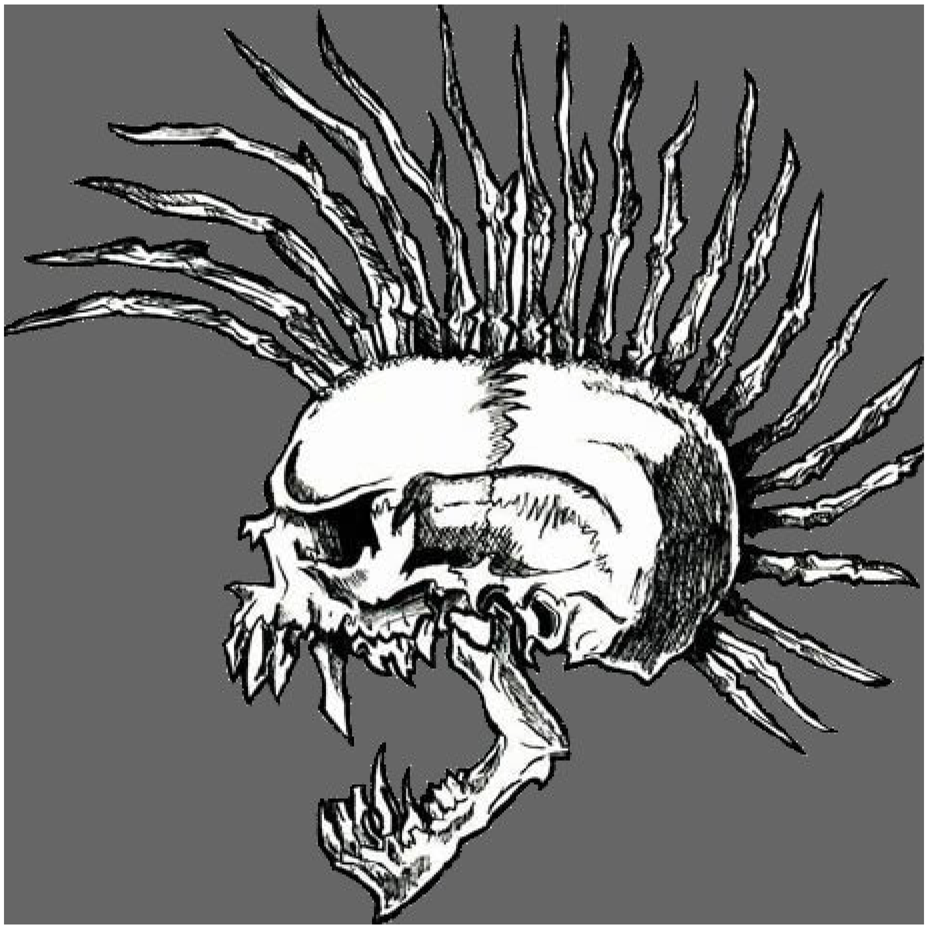
Virtual Machine Detection In The Browser

bannedit.github.io/Virtual-Machine-Detection-In-The-Browser.html

August 20, 2019



bannedit

Vulnerability Researcher

Introduction

Virtual Machine (VM) detection is nothing new. Malware has been doing it for over a decade now. Over time the techniques have advanced as defenders learned new ways of avoiding VM detection.

A while back a friend and I were working on a project related to exploit delivery via a web application for redteaming purposes. I wanted a way to fingerprint visitors of the site and hash the fingerprint data so I could look for potential repeat visitors. While investigating fingerprinting I stumbled upon something pretty interesting. I was looking at some code that collected information about WebGL capabilities. I quickly realized that some of the fingerprinting information could be useful for VM detection because vendor names were exposed. In this particular instance the string "VMWare" was contained within the WebGL information. After some more testing I also discovered that VirtualBox reported the same kind of information.

Once I realized it was potentially possible to detect VMs from the browser I started to dig deeper and began searching for other research related to this discovery. I found a pretty well researched academic paper [1] related to tracking users across multiple browsers. This gave me some other potential techniques that could be applied to VM detection.

The end goal of this research is to have multiple techniques for VM detection. Multiple techniques lead to much more accurate detection. Since some techniques are more false-positive prone than others, a weighting system can be applied to the detection capabilities. This allows us to generate detection confidence scoring. This can help account for inaccuracies of certain detection methods. Given enough testing and data it would then be possible to come up with a reasonable threshold value. If a browser scores above the threshold then it would most likely be within a VM. Alternatively, if the browser scored below the threshold value it could be considered to be running on physical hardware.

Techniques

Now that I have covered some of the background information and history leading up to this blog post we can start to dig into the actual techniques.

As mentioned prior in the introduction, WebGL can provide a lot of information about the OpenGL implementation including vendor information. The WEBGL_debug_renderer_info extension [2] can be used to query for debug information such as the WebGL vendor and rendered.

