MalCoBox: Designing a 10 Gb/s Malware Collection Honeypot using Reconfigurable Technology

Sascha Mühlbach
Secure Things Group
Center for Advanced Security Research Darmstadt (CASED)
sascha.muehlbach@cased.de

Martin Brunner, Christopher Roblee
Network Security and Early Warning Systems
Fraunhofer-Institute for Secure Information Technology (SIT)
{*martin.brunner*|*christopher.roblee*}*@sit.fraunhofer.de*

Andreas Koch
Embedded Systems and Applications
Dept. of Computer Science
Technische Universität Darmstadt
koch@esa.cs.tu-darmstadt.de

***Abstract*—Honeypots present networked computer systems with known security flaws to attackers and can serve to collect the executable code (malware) aiming to exploit the vulnerability. We describe and evaluate the proof-of-concept NetStage Architecture for a high-speed honeypot realized in reconfigurable logic. Dedicated hardware accelerators for the different network processing and detection layers allow the honeypot to operate at full speed of a 10 Gb/s connection and project the illusion of thousands of vulnerable systems at once. Furthermore, compromising the honeypot itself is significantly more difficult than in software honeypots, since all processing is handled by specialized hardware blocks instead of general purpose processors.**

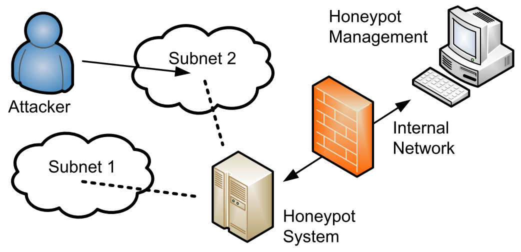

Figure 1. Integration of a honeypot system into a network

I. Introduction and Related Work

In today's highly networked and extremely heterogeneous computing environments, the increasingly sophisticated exploitation of security flaws has become a significant problem for private users, businesses, and even governments [1]. In general, attackers aim to gain control over or information from as many systems as possible. Thus, attacks themselves have been automated to a large degree. In one of the common techniques, the attacker attempts to inject a malware program (encompassing different types such as trojans, back-doors or spyware programs) through a security flaw (or user inattention) into the attacked system and have the malware executed.

To defend against current or anticipate future threats, security researchers must examine the malware currently distributed by attackers. Since reliable connections into the attacker ("black hat") community are rare, a common means is to collect the malware by setting up one (honeypot) or more (honeynet) computers with known vulnerabilities (simulated or real) on a publically accessible part of the Internet (but isolated from the researcher's internal network), and wait for them to be attacked (see Fig. 1). The malware can then be retrieved from the compromised honeypots [2].

A number of software implementations for honeypots exist (e.g., Nepenthes [3] or Honeyd [4]). However, software running on a general-purpose computer always carries the risk that the honeypot will be compromised for *real* and the attacker does indeed obtain full control over the machine. Even if the access to the researcher's internal network is blocked, the attacker could use the compromised honeypot to attack other external systems, leaving them to blame the honeypot operator (as one of his IP addresses will show up in address traces). In well-designed analysis environments, this risk can be minimized, but at the cost of a complex maintenance process. Beyond security concerns, the wide address ranges watched by the honeypot also strain network processing resources due to the high number of packets that must now be considered.

We propose the use of the specialized NetStage hardware Architecture (NSA) based on reconfigurable logic to cope with both the speed and security challenges. Prior studies of honeypot specific hardware-architectures do exist, but they have not considered high-performance aspects. For example, the architecture in [5] stores the FSMs modelling the protocols and vulnerabilities as tables in memory, which are then interpreted. This extends to embedding complete processor cores into the architecture to handle complex protocol steps in software. In addition to the security risks inherent in general-purpose processors, the work does not give throughput measurements at all. In our approach, all protocol handling (network processing and vulnerability emulation) are performed by dedicated hardware aiming for maximal throughput.

The paper is organized as follows: Section II describes the new architecture and its major characteristics. Section III

