CSE 451: Operating Systems
Spring 2005

Module 22
Security

Outline

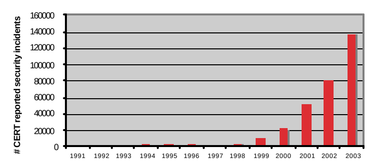
- “Classic” security topics
  - goal: safe sharing
  - general principles
  - Trusted Computing Base (TCB)
- Contemporary security problems
  - worms
  - spyware

Safe sharing

- Protecting a non-networked PC with one user is easy
  - Nobody can access the data on your computer
  - Nobody can install new code
  - Nobody can attack you over the network
- Sharing resources safely is hard
  - Prevent some users from reading private data
    - yet allow authorized users to access it
    - e.g., grades, keystrokes
  - Prevent some users from using too many resources
    - e.g., disk space
  - Prevent users from interfering with others’ programs
    - spoofing displays, replacing programs with malicious code, killing off processes …

Much of security is art, not science

- Difficult to “prove” a system secure
- Security is based on principles and best practices
  - experience reveals commonly occurring types of flaws
  - but clearly we need to do better …



Principle of Least Privilege

- Figure out exactly which capabilities a program needs to run, and grant it only those
  - start out by granting none
    - run program, and see where it breaks
    - add new privileges as needed.
- Unix: concept of root is not a good example of this
  - some programs need root just to get a small privilege
    - e.g., FTP daemon requires root:
      - to listen on network port < 1024
      - to change between user identities after authentication
    - but root also lets you read any file in filesystem

Principle of Complete Mediation

- Check **every** access to **every** object
  - in rare cases, can get away with less (caching)
    - but only if sure nothing relevant in environment has changed…and there is a lot that’s relevant!
- A TLB caches access control information
  - page table entry protection bits
  - is this a violation of the principle?

"Security through Obscurity" = bad

- Security through obscurity
  - "gain security" by hiding system implementation details
  - should be secure even if implementation is open!
    - in fact, publishing makes it more secure, since people can scour implementation and find/fix flaws
  - rely on mathematics and sound design to keep secure
- Counterexample: GSM cell phones
  - GSM committee designed own crypto algorithm, but hid it
    - "impossible to clone"
  - social + reverse engineering revealed the algorithm
    - it turned out to be very weak
    - could play "20 questions" with identity chip and learn its secret key in a few hours

Trusted Computing Base (TCB)

- Think carefully about what you are trusting with your information
  - if you type your password on a keyboard, you're trusting:
    - the keyboard manufacturer
    - your computer manufacturer
    - your operating system
      - including the keyboard device driver
    - the password library
    - the application that 's checking the password
  - what about the compiler that compiled all of this software (!!)
- TCB = set of components (hardware, software, wetware) that you must trust to preserve your secrets
  - should be as small as possible
    - public web kiosks should *not* be in your TCB
    - how about your web browser?

Cryptography

- Mathematics to secure data in a digital lockbox
  - ciphertext = **f(**key1, plaintext**)**
  - plaintext = **g(**key2, ciphertext**)**
  - hard to convert between ciphertext/plaintext without keys
- Preserve secrecy, integrity, authenticity of data
  - encrypt messages before sending them over Internet
  - encrypt files before storing on hard drive
  - makes it difficult for intermediaries to:
    - learn plaintext by sniffing messages
    - change plaintext undetectably
    - spoof fake messages

Modern security problems

- Internet experiencing a plague of attacks
  - *remote exploits:* attackers breaking into your system
  - *worms:* self-replicating attack code
  - *spyware:* software that tries to steal information from you
  - *phishing attacks:* web sites spoofing other web sites
- Underlying issues
  - most of our code is buggy
  - the Internet was designed to be "open"
    - easy to build new services, but easy to find/attack victims
  - understanding security is hard
    - haven't found simple conceptual models or usable UIs
    - e.g., what does the lock icon in IE really mean?

Worms 101

- Pseudocode for a simple worm

```
for (i = 0.0.0.0;  i < 255.255.255.255;  i++) {
   open network connection to 'i";
   if succeed {
     try to exploit vulnerability x on "i";
     if succeed {
       send code for self to victim and run it;
     }
     close connection to "i";
   }
}
```

- Will this worm propagate?
  - how quickly?

A "better" worm

```
while (1) {
   open network connection to random IP address 'i";
   if succeed {
      try to exploit vulnerability x on "i";
      if succeed {
        send code for self to victim and run it;
      }
      close connection to "i";
   }
}
```

- Why is this "better"?
- How quickly will this propagate?
- How can you do even better?

Random scanning worms

- Simple "random constant spread" growth model
  - population size = N =~ 2^32
    - # susceptible hosts = S(t)
    - # infected hosts = I(t);   I(t) + S(t) = S(0)
    - scan rate of infected host = B scans/second
  - simple differential equation solving leads to:
    - dI(t) / dt = I(t) x   B x   [S(t) / N]

S(t) / S(0)

QuickTime™ and a TIFF (LZW) decompressor are needed to see this picture.

