What the hell is Windows CLIP service?

Philippe Laulheret

CISCO TALOS

Philippe Laulheret

@phLaul

Senior Vulnerability Researcher, Cisco Talos

Focus: Windows, …

CISCO TALOS

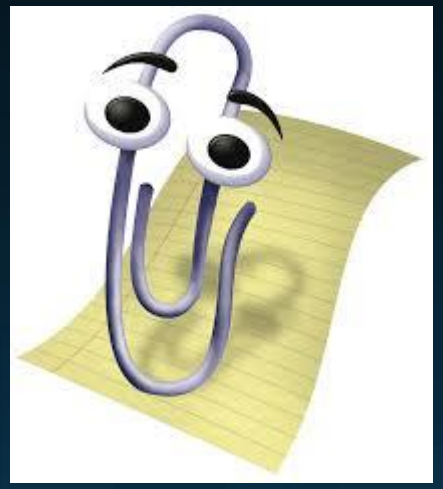
WHAT TO EXPECT FROM THIS TALK?

CISCO TALOS

WHAT IS CLIPSP?

CLIPSP.sys ?

- CLiP = Client License Platform
- SP = System Policy
- Bunch of components:
  - Clipc.dll → client
  - ClipSVC.dll → RPC backend
  - ClipSp.sys → Driver



Past Work

- ClipSp:
  - KiFilterFiberContext's github
    - https://github.com/KiFilterFiberContext/windows-software-policy
  - Keyhole - https://massgrave.dev/blog/keyhole (published after our research)
- Warbird:
  - Airbus-seclab's warbirdvm analysis
    - https://github.com/airbus-seclab/warbirdvm
  - Alex Ionescu's EkoParty talk
    - http://publications.alex-ionescu.com/EkoParty/EkoParty%202017%20-%20The%20Bird%20that%20killed%20Arbitrary%20Code%20Guard.pdf
  - DownWithUp's blog
    - https://downwithup.github.io/blog/post/2023/04/23/post9.html
- CVEs:
  - CVE-2023-28273 (@ezrak1e)
  - CVE-2023-35362 (@ezrak1e)

CISCO TALOS

Why?

- Looking for EoP in windows driver
  - Potential for Bug bounty
  - Figure out what threat actors tend to do
- Curiosity!
  - Importing in IDA looked like…

Output

```
1C00AC3C8: function entry has inval
1C00AC3D4: function entry has inval
1C00AC3E0: function entry has inval
1C00AC428: function entry has inval
too many function entries with inva
210 out of 1768 function entries ha
Applying fixups...
 18. Creating a new segment  (00000001C00AE548-00000001C00B0000) ... ... OK
PDB: using PDBIDA provider
PDB: downloading http://msdl.microsoft.com/download/symbols/clipsp.pdb/EA3FBCA0CBCD415D8F2C5AD3F06EDA071/clipsp.pdb =
Could not find PDB file 'clipsp.pdb'.
```

Please confirm

C:\re\temp\ClipSp_april.sys: failed to load pdb info.

Do you want to browse for the pdb file on disk?

Don't display this message again

Yes No

```
__int64 __fastcall ClipSpInitialize(unsigned int a1, __int64 a2)
{
  unsigned int v4; // ebx
  __int64 v5; // rcx

  v4 = -1073741595;
  if ( dword_1C00A7CB8 || (dword_1C00A7CB8 = 1, (int)sub_1C003427C() >= 0) )
  {
    if ( (int)sub_1C00014D0(&unk_1C0062B00, &unk_1C00A66F0) >= 0 )
    {
      if ( (int)sub_1C00014D0(&unk_1C0063610, &unk_1C00A69A0) >= 0 )
      {
        v4 = ((__int64 (__fastcall *)(__int64, __int64, _QWORD))loc_1C00F5008)(v5, a2, a1);
        sub_1C0001444(&unk_1C0063610, &unk_1C00A69A0);
      }
      sub_1C0001444(&unk_1C0062B00, &unk_1C00A66F0);
    }
  }
  return v4;
}
```

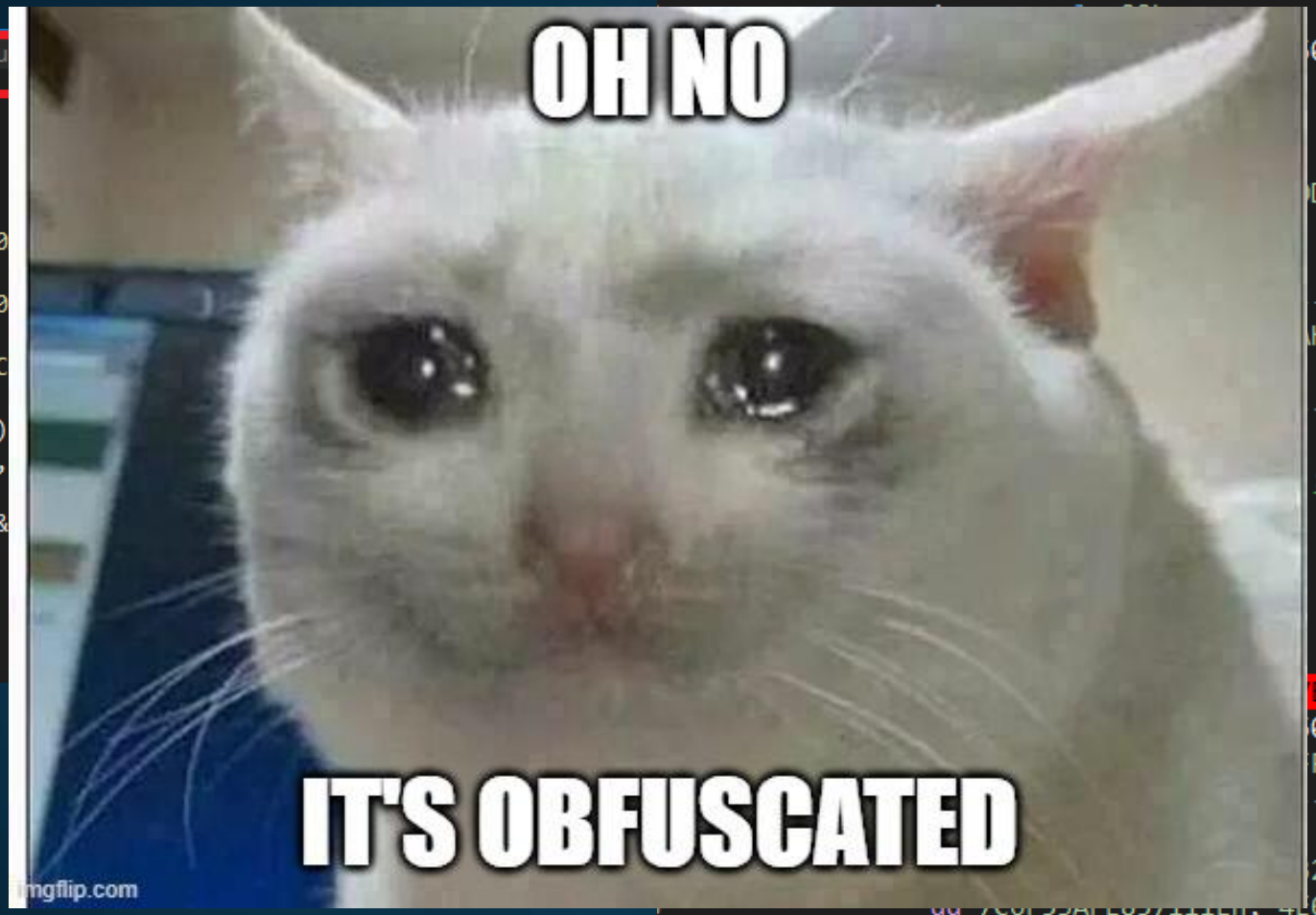
```
; ---------------------------------------------------------------------------

loc_1C00F5008:                          ; CODE XREF: ClipSpInitialize+69↑p
                                        ; DATA XREF: .pdata:00000001C00AC350↑o
                in      al, 22h
                jno     short loc_1C00F5024
                push    rcx
                rol     bl, 1
                popfq
                cmp     edi, [rbx+rcx-5DDC2BCAh]
                pop     rsi
                insb
---------------------------------------------------------------------------
                db 0C7h, 0A4h, 0B1h, 0DAh, 46h, 2Bh, 1Fh
                db 0Eh, 38h, 0CFh, 3Ch
---------------------------------------------------------------------------

4:                                      ; CODE XREF: PAGEwx1:00000001C00F500A↑j
                db      41h
                in      al, dx
                in      al, 50h
                db      36h
                xchg    eax, ebp
                call    near ptr 2140247BBh
                jbe     short loc_1C00F5076
                mov     word ptr ds:0FFFFFFFF86166427h[rax], es
; ---------------------------------------------------------------------------
                db 8Ch, 3Ah, 0CCh, 0BFh, 61h, 0EBh, 0C2h
                dq 0A89C25D356A4D0AEh, 3202A1955C704D98h, 0FBE6ED96481B2A55h
                dq 7C6F53AFE857111Eh, 4E6E8C53AF8E7CDAh, 835738AB949C224Bh
; ---------------------------------------------------------------------------
                mov     ebp, 84B1C11Ah
```

```
__int64 __fastcall ClipSpInitialize(u
{
  unsigned int v4; // ebx
  __int64 v5; // rcx

  v4 = -1073741595;
  if ( dword_1C00A7CB8 || (dword_1C00
  {
    if ( (int)sub_1C00014D0(&unk_1C00
    {
      if ( (int)sub_1C00014D0(&unk_1C
      {
        v4 = ((__int64 (__fastcall *)
        sub_1C0001444(&unk_1C0063610,
      }
      sub_1C0001444(&unk_1C0062B00, &
    }
  }
  return v4;
}
```




```
; ---------------------------------------------------------------------------

loc_1C00F5008:                          ; CODE XREF: ClipSpInitialize+69↑p
                                        ; DATA XREF: .pdata:00000001C00AC350↑o
                                                024
                                                DC2BCAh]
                                                h, 46h, 2Bh, 1Fh
                                                ; CODE XREF: PAGEwx1:00000001C00F500A↑j
                                                BBh
                                                076
                                                FFFFF86166427h[rax], es
                                                61h, 0EBh, 0C2h
                                                202A1955C704D98h, 0FBE6ED96481B2A55h
                                                6E8C53AF8E7CDAh, 835738AB949C224Bh
; ---------------------------------------------------------------------------
                                        mov     ebp, 84B1C11Ah
```

DEOBFUSCATION TIME!

