LPE vulnerabilities exploitation on Windows 10 Anniversary Update

Drozdov Yurii

Drozdova Liudmila

Windows 10 KASLR Improvements

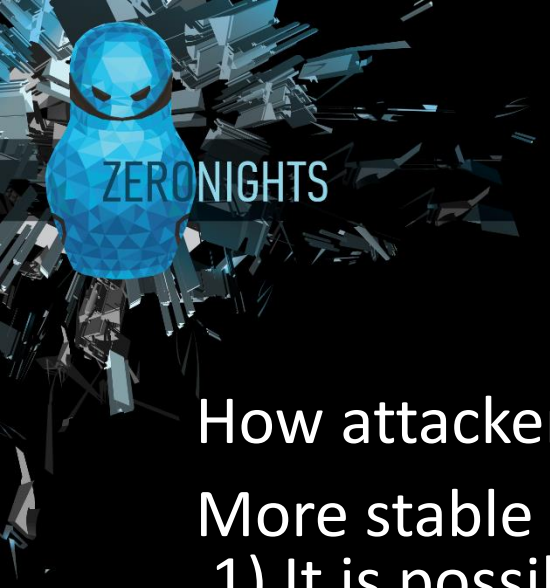
- Windows 10 Anniversary update contains many new exploit mitigations.
- Windows Kernel KASLR Updates for 10 x64 only.
- We'll focus on KASLR Improvements.

- We will describe bypass of GDI objects addresses mitigation (PEB.GDISharedHandleTable doesn't disclose GDI objects addresses after update).

GDI handle table before updates

- GDI objects - Bitmap, Brush, Pen, DC, Font .. etc
- Every GDI object must be added to the handle table after creation.
- Every object in handle table described by following structure
- typedef struct {
  PVOID64 pKernelAddress;
  USHORT wProcessId;
  USHORT wCount;
  USHORT wUpper;
  USHORT wType;
  PVOID64 pUserAddress;
  } GDICELL64;

- Handle table pointer saved in win32k variable gpentHmgr in kernel mode.
- Pointer to GDICELL array is located in PEB.GdiSharedHandleTable in usermode.
- So, we could get GDI objects kernel addresses from usermode by PEB.GdiSharedHandleTable.



Usage of GDI objects addresses

How attackers used GDI kernel objects addresses during exploitation ?

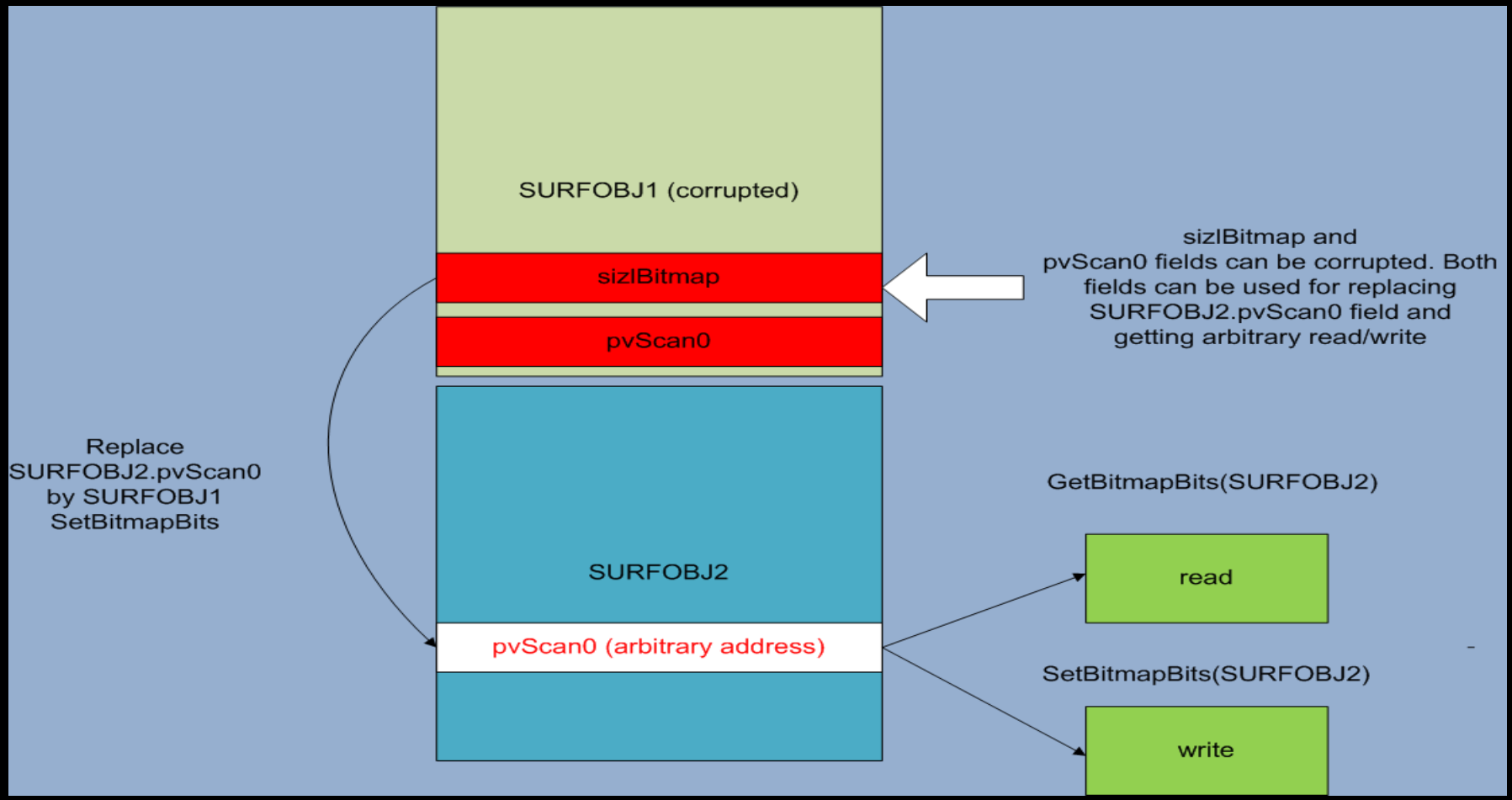
More stable exploitation:
1) It is possible to check if object was allocated on the right place after spray.
2) It is possible to change memory layout as necessary.

