Interpreting Opacity in the Context of Information-hiding and Obfuscation in Distributed Systems

Anirban Majumdar Clark Thomborson
Secure Systems Group
Department of Computer Science
The University of Auckland.
Auckland, New Zealand. Private Bag 92019.
{anirban,cthombor}@cs.auckland.ac.nz

Abstract

The concept of opacity has been investigated in two different contexts as means of expressing security properties in distributed systems. Opacity in the context of information-hiding assumes the existence of a black-box and is concerned with enforcing properties such as anonymity and secrecy on collaborating processes in an untrusted distributed computing environment. In the context of software obfuscation, opacity is a measure of the difficulty of reverse engineering of object code under the assumption that the adversary has access to grey-box information. In this contribution, we bring together these two contexts in which opacity has been defined and discuss how a specialized technique, called opaque predicates, can deter malicious reverse engineering.

1. Introduction

In a multi-party computation environment, it is important to hide sensitive information from unauthorized observers. Here, we imagine a set of agents executing in an untrusted distributed computing environment to collaborate in achieving a particular objective. Examples of such an objective could be an electronic voting system or an electronic commerce bidding scenario. In the first example, it might be required to hide the identity of the agents casting votes. This property is called anonymity and is a requirement for information-hiding systems. The second example demonstrates a case where the bidding agents contain privileged computation logic and must be protected from an adversary capable of reverse engineering agent code. This is the problem addressed by obfuscation.

Halpern and O'Neill [3] analyze information-hiding requirements by answering the following three questions:

What information needs to be hidden? Halpern and O'Neill have identified several requirements of information hiding systems, most notably anonymity. Anonymity requires that identity of an agent performing an action be hidden from a malicious observer agent. The goal of obfuscation is that the algorithm implemented within agent code and its data be made difficult for an adversary to retrieve. These requirements are distinct, in that an obfuscated agent may not be able to act anonymously (in the sense of Halpern and O'Neill); and an anonymous agent may not need to be obfuscated (if the system somehow prevents malicious agents from analysing another agent's code).

Who does it need to be hidden from? The adversary agents in the information-hiding model have access to a very limited portion of the system state vector. The adversaries (which may be either agents or humans) in our obfuscation model have access to the code of all agents, and to the system state.

How well does it need to be hidden? Information-hiding is an all-or-nothing property, much like cryptography. For enforcing anonymity, for example, if identity of an agent in the electronic voting system is revealed, security as a whole is compromised. The proof of security in an information-hiding system depends on its being able to prevent adversaries from reading arbitrary facts from the system state. Obfuscation is less stringent and provides another layer of defense. The assumption is that an adversary with sufficient resources will ultimately be able to reverse engineer the agent code. Even so, the objective is to deter the reverse engineering process until the lifetime of the agent expires by making the process of reverse engineering computationally prohibitive.

In the following two sections, we discuss the context, model the adversary, and show how opacity is enforced

1-4244-0549-1/06/$20.00 ©2006 IEEE.

for expressing security properties in information-hiding and obfuscated system of mobile agents.

2. Opacity in information-hiding

Halpern and O'Neill [3] provide a comprehensive summary of existing approaches to express information-hiding properties using CSP, epistemic logic, and function-view semantics. In this section, we discuss their newly introduced epistemic logic model since it suits well for describing information-hiding properties in distributed computing environment.

2.1. Modeling a distributed system

A distributed system of mobile agents consists of a set of n inter-communicating agents, denoted by $\{P_1, \ldots, P_n\}$, executing on multiple heterogeneous hosts. These agents neither share a global clock nor share memory and communicate only through asynchronous message-passing primitives. Each agent has a *local state* at a given point in time [3] and this could be the values of the variables in the agent code. Collectively, a global state of the system can be defined in terms of a tuple of local states $(s_e, s_1, \ldots, s_n)$, where s_e is the state of the environment and s_i is agent P_i's state for $i = 1, \ldots, n$.

2.2. Modeling the adversary

An adversary is modeled as one capable of observing the system in execution. For our purpose, we assume a malicious *sniffer* agent P_a as our adversary. Here, the agent is programmed to trace the messages exchanged between other (good) agents and deduce something *interesting* from the trace. What characterizes as an interesting property is dependent on what information-hiding property the agent owner wants to protect.

