Web Content Integrity: Tamper-Proof Websites Beyond HTTPS

Sven Zemanek(✉), Sebastian Tauchert, Max Jens Ufer, and Lilli Bruckschen

Fraunhofer FKIE, Bonn, Germany
{sven.zemanek,sebastian.tauchert,max.jens.ufer,
lilli.bruckschen}@fkie.fraunhofer.de

Abstract. We propose Web Content Integrity, a framework that allows a service provider to guarantee the integrity of their static website, even in the face of a compromised web server. Such integrity assurances can then be used to implement a secure end-to-end encryption application built in the form of a website. Our framework encompasses developers, the Domain Name System, and web browsers. To accomplish the integrity guarantees, our framework makes use of an index of queryable URLs and allowed redirects for the website, and publishes the cryptographic hash value of the index in the DNS. Web browsers can then use the information from the DNS to verify that the resources they retrieve from the web server have not been tampered with. The required data structures can be generated automatically, and the framework introduces an initial delay of about 4 ms and a recurring delay for each request of about 2 ms for a sample website.

Keywords: Web Security · Integrity · DNS

1 Introduction

Current web security measures such as HTTPS protect the integrity of data transmission between web server and client. However, the web server serving the website is not technically restricted from answering requests with arbitrary response content. An attacker who compromised the web server can leverage this to provide false or misleading information to website visitors. In addition, such an attacker can possibly extract secret information from clients or distribute malware. Some service providers have extra strong integrity requirements, like a voting authority that publishes election results, or a distributor of software used in classified contexts. Frequently, service providers do not host their websites themselves, on their own infrastructure, but use the services of dedicated web hosting providers. On the one hand, these hosting providers have all the options of an attacker who compromised a web server, and on the other hand, they could be breached themselves, giving an attacker control over a website's content. This means that even if one has trust in their legal framework to guarantee that

Published by Springer Nature Switzerland AG 2024
N. Meyer and A. Grocholewska-Czuryło (Eds.): SEC 2023, IFIP AICT 679, pp. 1–14, 2024.
https://doi.org/10.1007/978-3-031-56326-3_1

hosting providers behave properly, a technical solution to guarantee the integrity of one's website is still needed.

We propose *Web Content Integrity* (WCI), a framework that allows service providers to cryptographically ensure that visitors of their website receive only content that has not been tampered with. This is accomplished by first compiling an index of URLs and corresponding cryptographic hashes of the content available under the respective URL as well as allowed redirects. The cryptographic hash of this index is then published in the Domain Name System, next to the entries for the domain name that connects the domain name with IP addresses. WCI effectively eliminates the possibility for a web server to serve modified or entirely different content for a domain name. This leaves malicious actors only with the option to completely disable access to certain content. While our framework is tailored towards static websites, we also sketch a migration path for dynamic websites towards using WCI.

In summary, our contributions are as follows:

1. We propose a framework for service providers to make the integrity of their website verifiable using cryptographic hashes, and for web browsers to obtain those cryptographic hashes via DNS
2. We demonstrate that the creation of metadata required for the integrity verification can be automated, by implementing plugins for multiple static site generators
3. We determine properties of websites that can reasonably apply the framework
4. We provide upper bounds for the delays introduced when WCI is implemented by web browsers and deployed for a website

The remainder of this paper is structured as follows: After presenting related work in Sect. 2, we define the WCI framework in Sect. 3. We discuss the applicability of the framework, its compatibility with existing web infrastructure, possible attacks, and introduced overheads in Sect. 4, and summarize our findings in Sect. 5.

2 Related Work

There are already existing concepts for making sure the content of remote resources matches an expectation.

Subresource Integrity [11] allows resources from "script" or "link" tags that are included in a website, i.e., JavaScript or CSS files, to be accompanied by cryptographic hash values. The resource is checked against the hash values when it is retrieved. If no hash matches, the script or style is not applied, and it is treated as a network error where the resources could not be loaded. While this prevents tampering with the contents of specific types of resources, it does not cover all types of content (like images), and the HTML content which loads the resources is itself not protected from tampering. Nevertheless, the principle of checking loaded resources against cryptographic hashes before using them is

also used in WCI, and the widespread availability of Subresource Integrity in web browsers demonstrates the technical feasibility.

