CS162
Operating Systems and Systems Programming
Lecture 21

Security (II)

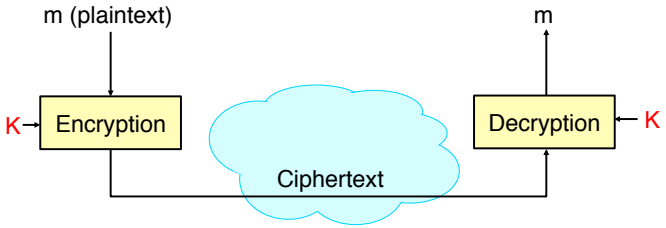
April 13, 2011
Ion Stoica
http://inst.eecs.berkeley.edu/~cs162

Recap: Security Requirements in Distributed Systems

- Authentication
  - Ensures that a user is who is claiming to be
- Data integrity
  - Ensure that data is not changed from source to destination or after being written on a storage device
- Confidentiality
  - Ensures that data is read only by authorized users
- Non-repudiation
  - Sender/client can't later claim didn't send/write data
  - Receiver/server can't claim didn't receive/write data

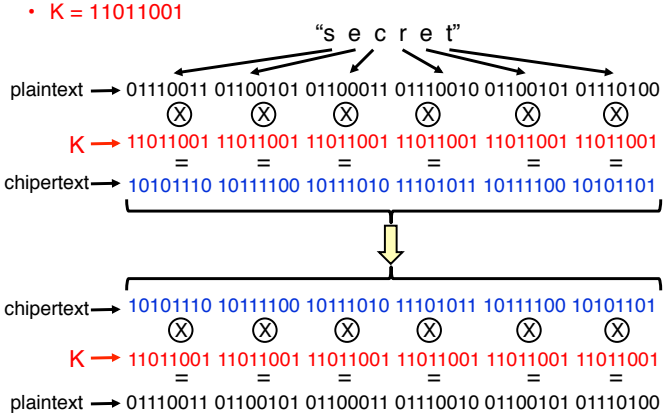
Recap: Confidentiality: Symmetric Key Cryptography

- K, secret key shared by both sender and recipient
- Assumption: to decrypt chipertext you need K



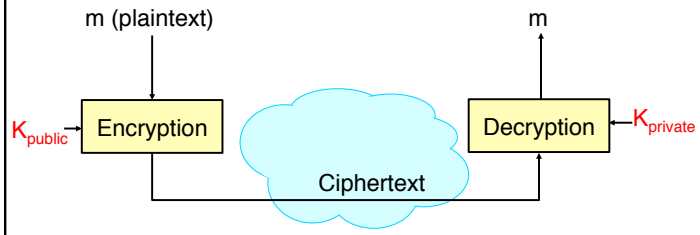
Recap: (Trivial) Example

- Use XOR for encryption/decryption
- K = 11011001



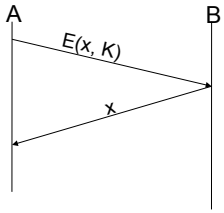
Recap: Confidentiality: Public Key Cryptography

- Two keys
  - K_{public}: public key, known by everyone
    $K_{private}$: private key, known only by recipient
- Cannot infer private key by knowing public key
- No need to securely distribute key, but slower than symmetric key cryptography



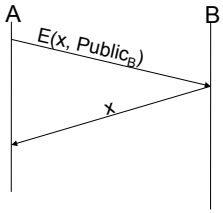
Recap: Authentication via Secret Key

- Main idea: entity proves identity by decrypting a secret encrypted with its own key
- K – secret key share only by A and B
- A can asks B to authenticate itself by decrypting a nonce, i.e., random value, x
  - Ignore man-in-the middle attack



Recap: Authentication via Public Key

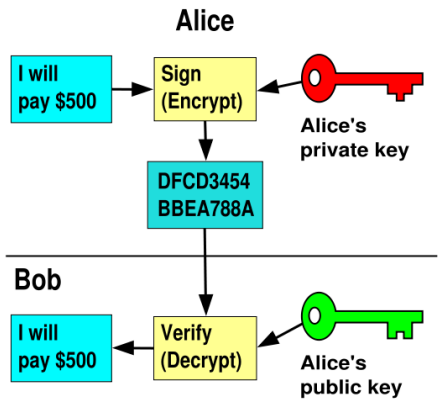
- Main idea: entity proves identity by decrypting a secret encrypted with its public key
- A asks B to authenticate itself by decrypting a nonce x that A has encrypted with B's public key



