Client-Side Mitigation of Web Supply Chain Attacks

A WHITE PAPER BY JSCRAMBLER

+1 650 999 0010 hello@jscrambler.com jscrambler.com

Table of Contents

Executive Summary

The ongoing Digital Transformation is pushing businesses of all sectors to bring to market highly innovative digital products. As development teams are pushed to develop advanced applications in record time, using third-party code has become a standard practice — 66% of modern applications' code comes from third-parties.

Often single developers or small companies, **these providers don't have enterprise-grade security systems.** Yet, this code has the same permissions as the code developed in-house. As attackers clearly identify this weakest link in the Web supply chain, major web supply chain attacks have taken place recently, including the **Magecart attacks** on British Airways and Macy's.

Companies must address Web Supply Chain Attacks now, as the attack surface of their applications has increased significantly. Mitigating Web Supply Chain Attacks like Magecart requires a **multi-layered approach**, including **CSP**, **SRI**, **limiting and vetting external code**, and using a holistic client-side security solution for **JavaScript protection** and **webpage monitoring**.

Jscrambler provides a complete JavaScript Application Shielding and Webpage Monitoring solution to detect client-side code injections and prevent tampering in real-time. As so, it enables companies to **react to Web Supply Chain Attacks as they happen, mitigating them before end-users are affected**.

Introduction

Today, innovation and disruption are major drivers of business performance. The ongoing digital transformation has urged businesses across all sectors to invest in their own digital platforms.

This has reshaped the software development industry itself. Today, software development teams are pushed to deliver highly advanced applications in record time — developing everything in-house has stopped being a sustainable practice. As the NIST puts it, "*This ecosystem has evolved to provide a set of highly refined, cost-effective, reusable ICT solutions*[1]". These " reusable solutions" encompass two dimensions: **code libraries/frameworks** and **third-party scripts**.

Code Libraries and Frameworks

Mainstream programming languages such as JavaScript led to the emergence of an open-source ecosystem where code sharing and reuse have become standard practices. Today, JavaScript libraries and frameworks in the Node ecosystem are two major promoters of development speed.

Creating a boilerplate application with a popular JavaScript framework such as React.js means installing **over 1,000** code dependencies — which are mostly **open-source** and maintained by volunteers. While this enables developers to build on top of solid code, it greatly increases the attack surface of the application.

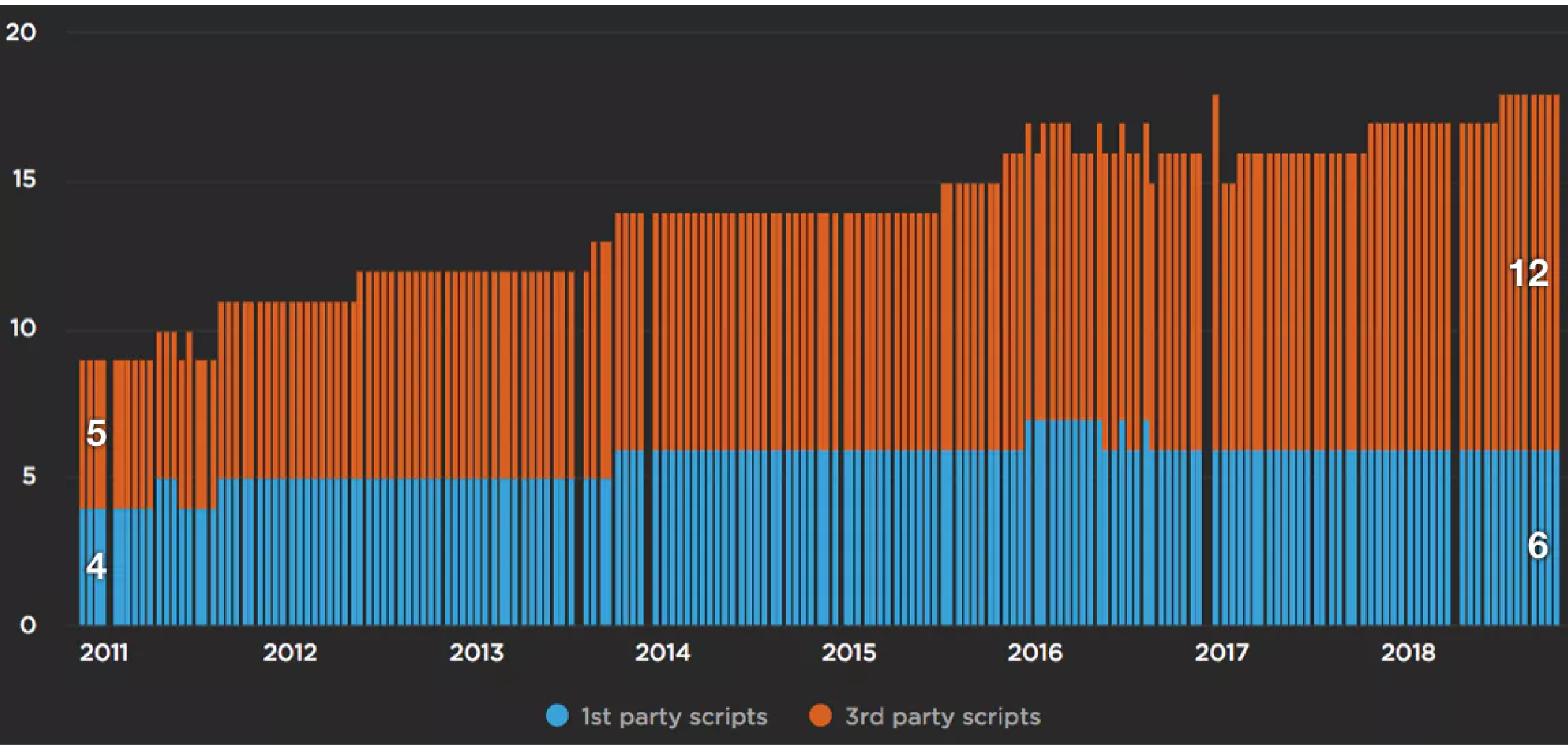
[1] https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-161.pdf