Mylar [6] ensures the integrity of static web applications by cryptographically signing the root HTML content and serving subresources from a second domain in order to leverage the web browser's same-origin policy. Included subresources are accompanied by a cryptographic hash that is checked similarly to Subresource Integrity. The key for verifying the signature of the root HTML page is stored in the TLS certificate. This may be problematic since the TLS certificate is being provided by the web server and can therefore be manipulated or replaced by an attacker with access to a Certificate Authority that is trusted by the web browser.

There are also approaches for integrity verification that have been developed before the widespread deployment of HTTPS.

Already in 1998, Peacock and Powell [7] described an algorithm to calculate a MD5 hash value over the contents of a website and selected subresources. This hash value was to be included in a metadata set for the BIBLINK project. The hash value could then be used to verify that the parts of a website covered by the hash value did not change since the metadata set had been created. The concept represents an early framework for expressing expectations about website contents, albeit partly outside the web browsing context.

Bayardo and Sorensen [1] suggested the construction of a Merkle tree over the contents of a website to provide integrity guarantees. The root hash value of that Merkle tree has to be transferred to clients over a secure channel that cannot be tampered with by web servers, for which the authors suggest among others to use the DNS. Besides guaranteeing the integrity of web content, the framework can also express that a specific resource does not exist on the website. The Merkle tree is constructed in a way that does leak the number of resources available on the website, but does not leak the actual URLs to those resources. Due to the properties of that construction, the use of wildcards or regular expressions in path specifiers is not possible. Also, since clients always need to know the current root hash of the Merkle tree in order to validate the website contents, updates of the website contents require immediate redistribution of the root hash value over the secure channel. When a resource is requested, parts of the Merkle tree below the root hash value are transferred in a special response header. This requires modifications to the web servers' behavior.

Sedaghat et al. [9] proposed to run additional software on the web server that validates the web server's responses against cryptographic hash values. This is not a suitable mechanism to protect against tampering from an attacker-controlled web server, since the integrity mechanism on the web server can be tampered with as well.

Singh et al. [10] advocated for HTTPi, a protocol that extends HTTP with integrity guarantees. Reis et al. [8] proposed what they called "Web Tripwires": JavaScript snippets that are included in websites, check client-side whether the site has been modified in transit, and report back to the web server if a modification is detected. Both approaches focus on the manipulation of content in transit

between a web server and a web browser. They do not protect against a compromised web server and are also made obsolete by the widespread deployment of HTTPS.

3 Web Content Integrity

We propose *Web Content Integrity* (WCI), a framework to cryptographically verify the integrity of content served by a web server. The framework involves three parties: Developers who generate the content that is served by web servers, the Domain Name System (DNS), and web browsers. Section 3.1 defines the framework, Sect. 3.2 describes scaled-down variants of the framework with reduced scope, Sect. 3.3 lays out considered alternatives for some design decisions, and Sect. 3.4 details how we automated the creation of the necessary data structures.

3.1 Framework

Web Content Integrity ensures the integrity of websites in the following way:

1. When a website is created, a WCI index for the website contents is created alongside it
2. The website and WCI index are deployed to a web server
3. A WCI DNS record is configured
4. Web browsers use the WCI DNS record to validate the WCI index, and use the WCI index to validate the integrity of resources they retrieve from the web server

The WCI index file represents a cryptographically verifiable expectation about the responses that a web server gives to requests. A website for which WCI is configured must provide a WCI index file under the path `/wci.txt`.

This WCI index file consists of lines of text and is structured as follows:

1. The file starts with a `REDIRECTS` section, mapping all paths for which the web server may send a "Location" header to redirect to another URL to a list of the allowed values of the "Location" header.
2. This is followed by a `PATHS` section, which maps all queryable paths the website is supposed to provide to the SHA-256 hash value of the expected response content for that path.

Figure 1 depicts a sample WCI index for a website with one redirect rule and three URLs with hashes in the `PATHS` section.

