(https://(https://(https://(text=WebAssembly%E2 webassembly- rising- rising- rising- ri role- role- role- r in- in- in- web- web- web- w development- development- development- and- and- and- a its- its- its- security- security- security- se concerns&via=GDATA)

Breaking New Ground: WebAssembly's Rising Role in Web Development and Its Security Concerns

11/19/2019

Reading time: 5 min (1456 words)

Techblog [(/blog/techblog)](/blog/techblog)

G DATA **Blog**

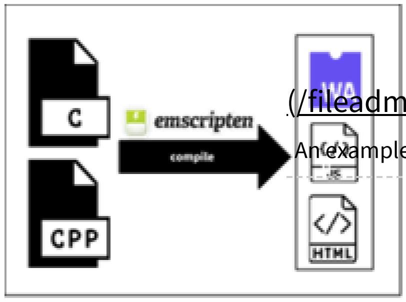
G DATA Security Lab
Virus-Analyst Team

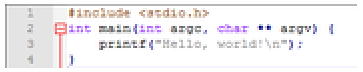
Getting to Know WebAssembly

Web applications development is continuously accelerating—from simple pages that use HTML/CSS/JavaScript, to the use of plugins like Flash and Java applets, to server–side tools that create programs over the Internet instead of installing them locally on users' desktops or mobile devices. The driving force for these developments is the need to provide more information and services that can be easily accessed by the end users.

One of the most popular and important components in developing these apps is JavaScript (JS), due to its functionality in creating dynamic/interactive web pages and cross-platform support across web browsers. However, websites and applications with complex JS code suffer from optimization problems and slow execution. Several technologies were developed to address these issues such as Portable Native Client (PNaCl) by Google and asm.js by Mozilla, which can convert low-level codes into programs that run very fast within the browser. Despite addressing the performance issues, these solutions still struggle to execute in a way that developers want their applications to perform. These efforts gave way to the creation of a new tool for web applications development called WebAssembly (Wasm).

How WASM works





```
#include <stdio.h>
int main(int argc, char ** argv) {
    printf("Hello, world!\n");
}
```

(/fileadmin/web/general/images/blog/2019/11_2(

An example program would be a simple “Hello World!” program c

(/fileadmin/web/general/images/blog/2019/11_2019/01_EmscriptenCompilationWorkflow.png)

The compilation workflow of Emscripten.

WebAssembly (https://webassembly.org/) is an open standard low-level assembly language and binary format that executes at near-native speeds, compiled from languages like C/C++. It is designed to help web apps coded with JS in executing more efficiently. Wasm has capabilities not limited to the following:

- Portable — able to run in different browsers and platforms
- Compact — files are already in bytecode and directly executed by the browser
- Fast execution — less time spent parsing and optimizing
- Support — can compile old programs coded in C/C++ that previously required dependencies

(https://
text=Web
webass
rising-
role-
in-
web-
develop
and-
its-
security
concerns

