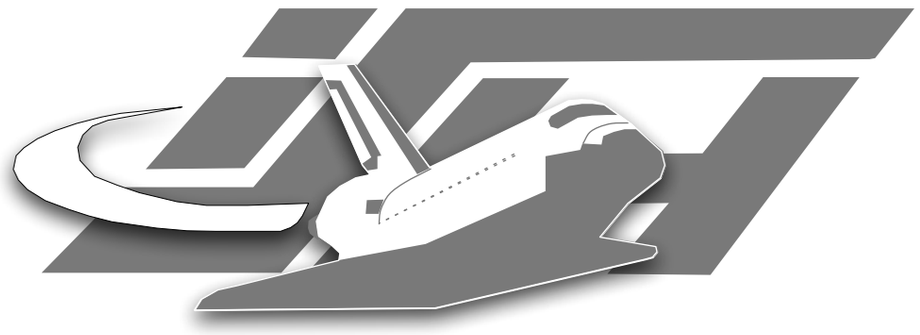
NoSEBrEaK - Defeating Honeynets

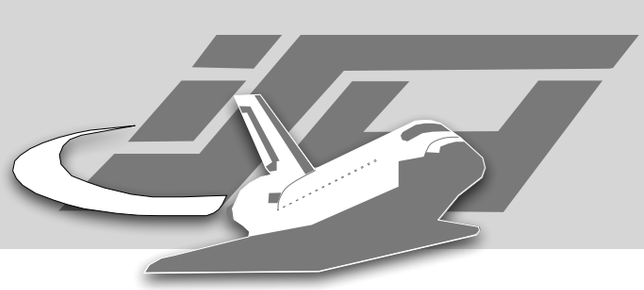
Maximillian Dornseif, Thorsten Holz, Christian N. Klein

