Interactive Runtime Monitoring of Information Flow Policies

Mohamed Sarrab

Software Technology Research Laboratory
De Montfort University
The Gateway
Leicester, UK LE1 9BH
Email: msarrab@dmu.ac.uk

Helge Janicke, Antonio Cau

Software Technology Research Laboratory
De Montfort University
The Gateway
Leicester, UK LE1 9BH
Email: (heljanic, acau)@dmu.ac.uk

Abstract

Computer systems are verified to check the correctness or validated to check the performance of the software system with respect to specific properties. Recently, the verification of security properties during run-time received increased attention from researchers. In particular security properties that relate to information that is made available by the end users of the software is achievable only to a limited degree using static verification techniques. The more sensitive the information, such as credit card data, government intelligence or personal medical information being processed by software, the more important it is to ensure the confidentiality of this information. In this paper we present a framework that provides a flexible verification approach to information security management so that the information flow within a program execution conforms to a defined set of rules at runtime. The approach concentrates on providing a dynamic and adaptable information security solution by interacting with the user while the system is running in response to information flow events

Keywords

Run-time verification, Usable Security, Information Flow, Interval Temporal Logic, Ana Tempura.

I. INTRODUCTION

Recently, the problem of security properties (confidentiality, integrity and availability) verification during run time received increased attention from the researchers [3,10,23]. Traditional static verification techniques [1,15,21] have only taken into account the software system and properties that are available at design time; however security properties are often expressed with respect to a variety of users and the concrete information that is being processed by the system. The identification of users and their responsibilities as well as the concrete data are not available until runtime and can only be insufficiently be taken into account using static verification techniques.

Static verification is process to verifying the conformance of a software system and its specification without executing the code [1,2] and can be used to detect vulnerabilities that are introduced during the development of the system, such as buffer overflows [22]. Static information flow analysis determines whether a given system implements confidentiality requirements correctly [4].

Assuming that some sensitive information is stored on a computer system, how can we prevent it from being leaked? The first approach that comes to mind is to limit access to the information, either by using access control mechanisms such as encryption or firewalls. These very useful approaches have their limitations. Standard security mechanisms are focused only on controlling the release of information but no restrictions are placed on the propagation of that information and thus are unsatisfactory for protecting confidential information.

Suppose that a program requires a piece of confidential information; Can we make sure that the information is not somehow being leaked? Simply trusting the program is dangerous as we cannot always trust its provider. A better approach is to execute the program in a safe environment and monitor its behaviour to prevent confidential information from flowing to untrusted entities.

Information flow occurs from source object to target object whenever information read from a source it is potentially propagated directly or indirectly to one or more target objects. There are two types of information flow: Firstly, direct information flow which means that information flow between two objects directly without any mediation between them to exchange, read, write or execute information. Secondly, indirect information flow which requires mediation between two objects [4,5].

Our approach provides a runtime monitoring technique that

checks and controls information flow more flexibly through the use of information flow policies, providing more usable security approach that is independent of the concrete software at hand. The approach concentrates on providing a dynamic security policy that can change over time based on the user's preferences and the changing context in which the monitored application is executing [12]. The ability for users to modify the information flow policy during runtime is a central objective of our framework, which allows for the alteration of the program behavior in the case that a potential leak of confidential information is detected by the monitor according to the user's decision. The key novelty of our research is that it allows for the controlling of information flow whilst maintaining the goals of the usable security paradigm.

The motivation for this work is that on the most of the previous work the information flow policy is not in the hand of the end user. Security requirements often depend on the users because what one user considers as secure others may consider as insecure. Usable security that enables users to manage their systems security without defining elaborate security rules before starting the application. Most research on information flow is based on a fixed policy; whereas, our approach supports configurable policies which can be modified according to the user decision.