Arbitrary read/write:
1) It is possible to change different fields of Bitmap (SURFACE in kernel) and gain arbitrary read and write.

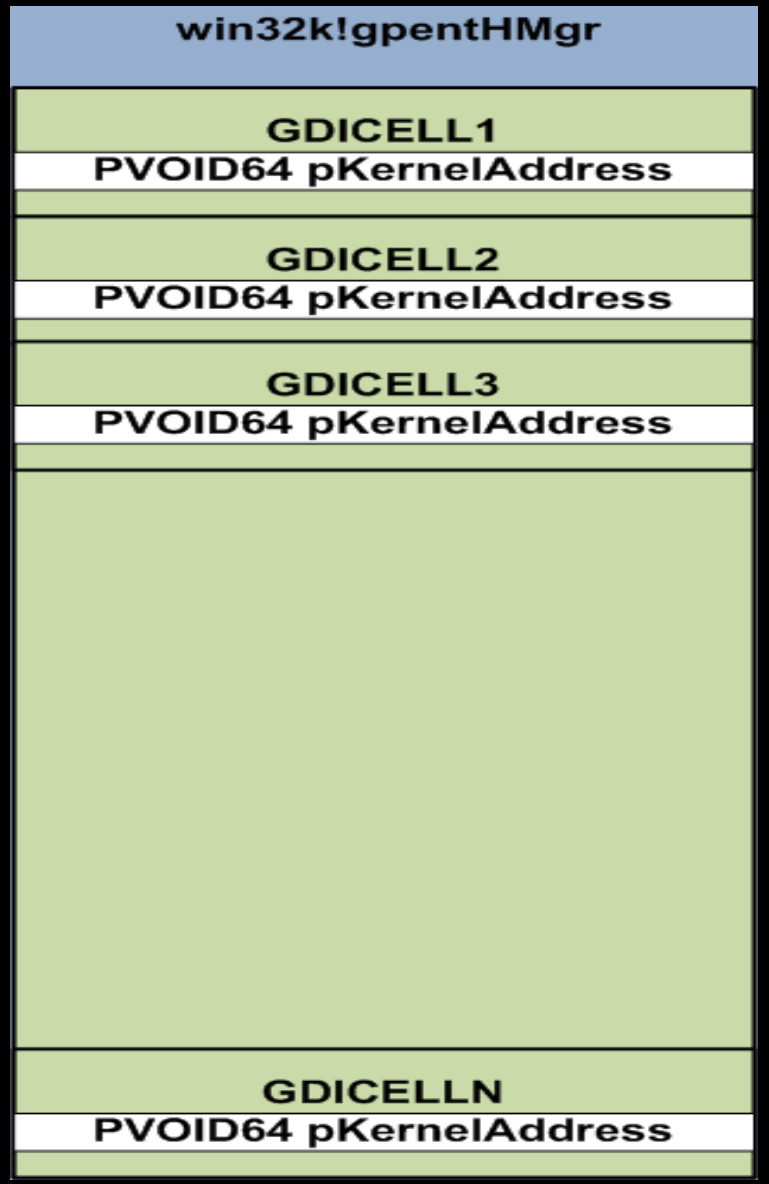
We can use GDI objects for exploitation even if we have vulnerability in different (not win32k) system component.

