Live-Forensics

Using Linux as example

Michael Sonntag

Institute for Information processing and microprocessor technology (FIM)

Johannes Kepler University Linz, Austria

sonntag@fim.uni-linz.ac.at

What is "Live-Forensics"

- Gathering data from running systems
- You **WILL** change the system through this!
  - Aims: As little as possible & changes are known
- Often used in incident response to determine whether an event occurred
- Ideally: Completely scripted → Little room for errors, complete, and repeatable
  - Consistent and verifiable
- Special problem: There's only one chance
  - What you forgot to copy, you won't get later in the lab
  - What you do incorrectly will change the running system
  - No possibility of "re-doing" it → A second expert has to fully trust what you produced

Why do it?

- Analyzing volatile data is only possible as long as the computer is running
  - — Volatile = Will be lost when cutting the power
- But this should not happen on the suspicious system!
  - — So we need to copy this data somewhere else
    - Different disk, so we don't overwrite non-volatile data!
  - — Or at least gather as much information as possible (printouts, photographs)
- Reasons for doing this:
  - — Encryption keys might still be in the memory
  - — Some attacks (esp. rootkits for servers) are not stored on the disk but are purely RAM-based → No trace will remain from them (apart from perhaps the paging file)
  - — Data not yet written to disk (e.g. drafts of documents)

What data to collect (1)

- System information
  - System version, patch level, and time + time zone
  - Running processes/services
    - Including who started them, parent, command-line arguments, directory, …
  - Memory content: Individual processes as well as complete RAM
  - Open files, current (established) network connections, processes listening on sockets
  - Hardware information (partition tables, …)
    - Windows: This registry part is dynamic and not stored on disk!
  - Loaded kernel modules/DLLs

What data to collect (2)

- User activity
  - Currently logged on users
  - Shell history (might be stored as file too → offline)
  - Login history (especially failures and date/time; might be stored as file too)
- Network data
  - Cached names/addresses (DNS, ARP, …)
  - Mounted file systems/shares
  - Routing table (actual and cache), scheduled jobs, system load
  - Firewall configuration
- Log data: Only as far as temporary, e.g. on RAM disks
  - Everything else is offline forensics!

“But if the system is running, it will still change!”

- Yes, leaving the system running means, it will still receive E-Mails, serve files, swap out to disk, add log entries, …
- **BUT:**
  - This would happen even without the investigation
  - It will not create new evidence (“illegal” mails will not be created, they might just be moved from one computer to another)
  - Leaving it running might result in data being deleted or overwritten automatically
    - If no investigation would take place this would happen anyway, so it is merely necessary to weigh the increased information because of keeping the system running against the possibility of losing data if not immediately shutting it down!

Elements of good toolkits for live forensics

- Minimize system impact
  - Don’t copy anything to the disk; binaries should be as small as possible
- Enforce the use of known binaries only – if possible
  - Make sure that no library from the system investigated is used
- Extensive logging and checksums
  - Ensuring that no later modification can occur and that verification is possible
- No drivers needed for installing (→ CD-ROM better than USB!)
  - Can be difficult → Depends on the system investigated
    - If very well secured, this might be difficult (IDS tries to prevent exactly this!)
- Copy data directly to another system (→ Network/Share or USB)

Using netcat for transferring output

- If a network connection is available, the investigation system can normally be connected to it easily (additional switch/plug in on open port/…)
  - Note: Preliminary investigation necessary what IP/network address is used, if this cannot be determined differently (asking, looking at another system)
- We can copy all the output from the investigated system directly there
- Usage: nc <options>
  - Server (investigator system): `nc –v –l –p 10000 > log.txt`
    - -v: Verbose output, -l: Listen, -p: Port number to listen on
  - Client (system being investigated): `[command] | nc [ip of inv. system] 10000`
    - Possibly: [command] **2>&1** | …. to capture also StdErr

Preconditions

- How to get at your special statically-linked (check: ldd <executable>!) programs for extracting the information as seen on the following slides?
  - First: You must log in!
    - How? Do you know a password (changed by attacker?)? Or is the last user still logged on (computer locked, e.g. through screensaver?)? If you are a “normal” user, can you do what is requested here (often impossible!)?
  - Second: Linux → CD-ROM/USB requires mounting
    - = running a program, = changing access times on several files
      - Example: “mount” will change atime of /etc/ld.so.cache, /lib/libc.so.6, /etc/fstab, /dev/cdrom, /bin/mount and perhaps several others!
    - Documentation is therefore important → We can check later what was affected by the commands we issued!

