CS162
Operating Systems and Systems Programming
Lecture 18

Security (II)

October 31, 2011

Anthony D. Joseph and Ion Stoica

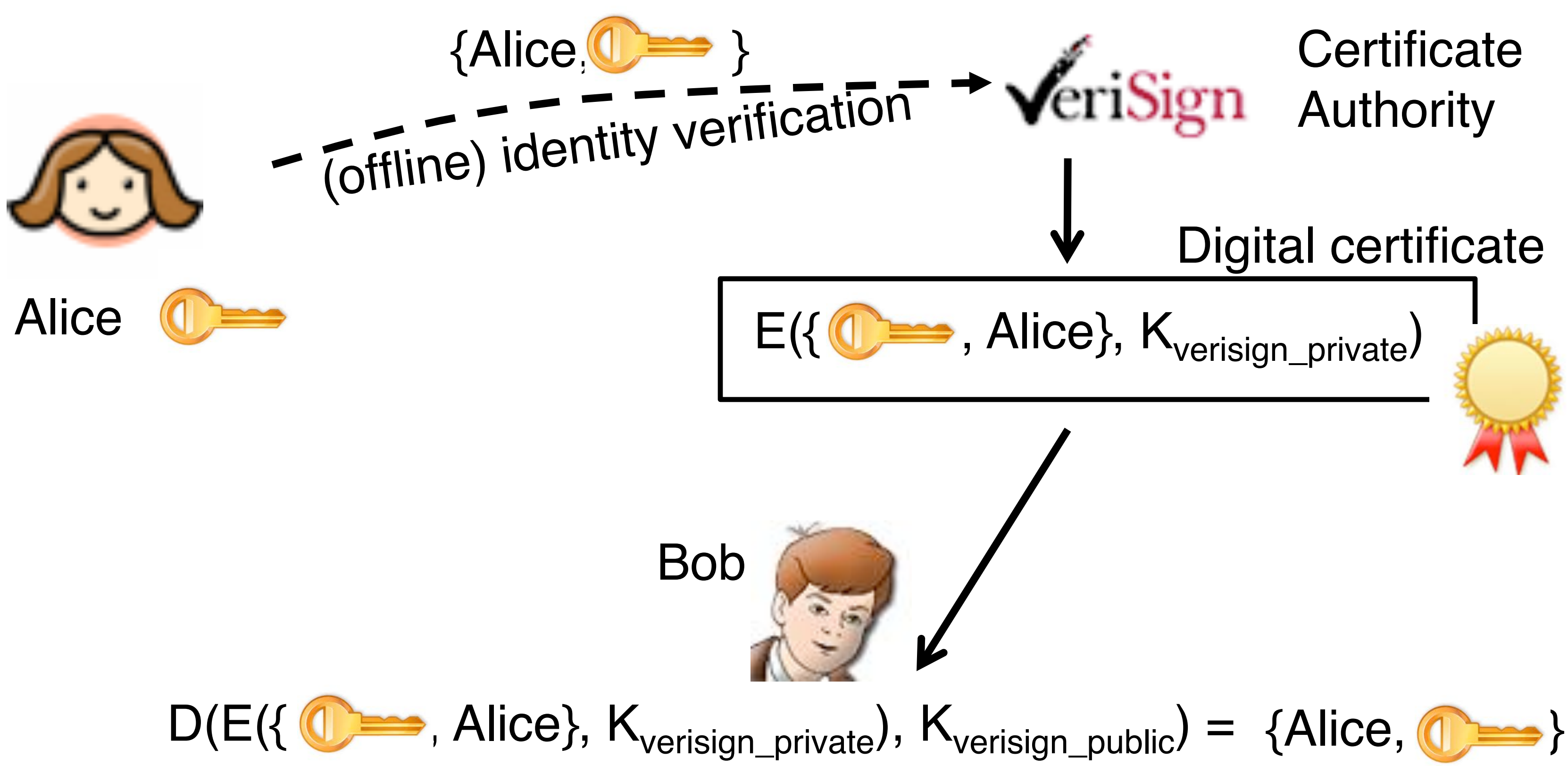
http://inst.eecs.berkeley.edu/~cs162

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: Digital Certificates

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



HTTPS Vulnerabilities

- Break into any Certificate Authority
  - 600+ Certificate Authorities that your browser will trust
  - This attack has occurred
- Compromise a router near any Certificate Authority
  - Read the CA’s outgoing email or alter incoming DNS packets, breaking domain validation
  - Compromise a router near the victim site to read incoming email or outgoing DNS responses
- Compromise a recursive DNS server used that is used by a Certificate Authority, or forge a DNS entry for a victim domain

https://www.eff.org/deeplinks/2011/10/how-secure-https-today

HTTPS Vulnerabilities (cont'd)

- Attack some other network protocol, such as TCP or BGP, in a way that grants access to emails to the victim domain
- A government could order a Certificate Authority to produce a malicious certificate for any domain
  - CAs are located in 52+ countries
  - There is circumstantial evidence that this may happen
- How often is a CA compromised?
  - 14 times total
  - *4 times since June!*

https://www.eff.org/deeplinks/2011/10/how-secure-https-today

This Lecture

- 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

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
  - Zero-day exploit
- Attacker gains control of a host
  - Reads data
  - Erases data
  - Compromises another host
  - Launches denial-of-service attack on another host

Stepping Stone Compromise

PHOTO: JOHN R. COUGHLIN/CNNMONEY

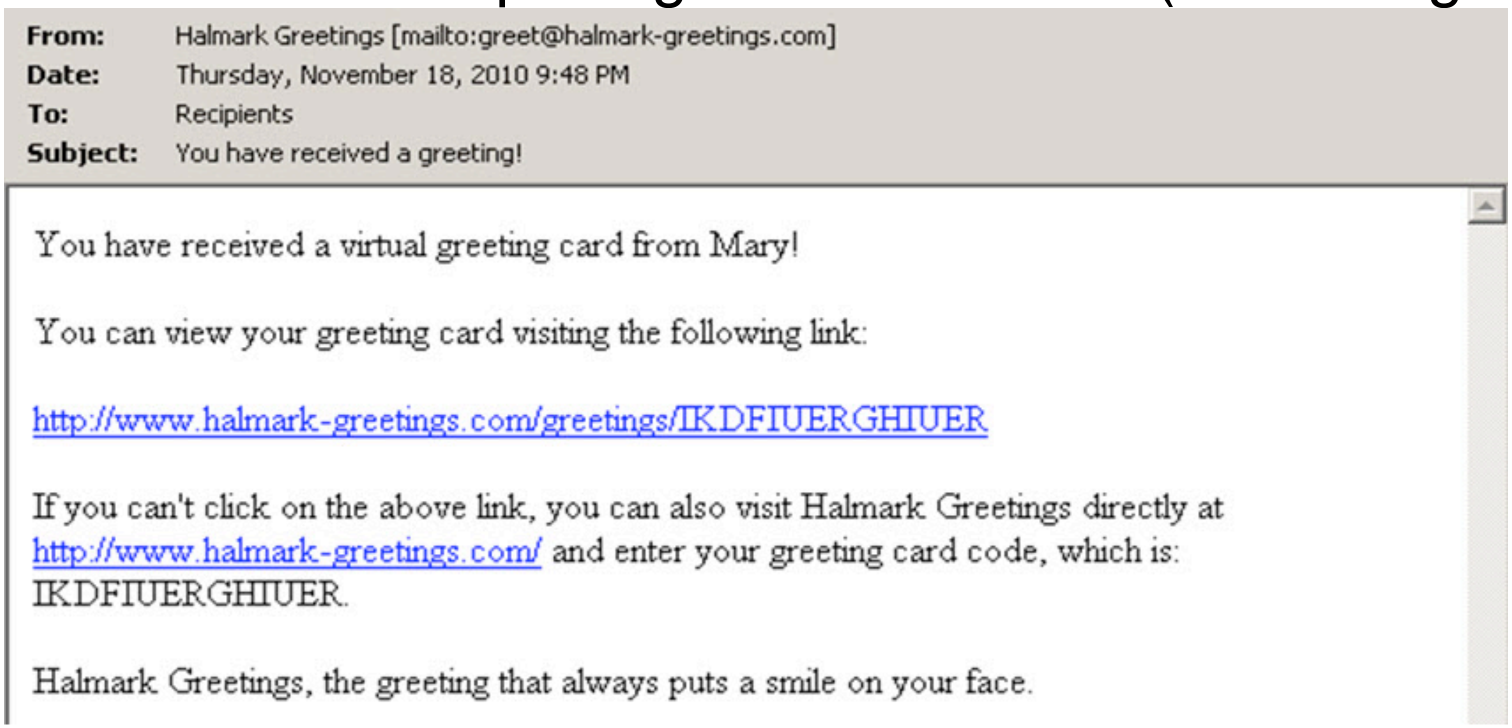
- RSA SecurID compromise (March 2011)
  - 2-factor authentication
  - Code changes every few secs
  - Data on codes stolen
- 760 companies attacked using stolen SecurID info
  - 20% of Fortune 100
  - Charles Schwabb & Co., Cisco Systems, eBay, European Space Agency, Facebook, Freddie Mac, Google, General Services Administration, IBM, Intel Corp., IRS, MIT, Motorola, Northrop Grumman, *Verisign*, VMWare, Wachovia, Wells Fargo, …
  - http://krebsonsecurity.com/2011/10/who-else-was-hit-by-the-rsa-attackers/

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 (Zombies)
  - A collection of programs running autonomously and controlled remotely
  - Can be used to spread out worms, mounting DDoS attacks

Trojan Example

- Nov/Dec e-mail message sent containing holiday message and a link or attachment
- Goal: trick user into opening link/attachment (social engineering)



From: Halmark Greetings [mailto:greet@halmark-greetings.com]
Date: Thursday, November 18, 2010 9:48 PM
To: Recipients
Subject: You have received a greeting!

