Application Containers without Virtual Machines

Micah Sherr
Dept. of Computer and Information Science
University of Pennsylvania
Philadelphia, PA USA
msherr@cis.upenn.edu

Matt Blaze
Dept. of Computer and Information Science
University of Pennsylvania
Philadelphia, PA USA
blaze@cis.upenn.edu

ABSTRACT

This position paper introduces lightweight *cryptographic jails* (CryptoJails) that protect the privacy of application data by intercepting write accesses and redirecting them to encrypted application containers. CryptoJails ensure that application data (for example, cached emails or web pages) cannot be read or undetectably altered by other applications. Unlike existing approaches, CryptoJails do not require kernel modifications or even superuser (i.e., root) privileges, do not impose significant performance overhead, and may even be used with already installed applications.

Categories and Subject Descriptors

D.4.6 [**Security and Protection**]: Access controls; D.4.3 [**File Systems Management**]: Access methods

General Terms

Design, Security

Keywords

Application Containers, File System Security, Virtual Machines

1. INTRODUCTION

Virtual machines (VMs) enable software isolation by partitioning system resources into separated containers. In principle, an operating system or application executing within one VM cannot access resources (in particular, files or memory) within another VM except via network protocols. From a security perspective, VM isolation confines the actions of a faulty, compromised, or malicious application to a particular VM, ensuring that a vulnerability in one service does not act as a stepping stone against other services or system resources [5].

We argue that although software isolation via VMs may be appropriate for server appliances, such VM techniques are often impractical for *client-side* personal computing devices. Users of such systems may operate a myriad of applications: web browsers, mail clients (MUAs), spreadsheets, word processors, instant messaging applications, etc. Restricting each application to its own VM provides software isolation (and consequently, increased security), but does so at the expense of usability. To ensure acceptable user experience, applications must share access to the same windowing system, exchange messages, etc. Maintaining separate VM environments – that is, installing OS patches, updating libraries, configuring virtual network devices, etc. – is too burdensome to all but the most security conscious computer operators.

In this position paper, we propose *cryptographic jails* or *CryptoJails* that share many of the advantages of more heavyweight virtual machine isolation techniques. Here, our goal is not to quarantine faulty or compromised software (although our techniques provide some protection against applications that behave maliciously), but rather to tradeoff some security in favor of usability to the end-user. CryptoJails consist of *lightweight application containers* (LACs) that store the encrypted contents and metadata of files and directories written to by their corresponding applications. CryptoJails do not require kernel modifications or even superuser (i.e., root) privileges, do not impose significant performance overhead, and may even be used with already installed applications.

CryptoJails provide the following privacy guarantee:

Data within an application's CryptoJail cannot be read or undetectably altered by another application outside of the CryptoJail, provided the kernel has not been compromised.

For example, the cache of emails stored by a CryptoJail-protected email client cannot be accessed by a compromised web browser on the same host. Similarly, a corrupted NTP client cannot decipher the logs belonging to a CryptoJail-protected instant messaging application. Although a compromised application may purposefully communicate private data (e.g., by sending them over the network), the private contents of a well-behaving application cannot be accessed by any other process. We do not attempt to correct vulnerable applications or to detect application compromises. Rather, our techniques provide a simple and lightweight mechanism in which users can protect the contents of application data from unauthorized processes.

VMSec'09, November 9, 2009, Chicago, Illinois, USA.

2. CRYPTOJAILS

At a high level, CryptoJails intercept filesystem functions, redirecting file and directory modification requests to *encrypted filesystem objects* (EFOs) stored in the running application's lightweight application container (LAC). An EFO is instantiated for each file or directory written to by the application. Future (read or write) requests to those files or directories are transparently intercepted and redirected towards the appropriate EFOs. To minimize storage overhead and simplify system administration, CryptoJails utilize a copy-on-write strategy. That is, requests to read files not stored in EFOs (e.g., libraries, system configuration files, etc.) are not modified and are handled by standard filesystem call implementations.

In our current prototype implementation, CryptoJails operate by overriding filesystem library functions and system call wrappers using Linux's LD_PRELOAD mechanism.[1] To use a CryptoJail for an application, the user simply invokes

```
cryptojail application-prog <application-args...>
```

For example, Firefox may be enclosed within a CryptoJail (keeping settings and cached webpages private) by executing `cryptojail firefox &`.

The (unmodified) application is unaware of CryptoJail and accesses the filesystem through standard APIs. However, CryptoJail changes the semantics of such filesystem functionality to redirect write requests to EFOs and read requests to EFOs if such EFOs exist.

2.1 Keying

When an application is invoked via the `CryptoJail` command, CryptoJail prompts the user for a password. The password is used to derive two cryptographic keys: (i) a symmetric *encryption key* for encrypting and decrypting content and metadata associated with EFOs and (ii) an *integrity key* used to verify that EFOs have not been modified by unauthorized processes.[2] While the application is executing, CryptoJail caches both keys in memory. Since an application's memory may be accessed through `/proc/<pid>/mem` by a process that attempts to PTRACE it, CryptoJail immediately aborts the application if it detects an attempted PTRACE.

2.2 Lightweight Application Containers (LACs)

An application's LAC consists of a number of EFOs as well as a *lookup table* that indexes path names (i.e., files and directories requested by the application) to their corresponding EFOs. To prevent unauthorized processes from discerning the number of EFOs or their relative sizes, the lookup table and the EFOs are enveloped within a file (i.e., the container) that resides on the standard filesystem. To ensure integrity and authenticity, each EFO contains an HMAC (keyed using the integrity key) over its metadata and content. Data privacy is achieved by encrypting the entire LAC with the encryption key.

