Lecture 3

Design Principles for Security-conscious Systems

Administrivia

From time to time, we may discuss vulnerabilities in widely-deployed computer systems. This is not intended as an invitation to go exploit those vulnerabilities. It is important that we be able to discuss real-world experience candidly; students are expected to behave responsibly.

Berkeley's policy (and our policy) on this is clear: you may not break into machines that are not your own; you may not attempt to attack or subvert system security. Breaking into other people's systems is inappropriate; and the existence of a security hole is no excuse.

Economy of Mechanism

- Keep your implementation as simple as possible
  - Note that *simple* is different from *small*: just because you *can* write a CGI program in 300 bytes of line-noise Perl, doesn't mean you *should*
  - All the usual structured-programming tips help here: clean interfaces between modules, avoid global state, etc.
- Interactions are a nightmare
  - You often need to check how each pair of subsystems interacts, and possibly even each *subset* of subsystems
  - For example, interactions between the password checker and the page-fault mechanism
  - Complexity grows as $\Omega(n^2)$, possibly even $\Omega(2^n)$

Bellovin's Fundamental Theorem of Firewalls

Axiom 1 (Murphy) All programs are buggy.

Theorem 1 (Law of Large Programs) Large programs are even buggier than their size would indicate.

Corollary 1.1 A security-relevant program has security bugs.

Theorem 2 If you do not run a program, it does not matter whether or not it is buggy.

Corollary 2.1 If you do not run a program, it does not matter if it has security holes.

Theorem 3 Exposed machines should run as few programs as possible; the ones that are run should be as small as possible.

The sendmail wizard hole

- Memory segments: `text` (code), `data` (initialized variables), `bss` (variables not explicitly initialized), `heap` (`malloc`ed)
- Config file parsed, then a "frozen" version written out by dumping the `bss` and `heap` segments to a file
- Wizard mode feature allows extra access for remote debugging
- Wizard mode implementation:

```
int wizflag; // password enabled?
char *wizpw = NULL; // ptr to passwd
```

- Results:
  - In production mode, wizard mode enabled without a password.
  - But in development, the password was tested, and worked fine. . .

Credits: Bellovin.

The ftpd/tar hole

- To save network bandwidth, `ftpd` allows client to run `tar` on the ftp server.
- This was fine, until people started using GNU tar.
- Security hole:

```
quote site exec tar -c -v
--rsh-command=commandtorunasftp -f somebox:foo
foo
```

- Beware the wrath of feeping creaturism…

Fail-safe Defaults

- Start by denying all access, then allow only that which has been explicitly permitted
  - By doing this, oversights will usually show up as “false negatives” (i.e. someone who should have access is denied it); these will be reported quickly
  - The opposite policy leads to “false positives” (bad guys gain access when they shouldn’t); the bad guys don’t tend to report these types of problems
- In production and commercial systems, the configuration *as shipped* is what matters. This has often been done wrong in the past:
  - SunOS shipped with + in `/etc/hosts.equiv`
  - Irix shipped with `xhost +` by default

Canonicalization Problem

- If you try to specify what objects are restricted, you will almost certainly run in to the *canonicalization problem*.
- On most systems, there are many ways to name the same object; if you need to explicitly deny access to it, you need to be able to either
  - list them all, or
  - canonicalize any name for the object to a unique version to compare against

  Unfortunately, canonicalization is hard.
- For example, if I instruct my web server that files under `~iang/private` are to be restricted, what if someone references `~iang//private` or `~iang/./private` or `~daw/../iang/private`?
- Both the NT webserver and the CERN webservers have suffered from vulnerabilities along these lines.

Canonicalization Problem, cont.

