19 - Unpacking

Unpackers
Packer Anatomy
Packed Executables
Identifying Packed Programs
Unpacking Methods
Manual Unpacking
IAT Rebuilding
OEP Finding
Unpackers

• Possible benefits:
  − Help malware avert antivirus software
  − Complicate malware analysis
  − Reduce the size of code.

• Packer transforms into executable containing transformed executable as data together with an unpacking stub.
Packer Anatomy

- A packer is an invertible function
  \[ f : \text{exe}_1 \rightarrow \text{exe}_2 \mid f^{-1}(f(e)) \approx e \]
- May pack entire executable or only the code and data sections.
- Import information must be stored so that it can be reconstructed at run time.
- OEP of packed program is start of the unpacking stub which
  - Unpacks original executable
  - Resolves imports
  - Transfers execution to original OEP
Packed Executables

• PE headers makes sure enough space is allocated to hold unpacked code.

• Nonpacked PEs contain import section identifying which functions to import and where to store their addresses. Windows loader does the work.

• Packers can
  - Preserve imports (note very stealthy)
  - Call `LoadLibrary` for each referenced DLL and `GetProcAddress` for each function imported.
  - Avoid all imports. Find `kernel32.dll` in memory and search for `LoadLibrary` and `GetProcAddress` inside it.

• Tail jump to unpacked OEP is last thing the unpacker does.
Identifying Packed Programs

- Indicators:
  - Few imports
  - Not much code visible in disassembly
  - OllyDbg may warn about packing
  - Section names indicate packers (UPX0)
  - Abnormal section sizes (raw size much smaller than virtual size)

- Entropy is high (frequency of different bytes is close to uniform)
Unpacking Methods

- **Automated Static**
  - Fastest and best when it works.
  - Must have access to unpacker.

- **Automated Dynamic**
  - Often available as OllyDbg scripts (ASProtect for FakeAV)
  - Dangerous because the malware is executed.

- **Manual Dynamic**
  - Discover the packing method and reverse engineer it, i.e., write an automated unpacker.
  - Run the packed program so that unpacking work is done, then dump the process and manually fix up PE header.
Manual Unpacking

- Find the OEP. You can do this manually.
- One method: use OllyDump plugin
  Some useful Plugins are included in the OllyPlugins Directory on Netlab, but they need to be copied to the directory containing OllyDbg.exe in order to work. Find out where that is and copy them there to test this.
  - Plugins->OllyDump->Find OEP by Section Hop
  - Program will break on the tail jump.
  - Write down OEP
  - Plugins->OllyDump->Dump Debugged Process
  - Check OEP in window and dump to file
Rebuild IAT with Import Reconstructor or ChimpRec

- Run ChimpRec
- Run your malware executable
- In ChimpRec select *Attach to an Active Process* dropdown and find malware process
- Enter RVA value of OEP (last 4 hex digits of OEP)
- Click *GetImports* (may not work)
- Click *Fix Dump*
Finding the OEP

- No strategy works will all packers
- Run program in OllyDbg and use single-stepping and breakpoints
  - Standard
  - Memory
  - Hardware
  - Tracing (with break conditions)
- Packing code may be self-modifying and contain mind-bending tricks.
Automated OEP Finding
OllyDump

- Most common method: OllyDump's Find by Section Hop (program jumps to another section)
  - *Trace Over* method does not enter called functions. Malware can use non-returning call to fool this.
  - *Trace Into* enters functions but may return false positives.
Automated OEP Finding
Manual Tail Jump Identification

- Tail jump is often last valid instruction before invalid instructions. (May be followed by 0 bytes.)
- Tail jump often jumps a long way.
- Tail jump likely jumps to location not containing instructions when execution begins.
Automated OEP Finding
Identifying End of Packer Execution

- Can put hardware breakpoint on stack memory address just after start of packing code hoping stack will shrink back to that point when packing is finished.

- Set breakpoints after every loop in the code. Manually intensive and may fail.

- May step over a function that does not return.

- Set breakpoint on calls to GetProcAddress. (May follow unpacking.)

- Set breakpoint on function you are sure is called by unpacked program. (GUI Programs call GetModuleHandleA.)

- Run Trace option in OllyDbg set breakpoint on .text section
For Next Time

- Try to unpack all the Chapter 18 lab executables manually.
- We will return to Chapter 11