Halpern and O'Neill [3] define a *run* r as a function of time t to global states of the system. A *point* is defined as a tuple (r, t). So, at a point (r, t), the system is in global state $r(t)$. If $r(t) = (s_e, s_1, \ldots, s_n)$, then $r_i(t) = s_i$. The points defined over a set of runs of the system $\mathcal{R}$ is denoted by $\mathcal{P}(\mathcal{R})$. Given this system $\mathcal{R}$, they define a knowledge function $\mathcal{K}_a(r, t)$ as the set of points in $\mathcal{P}(\mathcal{R})$ that the adversary P_a thinks are possible at (r, t). This is formally represented as:

$$\mathcal{K}_a(r, t) = \{(r', t') \in \mathcal{P}(\mathcal{R}) : r'_a(t') = r_a(t)\}$$

The adversary sniffer agent P_a knows a nontrivial fact φ (something that is not already valid) at a point (r, t) if φ is true at all points in $\mathcal{K}_a(r, t)$. This is represented in terms of an *interpreted* system $\mathcal{I}$ which consists of a pair $(\mathcal{R}, \pi)$, where $\mathcal{R}$ is the system and π is an interpretation that assigns each predicate (based on certain facts of the system) a true or false value at each point. So, the fact φ true at point (r, t) is written as $(\mathcal{I}, r, t) \models \varphi$ (and $\mathcal{I} \models \varphi$ if $(\mathcal{I}, r, t) \models \varphi$ for all points (r, t) in $\mathcal{I}$).

2.3. Modeling anonymity in terms of opacity of interpretation

For the anonymity property of information-hiding to be satisfied, the interpretation of observation has to be made opaque. As discussed in the previous section, information that need to be hidden for enforcing opacity is the identity of the agent (or the set of agents) which performs any particular action of interest to the adversary. Thus, if $\delta(P_i, x)$ implies agent P_i performs some action x, then the fact that P_i performed x needs to be hidden from adversary agent P_a.

Based on how well anonymity has to enforced, Halpern and O'Neill proposed two variations of anonymity:

- Action x performed by agent P_i is *minimally anonymous* to sniffer agent P_a in the interpreted system $\mathcal{I}$ if $\mathcal{I} \models \neg\mathcal{K}_a[\delta(P_i, x)]$. This definition tells that the fact $\delta(P_i, x)$ needs to be hidden from P_a. It is a weak information-hiding requirement since it suffices if P_a is not sure that P_i performed action x.

- Action x performed by agent P_i is *totally anonymous* to adversary agent P_a in the interpreted system $\mathcal{I}$ if

$$\mathcal{I} \models \delta(P_i, x) \Rightarrow \bigwedge_{i' \neq a} \neg\mathcal{K}_a[\neg\delta(P_{i'}, x)]$$

  This definition tells that if action x is performed by agent P_i, then the sniffer agent P_a thinks that x could have been performed by *any* of the agents except P_a.

Furthermore, Halpern and O'Neill prove that if an action x is totally anonymous, then it must be minimally anonymous as well, provided there are at least three agents in the system and it must be the case that x is performed only once in a given run of the system.

3. Opacity in obfuscation

The concept of opacity has been used to obscure understanding (for the purpose of malicious reverse engineering) of mobile agents code executing in distributed computing environment [4]. Even though Barak et al. in [1] showed some impossibility results for obfuscation by proving that every obfuscator will fail to completely obfuscate some programs, focus has shifted to finding obfuscating constructs

that are difficult but not necessarily impossible to reverse engineer.

In this section, we discuss one such construct, called *distributed opaque predicates*, which provides a way to obfuscated mobile agents code in an untrusted distributed computing environment.

3.1. Distributed Opaque Predicates

The concept of opaque predicates was originally introduced by Collberg and Thomborson in [2]. An opaque predicate is a construct with a true/false outcome. The opacity of these constructs is derived from the difficulty of reverse engineering their outcome by observation at execution time of program code.

