MIGRATING A HONEYPOT TO HARDWARE

Vukašin Pejović[1], *Ivana Kovačević*[2], *Slobodan Bojanić*[1], *Corado Leita*[3], *Jelena Popović*[2] *and Octavio Nieto-Taladriz*[1]

[1] Departamento de Ingeniería Electrónica, ETSI Telecomunicación, Universidad Politécnica de Madrid, Madrid, España
[2] Department of Electronics, School of Electrical Engineering, University of Belgrade, Serbia
[3] Institut Eurécom, BP193, F-06904 Sophia Antipolis cedex, France

ABSTRACT

A honeypot apparatus, as a perspective security technology has proven itself worth deploying by various malicious records made. The next step in deploying the technology can be an independent hardware device with the incorporated honeypot behaviour. Such a solution would bring an ease in deployment together with a high throughput it would be able to support to the area of network auditing and monitoring. Initial investigation and implementation steps have been conducted. A flexible base for a honeypot platform intended to be implemented on a modern Field Programmable Gate Array device, as a potential destination technology, has been developed. Correspondent results with a relevant set of details are being presented together with future perspectives and further investigation and deployment potential. No similar attempts have been documented.

1. INTRODUCTION

In a modern digital society the information security has an indisputable importance. Different technologies, all with their respective virtues and flaws, have been developed in order to enforce the safety of information. Some of these are globally accepted and deployed even in everyday life such as antivirus programs, firewalls, Virtual Private Networks and intrusion detection and prevention systems. The goal of this work is to present a step forward towards deploying a relatively new security technology, a honeypot that is becoming more and more actively used [8].

The essence of the functionality behind the honeypot mechanism is to provide a tool for detection and identification of hitherto unknown or unidentified security threats [2]. Honeypot is a decoy network, or a fake part of one. It has the intention to deceive an attacker by imitating a real network surrounding, a whole system or sometimes just a single system service. If it is well deployed, an attacker is unable to distinguish weather he communicates with the original or the artificial (honeypot based) servers or services. Two types of honeypots are generally distinguished: a low and a high interaction honeypot [9].

If an attacker was to determine the nature of the platform it communicates with, he would need to probe and actually perform some kind of an attack on all the potential targets, platforms of unknown type. While doing so, there is a great possibility of probing a honeypot platform, thus revealing an attempt and alerting a network administrator or another supervising entity.

A honeypot, being in a way a fake network device, must not have anybody interacting with it, since it has no production value at all. This creates a simple, straight forward, detection environment. Specifically, the only thing that a honeypot should do is to detect every presence and perform corresponding tasks in one such situation. Normally, the tasks consist in making the record of an event in order to analyse it later. In comparison with other security technologies, which have mostly defensive approach, honeypot is based on a proactive interaction principle with a goal to obtain useful data and prevent future attack scenarios.

Honeypots are generally implemented on the top of an operative system using its networking capabilities, to be more specific, TCP/IP stack. One such example certainly a low interaction HoneyD [1] honeypot platform. In order to clarify further the functionality that is behind the honeypot word the HoneyD mechanism will be described shortly.

HoneyD monitors the unused IP address space [1] within a local area network it is installed at. After noticing a connection attempt to an unused IP address the HoneyD intercepts the connection and then interacts with an attacker pretending to be a victim machine. This action is lead by a fact

This work has been funded in part by the Spanish Ministry of Education and Science under project TEC2006-13067-C03-03 and by the European Comission under Tempus Project CD-JEP-17028-02

0-7695-2989-5/07 $25.00 © 2007 IEEE
DOI 10.1109/SECURWARE.2007.35

that no legitimate communication can be established with an entity behind an unused address. A fundamental part of the platform is an operative system emulator implemented by scripting techniques and used to execute the interaction process.

2. A HARDWARE BASED HONEYPOT

The motivation for the work being exhibited is a potential that might be noted when observing a honeypot apparatus and the intention it has been engineered for. As previously mentioned, the most important role that a single honeypot platform has to play is to maintain a network communication with a potential intruder as long as possible, as it was a real system and not a fake honeypot one. All of this as to gather as much data as possible related to an intrusive communication it has been a part of. In such a context, more honeypots would provide more useful information. A hardware based honeypot can also be seen as a faciitation concept for the deployment of the Honeypot Sensors [**?**].

