Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Institut für
Technische Informatik und
Kommunikationsnetze

Marcel Marghitola

Towards Exploit Signature Generation using Honeypots II

Semester Thesis SA-2007-13
November 2006 to March 2007

Tutor: Bernhard Tellenbach
Co-Tutor: Daniela Brauckhoff
Supervisor: Bernhard Plattner

Acknowledgement

The semester thesis was from the beginning to the end an interesting task, a challenge and an opportunity to get to know the field of network security. Special thanks to Bernhard Tellenbach and Daniela Braukhoff for their support during the semester thesis. Additional thanks to the Communication System Group(CSG) of the ETH and the CSG head, Prof. Dr. B. Plattner, for providing the opportunity to write this thesis.

Abstract

It is a fact that there is little defense against the increasing number of Zero-Day attacks found in the wild. The problem with Zero-Day attacks is that the exploited vulnerability is yet unknown when they first appear. This makes it difficult if not impossible to detect them using signature based systems. A promising approach to counter Zero-Day attacks has been developed in the EU project NoAH. The project uses honeypot technology to catch these attacks at an early stage of their spreading using a technology called memory tainting. But because memory tainting suffers from bad performance, it can not be used to protect productive systems. That's why NoAH aims at automated signature generation for each new attack it detects. One way to generate the signatures is through the analysis of the network traffic. The difficulty of this approach lies in the prevention of false positives. Possible solutions are the use of techniques like protocol-aware analysis and state-tracking of the attacked host. An implementation of this functionality is given by the connection tracker framework, at tool which was developed at the ETH. The original implementation lacked some important features, like a state-aware TCP plug-in, or an interface to the honeypot system.

This thesis aimed at improving the existent connection tracker. The previous mentioned missing features were implemented, additional changes were made to improve the performance of the program. The new version of the tracker is capable of capturing and analysing a 100 MBit/s connection on a standard computer without any loss of packets. The following report gives a more detailed overview over the changes made, their results and the possible future development.

Contents

List of Figures

Chapter 1

Introduction

The information in this document is meant to be a addition to the documentation given in the sourcecode of the connection tracker. It is expected that the reader has already read [1], the semester thesis report of the first version of the connection tracker. Chapter two gives an overview over the part which the tracker plays in a honeypot system. Chapter three explains the addition and changes to the existing tracker framework. In chapter four a quick overview over security aspects is given. Subsequent chapter five describes the tests that were made with the written software. Finally, chapter six gives a summary and an outlook on possible future development. Appendix A informs about the task description for the semester thesis. A list of all files which were used and created in this semester thesis can be found in Appendix B.

Chapter 2

Honeypot Systems

2.1 NoAH

NoAH[2] stands for European Network of Affined Honeypots. It is a project financed by the European Union, aiming to develop an early warning system for internet cyberattacks in the form of a honeypot system. The connection tracker can be used to be a part of such a system, which is described more detailed in section 2.3. More information about NoAH can be found on http://www.fp6-noah.org/.

2.2 qemu and Argos

Argos[3] is a modified version of qemu[4], a solution for virtualisation. But instead of just running a virtual host, Argos has also the capability to detect an attack on the guest system over the network. For that, it tracks the way of the data, received through the network, to the places in the memory. These places are marked tainted. If such tainted memory gets executed by the CPU or if it modifies the value of the EIP (execution instruction pointer), Argos reports an attack. The additional functionality of Argos consumes a lot of resources, a program runs up to 25 times slower compared to native speed. Because of these performance issues, Argos is only usable as a honeypot, but not for the standard computer.

2.3 Structure of a Honeypot System

This semester thesis assumes a honeypot system according to figure 2.1. The attacked program runs under Argos, which detects the attack and informs the interface. The interface finds that responsible packet in the network dump from Argos. The same packet was also logged by the connection tracker, the interface finds this packet in the tracker log and all other packets, which belong to the same connection. The found information can be used by a signature generation unit to produce a signature for the attack.
At time of the attack, the tracker and Argos are required to run. In a future

