PCI DSS 针对恶意脚本防范的新要求及其方案探讨

atsec 唐冬生 2024 年 3 月
关键词：PCI DSS、Web Skimming、脚本安全性、防篡改、数据安全、客户端安全

2022 年 3 月，支付卡行业安全标准委员会（PCI SSC）公布了 PCI DSS v4.0 版本。新的 PCI DSS 版本中引入了有关恶意脚本防范的新要求，特别是条款 6.4.3 和条款 11.6.1。本文旨在探讨条款 6.4.3 和 11.6.1 的具体技术要求、对信息安全的影响以及如何实施其安全方案以满足这两项新要求。

1.PCI DSS v4.0 标准针对恶意脚本防范的新要求

1.1 PCI DSS 6.4.3 要求点

Requirements and Testing Procedures		Guidance
Defined Approach Requirements	**Defined Approach Testing Procedures**	**Purpose**
6.4.3 All payment page scripts that are loaded and executed in the consumer's browser are managed as follows: • A method is implemented to confirm that each script is authorized. • A method is implemented to assure the integrity of each script. • An inventory of all scripts is maintained with written justification as to why each is necessary.	**6.4.3.a** Examine policies and procedures to verify that processes are defined for managing all payment page scripts that are loaded and executed in the consumer's browser, in accordance with all elements specified in this requirement. **6.4.3.b** Interview responsible personnel and examine inventory records and system configurations to verify that all payment page scripts that are loaded and executed in the consumer's browser are managed in accordance with all elements specified in this requirement.	Scripts loaded and executed in the payment page can have their functionality altered without the entity's knowledge and can also have the functionality to load additional external scripts (for example, advertising and tracking, tag management systems). Such seemingly harmless scripts can be used by potential attackers to upload malicious scripts that can read and exfiltrate cardholder data from the consumer browser. Ensuring that the functionality of all such scripts is understood to be necessary for the operation of the payment page minimizes the number of scripts that could be tampered with. Ensuring that scripts have been explicitly authorized reduces the probability of unnecessary scripts being added to the payment page without appropriate management approval. Using techniques to prevent tampering with the script will minimize the probability of the script being modified to carry out unauthorized behavior, such as skimming the cardholder data from the payment page. **Good Practice** Scripts may be authorized by manual or automated (e.g., workflow) processes. Where the payment page will be loaded into an inline frame (IFRAME), restricting the location that the payment page can be loaded from, using the parent page's Content Security Policy (CSP) can help prevent unauthorized content being substituted for the payment page. *(continued on next page)*
Customized Approach Objective		
Unauthorized code cannot be present in the payment page as it is rendered in the consumer's browser.		
Applicability Notes		
This requirement applies to all scripts loaded from the entity's environment and scripts loaded from third and fourth parties. *This requirement is a best practice until 31 March 2025, after which it will be required and must be fully considered during a PCI DSS assessment.*		

在标准 6.4.3 已定义的方法要求（Defined Approach Requirements）中明确要求确保每个在消费者浏览器中被加载和执行的支付页面脚本都是经过授权使用的，并且是完整的没有被篡改的，同时还要求维护一个完整的脚本清单并书面说明为什么每个脚本都是必需的。

在标准 6.4.3 的指南（Guidance）中也明确提出了恶意脚本可能会造成的威胁，并指出“对于支付页面中加载和执行的脚本，其功能可以在实体不知情的情况下改变，也可以具有加载额外的外部脚本的功能（例如，广告和跟踪、标签管理系统）”。

