CS 422/522 Design & Implementation of Operating Systems

Lecture 19: Security and Trust

Zhong Shao
Dept. of Computer Science
Yale University

Self-replicating program (Thompson 84)

```
char s[] = {
        '\t',
        '0',
        '\n',
        '}',
        ';',
        '\n',
        '\n',
        '/',
        '*',
        '\n',
        (213 lines deleted)
        0
};
```

```
/*
 * The string s is a
 * representation of the body
 * of this program from '0'
 * to the end
 */

main( )
{
    int i;
    printf ("char\ts[]={\n");
    for (i=0; s[i]; i++)
        printf("\t%d, \n", s[i]);
    printf("%s", s);
}
```

Bootstrap a compiler (how a compiler learn?)

Initial C compiler

```
…
c = next ();
if (c != '\\')
      return (c);
c = next ();
if (c == '\\')
      return ('\\');
if (c == 'n')
       return ('\n');
….
```

C compiler that supports '\v'

```
…
c = next ();
if (c != '\\')
      return (c);
c = next ();
if (c == '\\')
      return ('\\');
if (c == 'n')
       return ('\n');
if (c == 'v')
        return (11);
….
```

C compiler that supports and uses '\v'

```
…
c = next ();
if (c != '\\')
      return (c);
c = next ();
if (c == '\\')
      return ('\\');
if (c == 'n')
       return ('\n');
if (c == 'v')
        return ('\v');
….
```

Thompson's "cutest" program

- Proposed by Ken Thompson in his Turing award lecture
- Bury trojan horse in binaries, so no evidence in the source
- Replicates itself to every Unix system in the world, and even to new Unixes on new platforms
- Two steps:
  - Make it possible (easy)
  - Hide it (tricky)
- Step 1: Modify login.c (**code snippet A**)

```
If (name == "ken")
      Don't check password
      Log in as root
```

- Next step: hide change, so no one can see it

Modify the C compiler

- Step 2: Instead of having the code in login.c, put it in the compiler (code snippet B):

```
If see trigger
    Insert A into input stream
```

- Whenever the compiler sees a trigger (/* gobbledygook */), puts A into input stream of the compiler
  - Now don't need A in login.c, just need the trigger
  - Need to get rid of the problem in the compiler

```
login.c:

   /* gobbledygook */
```

$\longrightarrow$

```
Compiler:
  if (str == "gobbledygook")
     emit code for
     trojan horse
```

Modify the C compiler (cont'd)

- Step 3: modify compiler to have (code snippet C)

```
If see trigger2
    Insert B + C into input stream
```

  - This is where the self-replicating code comes in!
  - Question: can you write a program that has no inputs and outputs itself (or a superset of itself)?

- Step 4: Compile the compiler with snippet C present
  - Now the intelligence is in the binary

```
Compiler code:

  /* gobbledygook2 */
```

$\longrightarrow$

```
Compiler binary:
  if (str == "gobbledygook2")
     emit code for trojan check
     and replicate this check
```

Self-replicating program (contd.)

- Step 5: replace snippet C with trigger2
  - Result: all of the intelligence is only in the binary and not in the source code!
- If you use binary to compile "login.c", it will recognize trigger to emit backdoor
- If you use binary to compile the compiler, it will recognize trigger2
  - It will emit code in the generated binary to watch out for invocations when you are compiling "login.c" or the compiler itself
- Summary: can't stop loopholes, can't tell if it's happened, can't get rid of it!

The rest of this lecture

- Types of misuse of computers:
  - Accidental
  - Intentional
- Protection is to prevent either accidental or intentional misuse; security is to prevent intentional misuse
- The security environment
- User authentication
- Attacks from inside the system
- Attacks from outside the system
- Trusted systems

The security environment: threats

Goal	Threat
Data confidentiality	Exposure of data
Data integrity	Tampering with data
System availability	Denial of service

Security goals and threats

Intentional misuse: intruders

Common categories:

1. Casual prying by non-technical users
2. Snooping by insiders
3. Determined attempt to make money
4. Commercial or military espionage

Accidental misuse

Common Causes

1. Acts of God
   - fires, floods, wars
2. Hardware or software errors
   - CPU malfunction, bad disk, **program bugs**
3. Human errors
   - data entry, wrong tape mounted

Three pieces to security

- ***Authentication***
  - Who is the user?
- ***Authorization***
  - Who is allowed to do what
- ***Enforcement***
  - Make sure people do only what they are supposed to do

Loophole in any of these means problem:

1. Login as super user and you have circumvented authentication
2. Login as self and you can do anything you want to your own resources. What if you run some program that decides to erase all your files?
3. Can you trust software to correctly enforce decisions about 1+2?

Authentication

- Common approach: passwords. Shared secret between you and the machine --- since only you know the password, machine can assume it is you.
- Private key encryption --- use an encryption algorithm that can be easily reversed if given the correct key (and hard to reverse without the key)
- Public key encryption --- an alternative (which separates authentication from secrecy)

Problems with using passwords

- System must keep copy of secret, to check against passwords. What if malicious user gains access to this list of passwords ?
- Encryption --- transformation that is difficult to reverse without the right key

  Unix /etc/passwd file:

  passwd → one way transform (hash) → encrypted passwd

  System stores only encrypted version, so OK even if someone reads the file!