- Exponential growth!
  - until population saturates
  - Code Red worms followed this quite closely

Local scanning worms

- A major poblem with random scan…
  - IP address space is non-uniformly populated
  - large swaths of empty IP space, but some dense regions
- Idea: scan nearby IP addresses preferentially
  - victim 128.95.4.1
    - with probability 37.5%, scan 128.95.X.Y
    - with probability 50%, scan 128.X.Y.Z
    - with probability 12.5%, scan X.Y.Z.W
- Code Red v2 used this technique
  - doubled in size every 37 minutes
  - took ~12 hours to saturate susceptible population
    - 1/2 million IIS web servers affected

Multi-vector worms

- Probe for many vulnerabilities, not just one
  - increases size of susceptible population
- Nimda worm used this approach
  - probed multiple IIS vulnerabilities
    - and left attack code in HTML on compromised IIS servers
  - bulk emailed itself
  - looked for backdoor left by Code Red v2 worm (!!)
- No good data on Nimda propagation speed
  - less than an hour to reach saturation

Faster scan rate

- Increased scan rate ==> faster spread
  - Code Red: approximately 5 scans per second
  - Sapphire worm: approximately 4000 scans per second
- Sapphire
  - attacked SQL server vulnerability
  - fit the probe + propagation in a single 376 byte UDP packet
    - very quick to send, no connection establishment timeouts
- Spread data
  - worm doubled in size every 8.5 seconds
  - saturated susceptible population of ~75,000 hosts in about 5-10 minutes (!!)

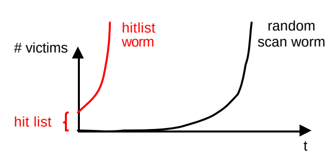
Sapphire fallout

- It propagated too fast for its own good!
  - no per-host damage
  - but massively clogged Internet backbones with scans
  - self-interference slowed its propagation rate

QuickTime™ and a TIFF (LZW) decompressor are needed to see this picture.

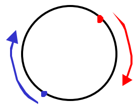
Theoretically possible worms

- Hit list scanning worm
  - gather large list of susceptible machines in advance
  - initial victim scans this hit list, then does random scan
  - upon propagation, partition hitlist across new victims
- Very quick spread through hitlist
  - avoids initial rampup in exponential curve



Theoretically possible (2)

- Permutation scanning worm
  - rather than true random scan, all worms pick same random ordering of IP address space
    - worm picks a random starting point in ordering



  - if scan reveals target is already a worm, jump to new starting point
  - avoids duplicating work, speeds up the “end game”

What’s the worst-case scenario?

- Flash worm
  - build a hitlist with all possible susceptible victims
    - possible to do with slow scanning in advance
- How fast would it spread?

What’s the worst-case scenario?

- Flash worm
  - build a hitlist with all possible susceptible victims
    - possible to do with slow scanning in advance
- How fast would it spread?
  - hard to predict precisely
    - estimate: saturate millions of susceptible hosts in 1-5 seconds
  - how do you defend against this??

Coping with worms

- Two basic approaches
  - prevention
    - avoid vulnerabilities that lead to worms
  - detection & filtering
    - notice a new worm is spreading
    - devise a filter that blocks it
    - install the filter in enough places around world to block it

Prevention

- Prevention techniques
  - turn off servers on home machines
  - use firewalls and NAT proxies on home machines
  - write less buggy code
- “Sneaky worm”
  - you don't need to run a server to be affected
    - compromised web server attacks web clients
    - compromised web client attacks web servers

Detecting worms

- How do you detect new worms?
  - approach 1: listen for increasing probe rate
    - worms knock on your door as they spread
      - an average of one probe every 5-7 minutes now
    - if probe rate grows anomalously high, must be a new worm
  - approach 2: look for probes with repeated content
    - derive “signature” based on repeated strings
    - called “content sifting”
    - much faster and more accurate

Filtering worms

- How widely must you filter?
  - turns out must cover most of the Internet "junctions"
  - Internet is well-connected by design
    - worm wriggles through nooks and crannies
- Major problem!
  - pushing filters out faster than worm spreads requires something that looks a lot like a worm!

Spyware

- Software that is installed that collects information and reports it to third party
  - key logger, adware, browser hijacker, …
- Installed one of two ways
  - piggybacked on software you choose to download
  - "drive-by" download
    - your web browser has vulnerabilities
    - web server can exploit by sending you bad web content
- Estimates
  - majority (50-90%) of Internet-connected PCs have it
  - 1 in 8 executables on the Web have it
  - 2% of Web pages attack you with drive-by-download

Wrap-up

- Security is hard
  - fundamentally an adversarial, escalating game
  - we're getting better, but so are the "bad guys"
- Complex systems are insecure
  - OS software one of the most complex artifacts of humankind
  - no surprise it has flaws!
- Current trends
  - reduce TCB to exclude OS
  - develop stronger sandboxes to contain flaws
    - virtual machine software (e.g., Vmware)
  - program with safer languages than C

Principle of Fail-Safe Defaults

- Policy should list what is allowed, not what is denied
  - security configuration should deny all access by default
  - allow only that which has been explicitly permitted
  - oversights show up as "false negatives"
    - users will quickly complain
- Opposite approach leads to "false positives"
  - the bad guys usually don't report this kind of failure…
- Counterexample: Irix OS
  - shipped with "xhost +" by default
    - Allows the world to open windows on your screen and grab the keystrokes you type

