DNSCHK: Malicious Download Detection with Highly-Available Distributed Systems

Anonymous Author(s)

Abstract

Downloading resources over the internet comes with many risks, including the chance that a malicious actor has replaced the resource you think you are accessing with a compromised version. The current standard for addressing this risk is the use of *checksums* coupled with a secure transport layer: users download a resource and compare its checksum with a posted checksum from the developers to ensure a match. Among the many problems with the current use of checksums are user apathy—for most users, hand-calculating the checksum and comparing to the published version are too tedious—and that it is often trivial for a malicious actor who compromises a resource to also compromise the checksum as they are hosted using the same infrastructure. In this paper we propose DNSCHK, a novel resource validation scheme meant as a complete replacement for current checksum based approaches. DNSCHK automates the tedious parts of verification to eliminate user apathy while leveraging highly-available distributed systems to separate resources from digests, making these systems and their end users more resilient to resource integrity attacks. We carefully evaluate the security, performance, and practicality of DNSCHK through implementing extensions in both a web browser (Chrome) and FTP client (FileZilla); implementations are tested versus common resource integrity violations. We find that DNSCHK is more effective than existing solutions, detects a wide variety of real-world integrity errors across a diverse set of platforms, and is a scalable and immediately deployable solution.

1 Introduction

In 2010, through compromising legitimate applications available on trusted vendor websites, nation-state actors launched the Havex malware, targeting aviation, defense, pharmaceutical, and petrochemical companies in Europe and the United States [24]. In 2012, hackers compromised an official phpMyAdmin download mirror hosted by reputable software provider SourceForge. The backdoored version of the popular database frontend was downloaded by hundreds of users, potentially allowing attackers to gain access to private customer data [37, 38]. In 2016, hackers broke into the Linux Mint distribution server and replaced a legitimate Mint ISO with one containing a backdoor, infecting hundreds of machines with the malware [34, 42]. Over a four day period in 2017, users of the popular HandBrake open source video transcoder on Mac/OSX were made aware that, along with their expected video software, they may have also downloaded a trojan that was uploading their sensitive data to a remote server [23]. HandBrake developers recommended users perform checksum validation to determine if their install was compromised [22].

As every internet user is certainly aware, downloading resources over the internet comes with considerable risk. This risk can be divided into three categories: response authentication, communication confidentiality, and resource integrity. Response authentication allows us to determine if a response received indeed originates from its purported source. This can be accomplished through, for instance, the adoption of a Public Key Infrastructure (PKI) scheme [9]. Communication confidentiality, on the other hand, allows us to keep the data transacted between two or more parties private except to said parties. This is can be accomplished through some form of encryption, such as AES [11]. Finally, resource integrity allows us to verify that the data we are receiving is the data we are expecting to receive.

When it comes to response authentication and communication confidentiality concerns on the internet, the state of the art in attack mitigation is Transport Layer Security (TLS) and its Hyper Text Transfer Protocol (HTTP)/PKI based implementation, HTTPS [7, 9, 14, 15, 39]. Assuming well behaved certificate authorities and modern browsing software, TLS and related protocols, when properly deployed, mitigate myriad attacks on authentication confidentiality.

However, as a *communication* protocol, TLS only guarantees the integrity of each *end to end communication* via message authentication code (MAC) [15]. But protected encrypted communications mean nothing if the contents of those communications are corrupted before the fact. Hence, the integrity of resources at the application layer (rather than the transport layer) is outside of the model addressed by TLS and HTTPS [15, 39].

Attacks on resource integrity can be considered a subset of *Supply Chain Attacks* (SCA). Rather than attack an organization directly, SCAs are the compromise of an organization's software source code (or any product) via cyber attack, insider threat, upstream asset compromise, trusted hardware/vendor compromise, or other attack on one or more phases of the software development life cycle. These attacks are hard to detect, even harder to prevent, and have the goal of infecting and exploiting victims by abusing the trust between consumer and software vendor [35].

