Exposing Vulnerabilities of Untrusted Computing Platforms

Yier Jin*, Michail Maniatakos* and Yiorgos Makris†
*Department of Electrical Engineering, Yale University
†Department of Electrical Engineering, The University of Texas at Dallas
{yier.jin@yale.edu, michail.maniatakos@yale.edu, yiorgos.makris@utdallas.edu}

***Abstract*—This work seeks to expose the vulnerability of untrusted computing platforms used in critical systems to hardware Trojans and combined hardware/software attacks. As part of our entry in the Cyber Security Awareness Week (CSAW) Embedded System Challenge hosted by NYU-Poly in 2011, we developed and presented 10 such processor-level hardware Trojans. These are split in five categories with various impacts, such as altering instruction memory, modifying the communication channel, stealing user information, changing interrupt handler location and RC-5 encryption algorithm checking of a medium complexity micro-processor (8051). Our work serves as a good starting point for researchers to develop Trojan detection and prevention methodologies on modern processor and to ensure trustworthiness of computing platforms.**

I. Introduction

The problem of maliciously intended modifications in manufactured integrated circuits (ICs) has recently emerged to be a main hardware security threat, mostly because of design outsourcing and migration of fabrication to low-cost areas across the globe. The booming of Intellectual Property (IP) exchange market causes further concerns since IP cores can be maliciously modified during various transaction stages before arriving at system integrators. Such modifications, known as hardware Trojans, introduce additional functionality that the designer, vendor, and customer are unaware of, and which may be exploited by the perpetrator after chip deployment. Depending on the field of application, the consequences of such attacks can range from minor inconvenience to major catastrophes, as a result of sabotaging a chip, or stealing sensitive data [1].

Many researchers have already proposed various hardware Trojan detection methods at both post-silicon and pre-silicon stages, e.g., side channel fingerprinting [2], [3], [4], [5], enhanced functional testing [6], code verification and functionality comparison logic [7], [8], etc. However, all of the previous work only focuses on small-scale ASIC designs while far less work has been performed to detect malicious modifications in processor-level designs to ensure trustworthiness of computing platforms [9], [10].

In order to protect processors and stimulate research in the field of processor-level hardware Trojan detection, we chose the 8051 micro-processor as the target platform trying to demonstrate that processor-level malicious modification can be exploited by software codes to cause harmful impact. Example malicious modifications are added in the processor, memory, or the communication channels in order to leak sensitive information, perform denial-of-service attacks or modify the processor's functionality. Figure 1 shows the block diagram of the 8051 micro-processor implemented on a Xilinx Spartan-6 FPGA board. Hardware description language (HDL) codes for the 8051 core, the UART module, testbenches and simulation models were provided by NYU-Poly as part of the CSAW Embedded System Challenge [11].

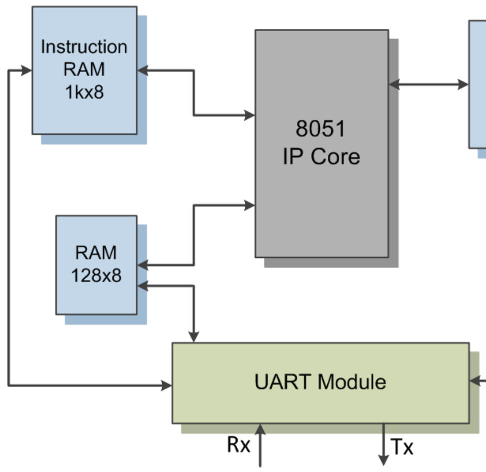

Fig. 1. Block diagram of 8051 micro-processor with UART channel

In the rest of the paper, ten processor-level hardware Trojan designs are introduced in five categories, including malicious instruction insertion, Trojan side channel, privacy hacking, malicious code execution and cryptographic algorithm tracking.

II. Instruction Monitoring and Modification

Different from ASIC design, processors and micro-processors contain both control logic and computation units and can perform general purpose operations. Here, the instruction set plays a key role to control the functionality of the processor. For security purposes, the integrity of the instruction memory where the program is stored is of highest priority for processor-level hardware Trojan prevention. We present three hardware Trojans targeting instruction memory to demonstrate that any malicious modifications in instruction memory can cause catastrophic impact to the processor. The area overhead and performance impact resulting from the insertion of hardware Trojans can be found in Table I.

Trojan I: Malicious instruction insertion

The first Trojan targets the address inconsistency between program loading and program operating. That is, the genuine 8051 micro-processor will always run the program from address 0x0000, disregarding the way programs are loaded. We show that only one line modification on the HDL code suffices to insert a `JUMP` instruction on top of any legitimate program so that attackers can put malicious instructions at a pre-defined location. The modified program generates correct results on a genuine processor but performs additional functions in Trojan-infected processors. It serves as a good example of how vulnerable hardware can be exploited by seemingly harmless software codes.

Trojan II: Program pattern checking

In cases where attackers cannot gain access to the program and considering the fact that not all programs are of high security level containing sensitive information, in order to avoid meaningless hacking, a Trojan performing on-chip program recognition will be helpful for attackers. Under this requirement, we designed a new type of hardware Trojan which can check the pattern of the program from simply detecting the `DPTR` address to more complicated RC-5 encryption algorithm matching. However, since we need to define the checking pattern beforehand, such Trojans are better used in systems of limited functionalities, e.g., encryption/decryption systems, secure communications, etc.

Trojan III: Instruction replacement

The type III hardware Trojan is another low overhead example modifying the instruction memory to cause malfunction of the system. Here we replace certain instructions with pre-defined malicious instructions to change the functionality of programs. This Trojan will be effective in the scenario where attackers can access the software program at the testing stage so that they can avoid using "contaminated" instructions which will be maliciously replaced during operation. After installation of the micro-processor, any programs containing "contaminated" instructions will cause erroneous results.

III. Trojan Side Channel (TSC)

Trojan side channel is a special hardware Trojan design which leaks internal sensitive information through malicious side channels. According to the availability of on-chip resources, attackers may either construct a new side channel (e.g. authors in [12] constructed a CDMA-like wireless channel to leak information with an additional LFSR; authors in [13] simply inserted ring oscillators to construct FM channels) or hide the Trojan side channel into legitimate communication channels (e.g. authors in [14] added a shadow RS232 channel with baud rate 19200 on top of the original RS232 channel with baud rate 9600). Trojan side channels have been proven to be a powerful attacking scheme for leaking internal sensitive information in a contactless way.

All previously proposed Trojan side channels are implemented in ASIC designs. The migration of these Trojans to processors is not straightforward because a modern processor will often have sensitive data protection schemes and users are restricted to access only some of the memory addresses. Inserting a Trojan side channel by invalidating the data protection scheme would result in a high Trojan overhead [15]. However, many of the on-board peripheral devices have the privilege to visit restricted memory addresses. We can then get around this problem by attacking those on-board devices to insert a Trojan channel on top of their normal functionality.

Trojan IV: Trojan side channel on AC'97 codec

Among all the hardware available on the FPGA board, we chose to hack the audio channel, a National Semiconductor LM4550 AC '97 audio codec, to hide our Trojan side channel [16]. An audio Trojan side channel is constructed, which leaks memory information by converting a '1' bit into a one second audio signal with half second beep and half second silence and a '0' bit into one second length silence. Necessary modifications are made on HDL codes for both the micro-processor and the audio driver module. The hidden AC '97 Trojan side channel can be used to attack most of the modern PC system with multimedia devices. The internal secret data is converted to a multimedia signal and is then leaked through the multimedia Trojan channel.

IV. Privacy Hacking

Most of the current computer systems including mission-critical systems dedicated for cryptographic purpose or data collection, do not contain sensitive information at the architecture level but rather rely on users' input to perform various operations. In order to protect user privacy, most of the current computer systems will encrypt the data from input devices such as the keyboard, mouse, touchscreen and only use the encrypted data when communicating with other systems, or they would simply discard the user inputs after user authentication. So for a general purpose processor, user input is the most critical information of the whole system and the most valuable information the attackers would like to collect, before it is encrypted or discarded.

Trojan V: User information leakage

Considering the fact that most of the user input is from the keyboard, the new type of hardware Trojan tries to hack the 8051 micro-processor by collecting all keyboard hits and then leaking user information. Whenever a key is released, an `F0` key-up code will be sent back to the micro-processor followed by the scan code of the released key. This key-up code will move the external RAM address bits to a pre-defined location, so-called "Trojan location" and store the ASCII key value in that address. This type of hardware Trojan can be applied to any computer system to steal user information since it gets around most of the hardware/software information protection methodologies by attacking the raw input directly.

V. Executing Malicious Code

Many of the modern attacks rely on attempts to execute unauthorized code. The most common practice in the past was buffer overflow attacks [17]. However, with the introduction of the No-Execute bit in modern micro-processors, buffer overflow attacks are extremely difficult to implement in the latest systems. Another type of attack, return-to-libc [18], was introduced recently and is the latest threat in terms of executing malicious code.

In this section, we present modifications in the micro-processor that allow the attacker to execute unauthorized code. These Trojans are generalizable to microprocessors.

Trojan VI: Interrupt handler location change

This Trojan involves the modification of the jump location of the interrupt service routines. Specifically, we modify the service location for External Interrupt 1 (IE1), which is originally located in location 0x13, to location 0x11. The result of this change is that whenever IE1 occurs, the micro-processor jumps to location 0x11 instead of location 0x13 (thus, 2 bytes before). The 2 bytes are enough to fit a SJMP or AJMP instruction; so, the programmer can place a jump instruction to malicious code.

Locations 0x0B-0x12 (8 bytes) are reserved for the Timer Interrupt 0 (TF0) interrupt handler. However, in most of the cases 8 bytes are not enough to handle an iterrupt. The most common practice is to place a jump instruction (2-3 bytes) in the address of the micro-processor defined interrupt service location. The remaining 4-5 bytes are filled with NOPs.

In order to achieve the routine location change, one line is modified in the VHDL of the micro-processor. This change incurs no overhead.

Trojan VII: LCALL $\rightarrow$ ACALL transformation

This Trojan converts an LCALL instruction to an ACALL instruction, whenever the Trojan is enabled. This enables a jump to another location, where malicious code is inserted. In order for the Trojan to be transparent, whenever the malicious code ends, the program counter returns to the original LCALL instruction, the Trojan is disabled and the originally requested routine is serviced.

Whenever an LCALL is substituted by an ACALL, the program jumps to a different address. LCALL is a 3-byte instruction, while ACALL requires 2-bytes:

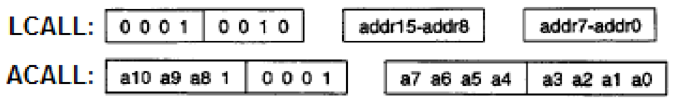

Fig. 2. LCALL and ACALL encoding [19]

Thus, whenever the LCALL is transformed to ACALL, the jump address is PC<15:11>000 addr15-addr8. For example, if LCALL 0444 at location 0x0403 is executed, then the jumping address would be 0x0004 (PC<15:11>= 00000, addr15-addr8 = 00000100 (0x04)). Since bits 10-8 are always 0, the micro-processor will jump within the 0x?000-0x?0FF or 0x?800-0x?8FF 256-byte segments. Thus, the program can align the code accordingly so that the malicious code starts at address 0x?0FF or 0x?8FF (end of 256-byte segment) and no code exists in the aforementioned 256-bytes segments (NOPs will be executed until the PC reaches the beginning of the malicious code).

Trojan VIII: Unused instruction executed as LCALL

Trojan VIII utilizes an unused operation code to jump to a malicious code location. Specifically, opcode '11110010' is unutilized and we can modify to decoder to interpret it as an LCALL (opcode '00010010'). Furthermore, during the malicious LCALL execution, the program counter is modified to point to location 0x03FD (last 3 bytes in memory), where the programmer can place a jump to the malicious code.

In order to implement the attack, the 'malicious' LCALL can replace a normal LCALL, jumping to a different address. Thus, the program size of the initial assembly is unchanged (unlike the case when an extra LCALL is added). The malicious and normal LCALL have different PC targets, depending on whether the Trojan is installed or not. Furthermore, the machine code has to be modified after the assembly, or during loading (the first three bits of the LCALL are set to 1).

Trojan IX: OISC

This Trojan is based on the concept of One Instruction Set Computing (OISC). OISC is an abstract machine that uses only one instruction, eliminating the need for a machine language opcode. With a judicious choice for the single instruction, OISC is capable of being a universal computer in the same manner as traditional computers that have multiple instructions [20]. In this example, we use `subleq` (Subtract and branch if less than or equal to zero).

OISC code is installed in memory, and executes there. Whenever the 8051 does not access the memory, an OISC cycle is executed by performing a memory operation. Nine cycles are needed to execute one `subleq` instruction (the memory contents of 3 operands need to be fetched). The process is completely transparent to the user, but the memory contents are being modified (and the OISC code should not be overwritten).

Since higher order instructions can be synthesized using `subleq`, the programmer can execute any desired program. There are several online tools for compiling C programs to `subleq` assembly.

VI. RC5 ATTACK

This section describes Trojans that are specific to the RC-5 encryption algorithm. Specifically, we try to exploit properties of the RC-5 algorithm and leak information that will help the attacker extract the parameters of the process.

Trojan X: RC5 rounds and rotations tracking

The Trojan X detects the RC-5 algorithm by tracking the number and type of arithmetic operations performed between registers. Specifically, every iteration of the RC-5 requires 2 4-byte additions, 2 4-byte XORs and 2 4-byte multiplications. This translates to 8 additions, 8 XORs and 4 multiplications (since operations are per byte, except for multiplication that produces a 16-bit result). These operations are always performed between the accumulator and the memory, thus can be easily distinguished from other arithmetic operations. Therefore, by tracking the number of operations, we can identify an RC-5 iteration. Furthermore, the Trojan also tracks the number of data-dependent rotations.

The Trojan is completely transparent to the user, is implemented purely in hardware, and places information about the number of iterations as well as the rotations to the last memory location. In order to minimize overhead, 4 bits are used for the number of rounds and 4 bits for the rotation count. Thus, after executing the RC-5 algorithm, the last location (0x3FF) of the external memory contains the value 0xXY, which means that num_rounds % 8 = Y and shift_count % 8 = X.

TABLE I
SUMMARY OF PROPOSED TROJAN DESIGNS AND THEIR OVERHEAD

Trojan type	Trojan Description	Slice Registers (Overhead)	Slice LUTs (Overhead)	Average Fanout
Genuine	N.A.	549	2661	5.42
Trojan 1	Malicious instruction insertion	548 (0%)	2666 (0%)	5.42
Trojan 2	Program pattern checking	554 ~ 568 (0.9% ~ 3.5%)	2646 ~ 2672 (-0.6% ~ 0.4%)	5.33 ~ 5.39
Trojan 3	Malicious instruction replacement	549 (0%)	2668 (0.2%)	5.41
Trojan 4	Trojan side channel (audio)	647 (17.9%)*	2614 (-0.2%)	5.11
Trojan 5	Private information collecting	616 (12.2%)*	2756 (3.6%)	5.23
Trojan 6	Interrupt handler location change	549 (0%)	2661 (0%)	5.41
Trojan 7	LCALL $\rightarrow$ ACALL transformation	546 (-0.55%)	2369 (-11.98%)	4.96
Trojan 8	Unused instruction executed as LCALL	551 (0.3%)	2711 (1.87%)	5.33
Trojan 9	OISC	603 (9.8%)	2718 (2.14%)	5.21
Trojan 10	RC5 rounds and rotations tracking	566 (3.09%)	2665 (0.01%)	5.32

* Note that the high overhead here includes an extra module to an drive AC '97 codec (or keyboard).
Modern computer systems often contain these drivers so this overhead may be lowered to less than 1%.

VII. Discussion and Conclusion

In this paper, we presented ten processor-level hardware Trojan designs. We demonstrated that malicious modifications in the micro-processor, with trivial overhead, can be exploited by software codes to cause catastrophic impact to the whole computing system. Further, by reusing on-chip computing resources, zero-overhead Trojans become possible which can easily evade previously proposed Trojan detection methods relying on side channel fingerprinting.

Our work raises the alarm that micro-processors are quite vulnerable to hardware Trojan attacks and similar attacks can be easily implemented on modern processors. There are clearly more points of vulnerability that can be exploited for malicious purposes in large-scale processors. This provides significant impetus to improve methods for detecting and combating hardware Trojans in modern computing platforms towards designing trusted computing basis for critical systems.