  When you type in your password, system compares encrypted versions

Passwords must be long and obscure

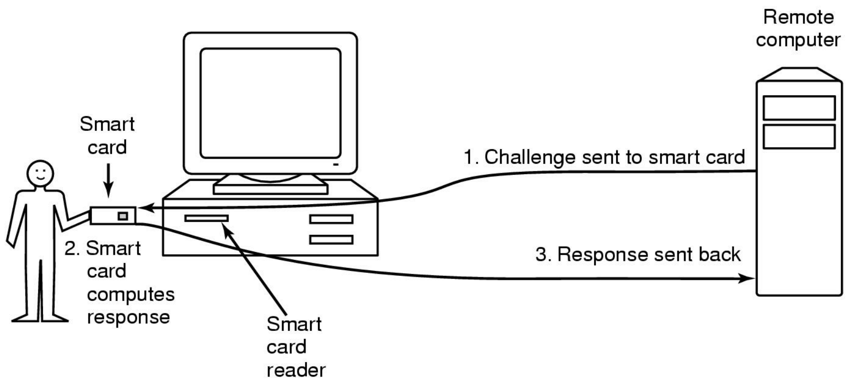
- Paradox: short passwords are easy to crack; long ones, people write down.
- Technology means we have to use longer passwords. Unix initially required only lowercase 5 letter passwords

  How long for an exhaustive search? 26^5 = 10 million

  In 1975, 10ms to check a password → 1 day
  In 2015, less than 10ms to do the entire search

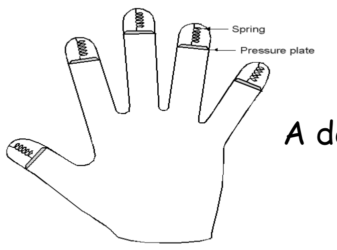
  Some people choose even simpler passwords such as English words --- taken even less time to check for all words in the dictionary.

Authentication using a physical object



- Magnetic cards
  - magnetic stripe cards
  - chip cards: stored value cards, smart cards

Authentication using biometrics



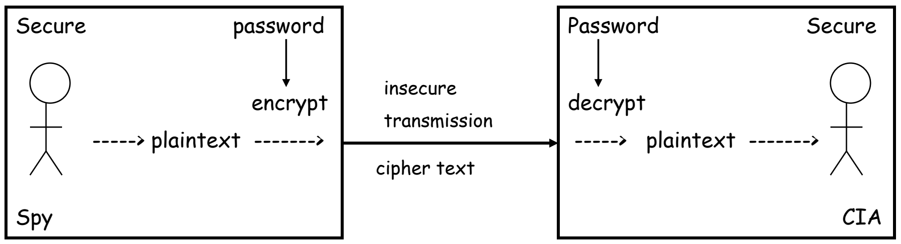
A device for measuring finger length.

Other ideas:

- fingerprints, voices, retinal pattern analysis
- crazy one: urine samples (cats do it this way);

Private key encryption

- Two roles for encryption
  - Authentication
  - Secrecy --- I don't want anyone to know this data



1. From cipher text, can't derive plain text (decode) without password;
2. From plain text and ciper text, can't derive password!

Private key encryption (cont’d)

- How do you get shared secret in both places ? Use **authentication server** (example: Kerberos)
- Main idea:
  - Server keeps list of passwords, provides a way for parties, A and B, to talk to one another, as long as they trust server.
- Notations
  - Kxy is a key for talking between x and y
  - K[…] means encrypt message (…) with the key K.

Example: using an authentication server

- A asks server for key

  A → S (Hi, I’d like a key for talking between A and B)

- Server gives back special session key encrypted using B’s key

  S → A **Ksa[** use Kab; **Ksb[** This is A! Use Kab**]** **]**

- A gives B the ticket

  A → B **Ksb[** This is A! Use Kab **]**

Using authentication server (cont' d)

Lots of details:

- Add in time stamps to limit how long a key will be used and to prevent a machine from replaying messages later
- Also have to include encrypted checksums to prevent malicious user from inserting stuff into the message or changing the message!
- Want to minimize number of times password must be typed in, and minimize amount of time password is stored on machine --- ask for a temporary password and use the real password for authentication.

Public key encryption

- What if A and B don't share a trusted authentication server? Use **public key encryption** --- each key is now a pair **(Kpublic, Kprivate)**

- With private key system (it is symmetric!)

  K[text] = ciphertext   K[ciphertext] = text

- With public key system

  Kpublic[text] = ciphertext   Kprivate[ciphertext] =text
  Kprivate[text] = ciphertext'   (not same ciphertext as above)
  Kpublic[ciphertext'] = text

  Can't derive Kpublic from Kprivate and vice versa
  Idea: Kprivate kept secret, Kpublic put in a telephone directory

Example: using public key encryption

- Authentication:
  Kprivate[I am Anthony! **]**
  Everyone can read it, but only I can send it!
- Secrecy:
  Kpublic [Hi! **]**
  Anyone can send it, but only the target can read it
- Secure communication
  K' public [Kprivate [I am Anthony! **]** Hi! **]**
  Only I can send it, and only you can read it!

