

---

## Problem Set 1

Submit this problem set in PostScript, PDF, or MS Word format to `6857-submit@csail.mit.edu` before the due date. We have provided templates for  $\LaTeX$  and Microsoft Word on the course website. Each solution must appear on separate sheets of paper. Mark the top of each sheet with your name(s), the course number (6.857), the problem set number and question, and the date.

This problem set consists of both group and individual problems. You are to work in groups of three or four people and should submit a single set of solutions for all problems parts designated [**Group**]. You should turn in a separate, individual solution to any problems designated [**Individual**].

Be sure that all group members can explain the solutions. See the course information webpage for our policy on collaboration. If you do not have a group, seek partners by emailing `6857-tas@csail.mit.edu`. Referring to bibles or solutions from previous semesters is forbidden.

**Grading and Late Policy:** Each problem is worth 10 points. Late homework will not be accepted without prior approval. Homework should not be submitted by hand except with prior approval.

We may distribute our favorite solution to each problem as the “official” solution. If you do not wish for your homework to be used as an official solution, or if you wish that it only be used anonymously, please note this on your homework.

### Problem 1-1. PGP

This problem has both individual and group elements to it. Your group should turn in one write-up answering each of the parts labeled [**Group**], but all key pairs, emails, etc. should be created and sent individually.

Read Alma Whitten’s paper, “*Why Johnny Can’t Encrypt.*” (There is a link to it on the “Lectures and Handouts” page of the course website.)

Locate and install a fresh version of PGP or GPG. There are versions for Unix flavors, Windows, and the Macintosh. See <http://web.mit.edu/network/pgp.html> for downloads. If for some reason you are not able to download from this site, <http://www.pgpi.org/> may be of use.

Find the PGP public keys for as many of the 6.857 staff as you can. Part of your assignment is figuring out how to locate PGP keys. Searching the Internet for PGP key servers may be of help. But beware; there may be fake keys out there...

- (a) [**Group**] **Trust and Signing.** PGP’s “web of trust” model allows users to “sign” each others’ public keys. Suppose Alice signs Bob’s key; what, in effect, is Alice declaring when she does this? Why is it useful for people to sign each other’s keys? What precautions should one take before signing someone else’s key, and why are these measures appropriate? If Bob subsequently signs Charlie’s key, what (if anything) does Alice, or one who personally knows and trusts Alice, know about Charlie?
- (b) [**Individual**] **Getting started.** Create a new public/private key pair for yourself (you may use an existing key pair if you already have one). Sign each of your group members’ public keys, and have them sign yours.

When all of your group members have signed your public key, email it to `6857-submit@csail.mit.edu` in ASCII-armored format, with the subject `My public key`.

- (c) [**Individual**] **Encrypting email.** Send an encrypted, signed email to `6857-submit@csail.mit.edu` with the subject `PGP is fun`. Do *not* send the mail to the TAs individually. In the body of the message,
  - Tell us what operating system, mail client, and version of PGP you are using.
  - Show us the public keys you found for the 6.857 staff; PGP fingerprints are sufficient.

- Search for keys for `president@whitehouse.gov` on a PGP key server. Based on your findings, explain (in a few sentences each) one useful feature and one drawback of PGP key servers. (You do not need to include any key(s) you found.) Remember to cite all your sources.
- In a few sentences, explain why you do or do not believe that the keys you found for the 6.857 staff in fact belong to us. If you do not trust a public key, explain what would convince you otherwise.

Your mail should be protected with PGP such that the 6.857 TAs, and *only* the 6.857 TAs, can obtain the plaintext contents. You must also sign the mail with your private key. We will only accept your first message, so make sure to get it right the first time. Are you able to finish the assignment in fewer than 90 minutes as in Whitten’s experiment? Remember to cite all your sources (books, manuals, friends, etc.) according to the guidelines in the course information handout.

### Problem 1-2. Open Source Security Model [Group]

In this problem, you will compare how an open source or proprietary model affects the number of security bugs in a piece of software. To do so, you will use a very simple software development model.

In this model, you will consider a piece of software with a fixed number of lines of code  $L$ . Software developers replace code at a fixed rate of  $\rho$  lines per time unit. A fraction  $\delta$  of new lines will contain a defect, or bug. Each bug is assumed to be on a single line and represents a single security vulnerability. At a given time  $t$ , the value  $V(t)$  represents the number of vulnerabilities currently in the code. Replaced lines are randomly chosen and may contain bugs proportional to  $V(t)$ .

In the proprietary model, a team of debuggers (the “good guys”) will check lines of code at a fixed rate of  $\hat{\gamma}$  lines per time unit, and will fix all bugs present in those lines. Meanwhile, adversarial “bad guys” will check the code for security exploits at a rate of  $\hat{\beta}$  lines per time unit. Once an adversary exploits a bug, the bug will be fixed. Assume that adversaries and debuggers never simultaneously discover the same bug. The number of bugs that debuggers and adversaries find will be proportional to  $V(t)$ .