Acknowledgements

This work was partially supported by the National Science Foundation (NSF-1017719). The travel grant to participate in the Embedded Systems Challenge were provided part by The National Science Foundation (0958510,1059328), Army (W911NF-11-1-0470), Air Force Research Labs and Intel. The FPGA platforms for the contest were donated by Xilinx.

References

[1] R. Karri, J. Rajendran, K. Rosenfeld, and M. Tehranipoor, "Trustworthy hardware: Identifying and classifying hardware Trojans," *Computer*, vol. 43, no. 10, pp. 39–46, 2010.
[2] D. Agrawal, S. Baktir, D. Karakoyunlu, P. Rohatgi, and B. Sunar, "Trojan detection using IC fingerprinting," in *IEEE Symposium on Security and Privacy*, 2007, pp. 296–310.
[3] Y. Jin and Y. Makris, "Hardware Trojan detection using path delay fingerprint," in *IEEE International Workshop on Hardware-Oriented Security and Trust*, 2008, pp. 51–57.
[4] R. M. Rad, X. Wang, M. Tehranipoor, and J. Plusquellic, "Power supply signal calibration techniques for improving detection resolution to hardware Trojans," in *IEEE/ACM International Conference on Computer-Aided Design*, 2008, pp. 632–639.
[5] Y. Jin and Y. Makris, "Hardware Trojans in wireless cryptographic ICs," *IEEE Design and Test of Computers*, vol. 27, pp. 26–35, 2010.
[6] F. Wolff, C. Papachristou, S. Bhunia, and R. S. Chakraborty, "Towards Trojan-free trusted ICs: Problem analysis and detection scheme," in *IEEE Design Automation and Test in Europe*, 2008, pp. 1362–1365.
[7] S. Drzevitzky, U. Kastens, and M. Platzner, "Proof-carrying hardware: Towards runtime verification of reconfigurable modules," *International Conference on Reconfigurable Computing and FPGAs*, pp. 189–194, 2009.
[8] E. Love, Y. Jin, and Y. Makris, "Proof-carrying hardware intellectual property: A pathway to trusted module acquisition," *IEEE Transactions on Information Forensics and Security*, vol. 7, no. 1, pp. 25–40, 2012.
[9] Tucek J. Cozzie A. Grier C. Jiang W. Zhou Y. King, S.T., "Designing and implementing malicious hardware," in *Proceedings of the 1st USENIX Workshop on Large-Scale Exploits and Emergent Threats (LEET)*, 2008, pp. 1–8.
[10] M. Hicks, M. Finnicum, S. T. King, M. M. K. Martin, and J. M. Smith, "Overcoming an untrusted computing base: Detecting and removing malicious hardware automatically," in *Proceedings of IEEE Symposium on Security and Privacy*, 2010, pp. 159–172.
[11] *http://www.poly.edu/csaw2011/csaw-embedded.*
[12] L. Lin, M. Kasper, T. Guneysu, C. Paar, and W. Burleson, "Trojan side-channels: Lightweight hardware Trojans through side-channel engineering," in *Cryptographic Hardware and Embedded Systems*, vol. 5747 of *LNCS*, pp. 382–395. Springer-Verlag Berlin, 2009.
[13] *http://isis.poly.edu/csaw/embedded.*
[14] Y. Jin, N. Kupp, and Y. Makris, "Experiences in hardware Trojan design and implementation," in *IEEE International Workshop on Hardware-Oriented Security and Trust*, 2009, pp. 50–57.
[15] Y. Jin and Y. Makris, "Is single trojan detection scheme enough?," in *Proceedings of the IEEE International Conference on Computer Design (ICCD)*, 2011, pp. 305–308.
[16] *www.digilentinc.com/atlys/.*
[17] I. Gerg, "An overview and example of buffer overflow," *Information Assurance Technology Professionals Newsletter*, vol. 7, no. 4, pp. 110–116, 2005.
[18] H. Shacham, "The geometry of innocent flesh on the bone: Return-into-libc without function calls (on the x86)," in *Proceedings of the 14th ACM conference on Computer and communications security*. ACM, 2007, pp. 552–561.
[19] Intel Corporation, "Intel mcs51 family user manual," 1981.
[20] F. Mavaddat and B. Parhami, "Urisc: the ultimate reduced instruction set computer," *International Journal of Electrical Engineering Education*, vol. 25, pp. 327–34, 1988.