Tools

- What we are using here are “standard” commands
- Several special “forensic” versions exist as well, but these have to be specially prepared (but we have to create statically linked versions anyway!)
  - These provide more information
  - They might directly access “deeper” information, e.g. the kernel process lists directly, instead of relying on the “output” of the kernel!
  - They might then require configuration/compilation for this specific system/kernel

Linux live response example: Date and time

- Note: Here shown as interactive, ideally this is a script file redirected to nc!
  - Also split over several slides
- Current date and time of system:

  -
    ```
    [root@mail ~]# date
    Wed Jul  6 14:56:29 CEST 2011
    ```

  -
    ```
    [root@mail ~]# date -u
    Wed Jul  6 12:56:31 UTC 2011
    ```

- Current local date and time
  - Note current real time separately!
- Current date and time in UTC
  - Timezone/DST corrected

LLRE: System version and patch level

- Where are we, who are we, what kind of system is this?
  - Patch level: Only for the “main” elements, i.e. the kernel
    - For all other “packages” this is distribution-specific!
  - Plus: Who are we currently (=which account; are we root?)
- General information:

  -
    ```
    [root@mail ~]# uname -a
    Linux mail.msv.at 2.6.18-238.12.1.el5.centos.plus #1 SMP Wed Jun 1 11:12:52 EDT
    2011 i686 i686 i386 GNU/Linux
    ```

  -
    ```
    [root@mail ~]# whoami
    root
    ```
- Packages (here: RedHat/rpm; only part of long list):

  -
    ```
    [root@mail ~]# rpm -qa
    basesystem-8.0-5.1.1.el5.centos
    bind-chroot-9.3.6-16.P1.el5
    httpd-2.2.8-1.el5s2.centos
    ……
    ```

LLRE: Cached ARP addresses

- Shows the other (**local**!) computers this system has been communicating with
  - Typically not that interesting because of very short timeout!
    - Note: Skipped several lines here!
  - [root@mail ~]# arp -vn

```
[root@mail ~]# arp -vn
Address                  HWtype  HWaddress           Flags Mask            Iface
192.168.2.47             ether   00:16:76:79:3F:42   C                     eth0
192.168.2.4              ether   00:01:E6:CE:68:C0   C                     eth0
62.218.130.126           ether   00:A0:F9:06:97:AD   C                     eth1
192.168.2.1              ether   00:0C:29:91:65:AC   C                     eth0
192.168.2.51             ether   00:13:20:C3:1A:79   C                     eth0
Entries: 17        Skipped: 0       Found: 17
```

  - Address (=IP), HWaddress (=MAC-address), HWtype (=Kind of interface), Iface (=Interface over which this device is reachable), Flags (C=Complete, M=Permanent, P=Published), Mask (=Netmask; only used in Proxy-ARP)

LLRE: IP configuration (1)

- List ip addresses
  - `[root@mail ~]# ip addr show`

```
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 16436 qdisc noqueue
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc htb qlen 1000
    link/ether 00:0d:9d:97:a0:31 brd ff:ff:ff:ff:ff:ff
    inet 192.168.2.240/24 brd 192.168.2.255 scope global eth0
3: gre1@NONE: <POINTOPOINT,MULTICAST,NOARP,UP,LOWER_UP> mtu 1420 qdisc noqueue
    link/gre 62.218.130.123 peer 192.168.0.250
    inet 10.0.1.1 peer 10.0.0.1/32 scope global gre1
```

  - `[root@mail ~]# ip maddr show`