1.2 PCI DSS 11.6.1 要求点

Requirements and Testing Procedures		Guidance
11.6 Unauthorized changes on payment pages are detected and responded to.		
Defined Approach Requirements	**Defined Approach Testing Procedures**	**Purpose** Many web pages now rely on assembling objects, including active content (primarily JavaScript), from multiple internet locations. Additionally, the content of many web pages is defined using content management and tag management systems that may not be possible to monitor using traditional change detection mechanisms. Therefore, the only place to detect changes or indicators of malicious activity is in the consumer browser as the page is constructed and all JavaScript interpreted. By comparing the current version of the HTTP header and the active content of payment pages as received by the consumer browser with prior or known versions, it is possible to detect unauthorized changes that may indicate a skimming attack. Additionally, by looking for known indicators of compromise and script elements or behavior typical of skimmers, suspicious alerts can be raised. *(continued on next page)*
11.6.1 A change- and tamper-detection mechanism is deployed as follows: • To alert personnel to unauthorized modification (including indicators of compromise, changes, additions, and deletions) to the HTTP headers and the contents of payment pages as received by the consumer browser. • The mechanism is configured to evaluate the received HTTP header and payment page. • The mechanism functions are performed as follows: – At least once every seven days **OR** – Periodically (at the frequency defined in the entity's targeted risk analysis, which is performed according to all elements specified in Requirement 12.3.1).	**11.6.1.a** Examine system settings, monitored payment pages, and results from monitoring activities to verify the use of a change- and tamper-detection mechanism. **11.6.1.b** Examine configuration settings to verify the mechanism is configured in accordance with all elements specified in this requirement. **11.6.1.c** If the mechanism functions are performed at an entity-defined frequency, examine the entity's targeted risk analysis for determining the frequency to verify the risk analysis was performed in accordance with all elements specified at Requirement 12.3.1. **11.6.1.d** Examine configuration settings and interview personnel to verify the mechanism functions are performed either: • At least once every seven days **OR** • At the frequency defined in the entity's targeted risk analysis performed for this requirement.	
Customized Approach Objective		
E-commerce skimming code or techniques cannot be added to payment pages as received by the consumer browser without a timely alert being generated. Anti-skimming measures cannot be removed from payment pages without a prompt alert being generated.		

根据标准 11.6.1 的具体要求，笔者认为需要满足以下技术关键点：

- 警告机制：当检测到消费者浏览器接收到的 HTTP 标头和支付页面内容发生未经授权的修改时，实施的系统应能够及时向有关人员发出警告。这些修改可能包括但不限于内容的更改、添加和删除。
- 评估功能：系统应具备评估接收到的 HTTP 标头和支付页面内容的能力，以确保这些内容的完整性和真实性没有被破坏。这意味着系统需要能够识别和验证页面内容的预期状态，并与实际接收到的内容进行比较。
- 定期执行：更改和篡改检测机制应至少每七天执行一次，或者根据对网站风险的分析来定期执行。这意味着组织需要根据其业务特性和面临的安全威胁水平，确定合适的检测频率，以确保及时发现并响应任何潜在的安全事件。

从以上 6.4.3 和 11.6.1 的要求，我们能看出标准的条款要求目标是通过实施有效的客户端安全防护和恶意脚本的检测机制，来保护消费者在浏览器端的支付信息安全和提升网站的整体安全性。

2.标准增加针对恶意脚本防范安全要求的原因分析

由此引发笔者思考的是：PCI DSS v4.0 中针对恶意脚本防范的安全要求为什么会延伸到了与消费者有关的客户端安全（Client-side Security）？

在回答这个问题之前，让我们先来了解 Web 应用程序的重要组成部分：客户端（Client-side）。

客户端应用程序通常由定制的 HTML、CSS 和 JavaScript 组成，当中会使用许多第三方库，这些第三方库经常与提供其自定义代码和库的第三方服务集成到同一客户端应用程序中。

所有这些第三方库或服务都在客户的浏览器中运行，而不是在应用程序所有者控制、管理和保护的服务器上运行。因此客户端的应用程序通常会与除原始服务器托管服务的应用程序以外的许多第三方服务进行交互，并向用户的浏览器提供客户端 JavaScript 核心元素，而这些不可控的第三方库或服务最终会导致客户端代码面临许多安全风险。

OWASP 团队目前已经针对客户端的安全风险收集了 10 个候选项，并计划最终发布，统计信息如下：

基于诸多实际项目经验，笔者对上述 10 个风险项进行了粗略的分析。其中第 5 个“Lack of Third-party Origin Control”是指未利用来源控制而增加了供应链风险。因为可能包含未知或不受控制的第三方代码，而这些代码可以访问站点源中的数据。而第 6 个“JavaScript Drift”的威胁主要来源于无法检测客户端使用的 JavaScript 资源代码的变化，包括无法检测这些代码的行为变化，以确定变化是否具有潜在的恶意性质。

以上第 5，6 两个风险项都与 PCI DSS v4.0 标准中的 6.4.3 和 11.6.1 技术要求所防范的恶意脚本的风险一致，这也表明 PCI DSS 标准紧跟 IT 产业技术更新和发展，及时针对信息安全风险给出响应和应对。

