# Process on a diet: anti-debug using job objects

secret.club/2021/01/20/diet-process.html

jm January 20, 2021

jm

Jan 20, 2021

In the second iteration of our anti-debug series for the new year, we will be taking a look at one of my favorite anti-debug techniques. In short, by setting a limit for process memory usage that is less or equal to current memory usage, we can prevent the creation of threads and modification of executable memory.

## Job Object Basics

While job objects may seem like an obscure feature, the browser you are reading this article on is most likely using them (if you are a Windows user, of course). They have a ton of capabilities, including but not limited to:

- Disabling access to user32 functionality.
- Limiting resource usage like IO or network bandwidth and rate, memory commit and working set, and user-mode execution time.
- Assigning a memory partition to all processes in the job.
- Offering some isolation from the system by “upgrading” the job into a silo.

As far as API goes, it is pretty simple - creation does not really stand out from other object creation. The only other APIs you will really touch is `NtAssignProcessToJobObject` whose name is self-explanatory, and `NtSetInformationJobObject` through which you will set all the properties and capabilities.

```
NTSTATUS NtCreateJobObject(HANDLE*            JobHandle,
                           ACCESS_MASK        DesiredAccess,
                           OBJECT_ATTRIBUTES* ObjectAttributes);

NTSTATUS NtAssignProcessToJobObject(HANDLE JobHandle, HANDLE ProcessHandle);

NTSTATUS NtSetInformationJobObject(HANDLE JobHandle, JOBOBJECTINFOCLASS InfoClass,
                                   void*  Info,      ULONG              InfoLen);
```

## The Method

With the introduction over, all one needs to create a job object, assign it to a process, and set the memory limit to something that will deny any attempt to allocate memory.

```
HANDLE job = nullptr;
NtCreateJobObject(&job, MAXIMUM_ALLOWED, nullptr);

NtAssignProcessToJobObject(job, NtCurrentProcess());

JOBOBJECT_EXTENDED_LIMIT_INFORMATION limits;
limits.ProcessMemoryLimit               = 0x1000;
limits.BasicLimitInformation.LimitFlags = JOB_OBJECT_LIMIT_PROCESS_MEMORY;
NtSetInformationJobObject(job, JobObjectExtendedLimitInformation,
                          &limits, sizeof(limits));
```

That is it. Now while it is sufficient to use only syscalls and write code where you can count the number of dynamic allocations on your fingers, you might need to look into some of the affected functions to make a more realistic program compatible with this technique, so there is more work to be done in that regard.

## The implications

So what does it do to debuggers and alike?

- Visual Studio - unable to attach.
- WinDbg
  - Waits 30 seconds before attaching.
  - cannot set breakpoints.
- x64dbg
  - will not be able to attach (a few months old).
  - will terminate the process of placing a breakpoint (a week or so old).
  - will fail to place a breakpoint.

Please do note that the breakpoint protection only works for pages that are not considered private. So if you compile a small test program whose total size is less than a page and have entry breakpoints or count it into the private commit before reaching anti-debug code - it will have no effect.

## Conclusion

Although this method requires you to be careful with your code, I personally love it due to its simplicity and power. If you cannot see yourself using this, do not worry! You can expect the upcoming article to contain something that does not require any changes to your code.