Scientific Research Publishing

Journal of Computer and Communications, 2022, 10, 34-49
https://www.scirp.org/journal/jcc
ISSN Online: 2327-5227
ISSN Print: 2327-5219

Reduce Malicious Activity in Trusted Programs

Elliot Ito, Depeng Li

Department of Information and Computer Sciences, University of Hawaii at Manoa, Honolulu, USA
Email: depengli@hawaii.edu

How to cite this paper: Ito, E. and Li, D.P. (2022) Reduce Malicious Activity in Trusted Programs. *Journal of Computer and Communications*, **10**, 34-49.
https://doi.org/10.4236/jcc.2022.1010003

Received: September 14, 2022
Accepted: October 17, 2022
Published: October 20, 2022

Abstract

The malicious activity comes in many forms, but many can come through trusted applications that we commonly use. Current systems have the capability to reduce damages, but implementations for the reduction are either outside of the system or are implemented in a manner that is unintuitive or confusing to users. In this paper, an access control method has been proposed that focuses on the alleviation of damage caused by such applications through the interactions between the user, application, and computer system. In details, the proposed model would work as a module or an interceptor to delegate permissions to applications through user input by using existing system calls. The evaluation about the proposed model as well as the first step implementation can show better security protection than existing systems.

Keywords

Malicious Activity

1. Introduction

Computer users often deploy security software on their computer to protect themselves from malware. This software uses varying techniques to identify potential malicious activity and prevent such actions from occurring on the user's system [1]. However, the problem often lies with the access control model that has been implemented in the system. These models that the systems run allow the malware to execute on the system.

There are three major problems that access control models face:

- Implementation
- Deployment
- Effectiveness

Liang *et al.* proposed a play and rewind feature for unknown applications through process isolation. However, in this system, it assumes user understanding what each file modification implies and the impact of such modifications. In addition this proposal required a custom Linux system for its implementation [2].

Zeldovich *et al.* proposed a confined information flow model that would minimize the amount of trusted code to only kernel. The user would then be able to specify data access policies. This proposal also required a custom Linux system for implementation and usage [3].

ZBAC (authoriZation Based Access Control) was proposed as a distributed authorization model, but it was designed for "enterprise architecture" and not for general computing systems [4].

Roesner *et al.* proposed a usage based access control to limit program access to user data through user's interactions with applications. The model was limited when it left what the application as accessing to information flow models despite the same data being able to compromise the application. The proposal was implemented on a modified OS system from a previous work [5] [6].

For these models, while effectiveness may be met, there is a problem with the implementation and deployment of these models due to the usage of customized systems or kernels.

In this paper, we propose a model that would alleviate malicious intent on trusted applications using existing systems. The model would limit its modification at the kernel level to that of a module, making the implementation and deployment of the model much easier. This is due to the lower entry barrier for the implementation of this model on current systems and improving current system security without the need for modifications to the kernel itself. However there are limitations on the model's effectiveness due to it's inherent nature of user interaction.

In this paper, Section 2 will go over the background of this problem with related work, Section 3 will cover current system implementations of security, Section 4 will go over the proposed model and how it would reduce damage, Section 5 will go over the attempt to implement the system with Section 6 covering the limitations and issues with the implementation. Section 7 will compare how the current solutions as compared to the proposed solution. Section 8 will cover what could be done for improvements to the proposed model and areas of future research. Section 9 will conclude the paper.

2. Background and Related Work

Access control is often based on what a user or application can access based on policies and mechanisms. However, it is common for access control measures to fall short of protecting users in the event that the user is compromised. This can occur though different means such as to incentivize users to run potentially malicious programs [7], to trick users through social engineering [8], compromised or stolen user credentials, and compromised or malicious applications.

There are three areas that are often not considered or alleviated by access control models.

1) *Blanket Authorization*: When a program runs at the permission level of the the user who initiated or granted the application permission to run. This goes against the principle of least privilege due to the program having access to any content that the user has permission to access. An example of this is the Windows UAC prompt and Linux sudo command providing the program high-level permissions without knowing how or where it would be used.

2) *Trusted Application Compromise*: When a program that is trusted by the user has a vulnerability that allows rogue commands to be executed through the program that could run at the user's permission level. An example of this is a backdoor being included with the program as an update.

