An Empirical Study of the Strength of Information Flows in Programs

Wes Masri
Computer Science Department
American University of Beirut
Beirut, Lebanon 1107 2020
wm13@aub.edu.lb

Andy Podgurski
Electrical Engineering & Computer Science Department
Case Western Reserve University
Cleveland, OH 44106
podgurski@case.edu

ABSTRACT
Dynamic information flow analysis aims at monitoring the flow of information among objects in an executing program. It is based on the assumption that if two objects are connected by a sequence of dynamic data and/or control dependences, then information actually flows between them. This paper seeks to empirically verify the validity of this assumption and to explore the relationship between the strength of an information flow and its length.

Categories and Subject Descriptors

General Terms
Security, Experimentation.

Keywords
Dynamic information flow analysis, program dependence, entropy, correlation.

1. INTRODUCTION
It has long been recognized that to validate the security of a software system it is necessary to analyze, among other things, the information flows that can occur between program variables or objects during its execution [1][7]. Certain information flows may indicate either leakage of sensitive information or tampering with such information. Both static [4][6][17] and dynamic [7] techniques for information flow analysis have been developed.

Dynamic information flow analysis (DIFA) is based on the assumption that the presence of a dynamic program dependence between two objects implies that information actually flows between them. That is, information flow occurs from object $x$ to object $y$ whenever there is a sequence of $n \geq 2$ program actions $a_1, a_2, \ldots, a_n$ such that $x$ is used by $a_i$, $y$ is defined by $a_n$, and for $i = 1, 2, \ldots, n - 1$, $a_{i+1}$ is dynamically data dependent or dynamically control dependent on $a_i$. In this paper we will try to empirically verify the validity of this assumption, i.e., we will try to answer the question: Is dynamic program dependence truly indicative of information flow in real programs? For this purpose, we will compute the entropy-based strength [5] of the information flows (identified using dependence analysis) that occurred during the execution of several programs and then analyze the frequency distribution of the computed strengths.

In [12] we presented a new approach to DIFA that can be used to detect, prevent and debug insecure flows in programs. A prototype tool implementing the proposed approach for Java byte code programs was also described. Our information flow monitoring tool, which we call DynFlow, can be used to prevent and detect violations of information flow policies offline with either synthetic test cases or captured operational inputs [18], and it can also be used online during deployment. This last capability is possible only with a forward computing algorithm. Forward computing algorithms operate in tandem with program execution and have the advantage that they do not require a previously stored execution trace. In [13] we applied DynFlow to a number of Java programs in order to assess its time and storage impact. In most cases there was a considerable slowdown and storage impact. The results made it clear that for processing intensive applications (those having relatively large execution traces per request), it is not feasible to apply our tool online, but only offline and for debugging. On the other hand, for interactive and/or non-processing-intensive applications, it seems feasible to apply our tool either offline or online. In this work, we extend DynFlow to enable our empirical study.

In this paper we will also try to empirically study the relationship between the strength of an information flow and its length, i.e. the length of the dependence chain. Specifically, we will try to answer the question: Is the length of an information flow indicative of its strength? If for example we were able to determine that the strength of a flow is inversely proportional to

---

1 An action is an execution of a program statement.
its length then we could disregard any detected illegal flows that are excessively long or even, for the purpose of optimization, exclude lengthy flows from our computation. On the other hand, if we concluded that lengthy flows could be as strong as short ones then no detected illegal flow could be dismissed as benign.

Sections 2 and 3 describe our computation of information flow strength and length, respectively. Section 4 presents our empirical findings about the relationship between program dependence and information flow. Section 5 presents our empirical findings about the relationship between the strength of an information flow and its length. Section 6 surveys related work. Finally, Section 7 presents our conclusions and future work.

2. INFORMATION FLOW STRENGTH

We rely on classical information theory [3] to determine whether information flow has occurred between two objects and specifically to quantify the amount of transferred information, i.e. to measure the strength of the flow between two objects.

