POSTER: Hidden in Plain Sight: A Filesystem for Data Integrity and Confidentiality

Anne Kohlbrenner*
Carnegie Mellon University
akohlbre@andrew.cmu.edu

Frederico Araujo Teryl Taylor Marc Ph. Stoecklin
IBM T.J. Watson Research Center
{frederico.araujo, teryllt}@ibm.com, mpstoeck@us.ibm.com

ABSTRACT

A filesystem capable of curtailing data theft and ensuring file integrity protection through deception is introduced and evaluated. The deceptive filesystem transparently creates multiple levels of stacking to protect the base filesystem and monitor file accesses, hide and redact sensitive files with baits, and inject decoys onto fake system views purveyed to untrusted subjects, all while maintaining a pristine state to legitimate processes. Our prototype implementation leverages a kernel hot-patch to seamlessly integrate the new filesystem module into live and existing environments.

We demonstrate the utility of our approach with a use case on the nefarious Erebus ransomware. We also show that the filesystem adds no I/O overhead for legitimate users.

KEYWORDS

Intrusion Detection and Prevention; Cyber Deception; Filesystems

1 INTRODUCTION

In today's modern digital age, the compromise or theft of data can have severe consequences on individuals, governments, enterprises, and cloud environments. Capitalizing on data as the new digital currency, cybercrime has become a big money business, with criminals stealing millions of credit card numbers [9] and holding data ransom, costing businesses millions of dollars to regain access to their data [5]. Given the alarming rate and scope of recent attacks, new approaches are needed to effectively identify and dissuade attackers trying to steal or destroy their targets' crown jewels.

Existing approaches to prevent data theft only work under special circumstances. For example, current ransomware protections focus on preventing malware from running, maintaining backups, or trying to reverse engineer custom cryptography schemes. Unfortunately, such reactive approaches have been proven inadequate, as 71% of companies attacked by ransomware still have their files successfully encrypted [7], with less than half being able to recover from backups [8].

Other protective measures, such as deceptive files [2, 4] and canaries [1], alert defenders of an attacker's presence by leaving deceptive breadcrumbs among the legitimate files on the filesystem, which trigger a beacon when accessed. However, to avoid confusing legitimate users, the users must either be aware of the decoys, which is difficult to maintain in shared systems, or the decoys must be identifiable by users while being indiscernible to attackers—a non-trivial property to achieve in practice. Unfortunately, such deceptive files do not prevent the attacker from stealing sensitive data.

To overcome these disadvantages, our work introduces a new filesystem, DcyFS, which protects files at their place of rest. DcyFS seeks three main objectives: (1) stop the theft, modification, and destruction of important data by untrusted subjects (e.g., applications, users), (2) deceive, misdirect, and disinform adversaries, and (3) construct a new sensor able to detect the presence of attackers on production systems.

Contributions. DcyFS takes a fundamentally different approach to the data theft and integrity problem with a new filesystem that can monitor file accesses transparently, hide sensitive data, create decoy files, and modify existing files to provide a fake system view to untrusted subjects (e.g., processes and users). DcyFS actively captures filesystem events and correlates them with other system features (e.g., user, process name, time) to create targeted filesystem *views* that hide high-value assets and expose enticing breadcrumbs to detect deliberate tampering with filesystem data. Such *context-awareness* minimizes false alarms by curtailing inadvertent, legitimate access to breadcrumbs—by exposing more "truthful" views of the filesystem to trustworthy processes—while maximizing chances of attack detection by strategically overlaying deceptive objects atop the base filesystem.

Specifically, our contributions can be summarized as follows:

- The design of a new *stacking* filesystem to augment standard filesystems with denial and deception capabilities, such as hiding resources from untrusted processes, redacting or replacing assets to protect sensitive data, and injecting breadcrumbs to disinform and misdirect attackers.
- The creation of a Virtual File System (VFS) module to enable transparent integration with legacy environments. A tiny kernel hot-patch interposing `exec` allows DcyFS to be installed without system restart.
- An evaluation showing that our approach can detect and resist real ransomware attacks, and preliminary benchmarks demonstrate that DcyFS can defend against data theft and filesystem tampering without incurring significant overhead.
- An approach that enforces file integrity protection without requiring file access mediation. It also supports the implementation of access control policies, and enables the automation of decoy injection in commodity filesystems.