Recap: Non-Repudiation

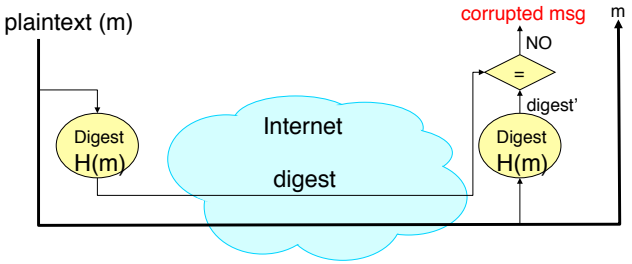
- Sender cannot deny she has sent a message
- Idea:
  - Sender signs (encrypts) message with its private key
  - Anyone can use sender's public key to check that sender has signed the message
    - » A message encrypted with private key can be decrypted by public key
  - Once sender signs the message, cannot deny it!

Recap: Example



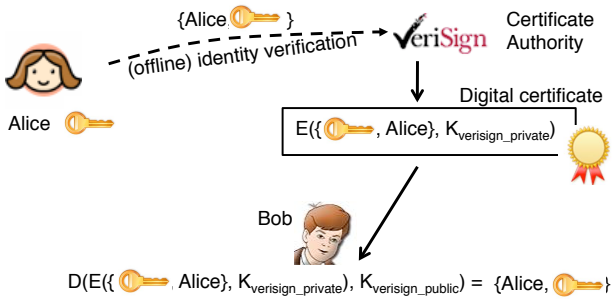
Recap: Integrity

- Compute a hash (i.e., digest) on the message
- Hash cannot be easily inverted
  - Very difficult for someone to modify the message without modifying the digest



Digital Certificates

- How do you know  is Alice's public key?
- Main idea: trusted authority signing binding between Alice and its private key



This Lecture

- More authentication
- Host Compromise
  - Attacker gains control of a host
- Denial-of-Service
  - Attacker prevents legitimate users from gaining service
- Attack can be both
  - E.g., host compromise that provides resources for denial-of-service

Authentication: Passwords

- Shared secret between two parties
- Since only user knows password, someone types correct password ⇒ must be user typing it
- Very common technique
- System must keep copy of secret to check against passwords
  - What if malicious user gains access to list of passwords?
    - » Need to obscure information somehow
  - Mechanism: utilize a transformation that is difficult to reverse without the right key (e.g. encryption)

Passwords: Secrecy

- Example: UNIX /etc/passwd file
  - passwd→one way transform(hash)→encrypted passwd
  - System stores only encrypted version, so OK even if someone reads the file!
  - When you type in your password, system compares encrypted version

Passwords: How easy to guess?

- Ways of Compromising Passwords
  - Password Guessing:
    - » Often people use obvious information like birthday, favorite color, girlfriend's name, etc…
    - » Trivia question 1: what is the most popular password?
    - » Trivia question 2: what is the next most popular password?
    - » Answer: http://www.nytimes.com/2010/01/21/technology/21password.html
  - Dictionary Attack:
    - » Work way through dictionary and compare encrypted version of dictionary words with entries in /etc/passwd
  - Dumpster Diving:
    - » Find pieces of paper with passwords written on them
    - » (Also used to get social-security numbers, etc)

Passwords: How easy to guess? (cont'd)

- Paradox:
  - Short passwords are easy to crack
  - Long ones, people write down!
- Technology means we have to use longer passwords
  - UNIX initially required lowercase, 5-letter passwords: total of 26^5=10million passwords
    - » In 1975, 10ms to check a password→1 day to crack
    - » In 2005, .01μs to check a password→0.1 seconds to crack
  - Takes less time to check for all words in the dictionary!

