Week 2: Trojans, Worms, and Attacks

Week 2

Worm

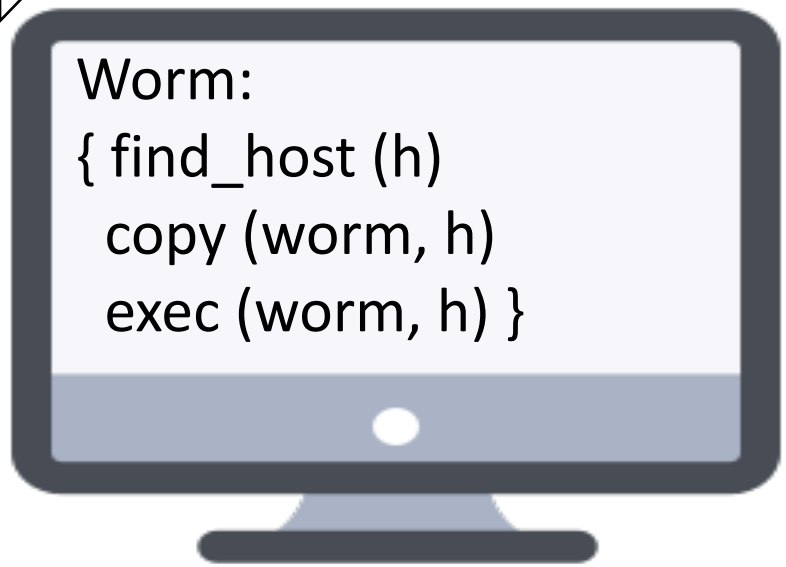
Find Someone's Computer

Copy ***Worm*** to Their Computer

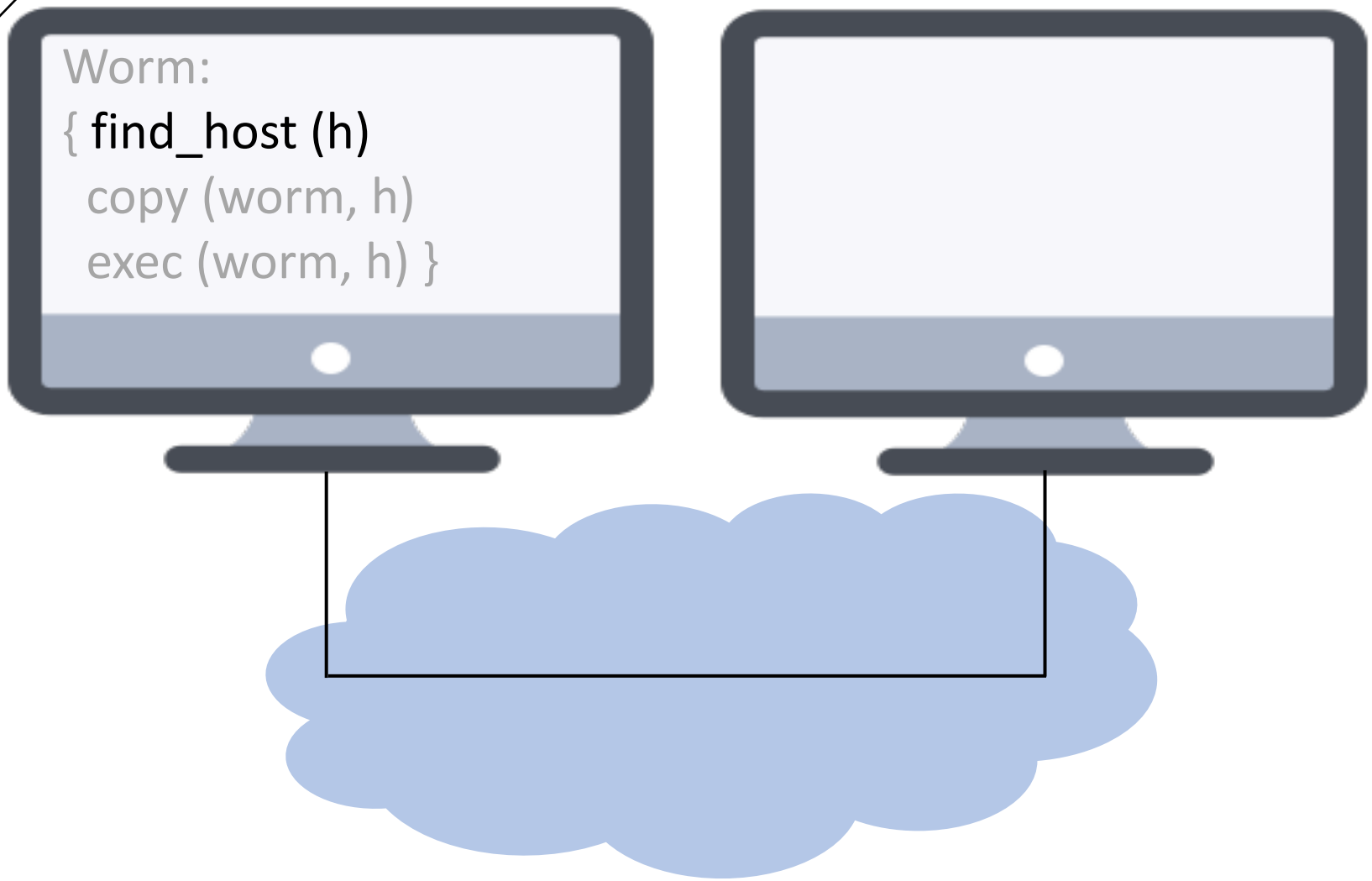
Run ***Worm*** on Their Computer

```
Worm:
{ find_host (h)
  copy (worm, h)
  exec (worm, h) }
```