Third-Party Scripts

Another very common practice in web development is integrating third-party scripts to access a myriad of services without having to develop them in-house. A typical application can rack up several external scripts to provide this extra functionality with little extra effort[2].

Two-thirds of code in web applications today are third-party scripts.

[2] https://discuss.httparchive.org/t/js-requests-size-1st-party-vs-3rd-party/1509



A comprehensible example here would be a live chat widget. Integrating this service requires loading the external script from the web app's client-side — enabling the application to directly load the third-party code and grant it access to events that are triggered by the end-users (button onclick, mouseover, form submit, among others).

JavaScript libraries/frameworks and **third-party scripts** are powering most modern applications. But **what happens when these third-party developers or providers are attacked?**

Weaknesses and Risks

Amidst this rush to maximize development speed, **security has remained an afterthought**. Open-source ecosystems are based on collaboration: projects are developed and maintained by volunteers, which have full control to modify their code. But what happens when we consider a project that is a dependency in a major framework such as React.js?

A single developer with malicious purposes can breach thousands of enterprises by compromising a single component which is used as a dependency.

A compromise of a direct dependency of a major framework may be detected and fixed quickly. However, a dependency can require other dependencies, and so on. Escalating through this web supply chain, attackers can trace dependencies to a smaller, not well-maintained project and compromise it, reaching their targets downstream.

The risk of **integrating third-party scripts** is similar: if the third-party script provider gets breached by attackers who inject malicious code, all applications that load this script will start serving this malicious code to their end-users. Third-party code is granted the same privileges as all the code developed in-house. This major risk urged OWASP to feature “Using Components with Known Vulnerabilities” on their Top 10 Application Security Risks[3].

Most third-party code providers don’t have enterprise-grade security systems.

It’s not likely that attackers compromise the scripts of companies the size of Google. But what about the majority of providers, which are small companies or developers? When we consider this scenario, it’s clear that most third-party code providers don’t have enterprise-grade security systems and become appealing targets.

Attackers have become able to breach high-profile companies without ever having to touch their servers or code. The holy grail is now to target third-party dependencies or scripts, in what we have come to know as **Web Supply Chain Attacks.**

[3] https://www.owasp.org/index.php/Top_10-2017_Top_10

Web Supply Chain Attacks

A Supply Chain Attack is characterized as "*an intentional malicious action taken to create and exploit a vulnerability in ICT (hardware, software, firmware) at any point within the supply chain*"[4].