Usage of GDI objects addresses

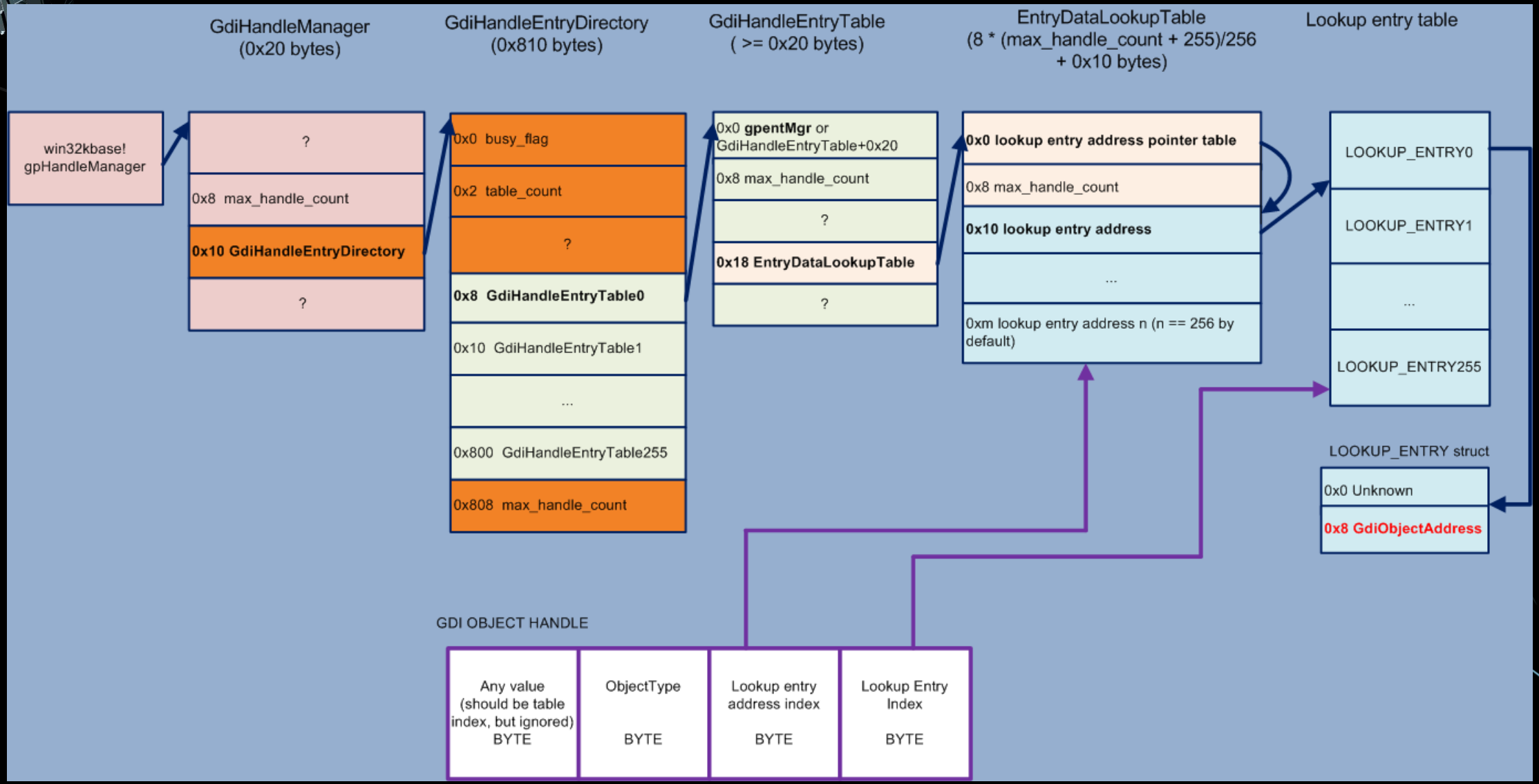
- SURFACE (corruption of it's substructure SURFOBJ) is one of the popular ways to achieve privilege escalation, which is working from Vista to 10.



GDI handle management before updates (kernel mode)



GDI handle management after updates (kernel mode)



HOW TO GET GDI OBJECT ADDRESS BY HANDLE (X64) ?

Before updates Windbg command looked like this (handle in this case - 0x3c05096a)

```
dq poi(poi(win32kbase!gpentHmgr) + 0x18*(0x3c05096a & 0xffff))
```

After changes (handle in this case - 0x1f0509e)

```
dq poi(poi(poi(poi(poi((poi(poi(win32kbase!gpHandleManager)+0x10) + 8 + 0*8)) + 0x18)) + ((0x1f0509ea & 0xffff) / 0x100) * 8) + (0x1f0509ea & 0xff)*0x10 + 8)
```

BYPASS METHODS

GENERAL INFORMATION

Allocation of GDI and USER objects

- USER objects (window, dde conv, hook, menu…):

1. Allocated by HMAllocObject.
2. Can be allocated on Desktop heap, Shared heap, PagedSessionPool (33) or PagedSessionPool|POOL_QUOTA_FAIL_INSTEAD_OF_RAISE (41) depending on object type.
3. We can read Desktop heap from usermode.

- GDI objects (bitmap, brush, pen…):

1. Allocated by HmgAlloc.
2. Can be allocated from PagedSessionPool. Also, there are lookaside lists with previously allocated blocks.
3. On the next slides we'll reference PagedSessionPool as GDI pool.

Desktop heap

- Desktop heap is created with Desktop object.
- Desktop heap is mapped into GUI process address space.
- Desktop heap contains some USER objects and structures.
- We can get user mode mapping of kernel mode address (located on desktop heap) by simple calculation

UserModeAddress = KernelModeAddress – TEB.Win32ClientInfo.ulClientDelta

- Some Windows API are using this feature to avoid entering to kernel when reading structures.

