Defeating MageCart Attacks in a NAISS Way

Cătălin Rus[a], Dipti Kapoor Sarmah[b] and Mohammed El-Hajj[c]
EEMCS/SCS, University of Twente, Drienerlolaan 5, Enschede, Netherlands

Keywords: Image Steganography, E-skimmers, MageCart, Middlebox, Digital Signatures, Network Filter, E-commerce.

Abstract: MageCart attacks pose a security threat to E-commerce platforms by using e-skimmers to steal payment details. Image steganography is used by attackers to conceal e-skimmers, making detection challenging. Existing solutions have limitations, such as incompatibility or insufficient functionality. This research proposes NAISS, a server-side middlebox solution that leverages digital signatures to filter unauthorized images without requiring client-side modifications. The proof-of-concept implementation demonstrates the efficacy of NAISS, filtering 100% of state of the art stegoimages, while indicating areas for further improvement.

1 INTRODUCTION

E-commerce is a rapidly growing market where goods and services are sold over the internet, with increasing numbers of businesses implementing e-commerce capabilities to increase sales and customers opting for online purchases (Zenkina, 2022). However, technical security measures are often overlooked, leaving hosted e-commerce platforms vulnerable to exploits such as the stealing of payment card details by threat group MageCart, which has become more prevalent in recent years (Clapp, 2022). The Federal Bureau of Investigation (FBI) has issued a warning (FBI, 2019) and the Payment Card Industry Data Security Standard has been revised in response to the rising threat (Leyden, 2022). E-skimmers deployed by MageCart are code snippets that instruct clients to forward payment details to a server controlled by attackers and can be hidden inside various HTML tags or images, including stegoimages (Jamil, 1999). Stegoimages produced with novel techniques (e.g. based on Deep Learning) are difficult to combat using steganalysis (Muralidharan et al., 2022), and hence other tools are required to address the threat of malware hiding in images (Wiseman, 2017b).

We searched for literature on tools and ideas that can prevent MageCart attacks, particularly those enabled by stegoimage e-skimmers. Our analysis of current techniques revealed that there are no specific solutions for preventing stegoimage e-skimmers, but some solutions can indirectly address this issue by preventing injection attacks, sanitizing data, enforcing file or code integrity, or blacklisting addresses. However, all of these solutions have limitations such as poor generality, performance, adoption scheme, or functionality under the presence of an attacker. Our research aims to fill these gaps by proposing a practical, efficient, and easily adoptable solution that effectively mitigates the risks posed by stegoimage e-skimmers.

The remainder of the paper is structured as follows: in Section 2, we showcase the solutions found and additional papers to provide insight into already proposed solutions and their shortcomings (2.4); in Section 3, our proposed solution is described and in Section 4, its proof-of-concept implementation and testing details are explained; Section 5 is used to present all the data collected during the testing procedure, while the significance of these findings is discussed in Section 6 together with future directions for research and development. Finally, a summary of our paper's contribution is given in Section 7.

2 RELATED WORK

In this section, we will provide a brief overview of the existing solutions and their limitations. As previously mentioned, there is no all-encompassing solution to the MageCart challenge, so we had to look beyond the existing literature. Our search led us to solutions that include protocols and frameworks suggesting significant changes on both the client and server sides, as

[a] https://orcid.org/0009-0004-2408-3012
[b] https://orcid.org/0000-0002-0802-4280
[c] https://orcid.org/0000-0002-4022-9999

Rus, C., Sarmah, D. and El-Hajj, M.
Defeating MageCart Attacks in a NAISS Way.
DOI: 10.5220/0012079300003555
In *Proceedings of the 20th International Conference on Security and Cryptography (SECRYPT 2023)*, pages 691-697
ISBN: 978-989-758-666-8; ISSN: 2184-7711

well as solutions that suggest modifications on either the client or server side alone.

