Honeypot Technology

Thorsten Holz

Laboratory for Dependable Distributed Systems

tho@koeln.ccc.de

Overview

1. Introduction
2. Honeynets
   - Overview
   - Setup and tools used
   - Preliminary results
3. NoSEBrEaK

Honeynets

"Suppose," he [Winnie the Pooh] said to Piglet, "you wanted to catch me, how would you do it?"
"Well," said Piglet, "I should do it like this: I should make a trap, and I should put a jar of honey in the trap, and you would smell it, and you would go in after it, and . . . "

A. A. Milne: Winnie the Pooh

Honeypots?

- Electronic bait, i.e. network ressources (e.g. computers, routers, switches, ...) deployed to be probed, attacked and compromised
- Lure in attackers and watch them as they exploit vulnerabilities
- Monitoring software permanently collects data, helps in post-incident forensics
- Clifford Stoll: *The Cuckoo's Egg*, 1988

Global Honeynet Project

The Honeynet PROJECT™

- Non-profit research organization of security professionals dedicated to information security
- “Learn the tools, tactics, and motives of the blackhat community and share these lessons learned”
- Aims of Project:
  - Raise awareness
  - Teach and Inform
  - Research on old and new attacking techniques
  - Active defense

Global Honeynet Project

- **Development of tools, for example monitoring software like *Sebek* or software for data analysis**
- **Experiences up to now:**
  - **Capturing of exploits and tools, e.g. exploit for known vulnerability (`dtspcd`, 2002)**
  - **Typical approach of attackers**
  - **Monitoring of conversations over IRC Botnets, organized card fraud, . . .**

Further information: honeynet.org

Sebek

- Kernel-module on Linux, Solaris, patch on OpenBSD / NetBSD, device driver for Window$
- Tries to capture all activities of an attacker
- Hijacks `sys_read` (access to SSH sessions, `burneye`-protected programs, . . .)
- Direct communication to ethernet driver, therefore mostly stealth
- Unlinking from module list to hide its presence

“Honeywall” and further monitoring

- **Transparent bridge, used for data capture and data control**
- **IDS `snort` / IPS `snort_inline` (now part of `snort`)**

```
alert ip $HONEYNET any -> $EXTERNAL_NET any
 (msg:"SHELLCODE x86 stealth NOOP"; rev:6; sid:651;
      content:"|EB 02 EB 02 EB 02|";
      replace:"|24 00 99 DE 6C 3E|";)
```

- **`netfilter/iptables` for traffic limiting**
- **Further monitoring**
  - `monit` or `supervise`
  - `swatch`

honeyd

- “Low interaction honeypot” in contrast to “high interaction honeypots” like described before
- Written by Niels Provos
- Daemon that creates virtual hosts on a network, virtualization of IP stack
- Hosts can be configured to run arbitrary services, and personality can be adapted
- Uses `nmap`, `xprobe` and `p0f` fingerprints to emulate hosts

honeyd

- arpd or static routes to route traffic into honeyd network
- Offered “services” are scripts or proxy <ip>:<port>