```
1:      lo
        inet  224.0.0.1
        inet6 ff02::1
2:      eth0
        link  01:00:5e:00:00:01
        link  33:33:00:00:00:01
        inet  224.0.0.1
        inet6 ff02::1
4:     gre1
        inet  224.0.0.5
        inet  224.0.0.1
        inet6 ff02::1
```

„Normal“ addresses

MAC address

GRE tunnel

Multicast addresses

All hosts (IPv4, IPv6)

33:33:00:00:00:01 is the “MAC address” for IPv6 ff02::1

01:00:5e:00:00:01 is for 224.0.0.1

Lower-layer multicast addresses

OSPF

LLRE: IP configuration (2)

- List devices
  - [root@mail ~]# ip link show

```
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 16436 qdisc noqueue
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc htb qlen 1000
    link/ether 00:0d:9d:97:a0:30 brd ff:ff:ff:ff:ff:ff
3: gre1@NONE: <POINTOPOINT,MULTICAST,NOARP,UP,LOWER_UP> mtu 1420 qdisc noqueue
    link/gre 62.218.130.123 peer 192.168.0.250
```

Queuing discipline: When/How to send packets

Lower level (layer) is up (Ethernet: Cable connected)

- List tunnels
  - [root@mail ~]# ip tunnel show

```
gre1: gre/ip  remote 192.168.0.250  local 62.218.130.123  ttl 255
```

- ARP cache: `ip neigh show` (but see arp before: should be the same information!)
- Multicast route cache: `ip mroute show` (very rare to see anything)

LLRE: Routing table

- Shows the routing table: Not only static (→ offline investigation), but also dynamically added ones (e.g. by routing daemons)

  -
    ```
    [root@mail ~]# route -vn
    Kernel IP routing table
    Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
    10.0.0.1        0.0.0.0         255.255.255.255 UH    0      0        0 gre1
    192.168.2.0     0.0.0.0         255.255.255.0   U     0      0        0 eth0
    0.0.0.0         62.218.130.126  0.0.0.0         UG    0      0        0 eth1
    ```
  - Destination, Gateway, Genmask, Metric, IFace (=Interface) → Standard meaning
  - Flags: U (=Up), H (=Host route), G (=Gateway), R/D/M (=dynamic routes), C (=Cache entry), B (=Blackhole), ! (=Reject route)
  - Ref: Number of references to this route, Use: Lookups for this route
    - Mostly useful only for cache lookup (with additional parameter “-C”)!

LLRE: Routing table cache

- Printing the cache shows also what connection were used recently
  - Additional flags:
    - i (=Direct source), d (=Direct destination), q (=Masquerade), r (=Redirect), Ns (=SNAT), Nd (=DNAT), b (=Broadcast), m (=Multicast), # (=Reject), l (=Local)
  -
```
[root@mail ~]# route -vnC
Kernel IP routing cache
Source          Destination     Gateway         Flags Metric Ref    Use Iface
192.168.2.4     192.168.2.240   192.168.2.240   il    0      0       31 lo
127.0.0.1       127.0.0.1       127.0.0.1       l     0      0       44 lo
192.168.2.15    192.9.200.223   62.200.100.120  i     0      0       22 eth1
192.168.11.205  192.168.10.13   62.200.100.120        0      0      169 eth1
```
  - 192.168.2.240 (=eth0): Local address

    62.200.100.120 (using eth1): Next hop

    192.168.11.205: Some other address (not in local net, comes from another router)