2.1 Protocols and Frameworks

Several proposed solutions modify both the client and server sides to address MageCart attacks, offering stronger security assurances. Verena (Karapanos et al., 2016) proposes a framework for end-to-end database integrity, where the server provides correctness proofs for queries, and the client filters data based on these proofs while verifying web page integrity. However, correctly using Verena's integrity policies is a challenge, and the framework does not provide confidentiality.

The authors of (Pöhls, 2007) suggests attaching digital signatures to web content's structured elements, but the signatures are located at the end of each element, allowing malicious payloads to be interpreted before the signatures are read. Another proposal (Lim et al., 2022) involves signing each resource with the author's private key and automatically checking the signatures against the author's public key retrieved from a Domanin Name Server (DNS) to prevent content manipulation in transit. However, this solution needs client-side modifications and may cause indirect Denial of Service (DoS) attacks as it does not filter only malicious elements. Additionally, JavaScript code is not checked since the authors suggest solving that by using Subresource Integrity (W3C, 2016), which is rarely used correctly and can be exploited by a malicious server to generate valid hash values for malicious elements

2.2 Server-Side

These solutions propose simpler approaches on the developer or hosting provider's side but have a single point of failure for attackers to exploit.

In (Gebre et al., 2010), the authors propose an upload filter based on regular expressions to prevent content-sniffing Cross-Site Scripting (XSS) attacks. However, the filter needs to keep up with evolving sniffers and client device heterogeneity.

The Content Threat Removal (CTR) (Wiseman, 2017a) fights steganography-based malware by eliminating redundancy in file encoding. CTR is vulnerable to being disabled by a malicious hosting provider and cannot remove dynamically loaded e-skimmers.

(Zuppelli et al., 2021) proposes the use of a neural network to remove malicious payloads from stegoimages created using Invoke-PSImages (github/peepw, 2017). This solution can eliminate harmful payloads in 25ms but does not detect stegoimages and adds a delay to transmission proportional to the number of images.

(Gupta and Gupta, 2016) proposes JS-SAN, which clusters and identifies attack vectors based on templates, achieving an 89% true positive rate for some types of injection attacks. The method has only been tested on two web applications, but the results are promising.

2.3 Client-Side

Server-side solutions are preferred over client-side solutions due to challenges in implementation and effectiveness, as per (Fryer et al., 2015). One Chrome extension proposed in (Bower et al., 2019) uses dynamic analysis to block outgoing requests to attacker domains and prevent JavaScript e-skimmers with a 97.5% detection rate, but is vulnerable to being out-of-date. Tools such as (Aljofey et al., 2022) and (Hiremath, 2021) use classification algorithms and hybrid analysis to detect malicious websites with high accuracy but may not detect malware enabled by certain technologies.

2.4 Shortcomings

We reviewed the literature and found some ideas and solutions that partially address the challenge of stegoimage e-skimmers, but we did not find any works that improve upon the progress made in a way that addresses MageCart attacks. We identified several gaps in the literature, which can be summarized as: 1) Incompatibilities with MageCart attack vectors, specifically images, and limitations in supporting all image formats and malicious payloads; 2) Impracticality of client-side solutions; 3) The vulnerability of server-side solutions to intruders; and 4) Difficulties in keeping up with the pace of state-of-the-art attacks.

3 PROPOSED SOLUTION

Our solution addresses the limitations and drawbacks of existing solutions, as highlighted in 2.4. We provide a simple and compatible approach that ensures integrity even in the event of a compromised hosting provider. Our approach uses cryptographic digital signatures, which we briefly explain before describing the details of our solution. We conclude by evaluating how our solution addresses the research gaps in 2.4.

3.1 Threat Model

To assess our solution's relevance, we consider the limitations and assumptions of the developer, the hosting server, and the attacker. The developer takes responsibility of the security of customers' payment credentials and web page source code (incl. third-party resources), but source code vulnerabilities may remain exploitable by injection attacks. We assume that hosting providers have inadequate internal and external security. Additionally, vulnerabilities in clients' code could allow an attacker to access the physical server and running services, leading to stegoimage e-skimmers being inserted into payment pages. We assume that the attacker has elevated access to the hosting server and uses novel techniques to create difficult-to-detect stegoimages, aiming to modify sites without raising suspicion. However, the attacker is unable or unwilling to perform lateral movement and overtake other nodes in the network - this is left as a future direction to explore.