Paths in the `PATHS` and the `REDIRECTS` section are treated as regular expressions. The `PATHS` and the `REDIRECTS` section may be empty, with only their respective section header being present. Location values in the `REDIRECTS` section may use capture groups from the regular expression in their respective path. This flexible definition of paths enables developers to define hashes for pages that are available under many URLs, without having to enumerate all of those URLs.

```
#REDIRECTS
"/goto/(.+)":"https://example.org/$1"
#PATHS
"/":858d80f786e52cdece36d141de383...c9ce
"/about":309d702383574787c1925ae13703...5bb2
"/download.zip":15cfd081fc216ea06d74b31ff343...7837
```

Fig. 1. Sample WCI index with a REDIRECTS section with one entry and a PATHS section with three entries. Three dots indicate shortened lines for display purposes.

This is especially useful for covering Single-Page-Applications. It also makes it possible to specify hashes for default pages, like a 404 error page that is displayed for URLs that are not available on the website. This reduces false-positive alerts on websites that deploy WCI.

The cryptographic hash value is calculated over the contents of the WCI index file. Note that this is purely an operation on a string. No whitespace or newlines are manipulated. This makes the construction robust against differences in serialisation/deserialisation of the various clients working with the file.

A web browser must know that WCI is configured for a domain in order to act accordingly. Also, the web server that serves the website under that domain must not be able to interfere with the web browser learning this information. Therefore, DNS is a suitable vehicle to convey this information and the cryptographic public key. Similar to DANE [3], which uses DNS records to pin TLS certificates, we announce the cryptographic hash value that can be used to verify the WCI index via DNS records.

We define a new WCI DNS record to consist of three components:

1. specification version indicator
2. identifier of a cryptographic hash function (sha256)
3. cryptographic hash value (base64 encoded)

Figure 2 shows an annotated example of such a DNS record. When a DNS server receives an A (IPv4) or AAAA (IPv6) query, and one or more WCI records exists for the queried domain, all WCI records must be sent in the additional records section of the response, alongside the answer to the original request. As "the additional records section contains RRs which relate to the query, but are not strictly answers for the question" [4], it is a suitable place to transmit this information, and existing clients that follow RFC 1035 should not malfunction due to the presence of data in the additional records section of a DNS response.

Web browsers query DNS information without having to interact with a web server first.[1] If a WCI DNS record is present, WCI is configured for the corresponding domain, and the absence of corresponding features from the responses of the web server must be considered a severe error, possibly indicating an attack.

[1] Chromium and Firefox implement DNS clients that issue their own queries and process the responses. They do not require DNS support from the underlying operating system.

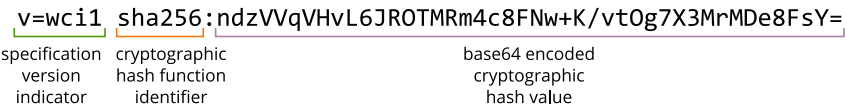

Fig. 2. Structure of a WCI DNS record

For the WCI framework to function, web browsers must implement verification and application of the WCI index. When a web browser queries a domain for which WCI is configured, it must first fetch the WCI index under the path `/wci.txt` and verify that the cryptographic hash value of the WCI index content is announced by at least one of the WCI DNS records.

If this verification fails, this must be considered a severe error, possibly indicating an attack.

When the verification of the WCI index was successful, for each queried path, the response from the web server must be checked against the given hash values in the `PATHS` section. If no entry for the queried path is present, or if there is no such entry with a matching hash value, which means the queried URL is not covered by the WCI index, this must again be considered a severe error, possibly indicating an attack.

If the response from the web server contains a Location header, its value must be checked against the values given for the path in the `REDIRECTS` section. If no entry for the queried path is present or none of the allowed values match the given value, this must once again be considered a severe error, possibly indicating an attack.

To prevent false positives, e.g. during a site update, we additionally mandate the following retry policy:

1. A validation mismatch of a resource triggers a reload of the WCI index. If the reloaded WCI index is equal to the previous one, the error is confirmed and access to the Website is blocked. Otherwise, the WCI index is validated against the known WCI DNS records.
2. A validation mismatch of the WCI index triggers a reload of the WCI DNS records. If the reloaded WCI DNS records are equal to the previous ones, the error is confirmed and access to the Website is blocked. Otherwise, the WCI index is validated against the new WCI DNS records.

