Network Information Flow Integrity: An Analysis on Subresource Integrity

Dieu-Anh Le
College of Engineering,
University of California, Davis
ndale@ucdavis.edu

Abstract
Subresource Integrity is a security feature developed by W3C in 2016 that allows web developers to ensure that the resources retrieved from the Content Delivery Network (CDN) are not modified in transit. By protecting the integrity of the loaded content, SRI effectively protects the system from different attacks like cross-site scripting (XSS), malicious code injections, or man-in-the-middle (MiTM) attacks. Along with the framework overview, this paper also analyzes any limitations the current SRI scheme has and proposes some enhancement features to tackle these challenges.

Keywords
Confidentiality, Integrity, Availability, Subresource Integrity, SRI, nonce

1. Introduction

For many years, developers have relied on the so-called 'CIA triad' as the core components to keeping their system secure, which stands for *Confidentiality* (restriction of access to information), *Integrity* (authenticity of data source and content), and *Availability* (accessibility of the system) **[1]**. In theory, any system satisfying the CIA triad is considered secure. However, in practice, there are trade-offs between different aspects of security. One of the concerns arises from the conflict between the functionality of a system and the design principle *Economy of Mechanisms* **[3]**. A system offering more functionality tends to be more complicated to develop and maintain. Therefore a secure system should be minimal to minimize exploitation loopholes and reduce inconsistencies. Since such trade-offs exist, this paper will focus on the integrity aspect of web content and will not emphasize much on the confidentiality and availability components.

Most of the time, web content is not composed of materials from a single origin. In general, the browser script would pull and gather content from multiple content delivery networks (CDN) and trust that it will give them the authentic expected content. In such a loading scheme, there are two vulnerability loopholes that an attacker could potentially exploit, either replacing the server with a malicious server or injecting arbitrary content to the script associated with the CDN.

There are two issues with the content loaded from the web: malicious server and malicious content. Multiple works have previously addressed the former issue. One typical approach is using a cryptographic protocol SSL/TLS to establish a secure channel between the client and the server. This encryption protects users from MiTM attacks when the attacker interferes with the traffic and may modify the communication between the user and server **[4]**.

Another example is the HTTP Strict Transport Security (HSTS), a policy that enforces the TLS mechanisms and ensures that website access over a secure connection. It sends an HTTP response header to tell the user agent that all communications with the server must be through the HTTPS protocol, replacing the regular HTTP protocol **[11,12]**. This is because the HTTPS protocol protects the header from being modified by the attacker, which may cause undesirable results. Lastly, another example is pinned public key, also known as Certificate Pinning. This security measure tells a particular web browser to accept a certificate from a list of given trusted Certificate Authorities, attempting to protect the browser from malicious file manipulation. The web browser may load the file if and only if the keys used in modifying the file were ensured by the Certificate Authorities. While all three methods were used to prevent MiTM attacks, the last one is more common with mobile applications **[13]**.

Due to the large overheads and high cost of maintenance of HTTPS, the majority of the web pages on the Internet remain loyal to the HTTP protocol. An experiment conducted by Felt et al. illustrates that between 2015-2017, the web browsing ecosystem experienced a significant change in the adoption of HTTPS protocol; yet, the figures for the adoption of the new secure protocol remain at the 50% range for both major browsers (Chrome and Firefox) conducted on two major operating systems (MacOS and Windows). In addition, they also found that HTTPS protocol usage varies globally, and the pattern is determined by browsing habits that differ across cultures. For instance, small nations or more developed countries have significantly higher adoptions of HTTPS protocol than East Asian countries such as South Korea, Japan, and China **[2]**.

Despite the previous rigorous work in exploring and enforcing web integrity, much work was done in server authentication but lacked attention in content authentication. For this project, I would like to study validating schemes that enforce website integrity and any concerns this extra feature introduces. This survey will begin with section 2 studies the framework of Subresource Integrity (SRI); section 3 outlines some limitations of the current framework and proposes possible enhancement features; section 4 will discuss any related work; lastly, section 6 is our conclusion.

2. Subresource Integrity

Websites and applications rely on various origins to retrieve the content. When a user requests, the web will fetch the file from the nearest *Content Delivery Network* (CDN). If the attacker replaces the file on the network, they can trick the user into executing arbitrary code. To overcome this issue, W3 introduced a subresource integrity (SRI) attribute in 2016 as part of the Content Security Policy to protect web applications from being modified. In this scheme, developers can add a cryptographic hash function to their script to ensure the integrity of the content. This approach ensures the CDN delivers the expected content. However, this doesn't solve all the problems.

Web content integrity has been divided into multiple integrity categories by developers for many years, defined as follows:
Web page development: To ensure such integrity, web developers need to check the availability of images and files used by web pages and validate the accuracy of web syntax **[5]**.
Data validity and security: This is the secure data collection over the web via Web forms **[7,8]** and verifying the integrity of the semantic web page content **[6]**.
Content authenticity: This ensures the document is neither compromised nor modified since its creation **[9]**.

In this survey, we will focus on the content authenticity aspect of web integrity. Subresource Integrity is a framework that ensures the embedded content obtained from the content delivery network is not modified.

2.1. The Integrity-trio Distinction

Web Integrity refers to the overall state performance of a web application. Specifically, it covers the accuracy and reliability aspects of its content, functionality, and security features. It encompasses a wide range of factors, such as the accuracy and completeness of the information presented on the site, with additional measures taken to ensure users' privacy.

Web Content Integrity is a component of Web Integrity, which refers to the idea that the content displayed on a website should reflect the intentions of the website's owners. In other words, the content should not be modified without the developer's consent and should not be free from malicious code or other harmful elements such as malware or phishing scams.

Subresource Integrity is a security mechanism used to enforce Web Content Integrity. It utilizes hashing to verify the appropriateness of external resources such as scripts, stylesheets, and images. In other words, each external source has a corresponding hash value at creation. When fetched, this hash value is compared against that stored in the HTML code in which the matching values indicate that the resource has not been tampered with. Section 2.4 discusses more about this.

2.2. Why is it a problem?

Web-based attacks occur in various forms such as code injection attacks (insertion of malicious code into a web application), malicious file manipulation (altering files without the web owners' consent), and data corruption (modifying content that results in inaccurate or incomplete information delivered to the client). As a result, W3C proposed the SRI algorithm as a solution to the problems above, which was first proposed in 2013 and soon got widely adopted by various major browsers.

Malicious code injection could cause devastating impacts on the companies. Some of the previous examples are the Target data breach, Equifax hack, and SolarWinds hack. In particular, the first incident happened in 2013 when Target suffered a massive attack that exposed customers' sensitive information. The attacker modified Target's web content to redirect users to a fake login web page and hence stole their credentials across 1,797 stores in the US **[14]**. Consequently, Target quickly lost customers' trust and experienced huge financial and legal impacts. In 2013, Equifax, which is one of the major credit bureaus in the US, exposed customers' personal and financial information in 2017 **[15]**. The last case happened to SolarWind in 2020, a software company providing network monitoring and management tools **[16]**. Attackers injected malicious code during the routine software update period to intercept the client's system. The last case was also an example of a supply chain attack that cascaded the impact and affected the US government and many large companies that consumed SolarWinds products. These three cases happened recently in which the targets were large entities, suggesting that even big organizations still suffer from web content integrity attacks.

2.3. Framework

SRI provides an additional layer of security for web pages, preventing unintended modification of web pages. The main components of the framework are Resource Integrity, Cryptographic Hash Function, and Content Security Policy **[17]**. The subsections below discuss each topic in detail.

2.3.1. Resource Integrity

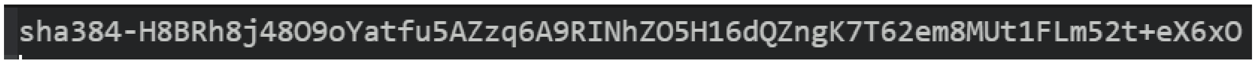
As part of the verification process, the user-agent requires that the *integrity metadata* is part of the HTTP request. The components of the metadata are the cryptographic hash function (shortened as *alg*), digest (*val*), and options (*opt*).

At a high level, a hash function is a mathematical function that takes inputs of variable lengths and outputs a bit string of a specified length. The common hash algorithms used in SRI are SHA-256, SHA-384, and SHA-512, which are hash collision-free functions. This is crucial because SRI heavily relies on hash functions to validate data, so if the chosen hash function is vulnerable to attack, the whole scheme becomes insecure. Therefore, the framework discourages developers from using previously broken hash functions, such as MD5 or SHA-1.