3.2 Design

Digital signatures, generated by a mathematical algorithm that relies on a private key and the input message, are widely used in e-commerce for message authenticity (Katz, 2010). Our proposed solution, Network Authentication of Images to Stop e-Skimmers (*NAISS*), utilizes digital signatures to tie every image of a website to its developer's private key. *NAISS* also involves a security testing process and a reverse proxy filter to check the images and signatures against the public key of the developer. The generated signatures are added to a new tag in the web page's head, and the filter performs filtering on any GET request. The proposed flow and architecture are shown in Figure 1.

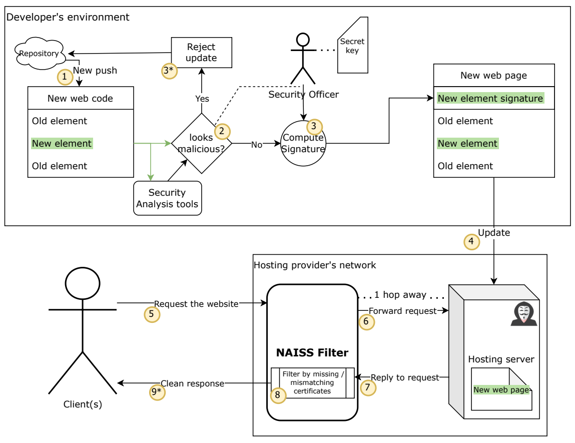


Figure 1: NAISS Flowchart.

3.3 Research Gaps Comparison with *NAISS*

We demonstrate how *NAISS* outperforms related work by addressing the shortcomings outlined in Subsection 2.4: 1) *NAISS* can effectively use digital signatures to ensure integrity for any image format or attack vector, 2) *NAISS* is a server-side solution and requires no client-side modification, increasing its adoptability, 3) even the slightest change made to hosted files or signatures is detected by the unique property of digital signatures and 4) *NAISS* is able to detect attacks that rely on the modification of already-pushed data, making it highly effective against novel attacks.

4 METHODS

In this section, we will outline the technical details, materials, and algorithms used to implement *NAISS*. The main programming language used was Python, and Docker was used to build the test environment. The hosting server was a basic Python HTTP server, and the *NAISS* filter was built using Flask. HTTP is used to access the server instead of HTTPS for ease of development and testing, although upgrading to HTTPS is easily achievable. During testing, the hosting server and filter containers were run on the same Docker network, while the automated tests were run outside the Docker network to emulate a real client. The source code for *NAISS* is available on GitHub (Rus, 2023), where more instructions can be found. The code is organized into four directories: *client*, *filter*, *server*, and *utils*. The *client* directory contains scripts for running automated tests and visualizing their results, while the *filter* directory contains the scripts and the Dockerfile needed to launch the *NAISS* filter container. The *server* directory contains various website variants and the Dockerfile for the hosting server container. Finally, the *utils* directory contains useful scripts for developers/attackers to attach signatures to websites, as well as scripts used to generate websites.

We employed python-ecdsa (github/tlsfuzzer, 2010) with the NIST256p curve (Adalier and Teknik, 2015) to generate and authenticate the signatures and their corresponding key. This curve was preferred due to its practical key size and better computational performance. As input for the signatures, the byte string of images stored locally were used, while for images stored on external sources, their URL access was used. As mentioned in Subsection 3.1, the content of third party resources falls under the developer's responsibility.

To verify the effectiveness of our proposed approach against advanced stegoimage techniques, we decided to use SteganoGAN (Zhang et al., 2019) to create our own stegoimages. It uses Generative Adversarial Networks for embedding payloads, which is considered to be the most challenging method to detect (Muralidharan et al., 2022). SteganoGAN can embed data of any size with an internal cutting point of 4 bits per pixel.