3) *Trusted Application Usage Compromise*: When a program that is trusted by the user is used in a manner where malicious commands can be executed at a user's permission level from using the program. An example of this would be opening malicious documents that run scripts that would install malware on the user's computer.

Traditional access control models like that of Role Based Access Control (RBAC) [9], Attribute Based Access Control (ABAC) [10] and implementation methods like Access Control Lists (ACL) [11] do not consider these problems as they are out-of-scope, as these models fall under the first area. Once the user or application obtains the permission level, it would be allowed access to everything at that level.

Moving beyond traditional access control models, there have been proposed access control models that attempt to solve the problem of least privilege. The primary problem is the need to implement a least privilege approach to applications that the user would normally trust. This is counter-intuitive because the user already trusts the application with certain permissions.

Liang *et al.* proposal had process isolation to present users what operations were done on files and to allow the user the option to allow the modifications for each file. There is one main problem with the proposal. It is the requirement for the user to check each change and verify that these changes are satisfactory. Even if there was a method of generalizing the file changes to lessen the number of prompts a user would be presented, the user would still have to go through the process each time an application is ran for the first time. This burden is on top of the user being able to understand what modification are and are not trustworthy. In addition to this, it has the problem with the second and third area where a trusted application would not undergo such screening [2].

Zeldovich *et al.* proposal was unique as they proposed a different form of information flow control to restrict access of data that applications can access. This was done through the usage of labels to resources from the kernel. However, the limitation of the model comes from the need for the user themselves designating what permissions a file. While there may be defaults for permission delegation, it would require users to always ensure that the settings on each file are set to what they want [3].

ZBAC (authoriZation Based Access Control) was an approach for "sharing information across traditional domain boundaries" that had a test use case in a NAVY Limited Technical Evaluation (LTE). Being a distributed model, it allowed the enforcement of least privilege for users and programs in addition to better control and delegation of permissions between users and programs. Despite it being a distributed system, it presents the notion of user delegated permissions. Users would designate to the system what access a program would have to a file and the system would only allow the application access to just that file with the specified permission. Later access control papers like Roesner *et al.* [6] have this concept. In addition, it presents a model that can be used to limit damage caused by application compromise, but it does not provide a measurable implementation for standard programs like word processing programs [4].

The usage-based approach taken by Roesner *et al.* showed a novel means of permission delegation through the user's interaction and intent with application. However, its assumptions present a problem when it looked at what data applications were accessing. Considering one aspect of data being accessed by applications as invalid data that could contain malicious content to be protected through input validation does not consider the cases where malicious content is valid input [6]. A prime example of this would be word processing macros that would download and install malware. Macros would use valid syntax to download and execute the downloaded content.

3. Current Systems

Current systems have generally four different types of permission delegation to applications.

1) No Permission Required. Resources that are available to the application like user documents or application required (renderer to draw an image on a smartphone) have no permission requirements.

2) Permission at Install time. Systems like Android use manifest files that dictate what applications are allowed to access to inform the user what the application would need when the user installs the application. Once the application is installed, the application would be able to utilize any of the granted permissions that were in the manifest file.

3) Permission at Runtime. Often done through a user prompt, systems would notify the user asking the user to grant the application permission to do an action.

4) No Permission Granted. For system specific resources, it may not be possible for the application to be granted access. This would also include systems where applications are not granted any permission at all.

Each type of permission delegation presents security implications and usability problems. It is inherently possible for malicious applications to abuse the resources that are allowed without any permission. For permissions at install time and runtime, the user may not know what it means for them to grant permissions to said application. For systems that have no permission delegation, it may be unusable in some cases where users would want to use local system resources.

4. Model Proposal

4.1. Goal

The goal of the security model is to alleviate the damage that a trusted application could do. In order to do this with current systems, there are assumptions about the system and attack surface that is available to adversaries.

4.2. Assumptions

Secure method of updating: Base Operating System (OS) and OS related programs have a secure method of updating.

Proper system configurations—The base OS have properly secured system configurations for the OS (*i.e*. No remote access for the root account for UNIX or UNIX-Like OS.).

Hardware compromise: Devices that run the base OS have hardware that has not been compromised by either the seller of the devices or manufacturer of the hardware.

Secure base Operating System: Devices that use this security model have a secure OS that does not have vulnerabilities that could bypass the proposed security model.

No User Compromise: For this model, it will be assumed that users themselves are not compromised where the adversary would obtain the user's credentials.