In software development, these attacks typically rely on inserting malicious code into a code dependency or third-party service. Common attack objectives include:

1. **Violating confidentiality** (intercept): gain unauthorized access to information;
2. **Reducing integrity** (modify, fabricate): cause the system to malfunction; cause end users to mistrust the information and information system; or cause end users to do unintended things;
3. **Reducing availability** (degrade, interrupt): making the system and information or resource unavailable when it is needed;
4. **Using resources for illegitimate purposes** (unauthorized use or usurpation): use for potentially harmful reasons and violate the confidentiality, integrity, or availability of other resources that trust the information asset being attacked by the adversary (as they don't know it is compromised).

When compared to typical cyber attacks, Web Supply Chain Attacks provide three main advantages to attackers:

1. **Lack of privilege separation on the Web** — all pieces of third-party code have the same privileges as code that is developed internally. As a result, external code can harvest any user input, add extra code, hijack events, fully modify the

[4] https://www.mitre.org/sites/default/files/publications/pr-18-0854-supply-chain-cyber-resiliency-mitigations.pdf

behavior of the web page, tamper with other code in the same scope, and contact any external domain, possibly exfiltrating data.

2. **Targeting multiple companies with a single attack** — the same dependency or script is used by multiple companies, greatly increasing the potential return on investment of the attack. Recent data shows that a breach to 20 maintainer accounts would trigger an attack to more than half of the entire Web ecosystem[5].
3. **Remaining undetected by perimeter defenses** — these attacks are often initiated by an embedded change to a component which is trusted by default; an approved delivery mechanism such as a software update can deliver the Web Supply Chain Attack without arising any suspicion by network defenders.

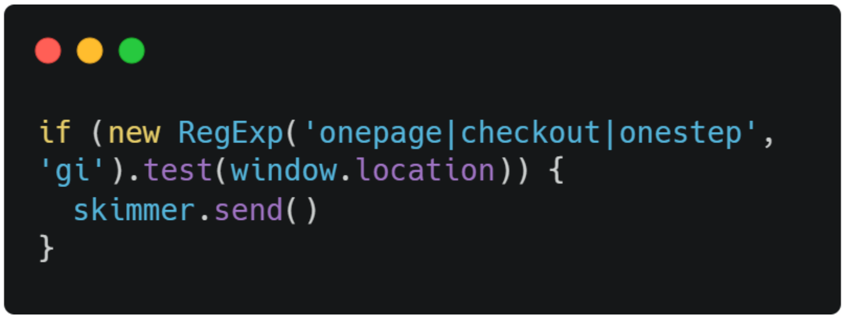
[5] https://www.usenix.org/system/files/sec19-zimmermann.pdf

Case Study 1 – Magecart (British Airways, Ticketmaster, others)

“Magecart” is an umbrella of cybercriminal groups which have been injecting digital credit card skimmers on e-commerce websites. In 2018 alone, Magecart attacks on Ticketmaster and British Airways (BA) stole details of at least 420,000 credit cards.

The BA attack was achieved by injecting malicious code on the Modernizr JavaScript library that the company was loading on its website and mobile app. This malicious code was able to detect when credit card details were written and send them to attackers’ servers. Analysis of this attack clearly shows that it specifically targeted the company, as attackers identified and exploited this weak link on BA’s supply chain.

However so, the groups’ *modus operandi* is usually to infect as many third-party script providers as possible, as the malicious code itself shows:


```
if (new RegExp('onepage|checkout|onestep',
'gi').test(window.location)) {
  skimmer.send()
}
```

With the regex check, the skimmer activates in web pages whose URL matches usual payment page keywords (onepage, checkout, onestep). It also grabs information inserted in any input fields to target a greater number of websites.

The Ticketmaster breach of June 2018 had this approach. Magecart compromised two third-party scripts — SociaPlus and Inbenta — to steal credit card details of

40,000 customers, a number greatly influenced by the two months that took the company to identify the breach. According to RiskIQ, Magecart attacks had already been detected over 2 million times as of 2019. In March 2020, with the pandemic accelerating the growth of e-commerce, web skimming grew by 26%[6]. The most noteworthy statistic about these attacks is that companies take, on average, 22 days to detect the breach, with some companies taking several months.