In the mobile agents scenario, Majumdar and Thomborson in [4] define a *distributed opaque predicate* (Φ) to be an opaque predicate which depends on local states of agents in the distributed computing environment for evaluation of its outcome. Distributed opaque predicates are designed to be *temporally unstable*; i.e., it can be evaluated multiple times at different program points $(t_1, t_2, \ldots)$ in the agent code during a single run such that its observed values $(v_1, v_2, \ldots)$ are not identical. Structurally, distributed opaque predicates are relational and of the form:

$$\Phi : [(a + b + c + \ldots + n)\ \Re\ K]$$

where $(a, b, c, \ldots, n)$ are integers whose values are set by individual agents (this forms the local state), $\Re$ denotes an equality (inequality) operator such as `'='` (`'!='`) and K is a constant.

We will use the running example from [4] for the rest of this section. Given a pool of following integers (forming the local states of agents),

$$S = \{11, 9, 18, 2, 12, 5, 17, 19, 4, 7, 1, 33\}$$

a dynamic data structure, such as a doubly linked-list, is initialized with the values in S and passed to three agents in the system, P_1, P_2, and P_3. We note that only one agent, P_1, is the candidate for obfuscation (checks for the outcome of Φ). Agents P_2 and P_3 serve as guards - they communicate with other agents using message-passing primitives to maintain the local state of the obfuscated agent in a consistent state. Local state of each agent, denoted by the pointer location on its copy of the data structure, changes when sending and receiving messages. Details of this message-passing technique and state update rules can be found in [4].

The distributed opaque predicate for this example could be formed as:

$$\Phi : p_1.v + p_2.v + p_3.v = 27$$

where $p_1.v$, $p_2.v$, and $p_3.v$ denote the local states of agents P_1, P_2, and P_3 respectively. We note that Φ is satisfied for the solution vector $\{0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0\}$; i.e., only when $p_1.v = 18$, $p_2.v = 5$, and $p_3.v = 4$, for this example.

3.2. Modeling the adversary

The adversary in this context is considered to be more powerful that the one assumed for information-hiding. Here, the sniffer agent P_a has grey-box information at its disposal - in the sense it knows what to observe in the execution. It is capable of using static analyses for finding out which particular variables in the agent code constitute its local state. Based on that, it can use the same observational capability to capture messages that contain those local state values. How it might then try to infer the predicate Φ value from observation traces is discussed in the following subsection.

However, restrictions are imposed on observational powers of P_a by not letting it have access to white-box information. This restriction effectively means that P_a cannot possess sufficient static analysis information that will facilitate it to insert debugging probes at obfuscated control-flow points of the obfuscated agent. If an adversary is allowed to do this, it can quite easily determine the outcome of distributed opaque predicates by just tracing probe values during execution of the obfuscated agent P_1.

3.3. Modeling opacity using distributed opaque predicates

Opacity of the system is attributed by indeterminacy of asynchronous message passing. This indeterminacy in turn induces concurrency within the system. This is illustrated in figures 1 and 2.

These figures depict event-time mapping of a run. Here, events can be thought of as message sends and receives. Each agent changes its local state (by moving the pointer location on its copy of circular linked-list) when it sends or receives a message. As we can see from the diagrams, each agent starts at an initial local state, $p_1.v = 9$, $p_2.v = 5$, and $p_3.v = 7$. The numbers along the time axis (within square brackets) denote vector clocks and serve to maintain causal ordering of messages. Let us see what outcome of Φ the adversary P_a will deduce if it observes the system right after message labeled n is delivered to P_1.

In Figure 1, when message labeled m reaches agent P_2, the vector clock value changes to $[0, 1, 1]$ and P_2's local state, $p_2.v$, changes from 5 to 17. Agent P_3's local state, $p_3.v$, changes from 7 to 4. After P_2 sends message labeled n at vector clock $[0, 2, 1]$, its local state reverts to 5. When the adversary $P_a$tries to infer the value of Φ after $[1, 2, 1]$, the local state values returned from agents P_2 and P_3 are

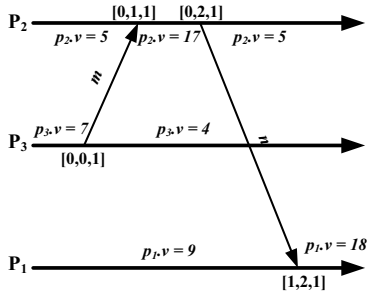

Figure 1. Ordered message delivery

$p_2.v = 5$ and $p_3.v = 4$ respectively. The local state values of agents P_1, P_2, and P_3 add up to 27 and thus Φ is satisfied (true).

