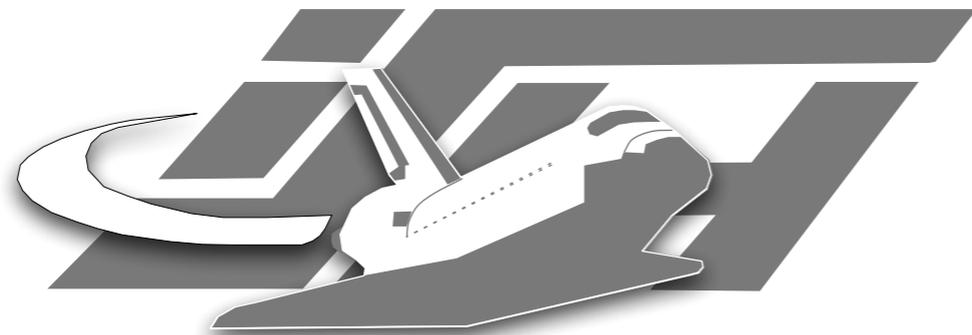


NoSEBrEaK - Defeating Honeynets

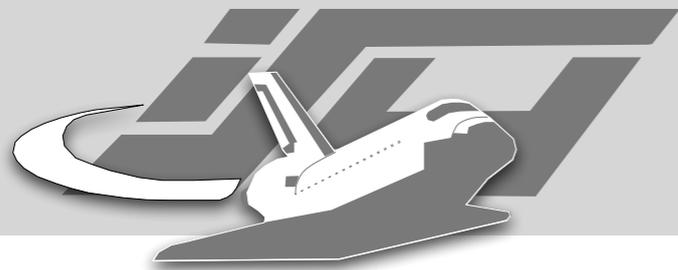
Maximillian Dornseif, Thorsten Holz, Christian N. Klein

at

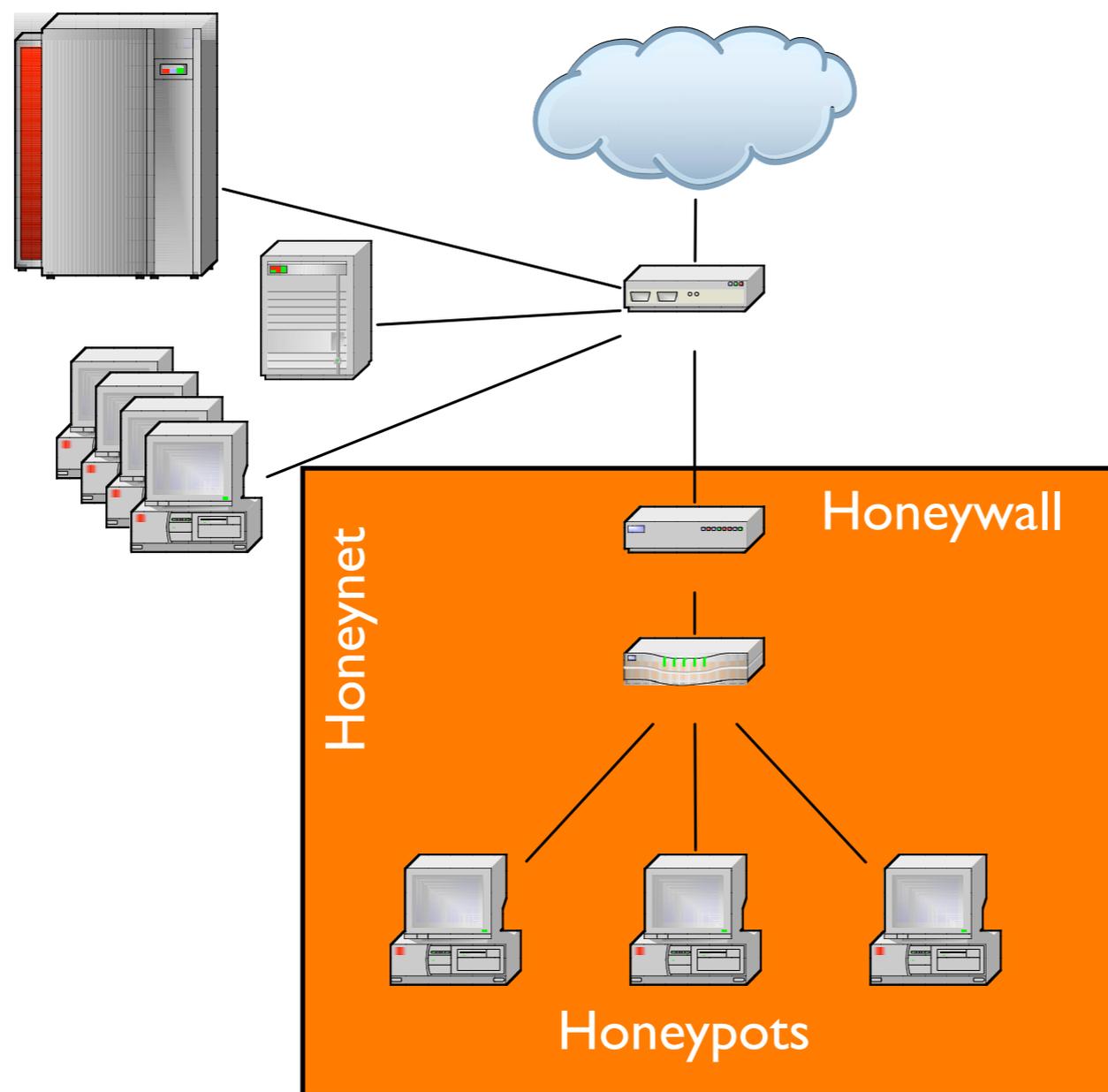
Black Hat Briefings 2004, Las Vegas

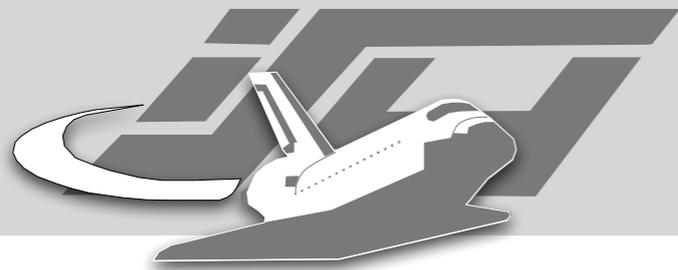


RWTHAACHEN
RHEINISCH-WESTFÄLISCHE TECHNISCHE HOCHSCHULE AACHEN



Honeynets



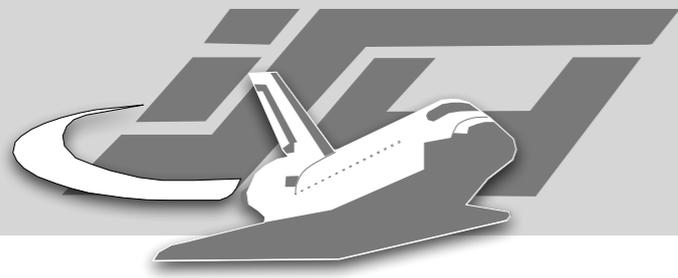


Sebek



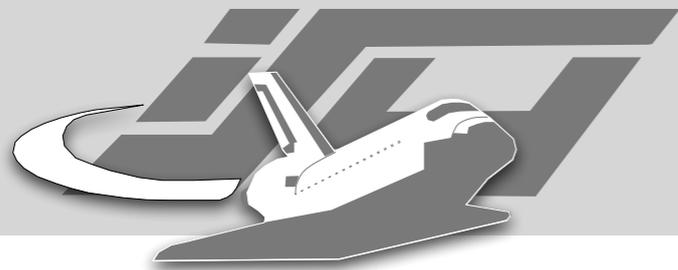
[...] monitoring capability to all activity on the honeypot including, but not limited to, keystrokes. If a file is copied to the honeypot, Sebek will see and record the file, producing an identical copy. If the intruder fires up an IRC or mail client, Sebek will see those messages. [...] Sebek also 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.

Know Your Enemy: Sebek



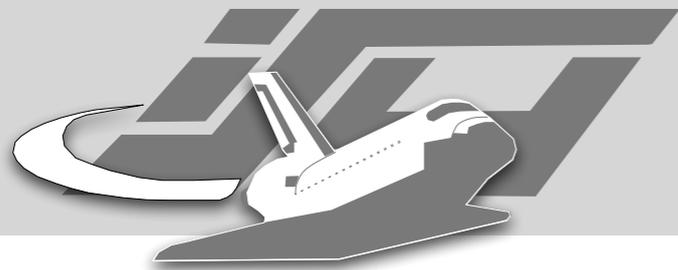
Workings of Sebek

- Hijacks `sys_read()`.
- Sends data passing through `sys_read()` over the network.
- Overwrites parts of the Network stack (`packet_recvmmsg`) to hide Sebek data passing on the network.
- Packages to be hidden are identified by protocol (`ETH_P_IP`, `IPPROTO_UDP`), port and magic.



Hiding of Sebek

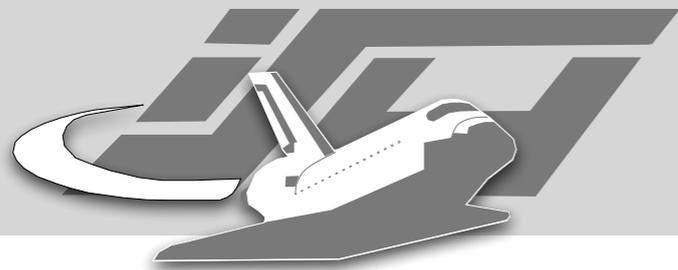
- Sebek loads as a kernel module with a random numeric name.
- Afterwards a second module is loaded which simply removes Sebek from the list of modules and unloads itself.