Now we define the entropy-based measure of the strength of an information flow that is based on the information-theoretic analysis presented in [5]. The entropy of a random variable $X$, denoted $H(X)$, is a measure of uncertainty in $X$:

$$H(X) = -\sum_{x \in X} P(x) \log_2 P(x)$$

(Note that by convention, $\log(0) = 0$, because $x \log(x) \to 0$ as $x \to 0$.) The entropy of $X$ given $Y$, denoted $H(X|Y)$, is a measure of the uncertainty about the value of $X$ when the value of $Y$ is known:

$$H(X|Y) = -\sum_{x \in X, y \in Y} P(x, y) \log_2 P(x|y)$$

Let $e$ be the execution of a sequence of statements, and let $X$ and $Y$ be objects in the program. The execution sequence $e$ causes a flow of information from $X$ to $Y$ if and only if $H(X|Y) < H(X)$. That is if we are more certain about the value of $X$ by observing $Y$ after $e$, or, we can deduce additional information about $X$ by observing $Y$ after $e$. We define our entropy-based measure of information flow strength as:

$$\textbf{StrengthFlow}(X, Y) = H(X) - H(X|Y)$$

$\textbf{StrengthFlow}(X, Y)$ will be zero if no measurable flow occurred between $X$ and $Y$, and greater than zero otherwise. We estimate the probabilities needed to compute $\textbf{StrengthFlow}(X, Y)$ based on a sample of actual flows between $X$ and $Y$ that were recorded. For the estimates to be reasonably accurate, the sample cannot be too small. In the experiments reported in this paper, we ignored flows that occurred fewer than five times. Note that $\textbf{StrengthFlow}(X, Y)$ is applicable to flows where $X$ and/or $Y$ are of scalar types or object types whose states can be represented by scalars. In the latter case, the scalar associated with an object state will be incorporated in the computation of $\textbf{StrengthFlow}(X, Y)$ as a substitute for the value of the object, i.e. $x_i$ or $y_j$. Also, note that in the experiments presented in this paper we ignored information flows that involved objects that are not of scalar types or the java.lang.String type.

As an illustration, consider the computation of $\textbf{StrengthFlow}(X, Y)$ where $(X, Y)$ took on the following values: $(5, 100), (5, 100), (5, 100), (6, 200), (6, 200), (7, 300), (7, 300), (7, 300), (7, 300)$. We have

$$H(X) = -P(X=5) \log_2 P(X=5) - P(X=6) \log_2 P(X=6) - P(X=7) \log_2 P(X=7) = \frac{1}{3} \log_2 (1/3) - 2/9 \log_2 (2/9) - 4/9 \log_2 (4/9) = 1.53$$

$$H(Y) = -P(Y=100) \log_2 P(Y=100) - P(Y=200) \log_2 P(Y=200) - P(Y=300) \log_2 P(Y=300) = -1/3 \log_2 (1/3) - 2/9 \log_2 (2/9) - 4/9 \log_2 (4/9) = 1.4$$

This yields a $\textbf{StrengthFlow}(X, Y)$ value of 0.13; suggesting a weak flow.

3. INFORMATION FLOW LENGTH

The length of an information flow is the length of the chain of the dynamic data and control dependences it comprises. In order to present our equation for computing the length of an information flow, we first need to provide some background definitions:

1) An action is an execution of a program statement. An action at position $k$ in an execution trace is denoted $s^k$ where $s$ is the executing statement.

2) $U(s)$ is the set of variables or objects used at action $s^k$.

3) $\textbf{DInfluence}(s^k)$ is the set actions that directly influence action $s^k$ through intra-procedural or inter-procedural data and control dependence.

4) $\textbf{InfoFlow}(s^k)$ is the set objects from which information directly or indirectly flowed to $s^k$.

Detailed descriptions of the above definitions are provided in [12].

Let $o_i$ be the source and $o_j$ the target of an information flow such that action $s^k$ last defined $o_i$ and action $t^l$ last defined $o_j$. The associated flow instance is uniquely identified by the quadruple $(o_i, o_j, s^k, t^l)$. Note that in the sequel the pairs $(o_i, o_j)$ or $(s^k, t^l)$ are used as shorthand for the quadruple $(o_i, o_j, s^k, t^l)$.

