Security Metrology and the Monty Hall Problem

Bennet S. Yee

April 2, 2001

Abstract

Evaluating computing systems and classifying them by the security properties they provide is not new [13, 14]. Other researchers [8, 9] have pointed out the difficulty of evaluating security and the apparent binary nature of security given discoveries of system vulnerability. Here, I compare the role of security evaluations with that of cryptographic security parameters, and relate the difficulty of arriving at security metrics with the Monty Hall Problem.

Additionally, I argue that trying to represent the security of a system by either a single numeric value or constructing some digraph using which systems are compared is a Quixotic affair: security needs are application dependent, and no single total or partial ordering can provide all the information needed. I give example scenarios that demonstrate the need for multi-faceted security rating systems.

1 Introduction

It seems obvious that measuring the "security level" of a system is a good thing. Quantifying and measuring security would enable us to easily compare two systems and determine which is better — perhaps even allowing security-conscious organizations to make purchase decisions based on the "security measurements" or some form of "price/security" ratio. Vendors would be motivated to improve the security of their systems.

While apparently quite desirable, before making the design of a scalar security measure (rating) or a method for partial ordering systems (ranking) a goal of systems security work, we must determine whether such a measure is indeed meaningful. This paper addresses this issue.

2 Existing Security Evaluations

Before we try to figure out what new security ratings might mean and how to use security ratings, we should look at some existing systems of security ratings. Here, we look at two important security evaluation standards: Trusted Computer System Evaluation Criteria ("Orange Book") ratings [13] and Security Requirements for Cryptographic Modules (FIPS 140-1) [14]. The Common Criteria [7] does not mandate security requirements per se, but attempts to standardize the process of evaluating the evidence supporting security claims. It is largely orthogonal to the subject of security metrics — instead of trying to measuring security, it essentially tries to measure the "error bounds" of the claimed security.

2.1 Orange Book Evaluation

The Orange Book [13] provided a classification of computer systems which has been very influential. Government procurement often required that computer systems possess a certain rating, which entailed an expensive evaluation process. Orange Book mandates the inclusion of various security provisions at different security levels. These provisions include audit logs, Discretionary Access Controls (DAC), Mandatory Access Controls (MAC), etc.

2.2 FIPS 140-1 Evaluation

The Security Requirements for Cryptographic Modules [14] provides classifications of hardware modules implementing cryptographic functions for use in environments where the adversary has physical access to the modules. FIPS 140-1 mandates resistance to various attacks at different security levels. The effort required to actually mount these attacks covers a wide spectrum. The attacks range from simple lock picking to physically or chemically drilling through any portion of the physical enclosure.

3 Does Security Have A Binary Nature?

A primary critique of the attempt to arrive at a single scalar measure of security can be summarized as the "binary nature of security." [8, 9] The view here is that a system either has a security flaw or it doesn't. This has a long tradition: either a (security) proof is correct, or it isn't; either an algorithm is correct, or it isn't. In this view, assigning a numeric value is problematic: while an initial security review that misses a subtle flaw would give a system a high security rating, a subsequent review that discovers the flaw *should* cause the rating to plummet to zero.

What is missing here is that security is truly binary — but only in an information theoretic setting. By this, I mean that the evaluators/attackers having *unrestricted code inspection* and *unlimited computational resources*. Here unrestricted code inspection is required because in live systems limits such as account lock-out when maximum login attempts are reached can limit the amount of information that is learned by the experimenter. In lieu of unrestricted code inspection, we can substitute instead the ability to revert the system to an earlier state.

Clearly, when in such an information theoretic setting where computational resources are unlimited, exhaustively testing *all* reachable states of a system is feasible. By doing so and seeing if any undesirable states are reachable, we can determine if the system is secure.

Of course, real computing systems do not operate in an information theoretic setting. If that were the case, not much beyond one-time pads could be used.[1] Just as modern cryptography is conducted in an complexity-theoretic setting where computational resources are limited, security must be considered in a similar manner. Before I make the analogy more explicit, let us first revisit the roles that parameters play in cryptographic protocol design.

3.1 Security Parameters in Cryptographic Protocols

Broadly, there are two classes of values that are security critical in cryptographic protocol/system design: keys, and security parameters. Cryptographic keys — especially long term keys such as those used in public key cryptosystems — must be kept secret, and any disclosure immediately compromises the particular instance of the system. Re-keying, however, is comparatively very cheap.