See Linux source code ONLY (net-tools - src-rpm): lib/inet_gr.c

LLRE: Current network connections/open ports

- Currently active network connections (output is only a small selection!):
  - `[root@mail ~]# netstat -an`

```
Proto Recv-Q Send-Q Local Address       Foreign Address    State
tcp   0      0      192.168.2.240:3128 0.0.0.0:*          LISTEN
tcp   0      0      127.0.0.1:25        0.0.0.0:*          LISTEN
tcp   0      0      192.168.2.240:3128 192.168.2.48:1226 ESTABLISHED
tcp   0      0      127.0.0.1:52860     127.0.0.1:3551     TIME_WAIT
tcp   0      0      :::22               :::*               LISTEN
udp   0      0      127.0.0.1:123       0.0.0.0:*
udp   0      0      192.168.2.240:123  0.0.0.0:*
```

  - Plus lots of unix domain sockets (one way of inter-process communication)
- Info: Port 3551 = USV (apcupsd); LISTEN = Waiting for connection; TIME_WAIT: Making sure a connection is really and successfully closed; udp ports do not have a connection state; :::22 is IPv6 port 22 (=SSH)

LLRE: RAM content (1)

- Copy the complete RAM content (=physical memory)
- Software access: Typically two options exist
  - /proc/kcore: Memory in ELF core format → Ideal for debugging!
  - /dev/mem: Completely raw memory
  - Simple approach: dd if=/dev/mem of=mem.bin bs=4096 conv=noerror,sync
    - Note: Creates a local file; better to redirect it to nc!
- Physical access:
  - Firewire can use DMA for direct access to the whole memory (but: slow!)
    - Obviously requires a Firewire port to be present and active…
      - May hang the system; 128 MB → 15 seconds
  - Adding PCI cards
    - Before the intrusion ☺ or special hot-plug ones – system must support this!
    - Not generally available

LLRE: RAM content (2)

- Problem: Many distributions now prevent access to these files/their content!
  - E.g. /dev/mem: Access only to BIOS and PCI (=mapped I/O range), but not the RAM
  - Option: Use the crash driver or the fmem kernel module (creates /dev/fmem which doesn't have any restrictions)
    - Note: Drivers, modules, … must be compiled (or at least loaded)

      → Even more changes to the system and the RAM!
  - Even if modules are removed after imaging, the memory has changed
- Note: Most kinds of memory dump suffer from potential race conditions!
  - While the RAM is copied, the RAM is still being modified (=system continues to run!)
  - A memory dump is therefore not necessarily consistent
    - Problems when arguing in court that this is a "correct" dump!

LLRE: RAM content (3)

- Example with fmem:
  - Compile fmem for (but not "on"!) the target operating system
    - On a separate system; requires the development libraries (kernel headers)!
  - Get “fmem.ko” and “run.sh” to the target system (USB, CD-ROM, …)
  - Run “run.sh”, which loads the module and provides information which address areas (=size) are really RAM (and not mapped …)
    - Memory size MUST be provided; dd will read beyond end otherwise!

    -
      ```
      [root@mail src]# ./run.sh
      Module: insmod fmem.ko a1=0xc041c8d8 : OK
      Device: /dev/fmem
      ----Memory areas: -----
      reg00: base=0x00000000 (   0MB), size=2048MB: write-back, count=1
      reg01: base=0xf8000000 (3968MB), size=   4MB: write-combining, count=1
      -----------------------
      !!! Don't forget add "count=" to dd !!!
      ```
  - dd if=/dev/fmem bs=1M count=2048 | /mnt/cdrom/nc <Remote IP> <Port>

LLRE: Loaded kernel modules

- Shows all loaded kernel modules
  - Interesting especially if there is something “strange”
    - Here only a part of the normally very long list is shown!
  -

```
[root@mail ~]# lsmod
Module                  Size  Used by
ipt_MASQUERADE          7617  1
ipt_REJECT              9537  2
ipt_LOG                10049  42
iptable_nat            10949  1
iptable_mangle          6849  1
iptable_filter          7105  1
ip_nat_pptp             9797  0
ip_conntrack_pptp      15441  1 ip_nat_pptp
ip_tables              17029  3 iptable_nat,iptable_mangle,iptable_filter
cifs                  231385  0
asus_acpi              19289  0
pl2303                 21829  0
usbserial              33065  1 pl2303
ext3                  125385  2
jbd                    57321  1 ext3
uhci_hcd               25421  0
ohci_hcd               24937  0
ehci_hcd               33741  0
```

LLRE: Who is listening?

