Global Magecart Campaign Puts Banks Under Pressure

By raptur3

Archived: 2026-04-05 19:50:17 UTC

A large-scale magecart operation remained active for over 24 months, leveraging an infrastructure of 100+ domains. While the targeted victims are e-commerce websites, the actual pressure falls on banks and payment systems.

As ANY.RUN's analysis shows, threat actors applied multi-step checkout hijacking, payment page mimicry, and WebSocket-based exfiltration of card data.

This report provides both executive-level insights and technical analysis of the campaign.

Key Takeaways

- The campaign demonstrates **long-term persistence** (24+ months) supported by highly resilient infrastructure.
- **Banks** (not merchants) **bear the primary impact**, as stolen card data leads to fraud losses and reputational risk.
- Payment system mimicry (notably Redsys) **significantly increases attack success** by embedding fraud into trusted user flows.
- Use of **WebSocket exfiltration** reduces visibility in traditional security monitoring tools.
- Multi-stage, dynamically delivered payloads allow attackers to **adapt quickly** and evade disruption.
- The campaign is global but **regionally tailored**, leveraging localized payment ecosystems to enhance credibility.

Campaign Overview

A large-scale magecart operation has been identified, active for at least 24 months and supported by over 100 domains. In observed cases, threat actors deployed a multi-stage checkout hijacking framework, incorporating:

- Payment step substitution
- WebSocket-based exfiltration of payment card data
- Payment page mimicry, including infrastructure-level impersonation of legitimate providers (notably Redsys)
- Dynamic frontend adaptation of payment interfaces matching different storefronts and scenarios

A total of 17 WooCommerce websites were infected between February 2024 and April 2025 and are likely linked to this campaign, reflecting its longevity and operational stability.

Industrial and Regional Context Behind Global Impact

The geographic scope is of the campaign is global. Among the victims are organizations from at least 12 countries, including the United Kingdom and Denmark. However, there's a notable concentration of such incidents in Spain, France, and United States.

Some cases are confirmed directly via telemetry and network traffic, while others are identified via infrastructural correlation.

From an industry perspective, mostly retail e-commerce companies were targeted, although in some cases, non-commercial organizations have been affected, too.

However, the primary pressure here falls on banks, as cardholders faced financial exposure and their trust in payment systems suffered.

Why Redsys and Spanish Payment Context Stand Out

Despite the global impact, the ties to Spain and its payment ecosystem in particular are obvious in this magecart campaign.

Mimicry of RedSys, a payment system used in Spain, lies in the foundation of the attacks. The campaign infrastructure features domains and visual artifacts designed to fit Spanish payment context. In some cases, user payment flows included legitimate Redsys domain sis.redsys.es for added credibility.

The approach made the malicious activity of the campaign convincing within Spanish payment context.

What Makes This Campaign Durable

Payment Mimicry

A significant portion of the infrastructure is registered via NICENIC INTERNATIONAL GROUP and disguised as legitimate web services, including analytics platforms, CDN resources, jQuery libraries, andpayment services. If you access them directly, they'll act as technical placeholders or will simulate legitimate redirects. This complicates attribution.

Multi-Stage Delivery Architecture

The injected JavaScript contains only a minor loader that connects to external infrastructure, receives configuration data, and loads the next stage. The loader uses the fallback mechanism: it iterates through backup domains until a valid response is received. This allows the campaign to go on even if some components of the infrastructure get blocked.

Dynamic Payload Delivery

The next stage isn't openly stored inside an infected file. It's delivered dynamically via a staging response. Thanks to this, the operators modify delivery domains, payload paths, and control infrastructurewithout infecting the website again.

Different domains aren't necessarily serve different campaigns. Instead, they have different roles: staging responses, payload delivery, or for WebSocket/C2 and command handlers.

Other Factors

- State persistence in localStorage
- Masquerading as legitimate external dependencies
- WebSocket usage as a channel for control and exfiltration

As a result, the compromised website becomes only an initial access point. Subsequent payload delivery and data exfiltration can be flexibly modified inside the external infrastructure.

Technical analysis

Initial Loader Delivery and Execution

Following the compromise of a website, attackers modify one of the site's embedded JavaScript files with a small, obfuscated loader. It doesn't feature the main card-stealing logic but acts as an initialdelivery tool. It executes in the victim's browser and receives parameters for the next stage from external infrastructure.