2 SYSTEM OVERVIEW

Figure 1 presents an architectural overview of DcyFS. The core components of the system are a set of filesystem overlays that are deployed on a per-process basis, providing each process with a different *view* of the filesystem—computed as the *union* of the

*Work completed while the author was at IBM T.J. Watson Research Center.

ACM ISBN 978-1-4503-4946-8/17/10.
https://doi.org/10.1145/3133956.3138841

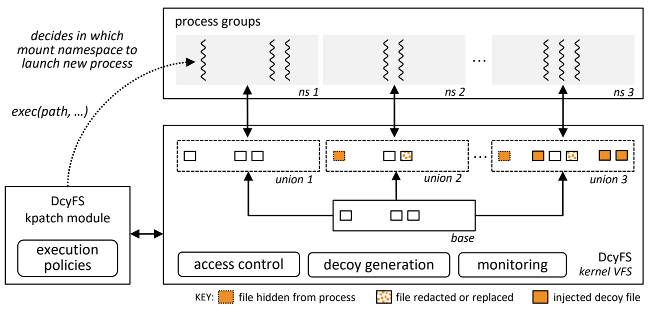

Figure 1: Architectural Overview of DcyFS

base filesystem and the *overlay*. To alter the resulting union, each overlay has the ability to (1) hide base files, (2) modify their content by overlaying a different file with the same name, and (3) inject new files that are not present in the host system. File writes are stored in the overlay protecting base files from being overwritten. This forms the basis of a stackable filesystem that can be mounted atop different base filesystem types (e.g., block, network) to offer data integrity protection and enhanced detection against data-stealing attacks.

To effectively and securely construct filesystem overlays, DcyFS leverages the kernel's mount namespace capability to swap the root filesystem to a specially-crafted union mount. The mount namespace is an operating system construct that provides an isolated mount point list for every process residing in a particular namespace—a process inside the namespace observes a different filesystem than the base system. Processes are moved, upon creation, into a mount namespace based on some notion of trust.

For our initial prototype, we built a simple trust model based on white/black listing. The model maps a user name, binary hash, or process name to a set of configurations describing an overlay. The configuration can specify which files and directories to show in the overlay, which ones to hide, and which ones to replace with another file. Trusted processes are presented with a *pristine* (unaltered) view of the filesystem.

2.1 Design & Implementation

To achieve transparency and minimize performance overhead, our DcyFS requires a small modification to the kernel—deployed as a kernel hot patch (patching the kernel while it is running)—and the installation of a kernel module implementing monitoring, access control, and decoy creation and injection capabilities. The hot patch modifies the kernel's `exec` family of functions to drop newly created processes into a new mount namespace protected by the union filesystem. The particular overlay is chosen based on a pre-defined trust model. Note that child processes automatically inherit their parent namespace, unless otherwise specified by the trust model.

We implemented DcyFS using Linux's OverlayFS union filesystem, which creates an upper mount and a lower mount, where the lower mount is the base file system, and the upper mount is the overlay. Figure 2 illustrates this concept, showing the base and overlay mounts, and the resulting union of the two mounts that serves as the namespace's pivot. To hide a base file or directory, DcyFS simply marks it as deleted in the overlay. Decoy files are similarly placed in carefully-chosen locations inside the upper mount, and existing files can be replaced or redacted for attacker deception [3].

Our initial prototype was developed for Ubuntu 16.04 LTS, leveraging VFS and its mature mount namespace implementation. Recently, Windows Server 2016 was released with native namespace support and an overlay filesystem driver, mirroring its open-source counterpart. We plan to port DcyFS to Windows as future work.