- We have seen which ports are open, but we also want to find out which programs have opened them
  - Lists open “files” (=file, directory, block/character file, library, stream, socket, …)
  - Attention: Really **LONG** listing … → Only very few selected lines shown!
    - [root@mail ~]# lsof -nP

-n: Numerical output, -P: no port name conversion

```
COMMAND     PID     USER  FD     TYPE    DEVICE     SIZE      NODE NAME
init          1     root  cwd     DIR     253,0    12288         2 /
init          1     root  rtd     DIR     253,0    12288         2 /
init          1     root  txt     REG     253,0    38652  10148267 /sbin/init
java       4017     root  mem     REG     253,0  1146570  13389759
                                   /usr/share/apache-tomcat-6.0.20/lib/catalina.jar
ntpd       3485      ntp   16u   IPv4      8600                UDP *:123
mysqld     3635    mysql  mem     REG     253,0           13038518
                                   /usr/lib/libkrb5.so.3.3 (path inode=12112988)
bash      28718     root  cwd     DIR     253,0     4096    621985 /root
bash      28718     root  rtd     DIR     253,0    12288         2 /
bash      28718     root  txt     REG     253,0   735004  15549661 /bin/bash
bash      28718     root  mem     REG     253,0   129900   4026535 /lib/ld-2.5.so
bash      28718     root    0u    CHR     136,1                  3 /dev/pts/1
```

What the “lsof –nP” command shows (1)

- Attention: Great care is necessary! Use documentation when interpreting output!
  - Most fields show different values/possess varying meanings depending on what is actually shown in this line (what kind of file it is)
- COMMAND: First nine characters of the command (no args); PID: Process ID
- USER: ID (or name) of the user the process belongs to
  - Differs on Linux if effective user ID has been changed
- FD: File descriptor number or special name
  - cwd (current working directory), rtd (root directory), pd (parent directory), txt (program “text” = code+data), mem (memory-mapped file), mmap (memory-mapped device), <number>r/w/u (open for read/write/read+write access), …
    - Standard file descriptor numbers: 0 = StdIn, 1 = StdOut, 2 = StdErr

What the “lsof –nP” command shows (2)

- TYPE: Type of the file
  - IPv4, IPv6, PIPE, DIR (directory), REG (“regular”=normal file), CHR (character file/device), BLK (block file/device), unix (Unix domain socket), LINK (symbolic link)
- DEVICE: Device numbers separated by commas
- SIZE: Size of the file
  - Great care must be taken for everything which is not a normal file: e.g. for sockets it might be the amount of data in memory (=buffer) at the moment!
- NODE: Node/Inode number of the file
- NAME: name of mount point(s), filesystem, file name, device name, local/local&remote addresses of network files, etc.

LLRE: Running processes

- This includes services; the command also shows which user runs them
  - Lots of lines (system as well as potentially interesting ones) removed!

– [root@mail ~]# ps aux

```
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root         1  0.0  0.0   2160   592 ?        Ss   May17  13:58 init [5]
root         2  0.0  0.0      0     0 ?        S<   May17   0:35 [migration/0]
apache     800  0.2  1.9  68236 41020 ?        S    15:14   0:08 /usr/sbin/httpd
ntp       3363  0.0  0.2   4508  4504 ?        SLs  May17   0:22 ntpd -u ntp:ntp -p
  /var/run/ntpd.pid –g
mysql     3513  0.0  3.8 184544 80332 ?        Sl   May17  62:16
  /usr/libexec/mysqld --basedir=/usr --datadir=/var/lib/mysql --user=mysql
  --federated --log-error=/var/log/mysqld.log --pid-file=/var/run/mysqld/mysqld.pid
  --socket=/var/lib/mysql/mysql.sock
root     28332  0.0  0.1   9908  2696 ?        Ss   14:55   0:00 sshd: root@pts/1
root     28335  0.0  0.0   4764  1480 pts/1    Ss   14:55   0:00 –bash
```

