2010

Justitie

Netherlands Forensic Institute, <*www.nederlandsforensischinstituut.nl*>
Matthieu Suiche <http://www.msuiche.net>
BlackHat Briefing, Washington DC (February 2010)

TABLE OF CONTENTS

ADVANCED MAC OS X PHYSICAL MEMORY ANALYSIS

In 2008 and 2009, companies and governments (e.g. Law Enforcement agencies) interests for Microsoft Windows physical memory grew significantly. Now it is time to talk about Mac OS X. This paper will introduce basis of Mac OS X Kernel Internals regarding management of processes, threads, files, system calls, kernel extensions and more. Moreover, we are going to details how to initialize and perform a virtual to physical translation under an x86 Mac OS X environment.

ADVANCED MAC OS X PHYSICAL MEMORY ANALYSIS

INTRODUCTION

In 2008 and 2009, companies and governments (e.g. Law Enforcement agencies) interests for Microsoft Windows physical memory grew significantly. Now it is time to talk about Mac OS X. This paper introduces Mac OS X Kernel Internals regarding management of processes, threads, files, system calls, kernel extensions and more. We provide details on how to initialize and perform a virtual to physical translation under a x86 Mac OS X environment.

Physical Memory is widely known in the UNIX world as /dev/mem.

MEMORY ADDRESS TRANSLATION

QUICK TRANSLATION FORMULA

Most Operating Systems have a way to compute the kernel physical address even if you do not have the `cr3` register value which is used as Directory Table Base for virtual to physical address translation. If you want to have more detailed information on this, please refer to *Intel64 and IA-32 Architectures Software Developer's Manuel: Volume 3A System Programming Guide*.[1]

By kernel physical addresses, I mean the kernel image (*`__DATA` & `__CODE` sections)* physical address. Both contain important information and variables we need. For instance, to reconstruct the kernel address space we need to be able to use Smart Translation Formula which requires variables we can retrieve using Quick Translation Formula. As I said above, with Quick Translation Formula we can only access to `__DATA` and `__CODE` sections of the kernel image and not to allocated buffers.

Here is a summary of some operating systems with their corresponding formula to translate from Kernel Virtual Address (KVA) to Kernel Physical Address (KPA).

Operating System	Quick translation formula
x86 Linux	KPA = KVA – 0xC0000000
PlayStation 3 Linux	KPA = KVA - 0xC000000000000000
x86 Windows	KPA = KVA & 0x1FFFF000
Mac OS X	KPA = KVA

As you can see the formula for Mac OS X, is the easiest existing formula.

SMART TRANSLATION FORMULA

Using Quick Translation Formula, we can retrieve variables from `__DATA` section and initialized by *`slave_pstart()`* function of Mac OS X Kernel, which is called during the Operating System initialization.

[1] 3.6 PAGING (VIRTUAL MEMORY) OVERVIEW.

There are ~4 variables which are interesting to perform the Smart Translation Formula: `IdlePDPT`, `IdlePDPT64`, `IdlePML4` and `IdlePTD`.

`IdlePML4` variable is initialized even on 32-bits Operating System. PML4 stands for Page Map Level 4 paging structure. This method can be used to address up to 2^27 pages, which spans a linear address space of 2^48 bytes.

Then, using `IdlePML4` variable we can cover a translation mechanism for a linear address space of 2^48 bytes even if the processor cannot do it. Internally, in kernel structures, Mac OS X is using 64-bits addressing for memory objects.

These variable are used later to initialize *kernel_map* and *kernel_pmap* kernel structures/variables.

Here is a common output of these variables under Mac OS X Leopard.

```
*_IdlePML4:   [0x004EB00C] = 0x01219000
0x01219000: 27 A0 21 01 00 00 00 00 - 00 00 00 00 00 00 00 00 ................
0x01219010: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00 ................
0x01219020: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00 ................
0x01219030: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00 ................
0x01219040: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00 ................
0x01219050: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00 ................

*_IdlePDPT64: [0x004EB010] = 0x0121A000
0x0121A000: 27 C0 21 01 00 00 00 00 - 27 D0 21 01 00 00 00 00 ................
0x0121A010: 27 E0 21 01 00 00 00 00 - 27 F0 21 01 00 00 00 00 ................
0x0121A020: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00 ................
0x0121A030: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00 ................
0x0121A040: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00 ................
0x0121A050: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00 ................

*_IdlePDPT:   [0x004EB008] = 0x0121B000
0x0121B000: 01 C0 21 01 00 00 00 00 - 01 D0 21 01 00 00 00 00 ................
0x0121B010: 01 E0 21 01 00 00 00 00 - 01 F0 21 01 00 00 00 00 ................
0x0121B020: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00 ................
0x0121B030: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00 ................
0x0121B040: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00 ................
0x0121B050: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00 ................

*_IdlePTD:    [0x004EB004] = 0x0121C000
0x0121C000: 63 50 02 01 00 00 00 00 - 63 60 02 01 00 00 00 00 cP......c`......
0x0121C010: 63 70 02 01 00 00 00 00 - 63 80 02 01 00 00 00 00 cp......c.......
0x0121C020: 23 90 02 01 00 00 00 00 - 23 A0 02 01 00 00 00 00 ................
0x0121C030: 63 B0 02 01 00 00 00 00 - 63 C0 02 01 00 00 00 00 c.......c.......
0x0121C040: 63 D0 02 01 00 00 00 00 - 63 E0 02 01 00 00 00 00 c.......c.......
0x0121C050: 63 F0 02 01 00 00 00 00 - 63 00 03 01 00 00 00 00 c.......c.......
```

Symbols

Symbols are a key element of volatile memory forensics without them an advanced analysis is impossible. Symbols of Microsoft Windows are available on a remote server as standalone files, but on Mac OS X symbols are directly stored inside the executable in a segment/section called `__LINKEDIT`.

The easiest way to retrieve kernel symbols is to extract them from the kernel executable of the hard-drive.

Symbols are firstly used to retrieve the address of memory variable for Smart Translation Formula.