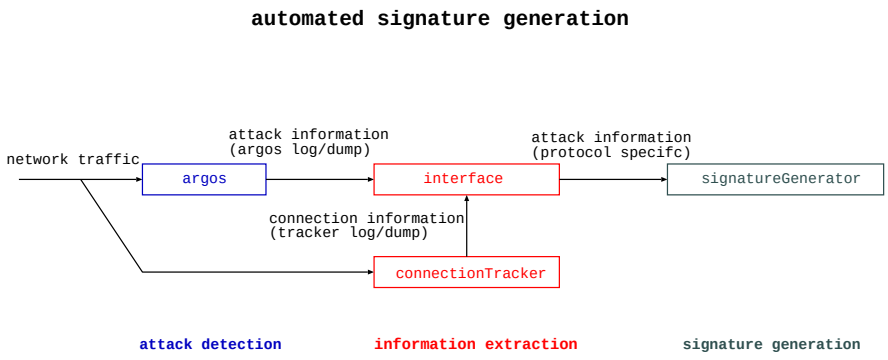

Figure 2.1: Structure of a Honeypot System

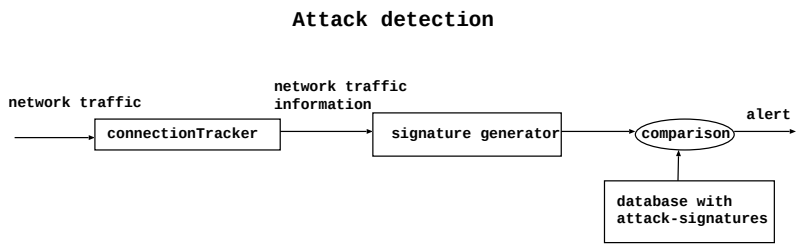

Figure 2.2: Structure of a simple IDS

version, one could add the capability for offline analysis. Then, the tracker would use the network dump from Argos as input.

A second possible use for the connection tracker is as part of a IDS (Intrusion Detection System) or a IPS (Intrusion Prevention System), see figure 2.2. The connection tracker logs the network traffic and gives the relevant information to the signature generation unit. The signature generator compares the generated signature with existing Attack signatures. On a match, an alarm is given. In an IPS, the packets are routed to the destination after the test was negative. The second approach introduces a delay which could be problematic, depending on the information sent over the network (e.g. VoIP needs small delays, while a file download is independent of the delay). Another issue is the reaction on high load. An IPS can either close the connection or open it completely, booth possibilities have disadvantages. Either there is no protection, or even regular traffic can't pass.

Chapter 3

Design

The task of the semester thesis contained two design parts. The first was the creation of a TCP plug-in, the second was the development of an interface to Argos. Additionally some other improvements were made to the connection tracker, e.g. a simplification of the reporting system to enhance the performance. The following sections describe these new features and modifications.

3.1 UDP Plug-in

The plug-in was implemented according to RFC 768[5]. Because of its simplicity, there is not much to say. The processing steps are:

- check for reasonable length
- verify the checksum (optional, only if checksum$\neq$0)
- do reporting
- call next plugin

The tracker configuration file determines which plugin gets called, in specifying for each interesting port number of the monitored host a plug-in name.

3.2 TCP Plug-in

The basis for the implementation of the TCP plug-in was the standard RFC 793[6]. Some additional features were implemented according to succeeding standards[7][8], like the scale window option from RFC 1323 (TCP Extensions for High Performance). The processing steps are:

- check for reasonable length
- verify the checksum
- analyse TCP options
- check TCP sequence number

- put the packet in the waiting queue if it is not next in line and the sequence number was in the receive window
- if the packet arrived in order update state information
- do reporting
- check the todo-list, if a packet gets ready for analysis, put in it in the state-tracker thread todo-list
- call next plugin

3.2.1 Todo List

There are two kind of todo lists: One for the state-tracker thread, one for the TCP plug-in. Booth were created for the case when packets arrive in disorder. In that case, the TCP plug-in can't analysis a packet until all its predecessors arrived. Such a packet, with a sequence number greater than the one expected but still in the receive window, is put in a todo list and EXIT_FAILURE is returned. This return value orders the calling state-tracker thread not to do the reporting. As soon as the TCP plug-in receives the preceding packet, it moves the delayed packet from the TCP todo-list to the state-tracker thread todo-list. At the next cycle, the state-tracker thread removes the packet from its todo-list and restarts the analysis cycle. This time the TCP-plug-in can finish the packets analysis.

3.2.2 State Model

RFC 793 describes the states a TCP implementation can be in, a simplified version is shown in figure 3.1. This state model is suitable for the receiver and the sender, but not for us as a outside listener. The tracker uses a modified version of this state model, as shown in figure 3.2. The following text describes the changes and the reasons for these.

merged states Through passive observation, the tracker can't always decide in which state the monitored host resides. E.g. it is not possible to distinguish between the closed and the listen state. Therefor, the plug-in treats these states as one state. A simplification was made in merging the states TIME_WAIT and CLOSED. The transition between these states is done after the expiration of a timeout.

additional states The transition between the LISTEN and the SYN_RECEIVED state consists of receiving a packet with a syn flag and as a reaction, sending a packet with a syn or a reset flag. While the receiver knows in which state he make the transitions after the arriving packet, the tracker has to wait for the monitored host to send a packet. An additional state has to be created for this situation. We call this state "SYN_REC_WITHOUT_ACK". The problem arises five times, leading to five additional states. The names of these additional states are:

- SYN_RECV_WITHOUT_ACK

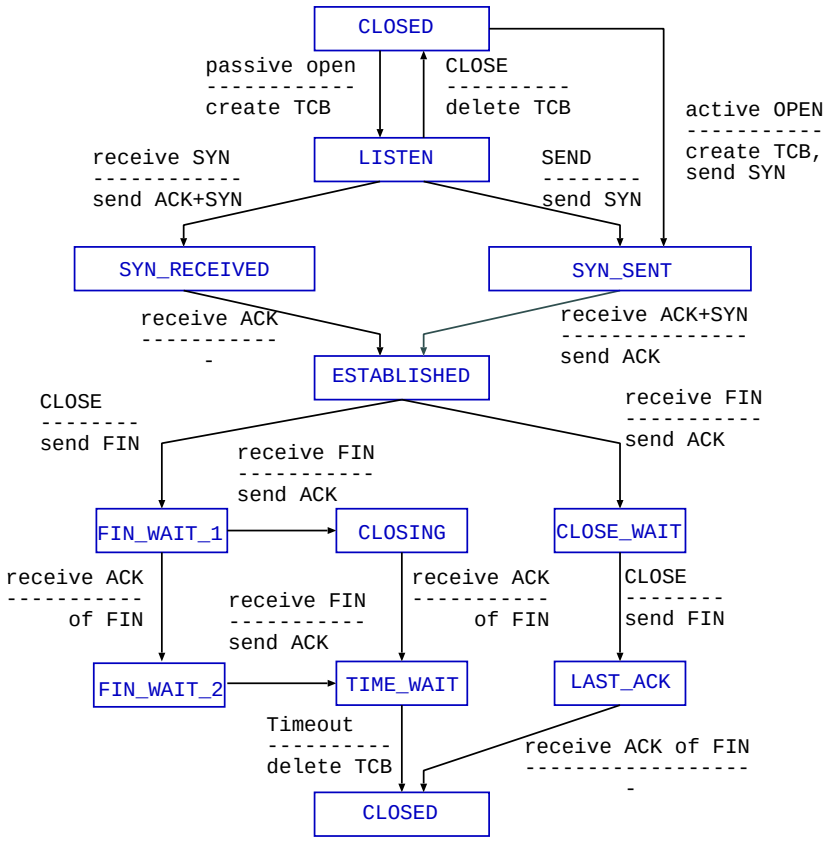

Figure 3.1: TCP state model

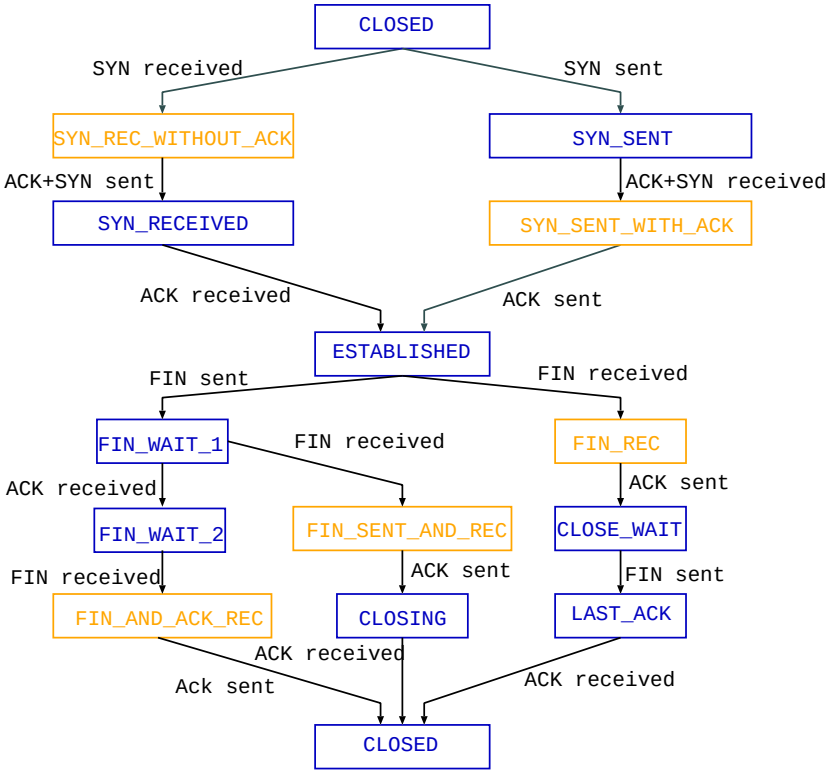

Figure 3.2: state model of tracker

- SYN_SENT_WITH_ACK
- FIN_AND_ACK_RECV
- FIN_SENT_AND_RECV
- FIN_RECV

The names of the old states weren't changed, leading to an asymmetric naming scheme. After sending a syn, the state changes to SYN_SENT, while after receiving as syn, the state becomes SYN_RECV_WITHOUT_ACK, not SYN_RECV. In figure 3.2, the additional states are shown in yellow colour.

3.3 Interface

Like the tracker, the Interface can be shutdown either with a SIGINT signal (Ctrl-C) or with a SIGTERM signal.

3.3.1 Overview

The functionality of the interface can be described through the following steps:

1. open socket connection to Argos
2. wait for attack event or shutdown signal
3. on attack, get the Argos logname
4. find the attacking packet in Argos netlog
5. find the attacking packet in tracker dump, extract packetId
6. find the attacking packet in tracker log through the packetId, extract the connectionId
7. find and print in tracker log all packets with the connectionId of step 6
8. go to step 2

3.3.2 Socket Connection to Argos

The interface opens a socket connection to Argos(SOCK_STREAM, TCP). The first transmission after a successful connection establishment contains the name of the Argos working directory. After each attack Argos sends two text lines. The first describes the type of attack, the second denotes the name of the new Argos attack log. In contrast to the Argos log, the Argos net log always has the name "argos.netlog". An implementation detail: the strings sent by Argos aren't zero terminated. The length is known through the return value of the read function.