Objects/structures in Desktop heap

Some objects/structures allocated on Desktop heap (following list is incomplete).

Name	Type	Comment
Window (tagWND)	Object	Can be allocated on pool when no desktop specified
Menu (tagMENU)	Object	
Hook (tagHOOK)	Object	
_CALLPROCDATA	Object	
Input context (tagIMC)	Object	
tagITEM	Structure	Popup menu items
tagDESKTOPINFO	Structure	
tagPROPLIST, tagPROP	Structure	Window prop list
tagCLS	Structure	Window class, can be allocated on pool when no desktop specified
tagMENULIST	Structure	

METHODS

Bypass methods

Our methods are based on the following points:

- Anniversary Update suppressed addresses of GDI objects, but USER objects were left intact.
- We still can get addresses of USER objects from gSharedInfo from usermode.
- Also, we can read content of some USER objects that were allocated on Desktop heap.
- Some USER objects and structures are allocated on the GDI pool.

Discussed methods:

- Arbitrary read/write via USER objects (as SURFOBJ alternative)
- Leak of addresses from GDI pool.
- Spray approach.

Arbitrary read/write via USER objects

- We need exact address for corruption when we have vulnerability where it is possible write specific kernel address.
- We could use SURFOBJ address and overwrite some SURFOBJ fields before Anniversary update.
- Now we can use USER objects to gain arbitrary read/write because we can get their addresses.
- The main idea is very similar to arbitrary read/write with 2 SURFACEs.

Arbitrary read/write via USER objects

- Method based on using two USER objects – Window (tagWND) and Menu (tagMENU).
- Both objects are placed on desktop heap.
- tagWND is structure which can have some “extra data”, controlled by user (via SetWindowLongPtr API).
- Size of tagWND extra data is located in field tagWND.cbwndExtra.
- Popup menu (CreatePopupMenu) can contain some items (InsertMenuItem) described by tagITEM structure.
- Pointer to items array is located in tagMENU.rgItems.

Arbitrary read/write via USER objects

- It is possible to read/write different fields of menu items via WinAPI functions (GetMenuItemRect,GetMenuItemInfo/SetMenuItemInfo).
- We can easily place tagWND before tagMENU in memory (because we can get addresses of both objects).
- If tagWND.cbwndExtra was corrupted because of some vulnerability, we can read/write tagMENU.rgItems pointer via SetWindowLongPtr.
- When pointer to rgItems was replaced to some arbitrary address, we can read/write it via menu items API.

Arbitrary read via USER objects

- Arbitrary read

1. tagITEM structure is located on desktop heap. Windows API GetMenuItemInfo is using this fact to read item's content from usermode. So, if we replaced tagITEM pointer in tagMENU to arbitrary kernel address, we have to use another api, which can get info from kernel mode.
2. GetMenuItemRect is able to do that, but it returns result in specific format.
3. GetMenuItemRect reads 2 fields – tagITEM.cxItem, tagITEM.cyItem, so we can calculate 64-bit value from 2 dwords.

Arbitrary read example

```
GetMenuItemRect(hWnd, hMenu, 0, &rect) ;
ldw =(rect.right - rect.left); //-> low dword
hdw=(rect.bottom - rect.top);// -> high dword
result = MAKEULONGLONG(ldw, hdw);
```

Arbitrary write via USER objects

BOOL WINAPI SetMenuItemInfo(HMENU hMenu, UINT uItem, BOOL fByPosition, LPMENUITEMINFO lpmii);

```
typedef struct tagMENUITEMINFO {
    UINT      cbSize;
    UINT      fMask; // MIIM_DATA , MIIM_ID …
    UINT      fType;
    UINT      fState; // MIIM_STATE
    UINT      wID; // -> MIIM_ID
    HMENU     hSubMenu; // ->MIIM_SUBMENU
    HBITMAP   hbmpChecked;
    HBITMAP   hbmpUnchecked;
    ULONG_PTR dwItemData; //-> MIIM_DATA
    LPTSTR    dwTypeData;  // -> MIIM_STRING
    UINT      cch;
    HBITMAP   hbmpItem;
} MENUITEMINFO, *LPMENUITEMINFO;
```

Arbitrary write via USER objects

- Arbitrary write

1. We can use API SetMenuItemInfo(MIIM_DATA) for writing content of arbitrary address.
2. MIIM_DATA allows to write DWORD64 on x64 system.
3. SetMenuItemInfo writes tagITEM.dwItemData field.
4. If we call “write” from wow64 process, we need to call service NtUserThunkedMenuItemInfo directly, because wow64 stub doesn’t allow to use x64 MENUITEMINFOW structure and as the result we can’t write 64 bit field (only 32-bit one).

In both cases (read and write) we need to calculate arbitrary address according to tagITEM field offset.