at

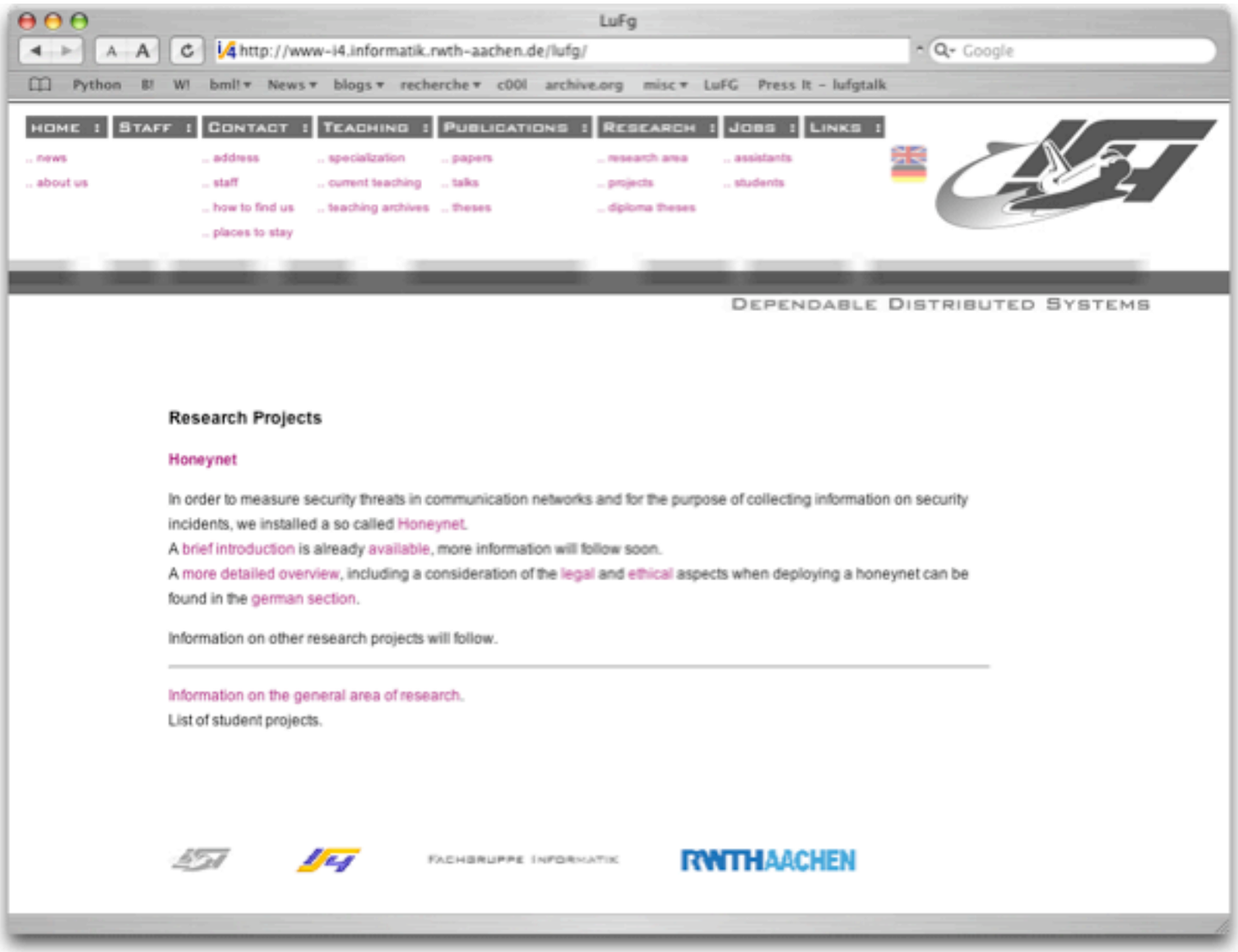




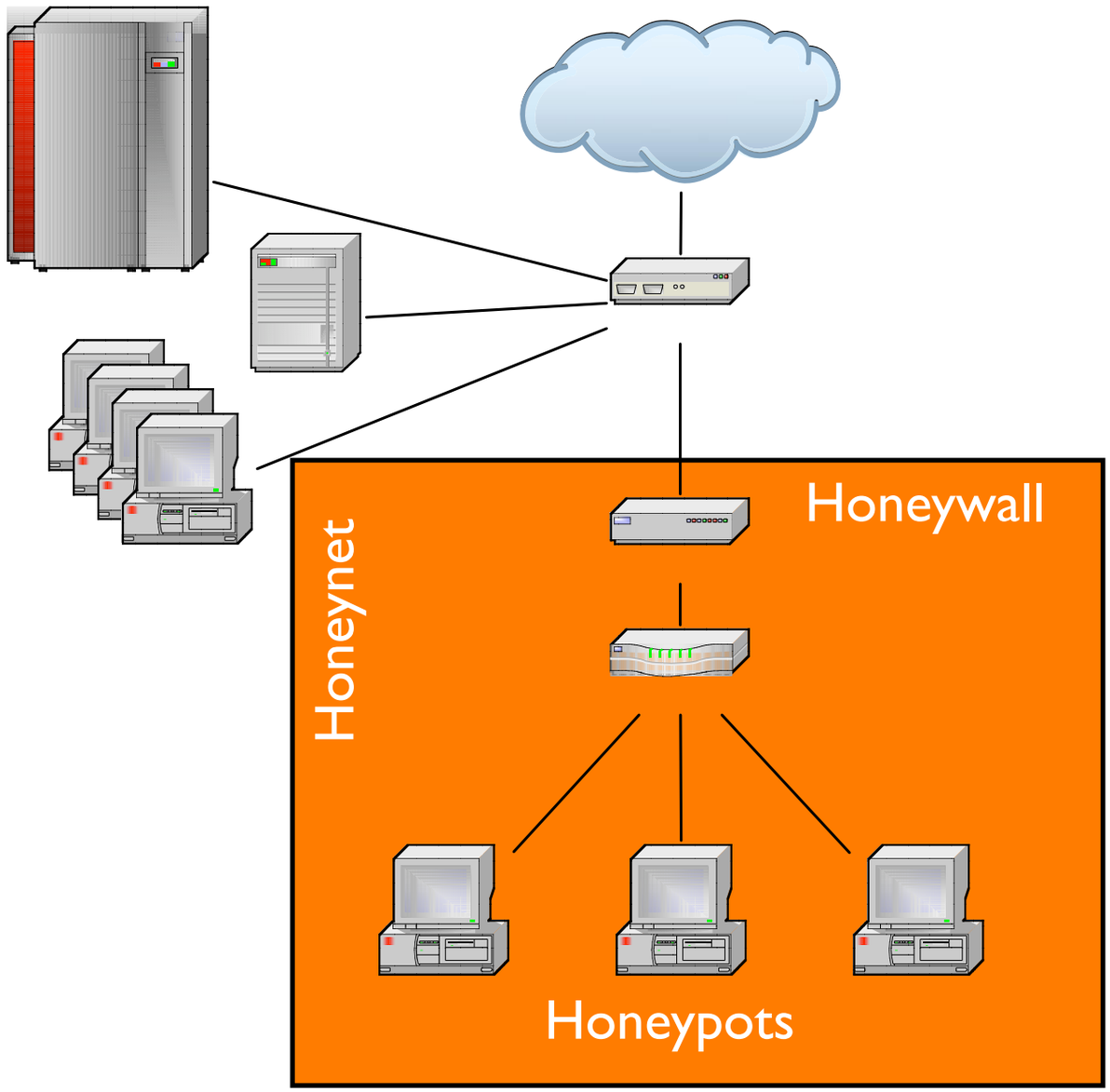
RWTH AACHEN
RHEINISCH-WESTFÄLISCHE TECHNISCHE HOCHSCHULE AACHEN



Who and Why


LuFg
http://www-i4.informatik.rwth-aachen.de/lufg/
Google
Python
HOME
STAFF
CONTACT
TEACHING
PUBLICATIONS
RESEARCH
JOBS
LINKS
.. news
.. about us
.. address
.. staff
.. how to find us
.. places to stay
.. specialization
.. current teaching
.. teaching archives
.. papers
.. talks
.. theses
.. research area
.. projects
.. diploma theses
.. assistants
.. students
DEPENDABLE DISTRIBUTED SYSTEMS
Research Projects
Honeynet
In order to measure security threats in communication networks and for the purpose of collecting information on security incidents, we installed a so called Honeynet.
A brief introduction is already available, more information will follow soon.
A more detailed overview, including a consideration of the legal and ethical aspects when deploying a honeynet can be found in the german section.
Information on other research projects will follow.
Information on the general area of research.
List of student projects.
FACHGRUPPE INFORMATIK
RWTH AACHEN