FAT HEADER

Mac OS X file format follows the FAT file format which contains magic signature of the header and the number of different architectures entries (i386, PowerPC or Both) inside the executable in big endian.

```
#define FAT_MAGIC 0xBEBAFECA

typedef struct _FAT_HEADER
{
      ULONG magic;
      ULONG nfat_arch;
} FAT_HEADER, *PFAT_HEADER;
```

To jump to the first architecture entry we add `sizeof(FAT_HEADER)` bytes to the pointer of the file header. Earch entry uses the following definition, and also uses the big endian endianess.

```
typedef struct _FAT_ARCH
{
    cpu_type_t cputype;
    cpu_subtype_t cpusubtype;
    ULONG offset;
    ULONG size;
    ULONG align;
} FAT_ARCH, *PFAT_ARCH;
```

The first field, `cpu_type`, indicates to the loader what kind of architecture this entry defines using the following description:

```
typedef enum
{
    CPU_TYPE_VAX = 1,
    CPU_TYPE_ROMP = 2,
    CPU_TYPE_NS32032 = 4,
    CPU_TYPE_NS32332 = 5,
    CPU_TYPE_MC680x0 = 6,
    CPU_TYPE_I386 = 7,
    CPU_TYPE_MIPS = 8,
    CPU_TYPE_NS32532 = 9,
    CPU_TYPE_MC98000 = 10,
    CPU_TYPE_HPPA = 11,
    CPU_TYPE_ARM = 12,
    CPU_TYPE_MC88000 = 13,
    CPU_TYPE_SPARC = 14,
    CPU_TYPE_I860 = 15,
    CPU_TYPE_ALPHA = 16,
    CPU_TYPE_POWERPC = 18,
    /* APPLE LOCAL 64-bit */
    CPU_TYPE_POWERPC_64 = (18 | CPU_IS64BIT),
    /* APPLE LOCAL x86_64 */
    CPU_TYPE_X86_64 = (CPU_TYPE_I386 | CPU_IS64BIT)
} cpu_type_t;
```

And the third field, `offset`, contains the raw offset of the architecture header.

We assume index x is the id of the **CPU_TYPE_I386** architecture. So we have `FAT_ARCH[x].cputype` equals to `CPU_TYPE_I386` and `FAT_ARCH[x].offset` as new pointer offset to the `MACH_HEADER` structure.

Mach Header

Now we have a pointer the i386 architecture binary using the following header definition and little-endian endianess.

```
#define MH_MAGIC 0xfeedface

typedef struct _MACH_HEADER
{
   ULONG Magic;
   cpu_type_t cputype;
   cpu_subtype_t cpusubtype;
   ULONG filetype;
   ULONG ncmds;
   ULONG sizeofcmds;
   ULONG flags;
} MACH_HEADER, *PMACH_HEADER;
```

This architecture validity can be verified using the `0xfeedface` magic key.

Now we can read what Apple calls *commands*, the field `MACH_HEADER.ncmds` indicates the number of commands inside the Mach-O binary.

We have to add `sizeof(MACH_HEADER)` to the Mach-O header pointer to have a pointer to the first command entry. There are different commands types and size of commands depends of their type. Most important commands types are `LC_SEGMENT` and `LC_SYMTAB`.

```
#define LC_SEGMENT 0x1    /* file segment to be mapped */
#define LC_SYMTAB  0x2    /* link-edit stab symbol table info (obsolete) */
```

And very two first fields contains information about the command's type and its size, using the following scheme:

```
typedef struct _LOAD_COMMAND {
    ULONG  cmd;      /* type of load command */
    ULONG  cmdsize;  /* total size of command in bytes */
} LOAD_COMMAND, *PLOAD_COMMAND;
```

Command type called `LC_SYMTAB`, contains raw pointers to two different tables. One, called `symoff`, with `NLIST` structures-based entries, and another, called `stroff`, with functions and variables names of each corresponding entry in the same order.

```
typedef struct _SYMTAB_COMMAND
{
   ULONG cmd;
   ULONG cmdsize;
```

```
    ULONG symoff;
    ULONG nsyms;
    ULONG stroff;
    ULONG strsize;
} SYMTAB_COMMAND, *PSYMTAB_COMMAND;

typedef struct _NLIST
{
    ULONG n_strx;
    UCHAR n_type;
    UCHAR n_sect;
    USHORT n_desc;
    ULONG n_value;
} NLIST, *PNLIST;
```

Both `symoff` and `stroff` are pointer into the *`__LINKEDIT`* segment. Please note we have to add `FAT_ARCH[x].offset` value to these fields. And `n_value` field from `NLIST` structure contains the symbol offset.

Here is a short dump of symbols retrieved from Mac OS X Leopard kernel.

```
[000000] .constructors_used                                          0x0050F254
[000001] .destructors_used                                           0x0050F25C
[000002] _AARPwakeup                                                 0x0029F6BD
[000003] _APTD                                                       0xFF7F8000
[000004] _APTDpde                                                    0xFEFF7FC0
[000005] _APTmap                                                     0xFF000000
[000006] _ASPgetmsg                                                  0x00293448
[000007] _ASPputmsg                                                  0x00293A3F
[000008] _ATPgetreq                                                  0x002A5A12
[000009] _ATPgetrsp                                                  0x002A5A79
[000010] _ATPsndreq                                                  0x002A592C
[000011] _ATPsndrsp                                                  0x002A599F
[000012] _ATgetmsg                                                   0x002A5844
[000013] _ATputmsg                                                   0x002A58B8
[..]
[000223] _IOZeroTvalspec                                             0x0050EE18
[000224] _IS_64BIT_PROCESS                                           0x00373952
[000225] _IdlePDPT                                                   0x004EB008
[000226] _IdlePDPT64                                                 0x004EB010
[000227] _IdlePML4                                                   0x004EB00C
[000228] _IdlePTD                                                    0x004EB004
[000229] _InitGlobals                                                0x002A3A08
[000230] _InsertKeyRecord                                            0x003477C8
[000231] _InsertOffset                                               0x003475A1
[000232] _InsertRecord                                               0x00347703
[..]
[002577] __ZN32IOServiceMessageUserNotificationC2EPK11OSMetaClass    0x004354AA
[002578] __ZN32IOServiceMessageUserNotificationC2Ev                  0x0043560E
[002579] __ZN32IOServiceMessageUserNotificationD0Ev                  0x0043553A
[002580] __ZN32IOServiceMessageUserNotificationD1Ev                  0x0043551A
[002581] __ZN32IOServiceMessageUserNotificationD2Ev                  0x004354FA
[002582] __ZN5IOCPU10gMetaClassE                                     0x0052E838
[002583] __ZN5IOCPU10superClassE                                     0x004BDE7C
[002584] __ZN5IOCPU11getCPUGroupEv                                   0x004300A2
[002585] __ZN5IOCPU11getCPUStateEv                                   0x00430082
[002586] __ZN5IOCPU11setCPUStateEm                                   0x0043008E
[002587] __ZN5IOCPU11setPropertyEPK8OSSymbolP8OSObject               0x0042FE68
[..]
```

```
[013859] _zt_ent_zindex                                              0x00299028
[013860] _zt_find_zname                                              0x00298D6D
[013861] _zt_getNextZone                                             0x0029910E
[013862] _zt_get_zmcast                                              0x00298F1D
[013863] _zt_remove_zones                                            0x00298CDD
[013864] _zt_set_zmap                                                0x002990B6
```

