Extended Abstract: Circuit CAD Tools as a Security Threat

Jarrod A. Roy†, Farinaz Koushanfar‡ and Igor L. Markov†
†The University of Michigan, Department of EECS, 2260 Hayward Ave., Ann Arbor, MI 48109
‡Rice University, ECE and CS Departments, 6100 South Main, Houston, TX 77005

The demand for trusted and tamper-resistant computing platforms has placed security at the leading edge of research and industrial practice. Reported hardware-security breaches have already led to loss of confidential information, identity theft, intercepted cellular communications, and IP burglary. Our work demonstrates that ICs can be easily compromised by tampering with CAD tools or scripts that run these tools, suggesting that developing effective countermeasures against such attacks is a major research challenge. Our work is especially relevant to industrial uses of open-source EDA.

Introduction. Breaches of hardware security facilitate numerous attacks on electronic systems that imperil important and ubiquitous applications. For example, in the *Greek cell-phone tapping case* of 2004-2005 [6], a still-unknown entity tapped personal cell phones of many top Greek government officials. In another system break-in, a New York City resident installed software key loggers at public computers at Kinko's to steal credit card numbers [5]. The key loggers went undetected for nearly two years, but eventually the crook was caught red-handed and pleaded guilty to computer fraud. Insertion and avoidance of such software-based *Trojan horses* has been an active area of research for many years [8]. While software exploits can be often caught by anti-virus programs and can be purged by reinstalling the OS, hardware exploits are more difficult to detect and remove. Recent work, in collaboration with IBM, focuses on detecting Trojan horses inserted by manufacturers into IC masks [1].

A major concern raised by our work is that similar key loggers and Trojan horses can be *planted in integrated circuits during the design flow* and remain undetectable. They may be activated by secret combinations through the network without risk to the perpetrator, or through physical contact between the adversary and smartcards, RFIDs, e-cash, etc.

We demonstrate how Trojan horses can be planted into ICs by maliciously altered logic CAD tools and simple pre-/postprocessing scripts. Concerns about sabotaged tools become more relevant as the industry embraces open-source EDA initiatives. As pointed out by Ken Thompson [8], open-source software can make it easy to alter any given tool (using its original source code) and surreptitiously replace the binary used at a particular site. We outline several variants of such attacks and discuss countermeasures, mostly short-term patches, with the purpose of articulating this arduous challenge to the research community.

To better blend into the host system, Trojan horses must be much smaller in size and power profile than genuine components. This can be achieved by tailoring exploits to a specific host system, as was done in most documented cases. If the host system is an IC, then a *trigger* can be implemented as a digital logic circuit, memory can be implemented using sequential logic, and the payload would depend on the application. In particular, some parts of the host system can be altered using multiplexer gates, or turned off (gated).

Avoiding Detection. Trojan horses in ICs may be viewed as design errors that alter observable behavior, and at the first sight appear detectable by standard design verification and test procedures. However, such detection is surprisingly easy to avoid by using a hidden binary counter that would keep the Trojan horse inactive for long enough to survive manufacturing test. A threshold of several hours to several weeks is sufficient to fool even extremely rigorous testing procedures. The same technique would prevent many sequential verification techniques, such as Bounded Model Checking (BMC) and even simulation, from catching altered behavior. In particular, BMC is limited to a small number of clock cycles, and simulation is orders of magnitude slower than the actual IC.

To prevent accidental triggering of a Trojan horse in a working system, an additional secret combination can be used that is unlikely to occur in normal circumstances. Without such a combination, the device will operate normally and no evidence of the Trojan horse will be present in the IC's outputs.

One may think that tools and methods that examine the circuit might detect exploits. And yet a Trojan horse may be unobservable to design for test (DFT) constructs such as scan chains. A better alternative is to inject the Trojan horse into the design before ATPG, so that the generated input patterns test the Trojan horse for faults as well. Carefully checking the generated patterns against the intended function of the IC after manufacturing might reveal the Trojan, but this is usually impractical for large circuits, and test compression makes it impossible to analyze expected responses.

Injecting Trojan Horses. We point out that Trojan horses can be injected at nearly any point in the design flow from logic synthesis to design for test (DFT) — either (1) by the CAD tools themselves, or (2) by the scripts that run them. The latter technique can be used with commercial tools, but cannot invoke the primitives available during optimization.[1] Scripts can plant Trojan horses during preprocessing — by editing Verilog — or during postprocessing — by altering the gate-

level netlist. Preprocessing appears easier because it primarily involves editing ASCII text, and allows the injected Trojans to better blend into the synthesized netlist and circuit technology. In all cases, downstream back-end optimizations will treat the Trojan horse as a regular circuit module, making it particularly difficult to detect by inspection.

To inject a Trojan horse, one first detects a target circuit such as a particular cryptography or communication primitive that can be compromised. Detection can be accomplished

- by pattern-matching in Verilog (discussed below),
- by matching unusual sub-circuits, such as bit permutations between registers,
- by combinational equivalence checking.[1]

An identified pattern or subcircuit can then be replaced with a compromised copy. We illustrate these concepts for standard cryptography circuits, which are among the hardest to compromise.