During 2020 and 2021, several other high-profile companies were blindsided by Magecart attacks, including Tupperware, Claire's, Intersport, Warner Music Group, and JM Bullion.

Additional details about Magecart attacks can be found on our Info Sheet.

[6] https://blog.malwarebytes.com/cybercrime/2020/04/online-credit-card-skimming-increases-by-26-in-march/

Case Study 2 – BrowseAloud (ICO, others)

In February 2018, a Web Supply Chain Attack exploited over 4000 websites, including government bodies such as the UK's Information Commissioner's Office (ICO).

The attack got affected websites to serve the CoinHive cryptocurrency miner to their end users — using users' computational power without their consent to generate cryptocurrency for attackers' benefit.

The attack was performed by injecting the CoinHive miner into a third-party script (BrowseAloud), which all affected websites were loading. The change to the script passed on as a regular update and all 4000+ websites automatically accepted this new script version by default — an imprudent yet standard practice.

Inducing the end-users of affected websites to mine cryptocurrency without their consent is illegal and prompted affected institutions and companies to shut down their websites while the issue remained unresolved by the third-party provider.

Unlike the Magecart attacks, no user data was stolen on this attack. Still, affected entities faced backlash over the incident and had to temporarily take down their services, effectively keeping a permanent dent on their reputation.

Case Study 3 – event-stream (Copay)

Both aforementioned attacks inserted malicious code into a third-party script. The November 2018 event-stream incident featured a compromised code dependency.

Copay, a cross-platform cryptocurrency wallet, is built with JavaScript and relies on several open-source modules. One such module, event-stream, was maintained by a single developer who eventually dropped the project and legitimately passed control to a volunteer. This new developer inserted a direct dependency in the event-stream module which contained malicious code. As so, the event-stream module began serving malicious JavaScript code to all projects which had it as a dependency.

While the malicious code affected thousands of projects, it had been created to target Copay's specific development environment setup. With no visibility over the compromised code, Copay inadvertently put the malicious code in production in its crypto wallet. As a result, account data and private keys from Copay accounts with a balance over 100 Bitcoin or 1000 Bitcoin Cash were stolen and sent to attackers.

The attack itself generated a gigantic response from the community. Using third-party code (dependencies) poses major security risks that are often disregarded. The event-stream incident came to show that attacks can breach the supply chain at its onset and remain undetected even after going to production and reaching end-users.

How to Address

The *Supply Chain Attacks and Resiliency Mitigations* report by the MITRE Corporation identifies several cyber resiliency techniques for mitigating Web Supply Chain Attacks, including:

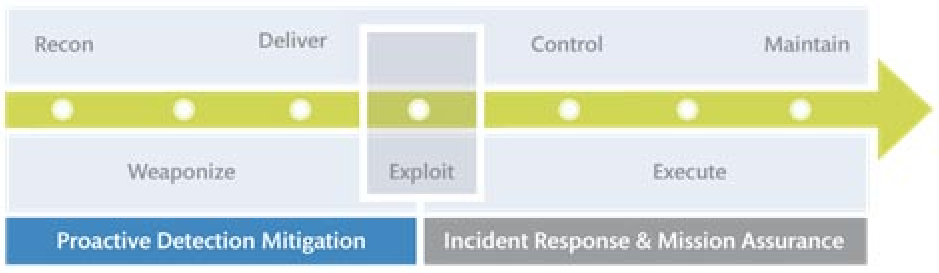
- **Adaptive Response** — Optimize the organization's ability to respond in a timely and appropriate manner to (...) attacks, thus maximizing the ability to maintain mission operations, limit consequences, and avoid destabilization.
- **Analytic Monitoring** — Gather, fuse, and analyze data on an ongoing basis and in a coordinated way to identify potential vulnerabilities, adverse conditions, stresses, or attacks, and damage;
- **Coordinated Defense** — Ensure that failure of a single defensive barrier does not expose critical assets to threat exposure. Require threat events to overcome multiple safeguards (...);
- **Deception** — Mislead, confuse, or hide critical assets from the adversary;
- **Diversity** — Use heterogeneity to minimize common mode failures, particularly attacks exploiting common vulnerabilities;
- **Redundancy** — Provide multiple protected instances of critical resources;
- **Substantiated Integrity** — Detect attempts by an adversary to deliver compromised data (...) as well as successful modification or fabrication;
- **Unpredictability** — Make changes randomly or unpredictable;

Foremost, mitigating Web Supply Chain Attacks requires a **security-in-depth approach**. Investing resources on periphery defenses alone is not an adequate approach. The same goes for SAST (Static Application Security Testing). Web Supply Chain Attacks exploit weaknesses - not vulnerabilities - to introduce malicious logic into existing code. As so**, it remains undetected by SAST**. Since these attacks frequently operate through changes that are manifested on the client-side, investing in client-side security becomes a key step of the process.

JavaScript Protection and Webpage Monitoring

Leading client-side security solutions such as Jscrambler provide a holistic client-side approach to meet the mitigations outlined by MITRE.

Jscrambler Code Integrity module serves as a first-line of defense to prevent Web Supply Chain Attacks. The Cyber Attack Lifecycle (shown below), begins with a “Recon” phase, where attackers search for weaknesses to exploit.


Recon
Deliver
Control
Maintain
Weaponize
Exploit
Execute
Proactive Detection Mitigation
Incident Response & Mission Assurance

Jscrambler Code Integrity uses obfuscation techniques, code locks, and self-defensive features to protect JavaScript code and block attackers from debugging the code in a *Recon* attempt. This is a best practice and an OWASP recommendation[7] for Mobile Applications that Jscrambler enables for

[7] https://www.owasp.org/index.php/Mobile_Top_10_2016-M9-Reverse_Engineering

JavaScript-based applications, whether they are running on a mobile device, web browser, or server-side.

Original Code

```
(function() {
    function add1(element, index, array) {
        array[index]=element+1;
    }
    function apply(8functions, integers) {
        if(!Array.isArray(functions)) {
             integers.forEach(functions);
        }
```

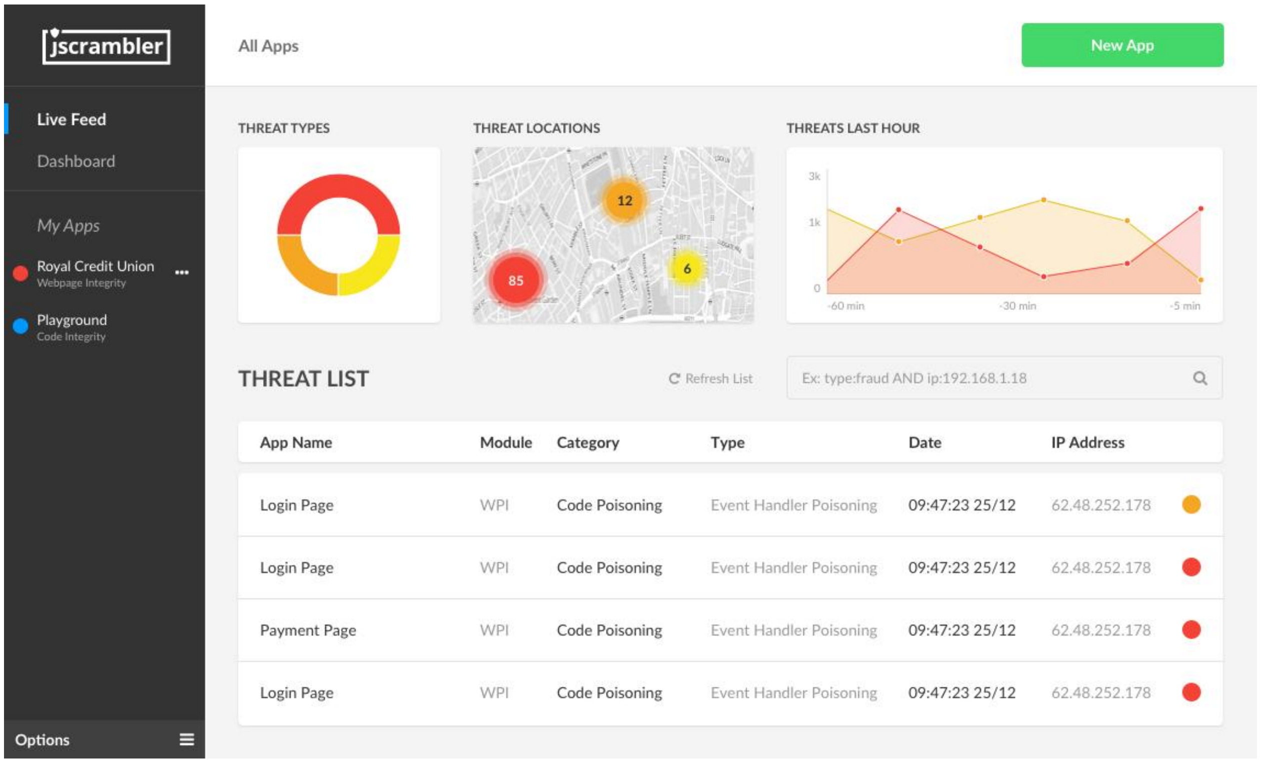
Obfuscated Code

```
/*...*/ ff)<<W);s=this[a](s,B);s=((s&0x1ff`
= A)|(s>>>Q` <)f);P^=s;P=((P&0x7`
H\"y)|(P>>>h);P=(P*z+((129.,0x187)<(91,0x1E0
)?(0x72,0xe6546b64):(120.7E1,0x35)))|R8H.j3H
;}s=` ##switch (t%x){case
X:s=(J[I](H+v)&0xff)<<c;` 8 v:s|` 5$R8H.t3H`
;$e` >!` 0\"` ?&`
;\";s=this[a](s,B);s=((s&0x1ff` [ A)|(s>>>Q`
<)f);P^=s;}P^=t;P^=P>>>c;P` :$P,0x85ebca6b`
G P>>>y` 2*c2b2ae35` 8$c;return P;}};var
r=K.T(\"3af\")?\"\":{\'F\':{(49.80E1>(0xAF,1
3.)?(22.40E1,0):65.>(137.8E1,0x209)?(74,0x84
):(0x1AF,46` K )),E=L>o,G;for
(;n0N<l0N.length;n0N++) /*...*/
```