Implementing public key encryption

Public encryption key: (e, n)
Private key: (d, n)
where e, d, n are positive integers
n = p x q where p & q are two large (>=100 digits) prime numbers
d satisfies GCD (d, (p-1)x(q-1)) = 1
e satisfies (e x d) mod (p-1)x(q-1) = 1

Each message is represented as an integer between 0..n-1
(long message can be broken into smaller messages and each can be represented as such an integer)

Encyption function: E(m) = m^e mod n = C
Decyption function: D(C) = C^d mod n

Authorization

- About who can do what
- Policy: access control matrix
- Mechanisms:
  - Access control list --- store all permissions for all users with each object
  - Capability list --- each process stores all objects the process has permission to touch
- The real problem
  - How fine-grained should authorization be?

  Example: suppose you buy a copy of new game from "Joe's Game World" and then run it. It is running with your userid. It removes all the files you own including the project due next day. How to prevent this?

Authorization (cont'd)

- Have to run the program using some special games userid (which has no write privileges).
- What if the game needs to write out a file recording scores? Would need to modify the special userid to allow to write to a special file.
- What about other non-game programs (e.g., Quicken)? Need to create another special userid.
- What about word processor programs ? Need to have read/write access to entire categories of files.

Semi-satisfactory solution: only use software from sources you trust!

Authorization (cont’d)

- Bigger risk:
  - Programs can appear on your machine in the form of macros attached to your documents (e.g., MS Word or Excel)
  - Java applets or Javascript programs that are part of the web pages!

Macros typically run with full privileges and may get automatically invoked as part of initially accessing a document, or as part of saving the document later on.

Macros can be used as virus vectors --- replicating themselves when documents are opened and copied.

Java applets are sand-boxed: the JVM inside the web browser runs them with no privileges except the ability to send and retrieve data from the server that the applet is from.

Authorization (cont’d)

- Sophisticated applets need limited access to the resources of the client machine !

- The general problems
  - How do I specify the exact privileges that something running on my behalf should have?
  - How to avoid making this specification task so onerous that nobody will put up with it.

Enforcement

- Enforcer checks passwords, access control lists, etc.
- Any bug in enforcer means: way for malicious user to gain ability to do anything
  - In Unix, super user has all the powers of the Unix kernel --- can do anything!
  - Because of coarse-grained access control, lots of stuff has to run as super user in order to work. If bug in any one of these programs, you are hosed!
- Bullet-proof enforcer --- only known way is to make enforcer as small as possible.

  (also known as "Trusted Computing Base" --- TCB)

Class of security problems

- **Abuse of privilege** --- if the super user is evil, we are all in trouble. Solution: eliminate super user!
- **Impostor** --- break into system by pretending to be someone else.
- **Trojan horse** --- appears helpful but really harmful
  - One army gave another a present of a wooden horse, army hidden inside
- **Salami attack** --- steal and corrupt something a little bit at a time

  (partial pennies from bank interest ……)
- **Eavesdropping** --- tap onto Ethernet and see everything typed in.

More attack examples

- Trojan horses
- Login spoofing
- Logic bombs
- Trap doors
- Buffer overflow
- Covert channels
- Tenex --- early 70's, BBN
- Internet worms
- Viruses
- Ken Thompson's self-replicating programs
- Stuxnet, Heartbleed, car hacking, …

Trojan horses

- Free program made available to unsuspecting user
  - Actually contains code to do harm
- Place altered version of utility program on victim's computer
  - trick user into running that program

Login spoofing

Login:
Login:
(a)
(b)

(a) Correct login screen
(b) Phony login screen

Logic bombs

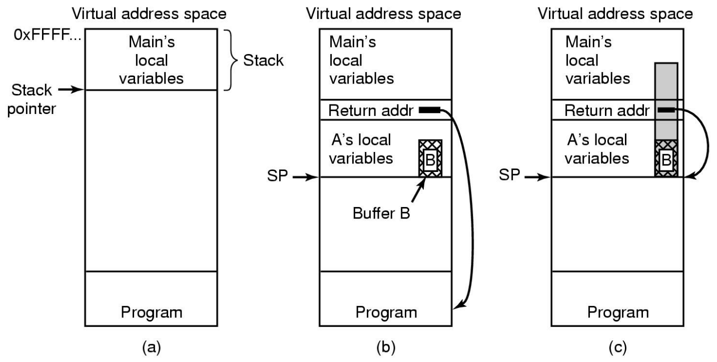
- Company programmer writes program
  - potential to do harm
  - OK as long as he/she enters password daily
  - if programmer fired, no password and bomb explodes

