Advances in Linux process forensics with ECFS

Quick history

- Wanted to design a process snapshot format native to VMA Vudu
- http://www.bitlackeys.org/#vmavudu
- ECFS proved useful for other projects as well

Problem space

- A process address space is complex with many components

  - ELF binary format (structural nuances)
  - Dynamic linking
  - Architecture specific data and structures
  - Kernel specific data and code (VDSO, VSYSCALL)
  - Multiple threads

Hackers infect processes

- Process infection is stealth and flexible
- Processes are attacked in many ways
  - Viruses
  - Rootkits
  - Backdoors
  - Exploitation

Process forensics capable tools

- Volatility
- Rekall
- Second Look
- *ptrace* system call
- GDB
- Core dumps

Volatility in kernel land

- Use full system memory dumps
- Dwarf symbols to acquire high resolution insight into the Linux kernel
- Can be used to detect virtually any kernel malware
- System.map, and libdwarf are friendly for this (Creating kernel profiles)

Volatility in process memory

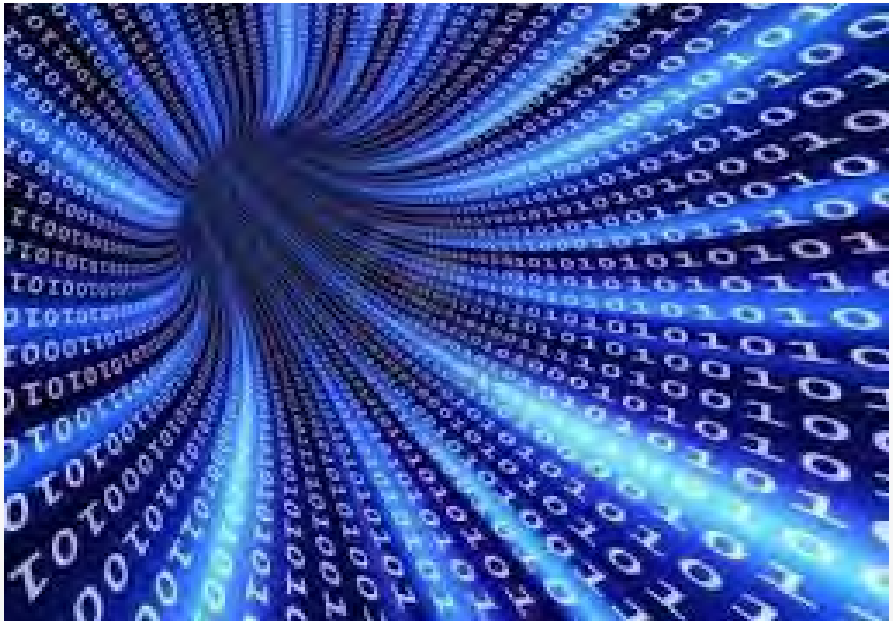
- ***detect_plt*** – A plugin for detecting PLT/GOT hooks by Georg Wicherski
- Process snapshots are raw
- Low resolution insight compared to kernel
- Plugin development is a big task
- **No profile can exist for each process**

Full memory dump vs. process memory dump

- Macrocosm: full memory dump
- Microcosm: process memory dump
- **ECFS** focuses on the Microcosm