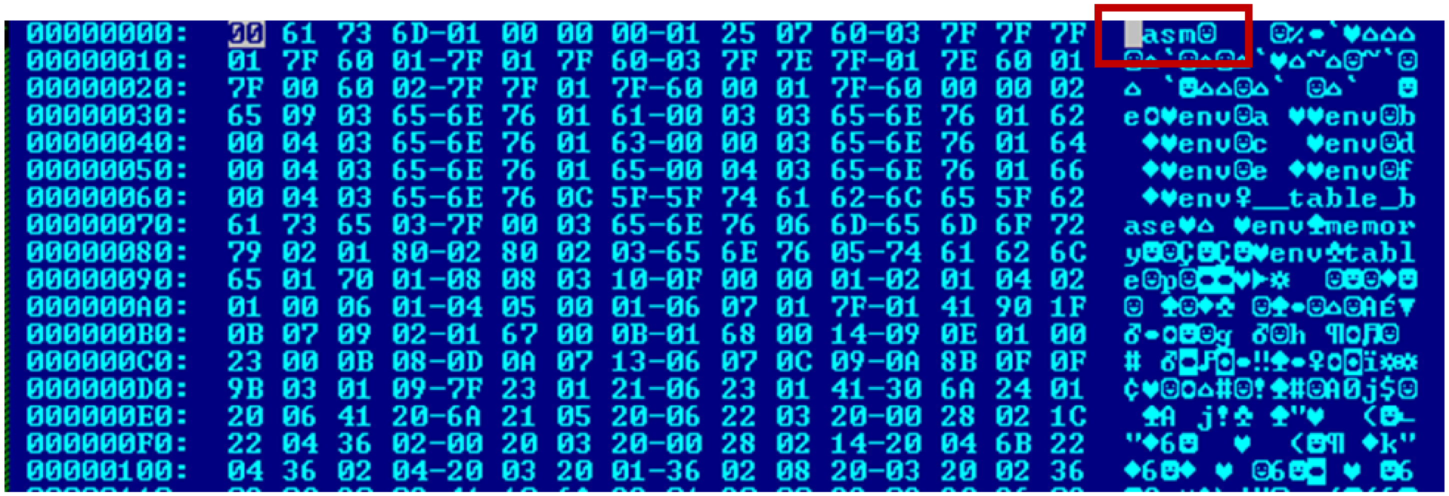
Reading time: 5 min (1456 words)

Techblog (/blog/techblog)

Several tools can be used to compile C/C++ code into Wasm such as Emscripten (https://emscripten.org/), a compiler toolchain that runs in operating systems like Linux and Windows.

Emscripten outputs three files: 1) the Wasm file that is the compiled C/C++ code; 2) the JavaScript file used to instantiate and call the Wasm file; 3) and the HTML page that loads the JS and Wasm files into a web browser. The Wasm, JS, and HTML files form the web application that is rendered in the browser.

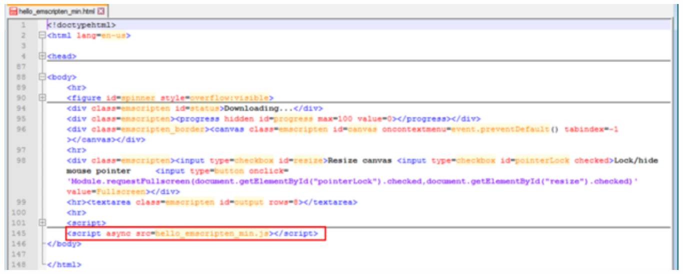
Converting the code using Emscripten would result in a Wasm bytecode as show in the image below. It should be noted that the first few hex bytes of the file indicating it as Wasm is **0x00 0x61 0x73 0x6D** (“ asm”).



```
00000000:  00 61 73 6D-01 00 00 00-01 25 07 60-03 7F 7F 7F   asm☺   ☺%•`♥⌂⌂⌂
00000010:  01 7F 60 01-7F 01 7F 60-03 7F 7E 7F-01 7E 60 01  ☺⌂`☺⌂☺⌂`♥⌂~⌂☺~`☺
00000020:  7F 00 60 02-7F 7F 01 7F-60 00 01 7F-60 00 00 02  ⌂ `☻⌂⌂☺⌂` ☺⌂`  ☻
00000030:  65 09 03 65-6E 76 01 61-00 03 03 65-6E 76 01 62  e○♥env☺a ♥♥env☺b
00000040:  00 04 03 65-6E 76 01 63-00 00 03 65-6E 76 01 64   ♦♥env☺c  ♥env☺d
00000050:  00 04 03 65-6E 76 01 65-00 04 03 65-6E 76 01 66   ♦♥env☺e ♦♥env☺f
00000060:  00 04 03 65-6E 76 0C 5F-5F 74 61 62-6C 65 5F 62   ♦♥env♀__table_b
00000070:  61 73 65 03-7F 00 03 65-6E 76 06 6D-65 6D 6F 72  ase♥⌂ ♥env♠memor
00000080:  79 02 01 80-02 80 02 03-65 6E 76 05-74 61 62 6C  y☻☺Ç☻Ç☻♥env♣tabl
00000090:  65 01 70 01-08 08 03 10-0F 00 00 01-02 01 04 02  e☺p☺◘◘♥►☼  ☺☻☺♦☻
000000A0:  01 00 06 01-04 05 00 01-06 07 01 7F-01 41 90 1F  ☺ ♠☺♦♣ ☺♠•☺⌂☺AÉ▼
000000B0:  0B 07 09 02-01 67 00 0B-01 68 00 14-09 0E 01 00  ♂•○☻☺g ♂☺h ¶○♫☺
000000C0:  23 00 0B 08-0D 0A 07 13-06 07 0C 09-0A 8B 0F 0F  # ♂◘♪◙•‼♠•♀○◙ï☼☼
000000D0:  9B 03 01 09-7F 23 01 21-06 23 01 41-30 6A 24 01  ¢♥☺○⌂#☺!♠#☺A0j$☺
000000E0:  20 06 41 20-6A 21 05 20-06 22 03 20-00 28 02 1C   ♠A j!♣ ♠"♥  (☻∟
000000F0:  22 04 36 02-00 20 03 20-00 28 02 14-20 04 6B 22  "♦6☻  ♥  (☻¶ ♦k"
00000100:  04 36 02 04-20 03 20 01-36 02 08 20-03 20 02 36  ♦6☻♦ ♥ ☺6☻◘ ♥ ☻6
```