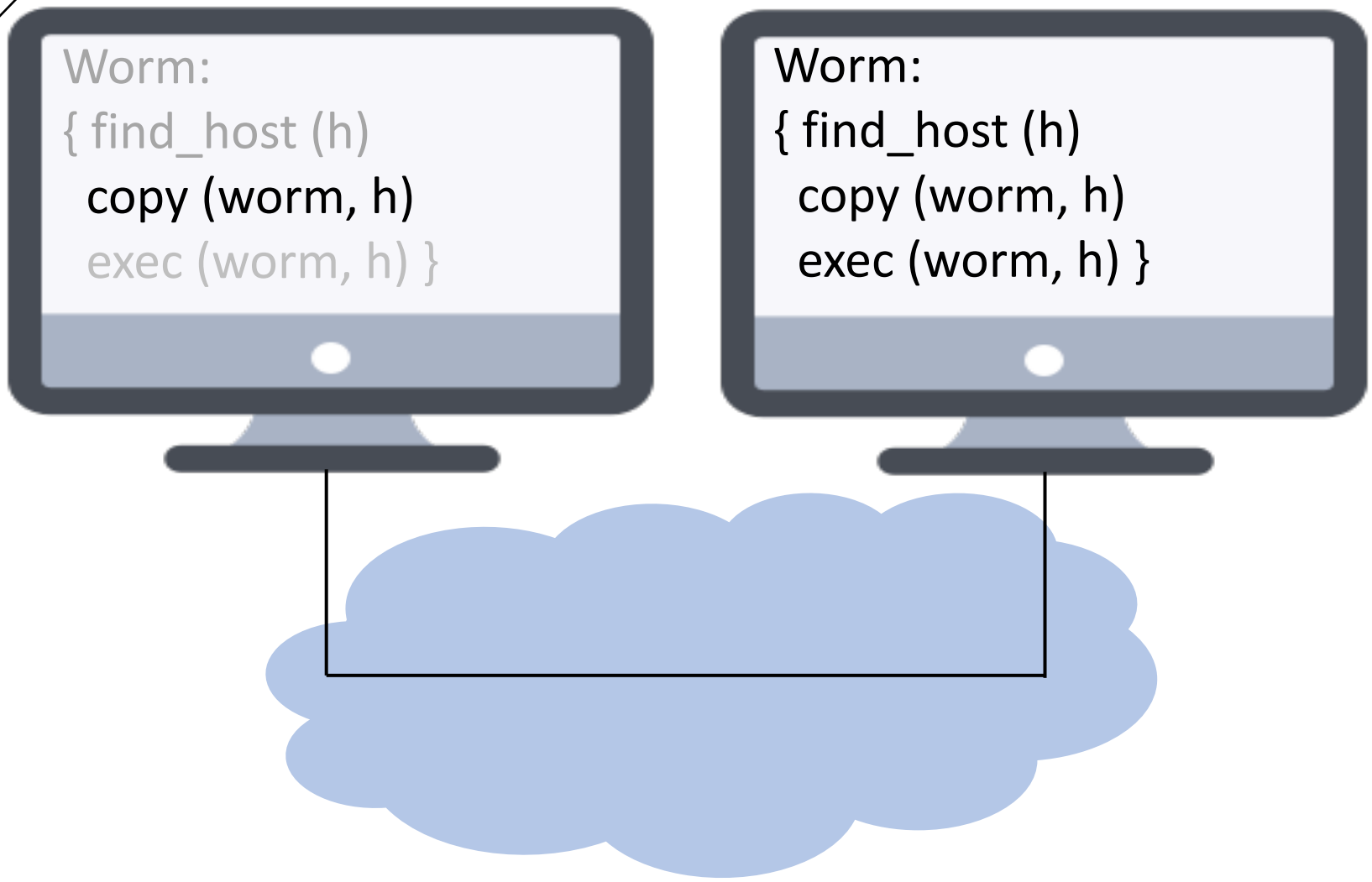
Week 2



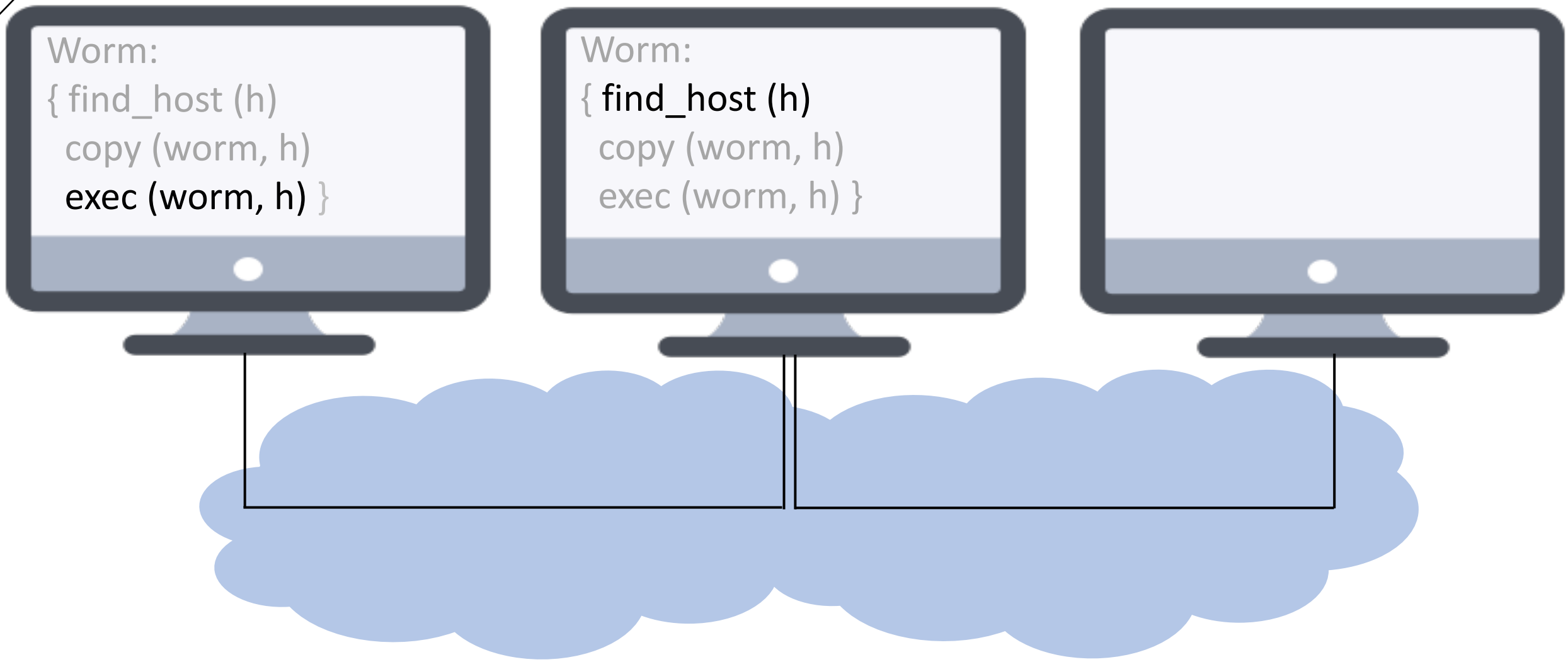
Week 2



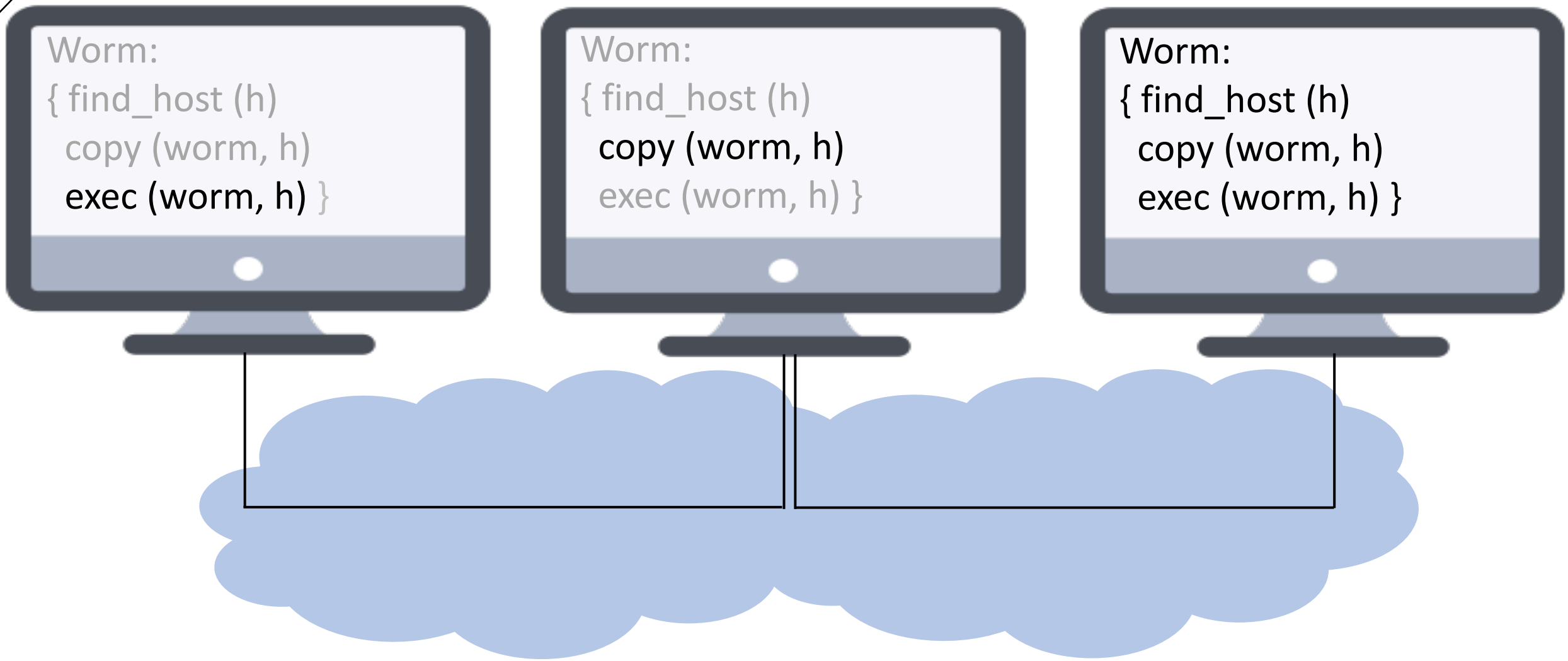
Week 2


Worm:
{ find_host (h)
copy (worm, h)
exec (worm, h) }
Worm:
{ find_host (h)
copy (worm, h)
exec (worm, h) }

Week 2


Worm:
{ find_host (h)
copy (worm, h)
exec (worm, h) }
Worm:
{ find_host (h)
copy (worm, h)
exec (worm, h) }

Week 2


Worm:
{ find_host (h)
copy (worm, h)
exec (worm, h) }
Worm:
{ find_host (h)
copy (worm, h)
exec (worm, h) }
Worm:
{ find_host (h)
copy (worm, h)
exec (worm, h) }

Week 2

The Morris Internet Worm source code

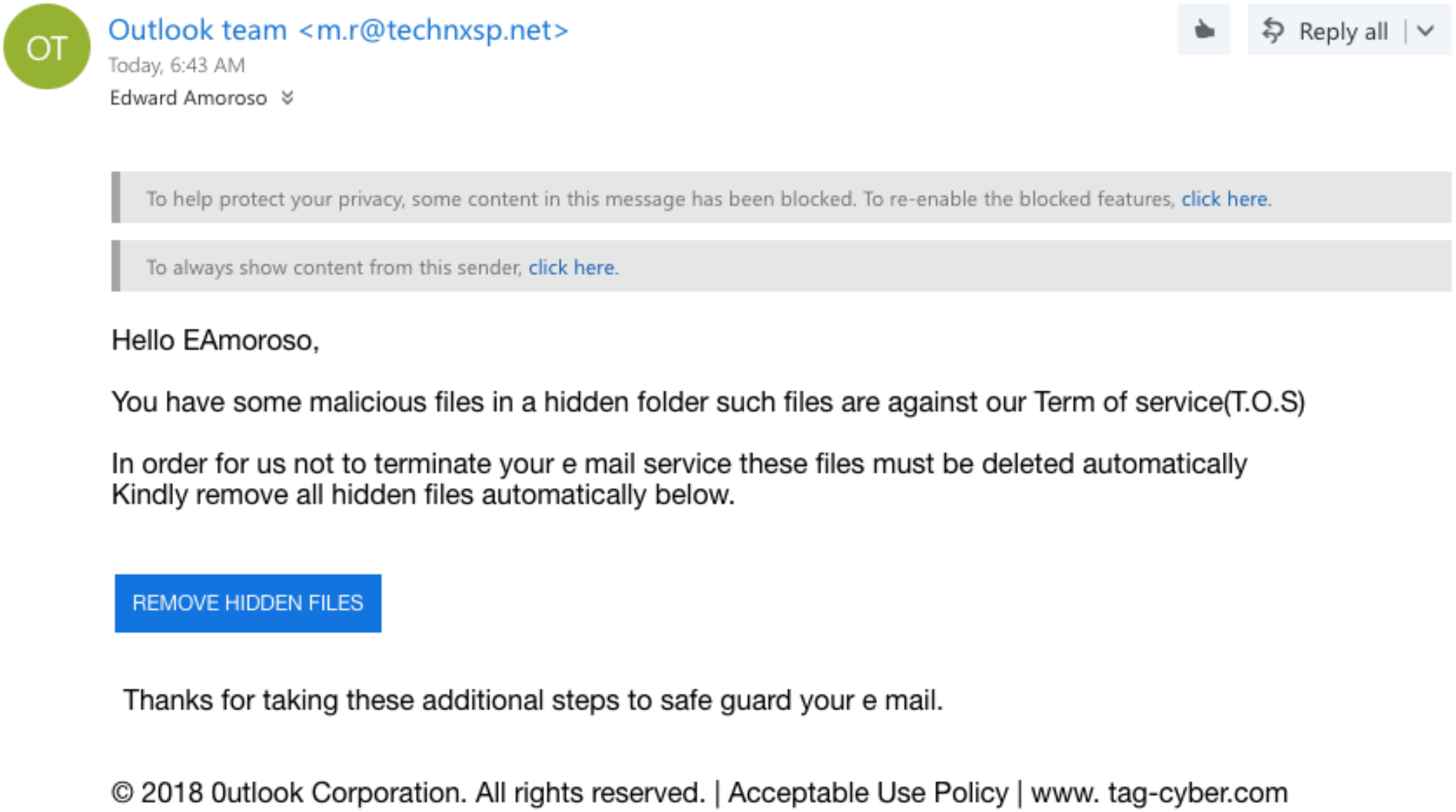
This disk contains the complete source code of the Morris Internet worm program. This tiny, 99-line program brought large pieces of the Internet to a standstill on November 2nd, 1988.

The worm was the first of many intrusive programs that use the Internet to spread.

Computer History Museum

Week 2

Most Common Hacking Method



OT

Outlook team <m.r@technxsp.net>

Today, 6:43 AM

Edward Amoroso

Reply all

To help protect your privacy, some content in this message has been blocked. To re-enable the blocked features, click here.

To always show content from this sender, click here.

Hello EAmoroso,

You have some malicious files in a hidden folder such files are against our Term of service(T.O.S)

In order for us not to terminate your e mail service these files must be deleted automatically
Kindly remove all hidden files automatically below.

REMOVE HIDDEN FILES

Thanks for taking these additional steps to safe guard your e mail.

