Debugging Injected Code with IDA Pro

By ~ by malwareninja on September 27, 2011.

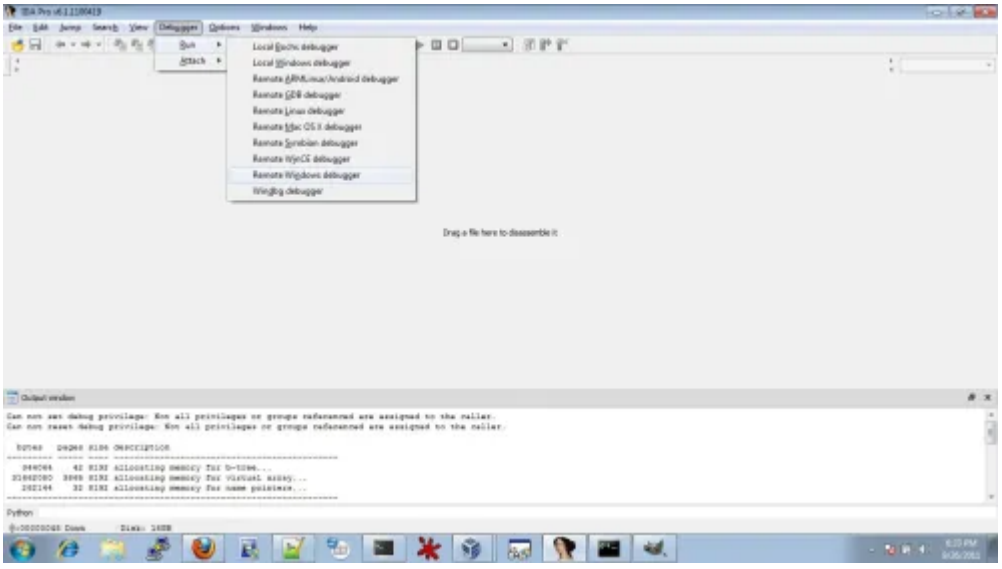
Published: 2011-09-27 · Archived: 2026-04-05 17:04:19 UTC

Hello all! Today I wanted to talk about how you go about debugging/analyzing injected code. In today's malware environment a lot of malicious code doesn't sit resident in memory in the context of it's own process. Back in the day you could look at task manager and recognize some weird executable that didn't belong. Those days are mostly over. The new(er) malware classes will typically inject malicious code and hook dll's in legitimate looking processes (explorer.exe, winlogon.exe, svchost.exe, etc.) to evade detection. This makes analyzing malware trickier as you need a wider skill set than opening up a bad binary in IDA. I'm going to shed some light on that process when you run into this type of malware.

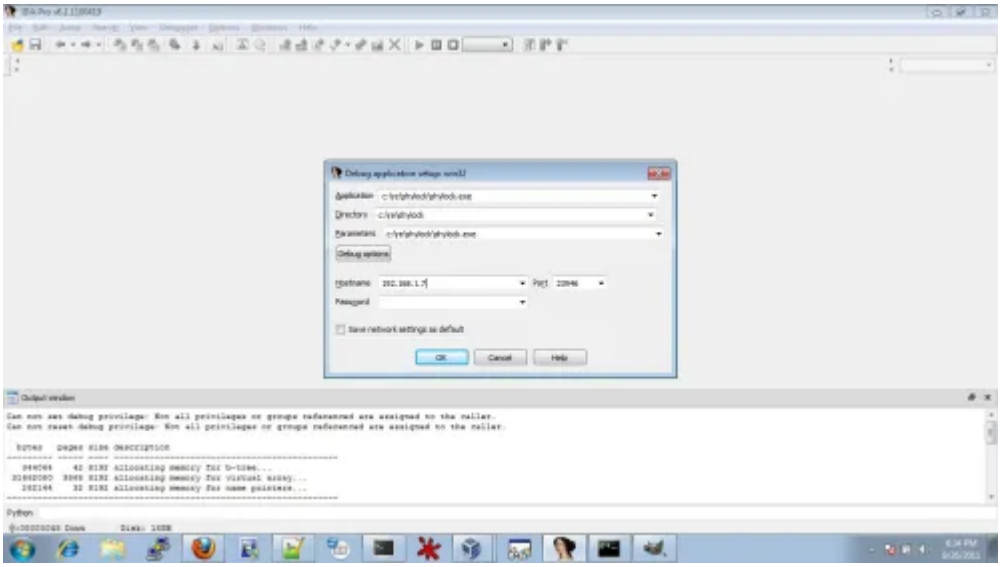
First off we need to find some malware that uses code injection. Code injection is usually done through the WriteProcessMemory API call through Windows. I've provided a sample here which just happens to be the shylock malware that was posted recently at Contagio. Download to follow along (the password is infected). This executable injects code into the explorer.exe process of the target machine (xp sp3 os running on virtualbox). This is what we will be working with if you want to follow along. Now I haven't done a complete in depth-analysis on this yet (it's coming) but I suspect there isn't any VM breakout that will totally hose your host OS. If there is well sorry bout that! 😛 You need to also make sure your vm is accessible from your host machine. I used 'Host-Only Networking' and made sure the guest was accessible from my host box.

