# Computer Security is Not a Science
(but it should be)

Michael Greenwald Carl A. Gunter Björn Knutsson
Andre Scedrov Jonathan M. Smith Steve Zdancewic

University of Pennsylvania

## 1 Introduction

Security research is sometimes referred to as the "Humanities of Computer Science" because, too frequently, "secure" systems are built using equal measures of folklore and black arts. Despite the humorous intention, there is a kernel of truth in this jest—computer security, at least "security in the large", is not currently a science.

This claim may seem unfair, given the progress made in security over the past decades. However, our present tools and methodologies are at most adequate for understanding systems security on a small scale. Cryptography, for example, is perhaps the most thoroughly studied and most rigorously modeled aspect of security. Despite its tremendous importance, cryptography alone is not sufficient for building secure systems. Indeed, the vast majority of all security flaws arise because of faulty software (*e.g.*, the ubiquitous buffer overflow problem). Such security holes cannot be avoided by cryptographic techniques, and despite widely known and accepted solutions to these kinds of software flaws, buggy code persists.

Why is security not a science? Some would argue that, by nature, security is fundamentally unscientific: security is hopelessly intertwined with social and economic forces beyond the purview of science. Yet, economists and psychologists have developed testable, scientific theories.

What sets science apart from other disciplines is that it produces hypotheses that can be experimentally verified (or falsified). But, despite the large amounts of security-relevant data collected by organizations like CERT and despite our decades of experience building systems, computer security research has produced little in the way of predictive models or experimentally verifiable hypotheses.

How can we establish security in the large on a more scientific footing? Over the last millenium, one way that disciplines have evolved into "sciences" is through a period of quantification. For example, Galileo, among others, transformed physics from an Aristotelian philosophy to a Baconian science by *describing* distance, speed, and time *quantitatively*, rather than *explaining* why objects fell, rolled, or flew.

Our belief is that the current pre-scientific state of security research is fundamentally due to a lack of reasonable metrics. Furthermore, although there exist a few experimental methods for assessing security (i.e., tiger-teaming [5]), these methods are not yet particularly meaningful in the context of science, where quantitative evaluation—for comparison, modeling, and measurement of achievement—is central.

The main questions we are interested in addressing are:

**Question 1:** How could one measure security quantitatively?

**Question 2:** What experiments ought one perform to assess security?

**Question 3:** How can we improve our models using these metrics?

We believe that we can eventually achieve a workfactor-like formulation to address the first question. Such a formulation will likely be a composite of a variety of measurements, with imprecise but meaningful weights. As in physics, an approximation that can gradually be refined with experience is very use-

ful, because it allows hypotheses to be tested, even without six sigmas of precision.

In this short note, we examine the problem of turning security into a science and suggest some possible future directions for research. We do so by considering these questions in the context of examples drawn from network security.

## 2 Denial of service attacks

One area that requires new research is the quantification of Denial of Service (DoS) threats. Traditional formal models such as Dolev-Yao [3] provide an excellent basis for finding logical/algebraic flaws in protocols. Work on cryptographic security guarantees, in which threats in protocols are reduced to cryptographic questions, is able to find an additional collection of risks. However, more work is needed to extract explicit information about workload, where constant factors are often critical.

For example, a protocol that relies on public key signatures may be vulnerable to an attack when a similar protocol relying on Message Authentication Codes (MACs) is not. On a stock PC, a signature may take 10ms, whereas a MAC may take 1$\mu$s. If an attacker can generate a load of 1000 such operations per second, the victim will not be able to keep up with the necessary public key signatures, but will require only .1% of its effort to deal with the MACs. On the other hand, if a DoS threat applies to public key verifications (rather than signatures) and these require about 80$\mu$s, while a MAC for the same protocol requires taking the hash of a 1500byte packet at a cost of about 15$\mu$s, then key management advantages of public keys may outweigh the comparatively small factor of extra cost entailed by using them rather than MACs.