Memory Protection: Programs would not be able to have direct access to the system's memory where other application's run time stack.

Network Whitelist: It is assumed that the system has a whitelist to control the domains that can be accessed.

4.3. Available System Features

Windows and Linux systems already have methods that can keep track and monitor application usage of system resources. This would include Window's Resource Monitor that would allow the user to see what files are being accessed from a particular application and Linux's comparable lsof command that shows currently opened files. In addition to file information, both Resource Monitor and lsof can show current network activity on the system. Lastly Windows can suspend processes with Resource Monitor and Linux has a comparable kill command that would suspend processes with the kill command's-STOP flag. This model can be seen as an additional feature to these existing systems to provide the user more control over their application.

4.4. Model

The primary way to reduce damage done by compromised program is limit what the program can do or access. Even if a program is trusted, we should not necessarily grant it access to anything accessible with the user's permission level. The proposed model would be used as a system service that would delegate and verify application requests. Figure 1 shows how the service would interact with the system and the user.

In order for the model to keep track of permissions, programs would have permission metadata that describe what the program can do when running. This would be similar to the manifest files in the Android and other systems described in Section 3. In the event of no metadata being available, the program is defaulted to having no access permissions. Irregardless, the program is initially not allowed any access rights. This is to ensure that it is the user that allows the actions and not a third-party. It is assumed that the user or third-party would not be able to access this metadata information. This metadata would be stored in a directory in the system where there would be an ACL or similar mechanism to prevent tampering.

Actions taken by a program are split into different four different categories:

1) Network. When a program attempts to access the internet.

2) Application. When a program attempts to call another program or application.

3) Kernel. When a program tries to access system level resources.

4) File System. When a program tries to access a file or folder outside of the allowed directories or a file whose file type is not allowed.

For network connections, the user would need to specify if the program is a browser or non-browser application. This is due to the issue of having a whitelist for network connections. While it would make a browser more secure, it would cause a usability problem for users.

Allowed directories are described as meeting the following criterion:

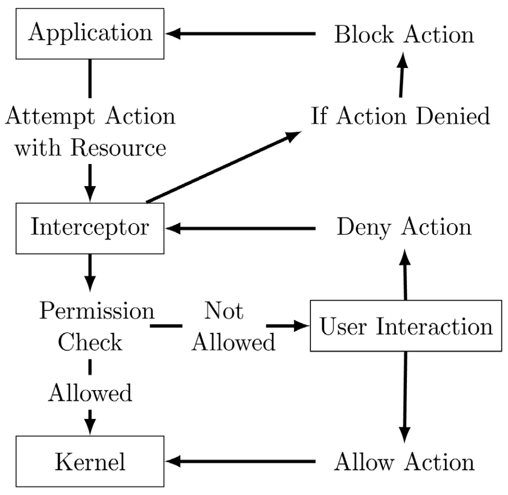

Figure 1. Proposed model using existing system calls, the proposed model would work as a module or an interceptor to delegate permissions to applications through user input.

1) Directories that the user allows at install time

a) Install Directory

b) Standard temporary directories that program requests at install time

2) Directories that the user allows after install time

A limitation to this is the user cannot allow the program to have directory access to all directories in the drive or drives.

Allowed file types are described as meeting the following criterion:

1) File types that the user allows at install time

2) File types that the user allows after install time

Similar to directories, the user cannot allow the application to open any and all file types.

Upon first time attempt of an action, (*i.e.* program accessing internet), the program would be granted explicit access by the user. This would be something similar to prompts may be seen from a security software for the application accessing the internet for the first time. Once this is granted, the program would have access limited by any domain limitations placed by the system via the whitelist. In the scenario that the application attempts to access a domain that is not on the whitelist, the user is prompted with a notification requesting if the user wants to access the domain and if the user wants add the domain to the whitelist for future access. This action for network connections is only for non-browser applications. This is to reduce the user having security fatigue while using the system. For application specific events like directory or filetype access, the model would update the metadata of the application on the user's choice from the notification.

Once permissions are granted to a program, the system would not repeatedly prompt the user. A prompt would not occur unless the program attempts to do something that wasn't done before.