For requests during a site update, there may be multiple rounds of reloading. However, a malicious web server cannot maintain an endless chain of reloads because an error is confirmed once the same WCI DNS records are observed twice, which the web server cannot influence.

Website operators usually wish to update the contents of their website from time to time. To update a website on a domain for which WCI is configured, the following steps should be executed in order:

1. Create the new version of the website, create the WCI index and calculate its hash value
2. Add a WCI DNS record with the new hash value
3. Set the TTL of the WCI DNS records with the old hash value to 0
4. Wait for the original TTL of the WCI DNS records with the old hash value
5. Deploy the new version of the website on the web server
6. Remove WCI DNS records with old hash values

This sequence of events ensures that the old hash value is no longer cached in intermediate DNS resolvers when the new version of the website is deployed on the web server. Clients that contact the website at that point will either already have fetched the new WCI DNS record, or be prompted to do so by a validation failure.

When the procedures outlined above are properly executed, our presented framework guarantees that a web browser only processes resources that have not been tampered with, and prevents access to the modified website in all other cases. It does so in a failsafe way that takes into account the effects of caching.

3.2 Scaled-Down Variants

The full WCI framework requires modifications to web browser and DNS server behavior. For evaluation purposes and local applications, less invasive scaled-down variants of the framework can be used. Instead of a native implementation, the web browser logic can be implemented by an add-on, which either performs its own DNS queries or uses hardcoded WCI DNS records. Such a browser add-on can intercept responses and perform the WCI validation before either handing off the unaltered response to the web browser or blocking access to the resource when a validation error occurs.

Current DNS servers can return the necessary information in TXT DNS records instead of dedicated WCI DNS records. Explicitly querying WCI data in the DNS effectively doubles the DNS traffic. For a full-scale implementation of WCI, the behavior as described in Sect. 3.1 is therefore preferable.

3.3 Considered Alternatives

The WCI index is represented in a custom line based text format. We considered using commonly found structured data formats like JSON or YAML to store the WCI index, as tools for parsing and representing data in those formats are usually broadly available. These data formats provide a lot more features than required, like nested properties. To keep the processing of the WCI index as simple as possible, we chose to use the custom line based format instead.

It would be possible to alternatively specify a binary format and representation for the WCI index. However, the web ecosystem generally tends to use plain text files. Since files can be compressed for transit, we expect the impact

of the decision to choose a plain text format for the WCI index to be negligible with regard to the actual transmission size.

Due to how it is constructed, the WCI index leaks all queryable paths of the website. An alternative would have been to use hash values of the paths instead of the path specifiers, like [1] have chosen to. This would however prevent the use of regular expressions as path specifiers, since correctly guessing the used regular expression from a given path is infeasible for nontrivial cases. We prefer the flexibility that the use of regular expressions enables over the concealment of all valid paths, and have therefore decided against this alternative.

The WCI framework requires the selection of a cryptographic hash algorithm. Output size is the main point to consider besides general security considerations. SHA-256 has an output size of 256 bits, high general availability, and is currently considered cryptographically secure [2,5]. It is therefore a suitable choice compared to alternatives with bigger output size, like SHA-512 or SHA3-512.

We considered distributing a cryptographic public key instead of the cryptographic hash value in the WCI DNS record. The WCI DNS record would then also include a version identification number that is incremented with each new version of the website to prevent rollback attacks, and the WCI index would be extended by a signatures section that contains signatures over the version identification number and the redirects and paths sections. While such an approach could eliminate retries in some scenarios during website updates, it introduces additional challenges with regard to cryptographic key management, key rotation, key loss and compromise, requires the calculation of cryptographic signatures over selected parts of a file as well as the application of public key cryptography in the first place, and finally makes calculations considerably slower compared to the caluclation of cryptographic hash values. We therefore opted for the current framework specification without public key cryptography.

3.4 Automation

To ease adoption of a new concept, it is desirable to make its implementation as easy as possible. Most of the steps required for configuring WCI for a domain can be automated. We have developed tooling to generate the WCI index and the content of the WCI DNS record. To demonstrate that this functionality can be integrated into the build processes of static websites, we have created plug-ins for the popular static site generators Gatsby, which is written in JavaScript, and MkDocs, which is written in Python. Additionally, we have developed a generic npm package that provides the same functionality when applied to a directory structure of files, and can be integrated in arbitrary build pipelines. We use the developed tools to generate valid WCI indexes and DNS record data in our performance measurement experiments. Our implementations are available under https://github.com/fkie-cad/Web-Content-Integrity.