Extended core file snapshot

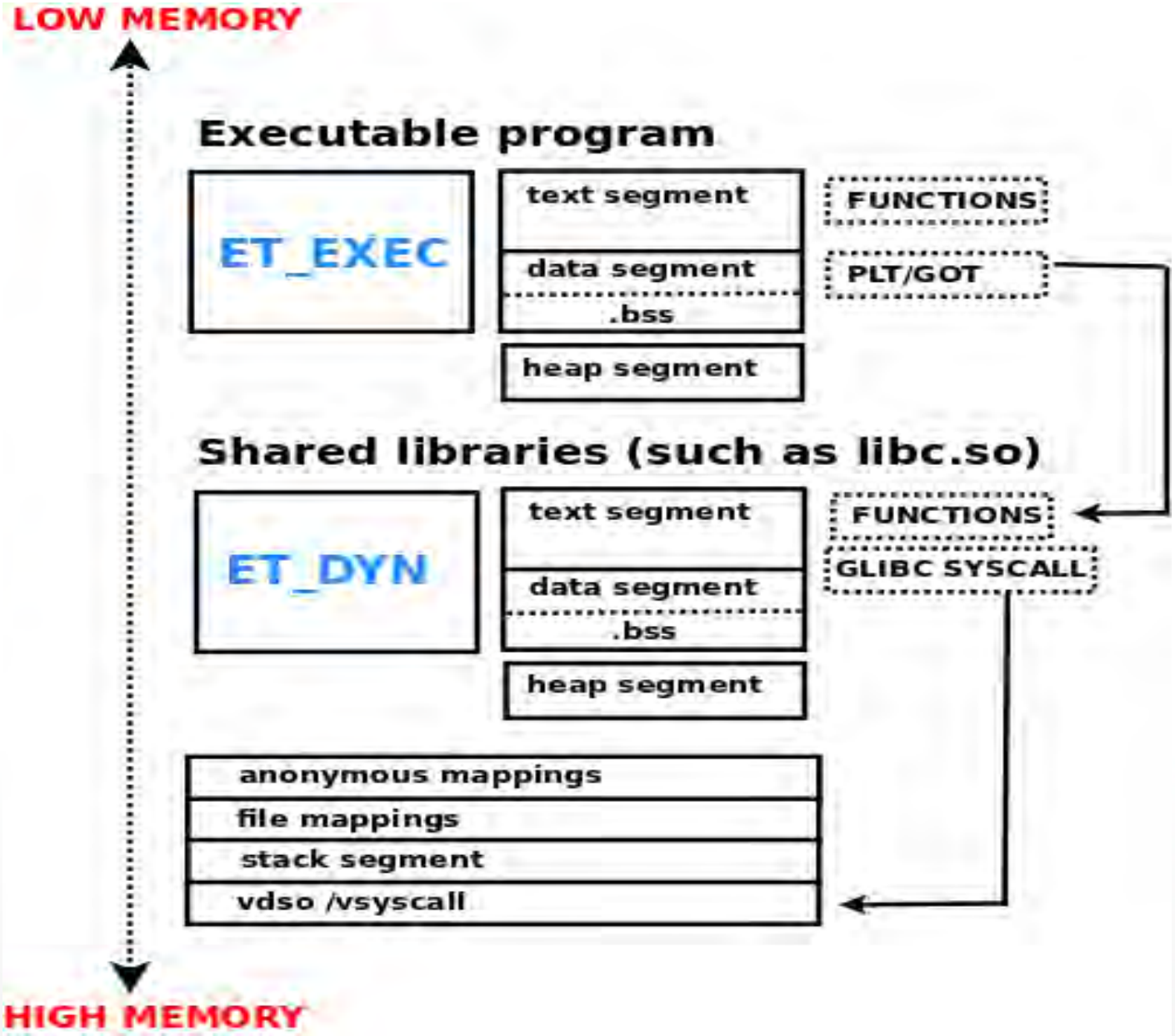
- A custom core file format for forensics analysis
- Backwards compatible with Linux Core files
- ***HI-DEF resolution process-snapshots***



Overview of attack surface

- ET_DYN Injection (.so files)
- ET_REL Injection (.o files)
- ET_EXEC Injection (exe files)
  - LD_PRELOAD
  - __libc_dlopen_mode
  - sys_ptrace
  - VDSO manipulation
  - Shellcode based loading
- Symbol and code hijacking
  - PLT/GOT poisoning
  - Trampolines (inline hooks)
  - .ctors/.dtors patching
  - Text segment modifications and other anomalies

Process memory layout



Definition of process memory forensics & analysis

- Understanding the process layout and structure
- Learning the programs runtime characteristics
- Identifying anomalous code or data
- Identifying process infection
  - Backdoors
  - Rootkits
  - Keyloggers
  - Viruses
  - protected binaries

Traditional core files .p1

- A snapshot of a process
- Contains segments (text, data, stack, heap)
- Contains all memory mappings
- File mappings and shared libraries
- ELF file header
- Program headers describing memory layout

Traditional core files .p2

- The PT_NOTE segment in a core file contains:
  - Register state *(struct elf_prstatus)*
  - Shared library paths
  - Auxiliary vector
  - Signal information

Traditional core files .p3

- A core file is dumped by the kernel when a process is delivered SIGSEGV
- /usr/src/linux/binfmt_elf.c
- Core files are useful for debugging a crashing application

Traditional cores are useless for forensics

- Highly dependent on the original executable being available
- Do not provide more than 4096 bytes of text images
- Does not give high resolution insight into a process

Recap on forensics goals

- Detect shared library injection
- Detect function hijacking (Trampolines)
- Detect PLT/GOT hooks
- Detect ELF object injection
- Function pointer redirection
- Shellcode injection
- Strange segment permissions
- ETC.