Let us assume a scenario of a common malicious infection. An infected document is loaded with macros that would load malicious data onto the computer. The system would prompt the user if the document reader program attempts to access a domain that is not on the whitelist. Even if the user accepts the network connection, the user would then get prompted for an application execution. The document reader program tried to run the unknown malicious program without having the permissions to do so. From this point, even if the user allows the program permission to run the downloaded program, the system would still not allow the malicious program to run at the same permission levels as that of the document reader program. Each program has a separate permission scheme. As such, the next prompt would be dependent on the malicious application. If the program is ransomware, the user would be prompted for file access to the directory that the ransomware would want to encrypt. If the program is something like a botnet client, the user would get prompts about application usage (depending on the botnet) and network connections (to the command and control server or peers).

However, this is not the only method of getting malicious content to the end user. Drive-by-downloads is another method, using ads or compromised sites to install rootkits on a user's computer. Even if browser programs are not affected by the whitelist requirement, the system is still able to reduce infections. Let us assume a different scenario where the user visits a compromised site or malicious advertisement on a web browser. There is no whitelist check, but the execution of any malicious program would present a prompt for any resource action, as the program was not granted any prior permissions.

Despite it's flexibility, there are limitations and usage issues in this model that are mentioned Section 6.

4.5. Implementation

To implement this model and show it's functionality in different systems, a Windows and Linux system was selected to have the model to be implemented. However, problems arose when the implementation was attempted. This shall be explained in Section 5.

5. Experiment Attempt

The model was attempted to be implemented in both a Linux and Windows environment. However due to time restraints and other issues creating the model, the experiment did not go as planned. The Linux experiment was to be done on a mobile system with a i5-5200U with 8GBs of RAM. The Windows experiment was to be run on a desktop system with a i5-4570 with 16GB of RAM.

5.1. Linux

The Linux implementation was not attempted due to time restraints, instead research into similar programs which were open source or had source-code available was done. From available findings on available source code, it appears to be possible to have a kernel module implementation that would intercept system calls [12] [13]. The alternative to this is if a program is compiled with a shared library or injected with a shared library and use the shared library to intercept function calls [14] [15].

For an implementation for the domain checker, the flow of the model would be as follows:

1) Check syscalls for connect() or calls from a similar function from a shared library injection for a domain request

2) Check requested domain against a hostfile like whitelist file for allowed domains

3) Notify user if the domain is not on the whitelist

4) User selection would determine allowance of

a) No access

b) Temporary access (session)

c) Permanent access (adding domain to whitelist)

5) System would proceed according to user selection

File System events can be seen in the model as follows:

1) Check syscalls for open(),write(), and read(), or calls from a similar function from a shared library injection for a file request

2) Notify user if call does not match the directory listing provided

3) User selection would determine allowance of

a) No access

b) Temporary access (session)

c) Permanent access (adding domain to whitelist)

4) System would proceed according to user selection

From provided examples of this implementation, it is possible to intercept standard system calls (syscalls) from processes and hijack/replace the syscalls with custom ones [13] or use as shared library to hook system calls. [15] Further investigation is needed to verify feasibility of shared library injection to hijack or hook non-syscalls.

5.2. Windows

The main problem that arose when attempting to create a Windows implementation was the time needed for research into creating a system application that would allow the model's feature of request interception.

The interception of system calls can fall under three forms. One would be interception via injection through specific calls [16], hooking of specific system calls through the Windows API [17] [18] [19], or something different—the methods by which security software prevent malicious executables from running. In this case there are two possible methods, one is the hooking being done through the kernel32 or the application specific dlls for command. The second option would be using injection methods [20]. The latter would be more effective for multiple applications whereas the former would specialize for particular applications.

For an implementation for the domain checker, the flow of the model would be the same to that of Linux except for the first step, where it would go as follows:

1) Check system calls in either kernel32 or custom dlls for a domain request

2) Check requested domain against a hostfile like whitelist file for allowed domains

3) Notify user if the domain is not on the whitelist

4) User selection would determine allowance of

a) No access

b) Temporary access (session)

c) Permanent access (adding domain to whitelist)

5) System would proceed according to user selection

Other than this, documentation that was found was primarily for monitoring and not interception of resource calls. An example provided in the File System Watcher Class page on the Microsoft Developer Network (MSDN) website with a slight modification allowed for the viewing of files being modified, but not which applications modifying said files [21]. The reason behind this is due to the events that the watcher captures are only the files that have been changed, created, deleted, disposed, error, or renamed [22].