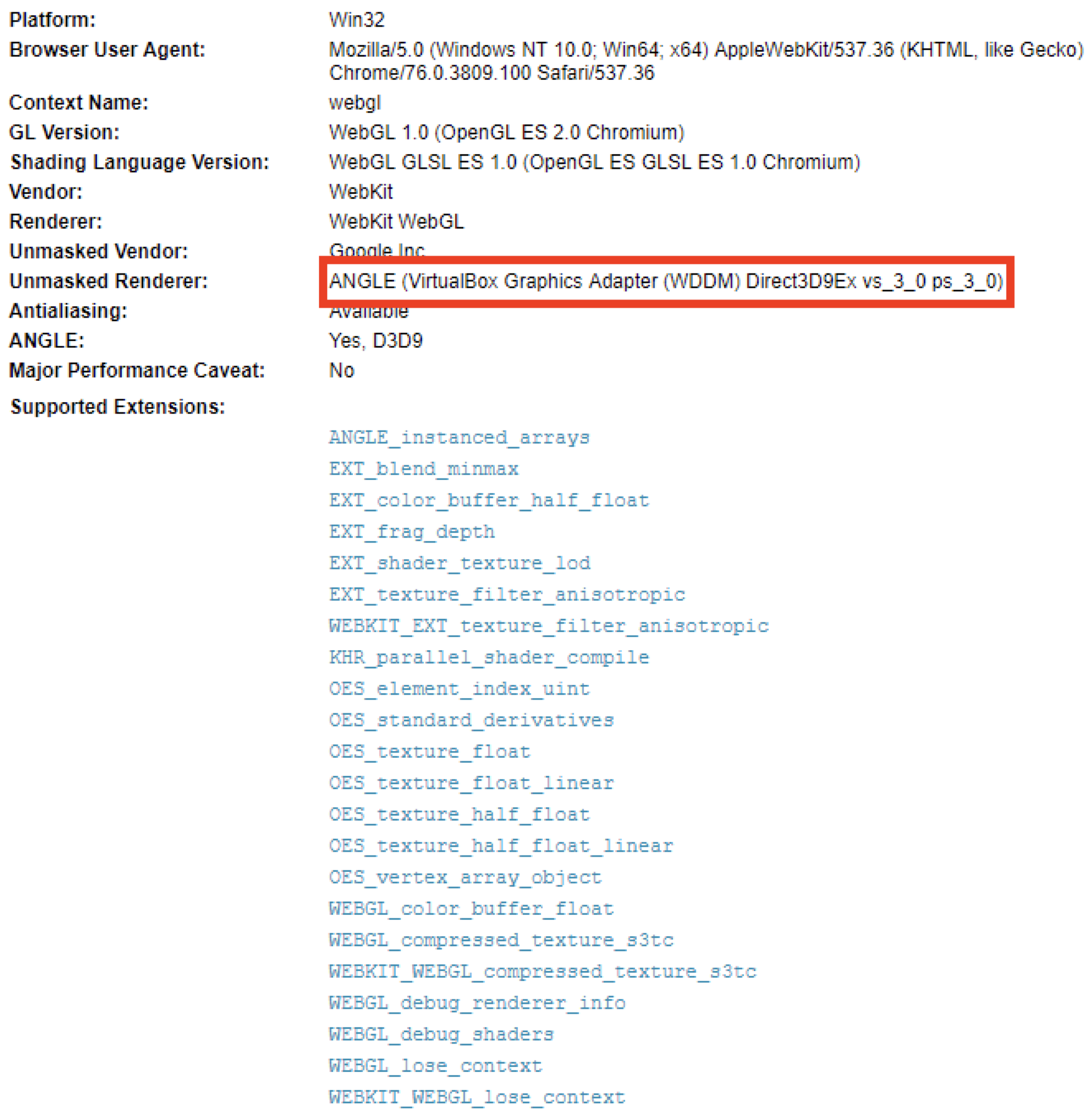
```
var canvas = document.createElement('canvas');
var gl = canvas.getContext('webgl');

var debugInfo = gl.getExtension('WEBGL_debug_renderer_info');
var vendor = gl.getParameter(debugInfo.UNMASKED_VENDOR_WEBGL);
var renderer = gl.getParameter(debugInfo.UNMASKED_RENDERER_WEBGL);

console.log(vendor);
console.log(renderer);
```

Additionally, extension availability can be queried using the ***getExtension*** method on a WebGL context. I have not fully explored this avenue but it might be possible to detect certain WebGL implementations provided by VMs based on the extensions available. Though this idea is likely very false-positive prone.

Below is a screenshot from [3] WebGLReport a website dedicated to fingerprinting WebGL.