Injected JavaScipt

Next, the obfuscated part of the loader refers to one of the pre-determined domains from the fallback infrastructure list. It returns a JSON configuration featuring the next stage's address, WebSocket/C2 server address, and an extra HTTP handler for auxiliary communication.

Domain examples

These values are delivered as encoded arrays of numeric character codes, which are then decrypted in the victim's browser.

An example of JSON configuration. ANY.RUN Interactive Sandbox

In case no response was received or the JSON was invalid, the loader automatically switches to the next domain from the list. **This mechanism ensures continued operation even in the presence of partial infrastructure disruption or blocking.**

Stage 1: Malicious Payload Delivery and Execution

After receiving a valid staging response, the loader takes the URL of the next JavaScript and dynamically adds it to the DOM via a new <script src=…> element.

Code fragment responsible for the execution of the malicious activity

At this point, the primary malicious payload is loaded into the page. Notably, this payload may be delivered from different domains, such as:

jquerybootstrap[.]com

newassetspro[.]com

assetsbundle[.]com

bundlefeedback[.]com

and others.

In any case, the delivery stage is the same. The operators **rotate payload sources** to increase the infrastructure's durability.

Stage 2: Payment Step Activation

After loading, the main payload begins executing within the context of the store's webpage and waits for the checkout/payment DOM to appear.

At this stage, it:

- monitors the opening of the payment step;
- interacts with checkout elements;
- replaces or overlays the legitimate payment interface;
- injects its own elements, including iframes and custom buttons;
- hides the real payment confirmation elements.

Once checkout is loaded, payment hijacking begins.

Observed Code Patterns Indicative of Payment Hijacking

Delayed activation ensures the user follows through until they reach the required payment step

Attackers conceal the legitimate payment button and replace it with a fake one

The script not only runs in the background but fully overlays/replaces the interface

The form isn't static but controlled and manageable

In some cases, the mimicry is built around a payment scenario that is visually and logically close to a legitimate PSP flow. In cases related to Spain Redsys mimicry is especially notable, but payment overall can a**dapt to storefronts, countries, and local PSPs.**

Script Deobfuscation

The core payload waits for the checkout form to appear and is responsible for the reception, validation, and sending payment data from the fake payment form.

Notable Code Features Inside the Script

The payload adapts to user environments with frontend localization capabilities and supports multiple languages: English, Spanish, Arabic, French.

There's a state machine with the following states: init, return, confirm, alert, getData, allowing for controlled progression through the attack lifecycle.

Code for handling WebSocket connections to the C2 server for the control of the attack flow. Part 1.

Code for handling WebSocket connections to the C2 server. Part 2

An example of the final result of the mimicry can be seen below:

Base64-encoded HTML page is responsible for displaying a fake payment interface

PayPlug SAS payment window imitation

There's a heavily obfuscated JavaScript inside the HTML page. It uses techniques like that to avoid detection:

- **Anti-tampering**: code integrity is verified via function serialization, as well as bitwise & arithmetical operations.

Code fragment confirming anti-tampering

- **Virtualization**: Custom VM's opcodes, symbolic execution, code strings executed via eval call.

A fragment of the raw load

VM’s opcode description fragment

The strings that are stored in an obfuscated form are decrypted using the VM:

Raw obfuscated strings

Deobfuscated strings

The payload is responsible for the formatting and validation of Visa/Mastercard payment data that are entered into the fake form, as well as UI state modification, and event or data delivery via postMessagemethod:

PostMessage method for data delivery

Stage 3: Connecting to Control Infrastructure

After activation, the malicious payload establishes a connection to the control infrastructure, e.g., via WebSocket.

W*ebSocket exfiltration code*

This channel is used for:

- transmitting service events;
- sending BIN (Bank Identification Number) data;
- transmitting full payment card details;
- receiving additional commands to control the replaced payment flow.

In one of the analyzed cases, WebSocket was used as the primary channel for card data exfiltration, while the C2 server was disguised as a Redsys domain (redsysgate[.]com).

During the skimmer’s operation, it retrieves malicious JavaScripts from URLs that look like so:
hxxps://<c2_domain>/<base64_text>.js?_=<digits>

Then, WebSocket connections are used for control and data transmission at:
wss://<c2_domain>/?token=<base64_data>

When the user enters their data, an event is sent containing the exfiltrated information. In response, the server provides instructions on what to do next and what content to display, such as the logo of the payment system associated with the entered card (Visa/MasterCard).