3.3.3 Cargos Library

The interface uses the cargos library[3] for interpreting the logs of Argos. The used version is 0.1, released in December 2006.

After every attack, the following functions from the cargos-library get called:

- create a cargos_lib_t instance with cargos_lib_create()
- open the Argos log with
  cargos_lib_csi_open(cargos_lib_t *, const char * argos_logname)
- open the Argos netlog with
  cargos_lib_nt_open(cargos_lib_t *, const char * argos_netlogname)
- find the index of the attacking packet with
  cargos_lib_csi_regnidx(cargos_lib_t *, CARGOS_LIB_EIP)
- create a cargos_lib_pkt_t instance for the attacking packet with
  cargos_lib_nt_pkt(cargos_lib_t *, int index)
- get the raw packet with
  cargos_lib_pkt_data(cargos_lib_pkt_t *, u_char * buffer, int maxLength)
- free memory with cargos_lib_destroy(cargos_lib_t *)

A summary of the log files could be printed with the function cargos_lib_nt_print(cargos_lib_t *). In the case of a problem, the function cargos_lib_error(cargos_lib_t *) is used to obtain an error message.

3.3.4 Reading the Tracker Log

Opening the tracker log seems to be simple, it's just an XML-file which can get processed with the XML-library in the xmlParser directory. But there are some pitfalls to consider. First of all, the data gets buffered before the writing, so maybe the needed data isn't yet written to disk. For that the interface sends the signal USR1 to the tracker, signaling him to do a flush the buffer to disk. The second problem arises if the tracker still runs. That means that the XML end tag is still missing, since the tracker wants to append some data. So the interface has to append the end tag if necessary. The last possible issue happens, if the tracker has written analysis data of succeeding packets, and if the last reporting entry isn't completed. In that case, the last unfinished packet entry is removed.

3.4 Changes to the Tracker Framework

3.4.1 Receiving Signals

The tracker performs a normal shut down when it receives the signal SIGINT (Ctrl-C) or SIGTERM (in contrary to the first version [0.1], where SIGUSR1 was used for that purpose). When it receives the signal SIGUSR1, it sends an IPC-message to the reporter thread, telling him to flush the write buffers for the dump- and log-file. This feature is used by the interface, as already explained.

3.4.2 Reporting System

The Old Way The old reporting system allowed every plug-in to send one or multiple messages to the reporting thread. There, these messages were copied to a buffer. After receiving a special message, indicating that the analysis of the packet was finished, this buffer was copied into another buffer before writing the data to disk. The last buffer was meant to prevent the writing of small data chunks to the disk. But because this functionality is already guaranteed through the underlying write function, it was just a waste of time.

The New Way To reduce the number of sent IPC-messages, the state tracker thread sends only one message. He does so after the plugins have done their analysis. As payload the message contains a pointer to a struct called packetInfo, which holds all information needed by the reporting thread. The analysis report is split into several buffers which are connected through a linked list. To append something to the report, the plugins just add another element to the end of the linked list. When the reporting thread receives a message, he calls the write-function without buffering the an additional time.

3.4.3 IPC-Messages

Because threads share a common address space, it is not necessary to copy data between the threads. Nevertheless the tracker-threads in version 0.1 copied the data used by others threads to the data part of the IPC-messages. With version 0.2, the threads only exchange pointers to the data, thus reducing the number of copy operations and improving the speed. The synchronisation between threads is done through IPC-messages, there is no need to protect the memory through semaphores against simultaneous accesses.

Chapter 4

Security

To prevent cyber attacks, security software was invented. These days they are found on most computers. Because of their wide deployment, security software itself gets more and more attacked, sometimes leading to the strange situation where an attacker was only successful because security software was installed. The following sections discuss how possible security issues in the connection tracker got addressed. The first section is about writing the code, the second about running the program. Additionally, the programs reaction on special input was tested in section 5.1.

4.1 Writing Secure Code

One way to improve code security is to review the written code by a different person. The connection tracker gets written by several persons (till now three, after the thesis "Towards Exploit Signature Generation using Honeypots III" four), so each person will read code by someone else, and so the chances to find existing bugs increases.
Security risks often arise because the programmer assumed the input, coming from the user, a file or the network, would be correct. While writing the TCP plug-in and reviewing the code which every packet passes through, I tried to critically test the input for possible malformed content. Another problem is the use of unsafe functions, e.g. strcpy or sprintf. Such unsafe functions are only used in the tracker where no threat is possible (no threat means e.g. copying constant strings with strcpy).

4.2 Running the Connection Tracker

Because the connection tracker needs raw access to the network, it has to run with root privileges. That makes the tracker a worthy target. The problem of programs which run as root and have network access is well known and lead to the creation of solutions like AppArmor and SELinux. Booth are Mandatory Access Control (MAC) systems. They have the goal to restrict the power of a program to the minimal set of rights, which the program needs to fullfill its normal functionality. With such MAC systems, even programs run by root have

restricted rights. It is strongly suggested that the tracker is only run under such systems.

