Online-Journals.org

iTDAF IETI Transactions on Data Analysis and Forecasting

iTDAF | eISSN: 2959-0442 | Vol. 2 No. 2 (2024) | OPEN ACCESS

https://doi.org/10.3991/itdaf.v2i2.51015

PAPER

# Review Analysis of Web Socket Security: Case Study

Malik Muhammad Nadeem[1], Yousaf Raza[1], Ahthasham Sajid[1](✉), Hamza Razzaq[1], Rida Malik[1], Sugandima Vidanagamachchi[2]

[1]Department of Cyber Security, Riphah Institute of Systems Engineering, Riphah International University, Islamabad, Pakistan

[2]Department of Computer Science, Faculty of Science, University of Ruhuna, Matara, Sri Lanka

ahthasham.sajid@riphah.edu.pk

**ABSTRACT**

Web sockets (WS) have revolutionized real-time online communication by enabling two-way communication channels using a single transmission control protocol (TCP) connection, significantly enhancing the user experience in web applications. However, this advancement has also presented certain security challenges that need to be addressed in order to ensure secure and reliable communication. This review paper delves into the security aspects of WSs, analyzing and contrasting various tactics and methodologies proposed for securing WS connections. By conducting a thorough analysis of notable study contributions dating back to 2015, we have found common vulnerabilities and risks, such as cross-site scripting (XSS), cross-site web socket hijacking (CSWSH), and man-in-the-middle (MITM) attacks. This paper evaluates the effectiveness of several security measures, including confirmation, encryption, and different anomaly detection algorithms. Further, this study scrutinizes the deficiencies and constraints that have been shown in these study initiatives, placing emphasis on areas that require further examination. The main objective of our comprehensive examination is to build a robust foundation for future studies on WS security, promoting the development of more resilient and impervious live communication networks.

**KEYWORDS**

XXS, WS, API, DOS, TCP

## 1 INTRODUCTION

In 2011, web socket (WS) was introduced, and it provides a two-way communication channel over a single WS connection.WS protocol was standardized by the internet engineering task force (IETF) as request for comment (RFC) 6455, and the WS application program interface (API) is standardized by the world wide web consortium (W3C). WS provides a way of creating a persistent, low-latency connection [12], which is capable of handling transactions initiated by either the client or the server, thus being full-duplex and support both data dispatch and data fetching [12].

Nadeem, M.M., Raza, Y., Sajid, A., Razzaq, H., Malik, R., Vidanagamachchi, S. (2024). Review Analysis of Web Socket Security: Case Study. *IETI Transactions on Data Analysis and Forecasting (iTDAF)*, 2(2), pp. 56–75. https://doi.org/10.3991/itdaf.v2i2.51015

Article submitted 2024-06-09. Revision uploaded 2024-08-02. Final acceptance 2024-08-02.

## 1.1 Overview of web socket

The WS is a revolutionary technology that greatly enhances real-time communication on the Internet. WS differs from typical hypertext transfer protocol (HTTP) requests by establishing a persistent connection between the web browsers and the servers, allowing for bi-directional, real-time data exchange without the need for constant page refresh. Two-way communication enables capabilities such as live chat and collaborative editing, allowing both the server and browser to initiate data transmission. The WS offers a reduced time delay, resulting in a prompter and more interactive user experience. So, in addition to web browsers and servers, its versatility also applies to a wide range of application environments that require instantaneous data exchange. The WS facilitates the development of a more dynamic and interactive online environment by enabling features that depend on uninterrupted data transfer.

The WS protocol was officially defined by the IETF in 2011, specifically as RFC 6455. Major accomplishments in the widespread deployment of WS technology include:

- In December 2009, Google Chrome 4 became the first browser to include complete WS capability.
- In 2011, Ian Fete issued RFC 6455, which finalized the WS protocol.
- In 2011, the responsibility for standardizing Web Socket's was transferred from the W3C and WHATWG to the IETF [4].

## 1.2 Historical revolution in web socket security

The historical growth of WS security demonstrates the dynamic interaction of technological improvements and increasing threats. WS usage outpaced security concerns, resulting in vulnerabilities including XSS and data interception. However, as the technology advanced, developers and academics concentrated on resolving these flaws. This progression saw the emergence of strong security mechanisms and standards designed expressly for WS communication. Moreover, advances in encryption algorithms and authentication processes improved the integrity and secrecy of the WS. Collaborative initiatives within the cyber security industry, as well as regulatory frameworks, all helped to improve the WS security standards. Today, WS security demonstrates the proactive efforts used to reduce risks and ensure the secure and dependable functioning of real-time online applications. The timeline below reveals when a new security feature was added to the WS technology. Figure 1 below gives an overview of web sockets.

**2010:** Introduction of web socket protocol (RFC 6455) [31]
**2012:** Implementation of secure web socket (WSS) [31]
**2013:** Cross-origin resource sharing (CORS) [31]
**2015:** Enhanced handshake mechanisms [31]
**2016:** Advanced encryption techniques [31]
**2018:** Multi-factor authentication (MFA) integration [31]

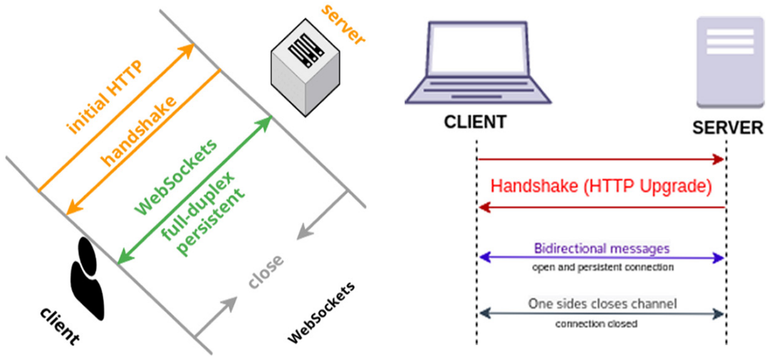


**Fig. 1.** Historical review of web socket [31]

The WS facilitates instantaneous, bidirectional communication between the web browser and server. It uses a solitary, enduring TCP connection for effective data interchange. Although often known as the HTML5 W-API, which is actually an independent standard that is currently being developed [3] and [6], it has not been officially standardized, so many web browsers already implement the feature. As WS is on the verge of transforming web communication, its security implications have become extremely important. This study examines the security risks associated with the WS protocol and API, despite their recent introduction and minimal public study. It is also investigating the security measures provided by this protocol [1] and [6]. Figure 2 below illustrates the basic communication between a web client and remote services using the HTTP protocol.

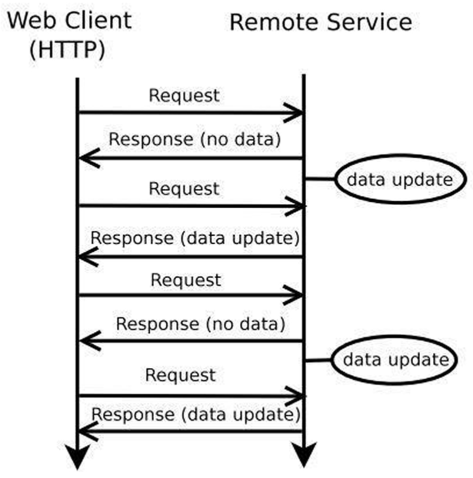


**Fig. 2.** Fetching real-time data with HTTP and web socket [1]

The WS protocol, originally intended for web browser and servers but also suitable for other applications, operates over TCP to provide rapid communication. The rising in popularity of WS technology has significant implications for web security in the coming years [2] and [5].

## 1.3 Importance of web socket

The WS allows two-way, real-time communication between web browser and server, which is essential to contemporary web applications. Instant data sharing is made possible by this, eliminating the requirement for frequent page refreshes. Moreover, WS reduces network overhead over standard HTTP queries by using a single, persistent TCP for effective data transmission. This type of adaptability is useful for a variety of real-time data interaction requirements that go beyond the web browser and server [7] and [12]. By providing capabilities such as live chat, collaborative editing, and a real-time data dashboard, WS ultimately improves the user experience and increases user engagement and response. Over conventional web communication, WS provides a potent boost. Full-duplex, two-way data exchange is made possible by the long-lasting, low-delay link it creates. Thus, real-time data pushing and pulling is made possible by the fact that both the web server and the browser can start data transfer [8]. As all of the main browsers now support this widely used technology, it is an essential tool for creating dynamic, interactive web applications that need real-time functionality.

