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

Anonymous Author(s)

Abstract

Downloading resources over the internet comes with considerable risk due to the general inability of the end user to verify the integrity of the resource they've received. An adversary could tamper with said resource in a variety of ways: a) by compromising it en route to its destination, b) by executing a successful supply chain or similar attack beforehand, or c) by compromising the server that physically hosts the resource, as was the case with the 2016 breach of the HTTPS-protected Avast distribution servers that provided downloads of the popular software suite CCleaner. The de facto standard for addressing this risk is with the use of *checksums* coupled with some secure transport medium like TLS/HTTPS. Checksums are fingerprints generated by cryptographic hashing functions early in the build process, are supposedly hosted on a separate system than a resource, and are used to verify that resources integrity to an end user. Checksums are problematic for a whole host of reasons, the foremost being the fact that the clear majority of end users will not be burdened with manually calculating the checksum of a resource they've received. Even if they do, said user must search for the corresponding "correct" checksum to verify their calculation. If they recognize the checksums are different, the user is then expected to "do the right thing" in context, whatever that happens to be.

With this research, we explore a novel method of verifying the integrity of resources downloaded over the internet with two key concerns: a) it is automatic, *i.e.,* no additional interaction is required from the end user during standard usage and b) configuring the validation method is simple for service administrators and system operators to integrate and deploy. Hence, we propose DNSCHK, a novel automated resource validation method that is transparent to end users and simple for administrators to deploy. We implement DNSCHK as a proof-of-concept Google Chrome extension as well as a patch to the FileZilla FTP client. We evaluate the security, scalability, and performance of the DNSCHK approach and provide a publicly accessible demonstration of its utility via a patched HotCRP instance.

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)]

Discuss "free," *i.e.,* no interface changes, no addition to resource download time, no additional burden on the end user (qualified statement).

[1]

In summary, our primary contributions are:

- We propose a novel practical defense against receiving malicious, corrupted, or compromised resources over the internet. Contrasted with current solutions, our defense requires no UI or API interface changes at any level, does not employ unreliable heuristics, does not interfere with other software or extensions that also handle resource downloads, and can be transparently deployed without adding to the *fragility* of the system. It protects end users whose software implements DNSCHK while remaining unnoticeable to users of whose software does not. Hence, DNSCHK could be deployed immediately.
- We present our prototype DNSCHK implementations for Google Chrome and FileZilla and demonstrate its effectiveness in mitigating the consumption of compromised resources, even when transmitted over a secure channel. To the best of our knowledge, this is the *first* system providing such capabilities with little implementation cost and at no cost to the end user. Further, we release the DNSCHK solution to the community as open source software [1].
- We carefully and extensively evaluate the security, scalability, and performance of our automated defense against resource corruption to demonstrate the effectiveness and high practicality of the DNSCHK approach. Specifically, we find no obstacles to efficient scalability given choice of distributed system and no performance overhead compared to downloads without DNSCHK. We further provide a publicly accessible demonstration of DNSCHK's utility via a patched HotCRP instance [2].

2 Background

In this section we ...

[1] The Chrome extension is available at https://tinyurl.com/dnschk-actual

[2] The patched HotCRP instance is available at https://tinyurl.com/dnschk-hotcrp

[TODO: Use the language of IETF RFC3552 to describe active attack]

2.1 Current Detection and Prevention Solutions

There are several. Blah blah.

2.1.1 Anti-Malware Software.

(what it is, why it fails; also talk about manual scanning of files for viruses)

2.1.2 HTTPS / Encrypted Channel.

[6, 11, 13, 24, 25]

2.1.3 Browser-based Heuristics and Blacklists.

(what it is, why it fails)

2.1.4 Checksums.

(what it is, why it fails)

2.1.5 Public Key Infrastructure.

[7, 12, 18, 30]

2.2 Motivation: Case Studies.

Blurb about case studies.
Case 1: x. Explain
Case 2: x. Explain
Case 3: x. Explain
Case 4: x. Explain

2.3 "Free" Highly-Available Distributed Systems.

Others are considering this as well, such as securitytxt draft [14]. A widely deployed example is DKIM [8].

3 The DNSCHK Approach

In this section we ...

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

Go over algorithms!

4 Implementations

In this section we ...

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)]

Reference hotcrp demo but leave the description for the evaluation.

Suggest chrome download API improvement ability to mark downloads dangerous.

4.2 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)]

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.

5.1 Threat Model and Assumptions

Include threat model for Chrome extension and FTP patch. Note the additional Chrome vulnerability solved with a timer and the fact that we're assuming Chrome is giving us an accurate referrer URL on its `DownloadItems`.

