Linux Memory Forensics: A Real-Life Case Study

Georg Wicherski, 中國人民解放軍綁架目標

Linux Memory Forensics

A Real-Life Case Study

- Evanescent Bat Intrusion as case study
  - Large European IT company breach Mid-2013
  - Focus on Linux server compromises
- ELF Format basics (the common Linux executable format)
  - Understanding dynamic linking
  - What is a PLT hook?
- Detecting PLT hooks with Volatility
  - Everything was harder in the old days!

EVANESCENT BAT

MOTIVATING INTRUSION

Evanescent Bat

Small Profile and Skill Level Estimate

- Highly skilled implant developers
  - Proper error and corner case handling
  - In-depth familiarity with Linux ecosystem
  - Use of GCC plugin to scrub local variables running out of scope
    - Curiously, overwriting with 0s apparently not deemed fit.
  - Use of GCC plugin to encrypt all global & heap variables
- Skilled operator team
  - Did not adjust all framework configuration to operational environment
  - Dropped the ball by leaving traces in specific log files

ELF FORMAT PRIMER

SECTIONS, SEGMENTS AND STUFF

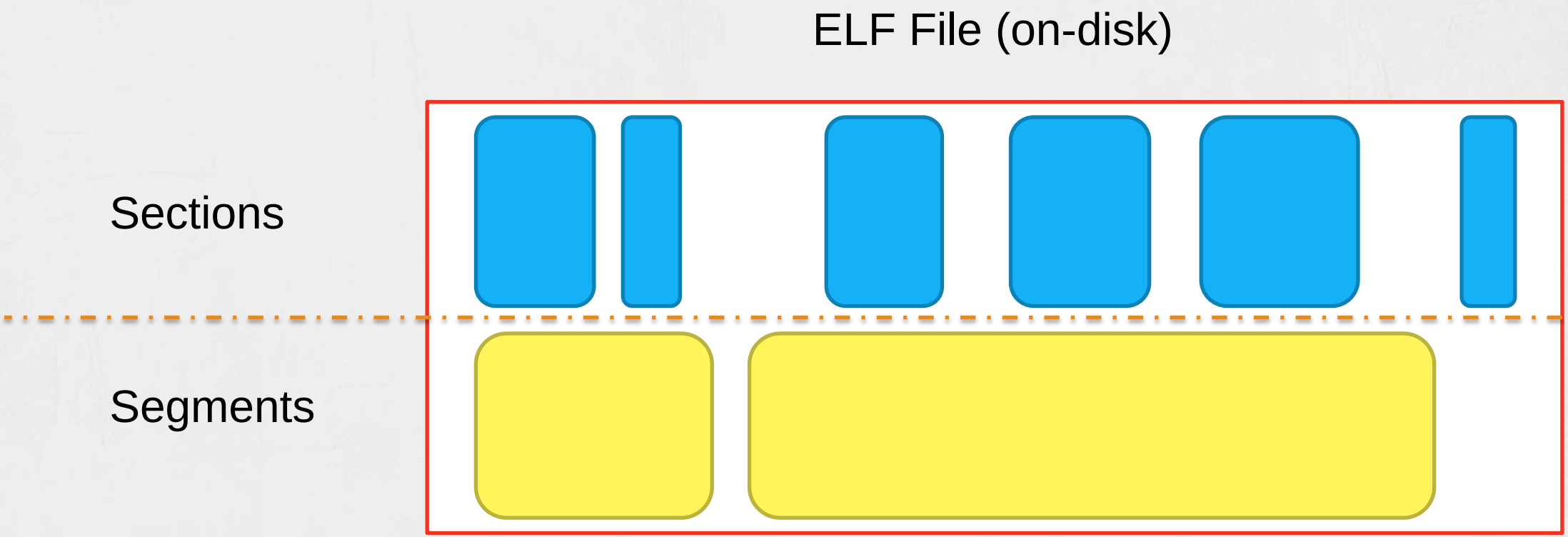
ELF Format

Segments vs. Sections

- Executable and Linkable Format
  - One file format for toolchain intermediates and finished executables
- Sections: Compiler toolchain view
  - (Many) arbitrary named sections of the binary file, arbitrary content
  - Meaning asserted by compiler and linker
  - File (offset) centric data structures
- Segments: Run-time loader view
  - Coarse grained, mostly two or three segments only (page permissions)
  - Memory address centric data structures (post-loading)

ELF Format

Segments vs. Sections



ELF Format

Segments vs. Sections

- Sections are irrelevant for execution, loader examines Segments
  - `PT_LOAD` segment headers describe to-be-loaded file parts
  - Segment headers and ELF header typically part of first `PT_LOAD` segment
  - Section headers typicall not part of any `PT_LOAD` segment!
- Stripping all sections headers from an executable leaves it intact
  - Can usually be done with a simple truncation of file
- Don’t rely on section information for parsing ELF files / dumps

```
truncate -s $(readelf -h elf | grep -F 'Start of section headers' | awk '{print $5}'); ./elf
```

ELF Format

Tools relying on section headers

- objdump, readelf, binutils suite
- pyelftools
- IDA Pro 6.5 (for dynamic symbols)
- …

All the tools!

