Intro to Cracking and Unpacking

Nathan Rittenhouse – nathan_@mit.edu

Keygenning

- Take this crackme:
  - http://crackmes.de/users/moofy/crackme_2
  - Write a key generator

Process

- Watch where user data is inputted
  - Easy to see since there's only one dialogue box, but you have to discern which GetDlgItemTextA() corresponds to the name input
- Follow all operations that are done based on the string
  - Decompile it into C or some other language
  - You could also rip their assembly
- Seriously, in this case, it's *that* easy
- Mention hilarious flaw in algorithm

Unpacking

- Take packed program
- Unpack it into its original form
- Target is on SIPB IAP page

Process

- Notice calls to “jmpOneOverRet”
  - inc [esp]
  - ret
- Notice next instruction after call doesn’t look right
- Do a hex dump
- First
  - call jmpOneOverRet: 0xE8 <4 byte offset to function>
  - 0xFF
  - <real instruction>
- This simply jumps over the 0xFF

But…

- This is annoying to defeat manually, but yes, you could do it
- IDAPython / IdaRub are perfect for this
  - I forgot to install IdaRub
- Scan the entire segment for 0xE8 <4 byte offset to jmpOneOverRet>
  - Note, this offset is relative the address of the instruction AFTER the call
- When one is hit, NOP it, including the 0xFF
- Hide that portion for better readability

Programmatic Obfuscation

- Lots of shellcode type code constructs
  - Program parsing its own PE headers, etc – ominous
  - Even a z0mbie style hashing function, which is a dead giveaway
- Finds addresses for various addresses such as GetProcAddress, LoadLibraryA and stores them in a local data structure (esi+offset)

Anti-Debugging

- Call to NtQueryInformationProcess
  - With parameter of 7
  - Quick look at MSDN shows that this is ‘DebugPort’
- Obviously anti-debugging
- Uses the value returned by this to decrypt/decode stuff
- When this function returns, null out the value
  - -1 = being debugged
  - 0 = not being debugged

More Anti-Debugging

- Patches DbgUiRemoteBreakin()
- Why?
  - This function is run every time that DebugActiveProcess() is called via CreateRemoteThread()
    - It's how that breakpoint shows up when you attach a debugger
  - Normal code:
    - If(isDebugged()){
      - __asm{ int3 }
      - }

More Anti-Debugging

- Watch as protected.exe VirtualProtect()'s the function to be writable
- Then proceeds to build a jmp [ptr_ExitProcess]
- This means if one tries to attach with olly while the program is running, it will simply exit
  - Advanced olly can defeat this, but for learning purposes, we can simply avoid using this technique

More Unpacking

- Parsing of PE headers and then HeapCreate()'s done based off of the size of the executable
  - These executables are ntdll.dll and kernel32.dll
  - Oh, and a section of the exe called '.lex'
    - Provides a lookup table (hash style) for the values to fix up the relative calls with
- Contents of these are copied into the new memory

Decryption

- Value from debugger check is used in process of decrypting (done by xor)
- No sanity checking
  - This means if your executable doesn't decrypt correctly, you'll jump into garbage code later

Address Fixups

- Remember the library copies?
- Program searches through .lex for relative call / jmp sequences
- Usually look something like: 0xE8 0xEE 0xEE 0xEE 0xEE
  - Looks for this signature
- Once it's found, the address of the call is hashed and then used as an index into an array to find out what to replace the 0xEE's with
- After this, jump right into the original code

Removing Anti-Debugging

- This can be bypassed manually
  - Set the output buffer to 0's instead of the 0xFF's
  - But it's a pain to do it every time
- Patch NtQueryInformationProcess by forcing a 'push 0' (see disassembly) and replacing the call with an 'add esp, 0x14'
  - Yay tricks I picked up from breaking ZoneAlarm!

Cracking

- The new code is located on the heap, so just using ollydump / LordPE to dump the executable space and rewrite the OEP isn't an option
- Why not modify the unpacker?
  - Tell the program to put code in a new section in the executable instead of on the heap
- Open up LordPE and create two new sections – one for the first copy and one for the second
- Then, set the addresses of the 'code migration pointers' to them
- Break on the last jmp [realProgram] and then dump memory, fix up OEP

Cracking

- Now that we have a dumped exe, we need to fix the imports
  - Rebuild the import table so that the IAT will contain valid pointers to library calls
  - Sometimes, impRec will find the IAT automatically and rebuild from that, but it's often wrong
  - Search memory for 0xFF 0x25 (jmp dword ptr [ptr]) or just find a call that uses the IAT and then look around the area that the jmp points to
    - It's usually easy to look at a pointer and tell if it is really pointing to something
  - Then, correct impRec's base address and size, then fix the dump

End

- Crackmes.de is great for finding… crackmes…
- Openrce.org for good win32 reversing related articles
- Uninformed.org