Passwords: Making harder to crack

- How can we make passwords harder to crack?
  - Can't make it impossible, but can help
- Technique 1: Extend everyone's password with a unique number (stored in password file)
  - Called "salt". UNIX uses 12-bit "salt", making dictionary attacks 4096 times harder
  - Without salt, would be possible to pre-compute all the words in the dictionary hashed with the UNIX algorithm: would make comparing with /etc/passwd easy!
- Technique 2: Require more complex passwords
  - Make people use at least 8-character passwords with upper-case, lower-case, and numbers
    - » 70^8=6x10^{14}=6million seconds=69 days@0.01µs/check
  - Unfortunately, people still pick common patterns
    - » e.g. Capitalize first letter of common word, add one digit

Passwords: Making harder to crack (con't)

- Technique 3: Delay checking of passwords
  - If attacker doesn't have access to /etc/passwd, delay every remote login attempt by 1 second
  - Makes it infeasible for rapid-fire dictionary attack
- Technique 4: Assign very long passwords
  - Long passwords or pass-phrases can have more entropy (randomness→harder to crack)
  - Embed password in a smart card (or ATM card)
    - » Requires physical theft to steal password
    - » Can require PIN from user before authenticates self
  - Better: have smartcard generate pseudorandom number
    - » Client and server share initial seed
    - » Each second/login attempt advances to next random number
- Technique 5: "Zero-Knowledge Proof"
  - Require a series of challenge-response questions
    - » Distribute secret algorithm to user
    - » Server presents a number, say "5"; user computes something from the number and returns answer to server
    - » Server never asks same "question" twice
  - Often performed by smartcard plugged into system

Host Compromise

- One of earliest major Internet security incidents
  - Internet Worm (1988): compromised almost every BSD-derived machine on Internet
- Today: estimated that a single worm could compromise 10M hosts in < 5 min
- Attacker gains control of a host
  - Reads data
  - Erases data
  - Compromises another host
  - Launches denial-of-service attack on another host

Definitions

- Worm
  - Replicates itself
  - Usually relies on stack overflow attack
- Virus
  - Program that attaches itself to another (usually trusted) program
- Trojan horse
  - Program that allows a hacker a back door to compromised machine
- Botnet
  - A collection of programs running autonomously and controlled remotely
  - Can be used to spread out worms, mounting DDoS attacks

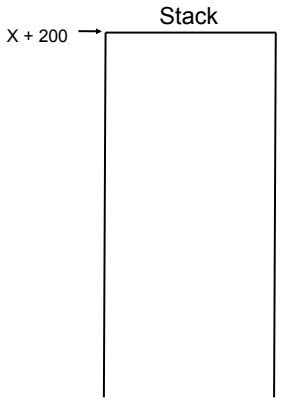
Buffer Overflow

- Part of the request sent by the attacker too large to fit into buffer server uses to hold it
- Spills over into memory beyond the buffer
- Allows remote attacker to inject executable code

```
void get_cookie(char *packet) {
  . . . (200 bytes of local vars) . . .
  munch(packet);
  . . .
}
void munch(char *packet) {
  int n;
  char cookie[512];
  . . .
  code here computes offset of cookie in
  packet, stores it in n
  strcpy(cookie, &packet[n]);
  . . .
}
```

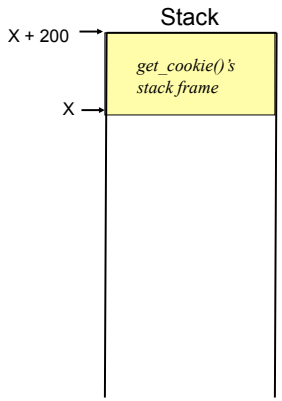

Example: Normal Execution

```
void get_cookie(char *packet) {
  . . . (200 bytes of local vars) . . .
  munch(packet);
  . . .
}
void munch(char *packet) {
  int n;
  char cookie[512];
  . . .
  code here computes offset of cookie in
  packet, stores it in n
  strcpy(cookie, &packet[n]);
  . . .
}
```



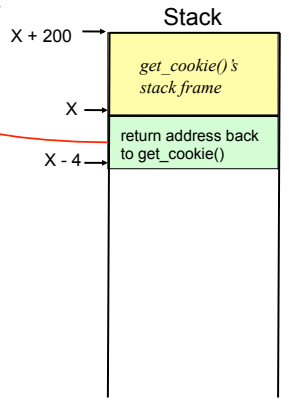
Example: Normal Execution

```
void get_cookie(char *packet) {
  . . . (200 bytes of local vars) . . .
  munch(packet);
  . . .
}
void munch(char *packet) {
  int n;
  char cookie[512];
  . . .
  code here computes offset of cookie in
  packet, stores it in n
  strcpy(cookie, &packet[n]);
  . . .
}
```



