SAND2011-6434C

Tradeoffs in Targeted Fuzzing of Cyber Systems by Defenders and Attackers

[Extended Abstract]

Jackson R. Mayo
Sandia National Laboratories
Livermore, California 94551
jmayo@sandia.gov

Robert C. Armstrong
Sandia National Laboratories
Livermore, California 94551
rob@sandia.gov

ABSTRACT

Automated randomized testing, known as fuzzing, is an effective and widely used technique for detecting faults and vulnerabilities in digital systems, and is a key tool for security assessment of smart-grid devices and protocols. It has been observed that the effectiveness of fuzzing can be improved by sampling test inputs in a targeted way that reflects likely fault conditions. We propose a systematic prescription for such targeting, which favors test inputs that are "simple" in an appropriate sense. The notion of Kolmogorov complexity provides a rigorous foundation for this approach. Under certain assumptions, an optimal fuzzing procedure is derived for statistically evaluating a system's security against a realistic attacker who also uses fuzzing.

Categories and Subject Descriptors

K.6.5 [**Management of Computing and Information Systems**]: Security and Protection; C.4 [**Computer Systems Organization**]: Performance of Systems—*measurement techniques*; D.2.5 [**Software Engineering**]: Testing and Debugging—*testing tools*

General Terms

Security, Testing, Measurement

Keywords

fuzzing, Kolmogorov complexity

1. INTRODUCTION

Digital devices and protocols for the smart grid are subject to the same basic security problem that has plagued general-purpose computers and other cyber infrastructure. The fundamental origin of this problem is the combinatorial complexity of digital systems, which provides a vast space for unknown faults and vulnerabilities [2]. Designers and defenders of such systems cannot find and eliminate every vulnerability, whereas an attacker only needs to find a single one. In the face of this problem, possible *analytical* approaches for establishing confidence in cyber systems include

- formal methods [8], which require considerable computational power and can give rigorous assessments for systems of moderate complexity, and
- analysis of emergent robustness using complexity theory [1], which can give probabilistic assessments of particular designs even with very high complexity.

While these approaches are promising for cyber infrastructure such as the smart grid, we focus here on an approach that is synergistic with them and is even more widely applicable: automated randomized testing, known as fuzzing [11]. Fuzzing can give a probabilistic assessment of reliability and security for an arbitrarily complex system and, unlike the analytical approaches above, does not *require* more than a black-box view of the system. The tradeoff is that the feasible coverage of combinatorial spaces through fuzzing is increasingly poor as systems become more complex, and thus the difficulty of finding any particular fault grows rapidly. Fuzzing has discovered previously unknown vulnerabilities in smart meters, and is a recommended practice for security assessment of smart-grid hardware and software, particularly given their dependence on network protocols [4, 7].

A major concern in fuzzing is the distribution of pseudorandom test inputs that are fed to the system. It is recognized that fuzzing with *uniformly* random bit strings is relatively ineffective [11]. Rather, fuzzing typically takes into account properties of a realistic system to generate a more targeted distribution of test inputs, e.g., by mutation starting from expected inputs [9, 11]. Here we develop a systematic "wedding cake model", based on the concept of Kolmogorov complexity, for how fuzzing can best be targeted in the simple setting of a black-box system. This model provides a basis for future generalizations that can account for additional information about the system.

2. WHY ASSESSING SECURITY IS HARD

Undecidability results exemplified by the halting problem and Rice's theorem [10] arise because a Turing machine has infinitely many possible states and the number of computational steps needed to reach a given state cannot be bounded in general. Real cyber systems have a finite state space that in principle can be explored exhaustively, and thus questions about their input-output behavior are not undecidable but

CSIIRW '11, October 12–14, Oak Ridge, Tennessee, USA

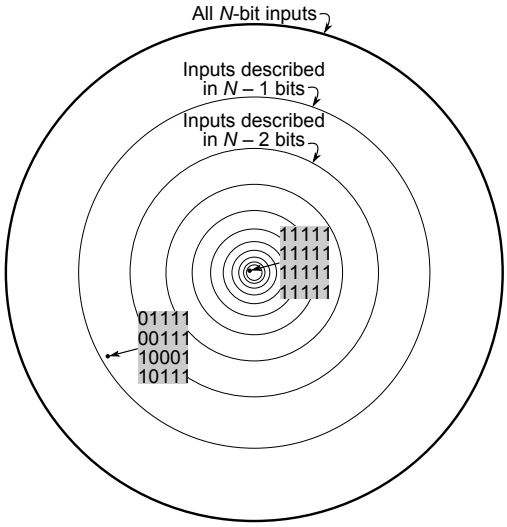

Figure 1: Space of program inputs organized by Kolmogorov complexity.