INFORMATION EXTRACTION (ALSO KNOW AS ANALYSIS)

Once memory manager is functional, we can now proceed to the extraction of information such as process list and so on.

MACHINE INFORMATION

Machine identification is a very important part to validate result. This section covers how to retrieve Darwin version, compilation date, number of CPUs and available memory on the current system.

There is a global variable, accessible from symbols, called `version` which contains a 100 bytes string with O.S. Type, O.S. Release version, username who compiled it.

There is another global variable, accessible from symbols, called `machine_info` defined by `machine_info` structure which contains information about CPUs and Memory of the target machine.

Definition of `machine_info` structure can be retrieved in *xnu/osfmk/mach/machine.h* header file.

Below is the definition of `machine_info` structure under Mac OS X Snow Leopard.

```c
struct machine_info {
  integer_t major_version; /* kernel major version id */
  integer_t minor_version; /* kernel minor version id */
  integer_t max_cpus; /* max number of CPUs possible */
  uint32_t memory_size; /* size of memory in bytes, capped at 2 GB */
  uint64_t max_mem; /* actual size of physical memory */
  uint32_t physical_cpu; /* number of physical CPUs now available */
  integer_t physical_cpu_max; /* max number of physical CPUs possible */
  uint32_t logical_cpu; /* number of logical cpu now available */
  integer_t logical_cpu_max; /* max number of physical CPUs possible */
};
```

```
Darwin Kernel Version 9.0.0: Tue Oct  9 21:35:55 PDT 2007; root:xnu-1228~1/RELEASE_I386
Major version:                9
Minor version:                0
Max number of CPUs:           4
Size of physical memory:      1024 MB
Number of physical CPUs:      0
Number of logical CPUs:       1
```

Above is a screenshot of extraction information showing the target machine is running Mac OS X Leopard 10.5.0 with 1GB of physical memory.

MOUNTED FILE SYSTEMS

Mounted file systems are defined by a global list-head, accessible from symbols, called `mountlist`. `mountlist` is a single link-list and contains a pointer called next which is a pointer to the next mounted file system entry both are defined by `mount` structure.

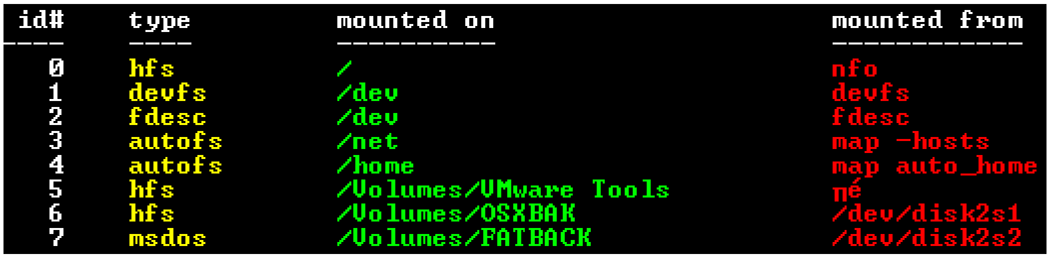
This structure contains 3 important fields including: file system type (`f_fstypename`), directory on which mounted (`f_mntonname`) and mounted file system (`f_mntfromname`).

Definition of `mount` structure can be retrieved in *xnu/bsd/sys/mount_internal.h* header file.