(/fileadmin/web/general/images/blog/2019/11_2019/03_HelloWorld_wasm.png)

The first four bytes indicate that this is was code.



hello_emscripten_min.html

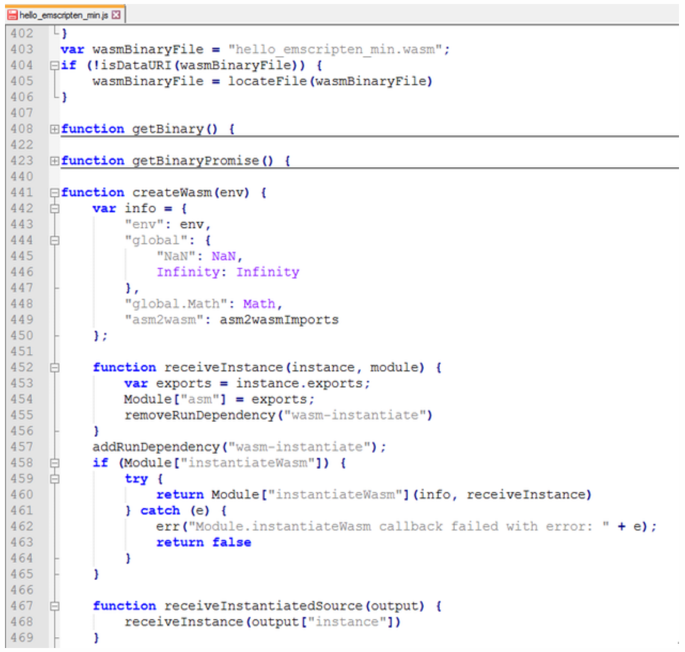
```
<!doctypehtml>
<html lang=en-us>

<head>

<body>
    <hr>
    <figure id=spinner style=overflow:visible>
    <div class=emscripten id=status>Downloading...</div>
    <div class=emscripten><progress hidden id=progress max=100 value=0></progress></div>
    <div class=emscripten_border><canvas class=emscripten id=canvas oncontextmenu=event.preventDefault() tabindex=-1
    ></canvas></div>
    <hr>
    <div class=emscripten><input type=checkbox id=resize>Resize canvas <input type=checkbox id=pointerLock checked>Lock/hide
    mouse pointer     <input type=button onclick=
    'Module.requestFullscreen(document.getElementById("pointerLock").checked,document.getElementById("resize").checked)'
    value=Fullscreen></div>
    <hr><textarea class=emscripten id=output rows=8></textarea>
    <hr>
    <script>
    <script async src=hello_emscripten_min.js></script>
</body>

</html>
```