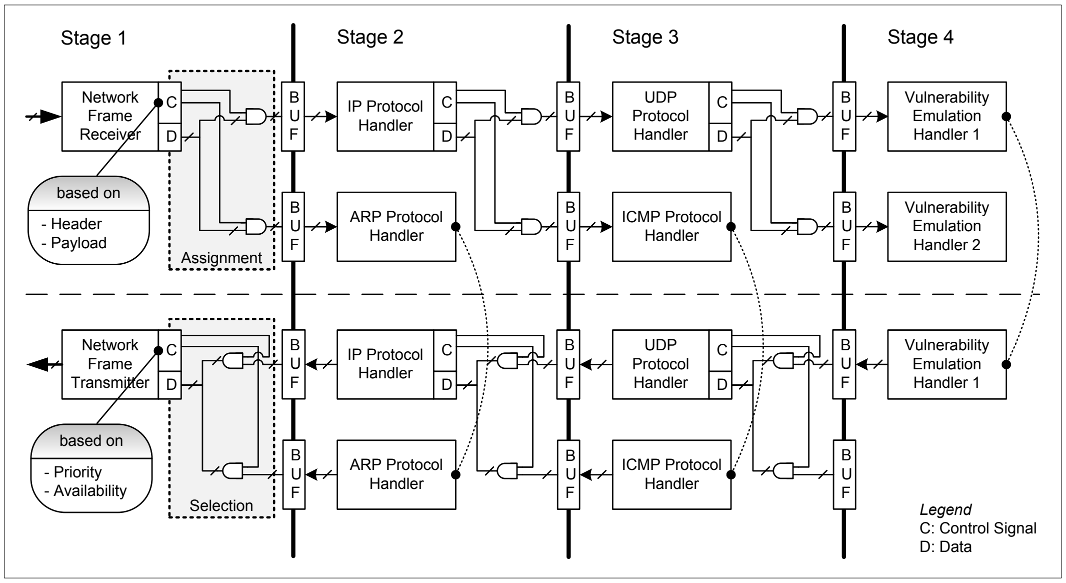

Figure 2. NetStage Architecture of the malware collection hardware honeypot (MalCoBox)

covers implementation details, followed by a discussion of synthesis results in Section IV. We close with a conclusion and an outlook towards further research in the last Section.

II. Architecture

Our architecture should be both flexible (to allow freedom for later experimentation) as well as support high-speed processing (10Gb/s streams on Virtex 5-generation devices). To achieve both ends, we use a hierarchical design reflecting the levels of the protocol stack [6]. Lower-level network operations (e.g., ARP requests) can be quickly handled in lower architecture stages. Only the application-level vulnerabilities are handled at the highest architecture stage. In this fashion, new protocol or vulnerability handlers can be easily “plugged-in” at the appropriate stage (Fig. 2).

Adjacent stages are loosely interconnected using buffers to keep brief variations in throughput in a single stage from interfering with the system-level throughput. This buffering capability also opens the way for future work on dynamically reconfiguring handlers. Note that the latency increase due to data passing through multiple buffers is not an issue for this application, only throughput is.

In addition to the stages, the NSA is also structured along the receive and transmit datapaths, with handlers also being split along those lines. Communication between the receive and transmit parts of a handler is possible. This is indicated by the dotted arcs in Fig. 2, e.g., the receiving part of an ICMP handler can initiate the sending of an ICMP reply.

A. Receive Path

Raw network packets received at the media interface are initially classified according to their link-level protocol information. Those matching the MAC address of our honeypot and requesting the IP or ARP protocols are forwarded to the corresponding handler in the next stage, all other packets will be discarded right away. While ARP packets can be processed directly in Stage 2, IP packets are just classified there and forwarded to Stage 3.

In Stage 3, ICMP requests will be processed directly. Other IP traffic will be handled by the corresponding transport protocol handler. The actual application-level vulnerability handlers will generally reside in Stage 4 (but could also be inserted earlier, if necessary).

As packets proceed upwards, lower-level protocol headers will be stripped away and only information required for further processing will be retained.

B. Transmit Path

The transmit path is organized similarly. A higher level stage wishing to send a packet puts the desired payload data and the internal header into the stage output buffer. The lower-level handler accepts the transmit request and then creates the appropriate “real” protocol header.

C. Vulnerability Emulation Handlers

The top-level stage consists of the custom vulnerability emulation handlers (VEH) currently active in the system. Figure 3 shows a sketch of their internal structure. The VEH uses a set of matching rules to extract information from incoming packets (e.g., a login user name). Outgoing packets are composed by filling in the appropriate fields in stored packet templates (e.g., echoing the user name provided back to the attacker).