## 2 SECURITY CHALLENGES OF WEB SOCKET

The security issues brought on by WS include XSS, cross-origin web socket hijacking (COWH), weak authentication, denial of service (DoS) attacks, message tampering, MITM attacks, and client impersonation. Strong authentication, encryption, and validation protocols, together with frequent security audits, can help reduce these threats. When WS is being used, it's common to ignore the recommended security procedures [10] and [11]. The WS handshake by itself does not naturally manage authentication. Although cookies or other techniques may be used in standard HTTP ways, the WS protocol implements authentication processes at the application layer. If not done, this can make the connection open to unwanted access. WS uses a permanent connection hence, hostile actors on the network may intercept or alter the data transferred [15]. Data confidentiality and integrity are therefore made essential by encryption. Data compromise is not at all likely because the program has no known security vulnerabilities, and any possible assaults would be contained [19]. Attackers might take over a web connection that a genuine user started by taking advantage of flaws in the handshake procedure. This enables them to pass for the user and obtain unapproved access to features or data. WS connections are prone to DoS attacks because of their persistent character, possible XSS site flaws, and private data exposure brought on by insufficient message and WS data neutralization [8] and [9]. A server may be overrun by attackers' connection requests, taxing its resources and preventing authorized users from connecting. These types of security issues emphasize the need to combine WS with strong security measures. These hazards can be reduced, and safe real-time communication can be guaranteed by encryption, appropriate authentication, and rate limiting.

Assuring that authenticated users have the rights required to access resources inside a system is known as authorization [17]. Usually, the role of the user inside the service determines their access to resources. A fundamental guideline in authorization, the principle of least privilege urges users to restrict access to resources necessary for their designated duties, therefore reducing the possibility of illegal access [16].

The WS commonly uses transport layer security (TLS) to establish a secure connection and encrypt data. Encryption fosters confidence, strengthens security, and may be obligatory for confidential information in regulated sectors [23]. Figure 3 illustrates secure communication vs. unsecure communication using HTTP and HTTPS.

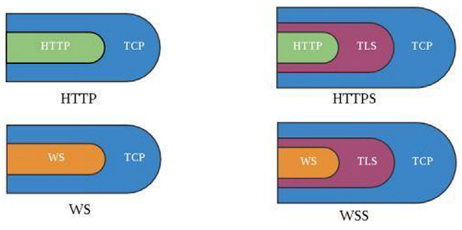


**Fig. 3.** Secure connection and unsecure connection in web socket [23]

Encryption keeps data private by restricting access to only the people who are supposed to. Attackers running the danger of MITM attacks can intercept and read communications without encryption, as illustrated further in Figure 4. Data is insecure because default WS usage on port 80 is not encrypted [23] and [24]. To protect data from MITM attacks, port 443 with SSL encryption is used.

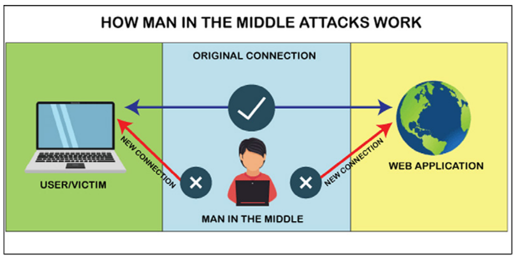


**Fig. 4.** Man-in the-middle attack [23]

Authorization in WS apps is mostly dependent on the particular application context. Attackers could, for example, read restricted material belonging to other users, use the WS service without any kind of authorization, or access functionalities needing higher-level authorization than first allocated [16] and [22]. Authorization using WS is mostly application-context-dependent. Three situations are conceivable, as illustrated in Figure 5. Further:

- An attacker can utilize a WS service function that needs higher-level authorization than the user was first granted.
- A function displaying other users' restricted material is open to an attacker.
- Without any authorization, an attacker can access the WS service.

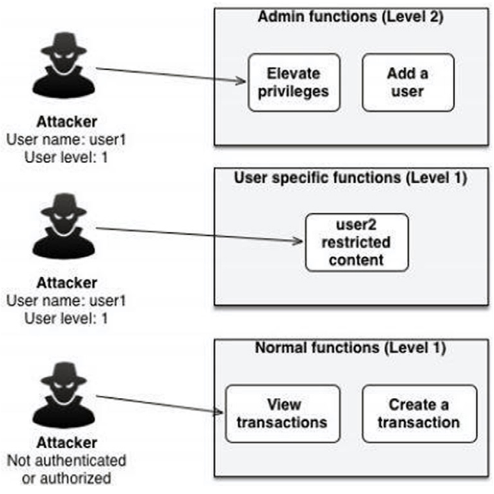


**Fig. 5.** Illustration of the listed attack scenarios [16]

### 2.1 Rational of conducting review on web sockets 2024

The rising dependency on WS technology for real-time, two-way communication in web applications needs a thorough analysis of the security implications. The WSs, although providing substantial advantages over typical HTTP polling and AJAX techniques, have unique security issues due to their permanent connection. As WSs are widely used in vital applications such as financial services, healthcare, and collective tools, it is very critical to understand their possible vulnerabilities and implement strong security methods. A thorough assessment of WS security is required to detect and mitigate vulnerabilities such as MITM attacks, unauthorized access, and data breaches [20]. Previous study has identified certain risks and recommended numerous solutions, but a comprehensive comparison of these strategies and an assessment of their efficacy remain unexplored. This study seeks to offer a complete overview of the current status of WS security by testing important study articles released since 2008, identifying prevalent flaws, and assessing the effectiveness of suggested fixes. This initiative will added to the academic body of knowledge while also providing practical insights for developers and security experts working to protect WS-enabled apps.

**Objectives of review 2024.** The review paper analyzes the security techniques and vulnerabilities of WSs in key study papers, highlighting the pros and cons.

**Structure of review**

This type of study utilizes a methodical framework to explore the security aspects of the WS. The abstract provides a concise overview of the study's extent and goals. The introduction offers a backdrop and delineates the importance of the WS security. The detailed analysis of existing studies and methodology is presented in the following comprehensive literature review. A comparative analysis is conducted to assess different security measures and provide valuable insights for improving WS security standards.

## 3 LITERATURE REVIEW

In order to select literature for a WSs review paper, the authors in this study access the following standards: topical relevance, citations, methodological relevance, time frame relevance, impact, and diversity of perspectives, as illustrated in Figure 6 below. Make sure that sources are recent, relevant to the subject matter, employ appropriate methodologies, have a significant impact, are well-cited, and provide a variety of perspectives.

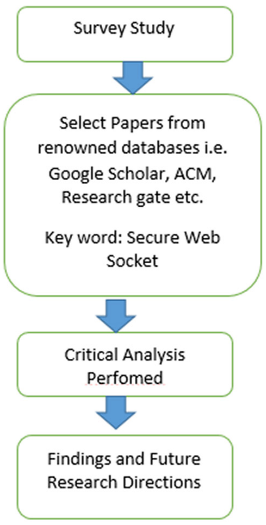

Survey Study
Select Papers from renowned databases i.e. Google Scholar, ACM, Research gate etc.
Key word: Secure Web Socket
Critical Analysis Perfomed
Findings and Future Research Directions

**Fig. 6.** Survey flow

The WS is susceptible to security vulnerabilities that are typical of other online technologies, underscoring the significance of installing suitable security measures. Although some security controls that are relevant to general web technologies may also be used for the WS, it is the duty of developers or service administrators to apply these controls. Regrettably, when humans are responsible for safeguarding systems, there is a risk of overlooking or improperly implementing security controls [13] and [18]. This issue may arise due to developers' inadequate understanding of appropriate implementation techniques or time limitations. As a result, if security rules are not built or performed correctly, WS applications can be easily exploited. There are three techniques that are defined.

The WS encryption employs cryptographic algorithms to obfuscate data using confidential keys, rendering it indecipherable to any unauthorized individuals intercepting the network traffic. This guarantees confidentiality by ensuring that only authorized parties, such as the server and browser, are able to comprehend the information that is being transferred. Integrity refers to the state of data remaining unchanged during transmission, thereby preventing any unauthorized modifications.

In any service to confirm the validity of a user. Still, WS by itself makes no demands about the kind of authentication. Frequently, developers use the "set-cookie" header method, giving the user the final say [14] and [19]. Permission-based access control to resources and user data is made possible in part via authentication.

