DNSCHK: Free Malicious Download Detection with Highly-Available Distributed Systems

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: (the rest; an expository summarization of research)]

[TODO: (discuss "free": no interface changes, no addition to resource download time, no additional burden on the end user (qualified statement))]

In summary, our primary contributions are:

[TODO: (contribution summary; bulletpoints)]

2 Background

[TODO: (lay out the problem more formally, more in depth, and with precise language)]

2.1 Current Detection and Prevention Methods

Anti-Malware Software

[TODO: (what it is, why it fails)] **[TODO: (also talk about manual scanning of files for viruses)]**

HTTPS / Encrypted Channel

[TODO: (what it is, why it fails)]

Browser-based Heuristics and Blacklists

[TODO: (what it is, why it fails)]

Checksums

[TODO: (what it is, why it fails)]

Public Key Infrastructure

[TODO: (what it is, why it fails)]

2.2 Motivation: Case Studies

[TODO: (there are so many; enumerate them)]

3 The DNSCHK Approach

[TODO: (describe generalized system using any sort of distributed highly-available key-value store that exists or could exist (like dnschk))]

3.1 Assumptions

[TODO: (list assumptions)]

3.2 Threat Model

[TODO: (enumerate threat model)]

4 Implementation

4.1 DNSCHK: Google Chrome Extension

[TODO: (describe implementation details; works with DNS or DHT and is published to Chrome store; no interface changes!–i.e. downloads work exactly the same with or without the extension; users still have to confirm/deny suspicious judgements, but they're rare occurrences)]

[TODO: (should we provide a link to/description of the hotcrp demo?)]

4.2 Expanded Threat Model

[TODO: (enumerate additional threats)]

4.3 DNSCHK: FileZilla (FTP) Patch

[TODO: (describe implementation details; minor interface change if the download is judged unsafe or suspicious, requires user to confirm/deny download)]

4.4 Expanded Threat Model

[TODO: (enumerate additional threats)]

5 Evaluation

The primary goal of any DNSCHK implementation is to alert end-users when the resource they've downloaded is something other than what they were expecting. We tested the effectiveness of our approach using the DNSCHK extension for Google Chrome, a real-world deployment of HotCRP, and a random sampling of papers published in previous Usenix proceedings.

[TODO: (other text)]

5.1 Real-world Resource Corruption Detection

[TODO: (describe the system, the alterations, and the test cases)]

5.2 Overhead

[TODO: (Additional Download Latency, Additional Network Load, Runtime overhead)]

6 Discussion

6.1 Related Work

[TODO: (adapt wiki entry)]
[TODO: (cite some usenix/oakland papers)]

6.2 Limitations

[TODO: (bulletpoints)]

6.3 Deployment

[TODO: (discuss envisioned deployment strategies for resource providers)]

6.4 Future Work

[TODO: (adapt wiki entry)]

7 Conclusion

[TODO: (summarize intro, contributions, evaluation, and discussion tidbits)]