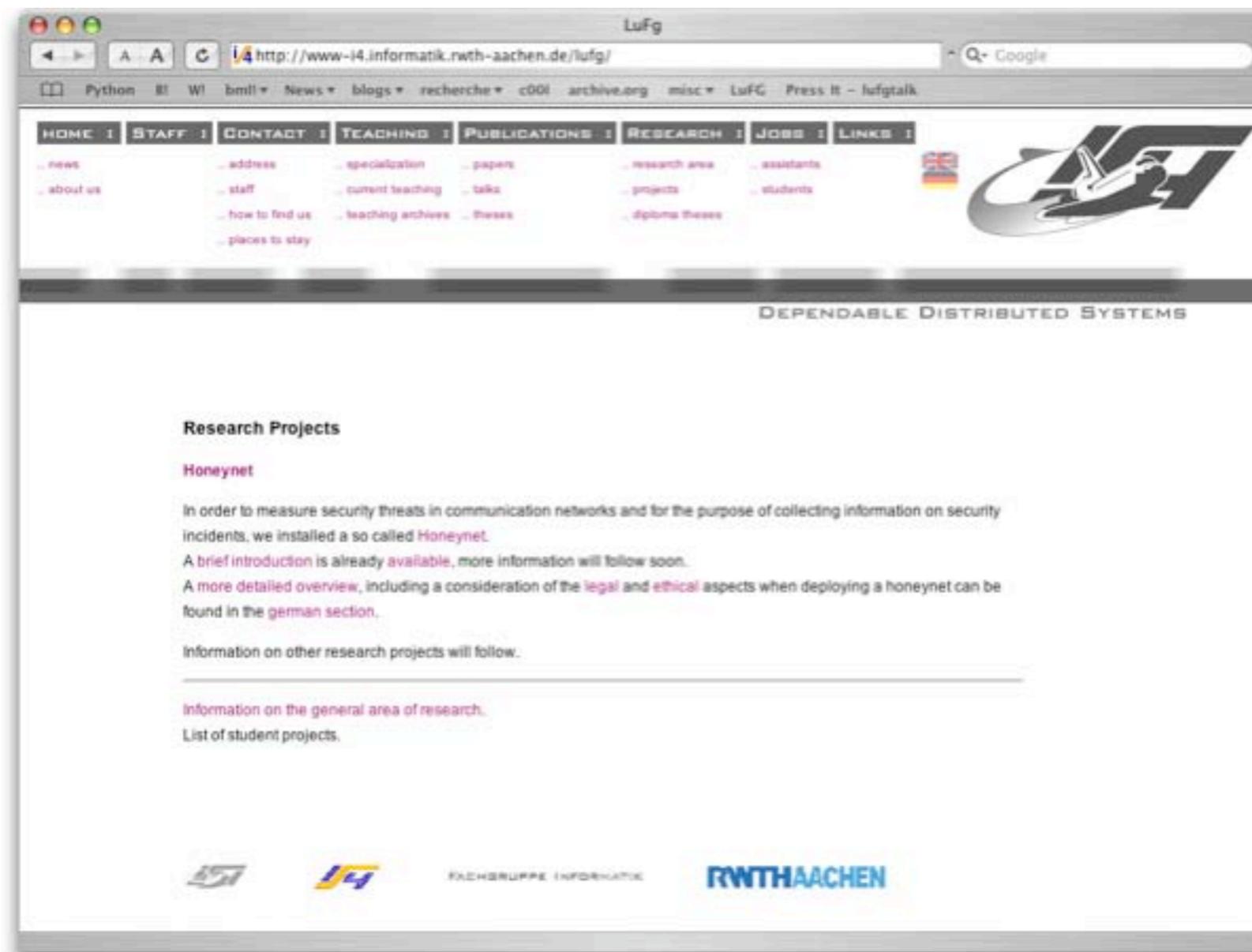
Demonstration

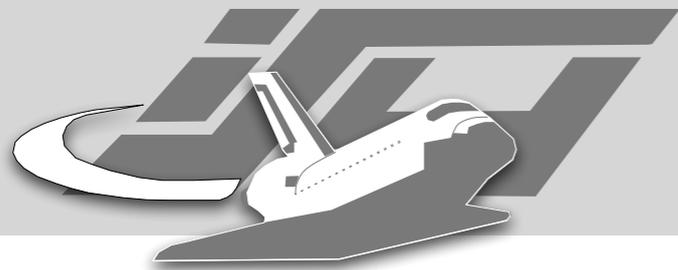
```
RedTeam@RWTH - SSH connection to the Honeypot
vampire:~# cd NoSEBrEaK/sebek-2.1.7/
vampire:~/NoSEBrEaK/sebek-2.1.7# sh sbk_install.sh
Installing Sebek:
  snd-pcmcia.o installed successfully
  cleaner.o installed successfully
  cleaner.o removed successfully
vampire:~/NoSEBrEaK/sebek-2.1.7# lsmod
Module                Size  Used by    Not tainted
ds                     7092   2
yenta_socket          10912   2
pcmcia_core            42912   0 [ds yenta_socket]
ipsec                  268100  0
uhci                   27452   0 (unused)
usbcore                64972   1 [uhci]
vampire:~/NoSEBrEaK/sebek-2.1.7# 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:174 errors:0 dropped:0 overruns:0 frame:0
          TX packets:104 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:19298 (18.8 KiB)  TX bytes:13376 (13.0 KiB)
          Interrupt:10 Base address:0x5800

vampire:~/NoSEBrEaK/sebek-2.1.7#
```



Who and Why





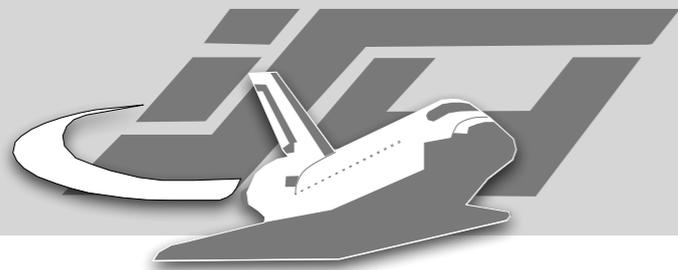
Sebek



[...] 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.

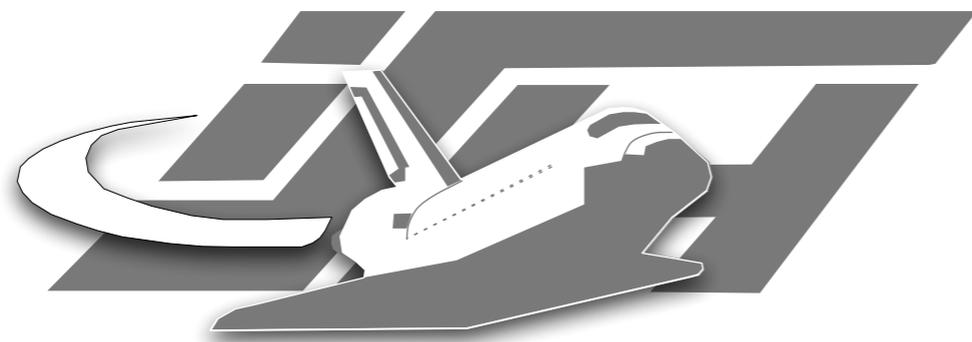
Know Your Enemy: Sebek



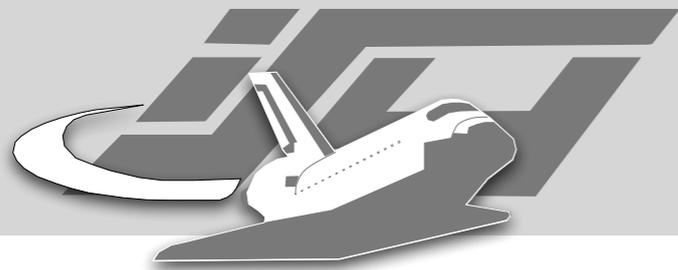
Demonstration

```
RedTeam@RWTH - SSH connection to the Honeypot
Last login: Wed Jul 28 06:29:28 on ttyp5
ssh root@10.11.12.2
Welcome to Darwin!
[c0ldcut:~] md% ssh root@10.11.12.2
root@10.11.12.2's password:
Linux vampire 2.4.23 #1 Fri Dec 19 01:32:48 CET 2003 i686 GNU/Linux
Last login: Wed Jul 28 14:51:44 2004 from 10.11.12.1
vampire:~# cd /var/www/users/kebes/
vampire:/var/www/users/kebes# sh demo-server.sh
gcc -o vulnerablesuid vulnerablesuid.c
vulnerablesuid.c:7:1: warning: no newline at end of file
sudo chmod 0755 vulnerablesserver.py
sudo chmod 04755 vulnerablesuid
sudo chown root vulnerablesuid
starting vulnerable http server
█
```

Detecting Sebek



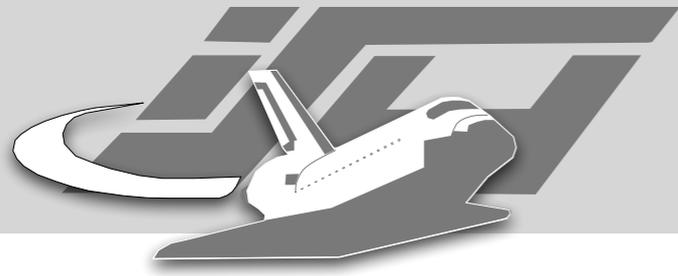
RWTHAACHEN
RHEINISCH-WESTFÄLISCHE TECHNISCHE HOCHSCHULE AACHEN



Detecting Sebek

Several ways to detect a Sebek infected host come to mind:

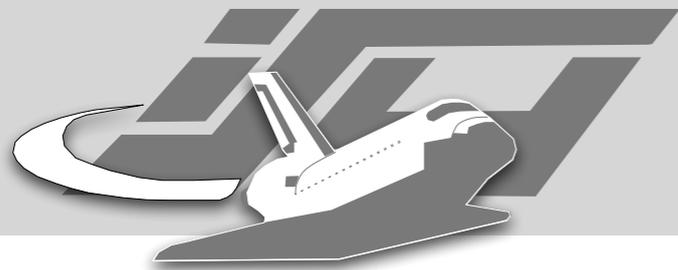
- (The Honeywall)
- Latency
- Network traffic counters
- Syscall table modification
- Hidden module
- Other cruft in memory



