; Jayasumana, A. Integrating IoT monitoring for security operation center. In Proceedings of the Global Internet of Things Summit (GIoTS), Dublin, Ireland, 3–5 June 2020; pp. 1–6.
50. Mao, R.; Xu, H.; Wu, W.; Li, J.; Li, Y.; Lu, M. Overcoming the challenge of variety: Big data abstraction, the next evolution of data management for AAL communication systems. *IEEE Commun. Mag.* **2015**, *53*, 42–47. [CrossRef]
51. Kuorilehto, M.; Hännikäinen, M.; Hämäläinen, T.D. A survey of application distribution in wireless sensor networks. *EURASIP J. Wirel. Commun. Netw.* **2005**, *5*, 859712. [CrossRef]
52. Blaze, M. A cryptographic file system for UNIX. In Proceedings of the 1st ACM Conference on Computer and Communications Security, Fairfax, VA, USA, 3–5 November 1993; pp. 9–16.
53. Pasquier, T.; Han, X.; Goldstein, M.; Moyer, T.; Eyers, D.; Seltzer, M.; Bacon, J. Practical whole-system provenance capture. In Proceedings of the 2017 Symposium on Cloud Computing, Santa Clara, CA, USA, 24–27 September 2017; pp. 405–418.
54. McDonald, G.; Papadopoulos, P.; Pitropakis, N.; Ahmad, J.; Buchanan, W.J. Ransomware: Analysing the impact on Windows active directory domain services. *Sensors* **2022**, *22*, 953. [CrossRef]
55. Lugo, P.C.; Garcia, J.M.G.; Flores, J.J. A system for distributed SELinux policy management. In Proceedings of the Third International Conference on Network and System Security, Queensland, Australia, 19–21 October 2009; pp. 254–261.
56. Cowan, C. Securing Linux Systems with AppArmor. *DEF CON* **2007**, *15*, 15–26.
57. Wang, J.; Li, D.; Yang, L.; Tan, L.; Wang, H. Security strategy and research of power protection equipment based on SELinux. In *Proceedings of Sixth International Congress on Information and Communication Technology;* Springer: Singapore, 2022; pp. 37–47.
58. Zhu, H.; Gehrmann, C. Lic-Sec: An enhanced AppArmor Docker security profile generator. *J. Inf. Secur. Appl.* **2021**, *61*, 102924. [CrossRef]
59. Reti, D.; Fraunholz, D.; Zemitis, J.; Schneider, D.; Schotten, H.D. Deep down the rabbit hole: On references in networks of decoy elements. In Proceedings of the International Conference on Cyber Security and Protection of Digital Services (Cyber Security), Dublin, Ireland, 15–19 June 2020; pp. 1–11.

60. Kindervag, J.; Balaouras, S. No more chewy centers: Introducing the zero trust model of information security. *Forrester Res.* **2010**, *3*, 7545.
61. Zhao, X.; Borders, K.; Prakash, A. Using a virtual machine to protect sensitive Grid resources. *Concurr. Comput. Pract. Exp.* **2007**, *19*, 1917–1935. [CrossRef]
62. Isohara, T.; Takemori, K.; Miyake, Y.; Qu, N.; Perrig, A. Lsm-based secure system monitoring using kernel protection schemes. In Proceedings of the International Conference on Availability, Reliability and Security, Krakow, Poland, 31 August–2 September 2010; pp. 591–596.
63. Win, T.Y.; Tianfield, H.; Mair, Q. Virtualization security combining mandatory access control and virtual machine introspection. In Proceedings of the IEEE/ACM 7th International Conference on Utility and Cloud Computing, London, UK, 8–11 December 2014; pp. 1004–1009.