关于客户端安全的详情可以参考：https://owasp.org/www-project-top-10-client-side-security-risks/。

2.1 针对客户端的恶意脚本攻击安全事件

正是由于客户端安全面临越来越多的风险，除了以上列出的 Top10 风险外，还包括运行时可见性不足、脚本变更频繁以及由于快节奏的开发而导致的安全审查不足等。因此针对客户端的攻击方式也非常多，针对客户端的攻击事件而导致的数据泄露事件也是层出不穷，以下是几个实际发生影响较大的事件。

- **英国航空（British Airways）事件：**
  时间：2018 年 8 月至 9 月。
  影响：约有 380,000 张支付卡信息，包括姓名、地址、银行卡详情和 CVV 码被泄露，影响了约 500,000 名客户。
  攻击方式：Magecart 组织在 BA 网站上的行李认领信息页面添加了恶意软件。
  结果：英国航空公司侵犯客户隐私的行为被证明是一场 GDPR 灾难。欧盟 GDPR 监管机构处以 2.3 亿美元罚款，并将其归咎于英国航空网站的安全措施不力。严厉的罚款使得商业组织对其网站上的第三方服务提供商造成的任何数据泄露负责。
- **Ticketmaster UK 事件：**
  时间：2018 年 6 月。
  影响：Ticketmaster 宣布遭受信用卡泄露，攻击者利用第三方 Web 应用供应商 Inbenta，将扒窃器（skimmers）放置在其他第三方网站上。
  结果：传统的安全工具未能实时检测到问题。
- **Newegg 事件：**
  时间：2018 年 8 月 14 日至 9 月 18 日。
  影响：Newegg 是一个受欢迎的在线零售商，拥有数百万注册用户。攻击期间，桌面和移动应用用户都受到了影响，攻击者利用了与 Magecart 相关的支付卡扒窃器（skimmers）。
  结果：尽管确切的受害者数量和被盗记录未知，但攻击持续了近一个月，表明可能有大量受害者。

这些案例展示了针对客户端攻击的严重性和对在线业务的潜在威胁。也提醒我们对引用的第三方脚本和服务进行持续监控的重要性，以及在日常业务中维护安全和合规性的必要性。

**以上事件引用来源：*

https://www.reflectiz.com/blog/ico-fines-ticketmaster-uk-1-25-million-for-security-failures-a-lesson-to-be-learned/

https://www.reflectiz.com/blog/all-you-need-to-know-about-web-skimming-attacksss

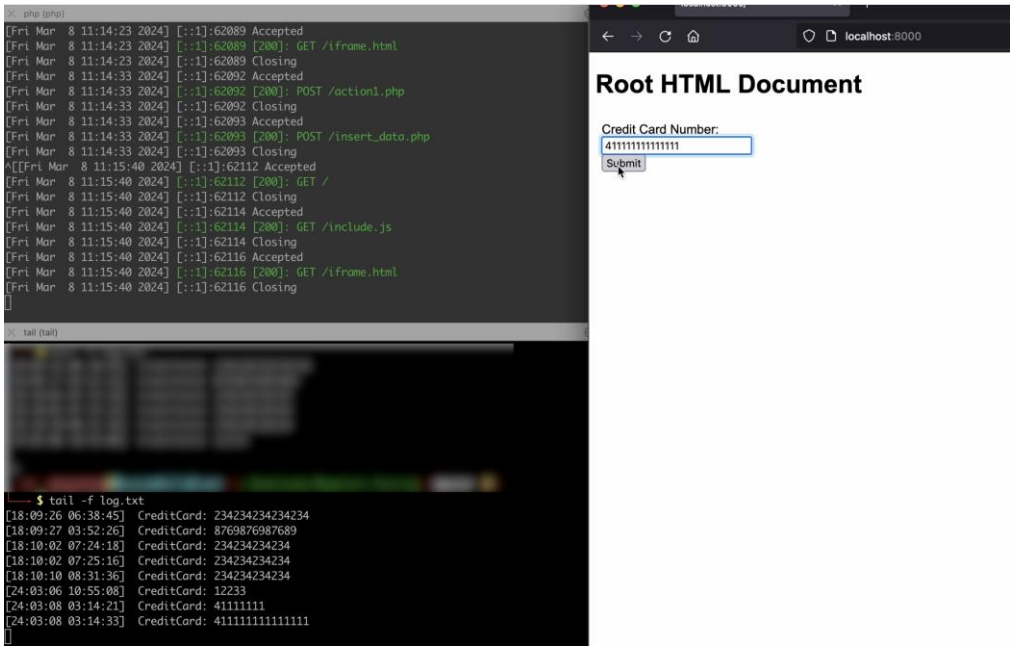
2.2 客户端的恶意脚本攻击方式分析

针对客户端的恶意攻击方式包括 XSS，Clickjacking，Session hijacking，Web skimming 和钓鱼攻击等，接下来我们来分析当中称之为“Web skimming”的专门针对