Latency

- dd-attack:

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

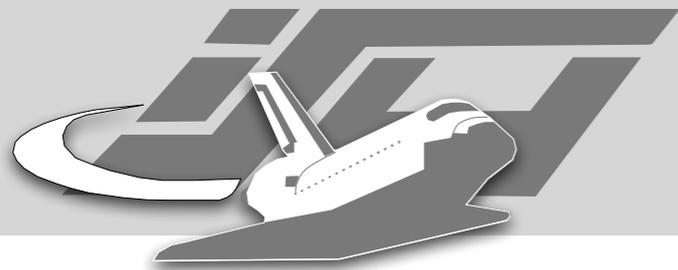


Demonstration

```
RedTeam@RWTH
vampire:~# ping -c 5 10.11.12.1
PING 10.11.12.1 (10.11.12.1) 56(84) bytes of data.
64 bytes from 10.11.12.1: icmp_seq=1 ttl=64 time=0.591 ms
64 bytes from 10.11.12.1: icmp_seq=2 ttl=64 time=0.297 ms
64 bytes from 10.11.12.1: icmp_seq=3 ttl=64 time=0.317 ms
64 bytes from 10.11.12.1: icmp_seq=4 ttl=64 time=0.315 ms
64 bytes from 10.11.12.1: icmp_seq=5 ttl=64 time=0.306 ms

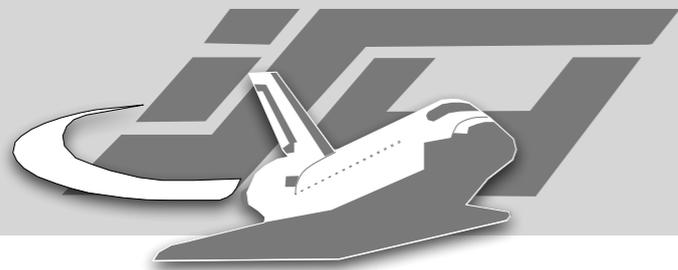
--- 10.11.12.1 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 3996ms
rtt min/avg/max/mdev = 0.297/0.365/0.591/0.113 ms
vampire:~# dd if=/dev/zero of=/dev/null &
[1] 1433
vampire:~# ping -c 5 10.11.12.1
PING 10.11.12.1 (10.11.12.1) 56(84) bytes of data.
64 bytes from 10.11.12.1: icmp_seq=1 ttl=64 time=0.917 ms
64 bytes from 10.11.12.1: icmp_seq=2 ttl=64 time=2.33 ms
64 bytes from 10.11.12.1: icmp_seq=3 ttl=64 time=2.76 ms
64 bytes from 10.11.12.1: icmp_seq=5 ttl=64 time=2.46 ms

--- 10.11.12.1 ping statistics ---
5 packets transmitted, 4 received, 20% packet loss, time 4026ms
rtt min/avg/max/mdev = 0.917/2.122/2.769/0.714 ms
vampire:~# killall dd
vampire:~#
```



Network Traffic Counters

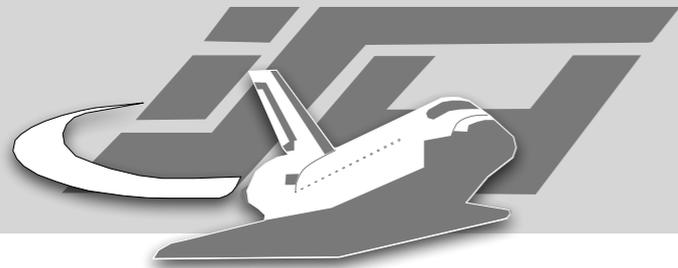
- dd-attack / backward running counters
- `dev->get_stats->tx_bytes` or `dev->get_stats->tx_packets`
vs. `/proc/net/dev` or `ifconfig` output.



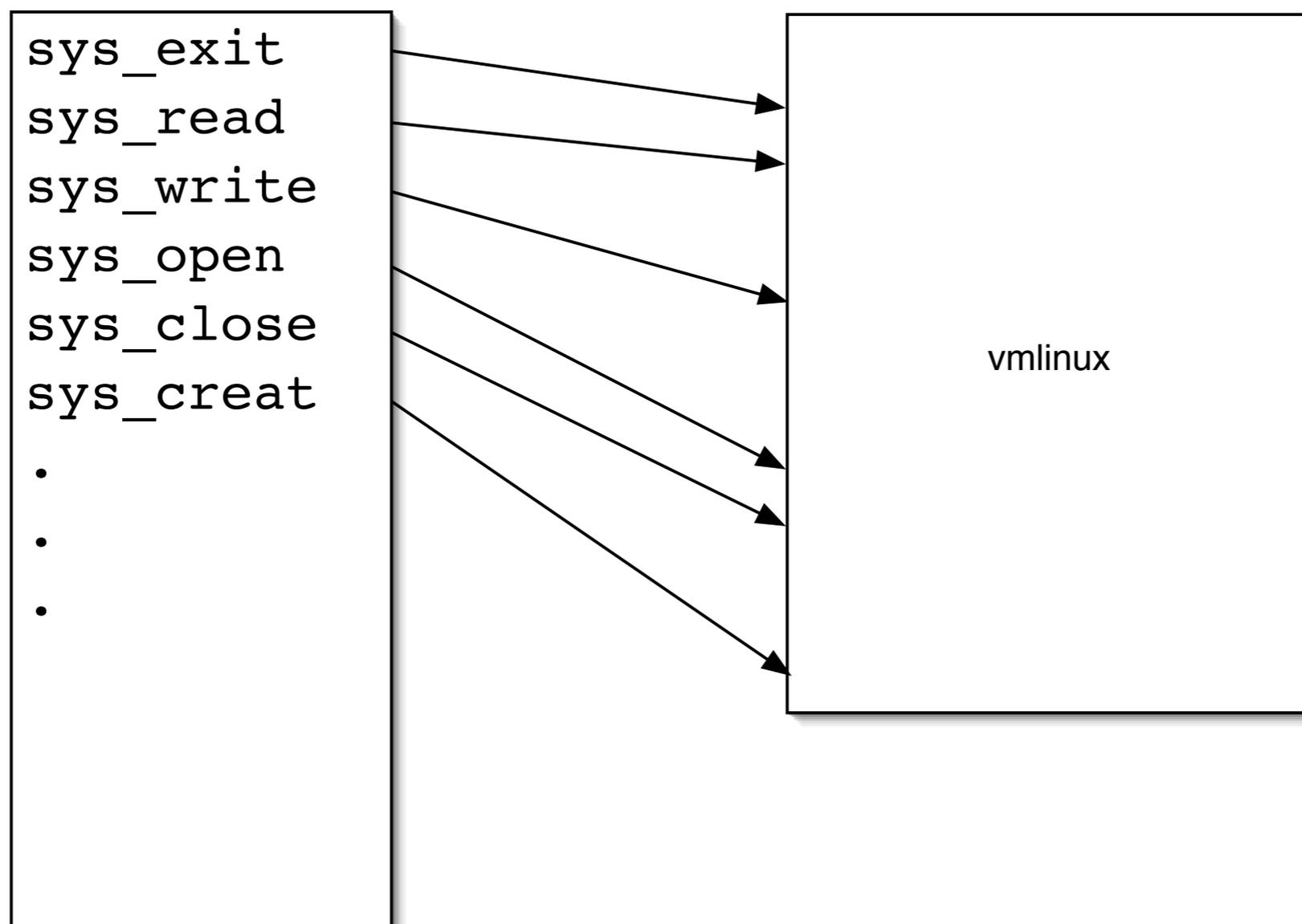
Demonstration

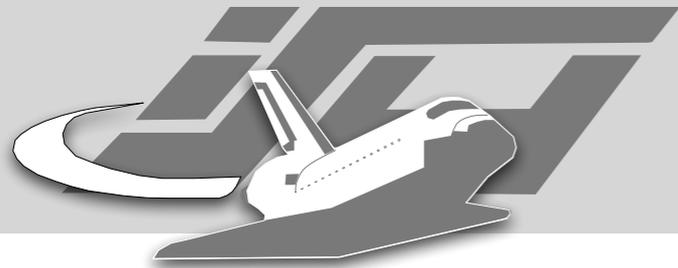
```
RedTeam@RWTH
Every 1s: ifconfig eth0                               Mon Jul 26 21:44:32 2004
eth0          Link encap:Ethernet  HWaddr 00:02:3F:74:5E:3D
              inet addr:10.11.12.2  Bcast:10.255.255.255  Mask:255.0.0.0
              UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
              RX packets:1386 errors:0 dropped:0 overruns:0 frame:0
              TX packets:238151 errors:0 dropped:0 overruns:0 carrier:0
              collisions:680 txqueuelen:1000
              RX bytes:117800 (115.0 KiB)  TX bytes:86128840 (82.1 MiB)

RedTeam@RWTH
vampire:~/NoSEBrEaK/kebes# dd if=/dev/zero of=/dev/null bs=1 count=1000000
1000000+0 records in
1000000+0 records out
1000000 bytes transferred in 2.466439 seconds (405443 bytes/sec)
vampire:~/NoSEBrEaK/kebes# insmod devchecker.o
vampire:~/NoSEBrEaK/kebes# dd if=/dev/zero of=/dev/null bs=1 count=1000000
1000000+0 records in
1000000+0 records out
1000000 bytes transferred in 2.473703 seconds (404252 bytes/sec)
vampire:~/NoSEBrEaK/kebes# rmmmod devchecker
vampire:~/NoSEBrEaK/kebes# dd if=/dev/zero of=/dev/null bs=1 count=1000000
1000000+0 records in
1000000+0 records out
1000000 bytes transferred in 2.458416 seconds (406766 bytes/sec)
vampire:~/NoSEBrEaK/kebes# insmod devchecker.o
```

