A Dynamically Reconfigured Network Platform for High-Speed Malware Collection

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

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

***Abstract*—Malicious software has become a major threat to computer users on the Internet today. To combat it, security researchers need to gather and analyze many samples to develop proper defense mechanisms. The setting of honeypots, which emulate vulnerable applications, is one method of gathering attack code. In contrast to the conventional software-based honeypots, we have proposed a dedicated hardware architecture for honeypots to both allow high-speed operation at rates of 10+ Gb/s as well as a higher resilience against attacks on the honeypot infrastructure itself. We now improve the flexibility of our prior solution by using partial dynamic reconfiguration to update the functionality of the honeypot during operation.**

I. Introduction

The significant increase of malicious software (malware) in recent years (see [1]) requires security researchers to analyze an ever increasing amount of samples for developing effective prevention mechanisms. One method for collecting a large number of samples is the use of low-interaction honeypots (e.g., [2]). Such dedicated computer systems emulate vulnerabilities in applications and are connected directly to the Internet, spanning large IP address spaces to attract many different attackers. But in addition to having performance limitations, such software systems also suffer from being compromisable themselves (can be subverted to attack even more hosts).

In this context, we have proposed the idea of a low-interaction malware-collection honeypot realized entirely in reconfigurable hardware without any software components in [3]. The core of our MalCoBox system is a high-speed implementation of the basic Internet communication protocols, attached to several independent vulnerability emulation handlers (VEH), each emulating a specific security flaw of an application (see Fig. 1). We have demonstrated the feasibility of that approach by implementing a prototype on a FPGA platform, fully employing the power of dedicated hardware resources to support 10+ Gb/s network traffic.

A common question of potential MalCoBox users is how the platform can be updated during operation with new or improved VEHs to react to the changing threat landscape. We have thus extended our solution by using partial dynamic reconfiguration [4] to update individidual VEHs while the system stays in operation.