(/fileadmin/web/general/images/blog/2019/11_2019/(

(https://(https://(https://(h
text=WebAssembly%20
webassembly webassembly webassembly
rising- rising- rising- ri
role- role- role-
in- in- in-
web- web- web-
development- development- development-
and- and- and-
its- its- its-
security security security se
concerns&via=CDATA)

Reading time: 5 min (1456 words)

Techblog (/blog/techblog)


hello_emscripten_min.js

```
}
var wasmBinaryFile = "hello_emscripten_min.wasm";
if (!isDataURI(wasmBinaryFile)) {
    wasmBinaryFile = locateFile(wasmBinaryFile)
}

function getBinary() {

function getBinaryPromise() {

function createWasm(env) {
    var info = {
        "env": env,
        "global": {
            "NaN": NaN,
            Infinity: Infinity
        },
        "global.Math": Math,
        "asm2wasm": asm2wasmImports
    };

    function receiveInstance(instance, module) {
        var exports = instance.exports;
        Module["asm"] = exports;
        removeRunDependency("wasm-instantiate")
    }
    addRunDependency("wasm-instantiate");
    if (Module["instantiateWasm"]) {
        try {
            return Module["instantiateWasm"](info, receiveInstance)
        } catch (e) {
            err("Module.instantiateWasm callback failed with error: " + e);
            return false
        }
    }

    function receiveInstantiatedSource(output) {
        receiveInstance(output["instance"])
    }
```

(/fileadmin/web/general/images/blog/2019/11_2019/04_JS_Instantiate_wasm.png)

The Wasm file will be instantiated by the JS file, which has been automatically generated by Emscripten.

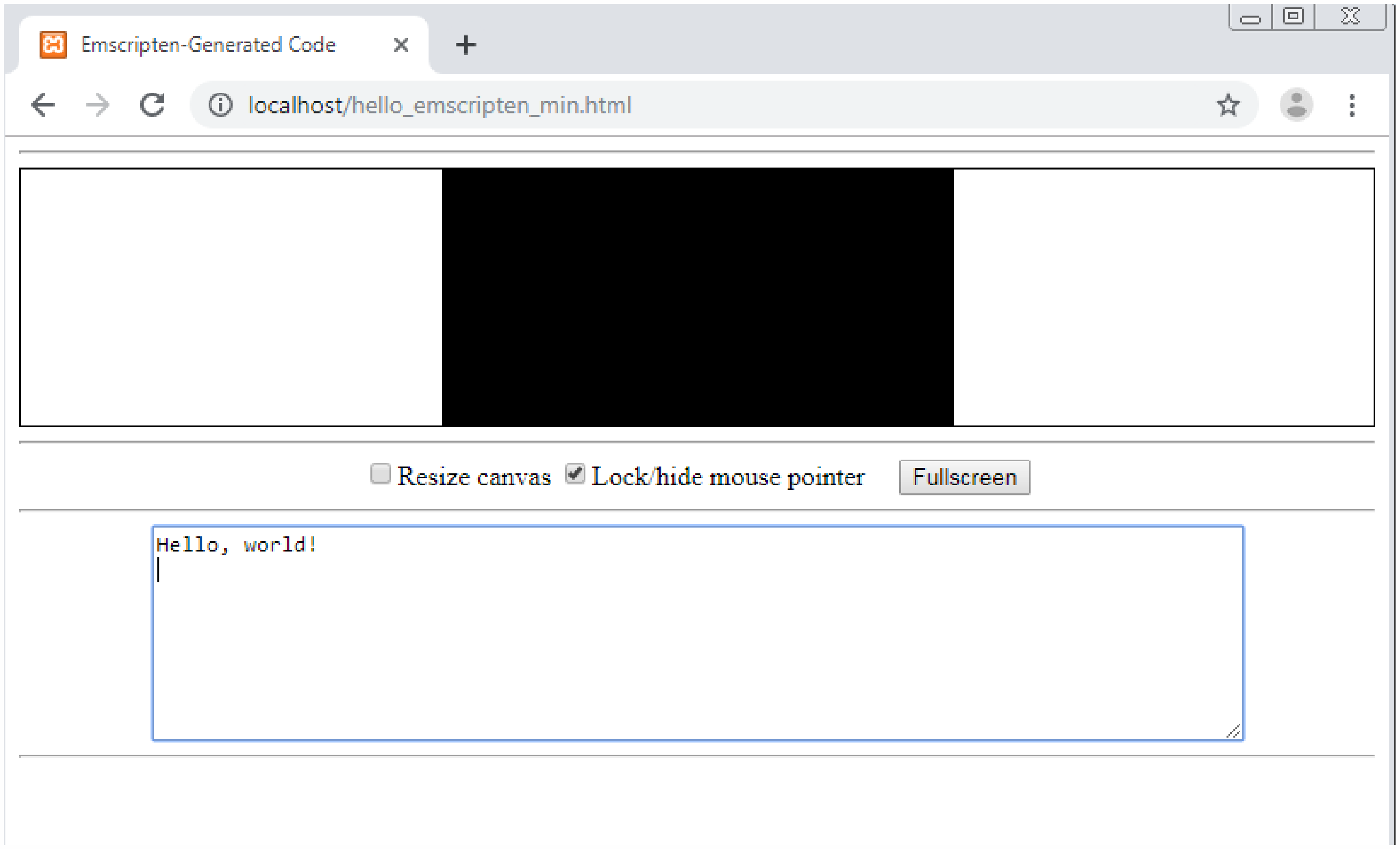
The compiled application is then loaded in the web browser via the HTML file.

