Univerzalna detekcija ring3 debuggera

Ovaj anti-debug sam sasvim slucajno otkrio dok sam se igrao sa ExeCryptorom, istom priliko sam napravio jednu veliku gresku koja se na kraju nije pokazala tako velikom.

Prica kaze vako… Igrao sam se sa ExeCryptorom i patchovao sam CreateThread kako bi izbegao pravljenje dodatnih threadova od strane samog ExeCryptora, normalno, ExeCryptor je radio bez problema. Medjutim, prebacim se ja u Olly debugger i uradim isto, patchujem CreateThread sa retn i program pakovan ExeCryptorom nije hteo da radi. Medjutim pod ovakvim uslovima ExeCryptor radi bez problema kad nije prisutan ring3 debugger. Ovaj zakljucak sam izveo iz proste cinjenica da moj stealth nonintrusive tracer radi bez problema dok u Olliju pod istim uslovima nesto se desava lose. Znajuci sta sam promenio resenje nije bilo daleko, logicno je da CreateThread vrati handle ka novom threadu, ukoliko mi patchujemo CreateThread sa retn onda ce povratna vrednost biti nedefinisana, tj. bice ono sto je u eax bilo pre pozivanja CreateThread. Ok, assemblujem ja patch postavim BreakPoint na moj patch pokrenem aplikaciju, sve lepo radi, no ubrzo zatim stize exception EXCEPTION_INVALID_HANDLE, ja ga rutinski prosledim sa SHIFT+F9 (DBG_EXCEPTION_NOT_HANDLED) i program zavrsi sa svojim radom. Fora je u tome sto se CloseHandle-u prosledila slucajna vrednost koja je dovela do generisanja exceptiona koji nikad ne bi bio generisan u normalnim uslovima.

Nabrzaka sam sklepao program da proverim da li je ovo novi anti-debug:

```
push    0deadc0deh
callW   CloseHandle
```

Da, izgledalo je kao da se exception samo generise kad se pokrece kroz ring3 debuger, da bi proverio da li ovo postoji u normalnim uslovima ja sam debugovao proces kroz SoftICE i jasno video da nema nikakvog exceptiona, medjutim, sad ulazi lepo SoftICE u igru, drugi korak je da se program ucita u ring3 debugger i da se prati iz SoftICE njegovog izvrsavanje. Trik je da se u SoftICE postavi bpint 3 i onda se program ucitava u olly. Prvi break je u DbgBreakPoint koji u SoftICE hvata preko bpint 3 i samo prosledjujemo izvrsavanje debuggeru sa :x ili F5, sledeci break point je na entry pointu procesa, e tu radimo malu magiju i u SoftICE kucamo “eb eip prvi_byte_sa_entry” I pratimo izvrsavanje u SoftICE i ulazimo u CloseHandle->ntdll!NtClose->ntoskrnl!NtClose i jasno vidimo kako se generise exception redirektujuci EIP na ZwRaiseException iz kad je u debugovan ring3 debuggerom proces, a CloseHandle dobija dummy parametar kao handle.

Posto sam ovaj hint dao jednoj osobi u nadi da to ostane private, sokirao sam se kad je ta osoba pokusala sebe kroz ovaj trik da promovise po raznim forumima. Rec je o jednom naduvanom liku cije je znanje kodiranja = 0 ali to malo znanja sto ima koristi da ismejava newbie, normalno da je malo pametan ovaj trik ne bi sirio jer je ultimate nacin detekotvanja ring3 debugera, samo sto doticni nije razumeo u cemu je fora, dovoljno da se izvuku zakljuci o znanju doticnog i njegovoj zelji da sebe prikaze kao nekog ko nesto zna. Smeh.

Da bih vam pokazao koliko je glupo siriti nesto sto je trebalo da ostane private, pokazcu u ovom tekstu.

Naime postoji Native API NtRaiseException koji je exportovan u ntdll.dll, prototip je veoma jednostavan:

```
NTSYSAPI NTSTATUS NTAPI
NtRaiseException(
IN PEXCEPTION_RECORD ExceptionRecord,
IN PCONTEXT ThreadContext,
IN BOOLEAN HandleException );
```

Sa ovim APIjem mozete izazvati bilo koji exception i izforsirati pozivanje thread exception handlera. Pitate se u cemu je vic? U normalnim uslovima CloseHandle NIKAD ne bi izazvao EXCEPTION_INVALID_HANDLE, nikad!. Ali zato koristeci NtRaiseException mozemo da taj exception izazovemo u nasem programu bez problema i da isforiramo pozivanje exception handlera.

Ako debugujete iz ring3 i odigra se EXCEPTION_INVALID_HANDLE zbog prosledjivanja loseg argumeta u CloseHandle trebalo bi da nastavite izvrsavanje sa DBG_CONTINUE prosledjenim ka ContinueDebugEvent, drugim recima, pritisnite F9 u Olly. Medjutima, kad NtRaiseException je upotrebljen da se generise exception EXCEPTION_INVALID_HANDLE trebalo bi koristiti DBG_EXCEPTION_NOT_HANDLED i pozvati thread exception handler.