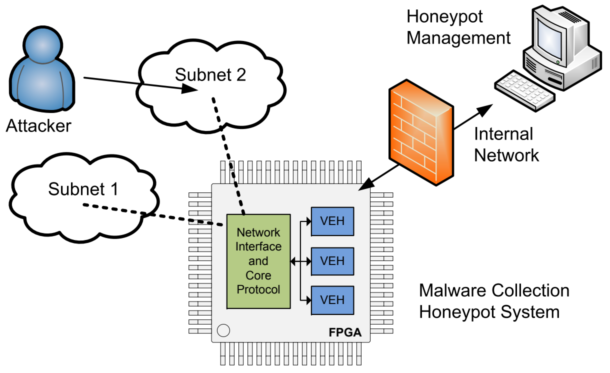


Figure 1. Hardware-Based Malware Collection

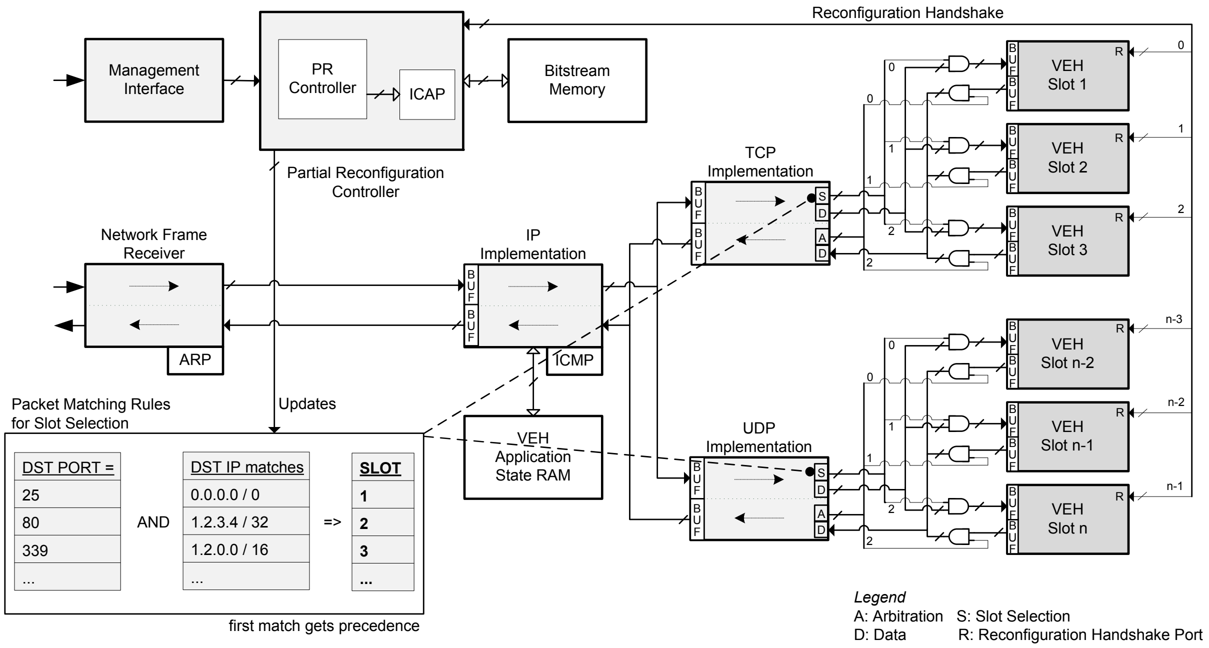
The paper is organized as follows: Section II briefly describes the core architecture and its major characteristics. The next Section III covers the details of the reconfigurable VEH slots, followed by a discussion of our partial reconfiguration strategy in Section IV. The implementation of the complete system on an actual FPGA platform is described in Section V, with Section VI giving experimental results. We close with a conclusion and an outlook towards further research in the last Section.

II. Key Architecture Components

The system architecture of the honeypot is based on the idea of a hierarchy reflecting the levels of the Internet protocol. Figure 2 shows our so-called NetStage Architecture (discussed in greater detail in [3]), now including the extensions to support partial reconfiguration which are the focus of this work.

The VEHs are modules loosely interconnected with the core system by buffers. The architecture provides slots into which VEHs can be configured. Each VEH slot provides the same inputs and outputs, and all the VEHs follow a similar structural pattern by reading and writing data only from and to the buffers. In that fashion, new modules can be easily "plugged-in" in any slot. The buffers also mitigate the effect of brief VEH-local stalls on system-level throughput.

VEHs share the underlying implementations of the core protocols (IP, TCP, UDP) in NetStage. These have been very



DST PORT =		DST IP matches		SLOT
25		0.0.0.0 / 0		1
80	AND	1.2.3.4 / 32	=>	2
339		1.2.0.0 / 16		3
...		...		...

Figure 2. Core Architecture of the Partially Reconfigurable Malware Collection System

carefully optimized to achieve a throughput of at least 10 Gb/s to keep up with the line-rate of the 10 Gb/s external network interface.

A. Vulnerability Emulation Handler

When a packet has passed the core, it will be forwarded to the responsible VEH performing the actual malware detection and extraction. To this end, the core contains a mapping table with matching rules for the different vulnerability emulations currently active in the system. In the MalCoBox refinement presented here, the mapping table is writable to allow dynamic alteration of the actual VEHs used, a feature that will be exploited when reconfiguring.

A basic set of mapping rules includes the destination port, destination IP and netmask. The latter allows us to set-up separate IP address ranges which use VEHs for different vulnerabilities on the same port (e.g., many handlers will listen on the HTTP port 80).

With the processing speed achievable in reconfigurable hardware, these basic rules could also be extended to directly match payload contents. However, this would require dynamic reconfiguration of the actual matching unit, instead of just writing new values into registers (as in the basic matcher). Since our current VEHs can be activated independently of the payload, we will use the simpler basic approach.

B. VEH Application State Memory

In some cases, VEHs have to track session state to generate an appropriate response. NetStage provides a central facility of storing per-connection state: When a packet is passing the IP implementation, the globally maintained state information is attached to the packet in a custom control header which accompanies every packet through the system. The VEH can read this information, act on it, and update it. The value is written back to the state memory when a response packet (or an empty state write packet) passes the IP implementation on the transmit path. Such a centralized storage is more efficient than attempting to store state in each VEH (which would fragment the capacity of the on-chip memories).