You have received a virtual greeting card from Mary!

You can view your greeting card visiting the following link:

http://www.halmark-greetings.com/greetings/IKDFIUERGHIUER

If you can't click on the above link, you can also visit Halmark Greetings directly at http://www.halmark-greetings.com/ and enter your greeting card code, which is: IKDFIUERGHIUER.

Halmark Greetings, the greeting that always puts a smile on your face.

- Adds keystroke logger or turns into zombie
- How? Typically by using a buffer overflow exploit

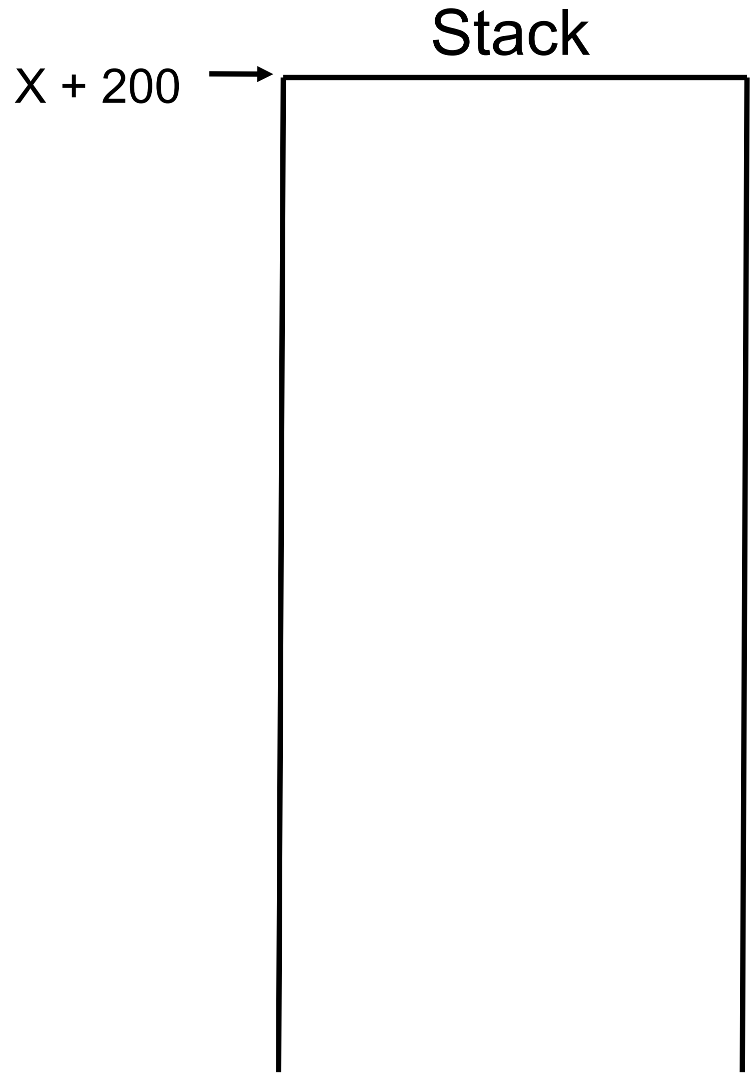
Buffer Overflow

- 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]);
    . . .
}
```

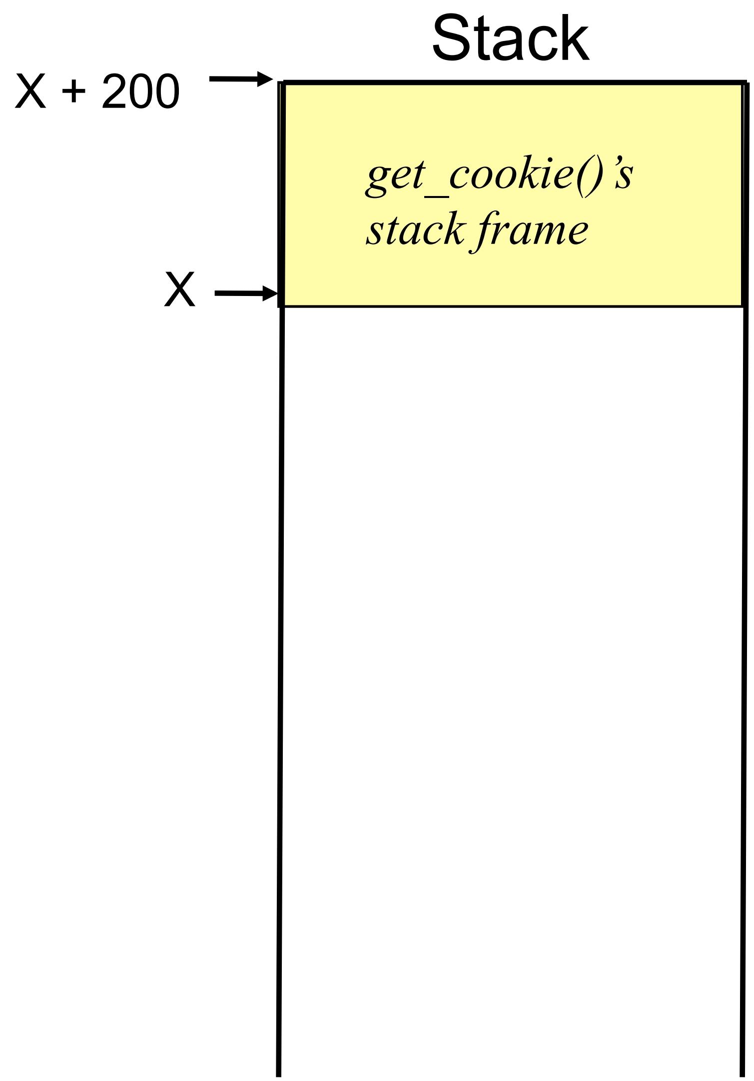
Example: Normal Execution

```
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]);
    . . .
}
```



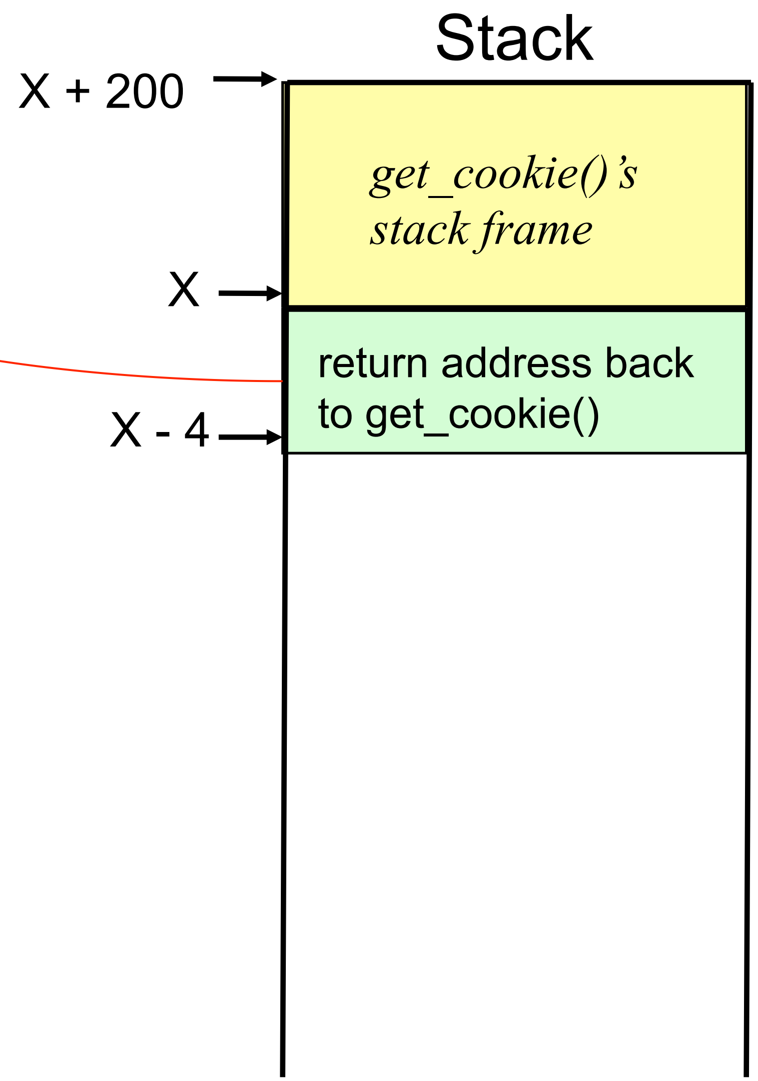
Example: Normal Execution