For a commonly used Windows utility suite SysInternals, the developers no longer provide source code to their applications suite. Within their application suite there is a program that would monitor process activity [23]. It appears that lower or custom dlls (Dynamically Linked Libraries) may be needed for the interception. The time needed to create custom system files to do this exceeded the available time for the project.

6. Limitations and Issues

6.1. Limitations

Due to the assumptions placed on the model, real-world effectiveness would be lower than expected. Some of these limitations are out-of-scope, but impact the mitigation that the model provides to users.

Depending on the implementation on how the OS and programs update, there may be no secure method of updating devices. Even if there is a method of doing so, it is at risk of being denied by adversaries from interception, rerouting, jamming or other means. The proposal is for a security model based on the usage of programs from a local or remote user, and as such the issue of denial of updating the OS or programs through external means is considered out-of-scope of the model.

Preventing and finding hardware compromise is a difficult problem to solve. With the exception of in-house production it is difficult to control the security protocols and procedures that manufacture that produce the different hardware components. Even if the supply chain is secure, computer manufactures may include their own software or firmware for hardware that can be exploited [24]. While the proposal is on access control for programs and this problem being out-of-scope of the model, it is important to note this limitation.

In addition to this, assuming that the base operating system is secure can be naive as it may prove difficult to have a secure base OS for devices due to bugs that may exist within the core OS or any later updates to an OS. Software bugs have been seen with open source projects [25], and Android [26] show the difficulty of ensuring non-exploitable code in production software. However, there are times when an OS feature can be exploited maliciously. Microsoft's atom tables were found to be vulnerable to malicious code injection [27]. What makes it difficult to minimize this threat is due to this being a feature in the OS, which was recently exploited by one of the newest iterations of the banking trojan Dridex [28]. While it may be possible to reduce damage, problems like this make it difficult for the model to be effective.

Memory protection is something that can be difficult to achieve in both old and new systems. RAM scrapers are an issue that can circumvent and eliminate the protections provided by the proposed model by obtaining information from memory without user interaction or knowledge.

By having proper system configuration, it requires that additional steps are required to ensure that the OS is properly setup before it is released or updated. Despite ensuring improved and proper protection against attacks, it may provide additional overhead to deployment. While this can impact the implementation of the system, this is a cost that is outside the scope of the model as it should be assumed that distributors should be focusing on security when it comes to development of devices. However, as noted with the hardware compromise assumption, Original Equipment Manufacturers (OEM) have faced problems with their installation and usage of custom software that is vulnerable or compromised. Both of these problems are considered out-of-scope, but the instances of these events occurring present yet another limitation on the model's effectiveness.

Aside from these limitations from assumptions, the real-world attack threats have capabilities that are not necessarily covered by the model. Fileless malware present a problem for system security. The problem with fileless malware in the scope of the model is how the model would interpret the execution of code. If the model does not see the fileless malware execution as a separate entity, the malware would be able to run without the model notifying the user that the program is executing a different executable.

The model itself is hindered from the usage of current systems. Current systems may not provide who did the system call, the program vs user, like that of Roesnen *et al.* [6]. This information provides a level of detail that would make the model more secure, but would require changes to the kernel and applications. Without this information the model has to attempt least privilege in systems that do not provide detailed information.

6.2. Issues

The model itself has issues with its compatibility with security software and computer policies. While not available to regular Window users, Windows has options for professionals, and users in business and education to control what applications can be ran. Despite this feature requiring user interaction and selection, this limitation feature presents a redundancy for capabilities [29]. Depending on its implementation, the model may conflict with security software as security software do similar actions by preventing execution of malicious code.

In addition to compatibility issues, both Linux and Windows models may face stability issue. Due to the implementation not being implemented, it is unknown the impact that the model would have with security software and kernel. If a Linux kernel module is used as the interceptor, it presents a problem of ensuring robustness in the program to prevent a kernel panic or other kernel related errors when intercepting system calls. A similar problem is possible with the creation of custom system files for system call interception in the Windows model.

The injection of code itself in any program presents a security problem in itself due to the intrusive nature of the model. Secure applications may not assume such level of interception with its system calls. An example of security impacts from intrusive security models is the interception of HTTPS. The issue from this has presented issues which have resulted in a recent US-CERT advisory [30].