(https:// text=We webass rising- role- in- web- develop and- its- security concern

Reading time: 5 min (1456 words)

Techblog (/blog/techblog)

The image below is the loaded web page of the string "Hello, world!" initially written in C and converted to Wasm. Emscripten also automatically generates a basic HTML layout that correctly outputs the printf command of C upon loading in the browser. Even though Wasm is relatively new, web developers are keen to try out this tool for their applications. Various companies and entities are already adapting this new technology for the web.


Emscripten-Generated Code
localhost/hello_emscripten_min.html
Resize canvas
Lock/hide mouse pointer
Fullscreen

```
Hello, world!
```

(/fileadmin/web/general/images/blog/2019/11_2019/06_HelloWorldBrowserOutput.png)

(https://(https://(https://(h
text=WebAssembly%E2
webassembly-webassembly-webassembly-w
rising- rising- rising- ri
role- role- role- r
in- in- in-
web- web- web- w
development-development-development-de
and- and- and- a
its- its- its-
security-security-security-se
concerns&via=GDATA)cs

Reading time: 5 min (1456 words)

Techblog (/blog/techblog)

Industry Adaption and Acceptance

All modern browsers support (https://developer.mozilla.org/en-US/docs/WebAssembly#Browser_compatibility)WebAssembly, although some browsers such as MS Edge Mobile and Extended Support Release of Firefox disable support for Wasm by default. Momentum is slowly gaining for industry acceptance and support as developers are implementing Wasm in their web applications.

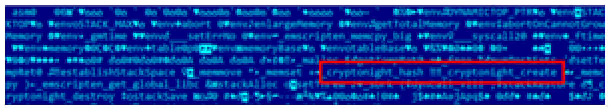
Video games are known for being high-load applications that have heavy CPU and graphics memory usage. Early demonstrations of Wasm therefore focused on video games since Wasm is designed for efficiency and a minimal memory footprint. Two examples are the D3Wasm project and Unity. D3Wasm (http://www.continuation-labs.com/projects/d3Wasm/) is a port of the id Tech 4 engine into Wasm, which can render the game “Doom 3”. Users are now able to play the game in any web browser that supports WebGL, showing that it is possible to load large programs with Wasm. In August 2018, the video game engine Unity (https://blogs.unity3d.com/2018/08/15/webassembly-is-here/) switched their default target web format from asm.js to Wasm. They noted that Wasm is more efficient compared with asm.js in terms of smaller code size, memory efficiency during load time, and faster performance.