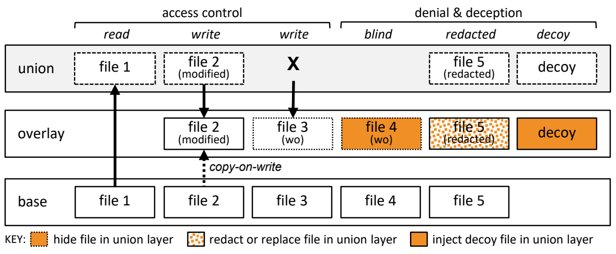

Figure 2: DcyFS**'s access control, denial, and deception**

2.2 Filesystem Denial & Deception

Changes made by untrusted processes do not affect the base filesystem, protecting legitimate users from seeing malicious changes as well as effectively keeping a uncorrupted copy of the filesystem immediately before the malicious process started. DcyFS can hide particular files and directories from the process, thus curtailing sensitive data leaks. Additionally, the filesystem module is capable of generating encrypted files and implanting decoys in the overlay to replace sensitive files in the base filesystem. DcyFS transparently monitors and logs access to such files. Moreover, only the untrusted process is affected by the hidden and decoy files, leaving legitimate users free of confusion.

3 EVALUATION

3.1 Experimental Setup

Since processes are moved to an overlay upon startup, DcyFS can potentially affect process startup time, specially for untrusted processes, where a crafted overlay is purveyed in lieu of the unaltered (empty) version presented to trusted processes. To test the overhead of the overlay creation, we built a program whose entry point simply exited, and timed its execution three different scenarios: a baseline evaluation without DcyFS, a whitelisted execution and a blacklisted execution, both with DcyFS. For the untrusted execution, an empty overlay was created. We used `perf stat -r 100000` to measure the execution time, running on a virtual machine with Ubuntu 16.04 LTS.

Table 1 shows the results, which demonstrate that DcyFS does not affect the startup time of trusted processes, and incurs an overhead of 1.66 ms for trusted processes. Figure 3 shows that untrusted processes startup time is quickly dominated by overlay creation and injection of decoys. However, both sources of startup overhead can be significantly reduced by simply creating overlays with decoy files ahead of time, reducing startup costs to moving the new program into an overlay. This is left as future work.

To assess the impact of the union filesystem on file I/O throughput, we used the Flexible Filesystem Benchmark and measured R/W throughput on the same machine, with and without DcyFS. For programs running in an overlay, the R/W throughput dropped from 634/641 MB/s to 456 MB/s for both read and write. This is due to the copy-on-write strategy implemented by the union filesystem. We plan to investigate techniques to reduce this overhead.

Table 1: Process startup time benchmark.

Benchmark	Startup time
Without DcyFS (baseline)	0.380 ms
Trusted process	0.378 ms
Untrusted process	2.046 ms

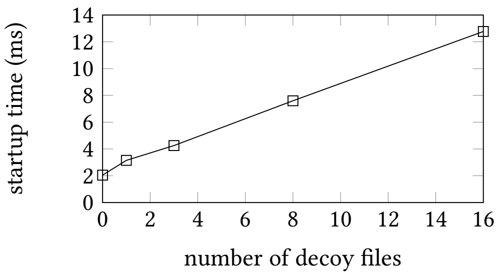

Figure 3: Process startup time vs. number of decoy files.

3.2 Use Cases

Data Theft Protection and Deception. A typical social engineering attack usually starts with a spear phishing email containing a malicious attachment. When the recipient (e.g., a governament official whose machine is protected with DcyFS) clicks on the attachment, a remote access terminal (RAT) is transparently installed. Since the RAT is untrusted, it is immediately given its own view of the filesystem, where sensitive files are hidden, or masked by fake versions. For example, the attacker gets to see a fake or redacted spreadsheet, while the original is safe from the attacker's view.

Ransomware Detection. Since DcyFS prevents processes from making changes to the host filesystem (integrity protection), it is an effective defense against ransomware. To evaluate DcyFS's ability to prevent damage from ransomware, we used the Erebus family of ransomware. Erebus has Windows and Linux variants that hit companies in South Korea. At least one company has paid $1.62 million in ransom [5]. To test the effectiveness of DcyFS against Erebus, we ran the ransomware under two conditions: (1) in a situation where the parent process was untrusted, such as a file browser or instance of a shell, and (2) when Erebus was run from a trusted parent, but Erebus itself was untrusted. In the first case, the files that Erebus encrypted were visible from the parent process only. The rest of the system was unaffected. In the second case, the damage from Erebus was not seen at all on the system, although Erebus could be seen to run by monitoring the internals of DcyFS.

