CET Updates – CET on Xanax

windows-internals.com/cet-updates-cet-on-xanax

By Yarden Shafir

Windows 21H1 CET Improvements

Since Alex and I first published our first analysis of CET, Windows' support for user-mode CET received a few important changes that should be noted. We can easily spot most of them by looking at the changes to the `MitigationFlags2` field of the `EPROCESS`, when comparing Windows `10` Build `19013` with `20226`:

```
    /* 0x09d4 */ unsigned long CetUserShadowStacksStrictMode : 1; /* bit position: 20 */
    /* 0x09d4 */ unsigned long BlockNonCetBinaries : 1; /* bit position: 21 */
    /* 0x09d4 */ unsigned long BlockNonCetBinariesNonEhcont : 1; /* bit position: 22 */
    /* 0x09d4 */ unsigned long AuditBlockNonCetBinaries : 1; /* bit position: 23 */
    /* 0x09d4 */ unsigned long AuditBlockNonCetBinariesLogged : 1; /* bit position: 24 */
    /* 0x09d4 */ unsigned long XtendedControlFlowGuard : 1; /* bit position: 25 */
    /* 0x09d4 */ unsigned long AuditXtendedControlFlowGuard : 1; /* bit position: 26 */
    /* 0x09d4 */ unsigned long PointerAuthUserIp : 1; /* bit position: 27 */
    /* 0x09d4 */ unsigned long AuditPointerAuthUserIp : 1; /* bit position: 28 */
    /* 0x09d4 */ unsigned long AuditPointerAuthUserIpLogged : 1; /* bit position: 29 */
    /* 0x09d4 */ unsigned long CetDynamicApisOutOfProcOnly : 1; /* bit position: 30 */
    /* 0x09d4 */ unsigned long UserCetSetContextIpValidationRelaxedMode : 1; /* bit position: 31 */
```

There are a lot of new mitigation flags here, and a few of them are related to CET:

- `CetUserShadowStackStrictMode` – annoyingly, this does not mean the same thing as Strict CFG. Strict CET means that CET will be enforced for the process, regardless of whether it's compiled as CET compatible or not.
- `BlockNonCetBinaries` – as the name suggests, this feature blocks binaries that were not compiled with CET support from being loaded into the process — just like Strict CFG.
- `CetDynamicApisOutOfProcOnly` – At first CET was supposed to block all non-approved RIP changes. That was too much, so it was toned down to only block most non-approved RIP targets. Then MS remembered dynamic memory, and couldn't force dynamic memory to comply with CET but insisted that allowing dynamic targets was only supported out of proc, so not really a security risk. And now it seems that in proc dynamic APIs are allowed by default and processes have to manually opt-out of that by setting this flag. In their defense, the flag is already set for most important Windows processes such as `winlogon.exe`, `lsass.exe`, `csrss.exe` and `svchost.exe`. But I'm sure that's OK and we'll never see CET bypasses abusing dynamic APIs in proc.
- `UserCetSetContextIpValidationRelaxedMode` – Even after all the adjustments that were made in order to not break any existing code, CET was still a bit too anxious, resulting in this new mitigation. This new flag has a pretty curious name that might draw your attention. If it did – good! Because this is the CET feature that this blog post will focus on.

But even without knowing the purpose of any of those, the amount of new CET flags alone hints that this we are not expected to see CET being fully enforced across the system any time soon.

Relaxed Mode

The least obvious of those new flags in the "relaxed mode" option. Was CET too anxious to handle 2020 and needed a bit of a break from everything? Well if it did, I think we can all relate to that and shouldn't judge to harshly.

This flag can be set on process creation, by calling `UpdateProcThreadAttribute` with `PROC_THREAD_ATTRIBUTE_MITIGATION_POLICY` and `PROCESS_CREATION_MITIGATION_POLICY2_USER_CET_SET_CONTEXT_IP_VALIDATION_RELAXED_MODE` as the mitigation policy flag.

It can also be set with a currently-undocumented linker flag, which will set the new `IMAGE_DLLCHARACTERISTICS_EX_CET_SET_CONTEXT_IP_VALIDATION_RELAXED_MODE` value in the PE header information (see the end of the post for the definition).