Ensuring the integrity of resources exchanged over the internet despite SCAs and other active attacks is a hard and well studied problem [1, 2, 8, 9, 18, 31, 33, 39]. For a long time, the de facto standard for addressing this risk in the generic case is with the use of *checksums* coupled with some secure transport medium like TLS/HTTPS. Checksums in this context are cryptographic digests generated by a cryptographic hashing function run over the resource's file contents. When a user downloads a file from some source, they're expected to run the same cryptographic hashing function over their version of the resource, yield a local checksum, and match it with the authoritative checksum given to them from said source.

However, checksums have come up short as a viable solution to the resource integrity problem. Foremost is a harsh reality: the clear majority of end users will not be burdened with manually calculating checksums for the resources they download. And even if they did, said user must search for the corresponding authoritative checksum to verify their calculation. As there is no standard storage or retrieval methods for checksums, end users are not guaranteed to find an authoritative checksum, even if they exist. If they manage to find the authoritative checksum and also recognize the checksums are different, the user is then expected to "do the right thing," whatever that happens to be in context.

Then there's the futility of co-locating a resource and its checksum on the same distribution system. While cost effective compared to hosting two or more discrete systems—one hosting the resource and one hosting the resource's checksum—an adversary that compromises a single distribution system hosting both a resource and its checksum can mutate both, rendering the checksum irrelevant. Among many examples of such breaches are the 2012 hack of phpMyAdmin [37, 38], the 2016 hack of Linux Mint [34, 42], and the 2017 hack of HandBrake [22].

Clearly, checksums as they are employed currently are not effective at guaranteeing resource integrity. Recognizing this, some corporations and large organizations rely instead on PKI-based solutions such as digital signature validation and code signing [9]. These solutions have been deployed successfully to mitigate resource integrity attacks in mature software package ecosystems (*e.g.,* Debian/apt, Red Hat/yum, Arch/pacman) and walled-garden app stores like Google Play, Apple App Store, and the Microsoft Store.

Unfortunately, not all resources available on the internet are acquired through software package ecosystems with builtin PKI support; moreover, these PKI schemes are not compatible with one another and cannot scale to secure arbitrary resources on the internet without significant cost and effort. Worse, roll-your-own PKI is *hard to get right* [9], implying systems built atop them are inherently more *fragile*, by which we mean susceptible to malfunction due to small errors or misconfigurations.

In this paper, we propose DNSCHK, a novel method of verifying the integrity of resources downloaded over the internet that is a complete replacement for traditional checksums. We approach the problem with four key concerns in mind: a) implementations provide security guarantees transparently without adding any extra burden on the end user, *i.e.,* neither grappling with a new user interface nor any additional labor is required during standard usage; b) configuring the validation method is simple for service administrators and system operators to integrate and deploy while ensuring configuration of a potentially-expensive discrete secondary system to host checksums is unnecessary; c) the validation method is not tightly coupled with any particular highly-available system; and d) no new HTML or JavaScript language additions, application source changes, or user-facing web server/infrastructure alterations are necessary. We implement DNSCHK as a proof-of-concept Google Chrome extension as well as a patch to the FileZilla FTP client.

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 empirical demonstration of DNSCHK's protective utility via a patched HotCRP instance[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 source code or infrastructure changes at any level other than DNS, 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 DNSSEC-enabled systems; it protects end users whose software implements DNSCHK while remaining unnoticeable to users of whose software does not.
- We present our prototype DNSCHK implementations for Google Chrome and FileZilla and demonstrate its effectiveness in automatically and transparently mitigating the accidental consumption of compromised resources from a compromised server hosting a compromised web portal. 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.
- We carefully and extensively evaluate the security, scalability, and performance of our automated defense against resource corruption and provide a publicly

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

accessible empirical demonstration of DNSCHK's protective utility.

We release the DNSCHK solution to the community as open source software to prompt exploration of the DNSCHK approach[2].

2 Background

[TODO: In this section we ...]

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

2.1 Current Detection and Prevention Solutions

[TODO: There are several. Blah blah.]

2.1.1 Anti-Malware Software.

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

2.1.2 HTTPS / Encrypted Channel.

[TODO: (what it is, why it fails)] [7, 14, 15, 19, 39, 40]

2.1.3 Browser-based Heuristics and Blacklists.

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

2.1.4 Checksums.

[TODO: (what it is, why it fails; perhaps move part of abstract definition here?)]