We created various website versions using a single template that represents a standard e-commerce website where customers enter their credentials. The websites feature several pictures of different sizes placed at different locations, which serve as attack vectors for stegoimage e-skimmers. The images used have sizes of 128x80 pixels for payment icons, 256x256 for favicons, and 505x446 for the website logo. An example of the website with all its images, except the favicon, can be seen in Figure 2.

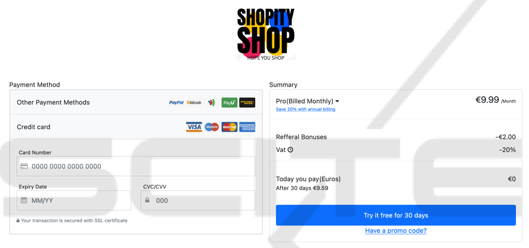


Figure 2: Fictitious e-commerce platform.

We aimed to improve test reproducibility and execution time by automating website access and data collection using Selenium Webdrivers. We also used selenium-wire (Keeling, 2018) to inspect network requests and responses. To ensure consistent results, tests were run on the same MacBook Pro M1 with 8GB of RAM, with no unnecessary background processes running and a battery level above 80%.

5 RESULTS

In this section, we present the testing variables, the collected data and the aggregated results. Moreover, we showcase how the change of one type of parameter affects the performance or behaviour of *NAISS*.

5.1 Test Parameters

We created a test setting to study the behavior of the *NAISS* filter under various conditions. To achieve this, we tuned several parameters such as the image format (PNG/JPEG), image loading method (internal/external links), embedded payload (none/stegoimages), and the type of signatures (none/developer's/attacker's). We also studied the impact of using different web browsers (Chrome, Firefox, Edge) and whether the test was run through the filter or not. We conducted 54 tests with all combinations of these parameters.

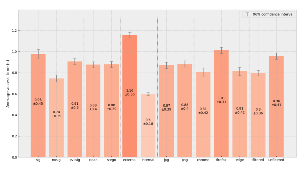
5.2 Measurements

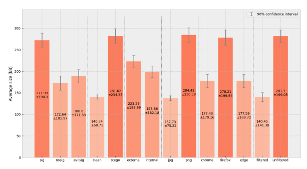
We gather data for each of the 54 test cases to assess the performance and behaviour of the *NAISS* filter, and also to support experimentation with specific parameters (e.g., encryption curve). We collect the following information for each test: (1) time taken to access the website in seconds, (2) amount of data transferred in kilobytes, and (3) percentage of images that load for the client. The third measurement helps us determine if the *NAISS* filter is working properly, while the first two measurements are used for performance evaluation.

5.3 Baseline Results

The measurements collected were used to generate results, starting with a baseline experiment using image of certain sizes, NIST256p curve, and the payload "RENAISSANCE". Further experiments explored the effects of increasing these parameters on performance and behavior. The results were grouped by the parameters in 5.1 and averaged. For each measurement in 5.2, a plot was produced: Figure 3(a) for access time, Figure 3(b) for transferred data, and Figure 3(c) for the percentage of unfiltered images based on the presence of the correct signature. To provide statistically viable results, we ran multiple iterations and found that the averages and standard deviation stabilizes from 5 repetitions onward. To this extent, all the measurements were taken 10 times and a 96% confidence interval alongside the standard deviation are provided.

The method of filtering images based on attached signatures is confirmed to be working as intended, as shown by the 100% arrival rate for images with the correct signatures. The 50% arrival rate of tests with incorrect signatures is also expected since half of the tests run through an unfiltered connection. The performance results indicate a connection between transferred data and access time, with an insignificant increase in access time expected with an increase in transferred data. Websites with no signatures have the lowest values, while those with a correct signature have the highest, as expected. Although stegoim-


(a) Access time per parameter


(b) Transferred data per parameter

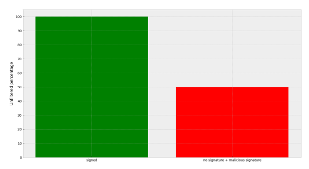

(c) Percentage of images reaching the client

Figure 3: Baseline results.

ages are close to double the size of clean images, the time difference in accessing them is orders of magnitude smaller. Chromium-based browsers are faster and transfer less data, with Chrome being the best performer. Websites accessed through the *NAISS* filter are transferred close to 50% less data and 20% faster than those accessed directly by the client.