© 2018 0utlook Corporation. All rights reserved. | Acceptable Use Policy | www. tag-cyber.com

Week 2

Here is a small piece of code that
implements a “login” process for an app:

Login

Print “Enter your name: ”
Get (Name)
Print “Enter your password: ”
Get (Password)
if OK (Name, Password) **then** Permit
  else Deny

End

Making Malicious Insertions Invisible

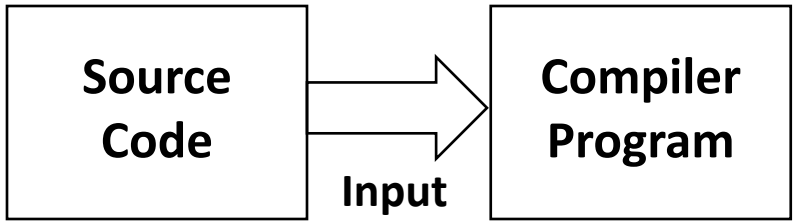
Week 2

Here is a small piece of code that implements a “login” process for an app:

```
Login

Print “Enter your name: ”
Get (Name)
Print “Enter your password: ”
Get (Password)
if OK (Name, Password) then Permit
  else Deny

End
```

Programmers refer to this as **“Source Code”** and use Programming Languages such as Java, C++, and Python

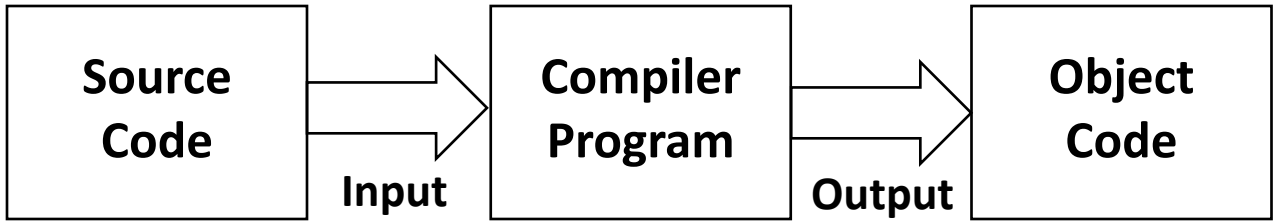
Here is the translation process to make the code executable on a computer:



Programmers obtain **“Compilers”** from software companies such as Microsoft (or get them for free on the Internet)

Week 2

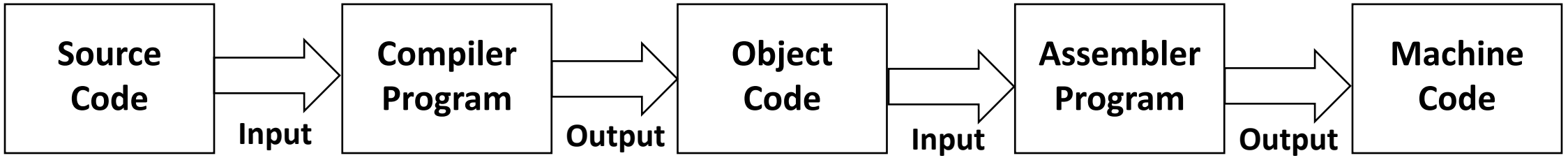
Here is the translation process to make the code executable on a computer:



Compilers “break down” Source Code into more rudimentary building block languages called “**Object Code**” for the computer being used

Week 2

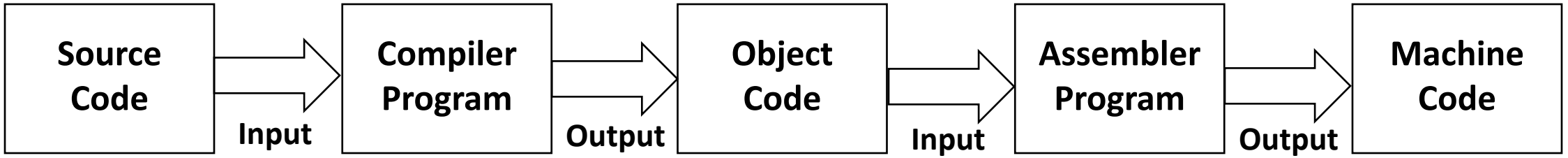
Here is the translation process to make the code executable on a computer:



Assemblers “break down” Object Code into raw binary languages (1’s and 0’s) called “**Machine Code**” for the computer being used

Week 2

Here is the translation process to make the code executable on a computer:



Login

Print "Enter your name: "
Get (Name)
Print "Enter your password: "
Get (Password)
if OK (Name, Password) **then** Permit
  else Deny

End

```
-- Login

LDA   8A34
STA   FF20
LDA   2001
STA   FF21
MOV   2A2B
. . .
```

```
-- Login

0010 1010
1100 1011
0000 1101
0001 1111
1110 1111
. . .
```

Here is the process to insert a Trojan horse into the code executable on a computer:

Login

Print “Enter your name: ”
Get (Name)
Print “Enter your password: ”
Get (Password)
if OK (Name, Password) **then** Permit
  else Deny

End

Here is the process to insert a Trojan horse into the code executable on a computer:

Login

Print “Enter your name: ”
Get (Name)
Print “Enter your password: ”
Get (Password)
if OK (Name, Password) **or** Password = “NYU_LAW1” **then** Permit
  else Deny

End

Week 2

Here is the process to insert a Trojan horse into the code executable on a computer:

Login

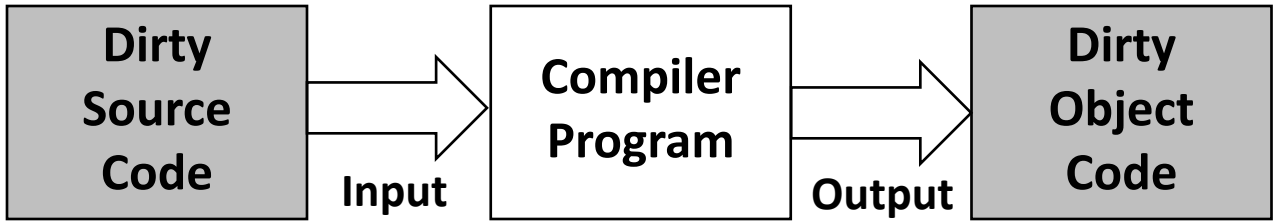
Print “Enter your name: ”
Get (Name)
Print “Enter your password: ”
Get (Password)
if OK (Name, Password) **or** Password = “NYU_LAW” **then** Permit
  else Deny

End

This is a Trojan Horse Insertion called a “Back Door”

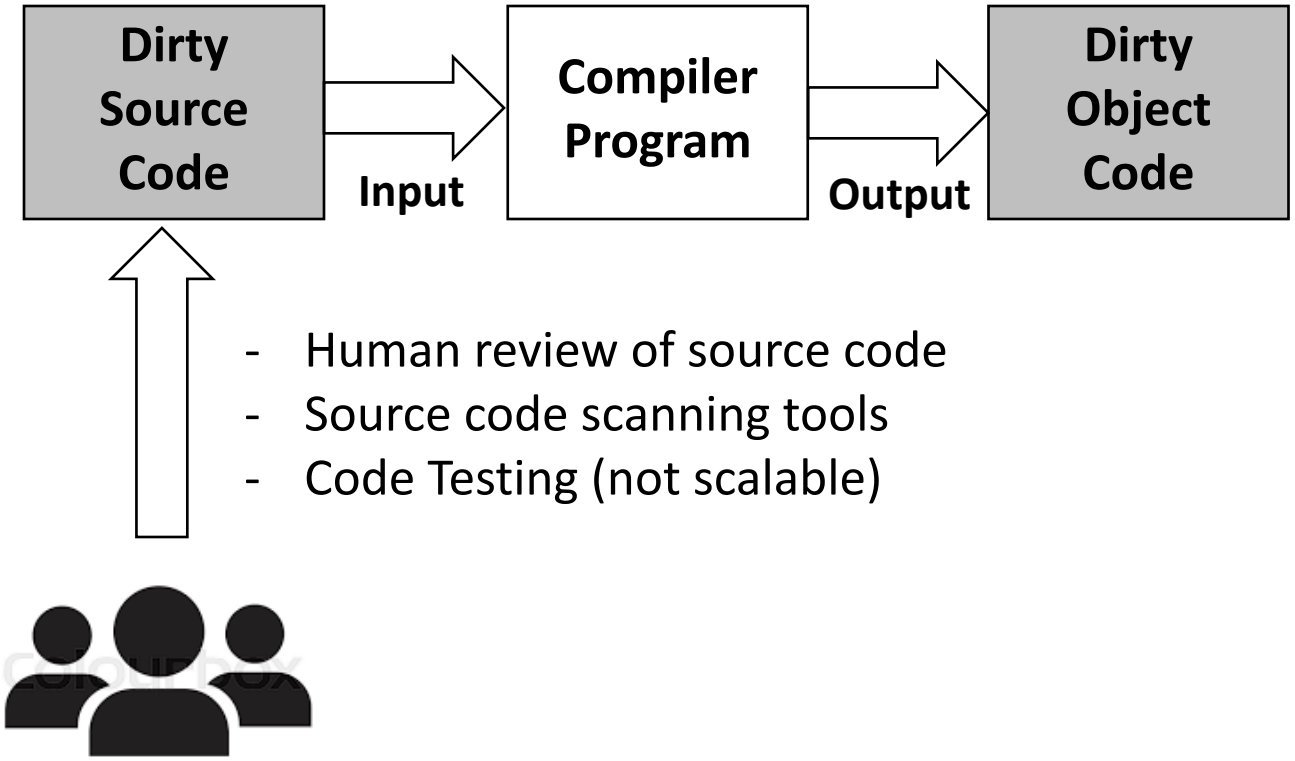
Week 2

Here is the process to translate “Dirty Source Code” into “Dirty Object Code:”



Week 2

Here is the review process to detect “Dirty Source Code” being used to create into “Dirty Object Code:”



Here is how a “Clean” compiler program works:

Compiler

Repeat
    Get (Line of Code)
    Translate (Line of Code)
Until Done

End

Here is how a “Clean” compiler program works:

Compiler

Repeat
    Get (Line of Code)

    Translate (Line of Code)
Until Done

End

Here is how a “Dirty” compiler program works:

Compiler

Repeat
  Get (Line of Code)

  If (Line of Code) = “**If** OK (Name, Password)”

  Translate (Line of Code)
Until Done

End

Week 2

Here is how a “Dirty” compiler program works:

Compiler

Repeat
  Get (Line of Code)

  If (Line of Code) = “**If** OK (Name, Password)”
    Then Translate (“**If** OK (Name, Password) **or** Password = “NYU_LAW”)
  Else

  Translate (Line of Code)
Until Done

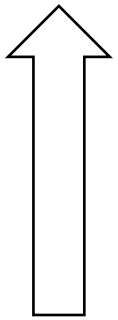
End

Week 2

Here is how a “Dirty” compiler program translates
Clean Source to Dirty Object Code:

Week 2

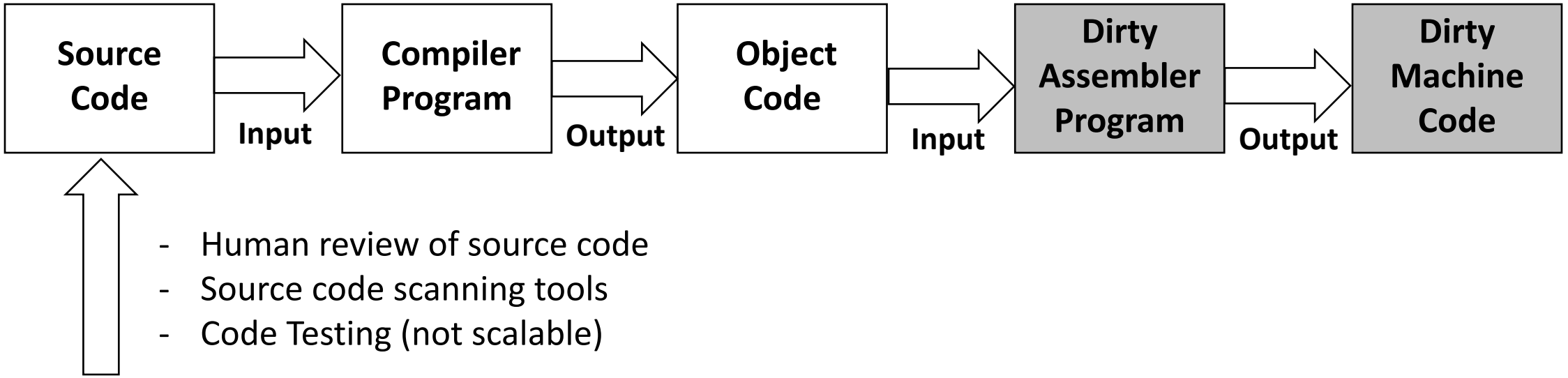
Here is how a “Dirty” compiler program hides Trojan Horse insertions in Object Code:



- Human review of source code
- Source code scanning tools
- Code Testing (not scalable)

Week 2

Here is how a Malicious Group “Hides” Trojan horses deeper into the translation process:



Week 2

Dirty Equipment in CONUS or OCONUS Locations

CIA Model

Confidentiality

Integrity

Availability

Network Transport of Command and Control (C&C), or Telemetry

1. Trojan Horse Designed to quietly "eavesdrop" on US Communications

2. Trojan Horse Designed to quietly "modify" US Operations

3. Trojan Horse Designed to noisily "block" US Operations

Foreign C&C Located in CONUS or OCONUS

Three US National Security Risk Areas of Trojan Horse Insertions

Week 2

1. **Review Vendor Software Development Process**
   - **Read documentation, review checklist answers, interview vendor team**

Normal Risk Mitigation for Vendor-Inserted Trojan Horses

Week 2

1. **Review Vendor Software Development Process**
   - Read documentation, review checklist answers, interview vendor team
2. **Inspect Vendor Hardware and Software**
   - Review source code, use static tools to scan software, read documentation

Normal Risk Mitigation for Vendor-Inserted Trojan Horses

Week 2

1. **Review Vendor Software Development Process**
   - Read documentation, review checklist answers, interview vendor team
2. **Inspect Vendor Hardware and Software**
   - Review source code, use static tools to scan software, read documentation
3. **Specify Vendor Integrity Requirements in Contract**
   - Include language in vendor contracts, specify consequences of Trojan detection

Normal Risk Mitigation for Vendor-Inserted Trojan Horses

Week 2

1. **Review Vendor Software Development Process**
   - Read documentation, review checklist answers, interview vendor team
2. **Inspect Vendor Hardware and Software**
   - Review source code, use static tools to scan software, read documentation
3. **Specify Vendor Integrity Requirements in Contract**
   - Include language in vendor contracts, specify consequences of Trojan detection
4. **Monitor Community for Evidence of Vendor Issues**
   - Digital Risk Management of vendor, review hacker community chatter