What the "ps aux" command shows (1)

- USER: Effective user name (preferably textual if existing)
  - Normally the same as the real user ID (who started it), but may be different for SUID executables (then it is the UID of the file owner)
- PID: Process ID (unique number of this process)
  - Other output formats also include PPID (=ID of parent process)
- %CPU: CPU this process currently uses in percent
- %MEM: Physical memory share (RAM) in percent of this process
  - This excludes memory in swap or parts of the program never loaded
- VSZ: Virtual memory in kB: Memory reserved, but swapped out
- RSS: Resident Set Size in kB: Non-swapped physical memory

What the "ps aux" command shows (2)

- TTY: Controlling terminal (only exists for processes associated with one!)
  - Services don't have a terminal
- STAT: Process state code(s)
  - R (Running/runnable), S (Interruptible sleep), T (Stopped, e.g. tracing), Z (Zombie – terminated, but not collected by parent), < (high priority), N (low priority), l (multi-threaded), L (has pages locked in memory), s (session leader; first of a group of processes; often a kind of shell), …
- START: Time when the process was started
  - Day+month; hour+minute only if less than 24 hours
- TIME: CPU time (user + system)
- COMMAND: Command line with all the arguments passed to it

LLRE: Currently logged on users

- Currently logged on users: /var/run/utmp (retrieved with the “w” command)
  - The history of logins is saved under /var/log/wtmp (binary file!) → Offline
  -
```
[root@mail ~]# w
 15:41:54 up 18:57,  1 user,  load average: 0.48, 0.29, 0.25
USER     TTY      FROM              LOGIN@   IDLE   JCPU   PCPU WHAT
root     pts/1    fim211.fim.uni-l 10:09    0.00s  0.12s  0.09s -bash
```
- USER: Name/ID of the user
- TTY: Where the user is logged in, FROM: Where he is connecting from
- LOGIN@: (Time of login), IDLE (Idle time), JCPU (CPU time of this + all child processes which are running), PCPU (CPU time of currently active processes)
- WHAT: What shell is being used