While this test was a success in preventing Erebus from damaging the system, it could also be used to detect that a program was malware. Ransomware often tries to hide its presence until it has finished its encryption. For example, Erebus initially sleeps for five minutes before starting to encrypt files, presumably to reduce the chance that the user will suspect which executable caused the problem. Once the user notices that their files have been encrypted, it is difficult to trace back and find out which program was malicious; however, since DcyFS runs each untrusted program in a separate overlay, we can examine which overlay has encrypted files in it to know which executable is to blame.

4 RELATED WORK

In recent years, there has been a few filesystem solutions specifically targeted towards defense against ransomware. Paybreak allows ransomware to encrypt the files on a system, but stores the crypto keys by hooking Windows Crypto API, so that it can reverse the encryption. While working well against ransomware, it cannot defend against malware that deletes or corrupts data. ShieldFS [6] is a copy-on-write filesystem, which enforces that all processes must write to an overlay as protection, until a detector determines that a process is not ransomware based on file I/O stats. Redemption [10] is similar to ShieldFS except that it uses more features to detect ransomware behavior, including the entropy of data blocks, number of file writes, number of directory traversals, and number of file conversions. While these approaches also provide integrity guarantees against ransomware, they are unable to deal with other types of malware, nor do they deal with data confidentially and deception, which is handled seamlessly by DcyFS.

5 DISCUSSION AND FUTURE WORK

DcyFS enables the construction of realistic, but completely false, views of the filesystem to untrusted processes. To a process running in an overlay, it appears that it is able to view, extract, and modify real data. However, it may be viewing decoy files or missing sensitive files, and its file modifications will not be seen outside its overlay. This is done transparently, without advertising itself to the untrusted process, and without affecting other legitimate processes. Next, to make decoy files both less visible to trusted users and more visible to attackers, DcyFS actively moves decoys into place for untrusted programs. This means that decoys can be stored out of the way of trusted users (e.g., in a hidden directory), as well as being visible in normal locations for untrusted programs.

Currently, our trust model is rule-based using basic white/black listing. In the future, we plan to expand this model to take into account which user is executing the program, how that user is authenticated, and past behaviors of the user and the process to determine its trustworthiness.

The changes made by untrusted processes are currently only visible to that process and disappear on reboot. In situations where an untrusted process should become trusted, such as being vouched for by a more trusted subject, those changes could be copied from the overlay and merged into the real filesystem. The prototype supports decoy files that are created manually; however, this is unnecessary. DcyFS could automatically create decoy files based on different formats, such as data that appears to be encrypted, or files containing fake keys or passwords. The system should also learn the content of overlays based on past process behaviors to streamline overlay generation.

REFERENCES

[1] Thinkst Canary . 2017. Canarytokens. (2017).
[2] Jim Yuill annd Mike Zappe, Dorothy Denning, and Fred Feer. 2004. Honeyfiles: deceptive files for intrusion detection. In *Proc. Annual IEEE SMC Info. Ass. Work.*
[3] Frederico Araujo and Kevin W Hamlen. 2015. Compiler-instrumented, Dynamic Secret-Redaction of Legacy Processes for Attacker Deception. In *Usenix Sec. Sym.*
[4] Brian M. Bowen, Shlomo Hershkop, Angelos D. Keromytis, and Salvatore J. Stolfo. 2009. Baiting Inside Attackers Using Decoy Documents. In *Proc. Int. ICST Conf. Security and Privacy in Communication Networks.*
[5] Ziv Chang, Gilbert Sison, and Jeanne Jocson. 2017. Erebus Resurfaces as Linux Ransomware. (2017).
[6] Andrea Continella, Alessandro Guagnelli, Giovanni Zingaro, Giulio De Pasquale, Alessandro Barenghi, Stefano Zanero, and Federico Maggi. 2016. ShieldFS: A Self-healing, Ransomware-aware Filesystem. In *Proc. Annual Computer Security Applications Conf.*
[7] Jonathan Crowe. 2017. 2017 Ransomware Trends and Forecasts. (2017).
[8] Brianna Gammons. 2017. 4 Surprising Backup Failure Statistics that Justify Additional Protection. (2017).
[9] Kevin Granville. 2015. 9 Recent Cyberattacks Against Big Businesses. (2015).
[10] Amin Kharraz and Engin Kirda. 2017. Redemption: Real-time Protection Against Ransomware at End-Hosts. In *Proc. Sym. Res. in Attacks, Intrusions and Defenses.*