Normal Risk Mitigation for Vendor-Inserted Trojan Horses

Week 2

1. **Review Vendor Software Development Process**
   - Read documentation, review checklist answers, interview vendor team
2. **Inspect Vendor Hardware and Software**
   - Review source code, use static tools to scan software, read documentation
3. **Specify Vendor Integrity Requirements in Contract**
   - Include language in vendor contracts, specify consequences of Trojan detection
4. **Monitor Community for Evidence of Vendor Issues**
   - Digital Risk Management of vendor, review hacker community chatter
5. **Proxy Outbound Communications to Unknown Sources**
   - Gateway interrupts all outbound communications, check target URL

Normal Risk Mitigation for Vendor-Inserted Trojan Horses

Week 2

1. **Use Social Engineered Deception to Expose Trojan Back Door Access**
   - **Ex/ Call vendor in distress, begging for assisted access to procured system**

Advanced Risk Mitigation for Nation-State Controlled Trojan Horses

Week 2

1. **Use Social Engineered Deception to Expose Trojan Back Door Access**
   - **Ex/ Call vendor in distress, begging for assisted access to procured system**
2. **Comparatively Analyze Multiple Instances of Vendor Product**
   - **Ex/ Purchase same product in different contexts (incl. critical and non-critical)**

Advanced Risk Mitigation for Nation-State Controlled Trojan Horses

Week 2

1. **Use Social Engineered Deception to Expose Trojan Back Door Access**
   - **Ex/ Call vendor in distress, begging for assisted access to procured system**
2. **Comparatively Analyze Multiple Instances of Vendor Product**
   - **Ex/ Purchase same product in different contexts (incl. critical and non-critical)**
3. **Learn from Present or Former Employees About Trojan Insertions**
   - **Ex/ Former Microsoft employees admit to Trojans in Word and Excel**

Advanced Risk Mitigation for Nation-State Controlled Trojan Horses

Week 2

1. Use Social Engineered Deception to Expose Trojan Back Door Access
   - Ex/ Call vendor in distress, begging for assisted access to procured system
2. Comparatively Analyze Multiple Instances of Vendor Product
   - Ex/ Purchase same product in different contexts (incl. critical and non-critical)
3. Learn from Present or Former Employees About Trojan Insertions
   - Ex/ Former Microsoft employees admit to Trojans in Word and Excel
4. Governments Can Utilize Surveillance and Signals Intelligence
   - Ex/ Lawful intercepts can provide evidence of integrity issues

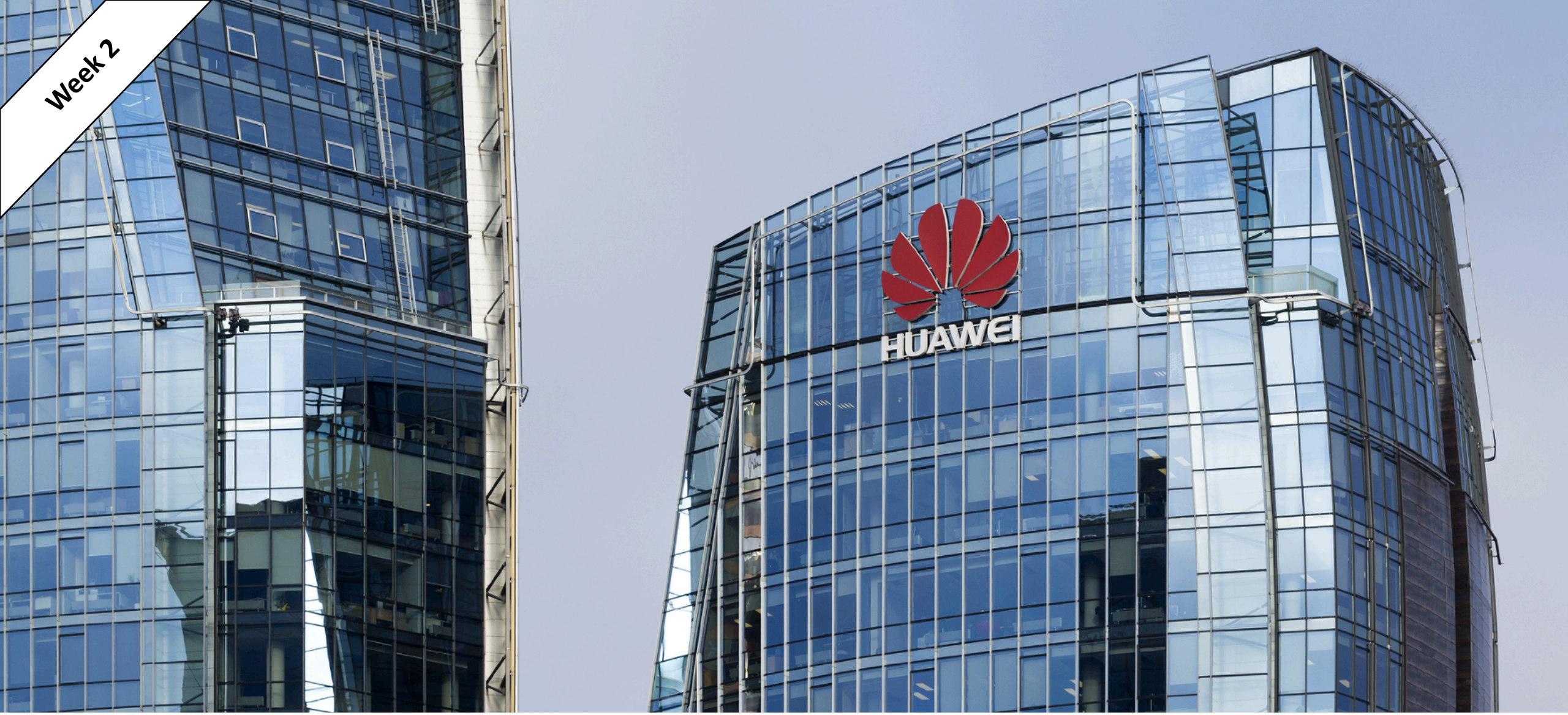
Advanced Risk Mitigation for Nation-State Controlled Trojan Horses

Week 2

1. Use Social Engineered Deception to Expose Trojan Back Door Access
   - Ex/ Call vendor in distress, begging for assisted access to procured system
2. Comparatively Analyze Multiple Instances of Vendor Product
   - Ex/ Purchase same product in different contexts (incl. critical and non-critical)
3. Learn from Present or Former Employees About Trojan Insertions
   - Ex/ Former Microsoft employees admit to Trojans in Word and Excel
4. Governments Can Utilize Surveillance and Signals Intelligence
   - Ex/ Lawful intercepts can provide evidence of integrity issues
5. Governments Can Embed Developers Into Target Vendor Environments
   - Ex/ Confidential relationship with developers employed by vendor

Advanced Risk Mitigation for Nation-State Controlled Trojan Horses

Week 2



US National Policy Regarding Chinese Telecommunications Firms

Major (non-Chinese) Telecommunications Breach (No Trojans)

Week 2

Stage 1: IFS Variable

- Set IFS variable to include ‘/’
- “/etc/file” same as “etc file”