2.1.5 Public Key Infrastructure.

[TODO: (what it is, why it fails)] [8, 16, 27, 47]

2.2 Motivation: Case Studies.

[TODO: Blurb about case studies. Information on breaches is spotty and incomplete because corps are scant on the details of their failure, so we are working with limited data.]

Case 1: Linux Mint. [TODO: Explain]

Case 2: Havex. [TODO: Explain]

Case 3: PhpMyAdmin. [TODO: Explain]

Case 4: HandBrake. [TODO: Explain]

3 The DNSCHK Approach

[TODO: In this section we ...]

[TODO: Resource Identifier (RI)] [TODO: Authoritative Hash (AH)] [TODO: Non-Authoritative Hash (NAH)] [TODO: Non-Authoritative Hash Validation (NAH Validation)] [TODO: Origin Domain (OD)] [TODO: Primary Label] [TODO: RI Sub-Label]

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

[TODO: Split explanation into "frontend" concerns and "backend" concerns]

[TODO: Go over algorithms!]

3.1 Proof-of-Concept Implementations

[TODO: (Google Chrome Extension: 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: Reference hotcrp demo but leave the description for the evaluation.]

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

4 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.

4.1 Threat Model

4.1.1 Compromised Resource

We consider the case where an adversary can influence or even completely control the victim's resource distribution mechanism (web page, file server, CDN, etc) in any way. In this context, the adversary can trick the user into downloading a compromised resource of the adversary's choice. This can be accomplished by compromising the resource on the victim's system or tricking the user into downloading a compromised resource on the adversary's remote system.

In this case, the adversary does not have control over any DNS zone(s) relevant to the function of DNSCHK.

If the adversary does not alter the Resource Identifier, the compromised resource will fail integrity validation during the NAH Validation step.

If the adversary does alter the Resource Identifier, there are two possibilities: a) the new Resource Identifier *does not* exist in the DNS zone, in which case DNSCHK will fail to resolve the NAH, hence the NAH Validation step will fail; b) the new Resource Identifier *does* exist in the DNS zone, therefore the "new" RI must be pointing to a different file's hash. Unless the adversary's goal is to swap one or more files protected

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

by DNSCHK and a particular DNS zone with another file also protected by DNSCHK and in that same zone, the NAH Validation step will fail. For the aforementioned ”swap” to work, the adversary would be required to both change the RI and also offer to the victim the DNSCHK protected file the ”new” RI corresponds to, which shrinks the attack surface significantly.

4.1.2 Compromised Authoritative Hash

We consider the case where an adversary can completely control the victim DNS zone(s) that allow DNSCHK to function. Therefore, the adversary can return an authoritative response of their choice to any DNS query.

In this case, the adversary does not have control over the victim’s file distribution mechanism (web page, file server, CDN, etc).

DNSSEC ensures the validity and authenticity of DNS responses. In order for the adversary to control any relevant DNS zones, they must have access to the authoritative DNS server and/or the appropriate DNSSEC keys.

Even if the adversary achieved this level of compromise, they do not have the ability to deliver a malicious payload in this case. However, the adversary could use control over the relevant DNS zones to cause denial-of-service attacks against those attempting to download the resource by causing all NAH Validation checks to fail. This is mitigated by DNSCHK allowing the user to ”override” its error states, similarly to Google Chrome’s invalid HTTPS certificate warning page allowing advanced users to pass through.

4.1.3 Compromised Resource and Authoritative Hash

We consider the case where an adversary can influence or even completely control the victim’s resource distribution mechanism (web page, file server, CDN, etc) in any way. Additionally, the adversary can completely control the victim DNS zone(s) that allow DNSCHK to function. Therefore, the adversary can make the user download a compromised resource and also return a (compromised) AH that legally corresponds to said compromised resource.

4.1.4 Determining the Origin Domain

If an adversary manages to compromise a web page/server, they have two options. They can mutate the resource directly, which would be observable via DNSCHK. They could also mutate the web/download page itself, replacing the anchor with a malicious one that points to a compromised resource on the adversary’s remote system. This system could be configured with valid DNSCHK DNS TXT records, allowing the adversary to trick DNSCHK into green lighting the resource without complaint. Similarly, an adversary could redirect the user to an valid and innocuous (but compromised) page that very quickly redirects the user again to the compromised resource with the goal of tricking DNSCHK.