LLRE: Mounted file systems

- Lists the mounted file systems

  -
    ```
    [root@mail ~]# mount -l
    /dev/mapper/VolGroup00-LogVol00 on / type ext3 (rw)
    proc on /proc type proc (rw)
    sysfs on /sys type sysfs (rw)
    devpts on /dev/pts type devpts (rw,gid=5,mode=620)
    /dev/cciss/c0d0p1 on /boot type ext3 (rw) [/boot]
    tmpfs on /dev/shm type tmpfs (rw)
    none on /proc/sys/fs/binfmt_misc type binfmt_misc (rw)
    ```
- Output: Device (where does it “come from”), Mountpoint (where is it), file system type, options (r=read only, rw=read/write, …)
  - “-l”: For ext2, ext3 and XFS the (optional) labels are shown too (above: “[/boot]”)

LLRE: Cached DNS addresses

- Linux doesn't do any DNS caching itself by default (→ every program must do it on his own), but often a full DNS server (in caching mode) is used (or nscd; Name Service Cache Daemon); or some other software!
- NSCD: Also caches other databases (e.g. users, groups)
  - nscd –g: Show statistics only, but no content (option not available everywhere!)
- BIND: rndc dumpdb –cache
  - Will write the complete cache (in normal configuration format!) to /var/named/data/named_dump.db
    - Exact path: Check with your distro, especially if chroot'ed as often!
  - Problematic for computer forensics as the destination is a fixed local file!

LLRE: System load

- Typically not very interesting, but may show "strange" things
  - Nothing happening according to the output of other tools, but heavy system load?!?
    - But take care: System load is not CPU alone, but also disk, …!
  -
```
[root@mail ~]# uptime
 14:12:06 up 4 days, 17:27,  1 user,  load average: 0.17, 0.21, 0.19
```
    - Load: System load average for 1, 5, and 15 minutes
  -
```
[root@mail ~]# top -bn 1
top - 14:17:16 up 4 days, 17:32,  1 user,  load average: 0.18, 0.20, 0.18
Tasks: 209 total,   1 running, 208 sleeping,   0 stopped,   0 zombie
Cpu(s):  4.2%us,  0.9%sy,  0.1%ni, 94.4%id,  0.3%wa,  0.0%hi,  0.1%si,  0.0%st
Mem:   2075012k total,  1832808k used,   242204k free,   182276k buffers
Swap:  4128760k total,      116k used,  4128644k free,   789964k cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
 1852 root      15   0  2424  964  696 R  5.5  0.0   0:00.04 top
    1 root      15   0  2160  604  524 S  0.0  0.0   1:08.82 init
    2 root      RT  -5     0    0    0 S  0.0  0.0   0:03.55 migration/0
    3 root      34  19     0    0    0 S  0.0  0.0   0:00.11 ksoftirqd/0
    4 root      RT  -5     0    0    0 S  0.0  0.0   0:00.00 watchdog/0
    5 root      RT  -5     0    0    0 S  0.0  0.0   0:03.29 migration/1
```

LLRE: Scheduled jobs

- This refers to the “at” command
  - Most jobs are scheduled through cron: These are excluded here, as they are specified in static files, which will be investigated in the forensic copy!
  - Note: We will get only the “meta-information” here. The actual commands are stored in files, e.g. under “/var/spool/at”
  -
```
[root@mail at]# atq
2       2011-07-11 12:00 a root
```
- Result:
  - Id of job (environment and command will e.g. be in file /var/spool/at/a00002014d3f18)
  - Date and time when it will run
  - Queue name (here: “a”)
  - User to run this under

LLRE: Current firewall configuration

- What is running need not necessarily be the same as what is configured
  - — Dynamically adding exceptions is trivial!
- Can be very difficult for custom firewalls (→ Memory analysis), but easy if the standard “iptables” is used:
  - –
```
[root@mail ~]# iptables-save
# Generated by iptables-save v1.3.5 on Wed May  9 12:33:06 2012
*filter
:INPUT DROP [2789:339230]
:FORWARD DROP [232:22301]
:OUTPUT DROP [81:222570]
-A INPUT -s ?.?.?.? -j ACCEPT
-A INPUT -s 192.168.0.0/255.255.255.0 -p tcp -m tcp --dport 80 -j ACCEPT
……………………………………………
COMMIT
# Completed on Wed May  9 12:33:06 2012
```
- Result: A complete dump of the current configuration
  - — Exactly; normally this would be used to recreate it after the next boot!

LLRE: Current firewall counters

- Additional data, especially counters, might also be interesting:
  - Must be done for each table separately

  -
    ```
    [root@mail ~]# iptables -t {filter, nat, mangle, raw} -L -vnx
    Chain INPUT (policy DROP 2800 packets, 339810 bytes)
        pkts      bytes target     prot opt in     out     source               destination
     1578852 216446630 ACCEPT     all  --  *      *       ?.?.?.?              0.0.0.0/0
    ```
- Counter: How many packets matched this rule and how many bytes were these
  - Header: Were dropped (2800 packets, which were 339.810 bytes in total)
- Alternative: `iptables-save –c` will print packet and byte counters too
- Result: General information on how often this rule matched (exact numbers are of little use!), especially for “DROP” (which are not automatically logged!)

