Deobfuscating and hunting for OSTAP, Trickbot's dropper and best friend

intrinsec.com/deobfuscating-hunting-ostap/

Equipe CERT

April 14, 2020

During **a recent investigation dealing with ransomware attack**, CERT Intrinsec faced OSTAP loader. This loader is **used to deliver other malwares** (such as Trickbot) on an infected system. It uses high obfuscation techniques to prevent the code from being read and to bypass detection processes.

Obfuscated loader

Figure 1 : Extract of the loader code (Javascript)

The ostap loader (Figure 1) we analysed was about 10 000 lines long. We started the **static analysis** by going through the code by hand to understand its structure. We identified a key part of the code that helped us to deobfuscate the loader (Figure 2).

```
} catch (a) {
    Trf4d = typeof(a);
    Ikol5 = 670;
    Rf8y = 'rom' + 'CharCode';
};
Rf9i = (Trf4d + 'String')['slice'](((this['Math'] + '').length) - 19);
Rfii6 = '';
var bDjSteWd = function () {
    return 0;
};
Trf4d = [];
function bDjSteW(ewnfBeth8, etvulike2) {
    try {
        ppfhair_3(ewnfBeth8);
    } catch (a) {
        if (etvulike2 != 'f') {
            return 1;
        } else {
            Tivczza = this[Rf9i][[Rfii6 = etvulike2 + Rf8y]](ewnfBeth8);
            try {
                return Trf4d(etvulike2);
            } catch (l) {
                return Tivczza;
            };
        }
        return bDjSteWd;
    }
};
```

Figure 2 : Key part of the loader code

The instructions, on *Figure 2*, aim at executing *String.fromCharCode* function **ewnfBeth8** parameter. There are lots of noise instructions in the program. For example, **ppfhair_3(ewnfBeth8)** instruction in *try* statement will never be triggered because the function does not exist. It is done on purpose to always enter the *catch*. Besides, **etvulike2** parameter is always equal to **'f'**. A large part of the program consists of a concatenation of functions such as the one shown on *Figure 3*.

```
+ (function (pkucu3) {
    pkucu3[Ikol5] = 1;
    pkucu3[Ikol5 + 5] = 70;
    return bDjSteW(bDjSteWd() + (pkucu3[675] - pkucu3[Ikol5]), (function (sfpyo5) {
            sfpyo5[Ikol5] = 2;
            sfpyo5[Ikol5 + 5] = 104;
            return bDjSteW(bDjSteWd() + (sfpyo5[675] - sfpyo5[Ikol5]), 'f');
        })(Trf4d, null, null));
```

Figure 3 : Repeated function model

The action of the function above (*Figure 3*) is to apply *String.fromCharCode* to 69, i.e. « E ». **The program uses this method to set all its instructions**. Knowing that, we decided to write a script to extract each obfuscated character.

Deobfuscation script

The main goal of the script is **to get indicators of compromise from the loader.** It has been developed using Node JS. It first goes through the obfuscated loader, retrieves the targeted numbers and apply *String.fromCharCode* to decode them. Then, it collects the indicators of compromise in the decoded payload using regular expressions. Extracted IOCs are IPs, URLs and User-Agents. The figure below represents the output of the script using a sample hunted on VirusTotal. We can see, at the top of *Figure 4*, a list of file extensions that are targeted (their content will be replaced by the Ostap JS code).

Figure 4 : Script execution output

Hunting

After deobfuscating the loader as a part of our investigation, **we decided to hunt recent and similar files on VirusTotal, using searches on static code patterns** (*content: "'String')['slice']"*, for instance). We found lots of samples (*Figure 5*) and process them so as to extract as many IOCs as possible.