5.4 Experiments

We conducted experiments to study the impact of changing individual parameters compared to the baseline. These parameters were selected based on their relevance to optimize performance and behavior for e-commerce website developers or *NAISS* implementers. We also ran an experiment with all changes simultaneously (Figure 4). Results can be found on separate branches on GitHub for uniform navigation across all experiments.

In the first experiment, the pixel size of all the used images is doubled. The results show the same topology as for the baseline experiment, except a 3% access time increase for the stegoimages category. The values are overall higher, especially for transferred data. This experiment further contributes to the idea that transferred data only slightly increases the time taken to load a web page.

In the second experiment, the stegoimage payload is changed to a realistic one. Through this, we emulate how *NAISS* would react to real e-skimmers hidden in images. As a realistic payload, we used an e-skimmer script (4,5 kilobytes) caught in the wild (MalwareBazaar, 2022). By increasing the size of the payload, we also slightly increase the storage size of the stegoimages. The results are similar to the image size experiment, with all values increasing by up to 11% compared to the baseline.

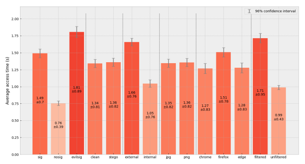
In the last experiment, we changed the elliptic curve used to NIST512p, increasing both the storage size of attached signatures and the time to verify a signature. The results are topologically different in the signature and connection type categories. The websites signed with an attacker key now have the longest access time. The other significant change is observed in the websites that are accessed through a *NAISS* filter, where the access time has more than doubled and transferred data increased by approximately 40% compared to the baseline results. One interesting finding is that the "evilsig" category has the largest latency, hinting that verifying an invalid signature is a slower process than verifying a valid one.

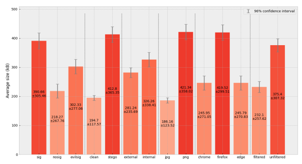
6 DISCUSSION

This section explains the results and implications of the experiment, using the payload experiment as a representative of a real-world case, mostly because of the stegoimage payload content. The *NAISS* filter effectively blocks missing or malicious images, resulting in faster access times for the client. The difference in access time between websites with and without signatures is minimal, with the filtering process being the only contributing factor. The study found that the filter can either reduce or add to the load time due to signature validation, which may slow down access time for websites with malicious signatures - something exploitable by an attacker. The factors that induce the highest latency are in order: the signature scheme, loading external images, and the use of Firefox. The former factors can be modified by developers to improve the filter's performance without compromising security.

6.1 Benchmark

Table 1 presents a comparison benchmark that evaluates the performance, behavior, robustness, and ease of adoption of various solutions, including *NAISS*. The criteria used in the evaluation are the imposed latency percentage (1), true positive rate (2), flexibility (3), adoptability (4), bypassability by intruder (5), and potential to cause indirect DoS (6). Flexibility (3) is the capability of functioning correctly with varied inputs and contexts, including novel attacks and adoptability is the ease of adoption and use from the servers' and, but especially, clients' side. The benchmark assesses whether a solution is applicable or as-


(a) Access time per parameter


(b) Transferred data per parameter

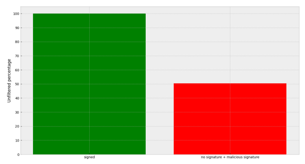

(c) Percentage of images reaching the client

Figure 4: All parameters experiment results.

sessable for each criterion and assigns a corresponding color-coded symbol, such as gray with a ╱ for not applicable, green with a △ for a strong point, red with a ▽ for a weak point, and yellow with a ◊ for neither strong nor weak.

Table 1: Comparison between solutions identified in the literature and *NAISS*.