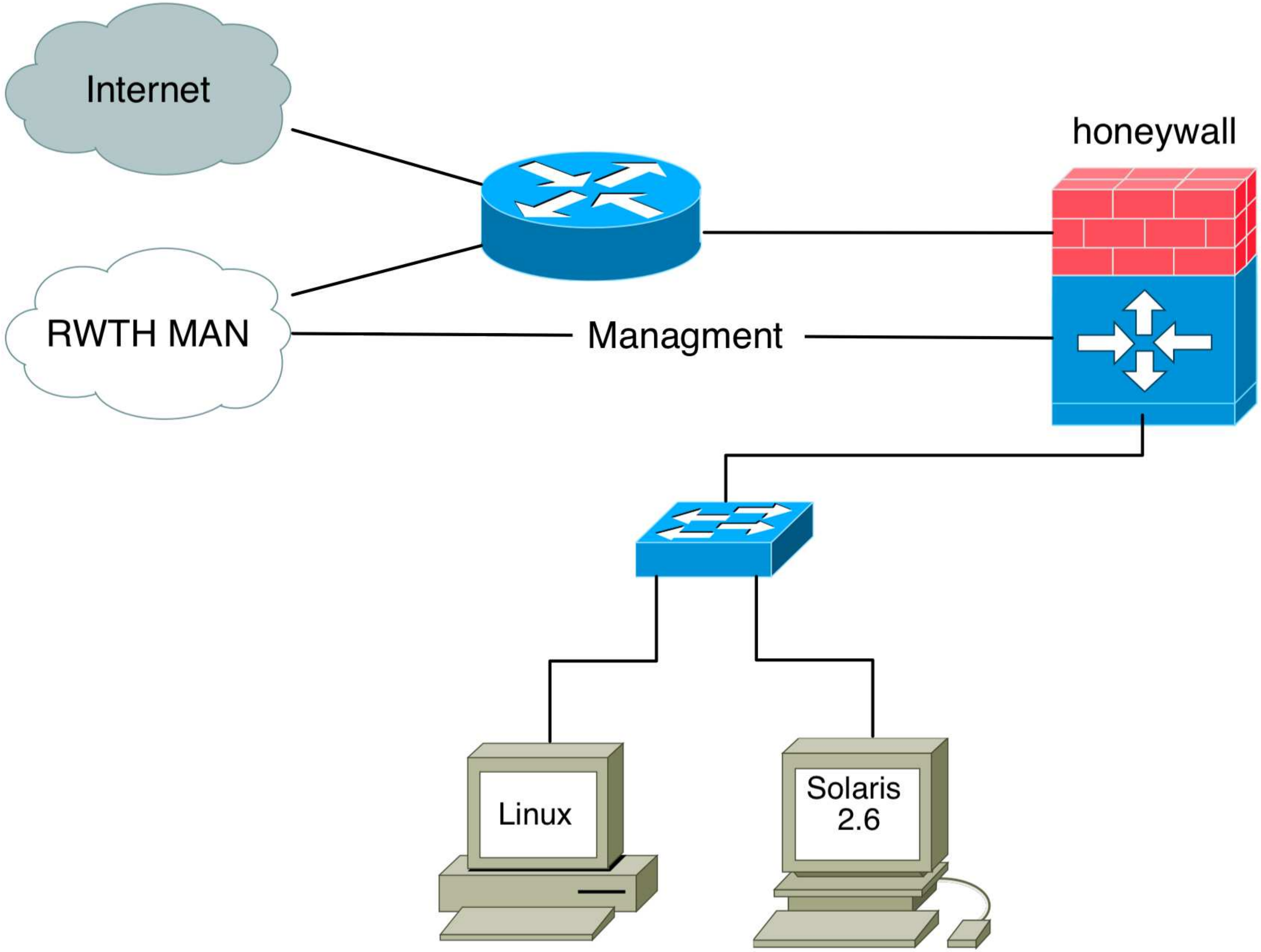
```
# Example of a simple host template and its binding
create template
set template personality "AIX 4.0 - 4.2"
add template tcp port 80 "sh scripts/web.sh"
add template tcp port 22 "sh scripts/ssh.sh $src $port"
add template tcp port 23 proxy 10.23.1.2:23
set template default tcp action reset

bind 10.21.19.102 template
```