Warbird TL;DR;

- Microsoft proprietary Obfuscator
- Many obfuscation options
  - Data structures (e.g communication w/ ClipSp)
  - Self-modifying code
    - Yes! In the kernel 😱
  - Syscall
    - Decrypt/Re-encrypt
    - Payload Execution (!!)

```
1  __int64 __fastcall ClipSpInitialize(unsigned int a1, __int64 a2)
2  {
3    unsigned int v4; // ebx
4    __int64 v5; // rcx
5
6    v4 = -1073741595;
7    if ( dword_1C00A7CB8 || (dword_1C00A7CB8 = 1, (int)sub_1C003427C() >= 0) )
8    {
9      if ( (int)sub_1C00014D0(&unk_1C0062B00, &unk_1C00A66F0) >= 0 )
10     {
11       if ( (int)sub_1C00014D0(&unk_1C0063610, &unk_1C00A69A0) >= 0 )
12       {
13         v4 = ((__int64 (__fastcall *)(__int64, __int64, _QWORD))loc_1C00F5008)(v5, a2, a1);
14         sub_1C0001444(&unk_1C0063610, &unk_1C00A69A0);
15       }
16       sub_1C0001444(&unk_1C0062B00, &unk_1C00A66F0);
17     }
18   }
19   return v4;
20 }
```

```
__int64 __fastcall ClipSpInitialize(unsigned int a1, __int64 a2)
{
  unsigned int v4; // ebx
  __int64 v5; // rcx

  v4 = -1073741595;
  if ( dword_1C00A7CB8 || (dword_1C00A7CB8 = 1, (int)sub_1C003427C() >= 0) )
  {
    if ( (int)decrypt1(&unk_1C0062B00, &unk_1C00A66F0) >= 0 )
    {
      if ( (int)decrypt1(&unk_1C0063610, &unk_1C00A69A0) >= 0 )
      {
        v4 = ((__int64 (__fastcall *)(__int64, __int64, _QWORD))do_ClipSpInitialize)(v5, a2, a1);
        encrypt1(&unk_1C0063610, &unk_1C00A69A0);
      }
      encrypt1(&unk_1C0062B00, &unk_1C00A66F0);
    }
  }
  return v4;
}
```

Yuck! (Feistel cipher)

```
v26 = v23 ^ (WORD1(v19) * __ROR4__(HIDWORD(v19) + v22, 10) - __ROR4__(v22, 29));
v27 = v22 ^ ((unsigned __int16)v19 * __ROR4__(v26 ^ HIDWORD(v19), 22) - __ROR4__(v26, 8));
v28 = v26 ^ ((unsigned __int16)v19 * (HIWORD(v19) ^ __ROR4__(v27, 15)));
v29 = v27 ^ ((v28 >> 9) + WORD2(v19) * __ROL4__(v28 ^ WORD1(v19), 3));
v30 = v28 ^ __ROR4__(v29, 28) ^ (WORD1(v19) * __ROR4__(HIDWORD(v19) ^ v29, 9));
v31 = v29 ^ (__ROR4__(v30, 12) + (unsigned __int16)v19 * __ROR4__(v30 - HIDWORD(v19), 14));
v32 = v30 ^ __ROR4__(v31, 11) ^ (HIWORD(v19) * __ROL4__(v31 ^ WORD2(v19), 2));
v33 = v31 ^ (v32 - WORD2(v19) - v19);
LODWORD(v106) = v32 ^ (WORD1(v19) * __ROL4__(v33 ^ HIWORD(v19), 2) - __ROR4__(v33, 18));
v34 = v33 ^ ((unsigned __int16)v19 * __ROR4__(v106 - HIDWORD(v19), 18) - __ROR4__(v106, 9));
v24 = v118;
v35 = &v116;
v36 = v118;
HIDWORD(v106) = v34;
v37 = (char *)v14;
do
{
  v38 = *v37++;
  *( BYTE *)v35 = v38;
```

CISCO TALOS

How to deal with the Binary Obfuscation?

- Dump the driver while loaded in memory?
  - Can't, code is being re-obfuscated once function is over
- Reimplement the algorithm?
  - Sure, but super tedious, multiple functions, weird API calls, etc.
  - We'll see that approach later….
- Lazy+smart move! Leverage existing code:
  - Option 1: load the driver and call the function yourself (might fail due to certain kernel api…)
  - Option 2: Emulate the code and stub all the code we don't need

Someone already did option 2!

See code at: https://github.com/KiFilterFiberContext/windows-software-policy/blob/master/clipsp-unpack.py

Tl;dr:

- Use Quiling to emulate the driver + other Windows components necessary to load it
  - Quling Framework does a lot of heavy lifting for us
  - Slow to run, quick to implement
- Run through the init of clipsp and dump all the Mdl being allocated
  - Problem: might miss some code, and pretty messy
  - Let's improve it!

Improving the Deobfuscation script

```
if ( (int)decrypt1(&unk_1C0062B00, &unk_1C00A66F0) >= 0 )
{
  if ( (int)decrypt1(&unk_1C0063610, &unk_1C00A69A0) >= 0 )
  {
    v4 = ((__int64 (__fastcall *)(__int64, __int64, _QWORD))do_ClipSpInitialize)(v5, a2, a1);
    encrypt1(&unk_1C0063610, &unk_1C00A69A0);
  }
  encrypt1(&unk_1C0062B00, &unk_1C00A66F0);
```

Idea:

1. Cross reference all the calls to decrypt1 function
2. Backtrack to recover the two arguments of the function call
3. List all the call to decrypt1(arg1, arg2)
4. Use Quiling to execute all the instances of the decryption function
5. Dump the whole memory range, and import that in IDA

1. Cross reference calls to decrypt1

```
deobfuscate_1_ea = 0x01C00014D0
deobfuscate_3_ea = 0x01C000155C

def get_unique_keys(func_ea):
    skip_section = [".pdata"]
    res =  {}
    for ea in XrefsTo(func_ea):
        call_site = ea.frm
        if get_segm_name(call_site) in skip_section:
            print("Skipping... " + hex(call_site) + " in seg " + get_segm_name(call_site) )
            continue
        rcx = get_register_assignement(call_site, "rcx")
        rdx = get_register_assignement(call_site, "rdx")
        print("0x{:x}, 0x{:x}, 0x{:x}".format(call_site, rcx, rdx))
        res[rcx] = rdx

    return  res


res1 = get_unique_keys(deobfuscate_1_ea)
res2 = get_unique_keys(deobfuscate_3_ea)
```

2. Retrieve rcx and rdx (arg1 and arg2)

```python
# Only returns RCX and RDX
def get_register_assignement(ea, reg):
    if reg == "rcx":
        reg_val = procregs.rcx.reg
    elif reg == "rdx":
        reg_val = procregs.rdx.reg
    else:
        raise "Unkwnon reg"


    # Should check we have a lea but for now let's skip that check
    for i in range(0, 20):
        ea = prev_head(ea)
        #print(ea)
        if get_operand_type(ea, 0) == o_reg:
            if get_operand_value(ea, 0) == reg_val:
                return get_operand_value(ea, 1)
    return None
```

```
                                        ; CODE XREF:
lea     rdx, unk_1C00A66F0
lea     rcx, unk_1C0062B00
call    deobfuscate_1
test    eax, eax
js      short loc_1C00BD706
lea     rdx, dword_1C00A69A0
lea     rcx, stru_1C0063610
call    deobfuscate_1
test    eax, eax
```

CISCO TALOS

3. List all the calls

```
res1 = get_unique_keys(deobfuscate_1_ea)
res2 = get_unique_keys(deobfuscate_3_ea)

print("Decrypt1")
for k in res1.keys():
    print("decrypt1(ql, 0x{:x}, 0x{:x})".format(k, res1[k]))


print("Decrypt2")
for k in res2.keys():
    print("decrypt2(ql, 0x{:x}, 0x{:x})".format(k, res2[k]
```

```
Skipping... 0x1c00a8024 in seg .pdata
Decrypt1
decrypt1(ql, 0x1c0063610, 0x1c00a69a0)
decrypt1(ql, 0x1c0064660, 0x1c00a6da0)
decrypt1(ql, 0x1c0062b00, 0x1c00a66f0)
Decrypt2
decrypt2(ql, 0x1c0063cc0, 0x1c00a6b40)
decrypt2(ql, 0x1c0063560, 0x1c00a6978)
decrypt2(ql, 0x1c0064900, 0x1c00a6e38)
```