Programing implementation issues aside, the User Interface for notification presents a different issue. If users are to be prompted needlessly or without proper information, the notification system would face the same problem with Windows UAC prompts [31]. The prompts need to be efficient and understandable where the user would want to read the message and choose the best option. The problem here is if the user already wanted to take the action and the prompt is a simple verification of it, as noted in Roesner *et al.* [6].

7. Solution Comparisons

Security software solutions provide a means for the identification and prevention of malicious file or code execution. However, the software still depends on signatures and execution heuristics. The problems with false positives and impact on applications create a problem for security software. For troubleshooting, installation, or even running an application, users are often suggested to turn off their security software for the application to run correctly. This prevents the security software from ensuring that the user is properly protected. In a manner similar to Roesner *et al.*, [6] by using using user interactions with applications, the model can reduce such problems through said interaction to dictate program permissions.

Current system security implementations rely heavily on traditional access control models utilizing what users have access to. Linux security modules like SELinux [32] have methods to reduce damage caused by malicious applications to system resources through access prevention, but have limitations for user specific content. Unless provided, users would often have to modify the security module configurations for stricter security policies. This model can limit application access with little user interaction with security configuration files.

Some of the newer features security features in Windows utilize sandboxing that require the application developers to modify their application code to provide further protection [33]. The current model allows for legacy support to applications that are no longer being supported as it utilizes the applications usage of system calls and not an application feature.

8. Future Work

8.1. Linux and Windows Implementation

Although some important work has already been accomplished, it was still not possible to fully implement a system that could be used in Linux and Windows due to the heavy workload. As a possible future work, an implementation of the model for both Linux and Windows could be created to show how well the model works when compared to the current security implementations in place.

More research would be needed on the Linux side to ensure compatibility with older kernel versions and robustness for updates to the Linux kernel. If the model is dependent on checking syscalls, backwards compatibility and the need for legacy checks are required. In addition, shared library injectors need to work for any application that does not use syscalls functions. For Windows, research into creating custom DLLs and methods of application preemption, similar to that of security products need to be researched further.

However, it needs to be noted that Microsoft has been updating the security features in Windows 10. With the latest Creators Update that was rolled out on April 11th, Microsoft is introducing new security features in Windows Defender and Windows 10 to harden the system against threats. However, some of these features like the Windows Defender Advanced Threat Protection are only for paying enterprise customers or systems that normal users may not use (*i.e.* Pro, Educational) [33] [34] [35].

With this in mind, the proposed alleviation model can still be applied to Windows due to the security difference between regular users and users who are professionals, part of an institution or business.

8.2. User Interface

As noted in the model issues in Section 6.2, the model relies on user interaction, which presets a problem when it comes to the User Interface (UI) that they would interact with to allow permissions. Users can often skip Windows UAC prompts not knowing the implications of what they were doing. [31] In addition, the type of prompts being presented to users may not be interpreted in a manner that is correct. This issue has been noted in research on browser information to users on their connection type [36].

Ensuring that there is an understandable UI for users to interact with is a difficult challenge. Even if the information is presented in a manner that is explicit and understandable for users, the manner that the information is presented may affect user's assumption on the prompt.

8.3. User Education

While a problem like user education is out-of-scope of the model, users are often the last line of defense when it comes to unknowingly runing a zero-day exploit. While security software and operating systems make it generalized for users to understand what is secure and what is not, such methods have a limit on the impact on users decision making. If users do not understand the threats well or if they have misconceptions on the threats, their interaction with systems and decision making will circumvent the system security.

Despite the proposed model making it easier for users to understand why they are getting the prompt, like accessing a new website that is not on a list, users may not see the prompt as a warning. Systems can only protect users to a given point as malicious activity can occur with and without user involvement.

8.4. Internet of Things (IoT)

One of the major limitations of the proposed model is its usage of user interaction, which hinders application for IoT devices. In general, there are three main problems that access control models face for IoT devices.

1) Constricted Resources
2) Automation
3) User Compromise

The primary problem is the constricted nature of IoT devices. Security models need to take into consideration the resource limitations that IoT devices have, while. The next problem is how IoT device are not usually managed with human interaction, this means that current proposed model cannot work efficiently. There are a wide array of services that IoT devices provide with devices like security cameras and sensors not requiring much human input. The model will need to consider how to handle permissions with automated services. The last problem is the handling of user compromise. If the model were to be generalized, the last problem would be considered out-of-scope for the model in the case of direct terminal connection to the device. If the device is being controlled via an application, it is theoretically possible for the proposed model to be generalized and adapted to cover this scenario.