5.2 Real-World Resource Corruption Detection with Google Chrome and HotCRP

It also seems from ACME that HTTP challenges are good enough of a proof to issue TLS certificates, so why not good enough for checksums? Threat model of ACME thoroughly goes through this [4].

5.3 Deployment and Scalability

Discuss envisioned deployment strategies for resource providers.

Can this be scaled? Yes it can. What are the practical limits? EDNS0 means it ain't DNS size, though packet fragmentation is still a concern. How about max record length? Maximum number of records? A service could have thousands or millions of files it serves! Can DNS handle that? DHT failover is still a solution anyway.

5.4 Performance Overhead

Additional Download Latency, Additional Network Load, Runtime overhead, etc. All nixed.

6 Discussion

In this section we ...

6.1 Additional Related Work

Cryptographic Data in DNS Resource Records. Storing metadata in the DNS network is not a new idea. The DNS-Based Authentication of Named Entities (DANE) specification [12, 18, 30] defines the "TLSA" and "OPENPGPKEY" DNS resource records to store cryptographic data. These resource record types, along with "CERT" [20], "IPSECKEY" [26], those defined by DNS Security Extensions (DNSSEC) [1], and others demonstrate that storing

useful cryptographic data retrievable through the DNS network is feasible at scale [18, 30]. With DNSCHK, however, we use "TXT" records to map resource identifiers to authoritative hashes. In accordance with RFC 5507 [23], an actual DNSCHK implementation would necessitate the creation of a new DNS resource record type.

PGP/OpenPGP. Despite PGP sufficiently addressing its threat model, it is notoriously hard for end users to use correctly, if they can manage it at all [29]. The same human factors that make the otherwise cryptographically solid PGP encryption solution so unpleasant and arduous for end users also exist in the context of download integrity verification and checksums. Users cannot and *will not* be burdened with manually verifying a checksum, and to expect otherwise has proven dangerous (see: Section 2).

More. (todo)

6.2 Limitations

6.2.1 DNSSEC Adoption is Slow

DNSSEC is hard to use and harder to use correctly. It does not make the DNS network, i.e. properly configured DNS servers protected with DNSSEC, any more vulnerable to reflection [22] and amplification [3, 9, 16, 17, 31] attacks than it already is as a UDP-based content service [27, 28] but does arguably make services significantly more fragile because DNSSEC is hard to get right.

The metrics of signed DNSSEC zones are not easy to come by for the entire Internet [2]. DNSSEC adoption across is small and slow. Worldwide, less than 14 percent of DNS requests have DNSSEC validated by the resolver [2] but thanks to community initiatives is on the rise [19] (use graph as figure from bit.ly/2zSR7A6).

From SO: DNSSEC does have risks! It's hard to use, and harder to use correctly. Often it requires a new work flow for zone data changes, registrar management, installation of new server instances. All of that has to be tested and documented, and whenever something breaks that's related to DNS, the DNSSEC technology must be investigated as a possible cause. And the end result if you do everything right will be that, as a zone signer, your own online content and systems will be more fragile to your customers. As a far-end server operator, the result will be, that everyone else's content and systems will be more fragile to you. These risks are often seen to outweigh the benefits, since the only benefit is to protect DNS data from in-flight modification or substitution. That attack is so rare as to not be worth all this effort. We all hope DNSSEC becomes ubiquitous some day, because of the new applications it will enable. But the truth is that today, DNSSEC is all cost, no benefit, and with high risks.

The overwhelming majority of domain name zone administrators appear to be just not aware of DNSSEC, or, even if they want to sign their zone, they cannot publish a signed zone because of limitations in the service provided by the registrar, or if they are aware and could sign their zone, then they dont appear to judge that the perceived benefit of DNSSEC-signing their zone adequately offsets the cost of maintaining the signed zone.

6.2.2 DNS-Specific Protocol Limitations

DNS [21] was not originally designed to transport or store relatively large amounts of data, though this has been addressed with EDNS0 [10]. The fingerprints stored in DNS shouldn't be much longer than 128 bytes or the output of the SHA512 function. Regardless, DNS resource record extensions exist that store much more than 128 bytes of data [18, 20, 26, 30].