Below is the definition of `mount` structure under Mac OS X Snow Leopard.

```
/*
 * Structure per mounted file system.  Each mounted file system has an
 * array of operations and an instance record.  The file systems are
 * put on a doubly linked list.
 */

struct mount {
	TAILQ_ENTRY(mount) mnt_list;		/* mount list */
	int32_t		mnt_count;		/* reference on the mount */
	lck_mtx_t	mnt_mlock;		/* mutex that protects mount point */
	struct vfsops	*mnt_op;		/* operations on fs */
	struct vfstable	*mnt_vtable;		/* configuration info */
	struct vnode	*mnt_vnodecovered;/* vnode we mounted on */
	struct vnodelst	mnt_vnodelist;		/* list of vnodes this mount */
	struct vnodelst	mnt_workerqueue;	/* list of vnodes this mount */
	struct vnodelst	mnt_newvnodes;		/* list of vnodes this mount */
	uint32_t	mnt_flag;		/* flags */
	uint32_t	mnt_kern_flag;		/* kernel only flags */
	uint32_t	mnt_lflag;		/* mount life cycle flags */
	uint32_t	mnt_maxsymlinklen;/* max size of short symlink */
	struct vfsstatfs	mnt_vfsstat;	/* cache of filesystem stats */
	qaddr_t		mnt_data;	/* private data */
	/* Cached values of the IO constraints for the device */
	uint32_t	mnt_maxreadcnt;	/* Max. byte count for read */
	uint32_t	mnt_maxwritecnt;	/* Max. byte count for write */
	uint32_t	mnt_segreadcnt;	/* Max. segment count for read */
	uint32_t	mnt_segwritecnt;	/* Max. segment count for write */
	uint32_t	mnt_maxsegreadsize;	/* Max. segment read size  */
	uint32_t	mnt_maxsegwritesize;	/* Max. segment write size */
	uint32_t	mnt_alignmentmask; /* Mask of bits that aren't addressable 
via DMA */
	uint32_t	mnt_devblocksize; /* the underlying device block size */
	uint32_t	mnt_ioqueue_depth; /* the maxiumum number of commands a 
device can accept */
	  uint32_t  mnt_ioscale; /* scale the various throttles/limits imposed 
on the amount of I/O in flight */
	uint32_t	mnt_ioflags;	/* flags for  underlying device */
	pending_io_t mnt_pending_write_size; /* byte count of pending writes */
	pending_io_t mnt_pending_read_size; /* byte count of pending reads */

	lck_rw_t mnt_rwlock;	/* mutex readwrite lock */
	lck_mtx_t mnt_renamelock; /* mutex that serializes renames that change 
shape of tree */
	vnode_t mnt_devvp; /* the device mounted on for local file systems */
```

```
	uint32_t	mnt_devbsdunit; /* the BSD unit number of the device */
	void	*mnt_throttle_info; /* used by the throttle code */
	int32_t mnt_crossref; /* refernces to cover lookups  crossing into mp
*/
	int32_tmnt_iterref; /* refernces to cover iterations; drained makes it
-ve  */

	/* XXX 3762912 hack to support HFS filesystem 'owner' */
	uid_t		mnt_fsowner;
	gid_t		mnt_fsgroup;

	struct label	*mnt_mntlabel;		/* MAC mount label */
	struct label	*mnt_fslabel;		/* MAC default fs label */

	/*
	 * cache the rootvp of the last mount point
	 * in the chain in the mount struct pointed
	 * to by the vnode sitting in '/'
	 * this cache is used to shortcircuit the
	 * mount chain traversal and allows us
	 * to traverse to the true underlying rootvp
	 * in 1 easy step inside of 'cache_lookup_path'
	 *
	 * make sure to validate against the cached vid
	 * in case the rootvp gets stolen away since
	 * we don't take an explicit long term reference
	 * on it when we mount it
	 */
	vnode_t		mnt_realrootvp;
	uint32_t	mnt_realrootvp_vid;
	/*
	 * bumped each time a mount or unmount
	 * occurs... its used to invalidate
	 * 'mnt_realrootvp' from the cache
	 */
	uint32_t		mnt_generation;
	  /*
	 * if 'MNTK_AUTH_CACHE_TIMEOUT' is
	 * set, then 'mnt_authcache_ttl' is
	 * the time-to-live for the per-vnode authentication cache
	 * on this mount... if zero, no cache is maintained...
	 * if 'MNTK_AUTH_CACHE_TIMEOUT' isn't set, its the
	 * time-to-live for the cached lookup right for
	 * volumes marked 'MNTK_AUTH_OPAQUE'.
	 */
	int		mnt_authcache_ttl;
	/*
	 * The proc structure pointer and process ID form a
	 * sufficiently unique duple identifying the process
	 * hosting this mount point. Set by vfs_markdependency()
	 * and utilized in new_vnode() to avoid reclaiming vnodes
	 * with this dependency (radar 5192010).
	 */
	pid_t		mnt_dependent_pid;
	void		*mnt_dependent_process;
};
```



id#	type	mounted on	mounted from
0	hfs	/	nfo
1	devfs	/dev	devfs
2	fdesc	/dev	fdesc
3	autofs	/net	map -hosts
4	autofs	/home	map auto_home
5	hfs	/Volumes/VMware Tools	né
6	hfs	/Volumes/OSXBAK	/dev/disk2s1
7	msdos	/Volumes/FATBACK	/dev/disk2s2

Above is a screenshot of mounted file systems including an external hard-drive.

BSD Processes

Every Operating System uses user-land processes, it is one of the key element of a working O.S.

Loaded processes are stored into `proc` structure which contains a double-list to walk into the list. There is a global variable, retrievable from symbols, called `kernproc` is the list-head of BSD processes list.

`p_list` field is a double link-list which contains a pointer to both, the previous and the next process.

Definition of `proc` structure can be retrieved in *xnu/bsd/sys/proc_internal.h* header file.