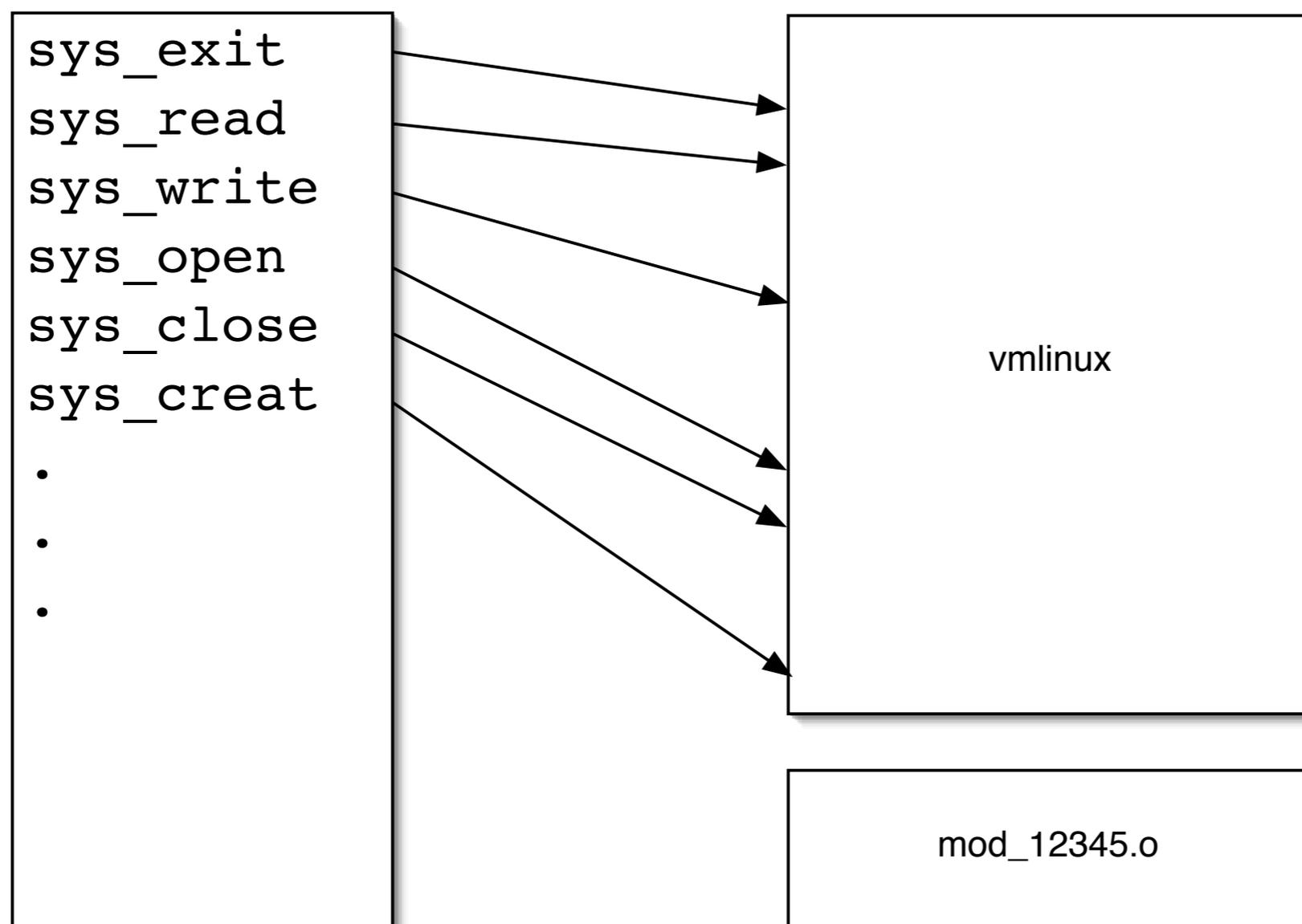


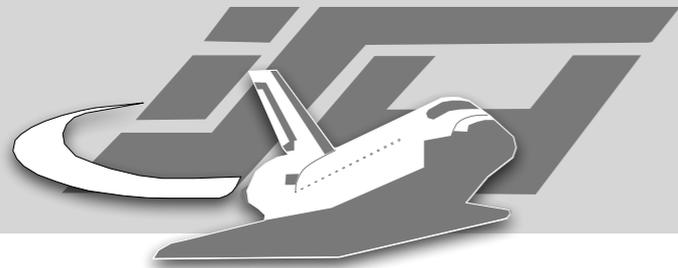
Syscall Table



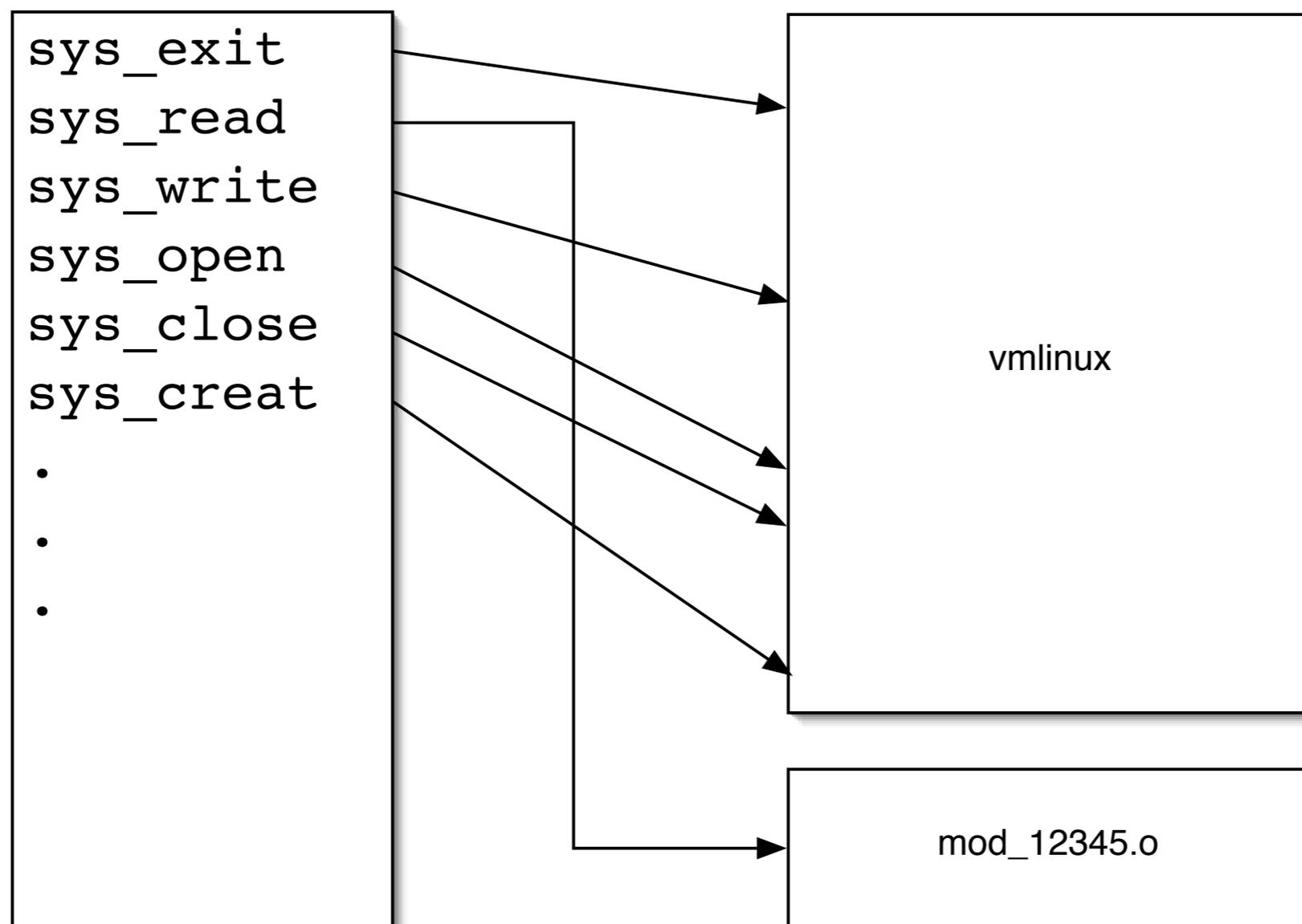


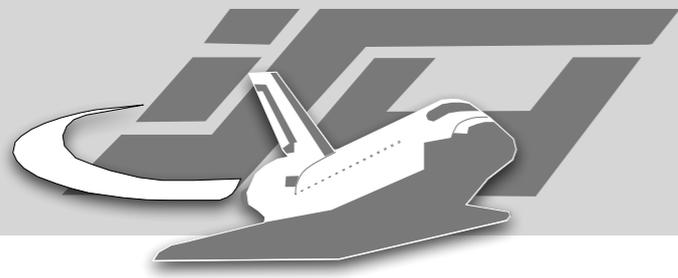
Syscall Table





Syscall Table





Syscall Table

before:

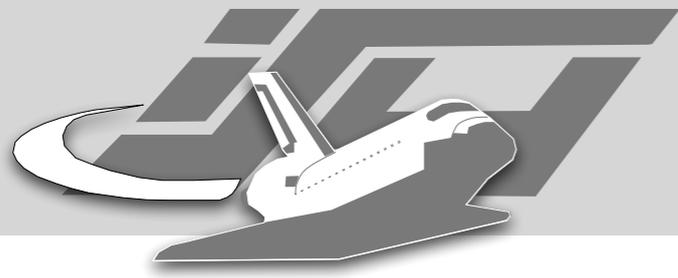
`sys_read` = `0xc0132ecc`

`sys_write` = `0xc0132fc8`

after:

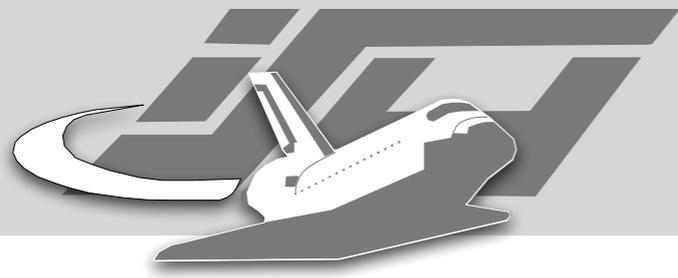
`sys_read` = `0xc884e748`

`sys_write` = `0xc0132fc8`



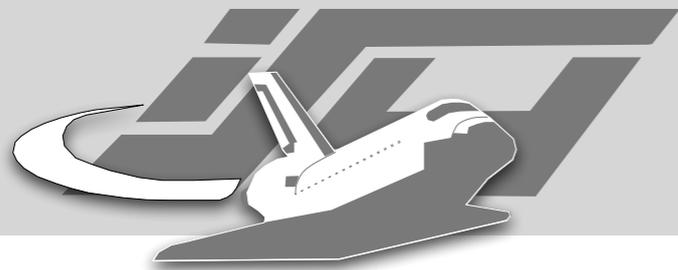
Finding Modules

- Find
- Extract variables
- Disable



Module Header

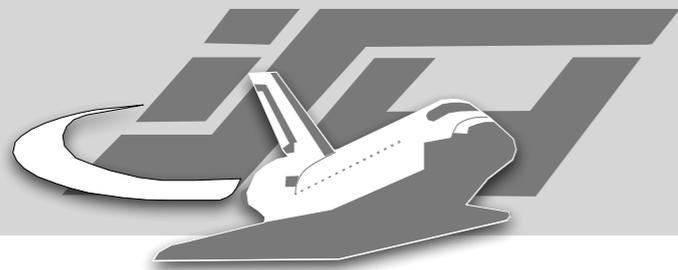
```
include/linux/module.h:
struct module
{
    unsigned long size_of_struct; /* sizeof(module) */
    struct module *next;
    const char *name;
    unsigned long size;
    union {
        atomic_t usecount;
        long pad;
    } uc; /* Needs to keep its size - so says rth */
    unsigned long flags; /* AUTOCLEAN et al */
    unsigned nsyms;
    unsigned ndeps;
    struct module_symbol *syms;
    struct module_ref *deps;
    struct module_ref *refs;
    int (*init)(void);
    void (*cleanup)(void);
};
```



Module Header

→ 96

```
include/linux/module.h:
struct module
{
    unsigned long size_of_struct; /* sizeof(module) */
    struct module *next;
    const char *name;
    unsigned long size;
    union {
        atomic_t usecount;
        long pad;
    } uc; /* Needs to keep its size - so says rth */
    unsigned long flags; /* AUTOCLEAN et al */
    unsigned nsyms;
    unsigned ndeps;
    struct module_symbol *syms;
    struct module_ref *deps;
    struct module_ref *refs;
    int (*init)(void);
    void (*cleanup)(void);
};
```

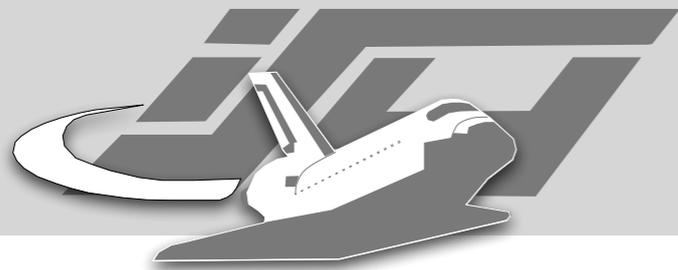


Module Header

→ 96

→ Pointers into Kernel

```
include/linux/module.h:
struct module
{
    unsigned long size_of_struct; /* sizeof(module) */
    struct module *next;
    const char *name;
    unsigned long size;
    union {
        atomic_t usecount;
        long pad;
    } uc; /* Needs to keep its size - so says rth */
    unsigned long flags; /* AUTOCLEAN et al */
    unsigned nsyms;
    unsigned ndeps;
    struct module_symbol *syms;
    struct module_ref *deps;
    struct module_ref *refs;
    int (*init)(void);
    void (*cleanup)(void);
}
```



Module Header



```
include/linux/module.h:
```

```
struct module
```

```
{
```

```
    unsigned long size_of_struct; /* sizeof(module) */
```

```
    struct module *next;
```

```
    const char *name;
```

```
    unsigned long size;
```

```
    union {
```

```
        atomic_t usecount;
```

```
        long pad;
```

```
    } uc; /* Needs to keep its size - so says rth */
```

```
    unsigned long flags; /* AUTOCLEAN et al */
```

```
    unsigned nsyms;
```

```
    unsigned ndeps;
```

```
    struct module_symbol *syms;
```

```
    struct module_ref *deps;
```

```
    struct module_ref *refs;
```

```
    int (*init)(void);
```

```
    void (*cleanup)(void);
```



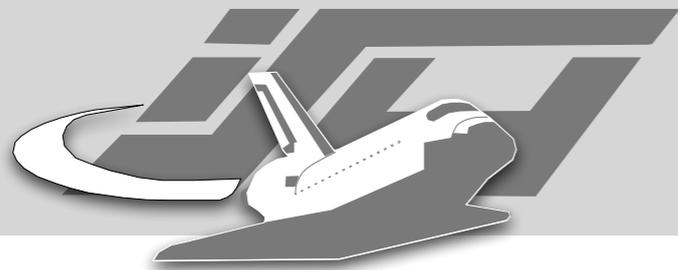
96



Pointers into Kernel



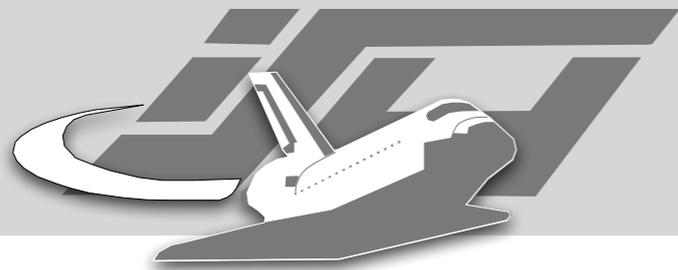
Pointers into the Module



Module Header

-
- 96
- Pointers into Kernel
- Pointers into the Module
- Variables with only a small range of “reasonable” values.

```
include/linux/module.h:
struct module
{
    unsigned long size_of_struct; /* sizeof(module) */
    struct module *next;
    const char *name;
    unsigned long size;
    union {
        atomic_t usecount;
        long pad;
    } uc; /* Needs to keep its size - so says rth */
    unsigned long flags; /* AUTOCLEAN et al */
    unsigned nsyms;
    unsigned ndeps;
    struct module_symbol *syms;
    struct module_ref *deps;
    struct module_ref *refs;
    int (*init)(void);
    void (*cleanup)(void);
}
```

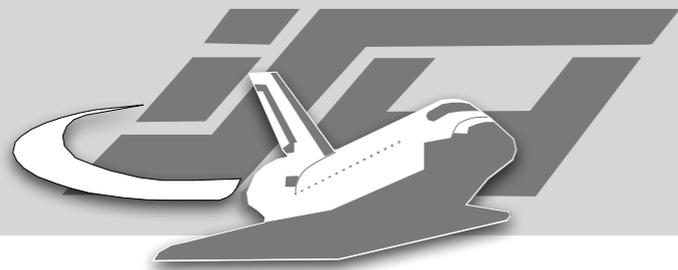


Finding Modules

- A module header is allocated by the kernel's `vmalloc`.
- The function `vmalloc` aligns 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_SIZE; \  
    p += PAGE_SIZE)
```

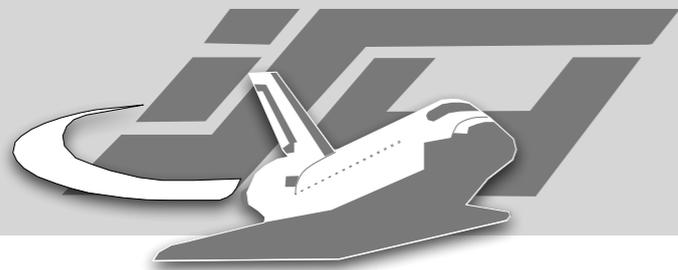
from `module_hunter.c` by madsys



Demonstration

```
RedTeam@RWTH
vampire:~/NoSEBrEaK/kebes/detectors# lsmod
Module                Size  Used by  Not tainted
ds                    7092   2
yenta_socket         10912   2
pcmcia_core          42912   0 [ds yenta_socket]
ipsec                268100  0
usbcore              64972   1
vampire:~/NoSEBrEaK/kebes/detectors# insmod module_hunter.o
vampire:~/NoSEBrEaK/kebes/detectors# cat /proc/showmodules
vampire:~/NoSEBrEaK/kebes/detectors# dmesg | tail -n 9
address                module

0xe0e6d000             usbcore size: 0xfdcc
0xe0e7e000             snd-pcmcia size: 0x50b4
0xe0e86000             ipsec size: 0x41744
0xe0ec9000             module_hunter size: 0x3ec
0xe0f2e000             pcmcia_core size: 0xa7a0
0xe0f3a000             yenta_socket size: 0x2aa0
0xe0f44000             ds size: 0x1bb4
vampire:~/NoSEBrEaK/kebes/detectors#
```



Retrieving Sebek's variables

```
$bs = 128 + int(rand(128));

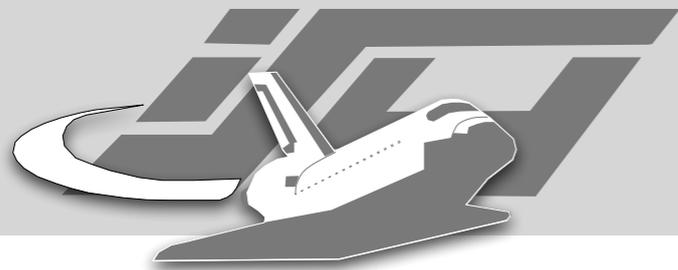
for($x=0;$x<38;$x++){
    $tmp = int(rand() * $bs);
    if(!defined($values{$tmp})) {
        $values{$tmp} = $x;
        push(@fun,$tmp);
    } else {$x--;}}

($dip, $dport, $sip, $sport, $kso, $magic, $smac0 ... $dmac5, $m_if,
$m_dip, $m_dmac, $m_dport, $m_sport, $m_kso, $m_magic, $m_block) = @fun;

$m_block = int(rand(1000000000)); $mod_name = int(rand(1000000000));