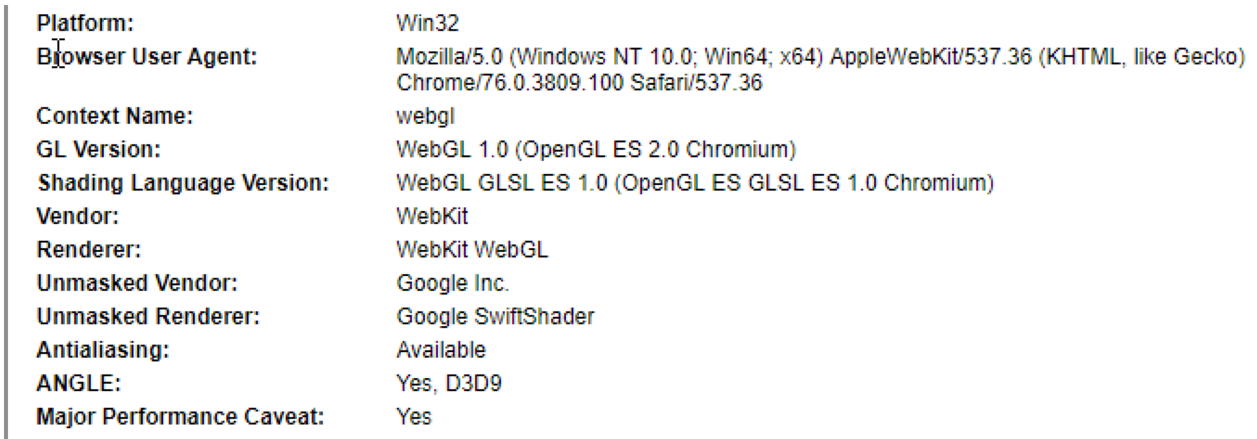


Platform:	Win32
Browser User Agent:	Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/76.0.3809.100 Safari/537.36
Context Name:	webgl
GL Version:	WebGL 1.0 (OpenGL ES 2.0 Chromium)
Shading Language Version:	WebGL GLSL ES 1.0 (OpenGL ES GLSL ES 1.0 Chromium)
Vendor:	WebKit
Renderer:	WebKit WebGL
Unmasked Vendor:	Google Inc.
Unmasked Renderer:	ANGLE (VirtualBox Graphics Adapter (WDDM) Direct3D9Ex vs_3_0 ps_3_0)
Antialiasing:	Available
ANGLE:	Yes, D3D9
Major Performance Caveat:	No
Supported Extensions:	

```
ANGLE_instanced_arrays
EXT_blend_minmax
EXT_color_buffer_half_float
EXT_frag_depth
EXT_shader_texture_lod
EXT_texture_filter_anisotropic
WEBKIT_EXT_texture_filter_anisotropic
KHR_parallel_shader_compile
OES_element_index_uint
OES_standard_derivatives
OES_texture_float
OES_texture_float_linear
OES_texture_half_float
OES_texture_half_float_linear
OES_vertex_array_object
WEBGL_color_buffer_float
WEBGL_compressed_texture_s3tc
WEBKIT_WEBGL_compressed_texture_s3tc
WEBGL_debug_renderer_info
WEBGL_debug_shaders
WEBGL_lose_context
WEBKIT_WEBGL_lose_context
```

VirtualBox Windows 10 x64 VM Google Chrome Visiting webglreport.com

Now, it is important to note that this depends on how the VM is configured. In Virtual Box for example, setting the graphics controller setting under Display to VMSVGA will report cause WebGL to use CPU based implementations of OpenGL which is browser dependent. However, this could still be a useful indicator that the client machine is running in a VM because most modern hardware has integrated GPUs and can provide access to OpenGL natively. Just keep in mind that CPU based OpenGL implementations do not necessarily mean it is a VM outright.



Platform:	Win32
Browser User Agent:	Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/76.0.3809.100 Safari/537.36
Context Name:	webgl
GL Version:	WebGL 1.0 (OpenGL ES 2.0 Chromium)
Shading Language Version:	WebGL GLSL ES 1.0 (OpenGL ES GLSL ES 1.0 Chromium)
Vendor:	WebKit
Renderer:	WebKit WebGL
Unmasked Vendor:	Google Inc.
Unmasked Renderer:	Google SwiftShader
Antialiasing:	Available
ANGLE:	Yes, D3D9
Major Performance Caveat:	Yes

VirtualBox Windows 10 x64 VM Google Chrome Using VMSVGA Visiting webglreport.com

This screenshot depicts Google Chrome utilizing the CPU based OpenGL implementation renderer Google SwiftShader [4].

Another technique seen in normal malware is to determine the screen width and height. This can be achieved in Javascript as well. Additionally, color depth and bits per pixel are other potentially good indicators related to the display.