So once you have your vm up (and it has an IP you can reach from your host box). You'll need to copy over to the share a file that exists in your IDA Pro file to enable remote debugging. The file is "win32_remote.exe". This is a server that allows IDA to connect up to a port on a remote server debugging to debug across the world or across memory in the sense of a VM. Now one caveat with this program is that it only allows one debugging session per server (depending on version, newer versions of IDA support multiple debugging sessions over the same port). So if you want to debug 2 programs at the same time (which we will be doing) you need two instances of this running on different ports. You specify the port with the -p flag and there is NO SPACES after the -p switch so if you want to set it up on port 1000 you'd run "win32_remote -p1000" from the command line. Tiga also has posted a video tutorial about remote debugging with IDA. His entire tutorial series is very good.

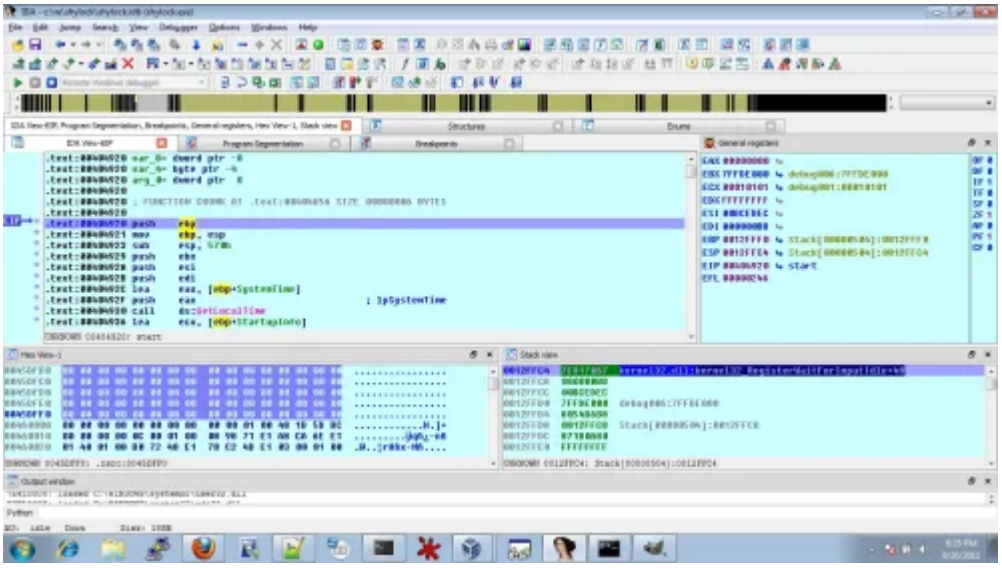
Open up a IDA Pro and Run -> Remote Win32 Debugger