Below is the definition of `proc` structure under Mac OS X Snow Leopard.

```
/*
 * Description of a process.
 *
 * This structure contains the information needed to manage a thread of
 * control, known in UN*X as a process; it has references to substructures
 * containing descriptions of things that the process uses, but may share
 * with related processes.  The process structure and the substructures
 * are always addressible except for those marked "(PROC ONLY)" below,
 * which might be addressible only on a processor on which the process
 * is running.
 */
struct      proc {
      LIST_ENTRY(proc) p_list;      /* List of all processes. */

      pid_t       p_pid;            /* Process identifier. (static)*/
      void *            task;       /* corresponding task (static)*/
      struct      proc *p_pptr;     /* Pointer to parent process.(LL) */
      pid_t       p_ppid;                 /* process's parent pid number */
      pid_t       p_pgrpid;         /* process group id of the process (LL)*/

      lck_mtx_t   p_mlock;          /* mutex lock for proc */

      char        p_stat;                 /* S* process status. (PL)*/
      char        p_shutdownstate;
      char        p_kdebug;         /* P_KDEBUG eq (CC)*/
      char        p_btrace;         /* P_BTRACE eq (CC)*/

      LIST_ENTRY(proc) p_pglist;    /* List of processes in pgrp.(PGL) */
      LIST_ENTRY(proc) p_sibling;   /* List of sibling processes. (LL)*/
      LIST_HEAD(, proc) p_children; /* Pointer to list of children. (LL)*/
      TAILQ_HEAD( , uthread) p_uthlist;   /* List of uthreads  (PL) */
```

```
	LIST_ENTRY(proc) p_hash;		/* Hash chain. (LL)*/
	TAILQ_HEAD( ,eventqelt) p_evlist;	/* (PL) */

	lck_mtx_t	p_fdmlock;	/* proc lock to protect fdesc */

	/* substructures: */
	kauth_cred_t	p_ucred;	/* Process owner's identity. (PL) */
	struct	filedesc *p_fd;	/* Ptr to open files structure. (PFDL) */
	struct	pstats *p_stats;	/* Accounting/statistics (PL). */
	struct	plimit *p_limit;	/* Process limits.(PL) */

	struct	sigacts *p_sigacts;	/* Signal actions, state (PL) */
	 int		p_siglist;	/* signals captured back from threads */
	lck_spin_t	p_slock;	/* spin lock for itimer/profil protection */

#define	p_rlimit	p_limit->pl_rlimit

	struct	plimit *p_olimit; /* old process limits  - not inherited by 
child  (PL) */
	unsigned int	p_flag;	/* P_* flags. (atomic bit ops) */
	unsigned int	p_lflag;	/* local flags  (PL) */
	unsigned int	p_listflag; /* list flags (LL) */
	unsigned int	p_ladvflag; /* local adv flags (atomic) */
	int	p_refcount; /* number of outstanding users(LL) */
	int	p_childrencnt; /* children holding ref on parent (LL) */
	int	p_parentref; /* children lookup ref on parent (LL) */

	pid_t p_oppid; /* Save parent pid during ptrace. XXX */
	u_int p_xstat; /* Exit status for wait; also stop signal. */

#ifdef _PROC_HAS_SCHEDINFO_
	/* may need cleanup, not used */
	u_int p_estcpu; /* Time averaged value of p_cpticks.(used by aio and 
proc_comapre) */
	fixpt_t p_pctcpu; /* %cpu for this process during p_swtime (used by 
aio)*/
	u_int p_slptime;  /* used by proc_compare */
#endif /* _PROC_HAS_SCHEDINFO_ */

	struct	itimerval p_realtimer;	/* Alarm timer. (PSL) */
	struct	timeval p_rtime;	/* Real time.(PSL)  */
	struct	itimerval p_vtimer_user; /* Virtual timers.(PSL)  */
	struct	itimerval p_vtimer_prof; /* (PSL) */

	struct timeval p_rlim_cpu; /* Remaining rlim cpu value.(PSL) */
	int	p_debugger; /*  NU 1: can exec set-bit programs if suser */
	boolean_tsigwait; /* indication to suspend (PL) */
	void	*sigwait_thread;	/* 'thread' holding sigwait(PL)  */
	void	*exit_thread; /* Which thread is exiting(PL)  */
	int	p_vforkcnt; /* number of outstanding vforks(PL)  */
	void *  p_vforkact;  /* activation running this vfork proc)(static)  */
	int	p_fpdrainwait;	/* (PFDL) */
	pid_t p_contproc; /* last PID to send us a SIGCONT (PL) */

	/* Following fields are info from SIGCHLD (PL) */
	pid_t si_pid;	/* (PL) */
```

```
	u_int	si_status;/* (PL) */
	u_int si_code;	/* (PL) */
	uid_t si_uid;	/* (PL) */

	void * vm_shm; /* (SYSV SHM Lock) for sysV shared memory */

#if CONFIG_DTRACE
	user_addr_t p_dtrace_argv; /* (write once, read only after that) */
	user_addr_t p_dtrace_envp; /* (write once, read only after that) */
	lck_mtx_t	p_dtrace_sprlock; /* sun proc lock emulation */
	int	p_dtrace_probes;	/* (PL) are there probes for this proc? */
	u_int p_dtrace_count;	/* (sprlock) number of DTrace tracepoints */
	struct dtrace_ptss_page* p_dtrace_ptss_pages; /* (sprlock) list of user
ptss pages */
	struct dtrace_ptss_page_entry* p_dtrace_ptss_free_list; /* (atomic)
list of individual ptss entries */
	struct dtrace_helpers* p_dtrace_helpers;  /* (dtrace_lock) DTrace per-
proc private */
	struct dof_ioctl_data*p_dtrace_lazy_dofs; /* (sprlock) unloaded
dof_helper_t's */
#endif /* CONFIG_DTRACE */

/* XXXXXXXXXXXXX BCOPY'ed on fork XXXXXXXXXXXXXXXX */
/* The following fields are all copied upon creation in fork. */
#define	p_startcopy p_argslen

	u_int p_argslen;	/* Length of process arguments. */
	int	p_argc;			/* saved argc for sysctl_procargs() */
	user_addr_t user_stack;		/* where user stack was allocated */
	struct	vnode *p_textvp;	/* Vnode of executable. */
	off_t p_textoff;		/* offset in executable vnode */

	sigset_t p_sigmask;		/* DEPRECATED */
	sigset_t p_sigignore;	/* Signals being ignored. (PL) */
	sigset_t p_sigcatch;	/* Signals being caught by user.(PL)  */

	u_char	p_priority; /* (NU) Process priority. */
	u_char	p_resv0;	/* (NU) User-priority based on p_cpu and
p_nice. */
	char	p_nice;		/* Process "nice" value.(PL) */
	u_char	p_resv1;/* (NU) User-priority based on p_cpu and p_nice. */

#if CONFIG_MACF
	int	p_mac_enforce;/* MAC policy enforcement control */
#endif

	char	p_comm[MAXCOMLEN+1];
	char	p_name[(2*MAXCOMLEN)+1];/* PL */

	struct	pgrp *p_pgrp;	/* Pointer to process group. (LL) */
	int		p_iopol_disk;	/* disk I/O policy (PL) */
	uint32_t	p_csflags;	/* flags for codesign (PL) */
	uint32_t	p_pcaction; /* action  for process control on starvation */
	uint8_t p_uuid[16];		/* from LC_UUID load command */

/* End area that is copied on creation. */
/* XXXXXXXXXXXXX End of BCOPY'ed on fork (AIOLOCK)XXXXXXXXXXXXXXXX */
```

```
#define     p_endcopy   p_aio_total_count
      int           p_aio_total_count;/* all allocated AIO requests for this
proc */
      int           p_aio_active_count;/* all unfinished AIO requests for this
proc */
      TAILQ_HEAD( , aio_workq_entry ) p_aio_activeq;  /* active async IO
requests */
      TAILQ_HEAD( , aio_workq_entry ) p_aio_doneq;    /* completed async IO
requests */

      struct klist p_klist;  /* knote list (PL ?)*/

      struct      rusage *p_ru;/* Exit information. (PL) */
      thread_t    p_signalholder;
      thread_t    p_transholder;

      /* DEPRECATE following field  */
      u_short     p_acflag;   /* Accounting flags. */

      struct lctx *p_lctx;    /* Pointer to login context. */
      LIST_ENTRY(proc) p_lclist;    /* List of processes in lctx. */
      user_addr_t       p_threadstart;          /* pthread start fn */
      user_addr_t       p_wqthread; /* pthread workqueue fn */
      int   p_pthsize;        /* pthread size */
      user_addr_t p_targconc; /* target concurrency ptr */
      void *      p_wqptr;    /* workq ptr */
      int   p_wqsize;         /* allocated size */
      boolean_t       p_wqiniting;  /* semaphore to serialze wq_open */
      lck_spin_t  p_wqlock;         /* lock to protect work queue */
      struct  timeval p_start;      /* starting time */
      void *      p_rcall;
      int         p_ractive;
      int   p_idversion;      /* version of process identity */
      void *      p_pthhash;  /* pthread waitqueue hash */
#if DIAGNOSTIC
      unsigned int p_fdlock_pc[4];
      unsigned int p_fdunlock_pc[4];
#if SIGNAL_DEBUG
      unsigned int lockpc[8];
      unsigned int unlockpc[8];
#endif /* SIGNAL_DEBUG */
#endif /* DIAGNOSTIC */
      uint64_t    p_dispatchqueue_offset;
};
```

Pointer to the process group, `pgrp` structure, allows us to retrieve the username of the person who launched the program because this structure contains a pointer to a structure called `session` with the username.