We want to quickly identify

- Userland memory rootkits
- Exploitation residuals
- Runtime malware/viruses

ECFS Technology

- ECFS is a technology that transforms a process image into an ELF file format
- ECFS makes process analysis much easier
- **Analogy (Photographing a process image)**

Core file (Low res)

ECFS file (Hi res)

ECFS Use cases

- Live malware analysis
- Process forensics
- Help break protected binaries
- Pausing and re-starting processes (Process necromancy)

ECFS Features outline

- Hooks into the Linux kernels core handler
- Backwards compatible with core files
- Full symbol table reconstruction
- Section header table reconstruction
- Built-in heuristics
- Custom sections containing

  - file descriptor data

  - socket data

  - IPC data

  - Signal data

  - Auxiliary vector

  - Compressed /proc/<pid> directory
- Re-execution (Pausing a process and running it later)
- Libecfs (API) for parsing ECFS files

Core handler (core_pattern)

- /proc/sys/kernel/core_pattern
- We tell *core_pattern* to pipe core files into our ecfs handler which then constructs an ecfs file
- Snapshots without killing the process are also possible (Not using core handler)

```
echo '|/opt/ecfs/bin/ecfs_handler -t -e %e -p %p -o
/opt/ecfs/cores/%e.%p' > /proc/sys/kernel/core_pattern
```

Symbol table reconstruction **.symtab**

- The PT_GNU_EH_FRAME segment contains FDE (Frame descriptor entries)
- .eh_frame data is used for stack unwinding
- Can be used to find the location and size of every function within the binary
- http://www.bitlackeys.org/#eh_frame

.symtab reconstruction is paramount

- Auto control flow (such as with IDA) fails when: ***Binary is encrypted***
- ECFS reconstructs symbol table with exact function location and size ***even with encrypted binaries***
- Show example of reconstructed ***Maya protected binary***

Symbol table reconstruction **.dynsym**

- located by looking at the dynamic segment and finding **DT_SYMTAB**
- resolve the address of every shared library function at runtime
- plug these values into the corresponding symbol table entry

ECFS Section headers

- Reconstructs most of the original section headers (**i.e., .text, .data, .plt, .got.plt,** etc.)
- ECFS adds many ***new*** never before seen section headers that are specific to process analysis

ECFS custom sections

- **.heap** – process heap
- **.stack** – process stack
- **.vdso** – virtual dynamic shared object
- **.vsyscall** – vsyscall page
- **._TEXT** – text segment (Not the same as .text)
- **._DATA** – data segment (Not the same as .data)

ECFS custom sections .p2

- **.procfs.tgz** - compressed /proc/pid
- **.prstatus** - process status info, registers, etc.
- **.fdinfo** – file descriptors, sockets, pipes
- **.siginfo** – Signal and fault info
- **.auxvector** – auxiliary vector from stack
- **.exepath** – path of original executable
- **.personality** – ECFS personality info

ECFS custom sections .p3

- **.arglist** – 'char **argv' of program
- **.fpregset** – Floating point registers

ECFS Custom section types

- **SHT_SHLIB** – Marks shared library segment mapping
- **SHT_INJECTED** – Marks injected ELF objects (ET_DYN, ET_REL, etc).
- **SHT_PRELOADED –** Marks shared libraries that were LD_PRELOAD'd

Injection detection heuristics

- ECFS uses techniques to detect injected ELF objects
- Can detect shared libraries that were not loaded by the dynamic linker
- Can detect any type of injected object file, executable or shared library
- Can differentiate between *dlopen and __libc_dlopen_mode*

Libecfs (API)

- ECFS parsing library
  - Tool development is made very easy
  - Program analysis on protected binaries
  - Detecting advanced process infections
  - Isolating the parasite code
  - Distinct access to program structures and data types

/usr/bin/readecfs

- Readecfs utility
- Similar to readelf
- Uses libecfs to parse ecfs files
- Can extract parasites, code, sections from ecfs files
- Still in early development

ECFS Re-execution

- ECFS snapshots can be taken and then re-executed later in time
- Can be used for live process migration
- Analysis of a suspicious process (re-executed within a sandbox)
- Beta stages
- https://github.com/elfmaster/ecfs_exec

Demo 1 – Detecting anti-forensics process cloaking technique

- Take snapshot of process infected with **Saruman** PIE executable injection
- Detect infection using simple ***readelf***
- Extract parasite code using ***readecfs***

http://www.bitlackeys.org/#saruman

Demo 2 – Detect userland rootkit

- Take snapshot of process infected with ***Azazel*** userland rootkit
- Use ***readecfs*** to extract the parasite code
- Use ***detect_plt_hooks*** to show PLT/GOT hooks in-place

Demo 3 – libecfs for tool development is easy

- The ***detect_plt_hooks.c*** is less ***than 60 lines of code***
- Can detect ELF Object injection
- Can detect Shared library injection (ptrace/mmap/__libc_dlopen_mode)
- Can detect LD_PRELOAD libraries
- Can detect PLT/GOT hooks

Demo 4 – ECFS snapshot execution

- Take a snapshot of a simple process that is reading from ***/etc/passwd*** and printing the results
- Restore the snapshot, and demonstrate how it restores the file streams, and continues reading from the file

Questions?

- ECFS
- https://github.com/elfmaster/ecfs
- ECFS snapshot execution
- https://github.com/elfmaster/ecfs_exec
- Saruman anti-forensics execve
- https://github.com/elfmaster/saruman