FEROOT.COM

feroot

Guide to Preventing JavaScript Supply Chain Attacks

Contents

Common Code.
Common Challenges.

This programming language is used in **98% of all global websites.**

Up to 80% of websites pull this code from open-source or third-party sources.

What is it? JavaScript. And while it is ubiquitous with 21st century web applications, it is also a notable contributor to the ongoing software supply chain attack issues

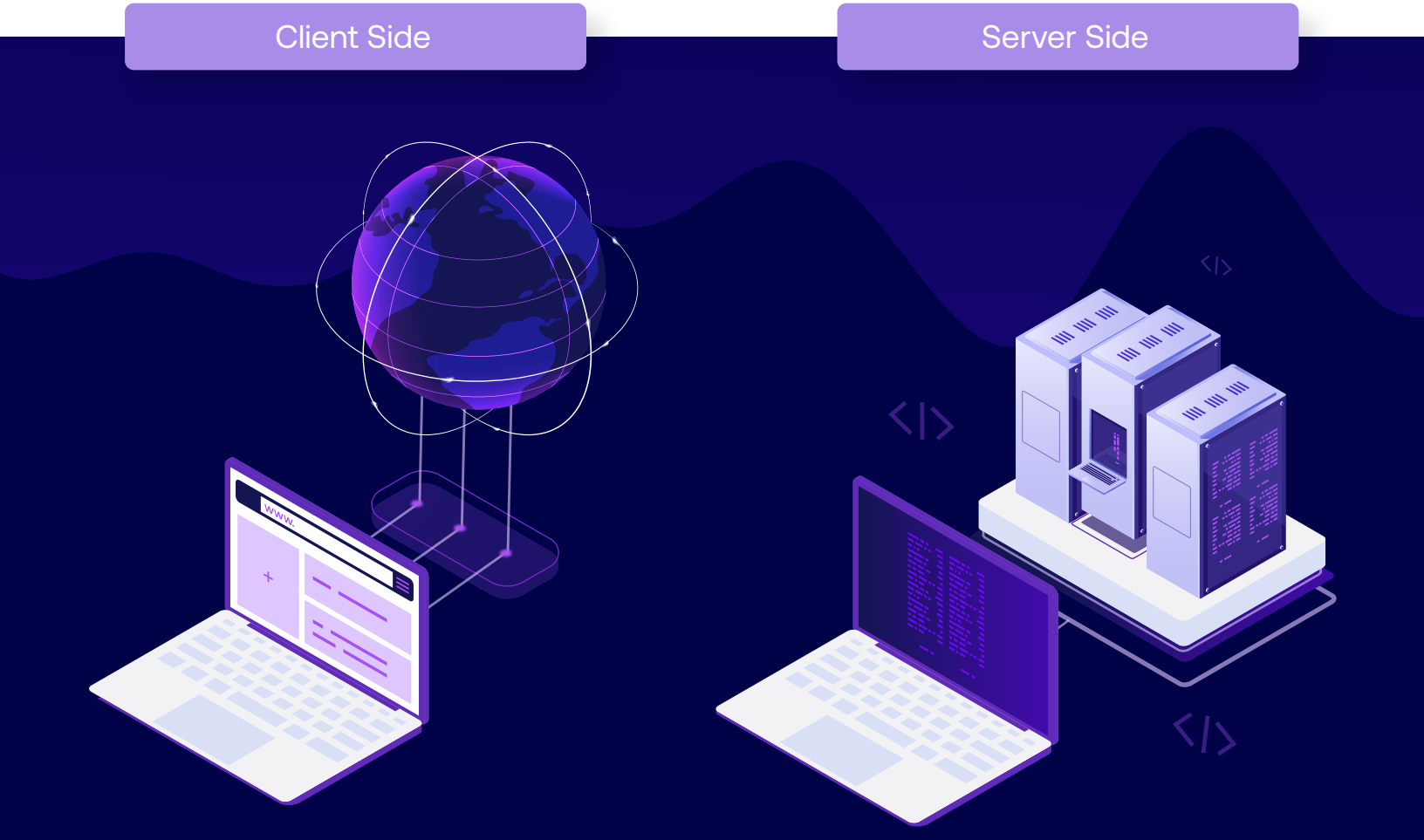
Recent research by Argon Security suggests a 300% increase in software supply chain attacks in 2021.

