LSE Week Crackme: Making-of

Pierre Bourdon

delroth@lse.epita.fr
http://lse.epita.fr

July 20, 2012

Plan

Crackme?

- Security and reverse engineering challenge
- You have an executable file and you need to crack it
- Patch, retrieve key, keygen, ...
- Done last year at LSE Week 2011
- Tried again this year with an harder challenge

Plan

Goals

- Last year the crackme was fully hand-written in x86 assembly
- This year I wanted to have an automated obfuscation and anti-debugging framework which did not required writing any assembly code
- I also had a few techniques I wanted to use in a "real" project

Packing layer 1

- Packing == storing a binary inside another binary in order to hide the code before it is executed
- You can't see packed code with hexedit, objdump or IDA
- The first layer uses a modified version of UPX
- No comment strings or UPX identifiers were kept
- Mostly obfuscation, really easy to dump

Packing layer 2

- The payload has a second layer of packing
- Custom, written in C
- Runs the packed code and wakes up on `SIGSEGV` to map the required memory pages
- Memory pages are stored encrypted and scrambled in the packed file to avoid simply reading it
- A lot harder to dump than the first payload: only loaded on demand, lots of signals that interrupt gdb, ...

Payload

- Written in C
- Asks the user to enter a password
- Computes a reversible result from this password and checks if it is equal to the one built in the compiled payload
- The reversible result is basically rotations and XOR operations on the 4 parts of the 32 bytes password

Toolchain

- Wouldn't be fun if you could simply read the compiled assembly code...
- Obfuscation!
- C compiled to QVM using the Quake3 C compiler
- QVM compiled to `x86_64` using a custom Python script
- Assembly is then shuffled, dead code is inserted, control flow obfuscation, on demand code decryption, etc.
- 5K of C code -> 620K of assembly -> 121K of compiled code

QVM assembly

- For more fun, QVM assembly is stack based
- `x86_64` is register based
- All instructions operate on memory, registers are only used to keep track of the stack pointer
- You can use all the registers you want for dead code!
- Also, stack based code is horrible to reverse (tools just aren't good enough)

Plan

Packing layer 1

- Unpacking UPX is very easy
- Find the jump to the OEP and dump memory
- Then reconstruct the binary if needed (UPX keeps the full ELF!)

Packing layer 2

- Hook in the SIGSEGV handler
- Dump the code pages when they are mapped in memory
- Another solution: reverse the packer (not very hard)

Defeating the payload obfuscation

- Tracing!
- Eliminate useless control flow instructions that way
- Then reverse the VM instruction set and find out how it matches to `x86_64`
- Idea: could machine learning be used to detect pattern in traces and defeat VM based obfuscation?

Plan

4 Conclusion

Questions?

- @delroth_
- delroth@lse.epita.fr