4 Discussion

4.1 Applicability

Currently, we expect the majority of websites to employ server-side processing. As WCI only works on static websites, such websites must be adapted before configuring WCI for them. Websites can be functionally separated into a static data component, a dynamic data component, and a static application component that works on the static or dynamic data. The integrity of the static data and the static application can be protected with WCI. Dynamic data can then be handled via another domain or subdomain for which WCI is not configured. Such data must be considered untrustworthy by the application component.

Investing the effort of separating dynamic data from the rest of the website in order to be able to configure WCI can be especially worthwhile for websites which provide authoritative information, like law texts or election results, or which provide a web application like a text or graphics editor for sensitive contents.

4.2 Compatibility with Existing Web Infrastructure

We want WCI to require only sparse modifications to existing web infrastructure. In this section we argue that many phenomena of the existing web infrastructure are not impaired by WCI.

DNS Caching. Resource records in DNS responses carry a time-to-live (TTL) field, indicating how long the contained information may be considered valid until it should be fetched again [4]. After a website update (cf. Section 3.1), clients will not yet accept the updated WCI index because its hash value has changed. The validation failure for the WCI index triggers a forced update of the hash value from the DNS, which resolves issue.

Websites Under Multiple Domains. Sometimes, websites are available under multiple domains or subdomains. A common example would be a website that is available under both example.org and www.example.org. As the WCI index does not encode the domain name in any way, it is possible to configure WCI for all of those (sub-) domains by setting the WCI DNS record, and serving the exact same files for all of them.

Specification Updates. The version identifier at the start of the WCI DNS record facilitates updates to the WCI framework. The involved entities can use the value of this field to adapt their behavior or content generation, or to determine that they do not recognize the specific version. As it is only present in the WCI DNS record, a compromised web server has no way to feign an older version of the specification. This prevents downgrade attacks based on the version of the WCI specification.

4.3 Defense Against Attacks

In order to connect to a website, web browsers query the DNS for the IP address(es) connected to a domain name and receive a response from a DNS server. Then, a connection is opened to an IP address from the response. While WCI does not protect against attackers who can control the content of responses to DNS queries, we argue that WCI does also not enable such an attacker to launch a new kind of attack. An attacker who controls the content of DNS responses has the following options with regard to WCI:

1. Remove the WCI record from a response if WCI is configured for the domain
2. Modify the WCI record so it no longer matches the content on the genuine web server
3. Add a WCI record for a domain for which WCI is not configured

The first case enables the web server to serve modified content. However, an attacker who can control the content of DNS responses can also replace the genuine IP address with one of a server under their own control, and serve modified content from there.

The second and third case lead to a denial of service, as the web server's responses no longer match the expectation that has been given by the DNS record. However, an attacker who can control the content of DNS responses can also suppress the genuine IP address from the response, or respond with an invalid IP address. This equally leads to a denial of service.

We conclude that control over the content of responses to DNS queries gives an attacker the ability to sidestep WCI, but WCI does not enable novel attacks. The following considerations are therefore all under the assumption that DNS queries and responses are not tampered with.

WCI protects against arbitrary modifications of the contents of a website. Even a compromise of the web server does not permit an attacker to circumvent the protection that WCI provides. Another attack method to consider are rollback attacks, where a compromised web server serves a previous version of the website. Such an attack can only be successful against clients which still have the WCI DNS record with the corresponding hash value cached. This only works for the duration of the TTL for the WCI DNS record. Since a rollback attack can only be conducted with a previous version of the website, the success of such an attack depends heavily on the goal of the attacker and whether that previous version of the website contains vulnerabilities or required content that enables the attack in the first place.

4.4 Overheads

We identify impacts on the network load caused by DNS requests and responses and on the processing of requests by DNS servers. We also look at the performance impact on the processing of responses by web browsers, and estimate a margin of improvement by comparison with SRI. We argue that the overhead needed for deployment of WCI in terms of development time and effort can be alleviated by use of automated tools.