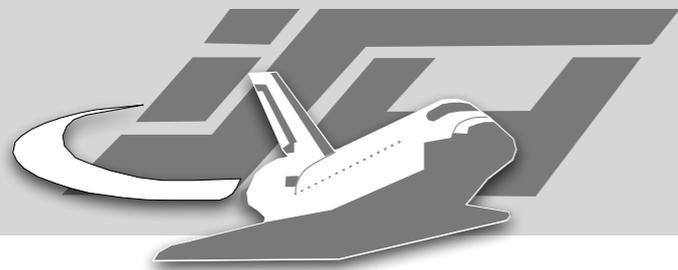
printf"//----- autogenerated fudge.h file\n\n\n";
print "#define BS                $bs\n";
print "#define DIP_OFFSET          $dip\n";
print "#define DPORT_OFFSET         $dport\n";
print "#define SIP_OFFSET           $sip\n";
print "#define SPORT_OFFSET         $sport\n";
print "#define KSO_OFFSET           $kso\n";
print "#define MAGIC_OFFSET         $magic\n";
print "#define SMAC_0_OFFSET        $smac0\n";
...
print "#define DMAC_5_OFFSET      $dmac5\n";

sebek.h: u32        BLOCK[BS];
```



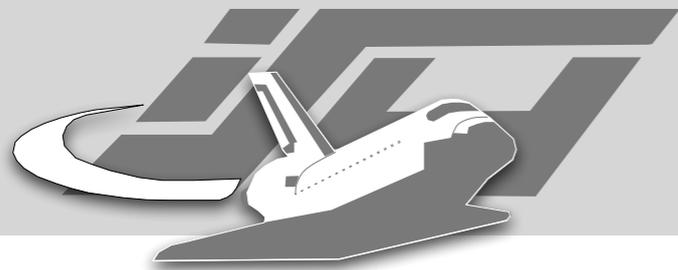
Retrieving Sebek's variables

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
00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000



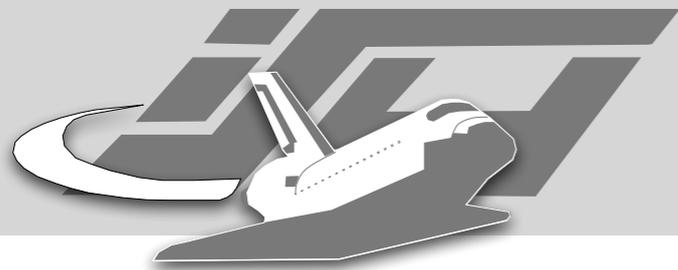
Retrieving Sebek's variables

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	00000000	MAC0	00000000	00000000
00000000	00000000	MAC3	00000000	00000000	00000000	00000000	00000000
00000000	00000000	00000000	00000000	00000000	00000000	IP	00000000



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	00000000	0000003a	00000000	00000000	00000000	00000000	00000000
00000000	00000000	00000000	00000000	00000000	00000000	d5495b1d	00000000



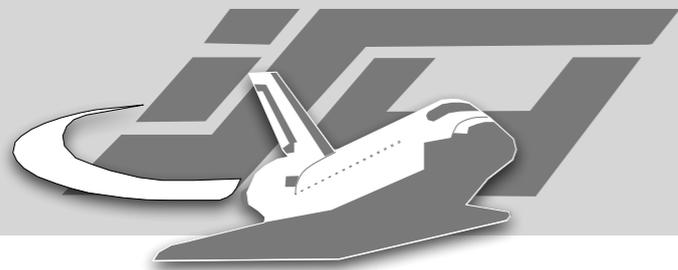
Retrieving Sebek's variables

IP?

~~240.1.192.222~~

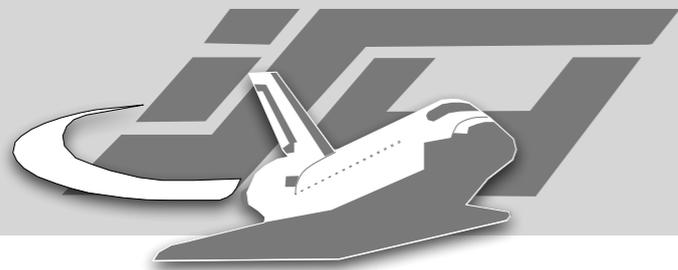
Magic?

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	00000000	0000003a	00000000	00000000	00000000	00000000	00000000
00000000	00000000	00000000	00000000	00000000	00000000	d5495b1d	00000000



Retrieving Sebek's variables

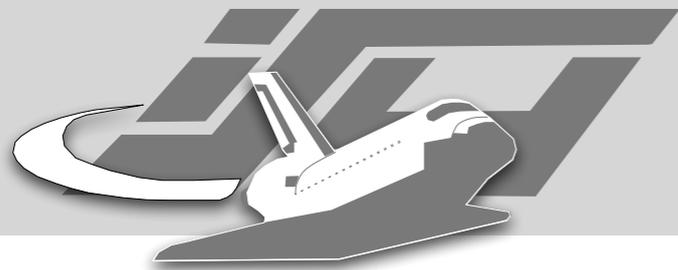
00000000	00000000	00007a69	00000000	00000000	00000000	00000000	000000d9
00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
00000000	00000000	00000000	00000000	000000dc	00000000	0000000d	00000000
00000000	f001c0de	00000000	00000000	00000000	00000000	00000000	00000000
000000e5	00000000	00000000	00000000	00000000	00000000	00000000	00000000
00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
00000000	00000000	0000003a	00000000	00000000	00000000	00000000	00000000
213.73.91.29	00000000	00000000	00000000	00000000	00000000	d5495b1d	00000000



Retrieving Sebek's variables

31337
Port?

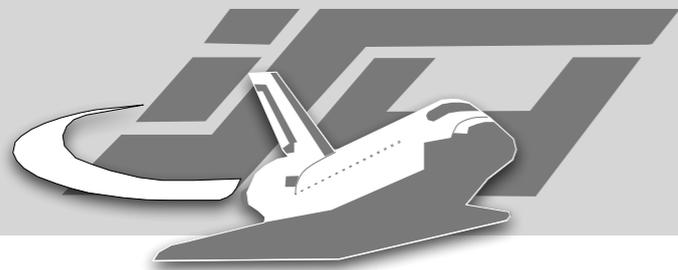
00000000	00000000	00007a69	00000000	00000000	00000000	00000000	000000d9
00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
00000000	00000000	00000000	00000000	000000dc	00000000	0000000d	00000000
00000000	f001c0de	00000000	00000000	00000000	00000000	00000000	00000000
000000e5	00000000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



Retrieving Sebek's variables

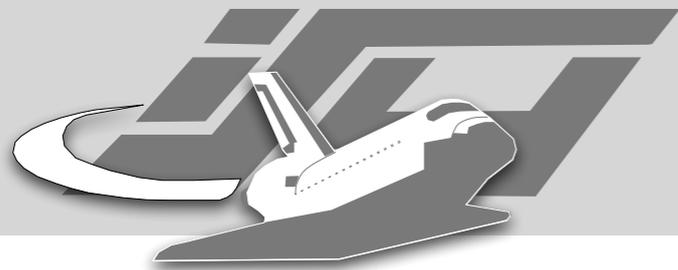
MAC?

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	00000000	0000003a	00000000	00000000	00000000	00000000	00000000
00000000	00000000	00000000	00000000	00000000	00000000	d5495b1d	00000000



Demonstration

```
RedTeam@RWTH
vampire:~/NoSEBrEaK/kebes# python NoSEBrEaKer.py
reading kernel symbol table ... done
resolving variables: high_memory loops_per_jiffy_addr boot_cpu_data sys_close
loops_per_jiffy = c02923c8, boot_cpu_data = c0292b20, distance = 758
sys_call_table claimed to be at c02925f0, verifying ...
sys_call_table found at c02925f0
sys_read = e0f4afb0 sys_write = c013b2b0 sys_open = c013a500
syscalls are far apart ... probably sys_write is hijacked
*** suspected sebek module at e0f47000 named 'snd-pcmcia', size 20876 bytes
* possible mac fragments: 0, 1, 2, 3, 11, 11, 11, 11, 11, 11, 18, 3d, 3f, 5e, 74, 79, ff, ff, ff, ff,
, ff, ff
* possible ports: 11111, 22222, 26466, 26470
      or: 1, 2, 3, 17, 17, 17, 17, 17, 17, 24, 61, 63, 94, 116, 121, 255, 255, 255, 255, 255
, 255
* possibles ips: 10.11.12.1, 10.11.12.2, 108.67.117.114, 144.177.19.192, 153.212.255.255, 153.212.25
5.255, 168.17.27.8, 184.17.27.8, 208.137.32.192, 76.67.85.82,
*      or: 115.111.114.0, 128.107.104.0, 128.253.62.0, 83.79.82.0, 87.183.110.0,
* possible magic values: 3EFD80, 524F53, 686B80, 6EB757, 726F73, 10C0B0A, 20C0B0A, 81B11A8, 81B11B8,
5255434C, 7275436C, C013B190, C02089D0, C02925F0, C15FD800, DEADBEEF, E0F4BCE3, E0F4BCEE, E0F4BD00,
FFFFD499, FFFFFD499
*      or: 11111, 22222, 26466, 26470
*      or: 1, 2, 3, 17, 17, 17, 17, 17, 17, 24, 61, 63, 94, 116, 121, 255, 255, 255, 2
55, 255, 255
* init() = e0f4aba0 ; cleanup() = e0f4ae00
vampire:~/NoSEBrEaK/kebes#
```

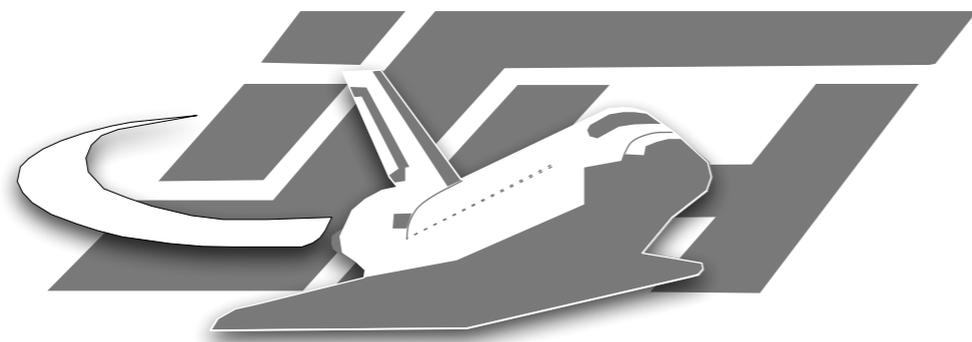


Disabling Sebek

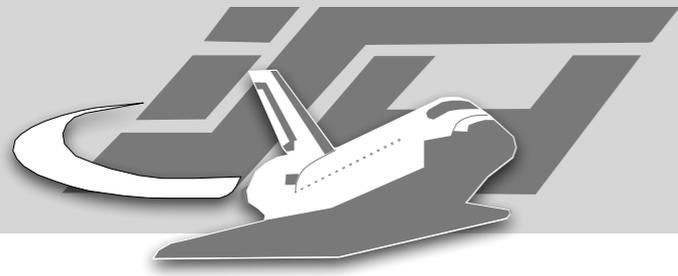
- The easy way: call `cleanup()`
- The obvious way: reconstruct `sys_read()` pointer from the kernel and fix it in the syscall table.
- The crazy way: patch in your own untainted `sys_read()`.