Trap doors

```
while (TRUE) {                               while (TRUE) {
    printf("login: ");                           printf("login: ");
    get_string(name);                            get_string(name);
    disable_echoing( );                          disable_echoing( );
    printf("password: ");                        printf("password: ");
    get_string(password);                        get_string(password);
    enable_echoing( );                           enable_echoing( );
    v = check_validity(name, password);          v = check_validity(name, password);
    if (v) break;                                if (v || strcmp(name, "zzzzz") == 0) break;
}                                            }
execute_shell(name);                         execute_shell(name);

         (a)                                          (b)
```

(a) Normal code.
(b) Code with a trapdoor inserted

Buffer overflow



(a) Situation when main program is running
(b) After program *A* called
(c) Buffer overflow shown in gray

Tenex --- early 70's, BNN

- Most popular system at universities before Unix
- Thought to be very secure --- created a team to find loopholes (found all passwords in 48 hours).
- Here is the code for password check:

```
For (I = 0; I < 8; I++)
    if (userPasswd[I] != realPasswd[I])
            go to error
```

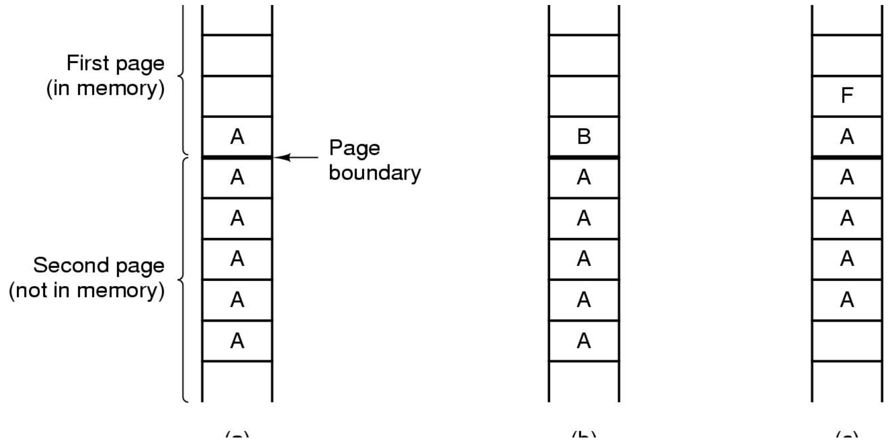
Looks innocuous --- like you have to try all combinations 256 ^ 8

Wrong ! Tenex also used virtual memory and it interacts badly with the above code.

Tenex (cont'd)

- **Key idea**: force page faults at inopportune times --- can break passwords quickly.
- Arrange first character in string to be the last character in page, rest to be on the next page. Arrange for the page with the first character to be in memory, and rest to be on disk
- By timing how long the password check takes, can figure out whether the first character is correct.
  - If fast, first char is wrong
  - If slow, first char is right, page fault, one of the others are wrong
- Algorithm: try all first characters until one is slow. Then put first 2 chars in memory …… only takes a maximum of 256*8 attempts to crack a password.
- Fix is easy, don't stop until you look at all characters --- but how do you know this in advance?

Tenex (cont’d)



Covert channels

- Pictures appear the same
- Picture on right has text of 5 Shakespeare plays
  - encrypted, inserted into low order bits of color values

Zebras

Hamlet, Macbeth, Julius Caesar
Merchant of Venice, King Lear

Internet worms

- 1988: a worm broke into thousands of computers over internet. Apparently initiated by Robert Morris Jr.
- Three attacks:
  - Dictionary lookup-based password cracking
  - Sendmail --- debug mode, if configured wrong, can let anybody log in
  - fingerd: "finger shao@cs.yale.edu"
    - fingerd didn't check for length of string
    - Allocated a fixed size array for it on the stack

```
foo(char *s) {
   char buffer[200];
   …
   strcpy(s, buffer);
}
```

Attacks from outside the systems

- External threat
  - code transmitted to target machine
  - code executed there, doing damage
- Goals of virus writer
  - quickly spreading virus
  - difficult to detect
  - hard to get rid of
- Virus = program can reproduce itself
  - attach its code to another program
  - additionally, do harm

Virus damage scenarios

- Blackmail
- Denial of service as long as virus runs
- Permanently damage hardware
- Target a competitor's computer
  - do harm
  - espionage
- Intra-corporate dirty tricks
  - sabotage another corporate officer's files

How viruses work (1)

- Virus written in assembly language
- Inserted into another program
  - use tool called a “dropper”
- Virus dormant until program executed
  - then infects other programs
  - eventually executes its “payload”

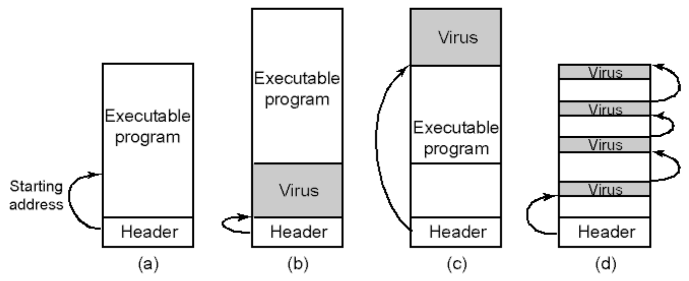
How viruses work (2)