We base our computation of the length of a flow instance on the following equation:

$$\textbf{LengthFlow}(s^k, t^l) = \begin{cases} 1, & o_i \in U(t^l) \\ \text{Min}(\textbf{LengthFlow}(s^k, r^p) + 1, & o_j \notin U(t^l) \end{cases}$$

where $r^p \in \textbf{DInfluence}(s^k)$ and $o_j \in \textbf{InfoFlow}(s^p)$.
**Figure 1** – Number of flows vs. strength for **Xerces**

**Figure 2** – Number of flows vs. strength for **JTidy**

**Figure 3** – Number of flows vs. strength for **Jigsaw**
The length is 1 if \( f^0 \) uses \( o_s \), otherwise it is the length of the shortest dependence chain through which information from \( o_s \) possibly flowed into \( o_t \). Note that when applying the above equation at an action \( f^0 \), all the values it depends on have already been computed and are available. This inductive nature of the equation makes our algorithm forward computing and hence easy to compute online and integrate into our tool.

4. IS PROGRAM DEPENDENCE TRULY INDICATIVE OF INFORMATION FLOW IN REAL PROGRAMS?

For the purpose of empirically answering the above question we conducted experiments using the XML parser Xerces 1.3, the XML pretty printer JTidy 3, the Servlet engine jigsaw 2.0.5, and the Servlet/JSP engines Tomcat 3.0 and Tomcat 3.2.1. Each subject program was executed on numerous inputs of varied types and complexity. Xerces and JTidy were executed using XML files that were selected from the XML Conformance Test Suite (www.w3.org/XML/Test). jigsaw 2.0.5 was executed using HTML and Servlet requests that were selected from the examples included in the distribution of Tomcat 3.0. Tomcat 3.0 and Tomcat 3.2.1 were executed using HTML, Servlet and JSP requests that were also selected from the examples included in the distribution of Tomcat 3.0.

For each flow instance we stored the quadruplet \( (o_s, o_t, value_s, value_t) \) where \( o_s \) is the source identifier, \( o_t \) the target identifier, \( value_s \) the source value and \( value_t \) the target value. We then computed the strength of each flow instance using the definition \( \text{StrengthFlow}(o_s, o_t) \). The required probabilities were estimated based on the recorded flows. Note that in our experiments we ignored information flows that involved objects that are not of scalar types or the \texttt{java.lang.String} type. In the case of an object of type \texttt{java.lang.String}, we assumed that the integer value returned by the \texttt{java.lang.String.hashCode()} method represents the state of the object and therefore we used it as a substitute to the value of the object when computing \( \text{StrengthFlow}(o_s, o_t) \).

This approach assumes that the hashing algorithm implemented in the \texttt{java.lang.String.hashCode()} method is collision free. Given
that it is not, there is a possibility that the results of our experiments will be slightly inaccurate due to the (very) rare occurrences of collisions. Note that we will address this problem in future work by devising techniques to generate scalars that faithfully represent the states of objects of type java.lang.String or any other type. Figures 1-5 show for each subject program the number of flows that occurred at a given flow length, i.e., the frequency distribution of flow strengths. Note that Figures 1-5 show the results of only a subset of the executions as the results for the rest of the executions are practically identical. Clearly, the majority of the flows are very weak, for all subject programs; only 14% to 23% of the flows have measurable entropy-based strengths and only a small proportion of them are very strong. Consequently, an illegal flow detected by our (program dependence based) tool has a 77% to 86% probability of being benign. Thus, although the presence of a static program dependence between two statements has been proven to be a necessary condition for one of them to affect the behavior of the other [15][16], our results suggest that even the occurrence of a dynamic dependence between program actions is often not indicative of measurable information flow between them.

5. IS THE LENGTH OF AN INFORMATION FLOW INDICATIVE OF ITS STRENGTH?

For the purpose of empirically answering the above question we conducted experiments similar to the ones described in the previous section, but for each flow instance we stored the quintuplet \((o_s, o_t, l, value_s, value_t)\) where \(l\) is the length of the flow computed using the definition \(LengthFlow(o_s, o_t)\). We then computed \(StrengthFlow(o_s, o_t)\) for the entries sharing the same source, target, and length. Finally, we computed the average of the strengths that were computed for a given length \(l\).