- Better if you tag somehow tag the object *directly*, instead of by name
  - check a file's device and inode number, for example
  - or run the webserver as uid `web`, and only ensure that uid `web` only has read access to public files
  - the `.htaccess` mechanism accomplishes this by putting the ACL file *in* the directory it protects: the *name* of the directory is irrelevant
- Better still if you have a fail-safe default: explicity *allow* access to a particular name; everything else is denied
  - Attempts to access the object in a non-standard way will be denied, but that's usually OK

Complete Mediation

- Check *every* access to *every* object
- In rare cases, you can get away with less (caching)
  - but only if you're *sure* that nothing relevant in the environment has changed
  - and there's a lot that's relevant...
- Note that this is not the distinction between ACLs and capabilities: both allow for complete mediation
  - For ACLs, the check involves knowing this identity of the requestor and checking it against a list
  - For capabilities, the check is simply "does the requestor have a valid capability for this object"

Incomplete mediation in NFS

- NFS is *not* a good example of complete mediation
- NFS protocol: contact `mountd` to get a filehandle, use the filehandle for all reads/writes on that file
- Access to an exported directory is checked only at mount time by `mountd`
- If you can sniff or guess the filehandle, you don't have to contact `mountd` at all, and you can just access the files directly, with no checks

Imperfect bookkeeping in sendmail

- Sendmail treats program execution as an address; for security, it tries to restrict it to alias expansion.
- This requires perfect bookkeeping:
  at every place an address can appear, one must check to ensure that it isn't program delivery.
- But there are too many different places that addresses could appear.
- Inevitable results: a few places where the check was forgotten, which has led to several security holes.

Credits: Bellovin.

Mediation in Java

- Access control in Java libraries is done like this:

```
public boolean mkdir() {
   SecurityManager security = System.getSecurityManager();
   if (security != null)
      security.checkWrite(path);
   return mkdir0(); // the real mkdir
}
```

- But forgetting just one such check leaves the access control wide open
- And there are 70 such calls in JDK1.1; what are the odds the developers forgot one?
- Just for kicks: a fun comment from `net/DatagramSocket.java`:

```
// The reason you want to synchronize on datagram packet
// is because you dont want an applet to change the address
// while you are trying to send the packet for example
// after the security check but before the send.
```

- Conclusion: it is not easy to convince oneself that Java exhibits complete mediation

Separation of Privilege

- Require more than one check before granting access to an object
  - A single check may fail, or be subverted. The more checks, the harder this should be
  - Something you know, something you have, something you are
  - e.g. Kerberos checks both your ticket and your IP address
  - e.g. Airport security checks both the shape of your hand and a PIN
- Require that more than one principal “sign off” on an attempted access before granting it
  - This is easy to do with cryptography: secret sharing can *mathematically* provide that a capability is released only when k out of n, for example, agree.

Anonymous Remailers

- Anonymous remailers allow people to send email while hiding the originating address
- They work by a process known as *chaining*: imagine the message is placed in a series of nested envelopes, each addressed to one of the remailers in the world
- Each remailer can open only his own envelope (cryptography is used here)
- Each remailer opens his envelope, and sends the contents to the addressee; he does not know where it's going after that, or where it came from before it got to him
- In order to trace a message, *all* the remailers in the chain need to cooperate

Least Privilege

- Figure out exactly what capabilities a program requires in order to run, and grant exactly those
- This is not always easy, but by starting with granting none, and seeing where errors occur, you can usually do OK
  - Watch out for unusual cases!
- This is the principle used to design policy for sandboxes (e.g. Janus)
- The Unix concept of `root` is *not* a good example of this
  - Some programs need to run as `root` just to get one small privilege, such as binding to a low-numbered port
  - This leaves them susceptible to buffer-overflow exploits that have complete run of the machine

Tractorbeaming wu-ftpd

- `wu-ftpd` tries to run with least privilege, but occasionally elevates its privilege level with

```
seteuid(0);
// privileged critical section goes here...
seteuid(getuid());
```

- However, `wu-ftpd` does not disable signals.
- Thus, when it is running in a critical section, it can be “tractorbeamed” away to a signal handler not expecting to be run with root privileges.
- Moreover, remote ftp users can cause `wu-ftpd` to receive a signal just by aborting a file transfer.
- Result: if you win a race condition, `wu-ftpd` never reliquinshes `root` privileges, and you get unrestricted access to all files.
- Conclusion: uid/euid not a robust mechanism for implementing least privilege.