Case Study: Crypto Circuits. Cryptographic circuits use numerous XOR gates, bit shifts, bit permutations, and distinctive, well-known constants. These are easy to detect and alter, compromising the original functionality.

- "Magic" numbers used in standard pseudo-random number generators (PRNGs), cryptographic hashes and ciphers are easy targets for alteration. PRNGs are of particular interest to security because randomization is used to defeat many types of attacks, such as Differential Power Analysis [7]. Depriving a PRNG of randomness would re-enable those attacks [2]. For example, a linear congruential PRNG relies on the fact that its constants are relatively prime to produce a pseudo-random sequence with a long period. Changing one digit can reduce this period by 10x, while zeroing out all digits removes randomness entirely.
- Bit permutations are used by many cryptographic ciphers to rearrange bits in keys and ciphertext. The Data Encryption Standard (DES) utilizes at least six distinctive 32-bit permutations, which make up a significant portion of DES implementations. Such gateless subcircuits, connecting pairs of 32-bit registers, are easy to detect, but critical to repealing known attacks (as of Fall 2007, triple-DES is considered cryptographically secure). Once a malicious tool identifies DES, it can compromise permutation circuits and hide the exploit by conditioning it on the state of a trigger, using a small number of MUX gates.
- Substitution functions take n bits as input and return n bits as output so that no two input combinations map to the same output combination (bijection). Such functions are used by may ciphers and are called S-boxes in the Advanced Encryption Standard (AES). 8-bit S-boxes used by AES (one each for encryption and decryption) perform well-known operations over the field GF(2^8). They are often implemented as look-up tables with "magic" numbers, but can also be implemented strictly with bit manipulations, as illustrated by the IWLS 2005 OpenCores benchmarks `aes_core` and `systemcaes`. An implementation of S-boxes can be detected using combinational equivalence checking, although for 8-bit circuits, bit-parallel exhaustive enumeration will do. Once detected, S-boxes in AES can be compromised by using published fault-injection methods [3], [4].

[1]Synthesis tools often use ROBDD packages which can check equivalence of small circuits. SAT-based equivalence checking with the open-source MiniSAT tool requires only 1000 lines in C.

Countermeasures & Summary. As we have demonstrated, Trojan horses can be easily injected into IC designs during the design flow. By remaining inactive for a long time, they become practically undetectable. Therefore, we suggest countermeasures based on the *dynamic verification* paradigm from [9], with the key insight that checking the results of a computation can be easier than performing the computation. The work in [9] shows that the results produced by a microprocessor can be checked at full speed by a small module. When a discrepancy is detected, the main processor falls back on a slow but surely-correct implementation, thus tolerating rare miscalculations. We adapt dynamic verification to harden IC designs against Trojan horse injection by checking vital statistics of the primary IC, e.g., sufficient randomness of PRNGs. When an anomaly is detected, the main IC should be shut down until further investigation. While this technique does not prevent sabotage, it may prevent subtle eavesdropping and leakage of keys. To enhance security, the checkers must be synthesized using different tools, or be simple enough not to be designed by hand.

In conclusion, we realize that our techniques may only slow down, but not stop a resourceful attacker. Thus the described research challenge remains open.

Acknowledgments. Prof. Koushanfar's work is partially supported by the DARPA/MTO Trust in Integrated Circuits/Young Faculty Award (YFA) under grant award W911NF-07-1-0198, and by the NSF ITR-CYBERTRUST under grant number 0716674. Prof. Markov's work is partially supported by the Microsoft Breakthrough Research Award.

REFERENCES

[1] D. Agrawal, S. Baktir, D. Karakoyunlu, P. Rohatgi and B. Sunar, "Trojan Detection using IC Fingerprinting," *IEEE Symp. on Security & Privacy'07*, pp. 296–310.

[2] L. Dorrendorf et al., "Cryptanalysis of the Random Number Generator of the Windows Operating System," *ACM CCS'07*.

[3] O. Faurax and T. Muntean, "Security Analysis and Fault Injection Experiment on AES," *SARSSI'07*.

[4] M. Karpovsky, K. J. Kulikowski and A. Taubin, "Robust Protection Against Fault-injection Attacks on Smart Cards Implementing the Advanced Encryption Standard," *DSN'04*, pp. 93–101.

[5] K. Poulsen, "Guilty Plea in Kinko's Keystroke Caper," SecurityFocus, July 18, 2003, `http://www.securityfocus.com/news/6447`

[6] V. Prevelakis and D. Spinellis, "The Athens Affair", *IEEE Spectrum*, vol. 44, no. 7, pp. 26-33, July 2007.

[7] F.-X. Standaert, S. B. Ors and B. Preneel, "Power Analysis of an FPGA - Implementation of Rijndael: Is Pipelining a DPA Countermeasure?", LNCS 3156, pp. 30–44, 2004.

[8] K. Thompson, "Reflections on trusting trust", *Comm. of ACM*, vol. 27, no. 8, pp. 761-763, Aug. 1984.

[9] C. Weaver and T. Austin, "A Fault Tolerant Approach to Microprocessor Design," *DSN'01*, pp. 411-420.