4. Run the Qiling script

```
# See ClipSpInitialize
def decrypt1(ql, rcx, rdx):
        ql.arch.regs.rcx = rcx
        ql.arch.regs.rdx = rdx

        if not gCanary:
                ql.run(begin=0x001C00014D0 , end=0x01C0001533)
        else:
                 ql.run(begin=0x001C0001740 , end=0x01C0001794)


# See ClipSpUnInitilaize
def decrypt2(ql, rcx, rdx):
        ql.arch.regs.rcx = rcx
        ql.arch.regs.rdx = rdx

        if not gCanary:
                ql.run(begin=0x01C000155C  , end=0x01C00015D1)
        else:
                ql.run(begin=0x01C00023dc  , end=0x01C0002442)

if __name__ == "__main__":
```

```
if gCanary:
        """
        decrypt1(ql, 0x1c009c810, 0x1c00a09b0)
        decrypt1(ql, 0x1c009d860, 0x1c00a0db0)
        decrypt1(ql, 0x1c009bd00, 0x1c00a0700)
        decrypt2(ql, 0x1c009cec0, 0x1c00a0b50)
        decrypt2(ql, 0x1c009c760, 0x1c00a0988)
        """

        decrypt1(ql, 0x1c009c990, 0x1c00a09b0)
        decrypt1(ql, 0x1c009d9e0, 0x1c00a0db0)
        decrypt1(ql, 0x1c009be80, 0x1c00a0700)

        decrypt2(ql, 0x1c009d040, 0x1c00a0b50)
        decrypt2(ql, 0x1c009c8e0, 0x1c00a0988)

        #dump_start = 0x1C00F5000
        #dump_end =   0x1C0109000
        dump_start = 0x01C0001000 # start of text
        dump_end =   0x01C00Fb000 # end of init
data = ql.mem.read(dump_start, dump_end - dump_start)

with open("dump_{:x}.bin".format(dump_start), "wb") as f:
        f.write(data)
```

TALOS

4. Patch the bytes

```
dump_start = 0x00001C0001000 # start of text
dump_end =   0x01C010F000 # end of init


with open("dump_{:x}.bin".format(dump_start), "rb") as f:
    data  = f.read()


ida_bytes.patch_bytes(dump_start, data)
```

CISCO TALOS

… wait for it …

```
; ---------------------------------------------------------------------------

loc_1C00F5008:                          ; CODE XREF: ClipSpInitialize+69↑p
                                        ; DATA XREF: .pdata:00000001C00AC350↑o
                in      al, 22h
                jno     short loc_1C00F5024
                push    rcx
                rol     bl, 1
                popfq
                cmp     edi, [rbx+rcx-5DDC2BCAh]
                pop     rsi
                insb
; ---------------------------------------------------------------------------
                db 0C7h, 0A4h, 0B1h, 0DAh, 46h, 2Bh, 1Fh
                db 0Eh, 38h, 0CFh, 3Ch
; ---------------------------------------------------------------------------

loc_1C00F5024:                          ; CODE XREF: PAGEwx1:00000001C00F500A↑j
                db      41h
                in      al, dx
                in      al, 50h
                db      36h
                xchg    eax, ebp
                call    near ptr 2140247BBh
                jbe     short loc_1C00F5076
                mov     word ptr ds:0FFFFFFFF86166427h[rax], es
; ---------------------------------------------------------------------------
                db 8Ch, 3Ah, 0CCh, 0BFh, 61h, 0EBh, 0C2h
                dq 0A89C25D356A4D0AEh, 3202A1955C704D98h, 0FBE6ED96481B2A55h
                dq 7C6F53AFE857111Eh, 4E6E8C53AF8E7CDAh, 835738AB949C224Bh
; ---------------------------------------------------------------------------
```

CISCO TALOS

5. Success

```
__int64 __fastcall ClipSpInitialize_(__unused __int64 a1, g_kernelCallbacks *a2, char a3)
{
  int v3; // edi
  __int64 v6; // rax
  g_kernelCallbacks *p
  __int128 v8; // xmm1

  v3 = 0;
  if ( !g_state.field_
  {
    v3 = CLIP_crypto_i
    if ( v3 < 0 )
      return v3;
    if ( (a3 & 1) != 0
      CLIP_delete_keys
    if ( (a3 & 2) == 0
      FltInitializePus
    CLIP_update_licens
    g_state.field_10 =
  }
  if ( a2 )
  {
    a2->ClipSpClepSign = ClipSpClepSign;
    a2->ClipSpClepKdf = ClipSpClepKdf;
    a2->ClipSpAcRequest = ClipSpAcRequest;
    a2->ClipSpAcHmac = ClipSpAcHmac;
    a2->ClipIsDeviceLicensePresent = ClipIsDeviceLicensePresent;
```

REVERSING TIME!

Reversing Plan

1. Understand how Windows communicate with the driver
2. Makes sense of data structures, logic, etc.
3. Keep an eye for potential vulnerabilities

Talking with the Driver

Part 1: the kernel

- Usually, we look for the creation of a device driver, search for IOCTL
  - None of that this time
- Instead, driver exports the ClipSpInitialize function
  - Takes one parameters that is used to store function pointers
  - Ntoskrnl.exe calls it and initialize a global array of callbacks:

```
__int64 ClipInitHandles()
{
  g_kernelCallbacks.ExUpdateLicenseData = ExUpdateLicenseData;
  g_kernelCallbacks.nt_SLQueryLicenseValue_ = (__int64)SLQueryLicenseValue_;
  g_kernelCallbacks.nt_ExUpdateOsPfnInRegistry = (__int64)ExUpdateOsPfnInRegistry;
  g_kernelCallbacks.field_138 = (__int64)SeExports;
  ClipSpInitialize(0i64, &g_kernelCallbacks);
  return wb_init_stuff();
}
```

Talking with the Driver

Part 1: the kernel

```
}
if ( a2 )
{
  a2->ClipSpClepSign = ClipSpClepSign;
  a2->ClipSpClepKdf = ClipSpClepKdf;
  a2->ClipSpAcRequest = ClipSpAcRequest;
  a2->ClipSpAcHmac = ClipSpAcHmac;
  a2->ClipIsDeviceLicensePresent = ClipIsDeviceLicensePresent;
  a2->ClipSpCreateLicenseEfsHeader = ClipSpCreateLicenseEfsHeader;
  a2->ClipCeate_some_EFS_key_blob_from_data_provided = kernel_call_207_create_some_EFS_key_blob_from_data_provided;
  a2->ClipSpLicenseEfsHeaderContainsFek = ClipSpLicenseEfsHeaderContainsFek;
  a2->CLIP_update_license_data_for_pfn__ = CLIP_update_license_data_for_pfn__;
  a2->ClipSpCheckLicense = ClipSpCheckLicense;
  a2->isPortableWithSerialNumber = isPortableWithSerialNumber;
  a2->ClipSpGetCurrentHardwareID = ClipSpGetHardwareBinding;
  a2->ClipSpQueryLicenseValueFromHost = ClipSpQueryLicenseValueFromHost;
  a2->ClipSpInsertTBActivationPolicyValue = ClipSpInsertTBActivationPolicyValue;
  a2->ClipSpGetPolicyValueFromCache = ClipSpGetPolicyValueFromCache must be type3;
```

- RE Trick: create a struct with the right size (51*sizeof(void*))
  - This way you can Xref where the function will be used, rename them, etc