Classic Unix Kernel Attack

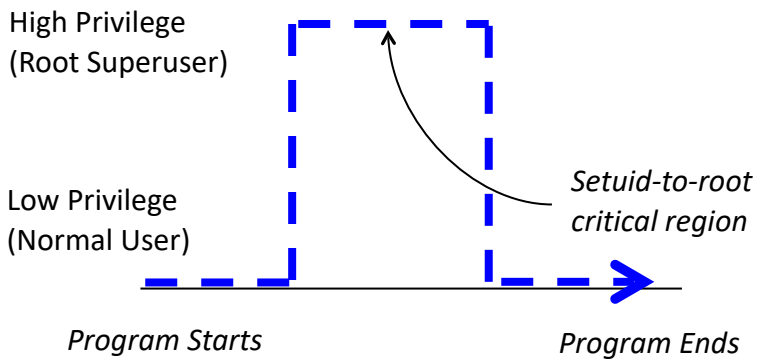
Week 2

Stage 1: IFS Variable

- Set IFS variable to include ‘/’
- “/etc/file” same as “etc file”

Stage 2: Find setuid-to-root program

- Allows increase in privilege
- Normal user to Unix “Root”

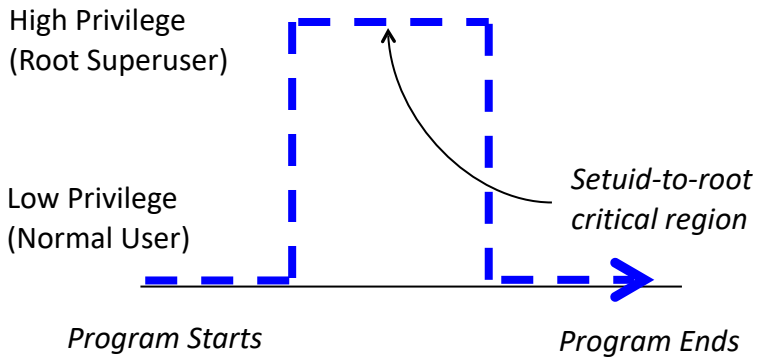


Classic Unix Kernel Attack

Week 2

Stage 1: IFS Variable
- Set IFS variable to include ‘/’
- “/etc/file” same as “etc file”

Stage 2: Find setuid-to-root program
- Allows increase in privilege
- Normal user to Unix “Root”



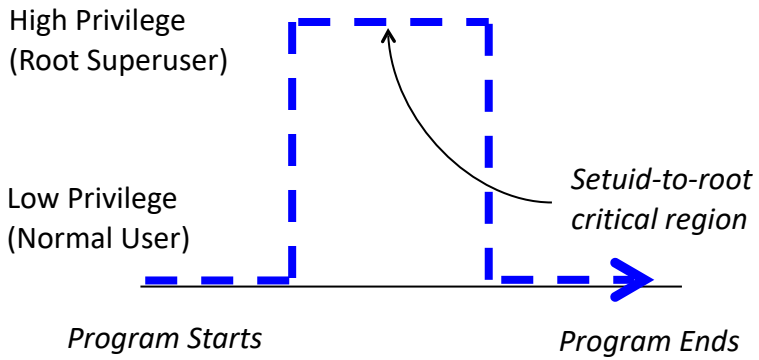
Stage 3: Notice Source Code in “At” Program
- Program has setuid-to-root critical region
- Region includes “exec /etc/protect/file”

Classic Unix Kernel Attack

Week 2

Stage 1: IFS Variable
- Set IFS variable to include '/'
- "/etc/file" same as "etc file"

Stage 2: Find setuid-to-root program
- Allows increase in privilege
- Normal user to Unix "Root"



Stage 3: Notice Source Code in "At" Program
- Program has setuid-to-root critical region
- Region includes "exec /etc/protect/file"

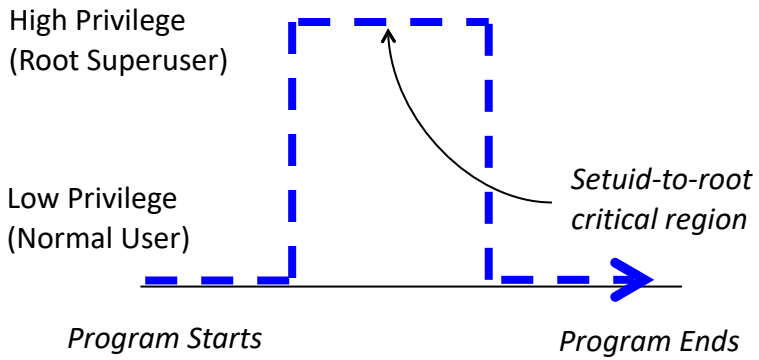
Stage 4: Unix Shell Program
- Allows copying ("cp /bin/sh myshell")
- Copied program inherits privileges

Classic Unix Kernel Attack

Week 2

Stage 1: IFS Variable
- Set IFS variable to include '/'
- "/etc/file" same as "etc file"

Stage 2: Find setuid-to-root program
- Allows increase in privilege
- Normal user to Unix "Root"



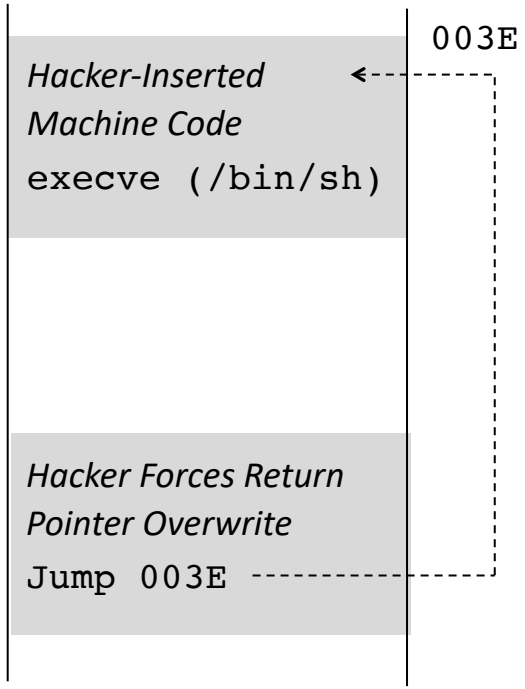
Stage 3: Notice Source Code in "At" Program
- Program has setuid-to-root critical region
- Region includes "exec /etc/protect/file"

Stage 4: Unix Shell Program
- Allows copying ("cp /bin/sh myshell")
- Copied program inherits privileges

Unix Kernel Attack:
- Step 1: Set IFS to include '/'
- Step 2: Write shell program to copy /bin/sh to myshell
- Step 3: Run "at" program
- Result: myshell is root shell owed by me!