Definition of `pgrp` structure can also be retrieved in xnu/bsd/sys/proc_internal.h header file.

Below is the definition of `pgrp` structure under Mac OS X Snow Leopard.

```
    /*
     * One structure allocated per process group.
     */
    struct pgrp {
```

```
        LIST_ENTRY(pgrp) pg_hash; /* Hash chain. (LL) */
        LIST_HEAD(, proc) pg_members; /* Pointer to pgrp members. (PGL) */
        struct session *pg_session; /* Pointer to session. (LL ) */
        pid_t pg_id; /* Pgrp id. (static) */
        int pg_jobc; /* # procs qualifying pgrp for job control (PGL) */
        int pg_membercnt; /* Number of processes in the pgrocess group (PGL) */
        int pg_refcount; /* number of current iterators (LL) */
        unsigned int pg_listflags; /* (LL) */
        lck_mtx_t pg_mlock; /* mutex lock to protect pgrp */
    };
```

Definition of `session` structure can also be retrieved in xnu/bsd/sys/proc_internal.h header file.

Below is the definition of `session` structure under Mac OS X Snow Leopard.

```
/*
 * One structure allocated per session.
 */
struct session {
        int    s_count;     /* Ref cnt; pgrps in session. (LL) */
        struct proc *s_leader;  /* Session leader.(static) */
        struct vnode *s_ttyvp;  /* Vnode of controlling terminal.(SL) */
        int    s_ttyvid; /* Vnode id of the controlling terminal (SL) */
        struct tty *s_ttyp; /* Controlling terminal. (SL + ttyvp != NULL) */
        pid_t s_ttypgrpid;              /* tty's pgrp id */
        pid_t s_sid;                    /* Session ID (static) */
        char   s_login[MAXLOGNAME];     /* Setlogin() name.(SL) */
        int    s_flags;                 /* Session flags (s_mlock)  */
        LIST_ENTRY(session) s_hash;     /* Hash chain.(LL) */
        lck_mtx_t    s_mlock;           /* mutex lock to protect session */
        int    s_listflags;
};
```

`s_login` field contains the name of the username in ASCII.

task#	pid	parent pid	name	username	started time
1	0	0	kernel_task		Thu 2009-March-26 12:44:43 (W. Europe Standard Time)
2	1	0	launchd	nfinfi	Thu 2009-March-26 12:44:43 (W. Europe Standard Time)
3	10	1	kextd	root	Thu 2009-March-26 12:44:45 (W. Europe Standard Time)
4	11	1	notifyd	root	Thu 2009-March-26 12:44:45 (W. Europe Standard Time)
5	12	1	syslogd	root	Thu 2009-March-26 12:44:46 (W. Europe Standard Time)
6	14	1	ntpd	root	Thu 2009-March-26 12:44:47 (W. Europe Standard Time)
7	16	1	update	root	Thu 2009-March-26 12:44:47 (W. Europe Standard Time)
8	19	1	securityd	root	Thu 2009-March-26 12:44:47 (W. Europe Standard Time)
9	21	1	mds	root	Thu 2009-March-26 12:44:47 (W. Europe Standard Time)
10	22	1	mDNSResponder		Thu 1970-January-01 01:00:00 (W. Europe Standard Time)
11	23	1	loginwindow	nfinfi	Thu 2009-March-26 12:44:47 (W. Europe Standard Time)
12	24	1	KernelEventAgent	root	Thu 2009-March-26 12:44:47 (W. Europe Standard Time)
13	26	1	hidd	root	Thu 2009-March-26 12:44:47 (W. Europe Standard Time)
14	27	1	fseventsd	root	Thu 2009-March-26 12:44:47 (W. Europe Standard Time)
15	28	1	dynamic_pager	root	Thu 2009-March-26 12:44:47 (W. Europe Standard Time)
16	31	1	diskarbitrationd	root	Thu 2009-March-26 12:44:47 (W. Europe Standard Time)
17	32	1	DirectoryService	root	Thu 2009-March-26 12:44:47 (W. Europe Standard Time)
18	34	1	configd	root	Thu 2009-March-26 12:44:47 (W. Europe Standard Time)
19	37	1	autofsd	root	Thu 2009-March-26 12:44:47 (W. Europe Standard Time)
20	38	1	socketfilterfw	root	Thu 2009-March-26 12:44:47 (W. Europe Standard Time)
21	40	1			Thu 1970-January-01 01:00:00 (W. Europe Standard Time)
22	46	1	coreservicesd	_securityagent	Thu 2009-March-26 12:44:51 (W. Europe Standard Time)
23	48	1	WindowServer	root	Thu 2009-March-26 12:44:51 (W. Europe Standard Time)
24	59	1	launchd	nfinfi	Thu 2009-March-26 12:44:53 (W. Europe Standard Time)
25	71	1	coreaudiod	nobody	Thu 2009-March-26 12:45:03 (W. Europe Standard Time)
26	78	59	Spotlight	nfinfi	Thu 2009-March-26 12:45:04 (W. Europe Standard Time)
27	79	59	UserEventAgent	nfinfi	Thu 2009-March-26 12:45:04 (W. Europe Standard Time)
28	80	59	Dock	nfinfi	Thu 2009-March-26 12:45:04 (W. Europe Standard Time)
29	81	59	SystemUIServer	nfinfi	Thu 2009-March-26 12:45:04 (W. Europe Standard Time)
30	82	59	Finder	nfinfi	Thu 2009-March-26 12:45:04 (W. Europe Standard Time)
31	83	59	ATSServer	nfinfi	Thu 2009-March-26 12:45:04 (W. Europe Standard Time)
32	85	59	pboard	nfinfi	Thu 2009-March-26 12:45:04 (W. Europe Standard Time)
33	96	14	ntpd	root	Thu 2009-March-26 12:46:10 (W. Europe Standard Time)
34	97	59	Terminal	nfinfi	Thu 2009-March-26 12:50:47 (W. Europe Standard Time)
35	98	97	login	nfinfi	Thu 2009-March-26 12:50:47 (W. Europe Standard Time)
36	99	98	bash	nfinfi	Thu 2009-March-26 12:50:48 (W. Europe Standard Time)
37	128	59	Preview	nfinfi	Thu 2009-March-26 12:56:36 (W. Europe Standard Time)
38	211	59	Xcode	nfinfi	Thu 2009-March-26 12:59:32 (W. Europe Standard Time)
39	228	97	login	nfinfi	Thu 2009-March-26 13:30:08 (W. Europe Standard Time)