```
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]);
   . . .
}
```



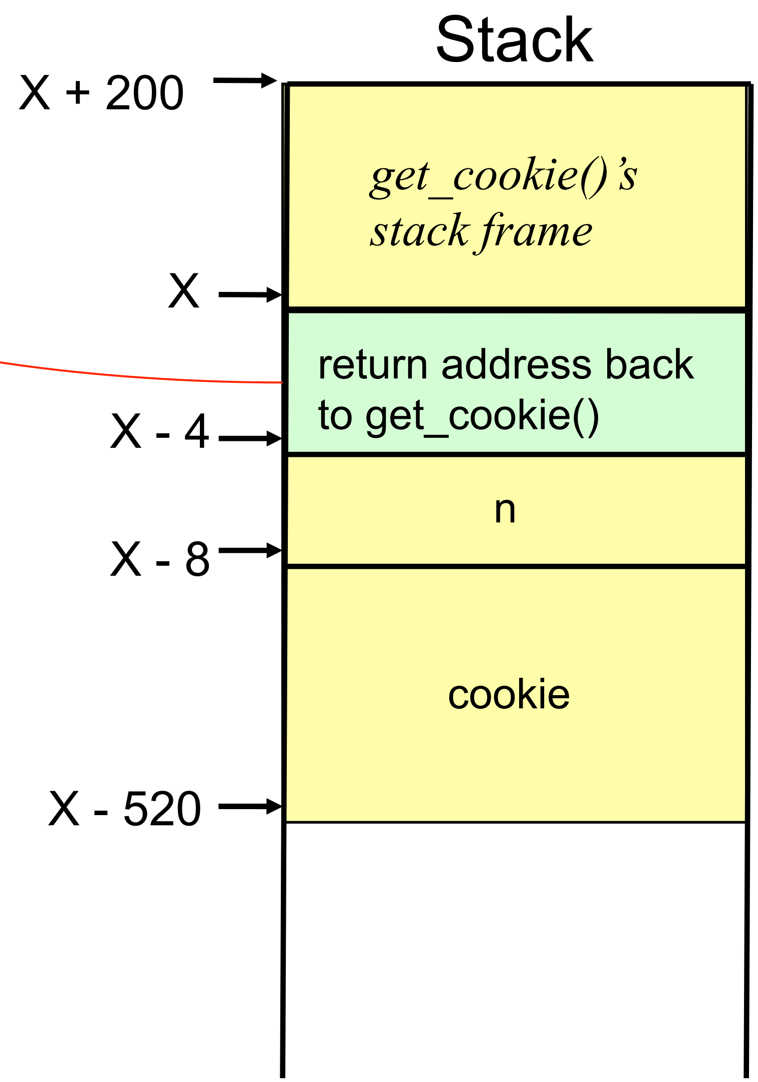
Example: Normal Execution

```
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]);
  . . .
}
```



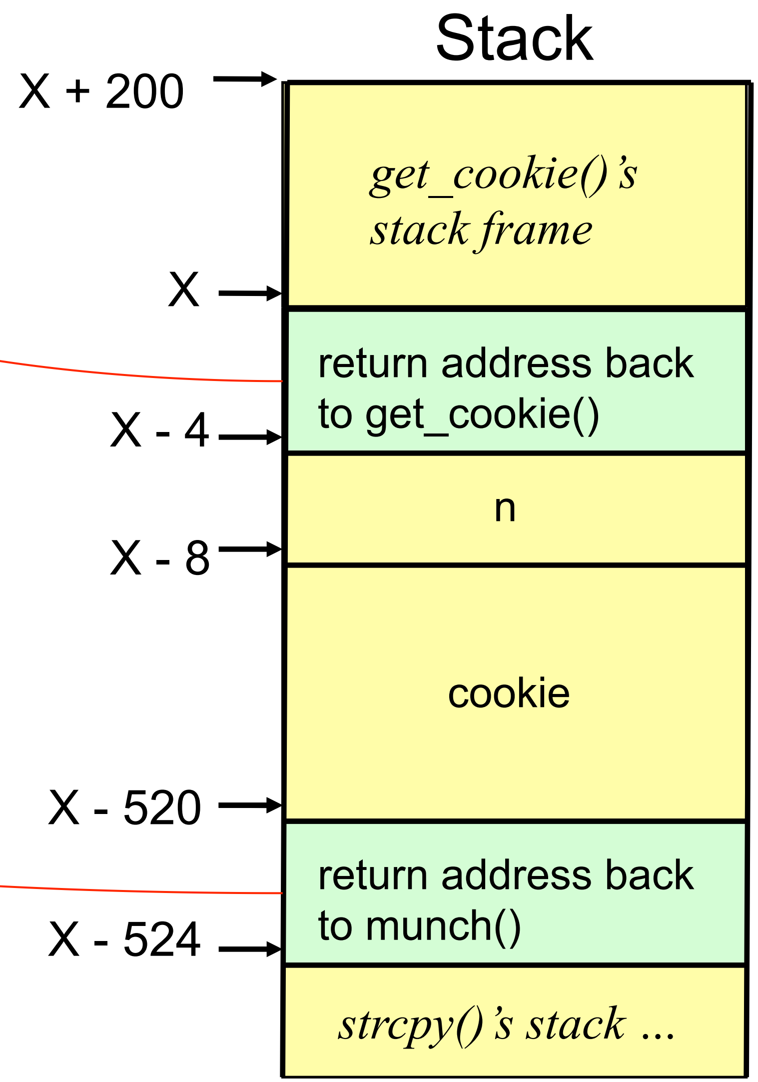
Example: Normal Execution

```
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]);
   . . .
}
```



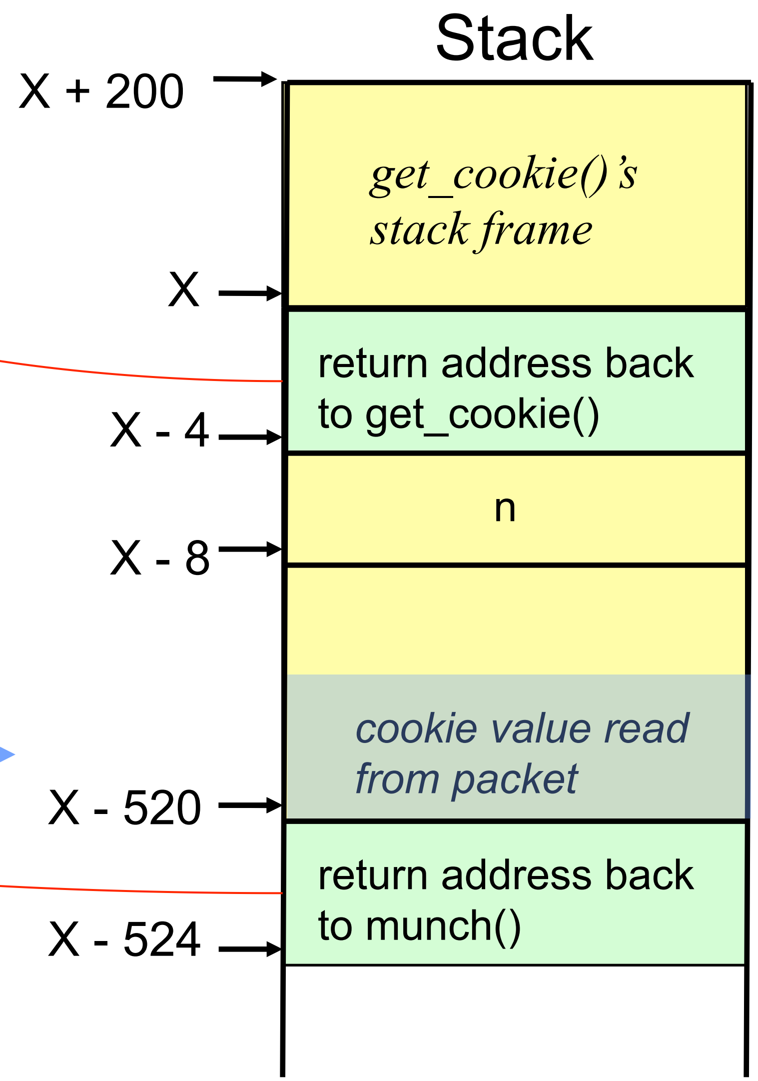
Example: Normal Execution

```
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]);
   . . .
}
```



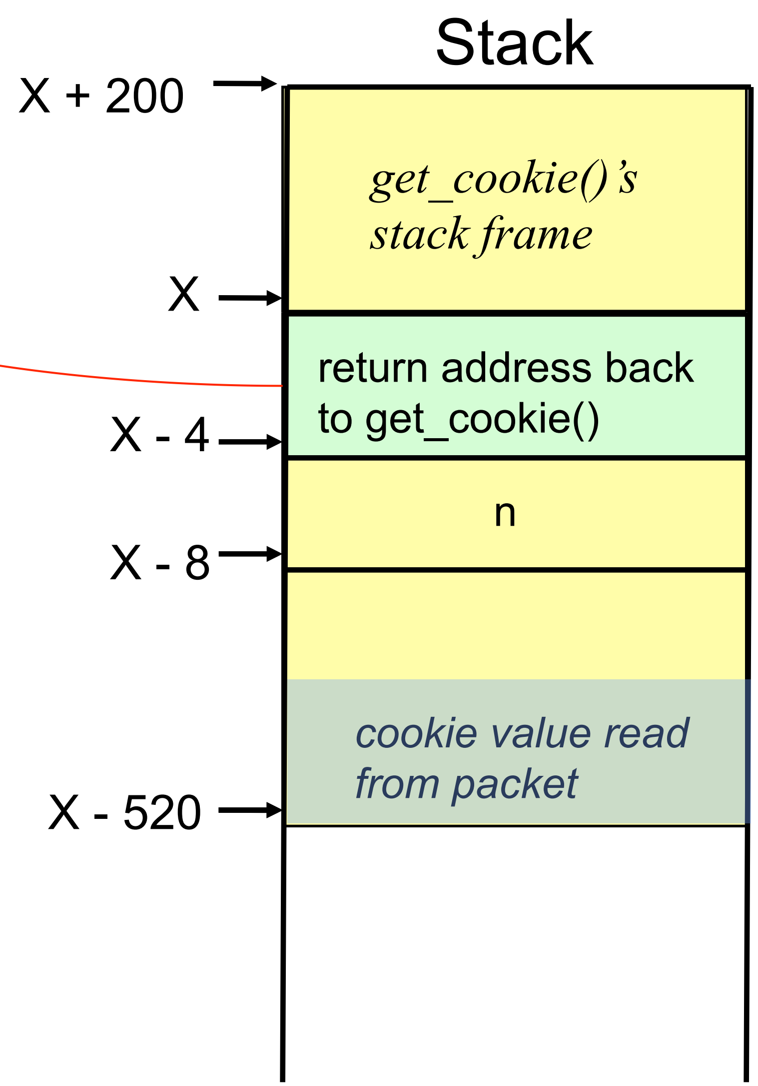
Example: Normal Execution