A digest is a compact, fixed-sized representation of the message after applying the cryptographic hash functions, often known as a hash function or checksum. This is also the object that we want to compare with when fetching content from the CDN to guarantee no unauthorized modifications. No options are defined by default, but they are there for future improvement. For instance, the options could be MIME type identifying the file types and formats and instructing the web browsers in handling the resources **[20]**.

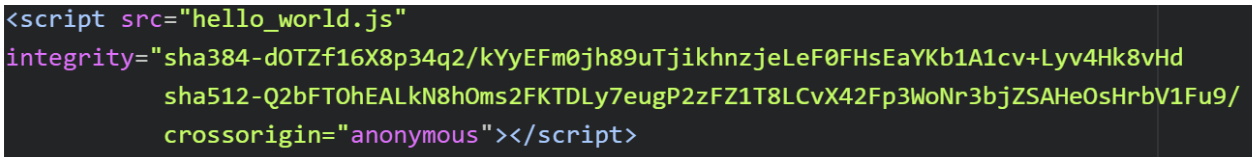
2.3.2. Cryptographic Hash Function

Figure 1.1 below illustrates an example of an integrity metadata for a script resource which contains only the string `alert(\'Hello, world.\');` with SHA-384 as the chosen cryptographic hash function **[20]**. The resulting digest turned out to be `H8BRh8j48O9oYatfu5AZzq6A9RINhZO5H16dQZngK7T62em8MUt1FLm52t+eX6xO` with the prefix specifying the hash function used.



```
sha384-H8BRh8j48O9oYatfu5AZzq6A9RINhZO5H16dQZngK7T62em8MUt1FLm52t+eX6xO
```

Figure 1.1. Example of script integrity metadata with hash function and digest ***[20]***



```
<script src="hello_world.js"
integrity="sha384-dOTZf16X8p34q2/kYyEFm0jh89uTjikhnzjeLeF0FHsEaYKb1A1cv+Lyv4Hk8vHd
          sha512-Q2bFTOhEALkN8hOms2FKTDLy7eugP2zFZ1T8LCvX42Fp3WoNr3bjZSAHeOsHrbV1Fu9/
          crossorigin="anonymous"></script>
```

Figure 1.2. Example of script integrity metadata with multiple hash functions ***[20]***

In addition, multiple hash functions may be applied to a single resource, which should be reflected in the integrity metadata as shown in *Figure 1.2*. In this example, there are two hash functions applied to the file `hello_word.js`, in which the user-agent will select the most robust hash function from the list and check the response with this metadata. The last attribute crossorigin specifies if the resource should be fetched via Cross Origin Resource Sharing (CORS) or not. This is a security mechanism enforcing data sharing of web browsers with other domains than where it originates from.

In the cases when there are multiple hash functions used, we must specify an order to which they should be used, which is implemented through a private helper function setting the order of whichever hash function is more secure (less likely to occur hash collision). In *Figure 1.2* above, SHA-512 is stronger than SHA-384 and hence the helper function such as `getPrioritizedHashFunction (‘sha384’,‘sha512’)` should return ‘`sha512`’ and for cases where the same hash function is used, it should return an empty string.

2.3.3. Verification

When loading the resource from the CDN, the browser computes the hash value of the retrieved content and compares it against the digest specified in the integrity attribute. If the hashes coincide, the browser executes the resource; otherwise, it prevents this resource from rendering and reports the error.

This algorithm helps secure the system because of the collision-free nature of certain hash functions. That is, given a particular string with a particular hash function. Modifying the original object to something arbitrary that yields the same hash value is challenging. Hence, the process ensures the resource remains untampered throughout the fetching process.

2.4. Goal

The main goal of SRI is to prevent various threats that occur from false web content, which therefore mitigates two main types of attacks:

Malicious Code Injection: By cross-checking the cryptographic hash function with the original hash function, any modification to the code would cause the hash function to change. The mismatch between the computed hash value and original digest indicates an unauthorized change, which helps prevent malicious code injection incidents like cross-site scripting (XSS) attacks.

Data Tampering: As a by-product of verifying that data has not been modified during the data fetching process from the CDN, SRI ensures that there was no MiTM attack since nothing was corrupted during the data transmission.

3. Possible Enhancement

While SRI offers a valuable tool for website developers to enhance web browser security, it does have some areas that need improvements. This section will discuss various drawbacks and possible enhancement solutions for the existing framework.

3.1. Hash Collision Attacks

Issue: Due to the heavy reliance on the cryptographic hash function in the SRI framework, it will only be as robust as the hash function used. Currently, SHA-2 is the most common and among the most robust hash functions, but this may change a few years down the line.