客户端的恶意脚本攻击方式，进而为如何防范针对客户端的恶意脚本攻击而建立基础。

Web skimming（也称为 Magecart attack）是一种针对数字业务的黑客技术。攻击者通过网站使用的第三方应用程序注入恶意脚本或恶意软件，从而破坏网站的支付页面。攻击者通常会在网站的支付页面上策略性地放置恶意 JavaScript 代码，并在不被发现的情况下收集信用卡信息和个人信息。

下图是笔者搭建的一个简单的测试页面，用于模拟黑客在 localhost:8000 页面插入恶意脚本后，用户输入的敏感信用卡信息被恶意读取并保存到本地 log 文件中的过程，而恶意的攻击者则会将用户每一次输入的信用卡信息保存到自己远程的服务器而不被用户发现。



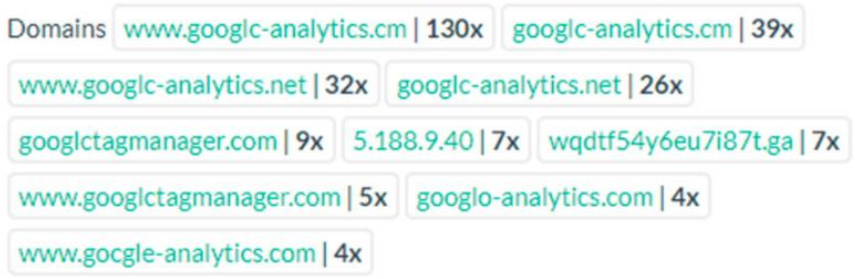
尽管这类攻击方式一次只能盗取一个用户的信息，但是如果被注入恶意脚本后的页面被大量用户用于输入和提交支付信息，则累积的信息量将是巨大的，就如同上一章节中的 Newegg 事件。

大规模和有针对性的 Web skimming 攻击可以通过多种方式进行，但通常涉及三个主要阶段：

01. **渗透**：攻击者通过各种手段获取网站的访问权限，例如：针对第三方供应商，利用网站中的脚本漏洞注入恶意代码或通过供应链攻击。

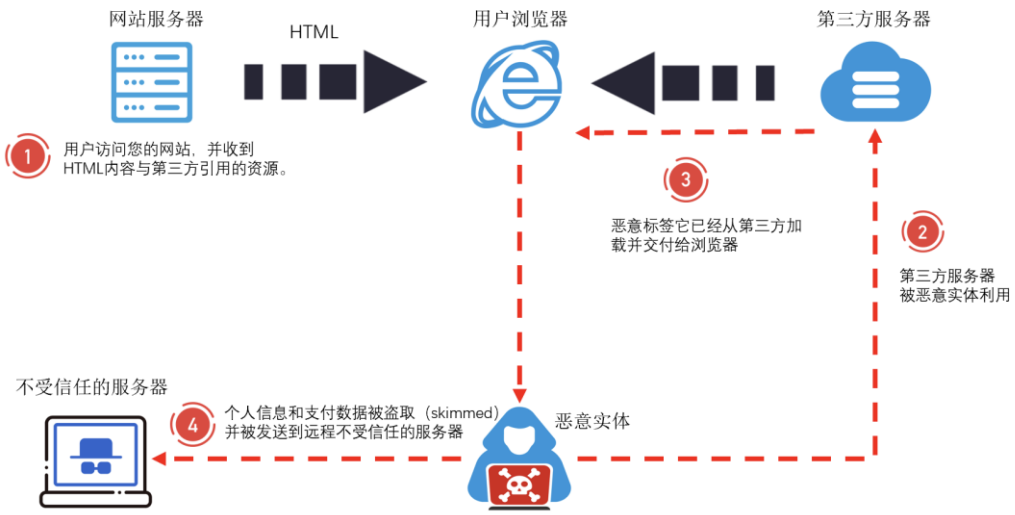
02. **植入**：攻击者不断发展植入方法来窃取敏感信息并避免被发现。攻击者植入的示例技术包括：

- 直接将恶意代码注入网站的敏感支付页面，以数字方式窃取信息。
- 通过代码注入或直接在真实网站上创建虚假付款表单。
- 重定向用户到有相似 URL 的欺诈网站上完成交易。
- 伪装已知的第三方供应商代码，例如 Google 跟踪代码管理器。如下图所示冒充 Google 分析或 Google 标签管理器的伪装域名：



03. **数据利用**：攻击者捕获客户输入的信用卡信息并将其发送到自己的服务器，可用于进行欺诈性购买或在暗网上出售。

下图完整的流程演示了 Web Skimming 是如何被恶意执行的：



3.针对客户端的恶意脚本防范的参考方案

了解 Web skimming 的攻击手法以及客户端恶意脚本攻击风险后，我们回到 PCI DSS 标准本身，来看标准是如何要求实施恶意脚本防护措施来保证持卡人数据安全的。

3.1 标准要求 6.4.3 的实施方案

我们首先来分析如何满足 PCI DSS v4.0 中 6.4.3 的具体要求。

- **必须实现一种方法来确认每个脚本都已获得授权**

针对这项要求，PCI DSS v4.0 建议通过工作流授权脚本或利用 Content Security Policy（CSP）防止未经授权的内容在支付页面上被替换。CSP 是一种安全机制，可让网站管理员识别允许在其网站上加载的内容来源。本质上 CSP 充当浏览器内防火墙，控制可以在消费者客户端加载内容的域，无论是脚本、图像还是其他资源。具体实现方法是通过 CSP 标签向 Web 服务器的响应添加特殊的 HTTP 标头来配置的。此标头向 Web 浏览器提供有关哪些内容源被认为是安全的并且应该允许在网页上执行的指令。

```
<!DOCTYPE html>
<html>
<head>
    <meta http-equiv="Content-Security-Policy" content="script-src 'self'">
    <title>CSP Example</title>
</head>
<body>
    <h1>Hello CSP!</h1>
    <script>
        // Legitimate script from the same origin
        console.log('This script is allowed by CSP.');
    </script>

    <script>
        // Malicious script from an external origin (blocked by CSP)
        console.log('This malicious script will not run due to CSP.');
    </script>
</body>
</html>
```

在上面的示例中，带有 Content-Security-Policy 标头的标记指定仅允许运行来自同一来源（self）的脚本。因此，同一源内的合法脚本将执行并向控制台记录消息，而来自外部源的恶意脚本将被 CSP 阻止并且不会执行。

关于 CSP 更多详细的使用可以参考：

https://developer.mozilla.org/zh-CN/docs/Web/HTTP/CSP

- **必须实施一种方法来确保每个脚本的完整性**

这项要求需要验证每个脚本的“完整性”。简而言之，即确保代码自被验证为非恶意以来保持完整性不变。可用于部分满足 6.4.3 要求的工具是 Subresource Integrity（SRI），它是 Web 浏览器中的一项安全功能，有助于确保从外部源加载的资源（例如脚本、样式表和字体）的完整性。SRI 的工作原理是允许您在 HTML 代码中与资源的 URL 一起包含哈希值。当浏览器加载资源时，它会计算其哈希值并将其与提供的哈希值进行比较。如果它们匹配，则该资源被认为是有效的并且未被篡改。

EXAMPLE 1

```
<script src="https://example.com/example-framework.js"
        integrity="sha384-Li9vy3DqF8tnTXuiaAJuML3ky+er10rcgNR/VqsVpcw+ThHmYcwiB1pbOxEbzJr7"
        crossorigin="anonymous"></script>
```

在上面的示例中，完整性属性是使用外部脚本的哈希值（sha384-Li9vy3DqF8tnTXuiaAJuML3ky+er10rcgNR/VqsVpcw+ThHmYcwiB1pbOxEbzJr7）设置的。哈希值是使用加密哈希函数（例如 SHA-384）计算的，并且对于脚本文件的内容来说是唯一的，如果脚本的内容发生变化，哈希值也会发生变化，浏览器将认为该脚本已被篡改而不对该脚本进行加载。