II. RELATED WORK

The static verification involves the analysis of source text by humans or software which can help to discovers errors early in the software process [3]. Security requirements in information systems change more frequently than functional requirements especially when new users or new data is added to the system. Runtime verification [6,7,8] has been used to increase the confidence that the system implementation is correct by making sure it conforms to its specification at runtime. Similar to [10] we employ runtime verification for information flow to determine whether a flow in a given program run violates a security policy. The problem is to prevent sensitive information such as credit card data, government intelligence or personal medical information, from leaking through the computation into untrusted data sinks. The specifications of what information flows are considered dangerous are set out in the security policy which represents a formal description of the user's security requirements. To manage change in security requirements and context of security mechanisms our approach concentrates on providing a dynamic and adaptable information security solution by interacting with the user while the system is running in response to information flow events. Despite a long history and a large amount of research on information flow control [1,14,15,16,17], there seems to be very little research done on dynamic information flow analysis and enforcing information flow based policies. One of the foundational work in this area is a lattice model of information flow which was presented in [13] by Denning. Considering this model Denning and Denning [14] provide static certification mechanism for verifying the secure flow of information through a program. Other approaches, eg. JFlow [2] and FlowCaml [17] use typing systems, checking information flow statically and work as source to source translator. Dynamic information flow analysis [18,19,20,22] has been less developed than static analysis. Dynamic analysis began very earlier in the 1970s by Bell and LaPadula aimed to deal with confidentiality of military information [21]. In their model they annotate each element of data with a security label and dynamically controlling information flow with two security properties. The simple security property 'no read up' which means any process can read from higher level security. The star property 'no writes down' which means that the process does not allow writing data to lower security level. Fenton [18] motivated research on dynamic analysis on a code level by his abstract data mark machine which dynamically checks information flow where each variable and program counter is tagged with a security label. Brown and Knight [19] provide a set of hardware mechanisms to ensure secure information flow. Lam and Chiueh [20] proposed a framework for dynamic taint analysis for C programs. Birznieks [22] provides an execution mode called Perl Taint Mode which is a special mode in the Perl script language where data are tagged with taint or untainted security level in order to detect and prevent the execution of bad commands. Vachharajani, Matthew, Bridges, Jonathan, Ram, Guilherme, Blome, Reis, Vachharajani and David. [23] proposed a framework for user centric information flow security at a binary code level, in this mechanism every storage location associated a security level. Cavadini and Cheda [10] presented two information flow monitoring techniques that use dynamic dependence graphs to track information flow during run-time. Previous work did not take in consideration the ability for users to modify the flow policy at runtime and the security requirements to be dependent on the requirement of individual users and their interaction with the monitoring system.

III. FRAMEWORK FOR INFORMATION FLOW CONTROL

Information flow occurs from source objects to target objects; whenever information is read from a source it is potentially propagated to the target object. There are two types of information flow: Direct information flow is defined as the operation that generates a flow between a source and a target independent of any other objects; indirect information flow means that there is an operation generating a flow

from a source to a target and the operation is dependent on the value of other objects.

Information flow control: Suppose that we have a classification of military security levels (top-secret ¿ secret ¿ confidential ¿ unclassified). Controlling information flow is concerned with preventing for example a subject (user or process) running at top-secret level from writing to objects (variables or files) at a lower security level (secret, confidential, unclassified). More generally the classification forms a lattice and information can only flow from sources to targets, if the targets' security label dominates the one of the source with respect to the lattice structure.

In a policy based system there is no fixed lattice structure and policy rules determine whether a flow is legal or not. A system controlling information flow using dynamic policies supports the evolution and change of policy rules while the system is executing. Controlling information flow with the user interaction allows the user to modify either program behaviour (by disallowing certain flows) or the policy (by dynamically modifying it) based on his current objective.

Information flow analysis vs. Runtime monitoring: Static information flow analysis verifies whether programs can leak secret information or not without running the program. Static analysis checks all possible execution paths including rarely executed ones. The advantage is that any program that has been successfully checked is indeed safe to execute as it cannot possibly leak information. The disadvantage is that any change in the underlying information flow policy means that the whole program needs to be analysed again. Another disadvantage is that a given program may be capable of leaking information, but in the way that it is executed by the user such leaks do not occur. Using static verification this program would be regarded unsafe.

Dynamic information flow analysis is concerned with monitoring and regulating a program execution at run time. It is potentially more precise than static analysis because it does only require that the current execution path does not leak information and can also handle language features such as pointers, arrays and exceptions easier that static analysis. Finally, using runtime monitoring of information flow it is possible to allow for user interaction that can influence the further execution of the program.

In static program analysis all possible paths of the program execution must be free of invalid flows. If any invalid information flow is detected then the static analysis mechanism will reject the whole program as insecure. Graphically we can depict the set of all possible program behavior by a blank circle and the set of all insecure program behaviour (defined in the policy) by the dotted circle. In these terms a program is rejected by static analysis if the intersection of both is not empty. In "Fig. 1" we depict the case for dynamic information flow analysis. Consider that a program is in a state 0 and performs an operation α that causes an