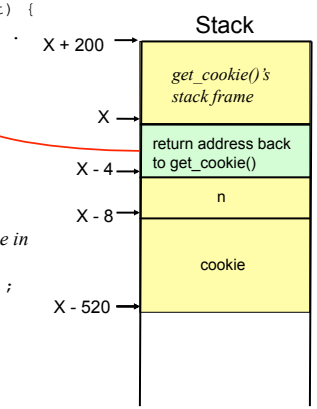
Example: Normal Execution

```
void get_cookie(char *packet) {
  . . . (200 bytes of local vars) . . .
  munch(packet);
  . . .
}
void munch(char *packet) {
  int n;
  char cookie[512];
  . . .
  code here computes offset of cookie in
  packet, stores it in n
  strcpy(cookie, &packet[n]);
  . . .
}
```



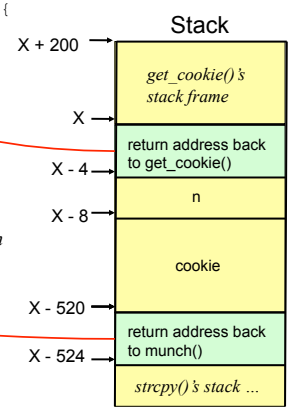
Example: Normal Execution

```
void get_cookie(char *packet) {
  . . . (200 bytes of local vars) . . .
  munch(packet);
  . . .
}
void munch(char *packet) {
  int n;
  char cookie[512];
  . . .
  code here computes offset of cookie in
  packet, stores it in n
  strcpy(cookie, &packet[n]);
  . . .
}
```



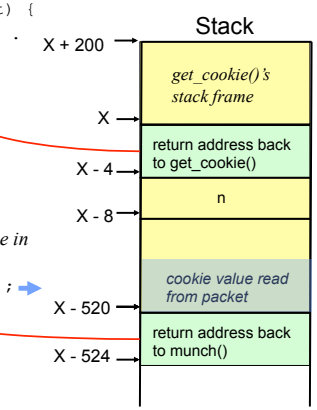
Example: Normal Execution

```
void get_cookie(char *packet) {
  . . . (200 bytes of local vars) . . .
  munch(packet);
  . . .
}
void munch(char *packet) {
  int n;
  char cookie[512];
  . . .
  code here computes offset of cookie in
  packet, stores it in n
  strcpy(cookie, &packet[n]);
  . . .
}
```



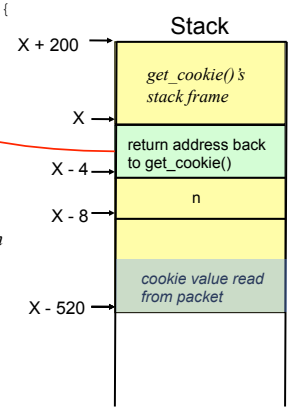
Example: Normal Execution

```
void get_cookie(char *packet) {
  . . . (200 bytes of local vars) . . .
  munch(packet);
  . . .
}
void munch(char *packet) {
  int n;
  char cookie[512];
  . . .
  code here computes offset of cookie in
  packet, stores it in n
  strcpy(cookie, &packet[n]);
  . . .
}
```



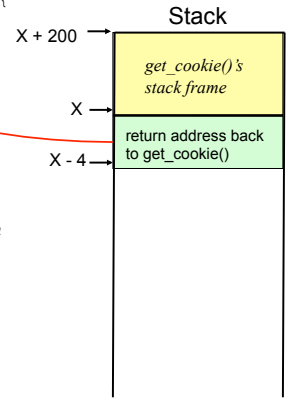
Example: Normal Execution

```
void get_cookie(char *packet) {
  . . . (200 bytes of local vars) . . .
  munch(packet);
  . . .
}
void munch(char *packet) {
  int n;
  char cookie[512];
  . . .
  code here computes offset of cookie in
  packet, stores it in n
  strcpy(cookie, &packet[n]);
  . . .
}
```



