System Configuration as a Privilege*

Glenn Wurster gwurster@scs.carleton.ca
Paul C. van Oorschot paulv@scs.carleton.ca
School of Computer Science
Carleton University, Canada

Abstract

We present a new approach for separating configuration privilege from traditional root privilege. We limit this new configuration privilege to a single (new) system daemon, `configd`. This daemon reads requests for changes in system configuration, either allowing or denying each request based on various criteria (possibly including user input). We do not allow any other application to run with configuration permission, forcing all requests for a change in system configuration to be processed by `configd`. We discuss the basic functionality required for `configd` to protect system configuration, and some preliminary improvements to a basic prototype design. We concentrate on only those system configuration changes performed through the modification of a file on disk.

1 Introduction and Overview

In the server environment, the root (or Administrator) privilege is normally restricted to system daemons and those applications responsible for changing the configuration of a system (loosely: what code runs and how). The same is increasingly the case in desktop environments, with the root user being reserved for performing special activities. The current root privilege, however, does not adequately protect a system against malicious configuration changes.

The principle of least privilege dictates that an application should only be given the privileges necessary to perform its task [13, 14]. While attempts to apply this principle to software running on a system abound, the principle of least privilege has apparently been applied less as it relates to the software installation process itself. Typically, software installers are given full access to the system, being allowed to make any configuration changes they desire. In today's computing environment, installers are often downloaded from the Internet (in addition to being shipped on CD). Almost all users who desire the functionality reported to be provided by a to-be-installed program will happily run the installer. Even in environments where users do not run as root for the majority of tasks, users purposefully become root to install software (Windows "helpfully" presents a dialogue box to elevate an application's privileges to Administrator if the executable is called `setup.exe`). The risk that follows from being able to easily modify arbitrary system configuration is compounded by the fact that the privileges required to install software are often not separated from those given to system daemons, which are also subject to exploit.

In this paper, we separate the privilege to configure a system (the *configure privilege*) from the traditional *root privilege*. While other systems have separated the configure privilege from root in the past, we believe the historical approach of treating the configure privilege as any other privilege given to applications is misguided. We approach the problem differently, granting the configure privilege to a single process which in turn mediates requests for changes in system configuration. We explore the benefits of mediating requests to configure a system through a configuration daemon, including the ability to involve the user.

We limit our discussion of system configuration to those operations having a direct visible effect on either configuration files or applications on disk (i.e., configuration operations which are persistent across reboots) – this includes scripts, startup files (related to both the operating system and applications) and any executable file (including libraries). While it may be possible to expand this approach to include other configuration operations (e.g. network interface modifications), they are not the focus of this paper.

In Section 2, we provide motivation for a core con-

*Version: July 8, 2009. USENIX HotSec 2009.

figuration daemon. Section 3 describes a possible configuration daemon. Section 4 discusses related work. We conclude in Section 5.

2 Motivating a Configuration State

We are not the first to propose a configuration state which is distinct from root privilege. We are also not the first to allow as a (optional) requirement user input to transition into the configuration state. Systems such as DTE-enhanced UNIX [20] and XENIX [18] have separated out configuration privilege and both hardware keys [1] and read-only media [7] have attempted to tie the granting of configuration privilege to a physical action performed by the user. To our knowledge, however, we are the first to attempt to restrict configuration privilege to a single process. While current systems attempt to limit the privilege to configure a system by separating it from the privileges required to perform routine tasks, this separation has proven to be ineffective in the presence of malware. The combined effect of both installing software and running daemons as root leads to a situation where it is altogether too easy to gain configuration privilege on a running system. To better protect the configuration of a system, it becomes clear that the granting of configuration privilege should be more tightly held and more granular.

The isolation of configuration privileges allows one to restrict what applications can do when performing standard tasks. Many configuration operations are inherently dangerous, including formatting hard drives, editing system configuration, and deleting program/system files. To allow these operations by any program (having broad privileges such as root) during standard system operation is an apparently unnecessary risk. Many viruses have exploited the fact that configuration operations can be performed without additional privileges to infect files, disable virus scanners, install themselves into the boot sequence, and infect the running kernel (among other activities) [15].

The end-user of a computer system (or IT department in corporate environments) is normally involved in any configuration changes within the system (either directly, by making them manually, or indirectly, by approving/initiating them). We believe that for a typical program, obtaining configuration privilege without the assistance of a physical user of that system should be much harder than it currently is. We propose that configuration permission should only be given to one process, `configd`.