Solution	(1)	(2)	(3)	(4)	(5)	(6)
(Pöhls, 2007)	╱	╱	△	▽	△	△
(Gebre et al., 2010)	╱	100%	▽	▽	▽	△
(Karapanos et al., 2016)	╱	100%	◊	▽	△	△
(Gupta and Gupta, 2016)	╱	89%	◊	△	▽	△
(W3C, 2016)	╱	╱	△	△	▽	△
(Wiseman, 2017a)	╱	100%	◊	△	▽	△
(Bower et al., 2019)	12%	98%	▽	▽	△	△
(Zuppelli et al., 2021)	╱	81%	▽	△	▽	△
(Hiremath, 2021)	╱	99%	◊	▽	△	△
(Aljofey et al., 2022)	╱	94%	▽	▽	△	▽
(Lim et al., 2022)	10%	100%	△	▽	△	▽
NAISS (Rus, 2023)	**10%**	**100%**	▲	▲	▲	▲

To summarize, implementing *NAISS* provides several benefits, including logically sound filtering based on content validation, a 100% true positive rate, agnosticism to novel image steganography techniques, no need for client-side modifications, minimal server-side modifications, the ability to verify image authenticity on the client-side through the Public Key Infrastructure (PKI), and the prevention of DoS attacks if signatures are altered.

6.2 Limitations

When reviewing our work, readers have to be aware of the limitations present within our implementation and testing procedure. Addressing these limitations shall yield an improvement in the proposed solution. The implementation limitations of our proof-of-concept may affect its practical usefulness in a real-world e-commerce platform. These limitations include the fact that the favicon is loaded but not displayed when accessing a website through *NAISS*, and that communication is done through simple HTTP, although upgrading to HTTPS is easily achievable. Additionally, an attacker can slow down a website's loading time by adding multiple incorrect signatures. Finally, the current implementation only supports images, although it is technically feasible to extend it to cover all types of MageCart attack vectors. The limitations regarding testing might introduce biases or conceal them in our results. Firstly, Scalable Vector Graphics (SVG) images were not included due to SteganoGAN incompatibility. Secondly, the ICO images could not be loaded from external sources due to incompatibility with online hosting services like imgur. Thirdly, data collection issues prevented the inclusion of the Safari browser in our study. Finally, we did not test any mobile browsers.

6.3 Future Directions

Our analysis shows that *NAISS* can be industry-ready with improvements. Future research should expand testing to more MageCart attack vectors, develop a integrated process for validating web elements, upgrade the server's connection to HTTPS, modify the filtering algorithm for addressing slowdown attacks, improve access time with faster programming languages and algorithms, study prevalent MageCart groups, and explore methods for better protecting the reverse proxy from intruders.

7 CONCLUSION

This research exposes the threat of stegoimage e-skimmers to e-commerce and the drawbacks of current solutions. To tackle these issues, a server-side solution called NAISS has been proposed, utilizing digital signatures to prevent unauthorized image transmission to clients. The results demonstrate NAISS's efficacy in filtering stegoimages with minimal loading time impact. However, NAISS may still encounter security and deployment challenges, necessitating further research to assess its scalability and practicality in real-world settings and enhance its design.

REFERENCES

Adalier, M. and Teknik, A. (2015). Efficient and secure elliptic curve cryptography implementation of curve p-256. In *Workshop on elliptic curve cryptography standards*, volume 66, pages 2014–2017.

Aljofey, A., Jiang, Q., Rasool, A., Chen, H., Liu, W., Qu, Q., and Wang, Y. (2022). An effective detection approach for phishing websites using url and html features. *Scientific Reports*, 12(1):1–19.

Bower, T., Maffeis, S., and Demetriou, S. (2019). Identifying javascript skimmers on high-value websites. *Imperial College of Science, Technology and Medicine, Imperial College London*, pages 1–72.

Clapp, K. (2022). Commodity skimming & magecart trends in first quarter of 2022. https://community.riskiq.com/article/017cf2e6.