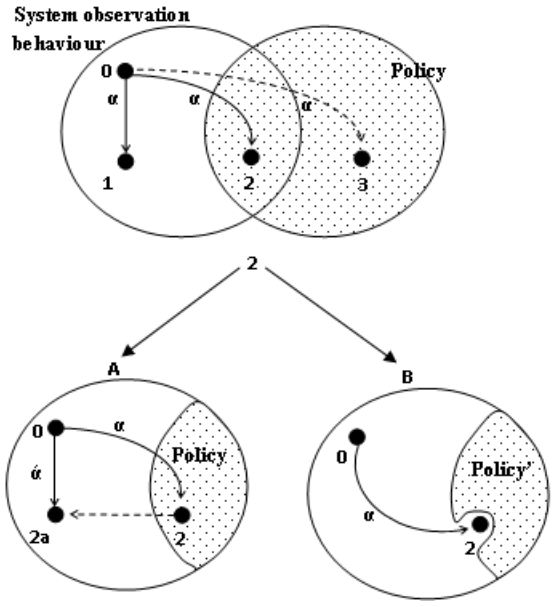

Figure 1: Runtime monitoring

information flow. We can distinguish two cases:
1) After the execution of α the program is in a secure state.
2) After the execution of α the program is in an insecure state.

The hypothetical third case, that the program exhibits a behavior that is defined by the policy as insecure, but is outside of the set of possible behaviours, can be ignored. In our framework we check whether the program is about to enter an insecure state by intercepting the operation α. In case 1, that α leads to another secure state the program will be allowed to perform α. In case 2, the runtime monitoring mechanism will send feedback to the user asking about the violation of information flow. The user has two options on how to proceed: (A) he changes the operation α to another operation $\acute{}\alpha$ in such a way that the resulting state is secure with respect to the policy. Such changes can for example be the termination of the program or (manually) sanitizing the information that flows in α. The other option (B) is to modify the Policy into a $\acute{\text{P}}$olicy for which α leads to a secure state. This could for example be introducing a one-off exception for the current flow or defining a separate context in which the information flow is considered legal.

IV. FRAMEWORK

Our approach is based on the observation of information flow during application run time.

The user feedback component handles all interactions with the system and the user. It runs in a separate thread of

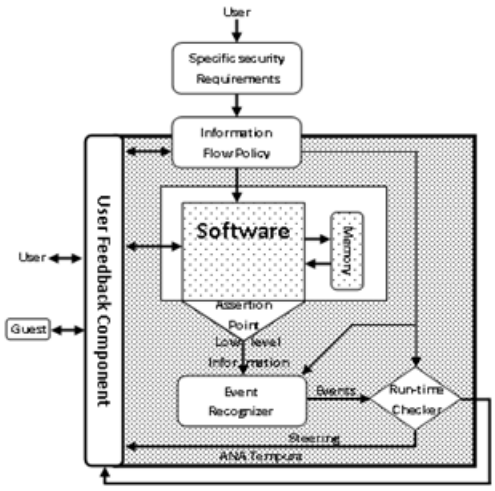

Figure 2: Runtime verification of information flow

control so that user interaction can be overlapped with information flow control. The user feedback component also allows the user to administrate the policy. When the software is running, the user feedback component receives feedback from the runtime checker (Steering). If the software is about to enter an insecure state then the user will be asked to determine whether the information flow should be aborted or allowed to flow and continue under a modified policy as illustrated in "Fig. 1". For example given a policy that states that user Bob's password must not be shared with any other user (Alice and Eve). If Bob now wants to give his password to Alice to perform some function on his behalf, our approach will detect this violation of information flow and ask the user how to proceed. Bob can then choose to stop the operation that would violate the flow (i.e. Alice does not obtain the password) or he allows this flow and changes the policy accordingly to reflect this decision. Moreover, this change can be temporary (a one of exception) or permanent (He can pass the password again to Alice).

- **Specific Security Requirements**. Stakeholders normally have a number of concerns that come with their desired system and are typically high-level strategic goals. In this component the stakeholders specifies the desired characteristics that a system or subsystem must possess with respect to sensitive information flow. In our previous example, this is the requirement that Bob's password must not be shared with any other user (Alice and Eve).

- **Information Flow Policy** is a security policy that defines the authorized paths, which can be a set of laws, rules, and practices that regulate how information must flow to prevent leak of information. In our framework, the information flow policy expresses the security requirements as specified by the stakeholder/user to a set of rules that are understandable by our monitoring mechanism. An information flow policy describes the events at the requirement level and is used to generate the assertion points and event recognizer. Information flow policy can make use of a complete lattice of security classes, where information is allowed to flow from objects of a specific security class, to objects of higher security classes [13], but also state more context dependent flows, such as Bob's password can only flow to Alice if Bob explicitly agrees" and can be paired with obligations such as Bob must change his password within 2 hours after passing it to Alice.

