A Tool for Estimating Information Leakage*

Tom Chothia, Yusuke Kawamoto, and Chris Novakovic

School of Computer Science, University of Birmingham, Birmingham, UK

Abstract. We present leakiEst, a tool that estimates how much information leaks from systems. To use leakiEst, an analyst must run a system with a range of secret values and record the outputs that may be exposed to an attacker. Our tool then estimates the amount of information leaked from the secret values to the observable outputs of the system. Importantly, our tool calculates the confidence intervals for these estimates, and tests whether they represent real evidence of an information leak in the system. leakiEst is freely available and has been used to verify the security of a range of real-world systems, including e-passports and Tor.

Introduction. Information leakage occurs when something about a system's secret data can be deduced from observing its public outputs. Not all information leakage is serious: many retailers' billing systems readily "leak" the last four digits of a credit card number, and password-checking functions "leak" some information about a secret password in response to an incorrect guess (e.g., that the guess is not the password). Information leakage is therefore quantitative and it is important to be able to answer the question "how much information does a system leak?". Information theory is a useful framework for quantifying these leaks in systems (see e.g. [9]), and two particular measures, mutual information and min-entropy leakage, place useful bounds on an attacker's ability to guess the secrets from the public outputs.

Our tool, leakiEst, estimates these leakage measures from datasets containing secrets and public outputs that are generated from trial runs of a system. Its methodology is based on our previous work that provides rigorous verification methods for estimating information leakage [3,4]; it performs statistical tests to distinguish an insecure system with a very small information leak from a secure one with no leaks. This is similar to detecting a correlation between two random variables, a well-investigated problem, and we compare leakiEst's performance to that of existing statistical tests. If a leak is found, leakiEst can display the conditional probability of observing each output from the system given a particular secret, which may be used to derive a concrete attack against the system.

There are several tools that calculate the amount of information that leaks from a program (e.g., [7,2]). These tools provide tight bounds, but require access to the source code of the program and a formalism that is powerful enough to model the underlying system. These requirements are often prohibitive, and

* This work was supported by EPSRC Research Grant EP/J009075/1.

prevent these tools from being used in the case studies below. By estimating leakage based on trial runs of a system, we trade precise leakage calculations for the ability to detect leaks in complex, real-world systems. Other tools, such as Weka [6], can estimate the mutual information of random variables, but do not calculate the confidence interval for their estimates, nor do they test for compatibility with zero leakage; it is therefore difficult to ensure that the estimates produced by these other tools are meaningful.

leakiEst, its documentation and sample datasets are available at [1].

Estimating Information Leakage. leakiEst can analyse data collected from systems that contain a secret value, and whose observable behaviour is probabilistic and possibly affected by the secret. We assume that there is a probability distribution X on the secret values and another probability distribution Y on the public outputs. The system is defined by the likelihood of observing each possible output given each possible secret, i.e., by the conditional probability distribution $P_{Y|X}$. As an example we consider a Java program in which two random integers between 1 and 10 are generated sequentially; the first is visible to an attacker, and the second must be kept secret. One potential flaw would be the use of the Java API's cryptographically weak `Random` class for pseudorandom number generation, rather than the stronger `SecureRandom` class: if `Random` were used, the value of the second integer may be related to the first, which would constitute an information leak. In this case, there would be a correlation between Y, the probability distribution on the integer that the attacker observes, and X, the distribution on the secret integer generated afterwards. X and Y's mutual information, or the min-entropy leakage from X to Y, defines how difficult it is for the attacker to guess the secret integer from the observable integer.

leakiEst estimates the magnitude of information leaks solely from trial runs of a system: a user must first run a system many times with a range of possible secret values and record the observed outputs to create a dataset that can be processed by leakiEst. This system-agnostic approach offers the greatest flexibility to users of the tool, and we note that for particular types of systems (e.g., RFID cards, web traffic, or Java programs) it would be possible to build a framework to automatically generate an appropriate dataset; we provide a Java API so leakiEst's functionality can easily be integrated with other tools.