The main task of our honeypot, the collection of malware, is also performed by the VEH, e.g., by regular expression

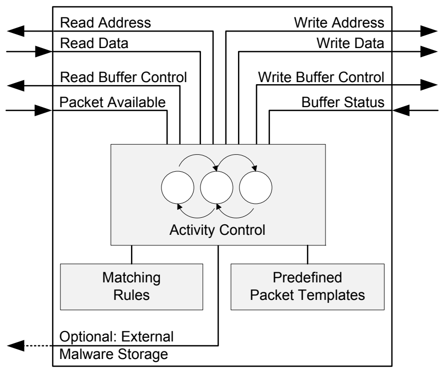

Figure 3. General vulnerability emulation handler structure

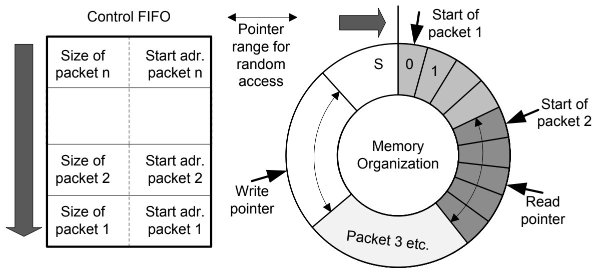

Figure 4. Ring buffer implementation

matching. This is highly vulnerability-dependent: The attacker could provide the malware directly during the attack as executable binary code, or cause the attacked system to download it from somewhere else. The malware (regardless of form) is stored in the honeypot system memory (which can be, e.g., SDRAM). It can be retrieved either using a dedicated network interface (isolated from the public network) or by PCI Express (when the honeypot is directly attached to a host computer) by the management station. Each malware collected is also stamped with the identifier of the VEH that found it, the collection time, and source- and destination IP addresses.

III. Implementation

We have implemented a proof-of-concept realization of the MalCoBox on an FPGA-based reconfigurable computing platform which has the required 10 Gb/s network interfaces.

A. Target Platform

We are using the BEEcube BEE3 platform, which has been fitted with four Xilinx Virtex 5 FPGAs (2x LX155T, 2x LX95T). In contrast to many other platforms, in has eight 10 Gb/s interfaces on board, externally available as CX4 connectors. The initial prototype discussed here uses one of the LX155T FPGAs and a single 10 Gb/s port, but could easily be extended.

B. Core System Functionality

The lowest levels of the network interfaces are realized using the Xilinx XAUI [7] and 10G MAC IP [8] cores. The external CX4 connectors are driven by RocketI/O transceivers connected to the XAUI block. The first processing stage of our hierarchical NSA connects to the 10G MAC. To achieve a raw throughput of 10 Gb/s (assuming no inter-frame gaps or unused bytes) on the 64b data words output by the 10G MAC, the entire system operates on a clock speed of 156.25 MHz.

However, beyond the network interface itself, all stages are designed to allow operation at 20 Gb/s, achieved by using 128b data paths. We anticipate the possibility of momentary stalls in the data transfer (e.g., due to complex rules matchable only on multiple cycles, or even partial reconfiguration of handlers) and want the affected handlers to be able to "catch-up" with the normal 10 Gb/s traffic by burst-processing the data accumulated in the inter-stage buffers at 20 Gb/s. On the transmit side, the 128b data words are converted to 64b streams for input into the 10G MAC.

The core handlers are implemented in a straight-forward manner reflecting the NSA described in Sec. II.

Since the efficient implementation of hardware-based TCP offload engines for 10 Gb/s is a complex design task itself that is outside the current focus of this work, we concentrated on the base architecture instead and implemented only UDP as a core handler in the proof-of-concept system (but see Sec. V).

C. Inter-Stage Buffers

The inter-stage buffers play a crucial role in decoupling the different processing stages within the NSA. To achieve most-efficient buffer use without fragmentation, we use a ring buffer-like structure as base. Furthermore, and in contrast to a FIFO, we allow random manipulation of the read and write pointers. This can greatly speed-up processing in Stage 2 and later, since handlers often require access only to a few bytes of the packet, while irrelevant data can be quickly skipped. The transmit path also profits from the random access. E.g., a checksum can be written into the header even after the packet body has already been stored, an approach not possible with pure FIFOs. However, FIFOs are useful for their order-preserving properties.

We thus use the combined approach in Figure 4. Packet data itself is stored in 128b wide, dual-port BlockRAM. Separate counters act as read/write-pointers into the RAM for traditional ring-buffer operation. In addition, we use a FIFO for buffer management: Once a packet has been completely written into the ring-buffer, a FIFO entry containing its 16b start address in the BlockRAM and 16b length is created. The handler in the next stage begins processing as soon as such a FIFO entry is available.

The ring buffer addressing logic is also set-up to gracefully discard entire packets instead of stalling should the

Table I
SYNTHESIS RESULTS FOR THE OVERALL SYSTEM

Stages	Handler	LUT	Reg. Bits	BRAM
Stage 1 to 3	Core Handlers	2,799	2,760	24
Stage 4	SIP VEH	1,119	364	6
NetStage Architecture (UDP/IP)		**3,965**	**3,133**	**30**
mapped NSA incl. MAC + XAUI		**7,176**	**6,832**	**47**
(% of XC5VLX155T)		(7%)	(7%)	(22%)

buffer begin to overflow. In practice, this should happen only rarely, with all stages aiming for 20 Gb/s operation.