4.3 DOS/DDOS-Attacks

Two tracker components store state information. The first is the tracker framework, it stores which connection belongs to which thread, so that all packets of a connection get processed by the same thread. The second component which stores state information is the TCP plug-in, which stores several data about each connection, e.g. the TCP-state of the monitored host, the current sequence numbers in booth directions or the MTU. An attacker could try a DOS(Denial Of Service)- or DDOS(Distributed Denial Of Service)- attack in sending a lot of connection requests, leading to a lot of memory consumption on the computer running the tracker. At the moment the only counter measurement is reducing the time until old connections gets removed. Future versions of the tracker should provide additional protection mechanisms, possible improvements are discussed in the section 6.2.4.

Chapter 5

Software Testing

5.1 Functional Test

5.1.1 Selection of a Packet Builder

Five packet builder(excalibur[9], packETH[10], packit[11], nemesis[12], colasoft packet builder[13]) were tested. Four of them weren't useful, because of problems like incompatibility to recent operating systems, a tendency to crash or wrong functional behaviour. The fifth, the colasoft packet builder, showed a decent behaviour and was used for testing. The program is available as freeware for windows 2000 and windows xp.

5.1.2 Tested Functionality

Only some special cases were tested, which occur in reality but which are unlikely to occur in the normal test environment. The following functionality was tested:

- sequence number wrap around
- packets arrive disordered
- reassembling of IP-fragments
- reassembling of IP-fragments, which arrive in disorder

These few cases which tested complex code, lead to the discovery and fixing of some problems.

5.2 Performance

5.2.1 Testing Goal

The goal of the test was to study the behaviour of the tracker for high loads. In such cases packet drop can occur, as the pcap library has problems to capture all packets. As a comparison, the tests were made with the connection tracker in a recent version (tracker-0.2) and the version prior to this semester thesis (tracker-0.1).

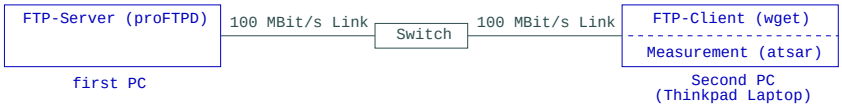

Figure 5.1: test structure

5.2.2 Test Environment

For the speed test, two computers were used. The first one acted as ftp-server, the second as ftp-client and as a host for the connection tracker. As second computer an IBM/Lenovo Thinkpad T60p laptop with a 2 GHz dual core processor was used. Proftpd[14] served as FTP-server, wget[15] as FTP-client. To stress both cpu cores, two FTP-downloads were started at the same time. On the first transmission the download rate was limited to 2 times 1.5 MByte/s, on the second transmission to 2 times 3 MByte/s and on the third time to 2 times 6 MByte/s. Each time an amount of 200 MByte was transferred, leading to number of about 100'000 packets. The whole test was repeated several times, and showed each time similar results. The results were gathered with the program atsar[16], for a graphical representation gnuplot[17] was used. The test scripts can be found in the directory /docu/measurements, a detailed explanation is given in the README file.

5.2.3 Results

As shown in figure 5.2, version 0.2 shows significant better performance results over version 0.1. The reason for this improvement is given in section 3. The new version doesn't show packet drop, while the old version drops about 13 percent of the packets already at a total speed of 3 MBytes/s. Over the whole test, the old version drops about every second packet.

5.3 Memory Leak Test

The connection tracker was searched for memory leaks with the help of a program called memprof[18]. It has the capability to show in which line of source code memory was allocated and never deallocated afterwards. According to memprof, there are no substantial memory leaks. The only warnings that were shown were at the creation of a thread and the loading of a plug-in. Booth warnings seem to be either a misinterpretation of memprof or a problem of the underlying library. Even if they were memory leaks, the damage is limited. Booth functions are only called a limited, small number of times at the start of the program. The interface didn't show any sign for memory leak at all.