Honeynets

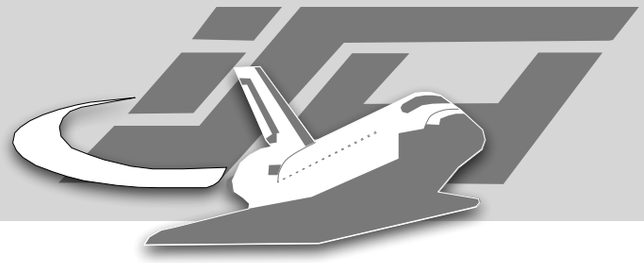


Sebek



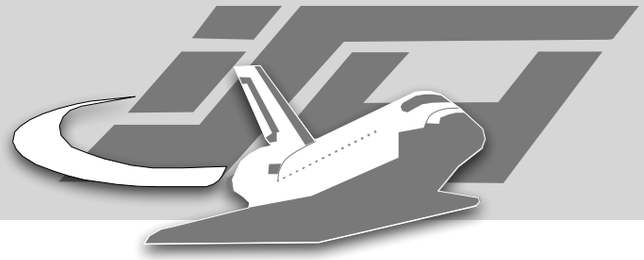
[...] monitoring capability to all activity on the honeypot including, but not limited to, keystrokes. If a file is copied to the honeypot, Sebek will see and record the file, producing an identical copy. If the intruder fires up an IRC or mail client, Sebek will see those messages. [...] Sebek also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

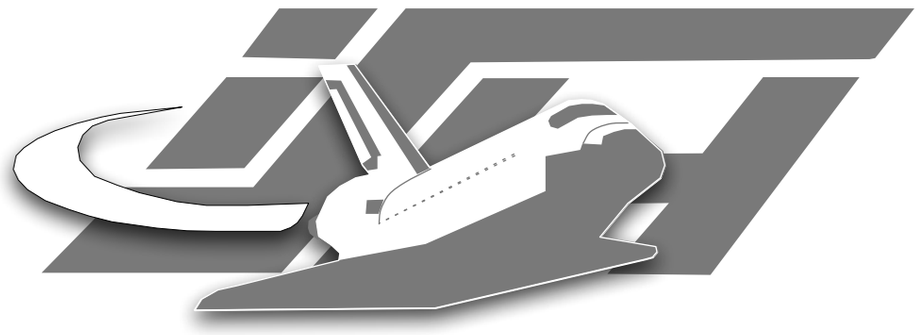
- Hijacks `sys_read()`.
- Sends data passing through `sys_read()` over the network.
- Overwrites parts of the Network stack (`packet_recvmsg`) to hide Sebek data passing on the network.
  - Packages to be hidden are identified by protocol (`ETH_P_IP`, `IPPROTO_UDP`), port and magic.



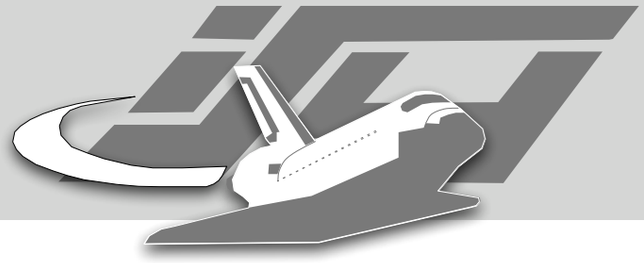
Hiding of Sebek

- 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.

Detecting Sebek



RWTH AACHEN
RHEINISCH-WESTFÄLISCHE TECHNISCHE HOCHSCHULE AACHEN



Detecting Sebek

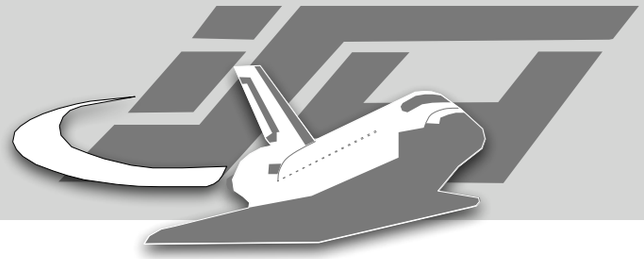
Several ways to detect a Sebek infected host come to mind:

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

Latency

- dd-attack:

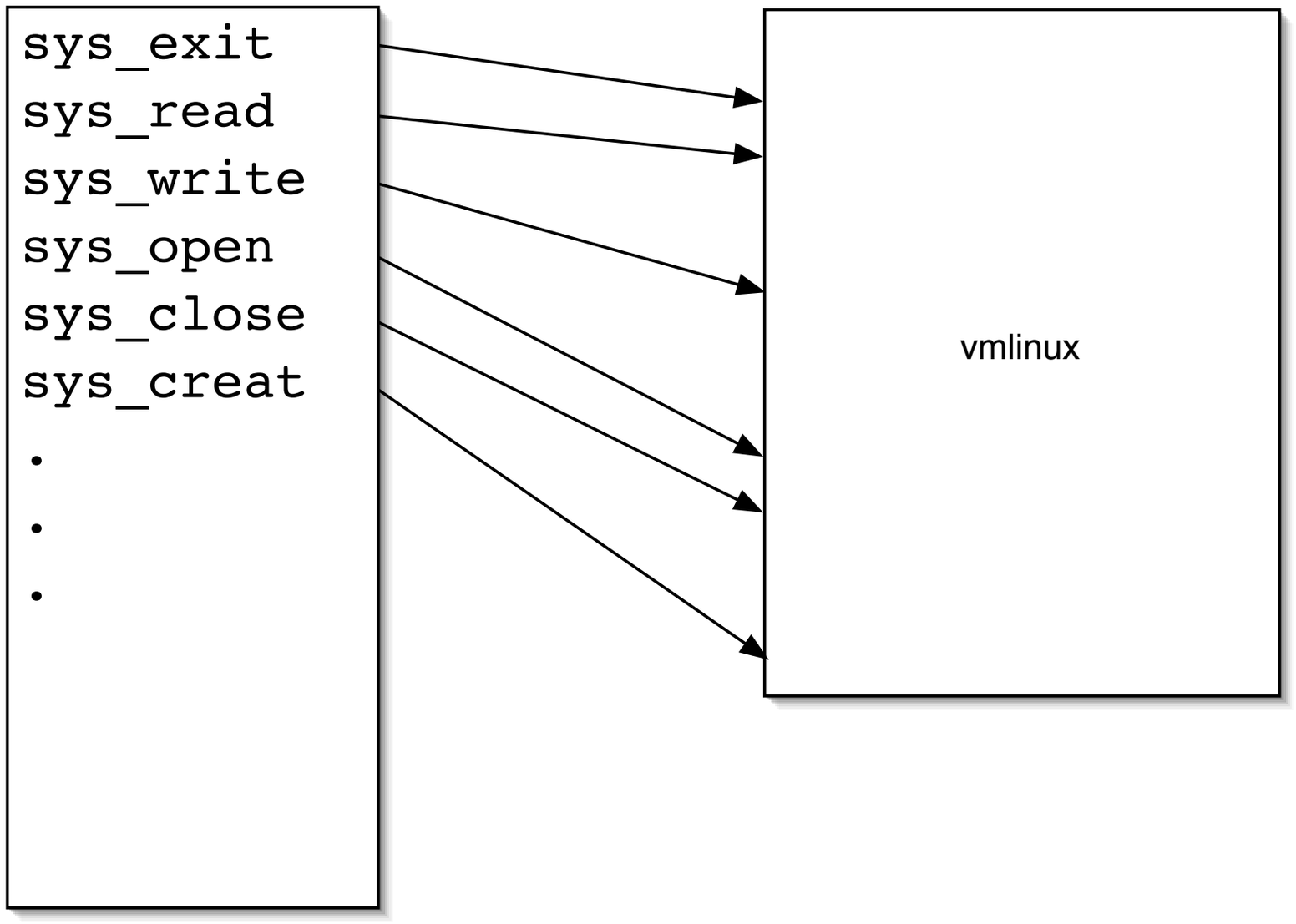
```
dd if=/dev/zero of=/dev/null bs=1
```