The global VEH application state memory can also be used to save/restore VEH state during reconfiguration, allowing the seamless swapping-in of newer (but state-compatible) versions of a VEH.

C. Partial Reconfiguration Controller

The partial dynamic reconfiguration of VEHs is managed by the Partial Reconfiguration Controller (PRC), which is connected to the FPGAs internal configuration access port (ICAP). On the application side, the PRC is connected to the MalCoBox management interface (either by a PCI Express endpoint or a dedicated 1 Gb/s network link, depending on the selected deployment mode of the system).

The PRC is also connected to the individual VEH slots by a number of handshake signals to inform the VEHs about their impending reconfiguration (for a clean shutdown, to save state in the global memory, etc.) and to check whether the VEH is idle.

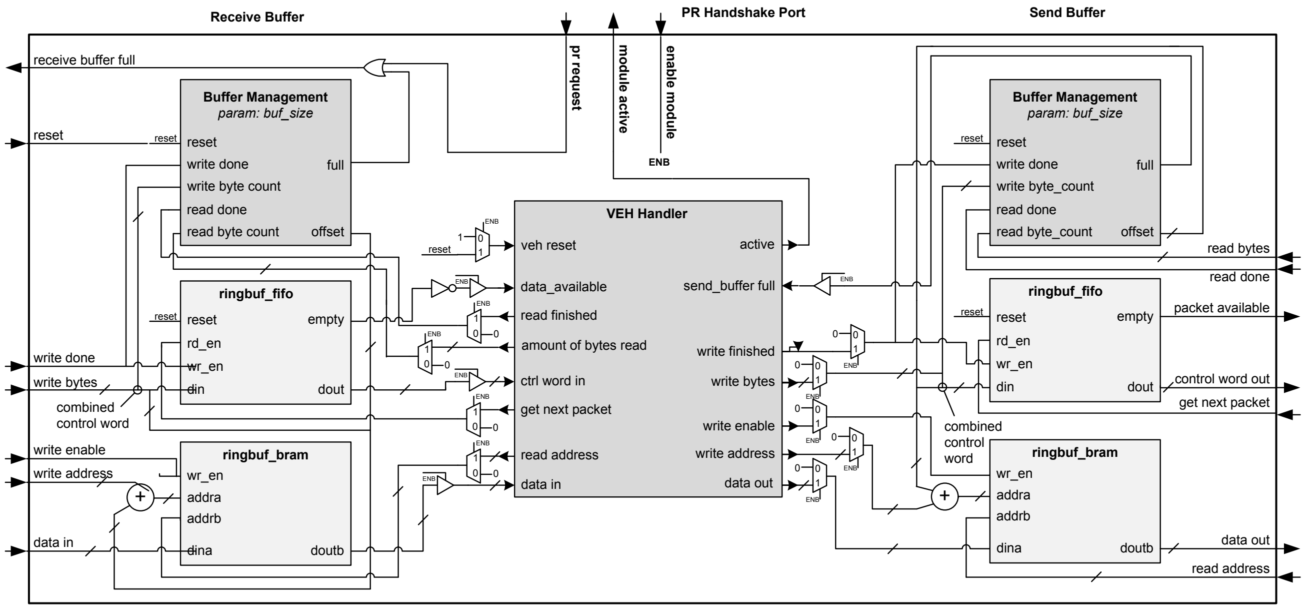


Figure 3. Wrapper encapsulating a Vulnerability Emulation Handler Slot

An attached bitstream memory can hold several partial bitstreams to allow the system to be reconfigured independently of the management station.

III. Reconfigurable VEHs

To support independent partial reconfiguration (PR) of any of the VEH slots, a wrapper encapsulates the actual VEH implementation module (see Fig. 3). This wrapper includes glue logic controlled by the PRC to disconnect/reconnect all inputs and outputs of the VEH module. This clean separation is essential to avoid introducing errors in the rest of the system when reconfiguring.

The wrapper also contains the send and receive buffers for each module. As all the handlers share the same buffer structure, it is more efficient to keep it static than configure it with each VEH. The inputs and outputs of the wrapper are directly connected to the UDP or TCP implementations within the MalCoBox core (see Fig. 2).

