AHA - Adaptive Honeypot Alternative

Gérard Wagener

October 29, 2010

Introduction

Related work

- Honeypots are resources designed to be under attack [5]
- End eighties / early nineties first experiments by Clifford Stoll [6], Steven Bellovin [1] and Bill Cheswick [2]
- They mainly reported how they trapped attackers and the related activities
- In 1998 Fred Cohen discussed the deception techniques that can be used while dealing with attackers [3]
- Lance Spitzner writes that honeypots are particularly useful to learn from attackers
- Jose Antonio Coret re-implemented an SSH server in python as honeypot [4]

Introduction

Attack Scenario

Step	Attacker	Honypot	Comment
0	SSH connect		Attacker penetration
1		Returns shell	Full access
2	id		System identification
3		Execute id	
4	uname		System identification
5		Execute uname	
6	ps aux		Already compromised?
7		Execute ps	
8	wget URL_0		Acquire tool
9		Execute wget	
10	$./ssh - brute$		Misuse the system
11		Return error	Strategical block
12	wget URL_1		Additional tool
13		Execute wget	
14	./configure	Build attacker tool	Source code
15		Allow	Make attacker happy

Introduction

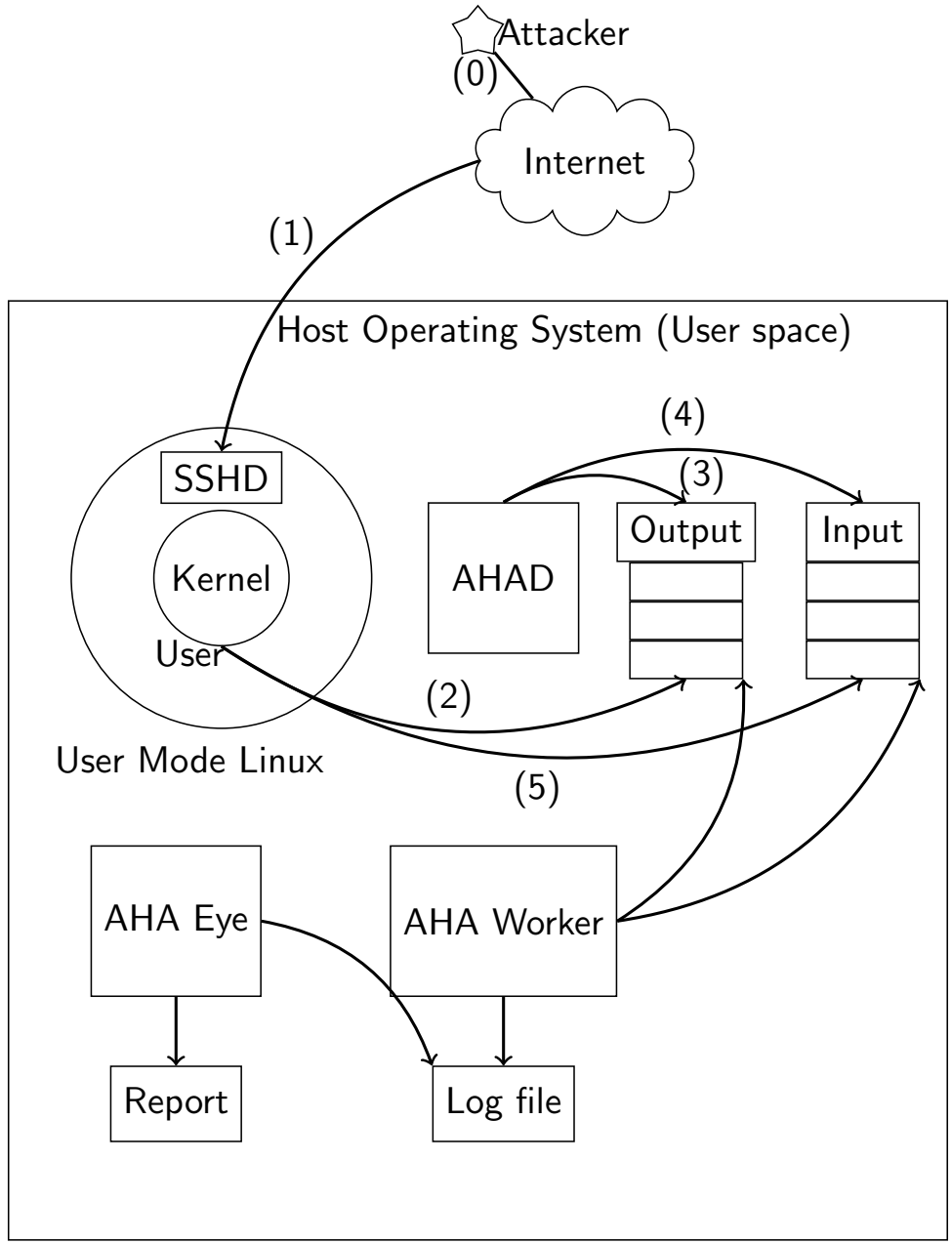
Contribution

- Create framework serving as building blocks for adaptive honeypots
- Optimize information retrieval from attackers (skills, tools, used time, social background, used language)
- Based on a Linux operating system exposing a vulnerable SSH server

Adaptation mechanisms

- Allow the execution of a program
  - Behave like a regular high-interaction honeypot
  - Do not interfere with the execution flow
- Block the execution of a program
  - Strategically block the execution of a program
  - Challenge the attacker
- Substitute the executed program
  - Make attacker believe that they downloaded the wrong program
  - Make attacker believe that their repository is not available
- Insult the attacker
  - Irritate attacker
  - Reveal his ethic background
  - Differentiate between automated attacks and human attackers
  - See if attackers bounce through compromised hosts

AHA framework - Overview



Components interaction

- Linux system call hooks
  - sys_execve
  - sys_clone
  - sys_exit
- Send messages to AHA daemon
- A decision must be taken (not included in the framework)
- Exchange Messages
  - Export message $\rightarrow$ export kernel information to the daemon
