NEWS

X41 D-Sec GmbH Security Advisory: X41-2021-001 - Multiple Vulnerabilities in YARA

Highest Severity Rating: Medium

Confirmed Affected Versions: YARA v4.0.3 and earlier

Confirmed Patched Versions: YARA v4.0.4

Vendor: VirusTotal (Google Inc.)

Vendor URL: https://virustotal.github.io/yara

Credit: X41 D-Sec GmbH, Luis Merino

Status: Public

Advisory-URL: https://www.x41-dsec.de/lab/advisories/x41-2021-001-yara

Summary and Impact
An integer overflow and several buffer overflow reads in libyara/modules/macho/macho.c in YARA v4.0.3 and earlier could allow an attacker to either cause denial of service or information disclosure via a malicious Mach-O file.

Product Description
According to the official project description:

YARA is a tool aimed at (but not limited to) helping malware researchers to identify and
classify malware samples. With YARA you can create descriptions of malware families
(or whatever you want to describe) based on textual or binary patterns. Each description,
a.k.a. rule, consists of a set of strings and a boolean expression which determine its logic.

Integer overflow in macho_parse_fat_file()

Severity Rating: Medium

Vector: Mach-O file sample

CVE: Pending

CWE: 190

Analysis
An integer overflow in `macho_parse_fat_file()` while processing the fat Mach-O file header can lead to arbitrary read.

```
    if (size < arch.offset + arch.size)
      continue;

    /* Force 'file' array entry creation. */
    set_integer(YR_UNDEFINED, object, "file[%i].magic", i);

    /* Get specific Mach-O file data. */
    macho_parse_file(
        data + arch.offset,
        arch.size,
        get_object(object, "file[%i]", i),
        context);
```

When the `arch.offset` + `arch.size` result does not fit in the uint64_t type the result will wraparound and might allow to bypass the `size` < `arch.offset` + `arch.size` sanity check. Afterwards, `macho_parse_file()` will be called with `buffer` and `size` values that could be invalid, resulting in an arbitrary read and plausible infoleak or denial of service.

`It is recommended to bail out when the addition during the sanity checks wraps around.`

```
-    if (size < arch.offset + arch.size)
+    if (size < arch.offset + arch.size ||
+        arch.offset + arch.size < arch.offset) // check int overflow
       continue;
```

Proof of Concept
Parse the https://github.com/x41sec/advisories/tree/master/X41-2021-001/yara-reproducers/int-overflow-macho-parse-fat-file sample via yr_rules_scan_mem().

Out-of-bounds reads in macho_parse_file() and others

Severity Rating: Medium

Vector: Mach-O file sample

CVE: Pending

CWE: 125

Analysis
Two for-loops iterate over input buffer `data` to extract and process segments and other commands. Incorrect sanity checks lead to out of bounds reads in several places.

It is also recommended to perform a sanity check on `command_struct.cmdsize`, discarding those values where `command_struct.cmdsize`< `sizeof(yr_load_command_t)`.

```
   for (unsigned i = 0; i < header.ncmds; i++)
   {
-    if (command - data < sizeof(yr_load_command_t))
+    if (data + size < command + sizeof(yr_load_command_t))
       break;

     memcpy(&command_struct, command, sizeof(yr_load_command_t));

     if (should_swap)
       swap_load_command(&command_struct);

-    if (size < header_size + command_struct.cmdsize)
+    if (size - (command - data) < command_struct.cmdsize || command_struct.cmdsize < sizeof(yr_load_command_t))
       break;
```

Please note this needs to be patched in the two similar for-loops.

Proof of Concept
Parse the https://github.com/x41sec/advisories/tree/master/X41-2021-001/yara-reproducers/oob-macho-parse-file sample via yr_rules_scan_mem().

Out-of-bounds reads in macho_handle_segment_64() and others

Severity Rating: Medium

Vector: Mach-O file sample

CVE: Pending

CWE: 125

Analysis
`macho_handle_segment_64()` reads `sizeof(yr_segment_command_64_t)` bytes from `command` without checking if the buffer is big enough.

```
  memcpy(&sg, command, sizeof(yr_segment_command_64_t));
```

This results in an out of bounds read when `command` is not big enough. Infoleak or denial of service could be a plausible outcome.

It is recommended to check at least `sizeof(yr_segment_command_64_t)` bytes are available in `command before calling` macho_handle_segment_64()`.

The same issue occurs when calling `macho_handle_segment()`, `macho_handle_unixthread()` and `macho_handle_main()`.

```
{
    case LC_SEGMENT:
+      if(command_struct.cmdsize < sizeof(yr_segment_command_32_t))
+        break;
       macho_handle_segment(command, seg_count++, object);
       break;
     case LC_SEGMENT_64:
+      if(command_struct.cmdsize < sizeof(yr_segment_command_64_t))
+        break;
       macho_handle_segment_64(command, seg_count++, object);
       break;
}

{
    case LC_UNIXTHREAD:
+      if(command_struct.cmdsize < sizeof(yr_thread_command_t))
+          break;
      macho_handle_unixthread(command, object, context);
      break;
    case LC_MAIN:
+      if(command_struct.cmdsize < sizeof(yr_entry_point_command_t))
+          break;
      macho_handle_main(command, object, context);
      break;
    }

}
```

Please note that we rely here on `cmdsize` having a safe value, which is checked in the fix proposed for the previous finding.

Proof of Concept
Parse the https://github.com/x41sec/advisories/tree/master/X41-2021-001/yara-reproducers/oob-macho-handle-segment, [https://github.com/x41sec/advisories/tree/master/X41-2021-001/yara-reproducers/oob-macho-handle-segment-64 samples via yr_rules_scan_mem().

Several out-of-bounds reads in macho_handle_unixthread()

Severity Rating: Medium

Vector: Mach-O file sample

CVE: Pending

CWE: 125

```
  command = (void*) ((uint8_t*) command + sizeof(yr_thread_command_t));
```

which could result in `command` pointing out of bounds and triggering invalid reads in subsequent `command` dereferences. It is recommended to check at least `sizeof(yr_thread_command_t)` bytes are available before calling `macho_handle_unixthread()`.

```
    case LC_UNIXTHREAD:
+     if(size - (command - data) < sizeof(yr_thread_command_t))
+       break;
      macho_handle_unixthread(command, object, context);
      break;
```