WebAssembly’s capabilities in image processing was recently demonstrated on Ebay’s online version of their bar code scanner (https://www.ebayinc.com/stories/blogs/tech/webassembly-at-ebay-a-real-world-use-case/). Users assumed that the old version of the application did not work properly because the processing time of the scanner was too slow. The new online scanner utilized both Wasm and JS and was found to be 50 times more efficient than the JS-only version of the application.
AutoCAD (https://blogs.autodesk.com/autocad/autocad-web-app-google-io-2018/) also created a version of their desktop software for the web that targets portability. Their initial development was built on the now-deprecated Adobe Flash, later switching to HTML5/JS. They were able to port their C++ code base into Wasm, some of which are 30 years old. The result is an application that can be accessed by the registered users over the web.
Lastly, Google Earth (https://blog.chromium.org/2019/06/webassembly-brings-google-earth-to-more.html) is the most recent and significant web application that uses Wasm. Like the above examples, the code base of Google Earth is in C++ and was ported using Emscripten. While the previous version of Google Earth was compiled in Native Client (NaCl) that only runs in Chrome, the newer version can be accessed by all major browsers that support Wasm. Currently the development of Wasm is a work in progress to adapt to newer Wasm-related technologies.

With more companies integrating WebAssembly in their applications due to its versatility and advantages which can complement and work alongside JavaScript, the future for this technology is optimistic. But much like JS and other web application tools, Wasm will also gain attention from cybercriminals and malware authors. Therefore, we also need to focus on Wasm’s security and its possible use in malware by cybercriminals.

Security Concerns

WebAssembly’s security architecture was designed in response to the



(/fileadmin/web/general/images/blog/2019/11_2019/07_Cryptonight_wasm.png)

(https://
text=WebAssembly
webassembly-
rising-
role-
in-
web-
development-
and-
its-
security-
concerns

Reading time: 5 min (1456 words)

Techblog (/blog/techblog)

Bytecode of the Cryptonight coinminer that was compiled in Wasm. Highlighted are the module names "_cryptonight_hash" and "_cryptonight_create".

shortcomings of previous technologies such as JavaScript and Flash. It runs in an isolated/sandboxed environment within the browser. When an application written in WebAssembly is executed, it cannot directly access or modify the functions or variables that are not yet called. Any calls needed by the application must be verified by the browser's engine/environment before it gives access to the called instruction or memory allocation.
However, Wasm can still be used for malicious activities. An example would be a file collected by G DATA that was found to be compiled into Wasm. Based on the modules from its header the file was used for Cryptonight mining, an algorithm for cryptocurrency mining such as Monero and Electroneum.

Although Cryptonight is not malicious by itself, malware authors are compiling it into Wasm instead of JS and injecting them in websites to illicitly mine cryptocurrencies from users' browsers by hijacking the CPU to use its processing power. This makes sense as JS is commonly used in crytpomining, the malware authors in this situation optimize their attack by leveraging Wasm to increase their mining efficiency as cryptomining requires large computing power (https://www.gdatasoftware.com/blog/2019/03/31575-eco-g-data-develop-cryptomining-rule).

According to research presented at Blackhat USA 2018 (https://i.blackhat.com/us-18/Thu-August-9/us-18-Lukasiewicz-WebAssembly-A-New-World-of-Native_Exploits-On-The-Web-wp.pdf), possible vulnerabilities that can use WebAssembly include cross-site scripting (https://i.blackhat.com/us-18/Thu-August-9/us-18-Lukasiewicz-WebAssembly-A-New-World-of-Native_Exploits-On-The-Web-wp.pdf) (XSS), where the function definitions in Emscripten calls JS to execute malicious code. If such vulnerabilities would exist in Wasm, then drive-by downloads that can be executed outside the browser environment are possible leading to malicious behavior like invoking PowerShell or process injection although no recent proof-of-concept are available yet to leverage these exploits in WebAssembly.

Is WebAssembly Future Friendly?

With its many possibilities, WebAssembly could be either a friend or a foe. It adds overwhelming advantages to applications development but can provide new avenues for vulnerabilities that may lead to exploits and malware attacks. However, adaption of Wasm for malicious activity is not yet imminent as development is still ongoing with continuous improvements in its security standards and general implementation in web browsers. The future of Wasm is anticipated, but we must prepare for any emerging threats that may surface.