It is not unusual for a conservative analysis of DoS to be carried out as follows. Assuming that an adversary is able to use all of the inbound channel of a victim, the victim is able to devote a sufficiently modest portion of its processing and memory to recognizing the packets of the attacker. The example above illustrates this reasoning when a victim is getting about 1000 service requests per second.

This is a sound way to analyze the problem but it is in many cases over conservative and leads to the adoption of excessive measures. In particular, if the attacker is able to control *all* of the inbound channel, then it typically will not matter what the cryptographic load is: the denial of service will be successful. Thus the assumption is only interesting in cases where the attacker and a legitimate claimant share the channel. Suppose, for instance, that the attacker can send 1000 service requests in a second, but the victim only needs to find one valid request per second. If the valid claimant sends 1 request per second, then the victim will need to check about 1000 requests. But if the claimant sends, say, 10 requests per second, then the victim will only need to check an average of 100 service requests per second. In general, if such tradeoffs exist, the attacker's rate cannot accurately measure the DoS threat without a value for the claimant rate. Thus, models that assume the attacker controls the whole channel are too conservative in many cases.

Models of prerequisite workload and payment have been considered in various contexts. For instance, there are efforts to reduce spam by making an email sender pay a fee to the receiver; presumably spammers will not be able to afford the resulting expenses for enough people to make it profitable for them to continue their activities. This fee could be money, but it could also be processor time.

Another approach is to design protocols so that the claimant must reveal an identity to some degree or somehow forced to have 'skin in the game'. For instance, a common strategy is to send back a cookie to a claimant, thus forcing the claimant to (be able to at least appear to) use a valid return address. The next step in research could well be more formal ways to model and account for these gains. For instance, one could imagine a simple cost function that can be run over a protocol message sequence chart to reveal potential DoS risks. In general, DoS experiments and models have been under-emphasized so far by both research and development communities.

## 3 End-system security

While end-system security is not the key to all security problems, it is *a* key to many. The execution of most observed DoS attacks have hinged on the ability to co-opt third-party end systems to execute the

attack as well as hiding the identity of the attacker.

While DoS attacks are still possible in the presence of ubiquitous end-system security, they would be much more limited. Further, many of the type of attacks we are seeing today would either be easily traceable, or require large-scale fraud to set up.

One common argument heard for not implementing adequate security on end systems, especially those of home users and small businesses, is that "*There is nothing important on this machine anyway.*" But, this argument is erroneous: network connectivity and processor time are important to a DoS attacker, regardless of what other resources the end system provides. Besides, in other areas of everyday life, such an attitude would be considered criminal negligence and could be prosecuted, should the end system be used in the perpetration of a crime.

But how can this negligence be proved without a metric of security? And how can vendors provide adequate security, when no objective definition exists?

A simple metric, often used by systems administrators, is to count the number of *vectors* for an attack. Due to the way vendors set their machines up—for ease of use—this number is often both high and hard to reduce without special knowledge about the specific product(s) involved.

A quick approximation of the vulnerability would be the number of services that can be contacted on a machine from the outside—the number of *open ports*. The, equally quick, fix is often to put machines behind a *filtering firewall* that only allow traffic to specific ports from the outside.

This, however, ignores the problems of viruses and worms that can piggyback on allowed services. Simple examples would be viruses attached to e.g. Word documents or emails proclaiming "Try this!" with a link to a Trojan Horse. While firewalls can be extended with inspection of all traffic, this at best creates a hard outer shell, that once penetrated, leave the machines on the inside soft targets.

# 4 System Administration

Investigating and introducing security metrics has benefits other than simply to advance the program of making cyber-security a science. Consider the job of system administrators. System administrators must choose to deploy specific security measures and choose between software packages. Security measures provide a certain degree of protection for a certain level of inconvenience. Software packages provide a set of features but bring with them certain security risks. Which to choose? How to balance the risks to security against the desirable features? The naive notion that security is a binary property ("A system is either secure, or it is not") makes it difficult to make such decisions in a principled way.