D. Vulnerability Emulation Handlers

The actual honeypot functionality will be implemented in the VEHs, specific for a single or a class of related vulnerabilities. We have developed an HDL template for the VEHs (see Fig. 3) to simplify the creation of new handlers.

The proof-of-concept implementation presented here emulates a stack overflow vulnerability of the software SIP SDK sipXtapi [9]. The exploit uses a buffer overflow occurring if a SIP INVITE packet contains a CSeq field value exceeding 24 bytes in length. Matching rules look for the beginning of the CSeq field (the string CSeq:) and its end (indicated by the next field identifier Max-). If the difference between their positions exceeds 24, a UDP message containing the matched pattern (the malware) and logging data is sent back to the management station.

IV. SYNTHESIS RESULTS

The design was synthesized using Synplify Pro 9.6.2 and mapped with Xilinx ISE 11.4, targeting a Virtex 5 LX155T and aiming for a clock speed of 156.25 MHz. Table I gives a summary for the different handlers of the NSA and the entire system. The separated receive and send handlers for the core protocols (see Fig. 2) are counted together. The largest amount of logic is used by the VEH as the parallel regular expression matching needs more combinatorial logic than simple rules in handlers only forwarding bytes (see the ratio of LUTs vs. Register Bits).

Including the 10G MAC and XAUI blocks, roughly 7% of the LX155T device resources are used and the mapped MalCoBox design occupies 3291 slices (13%) on the chip. From this, we estimate that we can put 20-40 VEHs on a single chip (depending on the complexity), which could easily be extended due to our hierarchical design to all four devices in the BEE3. In practice, a honeypot will be looking at malware injected through *current* exploits. Thus, being able to attract roughly a hundred different exploits in parallel seems sufficient for this purpose.

V. CONCLUSION AND NEXT STEPS

With NetStage, we have presented a flexible system architecture to build our MalCoBox, a high-speed hardware-accelerated malware collection honeypot. It can keep up with 10 Gb/s traffic while spanning large IP address ranges and has spare processing capacity even for complex vulnerability emulation. By using a hierarchical and modular architecture modelled on the protocol stack, it is easy to add new functionality. Similarly, new vulnerability handlers can also be developed quickly due to their regular structure.

Our proof-of-concept implementation has covered the basic core modules of the system as well as a sample vulnerability emulation handler and was successfully tested on a current reconfigurable computer platform. Both the limited chip area required as well as the spare processing capacity show that this approach is feasible even with current FPGAs.

We will continue our work in this area. In addition to developing more vulnerability handlers, we also intend to integrate a handler for the TCP protocol. This, in turn, would enable the addition of HTTP handlers to emulate vulnerabilities in web servers. All of these features will also need to be stress-tested in a real production environment (e.g., university or ISP), preliminary talks to this end have already been initiated.

Mid-term research will also consider dynamic reconfiguration to swap vulnerability handlers at run-time as well as different memory organizations, optimized for more complex higher-level protocols such as TCP.

ACKNOWLEDGMENT

This work was supported by CASED (www.cased.de).

REFERENCES

[1] "Security threat report: 2010," Sophos Group, 2010. [Online]. Available: http://www.sophos.com/security/topic/security-report-2010.html

[2] N. Provos and T. Holz, *Virtual Honeypots: From Botnet Tracking to Intrusion Detection*. Addison-Wesley Professional, 2007.

[3] "Nepenthes." [Online]. Available: http://nepenthes.carnivore.it

[4] "Honeyd." [Online]. Available: http://www.honeyd.org

[5] V. Pejovic, I. Kovacevic, S. Bojanic, C. Leita, J. Popovic, and O. Nieto-Taladriz, "Migrating a honeypot to hardware," in *SECUREWARE '07: Proc. Intl. Conf. on Emerging Security Information, Systems, and Technologies*. IEEE Computer Society, 2007, pp. 151–156.

[6] R. Braden, "Requirements for Internet Hosts - Communication Layers," RFC 1122 (Standard), Internet Engineering Task Force, Oct. 1989, updated by RFCs 1349, 4379. [Online]. Available: http://www.ietf.org/rfc/rfc1122.txt

[7] "Xaui v9.1 user guide," Xilinx, 2009.

[8] "10-gigabit ethernet mac v9.3 user guide," Xilinx, 2009.

[9] M. Thumann, "Buffer overflow in sip foundry's sipxtapi," 2006. [Online]. Available: http://www.securityfocus.com/archive/1/439617