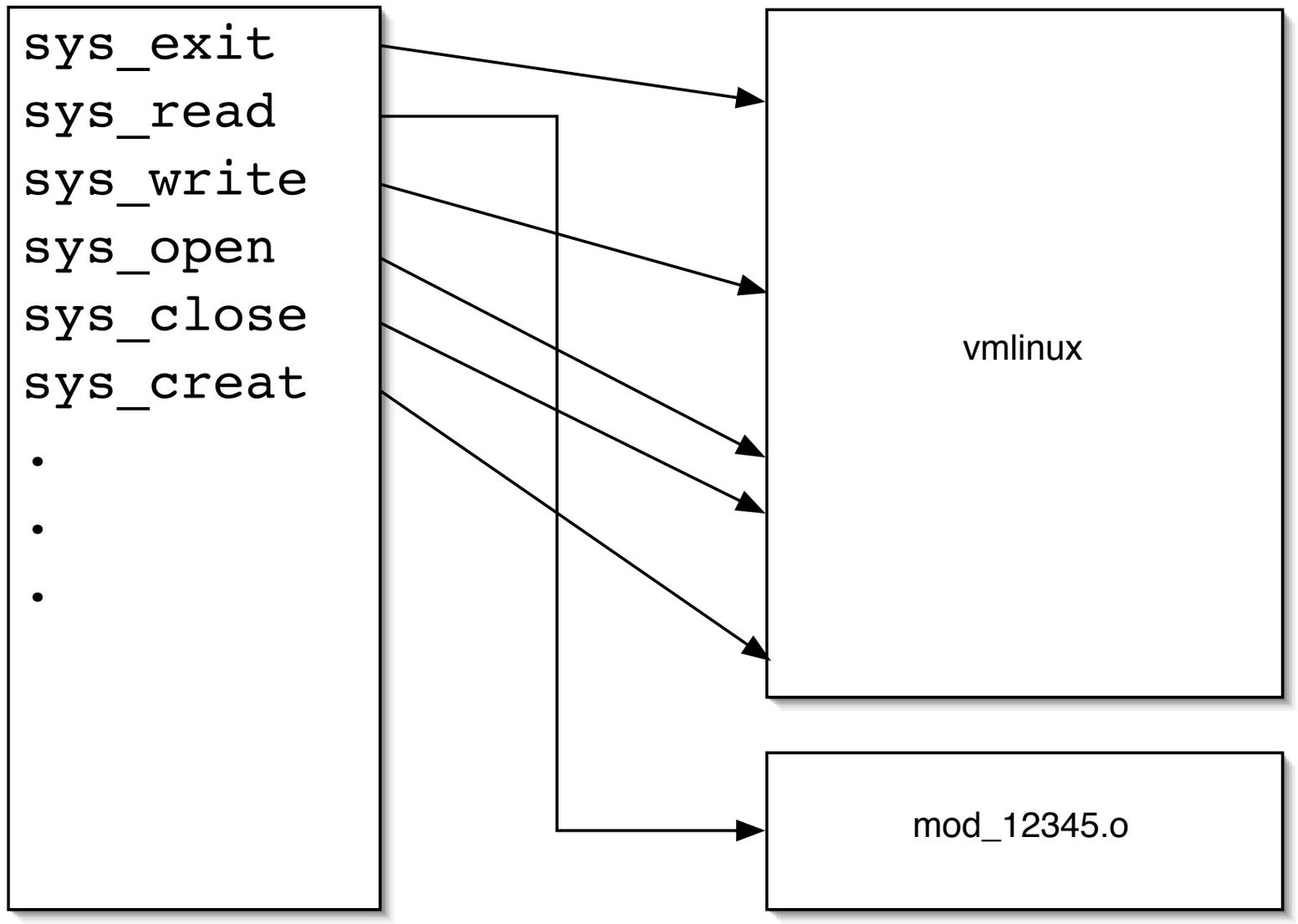
Network Traffic Counters

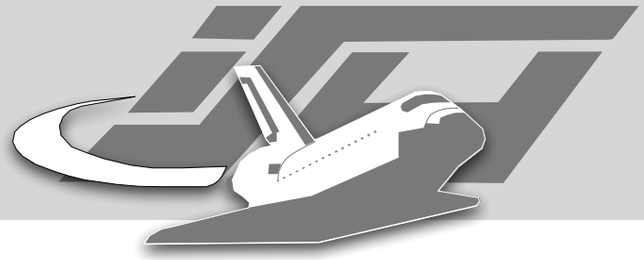
- dd-attack / backward running counters
- `dev->get_stats->tx_bytes` or `dev->get_stats->tx_packets` vs. `/proc/net/dev` or `ifconfig` output.

Syscall Table



Syscall Table





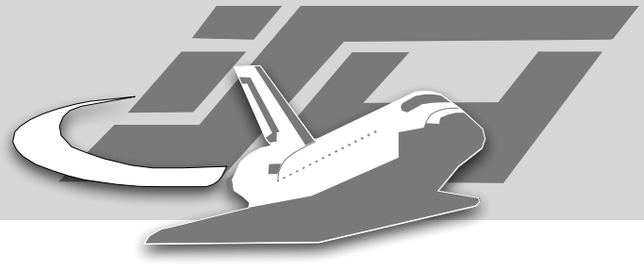
Syscall Table

before:

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

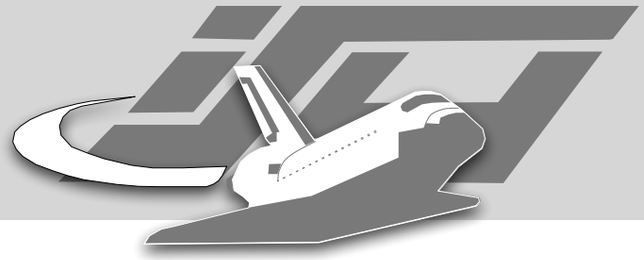
after:

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



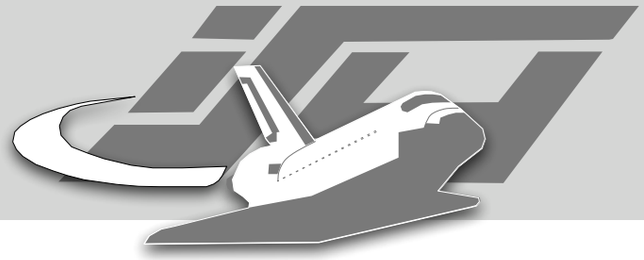
Finding Modules

- 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

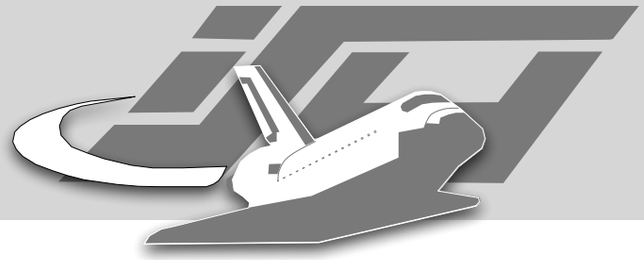
- 96
- Pointers into Kernel

```
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

- 96
- Pointers into Kernel
- Pointers into the Module

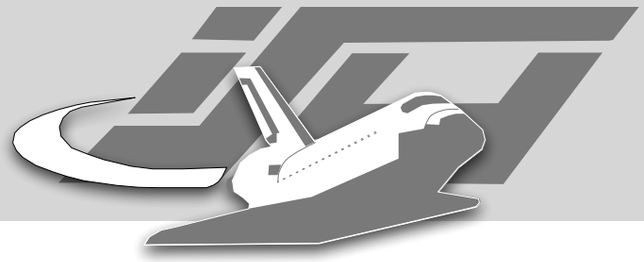
```
include/linux/module.h:
struct module
{
  unsigned long size_of_struct; /* sizeof(module) */
  struct module *next;
  const char *name;
  unsigned long size;
  union {
    atomic_t usecount;
    long pad;
  } uc;  /* Needs to keep its size - so says rth */
  unsigned long flags;          /* AUTOCLEAN 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
- 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);
```

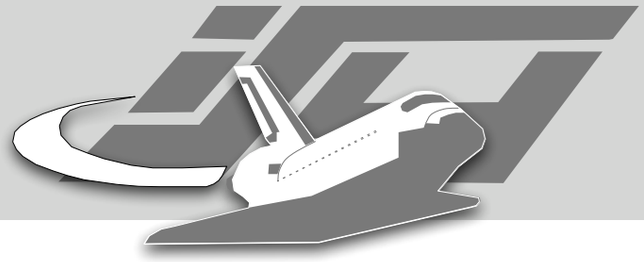


Finding Modules

- A module header is allocated by the kernels `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`.

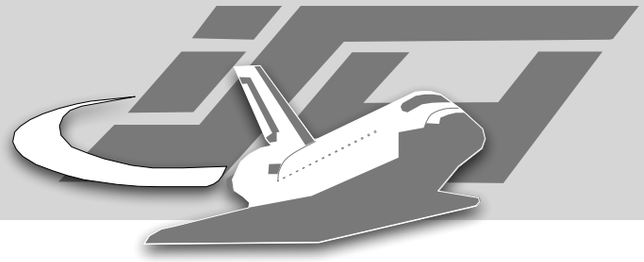
```
for(p = VMALLOC_START; \
    p <= VMALLOC_START + VMALLOC_RESERVE-PAGE_SIZE; \
    p =+ PAGE_SIZE)
```

from `module_hunter.c` by madsys



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

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

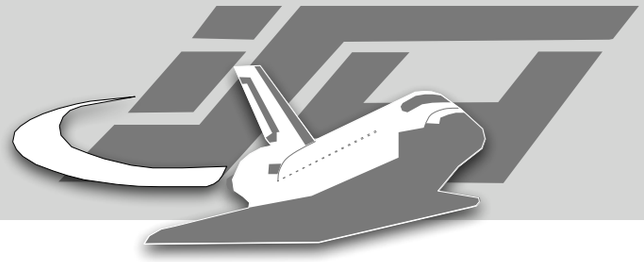
Retrieving Sebek's variables

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

IP?

~~240.1.192.222~~

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	00000000	0000003a	00000000	00000000	00000000	00000000	00000000
213.73.91.29	00000000	00000000	00000000	00000000	00000000	00000000	d5495b1d	00000000

Retrieving Sebek's variables

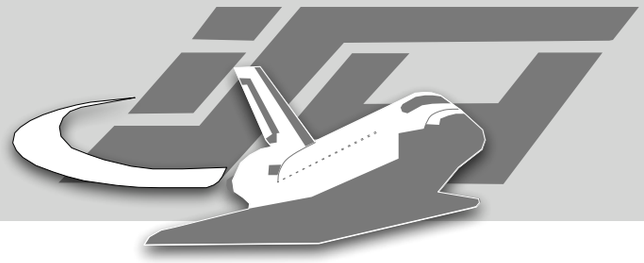
31337 Port?

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

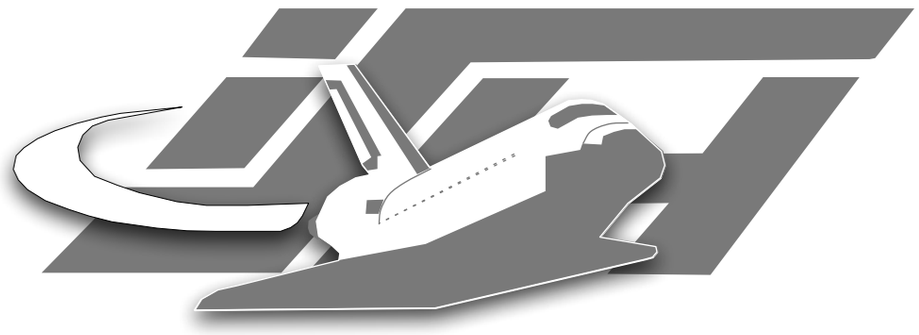


Disabling Sebek

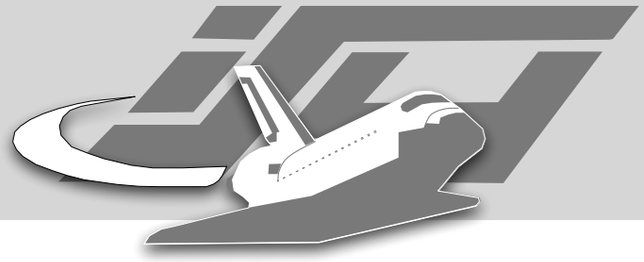
- The easy way: call `cleanup()`
- The obvious way: reconstruct `sys_read()` pointer from the kernel and fix it in the syscall table.
- The crazy way: patch in your own untainted `sys_read()`.

```
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

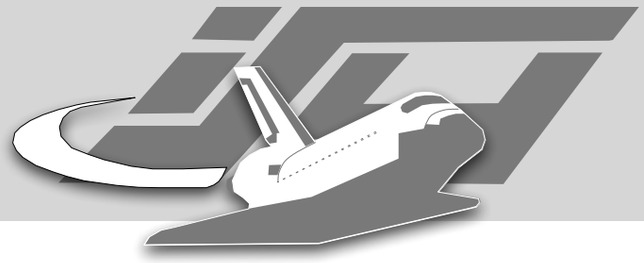


RWTH AACHEN
RHEINISCH-WESTFÄLISCHE TECHNISCHE HOCHSCHULE AACHEN



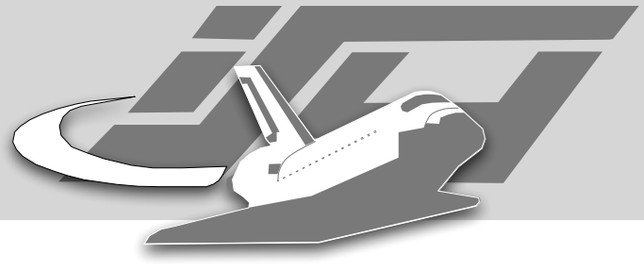
What can be logged?