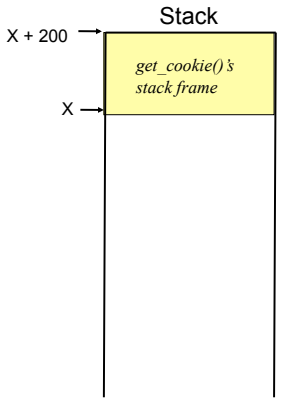
Example: Normal Execution

```
void get_cookie(char *packet) {
  . . . (200 bytes of local vars) . . .
  munch(packet);
  . . .
}
void munch(char *packet) {
  int n;
  char cookie[512];
  . . .
  code here computes offset of cookie in
  packet, stores it in n
  strcpy(cookie, &packet[n]);
  . . .
}
```



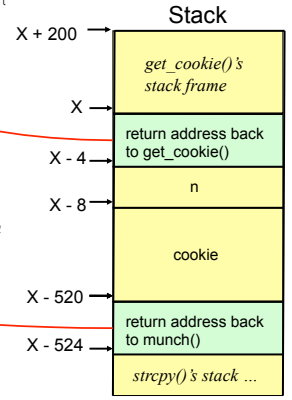
Example: Normal Execution

```
void get_cookie(char *packet) {
  . . . (200 bytes of local vars) . . .
  munch(packet);
  . . .
}
void munch(char *packet) {
  int n;
  char cookie[512];
  . . .
  code here computes offset of cookie in
  packet, stores it in n
  strcpy(cookie, &packet[n]);
  . . .
}
```



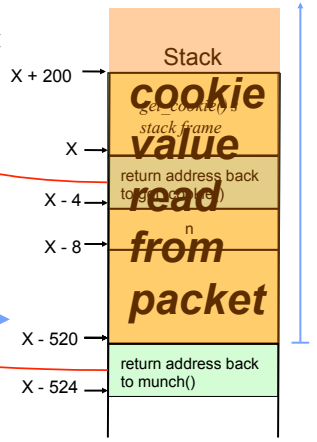
Example: Buffer Overflow

```
void get_cookie(char *packet) {
  . . . (200 bytes of local vars) . . .
  munch(packet);
  . . .
}
void munch(char *packet) {
  int n;
  char cookie[512];
  . . .
  code here computes offset of cookie in
  packet, stores it in n
  strcpy(cookie, &packet[n]);
  . . .
}
```



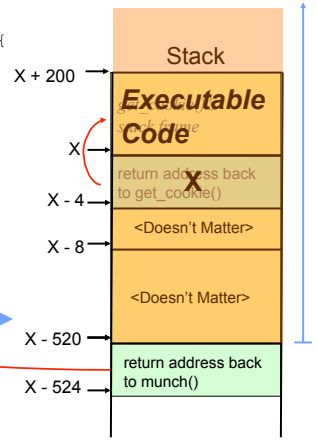
Example: Buffer Overflow

```
void get_cookie(char *packet) {
  . . . (200 bytes of local vars) . . .
  munch(packet);
  . . .
}
void munch(char *packet) {
  int n;
  char cookie[512];
  . . .
  code here computes offset of cookie in
  packet, stores it in n
  strcpy(cookie, &packet[n]);
  . . .
}
```



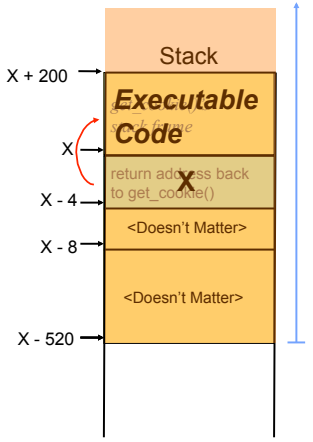
Example: Buffer Overflow

```
void get_cookie(char *packet) {
  . . . (200 bytes of local vars) . . .
  munch(packet);
  . . .
}
void munch(char *packet) {
  int n;
  char cookie[512];
  . . .
  code here computes offset of cookie in
  packet, stores it in n
  strcpy(cookie, &packet[n]);
  . . .
}
```