This code protection is applied in a polymorphic fashion. As so, each new version of the protected code will be entirely different, which goes in line with the **Deception, Diversity,** and **Unpredictability** recommendations by MITRE.

Moving beyond the *Recon* stage, mitigating Web Supply Chain Attacks during runtime can be achieved with Jscrambler's **Webpage Integrity** module.

Webpage Integrity is a **client-side monitoring** solution that detects any tampering to the DOM during runtime, independently of its delivery mechanism. As such, Webpage Integrity **detects in real-time** when any compromised third-party script or code dependency attempts to exploit the application's end user.



App Name	Module	Category	Type	Date	IP Address	
Login Page	WPI	Code Poisoning	Event Handler Poisoning	09:47:23 25/12	62.48.252.178	●
Login Page	WPI	Code Poisoning	Event Handler Poisoning	09:47:23 25/12	62.48.252.178	●
Payment Page	WPI	Code Poisoning	Event Handler Poisoning	09:47:23 25/12	62.48.252.178	●
Login Page	WPI	Code Poisoning	Event Handler Poisoning	09:47:23 25/12	62.48.252.178	●

Framing this into some of the major Web Supply Chain Attacks to date:

- For **Magecart attacks**, Webpage Integrity immediately registers changes to the DOM when the skimmer loads on the end-users' device. **It triggers a warning in real-time** to the affected companies, with details that a serious credit card skimming threat is running on the payment pages. **Companies can react in real-time to the attack**, removing the infected script or blocking the compromised pages before the attack is able to reach a second end-user.
- For **cryptojacking attacks** such as the BrowseAloud (ICO) one, Webpage Integrity identifies the injected crypto miner script as soon as it starts being served. This **enables companies to immediately block the malicious script** without shutting down their application and before users are affected.

- For **attacks that compromise a code dependency**, once a compromised version of the application goes into production, Webpage Integrity is able to identify it, provided that the attack results in a malicious change to the DOM — which it did on the event-stream incident. Much like the previous examples, **this real-time detection promotes a timely and accurate response**.

In the current panorama of Application Security, there's no infallible way of being sure malicious code or markup isn't injected into companies' applications. The next best thing is to gain visibility about malicious injections and be able to react in real-time.

Past Web Supply Chain Attacks have one common metric that contributed to their magnitude: the very long time from attack to detection.