DNS Load. In order to be able to grant the security guarantees that WCI provides, web browsers must query the WCI DNS record for each domain name they resolve. A scaled-down implementation (cf. Sect. 3.2) would send out two queries instead of one, effectively doubling the amount of DNS traffic caused by web browsers, which is undesirable. The alternative is to mandate that each response for an A or AAAA query include the WCI DNS record if one exists in the additional records section. Similar lookup behavior is already happening for A or AAAA queries for which a CNAME record is defined. This alternative does not increase the number of DNS requests. In this scenario, DNS servers do have to look up whether a WCI DNS record exists for each A and AAAA request they receive and have an answer for. This likely increases processing times of A and AAAA requests on dns servers, similar to how CNAME chains are resolved. Compared to a scenario where most clients send two DNS requests that have to be processed independently by the DNS servers, the version where WCI DNS records are automatically included in A and AAAA responses seems favorable both in terms of network usage and used processing power of DNS servers.

Performance Considerations and Comparison with SRI. Introducing WCI has a negative impact on performance. We have set up a local test setup that minimizes latency introduced by network communications and uses the minimal variant (see Sect. 3.2) with a browser extension that uses a hardcoded DNS WCI record. Using this test setup, we have measured two kinds of delay: Initial delay, which occurs once for each domain that is being connected to, and recurring delay, which occurs for every retrieval of a resource. Our performance measurements show about 4 ms initial delay and about 2 ms recurring delay for single resources, and about 3 ms initial delay and 13 ms recurring delay in total for a test of a full sample website with multiple subresources. Initial delay depends on the size of the WCI index, and recurring delay depends on the size of the fetched content.

These delays are composed as shown in Fig. 3. Depicted are three browser setups for testing to separate out the interesting kinds of delay (S1 is a baseline setup with a dummy extension for comparison):

S2 The WCI index is already cached in the web browser extension, so no initial delay occurs
S3 The WCI index is not cached in the web browser extension, so initial delay occurs every time
S4 An outdated WCI index is cached initially in the web browser extension, so the WCI index has to be reloaded once when a request is processed

The “WCI setup” step is where most of the initial delay occurs ($\sim$ 4 ms). Fetching and verifying the response, on the other hand, require around 4 ms and 2 ms respectively. Note that the concrete load time only refers to internal network traffic. These times are expected to increase in an actual deployment.

A special case is when a retry occurs. In this case, the time required to fetch and validate the WCI file is shifted to the “Verify content” step. In addition,

two hash validations are performed. A failing one, which triggers the retry, and a successful one after updating the cached WCI file. Thus, the time required in this case is about the same as if the WCI file had to be loaded directly from the server.

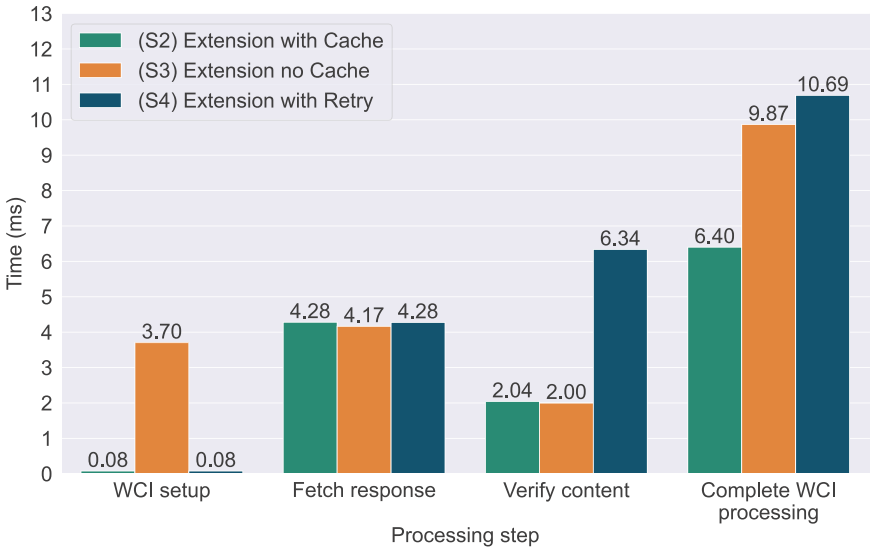

Fig. 3. Durations for sub-steps during the WCI validation.

