Windows 0-day exploit CVE-2019-1458 used in Operation WizardOpium

securelist.com/windows-0-day-exploit-cve-2019-1458-used-in-operation-wizardopium/95432

Research

Research

10 Dec 2019

minute read

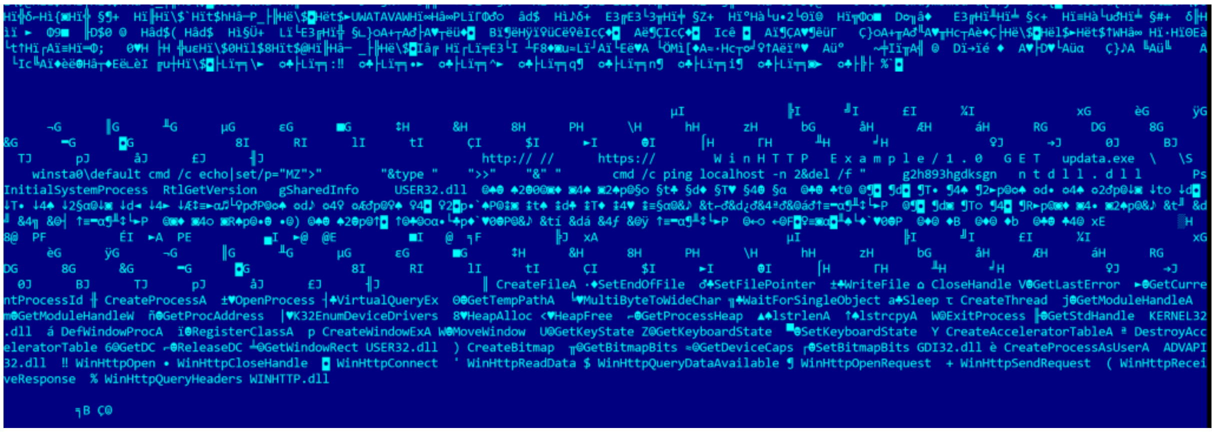
Authors

- Expert AMR
- Expert GReAT

In November 2019, Kaspersky technologies successfully detected a Google Chrome 0-day exploit that was used in Operation WizardOpium attacks. During our investigation, we discovered that yet another 0-day exploit was used in those attacks. The exploit for Google Chrome embeds a 0-day EoP exploit (CVE-2019-1458) that is used to gain higher privileges on the infected machine as well as escaping the Chrome process sandbox.

The EoP exploit consists of two stages: a tiny PE loader and the actual exploit. After achieving a read/write primitive in the renderer process of the browser through vulnerable JS code, the PE exploit corrupts some pointers in memory to redirect code execution to the PE loader. This is done to bypass sandbox restrictions because the PE exploit cannot simply start a new process using native WinAPI functions.

The PE loader locates an embedded DLL file with the actual exploit and repeats the same process as the native Windows PE loader – parsing PE headers, handling imports/exports, etc. After that, a code execution is redirected to the entry point of the DLL – the DllEntryPoint function. The PE code then creates a new thread, which is an entry point for the exploit itself, and the main thread simply waits until it stops.



EoP exploit used in the attack

The PE file encapsulating this EoP exploit has the following header:

Count of sections	5	Machine	AMD64
Symbol table	00000000[00000000]		Wed Jul 10 03:50:48 2019
Size of optional header	00F0	Magic optional header	020B
Linker version	12.00	OS version	6.00
Image version	0.00	Subsystem version	6.00
Entry point	0000135C	Size of code	00002A00
Size of init data	00002200	Size of uninit data	00000000
Size of image	00009000	Size of header	00000400
Base of code	00001000		
Image base	00000001`80000000	Subsystem	GUI
Section alignment	00001000	File alignment	00000200
Stack	00000000`00100000	Heap	00000000`00100000
Stack commit	00000000`00001000	Heap commit	00000000`00001000
Checksum	00000000	Number of dirs	16

The compilation timestamp of Wed Jul 10 00:50:48 2019 is different from the other binaries, indicating it has been in use for some time.