Figures 6-10 show the relationship between the length of flows and the average entropy-based strengths following the executions of our five subject programs. Note that the length reflects the number of executed byte code instructions and not Java statements. Also, as with Figures 1-5, Figures 6-10 show the results of only a subset of the executions, because the results for the rest of the executions are practically identical. Figure 6 and Figure 9 indicate that for Xerces and Tomcat 3.0, the strength of an information flow is to some extent inversely related to its length. However, this is not the case for the remaining three applications, especially not for jigsaw and JTidy where longer flows tend to be almost as strong as short ones.

These results suggest that, in general, the length of an information flow is not indicative of its strength. This in turn suggests that a long illegal flow could be as harmful as a short one and therefore should not be disregarded.

6. RELATED WORK

Lampson motivated research on information flow analysis by describing the problem and listing a number of possible information leaks [8]. Fenton [7] proposed an abstract machine called the Data Mark Machine to support dynamic checking of information flows. Denning and Denning [4][6] proposed a technique based on static control flow and data flow analysis for verifying a program’s compliance with an information flow policy. A number of language-based static type checking systems have also been proposed [17]. In such systems every program expression is assigned a security type in addition to its ordinary type. In type checking a program, the compiler ensures that the program cannot exhibit illegal information flows at runtime.

In [12] we implemented a prototype tool for DIFA. In [9] we used the tool to generate information flow and slicing execution profiles to be used in test case minimization experiments. In [14] we used it to demonstrate the efficiency of DIFA in online signature-based and offline anomaly-based intrusion detection.

Clark et al [2] used information theory to provide a formal approach to analyze the amount of confidential information which may be leaked by programs. Although their approach can be automated it applies only to programs written in a very simple imperative language that contains no iterations. Lowe [11] quantified the amount of information passed through covert (timing) channels. His proposed approach is based upon counting the number of different behaviors of a high level user that can be distinguished by a low level user. Note that our study did not involve information flows through covert channels and our tool is not capable of detecting them.

In [1] the dependence of predicates on global variables and formal parameters was empirically studied. The results show that as the number of formal parameters available to a predicate increases, there is a decrease in the proportion of these formal parameters that influence the predicate. No such correlation was found for global variables. The work also presents results that provide strong evidence that the globals and formals are independent of one another and that their numbers are independent of the size of the procedure that contains them.

7. CONCLUSION AND FUTURE WORK

The results of our empirical study of dynamic dependences suggest the following:

1) The existence of dynamic program dependence between two objects does not indicate that with high probability there is a measurable information flow between them. That is, dynamic program dependence is not necessarily indicative of actual information flow in real programs.

2) No general relationship exists between the length of an information flow and its strength. That is, the length of an information flow is not indicative of its strength.

Our results suggest that an illegal flow detected using dependence analysis has a high probability of being benign, i.e., a false positive. They also suggest that an excessively long illegal flow could be as harmful as a short one and therefore cannot be dismissed, i.e. every detected illegal flow must be looked at carefully, regardless of its length.

To try to confirm these results, we plan to replicate our study with additional subject program from a variety of application domains. Also, we plan to investigate other techniques for computing the strength of information flows including techniques that are not based on information theory, such as generalized-correlation based techniques. Finally, considering our results indicating that long information flows can also be strong, we plan to investigate how the strength/length ratio is affected by factors such as program structure, coupling, cohesion, conformance to the “Law of Demeter” [10], and the presence or absence of defects.
Figure 6 – Length vs. entropy-based strength for Xerces

Figure 7 – Length vs. entropy-based strength for JTidy

Figure 8 – Length vs. entropy-based strength for Jigsaw
Figure 9 - Length vs. entropy-based strength for Tomcat 3.0

Figure 10 - Length vs. entropy-based strength for Tomcat 3.2.1

8. REFERENCES