Others are considering this as well, such as securitytxt draft [14]. A widely deployed example is DKIM [8]. Whether we created our own DNS resource record type or settled for a well-known TXT record, there are other DNS records that could be far larger. Resource records like CERT, and OPENPGPKEY TLSA with a complete certificate [18] all may hold several kilobytes of data

DNS data can handle a lot more then the amount of data needed here. And amplification protection can be done simply by using TCP or insisting on DNS-COOKIES for these answers. This is a generic DNS problem is solved by DNS people and there is no need for additional requirements here.

6.2.3 Chrome Implementation

Our current JavaScript proof-of-concept implementation, as a Chrome extension, isn't allowed to touch the resource file downloaded by Chrome and so can't prevent the potentially-malicious resource file from being executed by the end usera feature Chrome/Chromium reserves for its own internal use. The Chrome *app* API [15] might have been of assistance as it allowed for some limited filesystem traversal via a now deprecated native app API; there is also a non-standard HTML5/WebExtensions API that would provide similar functionality were it to be widely considered [5].

DNSCHK would be even more effective as a browser extension if Chrome/Chromium and the WebExtensions API allowed for an explicit `onComplete` event hook in the downloads API. This hook would fire immediately before a file download completed and the file became executable, *i.e.,* had its `.crdownload` or `.download` extension removed. The hook would consume a `Promise`/`AsyncFunction` that kept the download in its non-complete state until said `Promise` completed. This would allow DNSCHK's background page to do something like alter the download's `DangerType` property and alert the end user to the dangerous download naturally. This would have the advantage of communicating intent through the browser's familiar UI while preventing the potentially-malicious download from becoming immediately executable. Unfortunately, the closest the Chrome/WebExtensions API comes to allowing `DangerType` mutations is the `acceptDanger` method on the downloads API, but it is not suitable for use with DNSCHK as it is a background page based extension.

6.3 Future Work

(adapt wiki entry)

7 Conclusion

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

References

[1] D. E. E. 3rd. *Domain Name System Security Extensions*. RFC 2535. http://www.rfc-editor.org/rfc/rfc2535.txt. RFC Editor, 1999. URL: http://www.rfc-editor.org/rfc/rfc2535.txt.

[2] APNIC. 2018. URL: https://stats.labs.apnic.net/dnssec/XA?c=XA&x=1&g=1&r=1&w=7&g=0.

[3] S. Ariyapperuma and C. J. Mitchell. "Security vulnerabilities in DNS and DNSSEC". In: *The Second International Conference on Availability, Reliability and Security (ARES'07)* (2007). DOI: 10.1109/ares.2007.139.

[4] R. Barnes, J. Hoffman-Andrews, D. McCarney, and J. Kasten. *Automatic Certificate Management Environment (ACME)*. Internet-Draft draft-ietf-acme-acme-16. Work in Progress. Internet Engineering Task Force, 2018. 90 pp. URL: https://datatracker.ietf.org/doc/html/draft-ietf-acme-acme-16.

[5] A. Barstow and E. U. *Re: [fileapi-directories-and-system/filewriter]*. 2014. URL: http://lists.w3.org/Archives/Public/public-webapps/2014AprJun/0012.html.

[6] S. Blake-Wilson, M. Nystrom, D. Hopwood, J. Mikkelsen, and T. Wright. *Transport Layer Security (TLS) Extensions*. RFC 3546. RFC Editor, 2003.

[7] J. Callas, L. Donnerhacke, H. Finney, D. Shaw, and R. Thayer. *OpenPGP Message Format*. RFC 4880. http://www.rfc-editor.org/rfc/rfc4880.txt. RFC Editor, 2007. URL: http://www.rfc-editor.org/rfc/rfc4880.txt.

[8] D. Crocker, P. Hallam-Baker, and T. Hansen. *DomainKeys Identified Mail (DKIM) Service Overview*. RFC 5585. 2009. DOI: 10.17487/RFC5585. URL: https://rfc-editor.org/rfc/rfc5585.txt.

[9] T. Dai, H. Shulman, and M. Waidner. "Dnssec misconfigurations in popular domains". In: *International Conference on Cryptology and Network Security*. Springer. 2016, pp. 651–660.

[10] J. Damas, M. Graff, and P. Vixie. *Extension Mechanisms for DNS (EDNS(0))*. STD 75. RFC Editor, 2013.

[11] T. Dierks and C. Allen. *The TLS Protocol Version 1.0*. RFC 2246. http://www.rfc-editor.org/rfc/rfc2246.txt. RFC Editor, 1999. URL: http://www.rfc-editor.org/rfc/rfc2246.txt.

[12] V. Dukhovni and W. Hardaker. *The DNS-Based Authentication of Named Entities (DANE) Protocol: Updates and Operational Guidance*. RFC 7671. RFC Editor, 2015.

