Equo ne credite: Trojans

Bonn-Aachen International Center
for Information Technology

Seminar Malware

03. December 2007

Daniel Rosenthal

Contents

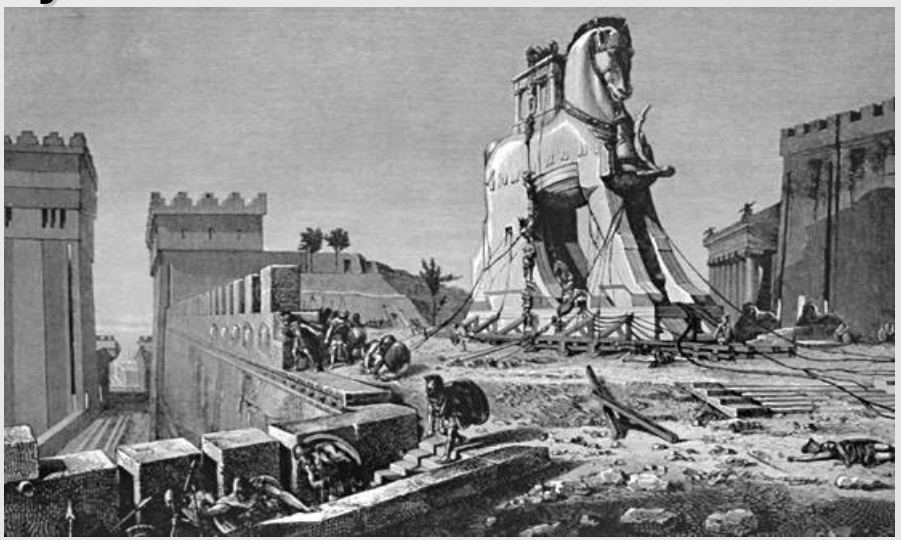
- Some expamples
- Definition of a Trojan Horse
- Trojans in compilers
- Harrier as part of HTH framework
- Summary
- References

Contents

- Some expamples
- Definition of a Trojan Horse
- Trojans in compilers
- Harrier as part of HTH framework
- Summary
- References

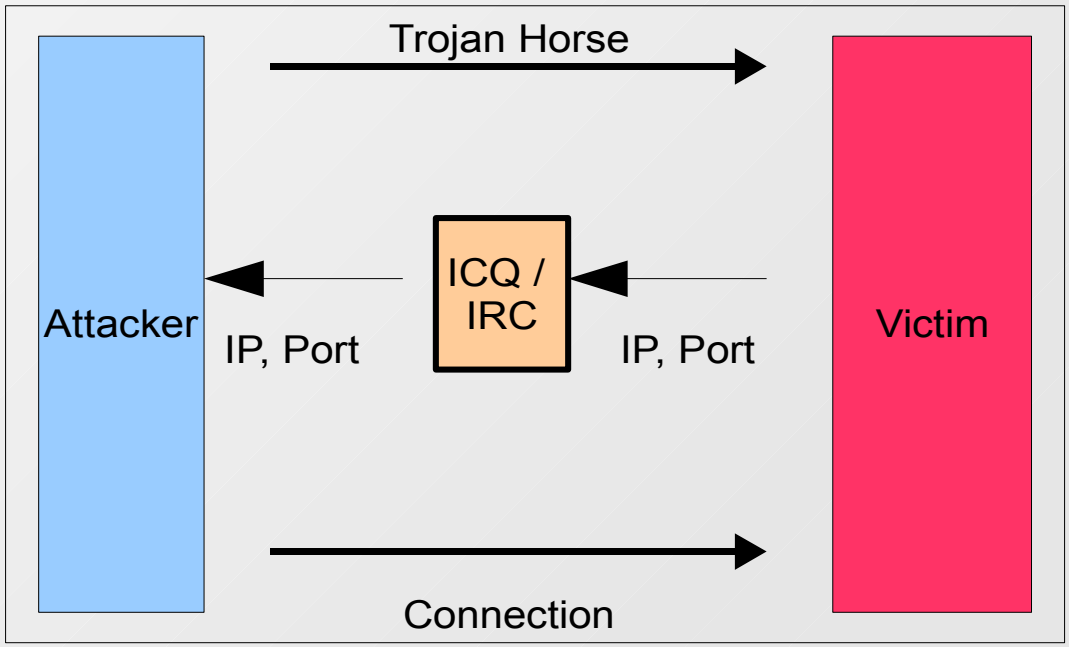
Expamles of Trojan Horses (1/6)

- Greek mythology
  - Trojan war
  - After 10 years of siege, the Greek built a wooden horse, inside some soldiers and left the battlefield
  - Trojans expected the horse to be a present and carried it into the city
  - At night, the hidden soldiers opend the gates, the greek army entered Troy and defeated the Trojans



Expamles of Trojan Horses (2/6)

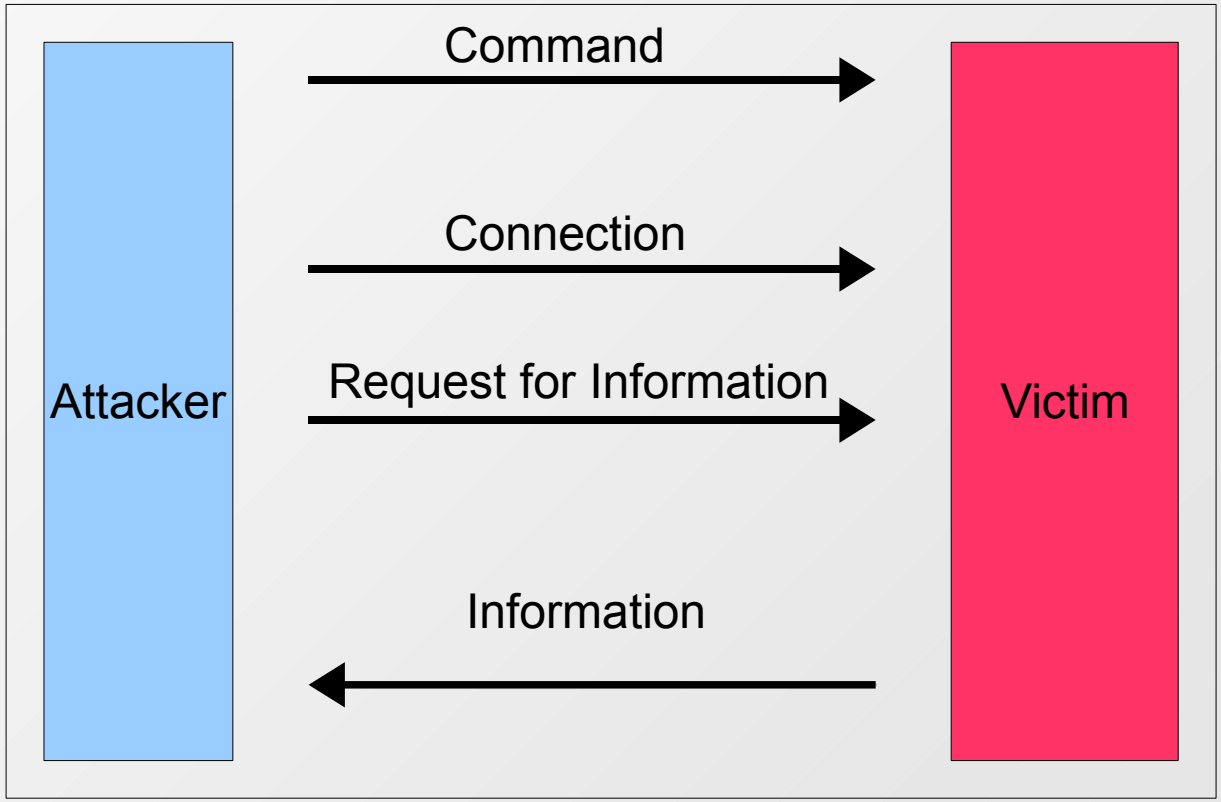
- Back Orifice (~1998)
  - developed by the “Cult of the Dead Cow“
  - released at DefCon 6 in 1998
  - affects Windows 95 and 98
  - modular system of plugins
    - *authentication and encryption possible*



Infection stage

Expamles of Trojan Horses (3/6)

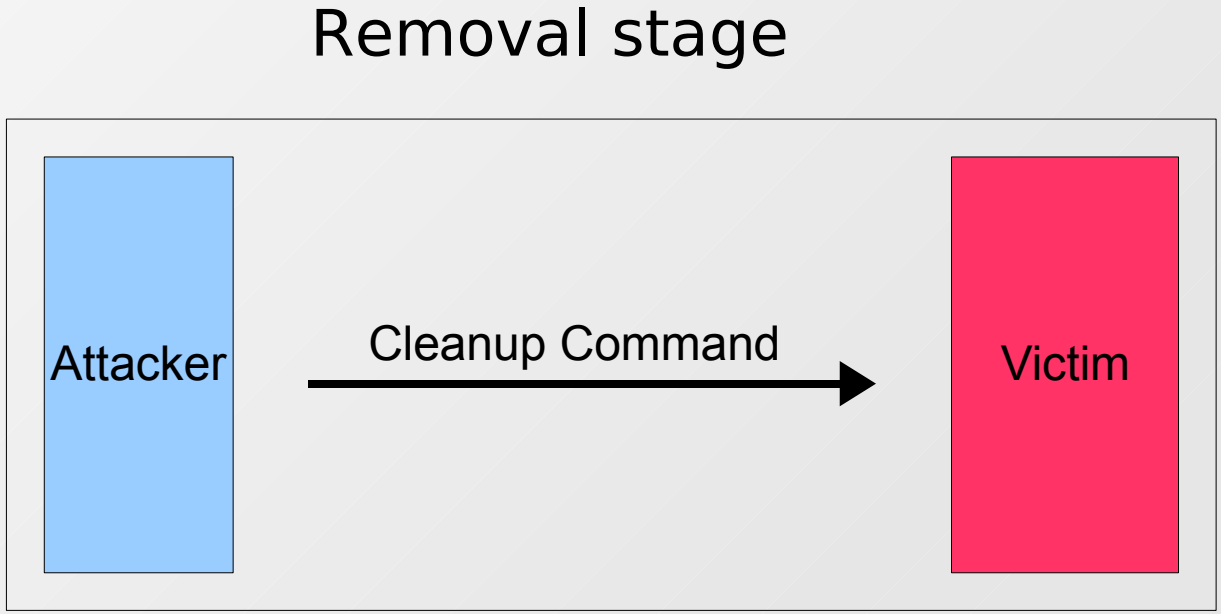
- Back Orifice



Execution stage

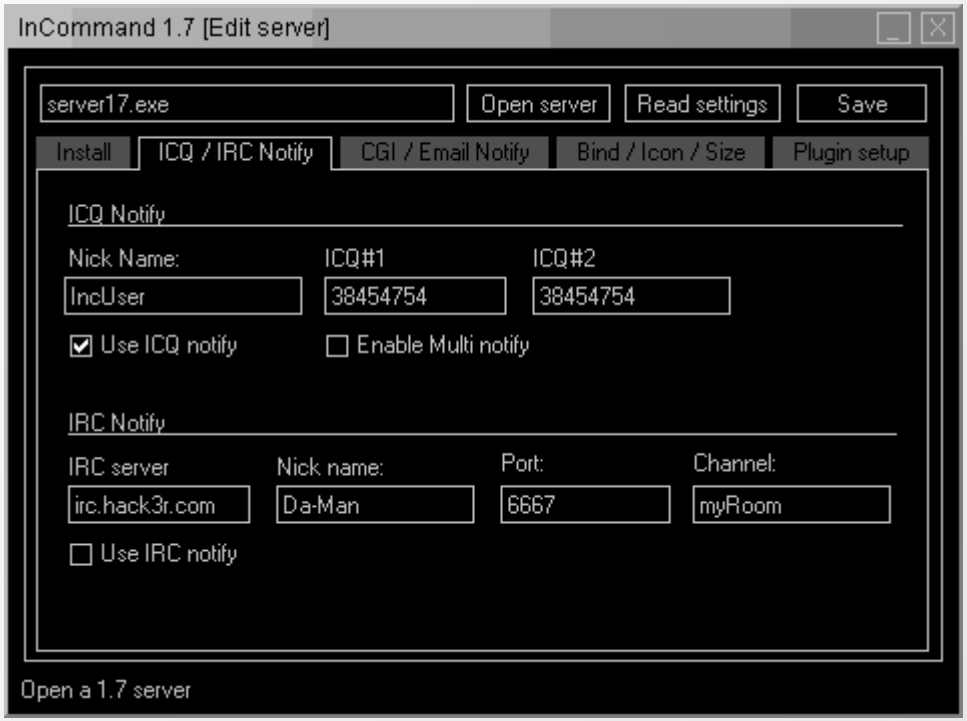
Expamles of Trojan Horses (4/6)

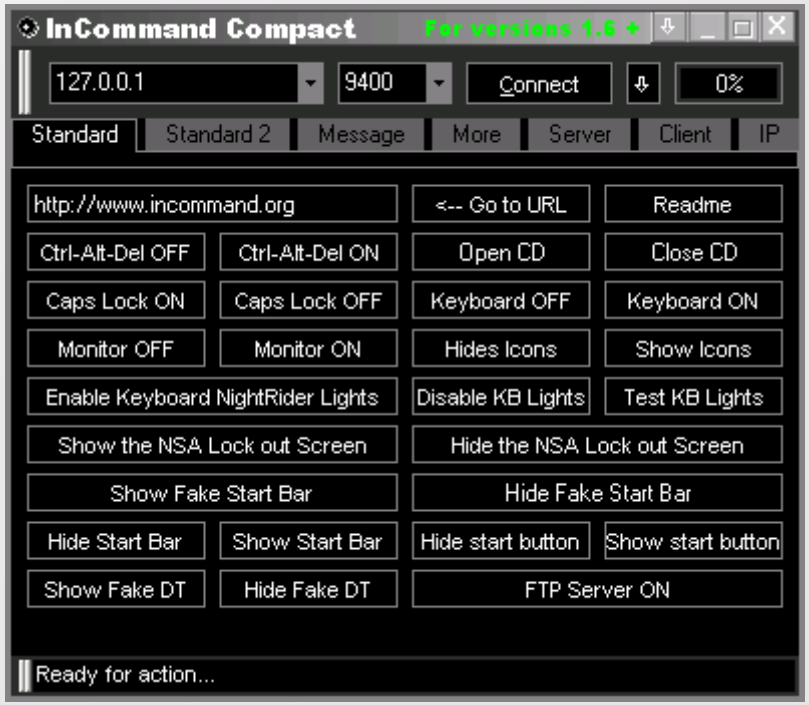
- Back Orifice



Expamles of Trojan Horses (5/6)

- InCommand (~July 2002) by Stoner and Bogart

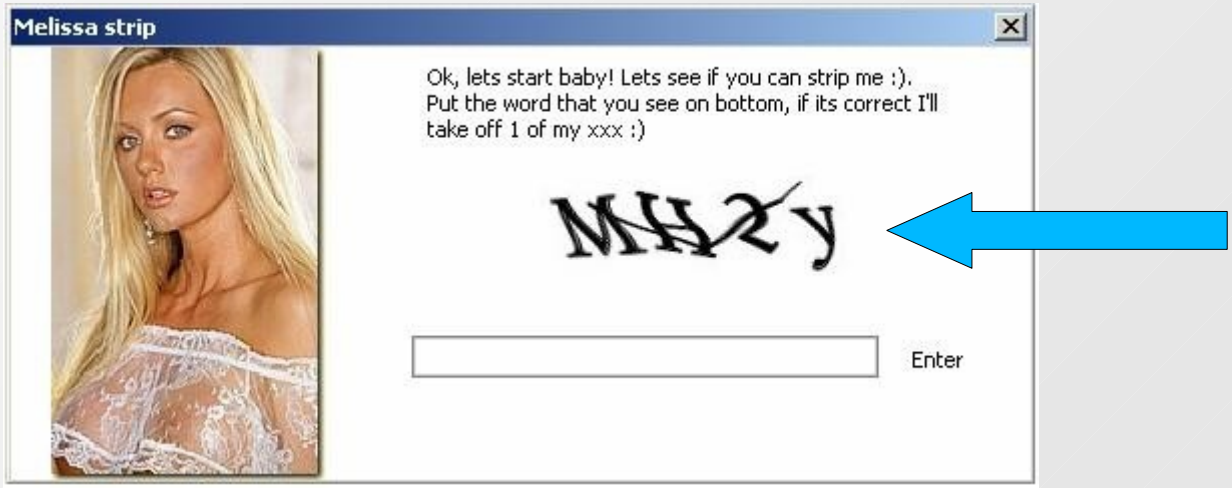




Files: editserver.exe, icon.dll, server17-b2.exe, incsrv.exe in Windows\

Expamles of Trojan Horses (6/6)

- Captcha-Breaker (29.10.2007)



• Captcha = Completely Automated Public Turing test to tell Computers and Humans Apart
• Spammers should be avoided

Contents

- Some expamples
- Definition of a Trojan Horse
- Trojans in compilers
- Harrier as part of HTH framework
- Summary
- References

Definition: What is a trojan horse ?

"An apparently useful and innocent program containing additional hidden code which allows the unauthorized collection, exploitation, falsification, or destruction of data."

from: Texas State Library and Archives Commission
http://www.tsl.state.tx.us/ld/pubs/compsecurity/glossary.html

- no self reproduction
- user must run the trojan horse program

A little more history (1/2)

- Trojan horses known since joint use of mainframe computers
  - pay per CPU time
    - *sniff username/ password by faked login screen*
    - *use account of someone else*
- Internet Service Provider: AOL
  - sniffed accounts
- Mostly Microsoft DOS and Windows systems harmed
  - huge distribution, low security standards

A little more history (2/2)

- Nowadays:
  - capturing private/confidential data
    - *online banking*
  - manipulation/ deletion of data and/or services
    - *even within a (local area) network*
  - remote access to machines
    - *Sub7even, Back Orifice*
  - mostly sended as email attachments

Characteristics

- Propagation Methods — P
- Activation — A
- Placement — H
- Effectiveness — E
- Communication — C
- Functions — F
- Guarding Mechanisms — G

- Describtion of a trojan horse as a tupel:
- T = (P|G, A|G, H|G, E|G, C|G, F|G)

Propagation Methods P

- p_1:Executables
  - $p_{1,1}$: Email attachment
  - $p_{1,2}$: Instant messaging
  - $p_{1,3}$: File sharing
  - $p_{1,4}$: FTP / HTTP
  - $p_{1,5}$: Wireless communication
  - $p_{1,6}$: Data mediums (Floppies, USB-Sticks, ...)
- p_2:Social Engineering
- p_3:Exploits
- p_4:Malformed data objects
- p_5:Physical access to computer

Activation A

- a_1: startup of operating system
  - Starting scripts / programs
  - Entries of registry (Windows)
  - Kernel module
- a_2: running a program (unintentionally)
  - Modified programs
  - Using mix-ups (unix „cp“ ⟺ windows „copy“)
  - Execution of programs treated by social engineering

Placement H

- h_1: as file somewhere on the mediums
- h_2: indepentent of the file system on the harddisk
  - marked as bad clusters
  - using free space in used clusters
  - outside of the partition in free space of harddisk
- h_3: in modules / memory of any hardware (RAM, Flash, USB-Stick, ...)
- h_4: distributed in several files

Effectiveness E

- e_1: DLL-injection (dynamic link library)
- e_2: process injection / code injection
- e_3: modifications to configurations
- e_4: loading of program modules (puzzle trojan horse)

Communication C

- c_1: active communication
  - open port (waiting / polling server)
  - closed port (port knocking)
  - stealth method (sniffer)
- c_2: passive communication
- c_3: email, IRC, ICQ, http
- c_4: tunneling (ICMP, DNS, HTTP)

Functions F

- f_1: file manager
- f_2: process manager
- f_3: keylogger
- f_4: update function
- f_5: registry
- f_6: gathering informations
- f_7: spying
- f_8: starting / providing services
- f_9: portscanner
- f_{10}: attacks to other systems
- f_{11}: destroying hardware
- f_{12}: adware

Guarding Mechanisms G

- g_1: none
- g_2: armoring
- g_3: polymorphism
- g_4: stealth
- g_5: stenography
- g_6: encryption
- g_7: manipulation of (antiviral) software

Application of the tuple specification

- Example:
- $T = (\{p_{1,1}, \{a_{1,2}, a_3\}, h_1, \{\}, c_{1,1}, \{\}\} | g^*)$

- email attachment ($p_{1,1}$)
- Activation through executing files (a_3) using a registy entry ($a_{1,2}$)
- stored in the filesystem (h_1)
- communication over an open TCP port ($c_{1,1}$)
- Unspecified self protection method for all tuple elements (g^*)

Contents

- Some expamples
- Definition of a Trojan Horse
- Trojans in compilers
- Harrier as part of HTH framework
- Summary
- References

Trojans in compilers

- Demonstration by Ken Thompson (inventor of Unix) in Turing Award lecture 1984
- Trojan Horse in C compiler binary inplementation
  - not visible in compiler source code,
  - but reproducing itself when source code is recompiled in a bootstrapping process
  - intruding back-door into the Unix „login“ command
- will pass nearly every test
  - state of the art compiler validation and verification
  - bootstrap test
  - any amount of source code inspection and verification
  - might cause a catastrophe

Trojans in compilers – How is this possible ? (1/2)

- Example: self reproducing progam (by substitution)

```
main(){
char *b = "main(){
char *b = %c%s%c;
printf(b,34,b,34);
}";
printf(b,34,b,34);
}
```

```
%c : replace character
%s : replace string
34 : "
```

Trojans in compilers – How is this possible ? (2/2)

- Example: conditional self reproducing

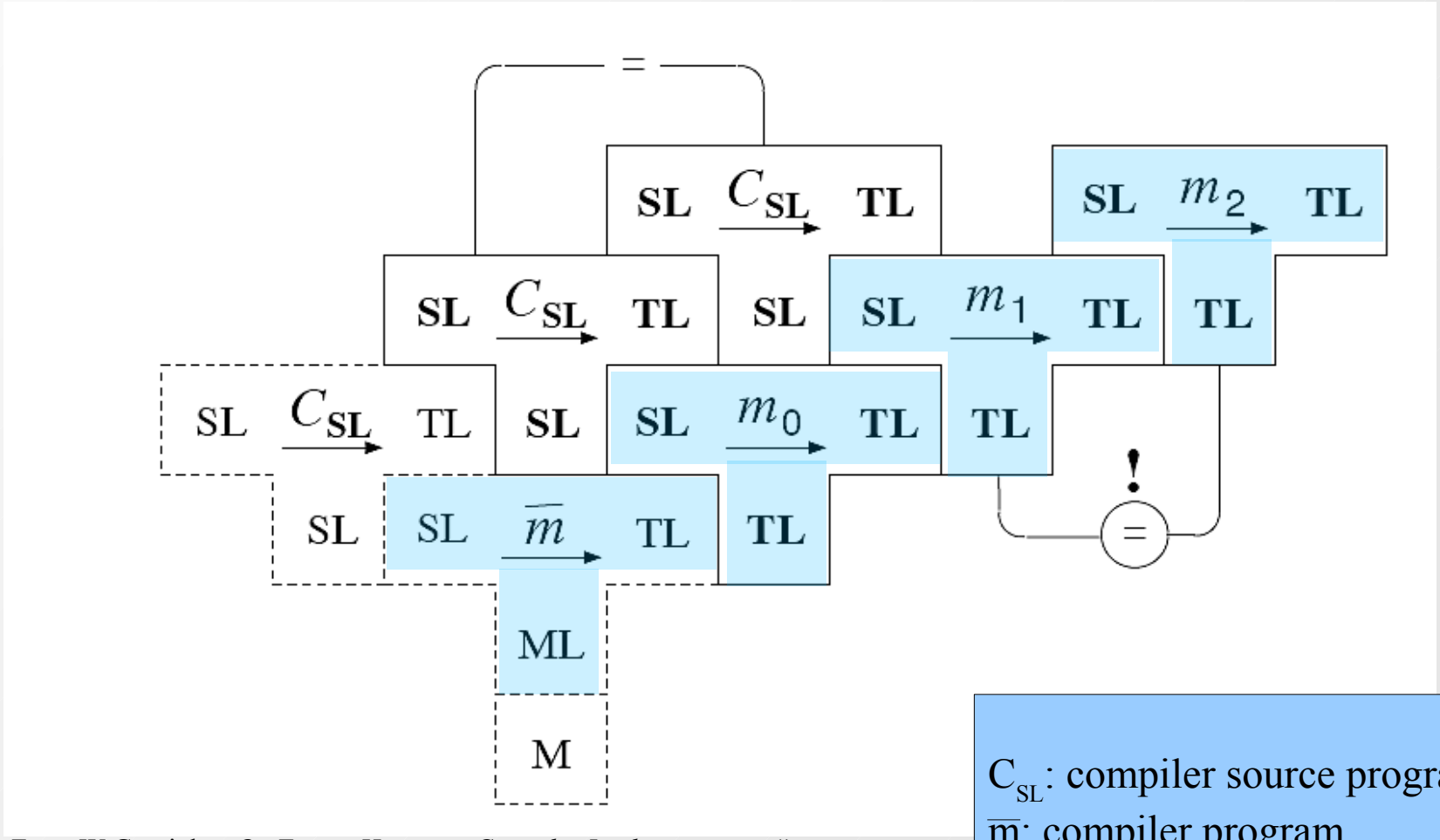
```
//file: reproduce.c
char *buf =“
//file: reproduce.c
char *buf = %c%s%c;
int main(int argc, char *argv[]){
    if (argv[1] && (strcmp(argv[1], %cident%c) == 0))
        printf(buf,34,buf,34,34,34,34,34,34,34);
    else if ((argv[1] && (strcmp(argv[1], %clogin%c) == 0))
        printf(%cOops%c);
    else
        printf(argv[1]);
}
void cheat () {}
“;

int main(int argc, char *argv[]){
    if (argv[1] && (strcmp(argv[1], “ident“) == 0))
        printf(buf,34,buf,34,34,34,34,34,34,34);
    else if ((argv[1] && (strcmp(argv[1], “login“) == 0))
        printf(“Oops“);
    else
        printf(argv[1]);
}void cheat () {}
```

Compiler Bootstrapping

- Compiling a compiler where
  - source language and
  - implementation language are the same
- Example:
  - C++ compiler used to compile a new version of it
  - where source code for the new version is written in C++

Bootstrap Test



From W.Goerigk: *„On Trojan Horses in Compiler Implementations“*

C_{SL}: compiler source program
$\overline{m}$: compiler program
ML: M's machine language

Bootstrapping Theorem

If m_0 and C_{SL} are both correct, if m_0, applied to C_{SL}, terminates with regular result m_1, and if the underlying hardware worked correctly, then m_1 is correct.

Bootstrap Test Theorem

If m_0 and C_{SL} are both correct and deterministic, if m_0, applied to C_{SL}, terminates with regular result m_1, if m_1, applied to C_{SL}, terminates with regular result m_2, and if the underlying hardware worked correctly, then $m_1 = m_2$.

Passing the Bootstrap Test (1/3)

- Now consider $\overline{m_0}$ to be an compiler implementation including a Trojan Horse
  - reproducing $\overline{m_0}$ if applied to C_{SL}
  - Compiling a bug to login.c if applied to this
  - Working correctly as m_0 (unmodified compiler) for any other case

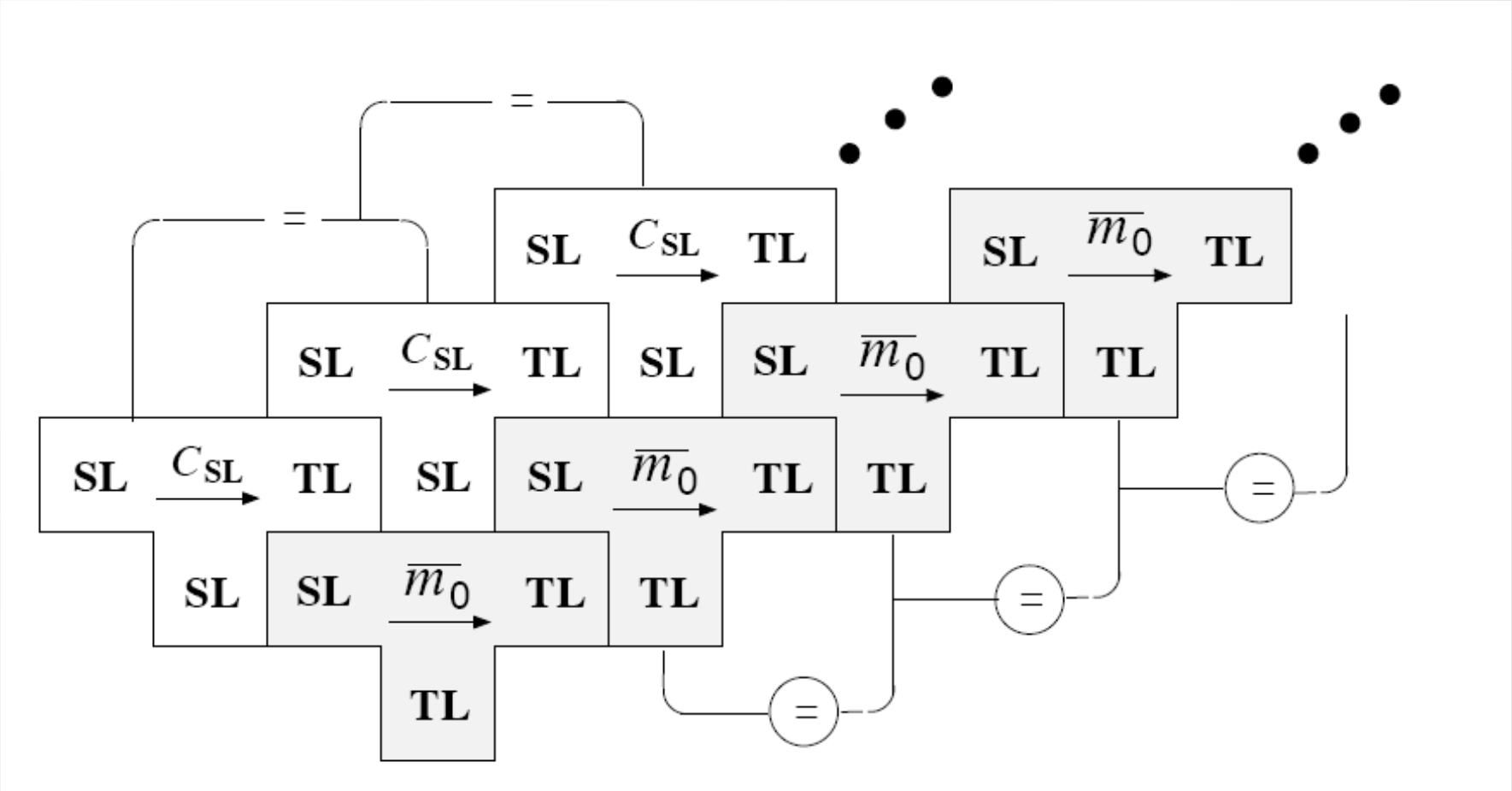
Passing the Bootstrap Test (2/3)

```
//file: compile-incorrect.c
#include<string.h>
#include<stdio.h>

char login[255] = „void main() {printf\"Cheating Login (Oops):\";}";
char *buf = "...
";
char cmdbuf[255] = "make CC=gcc 'basename ";
FILE* handle;

void main (int argc, char *argv[]) {
       if (argv[1] && (strcmp(argv[1],"compile.c") == 0)) {
                system("mv compile.c .compile.c.orig");
                if ((handle = fopen("compile.c", "w+"))!= NULL {
                     fprint(handle,buf,
                          34,92,34,92,34,34,34,buf,34,34,...,34);
                     fclose(handle);
                     system("make CC=gcc compile");
                     system("mv .compile.c.orig compile.c");
                };
       }
       else if (argv[1] && (strcmp(argv[1],"login.c") == 0)) {
                system("mv login.c .login.c.orig");
                if ((handle = fopen("login.c", "w+"))!= NULL {
                     fprint(handle,login);
                     fclose(handle);
                     system("make CC=gcc login");
                     system("mv .login.c.orig login.c");
                };
       }
       else {
                strcat(cmdbufm argv[1]); strcat(cmdbuf," .c'");
       }
}
```

Passing the Bootstrap Test (3/3)



Avoiding Trojan Horses in compilers

- Seen: Source level verification does not work
- Sufficient: Syntactical Code Inspection
  - Let $CC_{SL,TL}$ be a semantically correct compiling relation between source and target language
  - C_{SL} is correct refinement of $CC_{SL,TL}$
  - If m applied to C_{SL} it is element of $CC_{SL,TL}$
  - ⟹ m is correct implementation of $CC_{SL,TL}$
  - ⟹ m is a correct compiler executable from SL to TL

Contents

- Some expamples
- Definition of a Trojan Horse
- Trojans in compilers
- Harrier as part of HTH framework
- Summary
- References

Hunting Trojan Horses (HTH)

- is a security framework
- developed for detecting diffucult types of intrusions
- intended to be a complement to antiviral software
- zero day attacks and new malicious code can go undetected by even most up-to-date anti-virus-program
- some trojan horses executes as plugins or DLL
- many have little impact on system behaviour
  - difficult for the user to detect
  - being undetected for a long time
  - providing attacker vulnerability for this time

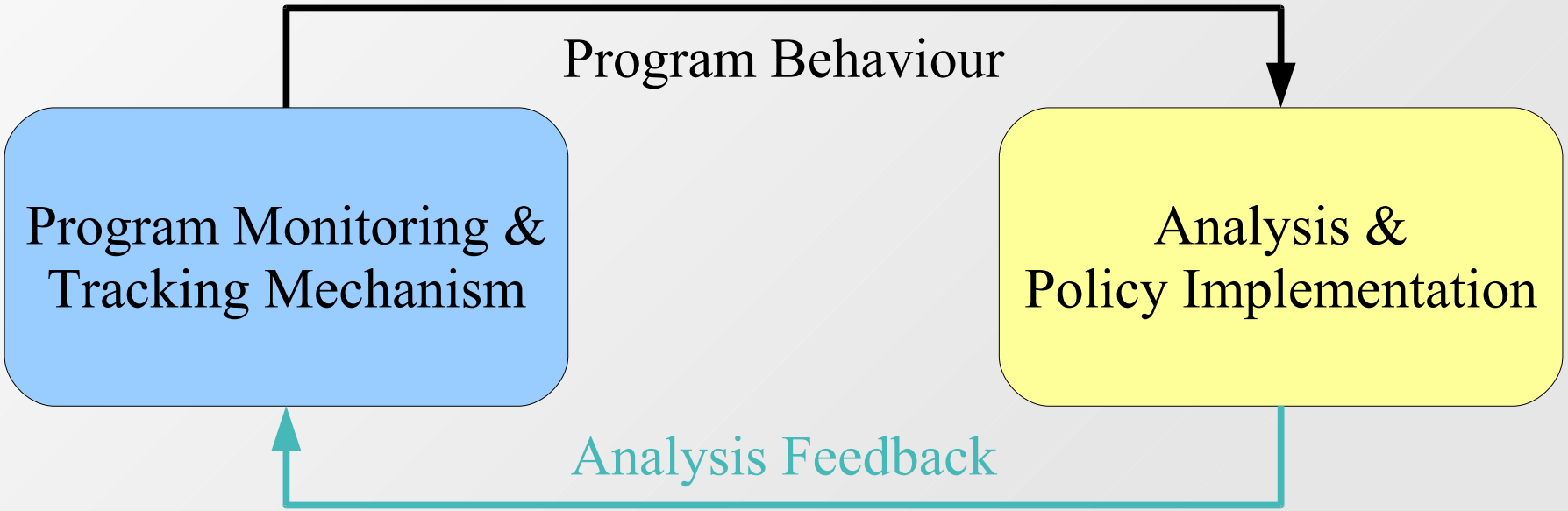
Harrier as part of HTH (1/2)

- Heart of HTH
- Application security monitoring program
- Runtime monitor collecting dynamically execution related data
- Collecting information across different abstraction layers
  - Architectural events
  - System calls
  - Library (API) routines
  - 3 – 4 times faster than other available products

Harrier as part of HTH (2/2)

- allows identification of abnormal program behaviour
- good detection rate with low rate of false positives
- enables defending against harmful activities
- no source code analyzing
- works with program binaries
  - Linux
- restricted monitoring to shared objects with a defined API

HTH software architecture



Harrier: Data sources

- Divided into 5 resource types

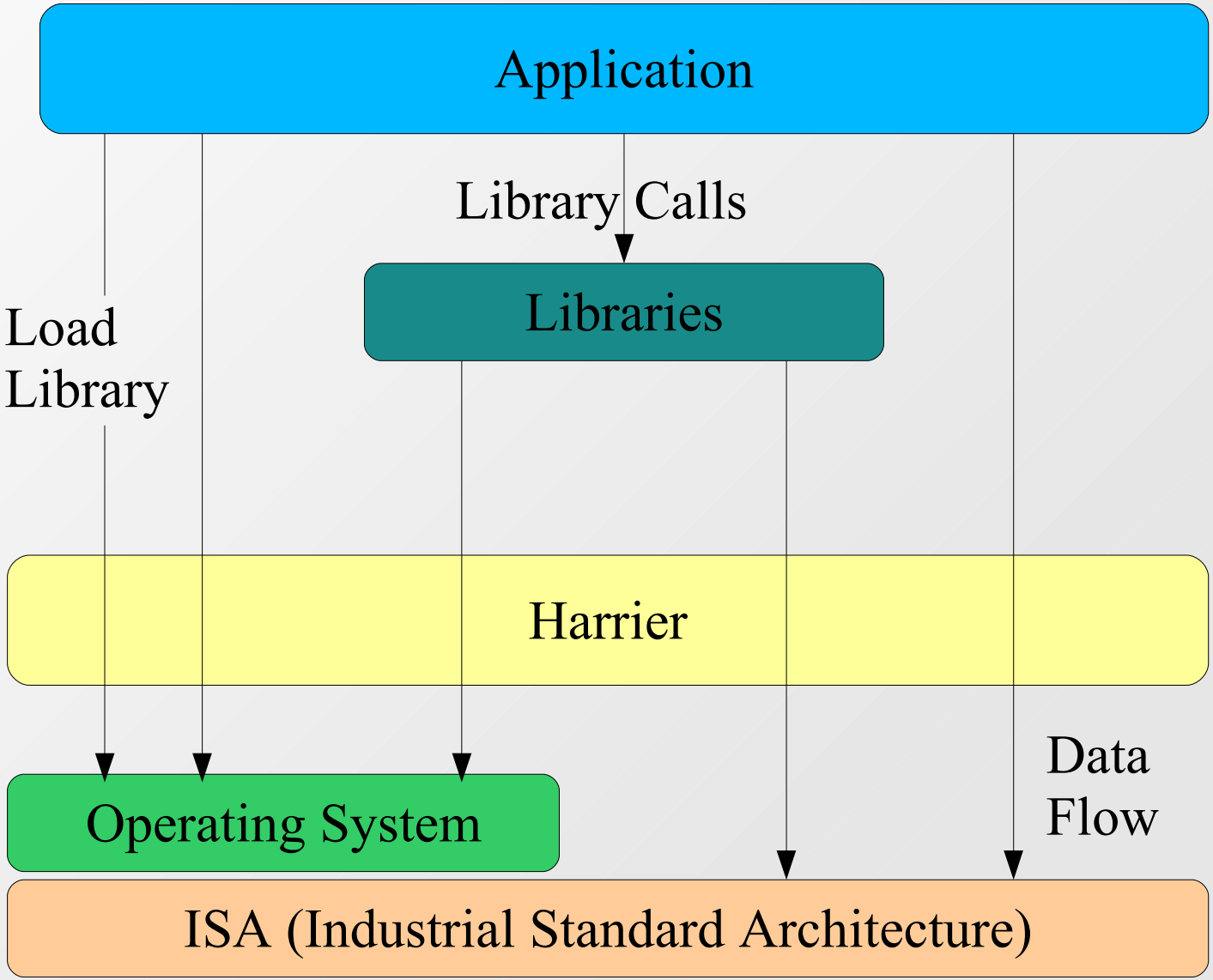
Resource Type	**Description**
User Input	data is retrieved via user interaction
File	data is read from a file
Socket	data is retrieved from a socket interface
Binary	data is part of the program binary image
Hardware	data originated from hardware (e.g. cpuid)

Harrier: Abstraction levels

- Architectural (ISA)
  - Instructions executed
- Operating System (API)
  - system calls
  - (clone, execve, open, close, read, write)
- Library (API)
  - Library routines
  - (only small set of library API functions monitored)

Collect information about
- program semantics
- program information flow

Harrier: Events collected



Harrier: Security policy

- Execution flow
  - Target: detecting malicious code being executed
- Resource abuse
  - monitor number of new processes and rate of creation of these
- Information flow
  - enforce flow between different sources and targets for the different resource types

Contents

- Some expamples
- Definition of a Trojan Horse
- Trojans in compilers
- Harrier as part of HTH framework
- Summary
- References

Summary

- Trojan Horse
  - program containing additional hidden code
  - unauthorized collection, exploitation, falsification, or destruction of data
- Trojans in compilers
  - source level verification not sufficient to guarantee compiler correctnes
  - binary compiler implementation verification needed
- Harrier within the HTH framework
  - complement to anti-virus software
  - runtime security monitor analyzing program binaries
  - tracks ISA, OS and selected library events

References

- Dr.-Ing. Claus Vielhauer, Dipl.-Inform.(FH) Andreas Lang: *„Lecture slides: Sicherheit verteilter Systeme SS2007“* FH Brandenburg, 2007
- Paul A. Karger: *„Limiting the Damage Potential of Discretionary Trojan Horses“*, in 1987 IEEE Symposium on Security and Privacy, pp. 32-37, 1987
- Wolfgang Goerigk: *„On Trojan Horses in Compiler Implementations“*, in Proceedings of the Workshop Sicherheit und Zuverlässigkeit softwarebasierter Systeme, IsTec Report, IsTec-A-367, Garching 1999
- M. Moffie, W. Cheng, D. Kaeli, Q. Zao: *„Hunting Trojan Horses“*, AsiD'06: Proceedings of the 1st Workshop on Architectural and System support for improving software dependability, pp. 12-17, San Jose, California, 2006
- A. Brown, T. Cocks, K. Swampillai: „Spyware and Trojan horses“, Seminar on Computer Security, 01-04 2004, University of Birmingham

What Questions do you have ???

Thanks for your attention !

Joke: Trojan Horse - The Chaser