Enhancement: In line with regular system updates, developers should schedule regular checks and re-evaluates the strengths of hash functions used in the system. Although the commonly used hash functions in SHA-2 have not yet been broken, it's more secure to assume that they are at risk and so the developers should schedule a routine check on all the resources that used SRI to protect the integrity, ensuring that the hash functions are not recently broken and updated to new, more robust hash functions if needed. For instance, since its introduction in 2016, most web browsers implemented SRI with SHA-2 after SHA-1 vulnerabilities were established. Shortly, we expect SHA-3 to replace SHA-2 to become the most secure hash functions, then the developers would be expected to update all the integrity metadata of SRI to reflect the usage of SHA-3 if needed **[21]**. However, the advancement of quantum computing allows complex calculations to be completed in a much shorter period of time. As a result, hash functions are more vulnerable to attack.

3.2. Browser Incompatibility

Issue: Although SRI is supported by major browsers such as Google Chrome, Mozilla Firefox, Microsoft Edge, etc, this does not mean that all browsers support it. Because it is still a new feature, some browsers that are newly developed or those that were outdated may not support it. However, we could also argue that the more niche the browser, the less people using it and hence the effect of malicious code injection on those servers would not be as devastating as on more prevalent ones and so SRI serves its purpose of enhancing the system security on more popular platforms.

Enhancement: One approach is the web server only serves requests coming from SRI-supported browsers. This can be done by inspecting the user agent HTTP request header. In addition, developers could still choose other mechanisms to protect content integrity such as Content Security Policy (CSP) or HTTP Strict Transport Security (HSTS) as discussed earlier. Although it is best to use this SRI newly introduced feature, if it was not available on the current browser, the developer can always revert back to older mechanisms.

3.3. Limited Protection

Issue: SRI offers a scheme to validate the integrity of data. That is, when fetching data from the CDN, SRI ensures that the data is delivered without being tampered with, which effectively protects against man-in-the-middle attacks. However, there are many other types of attacks that are not protected by the SRI algorithm. For instance, if the attacker compromised the original server and modified the contents, then they can change the original digest value. Then, when the user retrieves the resource and computes the hash value, both values will match because the one stored in the script was compromised in the first place. In addition, SRI only protects original data from being modified but fails to capture the changes with injection of new resources.

Enhancement: In the first example, the origin server is compromised and approaches to mitigate this goes beyond the scope of this paper so we'll be focusing on the second case.

To prevent injection of new resources, we can introduce nonce (a one time random number) via the CSP HTTP response header. When the browser sends in a request to load a web page, the CDN will first respond with the HTTP Response header with this nonce attribute. When the HTTP Response body arrives, every resource in the body will also contain a nonce. These nonces will each be checked against that in the header. If the two match, the resource will be loaded, otherwise it'll be discarded by the browser. If we use nonce along with SRI, then the web browser ensures that the number of resources are as intended by the creator (no new script is injected) and all scripts are untampered. In other words, the quantity and quality of scripts are secured.

3.4. Overhead

Issue: The additional calculation required when fetching resources may add some overhead to the loading time of a web page which could interfere with user experience.

Enhancement: While there is a tradeoff between security and functionality, developers need to take some consideration to balance out the two elements when embedding SRI features into the system. In account for the additional time SRI is required to compute the hash value of each retrieved resource, the developer could reduce the rendering time of the actual resource. For instance, we could use gzip compression to reduce CSS and HTML files and hence reduce package transmission time. Additionally, a more complex SHA provides more security to a system but also requires more computation time. It is worth noting that having a robust system is important but it is not everything; user experience also plays a crucial role in determining the success of a web application. Therefore, developers should perform a cost-benefit analysis between the two elements to select the appropriate SHA version to protect their system.

4. Related work

This section discusses some previous research in securing web integrity.

4.1. HTTPS, HTTPA and HTTPi

While HTTP is the foundation for the World Wide Web communication, it is very insecure, so HTTPS was introduced as a secure version of HTTP. It uses the SSL/TLS protocol to protect the interaction between the client and server. However, the overheads that come with HTTPS limit many web domains from using it. Therefore, Choi and Gouda proposed HTTPi as an alternative solution with improved flexibility and reduced overheads **[24]**. Similarly, HTTPA is presented as an alternative to HTTP, focusing on authentication **[25]**.