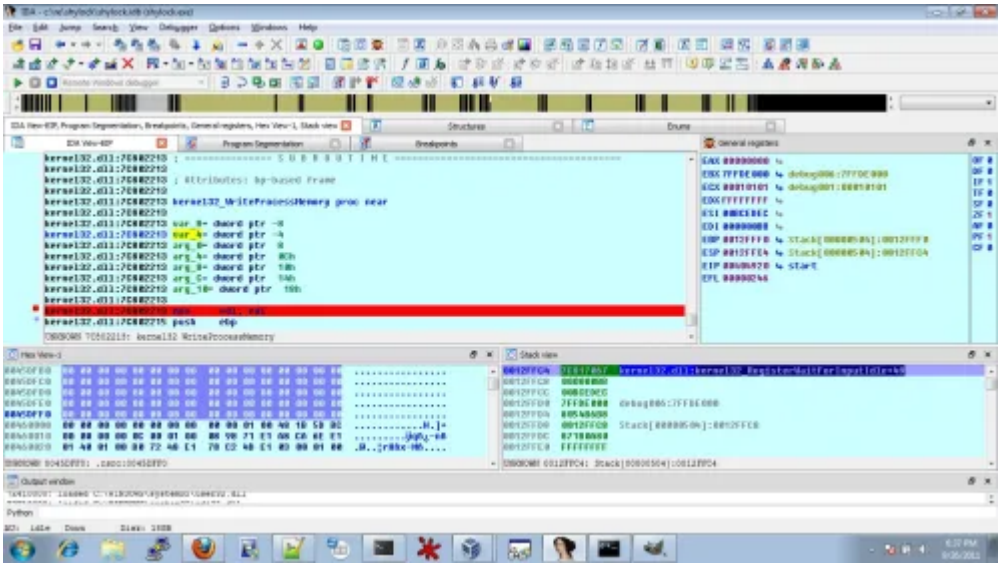
Make sure your connection/paths are correct.



Click ok and you'll break at the entry point of the module



Now we're going to set a breakpoint at WriteProcessMemory() (In IDA that equates to kernel32_WriteProcessMemory. From here on out it will be referred to as WriteProcessMemory)



Hit f9 to go and it breaks on WriteProcessMemory() (How did I know how to break here? I reversed the program roughly to get a feel for the program from the beginning up until this point.)

Now the code injection routine is a separate link here. shows why we want to break on WriteProcessMemory(). There are a few basic methods on how to inject code into a process that is not yours on Windows. Here is a good breakdown describing those methods. Most of the tactics revolve around WriteProcessMemory system call. This particular piece of malware uses the third type of injection mentioned in the code project article. Before this specific function was reached the malware took a snapshot of the system state and iterated through the processes until it found explorer.exe then called this function. So the short version of the disassembly is that it opens the target process, allocates some memory inside the process, writes memory that was allocated (repeats 3 times), then starts a remote thread to execute this new code, wait for thread to exit then cleanup handles. The reason 3 sections of memory are mapped into the target process is there is a loader there that reconstructs a dll in memory that is allocated inside Explorer. This happens all before the exit status code is returned from the thread and the

code is successfully injected.

Let's fire up another IDA instance and use the Attach -> Remote Win32 Debugger and put in the port for the second server that was different than the first. Hit ok then we should see a process listing and let's choose our injected process (explorer.exe) from the menu. If you took note of the injected code locations from CreateRemoteThread structure.

```
HANDLE WINAPI CreateRemoteThread(
  __in   HANDLE hProcess,
  __in   LPSECURITY_ATTRIBUTES lpThreadAttributes,
  __in   SIZE_T dwStackSize,
  __in   LPTHREAD_START_ROUTINE lpStartAddress,
  __in   LPVOID lpParameter,
  __in   DWORD dwCreationFlags,
  __out  LPDWORD lpThreadId
);
  __in   LPTHREAD_START_ROUTINE lpStartAddress,
```

lpStartAddress [in]

> A pointer to the application-defined function of type **LPTHREAD_START_ROUTINE** to be executed by the thread and represents the starting address of the thread in the remote process. The function must exist in the remote process. For more information, see **ThreadProc**.

We can mark this location with a breakpoint once we attach to explorer.exe (before the thread is started but after the memory was written). Then we hit run in the shylock.exe (injector process) and then we should have a breakpoint hit in explorer.exe and sure enough we do. We can continue on reversing from here but let's dump this segment and save it so we can annotate our debugging sessions and build on this previous knowledge. The way we can do this in IDA is take a memory snapshot. We have to View -> Open Subviews -> Segments so that we can view a memory map. Noting our addresses from WriteProcessMemory we need to change those segments to Loader segments. Next up go to Debugger and take memory snapshot and choose only Loader Segments. If you notice in our column our only dump will be of the three sections we marked 'Loader' segment. If you don't mark them as Loader segments IDA will ignore them and exclude from putting them into the database/idb. Here you have it and that's how you dump injected code from any process with IDA Pro. Hope you enjoyed reading this article.

References:

[1] – Tiga's IDA video tutorials

[2] – CodeProject Code Injection methods

[3] – Contagio malware dump

[4] – IDA Docs Page

[5] – Virtualbox Networking Doc

Posted in Uncategorized