Read/Write address corrections

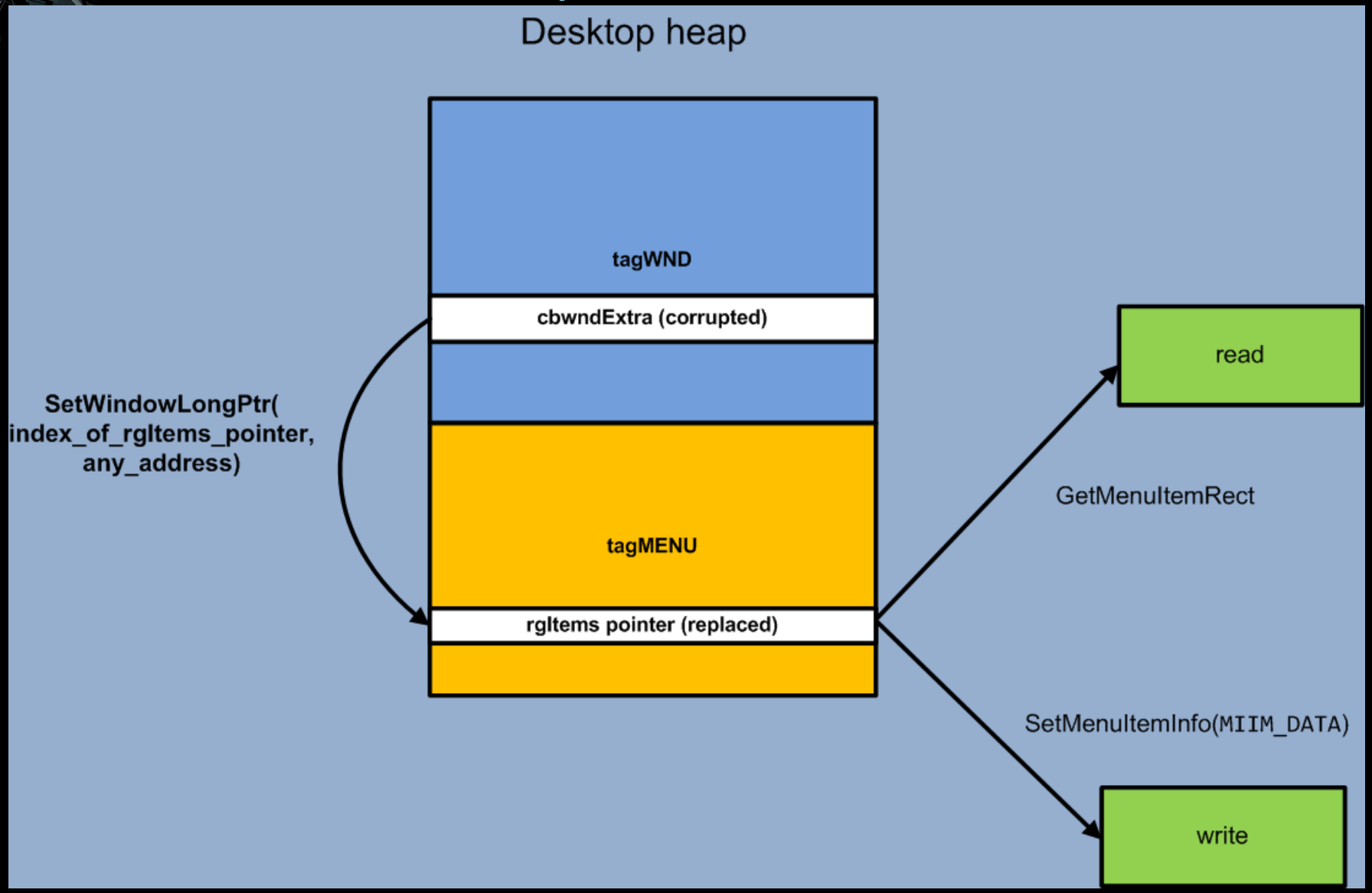
- Set arbitrary address for read (ArbAddr)

```
SetWindowLongPtr(hWndCorrupted, ExtraDataIndex, (LONG_PTR) ArbAddr - 0x48);
```

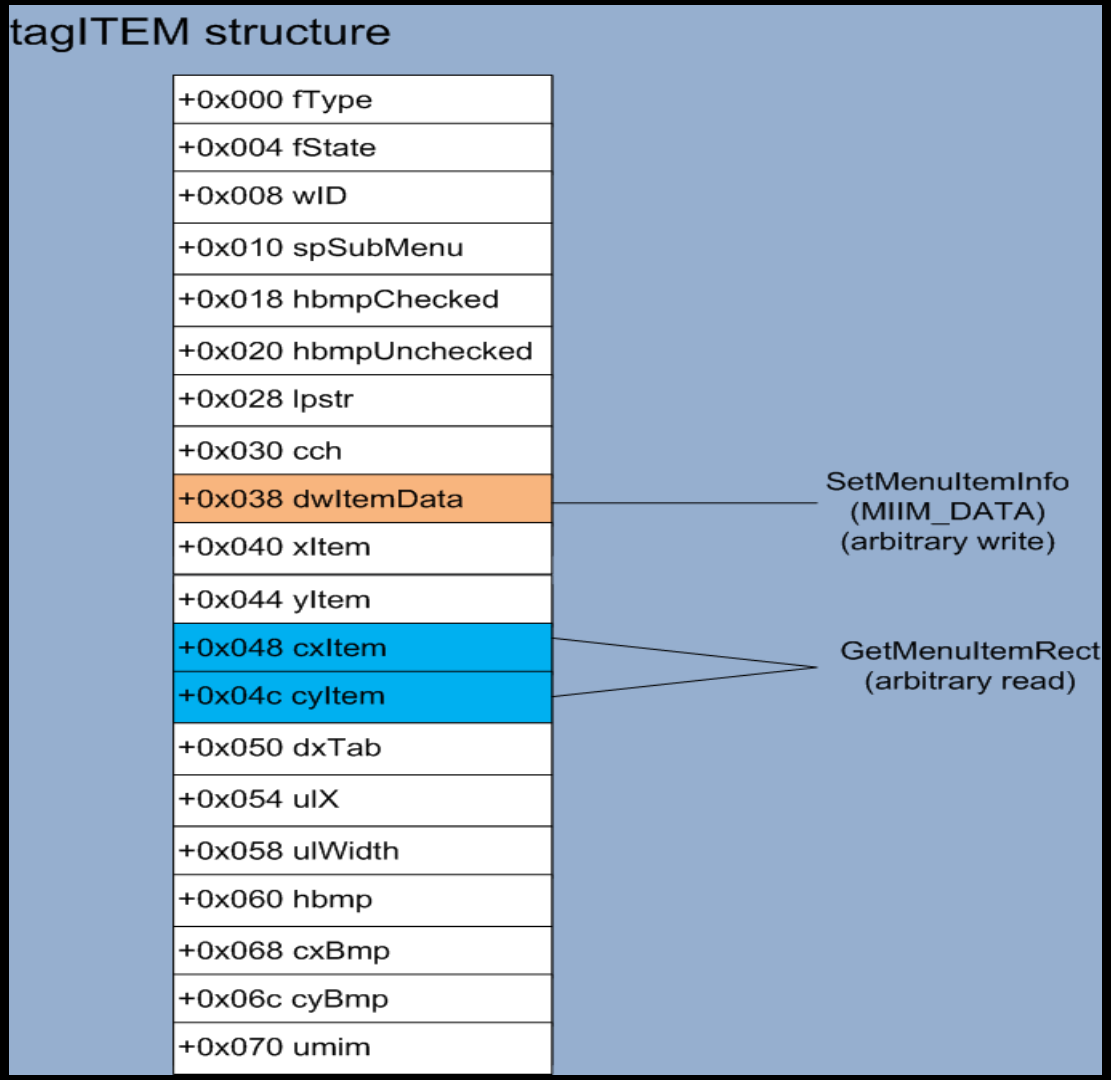
- Set arbitrary address for write

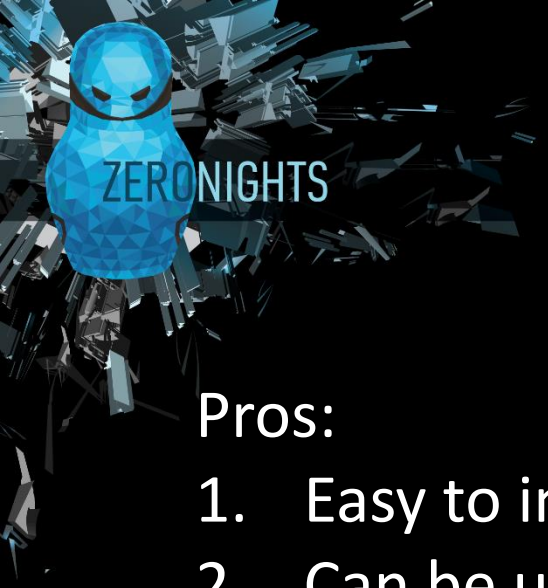
```
SetWindowLongPtr(hWndCorrupted, ExtraDataIndex, (LONG_PTR) ArbAddr - 0x38);
```

Arbitrary read/write via USER objects



Arbitrary read/write via user objects





Pros and Cons

Pros:

1. Easy to implement.
2. Can be used under wow64 process and pure x64 process.
3. Access to full address space on x64.
4. Works under low integrity process.

Cons (Limitation):

1. We need vulnerability, which can corrupt memory in desktop heap or vulnerability where we can write specific address (tagWND. cbwndExtra offset).
2. Desktop heap must be mapped in exploitation context.

GDI pool addresses leaks

- Preparing memory and use old SURFACE technique:

- We can leak addresses of some GDI structures or USER structures/data allocated on GDI pool.
- We can find USER objects which are allocated on the GDI pool.

- Based on both methods we can prepare memory for exploitation.
- Main technique is allocation of GDI object (SURFOBJ) on place of freed object/structure which address we can get. Then we can use old SURFACE corruption technique.

- Using of USER objects was described at Ekoparty 2016, we'll show using of data and structures.

GDI structure address leak

We can get address of specific GDI structure by reading Desktop heap.

Base information:

- Every window class (RegisterClass API) described by tagCLS structure.
- tagCLS contains pointer to tagDCE structure (pdce field) if class was created with CS_CLASSDC style.
- tagCLS is located on Desktop Heap, so we can read it content.
- tagDCE is allocated on PagedSessionPool as GDI objects (CreateCacheDC).
- tagDCE has constant length – 0x60 (Win 10 x64)

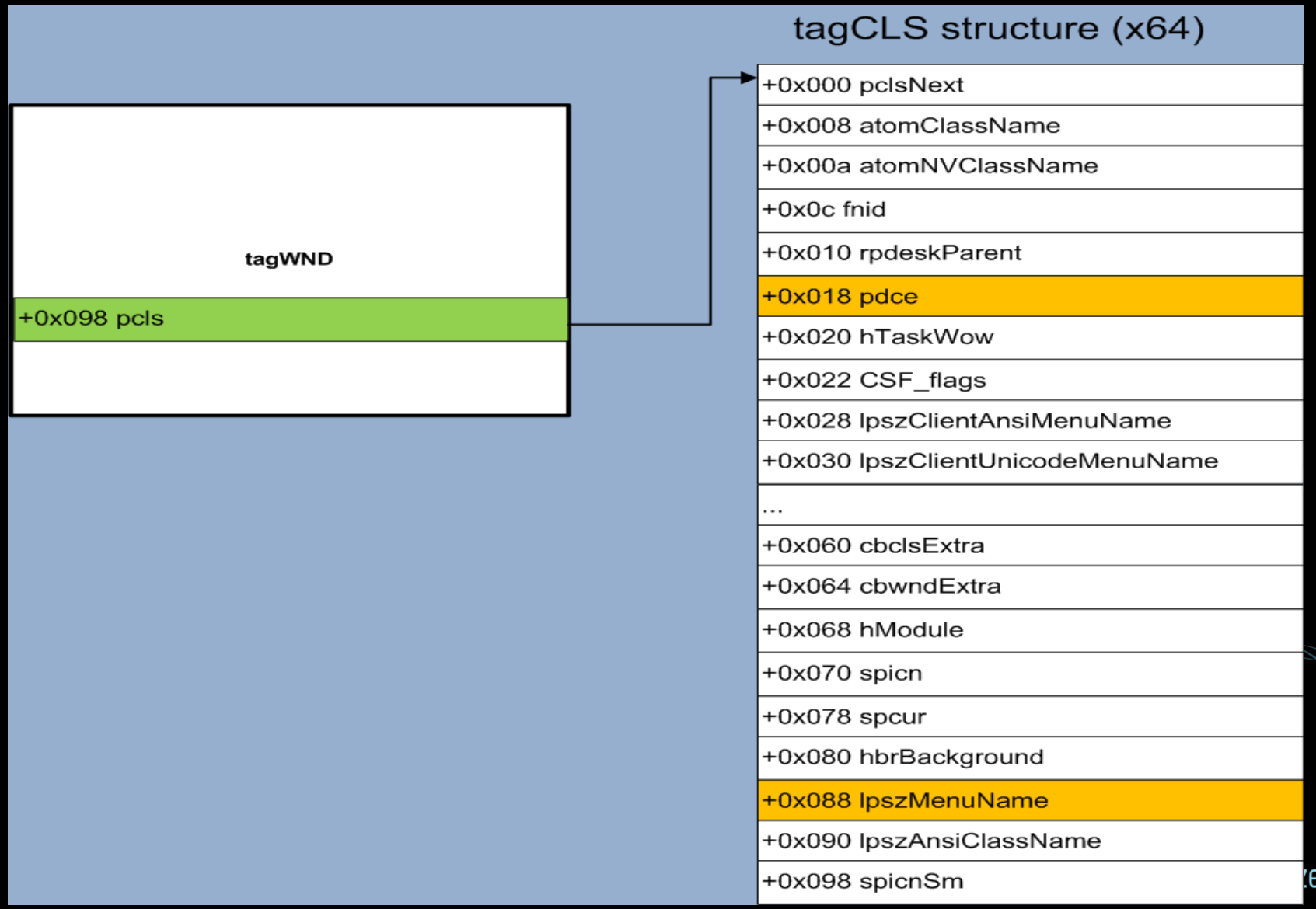
GDI structure address leak

Getting tagDCE pointer:

1. RegisterClass with CS_CLASSDC style.
2. Create window for this class in order to cache DC and allocate tagDCE.
3. Get pointer to tagWND, then to tagCLS (tagWND.pcls) and finally tagCLS.pdce.

We need to call DestroyWindow + UnregisterClass in order to free tagDCE.

GDI structure address leak



Getting address of tagCLS.pdce

We can get pdce pointer by using following code

```
DWORD64 pWndUMAddr = GetUserObjectKAddr(hWnd) - dwClientDelta;
DWORD64 pClassUMAddress = *(DWORD64 *)(pWndUMAddr + 0x98) - dwClientDelta;
DWORD64 pDceAddress = *(DWORD64 *)(pClassUMAddress + 0x18);
```

USER structures/data allocated on GDI pool

- Some user objects are allocated on GDI pool, but they are not alone!
- There are many user structures allocated on GDI pool (tagPOPUPMENU, tagWND.pTransform, tagSBTRACK …). We can get their addresses by reading object's content from desktop heap (as USER objects contain pointers to this structures).

- We found, that tagCLS.lpszMenuName (see previous slide) allocated on GDI pool.
- This tagCLS field represents WNDCLASSEX.lpszMenuName (UNICODE).
- We can easily allocate it by RegisterClass and free by UnregisterClass.
- Also we can control size of this allocation.