We have conducted similar experiments with Subresource Integrity. The performance results from our tests with SRI are shown in Fig. 4.

SRI and WCI appear similar in terms of load time for files below 100 KiB. Once file sizes surpass this limit however, the native implementation of SRI outclasses our sample WCI implementation, taking only one third of the time of our WCI implementation at a file size of 10,000 KiB. The results spark hope that an efficient implementation of WCI can further reduce the processing time of larger resources by a factor of 2 to 3 or more. This transferability of results is justified by the similar nature of the processes in SRI and WCI.

Automation. The adoption of a new technology like WCI by developers can be facilitated by appropriate tooling support. We demonstrate that the creation of the WCI index file can effectively be fully automated by implementing plug-ins for MkDocs and Gatsby, two popular static site generators, as well as a generic tool that can be applied to arbitrary collections of files. The availability of easy-to-use and easy-to-integrate tools for WCI index creation keeps overheads for adopting this technology for existing static websites low.

Fig. 4. Comparison of average processing times for a minimal HTML page with inline (WCI) or linked (SRI) JavaScript of different sizes

5 Conclusion

Web Content Integrity focuses on an advanced understanding of the relationship between service providers and website visitors that goes beyond what HTTPS covers. It provides integrity guarantees for static websites that are resistant to tampering by a compromised web server. WCI is also resilient with regard to common maintenance tasks. We have demonstrated that the creation of the data structures required for WCI can be automated, hence facilitating adoption. With our sample implementation of the web browser logic in the form of a web browser extension, we have determined that the overhead in terms of additional delay introduced by WCI is expected to be around 9 ms initially, and about 2 ms for every resource. Initial delay increases with the size of the WCI index, and recurring delay increases with the size of the fetched content. Experiments with Subresource Integrity make it seem probable that the performance of our extension for bigger files can significantly be undercut by a native implementation. The evaluation of processing delays introduced at DNS servers by the deployment of WCI is considered future work, as is the development of methods to efficiently handle websites with many distinct queryable URLs.

References

1. Bayardo, R., Sorensen, J.: Merkle tree authentication of http responses, pp. 1182–1183 (2005). https://doi.org/10.1145/1062745.1062929
2. Bundesamt für Sicherheit in der Informationstechnik: Kryptographische Verfahren: Empfehlungen und Schlüssellängen (BSI TR-02102-1), Version: 2023-01 (2023)

3. Hoffman, P.E., Schlyter, J.: The DNS-based authentication of named entities (DANE) transport layer security (TLS) protocol: TLSA. RFC 6698 (2012).https://doi.org/10.17487/RFC6698, https://www.rfc-editor.org/info/rfc6698
4. Mockapetris, P.V.: Domain names - implementation and specification. RFC 1035 (1987). https://doi.org/10.17487/RFC1035, https://www.rfc-editor.org/info/rfc1035
5. National Institute of Standards and Technology: Fips pub 180-4 – secure hash standard (shs) (2015). https://doi.org/10.6028/NIST.FIPS.180-4
6. Popa, R.A., Stark, E., Valdez, S., Helfer, J., Zeldovich, N., Balakrishnan, H.: Building web applications on top of encrypted data using mylar. In: 11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14) (2014)
7. Powell, A., Peacock, I.: Metadata: Biblink.checksum. Ariadne (17) (1998). http://www.ariadne.ac.uk/issue/17/biblink/
8. Reis, C., Gribble, S.D., Kohno, T., Weaver, N.C.: Detecting in-flight page changes with web tripwires. In: NSDI, vol. 8 (2008)
9. Sedaghat, S., Pieprzyk, J., Vossough, E.: On-the-fly web content integrity check boosts users' confidence. Commun. ACM **45**(11), 33–37 (2002)
10. Singh, K., Wang, H.J., Moshchuk, A., Jackson, C., Lee, W.: Practical end-to-end web content integrity. In: Proceedings of the 21st International Conference on World Wide Web (2012)
11. Weinberger, J., Braun, F., Akhawe, D., Marier, F.: Subresource integrity. W3C recommendation, W3C (2016)