Software supply chain attacks are no joke. They currently dominate news headlines. As businesses leverage dynamic websites features to engage users and improve functionality, it becomes incredibly important to prevent web application exploitation. Businesses need to extend their security investments specifically to account for the growing client-side JavaScript attack risk.

Server Side vs Client Side

Before we get too deep into client-side attacks, it's important to understand the origins of the term. The words "client side" and "server side" describe different types of connected devices. So, end-user devices, like desktops, laptops, mobile phones, and tablets are considered 'clients,' while the systems that the devices are connected to are referred to as 'servers.' In this paper, when we talk about the "client side," we're referring to activities that occur on the front end of a web application—the side that the end user can access. "Server side" refers to things happening on the back end—or the side that is only visible to authorized individuals within the business.

When we discuss client-side vulnerabilities, we're referring to flaws, bugs, or malicious code that exist within any scripts, text, videos, images, or chat windows that live in a web application accessible by an end-user device.



The Problems with JavaScript Begin with JavaScript

JavaScript is a text-based programming language used on both the client side and server side. On the client side, it allows developers to build web applications with interactive elements that engage the user and enhance their experience. JS is relatively easy to learn and easy to share. Developers and marketers love JavaScript because of its flexibility and ease of use. Most web applications, and the elements within them, are written using JavaScript code. **However, because JavaScript was not designed with security in mind, it is extremely common for threat actors to exploit the code, often with the end goal of exfiltrating sensitive information, such as PII, end-user data, credit card numbers, or other types of financial information.**

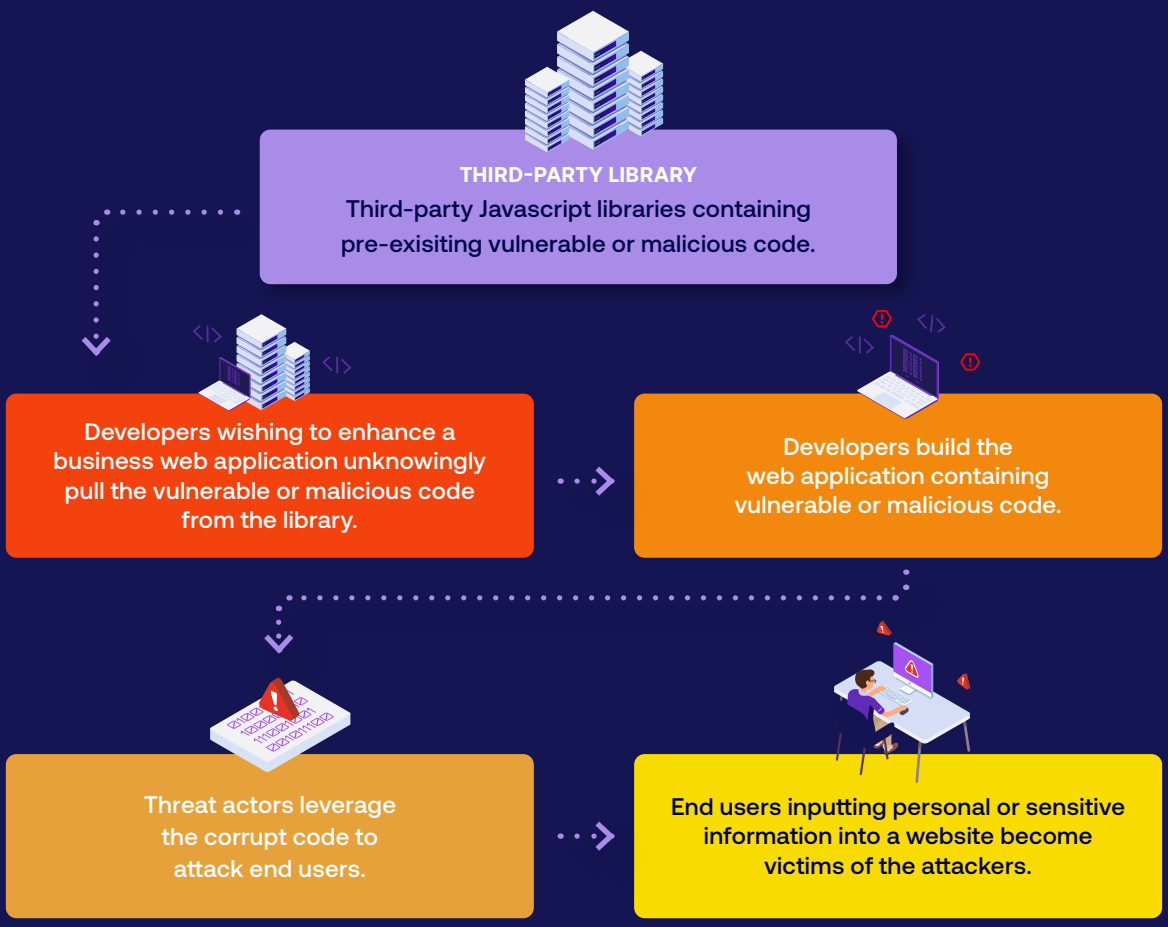
Hacked JavaScript can end up in web applications one of two ways:

- → Threat actors may alter existing JavaScript or inject malicious code directly into the current web application.
- → Flawed for intentionally malicious JavaScript can find its way onto a web application via the software supply chain.

The Connection Between the Client Side and the Supply Chain

A software supply chain attack begins with software—in this case, flawed or malicious code written in JavaScript. Since JavaScript is inherently vulnerable, and so many web applications are assembled from open source and third-party JavaScript libraries, it's inevitable that web applications, cobbled together by developers in third-party JavaScript, will come under attack. Any business using JavaScript code, add-ons, or plug-ins from third-party sources is placing themself and their end users at risk.

The Client-Side Software Supply Chain Attack Process



Client-side software supply chain attacks can dramatically affect businesses, even when other server-side cybersecurity defenses are comprehensive and current. Sometimes the business doesn't even know they've been compromised until their customers begin to experience problems, such as identity theft or credit card fraud. Ultimately, any business using JavaScript is at risk—making a client-side software supply chain attack one of the most dangerous types of cyberattacks facing businesses today. We should also mention that there's nothing novel or sophisticated about most of these attacks. Attackers are merely taking advantage of insecure code assembled into software that is commonly used in websites and web applications.

Software supply chain attacks can devastate businesses. Immediate effects of an attack can include operational delays, system infiltration, and the theft of sensitive credentials or customer data. But it's the long-term consequences that can cause significant impact, ranging from regulatory fines and compliance concerns to reputation damage, attacks on connected businesses, and lost customers.

Types of JavaScript Exploits

E-skimming Attacks: Magecart and Formjacking

Used interchangeably, the terms e-skimming, formjacking, and Magecart are all client-side attacks. The attack mechanism is fairly straightforward: skimming makes use of malicious code within web applications to exfiltrate information end users input into the application (e.g., credit card data inputted into check-out pages on shopping websites). The information is then transferred to the threat actor's command and control server either for reuse by the threat actor or for sale on the dark web.

Smart attackers have learned how easy and successful it is to engage in skimming attacks. JavaScript code is easy to manipulate and, as we stated previously, is used on almost every website, so e-skimming remains a favored attack type among hackers.

JavaScript Injection

In a JavaScript Injection attack, an adversary injects malicious code directly into the client-side JavaScript by changing or adding to the existing code. This allows the threat actor to manipulate the web application and collect sensitive data, such as PII or payment information. Threat actors sometimes even use JavaScript code obfuscators or scramblers to make it difficult for web application developers or security experts to see the malicious code and detect threats.