**Table 1.** Critical analysis

| Technique | Description | Benefits | Risks | Implementation |
|---|---|---|---|---|
| Encryption | Makes data unintelligible during transmission by jumbling it with secret keys. | • The material is understood only by those with the necessary authorization.<br>• Integrity is the untouched data.<br>• Increases security and fosters trust.<br>• Maybe required for private data. | • The default Web Socket (port 80) is not encrypted, making it open to MITM attacks.). | • All Constructs a secure connection and encrypts data using Transport Layer Security (TLS/SSL). TLS implementation is made easier using several modules and frameworks |
| Authentication | Whether a user is legitimate to use the Web Socket service. manages, according to rights, user data and resource access. | • Controls access to resources and user data based on permissions. | • No built-in method in Web Socket.<br>• Developers rely on approaches like "set-cookie" header, leaving room for vulnerabilities. | • Developer-defined: Utilize application-specific methods like custom headers or tokens. |
| Authorization | Ensures authenticated users have the necessary privileges to access specific resources within the system. | • Minimizes risks of unauthorized access based on the Principle of Least Privilege. | • Heavily relies on application context.<br>• Attackers could potentially gain access to unauthorized functions, data, or the entire service. | • Context-dependent: Developers define access levels based on user roles and permissions. |

## 4 CRITICAL ANALYSIS

In order to guarantee the confidentiality, integrity, and controlled access of data transmitted between clients and servers, WS technology is dependent on a number of critical security mechanisms. Encryption, which is predominantly based on TLS, is responsible for the encryption of data, rendering it indecipherable to unauthorized entities, as illustrated in Table 1. This encryption process guarantees that sensitive information is safeguarded from potential attacks, including MITM interceptions. Although authentication is not expressly required by WS standards, it is essential for legitimately verifying the identity of users who access the service. Typically, developers employ methods such as the "set-cookie" preamble to implement authentication, which allows for greater flexibility in user validation. In contrast, authorization is responsible for determining the level of access that authenticated users are granted, which is typically determined by their duties and permissions within the system. The risk of unauthorized data exposure or misconduct is mitigated by ensuring that users only access resources necessary for their designated tasks by adhering to the Principle of Least Privilege. WS applications may be susceptible to exploitation if these security mechanisms are not implemented effectively, which could result in data breaches and compromise. Therefore, it is imperative to comprehend and effectively implement encryption, authentication, and authorization in order to enhance the security of WSs against the ever-changing nature of cyber threats.

## 4.1 Vulnerability mitigation

Integrity and security of client-server communication channels depend on WS technology vulnerabilities being mitigated. WS applications run the danger of XSS, COWH, and authentication method flaws. With any luck, this tool will be helpful and combat XSS, [25]. Effective mitigation of these weaknesses calls for a multi-dimensional strategy. First off, preventing XSS attacks—which can jeopardize the integrity of data transmitted across WS connections—needs the implementation of strong input validation and sanitization systems. Effective user input validation and sanitization help developers reduce the possibility of unwanted data modification and malicious script insertion. This is made feasible by WS, which enables distant servers to send inexpensive heartbeat queries to many websites [20]. Second, by using protocols such as WSS to secure WS connections, one may reduce the possibility of COWH assaults, in which criminals try to take over WS connections in order to capture private data. Through SSL/TLS protocol encryption of WS communication, businesses can guarantee that data transmitted between clients and servers is private and unaffected by manipulation or listening in. Because SSL's constituent protocols are modular, future expansions will be simple to integrate. [21] and [26]. Moreover, it is imperative to improve authentication and permission procedures in order to manage WS resource access and stop unwanted access. Putting robust authentication techniques into practice, including OAuth or token-based authentication, may help confirm user identities and guarantee that only authorized users can use WS services. Further limiting access to sensitive resources and preventing illegal activities inside WS applications is the enforcement of fine-grained authorization restrictions based on user roles and privileges. Additionally, crucial elements of vulnerability mitigation plan for WS apps include proactive monitoring, the vulnerability assessments, and regular security audits. Organizations can spot and fix any dangers before malevolent actors take advantage of them by routinely evaluating and fixing security flaws. According to the test findings, the issue stems mostly from susceptible programs [28]. Finally, developing a culture of security consciousness inside organizations requires raising developers and administrators' knowledge of best practices in WS security. Organizations may improve their defenses against changing threats and protect WS-based apps from possible vulnerabilities by giving teams the knowledge and ability to properly detect and reduce security concerns.

## 5 SECURITY FRAMEWORK'S AVAILABLE

With the help of several security frameworks, developers may improve the security of WS-based apps by giving them access to tools and recommendations. Notable WS security frameworks include

### 5.1 The OWASP web socket security cheat sheet

Provides comprehensive guidance and recommendations for ensuring the security of WS communication. In order to help secure WS implementations, the Open Web Application Security Project (OWASP) created this handy cheat sheet. This document encompasses several facets of WS security, encompassing authentication, authorization, encryption, and safeguarding against prevalent vulnerabilities such as XSS and COWH. This section outlines a standard testing methodology that may be created inside an organization [27].

### 5.2 Socket.IO security

Socket.IO is a widely used JavaScript library that enables real-time online applications, including support for WS. The socket.IO security documentation offers guidance and insights on how to secure socket.IO-based applications. It covers several aspects, such as message validation, authentication, and authorization.

### 5.3 Spring security

Provides extensive support for safeguarding WS communication for developers using the Spring Framework in Java applications. The Spring Security manual provides instructions and illustrations on how to configure authentication, authorization, and encryption in applications that use WS technology. The process involves the conversion of a sequence of bytes into distinct frames. A frame has the capability to hold either text or a binary message [28].

### 5.4 Securing SignalR with ASP.NET core

SignalR is a library for ASP.NET Core applications that enables real-time communication. It supports WS as one of its transport protocols. The SignalR security literature offers recommendations on ensuring the security of SignalR connections, encompassing authentication, authorization, and safeguarding against prevalent security vulnerabilities.

### 5.5 WAMP, which stands for web socket application messaging protocol

It is an open standard that enables real-time communication over WS connections. It is commonly used alongside frameworks such as Autobahn, Python, and Crossbar.io. The WAMP standard incorporates measures for authentication, authorization, and encryption, allowing developers to construct secure applications based on WS. The WAMP utilizes a central router to facilitate the routing of messages between components [29] and [30]. It facilitates two communications patterns: Publish & Subscribe (PubSub) is a mechanism that enables components to send messages to channels and subscribe to receive messages that are relevant to them.

### 5.6 Routed remote procedure calls (rRPC)

It allows components to execute remote procedures hosted by other components, allowing distributed communication across WS connections. These frameworks provide developers with significant tools, documentation, and code samples to assist them in implementing strong security mechanisms in WS applications. This helps to guarantee the confidentiality, integrity, and availability of real-time communication channels.

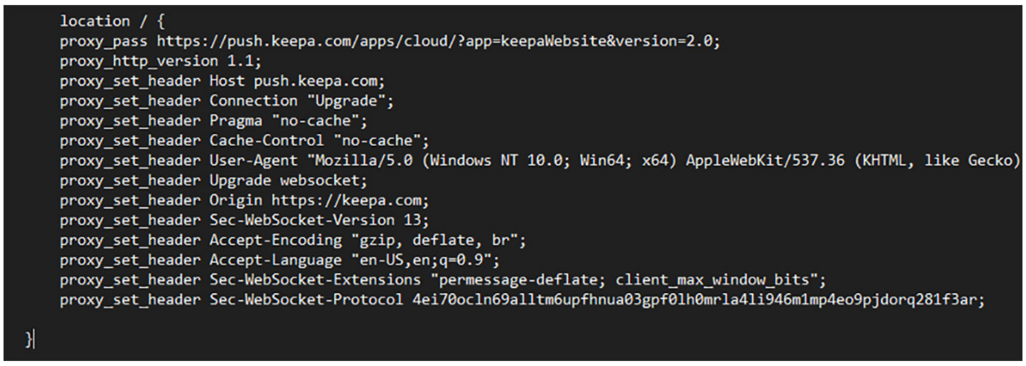
## 6 CASE STUDIES

### 6.1 Ip authorization bypass

A serious vulnerability in Keeps com's Web Socket implementation—a top Amazon data analysis SaaS service—is examined in this case study. Using Nginx to build a WS reverse proxy with certain headers allowed one to replicate the website and get around IP use limitations. This weakness opens up the service to possible exploitation by enabling unauthorized users to utilize the data against usage restrictions.

### 6.2 Configuring reverse proxy

The Nginx setup, as illustrated in Figure 7, shows how to configure a reverse proxy to the WS endpoint of Keepa. Carefully adjusting headers allows the proxy to mimic real traffic, therefore mirroring the website and removing IP use limitations.