Operating System Distribution

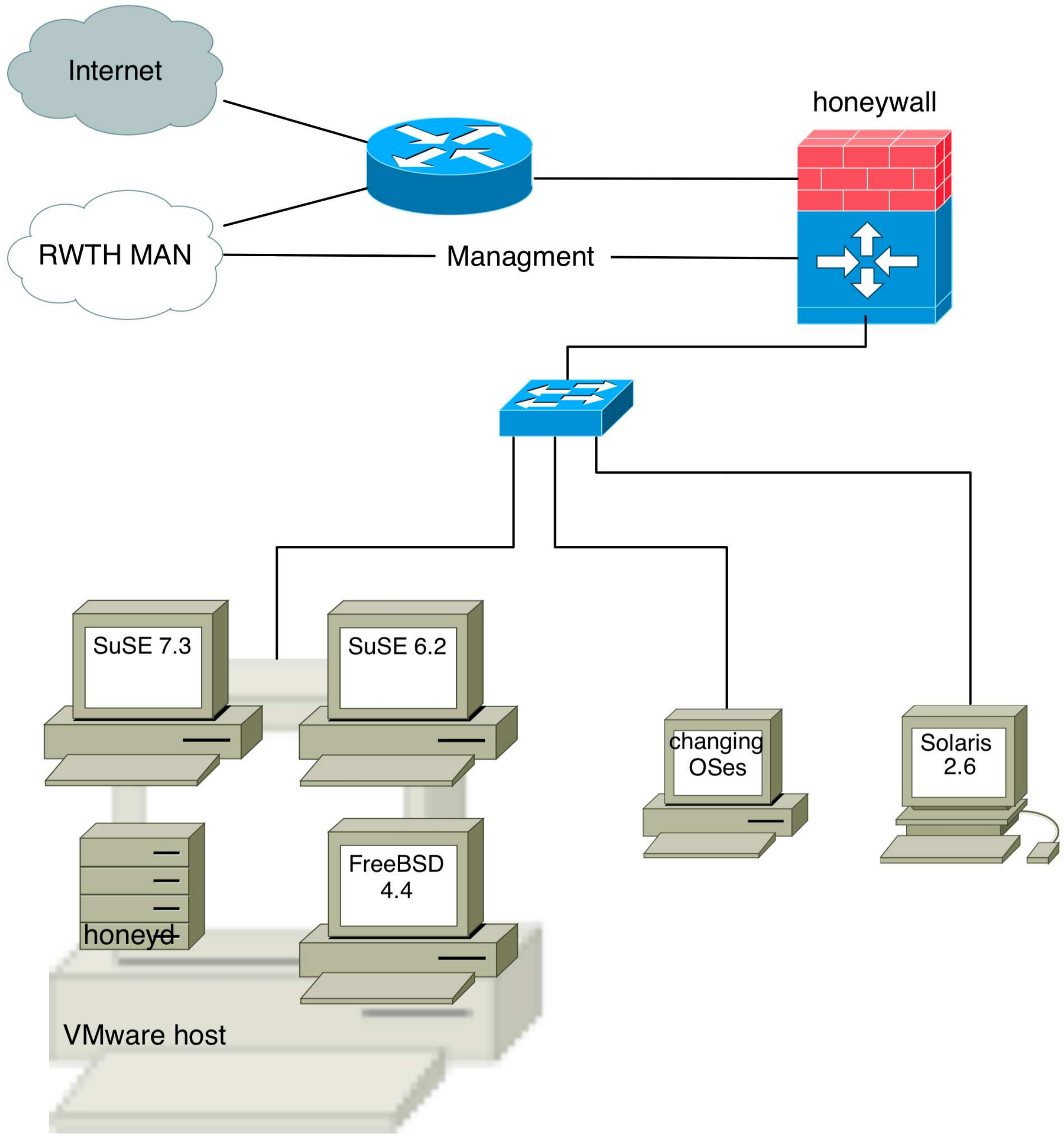
Distribution of operating systems for machines randomly scanning the Internet:

```
756k    Windows XP SP1
 50k    Windows 2000 SP4
 43k    Windows NT 4.0
 30k    Windows 2000
 30k    Linux 2.6
 13k    Unknown
 12k    Windows 2000 RFC1323
 12k    Windows 98
 2.3k   Linux google
 1.7k   Windows 98 noSACK
 1.4k   NetApp CacheFlow
 1.4k   FreeBSD 5.0-5.1
 1.0k   Linux 2.2
 1.0k   Windows 2000 Cisco
```

Setup February – June



Current Setup



Preliminary results I

- NO breaking until now
- First attack on one of the honeypots after about ten minute
- Seeing more or less automated malware, worms and similar things
- PHP-Nuke (installed on the Linux honeypot) was hit several times, but nobody actually tried to exploit it
- Mostly attacks on port 80, seldom also port 22 or 21 (`put speed-test1MB`)