```
include/linux/module.h:
struct module
{
    unsigned long size_of_struct; /* sizeof(module) */
    struct module *next;
    const char *name;
    unsigned long size;
    union {
        atomic_t usecount;
        long pad;
    } uc; /* Needs to keep its size - so says rth */
    unsigned long flags; /* AUTOCLEAN et al */
    unsigned nsyms;
    unsigned ndeps;
    struct module_symbol *syms;
    struct module_ref *deps;
    struct module_ref *refs;
    int (*init)(void);
    void (*cleanup)(void);
};
```

Avoid logging

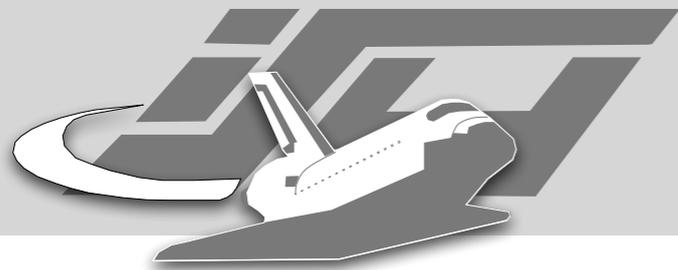


RWTHAACHEN
RHEINISCH-WESTFÄLISCHE TECHNISCHE HOCHSCHULE AACHEN



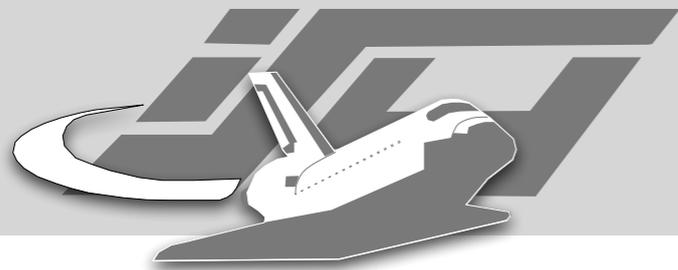
What can be logged?

- Unconditionally obtained by the adversary
 - All network traffic
 - All calls to `read()`
- Possible obtained
 - Forensic data obtained by disk analysis
 - `syslog` data



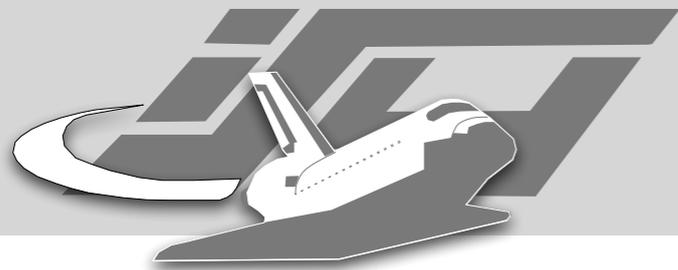
Logging of network traffic

- The adversary completely controls the network. What can we do about it?
- Use encrypted communication
 - Problem: how to deliver our initial payload? HTTPS-Exploit?
- Disable the logging host or gain access to it and delete data.

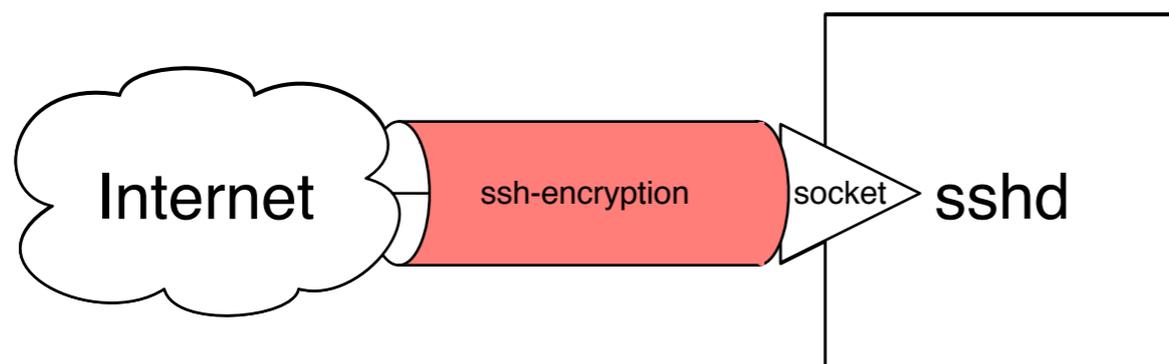


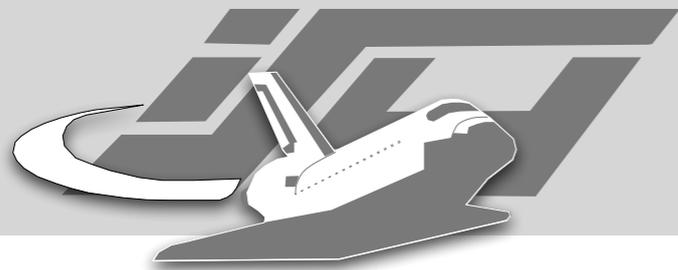
Intercepting `read()`

- Every interactive Program uses `read(1)`.
- Many Programs use `read()` for reading configuration files etc.
- Network Programs usually use `recv()` instead of `read()`.

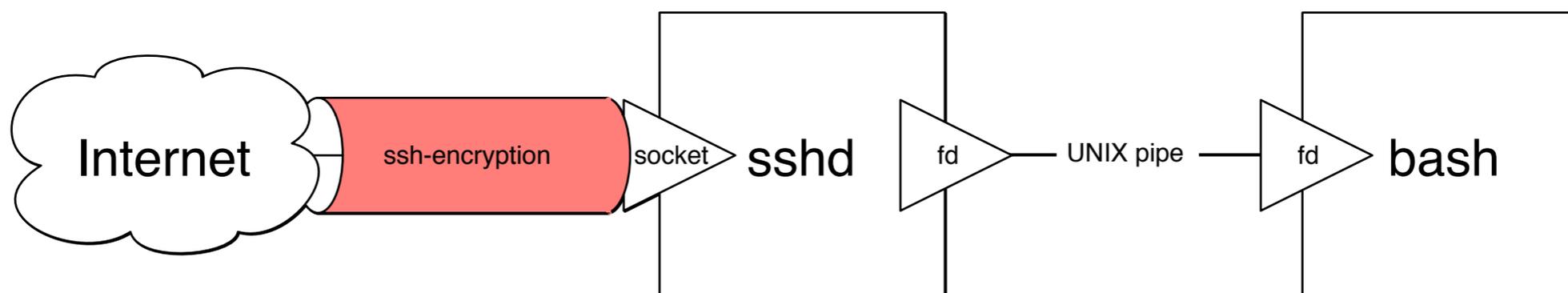


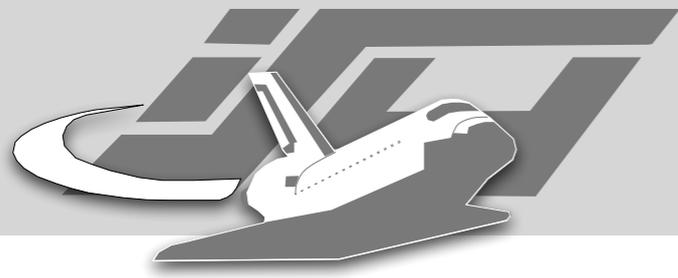
The power of read()





The power of read()

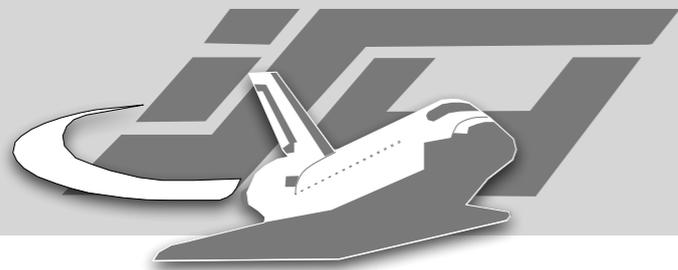




What is logged?

- data read
- pid, uid calling read()
- filedescriptor used
 - we can fiddle with this
- name of the process calling read() (max 12 bytes)
 - we can fiddle with this