SRI 的使用详情可以参考：https://www.w3.org/TR/SRI/

Hash 值的生成网站可以参考：https://www.srihash.org

SRI 检测工具：https://github.com/4armed/sri-check

- **保留所有脚本的清单，并以书面形式说明为什么每个脚本是必要的**

第三个要求是维护一份清单，该清单需列出在支付页面上运行的所有脚本，并为每个脚本的必要性提供书面理由。此要求可确保仅执行已授权的脚本，并清楚的记录每个脚本对于支付页面功能至关重要的原因。从本质上讲，此要求旨在通过提高透明度和对与敏感支付数据交互的脚本的控制来增强安全性，这将有助于防止包含不必要的或有潜在风险的脚本，这些脚本可能会危及支付过程的安全性。通过记录和证明每个脚本的存在，组织可以更好地管理其 Web 应用程序的攻击面，并降低执行未经授权或恶意代码的可能性。

笔者认为可以通过一些可用于管理库存脚本的工具进行脚本的清单维护，例如配置管理数据库（CMDB），如 Ansible Tower、Ralph (https://github.com/allegro/ralph)、iTop (https://github.com/Combodo/iTop) 等。

3.2 标准要求 11.6.1 的实施方案

接下来我们来分析如何满足 PCI DSS v4.0 11.6.1 的具体要求，标准中列出了一些可用于遵守 11.6.1 要求的现有技术的指导和示例。

Requirements and Testing Procedures		Guidance
Applicability Notes The intention of this requirement is not that an entity installs software in the systems or browsers of its consumers, but rather that the entity uses techniques such as those described under Examples in the Guidance column to prevent and detect unexpected script activities. *This requirement is a best practice until 31 March 2025, after which it will be required and must be fully considered during a PCI DSS assessment.*		**Examples** Mechanisms that detect and report on changes to the headers and content of the payment page include but are not limited to: • Violations of the Content Security Policy (CSP) can be reported to the entity using the *report-to* or *report-uri* CSP directives. • Changes to the CSP itself can indicate tampering. • External monitoring by systems that request and analyze the received web pages (also known as synthetic user monitoring) can detect changes to JavaScript in payment pages and alert personnel. • Embedding tamper-resistant, tamper-detection script in the payment page can alert and block when malicious script behavior is detected. • Reverse proxies and Content Delivery Networks can detect changes in scripts and alert personnel. Often, these mechanisms are subscription or cloud-based, but can also be based on custom and bespoke solutions.

➢ 可以使用 report-to 或 report-uri CSP 指令向实体报告违反内容安全策略（CSP）的行为

在 6.4.3 的要求中，已经建议使用 CSP 对页面可以加载的资源进行授权处理，此处的 report-to 指令是 CSP Level 3 中引入的新特性，用于指定一个上报组（report group）。这个上报组包含了一个或多个报告接收端（Reporting Endpoints），用来接收关于网页中违反 CSP 的报告。与 report-uri 相比，report-to 提供了更灵活和可配置的方式来管理报告接收端。

发送到一个报告接收端的配置参考如下：

```
HTTP

Reporting-Endpoints: endpoint-1="https://example.com/reports"

Content-Security-Policy: …; report-to endpoint-1
```

发送到多个报告接收端的配置参考如下：

```
HTTP

Report-To: { "group": "csp-endpoint",
             "max_age": 10886400,
             "endpoints": [
               { "url": "https://example.com/csp-reports" }
             ] },
           { "group": "hpkp-endpoint",
             "max_age": 10886400,
             "endpoints": [
               { "url": "https://example.com/hpkp-reports" }
             ] }
Content-Security-Policy: …; report-to csp-endpoint
```

report-to 更多信息可以参考：https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Content-Security-Policy/report-to

report-uri 指令，根据官方的介绍，它用于指定一个 URL，该 URL 会接收到关于网页中违反 CSP 的报告。当浏览器检测到违反 CSP 的行为时，它会向指定的 report-uri 发送一个