Credits: Wietse Venema.

Sandboxes and code confinement

- Least privilege is the whole motivation behind the use of sandboxes to confine partially-untrusted code.
- Example: sendmail
  - Once sendmail is broken into, intruder gains root access, and the game is over.
  - Better would be for sendmail to run in a limited execution domain with access only to the mail subsystem.
- Example: Netscape plugins
  - Netscape plugins run in the browser's address space, with no protection.
  - At one point, a bug in the popular Shockwave plugin could be used by malicious webmasters to read your email, by abusing `mailbox:`-style URLs.

Least Common Mechanism

- Be careful with shared code
  - The assumptions originally made may no longer be valid
- LiveConnect: allows Java and Javascript and the browser to talk to each other
  - But Java and Javascript have different ways to get at the same information, and also different security policies
  - A malicious Java applet could cooperate with a malicious Javascript page to communicate information neither could have communicated alone
- Some C library routines (and the C runtime) have excess features that lead to security holes

Eudora and Windows

- Windows exports an easy interface to IE's HTML-rendering code
- Eudora, among other programs, uses this interface to display, for example, HTML-formatted email
- By default, parsing of Java and Javascript (J-Script) are *enabled*
- However, the HTML-rendering code "knows" that Java and J-Script are unsafe when loaded from the Internet, but safe when loaded from local disk
- But the email *is* loaded from local disk!
- Oops...
- Enabling Java and J-Script by default in the common HTML-rendering code violated the Principle of Least Common Mechanism

Psychological Acceptability

- Very important for your users to buy into the security model.
  - If you force users to change their password every week, very soon most of them will simply write their password on a yellow sticky note attached to their monitor.
  - If users think your firewall is too restrictive, they'll hook up a modem to their machine so they can dial in from home, and you're hosed.
  - Never underestimate the ingenuity of engineers at bypassing obstacles that prevent them from getting work done!
  - Also important that the management buys into security.
    (Proof by reading Dilbert.)
- And the user interface to security mechanisms should be in an intuitively understandable form.
  - NSA crypto gear stores keying material on a physical token in the shape of a key. To enable a ciphering device, you insert the key and turn it.

Work Factor

- Work factor: an attempt to quantify the cost of breaking system security.
- Work factor issues are increasingly important today:
  - More and more “script kiddies”
  - And with `www.rootshell.com` etc., discovery of a security hole is likely to lead to widespread exploitation with days
  - So you should concentrate on increasing the cost of exploiting bugs, rather than focusing on the cost of discovering bugs
- One important distinguishing feature of crypto is the relative ease which which you can put concrete numbers on the required work factor (in terms of computational complexity).

Work factor, cont.

- Remember, *security is economics*.
  - You can always improve site security with an increased investment, but you need to decide when such expenditures are economically sensible.
  - Don't buy a $10,000 firewall to protect $1000 worth of trade secrets.

Compromise Recording

- Compromise recording: if you can't prevent breakins, in some cases it is enough to just detect them after-the-fact.
- This is the whole idea behind modern-day intrusion detection systems (network "burglar alarms").
- And just saving audit logs can be useful, even if you don't have an automated intrusion detection system.
  - If you discover an intruder has broken into one CS machine via the `rpc.statd` hole (say), you might like to know how many other machines he broke into.

Compromise Recoding, cont.

- An important principle in hardware tamper resistance, e.g. FIPS 140-1 standard:
  - Type II device is tamper-evident
  - Type III device is tamper-resistant (but more expensive)
- Example: casinos often don't bother looking for fraud unless their daily take differs significantly from expectations.
  - Principle: you don't care about fraud if it doesn't affect your bottom line enough to notice.