In order to prevent such implementation-level attacks, we make a distinction between the domain that the hyperlink containing the desired resource references and the ODor the domain of the document within which said hyperlink exists. The extension must be implemented such that the OD is resolved as early as possible in the page loading process. The scope of the OD is at the tab level, meaning there is one OD determined for each open browser tab. Once determined for a tab, the OD should not be recalculated for some period of time. If the browser is navigated within this time period, the user will be asked to verify that the OD is what they expect it to be (should be a familiar URL).

We implement this mitigation using chrome Download API’s DownloadItem::referrer property as the OD. We catch redirection attacks by assuming any page that begins a download in under 3 seconds is suspicious and requires affirmation by the user.

[TODO: Figuring out OD for FTP is really easy, though]

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

[TODO: 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 [5].]

4.3 Deployment and Scalability

[TODO: Discuss envisioned deployment strategies for resource providers.]

[TODO: 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.]

4.4 Performance Overhead

[TODO: Additional Download Latency, Additional Network Load, Runtime overhead, etc. All nixed.]

5 Discussion

In this section, we examine current and previous DNS-based and other cryptographic schemes, most of which are based on public key cryptography. Further, we note PGP’s limiting human factors, how those factors also apply to the checksum solution, and how the DNSCHK solution avoids them. Thereafter, we discuss some limitations of the DNSCHK methodology, implementation, and DNS itself.

5.1 Additional Related Work

Cryptographic Data in DNS Resource Records. Storing cryptographic data in the DNS network is not a new idea. The DNS-Based Authentication of Named Entities (DANE) specification [16, 27, 47] defines the “TLSA” and “OPENPGPKEY” DNS resource records to store cryptographic data. These resource record types, along with “CERT” [29], “IPSECKEY” [41], 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. With DNSCHK, however, we use “TXT” records to map Resource Identifiers to Authoritative Hashes. In accordance with RFC 5507 [17], an actual DNSCHK implementation would necessitate the creation of a new DNS resource record type.

PGP/OpenPGP. Though PGP addresses a fundamentally different threat model than DNSCHK, it is useful to note: many of the same human and UX factors that make the cryptographically solid OpenPGP standard and its various implementations so unpleasant for end users also exist in the context of download integrity verification and checksums. End users cannot and *will not* be burdened with manually verifying a checksum; as was the case with PGP 5.0 [46], some users are likely confused by the very notion of a checksum, if they are aware of checksums at all. If PGP's adoption issues are any indication, users of a security solution that significantly complicate an otherwise simple task are more likely to bypass said solution rather than be burdened with it. To assume otherwise can have disastrous consequences [46] (also see: Section 2).

Link Fingerprints and Subresource Integrity. The Link Fingerprints (LF) draft describes an early HTML anchor and URL based resource integrity verification scheme [31]. Subresource Integrity (SRI) describes a similar production-ready HTML-based scheme designed with CDNs in mind. Like DNSCHK, both LF and SRI employ cryptographic digests to ensure no changes of any kind have been made to a resource file [2]. Unlike DNSCHK, LF and SRI rely on the server that hosts the HTML source to be secure; specifically, the checksums contained in the HTML source must be accurate for these schemes to work. An attacker that has control of the web server can alter the HTML and inject a malicious checksum. With DNSCHK, however, an attacker would also have to compromise the DNS zone or whichever distributed system hosted the mappings between Resource Identifers and Authoritative Hashes.

Content-MD5 Header. The Content-MD5 header field is a deprecated email and HTTP header that delivers a checksum similar to those used by Subresource Integrity. It was removed from the HTTP/1.1 specification because of the inconsistent implementation of partial response handling between vendors [18]. Further, the header could be easily stripped off or modified by proxies and other intermediaries [33].