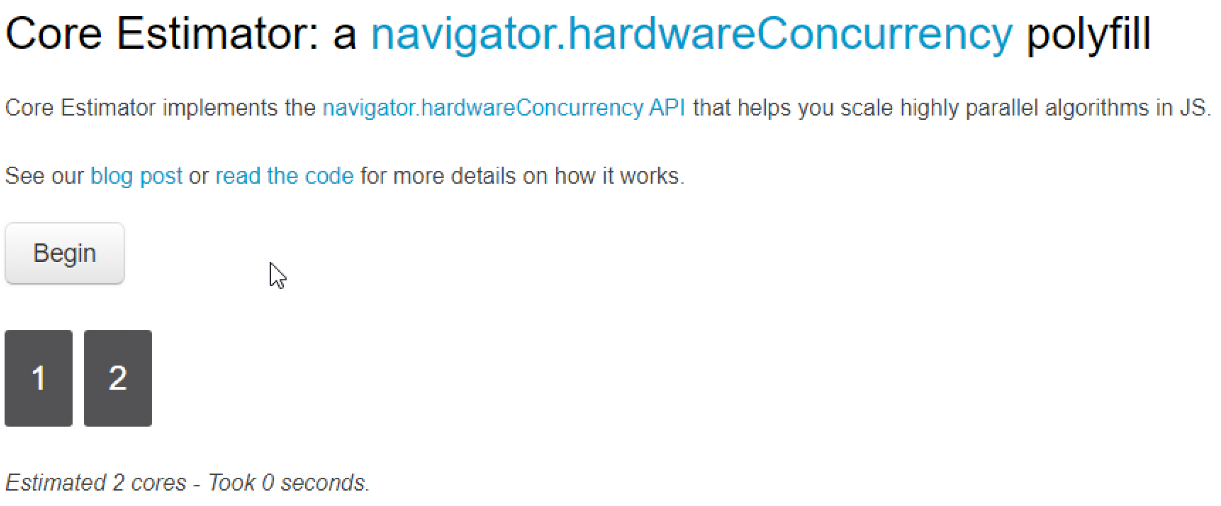
```
var width = screen.width;
var height = screen.height;
var color_depth = screen.colorDepth;
var bitspp = screen.pixelDepth;
```

More details on the screen object can be found at [5].

Can we detect the amount of RAM on the client? You bet. Again using Javascript we can determine roughly the amount of RAM available on the browser. One quirk to note here is that the browser will only report RAM values in gigabytes (gb). It also has a quirk where it will only report up to 8gb and as low as 256mb (0.25gb). These ranges of values however, are still enough to use as a VM detection method. Most physical workstations these days come with at least 8gb of RAM. Detecting smaller amounts of RAM such as 2gb or less would be a good indicator the client browser is in a VM. The specification for the Device Memory can be found at [6]

```
var ram = navigator.deviceMemory;
```

Finally, the last technique I will be covering detects the number of CPU cores. This is done by performing timing attacks using multiple web workers running simultaneously. During my testing of this technique I found it to be very accurate. I tested this concept out using the [7] Core Estimator Demo site. A small number of CPU cores can be a decent VM indicator and has been used by malware in the past. Core Estimator also provides the Javascript libraries on github [8].


Core Estimator: a navigator.hardwareConcurrency polyfill

Core Estimator implements the navigator.hardwareConcurrency API that helps you scale highly parallel algorithms in JS.

See our blog post or read the code for more details on how it works.

Estimated 2 cores - Took 0 seconds.

VirtualBox Windows 10 x64 VM Google Chrome 2 CPU Cores Tested with Core Estimator

Conclusion

This blog post covered four unique VM detection capabilities that can be performed from Javascript. When I first discovered these techniques my initial thought was to apply the concepts toward VM detection. Hopefully, both defenders and offensive security professions can find something useful to apply these techniques toward.

It is interesting to see that academics and various other researchers have applied some of the same concepts toward fingerprinting and privacy issues.

References

1. (Cross-)Browser Fingerprinting via OS and Hardware Level Features http://yinzhicao.org/TrackingFree/crossbrowsertracking_NDSS17.pdf
2. MDN web-docs WEBGL_debug_renderer_info https://developer.mozilla.org/en-US/docs/Web/API/WEBGL_debug_renderer_info
3. WebGL Report https://webglreport.com
4. Google Swiftshader Github https://github.com/google/swiftshader
5. W3 Device Memory Specification https://www.w3.org/TR/device-memory/
6. W3 Schools - The Screen Object https://www.w3schools.com/jsref/obj_screen.asp
7. Core Estimator Demo https://oswg.oftn.org/projects/core-estimator/demo/
8. Core Estimator Github https://github.com/oftn-oswg/core-estimator