Secondly, when reading the entry point for the different archs, it is assumed the buffer is big enough to read the corresponding `yr_*_thread_state_t` object. A not big enough buffer would lead to out of bounds reads and maybe denial of service or infoleaks. It is recommended to check at least enough bytes are available.

```
  int should_swap = should_swap_bytes(get_integer(object, "magic"));
  bool is64 = false;
  uint32_t s = ((yr_thread_command_t*)command)->cmdsize - sizeof(yr_thread_command_t);
  command = (void*) ((uint8_t*) command + sizeof(yr_thread_command_t));
  uint64_t address = 0;

  switch (get_integer(object, "cputype"))
  {
   case CPU_TYPE_MC680X0:
   {
+    if (s < sizeof(yr_m68k_thread_state_t))
+      break;
     yr_m68k_thread_state_t* m68k_state = (yr_m68k_thread_state_t*) command;
     address = m68k_state->pc;
     break;
   }
   case CPU_TYPE_MC88000:
   {
+    if (s < sizeof(yr_m88k_thread_state_t))
+      break;
     yr_m88k_thread_state_t* m88k_state = (yr_m88k_thread_state_t*) command;
     address = m88k_state->xip;
     break;
   }
   case CPU_TYPE_SPARC:
   {
+    if (s < sizeof(yr_sparc_thread_state_t))
+      break;
     yr_sparc_thread_state_t* sparc_state = (yr_sparc_thread_state_t*) command;
     address = sparc_state->pc;
     break;
   }
   case CPU_TYPE_POWERPC:
   {
+    if (s < sizeof(yr_ppc_thread_state_t))
+      break;
     yr_ppc_thread_state_t* ppc_state = (yr_ppc_thread_state_t*) command;
     address = ppc_state->srr0;
     break;
   }
   case CPU_TYPE_X86:
   {
+    if (s < sizeof(yr_x86_thread_state_t))
+      break;
     yr_x86_thread_state_t* x86_state = (yr_x86_thread_state_t*) command;
     address = x86_state->eip;
     break;
   }
   case CPU_TYPE_ARM:
   {
+    if (s < sizeof(yr_arm_thread_state_t))
+      break;
     yr_arm_thread_state_t* arm_state = (yr_arm_thread_state_t*) command;
     address = arm_state->pc;
     break;
   }
   case CPU_TYPE_X86_64:
   {
+    if (s < sizeof(yr_x86_thread_state64_t))
+      break;
     yr_x86_thread_state64_t* x64_state = (yr_x86_thread_state64_t*) command;
     address = x64_state->rip;
     is64 = true;
   }
   case CPU_TYPE_ARM64:
   {
+    if (s < sizeof(yr_arm_thread_state64_t))
+      break;
     yr_arm_thread_state64_t* arm64_state = (yr_arm_thread_state64_t*) command;
     address = arm64_state->pc;
     is64 = true;
   }
   case CPU_TYPE_POWERPC64:
   {
+    if (s < sizeof(yr_ppc_thread_state64_t))
+      break;
     yr_ppc_thread_state64_t* ppc64_state = (yr_ppc_thread_state64_t*) command;
     address = ppc64_state->srr0;
     is64 = true;
   }
```

Please note that we rely here on `cmdsize` having a safe value, which is checked in one of the fixes proposed above.

Proof of Concept

Parse the https://github.com/x41sec/advisories/tree/master/X41-2021-001/yara-reproducers/oob-macho-handle-unixthread sample via yr_rules_scan_mem().

Out-of-bounds read in macho_is_32()

Severity Rating: Medium

Vector: Mach-O file sample

CVE: Pending

CWE: 125

Analysis

`macho_parse_file()` calls `macho_is_32(data)` without checking that `size` is at least 4 bytes. An out-of-bounds read happens when `size` < 4. Depending on the initial parsed file size, `data` buffer could come from a mmaped region and the OOB read could lead to a denial of service.

It is recommended to return from the function when `size` < 4.

```
+  if (size < 4)
+    return;
+
   size_t header_size = macho_is_32(data) ? sizeof(yr_mach_header_32_t)
                                          : sizeof(yr_mach_header_64_t);
```

Proof of Concept

Parse the https://github.com/x41sec/advisories/tree/master/X41-2021-001/yara-reproducers/oob-macho-is-32-v2 sample via yr_rules_scan_mem().

Timeline

2021-01-16 Issues found

2021-01-20 Issues and patches reported to the vendor

2021-01-21 Vendor reply with acknowledge and final patches

2021-01-22 CVEs request (pending)

2021-01-27 Fixed release (v4.0.4)

2021-01-28 Advisory published

About X41 D-SEC GmbH

X41 is an expert provider for application security services. Having extensive industry experience and expertise in the area of information security, a strong core security team of world class security experts enables X41 to perform premium security services.

Fields of expertise in the area of application security are security centered code reviews, binary reverse engineering and vulnerability discovery. Custom research and IT security consulting and support services are core competencies of X41.

Author: Luis Merino
Date: January 28, 2021

«
Microsoft Exchange Remote Code Execution - CVE-2020-16875
»
New Ordering System (April Fools')

CONTACT

X41 D-SEC GmbH
Krefelder Str. 123
52070 Aachen

+49 (0) 241 9809418-0
+49 (0) 241 9809418-9
info@x41-dsec.de

PGP Key