- **Assertion points** are program fragments as a collection of probes that will be inserted into the software. The essential functionality of the assertion point is to send pertinent state information to the event recognizer. This will ensure the monitoring of relevant objects during the execution of the software. The probes are inserted into all locations where monitored objects are updated such as (program variables and function calls); unlike traditional runtime verification approaches our assertion points are inserted *before* the operation to be able to intercept updates and thus prevent the system from entering an insecure state.

- **Event recognizer** is used as a communication interface between the assertion points and the runtime checker. The event recognizer sends any event that attempts to change the state of the software (with respect to the information flow policy). In addition, to sending events, the event recognizer can also send variables' value to the runtime checker to use them for checking against information flow policy and to provide intelligible feedback to the user. Although, it is possible to combine these two components the event recognizer with the assertion point, we separated them to make the implementation of our architecture more extensible, allowing for example for the integration of several runtime checkers that verify different type of requirements. For example, the management of Obligations related to information flow could be placed in a logically separate component.

- **Runtime checker** checks that the execution satisfies the information flow policy. The runtime checker determines whether or not the current execution trace as obtained by the event recognizer satisfies the information flow policy and sends feedback to the user feedback component when it determines that the software is about to enter insecure state (For example any information flow that violate the policy).

- **User feedback component** is an interface between our system and the user. An essential functionality of the user feedback component is that all user interaction passes through this component. The user feedback component informs the user about any feedback received from the runtime checker. As illustrated in "Fig. 1" 1 if the runtime checker determined that this state execution would violate the information flow policy then it sends feedback to the user, the system behavior will be changed accordingly, and the policy will be modified according to the user decision.

V. TECHNOLOGY

In order to express information flow policies unambiguously, we will base our policy language on a sound, formal foundation. Based on our previous experience with the formalization of access control policies we chose Interval Temporal Logic (ITL) as our logical framework to express information flow. ITL is a flexible notation for both propositional and first-order reasoning about periods of time found in descriptions of hardware and software systems. Unlike most temporal logics, ITL can handle both sequential and parallel composition and offers powerful and extensible specification and proof techniques for reasoning about properties involving safety, liveness and projected time. Timing constraints are expressible and furthermore most imperative programming constructs can be viewed as formulas in a slightly modified version of ITL. We will use this to formally describe the operations of a program and relate to the flow of information. As information flow is inherently temporal, we believe that ITL is expressive enough to and indeed suitable for this task. Alternative formalisms that can be used are TLA [25] (Lamport) or Event Calculus [26]. In addition ITL has an executable subset, Tempura, which means that an interpreter for our information flow policies is readily available, if expressible in this subset. Previous work [6, 27] indeed has used Tempura for the runtime verification of functional (safety, i.e., nothing bad will happen) and real-time properties. Since information flow policies are in essence safety properties we can express in Tempura. We furthermore can use AnaTempura [27] to verify those properties at runtime.

AnaTempura: tool for the runtime validation of timing and safety properties. The runtime verification technique uses assertions, similar as in our Framework, to check whether a system satisfies safety properties expressed in ITL [24, 27]. The assertions are inserted in the source code of the software and will generate a sequence of events (system states), like values of variables while the software is running. Since an ITL property corresponds to a set of sequences of states (intervals), runtime verification is just checking whether the sequence generated by the system is a member of the set of sequences corresponding to the safety property which we want to check. The Tempura interpreter is used to do this membership check [6, 27]. So, we can use AnaTempura as our runtime checker of information flow policies.

VI. CONCLUSION

This article presents a framework that provides a flexible approach to information security management by verifying that the information flow within a program execution conforms to a defined set of rules at run-time. The approach concentrates on providing a dynamic and adaptable information security solution by interacting with the user while the system is running in response to information flow events. As consequences:

- The ability for users to modify the information flow policy during runtime.
- Potential leaking behaviour of a program will be detected by the monitor and the user decides whether to abort or continue the program execution.
- Our Framework ensures that the program contains only those flows approved by the user.

VII. FUTURE WORK

So far we have only identified the technology to be used in our framework. Next step will be to actually use a medium sized case study to see if the technology is sufficient and whether it scales. Another problem that needs to be solved is the gap between user understandable information flow policies and machine understandable ITL/Tempura. Furthermore, we must be able to detect conflicts in information flow policies

References

[1] A. Banerjee and D. A. Naumann (2005) History-Based Access Control and Secure Information Flow. 3362/2005, 27-48.