In the open source model, debuggers and adversaries can analyze code at fixed rates of  $\gamma$  and  $\beta$  lines per time unit, respectively. Since everyone has easier access to the source code, you may assume that  $\gamma > \hat{\gamma}$  and  $\beta > \hat{\beta}$ . You may also assume that  $V(0) = \delta L$ . To summarize the model:

$L$	=	Total lines of code.
$\rho$	=	Rate of line replacement.
$\delta$	=	Density of bugs in replaced line.
$V(t)$	=	Total number of vulnerabilities at time $t$ .
$\gamma$	=	Debugger inspection rate in the open source model.
$\hat{\gamma}$	=	Debugger inspection rate in the proprietary model .
$\beta$	=	Adversarial inspection rate in the open source model.
$\hat{\beta}$	=	Adversarial inspection rate in the proprietary model.

- What is the net change in bugs per time unit due to the replacement of lines of code? Your answer should be independent of debuggers or adversaries, and may depend on  $V(t)$ .
- How many security bugs can a proprietary adversary exploit per time unit? How many can an open source adversary exploit per time unit? Your answer may depend on  $V(t)$ .
- Find the number of vulnerabilities  $V^*$  that the system converges to in the open source model. In other words, find a value  $V^*$  such that if  $V(t) = V^*$ , then the expected value  $V(t + 1) = V^*$ .

- (d) Determine a relationship between the rates  $\rho, \gamma, \beta, \hat{\gamma}$ , and  $\hat{\beta}$  such that there are fewer expected security exploits in the open source than in the proprietary model. You may assume the number of vulnerabilities have converged to  $V^*$  and  $\hat{V}^*$ , respectively.
- (e) This model of software development is obviously very limited. Read Ross Anderson's "*Security in Open versus Closed Systems - The Dance of Boltzmann, Coase and Moore*", available on the course web page and online at: <http://www.cl.cam.ac.uk/ftp/users/rja14/toulouse.pdf>. Which aspects of software development does Anderson's model capture that this model does not? What aspects of software development and debugging do you think Anderson's model lacks?

### Problem 1-3. Vulnerability/Mechanism Chains [Group]

Consider some system  $S$  (such as a voting system) with an associated security policy  $P$  consisting of several parts  $(P_1, P_2, \dots, P_n)$ .

There may be a vulnerability such that  $P_i$  is violated by an adversary, who may be a participant. To address the violation, a new security mechanism is added to  $S$ . This new mechanism may be for *protection* against the risk that the vulnerability is exploited, or may be for *detection* that the vulnerability has actually been exploited (in which case some additional *recovery mechanism* may also be required, unless the detection mechanism is included only for its deterrence effect).

For example, there may be a vulnerability in a voting system where a voter casts more than one vote. So, the initial voting system is augmented by adding a protective mechanism  $M_1$  involving smart-cards:

**Mechanism 1** *The voter is given one smart card when she identifies herself at the polling site, and the voting terminal only allows each card to be used once. This may be implemented by erasing the card contents once the voter has voted.*

But the newly added protection, detection, or recovery mechanism  $M_1$  may also introduce new violations of the security policy: it may be defeated by an adversary so that  $M_1$  doesn't protect, detect, or recover as it should. So a new security mechanism  $M_2$  is added to help protect against, detect, or recover from attacks exploiting this new vulnerability in  $M_1$ .

For example, with  $M_1$  there may be a vulnerability that allows a dishonest voter to forge the appropriate smart cards (see the Kohno et. al paper, "*Analysis of an Electronic Voting System*", for example). So, an additional prevention mechanism  $M_2$  may be added to prevent forgery. (Note that this is intended to be a chain, so we are talking about prevention of card forgery now only, not prevention more generally of multiple voting.)

To prevent forgeries, we can add a public key cryptographic mechanism. Each voting terminal may be pre-loaded with list of public keys. These public keys will correspond to a secret key stored on each smart card that may be used at that polling site. A challenge-response mechanism is used at the voting terminal to verify that the card possesses the secret key corresponding to one of the authorized public keys. Once the card is used to vote, its corresponding public key is marked as "used" and cannot be used for other votes.

There may be new vulnerabilities with this countermeasure, e.g. a valid card may be still be duplicated and used on different machines. So a detection mechanism  $M_3$  is added comparing the list of used public keys at each machine at the end of the voting day, etc. In this way, chains of vulnerabilities and security mechanisms can be built up.

Read the Kohno et al. paper, or any other paper or web site about voting systems, and describe the longest such chain that you can that seems consistent with how voting is actually done today on some voting system. Don't bother to report a chain longer than eight. (By then you've got the point!)

Here are some potentially useful links:

<http://theory.lcs.mit.edu/~rivest/voting/index.html>  
<http://avirubin.com/vote/analysis/index.html>  
<http://www.csl.sri.com/users/neumann/book-voting.html>