3 A configd Prototype

One of the key aspects in limiting configuration is preventing programs from routinely running in configuration mode. It seems creating a configuration state on a system and then allowing arbitrary installers to operate in this space is a flawed approach. We can not be guaranteed that installers will limit their actions when given configuration permission (the installer may be malicious or badly written).

While operating systems mediate what applications can do to a system, traditional access control methods have focused on a preconfigured allow/deny rule-set, giving the user no real-time say in what configuration changes should be allowed and denied. While the operating system (OS) could be modified to query the user every time a configuration change is requested, the constant barrage of fine-grained queries would be a usability nightmare (we discuss user interaction in Section 3.3). In an effort to keep the OS kernel from becoming much bigger, we focus on a solution running in user-space, working alongside the kernel to involve the user in configuration changes.

In installing software, a user will typically become root (e.g., by entering a password, running `sudo`, or logging in as Administrator) and then run the application installer. To date, this explicit transition to root level control and subsequent running of an installer was considered sufficient security. It was assumed a user would not transition to root and run the installer unless they really did intend to install the application associated with the installer, and that the installer would restrict its system changes to only those required to get the specific application running on the system – in effect, trusting that all installers are non-malicious. Indeed, today installers may make any number of configuration changes to a system against the user's wishes (a user has no way of knowing what an installer is doing). Typically, the scope of configuration changes made by an installer is at the sole discretion of the software's developer, putting us at the mercy of the imagination of an attacker. Even when installers are created by legitimate entities, the configuration changes can still involve privileges beyond those expected (e.g., consider P2P file sharing clients [4] or the Sony Root-Kit [5]).

That we cannot trust installers to self-police the

scope of their actions clearly motivates the need for a mediator with the ability to accept or reject system configuration changes. We thus present `configd`, an in-progress prototype daemon designed to mediate requests for configuration changes. Its purpose is to work alongside the operating system kernel (a modified Linux 2.6.28.7 kernel in our prototype) to protect the configuration of a system. In this proposal, the OS kernel is responsible for:

1. Restricting to programs running with configure permission the ability to delete, move, and write to configuration-related files. "root" should no longer be allowed to make arbitrary configuration changes to a system (including the kernel) – these changes should be limited to the program running with configuration privilege (`configd`). While we do not mandate any specific method for distinguishing configuration-related files, storing a mark (discussed in Section 3.1) in file metadata is one suitable approach (and the approach used in our prototype).
2. Restricting the ability to obtain configuration permission. In this proposal, this is done by allowing only a single process to have configuration permission.
3. Restricting the ability to control the process running with configuration permissions (e.g. by not allowing the process to be killed or modified by a debugger).

The `configd` daemon is responsible for:

1. Responding to requests for configuration changes from other applications on the system (e.g., through a UNIX socket).
2. Communicating with the user regarding configuration changes on the system (e.g., through USB keys, a keyboard, and the display).
3. Notifying the kernel (through a yet to be specified interface) which files are marked as being configuration related.
4. Performing allowed changes to the configuration of the system.

We rely on `configd` being started early during the boot process (`configd` itself prevents malicious changes to the boot process). Once `configd` has started, other programs are prevented by the OS kernel from obtaining configuration permission.

In developing `configd`, we take advantage of the temporal nature of software installs. We assume the operating system and `configd` are already installed and running on a system. If `configd` is made part of the OS or core system, this assumption is reasonable.

3.1 Mediating Configuration Changes

By proxying all configuration requests through `configd`, we can limit the configuration privilege on a system. `configd` will respond to requests from other applications, queueing up configuration change requests until such time as a USB key previously associated with the request is inserted. In relying on USB keys, we borrow pieces from the work of Butler et al. [1], who coarsely tie configuration of a system to USB keys, relying on the user to insert the USB key associated with a disk block before that block can be modified (in their work, a specific USB key can be associated with one or more different disk blocks). We adapt their work, tying files to a particular USB key (instead of blocks; blocks are too fine a granularity for our goal). Butler et al. prevent modifications to the disk when the USB key is not inserted, but do not limit which applications can run once the key is inserted; an installer need only ask the user to insert the USB key in order to obtain the privileges required to modify the file of its choosing. Any malicious daemon running on their system would be equally capable of waiting for a USB key to be inserted. In contrast, when a USB key is inserted in the prototype `configd` system, only `configd` and the kernel remain running.