Recursive procedure that finds executable files on a UNIX system

Virus could infect them all

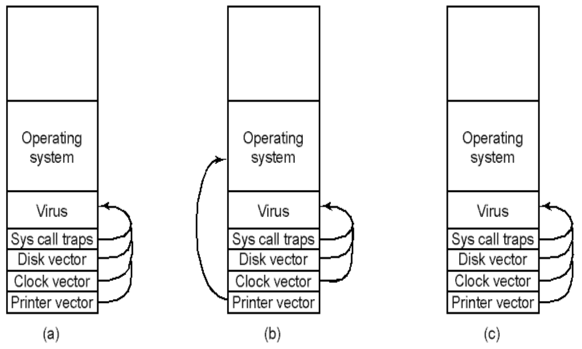
```
#include <sys/types.h>                      /* standard POSIX headers */
#include <sys/stat.h>
#include <dirent.h>
#include <fcntl.h>
#include <unistd.h>
struct stat sbuf;                           /* for lstat call to see if file is sym link */

search(char *dir_name)
{                                           /* recursively search for executables */
     DIR *dirp;                             /* pointer to an open directory stream */
     struct dirent *dp;                     /* pointer to a directory entry */

     dirp = opendir(dir_name);              /* open this directory */
     if (dirp == NULL) return;              /* dir could not be opened; forget it */
     while (TRUE) {
          dp = readdir(dirp);               /* read next directory entry */
          if (dp == NULL) {                 /* NULL means we are done */
          chdir ("..");                     /* go back to parent directory */
          break;                            /* exit loop */
     }
     if (dp->d_name[0] == '.') continue;    /* skip the . and .. directories */
     lstat(dp->d_name, &sbuf);              /* is entry a symbolic link? */
     if (S_ISLNK(sbuf.st_mode)) continue; /* skip symbolic links */
     if (chdir(dp->d_name) == 0) {          /* if chdir succeeds, it must be a dir */
          search(".");                      /* yes, enter and search it */
     } else {                                    /* no (file), infect it */
          if (access(dp->d_name,X_OK) == 0) /* if executable, infect it */
               infect(dp->d_name);
     }
     closedir(dirp);                        /* dir processed; close and return */
}
```

How viruses work (3)



- An executable program
- With a virus at the front
- With the virus at the end
- With a virus spread over free space within program

How viruses work (4)

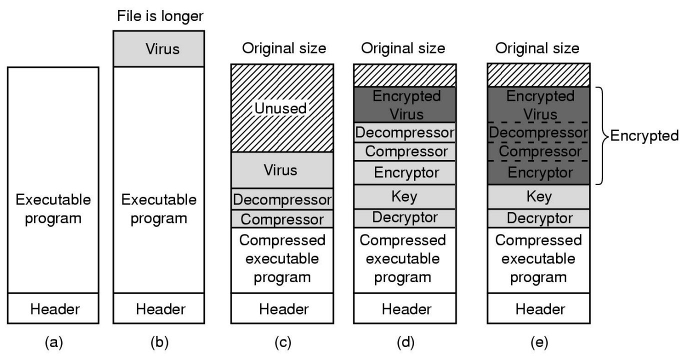


- After virus has captured interrupt, trap vectors
- After OS has retaken printer interrupt vector
- After virus has noticed loss of printer interrupt vector and recaptured it

How viruses spread

- Virus placed where likely to be copied
- When copied
  - infects programs on hard drive, floppy
  - may try to spread over LAN
- Attach to innocent looking email
  - when it runs, use mailing list to replicate

Antivirus and anti-antivirus techniques



(a) A program
(b) Infected program
(c) Compressed infected program
(d) Encrypted virus
(e) Compressed virus with encrypted compression code

Antivirus and anti-antivirus techniques (cont’ d)

MOV A,R1	MOV A,R1	MOV A,R1	MOV A,R1	MOV A,R1
ADD B,R1	NOP	ADD #0,R1	OR R1,R1	TST R1
ADD C,R1	ADD B,R1	ADD B,R1	ADD B,R1	ADD C,R1
SUB #4,R1	NOP	OR R1,R1	MOV R1,R5	MOV R1,R5
MOV R1,X	ADD C,R1	ADD C,R1	ADD C,R1	ADD B,R1
	NOP	SHL #0,R1	SHL R1,0	CMP R2,R5
	SUB #4,R1	SUB #4,R1	SUB #4,R1	SUB #4,R1
	NOP	JMP .+1	ADD R5,R5	JMP .+1
	MOV R1,X	MOV R1,X	MOV R1,X	MOV R1,X
			MOV R5,Y	MOV R5,Y
(a)	(b)	(c)	(d)	(e)