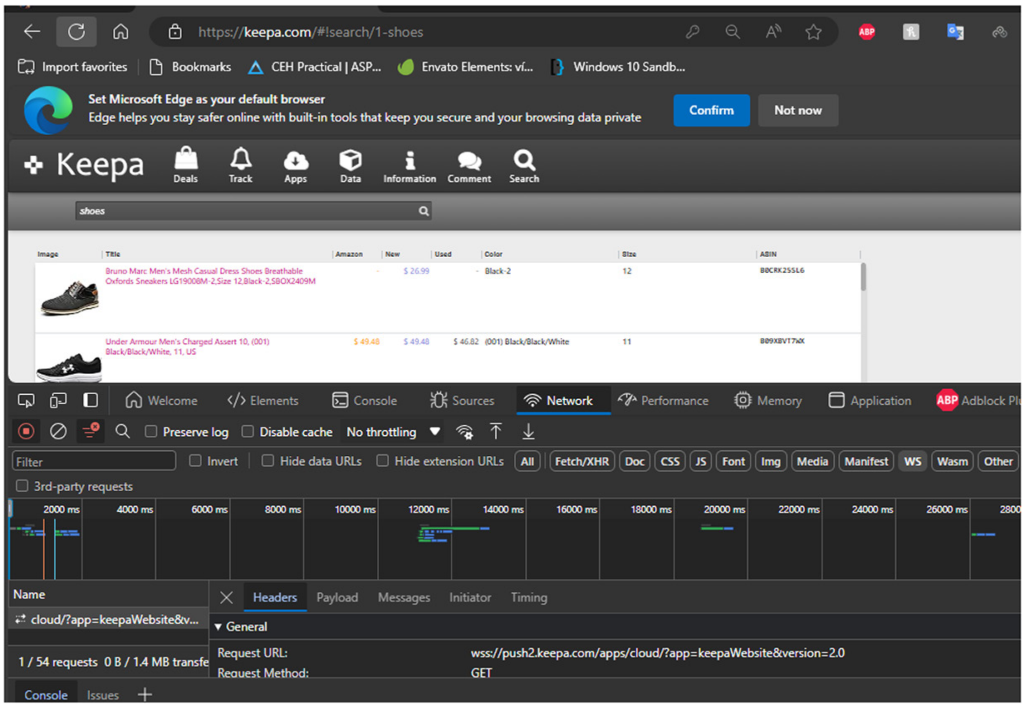
```
    location / {
    proxy_pass https://push.keepa.com/apps/cloud/?app=keepaWebsite&version=2.0;
    proxy_http_version 1.1;
    proxy_set_header Host push.keepa.com;
    proxy_set_header Connection "Upgrade";
    proxy_set_header Pragma "no-cache";
    proxy_set_header Cache-Control "no-cache";
    proxy_set_header User-Agent "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
    proxy_set_header Upgrade websocket;
    proxy_set_header Origin https://keepa.com;
    proxy_set_header Sec-WebSocket-Version 13;
    proxy_set_header Accept-Encoding "gzip, deflate, br";
    proxy_set_header Accept-Language "en-US,en;q=0.9";
    proxy_set_header Sec-WebSocket-Extensions "permessage-deflate; client_max_window_bits";
    proxy_set_header Sec-WebSocket-Protocol 4ei70ocln69alltm6upfhnua03gpf01h0mrla41i946m1mp4eo9pjdorq281f3ar;

}
```

**Fig. 7.** Reverse proxying and header manipulation

## 6.3 Keepa.com

Figure 8 illustrates the search results for “shoes” on Keepa.com. The developer tools disclose the WS URL as push2.keepa.com. This demonstrates that the original site is utilizing this particular WS endpoint to enable real-time data communication. The client’s active connection to Keepa’s server is verified by the continuous data transfer displayed in the network tab. The true site is subject to stringent IP utilization policies, which guarantee that each client session complies with Keepa’s terms and conditions, thereby preserving the security and integrity of the data. The real site operates under strict single-IP usage policies.



https://keepa.com/#!search/1-shoes

Set Microsoft Edge as your default browser
Edge helps you stay safer online with built-in tools that keep you secure and your browsing data private

shoes

| Image | Title | Amazon | New | Used | Color | Size | ASIN |
|---|---|---|---|---|---|---|---|
| | Bruno Marc Men's Mesh Casual Dress Shoes Breathable Oxfords Sneakers LG19008M-2,Size 12,Black-2,SBOX2409M | - | $ 26.99 | - | Black-2 | 12 | B0CRK2SSL6 |
| | Under Armour Men's Charged Assert 10, (001) Black/Black/White, 11, US | $ 49.48 | $ 49.48 | $ 46.82 | (001) Black/Black/White | 11 | B09XBVT7WX |

Preserve log · Disable cache · No throttling

Filter · Invert · Hide data URLs · Hide extension URLs · 3rd-party requests

Name: cloud/?app=keepaWebsite&v...

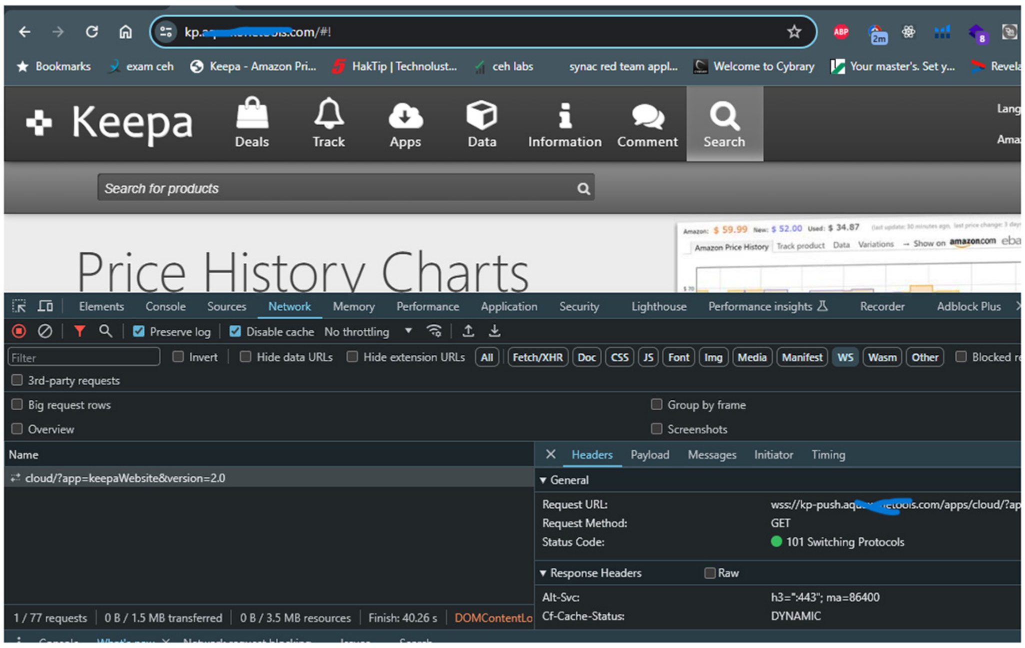
1 / 54 requests 0 B / 1.4 MB transfe

General
Request URL: wss://push2.keepa.com/apps/cloud/?app=keepaWebsite&version=2.0
Request Method: GET

**Fig. 8.** Developer tools interacting to real socket wss://push2.keepa.com

## 6.4 Mirrored site

Conversely, Figures 9 and 10 illustrate the mirrored site, which is identified by a WS URL that commences with kp-push, indicating the effective implementation of the reverse proxy. Despite the fact that the domain is obscured, it is evident that the reverse proxy is successfully replicating the connection to the original site. The network page is revealed in the final image of the series, which depicts the binaries that are being sent and received in the same manner as they would on the genuine Keepa.com. This illustrates that the reverse proxy configuration circumvents IP restrictions and replicates Keepa.com’s data transmission, thereby emphasizing a substantial security vulnerability. This configuration could potentially facilitate data scraping and misconduct by permitting unauthorized access to Keepa’s services.


kp.[redacted]s.com/#!

| Name | |
|---|---|
| cloud/?app=keepaWebsite&version=2.0 | |

General

| | |
|---|---|
| Request URL: | wss://kp-push.aq[redacted]s.com/apps/cloud/?ap |
| Request Method: | GET |
| Status Code: | 101 Switching Protocols |

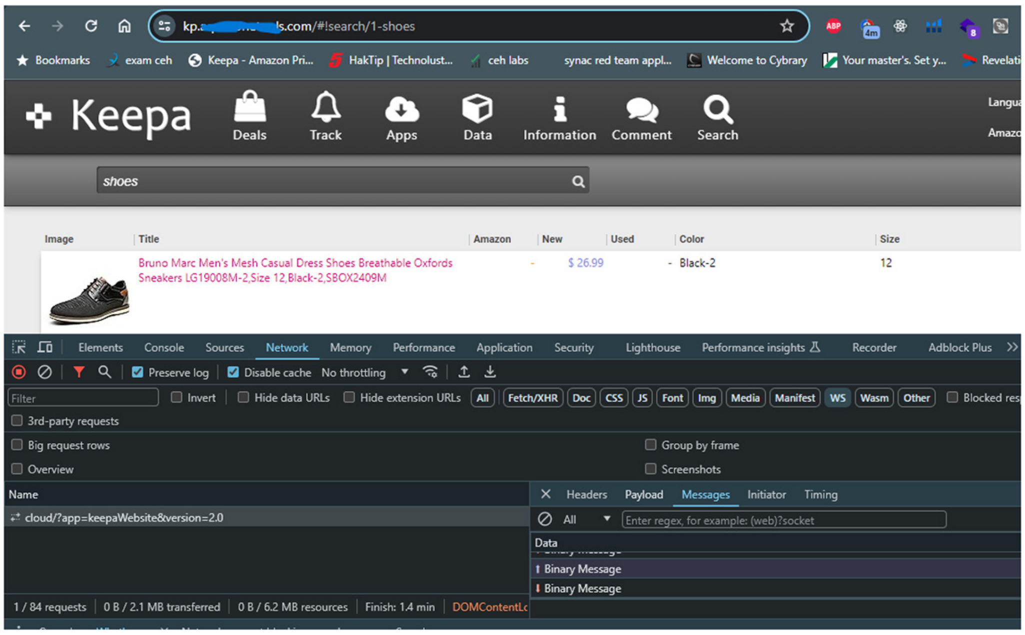
Response Headers

| | |
|---|---|
| Alt-Svc: | h3=":443"; ma=86400 |
| Cf-Cache-Status: | DYNAMIC |