(https://[illegible] text=WebAssembly[illegible] webassembly-rising-role-in-web-development-and-its-security-concerns[illegible]

Reading time: 5 min (1456 words)

Techblog (/blog/techblog)

While developers are responsible for the security of Wasm as compiled modules are the most probable targets, end-users must also constantly update their browsers and plugins. Users can download browser extensions such as NoScript (https://noscript.net/), a software that blocks dynamic web content, to stop any Wasm files from loading as it currently uses Javascript as the glue code to execute on a web page. The maximum privacy and security settings for browsers can be enabled for a safer web surfing. It is also best to be on the lookout for suspicious activities on the web.

Sample IOC

SHA256: 3f5961a80d3aa7cb06520fd8e89558170936a1a4a3fe16e9fc84c379518c0759
Malware type: Cryptonight Wasm
Detection: Application.BitCoinMiner.ZV

References

1. https://webassembly.org/ (/blog/2019/11/35494-webassemblys-rising-role-in-web-development-and-its-security-concerns)
2. developer.mozilla.org/en-US/docs/WebAssembly (https://developer.mozilla.org/en-US/docs/WebAssembly#Browser_compatibility)
3. techcrunch.com/2015/06/17/google-microsoft-mozilla-and-others-team-up-to-launch-webassembly-a-new-binary-format-for-the-web/ (https://techcrunch.com/2015/06/17/google-microsoft-mozilla-and-others-team-up-to-launch-webassembly-a-new-binary-format-for-the-web/)
4. i.blackhat.com/us-18/Thu-August-9/us-18-Lukasiewicz-WebAssembly-A-New-World-of-Native_Exploits-On-The-Web-wp.pdf (https://i.blackhat.com/us-18/Thu-August-9/us-18-Lukasiewicz-WebAssembly-A-New-World-of-Native_Exploits-On-The-Web-wp.pdf)
5. emscripten.org (https://emscripten.org/)
6. blogs.unity3d.com/2018/08/15/webassembly-is-here/ (https://blogs.unity3d.com/2018/08/15/webassembly-is-here/)
7. www.continuation-labs.com/projects/d3Wasm/ (http://www.continuation-labs.com/projects/d3Wasm/)

Reading time: 5 min (1456 words)

8. blog.chromium.org/2019/06/webassembly-brings-google-earth-to-more.html (https://blog.chromium.org/2019/06/webassembly-brings-google-earth-to-more.html)
9. www.ebayinc.com/stories/blogs/tech/webassembly-at-ebay-a-real-world-use-case/ (https://www.ebayinc.com/stories/blogs/tech/webassembly-at-ebay-a-real-world-use-case/)
10. blogs.autodesk.com/autocad/autocad-web-app-google-io-2018/ (https://blogs.autodesk.com/autocad/autocad-web-app-google-io-2018/)
11. www.gdatasoftware.com/blog/2019/03/31575-eco-g-data-develop-cryptomining-rule (https://www.gdatasoftware.com/blog/2019/03/31575-eco-g-data-develop-cryptomining-rule)

back to list (/blog)

Reading time: 5 min (1456 words)

(https://www.facebook.com/sharer/sharer.php?
webassemblys-
rising-
role-
in-
web-
development-
and-
its-
security-
concerns)

Techblog (/blog/techblog)

YOU ARE HERE: BLOG (EN) (/BLOG) / WEBASSEMBLY'S RISING ROLE IN WEB DEVELOPMENT AND ITS SECURITY CONCERNS

(https://(https://(https://(h
text=WebAssembly%E2
webassembly-webassembly-webassembly-w
rising- rising- rising- ri
role- role- role- r
in- in- in-
web- web- web- w
development-development-development-de
and- and- and- a
its- its- its-
security-security-security-se
concerns&via=GDATA)concerns)concerns)c

Reading time: 5 min (1456 words)

Techblog [(/blog/techblog)](/blog/techblog)