FBI (2019). Oregon fbi tech tuesday: Building a digital defense against e-skimming. https://www.fbi.gov/contact-us/field-offices/portland/news/press-releases/oregon-fbi-tech-tuesday-building-a-digital-defense-agaist-e-skimming.

Fryer, H., Stalla-Bourdillon, S., and Chown, T. (2015). Malicious web pages: What if hosting providers could actually do something. *Computer Law & Security Review*, 31(4):490–505.

Gebre, M. T., Lhee, K.-S., and Hong, M. (2010). A robust defense against content-sniffing xss attacks. In *6th International Conference on Digital Content, Multimedia Technology and its Applications*, pages 315–320.

github/peepw (2017). GitHub - peewpw/Invoke-PSImage: Encodes a PowerShell script in the pixels of a PNG file and generates a oneliner to execute — github.com. https://github.com/peewpw/Invoke-PSImage.

github/tlsfuzzer (2010). GitHub - tlsfuzzer/python-ecdsa: pure-python ECDSA signature/verification and ECDH key agreement — github.com. https://github.com/tlsfuzzer/python-ecdsa.

Gupta, S. and Gupta, B. B. (2016). Js-san: defense mechanism for html5-based web applications against javascript code injection vulnerabilities. *Security and Communication Networks*, 9(11):1477–1495.

Hiremath, P. N. (2021). *A novel approach for analyzing and classifying malicious web pages*. PhD thesis, University of Dayton.

Jamil, T. (1999). Steganography: the art of hiding information in plain sight. *IEEE potentials*, 18(1):10–12.

Karapanos, N., Filios, A., Popa, R. A., and Capkun, S. (2016). Verena: End-to-end integrity protection for web applications. In *2016 IEEE Symposium on Security and Privacy (SP)*, pages 895–913. IEEE.

Katz, J. (2010). *Digital signatures*, volume 1. Springer.

Keeling, W. (2018). GitHub - wkeeling/selenium-wire: Extends Selenium's Python bindings to give you the ability to inspect requests made by the browser. — github.com. https://github.com/wkeeling/selenium-wire.

Leyden, J. (2022). Credit card industry standard revised to repel card-skimmer attacks. https://portswigger.net/daily-swig/credit-card-industry-standard-revised-to-repel-card-skimmer-attacks.

Lim, Z. X., Ho, X. Q., Tan, D. Z., and Goh, W. (2022). Ensuring web integrity through content delivery networks. In *2022 IEEE World AI IoT Congress (AIIoT)*, pages 494–500. IEEE.

MalwareBazaar (2022). Malwarebazaar magecart sample. https://bazaar.abuse.ch/sample/f9274347590156c3e86e\\b7015b6dbd3587de034c51cb52e5161cee671c1107e4/.

Muralidharan, Trivikram, Aviad, Cohen, Assaf, Nissim, and Nir (2022). The infinite race between steganography and steganalysis in images. *Signal Processing*, page 108711.

Pöhls, H. C. (2007). Authenticity and revocation of web content using signed microformats and pki.

Rus, C. (2023). GitHub - ruscatalin/NAISS: Research Project for NAISS: Network Authentication of Images to Stop e-Skimmers — github.com. https://github.com/ruscatalin/NAISS.

W3C (2016). Subresource integrity. https://www.w3.org/TR/SRI/.

Wiseman, S. (2017a). Content security through transformation. *Computer Fraud & Security*, 2017(9):5–10.

Wiseman, S. (2017b). Stegware–using steganography for malicious purposes.

Zenkina, E. (2022). About current trends in global e-commerce. *BENEFICIUM* , (1):68–73.

Zhang, K. A., Cuesta-Infante, A., and Veeramachaneni, K. (2019). Steganogan: High capacity image steganography with gans. *arXiv preprint arXiv:1901.03892*.

Zuppelli, M., Manco, G., Caviglione, L., and Guarascio, M. (2021). Sanitization of images containing stegomalware via machine learning approaches. In *ITASEC*, pages 374–386.