We start from the observation that no system is secure against all attacks and all adversaries. (Even if we could create technically foolproof systems, they would still be subject to "social engineering" attacks.) Therefore, we seek to quantify the *relative* security of systems and software, to allow system administrators to make informed tradeoffs between security and overhead on the one hand, and features and convenience on the other hand. How much security (and at what price) does a given piece of software provide?

We aim to find quantitative models that would make such decisions easier. Consider, for example, statistically-based models that quantitatively capture some rules of thumb relating salient properties of software packages. "Popular" programs are attractive to system administrators because of familiarity. The more popular a program is, the more users are likely to be already familiar with the interface. On the other hand, the more popular a program is, the more likely it is to be the target of an attack. Popularity, however, also increases the likelihood that bugs (security-related or not) will be discovered and fixed rapidly. Other similar and related tradeoffs exist. For example, programs with more useful features are often more popular. However, a large feature-set increases complexity and also increases the likelihood of bugs. Bugs decrease robustness, can be exploited as security holes, and reduce the attractiveness of a program. On the plus side, popular featureful software is more likely to be ported to a wide variety of hardware architectures and operating systems. Such diversity of platforms is good because it reduces the likelihood and magnitude of the worst-case failure scenarios; on the other hand diversity represents a maintenance and administrative headache. Large programs also may increase the size

of the Trusted Computing Base (TCB).

A moment's reflection reveals that the notion of popularity (some measure of "community size") must be carefully refined. In order to understand the impact of community size on the expected time for bug-discovery and repair, one must characterize community with finer granularity. Open source software, for example, is likely to have a community with a higher ratio of developers to users. Developers are more likely to fix bugs, so the mix of users and developers will affect the rate of bug-repair. Similarly, web servers may have many users, but configuration is mostly done by a small set of (trained? informed?) administrators. In contrast, web browsers that may allow remote applets to run are typically configured by each user. Administrators are ostensibly less likely to misconfigure a web server than users are likely to misconfigure a browser, so again the components of the community are relevant.

Such complex, qualitative, interrelated tradeoffs are not helpful without some way of weighing one attribute against another. The first step seems to be to develop crude quantitative models and gradually refine them as more data is collected. Many of these factors may seem hard to quantify. However, we are heartened by recent work by groups such as Engler's at Stanford [4, 2, 1], who have some simple models that quantify attributes such as rate of bugs and bug-fixes in succeeding versions of software such as operating systems.

## 5 Conclusion

Experience with computer systems has led to some general principles of computer security [6], but despite their importance, these principles do not yield any way to ascertain whether a system is secure. Our belief is that to make real progress, we must establish better experimental techniques, better metrics of security, and better models that have real predictive power; we must put security research on a foundation of science.

## References


[1] Ken Ashcraft and Dawson Engler. Using programmer-written compiler extensions to catch security holes. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 143–159, May 12-15 2002. Oakland, CA.

[2] Andy Chou, Jun-Feng Yang, Benjamin Chelf, Seth Hallem, and Dawson Engler. An empirical study of operating systems errors. In *Proceedings of 18th ACM Symposium on Operating Systems Principles (SOSP '01)*, pages 73–82, October 2001. Chateau Lake Louise, Banff, Alberta.

[3] D. Dolev and A. Yao. On the security of public key protocols. *IEEE Transactions on Information Theory*, 2(29), 1983.

[4] Dawson Engler, David Yu Chen, Seth Hallem, Andy Chou, and Benjamin Chelf. Bugs as deviant behavior: A general approach to inferring errors in systems code. In *Proceedings of 18th ACM Symposium on Operating Systems Principles (SOSP '01)*, pages 57–72, October 2001. Chateau Lake Louise, Banff, Alberta.

[5] Erland Jonsson and Tomas Olovsson. A quantitative model of the security intrusion process based on attacker behavior. *IEEE Transactions on Software Engineering*, 23(4), April 1997.

[6] J. H. Saltzer and M. D. Schroeder. The protection of information in computer systems. *Proceedings of the IEEE*, 63(9):1278–1308, September 1975.