Internally, the VEHs themselves all have very similar message-oriented architectures: Packet data is read as 32b words from the receive buffer, processed within the VEH, and output packet data is written back to the send buffer (if required). The output packet can be a response to the client, or a system-internal control packet (e.g., state write information for the global memory). This structure allows all VEHs to share the same interface (important for PR) and is sufficiently flexible for the MalCoBox' needs.

A. *Buffer Management*

The packet buffers in the wrappers are organized as modified ring-buffers, which avoids fragmentation. The packet data itself is stored in a dual-port BlockRAM, allowing producers and consumers of packet data to work independently. An additional FIFO is used to manage the BlockRAM addressing. Each FIFO entry consists of a 16b start adress and a 16b length field, and presence of an entry indicates that a complete packet has been stored in the BlockRAM.

To control the buffer fill status, a buffer management unit (BMU) keeps track of the number of bytes read and written. When the number of bytes in the buffer exceeds the capacity, the buffer full signal is asserted. Then, as a system-wide policy, the TCP and UDP implementations in the core will gracefully discard entire packets (instead of stalling) to avoid congestion. The BMU also keeps track of the start address for the next packet within the buffer, avoiding the need to save/restore this state when a VEH is reconfigured.

IV. Partial Reconfiguration

Partial bitstream data is transferred from the management station (usually an external PC) to the MalCoBox via the management interface. The underlying protocol consists of the raw bitstream preceded by a reconfiguration header (see Figure 4) and an optional address offset in the MalCoBox configuration memory (which can thus hold multiple bitstreams locally).

A. *Partial Reconfiguration Process*

When the PRC receives a reconfiguration request (which includes a pointer to the desired bitstream in configuration memory), it initially informs the wrapper of the target slot that the slot is about to be reconfigured. This will stop the receive buffer of the VEH from accepting new packets (which will thus accumulate in an earlier stage). The VEH

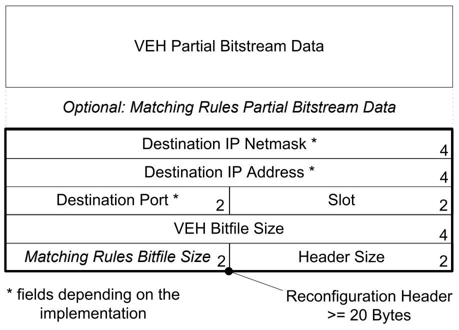

VEH Partial Bitstream Data
Optional: Matching Rules Partial Bitstream Data
Destination IP Netmask * 4
Destination IP Address * 4
Destination Port * 2
Slot 2
VEH Bitfile Size 4
Matching Rules Bitfile Size 2
Header Size 2
* fields depending on the implementation
Reconfiguration Header >= 20 Bytes

Figure 4. Custom PR header and bitstream data

is allowed to process all of the packets held in the buffer at this time, asserting a signal to the PRC on completion. The PRC then deactivates the VEH and, using information in the incoming VEH's reconfiguration header, updates the mapping tables with the characteristics of the incoming VEH about to be configured. This will forward matching packets to the incoming VEH's receive buffer.

Meanwhile, the now inactive VEH is disconnected from the slot wrapper by the PRC setting the appropriate control signal. Now the actual PR occurs: The bitstream data is read from memory and transformed into a Select MAP protocol [5] compatible form, before being fed into the ICAP. Once reconfiguration is completed, the PRC re-enables the VEH-wrapper connections and allows the new VEH to wake up in its reset state.

Since the management console is assumed to run trusted software under control of trusted operators, we do not perform security measures during this process. But such functionality could be easily added without affecting the base architecture (see, e.g., [6], [7]).

V. Implementation

The system has been implemented on the BEEcube BEE3 FPGA-based reconfigurable computing platform equipped with eight 10 Gb/s network interfaces and four Xilinx Virtex 5 FPGAs (2x LX155T, 2x LX95T). Currently, we are only using one of the LX155T FPGAs, but this could be easily extended.

