Protection and Security

Jingtao Wang

cs162-tb@imail.eecs

4/22/2009

Access Enforcement

- Some part of the system must make sure the only authorized actions take place
  - Enforcer checks passwords, ACLs, etc
  - Bugs in enforcer⇒things for malicious users to exploit
- In UNIX, superuser can do anything
  - Because of coarse-grained access control, lots of stuff has to run as superuser in order to work
  - If there is a bug in any one of these programs, you lose!

Access Enforcement - Continue

- Paradox
  - Bullet-proof enforcer
    - Only known way is to make enforcer as small as possible
    - Easier to make correct, but simple-minded protection model
  - Fancy protection
    - Tries to adhere to principle of least privilege
    - Really hard to get right

State of the World

- Authentication: Encryption
  - But almost no one encrypts or has public key identity
- Authorization: Access Control
  - But many systems only provide very coarse-grained access
  - In UNIX, need to turn off protection to enable sharing
- Enforcement: Kernel mode
  - Hard to write a million line program without bugs
  - Any bug is a potential security loophole!

Challenges to Access Enforcement

- Abuse of valid privileges
  - A super-user in Unix can do anything
- Imposter or Trojan Horse
- Listener
  - Eavesdrop on terminal wire, or listen in on local network traffic
- Spoiler
  - Use up all resources and make system crash
- Create doctored version of some standard program

Examples of Penetration

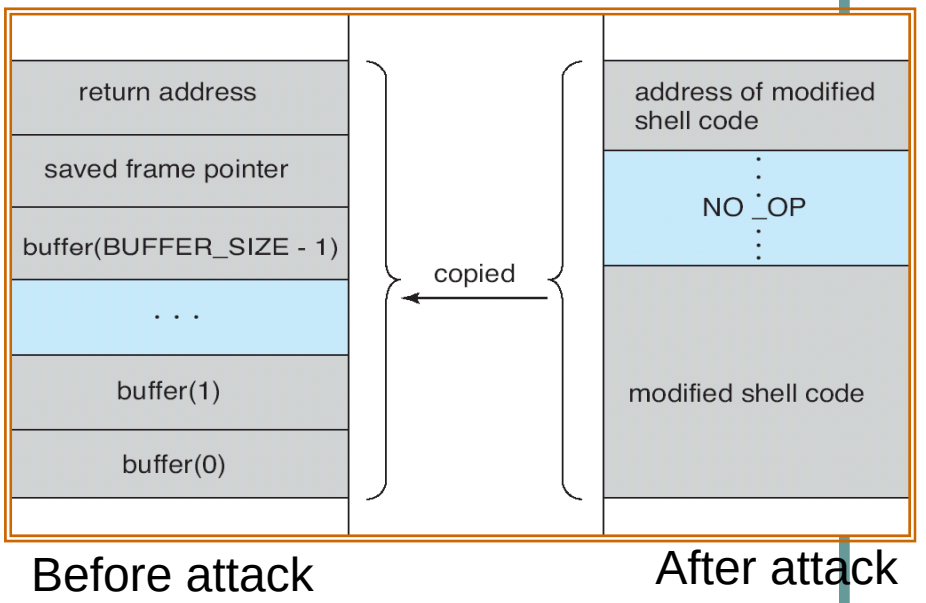
- Permission on lists of /dev files will lead to access to raw I/O devices
- Users leaves fake shell on terminal
- Email based Phishing
- Walk up to terminal that is still logged on
- Find Account with null password
- Fake distributions – distribute a version of the software with doctored code

Examples of Penetration

- Create a fake file system and have the system mount it. Can put a program there “owned” by the superuser, with setuid bit set. User runs program and becomes superuser.
- Buffer Overflow – many systems are vulnerable to argument buffers overflowing.

Security Problems: Buffer-overflow Condition

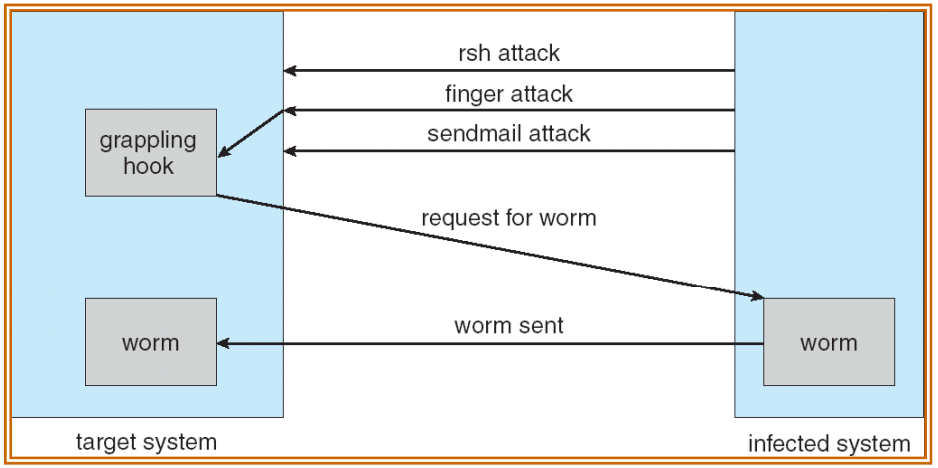
```
#define BUFFER SIZE 256
int process(int argc,
            char *argv[])
{
   char buffer[BUFFER SIZE];
   if (argc < 2)
       return -1;
   else {
       strcpy(buffer,argv[1]);
       return 0;
   }
}
```



- Technique exploited by many network attacks
  - Anytime input comes from network request and is not checked for size
  - Allows execution of code with same privileges as running program – but happens without any action from user!