Classic Unix Kernel Attack

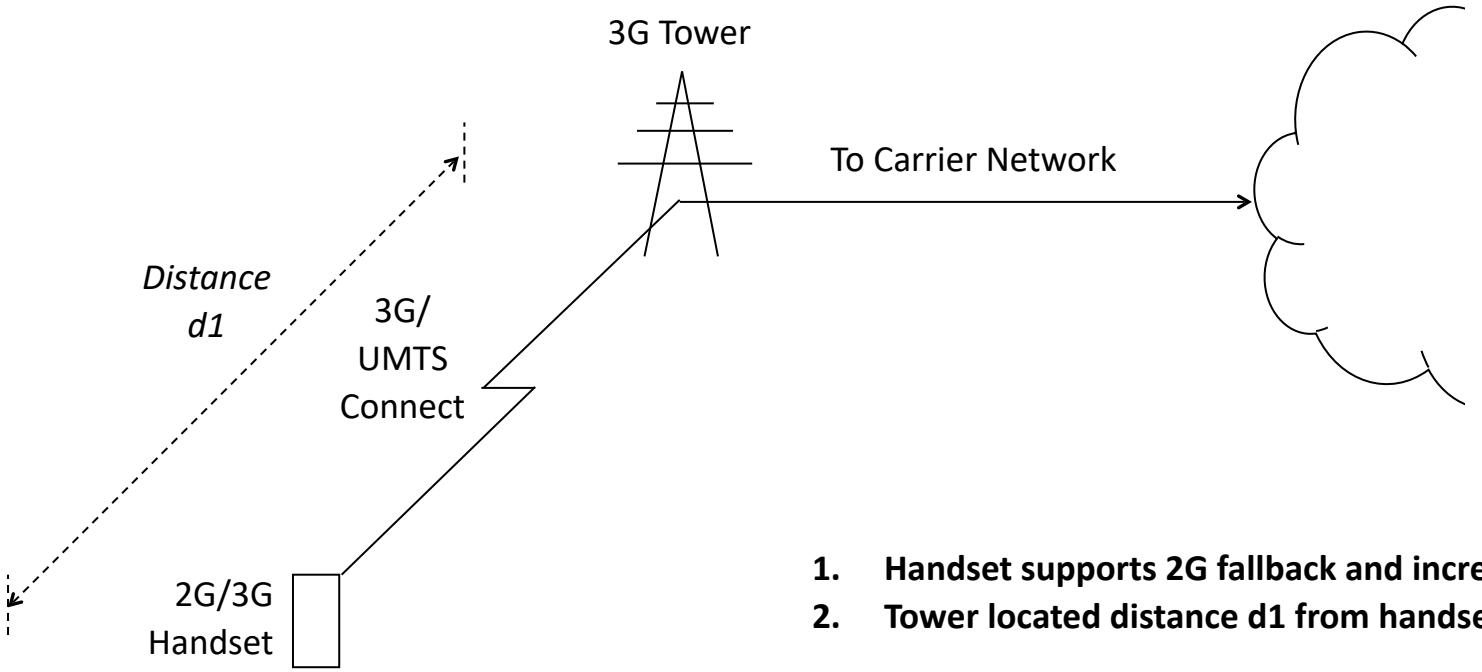
Week 2




```
void overflow_function(char *string)
    {
            char buffer[16];
            strcpy(buffer, string);
            return;
    }

void main()
    {
            char buffer[256];
            int i;

            for(i=0; i<255; i++)
                    large_buffer[i]='A';

                    overflow_function(large_buffer);
    }
```

Classic Buffer Overflow Attack Code

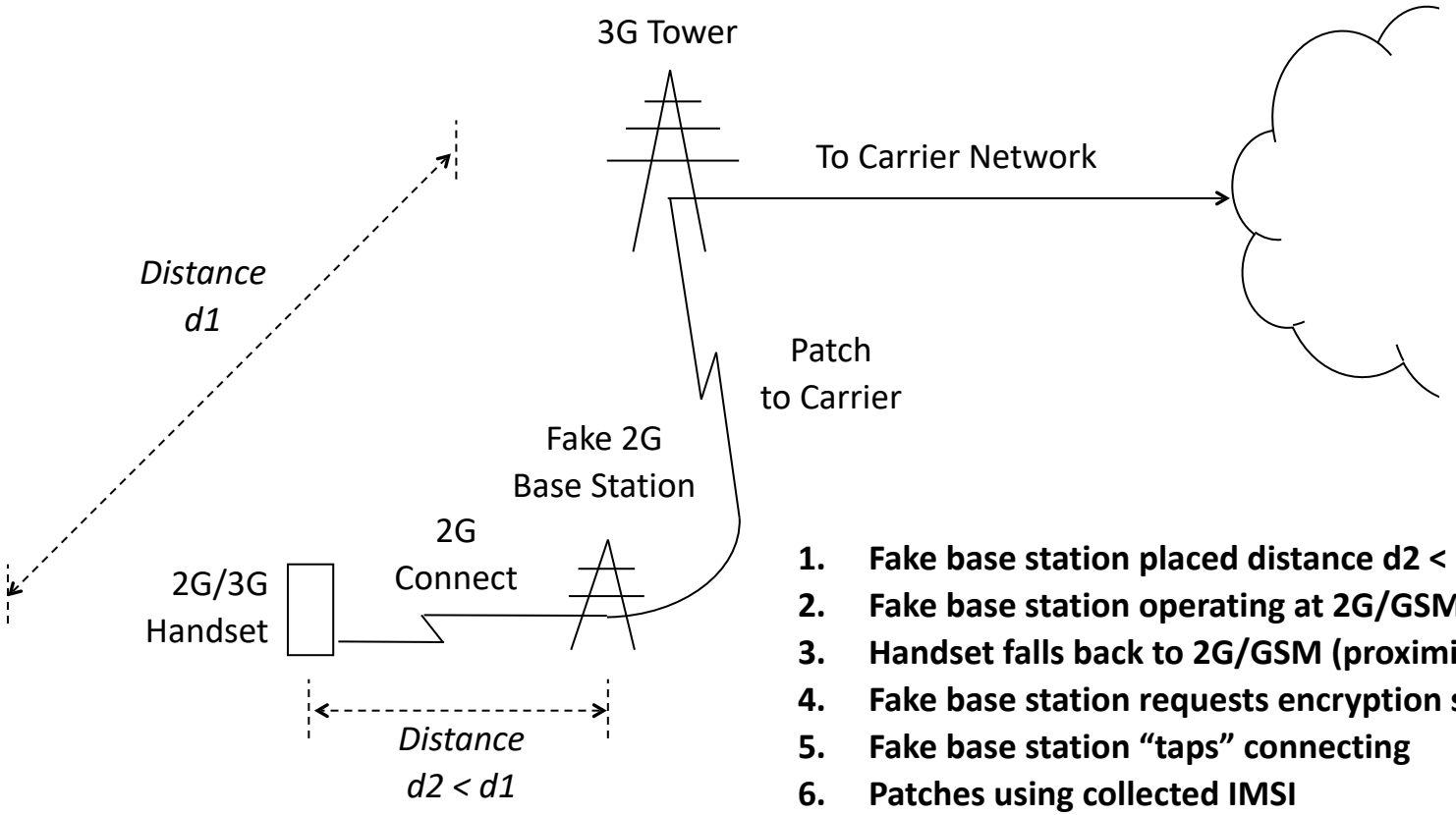
Week 2



1. **Handset supports 2G fallback and increased coverage**
2. **Tower located distance d1 from handset**

3G Mobile Intercept Attack (Fake Base Station)

Week 2



1. **Fake base station placed distance d2 < d1 from handset**
2. **Fake base station operating at 2G/GSM**
3. **Handset falls back to 2G/GSM (proximity rule)**
4. **Fake base station requests encryption suppression**
5. **Fake base station "taps" connecting**
6. **Patches using collected IMSI**

3G Mobile Intercept Attack (Fake Base Station)