The Xilinx XAUI and 10G MAC IP cores provide the network connectivity. Within the system, the processing throughput is doubled by extending the 64b data path of the 10G MAC to 128b, while keeping the 156.25 MHz clock speed. This allows us to react to brief stalls in the data flow: Affected handlers are able to "catch-up" with the normal 10 Gb/s traffic by burst-processing the data accumulated in the buffers.

The ICAP in the PRC is driven by a separate clock to allow varying reconfiguration speeds. The data width is set to the maximum of 32b. Currently, configuration data is stored on-chip in 256 KB of BlockRAM. Management access is implemented as a dedicated network interface, directly connected to a standard PC. The PRC receives bitstream data and control operations over the network using a custom protocol. Perl scripts are used to assemble the appropriate network packets.

A. Vulnerability Emulation Handler

To test the system, we have created a number of VEHs emulating different vulnerabilities and applications. Extracted malware is currently sent to the management PC via an UDP packet with a fixed (set during compile time) IP and MAC address. In addition to controlling FSMs, the VEHs contain additional hardware logic to perform tasks such as parallel pattern matching for high speed operations.

One of the UDP-based VEHs looks for packets exploiting a vulnerability of the software SIP SDK sipXtapi [8]. Another UDP-based VEH has a similar structure and is emulating a vulnerable MSSQL 2000 server [9]. As example for TCP-based VEHs, we implemented simple web and mail server emulations. They could be used, e.g., to monitor attack attempts on a login page or to receive large amounts of SPAM mails that often contain malware attachments or hidden links to malware download sites.

B. Buffers

MalCoBox is intended to attract attacks incoming from the Internet. Thus, their packets will generally have transited wide-area networks and will only rarely have sizes exceeding the Ethernet MTU limit of 1500 bytes. We size our BlockRAM-based buffers accordingly: Assuming all handlers in our implementation achieve a steady-state throughput of at least 10 Gb/s, a buffer size of two maximum size packets (3000 B) will ensure stall-free operation. Practically, we have to use two 36Kb simple dual-port BlockRAMs (each providing a bus width of 72b) to achieve a bus width of 128b. This results in an actual buffer size of 8 KB (512 x 128b).

VI. Results

The design was synthesized using Xilinx ISE 12.1 and mapped with PlanAhead 12.1, targeting a Virtex 5 LX155T FPGA and aiming for a clock speed of 156.25 MHz. Partial reconfiguration was implemented using the newest partial reconfiguration flow available in PlanAhead 12.1 [10].

System tests were performed by simulation as well as on an actual BEE3 machine connected to a host PC, sending data to the VEHs. Partial reconfiguration was performed under operator control, loading in new bitstreams via network from the management station.

For our prototype, we placed 20 VEH slots on the FPGA, which seems to be a reasonable value according to the total number of BlockRAMs available on the LX155T.

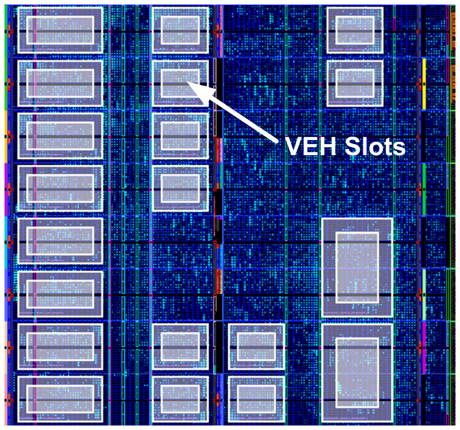


Figure 5. FPGA Layout for 20 VEH Slots

Table I
SYNTHESIS RESULTS FOR THE SYSTEM

Module	LUT	Reg. Bits	BRAM
SIP VEH	1082	358	0
MSSQL VEH	875	562	0
Web Server VEH	1026	586	0
Mail Server VEH	741	362	0
Core Implementation	20,764	12,226	190
Core with MAC and XAUI	25,810	17,173	202

A. Synthesis Results

Table I gives a summary of area requirements for the core and the various VEHs, the latter showing only little variation. Amongst them, the SIP VEH requires the most LUTs, as it contains the most complex pattern matching algorithms. The core implementation (ARP, ICMP, IP, UDP, TCP) including the 20 VEH wrapper slots occupies around 26% of the LUTs available on a LX155T. 100 of the Block-RAMs are used for the wrapper send and receive buffers (four as RAM and one as FIFO, for each of the 20 slot wrappers).