In addition to `configd` using USB keys, standard system IO devices such as the keyboard and video display can be used to interact with the user. By interacting with the user through these, configuration changes performed when a key is inserted can be further restricted (for example, in our prototype, all processes other than `configd` are suspended by the kernel to prevent standard system IO devices being controlled by malware during configure operations). While `configd` is capable of supporting many USB keys, we foresee most non-technical users typically using only one or two.

In order for system configuration to be possible at all, `configd` must support some basic operations. We now discuss some simple, basic requests that it should be able to process, respond to, or queue. We note several caveats which should be considered in implementing these operations.

1. **Marking a File**. If modifications to a file are to be restricted based on the USB key inserted, it must be possible to mark a file as requiring a specific key to be inserted. While Butler et al. make the association implicit by tagging all files modified while the USB key is inserted, the association is made explicit in our prototype

(by requiring applications explicitly mark files as configuration-related by calling `configd`). While implicit tagging has the benefit of being more easily integrated into current systems, explicit tagging avoids the problem of mistakenly tagging certain files. We foresee software installers explicitly marking files by calling `configd`.

2. **Move a File**. In order to properly configure a system, one needs to be able to at least replace a file with a new version. `configd` therefore supports the move operation, allowing a new version of a marked file to replace the version currently installed on a system.
3. **Delete a File**. In order to uninstall an application, `configd` must support being able to delete a file. We note that when a file is deleted, the associated mark (discussed in item 1) is also removed.

3.2 Creating USB Configuration Keys

To ease deployment of systems supporting `configd` and associated USB keys, we suggest using off-the-shelf USB flash drives. In using standard USB flash drives, however, one must ensure that software without configuration permission can not "create" a USB key recognized by `configd` from any USB drive inserted into the machine (and hence enable configuration mode).

As a solution, we propose that the file mark be of the form $m = H(p)$, where H is a one-way hash function, p is a random and suitably long secret which is stored on the USB key, and m is the mark associated with a file. All files on a given machine protected with the same USB key would have the same mark m. In order to verify the appropriate key is inserted, p is hashed and compared to m. Assuming p can not be easily guessed, an attacker is not able to create a new USB key which can be used to authorize changes to a file protected with hash m. This approach also protects against what we term the *Curious George attack* – attackers spreading portable storage devices (e.g., CD's and USB keys) in an area like a parking lot or lobby with the intention that victims will be curious and insert the media into their computer to have a look. With cryptographic file marks, malicious media found by the user will not be recognized as a valid USB key for the system.

3.3 Confirming Configuration Changes with the User

When a USB key pre-configured with `configd` marks is inserted, `configd` will go through all queued configuration change requests and determine which files are associated with the key that has been inserted. For those configuration operations involving files which are associated with the currently inserted USB key, `configd` will furthermore query the user for permission to perform the change.

The naïve approach to querying the user would be to list each requested configuration change, including the old and new versions of the file. Most users will not find such information helpful – indeed, it is likely that many users would accept all configuration requests. How best to reliably involve the user remains an issue that is obviously very important, but not one that we claim to solve in the present note – our main focus here is on proposing a base system to isolate configuration privileges which in its first instance, appears suited mainly for technical users or IT staff. We present, however, a possible improvement in Section 3.5.2.

3.4 Deployment

Because `configd` prevents installers from directly modifying system configuration, we acknowledge it can not be deployed in current mainstream computer systems. Installers (and auto-updaters) will fail if they are denied permission to install, update, and modify configuration files and binaries on a system. More and more application environments, however, are also providing generic application installer frameworks. These frameworks provide the functionality required to install an application, not requiring the application to provide its own installer. This trend toward providing install frameworks exists in both the open source software community (through package managers [2]) and in Windows (through the Microsoft Installer (MSI) [11]). The introduction of these installer frameworks may ease the future deployment of `configd`. As long as configuration changes on a system are not appropriately constrained however, we will continue to see malware and other application installers take advantage of overly broad permissions.

3.5 Extensions to configd

While `configd` and its supporting system does protect against malicious configuration changes on a running system, the basic system, as noted, is at best suitable for advanced users. We now discuss two extensions to the base system. We expect these and other possible extensions will improve the usability of `configd`, and encourage research in this area.