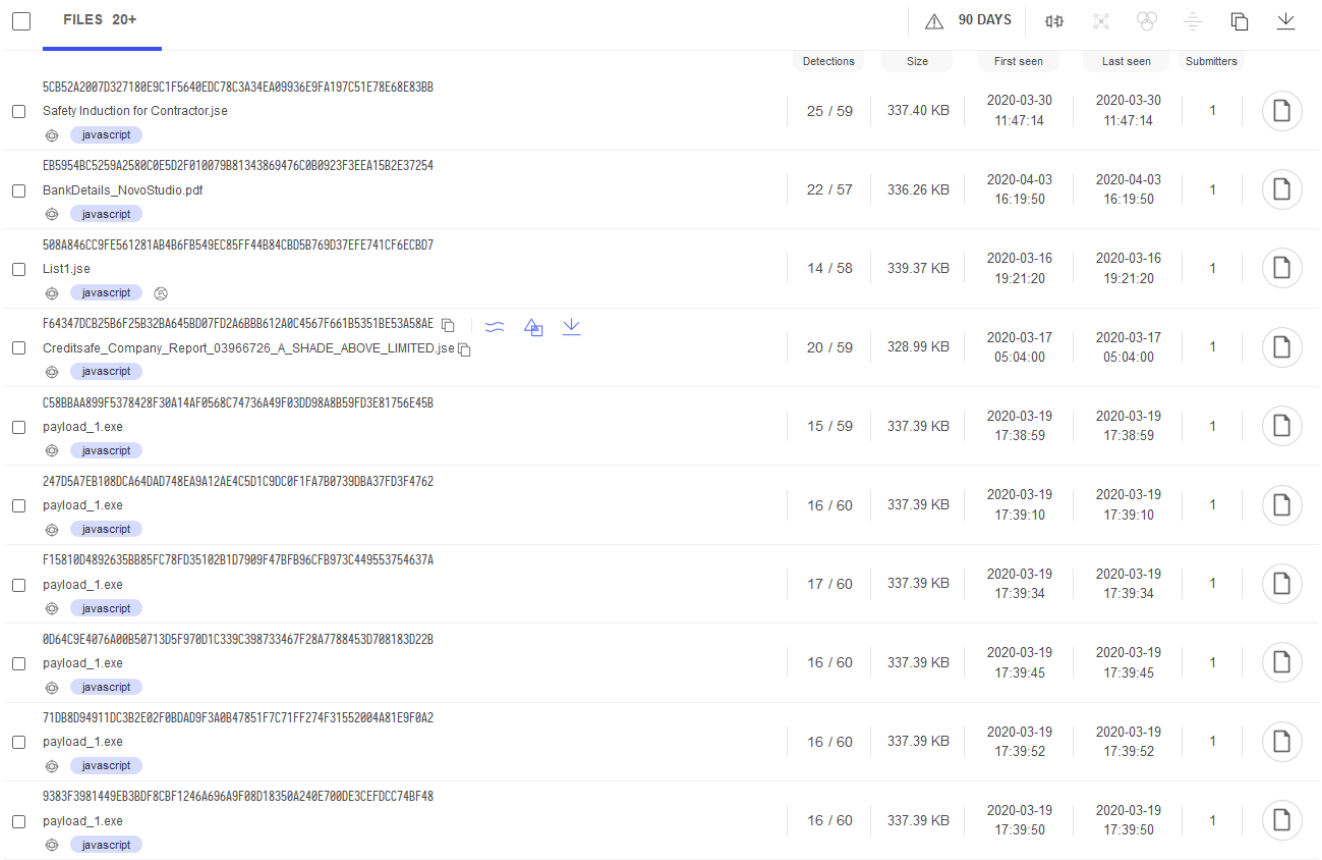

Figure 5 : Ostap samples from VirusTotal

We collected about **140 samples from VirusTotal using the script**. We analysed them and extracted the indicators of compromise presented in the table below. We can say that at least one of the IP addresses (185[.]234[.]73[.]125) **is related to the Trickbot campaign happening since the Coronavirus appeared such as in Italy, as reported by Sophos** (1).

IP	URL	User-Agent
141[.]98[.]214[.]14	hxxps[://]141[.]98[.]214[.]14/6BcsTO/AGVV5r[.]php	Mozilla/5.0
185[.]159[.]82[.]205	hxxps[://]185[.]159[.]82[.]205/2/1[.]php	(Windows NT 6.; Win64; x64; Trident/7.0; rv:11.0) like Gecko
185[.]216[.]35[.]10	hxxps[://]185[.]216[.]35[.]10/VYut68/L2KSUN[.]php	
185[.]234[.]73[.]125	hxxps[://]185[.]234[.]73[.]125/wMB03o/Wx9u79[.]php	
194[.]87[.]96[.]100	hxxps[://]194[.]87[.]96[.]100/2/1[.]php	
45[.]128[.]133[.]41	hxxp[://]45[.]128[.]133[.]41/jTlp8P/3OXkud[.]php	
91[.]196[.]70[.]126	hxxps[://]91[.]196[.]70[.]126/2/zsQX9M[.]php	

References

1. https://news.sophos.com/en-us/2020/03/04/trickbot-campaign-targets-coronavirus-fears-in-italy/
2. https://www.esentire.com/blog/oh-snap-new-ostap-variant-observed-in-the-wild
3. https://www.bromium.com/deobfuscating-ostap-trickbots-javascript-downloader/
4. https://blog.trendmicro.com/trendlabs-security-intelligence/latest-trickbot-campaign-delivered-via-highly-obfuscated-js-file/
5. https://github.com/cryptogramfan/Malware-Analysis-Scripts
6. https://www.cert.pl/en/news/single/ostap-malware-analysis-backswap-dropper/
7. Link to the script on Intrinsec Github : https://github.com/Intrinsec/CERT/tree/master/Scripts/ostap_deobfuscator

Want to learn more about our Computer Emergency Response Team (CERT) ?

Discover