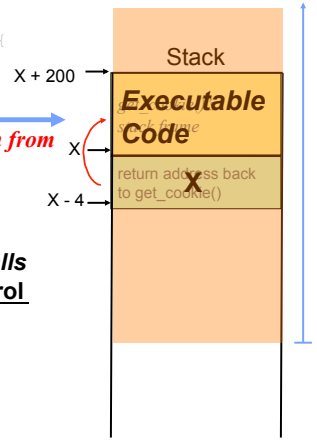


Example: Buffer Overflow

```
void get_cookie(char *packet) {
  . . . (200 bytes of local vars) . . .
  munch(packet);
  . . .
}
void munch(char *packet) {
  int n;
  char cookie[512];
  . . .
  code here computes offset of cookie in
  packet, stores it in n
  strcpy(cookie, &packet[n]);
  . . .
}
```



Example: Buffer Overflow



Now branches to code read in from the network

From here on, machine falls under the attacker's control

Buffer Overflows: Potential Solutions

- Don't write buggy software
  - It's not like people try to write buggy software
- Type-safe Languages
  - Unrestricted memory access of C/C++ contributes to problem
  - Use Java, Perl, Python instead
- OS architecture
  - Compartmentalize programs better, so one compromise doesn't compromise the entire system
  - E.g., DNS server doesn't need total system access
- Firewalls - restrict remote access to services
- *Intrusion detection*: recognize attack & block it

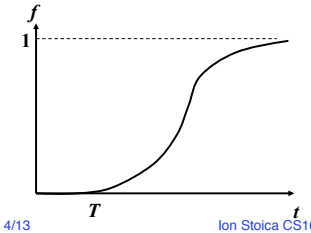
Automated Compromise: Worms

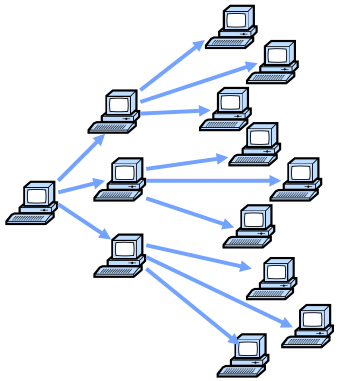
- When attacker compromises a host, they can instruct it to do whatever they want
- Instructing it to find more vulnerable hosts to repeat the process creates a worm: a program that self-replicates across a network
  - Often spread by picking 32-bit Internet addresses at random to probe …
  - … but this isn't fundamental
- As the worm repeatedly replicates, it grows *exponentially fast* because each copy of the worm works in parallel to find more victims

Worm Spreading

$$f = (e^{K(t-T)} - 1) / (1 + e^{K(t-T)})$$

- f – fraction of hosts infected
- K – rate at which one host can compromise others
- T – start time of the attack





Worm Examples

- Morris worm (1988)
- Code Red (2001)
  - 369K hosts in 10 hours
- MS Slammer (January 2003)
- Theoretical worms
  - 1M hosts in 1.3 sec
  - $50B+ damage

Morris Worm (1988)

- Infect multiple types of machines (Sun 3 and VAX)
  - Was supposed to be benign: estimate size of Internet
  - Spread using a Sendmail bug
- Attack multiple security holes including
  - Buffer overflow in fingerd
  - Debugging routines in Sendmail
  - Password cracking
- Intend to be benign but it had a bug
  - Fixed chance the worm wouldn't quit when reinfecting a machine → number of worm on a host built up rendering the machine unusable

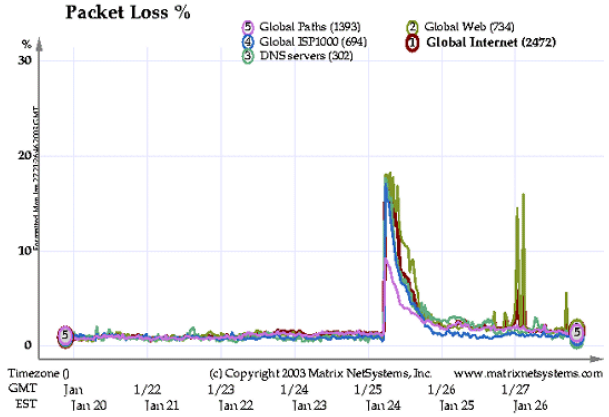
Code Red Worm (2001)