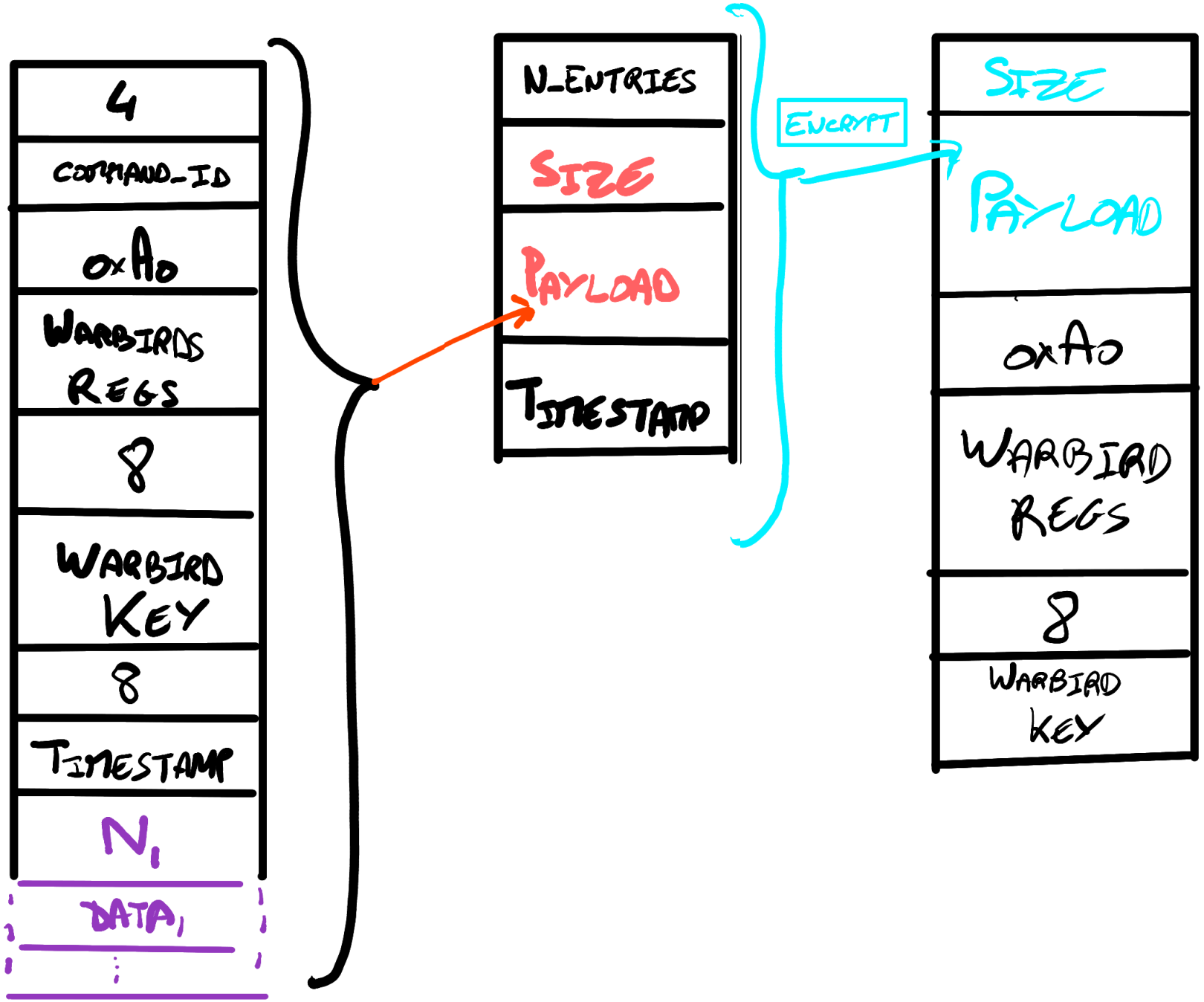
CISCO TALOS

Talking with the Driver

Part 2: Kernel-Userland interface

- Xref the callbacks to see how they are called
  - ExpQuerySystemInformation (SystemPolicyInformation)
    → ExHandleSPCall2 → ExHandleSPCall2-internal → ExHandleSPCall2Callout
    → Do-SPCall2
  - To talk with the driver, issue a NtQuerySystemInformation call with SystemPolicyInformation class
- command_id variable used to decide which callback function to call.
- ….but the whole payload is obfuscated

Diag



A big bird with missiles

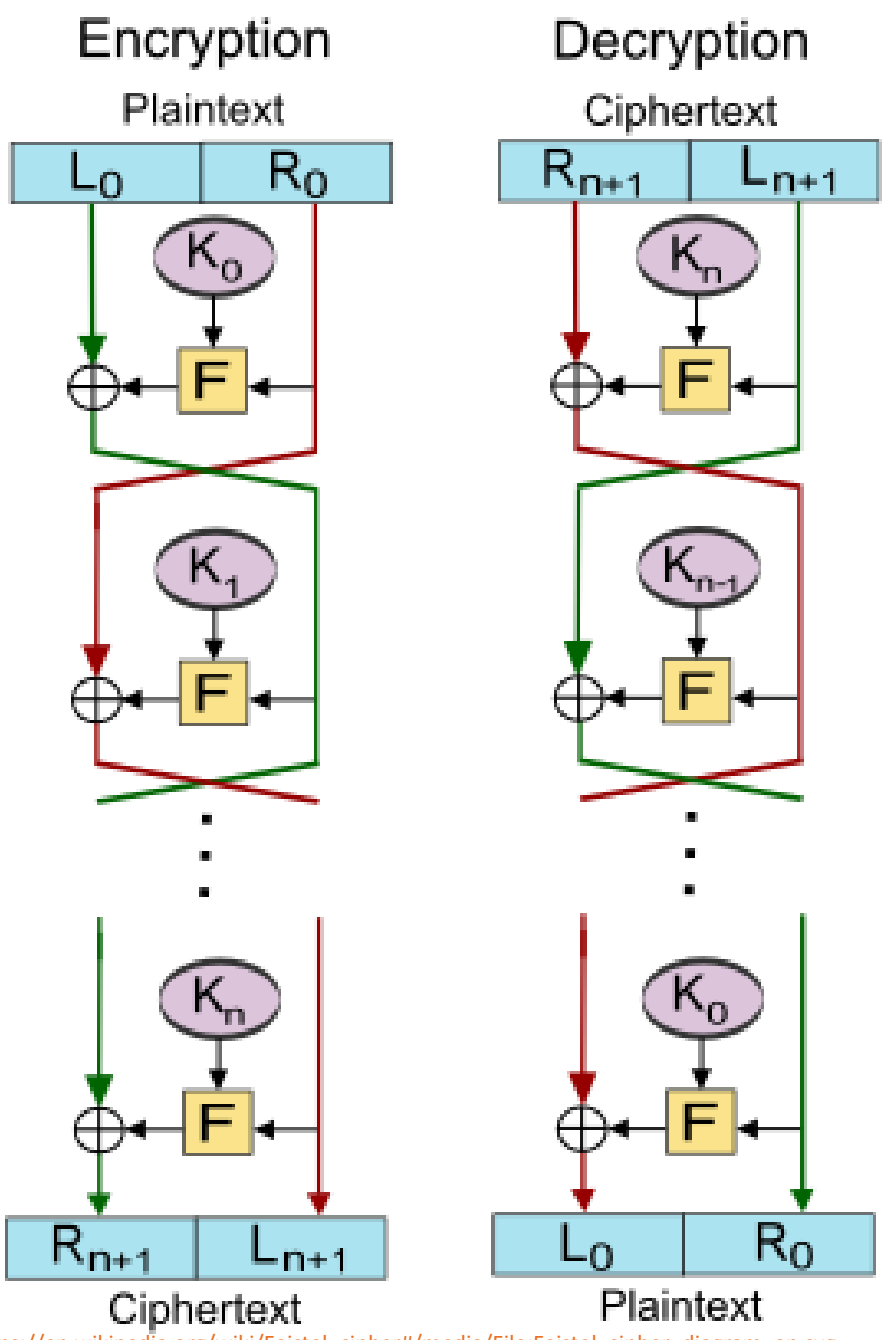
Image Creator in Bing | 1024 × 1024 jpg | 1 min ago

Content credentials
Generated with AI · August 22, 2024 at 8:40 PM

```
{
  v80 = v79[2] << 8;
  v81 = v79[3];
  v82 = (v79[1] | ((*v79 | ((*(v79 - 1) | (*(v79 - 2) << 8)) << 8)) << 8)) ^ v66;
  warbird_funcs_size_1 = v79[1] | ((*v79 | ((*(v79 - 1) | (*(v79 - 2) << 8)) << 8)) << 8);
  v83 = v79[5] | ((v79[4] | ((v81 | v80) << 8)) << 8);
  encrypted_size_without_xor_aligned = (size_t)(v79 + 8);
  warbird_key_size_1 = v83;
  v84 = (unsigned __int16)warbird_key_2;
  v85 = v82 ^ warbird_key_2 ^ HIDWORD(warbird_key_3) ^ v83 ^ v76;
  v86 = (v85 >> 8) ^ (WORD2(warbird_key_3) * (v85 ^ WORD1(warbird_key_3))) ^ v82;
  v87 = (WORD1(warbird_key_3) * __ROR4__(HIDWORD(warbird_key_3) - v86, 11) - __ROR4__(v86, 12)) ^ v85;
  v88 = ((unsigned __int16)warbird_key_2 * __ROL4__(v87 ^ HIDWORD(warbird_key_3), 8) - __ROL4__(v87, 2)) ^ v86;
  v89 = __ROR4__(v88, 9) ^ (HIWORD(warbird_key_3) * __ROR4__(v88 - warbird_key_2, 4)) ^ v87;
  v90 = (__ROR4__(v89, 4) + WORD2(warbird_key_3) * __ROR4__(warbird_key_2 - v89, 10)) ^ v88;
  v91 = (WORD1(warbird_key_3) * __ROL4__(HIWORD(warbird_key_3) ^ v90, 4) - __ROR4__(v90, 16)) ^ v89;
  v92 = 30i64;
  v93 = ((unsigned __int16)warbird_key_2 * (WORD1(warbird_key_3) ^ v91) - __ROR4__(v91, 7)) ^ v90;
  v94 = 16i64;
  v95 = (v93 - warbird_key_2 - HIWORD(warbird_key_3)) ^ v91;
  v359 = 16i64;
  v96 = __ROR4__(v95, 11) ^ (WORD2(warbird_key_3) * __ROR4__(warbird_key_2 - v95, 9)) ^ v93;
  v97 = (unsigned __int8 *)warbird_funcs_size_2;
  v98 = (WORD1(warbird_key_3) * (v96 - WORD2(warbird_key_3)) - (v96 >> 13)) ^ v95;
  v99 = (unsigned __int8 *)v363;
  warbird_funcs_1 = (PVOID)v363;
  v100 = (v98 >> 15) ^ (v84 * __ROL4__(v98 - WORD2(warbird_key_3), 3)) ^ v96;
  do
```

Feistel Rounds

- Same functions used for encryption/decryption
- Order is reversed for decryption
- Function Fn doesn't have to be invertible



https://en.wikipedia.org/wiki/Feistel_cipher#/media/File:Feistel_cipher_diagram_en.svg

WARBIRD REGS? WARBIRD KEY?

- Feistel Cipher
  - The 0xA0-sized buffer is a list of operation that were executed to encrypt the data
  - Tells the other side how to decrypt
  - The 8-byte key is fed to the Feistel functions
- The Warbird material in the inner payload is used by the kernel to encrypt the reply
- The Warbird material in the outer payload is used to decrypt

Where to get the algorithm / key material?

1. Find a binary that calls the clipsp API and rip it out!
   1. Grep for clip/license/system policy strings
   2. wlidsvc.dll! DeviceLicenseFunctions::SignHashWithDeviceKey
2. Use hex-ray to copy the encryption/decryption algorithm
3. Keep the same key/warbird material

Summary – Communication w/ Driver

1. Using NtQuerySystemInformation and SystemPolicyInformation class
2. Nested/obfuscated payloads with variable [length,data] fields
3. One of the field is a command_id, that defines the expected fields
4. Warbird material is passed around for encryption/decryption of payloads

So, what does CLIPSP do?

TL;DR;

- System Policy / License Value
  - NtQueryLicenseValue → ClipSpQueryLicenseValue
  - Tons of keys (e.g. notepad.exe: Security-SPP-GenuineLocalStatus)
- EFS
  - Encrypted Filesystem
  - Keys, headers, etc.
- Device attestation
  - Hardware binding
  - HMAC/Signature
- Windows Application Licenses (UWP)
  - Actual License
  - Cryptography material

How to find this info?

- ClipSp has no symbols, but other dlls/binaries may…
  - Ntoskrnl has tons of symbols, including functions calling ClipSP callbacks…
  - Grep for warbird constants to find other binaries calling into ClipSp / SystemPolicy
  - Time travel 🙃
    - Symbols were available ~1 year ago (c.f. keyhole blog)
- Many calls to logging give function / variable names away
  - Wrapper around EtwWriteTransfer
  - Last two variables are `n_entries` and an array of `EVENT_DATA_DESCRIPTOR`
  - Ptr to event description w/ function name and variables

CISCO TALOS

```
  v18 = 1000000 * (*&KeQueryPerformanceCounter(&Header) - PerformanceCounter.QuadPart);
  v19 = v18 / Header;
  if ( g_log_context > 5 && should_log(&g_log_context, 0x200000000000i64) )
  {
    v26 = v19;
    v30[2].Ptr = &v25;                          // Count
    v27 = v19;
    v30[3].Ptr = &v26;                          // qDurationTotal_ms
    v30[4].Ptr = &v27;                          // qDurationMin_ms
    v30[5].Ptr = &v28;                          // rDurationMax_ms
    v30[6].Ptr = &HeaderSize;                   // sKeyIdsSize
    v30[7].Ptr = &sKeyIdsSize_1;                // FekSize
    v30[8].Ptr = &v24;                          // LicenseEfsHeaderSize
    v30[9].Ptr = &Header;                       // Result
    v30[10].Ptr = &v29;                         // PartA_PrivTags
    v28 = v19;
    v25 = 1i64;
    *&v30[2].Size = 8i64;
    *&v30[3].Size = 8i64;
    *&v30[4].Size = 8i64;
    *&v30[5].Size = 8i64;
    HeaderSize = sKeyIdsSize;
    *&v30[6].Size = 4i64;
    sKeyIdsSize_1 = FekSize;
    *&v30[7].Size = 4i64;
    v24 = 0;
    *&v30[8].Size = 4i64;
    LODWORD(Header) = v10;
    *&v30[9].Size = 4i64;
    v29 = 0x1000000i64;
    *&v30[10].Size = 8i64;
    CLIP_log_stuff(&g_log_context, dword_1C0064E7D, v17, 0xBu, v30);
  }
  return v10;
}
```

UserData: struct _EVENT_DATA_DESCRIPTOR v30[11]; // [rsp+80h] [rbp-80h] BYREF

```
a:00000001C0064E7D ; int dword_1C0064E7D[47]
a:00000001C0064E7D dword_1C0064E7D dd 50Bh                ; DATA XREF: ClipSpCreateLicenseEfsHeader+1CA↓o
a:00000001C0064E81                 dd 0
a:00000001C0064E85                 dd 0B2000020h
a:00000001C0064E89                 dw 8000h
a:00000001C0064E8B                 db    0
a:00000001C0064E8C aClipspcreateli db 'ClipSpCreateLicenseEfsHeader',0
a:00000001C0064EA9 aCount_7        db 'Count',0
a:00000001C0064EAF                 dd 80808089h
a:00000001C0064EB3                 db  80h
a:00000001C0064EB4 aQdurationtotal_6 db 'qDurationTotal_ms',0
a:00000001C0064EC6                 dd 80808089h
a:00000001C0064ECA                 db  80h
a:00000001C0064ECB aQdurationminMs_6 db 'qDurationMin_ms',0
a:00000001C0064EDB                 dd 80808089h
a:00000001C0064EDF                 db  80h
a:00000001C0064EE0 aRdurationmaxMs_6 db 'rDurationMax_ms',0
a:00000001C0064EF0                 dd 80808089h
a:00000001C0064EF4                 db  80h
a:00000001C0064EF5 aSkeyidssize    db 'sKeyIdsSize',0
a:00000001C0064F01                 db    8
a:00000001C0064F02 aFeksize_1      db 'FekSize',0
a:00000001C0064F0A                 db    8
a:00000001C0064F0B aLicenseefshead_0 db 'LicenseEfsHeaderSize',0
a:00000001C0064F20                 db    8
a:00000001C0064F21 aResult_18      db 'Result',0
a:00000001C0064F28                 db  88h
a:00000001C0064F29                 db  0Eh
a:00000001C0064F2A aPartaPrivtags_19 db 'PartA_PrivTags',0
a:00000001C0064F39                 db  0Ah
a:00000001C0064F3A                 db    6
a:00000001C0064F3B ; int dword_1C0064F3B[27]
```

Function Name

Variables

See Logs

```
// See https://github.com/Biswa96/TraceEvent/tree/master

//Enable logging, first command needs Admin privileges

C:\re\tools\TraceEvent.exe -S yolo -g {b4b126de-32fe-4591-9ac5-b0778d79a0e7}
C:\re\tools\TraceEvent.exe -l yolo


// GUID provided when registering ETW context via EtwRegister
```

We RE for a while…

CISCO TALOS

```
00000000 g_kernelCallbacks struc ; (sizeof=0x198, ma
00000000                                         ; X
00000000                                         ; C
00000000 field_0         dd ?                    ; X
00000004 field_4         dd ?
00000008 ClipSpIsWinPortableWithSerialNumber dq ?
00000008                                         ; X
00000008                                         ; C
00000010 probably_collect_hardware_info_for_bindings
00000010                                         ; X
00000018 ClipSpGetLicenseChallenge dq ?          ; X
00000020 ClipSpRemoveLicense dq ?                ; X
00000028 ClipSpUpdateLicense dq ?                ; X
00000030 CLIP_delete_keys dq ?                   ; X
00000038 ClipSpDoNothing dq ?                    ; X
00000040 ClipSpFree      dq ?                    ; X
00000048 ClipSpDecryptFek dq ?                   ; X
00000050 ClipSpDecryptFekEx dq ?                 ; X
00000058 ClipSpCreateDirectoryLicenseHeader dq ? ; X
00000060 ClipSpCreateFileLicenseHeaderAndKey dq ?
00000060                                         ; X
00000068 ClipSpQueryLicenseStatusForApp dq ?     ; X
00000068                                         ; Q
00000070 ClipSpGetBaseContentKeyFromLicense dq ? ; X
00000078 ClipSpGetAppPolicyValue dq ?            ; X
00000078                                         ; C
00000080 ClipSpGetLicenseExpiryInfo dq ?         ; X
00000088 ClipSpGetBaseContentKeyFromKeyID dq ?   ; X
00000090 ClipSpDebugInfo_not_impl dq ?           ; X
00000098 ClipSpClepSign  dq ?                    ; X
000000A0 ClipSpClepKdf   dq ?                    ; X
000000A8 ClipIsDeviceLicensePresent dq ?         ; X
000000A8                                         ; C
```

```
000000A8                                         ; Clip
000000B0 SpCanAppLaunch  dq ?                    ; XREF
000000B8 ClipSpCreateLicenseEfsHeader dq ?       ; XREF
000000C0 ClipCeate_some_EFS_key_blob_from_data_provided
000000C0                                         ; XREF
000000C8 ClipSpLicenseEfsHeaderContainsFek dq ?  ; XREF
000000D0 CLIP_update_license_data_for_pfn__ dq ? ; XREF
000000D8 ClipSpCheckLicense dq ?                 ; XREF
000000E0 isPortableWithSerialNumber dq ?         ; XREF
000000E8 ClipSpGetCurrentHardwareID dq ?         ; XREF
000000F0 ClipSpQueryLicenseValueFromHost dq ?    ; XREF
000000F0                                         ; Quer
000000F8 ClipSpInsertTBActivationPolicyValue dq ?
000000F8                                         ; XREF
000000F8                                         ; Clip
00000100 ClipSpGetPolicyValueFromCache dq ?      ; XREF
00000100                                         ; sub_
00000108 ClipSpUpdateImdsResponse dq ?           ; XREF
00000110 ClipSpAcRequest dq ?
00000118 ClipSpAcHmac    dq ?
00000120 nt_ExUpdateLicenseData dq ?
00000128 nt_SLQueryLicenseValue_ dq ?
00000130 nt_ExUpdateOsPfnInRegistry dq ?
00000138 nt_SeExports    dq ?
00000140 field_140       dq ?
00000148 field_148       dq ?
00000150 field_150       dq ?
00000158 field_158       dq ?
00000160 ExpTimeRefresh  dq ?                    ; XREF
00000168 field_168       dq ?                    ; XREF
00000170 ClipSpQueryLicenseValue dq ?            ; XREF
00000178 ExUpdateOsPfnInRegistry dq ?            ; XREF
00000180 field_180       dq ?                    ; XREF
00000188 ExSetLicenseTamperState dq ?            ; XREF
00000190 ExUpdateLicenseData dq ?                ; XREF
00000198 g_kernelCallbacks ends
```

SystemPolicyInformation – command_id

```
0 - SPQueryLicenseValue
1 - SPUpdatePolicies
2 - SPAuthenticateCaller
4 - ???
5 - SPWaitForDisplayWindow
6 - ???
7 - ???
22 - SPFileUsnQuery
23 - SPFileIntegrityUpdate
24 - SPFileIntegrityQuery
```

```
100 - SPUpdateLicense
101 - SPRemoveLicense
103 - SPCreateEFSHeader
104 - SPEfsHeaderContainsFek
105 - SPGetLicenseChallenge
106 - SPClipSpGetBaseContentKeyFromLicense
107 - SPClipSpGetBaseContentKeyFromKeyID
109 - SPIsAppLicensed
110 - SPDebugLicense (not implemented?)
111 - SPDeleteKeys
112 - SPClepSign
113 - SPClepKdf
```

```
204 - Update license data for PFN
205 - SPCheckLicense
206 - SPGetCurrentHardwareID
207 - ?? EFS Create key blobs ???
208 - SPGetAppPolicyValue
209 - ??? Get time info for keyholder ???
210 - SPAcRequest
211 - SPAcHmac
212 - ? some IDMSLicensingIntegration check ?
```

CISCO TALOS

License blob

License Blob

What does it do?