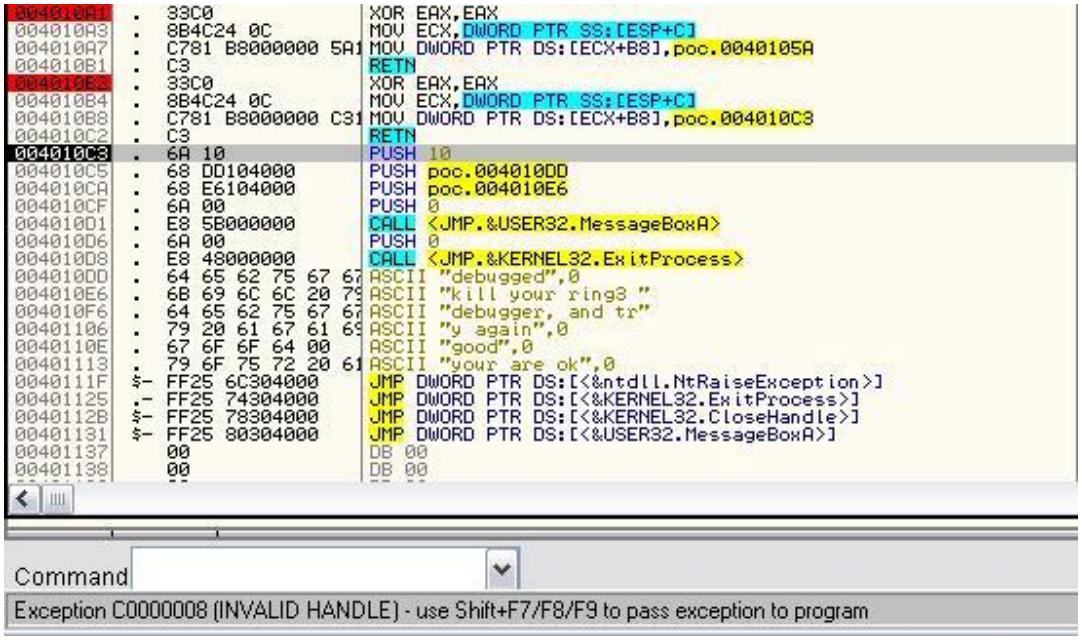
Drugim recima jednom morate koristiti DBG_EXCEPTION_NOT_HANDLED, a u drugom slucaju, pak, treba koristiti DBG_CONTINUE (shift+f9 and f9). Tako da ako u debuggeru stavite ovaj exception na ignore, sto znaci da ce se on prosledjivati sa DBG_EXCEPTION_NOT_HANDLED, exception koji je generisan sa NtRaiseException ce raditi sasvim lepo, ali cete biti detektovani od onog koji je geneirsan od strace CloseHandle. Takodje vazi i obratno, ako koristite DBG_CONTINU (samo f9), uhvatice vas exception generisan od NtRaiseException, ali ce CloseHandle raditit sasvim ok.

Da bih dokazao moju teroiju I pokazao kako su protection developeri glupi kao i oni koji su koristili moje hintove da sebe prikazu u boljem svetlu dajem vam ovaj mali poc code za detektovanje ring3 debuggera. Takodje sam cuo da su neki protection developeri poceli da koriste CloseHandle samo, ne razumejuci ni sami sta i zasto koriste. Smeh.

```
start:                  push    offset sehhandle1
                        push    dword ptr fs:[0]
                        mov     dword ptr fs:[0], esp

                        mov     ctx.context_ContextFlags, 10007h
                        mov     ctx.context_esp, esp
                        mov     ctx.context_eip, offset __debugged
                        mov     ctx.context_segCs, cs
                        mov     ctx.context_segDs, ds
                        mov     ctx.context_segFs, fs
```