The Morris Internet Worm

- Internet worm (Self-reproducing)
  - Author Robert Morris, a first-year Cornell grad student
  - Launched close of Workday on November 2, 1988
  - Within a few hours of release, it consumed resources to the point of bringing down infected machines



- Techniques
  - Exploited UNIX networking features (remote access)
  - Bugs in *finger* (buffer overflow) and *sendmail* programs (debug mode allowed remote login)
  - Dictionary lookup-based password cracking
  - Grappling hook program uploaded main worm program

Timing Attacks: Tenex Password Checking

- Tenex – early 70's, BBN
  - Most popular system at universities before UNIX
  - Thought to be very secure, gave "red team" all the source code and documentation (want code to be publicly available, as in UNIX)
  - In 48 hours, they figured out how to get every password in the system
- Here's the code for the password check:

```
for (i = 0; i < 8; i++)
  if (userPasswd[i] != realPasswd[i])
    go to error
```

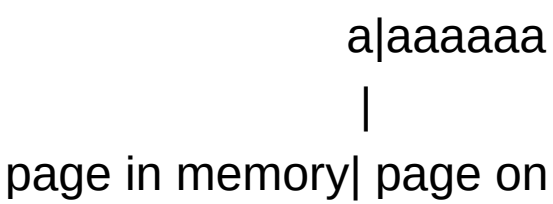
- How many combinations of passwords?
  - 256^8?
  - Wrong!

How to Prevent Buffer Overflow?

- Use a type safe language such as Java/C#/Python
- Use static source code scanner to check existing code
- Make stack not executable
- Implement some kind of dynamic stack-validity checking algorithm

Defeating Password Checking

- Tenex used VM, and it interacts badly with the above code
  - Key idea: force page faults at inopportune times to break passwords quickly
- Arrange 1st char in string to be last char in pg, rest on next pg
  - Then arrange for pg with 1st char to be in memory, and rest to be on disk (e.g., ref lots of other pgs, then ref 1st page)



- Time password check to determine if first character is correct!
  - If fast, 1st char is wrong
  - If slow, 1st char is right, pg fault, one of the others wrong
  - So try all first characters, until one is slow
  - Repeat with first two characters in memory, rest on disk
- Only 256 * 8 attempts to crack passwords
  - Fix is easy, don’t stop until you look at all the characters

Consequences of System Break-in

- Once the system has been penetrated, it may be impossible to secure it again
- It’s not always possible to tell when the system has been penetrated, since the villain can clean up all traces behind himself
- If we can never be sure that there are no bugs, then we can never be sure that the system is secure, since bugs could provide loopholes in the protection mechanisms.

Countermeasures

- Logging
- Get humans involved at key steps
- Principle of minimum privilege
- Correctness proofs
- Callback used to avoid abuse of accounts
- Consistency or plausibility check
  - E.g. is this user spending $10,000 when his largest previous purchase was $100?

Inference Control

- The goal – allowing users to be able to get statistical information (e.g. average) out of a database, but not get individual data
- The problem – can design sets of queries that will generate individual information
  - Average salary of all X
  - Average salary of X - delta
  - Size of X

Inference Control

- No good solution to this problem, can do some things
  - Randomize data (slightly) – i.e. introduce small errors
  - Permit only queries on predefined groups – e.g. zip codes

The Confinement Problem

- Problem: Mutually suspicious customer and service – want to insure that the service can only reach information provided by the customer, and that the service is protected from the customer
  - Idea is concept of information utility. Idea currently resurfacing as server/web based software
- Two problems remain
  - Service may not perform as advertised
  - Service may leak – i.e. transmit confidential data

List of Possible Leaks

- If the service has memory, it can collect data
- The service can send a message to a process controlled by its owner
- The information can be encoded in the bill rendered for service
- If the file system has interlocks, the service can lock and unlock a file, and the spy can watch to see if the file is locked, can use like morse code
- The service can vary the paging rate (which affects performance)

Viruses

- Computers transfer around executable files and code, e.g. in email.
- User executes this code, and bad things happen
  - Virus usually replicates itself elsewhere
  - And does something unpleasant to your machine

General Anti-Virus Techniques

- Search for known viruses by looking for their object code
  - Problem is that viruses encrypt themselves
    - Solution is to search for decryption code
  - Virus may change the decryption code
    - Solution is to interpretively execute the suspected virus code for some portion of time, to see if the code decrypts itself into something that is recognized as common virus.
  - There is no good defense against an unknown virus, since the code patterns can’t be recognized