Given a dataset, the tool estimates the conditional probability distribution $\hat{P}_{Y|X}$ of the system and implements tests we have proposed in previous work [3,4]. In [3] we calculate the bias and distribution that the repeated estimates of discrete mutual information will follow in terms of the true mutual information. This allows us to estimate a confidence interval, and therefore test whether the apparent leakage indicates a statistically significant information leak in the system or whether it is in fact consistent with zero leakage. The estimates are non-parametric; i.e., they do not assume that secrets and outputs fit any particular distribution. In [4] we extend this technique to cover estimates of continuous mutual information using kernel density estimation, allowing leakiEst to estimate leakage from systems with continuous outputs.

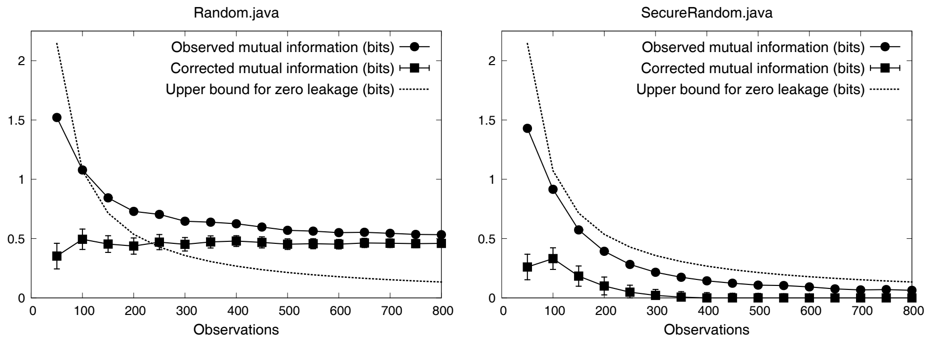

Fig. 1. A depiction of leakiEst's output for the Java `Random` and `SecureRandom` sample programs. The graphs' data sources are produced by leakiEst's `-csv` option, which calculates leakage estimates at user-defined intervals in a dataset and writes them to a CSV file.

Two further novel features of our tool are the calculation of an upper bound on the expected mutual information for systems containing no information leaks and the calculation of confidence intervals. It is these features that allow us to interpret leakiEst's output in a meaningful way. Fig. 1 depicts leakiEst's output for datasets generated by programs that utilise the Java `Random` and `SecureRandom` classes in the manner described earlier (their source code is available at [1]). Each graph denotes both the observed and corrected mutual information of the secret values and public outputs in the dataset, with the x-axis denoting the number of observations that produce those estimates. Other tools (e.g., Weka) can calculate the uncorrected observed mutual information value, but this is of limited use in isolation, as it can be difficult to tell whether that value represents a true information leak or is just noise in the data.

The dashed line shows the 95% upper bound on the measurement expected if the true mutual information were zero. For values of $x > 150$, when the observed mutual information falls below the upper bound for zero leakage, the left-hand graph provides clear evidence that there is an information leak from the first random integer to the second when the `Random` class is used to generate random numbers. For cases where the mutual information is not zero (i.e., there is a leak), [3] provides a prediction of the bias and variance of the estimate. This allows us to make a prediction of the true mutual information, labelled as "corrected mutual information" in both graphs. This quantifies the information leakage (approximately 0.5 bits) accurately, even for a very small number of observations. The right-hand graph shows that when `SecureRandom` is used to generate random numbers the mutual information is always below the expected confidence interval for zero leakage, therefore there is no evidence of leakage from the first random integer to the second. While this does not guarantee that `SecureRandom` is secure (and for much larger ranges of numbers it may not be), the data processing inequality guarantees that an attacker learns nothing statistically significant from this particular dataset.