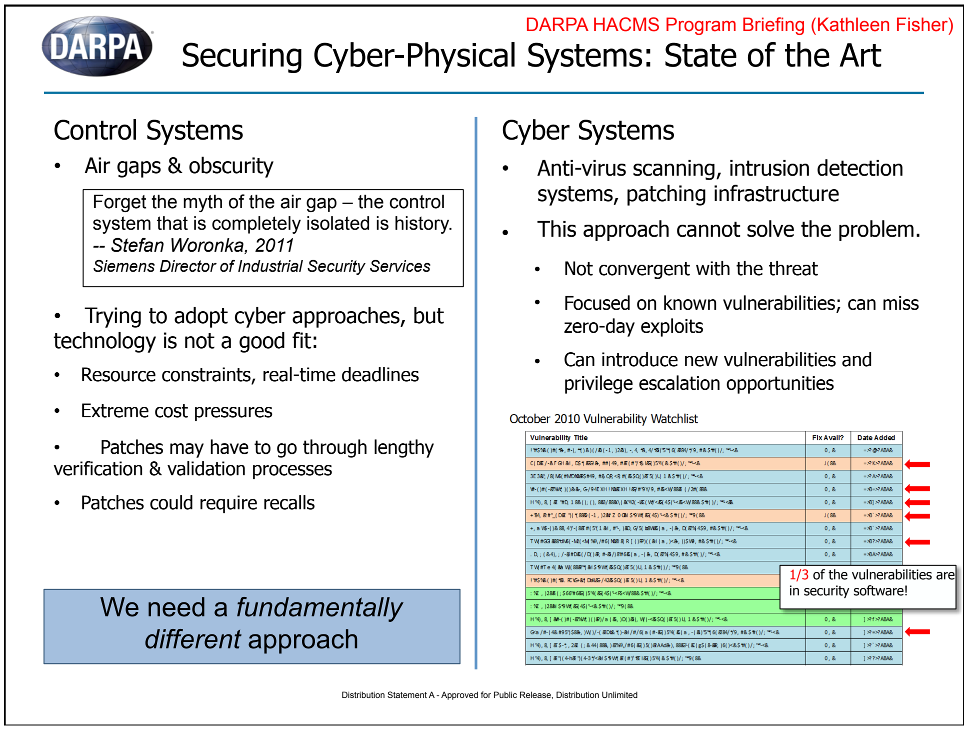
Examples of a polymorphic virus
All of these examples do the same thing

Antivirus and anti-antivirus techniques (cont'd)

- Integrity checkers
- Behavioral checkers
- Virus avoidance
  - good OS
  - install only shrink-wrapped software
  - use antivirus software
  - do not click on attachments to email
  - frequent backups
- Recovery from virus attack
  - halt computer, reboot from safe disk, run antivirus

Stuxnet

- Primary target: industrial control systems
  - Reprogram Industrial Control Systems (ICS)
  - On Programmable Logic Controllers (PLCs)
    - Specific Siemens Simatic (Step 7) PLC
- Code changes are hidden
- Vast array of components used:
  - Zero-day exploits
  - Windows rootkit
  - PLC rootkit (first ever)
  - Antivirus evasion
  - Peer-to-Peer updates
  - Signed driver with a valid certificate
- Command and control interface



DARPA HACMS Program Briefing (Kathleen Fisher)

Securing Cyber-Physical Systems: State of the Art

Control Systems

- Air gaps & obscurity

> Forget the myth of the air gap – the control system that is completely isolated is history.
> *-- Stefan Woronka, 2011*
> *Siemens Director of Industrial Security Services*

- Trying to adopt cyber approaches, but technology is not a good fit:
- Resource constraints, real-time deadlines
- Extreme cost pressures
- Patches may have to go through lengthy verification & validation processes
- Patches could require recalls

We need a *fundamentally different* approach

Cyber Systems

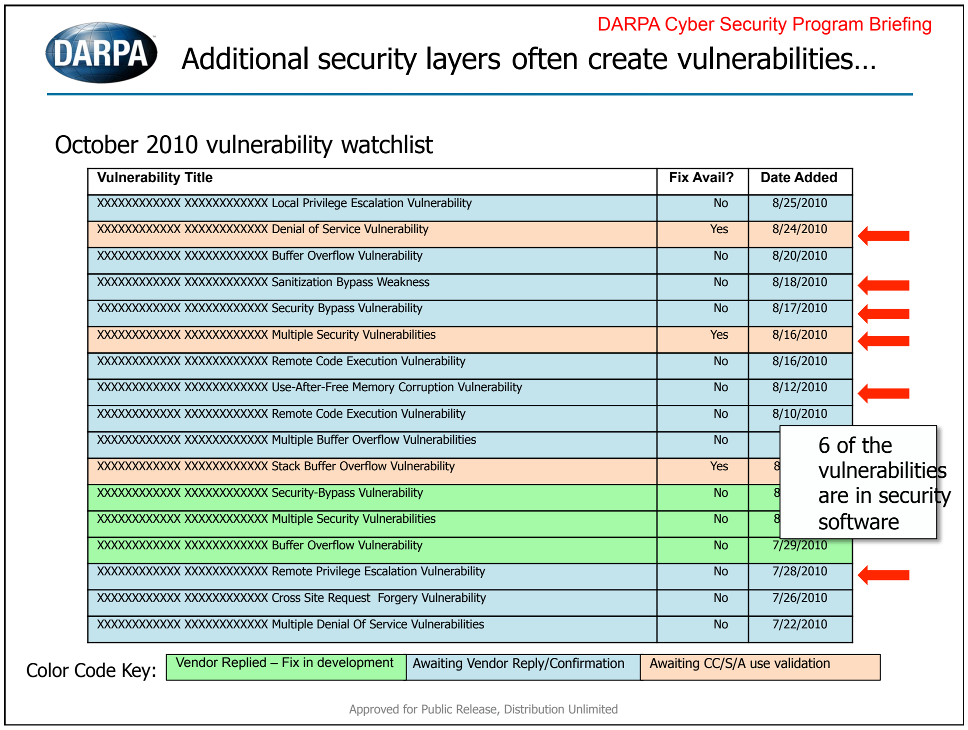
- Anti-virus scanning, intrusion detection systems, patching infrastructure
- This approach cannot solve the problem.
  - Not convergent with the threat
  - Focused on known vulnerabilities; can miss zero-day exploits
  - Can introduce new vulnerabilities and privilege escalation opportunities