Future Solutions

- Ryan O'Neil's http://bitlackeys.org/
  - **2011 - VMA Voodoo (DARPA CFT) (TR2)**
  - Not to be published before May 2014 (?)

- Your favorite A/V ;)

RUNTIME DYNAMIC LINKED IMAGES

LET ME SCRAMBLE THAT FOR YOU

Dynamic Linker

Symbol Resolution

- Statically linked executables just need to be mapped
- Dynamically linked executables are *interpreted* by `ld.so`
  - Kernel parses program header and maps memory
  - Dynamic linker “only” needs to resolve symbols in-memory
  - But section headers are not mapped to memory?!

Dynamic Linker

Symbol Resolution

- `PT_DYNAMIC` segment type just for the linker
  - Contains Tag / Value pairs (of native pointer size each)
  - Contains tags for many sections but points to memory addresses
    - String table, symbol table, relocations, …
- All the information is available through this segment
  - Using this for on-disk files requires reverse-translation of addresses
  - Section headers are much more convenient and thus tools rely on them
- Segment pointers get mangled / relocated during loading
  - Image base needs to be subtracted before translation

Dynamic Linker

Procedure Linkage Table

- PLT (or GOT) is a logical concept, not mandated by ELF spec
  - There is no formal import table, such as the PE IAT
- The PLT is a table of function pointers allocated by the compiler
  - Imported functions consist of `jmp [plt+offset]` stubs (x86 ⇒ "`JMPREL`")
  - The PLT is initially empty and just propagated by symbol-relative relocations
    - Typically with zero symbol-offset
  - Modern compilers enable lazy linking by initially placing a call to the linker
    - The slot is then resolved when called the first time

PLT Hooking

- Hooking: redirect function calls for arbitrary interception
  - Change arguments or add additional behavior, etc.
- Overwrite function pointer in `JMPREL` slot to redirect to hook
  - Save original value for calling original function from hook
- Challenge is locating the proper `JMPREL` slot(s)
  - Requires implementation of a dynamic linker in framework
  - Very generic technique because enforced by ABI
  - Probably still easier than inline hooking (requiring disassembly)

PLT HOOK DETECTION WITH VOLATILITY

HOW TO FALSE-POSITIVE

PLT Hook Detection

Volatility

- Volatility is a full-memory-dump forensics solution
  - Originally for Windows, added OS X and Linux support
  - Kernel centric point of view, little interpretation of user-space on Linux
- Plugin centric architecture, great for extending
  - New `linux_elfs` scans user-space mappings for ELF binaries
    - Uses "`\7fELF`" signature and pyelftools
  - New `linux_plthook` scans ELF binaries for PLT hooks
    - Uses `linux_elfs` and patched pyelftools

PLT Hook Detection

Defining a PLT Hook

- PLT legitimately contains function pointers to other modules
  - Not sorted by module, so even grouping by module does not help
  - `JMPREL` just specifies symbol name, not defining module
  - How can we distinguish a hook from a legitimate import?
- `DT_NEEDED` lists required / to-be-loaded dynamic .so libraries
  - A library should list all libraries it is importing symbols from
- Build transitive `DT_NEEDED` hull and check if `JMPREL` pointers point into any such module

PLT Hook Detection

DT_NEEDED Reality & False Positives

- DT_NEEDED is propagated based on -l compiler switches
  - Symbols can also be imported from main executable
  - Symbols can also be imported from any other module
    - If symbol not present at runtime, results in run-time dynamic linker error
- Some applications are linked wrong
  - apache2 modules implicitly link against libapr
- A real system has some legitimate PLT hooks (LD_PRELOAD)
  - libnpm_compat, bash (redirects malloc family to main executable), …
- Whitelist via --ignore switch / expert input

Demo

Getting the Code

http://ge.tt/1NscLaW1/v/0

- Volatility upstream in Summer
- Depends on patched pyelftools
  - Scheduled to be merged upstream at some point…
  - Thanks to Eli Bendersky for writing pyelftools!
- https://github.com/oxff/

CROWDSTRIKE

@ochsff — georg@crowdstrike.com