- Attempts to connect to TCP port 80 (i.e., HTTP port) on a randomly chosen host
- If successful, the attacking host sends a crafted HTTP GET request to the victim, attempting to exploit a buffer overflow
- Worm "bug": all copies of the worm use the same random generator to scan new hosts
  - DoS attack on those hosts
  - Slow to infect new hosts
- 2nd generation of Code Red fixed the bug!
  - It spread much faster

MS SQL Slammer (January 2003)

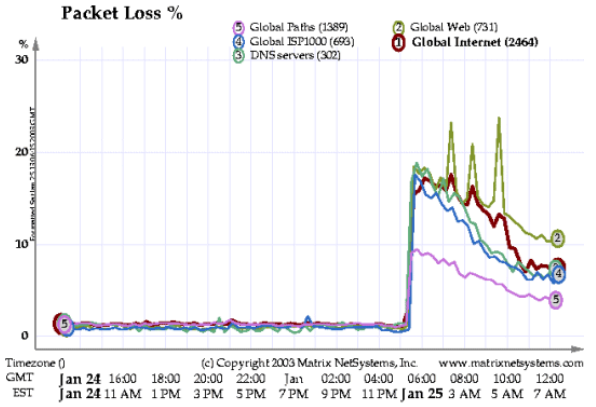
- Uses UDP port 1434 to exploit a buffer overflow in MS SQL server
- Effect
  - Generate massive amounts of network packets
  - Brought down as many as 5 of the 13 internet root name servers
- Others
  - The worm only spreads as an in-memory process: it never writes itself to the hard drive
    - » Solution: close UDP port on fairewall and reboot

MS SQL Slammer (January 2003)



(From http://www.f-secure.com/v-descs/mssqlm.shtml)

MS SQL Slammer (January 2003)



(From http://www.f-secure.com/v-descs/mssqlm.shtml)

Hall of Shame

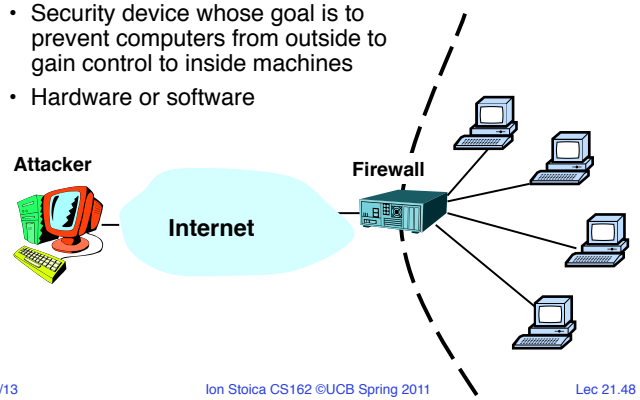
- Software that have had many stack overflow bugs:
  - BIND (most popular DNS server)
  - RPC (Remote Procedure Call, used for NFS)
    - » NFS (Network File System), widely used at UCB
  - Sendmail (most popular UNIX mail delivery software)
  - IIS (Windows web server)
  - SNMP (Simple Network Management Protocol, used to manage routers and other network devices)

Potential Solutions

- Don't write buggy software
  - It's not like people try to write buggy software
- Type-safe Languages
  - Unrestricted memory access of C/C++ contributes to problem
  - Use Java, Perl, or Python instead
- OS architecture
  - Compartmentalize programs better, so one compromise doesn't compromise the entire system
  - E.g., DNS server doesn't need total system access
- Firewalls

Firewall

- Security device whose goal is to prevent computers from outside to gain control to inside machines
- Hardware or software



Firewall (cont'd)

- Restrict traffic between Internet and devices (machines) behind it based on
  - Source address and port number
  - Payload
  - Stateful analysis of data
- Examples of rules
  - Block any external packets not for port 80
  - Block any email with an attachment
  - Block any external packets with an internal IP address
    - » Ingress filtering

Firewalls: Properties

- Easier to deploy firewall than secure all internal hosts
- Doesn't prevent user exploitation
- Tradeoff between availability of services (firewall passes more ports on more machines) and security
  - If firewall is too restrictive, users will find way around it, thus compromising security
  - E.g., have all services use port 80