```
struct sbk_h{
    u32  magic
    u16  ver
    u16  type
    u32  counter
    u32  time_sec
    u32  time_usec
    u32  pid
    u32  uid
    u32  fd
    char com[12]
    u32  length
};
```

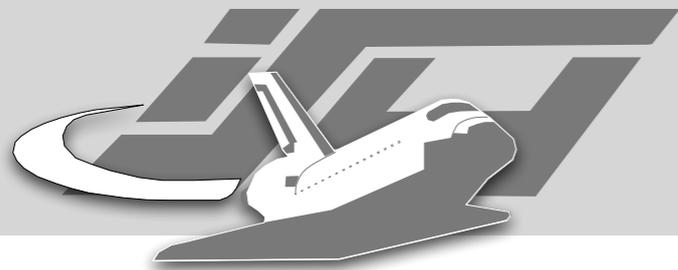


Making intercepting `read()` unreliable

- As long as you can squeeze more data through `read()` than can be transferred through the network, something will get lost.

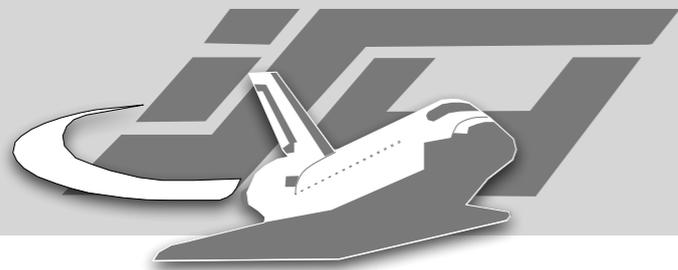
- dd-attack

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



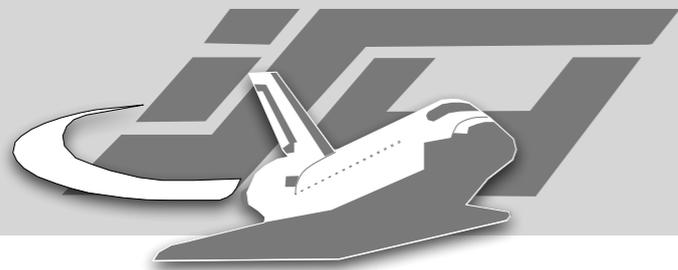
Living without read()

- Can we? Nearly!
- mmap() is our friend
 - it's very hard to intercept
 - it works on all regular files
 - ugly exception: /dev/random, pipes, etc.



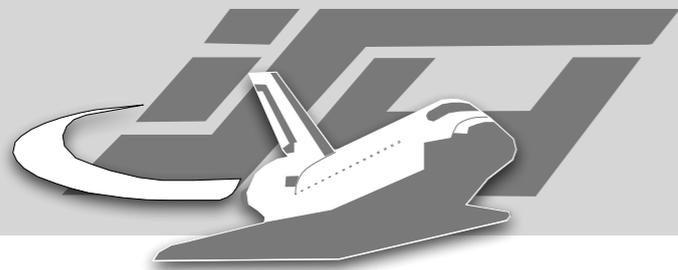
Better living without read()

- say goodbye to your shell
- you need something which directly talks to the network and executes your commands without calling other programs wherever possible.
- Nice bonus: `exec()` does not call `read`
 - but importing libraries may do so



Messing with the process name

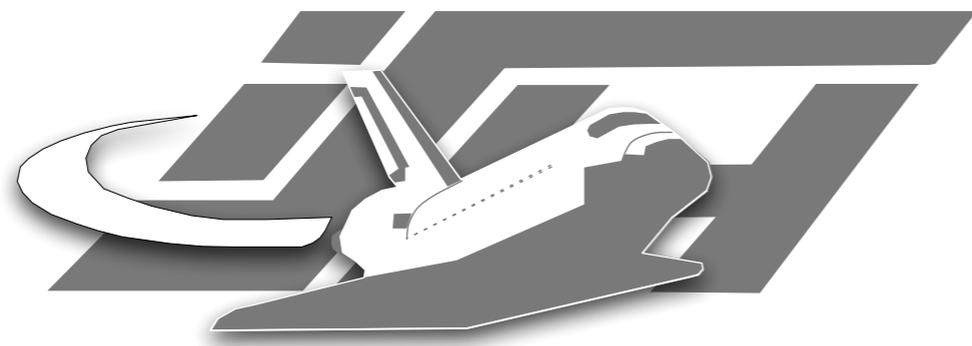
- Just copy & rename the binary.



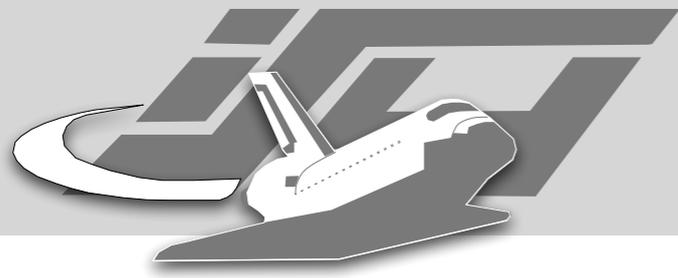
Results

- Reading files unnoticed.
- Possibly executing programs unnoticed.
- Since filenames are not logged, we can give the impression of reading certain files.
- Giving the impression we are executing programs which we don't.

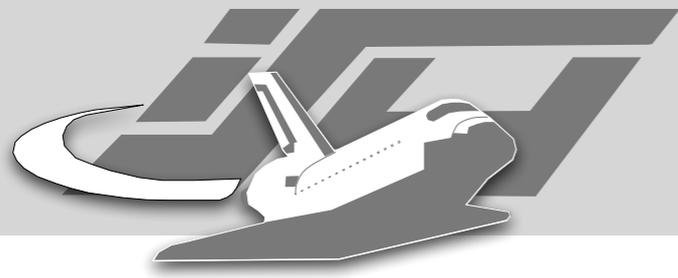
Kebes



RWTHAACHEN
RHEINISCH-WESTFÄLISCHE TECHNISCHE HOCHSCHULE AACHEN

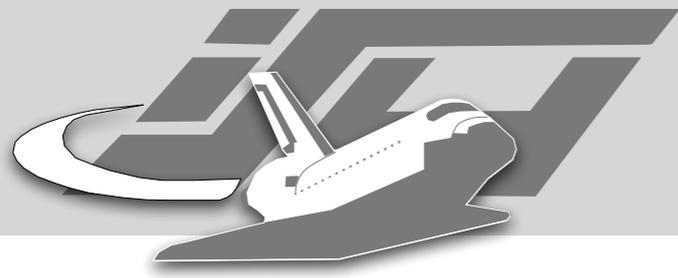


- Proof of concept code.
- Entirely written in Python 2.3 for portability with no external dependency.
- Can do everything you can expect from a basic shell.
- Highly dynamic.



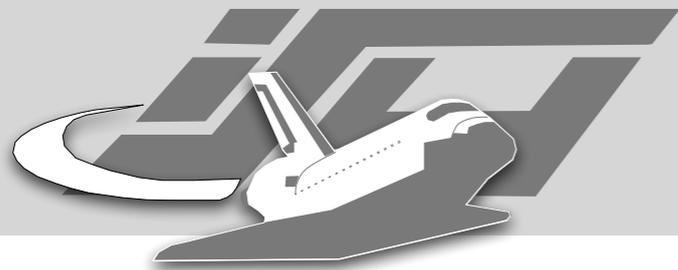
Kebes: networking

- Uses TCP-sockets for networking but could also be adopted to use stdout/stdin 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.
- Python specific “kebes layer” using serialized objects to transfer commands and results back and forth.



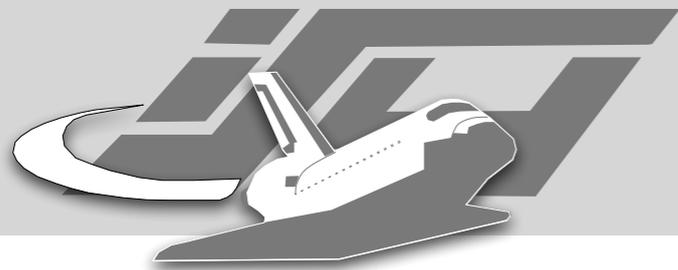
“Kebes layer”

- Can work asynchronous and send multiple commands at once.
- (Asynchronous commands are not implemented by the server at this time.)
- Commands can usually work on several objects on the server at once.



“Kebes layer”

- The Kebes server initially knows only a single command: `ADDCOMMAND`
- Code for all additional commands is pushed by the client into the server at runtime as serialized Python objects.
- So most of the kebes code will only exist in the server's RAM.
- Implemented commands: reading/writing files, secure deletion, direct execution, listing directories, ...



More Information

T1B2 1555

Proceedings of the 2004 IEEE
Workshop on Information Assurance and Security
United States Military Academy, West Point, NY, 7-9 June 2004

NoSEBrEaK – Attacking Honeynets

Maximillian Dornseif Thorsten Holz Christian N. Klein

Abstract— It is usually assumed that Honeynets are hard to detect and that attempts to detect or disable them can be unconditionally monitored. We scrutinize this assumption and demonstrate a method how a host in a honeynet can be completely controlled by an attacker without any substantial logging taking place.

I. INTRODUCTION

At the Laboratory for Dependable Distributed Systems at RWTH Aachen University, Germany, we run a Linux based honeynet for gathering information on security incidents. The scientific method dictates that we must attack our own assumptions vigorously to get meaningful results. Under the code name “NoSEBrEaK” we attacked our original assumptions about undetectability and monitorability of honeynets by turning the table and taking the view of an attacker trying to control a honeypot. In the following paper we present the results of this red team approach.

accessed by users via the `read()` system call on the honeynet. It replaces the normal `read()` system call with a new entry in the system call table pointing to its own version of this system call. It is then able to record all data accessed via `read()` [3]. Because Sebek lives entirely in kernel-space and has access to all data read, this rootkit is able to access most communication unencrypted and it can for example log SSH-sessions, recover files copied with SCP and record all passwords used by intruders. The recorded data is sent via UDP to the Sebek server, the other part of Sebek’s client/server architecture. This transmission is done by modifying the kernel in order to hide these packets such that an intruder can not see them. In addition, all network counters and data structures have to be readapted in order to make detecting these changes more difficult. Further information about Sebek and its architecture can be found in [3].

In this paper we show that a qualified attacker – in con-

Questions?

Maximilian Dornseif <dornseif@informatik.rwth-aachen.de>

Thorsten Holz <holz@i4.informatik.rwth-aachen.de>

Christian N. Klein <kleinc@cs.bonn.edu>

Slides & Software at

<http://md.hudora.de/presentations/#NoSEBrEaK-BH>