LLRE: Recovering deleted files still in use

- Malware often uses a file, starts a process, and then deletes itself
  - — Analysis becomes difficult, no “strange” file on disk, …
- But how can it still work?!? A (partial) copy remains in RAM!
  - — Carving/extraction from the RAM image
- Alternative: File carving in unallocated disk space
  - — Note: Might be overwritten on disk already, although it still exists in RAM!
- Alternative: As long as the process is running, a (hard-)link to it still exists under “/proc/<process id>/fd/<file descriptor number>”
  - — Just copy the file to another location (→ nc → to another computer!)
  - — Process id and FD can be obtained by lsof

LLRE: Example of recovering files still in use (1)

- Shell 1:

  -
    ```
    cp /var/log/messages /tmp
    vi /tmp/messages
    ```
- Shell 2:

  -
    ```
    rm –f /tmp/messages /tmp/.messages.swp
    lsof |grep vi
    cp /proc/2326/fd/4 /tmp/.messages-recovered.swp
    vi –r .messages-recovered.swp
    ```
- Note: In the example here the file is incomplete, as the swap file of vi only contains what has been shown to the user!
  - I.e., we recovered the complete swap file, but not the complete “source” file, as this one is just held in memory, but no longer has a file descriptor!

LLRE: Example of recovering files still in use (2)

```
[root@localhost tmp]# lsof |grep vi
vi         2326       root  cwd       DIR       253,0      4096      425985 /tmp
vi         2326       root  rtd       DIR       253,0      4096           2 /
vi         2326       root  txt       REG       253,0    594740     1441884 /bin/vi
vi         2326       root  mem       REG       253,0    129832      557087 /lib/ld-2.5.so
vi         2326       root  mem       REG       253,0   1689388      557289 /lib/libc-2.5.so
<… more libraries …>
vi         2326       root  mem       REG       253,0 56465584      831291
/usr/lib/locale/locale-archive
vi         2326       root    0u      CHR         4,1                   659 /dev/tty1
vi         2326       root    1u      CHR         4,1                   659 /dev/tty1
vi         2326       root    2u      CHR         4,1                   659 /dev/tty1
vi         2326       root    4u      REG       253,0     16384      425989 /tmp/.messages.swp
(deleted)
```

Process ID | File descriptor number | Filename

```
cp /proc/2326/fd/4 /tmp/.messages-recovered.swp
```

Summary

- Quite a lot of information can be gathered from a running system
  - Sometimes this is necessary, as some viruses/rootkits reside in memory only
- Careful preparation is needed
  - You need binaries working on this specific system
    - Which can be the latest OS version or a very old one, a common or an exotic one!
  - You have exactly one chance (and for caches, it is a brief one too!)
- Accessing the memory can get very difficult today with software only
  - A lot of contamination happens (installing kernel modules, …)
- Not touched here: Evaluation
  - Very difficult, as few tools exist. Mostly: Simple (string search) or very complex (debugging the OS itself, with manually interpreting the kernel data structures)

Thank you for your attention!

Michael Sonntag

Institute for Information processing and microprocessor technology (FIM)

Johannes Kepler University Linz, Austria

sonntag@fim.uni-linz.ac.at

Appendix: Programs used

- nc, date, uname, rpm, arp, ip, route, netstat, insmod, lsmod, lsof, ps, w, mount, nscd/mdc, top, at, atq, iptables, iptables-save, df, find, hostname, who
- Note: This is directly used software
  - If these are scripts/aliases, they might use different programs too
  - They will definitely use lots of libraries, unless statically linked
  - If automated as a script, some shell will be required as well (e.g. bash)
  - General tools (ls, cat, last, grep, md5sum, more, strings, telnet, vi, wc) might be needed as well
- Potentially as well:
  - dd/dcfldd, strace, unhide

Literature

- Jones, Bejtlich, Rose: Real Digital Forensics, Addison-Wesley 2006
- http://www.forensicswiki.org/wiki/Linux_Memory_Analysis
- Ivor Kollár: Forensic RAM dump image analyser

  http://hysteria.sk/~niekt0/foriana/doc/foriana.pdf

  fmem source: http://hysteria.sk/~niekt0/foriana/fmem_current.tgz
- Mariusz Burdach: Physical Memory Forensics

  http://www.blackhat.com/presentations/bh-usa-06/BH-US-06-Burdach.pdf