POST 请求，将违规信息发送给服务器端进行记录和分析。这有助于网站管理员监控和修复 CSP 违规问题，提高网站的安全性。

上图的 Examples 中的/csp-violation-report-endpoint/可以使用以下的 PHP 代码来运行，该代码会记录详细违规的 JSON，如果是添加到日志文件的第一条违规记录，则发送电子邮件给管理员 admin@example.com。

```php
<?php

// Start configure
$log_file = dirname(__FILE__) . '/csp-violations.log';
$log_file_size_limit = 1000000; // bytes - once exceeded no further entries are added
$email_address = 'admin@example.com';
$email_subject = 'Content-Security-Policy violation';
// End configuration

$current_domain = preg_replace('/www\./i', '', $_SERVER['SERVER_NAME']);
$email_subject = $email_subject . ' on ' . $current_domain;

http_response_code(204); // HTTP 204 No Content

$json_data = file_get_contents('php://input');

// We pretty print the JSON before adding it to the log file
if ($json_data = json_decode($json_data)) {
  $json_data = json_encode($json_data, JSON_PRETTY_PRINT | JSON_UNESCAPED_SLASHES);

  if (!file_exists($log_file)) {
    // Send an email
    $message = "The following Content-Security-Policy violation occurred on " .
      $current_domain . ":\n\n" .
      $json_data .
      "\n\nFurther CPS violations will be logged to the following log file, but no further email notifications will be sent until this log file is deleted:\n\n" .
      $log_file;
    mail($email_address, $email_subject, $message,
        'Content-Type: text/plain;charset=utf-8');
  } else if (filesize($log_file) > $log_file_size_limit) {
    exit(0);
  }

  file_put_contents($log_file, $json_data, FILE_APPEND | LOCK_EX);
}
```

**提示：官方提示该 report-uri 指令不再推荐使用。虽然一些浏览器可能仍然支持它，但它可能已经从相关的 web 标准中删除，或者可能正在被删除，现在只是出于兼容性的目的而*

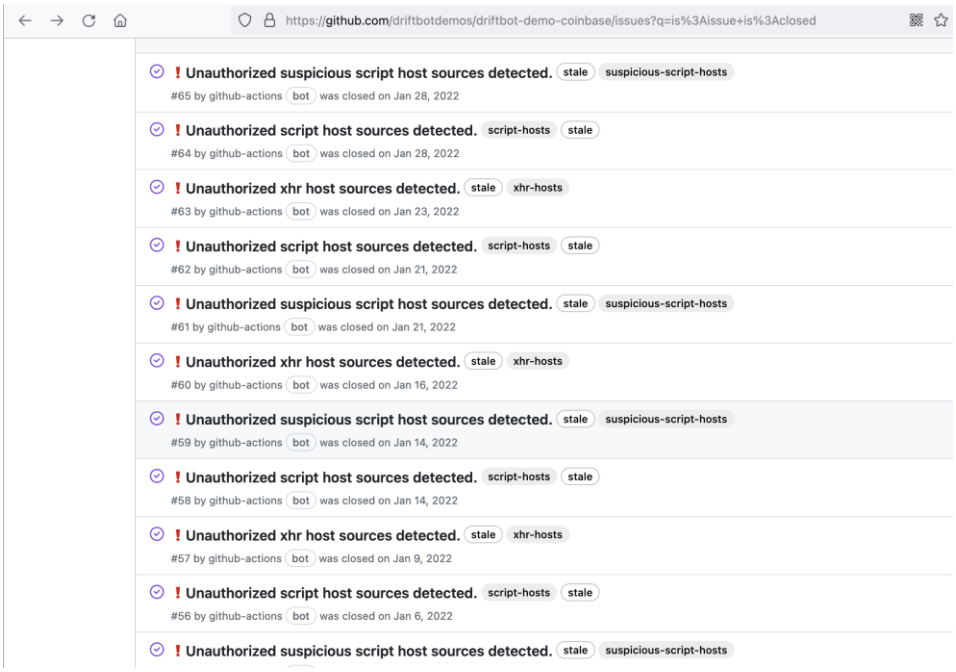
保留。report-uri 详情可以参考：https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Content-Security-Policy/report-uri

- **由请求和分析收到网页的系统进行外部监控（也称为合成用户监控）可以检测到支付页面中 JavaScript 的变更，并向人员发出警报**

Synthetic User Monitoring（合成用户监控），是一种监控和测试网站性能的方法，通过模拟真实用户在访问网站时的行为来评估网站的性能和可用性。Synthetic User Monitoring 通常会使用自动化工具或脚本来模拟用户与网站的交互，例如点击链接、填写表单、浏览页面等，以便检测潜在的问题并及时发现性能瓶颈。