  - Reply message $\rightarrow$ decision taken by the daemon
  - Export and reply messages are tighly linked $\rightarrow$ unique message identifier

Components interaction

```
1  type=1
2  file=/usr/bin/vi
3  argument=vi
4  env=TERM=screen
5  env=SHELL=/bin/bash
6  env=SSH_CLIENT=192.168.1.2 41836 22
7  env=SSH_TTY=/dev/pts/0
8  env=USER=gabriela
9  env=PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin
10 env=LANG=en_US.UTF-8
11 env=HISTCONTROL=ignoreboth
12 env=SHLVL=1
13 env=HOME=/home/gabriela
14 env=LOGNAME=gabriela
15 env=SSH_CONNECTION=192.168.1.2 41836 192.168.1.1 22
16 env=_=/usr/bin/vi
17 pid=1100
18 ppid=1075
19 rppid=1075
20 DONE=1
```

Reply message

```
1  struct ReplyMessage{
2      int block;
3      int substitue;
4      int insult;
5  };
```

Components description

User Mode Linux surgeries

Building an UML from a vanilla kernel

```
1 make defconfig ARCH=um
2 make ARCH=um
```

Modified kernel files

File	Function
arch/um/kernel/exec.c	`sys_execve`
arch/um/kernel/process.c	`exit_thread`
arch/um/sys-i386/syscalls.c	`sys_clone`
os-Linux/main.c	`__init`

Purpose: Export program execution data and let the daemon take the decisions

Components description

User Mode Linux surgeries

Sys_execve hook

```
long sys_execve(char __user *file, char __user *__user *argv,
        char __user *__user *env)
{
    long error;
    char *filename;
    struct ReplyMessage msg;
    filename = aha_dump_execve(file,argv,env);
    if (filename){
        aha_get_reply_message(filename,&msg);
        kfree(filename);
        /* Implement decisions taken by AHA */
        if (msg.block) {
            error = msg.block;
            goto out;
        }
        if (msg.insult) {
            aha_handle_insult_messages(&msg,file,argv);
        }else {
            if (msg.substitute) {
                aha_handle_substitutes(&msg,file,argv);
            }
        }
   }
```

Components description

AHA daemon

Operation

- Read messages initiated by the User Mode Linux
- Is the program execution related to an attacker?
- Take a decision and put it in the input queue

Code Organization

- AHAActions → core functions to interact with the User Mode Linux
- KERNEL_ERRORS → Strategical blocking (taken from the Kernel Source)
- ReplyMessage → Create a binary reply message for the User Mode Linux
- ProcessTree → Maintain in the daemon a clone of the process tree of the UML

Components description

AHA Worker

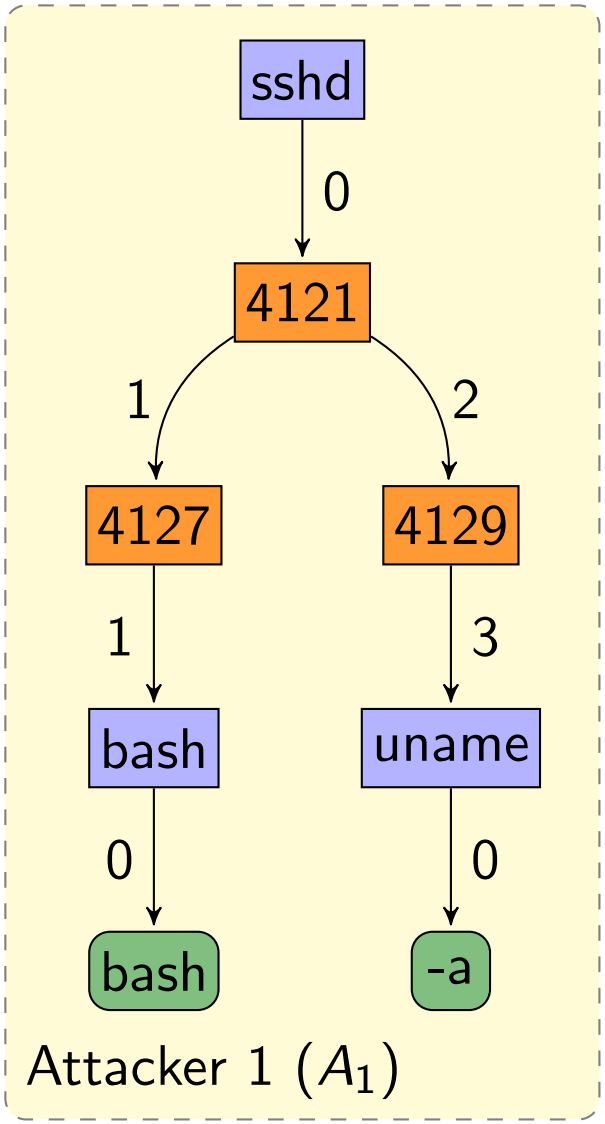
- Execution performance is critical
- AHA daemon only takes decisions
- AHA Worker periodically polls the queues
- Merges messages in a log file
- Avoid overfilled queues

Components description

AHA Eye

- Monitoring is essential for honeypot operation
- Human readable form is desired
- AHA Eye uses the log file from AHA Worker
- Creates a report $\rightarrow$ attacker's bash session

Is the program related to an attacker or the system?



Legend

Program name

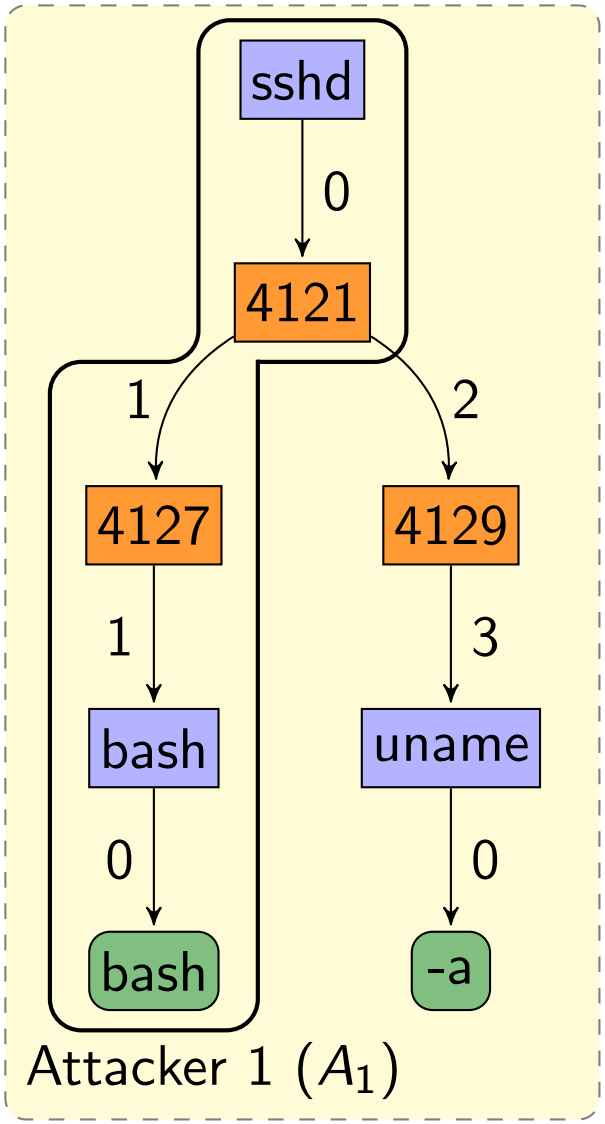
PID

Command line argument

Classified programs

$\sum ts$	Program name
0	*sshd*
2	
5	

Is the program related to an attacker or the system?



Legend

Program name

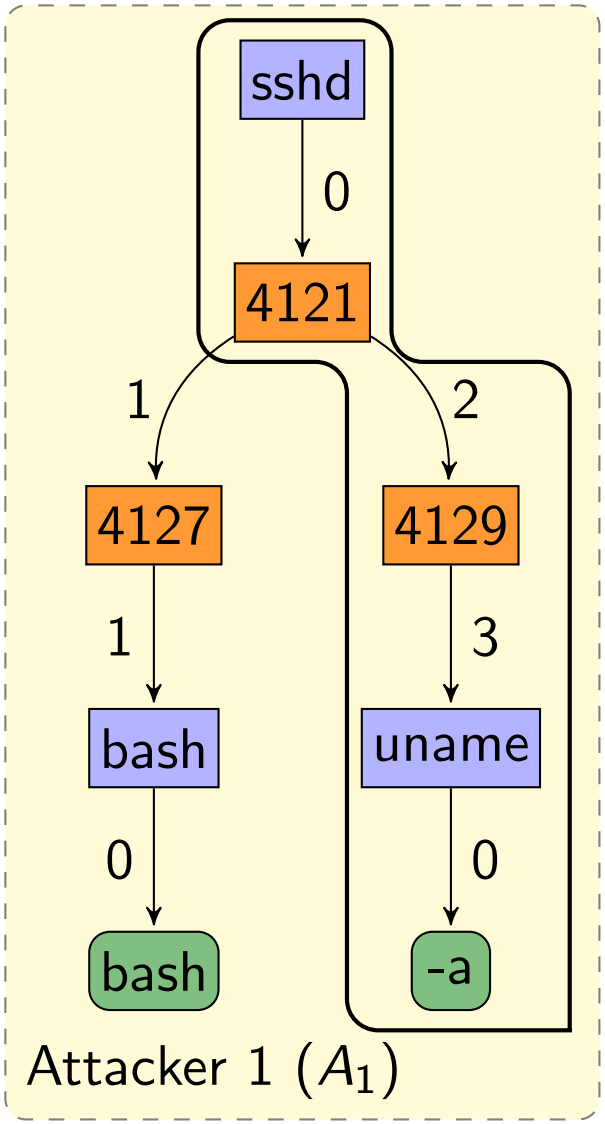
PID

Command line argument

Classified programs

$\sum ts$	Program name
0	*sshd*
2	*bash*
5	

Is the program related to an attacker or the system?



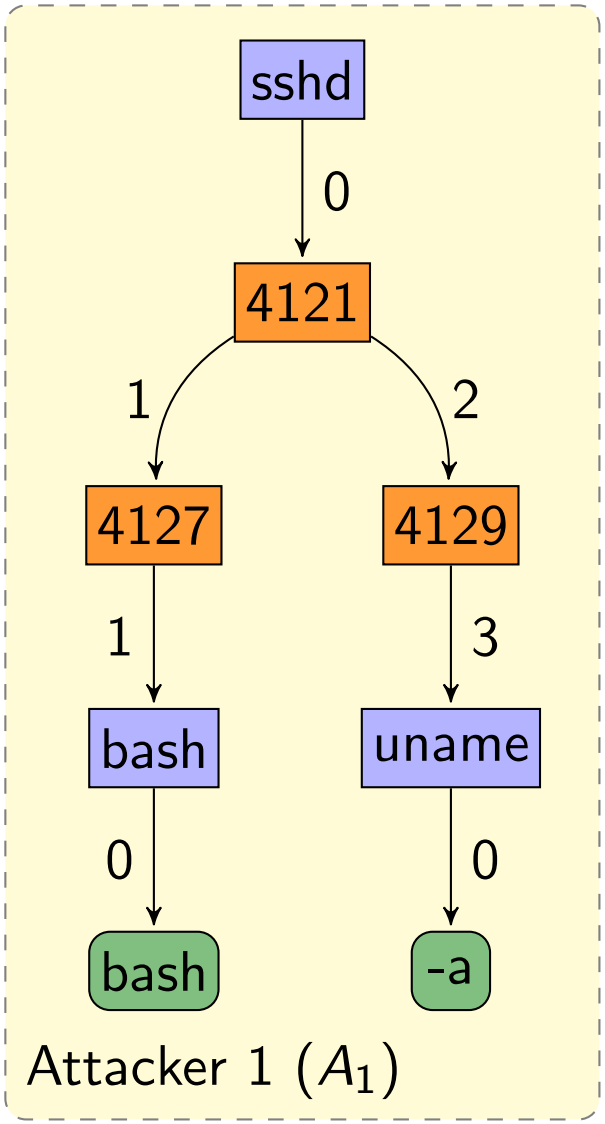
Legend

- Program name
- PID
- Command line argument

Classified programs

$\sum ts$	Program name
0	*sshd*
2	*bash*
5	*uname*

Is the program related to an attacker or the system?



Legend

- Program name
- PID
- Command line argument

Classified programs

$\sum ts$	Program name
0	*sshd*
2	*bash*
5	*uname*

$\vec{A_1} =< sshd, bash, uname >$

Insulting the attacker

```
1  void aha_handle_insult_messages(struct ReplyMessage *msg,
2                                  char __user* file,
3                                  char __user* __user* argv)
4
5      char buf[16];
6      char* addr;
7      int cnt;
8
9      if(!copy_to_user(file,"/sbin/insult",13)){
10         cnt = snprintf((char*)&buf,16,"%d",msg->insult);
11         if ((cnt > 0) && (cnt<15))
12             buf[cnt+1]=0;
13             if (!get_user(addr,argv))
14                 copy_to_user(addr,buf,cnt+1);
15      }
16
```

Substituting programs works similarly

Gathering insults from an attacker

Problem

- Insults = invalid programs
- Handled by Bash

Solution (ugly)

- Hook bash using the `NOTFOUND_HOOK`
- Use helper application that just accepts the arguments
- When the helper application is started a `sys_execve` is induced
- This is then visible for the AHA daemon

Case Studies

Example Session (94.52.64.x username: test)

w

. .. scbrute.tar .wp

w

18:28:21 up 6:46, 1 user, load average: 0.15, 0.03, 0.01

bash

I dont wanna do that

sh

wget http://www.dragutrau.xxx.su/xxx/yyy

I love you

kill -9 1

Core dumped

l

. .. scbrute.tar .wp

fetch

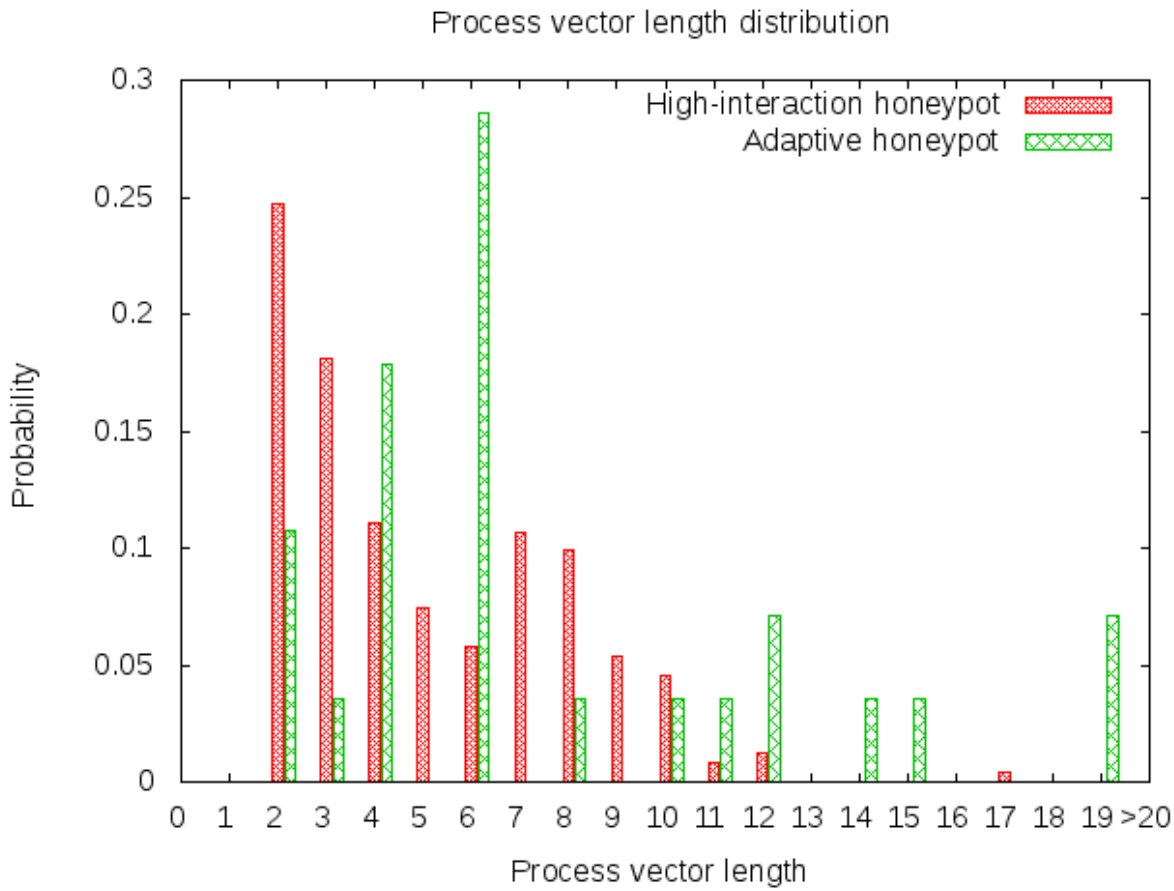
fuck you

exit

Case Studies

Experiment #1

Adaptive Honeypot vs High-Interaction Honeypot



Case Studies

Experiment #2

Insult Analysis

Language	Proportion	Country Code	Proportion
Undefined	51.8 %	RO	47%
Typographic errors	17.1%	DE	16%
Romanian	11.8%	ES	4%
English	9.2%	LU	4%
Smiley	5.3%	IT	4%
Slovak	5.3%	MK	4%
Croatian	1.0	LB	3%
Polish	1.0%	NL	2%
German	0.2%	GB	1%
others	33.06%	others	15%

Examples

muie, sex, fuck me, gogo, beto,hahahah, :)), pla, sugeo, please, sucker, bine, ?, noaon, qwerty ...

Future work and conclusions

- Future work
  - Execution slow-down $\rightarrow$ AHA is slower than an high-interaction honeypot
  - Evaluate timing attacks
  - Explore faster interprocess communication techniques
  - Insult program needs to be protected with rootkit techniques
  - Substituting a program can crash the program when the stack frame is too small
  - Vulnerable against indirect attacks $\rightarrow$ let the system continue the attack
  - Tests with the SKAS patch could be done
  - tty_read and tty_write could be monitored $\rightarrow$ insights about keystrokes
  - Instrument a virtual machine instead of User Mode Linux

Future work and conclusions

- Conclusions
  - Honeypots should become more intelligent and adaptive
  - Optimize information retrieval from attackers
  - Created an adaptive honeypot framework to investigate learning techniques
  - Extended User Mode Linux
  - Each system call related to program execution needs to be acknowledged by the AHA daemon
  - Freely available at git.quuxlabs.com

Demo in progress ...

Thank you for your patience ...

Questions and Answers

Thank you for your attention
Questions?
Comments for improvement?

Bibliography

Steven M. Bellovin.

There be dragons.
In *Proceedings of the Third Usenix Unix Security Symposium*, pages 1–16, September 1992.

Bill Cheswick.

An evening with Berferd in which a cracker is lured, endured, and studied.
In *In Proc. Winter USENIX Conference*, pages 163–174, 1992.

Fred Cohen.

A note on the role of deception in information protection.
Computers & Security, 17(6):483–506, 1998.

Jose Antonio Coret.

Kojoney - a honeypot for the SSH service.
http://kojoney.sourceforge.net/.

L. Spitzner.

Honeypots: Tracking Hackers.
Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2002.

Clifford Stoll.

Stalking the wily hacker.
Commun. ACM, 31(5):484–497, 1988.