- Multiple usages:
  - Device configuration / device ID
  - Application license for app store
    - Key material
    - Lease
    - …
- Automatically installed behind the scene
  - On UWP app start if license is expired and connected to the internet
- Error messages:
  - License is expired
  - Invalid device id
  - etc.

CISCO TALOS

License Blob

How does it work?

- Installed via command_id 100 (SpUpdateLicense)
- Serie of TLV (Tag, Length, Value) entries
- Not documented, XML mapping gives some variable names
- Map: Tag → Internal Index
  - Most of the data is loaded into an array
- Signed
  - License Type → different signing authorities
  - Hardcoded public keys

```
type 0  is the whole blob minus the hash I think
type 1  probably expiration date
type 2  maybe start date
type 3  expiration for device bound license
type 4  clep blob used for signing, supposedly device signing key (material?)
type 5  DeviceID the license is bound to
type 7  is what becomes the key name in regs[0]  (maybe hash/uniqueID of something)
type 8  LicenseHardwareBinding
type 10 id of sub-license probably for content key
type 14 pfn
type 15 license info (type, usage, ...)
type 16 llv buffer of encrypted key material  (likely stored in {D73E01AC-F5A0-4D80-928B-33C1920C38BA})
type 18 device id
type 21 Public key (RSA1 or ECC) (see 0x001C00FD35B )
type 22 keyslot stuff: n_entries;  (type, data) ... (type, data) -> type1 is wstr, type2 is a depth or something
type 23 keyholderID probably list of keyholder license (point to the next and next and ...)
type 24 signature type and hash
```

```
License_type_conversion <1, 1Fh>
                        ; DATA XREF:
                        ; license_con
License_type_conversion <2, 0D3h> ;
License_type_conversion <3, 20h> ;
License_type_conversion <4, 12Dh> ;
License_type_conversion <5, 0D2h> ;
License_type_conversion <6, 0D1h> ;
License_type_conversion <7, 0CBh> ;
License_type_conversion <8, 0D0h> ;
License_type_conversion <14, 0CEh> ;
License_type_conversion <15, 0C9h> ;
License_type_conversion <16, 0CAh> ;
License_type_conversion <17, 1> ;
License_type_conversion <18, 2> ;
License_type_conversion <17, 1> ;
License_type_conversion <18, 2> ;
License_type_conversion <19, 0CDh> ;
License_type_conversion <0, 14h>
License_type_conversion <10, 18h>
License_type_conversion <11, 0>
License_type_conversion <24, 0CCh>
License_type_conversion <9, 0CFh>
License_type_conversion <12, 12Eh>
License_type_conversion <13, 0D5h>
License_type_conversion <20, 0D4h>
License_type_conversion <21, 0DCh>
License_type_conversion <22, 0DDh>
License_type_conversion <23, 0DEh>
```

CISCO TALOS

License blob

Stored in registry

- `HKLN\SYSTEM\CurrentControlSet\Control\{7746D80F-97E0-4E26-9543-26B41FC22F79}`
  - Key used by Clipsp to store data
  - Only accessible to SYSTEM
  - PsExec64.exe -s -i regedit

    → Used for running regedit as SYSTEM

License blob

Stored in registry

- Subkeys:
  - `{A25AE4F2-1B96-4CED-8007-AA30E9B1A218}` → License data
  - `{D73E01AC-F5A0-4D80-928B-33C1920C38BA}` → ContentKey ?
  - `{59AEE675-B203-4D61-9A1F-04518A20F359}` → AppPolicy1 ?
  - `{FB9F5B62-B48B-45F5-8586-E514958C92E2}` → AppPolicy2 ?
  - `{221601AB-48C7-4970-B0EC-96E66F578407}` → Key timeinfo (expiration/start date)

Vulnerabilities

Vulnerability 1

Parsing the license…..

CISCO TALOS

```
  while ( 1 )
  {
    cur_tlv_entry = &LicenseData[cur_offset];
    entry_size = cur_tlv_entry->entry_size;
    if ( entry_size >= 0xFFFFF || cur_offset + entry_size > LicenseDataSize )
      return STATUS_INVALID_PARAMETER;
    status = license_validate_entry(&LicenseData[cur_offset], LicenseDataSize - cur_offset);
    if ( (status & 0x80000000) != 0 )         // bad
      return status;
    idx = license_convert_type(cur_tlv_entry->type);// 0xce -> type 14 -> pfn
    if ( idx < 25 )
    {
      idx_ = idx;
      License->entries[idx_].entry_size = cur_tlv_entry->entry_size;
      License->entries[idx_].entry_field2 = cur_tlv_entry->field_2;
      License->entries[idx_].entry_ptr = &cur_tlv_entry->first_byte;
      if ( idx == 24 )                        // type 0xCC aka signature
        break;
    }
    cur_offset += 8;
    if ( idx )                                // case 0 (aka 0x14 in license file) is the whole blob minus the signature blob
      goto NEXT_ENTRY;
CONTINUE_IF_MORE_DATA:
    if ( cur_offset >= LicenseDataSize - 8 )
      return status;                          // no more data
  }
  License->LicenseData_ = LicenseData;
  License->entry_of_type_24 = cur_tlv_entry;  // store pointer to signature
  ExFreePoolWithTag_noTag(0i64);
  status = 0;
  cur_offset += 8;
NEXT_ENTRY:
  cur_offset += cur_tlv_entry->entry_size;
  goto CONTINUE_IF_MORE_DATA;
}
```

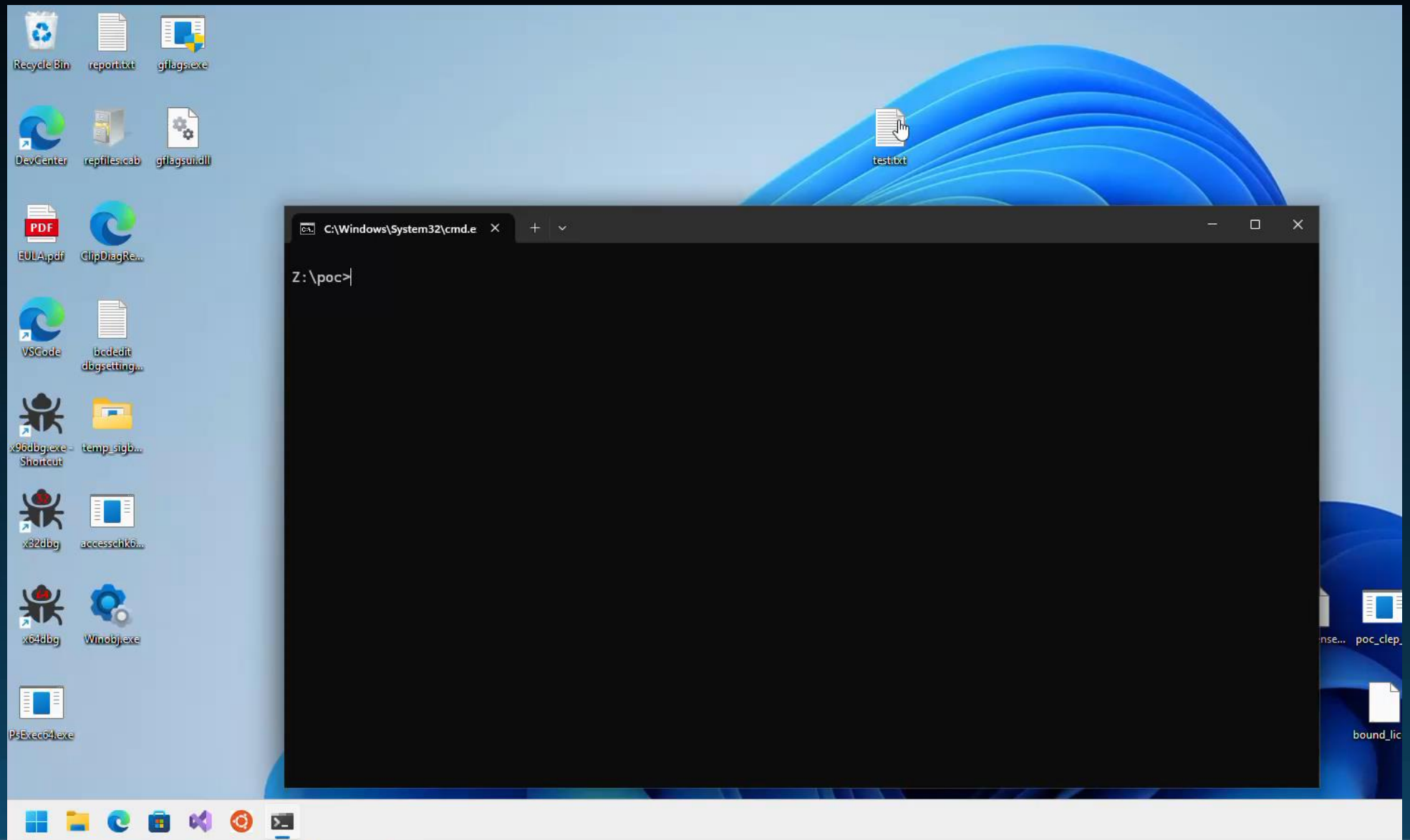
```
CHECK_SIG:
      sig_len = LicenseStruct_get_some_size_for_type24(License);
      sig_data = LicenseStruct_get_some_ptr_for_type24(License);
      status_ = License_validate_signature(
                  context->alg_SHA256,
                  hKey,
                  bcryptFlags,
                  License->LicenseData_,
                  LODWORD(License->entry_of_type_24) - LODWORD(License->LicenseData_),// length
                  &deviceId_details,
                  UNUSED_ARG(),
                  sig_data,
                  sig_len);
      status = status_;
      if ( status_ >= 0 )
      {
        License->license_status_bitmask &= ~CLIP_LICENSE_INVALID_SIGNATURE;
        goto DONE;
      }
      if ( status_ != STATUS_INVALID_SIGNATURE )
        goto DONE;
INVALID_SIGNATURE:
      License->license_status_bitmask |= CLIP_LICENSE_INVALID_SIGNATURE;
      status = 0;
      goto DONE;
```