JavaScript Sniffers

JavaScript sniffers are malware designed to steal financial transaction data from websites, web applications, and forms where customers input their information to purchase goods or services (e-commerce websites, primarily). In a real-world example, over the course of five years, a criminal group known as UltraRank led sniffing attacks on more than 700 e-commerce sites and 13 third-party suppliers globally.

The Impact of JavaScript Supply Chain Attacks

With JavaScript dominating web applications, the impact of a JavaScript supply chain attack can be broad. Ultimately, web application exploitation doesn't really require much vulnerable JS code. In fact, the 2018 well-documented breach of British Airways involving Magecart used only 22 lines of code. The attack succeeded in stealing 380K credit card numbers, names, and other types of personal information.

Regardless of the attackers' goals, supply chain attacks are growing both in number and impact. Recent studies suggest that supply chain attacks tripled in 2021, compared to 2020, and there's no reason not to expect an equal or higher growth rate in 2022.

It's also important to understand that it is both vulnerable JavaScript code and malware specifically designed to target flawed JS that is being used by attackers to exploit systems and cause damage. According to a 2021 report published by ENISA (the European Union Agency for Cybersecurity), on the topic of the threat landscape for supply chain attacks, an estimated 66% of attacks focused on the supplier's code and 62% of attacks relied on malware to exploit that code.

Suffice to say, these attacks are widespread and can have a severe impact on any web application.

But let's cut to the chase and discuss why you're really here: how to stop JavaScript supply chain attacks from impacting your company.

How to Prevent Client-Side JavaScript Supply Chain Attacks

How you address vulnerable or malicious JavaScript depends on how you use JavaScript in your business. This next section provides approaches for protecting against JavaScript supply chain attacks based on whether you're building the software using JavaScript or whether you're a business wanting to stay ahead of supply chain attacks against your environment because of the existence of potentially vulnerable JavaScript code contained in your web application.

It's worth noting that some of these approaches are potentially ineffective when it comes to preventing dynamic client-side level protections, but they're worth including here so that you can deploy a defense-in-depth approach to client-side security. Of course, remember there's no one-size-fits-all solution and a strategic approach based on your own risk tolerance level is best.

Software Developers

Use Secure Software Development Practices

The industry trend of moving to a DevSecOps process focuses on building security into software during the development phase, rather than treating security as an afterthought. It's important to begin building software with security in mind and to evaluate and test your code to detect vulnerabilities during the development phase.

The DevSecOps ethos (aka "move security to the left") helps dev teams build more secure software by focusing on writing secure code, and it will help eradicate software vulnerabilities that can be utilized for supply chain attacks.

Maintain Safe JavaScript Libraries

Make sure you're not using any blocklisted external libraries in your product. Regularly patch and update any libraries in use. If possible, minimize dependencies on third-party JavaScript libraries.

Be Highly Selective with Third-Party Scripts

Developing your own code is expensive and time-consuming. That's why many developers will snag existing code from external libraries and build on top of that. Unfortunately, third-party code is often the origin of the vulnerabilities that lead to supply chain attacks. So make sure the code is tested, and acknowledge that by using any code that you haven't written and tested yourself, you're accepting a certain level of risk in your web application and with your clients.

Automate Code Testing and Monitoring

Code testing is also expensive and time consuming, so take advantage of automated tools built by security experts that can do the work for you. Automated solutions built specifically to protect the client side include Feroot Security Inspector, which automatically discovers all web assets a company utilizes and reports on their data access, and PageGuard, which runs continuously in the background to automatically detect unauthorized scripts and anomalous code behavior, and block them from accessing and sharing data.

Security Teams

Audit Your Web Assets

Continuously monitor the assets you own for threats and known vulnerabilities. You can do this with automated attack surface management tools, by doing your own deep-dive scans to reveal intrusions and anomalies (attacker methodologies, asset changes, user access changes, etc.), and by leveraging other threat intelligence.