Card data (random numbers used an example) in a code fragment

This is important for the understanding of the campaign: attackers are not simply stealing card data, they **embed exfiltration** into a seemingly legitimate payment context.

Stage 4: Interception and Transmission of Payment Data

When a user enters their card details into the spoofed payment interface, the payload takes them to the attackers' external infrastructure.

The following data was being transmitted in network traffic:

- BIN
- full card number
- expiration date
- CVV

The transmission does not occur via a standard form POST request, but instead through a separate WebSocket channel, making detection via conventional HTTP logs more difficult.

Importantly, within the same cluster, **the visual scenario of the attack may vary**. In some cases, Redsys-themed mimicry is observed; in others, PayPlug-like or generic card form scenarios are used.

This does not necessarily indicate different campaigns: within a single malware family, the same loader, staging infrastructure, and exfiltration mechanism may be reused while applying different front-end disguises.

Additional Vector: Distribution of Android APK via the Same Inject

In addition to manipulating the payment step and stealing card data, the same malicious payload was also used as a platform to push the installation of an Android application in APK format.

The script checked the user's environment and, if certain conditions were met, displayed a separate mobile scenario offering the user to download an app. This included promises of discounts or bonuses, along with instructions on how to enable installation from "Unknown Sources."

Based on the contents of the payloads, this scenario was localized into at least several languages, including English, Spanish, Arabic, and French. This indicates that the campaign was targeting a broad international

audience and relied on a prepared, rather than ad hoc, infrastructure.

Code fragment for Android-specific flow

This scenario had several localization options, including English, Spanish, Arabian, and French, indicating the campaign's global focus targeting particular, not random infrastructures.

Conclusion

This magecart campaign reflects a shift from opportunistic skimming toward structured, infrastructure-driven payment attacks. By combining checkout hijacking, high-fidelity payment mimicry, and real-time exfiltration, attackers embed malicious activity directly into legitimate transaction flows. This not only increases effectiveness but also complicates detection and response.

Deep visibility into active attacks and continuous threat monitoring are required for efficient detection and prevention of such breachers.

About ANY.RUN

ANY.RUN delivers interactive malware analysis and actionable threat intelligence, enabling security teams to investigate threats more efficiently, gain clearer visibility into attacker behavior, and respond with greater confidence.

We focus on:

- Maintaining SOC 2 Type II certification and a strong commitment to safeguarding customer data
- Continuously enhancing our Interactive Sandbox, Threat Intelligence Lookup, and Threat Intelligence Feeds to support monitoring, triage, and incident response workflows
- Enabling SOC and MSSP teams to accelerate analysis, improve investigative context, and detect emerging threats at early stages

Analysis and Investigation Data

Link to TI Lookup query

Browse TI Lookup for related threats

Links to sandbox analyses

Case 1: Confirmed checkout hijacking and WebSocket exfiltration of BIN, PAN, expiry date, and CVV.

View analysis

Case 2: The same loader cluster and staging infrastructure but without confirmed card exfiltration (possibly due to redirection to a legitimate external payment flow)

View analysis

Case 3: Confirmed use of the same loader cluster and staging infrastructure.

View analysis

Indicators of Compromise (IOCs)

Payload URL: hxxps[:]//<c2_domain>/<base64_text>.js?_=<digits>

C2 WebSocket URL: wss[:]//<c2_domain>/?token=<base64_data>

bundle-feedback[.]com

doubleclickcache[.]com

analyticsgctm[.]com

hotjarcdn[.]com

firefoxcaptcha[.]com

solutionjquery[.]com

jquerybootstrap[.]com

assetsbundle[.]com

bundle-referrer[.]com

categorywishlist[.]com

cachesecure[.]com

securedata-ns[.]com

analysiscache[.]com

newassetspro[.]com

explorerpros[.]com

redsysgate[.]com

khr0x

I'm 21 years old and I work as a malware analyst for more than a year. I like finding out what kind of malware got on my computer. In my spare time I do sports and play video games.

raptur3

Network Analyst at ANY.RUN | + posts

Network Analyst at ANY.RUN. Keen to become a 'cybersec Swiss Army knife' man. Enjoys reading and writing deep-dive tech research.

I'm 21 years old and I work as a malware analyst for more than a year. I like finding out what kind of malware got on my computer. In my spare time I do sports and play video games.

raptur3

Network Analyst

Network Analyst at ANY.RUN. Keen to become a 'cybersec Swiss Army knife' man. Enjoys reading and writing deep-dive tech research.