1 / 77 requests | 0 B / 1.5 MB transferred | 0 B / 3.5 MB resources | Finish: 40.26 s | DOMContentLo

**Fig. 9.** Reverse proxied site successfully bypassing IP restriction on web socket

## 6.5 Authentication bypass


kp.[redacted]s.com/#!search/1-shoes

shoes

| Image | Title | Amazon | New | Used | Color | Size |
|---|---|---|---|---|---|---|
| | Bruno Marc Men's Mesh Casual Dress Shoes Breathable Oxfords Sneakers LG19008M-2,Size 12,Black-2,SBOX2409M | - | $ 26.99 | - | Black-2 | 12 |

| Name |
|---|
| cloud/?app=keepaWebsite&version=2.0 |

| Data |
|---|
| Binary Message |
| Binary Message |

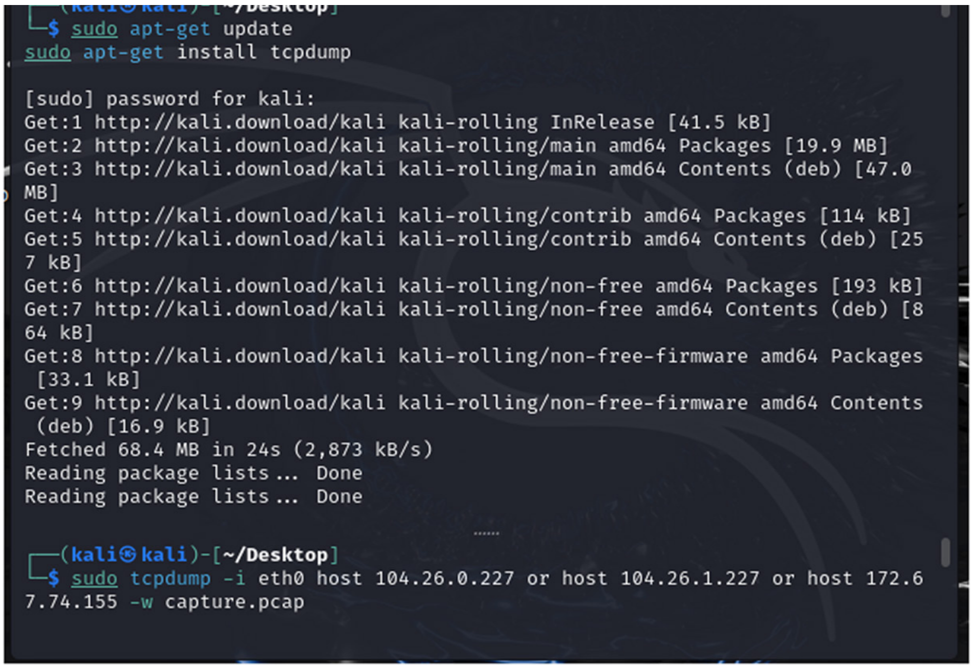
1 / 84 requests | 0 B / 2.1 MB transferred | 0 B / 6.2 MB resources | Finish: 1.4 min | DOMContentLo

**Fig. 10.** Proxied socket getting binaries as original

For the authorization It has been observed that in the mechanism of keepa.com authentication, we have noticed that keep is authenticating users in its WS, where the WS-protocol is being used as a token for user login sessions. For this purpose, we have setup our Kali machine in Vmware and installed tcpdump and wireshark in our machine to mimic the concept of capturing packets over the network. Our network interface was ETH0, and we have examined the IP addresses of their WS urls, push2.keepa.com, by using lookup push2.keepa.com, which gave us these results.

- Non-authoritative answer:
- Name: push2.keepa.com Address: 104.26.0.227
- Name: push2.keepa.com Address: 104.26.1.227
- Name: push2.keepa.com Address: 172.67.74.155
- Name: push2.keepa.com Address: 2606:4700:20::ac43:4a9b
- Name: push2.keepa.com Address: 2606:4700:20::681a:e3
- Name: push2.keepa.com Address: 2606:4700:20::681a:1e3

By identifying their ips, we have crafted our tcpdump command to capture only those packets that are sent to the WS of keepa.com, as shown below in Figure 11. sudo tcpdump -i eth0 host 104.26.0.227 or host 104.26.1.227 or host 172.67.74.155 -w capture.

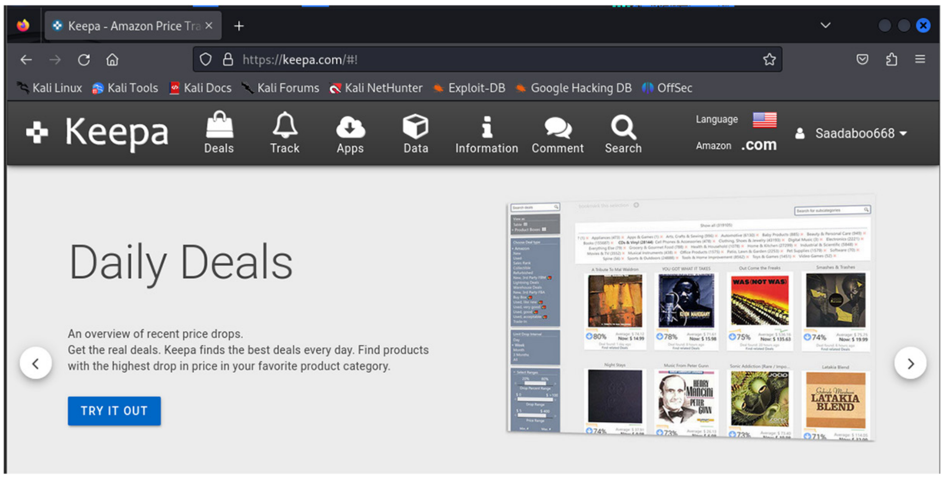


```
┌──(kali㉿kali)-[~/Desktop]
└─$ sudo apt-get update
sudo apt-get install tcpdump

[sudo] password for kali:
Get:1 http://kali.download/kali kali-rolling InRelease [41.5 kB]
Get:2 http://kali.download/kali kali-rolling/main amd64 Packages [19.9 MB]
Get:3 http://kali.download/kali kali-rolling/main amd64 Contents (deb) [47.0
MB]
Get:4 http://kali.download/kali kali-rolling/contrib amd64 Packages [114 kB]
Get:5 http://kali.download/kali kali-rolling/contrib amd64 Contents (deb) [25
7 kB]
Get:6 http://kali.download/kali kali-rolling/non-free amd64 Packages [193 kB]
Get:7 http://kali.download/kali kali-rolling/non-free amd64 Contents (deb) [8
64 kB]
Get:8 http://kali.download/kali kali-rolling/non-free-firmware amd64 Packages
 [33.1 kB]
Get:9 http://kali.download/kali kali-rolling/non-free-firmware amd64 Contents
 (deb) [16.9 kB]
Fetched 68.4 MB in 24s (2,873 kB/s)
Reading package lists... Done
Reading package lists... Done
                                          ......
┌──(kali㉿kali)-[~/Desktop]
└─$ sudo tcpdump -i eth0 host 104.26.0.227 or host 104.26.1.227 or host 172.6
7.74.155 -w capture.pcap
```

**Fig. 11.** Dumping packets over network

After that, we browsed keepa.com and signed in with user login credentials, as illustrated in Figure 12 below.