Our estimates make the assumption that terms of the order $(\#\text{samples})^{-2}$ are small, but for a large number of secrets and outputs and a small number of samples this may not be the case. For instance, when the product of the number of secrets and outputs is in the hundreds, tens of thousands of samples may be required for reliable results. leakiEst can test whether enough samples have been provided and warn the user if more are required.

leakiEst uses a new technique [1] to calculate the confidence interval for the estimated min-entropy leakage, which measures the vulnerability of secrets to single-attempt guessing attacks.

The Tool. leakiEst is developed in Java and may be used as either a Java API or as a standalone JAR file that can be invoked from the command line, so it can be integrated into the development workflow of software written in any programming language; it is a suitable component of system testing (to uncover new information leaks) as well as regression testing (to ensure that previously-discovered leaks have not been reintroduced). leakiEst's Java API also exposes common information theory and statistical functions that developers may find useful. We chose Java, as opposed to (e.g.) MATLAB or R, to make the tool more accessible to non-specialists and to simplify standalone execution.

The simplest datasets processed by leakiEst are text files with lines of the form `("secret","output")` describing a single trial run of the system. From this input, the tool calculates the conditional probability distribution $\hat{P}_{Y|X}$, estimates min-entropy leakage and mutual information using $\hat{P}_{Y|X}$, calculates the confidence intervals, and (for discrete mutual information) performs tests on the estimate in search of statistical evidence of non-zero leakage. For more complex systems (e.g., those containing multi-part secrets or producing multiple outputs per secret), leakiEst can process datasets recorded in Weka's ARFF file format. While each line of the file still describes a single trial run, the format of each line is more structured, with named "attributes" allowing the system's various secrets and outputs to be distinguished. Command-line options can be supplied to instruct leakiEst to treat arbitrary sets of attributes as secrets or outputs. Given an ARFF file, the tool calculates the mutual information and min-entropy leakage confidence intervals for the specified secret and each of the outputs individually. Another function orders all of the outputs by the amount of information they leak, allowing users to focus their attention on minimising leakage caused by particular outputs.

Scalability. leakiEst generates the conditional probability matrix for $\hat{P}_{Y|X}$ for a given list of observations. The tool updates the matrix in-place for each observation read, meaning it scales well for datasets containing a large number of observations: our `Random` and `SecureRandom` datasets, each containing 500 million observations forming 10×10 matrices, can be analysed in 3 minutes on a modern desktop computer. The tool scales less well for datasets containing a large number of unique secrets and outputs (which result in matrices with larger dimensions), but is nevertheless able to estimate leakage in many real-world scenarios: systems where the secret is a 4-digit PIN and the output is a binary value

Table 1. The p-values of leakiEst and other non-parametric tests when applied to an e-passport dataset containing 500 observations

Nationality	leakiEst	KS test	CVM test	AD test	BWS test
British	0	0	0	0	0
Irish	0	0.001	0	0	0
Greek	0.075	0.718	0.544	0.367	0.408
German	0	0.257	0.743	0.302	0.271

(i.e., a $\approx 2^{13} \times 2$ matrix) can be analysed in under 10 seconds, and systems where the secret is 19 bits of a key and the output is a binary value (i.e., a $2^{19} \times 2$ matrix) can be analysed in a day.

Case Study: Fixing an e-Passport Traceability Attack. The RFID chip in e-passports is designed to be untraceable; i.e., without knowing the secret key for a passport, it should be impossible to distinguish it from another passport across sessions. In [4,5] we observed that e-passports fail to achieve this goal due to a poorly-implemented MAC check: passports take longer to reject replayed messages. This means that a single message can be used to test for the presence of a particular passport. Here, the secret is a binary value indicating whether the passport is the one the attacker is attempting to trace, and the output is the time taken for the passport to reply. We collected timing data from an e-passport and analysed it with leakiEst, which clearly detects the presence of an information leak from a dataset containing 100 observations. Attempting to fix the leak, we developed a variant of the e-passport protocol that pads the time delays so that the average response time is equal in all cases [4]. leakiEst still indicated the presence of a small information leak: while the average times are the same, it appears that the actual time measurements come from a different distribution. After modifying the protocol to continue processing a message even when the MAC check fails, and only reject it at the end of the protocol, leakiEst indicates that it is free from leaks.