Automate Monitoring and Detection of Your Web Application

Purpose-built automated software-as-a-service (SaaS) products can automatically run in the background and detect issues in all JavaScript code running on your site, including the apps and code you may not know is in use. Check it out to see if it can help you get visibility into these unknowns.

Consider Using Subresource Integrity (SRI)

SRI is a document-level approach that enables browsers to check resources against known hashes within the browser to make sure it hasn't been modified or tampered with since its creation. SRI takes advantage of how a web page loads: an HTML "file" that is part of a larger DOM object is rendered for the user to see when a web page loads. SRI looks at the HTML file "document" to assess its risk. Essentially, SRI would flag any anomalies or changes so that you can address them and shut down attacks once they've already been launched.

The limitations include being put into a reactive state of responding to attacks while they're in process, but this is better than waiting until you've been exploited and then trying to repair the damage after the fact. There's also nothing automated or continuous about this approach. It's a single, point-in-time check, but it is effective for static, content-like, cryptographically-signed messages. For JavaScript resources in particular, SRI can fall short, as these are dynamic, ever-changing elements that require a continuous, automated approach, ideally one that alerts within the tools (SIEMs, incident detection and response, etc.).

Enable Policies and Restrictions, But Know That They May Be Bypassed

If you're in security, you know that sometimes you just need to restrict certain types of code or applications from being used within your environment. The downside to this approach is that it harkens back to the days of old, when security teams were the people who said "no" to all the cool new features other teams wanted to roll out. Consider enabling policies, which can feel like a more empowering approach, though they basically do the same thing of restricting. In the case of supply chain attacks, policies can prevent dangerous or commonly insecure code from being used on your website.

Policies can be enforced automatically, using automated security tools, but they don't always detect issues on unknown assets, which is why you need to be using an automated monitoring and detecting tool in addition to your security policies.

Implement Content Security Policies (CSP)

CSPs enable the website administrator to define who (or what) the website can communicate with. You define what kinds of things can be loaded and from which domains and then set up alerts on a reporting endpoint for when the policy has been violated. Think of this one as a kind of whitelisting approach.

The limitations of a CSP are that the initial source (domain, element, application, etc.) must be trusted completely because it won't flag any malicious changes or anomalies once the application (or resource) has been "safelisted" or "allowed" within the policy. There are very few, if any, third-party resources that should warrant that level of trust.

OWASP has created a useful guide on how to make CSPs work in your environment, and they're certainly worth including, despite their limitations when it comes to third-party JavaScript vulnerabilities.

Use Web Application Firewalls (WAFs)

WAFs are a special category of firewall that are designed specifically to protect web applications and websites at the application layer. They're deployed in front of web applications to analyze web traffic in order to detect and block malicious or unauthorized activity or threats.

While you should definitely think about using a WAF, they have a few noteworthy limitations: They are not designed to protect the browser-level client-side interactions, so they can't detect and protect you from sophisticated skimming malware, drive-by skimming, supply chain attacks, or side loading and chain loading attacks.

We recommend that you consider an automated tool for scanning your entire client-side attack surface, along with enforcing the policies mentioned here. Policies make a great foundation for website security, but they can't prevent everything, particularly third-party elements and apps. With automated detections working in the background and immediately flagging issues for your team, you have a holistic, full-scale approach that's both manageable, end-user friendly, and secure.

Demo Inspector and PageGuard, Feroot's automated client-side attack surface management tools!

feroot

Feroot Security

Our team of security experts is ready to give you a hands-on demo of how our automated client-side attack surface management tools can help you stop supply chain attacks before they ever get started. Get out of the reactive cycle and refocus on prevention!

And to learn more about JavaScript security, check out our new e-book
The Ultimate Guide to JavaScript Security!

Additional Resources

E-book: The Ultimate Guide to JavaScript Security

E-book: The Ultimate Guide to Client-Side Security

E-book: The Skimming Threat

The Ultimate Guide to Client-Side Security Executive Summary

Defending the Client-Side Attack Surface

OWASP Content Security Policy Cheat Sheet