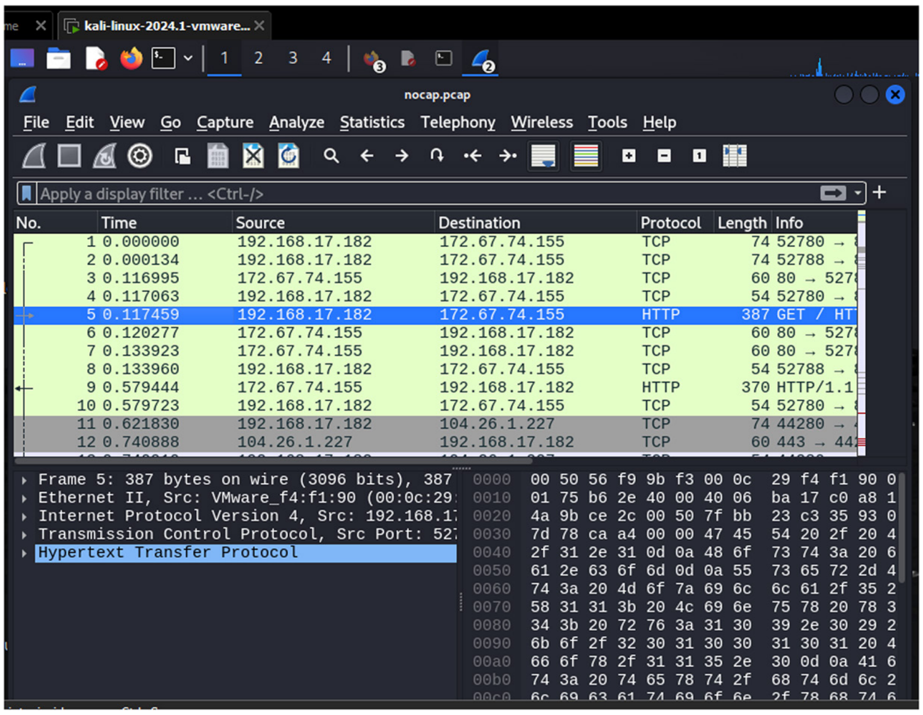


Daily Deals

An overview of recent price drops.
Get the real deals. Keepa finds the best deals every day. Find products with the highest drop in price in your favorite product category.

**Fig. 12.** Logged in session on same network

Then the authors closed the tcpdump capture and then opened the captured file via wire shark, which showed us multiple packets sent over different protocols. Our interest was in HTTP, so the identified packets 5 and 9 are HTTP, as presented in Figure 13.

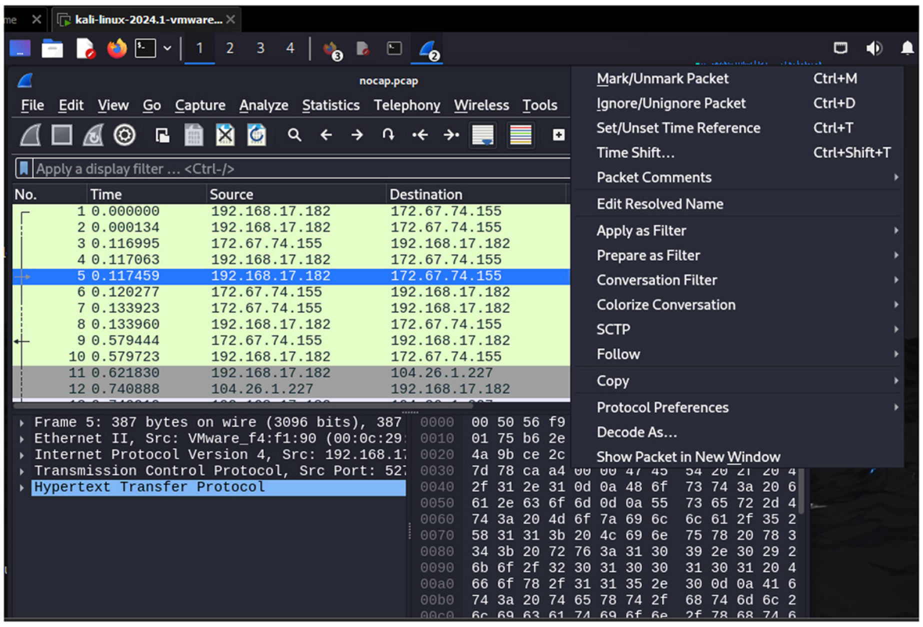

kali-linux-2024.1-vmware...
nocap.pcap
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter … <Ctrl-/>

```
No.   Time      Source          Destination     Protocol Length Info
    1 0.000000  192.168.17.182  172.67.74.155   TCP          74 52780 →
    2 0.000134  192.168.17.182  172.67.74.155   TCP          74 52788 →
    3 0.116995  172.67.74.155   192.168.17.182  TCP          60 80 → 527
    4 0.117063  192.168.17.182  172.67.74.155   TCP          54 52780 →
    5 0.117459  192.168.17.182  172.67.74.155   HTTP        387 GET / HT
    6 0.120277  172.67.74.155   192.168.17.182  TCP          60 80 → 527
    7 0.133923  172.67.74.155   192.168.17.182  TCP          60 80 → 527
    8 0.133960  192.168.17.182  172.67.74.155   TCP          54 52788 →
    9 0.579444  172.67.74.155   192.168.17.182  HTTP        370 HTTP/1.1
   10 0.579723  192.168.17.182  172.67.74.155   TCP          54 52780 →
   11 0.621830  192.168.17.182  104.26.1.227    TCP          74 44280 →
   12 0.740888  104.26.1.227    192.168.17.182  TCP          60 443 → 44
```

```
› Frame 5: 387 bytes on wire (3096 bits), 387     0000  00 50 56 f9 9b f3 00 0c  29 f4 f1 90 0
› Ethernet II, Src: VMware_f4:f1:90 (00:0c:29     0010  01 75 b6 2e 40 00 40 06  ba 17 c0 a8 1
› Internet Protocol Version 4, Src: 192.168.1     0020  4a 9b ce 2c 00 50 7f bb  23 c3 35 93 0
› Transmission Control Protocol, Src Port: 52     0030  7d 78 ca a4 00 00 47 45  54 20 2f 20 4
› Hypertext Transfer Protocol                     0040  2f 31 2e 31 0d 0a 48 6f  73 74 3a 20 6
                                                  0050  61 2e 63 6f 6d 0d 0a 55  73 65 72 2d 4
                                                  0060  74 3a 20 4d 6f 7a 69 6c  6c 61 2f 35 2
                                                  0070  58 31 31 3b 20 4c 69 6e  75 78 20 78 3
                                                  0080  34 3b 20 72 76 3a 31 30  39 2e 30 29 2
                                                  0090  6b 6f 2f 32 30 31 30 30  31 30 31 20 4
                                                  00a0  66 6f 78 2f 31 31 35 2e  30 0d 0a 41 6
                                                  00b0  74 3a 20 74 65 78 74 2f  68 74 6d 6c 2
```

**Fig. 13.** Dumped packets of our interests

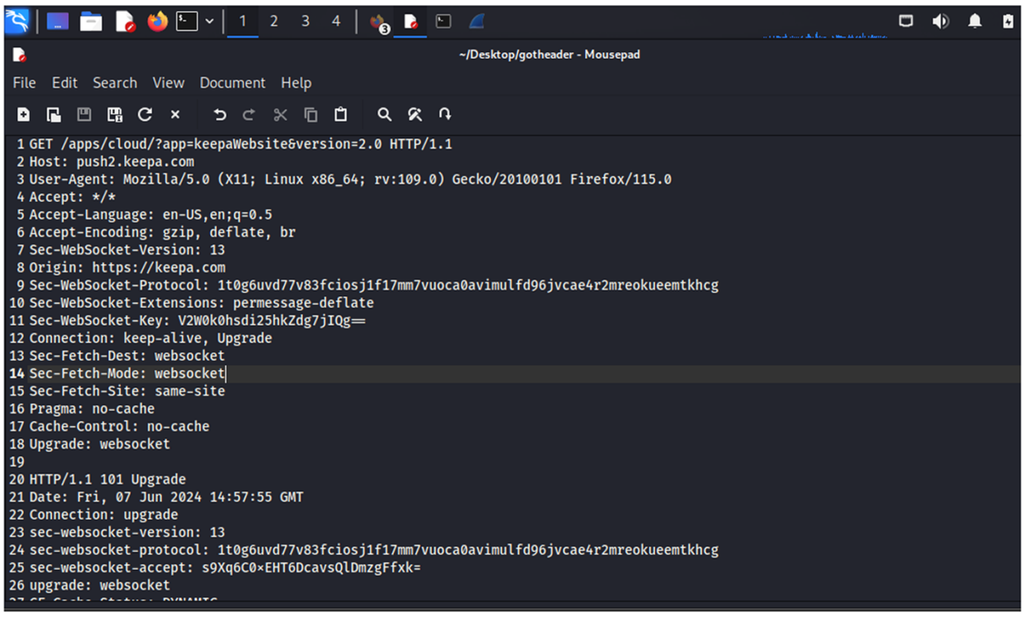
After using the follow TCP stream, authors have collected the complete header of the handshake, which included the WS protocol, which is then injected into any other browser to bypass login and get the same session as logged in before. This session continues until no one logs it out. Figures 14 and 15 demonstrate the WS retrieval from the headers.


kali-linux-2024.1-vmware...
nocap.pcap
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools
Apply a display filter … <Ctrl-/>
Mark/Unmark Packet Ctrl+M
Ignore/Unignore Packet Ctrl+D
Set/Unset Time Reference Ctrl+T
Time Shift… Ctrl+Shift+T
Packet Comments
Edit Resolved Name
Apply as Filter
Prepare as Filter
Conversation Filter
Colorize Conversation
SCTP
Follow
Copy
Protocol Preferences
Decode As…
Show Packet in New Window

```
No.   Time      Source          Destination
    1 0.000000  192.168.17.182  172.67.74.155
    2 0.000134  192.168.17.182  172.67.74.155
    3 0.116995  172.67.74.155   192.168.17.182
    4 0.117063  192.168.17.182  172.67.74.155
    5 0.117459  192.168.17.182  172.67.74.155
    6 0.120277  172.67.74.155   192.168.17.182
    7 0.133923  172.67.74.155   192.168.17.182
    8 0.133960  192.168.17.182  172.67.74.155
    9 0.579444  172.67.74.155   192.168.17.182
   10 0.579723  192.168.17.182  172.67.74.155
   11 0.621830  192.168.17.182  104.26.1.227
   12 0.740888  104.26.1.227    192.168.17.182
```

```
› Frame 5: 387 bytes on wire (3096 bits), 387     0000  00 50 56 f9
› Ethernet II, Src: VMware_f4:f1:90 (00:0c:29     0010  01 75 b6 2e
› Internet Protocol Version 4, Src: 192.168.1     0020  4a 9b ce 2c
› Transmission Control Protocol, Src Port: 52     0030  7d 78 ca a4
› Hypertext Transfer Protocol                     0040  2f 31 2e 31 0d 0a 48 6f  73 74 3a 20 6
                                                  0050  61 2e 63 6f 6d 0d 0a 55  73 65 72 2d 4
                                                  0060  74 3a 20 4d 6f 7a 69 6c  6c 61 2f 35 2
                                                  0070  58 31 31 3b 20 4c 69 6e  75 78 20 78 3
                                                  0080  34 3b 20 72 76 3a 31 30  39 2e 30 29 2
                                                  0090  6b 6f 2f 32 30 31 30 30  31 30 31 20 4
                                                  00a0  66 6f 78 2f 31 31 35 2e  30 0d 0a 41 6
                                                  00b0  74 3a 20 74 65 78 74 2f  68 74 6d 6c 2
```

**Fig. 14.** Selecting follow then TCP stream



~/Desktop/gotheader - Mousepad
File Edit Search View Document Help

```
GET /apps/cloud/?app=keepaWebsite&version=2.0 HTTP/1.1
Host: push2.keepa.com
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:109.0) Gecko/20100101 Firefox/115.0
Accept: */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Sec-WebSocket-Version: 13
Origin: https://keepa.com
Sec-WebSocket-Protocol: 1t0g6uvd77v83fciosj1f17mm7vuoca0avimulfd96jvcae4r2mreokueemtkhcg
Sec-WebSocket-Extensions: permessage-deflate
Sec-WebSocket-Key: V2W0k0hsdi25hkZdg7jIQg==
Connection: keep-alive, Upgrade
Sec-Fetch-Dest: websocket
Sec-Fetch-Mode: websocket
Sec-Fetch-Site: same-site
Pragma: no-cache
Cache-Control: no-cache
Upgrade: websocket

HTTP/1.1 101 Upgrade
Date: Fri, 07 Jun 2024 14:57:55 GMT
Connection: upgrade
sec-websocket-version: 13
sec-websocket-protocol: 1t0g6uvd77v83fciosj1f17mm7vuoca0avimulfd96jvcae4r2mreokueemtkhcg
sec-websocket-accept: s9Xq6C0×EHT6DcavsQlDmzgFfxk=
upgrade: websocket
```

**Fig. 15.** Handshake header containing sec-web socket-protocol (secure token)

This indicates how someone can easily bypass authentication if some legit user is using the same network.

## 7 RESEARCH GAPS

The authors, after a comprehensive literature review and testing, highlight the following study gaps, which could be further investigated to improve the security of web socket applications.

### 7.1 Comprehensive threat models

Although most study focus on individual attacks such as XSS and MITM, there is a dearth of comprehensive threat models that incorporate a broad spectrum of potential vulnerabilities in WS communication.

**Implication:** By developing a comprehensive threat model, one can gain a deeper understanding of the interplay between different threats and how to collectively counteract them.

**Implication:** By developing a comprehensive threat model, one can gain a deeper understanding of the interplay between different threats and how to collectively counteract them.

### 7.2 Standardization and best practices

While there are suggestions and recommended methods, there is currently no globally recognized standard that is expressly designed for WS security.

**Implication:** The establishment of standardized protocols and processes can facilitate the consistent implementation of secure WS communication across various apps by developers.

### 7.3 Performance vs. security trade-offs

There is a scarcity of studies on the trade-offs that occur when balancing performance and security in WS implementations. Certain security measures may impose latency or overhead that can have a negative impact on real-time communication.

**Implication:** Examining and enhancing these compromises can assist in achieving a balance between security and performance, especially for activities such as high-frequency trading, gaming, or real-time collaboration tools.

### 7.4 Advanced authentication mechanisms

Although there is extensive study on fundamental authentication mechanisms, further investigation is required for sophisticated techniques such as biometrics, zero-knowledge proofs, and continuous authentication, specifically in relation to web sockets.

**Implication:** Implementing advanced authentication methods can improve security measures without causing substantial negative effects on user experience or system performance.

### 7.5 The influence of emerging technologies

There is a lack of comprehensive study on the influence of new technologies such as blockchain, quantum computing, and AI on WS security.

**Meaning:** Gaining a comprehensive understanding of how these technologies can be utilized or present potential risks to WS communication is essential for ensuring the long-term effectiveness of security measures.

## 8 COMPARATIVE ANALYSIS

A comparative study of WS security review papers offers important information on how security protocols and techniques have changed over time. It can be found and shared by looking at different study, such as coverage of security concerns and useful suggestions for developers. Recurrent flaws, on the other hand, point to areas that require more study, including a dearth of empirical data and thorough mitigation plans. This study enables us to identify both well-addressed and neglected facets of WS security. The authors have included Table 2 below that lists the advantages and disadvantages of every reviewed paper.

**Table 2.** Weaknesses in the existing literature

| Sr.# | Paper Name | Strengths | Weaknesses |
|---|---|---|---|
| 1 | "The Web Socket Protocol and Security" by Juuso Karlström (2016) | Comprehensive analysis of Web Socket protocol and security vulnerabilities; practical recommendations. | Limited coverage on emerging threats; lack of detailed case studies and empirical data. |
| 2 | "Web Socket Security Analysis" by Jussi-Pekka Erkkilä (2012) | Detailed examination of Web Socket security issues; development of a custom testing tool; practical focus. | Inadequate coverage of long-term trends and emerging threats. |

*(Continued)*

**Table 2.** Weaknesses in the existing literature *(Continued)*

| Sr.# | Paper Name | Strengths | Weaknesses |
|---|---|---|---|
| 3 | "Web Socket Adoption and the Landscape of the Real-Time Web" by Paul Murley et al. (2021) | Extensive data collection and analysis; identification of misconfigurations and malicious uses; public dataset. | Limited performance impact analysis; inadequate coverage of long-term trends and emerging threats. |
| 4 | "Security Testing of Web Sockets" by Harri Kuosmanen (2016) | Detailed examination of security vulnerabilities; practical testing scenarios; comprehensive methodological approach. | Lack of automation in the custom tool; limited coverage of emerging threats and economic impact analysis. |
| 5 | "Web Socket Security" by Vanessa Wang, Frank Salim, Peter Moskovits (2013) | Comprehensive security overview; clear explanation of protocol design; practical recommendations. | Limited empirical data; lack of coverage on emerging threats; absence of detailed implementation examples. |

## 9 FUTURE WORK DIRECTIONS

In the changing WS security environment, ongoing study is required to counter new attacks and enhance current security measures. We review current methods and point up their shortcomings before proposing a number of future directions for study to improve WS security even more.

### 9.1 Developing advanced anomaly detection

Systems that utilize machine learning and artificial intelligence can greatly improve the detection and mitigation of new and changing threats. These systems must possess the ability to analyze and respond to WS communications in real-time in order to minimize any disruptions.

### 9.2 Advanced authentication protocols

While current authentication procedures provide some level of security, there is a need for more robust and adaptable protocols. Future investigations should give priority to studying multi-factor authentication and continuous authentication solutions that verify user identification during the entire session rather than just during the first phase.