Once the flag is set, it is only used in two places – `KeVerifyContextIpForUserCet` and `KiContinuePreviousModeUser`. Both read it from the `EPROCESS` and pass a Boolean value into `KiVerifyContextIpForUserCet` to indicate whether it's enabled or not. Inside `KiVerifyContextIpForUserCet` we can see this new addition that checks this argument:

```
RtlZeroMemory(&unwindState, sizeof(unwindState));
if (continueType == KCONTINUE_UNWIND)
{
    status = RtlVerifyUserUnwindTarget(userRip, KCONTINUE_UNWIND, &unwindState);
    if (NT_SUCCESS(status))
    {
        return status;
    }

if ((RelaxedMode != FALSE) && (continueType != KCONTINUE_RESUME))
{
    if (unwindState.CheckedLoadConfig == FALSE)
    {
        status = RtlGetImageBaseAndLoadConfig(userRip, &unwindState.ImageBase,
&unwindState.LoadConfig);
        unwindState.CheckedLoadConfig = NT_SUCCESS(status) ? TRUE :
unwindState.CheckedLoadConfig;

    if (unwindState.CheckedLoadConfig != FALSE)
    {
        if (unwindState.ImageBase != NULL)
        {
            __try
            {
                ProbeForRead(unwindState.LoadConfig,

 RTL_SIZEOF_THROUGH_FIELD(IMAGE_LOAD_CONFIG_DIRECTORY64,
GuardEHContinuationCount),
                              sizeof(UCHAR));

                if ((unwindState.LoadConfig != NULL) &&
                    (unwindState.LoadConfig->Size >=
RTL_SIZEOF_THROUGH_FIELD(IMAGE_LOAD_CONFIG_DIRECTORY64,
GuardEHContinuationCount)) &&
                    (BooleanFlagOn(unwindState.LoadConfig->GuardFlags,
IMAGE_GUARD_EH_CONTINUATION_TABLE_PRESENT)))
                {
                    goto CheckAddressInShadowStack;
                }
            }
            __except
            {
                goto CheckAddressInShadowStack;
            }
            return STATUS_SUCCESS;
        }
        return STATUS_SUCCESS;
    }
}
```

At first look, this might seem like a lot and could be confusing. But with some context it becomes a lot clearer. When implementing CET support, Microsoft ran into a problem. `NtSetContextThread` is widely used across the system by processes that don't necessarily respect the new "rules" of CET, and might use it to set RIP to addresses that are not found in the shadow stack. Those processes might also unwind into addresses that are not considered valid by CET, and since they were not compiled with proper CET support they will not have Static nor Dynamic Exception Handler Continuation Targets (which we wrote about in the previous post) that are recognized by CET. It won't be possible to enable CET across the system without breaking all those processes, some of which, like python, are very common. So, an option was added to "relax" `CetSetContextIpValidation` for those cases.

This check will be done for 2 continue types – all cases of `KCONTINUE_SET`, and cases of `KCONTINUE_UNWIND` where `RtlVerifyUserUnwindTarget` failed.

To know whether we are looking at such a case, `KiVerifyContextIpForUserCet` reads the `IMAGE_LOAD_CONFIG_DIRECTORY` structure from the headers of the module that contains the new RIP value. If the module has no image base, no load config or no Exception Handler Continuation Table, the function assumes that this is a module that is incompatible with CET and allows the action. But if the module has as Exception Handler Continuation Table, the new `RIP` value will be checked against the shadow stack, just as if relaxed mode would not have been enabled.

A fun side effect of this is that for any process where "relaxed mode" is enabled, setting the context or unwinding into JIT'ed code will always be permitted.

Load Config Directory Capturing

As part of this change MS also added a new `UNWIND_STATE` structure (that is our name, as this new structure is not in the public symbols) to hold the load configuration pointer and avoid reading the headers more than once. The new structure looks like this:

```
struct _UNWIND_STATE
{
    PVOID ImageBase;
    PIMAGE_LOAD_CONFIG_DIRECTORY64 LoadConfig;
    BOOLEAN CheckedLoadConfig;
} UNWIND_STATE, *PUNWIND_STATE;
```

The `CheckedLoadConfig` flag is used to indicate that the `LoadConfig` pointer is already initialized that does not need to be read again. We'll leave it as an excercise for the reader as to why this change was introduced.

Forward-thinking Downgrades

As hardware supporting CET is about the be released and hopefully become common over the next few years, the Windows implementation of CET doesn't seem to be fully prepared for the change and it looks like new challenges are only being discovered now. And judging by these "reserved" image flags, it seems that some developers are expecting more CET changes and downgrades in the future…

```
#define IMAGE_DLLCHARACTERISTICS_EX_CET_COMPAT                                 0x01
#define IMAGE_DLLCHARACTERISTICS_EX_CET_COMPAT_STRICT_MODE                     0x02
#define IMAGE_DLLCHARACTERISTICS_EX_CET_SET_CONTEXT_IP_VALIDATION_RELAXED_MODE  0x04
#define IMAGE_DLLCHARACTERISTICS_EX_CET_DYNAMIC_APIS_ALLOW_IN_PROC             0x08
#define IMAGE_DLLCHARACTERISTICS_EX_CET_RESERVED_1                             0x10  // Reserved for CET policy *downgrade* only!
#define IMAGE_DLLCHARACTERISTICS_EX_CET_RESERVED_2                             0x20  // Reserved for CET policy *downgrade* only!
```

Read our other blog posts: