УДК 004.056

SECURE INFORMATION FLOW ANALYSIS FOR HARDWARE DESIGN VERILOG AND VHDL LANGUAGE

Lahno V.A.

АНАЛИЗ ИНФОРМАЦИОННОЙ БЕЗОПАСНОСТИ АППАРАТНОГО ОБЕСПЕЧЕНИЯ НА БАЗЕ ИСПОЛЬЗОВАНИЯ ЯЗЫКОВ VERILOG И VHDL

Лахно В.А.

The paper analyzes possibility of hardware description languages VHDL and Verilog to create various components of information security. This can be used to ensure private keys are never leaked (for secrecy), and that untrusted information will not be used in the making of critical decisions (for safety and fault tolerance).

Keywords: *Information security, Information security, Information Flow Analysis, Hardware Security, Language-Based Automated Verification, firewall*.

Introduction. The influence of information automation systems pervades many aspects of everyday life in most parts of the world. In the shape of factory and process control systems they enable high productivity in industrial production, transport systems they provide the backbone of technical civilization. One of the foremost transport businesses security concerns is the protection of critical information, both within their internal financial infrastructures and from external elements. Up to now, most of these systems are isolated, but for the last couple of years, due to market pressures and novel technology capabilities, a new trend has been rising to interconnect automation systems to achieve faster reaction times. Initially, such interconnections were based on obscure, specialized, and proprietary communication means and protocols. Now more and more open and standardized Internet technologies are used for that purpose. Studies show that most cyber-attacks occur inside organizations, instigated by personnel with valid access to the system. This work describes the design, implementation, and testing of a security system that enhances the capability of transport businesses to protect information within the boundary of their networks [1, 2].

Hardware designers need to precisely analyze high-level descriptions for illegal information flows. Language-based information flow analyses can be applied to hardware description languages, but a straight-forward application either conservatively rules out many secure hardware designs, or constrains the designers to work at impractically low levels of abstraction. We demonstrate that choosing the right level of abstraction for the analysis, by working on Finite State Machines instead of the hardware code, allows both precise information flow analysis and high-level programmability.

VHDL has been at the heart of electronic design productivity since initial ratification by the IEEE in 1987. For 16 years the electronic design automation industry has expanded the use of VHDL from initial concept of design documentation to design implementation and functional verification. The use of VHDL has evolved and its importance increased as semiconductor devices dimensions have shrunk. Not more than 10 years ago it was common to mix designs described with schematics in favor of the hardware description language only. The use of VHDL has evolved and its importance increased as semiconductor devices dimensions have shrunk. Not more than 10 years ago it was common to mix designs described with schematics in favor of the hardware description language only.

In 1986, VHDL was proposed as an IEEE standard, It went through a number of revisions and changes until it was adopted as the IEEE 1076 standard in December 1987. The IEEE 1076-1987 standard VHDL is the VHDL used in this book. All the examples have been described in IEEE 1076 VHDL, and compiled and simulated with the VHDL simulation environment from Model Technology Inc. The synthesis examples were synthesized with the Exemplar Logic Inc. synthesis tools.

Verilog was started initially as a proprietary hardware modeling language by Gateway Design Automation Inc. around 1984. Verilog simulator was first used beginning in 1985 and was extended substantially through 1987. The standard, which combined both the Verilog language syntax and the PLI in a single volume, was passed in May 1995 and now known as IEEE Std. 1364-1995. After many years, new features have been added to Verilog, and the new version is called Verilog 2001. This version seems to have fixed a lot of problems that Verilog 1995 had. This version is called 1364-2001.

The main part. Hardware Information Security. We use the Network Site Controller as an example hardware design and Verilog as an example hardware description language. Network Site Controller provides a centralized repository for all information pertaining to network sites of Transport Company. It manages all operational Service Agreements, such as Leases, Licenses, Utilities and Transmission, to simplify operational planning and management.

We explore a new direction where we base the information flow analysis in hardware design, namely Finite State Machines. State machines are widely recognized as a natural way to describe hardware controllers, and most commercial Computer-aided design (CAD) tools can extract state machines from Verilog or VHDL programs automatically.

A state machine can be expressed as a set of states and transitions among those states triggered by signals which are either inputs to the state machine or local data. The most natural way to implement a state machine using programming languages is to have a variable *cur_state* to store the current state, and case-style statements to decide state transitions based on *cur_state* and some other conditions.

Figure 1 shows how conventional information analysis is applied to such state machines implementations.

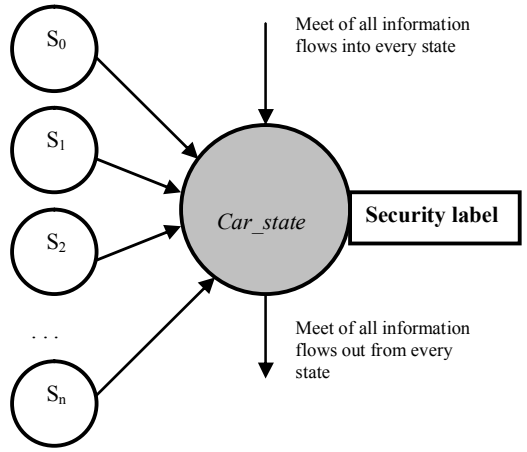

Fig. 1. Existing program analysis on state machines

The value of *cur_state* can be one of $S_0, S_1, S_2, \ldots, S_n$, indicating the current state. When information flows are analyzed, *cur_state* is associated with a single security label. Such analysis does not take into consideration the fact that information flows are actually flowing through each individual state, hence there is no way to track the security labels of individual states when states are represented only as different values of the variable *cur_state*, making the analysis conservative.

The key insight of our approach is that by analyzing hardware descriptions explicitly as state machines (i.e., as a reified set of individual states with accompanying transitions) rather than as an implicit state machine encoded using variables, the analysis can precisely track security labels for individual states.

For example, execution leases are an architectural mechanism that enables trusted code to grant access to a limited amount of machine resources to untrusted code for a fixed amount of time. One can imagine a lease to be a sandbox of space and time which untrusted components can never go beyond. A lease starts by setting up a timer and transferring the control to untrusted components. After the timer expires, the control will be automatically transferred back to the trusted system. We assume a two label security lattice with security labels High and Low, where High indicates secret or untrusted, and Low represents unclassified or trusted. In such security lattice, information is allowed to flow from any data with a Low security label to data marked as High, while the other direction is illegal. As can be seen from the figure, the inputs to the lease controller include the timer_low which is a trusted value and data_high which is an untrusted value. The output is generated by state S_0. The lease mechanism works as follows: Some trusted component (master state S_0) initiates a lease to some untrusted cloud (simplified as two states in this example) by specifying a timer_low boundary.

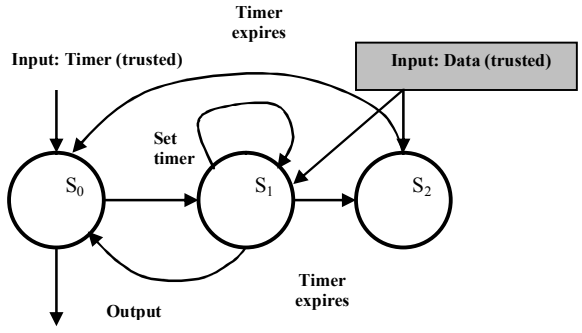

Fig. 2. State machine diagram of the lease controller

The control transfers to slave state S_1 which is inside the untrusted cloud, and either stays at S_1 or jumps to S_2 based on some untrusted data_high and loops inside the cloud until the timer_low expires. When the timer low expires the control automatically

transfers back to the master state no matter what the current state is. The corresponding Verilog program is shown in Figure 1.

Research results. The value of the state variable *cur_state* can be either 0 (master state) or 1, 2, 3 . . . (in the untrusted cloud). In master state S_0, if the *timer_low* is activated, the state will transfer to S_1 by assigning *cur_state* to 1. In slave state S_1, if the timer expires, the state will transfer back to S_0, otherwise *data_high* is processed and *timer_low* will be decremented.

The corresponding program Verilog is shown below.

```
module SM (...);
input ...;
output ....;
wire ....;
....
// Indicating the state machine will execute
// repeatedly
always @ *
begin
case (cur_state)

//Master State (trusted)
0:
if(timer) begin
//Jump to slave state
next_state <= 1;

//Trusted Timer
next_timer <= timer;
end
else begin
next_state <= 0;
next_timer <= 0;

//Generate Output
output = ...
end
//Untrusted Cloud (untrusted):
1:
if (timer == 0) begin

//Jump back to master state
next_state <= 0;
end
else begin

//Do something with untrusted data
//Untrusted Data

if (data) begin
//Keep jumping inside untrusted cloud
next_state <= 2;
end

next_timer <= timer - 1;
end
2: ...
3: ...
end
```

More complex state machine, such as a firewall, could not immediately be described in the language of Verilog or VHDL.

A firewall is a dedicated appliance, or software running on a computer, which inspects network traffic passing through it, and denies or permits passage based on a set of rules. Its basic function is to regulate the flow of traffic between computer networks of different trust levels.

Conventional firewalls operate at the network layer and their operation is based on stateful or non-stateful type. The later type has packet-filtering capabilities however; it is unable to make more complex decisions as regards to the stage or level of communications between the hosts. This leads to less security and functioning more like a router from the packet filtering point of view. Conventional firewalls working on the principle of stateful analysis poses a typical tradeoff of security Vs latency. More tightly the security implementations lead to increased latency causing jamming and congestion over the network. As mentioned above, building custom silicon in FPGAs leads to significant advantages such as rapid design cycle, early time-to-market, easy transition to structured ASICs and reduced non recurring cost of engineering (NRE) costs. Although the designers are choosing FPGA as a prototyping element, it is observed that the target FPGA is just treated as a black box. This makes the system designer to miss many opportunities to optimize the design to fit within the FPGA [3-5].

There are three essential components of the behavioral synthesis model first the inputs or stimulus to the system, second a module library and third the spatial and temporal constraints.

The behavioral specification is generally written in a high level general purpose language like C++ or in a Hardware Description Language like VHDL or Verilog.

The second component of the behavioral model i.e. module library consists of storage units like registers, memories or FIFOs, execution units like adders and multipliers, and interconnect units like multiplexers and buses. Overall, the behavioral model provides valuable information like the constraints such as area, clock speed, power and the data dependcy or the temporal model of execution. The behavioral synthesis is a process of constructing the register transfer model from its behavioral counterpart by adopting a process called as binding.

Conclusion. This paper presents the possibility of using modern hardware description languages for the design of the system hardware information security. It is shown that the use of languages VHDL, Verilog can significantly reduce the development time of the individual components of information security systems and improve the efficiency of project.

References

1. M. Tiwari. A hardware-supported mechanism for enforcing strong non-interference. / M. Tiwari, X. Li, H. Wassel, F. Chong, and T. Sherwood. // In Proceedings of the International Symposium on Microarchitecture (MICRO), 2009.
2. James Newsome. Dynamic taint analysis for automatic detection, analysis, and signature generation of exploits on commodity software. / James Newsome, Dawn Song.

// In 12th Annual Network and Distributed System Security Symposium, 2005.
3. M. Dalton. A Flexible Information Flow Architecture for Software Security. / M. Dalton, H. Kannan, C. Kozyrakis. // In Proceedings of the 34th annual international symposium on Computer architecture, June 2007.
4. D. E. Denning. Certification of programs for secure information flow. /D. E. Denning, P. J. Denning. // Communications of the ACM, 20 (7): 504 – 513, 1977.
5. G.E.Suh. Secure program execution via dynamic information flow tracking. / G.E.Suh, J.W.Lee, D.Zhang, and S.Devadas. // In Proceedings of the 11-th international conference on Architectural support for programming languages and operating systems, 2004.

Лахно В.А. Анализ информационной безопасности аппаратного обеспечения на базе использования языков Verilog и VHDL.

В статье анализируется возможность *использования* языков описания аппаратуры VHDL и Verilog для проектирования различных цифровых компонентов *систем защиты информации*.

Рассмотрены примеры реализации конечных автоматов применяемых в аппаратной части средств защиты информации.

Ключевые слова: *Информационная безопасность, защита информации, анализ информационных потоков, языка Verilog и VHDL.*

Лахно В.А. Аналіз інформаційної безпеки апаратного забезпечення на базі використання мов Verilog і VHDL.

У статті аналізується можливість використання мов опису апаратури VHDL і Verilog для проектування різних цифрових компонентів систем захисту інформації.

Розглянуто приклади реалізації кінцевих автоматів застосовуваних в апаратній частині засобів захисту інформації.

Ключові слова: *Інформаційна безпека, захист інформації, аналіз інформаційних потоків, мови Verilog та VHDL.*

Лахно Валерій Анатолійович – к.т.н, доцент кафедри економічної кібернетики, ss21@meta.ua.

Рецензент**: Леві Л.І.,** доктор технічних наук, професор Луганського національного аграрного університету.

Стаття подана 30.03.2013