Anti-debugging techniques

IsDebuggerPresent WinAPI

Now in order to frustrate the malware analyst, malware can be detected in the presence of debuggers and show up in unexpected events. In order to detect the presence of a debugger, malware can either read some values or it can use API present to detect if the malware is being debugged or not.

One of the simple debugger detection tricks includes using the winAPI function known as KERNEL32.IsDebuggerPresent.

```
#define WIN32_LEAN_AND_MEAN
#include <windows.h>
#include <stdio.h>

int main(int argc, char **argv)
{
if (IsDebuggerPresent())
{
MessageBox(HWND_BROADCAST, "Debugger Detected", ""Debugger Detected"", MB_OK);
exit();
}
MessageBox(HWND_BROADCAST, "Debugger Not Detected", ""Debugger Not Detected"", MB_OK);
return 0;
}
```

Detecting a debugger using PEB:

When the process is created using CreateProcess API, and if the creation flag is set as DEBUG_ONLY_THIS_PROCESS then a special field is set in the PEB data structure in the memory

---

```
#define WIN32_LEAN_AND_MEAN
#include <windows.h>
#include <stdio.h>

int __naked detectDebugger()
{
__asm
{
ASSUME FS:NOTHING

MOV EAX,DWORD PTR FS:[18]
MOV EAX,DWORD PTR DS:[EAX+30]
MOVZX EAX,BYTE PTR DS:[EAX+2]
RET
}
}
```

```
int main(int argc, char **argv)
{
if (detectDebugger())
{
MessageBox(HWND_BROADCAST, "Debugger Detected", ""Debugger Detected"", MB_OK);
exit();
}
MessageBox(HWND_BROADCAST, "Debugger Not Detected", ""Debugger Not Detected"", MB_OK);
return 0;
}
```

Detection using HEAP flags:

When a program is run under a debugger, and is created using the debug process creation flags. The heap flags are changed. These Flags exit at a different location depending upon the version of the operating system.

On Windows NT based systems these flags exist at 0x0c offset from heap base.

ON Windows Vista based systems and later they exist at location 0x40 offset from the heap base.

These two flags initialized are ‘Force flags’ and ‘flags’.

ProcessHeap Base Points towards a _HEAP structure are defined as:

Reference :http://www.nirsoft.net/kernel_struct/vista/HEAP.html

```
typedef struct _HEAP
{
    HEAP_ENTRY Entry;
    ULONG SegmentSignature;
    ULONG SegmentFlags;
    LIST_ENTRY SegmentListEntry;
    PHEAP Heap;
    PVOID BaseAddress;
    ULONG NumberOfPages;
    PHEAP_ENTRY FirstEntry;
    PHEAP_ENTRY LastValidEntry;
    ULONG NumberOfUnCommittedPages;
    ULONG NumberOfUnCommittedRanges;
    WORD SegmentAllocatorBackTraceIndex;
    WORD Reserved;
    LIST_ENTRY UCRSegmentList;
    ULONG Flags;
    ULONG ForceFlags;
    ULONG CompatibilityFlags;
    ULONG EncodeFlagMask;
    HEAP_ENTRY Encoding;
    ULONG PointerKey;
    ULONG Interceptor;
    ULONG VirtualMemoryThreshold;
    ULONG Signature;
    ULONG SegmentReserve;
```

```
    ULONG SegmentCommit;
    ULONG DeCommitFreeBlockThreshold;
    ULONG DeCommitTotalFreeThreshold;
    ULONG TotalFreeSize;
    ULONG MaximumAllocationSize;
    WORD ProcessHeapsListIndex;
    WORD HeaderValidateLength;
    PVOID HeaderValidateCopy;
    WORD NextAvailableTagIndex;
    WORD MaximumTagIndex;
    PHEAP_TAG_ENTRY TagEntries;
    LIST_ENTRY UCRList;
    ULONG AlignRound;
    ULONG AlignMask;
    LIST_ENTRY VirtualAllocdBlocks;
    LIST_ENTRY SegmentList;
    WORD AllocatorBackTraceIndex;
    ULONG NonDedicatedListLength;
    PVOID BlocksIndex;
    PVOID UCRIndex;
    PHEAP_PSEUDO_TAG_ENTRY PseudoTagEntries;
    LIST_ENTRY FreeLists;
    PHEAP_LOCK LockVariable;
    LONG * CommitRoutine;
    PVOID FrontEndHeap;
    WORD FrontHeapLockCount;
    UCHAR FrontEndHeapType;
    HEAP_COUNTERS Counters;
    HEAP_TUNING_PARAMETERS TuningParameters;
} HEAP, *PHEAP;
```

Following the C program can be used to detect the presence of a debugger using heap flags

---

```
int main(int argc, char* argv[])
{
  unsigned int var;
  __asm
  {
            MOV EAX, FS:[0x30];
            MOV EAX, [EAX + 0x18];
            MOV EAX, [EAX + 0x0c];
            MOV var,EAX
  }

  if(var != 2)
  {
            printf("Debugger Detected");
  }
  return 0;
}
```

GetTickCount WinAPI

This technique is using time difference between two task. Every milisecond GetTickCount value is increase 1 unit. Actually cpu' makes tasks very fast and after normally following code runs the time difference is 0. But when this code runs on debugger also debugger makes time duration between tasks. So difference is different from 0.

```
int count = GetTickCount();
   int count2 = GetTickCount();
   printf("Tick Count 2 = %d\n",count);
   printf("Tick Count 1 = %d\n",count2);

   if(count == count2)
           {
                   printf("No debugger detected");
           }
   else printf("debuger detected");
```