```
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]);
   . . .
}
```



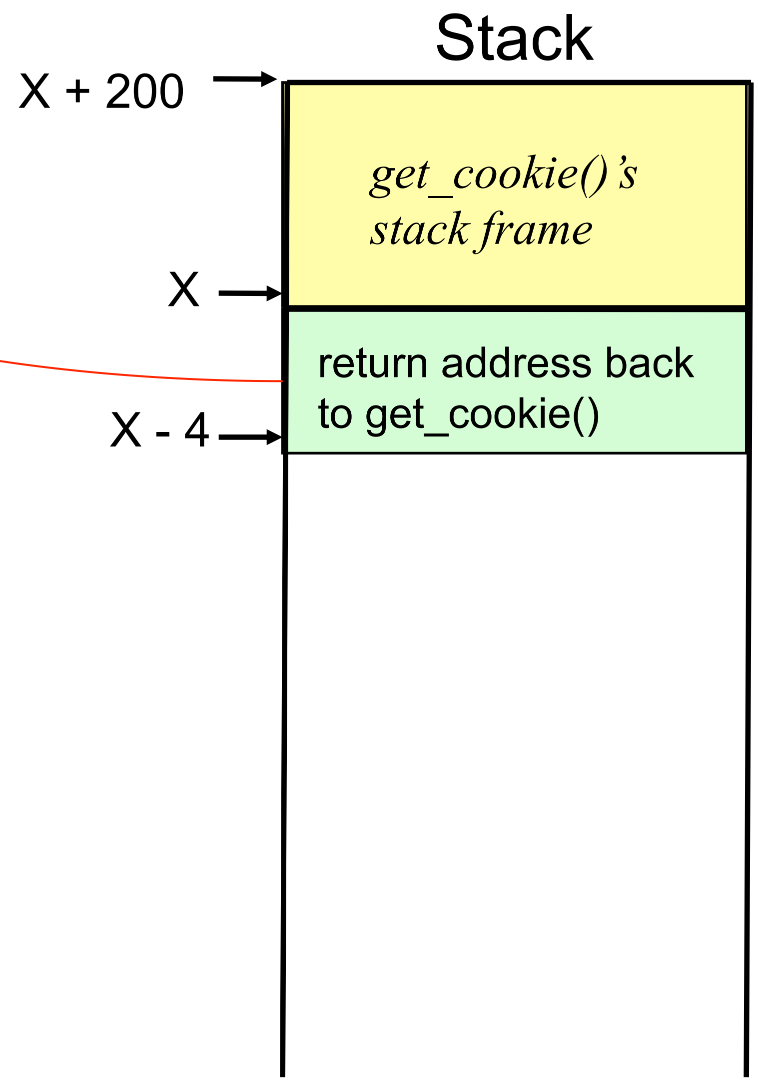
Example: Normal Execution

```
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]);
   . . .
}
```



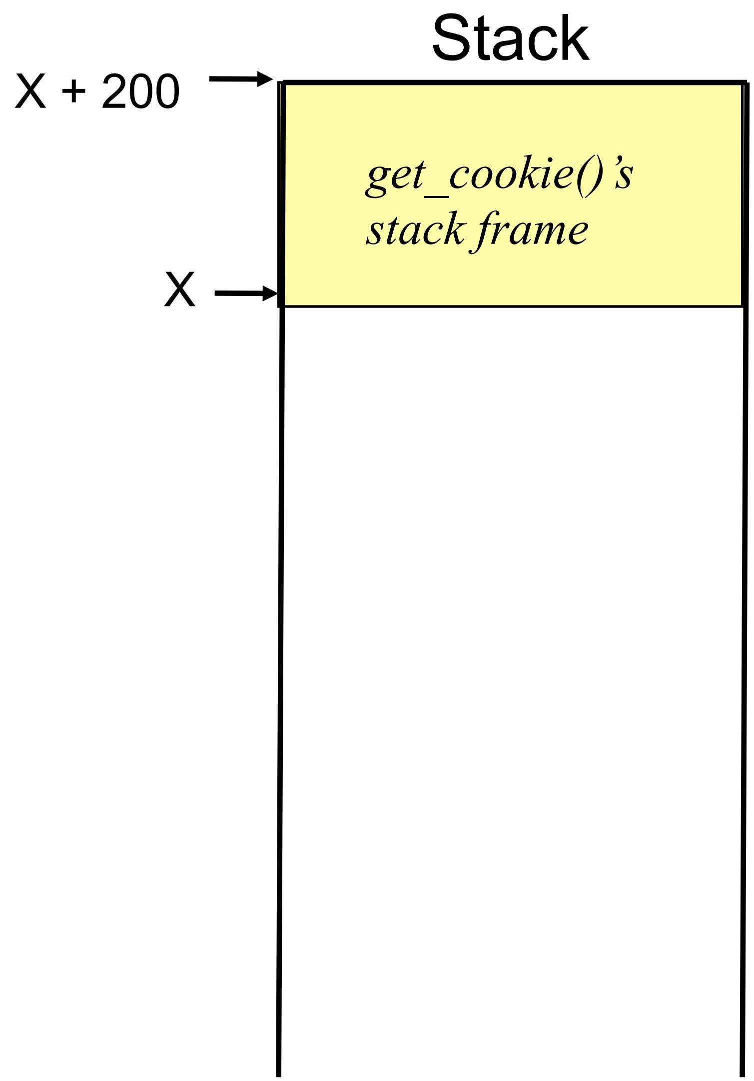
Example: Normal Execution

```
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]);
  . . .
}
```



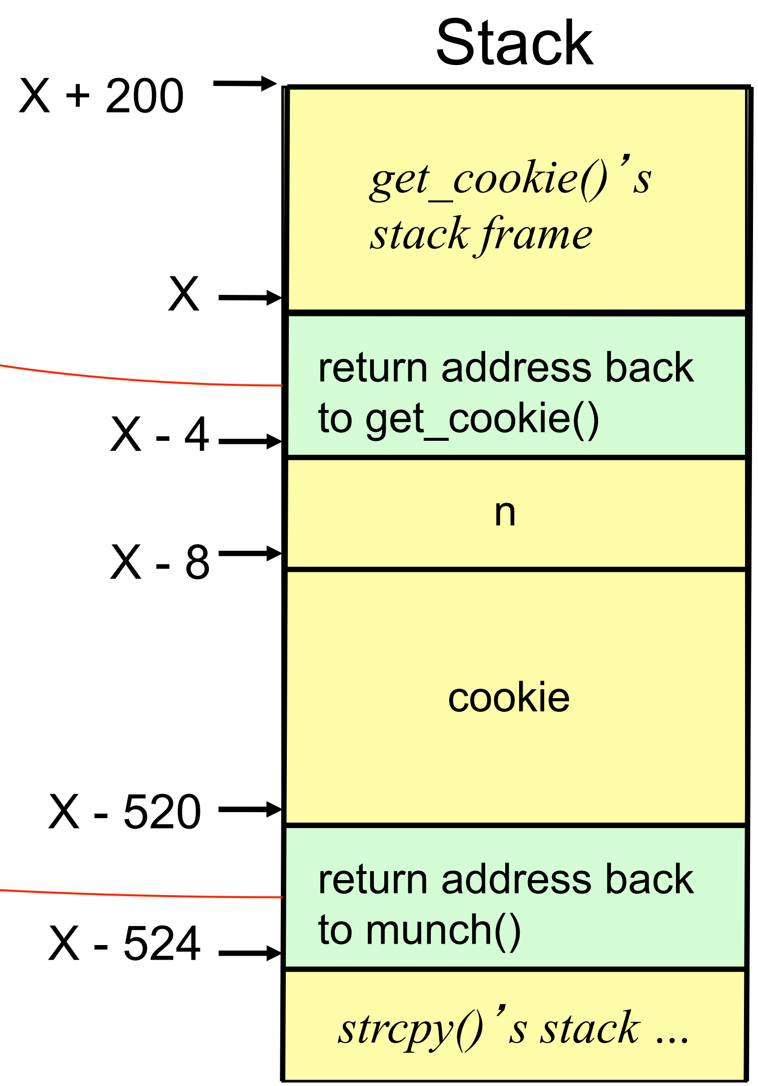
Example: Normal Execution