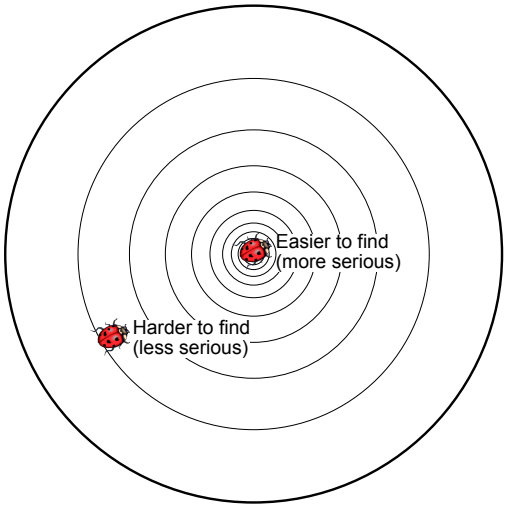

Figure 2: Position of faults in input space, and difficulty of finding them.

merely combinatorially hard. In practice, the state space is typically so large that exhaustive exploration is infeasible and the questions of interest are *effectively* undecidable.

In this paper we consider the problem of verifying that a program, given any possible input from a finite (but combinatorially large) set, does not exhibit some behavior deemed a fault. Assume that, for any given input, the presence or absence of a fault can be determined in a *fixed* time by running a test. For example, if the program is intended to produce a particular output, then failure to produce the intended output within a fixed time is considered a fault even if the program would eventually produce this output. (This is the paradigm of real-time systems.) Here the number of states reachable within the time limit is finite, and the verification question posed is decidable because each possible input can be tested. But it is still combinatorially hard because of the need to cover the vast input space.

The practical impossibility of exhaustive verification for a generic complex program implies that a defender cannot gain confidence that no faults occur for untested inputs. As a result, such a program must always be considered as vulnerable to a sufficiently capable attacker, for whom the existence of a single fault enables, in principle, a successful compromise. This view – which argues against the utility of statistical characterization of faults since the attacker's behavior is not assumed to be describable statistically – may be unduly pessimistic when the computational limits of a real attacker are accounted for. Through suitable testing, it may be possible for the defender to characterize faults statistically such that a computationally limited attacker seeking a fault can do no better than a random search of the input space. In this way, statistical measures of security can be meaningful even for an effectively undecidable system.

3. KOLMOGOROV COMPLEXITY AND THE WEDDING CAKE

The key insight is that the vast majority of inputs in a large combinatorial space are algorithmically random, i.e., the shortest description from which the input can be constructed is not significantly shorter than the input itself. By "description" we mean a *program* for generating a bit string; we refer to it as a "description" to avoid confusion with the program being fuzzed, to which the bit string is fed as input. The length of the shortest description of a bit string is known as its Kolmogorov complexity [6]. The choice of (Turing-complete) language used for the description makes only an $O(1)$ difference in this complexity measure. If the possible inputs are strings of N bits, then the vast majority have a Kolmogorov complexity within $O(1)$ of N; see Figure 1. *A priori*, without receiving magic information of at least $O(N)$ bits, an attacker has no way of preferentially distinguishing any one algorithmically random input from others. Thus, within the algorithmically random inputs – which, again, cover the vast majority of the input space – the attacker is effectively reduced to discovering faults through testing of uniformly pseudorandom inputs.