3.5.1 Updates without the USB Key

In the basic configuration daemon setup, all operations to a marked file require user authorization and the presence of the associated USB key. As an improvement, operations involving the replacement of one binary from a software vendor with a new version from the same vendor could use a system such as self-signed executables (SSE) [21] instead of requiring the USB key to be inserted. SSE restricts binary replacement to new versions of the binary provided by the same software source, significantly reducing the risk to system configuration. Because software updates (including security updates) are now common on binaries, SSE greatly reduces the number of times a user would need to insert/remove the USB key. Application installs/removals and configuration file changes would still require the USB key. The prototype `configd` is being extended to enforce SSE (shifting SSE enforcement from the kernel into `configd`).

3.5.2 Path-Description Aliasing

Because few users can be expected to know what each file on a system is used for, we suggest aliasing as an improvement on the naïve approach. We expand the list of commands accepted by `configd` (see Section 3.1) to include one more: the associate command, allowing a specific path prefix to be associated with a text description. As an example, `C:\Windows\System` could be associated with "The Windows System". Such associations would be permanent (as long as files with the path prefix remain on the system) and stored on the USB key along with p, the random key associated with files under the prefix. The stored description on the USB key allows `configd` to ask the user whether they want to allow changes to "The Windows System" instead of `C:\Windows\System`. We envision that the alias association would be set during the initial install of an application by the installer, similar to how an application would call `configd` to mark each file (as discussed above). In order to prevent abuse of the aliasing system, new aliasing prefixes would be limited to those files not already associated with a prefix (e.g., `C:\Windows\System32` could not be associated with a description if `C:\Windows\System` had already been associated). Like the core `configd` proposal, this technique leverages, for the benefit of security, the order software is installed.

Once path prefixes are associated with a description, it becomes possible to group configuration requests based on the description which will be presented to the user. Instead of asking 10 questions, one for each file in the `C:\Windows\System` directory, `configd` could be programmed to ask whether modifications to "The Windows System" should be allowed. This change may provide considerable benefit to the user but the design requires further study.

While it remains unclear how to involve non-technical users in configuration tasks while minimizing the chance of error (both accidental and through social engineering), both SSE and aliasing provide some initial steps.

3.5.3 Compatibility with Current Installers

While the basic proposal requires installers to explicitly request system configuration changes through `configd`, our system could be extended to work with current installers as follows. Attempts to modify protected files are already detected by the modified kernel (as discussed above) and rejected. These requests could instead be forwarded to `configd` by the modified kernel, being processed like any other request to `configd`. This approach allows current installers to operate within the `configd` framework.

4 Related Work

While projects such as DTE-enhanced UNIX [20] and XENIX [18] restrict the privileges of root (including root's ability to configure the system), we are unaware of any such privilege systems restricting which applications can run with configuration privileges – i.e., it seems any installer can be run with such privileges, again still having full access to all files on disk. For systems using the OpenBSD `schg` [8] and ext2 immutable [19] flags, any application can be given the ability to change an immutable file – the user can simply be asked to run an application after acquiring sufficient configuration privileges. SVFS [22] protects files on disk but is susceptible to the same problem of inadequate control over installation applications.

There have been many attempts to detect malicious modifications to system configuration. Windows file protection (WFP) [9, 3] maintains a database of specific files which are protected, along with signatures of them. WFP is designed, however, to protect against a non-malicious end-user, preventing only accidental system modification. Pennington et al. [12] proposed implementing an intrusion detection system in the storage device to detect suspicious modifications. Strunk et al. [17] proposed logging all file modifications for a period of time to assist in the recovery after malware infection. Tripwire [6] maintains cryptographic hashes of all files in an attempt to detect modifications. All these attempts rely on detecting modifications after the fact.

Applications such as registry watchers [16] and clean uninstallers [10] attempt to either detect or revert changes made to a system by an application installer. These systems don't actually prevent changes in system configuration. The separation of configuration privileges as proposed in `configd` prevents installers from making unauthorized changes to system state, leading to a proactive rather than reactive approach to limiting system configuration changes. Package managers [2] and MSI [11] both limit system configuration actions allowed by packages designed for their system, but do not prevent applications from simply providing their own installer, bypassing the limits enforced by the package manager.