Our detailed analysis of the EoP exploit revealed that the vulnerability it used belongs to the win32k.sys driver and that the EoP exploit was the 0-day exploit because it works on the latest (patched) versions of Windows 7 and even on a few builds of Windows 10 (new

Windows 10 builds are not affected because they implement measures that prevent the normal usage of the exploitable code).

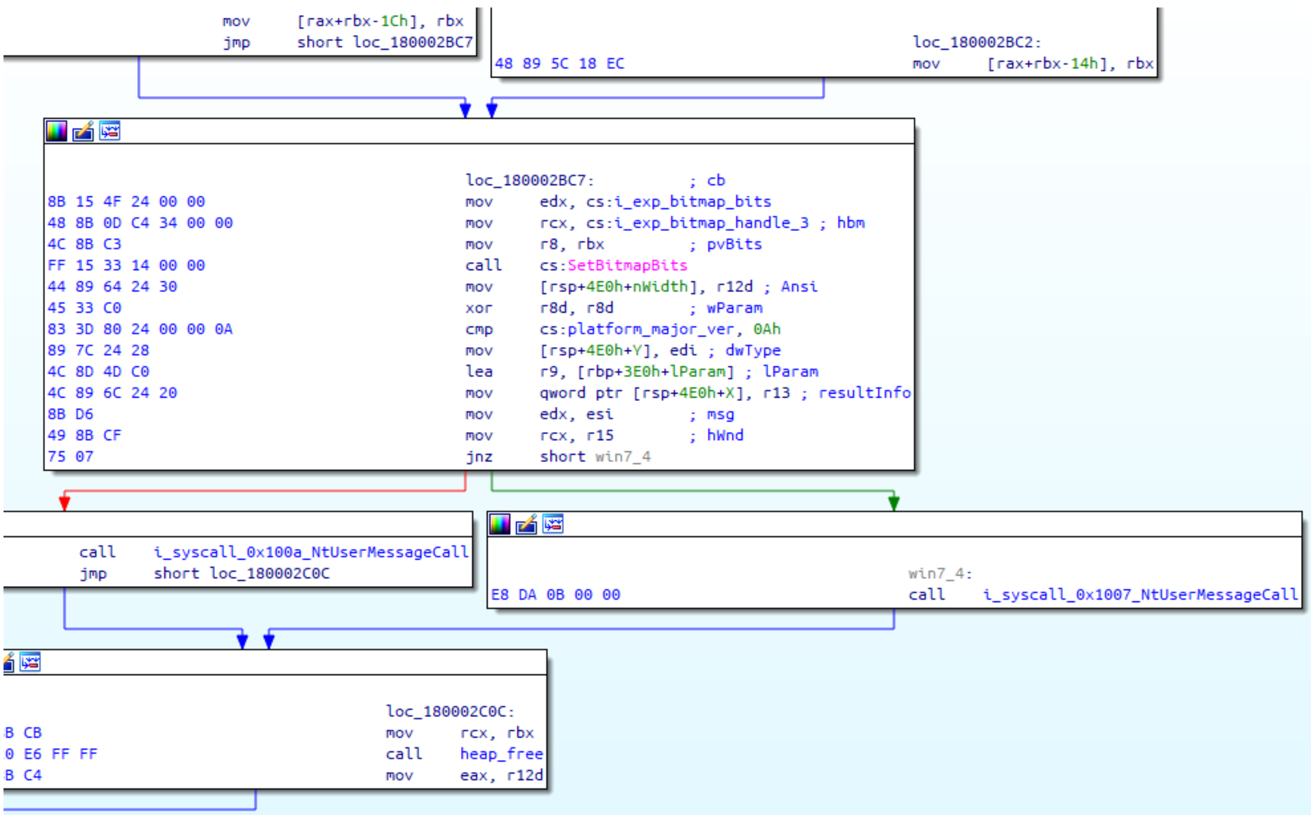
The vulnerability itself is related to windows switching functionality (for example, the one triggered using the Alt-Tab key combination). That's why the exploit's code uses a few WinAPI calls (GetKeyState/SetKeyState) to emulate a key press operation.

At the beginning, the exploit tries to find the operating system version using ntdll.dll's RtlGetVersion call that's used to find a dozen offsets needed to set up fake kernel GDI objects in the memory. At the same time, it tries to leak a few kernel pointers using well-known techniques to leak kernel memory addresses (gSharedInfo, PEB's GdiSharedHandleTable). After that, it tries to create a special memory layout with holes in the heap using many calls to CreateAcceleratorTable/DestroyAcceleratorTable. Then a bunch of calls to CreateBitmap are performed, the addresses to which are leaked using a handle table array.

```
  i_syscall_0x1469_NtUserSetWindowLongPtr();
  i_syscall_0x1469_NtUserSetWindowLongPtr();
  i_syscall_0x1469_NtUserSetWindowLongPtr();
ABEL_20:
  CreateWindowExA(
    0,
    (LPCSTR)32771,                          // #32771 (task switch window)
                                            //
    i_exp_window_name,
    0x10000000u,                            // WS_VISIBLE
    0,
    100,
    100,
    100,
    0i64,
    0i64,
    wnd_class.hInstance,
    0i64);
  i_toggle_alt_key_2();
  clear_memory(v0, 0, (int)i_exp_bitmap_bits);
  if ( byte_180005058 )
    *(_QWORD *)&v0[i_exp_bitmap_bits - 0x18] = v0;
  else
    *(_QWORD *)&v0[i_exp_bitmap_bits - 0x10] = v0;
  SetBitmapBits(i_exp_bitmap_handle_3, 0x1000u, v0);
  if ( platform_major_ver == 10 )
    i_syscall_0x100a_NtUserMessageCall(i_popup_wnd_handle3, 0x14u, 0i64, (LPARAM)lParam, 0i64, 0xE0u, 1);
  else
    i_syscall_0x1007_NtUserMessageCall(i_popup_wnd_handle3, 0x14u, 0i64, (LPARAM)lParam, 0i64, 0xE0u, 1);
  clear_memory(v0, 0, (int)i_exp_bitmap_bits);
  if ( byte_180005058 )
    *(_QWORD *)&v0[i_exp_bitmap_bits - 28] = v0;
  else
    *(_QWORD *)&v0[i_exp_bitmap_bits - 20] = v0;
  SetBitmapBits(i_exp_bitmap_handle_3, i_exp_bitmap_bits, v0);
  if ( platform_major_ver == 10 )
    i_syscall_0x100a_NtUserMessageCall(i_popup_wnd_handle4, 0x14u, 0i64, (LPARAM)lParam, 0i64, 0xE0u, 1);
  else
    i_syscall_0x1007_NtUserMessageCall(i_popup_wnd_handle4, 0x14u, 0i64, (LPARAM)lParam, 0i64, 0xE0u, 1);
  heap_free(v0);
```