4.2. PSK-TLS

Pre-Shared Key Transport Layer Security (PSK-TLS), a variant of TLS protocol, uses a symmetric shared key to authenticate and secure the communication initiated by the TLS handshake **[26]**. On the other hand, the traditional approach uses a public-private asymmetric key pair, which may add some overhead when computing the key. However, one limitation to PSK-TLS is that if the attacker compromises this shared key, they can make undetectable changes. One advantage of PSK-TLS over the traditional TLS protocol is that it offers more resistant protection to man-in-the-middle attacks. Because the key is shared before the communication, if the attacker intercepts in between, they cannot obtain the key to read the message. Furthermore, the only way to get this key is by compromising either the client or the server, which is another security problem.

4.3. Web Tripwire

Tripwire is a program in the file system used to report the consistency of the file content. In the first run, tripwire logs the initial state of all files in the system. For any following execution, it reports any detected changes made to the file, hence enforcing the integrity of the file system **[1]**. Reis et al. adopted a similar approach to assure web content integrity via the Web Tripwires model **[10]**. This model is a JavaScript code inserted into the client side of the script that identifies alteration to the web server at run-time. Despite that, there are multiple limitations to this approach. Firstly, it adds a third party to the web application **[27]**. This increases the overhead. Secondly, both the attack and defense JavaScripts share the same privilege. This could potentially allow the attacker to bypass security measures. Thirdly, the browser cannot guarantee the loading sequence of JavaScripts. As a result, malicious JavaScripts might be loaded before the protection mechanism.

5. Conclusion

Overall, SRI is a powerful tool to enhance web security. With a cryptographic hash, SRI validates whether data is tampered with and effectively protects the system against various attacks, such as cross-scripting, malicious code injections, and MiTM attacks. Nevertheless, it is not a solution to all security problems in web applications. Developers should use SRI alongside other security mechanisms, such as nonce via CSP, HTTPS protocol, and PSK-TLS secure channel to enhance the robustness of their websites.

6. Reference

[1] Bishop, M., Sullivan, E., & Ruppel, M. (2019). *Computer security: Art and science*. Addison-Wesley.

[2] Felt, A. P., Barnes, R., King, A., Palmer, C., Bentzel, C., & Tabriz, P. (2017, August). *Measuring https adoption on the web*. Retrieved February 2023, from https://www.usenix.org/system/files/conference/usenixsecurity17/sec17-felt.pdf

[3] R. E. Smith, "A Contemporary Look at Saltzer and Schroeder's 1975 Design Principles," in IEEE Security & Privacy, vol. 10, no. 6, pp. 20-25, Nov.-Dec. 2012, doi: 10.1109/MSP.2012.85.

[4] Krawczyk, H., Paterson, K.G., Wee, H. (2013). On the Security of the TLS Protocol: A Systematic Analysis. In: Canetti, R., Garay, J.A. (eds) Advances in Cryptology – CRYPTO 2013. CRYPTO 2013. Lecture Notes in Computer Science, vol 8042. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-642-40041-4_24

[5] Sedaghat, S., Josef Pieprzyk Macquarie University in Australia, J., Pieprzyk, J., & Vossough, E. (2002, November). *On-the-fly web content integrity check boosts users' confidence*. Communications of the ACM. Retrieved February 2023, from https://dl.acm.org/doi/pdf/10.1145/581571.581590

[6] Peacock, I. and Powell, A. BIBLINK.Checksum—an MD5 message digest for Web pages. Ariadne, 1998; www.ariadne.ac.uk/issue17/biblink.

[7] Sedaghat, S. Designing Electronic Forms in Web Applications: Integration of Form Components. Springer Verlag, Vol. 1987 (2001).

[8] Sedaghat, S. and Pieprzyk, J. Secure online data acquisition systems in a Web-based environment. In Proceedings of the International Workshop on Cooperative Internet Computing. Hong Kong Polytechnic University, Hong Kong, 2000, pp 37–42

[9] Harmelen, F.V. and Meer, J. WebMaster: Knowledge-based verification of Web-pages.. In Proceedings of Practical Applications of Knowledge Management, PAKeM'99. The Practical Applications Company, London, pp 147–166.

[10] C. Reis, S. D. Gribble, T. Kohno, and N. C. Weaver. Detecting In-Flight Page Changes with Web Tripwires. In Proceedings of the 5th USENIX Symposium on Networked Systems Design and Implementation (NSDI), San Francisco, CA, Apr. 2008.