5 Concluding Remarks

The separation of configuration privilege from the traditional root privilege better facilitates applying the principle of least privilege to application installers and other root-level processes, to better protect a system against unauthorized or unwanted configuration changes. While we mention two improvements to the basic `configd` framework in this note, this proposal and related prototype is work-in-progress. While we do not propose that it is suitable for universal adoption, we expect that `configd` will prove suitable in security-conscious environments (e.g., national defence, government, financial infrastructure).

Acknowledgements. We thank anonymous referees for helpful comments. The second author acknowledges NSERC for an NSERC Discovery Grant and his Canada Research Chair in Network and Software Security. Partial funding from NSERC ISSNet is also acknowledged.

References

[1] K. R. B. Butler, S. McLaughlin, and P. D. McDaniel. Rootkit-resistant disks. In *Proc. 15th ACM CCS*, Oct 2008.

[2] J. Cappos, J. Samuel, S. Baker, and J. H. Hartman. A look in the mirror: Attacks on package managers. In *Proc. 15th ACM CCS*, Oct 2008.

[3] J. Collake. Hacking Windows file protection. Web Page, May 2007. `http://www.bitsum.com/aboutwfp.asp`.

[4] B. Edelman. Comparison of unwanted software installed by P2P programs. Web Page, Mar 2005. `http://www.benedelman.org/spyware/p2p/`.

[5] J. A. Halderman and E. W. Felten. Lessons from the Sony CD DRM episode. In *Proc. 13th USENIX Security Symp.*, Aug 2006.

[6] G. H. Kim and E. H. Spafford. The design and implementation of Tripwire: A file system integrity checker. In *Proc. 2nd ACM CCS*, 1994.

[7] Knoppix Linux. Web Page (accessed 15 Dec 2008). `http://www.knoppix.net`.

[8] Y. Korff, P. Hope, and B. Potter. *Mastering FreeBSD and OpenBSD Security*, chapter 2.1.2. O'Reilly, 2005.

[9] Microsoft. Description of the Windows file protection feature. Web Page, May 2007. `http://support.microsoft.com/kb/222193`.

[10] Microsoft. Description of the windows installer cleanup utility. Technical Report Q290301, Microsoft, 2008. `http://support.microsoft.com/kb/290301`.

[11] J. Moskowitz and D. Sanoy. *The Definitive Guide to Windows Installer Technology*. Realtimepublishers.com, 2002. `http://nexus.realtimepublishers.com/dgwit.php`.

[12] A. Pennington, J. Strunk, J. Griffin, C. Soules, G. Goodson, and G. Ganger. Storage-based intrusion detection: Watching storage activity for suspicious behavior. In *Proc. 12th USENIX Security Symp.*, Aug 2003.

[13] J. H. Saltzer and M. D. Schroeder. The protection of information in computer systems. *Proceedings of the IEEE*, 63(9):1278–1308, Sep 1975.

[14] F. B. Schneider. Least privilege and more. *IEEE Security & Privacy Magazine*, 1(5):55–59, Sep 2003.

[15] E. Skoudis and L. Zeltser. *Malware: Fighting Malicious Code*. Prentice Hall, 2003.

[16] E. Software. Registry watch. Software Application (viewed 23 Apr 2009). `http://www.easydesksoftware.com/regwatch.htm`.

[17] J. Strunk, G. Goodson, M. Scheinholtz, C. Soules, and G. Ganger. Self-securing storage: Protecting data in compromised systems. In *Proc. 4th USENIX Symp. on Operating Systems Design and Implementation*, Oct 2000.

[18] Trusted XENIX version 3.0 final evaluation report. Technical Report CSC-EPL-92-001, National Computer Security Center, Apr 1992.

[19] C. Tyler. *Fedora Linux*, chapter 8.4. O'Reilly, 2007.

[20] K. M. Walker, D. F. Sterne, M. L. Badger, M. J. Petkac, D. L. Sherman, and K. A. Oostendorp. Confining root programs with domain and type enforcement (DTE). In *Proc. 6th USENIX Security Symp.*, Jul 1996.

[21] G. Wurster and P. van Oorschot. Self-signed executables: Restricting replacement of program binaries by malware. In *USENIX 2007 HotSec*, Aug 2007.

[22] X. Zhao, K. Borders, and A. Prakash. Towards protecting sensitive files in a compromised system. In *Proc. Third IEEE International Security in Storage Workshop (SISW'05)*, 2005.