Above is a sample screenshot of a processes list, mainly executed by nfinfi user around March 2009.

Kernel Extensions (Also Known As Drivers, Kernel Modules)

Kernel-Mode, the God Mode, is the most privileged level of an Operating System. Loaded Kernel Extensions can be retrieved by a global list-head variable, accessible from symbols, called `kmod` defined by `kmod_info` structure.

`next` field points to the next kernel extension.

Definition of `session` structure can also be retrieved in xnu/osfmk/mach/kmod.h h header file.

Below is the definition of `kmod_info` structure under Mac OS X Snow Leopard.

```
typedef struct kmod_info {
  struct kmod_info *next;
  int info_version; // version of this structure
  int id;
  char name[KMOD_MAX_NAME];
  char version[KMOD_MAX_NAME];
  int reference_count; // # refs to this
  kmod_reference_t *reference_list; // who this refs
  vm_address_t address; // starting address
  vm_size_t size; // total size
  vm_size_t hdr_size; // unwired hdr size
   kmod_start_func_t *start;
   kmod_stop_func_t *stop;
} kmod_info_t;
```

As you can see here we have both kernel extensions image base start and size. Since we have a functional kernel address space, we can easily extract the image of the kernel extension.

```
id    ref  address    size       code size  name (version)
--    ---  -------    ----       ---------  --------------
 106    0 0x226F0000 0x00003000 0x00002000 com.apple.driver.iTunesPhoneDriver (1.0)
 104    0 0x226BD000 0x0000A000 0x00009000 com.apple.iokit.IOUSBMassStorageClass (2.0.0)
 102    0 0x226D1000 0x0001B000 0x0001A000 com.apple.filesystems.ntfs (2.0)
  97    0 0x225ED000 0x0000C000 0x0000B000 com.apple.filesystems.msdosfs (1.5)
  94    0 0x225CA000 0x00002000 0x00001000 com.osxbook.kext.KernelMemoryAccess (1.0.0)
  93    0 0x00AC1000 0x00005000 0x00004000 com.apple.driver.AppleHWSensor (1.7.0d0)
  92    0 0x00D58000 0x0000B000 0x0000A000 com.apple.filesystems.autofs (2.0.0d1)
  91    0 0x00D35000 0x00023000 0x00022000 com.apple.driver.AppleHDA (1.4.0a23)
  90    1 0x00CF4000 0x00041000 0x00040000 com.apple.driver.DspFuncLib (1.0.0a1)
  89    0 0x00656000 0x00003000 0x00002000 com.apple.Dont_Steal_Mac_OS_X (6.0.2)
  88    0 0x00DC6000 0x00007000 0x00006000 com.apple.iokit.CHUDUtils (200)
  87    0 0x00C21000 0x0000C000 0x0000B000 com.apple.iokit.CHUDProf (207)
  86    0 0x00DCD000 0x00082000 0x00081000 com.apple.GeForce (5.1.6)
  85    2 0x00C1C000 0x00005000 0x00004000 com.apple.iokit.CHUDKernLib (196)
  84    0 0x00909000 0x001B8000 0x001B7000 com.apple.nvidia.nv40hal (5.1.6)
  83    2 0x006DC000 0x0022D000 0x0022C000 com.apple.NVDAResman (5.1.6)
  82    2 0x006CE000 0x0000E000 0x0000D000 com.apple.iokit.IONDRVSupport (1.5)
  81    3 0x006B2000 0x0001C000 0x0001B000 com.apple.iokit.IOGraphicsFamily (1.5)
  80    0 0x00C90000 0x00007000 0x00006000 com.apple.driver.AppleHDAController (1.4.0a23)
  79    2 0x00C8A000 0x00006000 0x00005000 com.apple.iokit.IOHDAFamily (1.4.0a23)
  78    0 0x00C03000 0x00009000 0x00008000 com.apple.iokit.IOFireWireIP (1.7.0)
  77    0 0x0069F000 0x00003000 0x00002000 com.apple.driver.AppleUSBDisplays (2.0)
  76    0 0x00E4F000 0x00004000 0x00003000 com.apple.driver.AudioIPCDriver (1.0.4)
  75    3 0x00CDD000 0x00017000 0x00016000 com.apple.iokit.IOAudioFamily (1.6.4b7)
  74    1 0x00CDA000 0x00003000 0x00002000 com.apple.kext.OSvKernDSPLib (1.1)
  73    0 0x00AC6000 0x0000A000 0x00009000 com.apple.driver.AppleMCEDriver (1.1.5b4)
  72    0 0x00C71000 0x0000A000 0x00009000 com.apple.driver.ACPI_SMC_PlatformPlugin (3.0.0d11)
  71    1 0x00C63000 0x0000E000 0x0000D000 com.apple.driver.IOPlatformPluginFamily (3.0.0d11)
  70    2 0x0064D000 0x00009000 0x00008000 com.apple.driver.AppleSMC (2.0.0d5)
  69    0 0x00B0A000 0x00003000 0x00002000 com.apple.driver.AppleLPC (1.2.2)
  68    0 0x00699000 0x00003000 0x00002000 com.apple.driver.AppleUSBHIDMouse (1.2.0b3)
  67    1 0x00691000 0x00003000 0x00002000 com.apple.driver.AppleHIDMouse (1.2.0b3)
  66    2 0x00694000 0x00005000 0x00004000 com.apple.iokit.IOUSBHIDDriver (3.0.3)
  65    0 0x00647000 0x00002000 0x00001000 com.apple.driver.AppleUSBMergeNub (3.0.3)
  64    0 0x00C4D000 0x00004000 0x00003000 com.apple.driver.AppleUSBComposite (3.0.0)
  62    0 0x00BD4000 0x00016000 0x00015000 com.apple.iokit.IOSCSIMultimediaCommandsDevice (2.0.0)
  61    2 0x00BBE000 0x00016000 0x00015000 com.apple.iokit.IOSCSIBlockCommandsDevice (2.0.0)
  60    1 0x00BB9000 0x00005000 0x00004000 com.apple.iokit.IOBDStorageFamily (1.5)
  59    2 0x00BB3000 0x00006000 0x00005000 com.apple.iokit.IODVDStorageFamily (1.5)
  58    3 0x00BAB000 0x00008000 0x00007000 com.apple.iokit.IOCDStorageFamily (1.5)
  57    0 0x00B04000 0x00006000 0x00005000 com.apple.iokit.SCSITaskUserClient (2.0.0)
  56    0 0x00C97000 0x00005000 0x00004000 com.apple.driver.XsanFilter (2.7.91)
```