Security parameters are values used to restrict the range of various values in a cryptographic system. Once chosen, they are often difficult to change: the parameter values determine sizes of registers and widths of data paths in hardware implementations; in software implementations, they are often compiled-in values, and updating them may require recompiling and reinstallation of software (possibly at many sites). Typical parameters are key sizes or number of rounds of interaction, e.g., in an interactive proof of knowledge protocol. Selecting values for these parameters determines the resources needed by an adversary to break the cryptographic protocol. Concrete security [1] relates values of computation time, space usage, probability of success, etc for a cryptographic scheme to corresponding values for a cryptographic primitive (based on reductions), so that the discovery of a method to efficiently break the cryptographic scheme under investigation would necessarily entail that some underlying cryptographic primitive would also be broken — by using the discovered method as a subroutine. The resources needed to break the cryptographic primitive can be back-calculated from the resources needed to run the subroutine.

[1] Not only can we not figure out what states are reachable, just determining what states are undesirable can be extremely difficult. For the purposes of this exposition, we ignore the specification problem.

Most cryptography papers give resource usages for breaking the cryptographic scheme being studied as functions of the security parameters. This is, of course, very useful when selecting security parameters, since this enables the system designers to chose the parameter values so that the cost of the work that attackers will have to do will be more expensive than the benefits that they will gain from a successful attack. Determining the actual monetary cost to be incurred by the attacker must necessarily involve estimation — the formulae give only the cost in terms of computational resources required, and not only are the cost of computational resources likely to continue to fall rapidly over time, but the benefits to the attacker of a successful attack must also be estimated for the foreseeable lifetime of the system.

3.2 Security Measures as Resource Estimates

Existing established security evaluation criteria do not provide good resource estimates. Security classifications in the Orange Book simply indicate whether some desired security property — such as MAC — is available. It is not an information assurance measure, per se: the system design may include MAC, but implementation errors may make it possible to bypass TCB enforcement.

Security classifications used in FIPS 140-1 evaluation [14] are bald naked numbers without associated units — they are simply intended as ordinal numbers for the FIPS 140-1 list of security classes, rather than actual numeric measures. There is some given interpretation of these numbers, but they are *immetrical*: the so-called measurements do not use well-established standard units.

The properties measured cannot readily be converted from one form to another using standard equations like gravitational potential energy can be converted into light. Using some reasonably well-understood techniques, we can use system integrity and a little bit of confidentiality to create a larger amount of confidentiality: a secure coprocessor with a small amount of tamper-responding private memory can hold and process a much large amount of confidential information by "crypto-paging" to non-private memory [10, 15] (modulo traffic analysis). However, this is not "conversion": we don't lose system integrity when we "gain" more private memory! (N.B., the memory privacy of crypto-paged memory is not strictly equivalent to the non-paged memory, since attackers might learn some partial information from the paging behavior.) Instead, a properly constructed cryptography engine coverts CPU cycles into confidentiality. Unfortunately, the security here is still binary in nature: the private memory is axiomatically private — and any loss of confidentiality will cause the scheme to fall apart.

3.3 Work Factor Estimates Derived From Penetration Testing

Many researchers have proposed using Red Team man-hours as a basis for security metrics. The total amount of work required (1) to discover a security vulnerability — say D man hours, (2) to engineer an attack based on the vulnerability — say E man hours, and then (3) to run the engineered exploit — say X man hours (or CPU cycles) — seems like a natural, non-binary starting point for constructing a security metric. There are some difficulties — the amount of time required by a Red Team depends greatly on the skill levels involved and to a smaller extent on luck, so $D + E + X$ is a measurement that is not only relative to the given Red Team skill set, but also a randomly sampled value from an unknown "luck" distribution about the skill-set determined data point. It remains to be seen whether the inherent difficulties in accurately determining the true value can be overcome.

Another critical problem with penetration testing derived statistics is that the directly measured values are like the positions of atoms: they change as a result of the act of measuring them! Certainly, the same Target of Evaluation (TOE) will be much easier to penetrate for the second Red Team if the results from the first measurement is available: even if the first team did not create an automated exploit tool, just knowing where the vulnerability is will make it much easier the second time around ($E + X$). Even if the only information available is that some vulnerability exists that can be discovered, engineered, and exploited in $D + E + X$ man-hours, this can aid in directing the search (e.g., do not bother to look for difficult-to-exploit vulnerabilities). Similarly, so-called "script kiddies" enjoy the fruits of more advanced attackers' labors,

and by downloading the appropriate automated exploits can penetrate systems by expending only about X man-hours of work.

3.4 The Monty Hall Problem

As promised, I will now relate the problem of security metrics to the Monty Hall Problem. To briefly review, the Monty Hall Problem is as follows:

Set up Monty Hall tells the contestant — Alice — that behind one of the three doors A, B, and C is the grand prize (say, a car). Behind the other two doors are consolation prizes.