```
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]);
  . . .
}
```



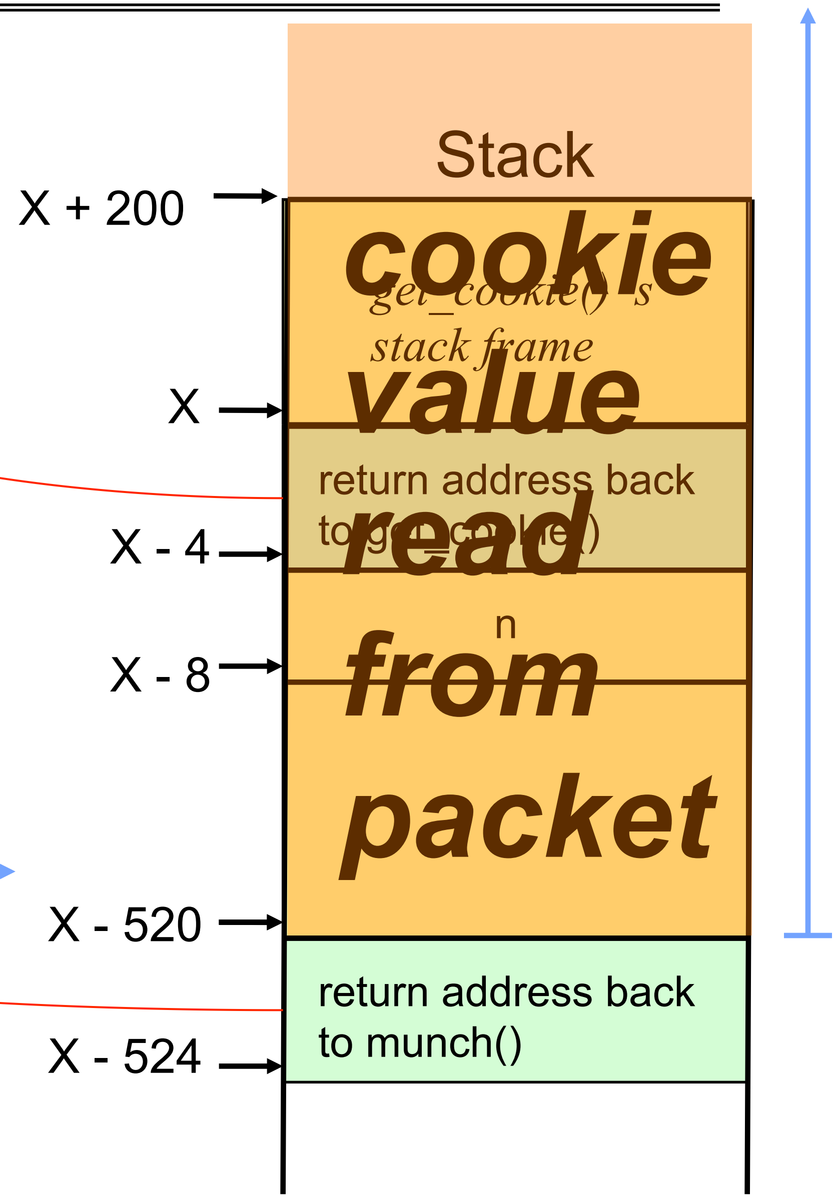
Example: Buffer Overflow

```
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]);
   . . .
}
```



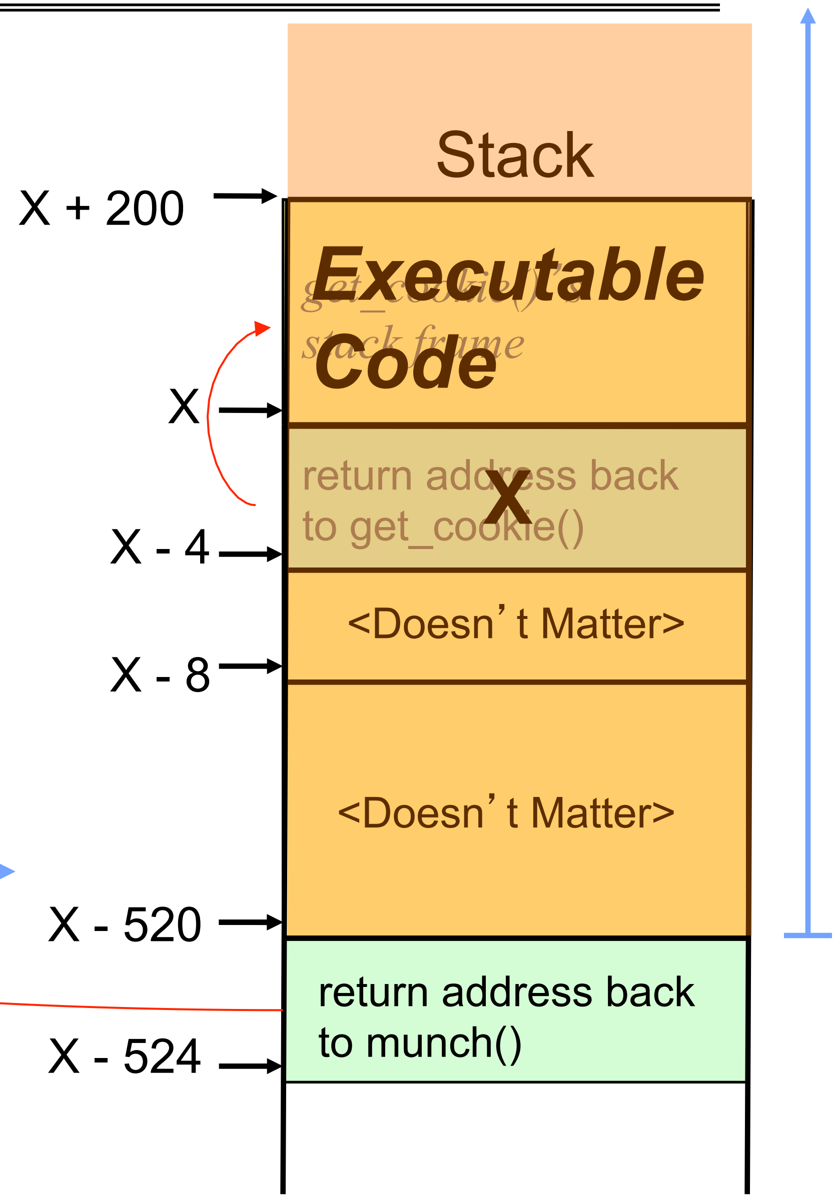
Example: Buffer Overflow

```
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]);
  . . .
}
```



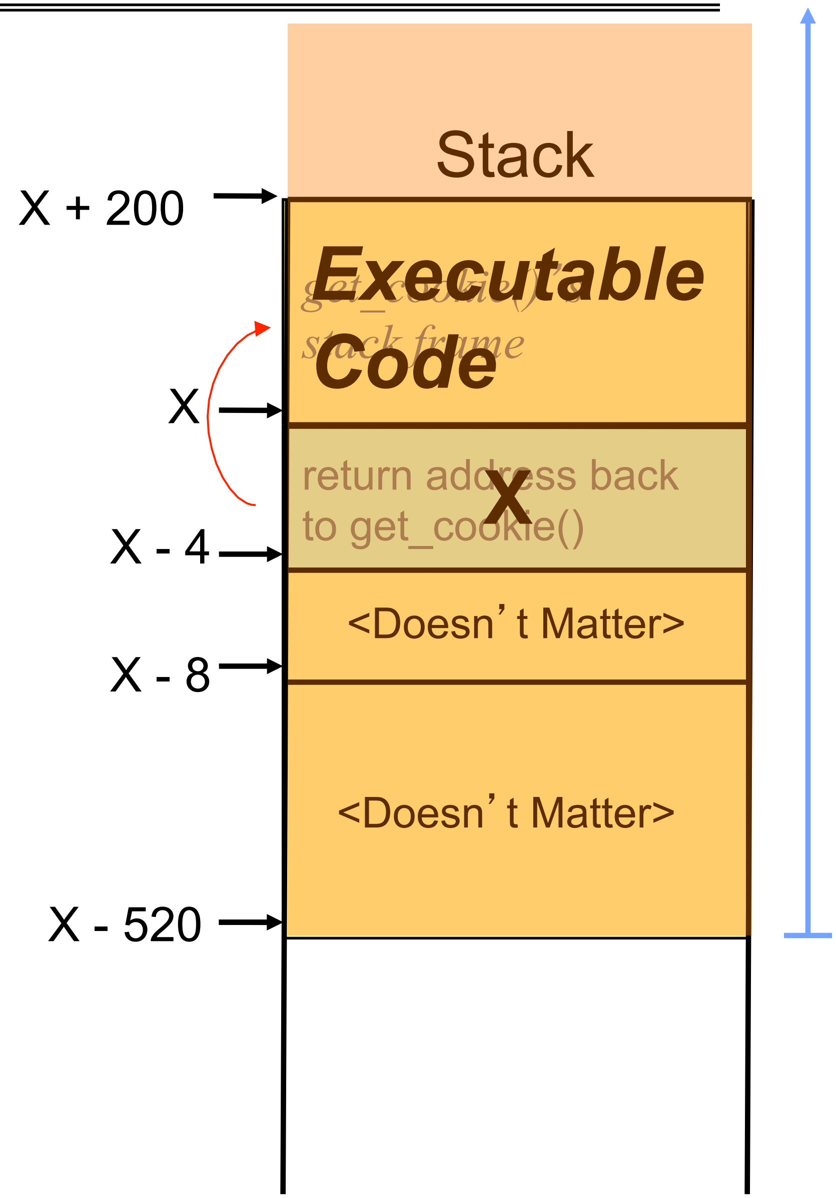
Example: Buffer Overflow