Yep! Signature bypass

Signature Bypass

- Two problems:
  - Can append data after the signature, license blob still valid
  - If a tag was already loaded, it can get overwritten if the same tag appears again
- Consequences:
  - Tamper with license content
    - Legitimate license can be dumped from the registry,
  - Change expiration date, encryption keys, hardware bindings, …
  - Change device id
  - Increased attack surface 😈

Expiring notepad...



Vulnerability 2

Let’s have a look at the device id….

CISCO TALOS

During Device License Install…

```
if ( (License->license_status_bitmask & CLIP_LICENSE_HARDWARE_ID_OUT_OF_TOLERANCE) == 0 )
{
  version_for_type17_via_deref_clepstuff = License_get_version_for_type17_via_deref_clepstuff(License);
  ptr_for_clep_type17 = License_get_ptr_for_clep_type17(License);
  status = ReleaseKeyFromEncState_internal(UNUSED_ARG(), ptr_for_clep_type17, version_for_type17_via_deref_clepstuff, &hKey);
  if ( status < 0 )
    goto DONE;
  deviceIdSize_ = License_get_DeviceIDSize(License);
  status = CLIP_allocate_buffer(deviceIdSize_, &deviceIdBuffer);
  if ( status < 0 )
  {
    v3 = deviceIdBuffer;
    goto DONE;
  }
  deviceIdSize_2 = License_get_DeviceIDSize(License);
  deviceIdSize_3 = License_get_DeviceIDSize(License);
  deviceId_ptr = License_get_DeviceID(License);
  v3 = deviceIdBuffer;
  some_memcpy_probz(deviceIdBuffer, deviceId_ptr, deviceIdSize_3);
}
entrysize_for_type4 = LicenseStruct_get_entrysize_for_type4(License);
```

CISCO TALOS

```
__int64 __fastcall __spoils<rax,rcx> License_get_DeviceIDSize(LicenseStruct *license)
{
  __int64 result; // rax
  unsigned __int16 *entry_ptr; // rcx

  result = license->entries[5].entry_size;
  if ( !license->entries[5].entry_size )
  {
    entry_ptr = license->entries[18].entry_ptr;
    if ( entry_ptr )
      return *entry_ptr;                        // read first word
  }
  return result;
}
```

```
__int64 __fastcall __spoils<rax,rcx> License_get_DeviceID(LicenseStruct *a1)
{
  __int64 result; // rax
  __int64 entry_ptr; // rcx

  result = a1->entries[5].entry_ptr;
  if ( !result )
  {
    entry_ptr = a1->entries[18].entry_ptr;
    if ( entry_ptr )
      return entry_ptr + 2;                     // returns buffer after the 2-bytes size field
  }
  return result;
}
```

OOB-Read!

OOB-Read in Device ID Field

- Two OOB-reads:
  - Size field can be read out of bound
    - If the data provided for entry[18] is less than 2 bytes.
  - If not, the deviceId data can be read out of bound as well
    - If size provided is larger than length of entry[18] buffer

But wait. There’s more!

```
if ( (License->license_status_bitmask & CLIP_LICENSE_HARDWARE_ID_OUT_OF_TOLERANCE) == 0 )
{
  version_for_type17_via_deref_clepstuff = License_get_version_for_type17_via_deref_clepstuff(License);
  ptr_for_clep_type17 = License_get_ptr_for_clep_type17(License);
  status = ReleaseKeyFromEncState_internal(UNUSED_ARG(), ptr_for_clep_type17, version_for_type17_via_deref_clepstuff, &hKey);
  if ( status < 0 )
    goto DONE;
  deviceIdSize_ = License_get_DeviceIDSize(License);
  status = CLIP_allocate_buffer(deviceIdSize_, &deviceIdBuffer);
  if ( status < 0 )
  {
    v3 = deviceIdBuffer;
    goto DONE;
  }
  deviceIdSize_2 = License_get_DeviceIDSize(License);
  deviceIdSize_3 = License_get_DeviceIDSize(License);
  deviceId_ptr = License_get_DeviceID(License);
  v3 = deviceIdBuffer;
  some_memcpy_probz(deviceIdBuffer, deviceId_ptr, deviceIdSize_3);
}
entrysize_for_type4 = LicenseStruct_get_entrysize_for_type4(License);
```

CISCO TALOS

Double Fetch!

OOB-Write in Device ID Field

- The size of DeviceId is read multiple times
  - One for malloc
  - One for memcpy
- Race condition during the double fetch
  - Need to read size field OOB
  - Swap value between reads
  - On success, memcpy could lead to heap overflow (allocated_size < copied_size)

CISCO TALOS

Exploitation strategy

1. Create a license file with junk data so it's large and page-aligned
2. Add a DeviceId field at the end of the license blob
    1. Make the size field of DeviceId is OOB
3. Perform heap feng-shui so an object we control lands right after the license
    1. Can use CreatePrivateNamespace and CreateBoundaryDescriptor
    2. Good edge case: Alloc size % 0x1000 is bigger than 0xFE0
4. Spam Create/Destroy Namespace while installing license
    1. Bad luck: BSOD
    2. Neutral: nothing
    3. Good: OOB Write

Debug in Windbg

```
sxi sse

// log size before calling malloc
ba e 1 clipsp+0x0FEC78 "r $t1 = @ecx; .echo ;  g;“


// log size before calling memcpy
ba e 1 clipsp+0xfecaf  "r $t2 = @r8d; .echo -------; .if ($t1 < $t2) {.echo ------;
.echo wooooooooooot!!!!!; .printf \"alloc size: %hx, ptr:%p\",$t1, poi(rdi+0x128);
.echo ;  .printf \"memcpy size: %hx\", $t2;  .echo -------;  } .else { g; };"
```

File Home View Break... Time... Model Scripti... Source Mem... Exten... Com...

Break Go Step Out Step Into Step Over Step Out Back Step Into Back Step Over Back Go Back Restart Stop Debugging Detach Settings Source Assembly Local Help Feedback

Flow Control | Reverse Flow Control | End | Preferences | Help

Disassembly

Address: @$scopeip Follow current instruction

```
fffff803`48224608  nop     dword ptr [rax+rax]
    nt!DbgUserBreakPoint:
fffff803`48224610  int     3
fffff803`48224611  ret
fffff803`48224612  int     3
fffff803`48224613  int     3
fffff803`48224614  int     3
fffff803`48224615  int     3
fffff803`48224616  int     3
fffff803`48224617  int     3
fffff803`48224618  nop     dword ptr [rax+rax]
    nt!DbgBreakPointWithStatus:
fffff803`48224620  int     3
fffff803`48224621  ret
    nt!DbgBreakPointWithStatusEnd:
fffff803`48224622  int     3
fffff803`48224623  int     3
fffff803`48224624  int     3
fffff803`48224625  int     3
fffff803`48224626  int     3
fffff803`48224627  int     3
fffff803`48224628  nop     dword ptr [rax+rax]
```

Command

```
* If you did not intend to break into the debugger, press the "g" key, then   *
* press the "Enter" key now.  This message might immediately reappear.  If it *
* does, press "g" and "Enter" again.                                          *
*                                                                             *
*******************************************************************************
nt!DbgBreakPointWithStatus:
fffff803`48224620 cc              int     3
0: kd> sxi sse
0: kd> ba e 1 clipsp+0x0FEC78 "r $t1 = @ecx; .echo ;  g;"
0: kd> ba e 1 clipsp+0xfecaf  "r $t2 = @r8d; .echo -------; .if ($t1 < $t2) {.echo ------; .echo wooooooooooo
0: kd> g
Break instruction exception - code 80000003 (first chance)
*******************************************************************************
```

Debuggee is running...

File Edit View VM Tabs Help

Linux-fun | win-11-eval

Recycle Bin, report.txt, gflags.exe, DevCenter, repfiles.cab, gflagsui.dll, test.txt, EULA.pdf, ClipDiagRe..., VSCode, bcdedit dbgsettings ..., ClipSp_ntqu... - Shortcut, x96dbg.exe - Shortcut, temp_sigby..., x32dbg, accesschk64..., poc_oob_w..., go.bat, x64dbg, Winobj.exe, poc_license..., PsExec64.exe, poc_misc_..., bound_lice..., poc_clep_o...