The one potentially useful way the attacker can bias his search in the absence of prior information is to treat separately the relatively few possible inputs of low Kolmogorov complexity. These lie in the small inner rings of Figure 1 (with the very lowest-complexity inputs, such as all 1's, found at the center). From the attacker's perspective, it is at least possible that faults are more common among this small subset of inputs and, if so, much can be gained by sampling them in preference to the high-complexity inputs – a targeted form of fuzzing. As depicted in Figure 2, an individual fault is much more susceptible to discovery by the attacker, and thus of much greater concern to the defender, if it is associated with a low-complexity rather than a high-complexity input. We hypothesize that faults are in fact more common among low-complexity inputs in typical programs. A reason to expect this is the well-known observation that faults often arise from "corner case" inputs as opposed to generic ones [9].

A simple example is an ordinary password-checking program intended to deny access to attackers. Access is granted only when the input matches a specific password. The behavior upon entering the correct password can be viewed as a "fault" that is deliberately introduced for the benefit of

authorized users and is designed not to be found by attackers. A good password checker, although fairly simple, is like a generic Turing-machine program in that it operates as a black box; even if the code is visible, strong cryptography permits only brute-force (trial-and-error) attacks. The main issue is password strength – the low-complexity fault in Figure 2 corresponds to a weak, easily guessed password, while the high-complexity fault corresponds to a strong password. It is known that a password chosen randomly from a large space has a low, quantifiable probability of brute-force compromise. In accordance with the arguments made here, both defenders and attackers have reason to give special attention to the small space of low-complexity passwords. Such passwords are often employed in practice for user convenience, and correspondingly attackers benefit greatly by focusing brute force on them. To be sure, a password checker is not a generic complex program and may, for example, be formally verifiable by the defender. But it provides an example of a system that contains a deterministic fault (i.e., the fault, once discovered, works every time) and yet possesses statistically quantifiable security against a realistic attacker.

The Kolmogorov complexity of arbitrary bit strings is known to be non-computable [6]. The "description" used in the definition of Kolmogorov complexity is a generic Turing-machine program, for which the halting problem is undecidable. Thus, it is generally impossible to know that one has found the shortest program producing a given string, since one does not know whether a shorter program *eventually* produces it. In practice, because the purpose of invoking input complexity is to promote efficient sampling, a time limit will be applied and a replacement input will be generated if a string is not produced in time. The attacker will not wait indefinitely for a compact but long-running description to produce a low-complexity N-bit input string – and the defender need not either.

There is also no need to search for the shortest program; sampling from descriptions of length k results in sampling from inputs of Kolmogorov complexity k or smaller, i.e., from the interior of a given circle in Figure 1 (as opposed to a single ring). This is suitable because low-complexity inputs should, if anything, be oversampled; nothing significant can be gained by purposely *avoiding* low-complexity inputs, since at their unbiased weight they contribute negligibly to the cost of fuzzing. The combination of such sampling options for various values of k can be visualized as a three-dimensional "wedding cake" built on top of Figure 1. Since each sampling "tier" includes low-complexity inputs (i.e., extends to the center of the space), the tiers pile up and contribute incremental preferential sampling for such inputs. We assume that the defender and attacker arrive at effectively the same procedure for sampling inputs at a given complexity (same horizontal tier profiles), though they may of course use different distributions of input complexity (different tier "heights") for their respective testing.

4. FUZZING STRATEGIES

How, then, can a defender gain quantified confidence in the ability of a complex program to withstand attack? The defender wishes to estimate the probability of success per trial for an attacker seeking a fault, which is inversely related to the amount of work required of the attacker per fault discovered. In the generic Turing-complete regime, both defender and attacker are limited to fuzzing – optionally weighted by input complexity – as a means of finding faults. (It is possible that some or all of the faults found by the defender's fuzzing can be fixed before deployment; in this case, the goal of fuzzing is not only to evaluate the program but also to improve it.) The sampling procedure for a given tier of input complexity defines a statistical ensemble of inputs, and the results of fuzzing are a statistical reflection of the rates at which faults occur in these ensembles. To give a worst-case confidence estimate, it is assumed that the attacker knows the statistical results of the defender's fuzzing and will target his own fuzzing to the most promising tier of input complexity. Thus, the defender seeks to *minimize* the *maximum* estimated fault rate among the tiers.