9. Conclusion

The current security models and systems that are in use need to be updated. The proposed system can be considered as a stop-gap, as it does not fully resolve the current problems that security models face. Despite this, updates to models are a continuing cycle for security development. The inclusion of alleviation techniques on trusted applications is a point of interest to ensure protection for users who may trust their applications beyond the current security capabilities. The evaluation about the proposed model as well as the first step implementation can show better security protection than existing systems. The future work for this research is to fully implement the model for both Linux and Windows to demonstrate how well the model works while compared to the current security implementations.

Conflicts of Interest

The authors declare no conflicts of interest regarding the publication of this paper.

References

[1] Saeed, I.A., Selamat, A. and Abuagoub, A. (2013) A Survey on Malware and Malware Detection Systems. *International Journal of Computer Applications*, **67**, 25-31.

https://doi.org/10.5120/11480-7108

[2] Liang, Z.K., Venkatakrishnan, V.N. and Sekar, R. (2003) Isolated Program Execution: An Application Transparent Approach for Executing Untrusted Programs. 19*th Annual Computer Security Applications Conference*, 2003. *Proceedings*, Las Vegas, 8-12 December 2003, 182-191.

[3] Zeldovich, N., Boyd-Wickizer, S., Kohler, E. and Mazi'eres, D. (2006) Making Information Flow Explicit in Histar. *Proceedings of the* 7*th Symposium on Operating Systems Design and Implementation*, Seattle, 6-8 November 2006, 263-278.

[4] Karp, A., Haury, H. and Davis, M. (2010) From ABAC to ZBAC: The Evolution of Access Control Models. *Journal of Information Warfare*, **9**, 38-46.

[5] Wang, H.J., Moshchuk, A. and Bush, A. (2009) Convergence of Desktop and Web Applications on a Multi-Service OS. *Proceedings of the* 4*th USENIX Conference on Hot Topics in Security*, Montreal, August 2009, 11.

[6] Roesner, F., Kohno, T., Moshchuk, A., Parno, B., Wang, H.J. and Cowan, C. (2012) User-Driven Access Control: Rethinking Permission Granting in Modern Operating Systems. 2012 *IEEE Symposium on Security and Privacy*, San Francisco, 21-23 May 2012, 224-238. https://doi.org/10.1109/SP.2012.24

[7] Christin, N., Egelman, S., Vidas, T. and Grossklags, J. (2011) It's All about the Benjamins: An Empirical Study on Incentivizing Users to Ignore Security Advice. *International Conference on Financial Cryptography and Data Security*, Gros Islet, 28 February-4 March 2011, 16-30. https://doi.org/10.1007/978-3-642-27576-0_2

[8] Gupta, S., Singhal, A. and Kapoor, A. (2016) A Literature Survey on Social Engineering Attacks: Phishing Attack. 2016 *International Conference on Computing, Communication and Automation* (*ICCCA*), Greater Noida, 29-30 April 2016, 537-540. https://doi.org/10.1109/CCAA.2016.7813778

[9] Sandhu, R.S., Coyne, E.J., Feinstein, H.L. and Youman, C.E. (1996) Role-Based Access Control Models. *Computer*, **29**, 38-47. https://doi.org/10.1109/2.485845

[10] Hu, V.C., Kuhn, D.R. and Ferraiolo, D.F. (2015) Attribute-Based Access Control. *Computer*, **48**, 85-88. https://doi.org/10.1109/MC.2015.33

[11] McCollum, C.J., Messing, J.R. and Notargiacomo, L. (1990) Beyond the Pale of Mac and Dac-Defining New Forms of Access Control. 1990 *IEEE Computer Society Symposium on Research in Security and Privacy*, Oakland, 7-9 May 1990, 190-200. https://doi.org/10.1109/RISP.1990.63850

[12] Shtober, K. (2016) Execmon. https://github.com/kfiros/execmon

[13] Nestorov, A. (2016) Monks. https://github.com/alexandernst/monks

[14] gaffe23 (2016) linux-inject. https://github.com/gaffe23/linux-inject

[15] NetSPI (2013) Function Hooking Part I: Hooking Shared Library Function Calls in Linux. https://blog.netspi.com/function-hooking-part-i-hooking-shared-library-function-calls-in-linux