2.3 Detecting Modified Executables

Applications have unencumbered (and transparent) access to the contents of their LACs. A malicious process may attempt to reveal or alter the content of another application's LAC by replacing the latter's executable and hence duping the user into entering a CryptoJail password for the wrong application. To mitigate such an attack, CryptoJail conducts simple integrity checks to verify that the CryptoJail-protected application has not been altered. When a CryptoJail is first associated with an application, a cryptographic hash of the executable is stored within the LAC. On subsequent invocation, CryptoJail causes the program to abort if its hash does not match the hash stored in the LAC. If the executable changes (i.e., for program updates), the user must run a utility application, `cryptojail-update`, to update the hash stored in the LAC.

3. COMPARISON TO OTHER APPROACHES

The Janus system was the first to propose that untrusted software be executed in a restricted environment [3]. Janus leverages Solaris' process tracing capabilities to intercept filesystem and network accesses, and accept or reject such requests according to user-specified policies. CryptoJail provides some isolation by recording file and directory modifications only in a private container. However, unlike Janus, CryptoJail is not intended to sandbox untrusted applications. In contrast, CryptoJail protects the privacy of data belonging to trustworthy applications.

SELinux [6] security extensions provide fine-grained mandatory access control (MAC) over filesystems. As with Janus, SELinux is well-suited for restricting untrusted applications, and is not particularly useful to protect the privacy of application data, particularly given that the access patterns of even well-behaved applications may be difficult to predict *a priori*. Moreover, producing correct SELinux policies can be very complex [4]. Finally, maintaining SELinux requires administrative privileges and patches to the kernel. CryptoJail requires no superuser privileges and operates entirely in userspace.

Cryptographic filesystems [1, 7] encrypt files and directories to protect against unauthorized accesses. CryptoJail utilizes a similar approach to maintain encrypted LACs. Cryptographic filesystems by themselves, however, lack sufficient granularity to protect the privacy of application data. That is, any process running as the user who mounts a cryptographic filesystem can read the filesystem's contents. If an application stores its data in the cryptographic filesystem, then a rogue process running as the same user can also access the information. CryptoJail can be viewed as providing each application with its own cryptographic filesystem. However, CryptoJail's LACs are not mounted in a particular location. Using CryptoJail, *any* file modification is transparently migrated to secure storage, not just those that occur at a particular location in the directory structure.

There have been a myriad of proposals that leverage virtual machines (VMs) to achieve software isolation [5, 8, 2]. Although such approaches are generally aimed at instantiating sandboxes for testing untrusted applications, they have the side effect of providing secure storage for appli-

[1]Consequently, our current LAC implementation cannot protect statically compiled executables. A potential workaround is to detect static compilation and use PTRACE to intercept system calls. Such techniques have been used in previously proposed systems [3], often incurring significant performance overhead.

[2]To support efficient re-keying, the password is used to decipher a key that decrypts encrypted encryption and integrity keys. Changing a CryptoJail password therefore entails re-encrypting the encryption and integrity keys with the key derived from the new password.

cations. Unfortunately, as described above, VMs impose performance penalties, limit access to physical devices, partition system resources such as memory, often involve kernel patches (for performance gains), and create barriers to sharing resources (e.g., libraries). The copy-on-write semantics of lightweight CryptoJails enable private containers for storing sensitive application data while permitting sharing of system resources.

4. CONCLUSION AND FUTURE WORK

CryptoJails protect the privacy of application data by transparently intercepting filesystem modifications, ensuring that data is stored in cryptographically protected containers. Unlike VM-based approaches, CryptoJail is lightweight, requiring no superuser privileges or kernel modifications, and supports copy-on-write semantics.

We are actively developing a prototype implementation of CryptoJail. Our initial performance results indicate that CryptoJails impose little overhead and maximize usability by imposing very little user configuration.

5. ACKNOWLEDGMENTS

This work is partially supported by NSF Grant CNS-0831376. Any opinions, findings, and conclusions or recommendations expressed in this paper are those of the authors and do not necessarily reflect the views of the National Science Foundation.

6. REFERENCES

[1] M. Blaze. A Cryptographic File System for UNIX. In *1st ACM Conference on Computer and Communications Security (CCS)*, pages 9–16, 1993.

[2] K. Borders, E. V. Weele, B. Lau, and A. Prakash. Protecting Confidential Data on Personal Computers with Storage Capsules. In *18th USENIX Security Symposium*, August 2009.

[3] I. Goldberg, D. Wagner, R. Thomas, and E. A. Brewer. A Secure Environment for Untrusted Helper Applications. In *Sixth USENIX Security Symposium*, July 1996.

[4] T. Jaeger, R. Sailer, and X. Zhang. Analyzing integrity protection in the SELinux example policy. In *SSYM'03: Proceedings of the 12th Conference on USENIX Security Symposium*, 2003.

[5] Z. Liang, V. N. Venkatakrishnan, and R. Sekar. Isolated program execution: An application transparent approach for executing untrusted programs. In *ACSAC '03: Proceedings of the 19th Annual Computer Security Applications Conference*, page 182, Washington, DC, USA, 2003. IEEE Computer Society.

[6] National Security Agency (NSA). Security-Enhanced Linux (SELinux). `http://www.nsa.gov/research/selinux`.

[7] TrueCrypt Foundation. TrueCrypt: Free Open-Source On-The-Fly Disk Encryption Software for Windows Vista/XP, Mac OS X and Linux. `http://www.truecrypt.org/`.

[8] C. Weinhold and H. Härtig. VPFS: building a virtual private file system with a small trusted computing base. In *Eurosys '08: Proceedings of the 3rd ACM SIGOPS/EuroSys European Conference on Computer Systems 2008*, pages 81–93, 2008.