Denial of Service

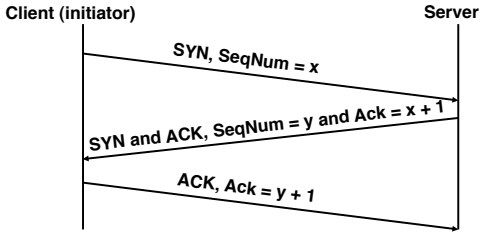
- Huge problem in current Internet
  - Major sites attacked: Yahoo!, Amazon, eBay, CNN, Microsoft
  - 12,000 attacks on 2,000 organizations in 3 weeks
  - Some more that 600,000 packets/second
    - » More than 192Mb/s
  - Almost all attacks launched from compromised hosts
- General Form
  - Prevent legitimate users from gaining service by overloading or crashing a server
  - E.g., SYN attack

Affect on Victim

- Buggy implementations allow unfinished connections to eat all memory, leading to crash
- Better implementations limit the number of unfinished connections
  - Once limit reached, new SYNs are dropped
- Affect on victim's users
  - Users can't access the targeted service on the victim because the unfinished connection queue is full → DoS

SYN Attack (Recap: 3-Way Handshaking)

- Goal: agree on a set of parameters: the start sequence number for each side
  - Starting sequence numbers are random.



SYN Attack

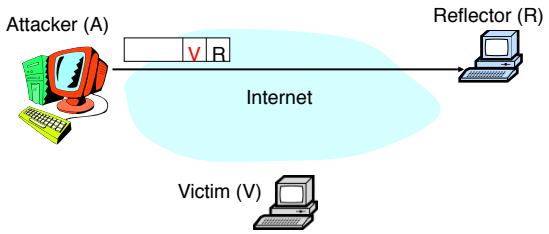
- Attacker: send at max rate TCP SYN with random spoofed source address to victim
  - Spoofing: use a different source IP address than own
  - Random spoofing allows one host to pretend to be many
- Victim receives many SYN packets
  - Send SYN+ACK back to spoofed IP addresses
  - Holds some memory until 3-way handshake completes
    - » Usually never, so victim times out after long period (e.g., 3 minutes)

Solution: SYN Cookies

- Server: send SYN-ACK with sequence number y, where
  - y = H(client_IP_addr, client_port)
  - H(): one-way hash function
- Client: send ACK containing y+1
- Sever:
  - verify if y = H(client_IP_addr, client_port)
  - If verification passes, allocate memory
- Note: server doesn't allocate any memory if the client's address is spoofed

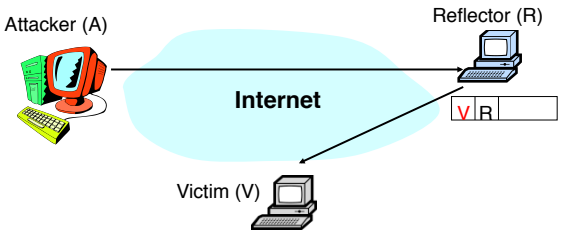
Other Denial-of-Service Attacks

- Reflection
  - Cause one non-compromised host to attack another
  - E.g., host A sends DNS request or TCP SYN with source V to server R. R sends reply to V



Other Denial-of-Service Attacks

- Reflection
  - Cause one non-compromised host to attack another
  - E.g., host A sends DNS request or TCP SYN with source V to server R. R sends reply to V



Other Denial-of-Service Attacks

- DNS
  - Ping flooding attack on DNS root servers (October 2002)
  - 9 out of 13 root servers brought down
  - Relatively small impact (why?)

Identifying and Stop Attacking Machines

- Defeat spoofed source addresses
- Does not stop or slow attack
- Egress filtering
  - A domain's border router drop outgoing packets which do not have a valid source address for that domain
  - If universal, could abolish spoofing
- IP Traceback
  - Routers probabilistically tag packets with an identifier
  - Destination can infer path to true source after receiving enough packets

Summary

- Security is one of the biggest problem today
- Host Compromise
  - Poorly written software
  - Partial solutions: better OS security architecture, type-safe languages, firewalls
- Denial-of-Service
  - No easy solution: DoS can happen at many levels