[16] Bremer, J. (2012) Intercepting System Calls on x86 64 Windows. http://jbremer.org/intercepting-system-calls-on-x86_64-windows/

[17] Hooks (2017). https://msdn.microsoft.com/en-us/library/windows/desktop/ms632589(v=vs.85).aspx

[18] Husse, C. and Stenning, J. (2012) Easyhook. https://github.com/EasyHook/EasyHook

[19] Batra, R. (2013) Api Monitor. http://www.rohitab.com/apimonitor

[20] Malik, A. DLL Injection and Hooking. http://securityxploded.com/dll-injection-and-hooking.php

[21] (2017) File System Watcher Class. https://msdn.microsoft.com/en-us/library/system.io.filesystemwatcher(v=vs.110).aspx

[22] (2017) File System Watcher Event. https://msdn.microsoft.com/en-us/library/system.io.filesystemwatcherevents(v=vs.110).aspx

[23] (2009) Sysinternals Licensing Faq. https://technet.microsoft.com/en-us/sysinternals/bb847944

[24] Merzdovnik, G., Falb, K., Schmiedecker, M., Voyiatzis, A.G. and Weippl, E. (2016) Whom You Gonna Trust? A Longitudinal Study on TLS Notary Services. *IFIP Annual Conference on Data and Applications Security and Privacy*, Trento, 18-21 July 2016, 331-346. https://doi.org/10.1007/978-3-319-41483-6_23

[25] Durumeric, Z., Kasten, J., Adrian, D., Halderman, J.A., Bailey, M., Li, F., Weaver, N., Amann, J., Beekman, J., Payer, M., *et al.* (2014) The Matter of Heartbleed. *ACM Proceedings of the* 2014 *Conference on Internet Measurement*, Vancouver, 5-7 November 2014, 475-488. https://doi.org/10.1145/2663716.2663755

[26] Shezan, F.H., Afroze, S.F. and Iqbal, A. (2017) Vulnerability Detection in Recent Android Apps: An Empirical Study. 2017 *IEEE International Conference on Networking, Systems and Security* (*NSysS*), Dhaka, 5-8 January 2017, 55-63. https://doi.org/10.1109/NSysS.2017.7885802

[27] Liberman, T. (2016) Atom-Bombing: A Code Injection That Bypasses Current Security Solutions. http://blog.ensilo.com/atombombing-acode-injection-that-bypasses-current-security-solutions

[28] Baz, M. and Safran, O. (2017) Dridex's Cold War: Enter Atombombing. https://securityintelligence.com/dridexs-cold-war-enter-atombombing

[29] Corio, C. and Sayana, D.P. (2008) Security: Application Lockdown with Software Restriction Policies. https://technet.microsoft.com/en-us/library/2008.06.srp.aspx

[30] Https Interception Weakens TLS Security—Us-Cert. https://www.us-cert.gov/ncas/alerts/TA17-075A

[31] Motiee, S., Hawkey, K. and Beznosov, K. (2010) Do Windows Users Follow the Principle of Least Privilege? Investigating User Account Control Practices. *ACM Proceedings of the Sixth Symposium on Usable Privacy and Security*, Redmond, 14-16 July 2010, 1. https://doi.org/10.1145/1837110.1837112

[32] Smalley, S., Vance, C. and Salamon, W. (2001) Implementing Selinux as a Linux Security Module. *NAI Labs Report*, **1**, 139.

[33] (2017) Mitigate Threats by Using Windows 10 Security Features (Windows 10). https://technet.microsoft.com/en-us/itpro/windows/keep-secure/overview-of-threat-mitigations-in-windows-10

[34] (2016) Introducing Windows Defender Application Guard for Microsoft Edge. https://blogs.windows.com/msedgedev/2016/09/27/application-guard-microsoft-edge

[35] (2017) Windows Defender Advanced Threat Protection (atp). https://www.microsoft.com/en-us/WindowsForBusiness/Windows-ATP

[36] Clark, J. and van Oorschot, P.C. (2013) Sok: Ssl and https: Revisiting past Challenges and Evaluating Certificate Trust Model Enhancements. 2013 *IEEE Symposium on Security and Privacy* (*SP*), Berkeley, 19-22 May 2013, 511-525. https://doi.org/10.1109/SP.2013.41