为满足标准要求，机构可以通过编写自动化的脚本来模拟用户提交支付请求，通过将消费者浏览器接收到的 HTTP 报头的当前版本和支付页面的活动内容与先前或已知版本进行比较，可以检测到 skimming 攻击相关的未经授权的更改。

基于笔者的产业调研，分享一款开源的工具（Driftbot）可供参考使用，该工具在安装配置成功后会默认每隔 4 个小时进行一次检测，并有类似如下图的监控结果：



该工具更详细的使用参考链接如下：

https://driftbot.io/howto/

https://github.com/joshlarsen/driftbot

- **嵌入篡改防范、篡改检测脚本到支付页面中，可以在检测到恶意脚本行为时发出警报并进行阻止**

实施嵌入篡改检测脚本的操作分步指南：

01. **了解网站架构：** 在部署篡改检测脚本之前，了解网站结构非常重要。其中包括了解并确定脚本的加载位置及其功能。

02. **创建篡改检测脚本：** 篡改检测脚本是一段 JavaScript 代码，用于检查网页的完整性。它可以像计算页面 HTML 的哈希值并将其与已知的正确哈希值进行比较的脚本一样简单。如果哈希值不匹配，脚本会警告发生的篡改情况。以下是篡改检测脚本的简单示例：

```javascript
var pageHash = 'known-good-hash';
var currentHash = calculateHash(document.documentElement.outerHTML);
if (pageHash !== currentHash) {
    alert('Page has been tampered with!');
}
```

03. **将脚本嵌入到网页中：**创建篡改检测脚本后，需要将其嵌入网页中。通常可以通过在 HTML 中添加标记来实现。建议将其加载到文档的头部，以便尽快检测篡改。
04. **测试脚本：**嵌入脚本后，应该对其进行测试以确保其正常工作。可以采取手动篡改的页面以查看脚本是否正确检测到更改的方式来进行。
05. **监控警报：**脚本上线后，需要监控它生成的警报，在检测到篡改时发送电子邮件或短信。

- **反向代理和内容交付网络可以检测到脚本的变更，并向人员发出警报**

该方案可以和 CDN 厂商来确认是否支持，经过调研初步了解部分支持客户端安全保护的 CDN 厂商的安全防护功能，如下：

Akamai 的客户端保护功能，详情参考：

https://www.akamai.com/resources/reference-architecture/client-side-protection-compliance

Cloudflare 的客户端保护功能，详情参考：

https://www.cloudflare.com/application-services/products/page-shield/

4.总结

PCI DSS v4.0 中 6.4.3 与 11.6.1 的安全要求目前阶段被认为是最佳实践，自 2025 年 3 月 31 日起将成为强制性要求，届时所有 PCI DSS v4.0 评估都将必须考虑这些要求。故而 atsec 建议被评估机构能够尽可能早的投入到相关标准的合规建设，从而更好地提高信息安全防护。

以上针对客户端的恶意脚本攻击的实施方案，机构可以根据实际情况进行选择使用，为了确保其符合您的安全策略，在实施任何措施之前对其进行彻底评估非常重要。定期风险分析可以帮助识别潜在的漏洞和威胁，采用最佳实践可以检测和防止恶意脚本的攻击，从而增强整个系统的安全性。

总之，PCI DSS v4.0 中对客户端恶意脚本防护的新要求将有助于应对不断演变的安全威胁，维护支付系统的安全和保护持卡人账户数据安全，并最终推动整个支付卡行业的安全标准提升。

参考资料：

1) https://blog.nasstar.com/magecart-time-to-focus-on-web-security-to-mitigate-digital-skimming-risk/
2) https://www.reflectiz.com/blog/the-gocgle-web-skimming-campaign/
3) https://en.wikipedia.org/wiki/Web_skimming
4) https://owasp.org/www-project-top-10-client-side-security-risks/
5) https://www.imperva.com/learn/application-security/magecart/
6) https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Content-Security-Policy/report-to#syntax
7) https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Content-Security-Policy/report-uri
8) https://blog.sucuri.net/2023/04/how-to-set-up-a-content-security-policy-csp-in-3-steps.html
9) https://east.pcisecuritystandards.org/document_library?category=pcidss&document=pci_dss
10) https://developer.mozilla.org/en-US/docs/Web/Performance/Rum-vs-Synthetic
11) atsec 官方网站：https://www.atsec.cn
12) PCI SSC 官方网站：https://www.pcisecuritystandards.org/