Actually, the approach used for the deployment of honeypots requires a PC platform on which a honey pot cn be installed at. Dependability of installed software and system characteristics applies here too, without any doubt. It is highly probable that a modern PC will not be used only as a honeypot device. In such a case, scheduling of active processes within an operative system, generally directly dependant of a type and a number of processes being executed at a certain moment of time, might cause a honeypot process to loose the processor at a crucial moment and not to be able to maintain an ongoing communication. On the other hand, older PCs that might be used as a honeypot only machines might not be fast enough to sustain traffic at high speeds such as 10 Gbps Ethernet, which is expected to be more actively invoked in a near future.

In order to overcome the cited problems an independent hardware honeypot device is proposed. As a consequence of its nature the device would be compact, easy to deploy, and hopefully of low cost. The basis of the device is implemented on an FPGA platform. It is expected that the final instance of the design becomes a fully functional, highly mobile and easily deployable honeypot system.

Transition of up to now purely software concept towards a hardware concept, proposed here, is based on a model presented in the work of Leita et al. [4]. The honeypot engine in the mentioned model behaves accordingly to the previously developed set of state machines. These state machines describe the server-client communication, which honeypot has to be aware of in order to behave as it was a real system and not an intentionally replicated one. The finite state machines are attained from gathered and analysed traffic records. The records are being processed with a especially designed set of algorithms [4] with a goal to develop a minimal yet sufficient state machine set to permit a honeypot platform to function to its best.

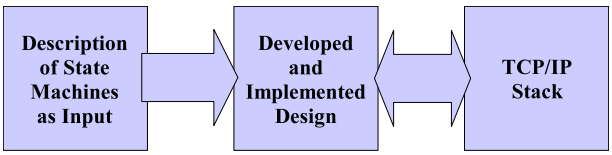

Fig. 1. Block scheme

Originally, state machines are described by three attributes [4]. These can be represented as a set of strings. More detailed explanation of the state machines related facts are encountered later on. Important fact now is that the string based model [4] can be easily stored in the memory resources of a hardware device. This will allows for the hardware design to up to certain extent program itself and behave according to the stored descriptions.

The design, presented in more details later, thus has as a foundation the shortly described model. Inputs of the design are state machine descriptions, whilst the outputs are excitations of a TCP/IP stack. More precisely, the outputs are payload values to be sent by the stack. Visual representation is in Fig. 1.

3. THE DEVELOPED SYSTEM

The destination platform for the described system was a chip from the Xilinx Virtex4 FX family [3]. This family permits a simultaneous usage of both logic-centric [5] design concept, as an essential virtue of FPGA technology, and microprocessor design concept, as a characteristics from the embedded system design background. The chip features a 700 DMIPS PowerPC [3] [6] embedded microprocessor hardware core which can be instantiated and used together with other mostly logic-centric soft core based parts of a design, comprising together a fully functional design.

As mentioned, input data for the system is a set of state machines used to mimic the behavior of real-world network communication. State machines, to be more specific, do describe the server side of communication, which coincides with the mechanism envisaged for the honeypot functioning in general. Honeypots, just to remind, are not used for any useful work except the detection of the intruders, so no client-side communication should be generated on such platforms.

The communication with clients should be happening as follows. After a request is sent by a client the information about the port and protocol of a service requested is completely defined. An active and valid connection should be established over a TCP/ IP stack. Simultaneously with that a corresponding state machine set is downloaded to the internal memory of the designed system. A TCP/IP stack for-

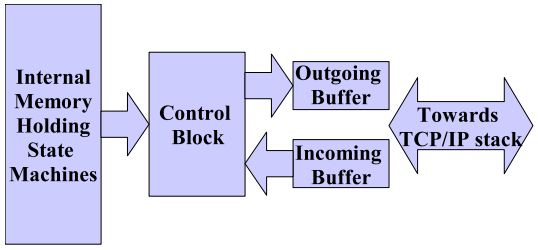

Fig. 2. Detailed block scheme

Table 1. Memory design details

	MEM0	MEM1	MEM2	MEM3	MEM4	MEM5	TOT
18Kb blocks used	1	15	15	1	2	2	36
Write port width	32	32	32	8	16	16	
Write port depth	7	13	13	11	11	11	
Read port width	8	32	32	8	8	16	
Read port depth	9	13	13	11	12	13	

wards the client request to the system which triggers internal logic actions destined to find the appropriate server answer from the preselcted and loaded state machine. Further data exchange between a client and the system simulating a server continues following the just described schema. The structural block diagram of the system is on Fig. 2. Downloading functionality, however, has not been implemented, since the focus of this initial work has mostly been at hardware aspects, with the simple goal of estimating the efforts and providing the initial platform concept for a complete system.