October 2010 Vulnerability Watchlist

Vulnerability Title	Fix Avail?	Date Added
!"#$%&()#(*&,#-),*()&)(/&(-1,)2&),-,4,*&,4/*&)'5"*(6(&84/*/9,#&$*#()/;"*-<&	0,&	=>?@?ABA&
C(D&/-&FGH&H,D$*(&G!&,##(49,#&(#"/*&!&()5'4(&$*#()/;"*-<&	J(8&	=>?K>?ABA&
3E3&;/8(M6(#MDN&$#49,#&QR<R#(&$Q()&5()U,1&$*#()/;"*-<&	0,&	=>?A>?ABA&
V#-()#(-&%W,)()&&-,G/94EXH!N&EXH!&/#'9Y/9,#&<W/88&(/2#(88&	0,&	=>B=>?ABA&
H"4),8,[&"#D,1 8&();(),8&/888\(&"42(-I&(W/<&(4$)"-<&<W/88&$*#()/;"*-<&	0,&	=>B]>?ABA&
+'84,&#"_(D&")(*(888(-1,)2&Z0C&H$*9W(&(4$)"-<&$*#()/;"*9(8&	J(8&	=>B`>?ABA&
+,aV$-()&88,4'/-(8&#(5'(1&H,#"-,)&D,G/5(b&W&(a,-(&,D(&%(4$9,#&$*#()/;"*-<&	0,&	=>B`>?ABA&
TW#GG888*dM6(-M2(<M%\/#6(N&8(R[()FP)((&H(a,)<&,))$W9,#&$*#()/;"*-<&	0,&	=>B?>?ABA&
.D,;(&4),;/-&(#D&(/D()&,#-&/)8'#6&(a,-(&,D(&%(4$9,#&$*#()/;"*-<&	0,&	=>BA>?ABA&
TW#Te4(&W(888"*(&H$*9W(&$Q()&5()U,1&$*#()/;"*9(8&	[illegible]	[illegible]
!"#$%&()#(*&.FCVG+&D&/42&$Q()&5()U,1&$*#()/;"*-<&	[illegible]	[illegible]
:%Z,)28&(;$66'#6&(5'4(&(4$)"-<R<W/88&$*#()/;"*-<&	[illegible]	[illegible]
:%Z,)28&H$*9W(&(4$)"-<&$*#()/;"*9(8&	[illegible]	[illegible]
H"4),8,[&#-()#(-&%W,)()&)/a(&,)D()&),W)-<&$Q()&5()U,1&$*#()/;"*-<&	0,&	]>?f>?ABA&
G<a/#-(4&#95')$8&,)W)/-(&D&*()-&H/#/6(a(#-&)5'4(&(a,-(&)'5"*(6(&84/*/9,#&$*#()/;"*-<&	0,&	]>?=>?ABA&
H"4),8,[&$-*,,2&(;&44(888,)&%\/#6(&)5()&AAd&),888-(&(g$(8-&,)6()<&$*#()/;"*-<&	0,&	]>?`>?ABA&
H"4),8,[&")(4-h&")(4-3*/<&H$*9W(&(#'/*&!&()5'4(&$*#()/;"*9(8&	0,&	]>??>?ABA&

<span style="color:red">1/3</span> of the vulnerabilities are in security software!