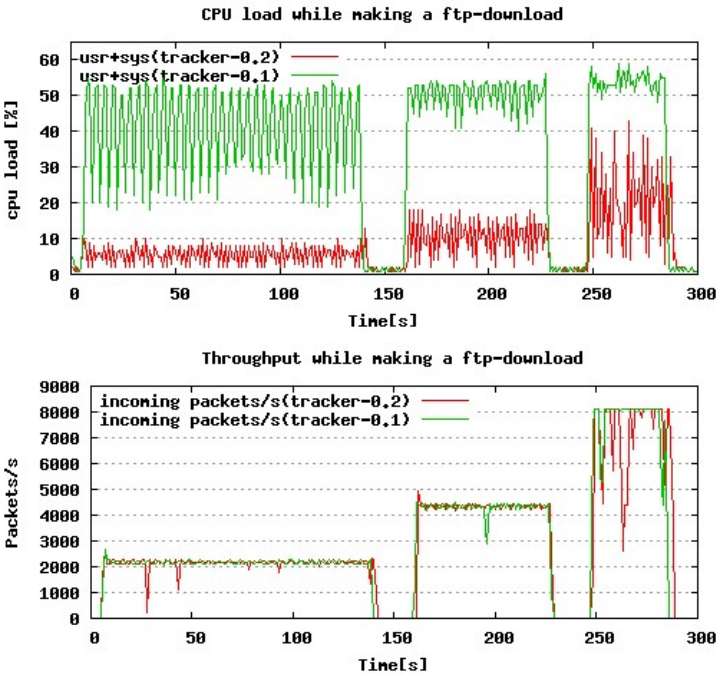

Figure 5.2: Measurement Results

Chapter 6

Summary

6.1 Conclusions

Another step is made in the life of the connection tracker. The program is now extended with a TCP plug-in, capable of recognising malformed packets or reordering rearranged packets. Also the other plug-ins went through an improvement-process, in which for example the detection of wrong checksums was added. With the implementation of the interface, an important step for the integration of the tracker in a honey pot system was made. Performance-wise a great improvement could be realized, giving the capability to capture the a connection on a 100 Mbit/s link without packet drop. Through the use of a packet generator special cases were tested, leading to the detection and removal of software bugs that would have been overseen otherwise.

6.2 Outlook

6.2.1 Tracker Framework

An nice addition to the tracker would be an offline mode, so that a network dump could be used as input instead of a network interface. This would be especially useful for developers. If the tracker fails to correctly analyse the network traffic, a developer could try to fix the problem and test his changes with the network dump. Additionally, there would be no need for the tracker to run simultaneously with Argos. The tracker could be run after an attack was detected by Argos, and the Argos network dump could be used as input.
Of course also some small improvements could improve the user experience, like the creation of a man page or the acceptance of command line arguments like –version, –help and –config-file=y.xml.

6.2.2 Plug-ins

Plug-ins for Higher Network Protocol Layers

For the generation of a useful signature, plug-ins for higher network protocol layers have to be created. The existing plug-ins can be used as a starting point.

At the moment, the dispatcher decides the analysing thread through the IP-addresses and the port numbers. If a future plug-in needs to analyse data sent to a different port, the dispatcher behaviour has to get adapted. A simple solution would be to make the decision about the analysing thread only by the IP-addresses of sender and receiver.

IPv6

It can be assumed that IPv6 will prevail someday. The TCP plug-in was made in the assumption, that an IPv6 plug-in will be made sometimes. For that reason, the TCP plug-in doesn't directly depend on the information of the IPv4 plug-in[1]. Instead of accessing the ipv4Address struct, it uses the functions from the pure virtual class ipAddress. If the creator of a future IPv6 plug-in derives a class from ipAddress and gives such an instance as a function argument to the TCP plug-in, he doesn't need to alter the TCP plug-in at all.
Additionally, some changes are needed to the tracker framework. The actual version of the tracker expects to read in an IPv4-address from the configuration file, and the function used by the dispatcher to determine the thread doing the analysis counts on the presence of an IPv4 header to determine the affiliation to a existing connection.

6.2.3 Additional Programs

To implement a honeypot system as shown in figure 2.1, a signature generation unit is needed. The challenge will be to detect the relevant data for each attack. A possible approach would be to feed the signature generator not only with the attacking packets, but also with normal traffic. Now the signature generation unit could search for differences, and also check if the signature only targets the infected packets.

6.2.4 Software Testing

Performance

To prevent packet dropping through libpcap, future extensions and changes to the tracker should be tested for their performance influence. Maybe packet dropping could also be avoided in changing the operating system, e.g. the use of FreeBSD or NetBSD instead of Linux[2]. For the use under Linux, the performance impact of security solutions like SELinux and AppArmor should get measured.

DOS/DDOS Resistance

To prevent DOS or DDOS-attacks, future versions of the tracker should provide a configuration option to limit the number of open connections per IP-address or the overall number of connections. An alternative would be to create a

[1] In a perfect world, the TCP layer would be independent from the IP Layer. But according to the TCP specification some information from the underlying protocol is needed, e.g. for the checksum calculation.

[2] It was never tested if the software even compiles on these operating systems, but I see no reason why it shouldn't work

dynamic limit, which gets set according to the system load and the unused memory. To estimate the maximum number of possible concurrent connections measurements have to made.

Functional Testing

In this semester thesis, besides normal network traffic only some special cases were tested. In the future a more comprehensive test should be made to ensure the correct functional behaviour.

Appendix A

Problem Statement

Today, intrusion prevention systems (IPS) are widely deployed to protect IT infrastructures from malicious activities. To detect and prevent malicious activities, most IPSes use some kind of signature. Basically, a signature is a description for a specific activity/class of activities on a system. The description has (ideally) the following property: it matches only the activity/class of activities it describes (unambiguous). Basically, signatures can be used at different levels of an IT system. The two main levels are the network level and the host-level.
Signatures operating at the network level usually describe some characteristic of the network traffic associated with a particular malicious activity. Examples for network-level signatures are simple byte strings, regular expressions, or byte frequency distributions for network traffic. Host-level signatures operate on host-level information such as system call patterns or control-flow data. This makes host-level signatures very OS-specific. Moreover, their generation requires quite some resources on the end-host. Network-level signatures are much easier to deploy and to distribute because they do not depend on host-specific information. Hence, they are widely used today. However, the drawback of network-level signatures is the high rate of false positives they cause. This is due to the fact, that e.g. malicious byte patterns frequently occur in benign traffic too.