3.1. State Machine Format

Each state of a state machine stored in the memory is initially [4] described by three types of attributes. The first attribute is a list of transitions to next states. This can be seen as a list of strings or specifically a set of names of next states. This list is accompanied by transition conditions that could also be represented as a set of strings. A correct transition is made by satisfying equlity conditions between an incoming client request and an appropriate transition label. Transition conditions have a frequency values assigned to them. Frequency values are used to decide which transition should be chosen in the case that more transitions have stisfied a transition condition.

The second attribute is a list of labels. A label corresponds to information that is to be sent back to a client as a consequence of a servers state. Label can be seen as the payload to be encapsulated in a packet. A label is additionally marked with a label frequency value, simply because there might exist more answers to be sent from a single state, and frequency value, presents a mechanism for deciding which answer option is the best or excatly the one used more often. Both mentioned frequency values, label and transition, are deducted from the number of their respective occurrences in the state machine generation process performed on the original traffic records.

Finally, the third attribute is a list of signals. A signal can be seen as information to be exchanged with another state machine, potentially active in another communication process in the same time, and carrying a certain amount of significance at the particular currently executed case. Enhancement of interoperability is implemented in this way as it might be important for types of communication that include utilisation of multiple ports.

The described original structure had to be adapted to fit the hardware implementation proposed. Therefore, the third attribute was not given significance as it had to have in the case of implementation of the fully functional model, though it was implemented up to some extent.

3.2. The Memory Storage Block

Having in mind the previously described state machine format, a memory storage block was carefully crafted to permit an easy access to different segments of the relevant data. It is composed of six smaller blocks, implemented as RAM cores, again, to minimise the number of clock cycles required to write and read the data.

The first functional memory block, marked with MEM0 in Fig. 3, contains general information relevant to every state in a state machine. That information is composed of the number of transitions from each state, the number of signals corresponding to that state and the number of states labels. This general information is necessary to allow a control block, described later, to move through the memory system and obtain the relevant data. The second block, marked with MEM1, contains the list of all the labels for all the states, whilst the third block, MEM2, contains the list of all the transitions. Values of label frequencies are stored in the fourth memory unit, MEM3. The transition frequency and the future state values are stored in the fifth MEM4 block. The list of intercommunication signals was memorised in the sixth memory block, MEM5.

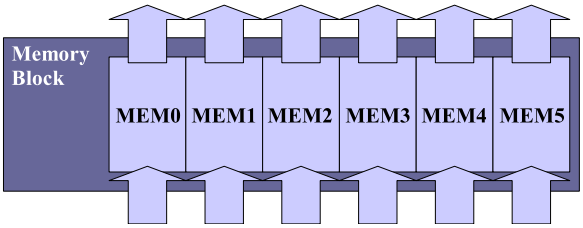

Fig. 3. Memory storage structure

From the implementation perspective, each memory block is a dual-port RAM dedicated to one type of the information. The design targeted the smallest chip from the Virtex4 family, the XCV4FX12 [3], on the Xilinx ML403 development board [7]. The sizes of memories were selected so as to achieve the maximum utilisation of the available RAM blocks. Details are presented in Table 1. The memory space on the selected chip equals to 36 18Kb RAM block units [3]. The maximum number of states in a state machine was fixed at 120, which for the initial state of the design development seemed completely acceptable equaly from the functional and implementation perspective.

3.3. Control Block and Buffers

The control block presents the core of the design. This part of the design coordinates all the other parts of the design, six memory units and TCP/IP interfaces, namely, to achieve a fully functional design. It has been designed as a state machine that combines the comparison of distinct factors as an essential part of its functionality.

The control block is foreseen to work according to the following scenario. Each client request triggers the start signal, presented by transition from *sx* to *s0* state in Fig. 4. The assumption is that memory blocks are filled correctly with the state machine related with the initial client request. The mechanism applied to fulfil this condition is still to be developed, but it is envisaged that this task will need to be implemented as a supervising software module.