The VEH module slots were placed manually on the FPGA and sized (see Table II) based on the resource usage indications provided by the sample VEH synthesis results. The resulting layout can be seen in Figure 5. Instead of setting the slots to an equal size, we decided to have diferent sizes to offer more flexibility while not wasting too much space, as the VEHs may vary in their functionality and size.

To be able to assign each handler to different slots, we ran the place and route process with different configurations. Each run produces the partial bitfiles for a specific VEH-Slot combination. As they are independent, we can use them in arbitrary combination later.

B. Partial Reconfiguration Results

An operator-managed PR operation (store and reconfigure) performed through the management interface could be completed in less than a second, which is more than sufficient for the common scenario of relatively infrequent in-the-field updates of VEHs (once every few days).

Table II
SLOT SIZE DISTRIBUTION AND RECONFIGURATION TIME

Qty.	LUT / BRAM	Bitfile Size	Reconfiguration Time			
			78,125MHz		156,25MHz	
			w/o SD	w/ SD	w/o SD	w/ SD
10	1496 / 0	64KB	209us	213us	105us	108us
4	2176 / 0	106KB	347us	350us	173us	176us
4	2176 / 2	118KB	387us	389us	193us	195us
2	4144 / 0	162KB	531us	534us	266us	269us

However, for later extension of the system to autonomous dynamic reconfiguration, we have already measured the performance when reconfiguring from a locally stored bitstream (instead of accepting a new one over the network): Table II lists the reconfiguration time for two different clock speeds (the second one operating the ICAP beyond the specified limits, but still reliable). As the scenario, we assume that the receive buffer of the VEH to be replaced is half full and that the VEH is able to process data at 10 Gb/s (being conservative, since all of our current VEHs can actually handle 20 Gb/s). We show reconfiguration times both including and excluding the clean shutdown sequence (abbreviated SD in the Table) for an outgoing VEH (allowing it to process the remaining packets).

One can see that the time required for cleanly shutting down the outgoing VEH is negligible. Most of the time is actually taken by feeding the reconfiguration bitstream into the ICAP, thus limiting the overall reconfiguration speed. Thus, the size of the VEHs will also be important for fast reconfiguration and justifies our approach of heterogeneously sized VEH slots (we can configure the 10 smaller VEH slots much faster than the 4+4+2 larger ones).

By taking the numbers of Table II and assuming a continous reconfiguration process at the highest clock rate, a new handler could be available on average every 155 us (assuming a memory being able to provide a throughput of 625 MB/s, easily achievable using current SDRAM). A new VEH could therefore be provided 6451 times per second to incoming packets, while the other slots still remain active processing incoming packets. This should be sufficient to later self-adapt the MalCoBox to the network situation by loading the corresponding VEHs from memory, as not every packet received by the honeypot will result in a reconfiguration event (certain exploits are more likely than others and especially TCP-based exploits consist of streams of multiple packets for a single handler).

In practice, our honeypot will be looking at malware injected through *current* exploits. Thus, we expect a set of roughly a hundred different VEHs. When implemented in a single static design, they will not fit on current FPGAs. Autonomous dynamic reconfiguration offers a smart way of extending the number of VEHs beyond this physical limit.

C. *Impact of data path width*

To evaluate the impact of the 128b data path on the VEH size, we created a 64b version of the SIP and the Web Server VEH and compared it to the 128b implementation (Table III). Data path conversion between the core and the VEHs can be easily performed by the wrappers at the cost of a reduced throughput for the attached VEH.

The area overhead of the 128b version is roughly 75% for the SIP VEH and 65% for the Web Server VEH. This was to be expected, since the VEHs mostly read data from the input buffer and write data to the output buffer. The area required is thus strongly related to the bus width.

Given these results, it will be an option to implement VEHs with a reduced data path while keeping the core architecture at 128b, as this could significantly reduce the size of the slots and, in turn, also the reconfiguration time by roughly 40%. By carefully optimizing the design of these VEHs, the impact of a potential speed decrease below 10 Gb/s could be reduced up to a certain extent. However, the 20 Gb/s burst capability of the core system could also compensate for this outside of the slow 64b VEH (as long as the input link is not saturated with requests).

Together with the data path area, the BlockRAM usage is also reduced: With 64b operation, we can now narrow the buffers and only require three BlockRAMs per wrapper instead of five for 128b VEHs (see Section V-B), again increasing the overall number of possible VEH Slots.

Table III
SYNTHESIS RESULTS FOR 128B AND 64B VEHS

Handler	LUT	Reg. Bits
SIP VEH 128 Bit	1082	358
SIP VEH 64 Bit	619	278
Web Server VEH 128 Bit	1026	586
Web Server VEH 64 Bit	663	244

VII. CONCLUSION AND NEXT STEPS

With this refinement of our MalCoBox platform, we have presented a scalable system architecture to build a high-speed hardware-accelerated malware collection solution that offers great flexibility through partial reconfiguration. A management interface allows instant updates or replacements of single vulnerability emulation handlers by loading new partial bitstreams, without interupting the operation of the remaining system.

With the high performance of the dedicated hardware, the VEHs actually performing the malware detection and extraction can contain a wide range of functionality: They can embed complex regular expression logic as well as simple request-response patterns, while still reaching the required throughput of 10 Gb/s. Furthermore, our hardware approach is resilient against compromising attacks and significantly reduces the risk of operating hoenypots in a production environment. The presented implementation covered the core architecture as well as a number of sample VEHs and showed the feasibility of the approach. The results indicate that operators have a great flexibility to adapt the system to their needs: Individual VEH complexity and total vulnerability coverage by different VEHs can be traded-of by altering the distribution of VEH slots sizes; throughput and area can be traded-of by selecting between VEH implementations with 64b and 128b processing widths.

We will continue our work in this area. MalCoBox will be stress-tested in a real production environment connected to the Internet (e.g., university or ISP), preliminary talks to this end have already been initiated. From this, we expect to gain valuable information on how to improve the architecture and its parameters in the future. We will also extend the system to multiple FPGAs (e.g., the four on the BEE3 platform) to further increase the number of VEH slots. Finally, with the high speeds of local reconfiguration, we can begin work on an autonomously reconfiguring honeypot that self-adapts to current network traffic to present the maximal attack surface.

REFERENCES

[1] "Internet security threat report, volume xv," Symantec, 2010. [Online]. Available: http://www.symantec.com

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

[3] S. Mühlbach, M. Brunner, C. Roblee, and A. Koch, "Malcobox: Designing a 10 gb/s malware collection honeypot using reconfigurable technology," in *FPL '10: Proceedings of the 20th International Conference on Field Programmable Logic and Applications*. IEEE Computer Society, 2010, pp. 592–595.

[4] J. W. Lockwood, N. Naufel, J. S. Turner, and D. E. Taylor, "Reprogrammable network packet processing on the field programmable port extender (fpx)," in *FPGA '01: Proceedings of the 2001 ACM/SIGDA ninth international symposium on Field programmable gate arrays*. ACM, 2001, pp. 87–93.

[5] "Virtex-5 fpga configuration user guide," Xilinx, 2009.

[6] K. v. d. Bok, R. Chaves, G. Kuzmanov, L. Sousa, and A. v. Genderen, "Fpga reconfigurations with run-time region delimitation," in *Proceedings of the 18th Annual Workshop on Circuits, Systems and Signal Processing (ProRISC)*, 2007, pp. 201–207.

[7] Y. Hori, A. Satoh, H. Sakane, and K. Toda, "Bitstream encryption and authentication using aes-gcm in dynamically reconfigurable systems," in *IWSEC '08: Proceedings of the 3rd International Workshop on Security*. Springer-Verlag, 2008, pp. 261–278.

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

[9] D. Litchfield, "Microsoft sql server 2000 unauthenticated system compromise." [Online]. Available: http://marc.info/?l=bugtraq\&m=102760196931518\&w=2

[10] "Partial reconfiguration user guide," Xilinx, 2010.