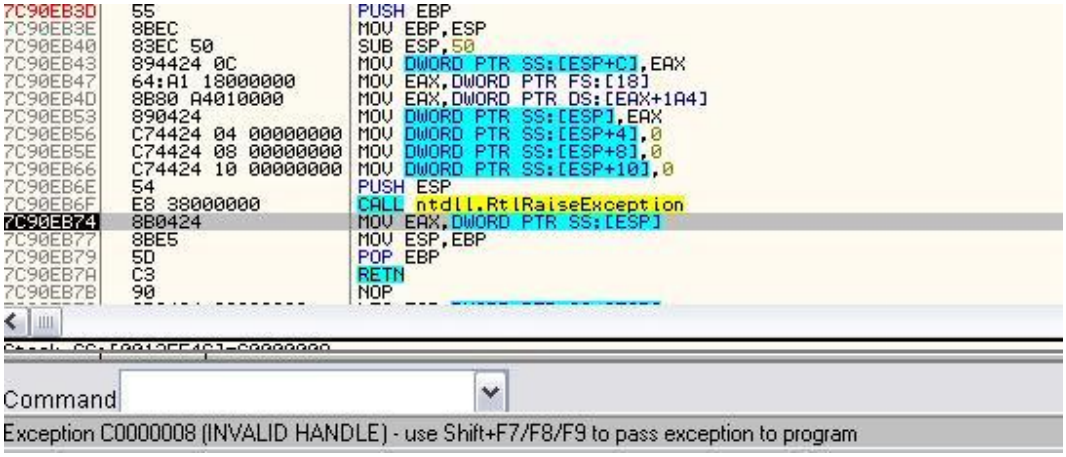
```
                        mov     ctx.context_segEs, es
                        mov     ctx.context_segSs, ss

                        push    1
                        push    offset ctx
                        push    offset exception
                        callW   NtRaiseException

__safe0:                pop     dword ptr fs:[0]
                        add     esp, 4

                        push    offset sehhandle2
                        push    dword ptr fs:[0]
                        mov     dword ptr fs:[0], esp

                        push    0deadc0deh
                        callW   CloseHandle
                        pop     dword ptr fs:[0]
                        add     esp, 4

                        push    40h
                        push    offset stitle
                        push    offset sabout
                        push    0
                        callW   MessageBoxA

                        push    0
                        callW   ExitProcess


sehhandle1:             xor     eax, eax
                        mov     ecx, [esp+0ch]
                        mov     [ecx.context_eip], offset __safe0
                        retn

sehhandle2:             xor     eax, eax
                        mov     ecx, [esp+0ch]
                        mov     [ecx.context_eip], offset __debugged
                        retn


__debugged:             push    10h
                        push    offset dabout
                        push    offset dtitle
                        push    0
                        callW   MessageBoxA
                        push    0
                        callW   ExitProcess

dabout                  db      "debugged", 0
dtitle                  db      "kill your ring3 debugger, and try
again",0
stitle                  db      "good", 0
sabout                  db      "your are ok", 0
```

Oki prvo treba popuniti context strukturu sa neophodnim podacima, stavljamo tamo ESP, EIP, segment registre i flagova. Obratite paznju da EIP pokazuje na __debugged label, jer mi govorimo procesu da idemo tamo (simuliramo da se exception odigrao na tom EIP), takodje prosledjujemo i exception code (0c0000008h), i 1 (bool HandleException, ako je 1 – pozovi thread exception handler, pokretanjem ovog koda u ollyiju sa unchekovanim exceptionima nas zaustavlja ovde:



Vidite EIP pokazuje __debugged label, ako pritisnemo F9 (DBG_CONTINUE) zavrsicemo na detekciji i na ExitProcess, ali ako pritisnemo shift+f9 I pozovemo thread exception handler, sto bi se inace dsilo u normalnim uslovima , bice pozvan nas SEH i nastavljamo sa izvrsavanjem dalje.

Ali uskoro imamo CloseHandle sa netacnim handlom prosledjenim:



Oki, CloseHandle je proizveo novi exception i sad bi trebalo samo pritisnuti F9 (DBG_CONTINUE) I izbeci pozvanje postavljenoh SEH handlera koji je lociran na : 4010B2h i koji ce redirektovati eip na __debugged label.

Da zavrsim, kada je ovaj exception izazvan preko NtRaiseException moramo ga proslediti sa DBG_EXCEPTION_NOT_HANDLED, no ukoliko je generisan od starne CloseHandle moramo ga proslediti sa DBG_CONTINUE. Sad uzmimo jednu hipoteticku situaciju gde nas protektor simultano izaziva exceptione, cas sa NtRaiseException, cas sa CloseHandle, zamislimo da se ovo ponavlja u kodu na nekoliko lokacija, recimo 10-20, dovoljno da izludi coveka J , a uzto se veoma lako implementira, recimo 2 exceptiona su preko NtRaiseException i onda jedan CloseHandle, itd. i kako cete vi znati kad koji da prosledite a koji da preskocite?

Da, da moze da se hookuju ova dva APIja, ali sta onda? Sto bi se ja uopste mucio i davao vam sansu da bez znanja pisanja drivera i hookovanja SSDT-a pobedite moj trik?

Ja lako mogu da koristim ovako nesto:

```
@sysenter               macro  syscall, parameters
                        local  __@@1, __@@2
                        push   eax
                        jmp    __@@2
__@@1:
                        mov    eax, syscall
                        mov    edx, esp
                        dw     340Fh                    ;sysenter 0F34h
__@@2:
                        call   __@@1
                        add    esp, (parameters*4) + 4 ; + 1 dummy EIP
endm
```

I direktno da komuniciram sa ntoskrnl.exe tako da bez hookovanja u ring0 teskocete uspeti da zaobidjete ovaj trik.

No svakako da trik nije nezaobilazan, cak sta vise lako se zaobilazi samo ako iole malo znate kako da programirate drivere, zaobilazenje ide ovako:

Hook -> ntoskrnl.exe!NtClose (ssdt), zatim koristimo ObreferenceObjectByHandle da proverimo da li postoji takav handle, ako postoji nastavljamo izvrsavanje preko starog NtClose, u usprotnom se vracamo nazad. Pa to je to…

Greetings: moji ortaci iz ARTeam, ap0x, LaFarge, Vrane, drustvo iz phearless, kao i generalno i odgovorno lice za phearless J

S verom u Boga, deroko/ARTeam