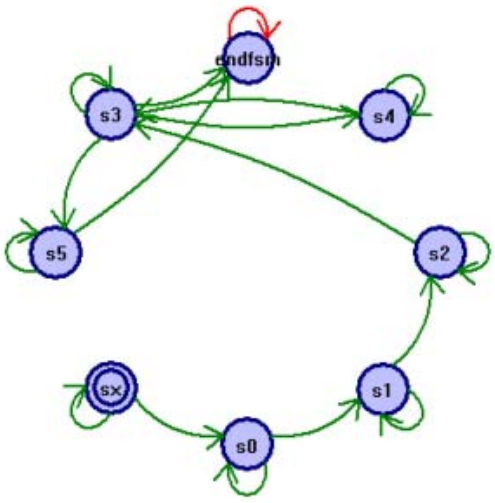

Fig. 4. Synthesized control block state machine

The State *s0* represents a process of reading the information from MEM0. To recall, MEM0 contains the number of transitions from each state together with other general information, which is needed to correctly establish the reading depth from all other memories. Machine then passes to state *s1*, which represents the selection process of an answer with the highest frequency value matching the payload received from a TCP/IP stack. This matching has to correspond to the initially received and recognised request sent by a client. After the selection of the answer with the highest frequency value the process continues by a simple retrieval of the correspondent reply from the MEM1 in the state *s2*.

The future state of the server side of the communication is then established in states *s3* and *s4*. Certain shifting between these two states is necessary in order to obtain the state with the highest frequency value. The state *s3* represents the reading process from MEM2 and *s4* from MEM4. The state *s5* is rarely used. It is implemented for the purpose of the communication with the supervising software and mentioned inter-state-machine communication that was not fully deployed, while *endfsm* state represents the sending of the determined best server answer to a TCP/IP stack. Finally, buffers in Fig. 2, are simply used to communicate with TCP/IP stack, they might develop themselves into FIFOs in future, whilst currently being represented as simple register structure.

Table 2. Xilinx Logic Utilization Summary Overview

	Used Resources	Available Resources	Utilization
Slices	1303	5472	24%
LUTs	1501	10944	13%
RAMBs	36	36	100%

3.4. Implementation and Results

The design as shown in Fig. 2, has been implemented on a Xilinx ML403 development board with a Virtex4 XCV4FX12 chip, using VHDL as a hardware description tool. The synthesis has been performed by Synplicity's SynplifyPro 8.6.2 tool; the simulation and functional verification in Aldec Active HDL 6.2 simulator; while place and route functions were performed by the Xilinx ISE Foundation 8.1 environment. The design has been tested in hardware, on mentioned board as a proof of proper and correct execution. The short Xilinx implementation summary is in Table 2.

4. CONCLUSION & FUTURE WORK

A view on potentials of the migration of a honeypot functionality to a hardware platform was given. The presented design should be seen as an initial work on the topic. It is clear that further work must include completion of the full design concept, including the supervising software and additional hardware components. In order to obtain a fully functional honeypot device it will be necessary to conduct more tests and to try different approaches for the TCP/IP

stack implementation, here referring to microprocessor option, using PowerPC core, or logic implementation of the stack.

Usage of bigger chips, from the same Virtex4 family, containing more block RAMs and more logic resources, even more PowerPC cores [3] might be suitable for complete transition of the design to an embedded platform. Two PowerPC cores would in that case allow an implementation of supervising software together with desribed hardware parts within a single embedded platform, thus the progress to a hardware based honeypot would obtain its usable form.

5. REFERENCES

[1] "Developments of the Honeyd Virtual Honeypot", available on-line at: http://www.honeyd.org

[2] S. Baldwin, "Lance Spitzner: Using Honeypots to Track the Bad Guys", available on-line at: http://www.raeinternet.com/ newsletter/interview_spitzner_050604.html

[3] Xilinx Inc., "Virtex4 User Guide", 2006.

[4] C. Leita, K. Mermoud, M. Dacier, "ScriptGen: an automated script generation tool for honeyd," *21st Annual Computer Security Applications Conference*, Dec. 2005.

[5] P. James-Roxby, G. Brebner and D. Bemmann, "Time-Critical Software Deceleration in an FCCM," *IEEE Symposium on FPGAs for Custom Computing Machines (FCCM04),* pp. 3-12, April 2004.

[6] Xilinx Inc., "PowerPC 405 Block Reference Guide," July 2005.

[7] Xilinx Inc., "Ml401/402/403 Evaluation Platform User Guide," May 2006.

[8] A. Lamb, "Survey of Trends in Honeypot Technology Users," March 2006, available on line at: http://www.rit.edu/ arl7969/whitepapers/alamb-3-2006.html

[9] The Honeynet Project: Tools for Honeynets, available on-line at: http://www.honeynet.org/tools/index.html

[10] P.T. Chen, C.S. Laih, F. Pouget and M. Dacier, "Comparative Survey of Local Honeypot Sensors to Assist Network Forensics," *International Workshop on Systematic Approaches to Digital Forensic Engineering*, November 2005.