We use a Bayesian approach for statistical inference of fault rates from the outcomes of the defender's fuzzing. A simple "uninformative" prior distribution for fault rates is the uniform distribution between 0 and 1; this prior is conservative (i.e., pessimistic) because fault rates in reasonably well-designed systems are expected to be small. Denote by f_k the fault rate for inputs sampled from the set of descriptions of length k, a set of size 2^k, for $k = 0, 1, \ldots, N$. Assume first that the defender's fuzzing is being done to evaluate the final version of the system, with no further fixes or other modifications being made.

Suppose that, at a given stage of fuzzing, R_k descriptions of length k have been tried and F_k of them have been found to result in faults. Inference of the fault rate f_k from these observations is similar but not identical to inference of a probability from Bernoulli trials (e.g., a series of biased coin flips). The difference is that the descriptions are sampled without replacement from a finite set, and so the posterior expected probability of a fault, $(F_k + 1)/(R_k + 2)$, applies only to the untested descriptions [3]. The number of faults among the R_k tested descriptions is known and need not be inferred statistically. Thus the posterior expected value of the fault rate for tier k as a whole is

$$\langle f_k \rangle = \frac{2^k - R_k}{2^k} \frac{F_k + 1}{R_k + 2} + \frac{F_k}{2^k}. \quad (1)$$

Before any observations are made ($R_k = F_k = 0$), the pessimistic estimate $\langle f_k \rangle = \frac{1}{2}$ is obtained; at the other extreme, for exhaustive testing ($R_k = 2^k$), the exact value $f_k = F_k/2^k$ is recovered. For large k, where testing can only scratch the surface, the first term in Eq. (1) dominates, whereas for small k, where exhaustive or nearly exhaustive testing is feasible, the second term eventually dominates. In the former case, in which $\langle f_k \rangle \simeq (F_k + 1)/(R_k + 2)$, it is useful to note that high confidence (low estimated fault rates) can be established with an amount of fuzzing that is large but need not scale with the combinatorial size 2^k; for example, if no faults are found, the estimated fault rate is nonzero but decreases inversely with the amount of fuzzing performed.

The concrete fuzzing procedure for the defender is then as follows. Test an input sampled from the complexity tier that has the largest estimated fault rate $\langle f_k \rangle$ as given by Eq. (1) based on the results obtained so far. This includes the initial condition where no observations exist and $\langle f_k \rangle = \frac{1}{2}$ for all k; ties are broken arbitrarily. Reevaluate the estimated fault rates based on the accumulated counts R_k and F_k, and identify the tier for sampling the next test input. Repeat as long as appropriate, either to obtain a desired maximum estimated fault rate or until the time allocated for fuzzing is exhausted. Early on in this procedure, the estimated fault

rate in a given tier will tend to decrease as more data are gathered by running tests from that tier, if the true fault rate is small. Thus, sampling each successive test from the tier with the current highest estimated fault rate will tend to reduce the maximum estimated fault rate as fuzzing proceeds. If the true fault rates are sufficiently small that very few faults are ever found, then the estimates will remain on the high or pessimistic side (due to the occurrence of $F_k + 1$ in the formula) and will continue to decrease slowly; the confidence obtained will be limited by the time available. But, if any fault rate is large enough that substantial numbers of faults are discovered, then the procedure will reduce to refining a precise estimate of the largest fault rate; the estimate will not continue to decrease, and the question is then whether this fault rate is deemed acceptable.

A slightly different analysis applies if an attempt is made to fix faults as they are found. Now suppose that, in each tier k, of the F_k faults observed, P_k have been successfully patched so that they will not recur. If it is assumed that patching a given fault does not affect any other behavior of the program, then the estimated fault rate is obtained from Eq. (1) by reducing the number of faults among tested inputs but not among untested inputs, i.e.,

$$\langle f_k \rangle = \frac{2^k - R_k}{2^k} \frac{F_k + 1}{R_k + 2} + \frac{F_k - P_k}{2^k}. \quad (2)$$