First choice Alice choses one of the three doors. Say this is door A. Monty does not yet reveal to her what is behind this door.

Consolation Prize Because there are two consolation prizes, behind at least one of the two remaining doors is a consolation prize. Monty picks one of these — say door B — and shows Alice the consolation prize behind it.

Second choice Alice is given the opportunity to switch to take what is behind door C or to stay with what is behind door A. What should she do?

The Monty Hall Problem has been widely analyzed [6], and the optimal strategy is to always switch. The crux of the problem is that the probabilities change as a result of information being revealed when door B is opened.

Just as the contestant's optimal strategy changes based on available information, the resource requirements for successfully penetrating a TOE also depend on available vulnerability information and penetration tools. The *conditional* nature of probabilities in the Monty Hall Problem naturally mirrors the conditional nature of available knowledge about the vulnerabilities of a system: knowing that security evaluation experts estimate that a system can be broken into after $D + E + X$ amount of work — but with no prior information — can greatly prune the vulnerability search space that a second, possibly malicious penetration team must explore.

3.5 The Security-Through-Obscurity Conundrum

Given the conditional nature of vulnerability testing values, we might be tempted to simply classify the detailed results of Red Team testing. By not revealing the nature of discovered vulnerabilities and not releasing engineered exploit tools, the work required in future penetration attempts is not influenced by the act of measurement. However, this implies that systems are also not allowed to benefit from Red Teaming: discovered vulnerabilities cannot be fixed. Worse yet, since the amount of work needed to discover and exploit a vulnerability can help to guide future search, the results of testing cannot actually be used, e.g., to compare systems, without leaking information that will aid future attackers!

3.6 Fixing Discovered Flaws

Of course, it is only by fixing vulnerabilities discovered in evaluation/testing efforts that systems may be improved. While desirable, fixing the vulnerabilities discovered by Red Team evaluations confuses the interpretation of the result. By making changes in the code, the TOE is changed. What is the relevance of work-effort estimates on earlier versions of the TOE to the current one? Given that finding the now-fixed vulnerability cost C man-hours, what can we say about the expected cost C' of finding the next one? In order to create an accurate, predictive model for how this conditional expected value C' depends on C, iterated Red Team evaluations is likely to be necessary.

Discoveries of implementation (or design) flaws — even by benign agencies — usually result in a significant amount of re-engineering. When the flaws are in deployed systems, operational costs can be even more

significant. While we might be tempted to compare the spread of information on vulnerabilities to the spread of information about cryptographic keys when a key exposure occurs, the update cost does not compare favorably: in cryptographic systems, re-keying is only an operational cost — no engineering analysis is required to determine whether the changeover to a new key (as opposed to a security patch) might introduce additional vulnerabilities.

4 Multifaceted Security Measures

In order to determine the desirability and meaningfulness of a single scalar measure or a single partial ordering for comparing systems by how well they provide security, let us imagine some "archetypical" applications with security needs. Doing so will enable us to determine what metrics, if any, might be suitable, as well as help us determine whether there are meaningful ways to combine values obtained into a single scalar value. Here, I argue that the security needs of different applications can differ greatly, that any single ordering will necessarily satisfy these requirements only in an incomplete fashion, and that a multi-faceted security measure is necessary.

A. The first archetypical application is a medical database for use by medical professionals. Since it contains patients' health records, confidentiality and availability are critical. Availability, however, is more important — emergency accesses might override privacy concerns, e.g., EMS technician would get access during health emergencies, despite lack of prior authorization. (Role-/situation-based access control.)

B. The second archetypical application is a publicly-accessible congressional records database. For simplicity, let us ignore records from closed sessions, etc, and assume that non-public-record data are segregated into a separate database. In this application, data integrity and availability are important requirements, but there are essentially no privacy concerns, especially if no access records are kept.[2] One could imagine implementations using read-only media, e.g., public kiosks with ROM-based operating systems and data in the form of CD-ROMs, as well as web-based implementations.

C. The last archetypical application is an Intelligence operatives database — code names, real names, drop locations, analysis, etc. For this application, confidentiality and strong access controls are presumably more important than availability. Implementations that favor data destruction, e.g., via tamper sensing[11, 12], over availability would likely be preferred.

These applications have different (primary) security needs. In the medical database application, availability is paramount. In the public records application, integrity is most important. Finally, in the intelligence database, confidentiality is most critical. I have chosen these three applications to highlight the "classical" security attributes of confidentiality, integrity, and availability. Other security properties, such as anonymity, may also be important for some applications (e.g., a web site for whistle-blowers to submit their concerns).