Protocol Awareness

A way to reduce the false positive rate when using network-level signatures is to implement protocol- and content-awareness. Nowadays, this approach is widely used in commercial applications and is an issue in several research projects (see e.g. [19]). Even some open-source IPS allow for protocol awareness since they support regular expressions [20]. Protocol- and content-awareness can provide a lot of useful information (e.g. protocol states, field lengths, and field types, expected content characteristics,...) for generating more accurate and meaningful signatures. The goal of this work is to provide information about protocol states (and if possible characteristics of the content of the protocol fields) for individual connections to ameliorate signatures generated by a honeypot system.

NoAH and Argos

NoAH is a European project which has the goal to develop an infrastructure for security monitoring based on honeypot technology. Honeypots will be used as early-warning systems capable of detecting attacks at the early stages of their infestation. In NoAH, honeypots are normal PC's running OSes such as Windows or Linux, and services like webservers or ftp-servers. Nevertheless, the honeypots in NoAH are different from ordinary office PCs because all remotely accessible software components (operating system services, web server,...) run within a containment environment called Argos [1] [21]. This containment environment serves mainly two purposes: First it contains the effects of an attack, and second it provides some information about the attack (e.g. for signature generation).

A.1 The Task

This thesis consists of two major subtasks:

- Continue the development of the ConnectionTracker framework
- Implement an interface between the ConnectionTracker framework and Argos

Continue ConnectionTracker framework development

This subtask consists of the following workpackages:

- Design the transport layer plugin for the ConnectionTracker framework
- Implementation of the plugin
- Evaluation of the plugin

We expect the following output:

Design the transport layer plugin for the ConnectionTracker framework: Prior the first line of code, a concept/blue-print on how to implement the requested extensions has to be developed and discussed with the tutors. This includes at least a diagram with the functional blocks, the relevant data to be logged and the data format for the log.

Implementation the plugin: Implementation: A working implementation of the transport layer plugin for tracking TCP and UDP transport layer states. The code should be well-written and -commented so that others can easily understand and extend it. We strongly recommend to use code documentation tools like Doxygen.

Evaluation of the plugin: Check if your plugin can accurately track TCP/UDP connection states. This includes considering the TIMEOUT/MULTI STATE PROBLEM ("Solvable" only after response from server has been seen)

[1] Argos runs the operating system together with its services inside the x86 emulator QEMU [4] in order to supervise their execution.

A.2 Deliverables

During this thesis the following deliverables will be produced:

1. Detailed documentation of the interface between Argos and the ConnectionTracker
2. Detailed documentation of the Connection Tracker plugin
3. A running prototype of the tracker framework with transport layer connection state tracking and logging.
4. An accurate evaluation of the designed components (measurements).

Further optional components are:

- Implementation of additional ConnectionTracker modules for an application level protocol

Documentation structure and presentations

A documentation that states the steps conducted, lessons learned, major results, an outlook on future work and unsolved problems is to be written. Additional information and tips are available on the thesis wiki at http://tikiwiki.ethz.ch/thesis/index.php/Main/HomePage. Furthermore, the developed code is expected to be well structured and documented. The same holds for its installation and configuration. At the end of this thesis, a presentation is to be given at TIK that states the core tasks and results of this thesis. If important new research results are found, a paper might be written as an extract of the thesis and submitted to a computer network or security conference.

Dates

General:

- This semester thesis starts on Monday, 13.11. 2006 and is finished on Monday, 05.03. 2006. It lasts 14 weeks in total.
- Informal meetings with the supervisors will be announced and organized on demand.

Presentations:

- One intermediate informal presentations for Prof. Plattner and all supervisors will be scheduled at half time into the thesis.
- A final presentation at TIK will be scheduled close to the completion date of the thesis.

Supervisors

Bernhard Tellenbach, tellenbach@tik.ee.ethz.ch +41 44 632 70 06, ETZ G97
Daniela Brauckhoff, brauckhoff@tik.ee.ethz.ch, +41 44 632 70 50, ETZ G97

Appendix B

Files and Directories

B.1 Important Directories

docu/ This Directory holds the report (report.pdf) and the presentation (presentation.pdf) files.

docu/measurements/ Look here for the measurement-scripts and their results (read the README file before you try to reuse the scripts!)

sourceDocu/ The doxygen documentation gets generated in this directory.

xmlParser/ Files from the used XML-parser.

