resources.infosecinstitute.com

Anti-debugging and Anti-VM techniques and anti-emulation

5-6 minutes

```
void EnterDebugLoop(const LPDEBUG_EVENT DebugEv)
{
DWORD dwContinueStatus = DBG_CONTINUE; //
exception continuation
char buffer[100];
CONTEXT lcContext;
for(;;)
{
// Wait for a debugging event to occur. The second
parameter indicates
// that the function does not return until a
debugging event occurs.
WaitForDebugEvent(DebugEv, INFINITE);
// Process the debugging event code.
switch (DebugEv->dwDebugEventCode)
{
case EXCEPTION_DEBUG_EVENT:
// Process the exception code. When handling
// exceptions, remember to set the continuation
// status parameter (dwContinueStatus). This value
// is used by the ContinueDebugEvent function.
switch(DebugEv->u.Exception.ExceptionRecord.ExceptionCode)
{
case EXCEPTION_ACCESS_VIOLATION:
// First chance: Pass this on to the system.
// Last chance: Display an appropriate error.
break;
case EXCEPTION_BREAKPOINT:
if (!fChance)
{
dwContinueStatus = DBG_CONTINUE; // exception
continuation
fChance = 1;
break;
}
lcContext.ContextFlags = CONTEXT_ALL;
```

```
GetThreadContext(pi.hThread, &lcContext);
ReadProcessMemory(pi.hProcess , (LPCVOID)
(lcContext.Esp ),(LPVOID)&rtAddr, sizeof(void *),
NULL );
if
(DebugEv->u.Exception.ExceptionRecord.ExceptionAddress
== pEntryPoint)
{
printf("\n%s\n", "Entry Point Reached");
WriteProcessMemory(pi.hProcess
,DebugEv->u.Exception.ExceptionRecord.ExceptionAddress,&OrgByte,
0x01, NULL);
lcContext.ContextFlags = CONTEXT_ALL;
GetThreadContext(pi.hThread, &lcContext);
lcContext.Eip--; // Move back one byte
SetThreadContext(pi.hThread, &lcContext);
FlushInstructionCache(pi.hProcess,DebugEv->u.Exception.ExceptionRecord.Ex
dwContinueStatus = DBG_CONTINUE   ; // exception
continuation
putBP();
break;
}
// First chance: Display the current
// instruction and register values.
break;
case EXCEPTION_DATATYPE_MISALIGNMENT:
// First chance: Pass this on to the system.
// Last chance: Display an appropriate error.
dwContinueStatus = DBG_CONTINUE ;
break;
case EXCEPTION_SINGLE_STEP:
printf("%s", "Single stepping event ");
dwContinueStatus = DBG_CONTINUE ;
break;
case DBG_CONTROL_C:
// First chance: Pass this on to the system.
// Last chance: Display an appropriate error.
break;
default:
// Handle other exceptions.
break;
}
break;
```

```
case CREATE_THREAD_DEBUG_EVENT:
//dwContinueStatus =
OnCreateThreadDebugEvent(DebugEv);
break;
case CREATE_PROCESS_DEBUG_EVENT:
printf("%s",
GetFileNameFromHandle(DebugEv->u.CreateProcessInfo.hFile));
break;
case EXIT_THREAD_DEBUG_EVENT:
// Display the thread's exit code.
//dwContinueStatus =
OnExitThreadDebugEvent(DebugEv);
break;
case EXIT_PROCESS_DEBUG_EVENT:
// Display the process's exit code.
return;
//dwContinueStatus =
OnExitProcessDebugEvent(DebugEv);
break;
case LOAD_DLL_DEBUG_EVENT:
char *sDLLName;
sDLLName =
GetFileNameFromHandle(DebugEv->u.LoadDll.hFile);
printf("\nDLl Loaded = %s Base Address 0x%p\n",
sDLLName, DebugEv->u.LoadDll.lpBaseOfDll);
//dwContinueStatus = OnLoadDllDebugEvent(DebugEv);
break;
case UNLOAD_DLL_DEBUG_EVENT:
// Display a message that the DLL has been
unloaded.
//dwContinueStatus =
OnUnloadDllDebugEvent(DebugEv);
break;
case OUTPUT_DEBUG_STRING_EVENT:
// Display the output debugging string.
//dwContinueStatus =
OnOutputDebugStringEvent(DebugEv);
break;
case RIP_EVENT:
//dwContinueStatus = OnRipEvent(DebugEv);
break;
}
// Resume executing the thread that reported the
```

```
debugging event.
ContinueDebugEvent(DebugEv->dwProcessId,
DebugEv->dwThreadId,
dwContinueStatus);
}
}
int main(int argc ,char **argv)
{
DEBUG_EVENT debug_event = {0};
STARTUPINFO si;
FILE *fp = fopen(argv[1], "rb");
ZeroMemory( &si, sizeof(si) );
si.cb = sizeof(si);
ZeroMemory( &pi, sizeof(pi) );
CreateProcess ( argv[1], NULL, NULL, NULL, FALSE,
DEBUG_ONLY_THIS_PROCESS, NULL,NULL, &si, &pi );
printf("Passed Argument is %s\n", OrgName);
pEntryPoint = GetEP(fp); // GET the entry Point of
the Application
fclose(fp);
ReadProcessMemory(pi.hProcess ,pEntryPoint,
&OrgByte, 0x01, NULL); // read the original byte
at the entry point
WriteProcessMemory(pi.hProcess
,pEntryPoint,"\xcc", 0x01, NULL); // Replace the
byte at entry point with int 0xcc
EnterDebugLoop(&debug_event);  // User-defined
function, not API
return 0;
}
int main(int argc ,char **argv)
{
DEBUG_EVENT debug_event = {0};
STARTUPINFO si;
FILE *fp = fopen(argv[1], "rb");
ZeroMemory( &si, sizeof(si) );
si.cb = sizeof(si);
ZeroMemory( &pi, sizeof(pi) );
CreateProcess ( argv[1], NULL, NULL, NULL, FALSE,
DEBUG_ONLY_THIS_PROCESS, NULL,NULL, &si, &pi );
printf("Passed Argument is %s\n", OrgName);
pEntryPoint = GetEP(fp); // GET the entry Point of
the Application
```

```
fclose(fp);

ReadProcessMemory(pi.hProcess ,pEntryPoint,
&OrgByte, 0x01, NULL); // read the original byte
at the entry point

WriteProcessMemory(pi.hProcess
,pEntryPoint,"\xcc", 0x01, NULL); // Replace the
byte at entry point with int 0xcc

EnterDebugLoop(&debug_event);  // User-defined
function, not API

return 0;

}
```