By detecting Web Supply Chain Attacks in real-time, Webpage Integrity enables companies to react **instantly** and mitigate them before any serious damage occurs.

Jscrambler's Code Integrity and Webpage Integrity solutions provide a holistic client-side security solution to mitigate Web Supply Chain Attacks — both proactively and reactively. Jscrambler can be easily integrated with existing SIEM, enabling an **in-depth security** solution that is simple to deploy and easy to use.

Limiting Third-Party Scripts and Code Dependencies

It's unfeasible to advocate putting an end to third-party code, even if it is the simplest way to prevent Web Supply Chain Attacks. The next best option is to **limit its use**.

Development teams must take a "lean" approach to address dependencies: sticking to the vital ones which can't be developed/maintained in-house. The same goes for third-party scripts; these represent add-ons which are not crucial to the application. Companies must prevent integrating unnecessary, risky, or deprecated scripts.

Each extra code dependency or external script must be seen as a substantial increase to the application's attack surface.

Assessing Third-Party Suppliers' Security

Even with a "lean" approach to third-party code, some will still prevail out of necessity. Organizations can still have some control over minimizing these risks. A suitable approach is to **assess the security level of third-party suppliers** during the procurement phase, on the following dimensions:

- **JavaScript code security** — the supplier should have in place a solution to protect JavaScript code, mitigating reverse-engineering and counterfeit code, as per MITRE's recommendations for **Deception** and **Substantiated Integrity**. The JavaScript security level can be assessed with peer-validated checklists[8].
- **JavaScript code polymorphism** — the supplier should guarantee that its protected code is polymorphic, *i.e.*, taking different forms for each new build. This helps increase the script's **Redundancy** and **Unpredictability**.
- **Subresource Integrity** — the supplier should provide the SRI token, enabling the procuring company to use SRI to ensure the integrity of loaded scripts.
- **Content Security Policy** — the supplier should have a CSP in place to limit the external sources to which they can send/receive content.

Content Security Policy (CSP)

CSP limits the external sources to which a website can connect. Trusted sources are whitelisted and every other connection is blocked.

[8] https://github.com/pfortuna/javascript-software-protections-checklist

In past Web Supply Chain Attacks, this simple security approach would have been able to protect the infected websites from data exfiltration, by blocking the external address to which attackers were sending stolen data.

Despite its ease and effectiveness, CSP presents some limitations. It requires substantial configuration and maintenance, is vulnerable to open-redirect attacks and can be bypassed CSP altogether, namely when Man-in-the-Browser (MitB) trojans and browser extensions are used as an attack vector.

Subresource Integrity (SRI)

Using Subresource Integrity (SRI) can serve as a simple yet effective prevention strategy against Web Supply Chain Attacks. By checking the file integrity, the application does not load scripts whose integrity check differs from the original, trusted script. As so, malicious scripts won't be loaded from third-parties and the Web Supply Chain Attack is mitigated.

However, **SRI comes with a major pitfall**: it's notably complex to apply to dynamic code. Most third-party script providers (like Google Analytics) keep improving their services, which results in frequent changes to the script itself. Adapting SRI to match this dynamic nature can be burdensome and, if SRI isn't properly set up, it can block a perfectly safe third-party script when it receives a much needed update. Also, SRI can be bypassed when a Man-in-the-Browser trojan is used as an attack vector.

A Multi-Layered Mitigation Approach

All of the mitigation strategies presented in this white paper go a long way into mitigating Web Supply Chain Attacks. However, an effective means of mitigation requires an approach which leverages most of these strategies.

The most advisable and effective mitigation of Web Supply Chain Attacks is a combination of CSP, SRI, limitation of external sources, third-party vetting, and resilient JavaScript protection, alongside webpage monitoring and real-time threat mitigation.

By pursuing this security-in-depth approach, companies can ensure that they gain total visibility and control over the client-side of their applications. This effectively translates to being able to prevent user data exfiltration and its ensuing liabilities.

Contact Us

If you want to know more about how Jscrambler can help you mitigate Web Supply Chain Attacks, don't hesitate to contact us

hello@jscrambler.com

+1 650 999 0010

Jscrambler is the leader in Client-Side Application Security Recognized in **Gartner's Market Guide for Online Fraud Detection** and in **Gartner's Market Guide for In-App Protection**