- 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

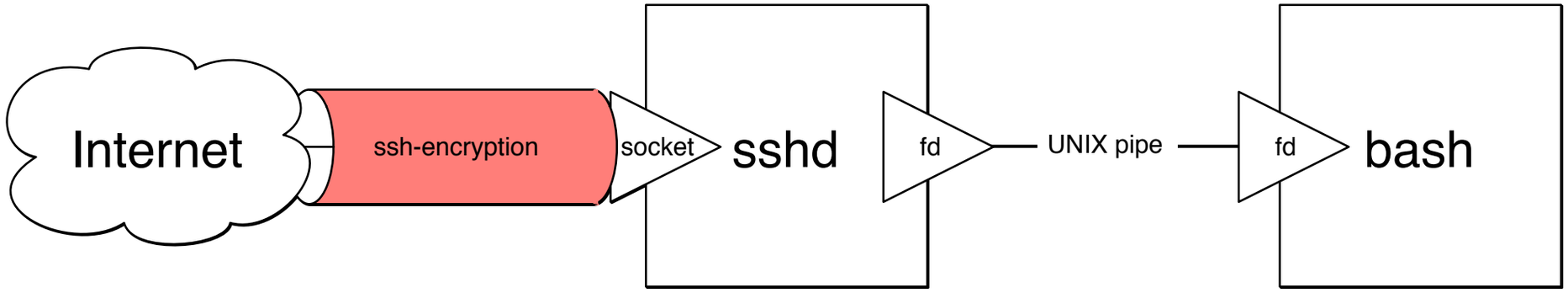
- The adversary completely controls the network. What can we do about it?
- Use encrypted communication
  - Problem: how to deliver our initial payload? HTTPS-Exploit?
- Disable the logging host or gain access to it and delete data.

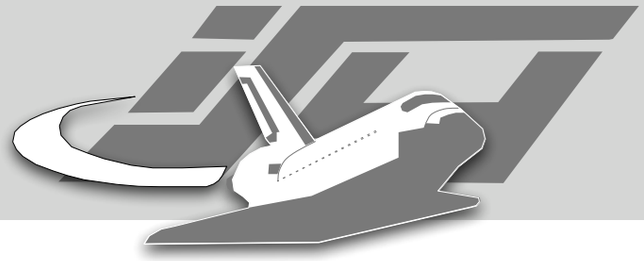


Intercepting read()

- 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()

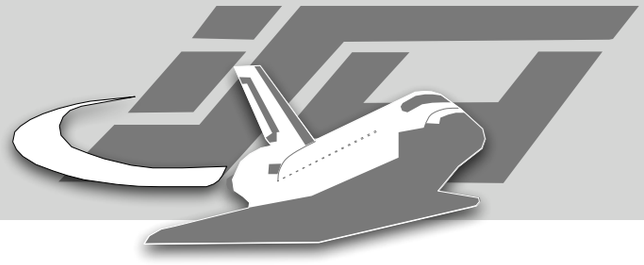




What is logged?