The improvement of encryption standards for WS connections is absolutely necessary in order to successfully prevent data breaches and reduce the risk of MITM. There is a need for study to examine lightweight encryption solutions that can retain high performance without compromising security.

### 9.3 User agent security

The security of WS connections is significantly influenced by user agents, such as web browsers and software programs. It is recommended that future study prioritize the development of best practices and recommendations for user agent developers to effectively manage WS security protocols. This encompasses the implementation of security measures such as proactive vulnerability patching, resilient WS API security, and stringent-origin restrictions.

### 9.4 Cross-site web socket security (CSWS)

The act of cross-site web socket hijacking (CSWSH) continues to pose a substantial risk. Subsequent efforts should focus on formulating all-encompassing approaches to thwart such assaults, including improved methods of segregating data, secure management of cookies, and strengthened validation processes on the server side.

### 9.5 Standardization and compliance

The use of defined security frameworks and compliance rules is necessary for WS implementations. Research should aim to further the development of these standards, guaranteeing that all WS-based apps strictly comply with a fundamental security protocol in order to avoid vulnerabilities.

## 10 CONCLUSION

Web sockets have emerged as a critical technology for enabling immediate, two-way communication over the internet, significantly improving user experiences in a variety of online applications. WSs, while beneficial, provide security issues that necessitate strong safeguards against numerous threats, such as XSS, CSWSH, and MITM attacks. This review article provides a thorough study of the WS security environment, analyzing major study contributions since 2015. The study evaluates existing security measures and their effectiveness in tackling security risks by comparing diverse methodologies and identifying common faults. Furthermore, the study exposes the inadequacies and limitations of present methodologies, providing useful insights into the areas that require further examination and improvement. In anticipation, the paper provides potential study options for improving WS security. These include the development of sophisticated anomaly detection systems based on machine learning, improved authentication techniques, and stronger encryption standards created specifically for WS connections. The significance of user agents is also emphasized, along with suggestions for adopting solid security measures. Emphasizing the importance of CSWS, standards, and user education in developing a more secure WS environment. To encourage the development of stronger and more secure real-time web apps, the study community should focus on these key areas. This ensures the continued dependability and security of connections.

## 11 REFERENCES

[1] J.-P. Erkkilä, "Web socket security analysis," 2012.

[2] M. Shema, S. Shekyan, and V. Toukharian, "Hacking with web sockets," *BlackHat USA*, 2012.

[3] R. R. Ganji *et al.*, "HTML5 as an application virtualization tool," in *2012 IEEE 16th International Symposium on Consumer Electronics,* 2012, pp. 1–4. https://doi.org/10.1109/ISCE.2012.6241695

[4] V. Wang, F. Salim, and P. Moskovits, "The definitive guide to HTML5 web socket," vol. 1, New York: Apress, 2013. https://doi.org/10.1007/978-1-4302-4741-8

[5] Web Sockets Standard. (2024, January 24). [Online]. Available: http://dev.w3.org/html5/WebSockets/

[6] R. M. Lerner, “At the forge: Communication in HTML5,” *Linux Journal*, vol. 2011, no. 202, p. 7, 2011.

[7] C. A. Gutwin, M. Lippold, and T. C. Nicholas Graham, “Real-time groupware in the browser: Testing the performance of web-based networking,” in *Proceedings of the ACM Conference on Computer Supported Cooperative Work,* CSCW, 2011, pp. 167–176. https://doi.org/10.1145/1958824.1958850

[8] J. Bai, W. Wang, M. Lu, H. Wang, and J. Wang, “TD-WS: A threat detection tool of web socket and web storage in HTML5 websites,” *Security and Communication Networks*, vol. 9, no. 18, pp. 5432–5443, 2016. https://doi.org/10.1002/sec.1708

[9] C. Schneider, “Cross-site web socket hijacking,” 2013. [Online]. Available: https://christian-schneider.net/blog/cross-site-websocket-hijacking/ [Accessed: May. 05, 2019].

[10] I. Fette and A. Melnikov, “RFC 6455 the WebSocket protocol,” RFC editor, 2011. https://doi.org/10.17487/rfc6455

[11] P. Murley *et al.*, “Web socket adoption and the landscape of the real-time web,” in *Proceedings of the Web Conference 2021 (WWW ’21)*, 2021, pp. 1192–1203. https://doi.org/10.1145/3442381.3450063

[12] T. Wirasingha and N. Ruwan Dissanayake, “A survey of WebSocket development techniques and technologies,” in *Professional Integration for Secure Nation*, vol. 9, 2016, pp. 1–9.

[13] Comet Daily. (2008, July 4). Independence Day: HTML5 web socket liberates comet from hacks. [Online]. Available: http://cometdaily.com/2008/07/04/html5-WebSocket/

[14] V. Chopra, *Web Socket Essentials–Building Apps with HTML5 Web Socket.* Birmingham, UK: Packt Publishing Ltd., 2015.

[15] M. Hribernik and A. Kos, “Secure WebSocket based broker and architecture for connecting IoT devices and web based applications,” *IPSI Transactions on Advanced Research*, vol. 16, no. 1, 2020.

[16] M. Howard and S. Lipner, *The Security Development Lifecycle*, vol. 8, Redmond: Microsoft Press, 2006.

[17] Web APIs, “The web socket API (Web Sockets),” MDN. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API

[18] K. Harri, “Security testing of WebSockets,” Thesis, JAMK University of Applied Sciences, 2016.

[19] G. L. Muller, “HTML5 web socket protocol and its application to distributed computing,” *arXiv Preprint arXiv:*1409.3367, pp. 1192–1203, 2014.

[20] P. Murley *et al.*, “Web socket adoption and the landscape of the real-time web,” in *Proceedings of the Web Conference 2021*, 2021. https://doi.org/10.1145/3442381.3450063

[21] S. Paudel, “Vulnerable web applications and how to audit them: Use of OWASP Zed attack proxy effectively to find the vulnerabilities of web applications,” Bachelor thesis, Oulu University of Applied Sciences, 2016.

[22] S. Guan, W. Hu, and H. Zhou, “Real-time data transmission method based on web socket protocol for networked control system laboratory,” in *2019 Chinese Control Conference (CCC)*, 2019, pp. 5339–5344. https://doi.org/10.23919/ChiCC.2019.8865690

[23] S. Ghasemshirazi and P. Heydarabadi, “Exploring the attack surface of Web Socket,” *arXiv Preprint arXiv:*2104.05324, 2021.

[24] R. M. Wibowo and A. Sulaksono, “Web vulnerability through Cross Site Scripting (XSS) detection with OWASP security shepherd,” *Indonesian Journal of Information Systems*, vol. 3, no. 2, pp. 149–159, 2021. https://doi.org/10.24002/ijis.v3i2.4192

[25] OWASP. [Online]. Available: https://owasp.org

[26] D. Omeiza and J. Owusu-Tweneboah, “Web security investigation through penetration tests: A case study of an educational institution portal,” *arXiv Preprint arXiv:*1811.01388, 2018.

[27] T. Dąbrowski, “Using spring boot for web socket implementation with STOMP,” *Toptal Engineering Blog*, 2019. [Online]. Available: https://www.toptal.com/java/stomp-spring-boot-WebSocket
[28] Wamp-proto.org/intro.htm. [Online]. Available: https://wamp-proto.org/intro.htm
[29] V. Wang, F. Salim, and P. Moskovits, “The web socket protocol,” in *The Definitive Guide to HTML5 Web Socket*, 2013, pp. 33–60. https://doi.org/10.1007/978-1-4302-4741-8_3
[30] J. Karlström, “The web socket protocol and security: Best practices and worst weaknesses,” MS thesis, University of Oulu, Department of Information Processing Science, 2016.
[31] N. Gibbins, “Cross origin resource sharing,” University of Southampton, 2016. https://edshare.soton.ac.uk/20595/

## 12 AUTHORS

**Malik Muhammad Nadeem** is with the Department of Cyber Security, Riphah Institute of Systems Engineering, Riphah International University, Islamabad, Pakistan.

**Yousaf Raza** is with the Department of Cyber Security, Riphah Institute of Systems Engineering, Riphah International University, Islamabad, Pakistan.

**Dr. Ahthasham Sajid** is with the Department of Cyber Security, Riphah Institute of Systems Engineering, Riphah International University, Islamabad, Pakistan (E-mail: ahthasham.sajid@riphah.edu.pk).

**Hamza Razzaq** is with the Department of Cyber Security, Riphah Institute of Systems Engineering, Riphah International University, Islamabad, Pakistan.

**Rida Malik** is with the Department of Cyber Security, Riphah Institute of Systems Engineering, Riphah International University, Islamabad, Pakistan.

**Dr. Sugandima Vidanagamachchi** is with the Department of Computer Science, Faculty of Science, University of Ruhuna, Matara, Sri Lanka.