In cases where the secret is a single binary value, a number of existing non-parametric tests can be used to test whether two samples originate from the same distribution. The most popular of these are the Kolmogorov-Smirnov (KS), Baumgartner-Weiß-Schindler (BWS), Anderson-Darling (AD) and Cramér-von Mises (CVM) tests. Table 1 compares the p-values of these tests when applied to 500 observations of the time-padded protocol variant for e-passports from a range of countries that all implement different variations of the protocol. The p-value indicates the proportion of tests that failed to detect the leak, so the table shows that leakiEst detects leaks more reliably than the other tests for datasets of this size.

Case Study: Fingerprinting Tor Traffic. Tor is an anonymity system that uses encryption and onion routing to disguise users' network traffic. Traffic is encrypted before it leaves the user's node, but an intermediary can still infer information about the web sites a user is requesting based on characteristics of the encrypted traffic: the time taken to respond to the request, the number of packets, the packet sizes, the number of "spikes" in the data stream, etc. Here, the secret is the URL of the web site whose encrypted traffic is being intercepted by the attacker, and the public outputs are the characteristics of the encrypted traffic that the attacker is able to observe.

Such an attack has previously been mounted against Tor [8], using Weka to fingerprint encrypted traffic with a 54% success rate; since fingerprinting the web site is possible, clearly a leak exists. We generated a dataset by accessing each of the Alexa top 500 web sites ten times through a Tor node and recording features of the encrypted traffic. leakiEst ranks them and identifies which features (or sets of features) leak information. Existing machine learning tools (e.g., Weka) can be configured to select features based on mutual information but, uniquely, leakiEst estimates a confidence interval for each measure of mutual information, ranks the features in the dataset by reliability, and identifies the features that do not leak information. leakiEst's analysis showed that a web site is most easily fingerprinted by the number of spikes in the data stream. It also showed that some features suggested by previous authors, such as the average packet size, do not contain any useful information; we verified this by removing these features from the dataset and rerunning the classification in Weka, and observed no drop in the identification rate.

References

1. leakiEst, `http://www.cs.bham.ac.uk/research/projects/infotools/leakiest/`
2. Backes, M., Köpf, B., Rybalchenko, A.: Automatic Discovery and Quantification of Information Leaks. In: Proc. S&P, pp. 141–153 (2009)
3. Chatzikokolakis, K., Chothia, T., Guha, A.: Statistical Measurement of Information Leakage. In: Esparza, J., Majumdar, R. (eds.) TACAS 2010. LNCS, vol. 6015, pp. 390–404. Springer, Heidelberg (2010)
4. Chothia, T., Guha, A.: A Statistical Test for Information Leaks Using Continuous Mutual Information. In: Proc. CSF, pp. 177–190 (2011)
5. Chothia, T., Smirnov, V.: A Traceability Attack against e-Passports. In: Sion, R. (ed.) FC 2010. LNCS, vol. 6052, pp. 20–34. Springer, Heidelberg (2010)
6. Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., Witten, I.H.: The WEKA Data Mining Software. SIGKDD Explorations 11(1), 10–18 (2009)
7. McCamant, S., Ernst, M.D.: Quantitative Information Flow as Network Flow Capacity. In: Proc. PLDI, pp. 193–205 (2008)
8. Panchenko, A., Niessen, L., Zinnen, A., Engel, T.: Website Fingerprinting in Onion Routing Based Anonymization Networks. In: Proc. WPES, pp. 103–114 (2011)
9. Smith, G.: On the Foundations of Quantitative Information Flow. In: de Alfaro, L. (ed.) FOSSACS 2009. LNCS, vol. 5504, pp. 288–302. Springer, Heidelberg (2009)