In order for a security rating or ranking system to be adopted, it must have relevance for its targeted audience. Suppose a system builder wishes to put together the above applications using Commercial Off-The-Shelf (COTS) systems. What information should the ranking system provide that would be useful? Clearly, because of the differing security needs, any single rating/ranking cannot completely satisfy the system builder's needs.

Suppose a hypothetical (cheap) operating system and web server are available which together provide high throughput web services. However, the web server provide little or no confidentiality: SSL [5, 2], HTTP basic authentication [3, 4], etc are not supported. Perhaps the web server is poorly designed and will happily serve the entire filesystem to the world. This system might, however, provide good system integrity and thus would adequately support the public records application, but it would be completely inappropriate for

[2] Recently personal privacy concerns have caused some information such as the social security numbers of public figures to be elided from some otherwise public data compilations. I assume that such elision has already occurred, so no sensitive data are in the system.

medical records or intelligence data. But because of its inability to handle confidential information, this COTS configuration would likely receive a low ranking. Even though confidentiality was not a requirement for this application, such a low ranking would likely cause this system to be ignored. Similarly, a database that provides high availability and easy-to-use access control overrides might be great for medical records, but access control overrides are also quite inappropriate for intelligence data.

Security needs of applications are different, and security measures need to provide data that are relevant to those needs. Minimally, separate rankings for confidentiality, integrity, and availability are needed to address the disparate security concerns.

5 Conclusions

The desire for a single security ranking or, less ambitiously, a single security partial ordering is a laudable one. However, the path towards achieving such a goal is unclear. In this paper I have argued that while security is binary in nature in an information-theoretic setting, security "measurements" can still be meaningful: in the more practical complexity-theoretic setting we can attempt to measure the resources needed by attackers to penetrate a system. However, I also showed that actually making security measurements is difficult for various reasons: the TOE changes when discovered bugs are fixed, raising questions about the applicability of Red Team resource usage due to TOE versioning; revealing or using the results of a security measurement can invalidate the measurement — or at least increase the uncertainty in its accuracy, since future attackers can optimize their security vulnerability testing based on this information. Furthermore, I argued that a single ranking or partial ordering does not serve the purposes for which such security measures would be useful — a multi-faceted or multi-dimensional security measure is much more useful, and it should be the goal of security measurement researchers to make this realizable.

References

[1] Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *Proceedings of the 5th ACM Conference on Computer and Communications Security*, 1993.

[2] Tim Dierks and Christopher Allen. The TLS protocol, version 1.0, November 1998. Internet Engineering Task Force Internet Draft; see `http://www.ietf.cnri.reston.va.us/internet-drafts/draft-ietf-tls-protoc%ol-06.txt`.

[3] R. Fielding, Jim Gettys, J. Mogul, H. Frystyk, and Tim Berners-Lee. Hypertext transfer protocol – HTTP/1.1. Internet RFC 2068, January 1997.

[4] J. Franks, P. Hallam-Baker, J. Hostetler, S. Lawrence, P. Leach, A. Luotonen, and I. Stewart. HTTP authentication: Basic and digest access authentication. Internet RFC 2617, June 1999.

[5] Alan Freier, Philip Karlton, and Paul Kocher. The SSL protocol version 3, December 1995.

[6] Thomas L. Magliozzi and Raymond "Chuckie" Magliozzi. Let's make a puzzler: Monty hall puzzler proof, 1997. `http://cartalk.cars.com/About/Monty/`.

[7] U. S. National Institute of Standards and Technology. Common criteria for information technology security evaluation, version 2.1, 1999.

[8] Bruce Schneier. A Cyber UL? Crypto-Gram Newsletter, January 2001. `http://www.counterpane.com/crypto-gram-0101.html`.

[9] Bruce Schneier. Are we ready for a cyber-UL? *ZDNet News*, January 2001. `http://www.zdnet.com/zdnn/stories/comment/0%2c5859%2c2669708%2c00.html`.

[10] Sean Smith and Dave Safford, 2001. Personal Communication.

[11] Sean W. Smith and Vernon Austel. Trusting trusted hardware: Towards a formal model for programmable secure coprocessors. In *Proceedings of the Third USENIX Workshop on Electronic Commerce*, pages 83–98. USENIX Association, 1998.

[12] Sean W. Smith and Steve H. Weingart. Building a high-performance, programmable secure coprocessor. Technical Report Research Report RC21045, IBM T. J. Watson Research Center, November 1997.

[13] U. S. Department of Defense, Computer Security Center. Trusted computer system evaluation criteria, December 1985.

[14] U. S. National Institute of Standards and Technology. Federal information processing standards publication 140-1: Security requirements for cryptographic modules, January 1994.

[15] Bennet S. Yee. *Using Secure Coprocessors*. PhD thesis, Carnegie Mellon University, 1994.