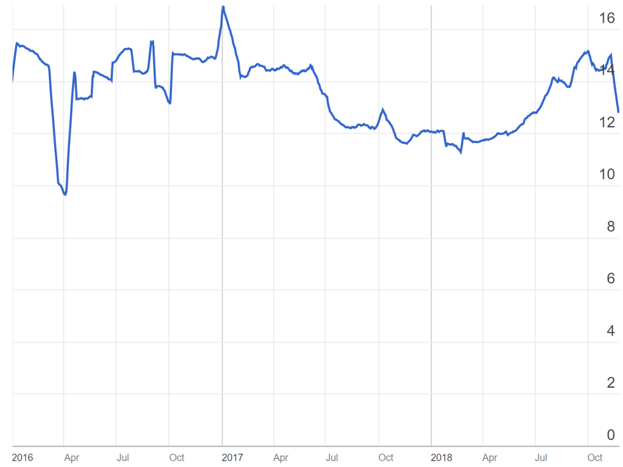

Figure 1. APNIC estimate of global DNS resolvers (Google PDNS as well as local resolvers) performing DNSSEC validation from January 2016 to December 2018.

Code Signing. [TODO: (specific case rather than the generic one; I think we should address the differences explicitly: we're simpler; we don't make the system more fragile; no central authority; applicable to more than just software binaries)]

5.2 Limitations

5.2.1 DNSSEC Adoption is Slow

DNSSEC is hard to configure correctly [12, 25, 26, 48], which make services that attempt to deploy it significantly more *fragile*. For instance, services deploying DNSSEC become vulnerable to small DNSSEC resource record misconfigurations that can take entire services offline with varying levels of conspicuousness. Further, any DNS-related issue becomes harder to diagnose with DNSSEC enabled. Some registrars and DNS operators do not offer DNSSEC capability at all, restrict access behind a paywall, or lack an interface for proper key management [28]. Despite ongoing efforts to simply DNSSEC deployment, adoption remains low for these reasons. Only around 3% of Fortune 1000 and 9% of university domains, a number that is very slowly on the rise [36].

On the client side, adoption of DNSSEC validation by DNS resolvers is low and growth is slow. Fig. 1 shows that, worldwide, some 14% of DNS requests have DNSSEC extensions validated by the resolver towards the end of 2018 [3]. The overall trend is positive thanks to large public resolvers like Google (8.8.8.8 and 8.8.4.4) and Cloudflare/APNIC (1.1.1.1), as well as the increasing number of compliant local/ISP resolvers.

It should be noted that DNSSEC does not necessarily make the DNS network, i.e. properly configured DNS servers, more vulnerable to amplification or other types of reflection attacks [4] than it already is as a UDP-based content delivery service [44, 45].

5.2.2 DNS-Specific Protocol Limitations

DNS [32] was not originally designed to transport or store relatively large amounts of data, though this has been addressed with EDNS0 [13]. The checksums 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 [27, 29, 41, 47].

Several working groups are considering DNS as a storage medium for checksums/hash output as well, such as securitytxt [20]. A widely deployed example of DNS "TXT" resource records being used this way is SPF and DKIM [10].

Additionally, DNSCHK does not add to the danger of amplification and other reflection attacks on DNS; these are generic DNS issues addressable at other layers of the protocol.

5.2.3 Supply Chain Attack Diversity

[TODO: There are many types of SCAs that can occur at any phase of the software development life cycle. DNSCHK only protects against attacks in the latter 3 categories. (Maybe bring the table back?) (Maybe talk about CCleaner falling out of scope?)]

5.2.4 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 [21] 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 FileSystem API that would provide similar functionality were it to be widely considered [6].

DNSCHK would be even more effective as a browser extension if Chrome/Chromium or 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 and 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 a background page based extension.

5.3 Future Work

5.3.1 Merkle Trees and Early Resource Validation

Using Merkle Trees instead of pure hashing functions to offer partial verification of large files, i.e. if the file we're downloading is 10TiB, we don't have to wait for it to finish downloading before we render a failing judgement. This saves the user time. Perhaps using the Tiger hash, since Tiger Merkle Trees seem to be popular among large P2P and file sharing applications.

5.3.2 Replacing RIs with URNs

The goal of the Resource Identifiers (RI) is very similar to that of Uniform Resource Names (URN). It may make sense to replace the mapping between RIs and Authoritative Hashes with purely URN-based DNS lookups that return specially formatted TXT records upon success. This would further simplify the deployment process for service administrators since DNS updates would be based upon the resource's contents instead of both its contents *and where it is located physically on a distribution server*. It may also allow for additional confirmation methods of the identical resources in different domains and in different locations.