```
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]);
   . . .
}
```

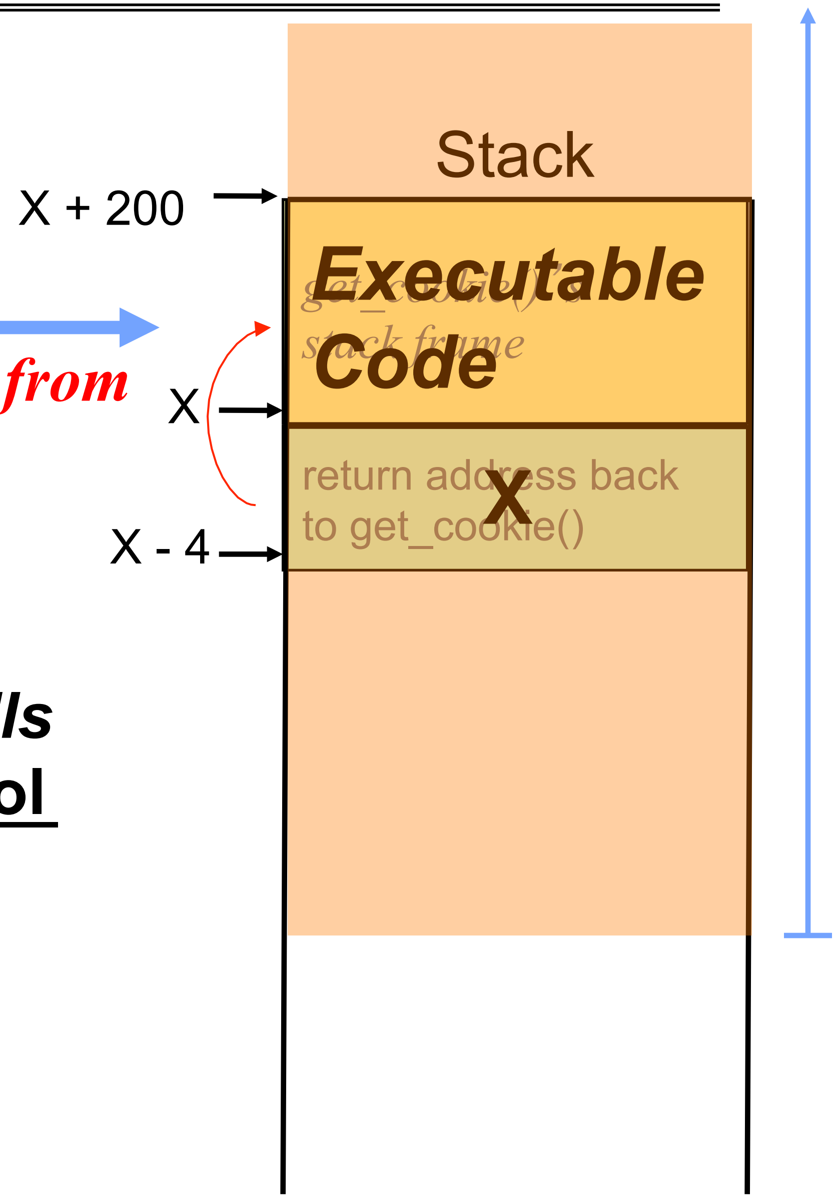


Example: Buffer Overflow