B.2 File Index

Total: 12 directories, 84 files

```
.
|-- AUTHORS
|-- COPYING
|-- ChangeLog
|-- INSTALL
|-- LICENSE
|-- Makefile.am
|-- README
|-- TODO
|-- configurationImport.cpp
|-- configurationImport.h
|-- configure.in
|-- docu
|   |-- Abstract.tex
|   |-- Bibliography.tex
|   |-- Design.tex
|   |-- Pictures
|   |   |-- README
|   |   |-- TCP_states.obj
|   |   |-- TCP_states.pdf
|   |   |-- TIKETHhdr.jpg
|   |   |-- attack_detection.obj
|   |   |-- attack_detection.pdf
|   |   |-- automated_signature_generation.obj
|   |   |-- automated_signature_generation.pdf
|   |   |-- tracker_states.obj
|   |   |-- tracker_states.pdf
|   |   |-- tracker_structure.obj
|   |   `-- tracker_structure.pdf
```

```
|   |-- Problem_Statement.tex
|   |-- Security.tex
|   |-- Summary.tex
|   |-- Title.tex
|   |-- acknowledgement.tex
|   |-- fileIndex.tex
|   |-- introduction.tex
|   |-- measurements
|   |   |-- README
|   |   |-- data_0_1.dat
|   |   |-- data_0_2.dat
|   |   |-- graph.gp
|   |   |-- graph.jpg
|   |   |-- konv.sed
|   |   |-- konv.sh
|   |   `-- measure.sh
|   |-- overview_honeypot_systems.tex
|   |-- presentation.pdf
|   |-- presentation.tex
|   |-- report.pdf
|   |-- report.tex
|   `-- tests.tex
|-- dynamicLibrary.cpp
|-- dynamicLibrary.h
|-- gpl.txt
|-- interface
|   |-- Makefile.am
|   |-- configI.xml
|   |-- interface.cpp
|   |-- interface.h
|   `-- xmlParser -> ../xmlParser
|-- messageQueuing.cpp
|-- messageQueuing.h
|-- packetCapture.cpp
|-- packetCapture.h
|-- plug-in
|   |-- IPTracker
|   |   |-- Makefile.am
|   |   |-- libIPTracker.cpp
|   |   `-- libIPTracker.h
|   |-- Makefile.am
|   |-- TCPTracker
|   |   |-- Makefile.am
|   |   |-- libTCPTracker.cpp
|   |   `-- libTCPTracker.h
|   `-- UDPTracker
|       |-- Makefile.am
|       `-- libUDPTracker.cpp
|-- reporting.cpp
|-- reporting.h
|-- sourceDocu
|-- stateTracking.cpp
|-- stateTracking.h
|-- stopTracker.sh
|-- test
|   |-- fragmentation.cscpkt
|   |-- fragments_arrive_in_wrong_order.cscpkt
|   |-- tcp_packets_arrive_in_wrong_order.cscpkt
|   `-- tcp_packets_with_wrap_around.cscpkt
|-- tracker.Doxyfile
|-- tracker.cpp
|-- tracker.h
|-- trackerConfig.xml
`-- xmlParser
    |-- Makefile
    |-- lgpl.txt
    |-- xmlParser.cpp
    `-- xmlParser.h
```

Bibliography

[1] Dominik Langenegger, Patrick Bichsel: *Design and Implementation of a Protocol Aware Connection Tracker Framework* (2006)

[2] NoAH: *http://www.fp6-noah.org/*

[3] Argos, cargos-lib: *https://gforge.cs.vu.nl/projects/argos/*

[4] Qemu: *http://fabrice.bellard.free.fr/qemu/*

[5] RFC 768 (UDP): *http://www.ietf.org/rfc/rfc0768.txt*

[6] RFC 793 (TCP): *http://www.ietf.org/rfc/rfc0793.txt*

[7] RFC 1071 (Computing the Internet Checksum): *http://www.ietf.org/rfc/rfc1071.txt*

[8] RFC 1323 (TCP Extensions for High Performance): *http://www.ietf.org/rfc/rfc1323.txt*

[9] Excalibur: *http://www.securitybugware.org/excalibur/*

[10] packETH: *http://packeth.sourceforge.net/*

[11] packit: *http://www.packetfactory.net/projects/packit/*

[12] Nemesis: *http://www.packetfactory.net/projects/nemesis/*

[13] Colasoft Packet Builder: *http://www.colasoft.com/packet_builder/*

[14] ProFTPD: *http://www.proftpd.org/*

[15] Wget: *http://www.gnu.org/software/wget/*

[16] atsar: *ftp://ftp.atcomputing.nl/pub/tools/linux/*

[17] gnuplot: *http://www.gnuplot.info/*

[18] MemProf: *http://www.gnome.org/projects/memprof/*

[19] Z. Liang and R. Sekar. Fast and automated generation of attack signatures: A basis for building self-protecting servers. In *Proceedings of the 12th ACM Conference on Computer and Communications Security (CCS)*, Alexandria, VA, November 2005.

[20] V. Paxson. Bro: a system for detecting network intruders in real-time. *Computer Networks (Amsterdam, Netherlands: 1999)*, 31(23-24):2435-2463, 1999.

[21] G. Portokalidis, A. Slowinska, and H. Bos. Argos: An emulator for fingerprinting zero-day attacks. In *Proc. ACM SIGOPS EUROSYS'2006*, Leuven, Belgium, April 2006.