For large k, where testing can only be very sparse, the effect of this change is insignificant because the first term dominates. For small k, however, the effect of P_k on confidence can be substantial; for example, if testing is exhaustive in the tier ($R_k = 2^k$) and all faults discovered are fixed ($P_k = F_k$), the result $\langle f_k \rangle = 0$ is as good as if no faults were present in the tier to start with. Because the program is being improved in the process of fuzzing, estimated fault rates will tend to decline due to both increasing statistical precision and elimination of faults. If the true fault rates are initially highest at small k, then the fuzzing procedure will tend to focus defender effort on patching faults from these low-complexity inputs, which is a particularly good investment because the estimated fault rate can be reduced substantially by exhaustive patching in smaller combinatorial spaces. In effect, this can be described as purging the "weak passwords" from the system.

5. CONCLUSIONS

We present an analysis of fuzzing strategies based on Kolmogorov complexity, applicable to smart-grid devices. Our hypothesis is that low-complexity input is a more likely place to find faults in typical programs. In a plain-text passphrase, for example, encodings based on the character set and grammar of the language in which it is written allow for a shorter representation than its naïve ASCII bit representation. Descriptions of length k are generating functions for inputs having complexity at or below k. More generally, the input description can be considered a generating function that operates within a "base representation" obtained from the program's nominal lexical structure or even semantics. In fact, Kolmogorov complexity can be defined as a measure of one string *relative* to another string that provides background information [5]. This generalization corresponds to "edit distance" in a Turing-complete editing language.

Such considerations are important when the amount of pertinent background information is comparable to or larger than the input complexity. Intuitively, programs should exhibit faults close to their nominal input space. In an SQL injection attack, for example, alien bit strings are more likely to be benignly ignored than input generated from SQL grammar. Pragmatically, it is more effective to draw a base representation for input descriptions from the "natural" representation of the nominal input for the program being fuzzed.

For hypothetical programs for which the input semantics are completely unknown, no choice can be made for the proper base representation for input descriptions. Because there will always be a finite-length program that can transform one base representation into another, the base representation is less important for high complexity. The choice of base representation does not affect the validity of our simple analysis for *asymptotically* long inputs since the description length becomes independent of its representation. However, because we expect the most fruitful application of fuzzing to occur at low complexity, pragmatically we expect that the choice of base representation will be a significant concern.

6. ACKNOWLEDGMENTS

Sandia is a multiprogram laboratory operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy under contract DE-AC04-94AL85000.

7. REFERENCES

[1] R. C. Armstrong and J. R. Mayo. Leveraging complexity in software for cybersecurity. In *Proceedings of the 5th Cyber Security and Information Intelligence Research Workshop*, Oak Ridge, TN, Apr. 2009.

[2] R. C. Armstrong, J. R. Mayo, and F. Siebenlist. Complexity science challenges in cybersecurity. Sandia Report SAND2009-2007, Mar. 2009.

[3] E. T. Jaynes. *Probability Theory: The Logic of Science*. Cambridge University Press, 2003.

[4] P. Kahlon. Security issues in system development life cycle of smart grid. Project (M.S.), California State University, Sacramento, 2011.

[5] M. Li and P. M. B. Vitányi. Two decades of applied Kolmogorov complexity. In *Proceedings of the 3rd Annual Structure in Complexity Theory Conference*, Washington, DC, June 1988.

[6] M. Li and P. M. B. Vitányi. *An Introduction to Kolmogorov Complexity and Its Applications*. Springer, third edition, 2008.

[7] S. McLaughlin, D. Podkuiko, S. Miadzvezhanka, A. Delozier, and P. McDaniel. Multi-vendor penetration testing in the advanced metering infrastructure. In *Proceedings of the 26th Annual Computer Security Applications Conference*, Austin, TX, Dec. 2010.

[8] J.-F. Monin. *Understanding Formal Methods*. Springer, 2003.

[9] P. Oehlert. Violating assumptions with fuzzing. *IEEE Security and Privacy*, 3(2):58–62, 2005.

[10] A. Singh. *Elements of Computation Theory*. Springer, 2009.

[11] M. Sutton, A. Greene, and P. Amini. *Fuzzing: Brute Force Vulnerabilty Discovery*. Addison-Wesley Professional, 2007.