Getting address of tagCLS.lpszMenuName

We can easily get lpzsMenuName address by using following code

```
DWORD64 pWndUMAddr = GetUserObjectKAddr(hWnd) - dwClientDelta;
DWORD64 pClassUMAddress = *(DWORD64 *)(pWndUMAddr + 0x98) - dwClientDelta;
DWORD64 lpszMenuName = *(DWORD64 *)(pClassUMAddress + 0x88);
```

Objects on GDI pool

- We can get addresses of different object/structures allocated in GDI pool.

Description	Disclose method	Examples
USER objects	Reading gSharedInfo	WinEvent, Clip data…
Pointers to structures allocated in GDI pool	Reading content of user objects, allocated in desktop heap.	tagCLS.pdce
Pointers to data allocated in GDI pool	Reading content of user objects, allocated in desktop heap.	tagCLS.lpszMenuName

Useful objects/structures in GDI pool

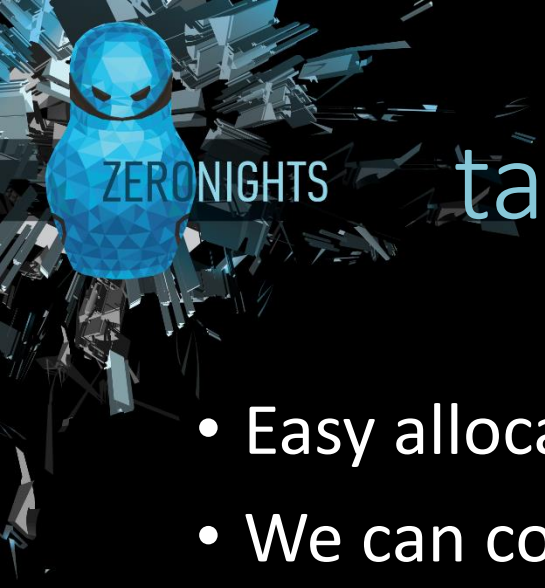
- Some USER objects/structures isn't possible to use during exploitation because we can't easily allocate/control them.
- We made list of potentially "exploitable" objects and structures with their pros and cons.
- Demo of usage of accelerator tables and clip data was shown at Ekoparty 2016. We'll show alternative.
- Following list is incomplete, there are other candidates.

Object/structure	Type	Size (x64)	Pros	Cons
Clip data	USER object	Controlled	Controlled size	Need clipboard access, which can be restricted by some sandboxes
WinEvent	USER object	0x60	Easy allocation and destruction	Static object size
Timer	USER object	0x88	Easy allocation and destruction	Static object size, need to scan user handle table to get object address as SetTimer doesn't return handle
tagCURSOR	USER object	0x98	Easy allocation and destruction	Static object size
Accelerator table	USER object	Controlled	Easy allocation and destruction, controlled size	We need to count tagACCEL to calculate size of allocation
tagCLS.lpszMenuName	USER data	Controlled	Easy allocation and destruction, controlled size (size of unicode string)	
tagCLS.pdce (tagDCE)	GDI structure	0x60	Easy allocation and destruction	Static structure size

Spray approach (tagCLS.lpszMenuName)

1. Allocate tagCLS structure by calling RegisterClass with controlled lpszMenuName.
2. Allocate Window for class (as we don’t want to scan desktop heap) and get addresses of allocated tagCLS.lpszMenuName.
3. DestroyWindow and free tagCLS by UnregisterClass.
4. Allocate SURFOBJ (Bitmap)*.
5. Now, we can expect that SURFOBJ will be allocated on place of lpszMenuName.

*As there is lookaside list for previous GDI objects allocations, we need to fill lookaside before trying to allocated bitmap on freed space.



tagCLS.lpszMenuName spray advantages

- Easy allocation and destruction.
- We can control size of tagCLS.lpszMenuName.
- tagCLS.lpszMenuName field allocated on GDI pool.
- We can easily get address of field.
- Method is working under low integrity process.
- Big strings are allowed (> 4kb).

Conclusion

- We can use method of arbitrary read/write via USER objects when we can write specific address.
- We can use tagCLS.lpszMenuName spray for other vulnerabilities.
- We also can use prediction of GDI objects addresses to make exploitation more stable.
- We can use GDI structures(tagDCE), USER objects, USER structures allocated on GDI pool for exploitation. Choice of object will depend on vulnerability.

DEMO

THANK YOU!

QUESTIONS?

Links

- https://www.blackhat.com/docs/us-16/materials/us-16-Weston-Windows-10-Mitigation-Improvements.pdf
- https://msdn.microsoft.com/ru-ru/library/windows/desktop/ms725486(v=vs.85).aspx
- https://www.coresecurity.com/system/files/publications/2016/10/Abusing-GDI-Reloaded-ekoparty-2016_0.pdf
- http://cvr-data.blogspot.ru/
- http://www.slideshare.net/DefconRussia/windows-10-gdi-68999382
- https://msdn.microsoft.com/ru-ru/library/windows/desktop/ms724291(v=vs.85).aspx