Start

Windows 11 Enterprise Evaluation
Windows License is expired
Build 22621.ni_release.220506-1250

CISCO TALOS

File Home View Break... Time... Model Scripti... Source Mem... Exten... Com...

Break Go Step Out Step Into Step Over Step Out Back Step Into Back Step Over Back Go Back Restart Stop Debugging Detach Settings Source Assembly Local Help Feedback

Flow Control | Reverse Flow Control | End | Preferences | Help

Disassembly

Address: @$scopeip Follow current instruction

```
fffff803`48224608  nop     dword ptr [rax+rax]
    nt!DbgUserBreakPoint:
fffff803`48224610  int     3
fffff803`48224611  ret
fffff803`48224612  int     3
fffff803`48224613  int     3
fffff803`48224614  int     3
fffff803`48224615  int     3
fffff803`48224616  int     3
fffff803`48224617  int     3
fffff803`48224618  nop     dword ptr [rax+rax]
    nt!DbgBreakPointWithStatus:
fffff803`48224620  int     3
fffff803`48224621  ret
    nt!DbgBreakPointWithStatusEnd:
fffff803`48224622  int     3
fffff803`48224623  int     3
fffff803`48224624  int     3
fffff803`48224625  int     3
fffff803`48224626  int     3
fffff803`48224627  int     3
fffff803`48224628  nop     dword ptr [rax+rax]
```

Command

```
* If you did not intend to break into the debugger, press the "g" key, then   *
* press the "Enter" key now.  This message might immediately reappear.  If it *
* does, press "g" and "Enter" again.                                          *
*                                                                             *
*******************************************************************************
nt!DbgBreakPointWithStatus:
fffff803`48224620 cc              int     3
0: kd> sxi sse
0: kd> ba e 1 clipsp+0x0FEC78 "r $t1 = @ecx; .echo ;  g;"
0: kd> ba e 1 clipsp+0xfecaf  "r $t2 = @r8d; .echo -------; .if ($t1 < $t2) {.echo ------; .echo wooooooooooo
0: kd> g
Break instruction exception - code 80000003 (first chance)
*******************************************************************************
```

Debuggee is running...

File Edit View VM Tabs Help

Linux-fun | win-11-eval

Recycle Bin, report.txt, gflags.exe, DevCenter, repfiles.cab, gflagsui.dll, test.txt, EULA.pdf, ClipDiagRe..., VSCode, bcdedit dbgsettings ..., ClipSp_ntqu... - Shortcut, x96dbg.exe - Shortcut, temp_sigby..., x32dbg, accesschk64..., poc_oob_w..., go.bat, x64dbg, Winobj.exe, poc_license..., PsExec64.exe, poc_misc_..., bound_lice..., poc_clep_o...

Start

Windows 11 Enterprise Evaluation
Windows License is expired
Build 22621.ni_release.220506-1250

CISCO TALOS

No exploit, why :'(?

Challenges in exploitation

- PagedPool vs NonePaged p
  - Not as much info?
- We need to "control" the fir ereafter
  - Lots of APIs will add hea
    - PipeAttributes and WN
  - Copy of string tend to b
    - Might be enough if d
- We need to race the malloc
  - NtCreateTransactionMa
- On the plus side, MS doesn
  - Kernel Access violation

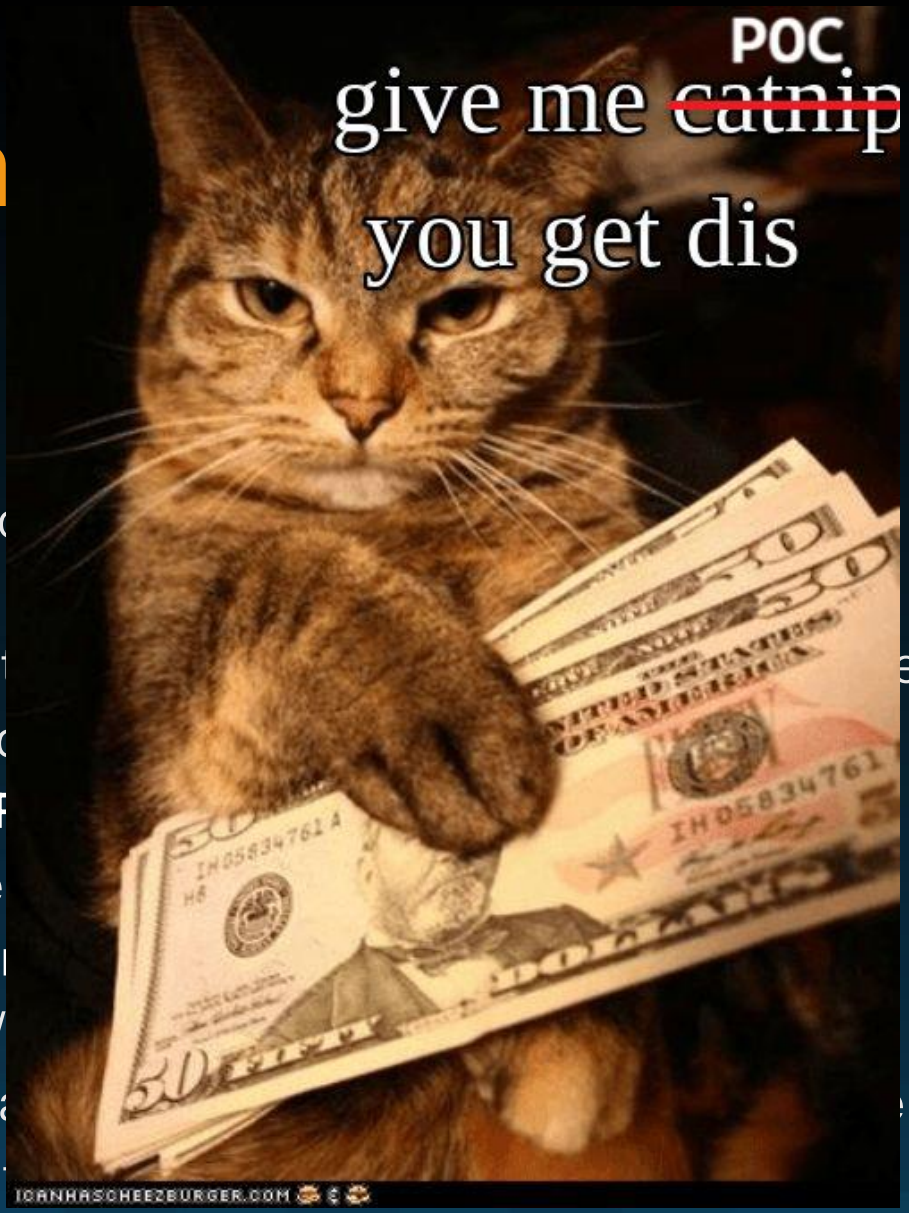
No exploit, why :'(?

Challenges in exploitation

- PagedPool vs NonePaged pool
  - Not as much info?
- We need to "control" the first 1-2 bytes of the data, and most of the content thereafter
  - Lots of APIs will add header to the data
    - PipeAttributes and `WNF_STATE_DATA` won't do
  - Copy of string tend to be transient
    - Might be enough if done right?
- We need to race the malloc/memcpy
  - NtCreateTransactionManager is probably too slow as a primitive ☹
- On the plus side, MS doesn't care and pays bounties…
  - Kernel Access violation is enough (https://aka.ms/windowsbugbar)

No exploit, wh

Challenges in exploitation



- PagedPool vs NonePaged p
  - Not as much info?
- We need to “control” the firs e content thereafter
  - Lots of APIs will add hea
    - PipeAttributes and WNF
  - Copy of string tend to be
    - Might be enough if do
- We need to race the malloc/
  - NtCreateTransactionMana ☹
- On the plus side, MS doesn’
  - Kernel Access violation is enough (https://aka.ms/windowsbugbar)

CISCO TALOS

More bugs…

Findings!

- Reporting:
  - Everything has been reported to Microsoft over 90 days ago
  - Fixed in July
  - Advisories available on Talos blog
    - `https://talosintelligence.com/vulnerability_reports`
- CVEs:
  - CVE-2024-38062
  - CVE-2024-38184
  - CVE-2024-38185
  - CVE-2024-38186
  - CVE-2024-38187

CISCO TALOS

Next Steps

What can be done with this?

- Regular EoP, but also Sandbox escape from LPAC container
  - LPAC = Less Privileged Application Container
  - Used to isolate browser content pages, parsers, …
- Tampering with License files
  - Changing device id
  - Changing expiration dates for licenses
  - etc.
- Self-modifying kernel code is probably a bad idea…
  - See https://downwithup.github.io/blog/post/2023/04/23/post9.html

Remaining attack surface

- EFS (Encrypted File System)
- Other System Policy objects
- Azure Integration ?
  - Feature flag in the ntoskrnl, maybe sign of active development

Going beyond ClipSP

- Other Warbird Obfuscated Dlls:
  - Ci, peauth, …
  - PlayReady
- Other surprising attack surface via NtQuerySystemInformation
  - Ex: SystemControlFlowTransition (syscall for Warbird)
    - CVE-2024-20698

CONCLUSION

Conclusion

- Obfuscation can hide trivial bugs
  - Makes it an interesting attack surface to look at from offensive perspective
  - Tradeoff with how difficult it is to deobfuscate
- Elevation of privileges are a key aspect of modern exploitation

THANKS!

Questions/comments:

@phLaul

Q&A

blog.talosintelligence.com

@talossecurity

TALOSINTELLIGENCE.COM

thank you!

blog.talosintelligence.com

@talossecurity

TALOSINTELLIGENCE.COM

CISCO

TALOS

TALOSINTELLIGENCE.COM