[2] A. C. Myers (1999) JFlow: Practical Mostly-Static Information Flow Control. Proceeding of 26th ACM Symposium on Principles of Programming Language.

[3] K. Havelind and A. Goldberg (2005) Verify Your Runs. Verified Software: Theories, Tools, Experiments.

[4] N. Zhang and C.Yang (2002) Information flow analysis on role-based access control model. Information Management : Computer Security, 10, 225-236.

[5] P.Herrmann (2001) Information Flow Analysis of Component-Structured Applications. IEEE Computer Society Press, 45-54.

[6] H. Janicke, F. Siewe, K. Jones, A. Cau AND H. Zedan. (2005) Analysis and Run-time Verification of Dynamic Security Policies. Proceedings of the Workshop on Defence Applications and Multi-Agent Systems (DAMAS05), at 4th international joint conference on Autonomous Agents : Multi Agent Systems (AAMAS05).

[7] M. Kim, M. Viswanathan, H. Ben-abdallah, S. Kannan, L. Insup, O. Sokolsky (1999) MaC: A Framework for Run-time Correctness Assurance of Real-Time Systems. Philadelphia, PA, Department of computer and Information Science University of Pennsylvania.

[8] L. Insup, H. Ben-abdallah, S. Kannan, M. Kim, O. Sokolsky, M. Viswanathan (1998) A Monitoring and Checking Framework for Run-time Correctness Assurance.

[9] K. Izaki, K. Tanaka and M. Takizawa (2001) Information Flow Control in Role-Based Model For Distributed Objects. IEEE, 363-370.

[10] S. Cavadini and D. Cheda (2008) Run-time Information Flow Monitoring based on Dynamic Dependence Graph. IEEE Computer society.

[11] H. Janicke, A. Cau, F. Siewe AND H. Zedan. (2007) Deriving Enforcement Mechanisms from Policies. In proceedings of Policy2007, Bologna, Italy, IEEE.

[12] H. Janicke, L. Finch. (2007)The Role of Dynamic Security Policies in Military Scenarios. In Proc. Digital In Journal of Information Warfare, William Hutchinson (Edt.), Volume 6, Issue 3, pages 1-14.

[13] D. E. Denning. A lattice model of secure information flow. Commun.ACM, 19(5):236-243, 1976.

[14] D. E. Denning and P. J. Denning. Certification of programs for secure information flow. Commun. ACM, 20(7):504-513, 1977.

[15] D. Volpano, C. Irvine, and G. Smith. A sound type system for secure flow analysis. J. Comput. Secur., 4(2-3):167-187, 1996.

[16] G. Smith and D. Volpano. Secure information flow in a multi-threaded imperative language. In POPL '98: Proceedings of the 25th ACM SIGPLAN-SIGACT symposium on Principles of programming languages, pages 355-364, 1998.

[17] F. Pottier and V. Simonet. Information flow inference for ML. ACM Trans. on Programming Languages and Systems, 25(1):117-158, 2003.

[18] J. Fenton, (1974). Memory less subsystems. The Computer Journal, 17(2):143-147, 1974b.

[19] J. Brown (2001) and F. Knight, JR. A minimal trusted computing base for dynamically ensuring secure information flow. Technical Report ARIES-TM-015, MIT, November 2001.

[20] L. Chung (2006) and T. Chiueh. A general dynamic information flow tracking framework for security applications. Pages 463-472, Washington, DC, USA, December 2006.

[21] D.E. Bell and J. Lapadula. Secure computer systems: A mathematical model, volume ii., 1975.

[22] G. Birznieks (1998). Perl Taint Mode *http://gunther.web66.com/FAQS/taintmode.html*.

[23] N. Vachharajani (2004), J. Matthew, Bridges, C. Jonathan, R. Ram O, Guilherme, A. Blome, A. Reis, M. Vachharajani, and I.David. August. Rifle: An architectural framework for user-centric information-flow security. In Proceedings of the International Symposium on Microarchitecture (MICRO), December 2004.

[24] *http://www.cse.dmu.ac.uk/STRL/ITL/index.html*

[25] L. Lamport. The temporal logic of actions. ACM Transactions on Programming Languages and Systems, 16(3):872-923, May 1994.

[26] R. Kowalski and M. Sergot (1986) *A Logic-Based Calculus of Events* New Generation Computing, vol. 4 pp. 67-95.

[27] S. Zhou, H. Zedan and A. Cau. Run-time analysis of time-critical systems Journal of System Architecture, 51(5):331-345, 2005.