```
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]);
   . . .
}
```

Now branches to code read in from the network

<u>From here on, machine falls under the attacker's control</u>



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

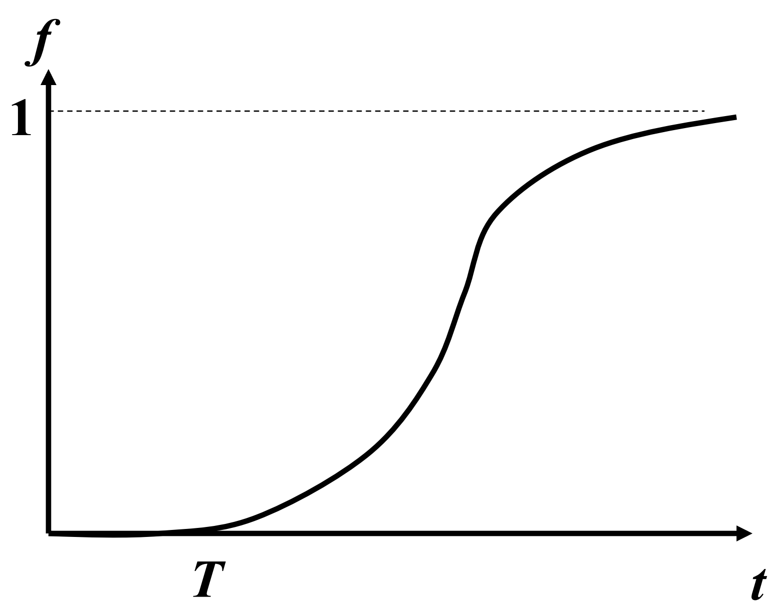
Automated Compromise: Worms

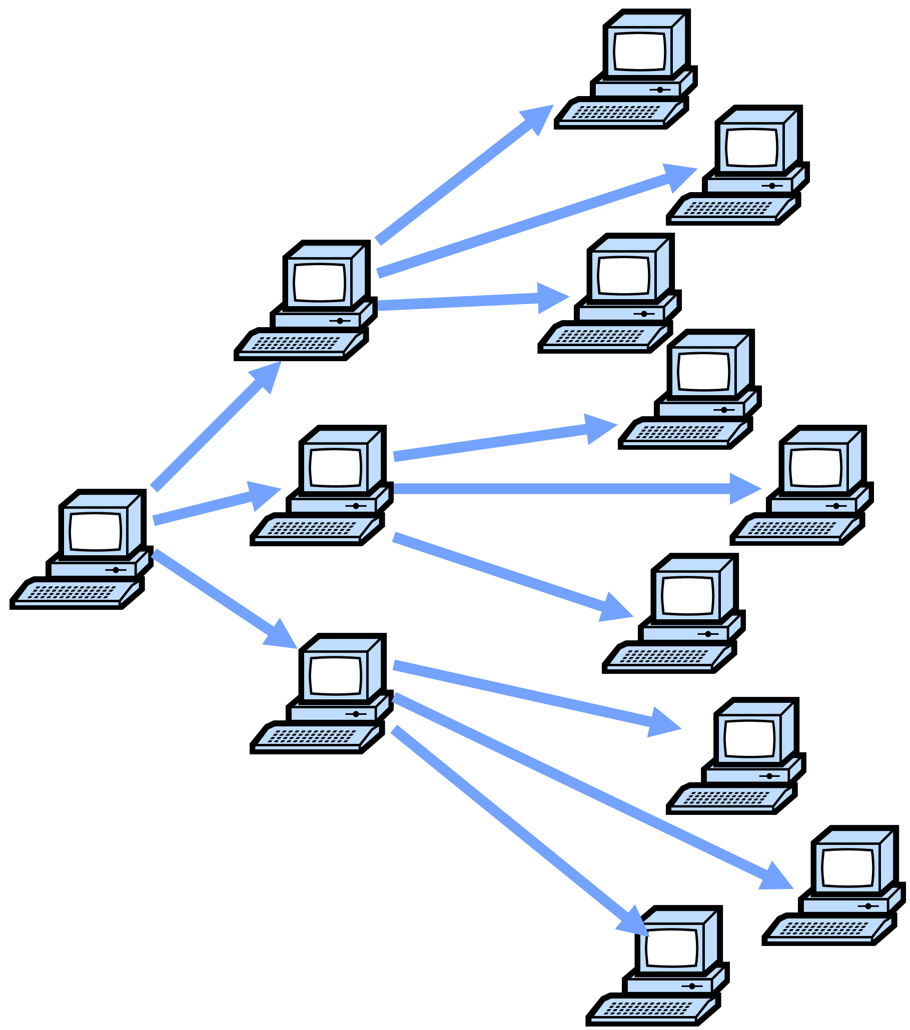
- When attacker compromises a host, they can instruct it to do whatever they want
- Instructing it to find more vulnerable hosts to repeat the process creates a worm: a program that self-replicates across a network
  - Often spread by picking 32-bit Internet addresses at random to probe …
  - … but this 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

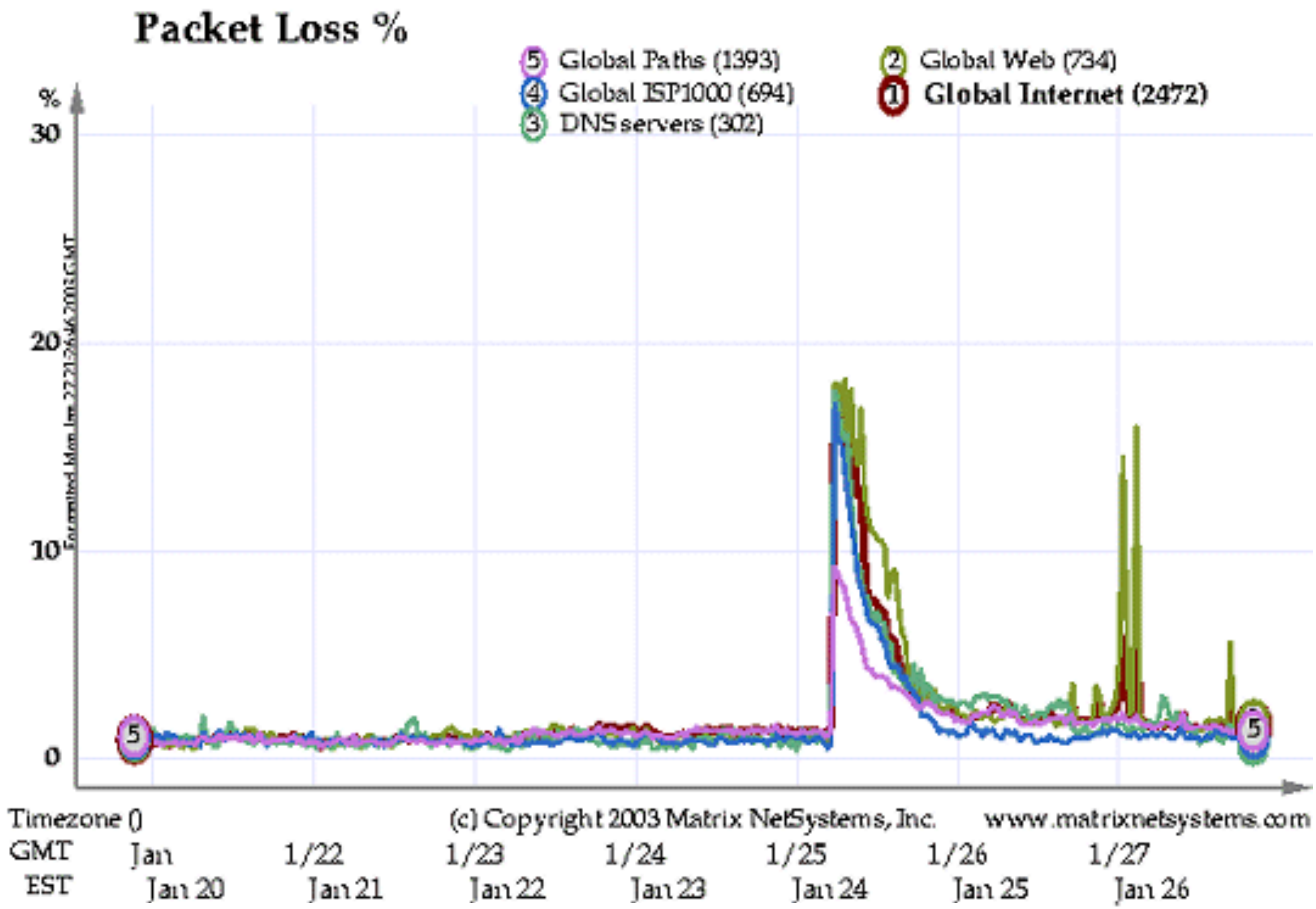
Code Red Worm (2001)

- 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)

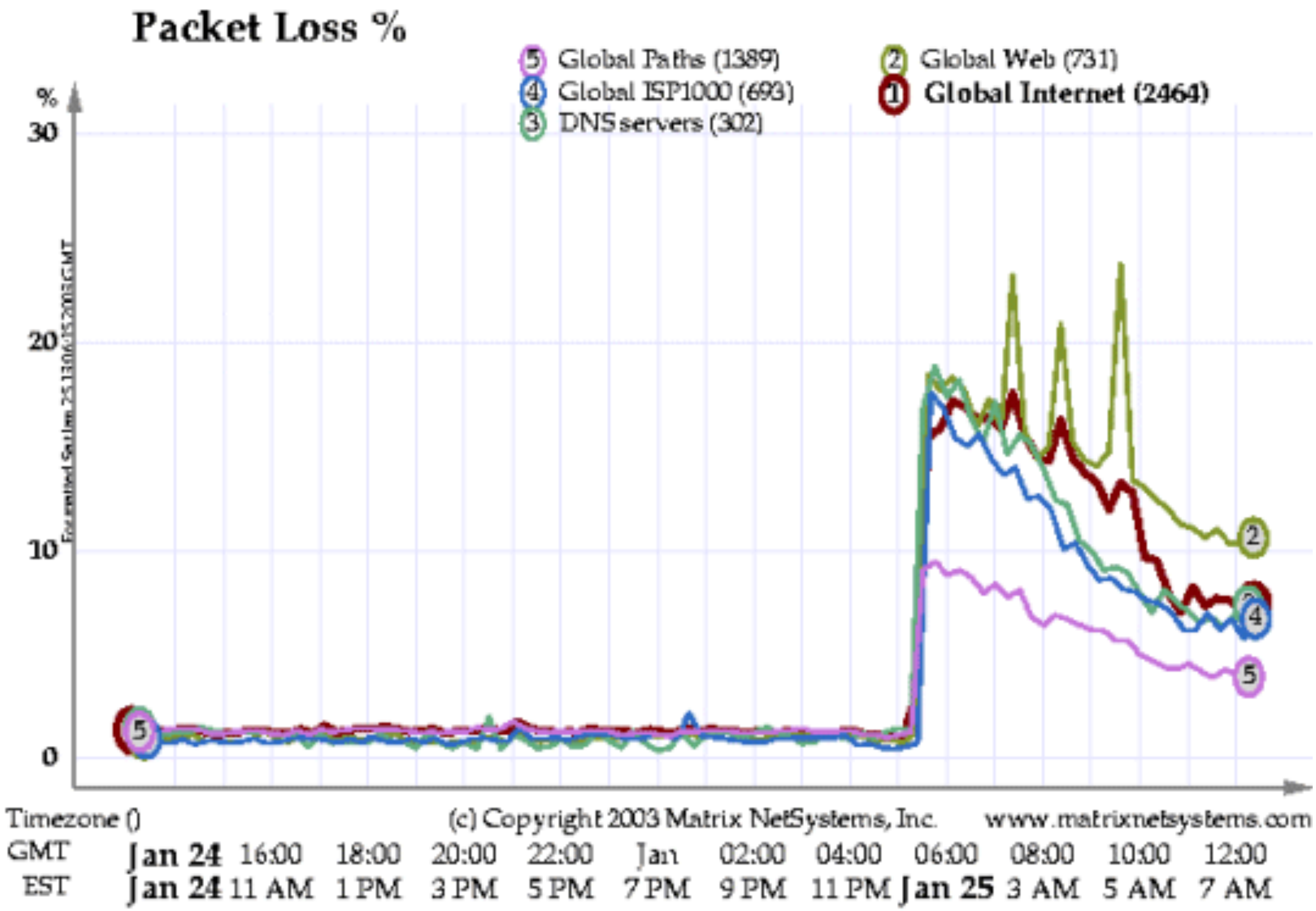
- Uses UDP port 1434 to exploit a buffer overflow in MS SQL server
- Effect
  - Generate massive amounts of network packets
  - Brought down as many as 5 of the 13 internet root name servers
- Others
  - The worm only spreads as an in-memory process: it never writes itself to the hard drive
    - » Solution: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

