Program Security II

Tom Kelliher, CS 325

Oct. 6, 2006

1 Administrivia

Announcements

Projects due Monday.

Assignment

Read 4.1–4.3.

From Last Time

Program security I.

Outline

1. Targeted malicious code.
2. Controls against program threats.

Coming Up

Operating systems.

2 Targeted Malicious Code

What do we mean? Malicious code written for a particular system, a particular application, and a particular purpose.

2.1 Trapdoors

1. An undocumented entry point to a module. Exploitable.

2. Example in a Perl CGI script I wrote and used for five+ years until a CS 102 student inadvertently tickled the trapdoor:

```
#!/usr/bin/perl -Tw

# Copyright 2001, Thomas P. Kelliher, Goucher College.


use strict;
use CGI qw(:standard);


##############################################################################
# Globals.

# Path to mail client.
my $MAILPATH = "/usr/bin/Mail";


##############################################################################
MAIN:
{
   ...

   # Send the form data as an e-mail.

   if ($sender eq "")
   {
       open(MAIL, "|$MAILPATH -s \"$subj\" $recip");
   }
   else
   {
```

```
        open(MAIL, "|$MAILPATH -s \"$subj\" -r \"$sender\""
           . " $recip");
    }

    for ($i = 0; $i < $numFlds; ++$i)
    {
       if ($fldn[$i] ne "")
       {
          print MAIL "$fldn[$i]:\n";
          print MAIL "$fldv[$i]\n";
          print MAIL "\n-----------------------------------";
          print MAIL "-----------------------------------\n\n";
       }
    }

    close(MAIL);

    ...

    exit(0);
}
```

3. Trapdoors in PHP.

   `open_basedir` — Default "jail" directory.

   Trying to prevent jail breaks: `disable_functions` (`shell_exec`, `system`, etc.).

4. Causes:

   (a) An intended functionality used in an unintended way (Mail).

   DEBUG mode in sendmail.

   (b) A design that does not consider consequences (PHP).

   (c) Forgot to remove, left in for testing, left in for maintenance, left in for covert access.

2.2 Salami Attacks

1. Merging inconsequential bits of data to end up with a substantial, meaningful result.

2. Classic example: Collecting rounded-off currency calculations into a specified account.

   No one notices!!!

3. Realistic (?) example: Allocating sections of memory or disk and sifting through the old data. (Or is this "dumpster diving?")

2.3 Covert Channels

1. A low bandwidth, unnoticed communication channel that allows the leakage of secure information from a (secure) program with access to a program or person not supposed to have access.

2. "Low" is relative — often just a bit at a time, but the signaling rate can be high.

3. Textbook example: Modifying aspects of a printed report.

2.3.1 Storage Channels

1. Basic idea: contention on a shared resource.

2. Pass a bit or more of information by presence/absence of certain files, lock status of a file, presence/absence of certain objects in an object-oriented database, locks on rows of a table, availability of certain ports, availability of disk space, etc.

3. How do we synchronize the communicating processes?

4. Interference from other processes?

2.3.2 Timing Channels

1. Here, the shared resource is time.

2. Communicate by measuring the rate of computation, or whether or not computation is occurring

3. The presence of other processes complicates matters, but not too much.

2.3.3 Related Cryptography Attack

Side-channel attacks:

1. Timing, power usage, memory access pattern (cache), electromagnetic leaks, sound, etc.

2. Require intimate knowledge of crypto system implementation details.

3. Extremely time consuming, requiring lots of plaintext results.

2.3.4 Identifying Covert Channels

1. Shared resource matrix: resources in rows; processes in columns.

	Secure	Spy
`/etc/shadow`	R	—
`/tmp/scratch`	RW	R

   Possible covert threat — theoretically, spy has access to `/etc/shadow`.

   According to this, any Linux process running as root could be a covert channel.

2. Information flow analysis. Can be automated by compiler.

   How useful in analyzing inter-process communication?

Uncovering covert channels is hard!

3 Controls Against Program Threats

3.1 Developmental

1. Follow a robust, methodical software engineering paradigm.

2. Code inspections find more faults per hour than running systems, white box testing, and black box testing.

3.2 Operating System

1. Useful when developmental controls can't be applied.
2. Use of trusted software as foundation for untrusted software.
3. Characteristics of trusted software:
   (a) Functional correctness.
   (b) Enforcement of integrity (properly handles garbage inputs).
   (c) Limited privilege (privilege not passed along).
   (d) Appropriate confidence level.
4. Mutual suspicion.
5. Confinement — Java's sandbox.
6. Access logs. "After-the-fact" analysis.

```
Oct  3 18:32:16 phoenix su(pam_unix)[32298]: authentication failure;
logname=ckonradi uid=509 euid=0 tty=pts/3
ruser=ckonradi rhost=  user=root
Oct  3 18:47:52 phoenix last message repeated 12 times
```

3.3 Administrative

1. Specification of the software engineering process.
2. Separation of duties.