We did not choose a URN-based scheme in our initial approach due to a new URN scheme requiring the registration of a unique identifier with the Internet Assigned Numbers Authority. Going forward, we can potentially adopt a URN scheme that already exists, such as Magnet links [30] or the informal IETF draft for hash-based URN namespaces [43]. With URNs, we can ensure our naming scheme is based solely on a resource's contents rather than both its contents and it's location on a web server.

6 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] D. Akhawe, F. Braun, J. Weinberger, and F. Marier. *Subresource Integrity*. W3C Recommendation. http://www.w3.org/TR/2016/REC-SRI-20160623/. W3C, 2016.

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

[4] 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.

[5] 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.

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

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

[8] 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.

[9] D. Cooper, S. Santesson, S. Farrell, S. Boeyen, R. Housley, and W. Polk. *Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile*. RFC 5280. http://www.rfc-editor.org/rfc/rfc5280.txt. RFC Editor, 2008. URL: http://www.rfc-editor.org/rfc/rfc5280.txt.

[10] 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.

[11] J. Daemen and V. Rijmen. "AES proposal: Rijndael". In: (1999).

[12] 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.

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

[14] 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.

[15] T. Dierks and E. Rescorla. *The Transport Layer Security (TLS) Protocol Version 1.2*. RFC 5246. http://www.rfc-editor.org/rfc/rfc5246.txt. RFC Editor, 2008. URL: http://www.rfc-editor.org/rfc/rfc5246.txt.

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

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

[18] R. Fielding and J. Reschke. *Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content*. RFC 7231. http://www.rfc-editor.org/rfc/rfc7231.txt. RFC Editor, 2014. URL: http://www.rfc-editor.org/rfc/rfc7231.txt.

[19] 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.

[20] 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.

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

[22] HandBrake. *Mirror Download Server Compromised*. 2017. URL: https://forum.handbrake.fr/viewtopic.php?f=33&t=36364.

[23] "HandBrake hacked to drop new variant of Proton malware". In: *MalwareBytes Labs* (2017). URL: https://blog.malwarebytes.com/threat-analysis/mac-threat-analysis/2017/05/handbrake-hacked-to-drop-new-variant-of-proton-malware/.

[24] "Havex". In: *New Jersey Cybersecurity and Communications Integration Cell Threat Profiles* (2017). URL: https://www.cyber.nj.gov/threat-profiles/ics-malware-variants/havex.

[25] 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.

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

[27] 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.

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

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

[30] *MAGNET-URI Project*. URL: http://magnet-uri.sourceforge.net/.

[31] G. Markham. *Link Fingerprints*. 2008. URL: http://www.gerv.net/security/link-fingerprints/.

[32] 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.

[33] J. Myers and M. Rose. *The Content-MD5 Header Field*. RFC 1864. RFC Editor, 1995.

[34] N. Nelson. "The impact of dragonfly malware on industrial control systems". In: *SANS Institute* (2016).

[35] NIST. *Cyber Supply Chain Risk Management*. URL: https://csrc.nist.gov/Projects/Supply-Chain-Risk-Management.

[36] NIST. *Estimating IPv6 and DNSSEC Deployment Status*. URL: https://usgv6-deploymon.antd.nist.gov/govmon.html.

[37] "phpMyAdmin corrupted copy on Korean mirror server". In: *SourceForge Official Blog* (2012). URL: https://sourceforge.net/blog/phpmyadmin-back-door/.

[38] "PMASA-2012-5". In: *PMASA-2012-5* (2012). URL: https://www.phpmyadmin.net/security/PMASA-2012-5/.

[39] 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.

[40] 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.

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

[42] L. M. Team. *Beware of hacked ISOs if you downloaded Linux Mint on February 20th!* 2016. URL: https://blog.linuxmint.com/?p=2994.

[43] P. Thiemann. *A URN Namespace For Identifiers Based on Cryptographic Hashes*. Internet-Draft draft-thiemann-hash-urn-01. Archived. Internet Engineering Task Force, 2003. 10 pp. URL: https://datatracker.ietf.org/doc/html/draft-thiemann-hash-urn-01.

[44] 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.

[45] 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.

[46] 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.

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

[48] 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.