[13] R. T. Fielding, J. Gettys, J. C. Mogul, H. F. Nielsen, L. Masinter, P. J. Leach, and T. Berners-Lee. *Hypertext Transfer Protocol – HTTP/1.1*. RFC 2616. http://www.rfc-editor.org/rfc/rfc2616.txt. RFC Editor, 1999. URL: http://www.rfc-editor.org/rfc/rfc2616.txt.

[14] E. Foudil and Y. Shafranovich. *A Method for Web Security Policies*. Internet-Draft draft-foudil-securitytxt-04. http://www.ietf.org/internet-drafts/draft-foudil-securitytxt-04.txt. IETF Secretariat, 2018. URL: http://www.ietf.org/internet-drafts/draft-foudil-securitytxt-04.txt.

[15] Google. *Chrome APIs*. URL: https://developer.chrome.com/apps/api_index.

[16] A. Herzberg and H. Shulman. "DNSSEC: Security and availability challenges". In: *Communications and Network Security (CNS), 2013 IEEE Conference on*. IEEE. 2013, pp. 365–366.

[17] A. Herzberg and H. Shulman. "Towards Adoption of DNSSEC: Availability and Security Challenges." In: *IACR Cryptology ePrint Archive* 2013 (2013), p. 254.

[18] P. Hoffman and J. Schlyter. *The DNS-Based Authentication of Named Entities (DANE) Transport Layer Security (TLS) Protocol: TLSA*. RFC 6698. http://www.rfc-editor.org/rfc/rfc6698.txt. RFC Editor, 2012. URL: http://www.rfc-editor.org/rfc/rfc6698.txt.

[19] S. Isasi. *Expanding DNSSEC Adoption*. 2018. URL: https://blog.cloudflare.com/automatically-provision-and-maintain-dnssec/.

[20] S. Josefsson. *Storing Certificates in the Domain Name System (DNS)*. RFC 4398. RFC Editor, 2006.

[21] P. Mockapetris. *Domain names - concepts and facilities*. STD 13. http://www.rfc-editor.org/rfc/rfc1034.txt. RFC Editor, 1987. URL: http://www.rfc-editor.org/rfc/rfc1034.txt.

[22] Neustar. *DNSSEC: How Savvy Attackers Are Using Our Defenses Against Us*. 2016. URL: https://www.telecomnews.co.il/image/users/228328/ftp/my_files/General%20Files/neustar-dnssec-report.pdf.

[23] and P. Faltstrom, R. Austein, and P. Koch. *Design Choices When Expanding the DNS*. RFC 5507. RFC Editor, 2009.

[24] E. Rescorla. *HTTP Over TLS*. RFC 2818. http://www.rfc-editor.org/rfc/rfc2818.txt. RFC Editor, 2000. URL: http://www.rfc-editor.org/rfc/rfc2818.txt.

[25] E. Rescorla and N. Modadugu. *Datagram Transport Layer Security Version 1.2*. RFC 6347. http://www.rfc-editor.org/rfc/rfc6347.txt. RFC Editor, 2012. URL: http://www.rfc-editor.org/rfc/rfc6347.txt.

[26] M. Richardson. *A Method for Storing IPsec Keying Material in DNS*. RFC 4025. RFC Editor, 2005.

[27] US-CERT. *US-CERT: United States Computer Emergency Readiness Team Vulnerability Alert TA14-017A: UDP-Based Amplification Attacks*. 2014. URL: https://www.us-cert.gov/ncas/alerts/TA14-017A.

[28] P. Vixie and V. Schryver. *DNS Response Rate Limiting (DNS RRL)*. Tech. rep. 2012. URL: http://ss.vix.su/~vixie/isc-tn-2012-1.txt.

[29] A. Whitten and J. D. Tygar. "Why Johnny Can't Encrypt: A Usability Evaluation of PGP 5.0". In: *Proceedings of the 8th Conference on USENIX Security Symposium - Volume 8*. SSYM'99. Washington, D.C.:

USENIX Association, 1999, pp. 14–14. URL: http://dl.acm.org/citation.cfm?id=1251421.1251435.

[30] P. Wouters. *DNS-Based Authentication of Named Entities (DANE) Bindings for OpenPGP*. RFC 7929. RFC Editor, 2016.

[31] Y. Yao, L. He, and G. Xiong. "Security and cost analyses of DNSSEC protocol". In: *International Conference on Trustworthy Computing and Services*. Springer. 2012, pp. 429–435.