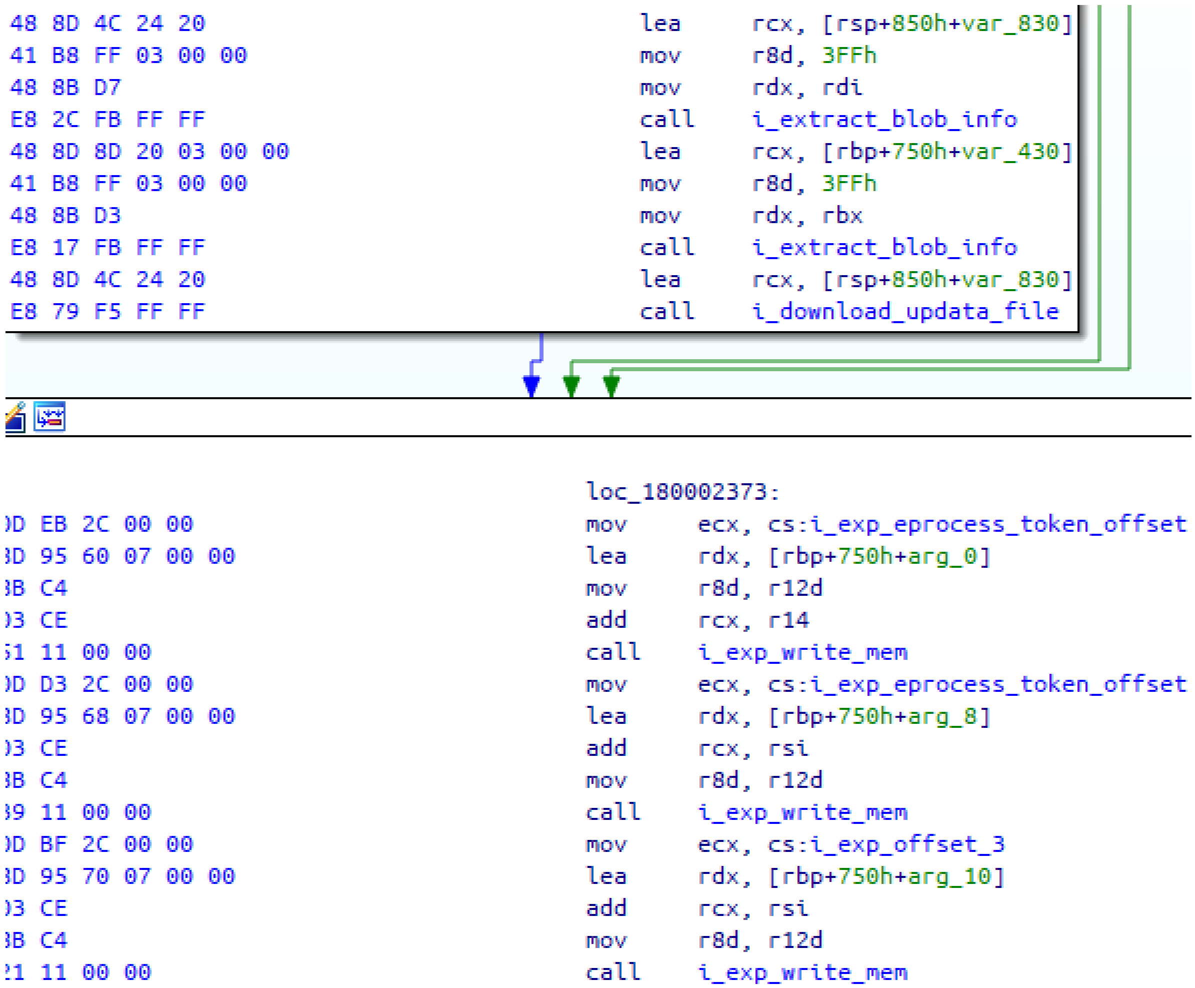
Triggering exploitable code path

After that, a few pop-up windows are created and an undocumented syscall NtUserMessageCall is called using their window handles. In addition, it creates a special window with the class of a task switch window (#32771) and it's important to trigger an exploitable code path in the driver. At this step the exploit tries to emulate the Alt key and then using a call to SetBitmapBits it crafts a GDI object which contains a controllable pointer value that is used later in the kernel driver's code (win32k!DrawSwitchWndHilite) after the exploit issues a second undocumented call to the syscall (NtUserMessageCall). That's how it gets an arbitrary kernel read/write primitive.



Achieving primitives needed to get arbitrary R/W

This primitive is then used to perform privilege escalation on the target system. It's done by overwriting a token in the EPROCESS structure of the current process using the token value for an existing system driver process.

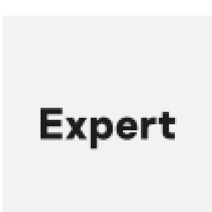


Overwriting EPROCESS token structure

Kaspersky products detect this exploit with the verdict PDM:Exploit.Win32.Generic. These kinds of threats can also be detected with our Sandbox technology. This detection component is a part of our KATA and Kaspersky Sandbox products. In this particular attack sandbox solution can analyze URL/malicious payload in isolated environment and detect the EPROCESS token manipulation.

- Microsoft Windows
- Vulnerabilities and exploits
- Zero-day vulnerabilities

Authors

- 

 AMR

- Expert GReAT

Windows 0-day exploit CVE-2019-1458 used in Operation WizardOpium

---

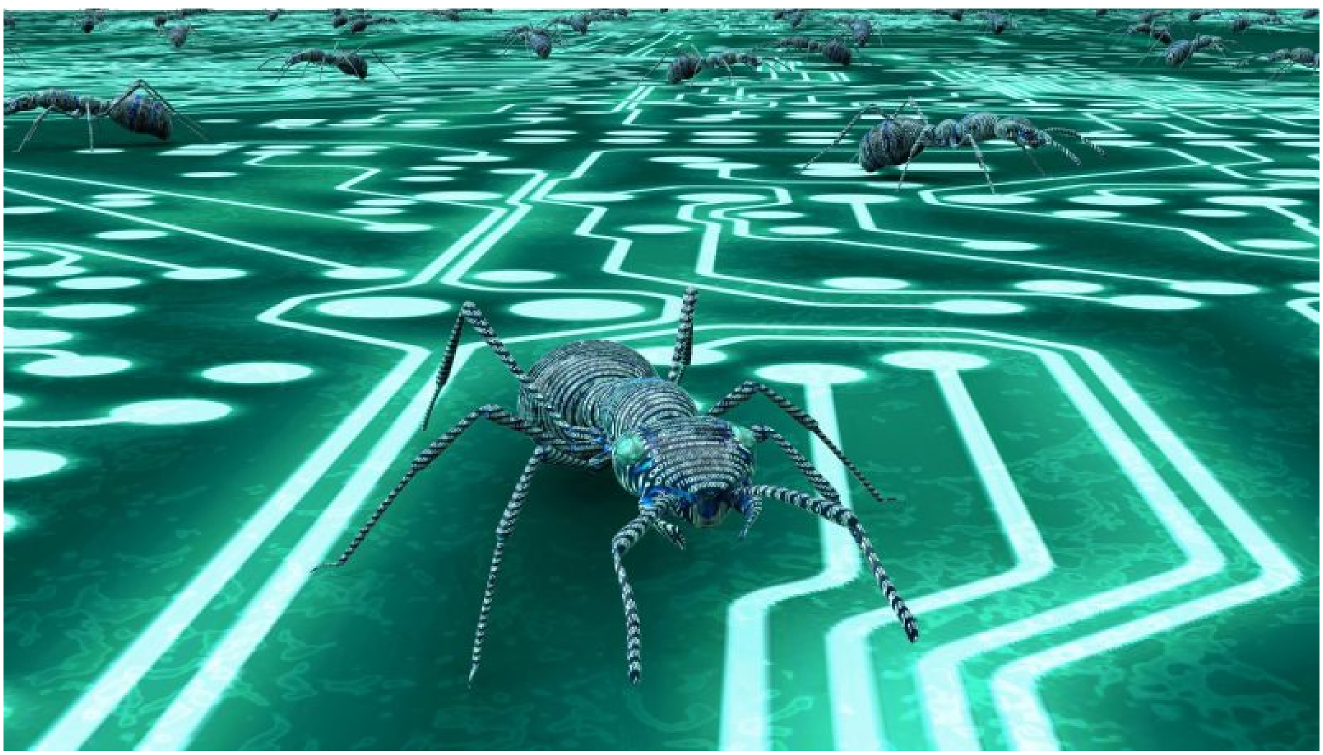
Your email address will not be published. Required fields are marked *

GReAT webinars

13 May 2021, 1:00pm

GReAT Ideas. Balalaika Edition

26 Feb 2021, 12:00pm
17 Jun 2020, 1:00pm
26 Aug 2020, 2:00pm
22 Jul 2020, 2:00pm
From the same authors



IT threat evolution in Q1 2022. Non-mobile statistics

[The Verizon 2022 DBIR](#)

[Evaluation of cyber activities and the threat landscape in Ukraine](#)

[New ransomware trends in 2022]

[APT trends report Q1 2022]

Subscribe to our weekly e-mails

The hottest research right in your inbox

-
-

-