DNSCHK: [TODO: Needs After Colon Title]

Anonymous Author(s)

Abstract

Downloading files from the internet comes with considerable risk due to the general inability of the end user to check the integrity of the file they acquired. An adversary could tamper with the file en route to its destination or even compromise the server that hosts the file, as was the case with the 2016 breach of the Avast distribution servers that provided downloads of the popular software suite CCleaner. The traditional strategy for addressing this risk is with the use of ”checksums” or signatures generated by cryptographic hashing functions to verify a files integrity. This process is problematic for a whole host of reasons, especially the fact that the clear majority of end users cannot be bothered to manually calculate a checksum, let alone find the corresponding correct checksum via some web resource, compare the two, and then be expected to ”do the right thing”.

With this research, we explore DNS-based methods of authenticating files downloaded over the internet that are completely transparent to end users yet dead simple for service administrators and system operators to comply with and integrate. [TODO: Two? insights] Leveraging these insights, we propose DNSCHK, a [TODO: description] . We implement DNSCHK on a DNSCHKwhere? and evaluate its efficacy when integrated into multiple popular applications. We find that DNSCHK [TODO: what do we get?] .

1 Introduction

Downloading resources over the internet is a remarkably simple and painless process for application developers and end users alike. The user (via their browser) requests a server resource at some URL. The server responds with the resource. The browser completes downloading the resource. Unfortunately, downloading content over the internet can be risky.

[TODO: (more)]

2 Specification

DNSCHK acts as a translation layer sitting between the drive and the operating system. It provides confidentiality and integrity guarantees while minimizing performance loss due to metadata management overhead. DNSCHK accomplishes this by leveraging the speed of stream ciphers over the AES block cipher and taking advantage of the append-mostly nature of Log-structured Filesystems (LFS) and modern Flash Translation Layers (FTL) [1].

3 DNSCHK Implementations

[TODO: todo]

4 Evaluation

4.1 Experimental Setup

[TODO: (todo)]

4.2 Experimental Methodology

[TODO: (todo)]

5 Related Work

[TODO: (adapt related work collected from wiki)] [TODO: (cite some usenix/oakland papers)]

6 Conclusion

[TODO: (summarize intro)]

References

[1] M. Cornwell. “Anatomy of a Solid-state Drive”. In: *Queue* 10.10 (Oct. 2012), 30:30–30:36. ISSN: 1542-7730. DOI: 10.1145/2381996.2385276. URL: http://doi.acm.org/10.1145/2381996.2385276.