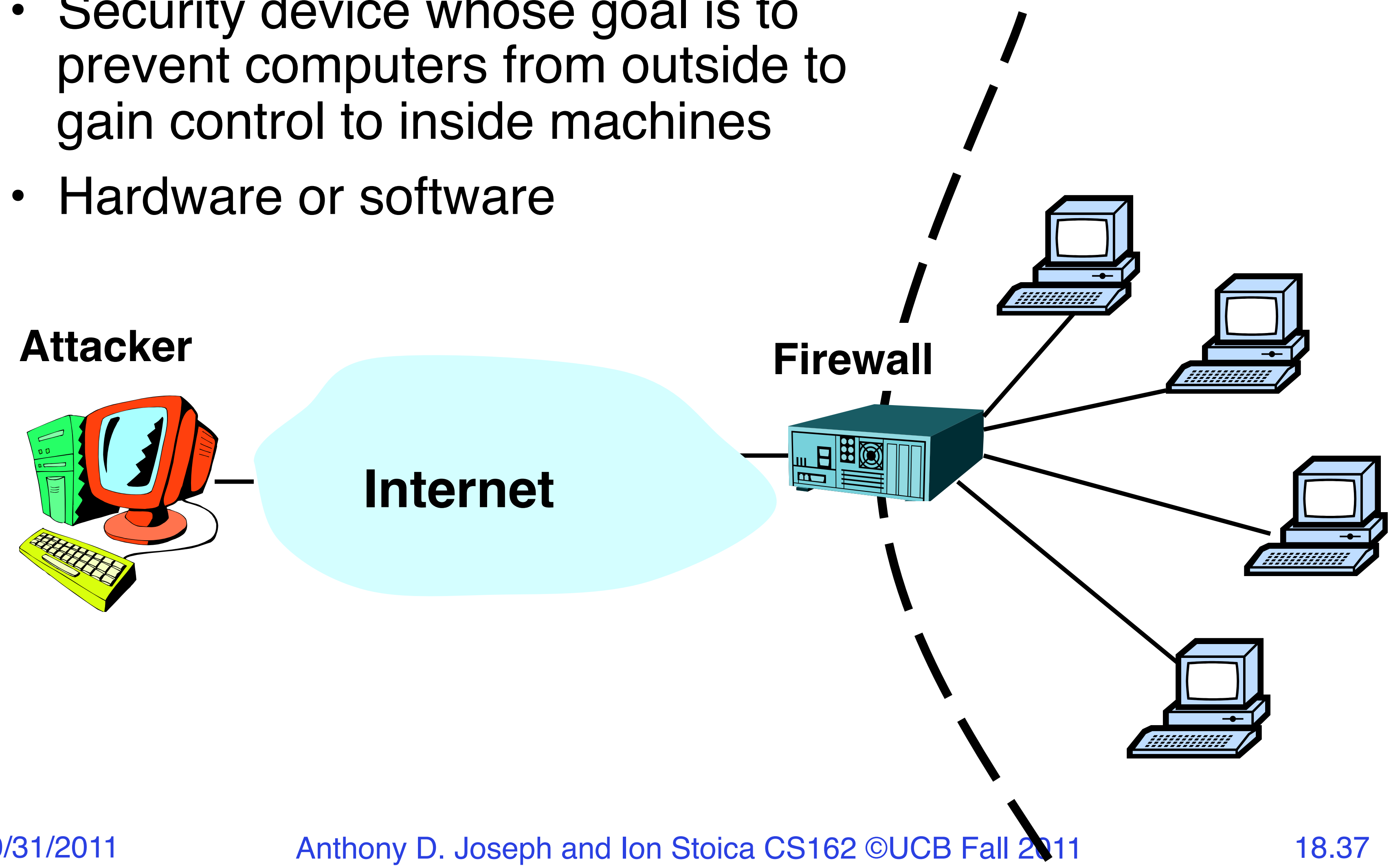
- Software that have had many stack overflow bugs:
  - BIND (most popular DNS server)
  - RPC (Remote Procedure Call, used for NFS)
    - NFS (Network File System), widely used at UCB
  - Sendmail (most popular UNIX mail delivery software)
  - IIS (Windows web server)
  - SNMP (Simple Network Management 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
  – Use code checkers (slow, incomplete coverage)
- 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/social networking
- 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

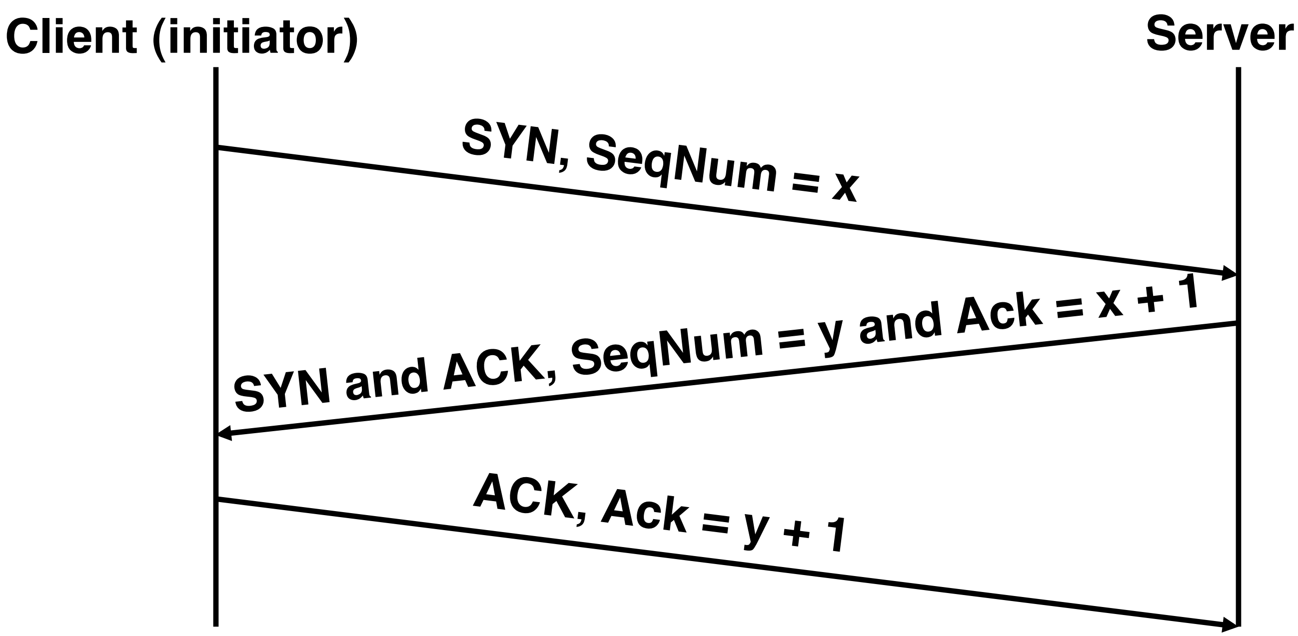
Affect on Victim

- 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

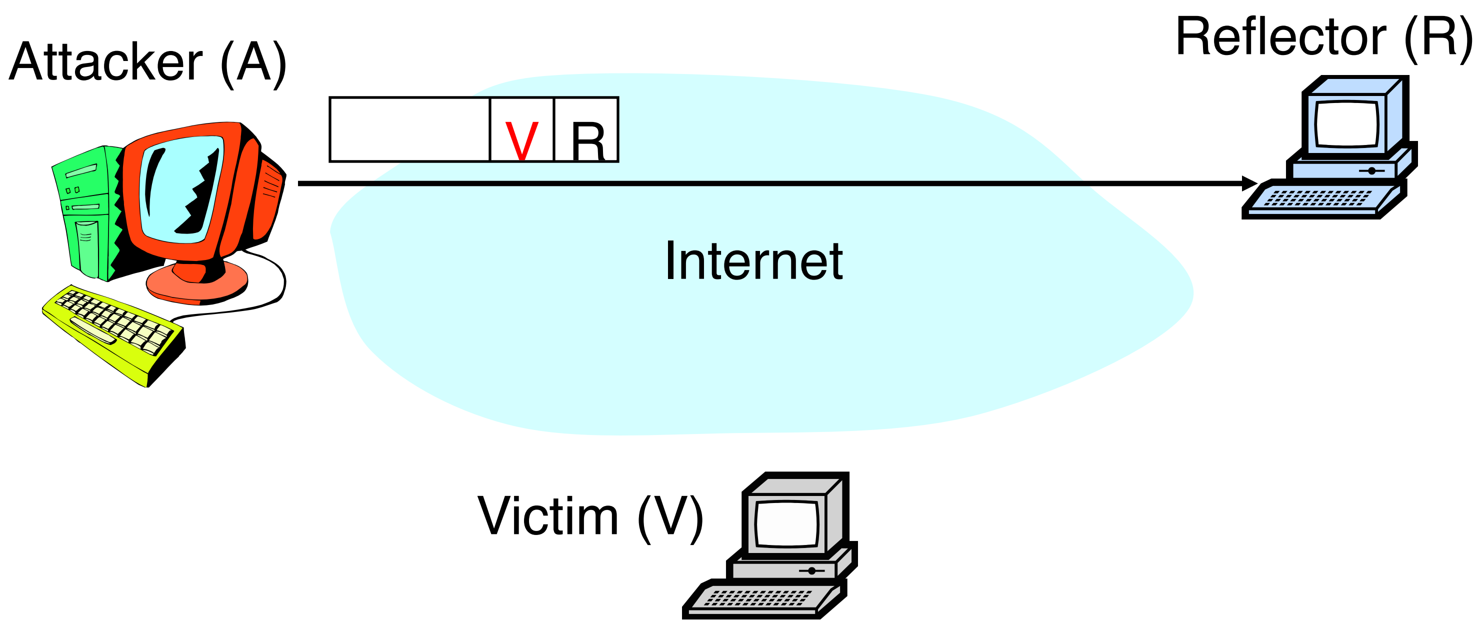
- Attacker: send at max rate TCP SYN with random spoofed source address to victim
  - Spoofing: use a different source IP address than own
  - Random spoofing allows one host to pretend to be many
- Victim receives many SYN packets
  - Send SYN+ACK back to spoofed IP addresses
  - Holds some memory until 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

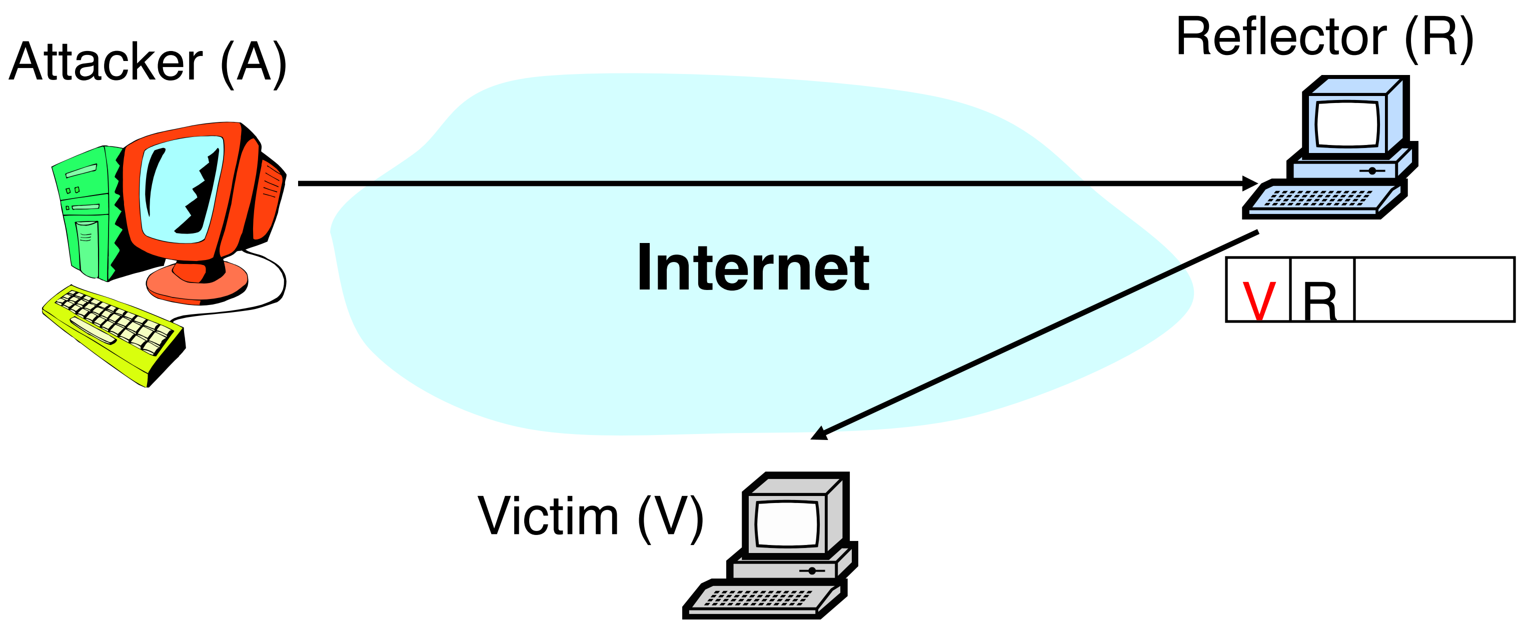
Other Denial-of-Service Attacks

- 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