Preliminary results II

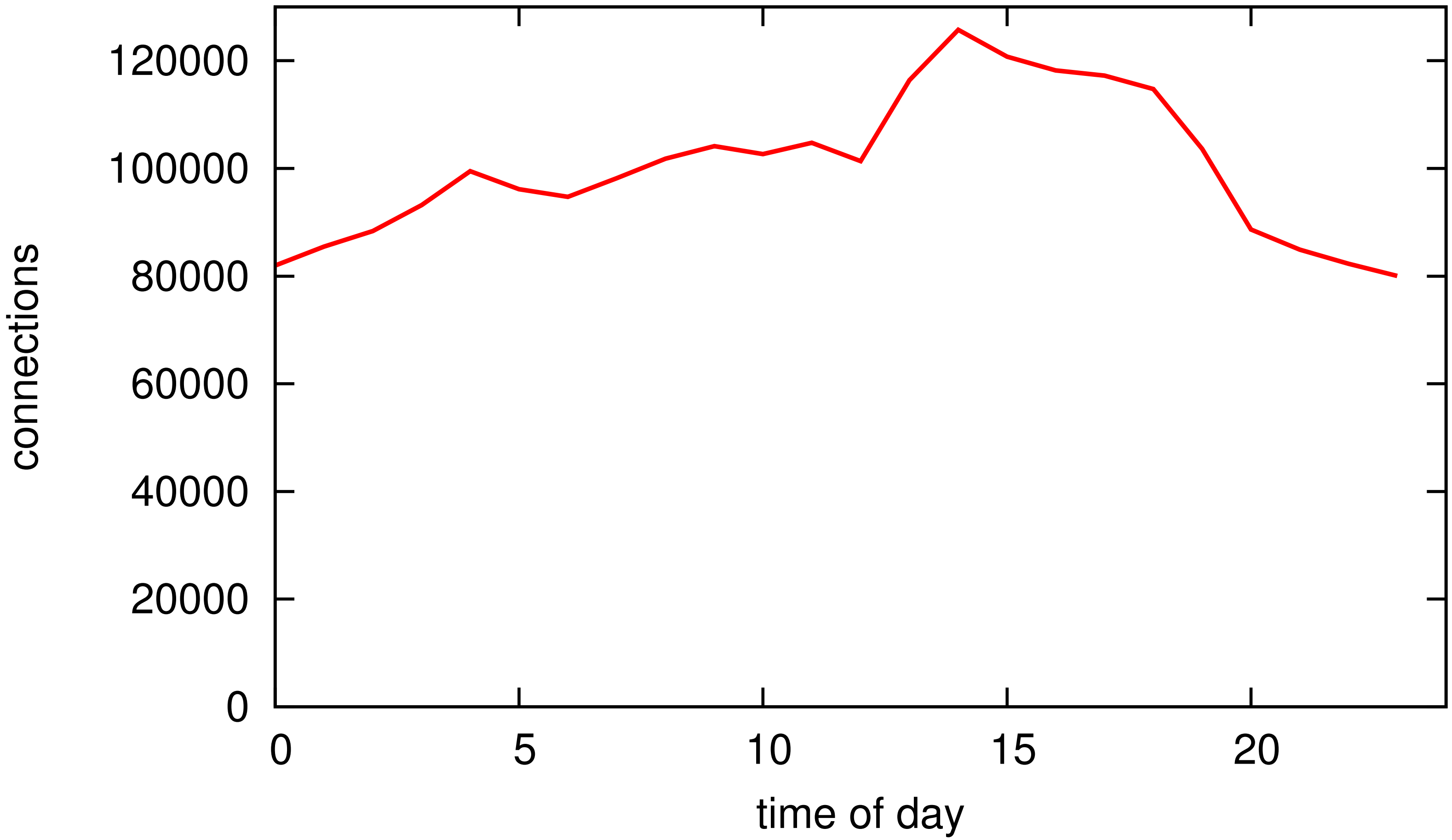
***Results for honeyd network in two week period between 07-21 and 08-04*:**

- **More than 2.4 million connection attempts**
- **Almost 48,000 unique IP-adresses**

Protocol	Number of packets
Total	2404766
TCP	2010921
UDP	363230
ICMP	30615

Preliminary results III

Connections per hour:



- **Similar results for other periods**

Preliminary results IV

Rank	Resource	Number of connections
1	445/TCP	965723
2	139/TCP	800960
3	137/UDP	357453
4	135/UDP	93275
5	80/TCP	33463
6	8/ICMP	28694
7	1025/TCP	23871
8	2745/TCP	23566
9	6129/TCP	17042
10	3127/TCP	16178

- **NetBIOS dominates before HTTP, ICMP Echo Request and various worms**

Preliminary results V

Rank	Source IP	Connections	DNS PTR record for Source IP
1	66.94.77.121	129528	
2	68.73.254.233	63471	adsl-68-73-254-233.dsl.chcgil. ameritech.net
3	69.156.110.172	53693	
4	64.229.170.202	47737	HSE-MTL-ppp64773.qc.sympatico.ca
5	64.144.104.227	40296	64-144-104-227.client.dsl.net
6	68.254.25.41	27993	
7	64.228.68.148	27856	HSE-Toronto-ppp130807.sympatico.ca
8	66.134.211.10	26971	h-66-134-211-10.lsanca54.covad.net
9	64.229.170.226	24963	HSE-MTL-ppp64797.qc.sympatico.ca
10	63.196.246.88	24424	adsl-63-196-246-88.dsl.lsan03. pacbell.net

Preliminary results VI

- More or less uniform distribution of source addresses, most common one:
  - ne.jp (7.5%)
  - verizon.net (4%)
  - hinet.net (2.5%)
- About 42% reverse DNS lookups failed

`honeyd` + `p0f`:

- About 70% of all connection attempts could be associated with an OS
- More than 90% of these were caused by Windows machines

NoSEBrEaK

NoSEBrEaK

- *We had no attacks on our honeynet, so . . .*
- Toolkit written in Python 2.3 to detect and remove Sebek from honeypot
- Work together with Maximillian Dornseif and Christian N. Klein
- Presented as academic paper at 5th IEEE Information Assurance Workshop, Westpoint. Available at arXiv as cs.CR/0406052
- Also presented at Black Hat / DefCon. Material available at md.hudora.de
- Now: Short presentation of our results

Sebek

[...] monitoring capability to all activity on the honeypot including, but not limited to, keystrokes. If a file is copied to the honeypot, Sebek will see and record the file, producing an identical copy. If the intruder fires up an IRC or mail client, Sebek will see those messages. [...] Sebek also provides the ability to monitor the internal workings of the honeypot in a glass-box manner, as compared to the previous black-box techniques. [...] intruders can detect and disable Sebek. Fortunately, by the time Sebek has been disabled, the code associated with the technique and a record of the disabling action has been sent to the collection server.

Sebek-paper from Ed Balas

Workings of Sebek *in short*

Concentrate on Linux version 2.1.7, no look at other versions or OS

Basic mechanism of Sebek and interesting points:

- **Hijack** `sys_read()`
- **Send data passing through** `sys_read()` **in covert manner over the network**
- **Overwrites part of the network stack (**`packet_recvmsg`**) to hide Sebek data passing on to the network**

Hiding of Sebek

- **Sebek loads as a kernel module**
- **Afterwards `cleaner.o` (part of `adore`) is loaded which removes Sebek from modules list**

From `cleaner.o`

```
if (__this_module.next)
    __this_module.next = __this_module.next->next;
```

This works because kernel maintains list of modules (`sys_create_module()`)

```
    spin_lock_irqsave(&modlist_lock, flags);
    mod->next = module_list;
    module_list = mod;    /* link it in */
    spin_unlock_irqrestore(&modlist_lock, flags);
```

Detecting Sebek

Several ways to detect Sebek come to mind:

- Latency
- Network traffic counters
- Modification of syscall table
- Finding hidden module
- Other cruft in memory

Latency

First detection method we found during tests:

"*dd-attack*"

```
$ dd if=/dev/zero of=/dev/null bs=1
```

Just call `sys_read()` a couple of thousand times per second...

Movie: `dd.mov`

Network Traffic Counters

- **dd-attack / backward running counters**
  - ***Issue solved in Sebek 2.1.7, changed packet counter manipulation technique (take a look at `sprintf_stats`)***
- `dev->get_stats->tx_bytes` **or** `dev->get_stats->tx_packets` **vs.** `/proc/net/dev` **or** `ifconfig` **output**

Movie: `devchecker.mov`

4 GB traffic in 4 minutes?

RedTeam@RWTH

```
vampire:~/NoSEBrEaK/kebes# ifconfig  eth0
eth0      Link encap:Ethernet  HWaddr 00:02:3F:74:5E:3D
          inet addr:10.11.12.2  Bcast:10.255.255.255  Mask:255.0.0.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:254 errors:0 dropped:0 overruns:0 frame:0
          TX packets:4294967295 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:25860 (25.2 KiB)  TX bytes:4294966268 (3.9 GiB)
          Interrupt:10 Base address:0x5800

vampire:~/NoSEBrEaK/kebes# uptime
 21:16:36 up 4 min,  2 users,  load average: 0.02, 0.07, 0.03
vampire:~/NoSEBrEaK/kebes#
```

Modification of Syscall Table

- **Before loading Sebek:**

```
sys_read  = 0xc0132ecc
sys_write = 0xc0132fc8
```

- **Afterwards:**

```
sys_read  = 0xc884e748
sys_write = 0xc0132fc8
```

Interesting things in `/usr/include/linux/module.h` Kernel 2.4.X

```
struct module {
    unsigned long size_of_struct; /* == sizeof(module) *
    struct module *next;          // Pointer into kernel
    const char *name;             // Pointer into kernel

    struct module_symbol *syms;  // Pointer into kernel
    struct module_ref *deps;     // Pointer into kernel
    struct module_ref *refs;     // Pointer into kernel
    int (*init)(void);           // Pointer into module
    void (*cleanup)(void);       // Pointer into module
}
```

(Note: Kernel 2.6 has different `module.h`)

Variables with only small range of “reasonable” values:

```
struct module {
    unsigned long size;

    union {
        atomic_t usecount;
        long pad;
    } uc;

    unsigned long flags;

    unsigned nsyms;
    unsigned ndeps;
}
```

Finding Modules

- **Module header is allocated by kernel's `vmalloc`**
- **Function `vmalloc` alligns memory to page boundaries (4096 bytes on IA32)**
- **Memory allocated by `vmalloc` starts at `VMALLOC_START` and ends `VMALLOC_RESERVE` bytes later**

```
for (p = VMALLOC_START;
     p <= VMALLOC_START + VMALLOC_RESERVE - PAGE_SYZE;
     p =+ PAGE_SIZE)
```

phrack issue 0x3d, phile #0x03 – `module_hunter.c`

Movie: module_hunter.mov

Retrieving Sebek's Variables

00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000

- **Initial memory layout**

Retrieving Sebek's Variables

00000000	00000000	PORT	00000000	00000000	00000000	00000000	MAC5
00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
00000000	00000000	00000000	00000000	MAC2	00000000	MAC1	00000000
00000000	MAGIC	00000000	00000000	00000000	00000000	00000000	00000000
MAC4	00000000	00000000	00000000	00000000	00000000	00000000	00000000
00000000	00000000	00000000	00000000	MAC0	00000000	00000000	00000000
00000000	MAC3	00000000	00000000	00000000	00000000	00000000	00000000
00000000	00000000	00000000	00000000	00000000	00000000	IP	00000000

- Random positions of parameters (`gen_fudge.pl`)

Retrieving Sebek's Variables

00000000	00000000	00007a69	00000000	00000000	00000000	00000000	000000d9
00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
00000000	00000000	00000000	00000000	000000dc	00000000	0000000d	00000000
00000000	f001c0de	00000000	00000000	00000000	00000000	00000000	00000000
000000e5	00000000	00000000	00000000	00000000	00000000	00000000	00000000
00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
00000000	0000003a	00000000	00000000	00000000	00000000	00000000	00000000
00000000	00000000	00000000	00000000	00000000	00000000	d5495b1d	00000000

- Memory layout after random insertion of parameters

Retrieving Sebek's Variables

00000000	00000000	00007a69	00000000	00000000	00000000	00000000	000000d9
00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
00000000	00000000	00000000	00000000	000000dc	00000000	0000000d	00000000
00000000	f001c0de	00000000	00000000	00000000	00000000	00000000	00000000
000000e5	00000000	00000000	00000000	00000000	00000000	00000000	00000000
00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
00000000	0000003a	00000000	00000000	00000000	00000000	00000000	00000000
00000000	00000000	00000000	00000000	00000000	00000000	d5495b1d	00000000

f001c0de = 240.1.192.222 (multicast address)

- Probably not the IP address
- But probably the Magic?

Retrieving Sebek's Variables

00000000	00000000	00007a69	00000000	00000000	00000000	00000000	000000d9
00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
00000000	00000000	00000000	00000000	000000dc	00000000	0000000d	00000000
00000000	f001c0de	00000000	00000000	00000000	00000000	00000000	00000000
000000e5	00000000	00000000	00000000	00000000	00000000	00000000	00000000
00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
00000000	0000003a	00000000	00000000	00000000	00000000	00000000	00000000
00000000	00000000	00000000	00000000	00000000	00000000	d5495b1d	00000000

d5495b1d = 213.73.91.29

- Probably not the Magic
- But probably the IP address!

Retrieving Sebek's Variables

00000000	00000000	00007a69	00000000	00000000	00000000	00000000	000000d9
00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
00000000	00000000	00000000	00000000	000000dc	00000000	0000000d	00000000
00000000	f001c0de	00000000	00000000	00000000	00000000	00000000	00000000
000000e5	00000000	00000000	00000000	00000000	00000000	00000000	00000000
00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
00000000	0000003a	00000000	00000000	00000000	00000000	00000000	00000000
00000000	00000000	00000000	00000000	00000000	00000000	d5495b1d	00000000

00007a69 = 31337

- Is this probably the port number?
- And are the other numbers part of the MAC address?

Movie: NoSEBrEaKer.mov

Disabling Sebek

- The easy way: Call `cleanup()`
  `kerneljumper.o` – jump to arbitrary memory location and execute code
- The obvious way: Reconstruct `sys_read()` pointer from the kernel and fix it in syscall table
  Saved inside memory, problem of locating it
- The crazy way: Patch in your own, untainted version of `sys_read()`
  Untested, but should work

What can be logged?

- Unconditionally obtained by operator of honeypot
  - All network traffic (⇒ use encrypted communication / attack logging host (hard!))
  - All calls to `read()` (⇒ avoid `read()`)
- Possibly obtained after break-in
  - Forensic data obtained by disk analysis (⇒ keep most things in memory only)
  - Syslog-data (⇒ avoid it as best as possible)

Intercepting `read()`

- What kind of programs use `read()`?
  - Almost every interactive program uses `read(1)`
  - Many programs use `read()` for reading configuration files etc.
  - Network programs usually use `recv()` instead of `read()`
- Making `read()` unreliable
  - Read in as much data as possible

$\Rightarrow$ dd-attack (*not reliable, no control*)

Living with `read()`

- Surprisingly it is possible to avoid `read()` in many cases
- Use `mmap()` instead :-)
  - It is very hard to intercept
  - Drawback: It works only on regular files
  - Things you can not access:
    - `/dev/random` (useful for getting random seed for crypto stuff)
    - pipes (useful for communication)
    - All devices

Better living without `read()`

- Talk directly to network, execute commands without calling other programs wherever possible
- Nice bonus: `exec()` does not call `read()` (but importing libraries may do so. . .)

Other stuff

- Messing with the process name – just copy & rename the binary
- Name of the command calling `read()` is logged (max 12 bytes) – we can play with it
- Since filenames are not logged, we can give impression of reading certain files (makes forensic harder)

Kebes

- Proof of concept code
- Entirely written in Python 2.3 for portability with no external dependency
- Can do everything you can expect from a basic shell
- Highly dynamic, leaves not much traces at honeypot

Kebes : Networking

- Uses TCP-sockets for networking but could also be adopted to use `stdin/stdout` or anything else
- On top of that implements a crypto layer based on Diffie-Hellman / AES providing compression and random length padding
- Main problem: Getting entropy for DH
  - Use race-conditions and similar things to get entropy
- Python-specific "Kebes layer" using serialized objects to transfer commands and results back and forth

Kebes : “Kebes layer” I

- Can work asynchronous and send multiple commands at once
  - *Asynchronous commands not implemented by the server at this time*
- Commands can usually work on several objects on the server at once
- Highly dynamic: Kebes layer initially knows only a single command; ADDCOMMAND

Kebes : "Kebes layer" II

- Code for all additional commands is pushed by client into server at runtime as serialized Python objects

⇒ So most of NoSEBrEaK-code will only exist in the server's RAM – makes forensic harder

- Implemented commands: Reading / writing files, secure deletion, direct execution, listing directories, . . .

Other versions of Sebek

- Sebek Win32 client
  - Traverse PsLoadedModuleList (similar to module list in Linux)
  - Watch out for hooked APIs (similar to changed memory locations in syscall table in Linux)
  - Disable Sebek through restoring of SDT ServiceTable (similar to reconstruction of syscall table in Linux)
  - Work by Tan Chew Keong
- In *BSD version, similar things should be possible if attacker is r00t

Further things

- Shameless plug: “Defeating Honeypots: Network Issues”, written by Laurent Oudot and me, available at **securityfocus** since yesterday evening
- Detection of UML- or VMWare-based honeypots also possible
- Can you escape from within UML or VMWare?
- Detection of other honeypot-related software also possible (e.g. LaBrea, Fake AP, . . .)

Further Questions?

- Thanks for your attention!
- If you are interested in setting up a honeypot, just contact me...
- Mail: `tho@koeln.ccc.de`