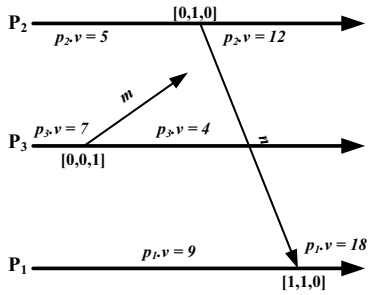

Figure 2. Delayed message delivery

In Figure 2, depicts a case where after the first receive of message labeled n at $[1, 1, 0]$ by agent P_1, it cannot be guaranteed that the message labeled m from agent P_3 originating at $[0, 0, 1]$ has reached agent P_2. This guarantee cannot be made because of indeterminacy of asynchronous message-passing. Consequently, agent P_2 changes its local state to $p_2.v = 12$, the local state values of agents P_1, P_2, and P_3 do not add up to 27. Therefore, the distributed opaque predicate (Φ) is not satisfied (false) after the delivery of message labeled n for Figure 2.

To mount a dynamic observation-based attack, the adversary agent P_a must gather relevant static analysis information that would help it trace the local state changes of individual agents. Majumdar and Thomborson argued that collecting all relevant static analysis information is impossible due to the computationally intractable issues of pointer aliasing problem. Even if the adversary manages to collect all local state information, the problem of global state monitoring is hard because the global state encoded by the temporally unstable predicates may not persist long enough for it to be true when the predicate is evaluated by the adversary.

Alternatively, P_a can sniff the agent interactions for portions of their local states that are referenced in Φ. It should then sequence each of these local states, one sequence per process, and build them up incrementally to construct the global state lattice. The state lattice thus formed is linear in the number of global states, and the number of global states formed is $O(e^n)$ where e is the maximum number of events monitored and n is the number of agents in the system. If any global state in the lattice satisfies Φ, the distributed opaque predicate is said to be detected. Again, Majumdar and Thomborson observed that if the number of guard agents in the system is large and a huge amount of message passing takes place, the adversary will face the problem of lattice state explosion. Moreover, failure to include all event changes would increase the concurrency within the system. Since there is no computationally inexpensive way for an adversary to find out if Φ was satisfied for a particular run, distributed opaque predicates aptly serve the purpose of deterring reverse engineering of agent code - quite possible for the entire lifetime of the obfuscated agent.

4. Conclusions

In this paper, we made a comparative analysis of the concept of opacity and discussed its use in context of information-hiding and software obfuscation in a distributed computing mobile agents scenario.

While information-hiding properties have a sound theoretical formalism, obfuscating constructs still lack a solid theoretical underpinning. In our future work, we will attempt to bridge this gap by trying to adapt the definitions of opacity in information-hiding to the field of software obfuscation. Relating the concept of distributed opaque predicates to the domain of epistemic logic is also a promising area worthy to be explored and will be part of our future work.

References

[1] B. Barak, O. Goldreich, R. Impagliazzo, S. Rudich, A. Sahai, S. Vadhan, and K. Yang. On the (im)possibility of obfuscating programs. In *CRYPTO-2001*, volume 2139, pages 1–18. Lecture Notes in Computer Science, Springer-Verlag, 2001.

[2] C. Collberg, C. Thomborson, and D. Low. Manufacturing cheap, resilient, and stealthy opaque constructs. In *POPL '98: Proceedings of the 25th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 184–196, New York, NY, USA, 1998. ACM Press.

[3] J. Halpern and K. O'Neill. Anonymity and information hiding in multiagent systems. In *16th IEEE Computer Security Foundations Workshop (CSFW 2003)*, IEEE, pages 75–88. IEEE Computer Society, 2003.

[4] A. Majumdar and C. Thomborson. Manufacturing opaque predicates in distributed systems for code obfuscation. In *Twenty-Ninth Australasian Computer Science Conference (ACSC 2006)*, volume 48 of *CRPIT*, pages 187–196, Hobart, Australia, 2006. ACS.