[11] Hodges, J., Jackson, C., & Barth, A. (1970, November). *HTTP strict transport security (HSTS)*. RFC Editor. Retrieved February 2023, from https://www.rfc-editor.org/rfc/rfc6797.html

[12] I. Dolnák and J. Litvik, "Introduction to HTTP security headers and implementation of HTTP strict transport security (HSTS) header for HTTPS enforcing," *2017 15th International Conference on Emerging eLearning Technologies and Applications (ICETA)*, Stary Smokovec, Slovakia, 2017, pp. 1-4, doi: 10.1109/ICETA.2017.8102478. https://ieeexplore.ieee.org/abstract/document/8102478

[13] Moonsamy, V., & Batten, L. (2014, December). *Mitigating man-in-the-middle attacks on smartphones – a discussion of SSL pinning and DNSSEC*. Research Online. Retrieved February 2023, from https://ro.ecu.edu.au/ism/165/

[14] Manworren, N., Letwat, J., & Daily, O. (2016, February). *Why you should care about the target data breach*. Business Horizons. Retrieved March 2023, from https://www.sciencedirect.com/science/article/pii/S0007681316000033

[15] H. Berghel, "Equifax and the Latest Round of Identity Theft Roulette," in *Computer*, vol. 50, no. 12, pp. 72-76, December 2017, doi: 10.1109/MC.2017.4451227. https://ieeexplore.ieee.org/abstract/document/8220474

[16] Willett, M. (2021, March 30). *Lessons of the SolarWinds Hack*. Taylor and Francis. Retrieved March 2023, from https://www.tandfonline.com/doi/citedby/10.1080/00396338.2021.1906001?scroll=top&needAccess=true&role=tab

[17] Akhawe, D., Braun, F., Marier, F., & Weinberger, J. (2016, June). *Subresource Integrity*. Subresource integrity. Retrieved March 2023, from https://www.w3.org/TR/SRI/

[18] Menezes, A. J., Oorschot , P. C. van, & Vanstone, S. A. (2001). *Handbook of Applied Cryptography*. CRC Press.

[19] Mozilla. (n.d.). *MIME types (IANA media types) - http: MDN*. HTTP | MDN. Retrieved March 2023, from https://developer.mozilla.org/en-US/docs/Web/HTTP/Basics_of_HTTP/MIME_types

[20] Frederik Braun, F. M. P. on S. 25. (n.d.). *Do not let your CDN betray you: Use subresource integrity – mozilla hacks - the web developer blog*. Mozilla Hacks – the Web developer blog. Retrieved March 2023, from https://hacks.mozilla.org/2015/09/subresource-integrity-in-firefox-43/

[21] N. Sklavos, "Towards to SHA-3 Hashing Standard for Secure Communications: On the Hardware Evaluation Development," in *IEEE Latin America Transactions*, vol. 10, no. 1, pp. 1433-1434, Jan. 2012, doi: 10.1109/TLA.2012.6142498. https://ieeexplore.ieee.org/abstract/document/6142498

[22] Aljawarneh, S., Laing, C., & Vickers, P. (2007). Verification of web content integrity: a new approach to protect servers against tampering.

[23] *What is gzip compression: Enabling gzip commands: CDN guide: Imperva*. Learning Center. (2019, September). Retrieved March 2023, from https://www.imperva.com/learn/performance/gzip/

[24] Singh, K., Wang, H. J., Moshchuk, A., Jackson, C., & Lee, W. (2012, April). *Practical end-to-end web content integrity: Proceedings of the 21st international conference on world wide web*. ACM Other conferences. Retrieved March 2023, from https://dl.acm.org/doi/10.1145/2187836.2187926

[25] Franks, J., Hallam-Baker, P., Hostetler, J., Lawrence, S., Leach, P., Luotonen, A., & Stewart, L. (1999, June). *HTTP Authentication: Basic and Digest Access Authentication*. IETF. Retrieved March 2023, from http://www.ietf.org/rfc/rfc2617.txt

[26] Blumenthal, U., & Goel, P. (2007, January). *RFC 4785: Pre-shared key (PSK) ciphersuites with null encryption for Transport Layer Security (TLS)*. IETF Datatracker. Retrieved March 2023, from https://datatracker.ietf.org/doc/rfc4785/

[27] Siddhesh, Y. (2021, July 20). How I Hacked Your Website and You Didn't Even Know [Information presentation]. RSA Conference, San Francisco, CA, United States.