- data read
- pid, uid calling `read()`
- filedescriptor used
  - we can fiddle with this
- name of the progress 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
};
```

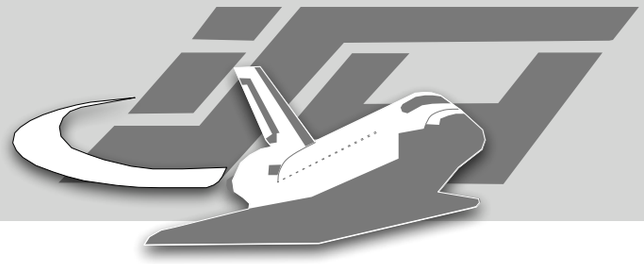


Making intercepting `read()` unrelyable

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

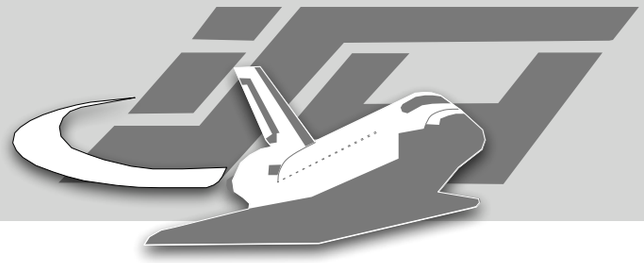
- dd-attack

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



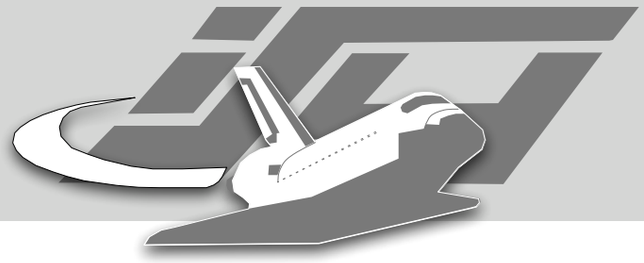
Living without read()

- 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.



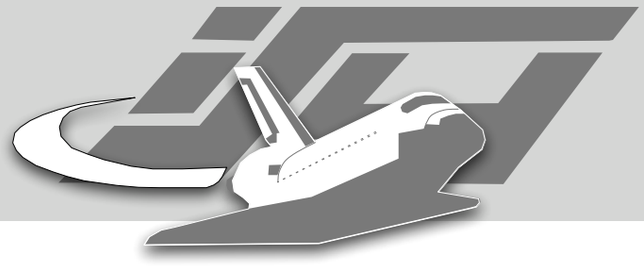
Better living without read()

- say goodby 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

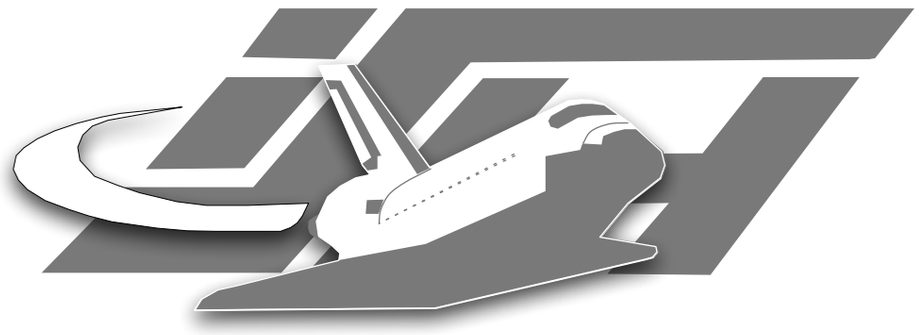
- Just copy & rename the binary.



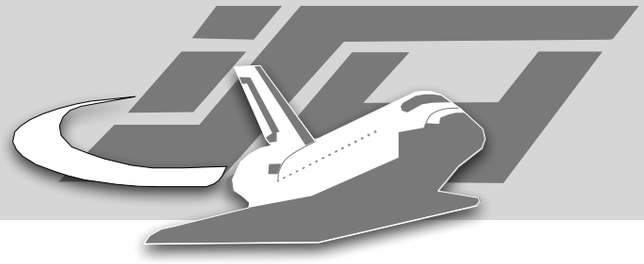
Results

- 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 programms which we don’t.

Kebes

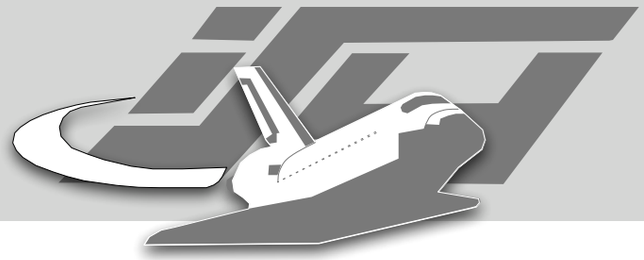


RWTH AACHEN
RHEINISCH-WESTFÄLISCHE TECHNISCHE HOCHSCHULE AACHEN



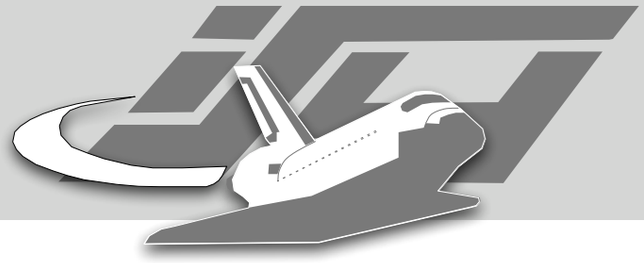
Kebes

- 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.



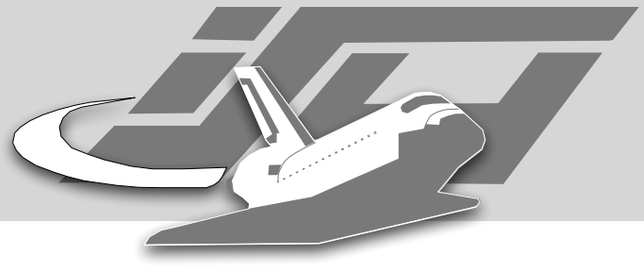
Kebes: networking

- 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
  implementing 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.



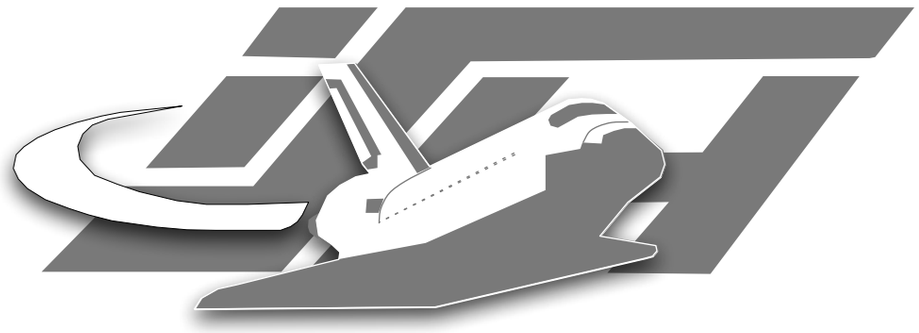
Kebes layer

- 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 NoSEBrEaK code will only exist in the server's RAM.
- Implemented commands: reading/writing files, secure deletion, direct execution, listing direcories, ...

Thank You!

Maximillian Dornseif <dornseif@informatik.rwth-aachen.de>
Thorsten Holz <holz@i4.informatik.rwth-aachen.de>
Christian N. Klein <kleinc@cs.bonn.edu>

Slides at http://md.hudora.de/presentations/#NoSEBrEaK-DC



RWTH AACHEN
RHEINISCH-WESTFÄLISCHE TECHNISCHE HOCHSCHULE AACHEN