Above is a screenshot of a loaded kext lists.

System Calls

The very first step is to localize the syscall table, called `sysent`, which is a non accessible variable from symbols. So using a magic trick we can retrieve its offset through `nsysent` exported variable which contains the number of syscall entries.

Under Mac OS X Leopard (10.5), as explained by Jesse D'Aguanno at BH US 2008, we have to add 0x20 to nsysent offset to obtain the offset of sysent table.

Under Mac OS X Snow Leopard (10.6), we have to proceed with a different methodology. First, we have to retrieve the value of `nsysent` variable, then we multiply its value with the size of `sysent` structure, and then we subtract this value to `nsysent` offset to obtain the offset of `sysent` table.

Definition of `sysent` structure can also be retrieved in xnu/bsd/sys/sysent.h header file.

Below is the definition of `sysent` structure under Mac OS X Snow Leopard.

```
struct sysent {          /* system call table */
      int16_t sy_narg;   /* number of args */
      int8_t sy_resv;    /* reserved  */
      int8_t sy_flags;   /* flags */
      sy_call_t   *sy_call;    /* implementing function */
      sy_munge_t *sy_arg_munge32; /* system call arguments munger for 32-bit
process */
```

```
        sy_munge_t  *sy_arg_munge64; /* system call arguments munger for 64-bit
process */
        int32_t sy_return_type; /* system call return types */
        uint16_t sy_arg_bytes;  /* Total size of arguments in bytes for
                                 * 32-bit system calls
                                 */
};
```

id#	offset	name	table
0	0x003907F5	_nosys	[OK]
1	0x00376F34	_exit	[OK]
2	0x00378B4A	_fork	[OK]
3	0x00390CAE	_read	[OK]
4	0x0039134C	_write	[OK]
5	0x001E425C	_open	[OK]
6	0x0036C75E	_close	[OK]
7	0x00375EB2	_wait4	[OK]
8	0x003907F5	_nosys	[OK]
9	0x001E4932	_link	[OK]
10	0x001E5540	_unlink	[OK]
11	0x003907F5	_nosys	[OK]
12	0x001E3925	_chdir	[OK]
13	0x001E3723	_fchdir	[OK]
14	0x001E43E8	_mknod	[OK]
15	0x001E6FD1	_chmod	[OK]
16	0x001E74B7	_chown	[OK]
17	0x0037A52D	_obreak	[OK]
18	0x001E335E	_getfsstat	[OK]
19	0x003907F5	_nosys	[OK]
20	0x0037DE30	_getpid	[OK]
21	0x003907F5	_nosys	[OK]
22	0x003907F5	_nosys	[OK]
23	0x0037E92E	_setuid	[OK]
24	0x0037DF0D	_getuid	[OK]
25	0x0037DF21	_geteuid	[OK]
26	0x0038C823	_ptrace	[OK]
27	0x003B0A4E	_recvmsg	[OK]
28	0x003B1701	_sendmsg	[OK]
29	0x003B07D8	_recvfrom	[OK]
30	0x003AFE73	_accept	[OK]
31	0x003B0EC4	_getpeername	[OK]
32	0x003B0CDA	_getsockname	[OK]
33	0x001E5D2D	_access	[OK]
34	0x001E6BD7	_chflags	[OK]
35	0x001E6C88	_fchflags	[OK]
36	0x001E22B5	_sync	[OK]
37	0x003836B2	_kill	[OK]
38	0x003907F5	_nosys	[OK]
39	0x0037DE42	_getppid	[OK]
40	0x003907F5	_nosys	[OK]
41	0x0036E487	_dup	[OK]
42	0x00394912	_pipe	[OK]
43	0x0037DFC7	_getegid	[OK]
44	0x0038FBA6	_profil	[OK]
45	0x003907F5	_nosys	[OK]
46	0x00382075	_sigaction	[OK]
47	0x0037DFB3	_getgid	[OK]
48	0x003829F2	_sigprocmask	[OK]
49	0x0037E544	_getlogin	[OK]
50	0x0037E5E5	_setlogin	[OK]
51	0x003582A7	_acct	[OK]
52	0x00381125	_sigpending	[OK]
53	0x00381539	_sigaltstack	[OK]
54	0x0039160C	_ioctl	[OK]
55	0x0038C732	_reboot	[OK]
56	0x001E9F24	_revoke	[OK]
57	0x001E4E09	_symlink	[OK]
58	0x001E6923	_readlink	[OK]

Above is a picture showing a list of syscalls from `sysent` table.

Integrity checks are done if entry value does not give the function name value. It does not sound complicated but this trick was enough to detect Jesse D'Aguanno Rootkit presented at HAR2009.

THANKS

For help, resources, etc.

- Dino Dai Zovi
- Vincenzo Iozzo, Zynamics
- Neil Archibald (nemo)
- Ruud van Baar, NFI
- Feico Dillema, NFI
- Raoul Bhoedjang, NFI