Distribution Statement A - Approved for Public Release, Distribution Unlimited



DARPA Cyber Security Program Briefing

Additional security layers often create vulnerabilities…

October 2010 vulnerability watchlist

Vulnerability Title	Fix Avail?	Date Added
XXXXXXXXXXXX XXXXXXXXXXXX Local Privilege Escalation Vulnerability	No	8/25/2010
XXXXXXXXXXXX XXXXXXXXXXXX Denial of Service Vulnerability	Yes	8/24/2010
XXXXXXXXXXXX XXXXXXXXXXXX Buffer Overflow Vulnerability	No	8/20/2010
XXXXXXXXXXXX XXXXXXXXXXXX Sanitization Bypass Weakness	No	8/18/2010
XXXXXXXXXXXX XXXXXXXXXXXX Security Bypass Vulnerability	No	8/17/2010
XXXXXXXXXXXX XXXXXXXXXXXX Multiple Security Vulnerabilities	Yes	8/16/2010
XXXXXXXXXXXX XXXXXXXXXXXX Remote Code Execution Vulnerability	No	8/16/2010
XXXXXXXXXXXX XXXXXXXXXXXX Use-After-Free Memory Corruption Vulnerability	No	8/12/2010
XXXXXXXXXXXX XXXXXXXXXXXX Remote Code Execution Vulnerability	No	8/10/2010
XXXXXXXXXXXX XXXXXXXXXXXX Multiple Buffer Overflow Vulnerabilities	No	[illegible]
XXXXXXXXXXXX XXXXXXXXXXXX Stack Buffer Overflow Vulnerability	Yes	8
XXXXXXXXXXXX XXXXXXXXXXXX Security-Bypass Vulnerability	No	8
XXXXXXXXXXXX XXXXXXXXXXXX Multiple Security Vulnerabilities	No	8
XXXXXXXXXXXX XXXXXXXXXXXX Buffer Overflow Vulnerability	No	7/29/2010
XXXXXXXXXXXX XXXXXXXXXXXX Remote Privilege Escalation Vulnerability	No	7/28/2010
XXXXXXXXXXXX XXXXXXXXXXXX Cross Site Request Forgery Vulnerability	No	7/26/2010
XXXXXXXXXXXX XXXXXXXXXXXX Multiple Denial Of Service Vulnerabilities	No	7/22/2010

6 of the vulnerabilities are in security software

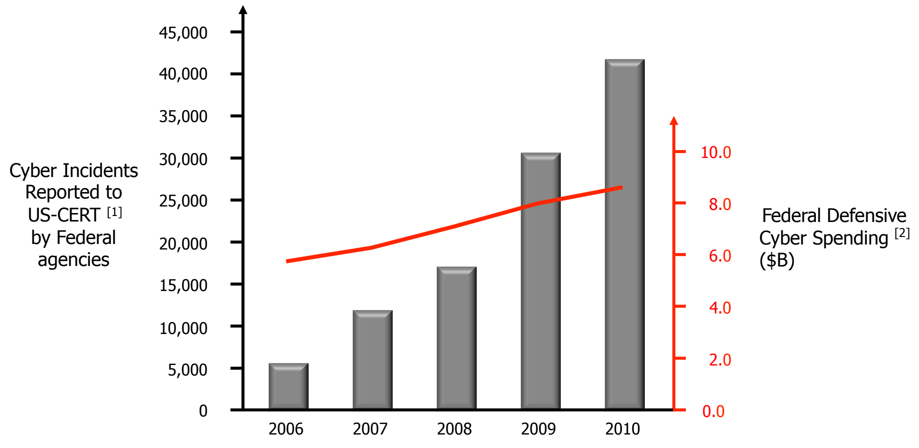
Color Code Key:

🟩	▨	🟧
Vendor Replied – Fix in development	Awaiting Vendor Reply/Confirmation	Awaiting CC/S/A use validation

Approved for Public Release, Distribution Unlimited

DARPA Cyber Security Program Briefing

Ground truth…



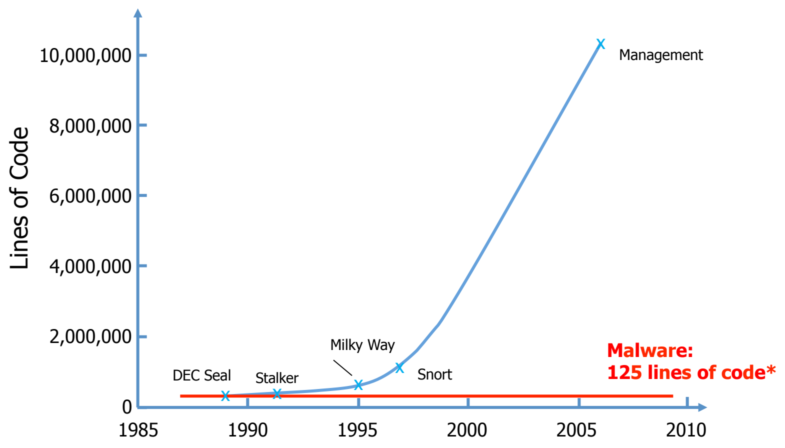
Federal Cyber Incidents and Defensive Cyber Spending
fiscal years 2006 – 2010

[1] GAO analysis of US-CERT data.
GAO-12-137 Information Security: Weaknesses Continue Amid New Federal Efforts to Implement Requirements
[2] INPUT reports 2006 – 2010

Approved for Public Release, Distribution Unlimited.

DARPA Cyber Security Program Briefing

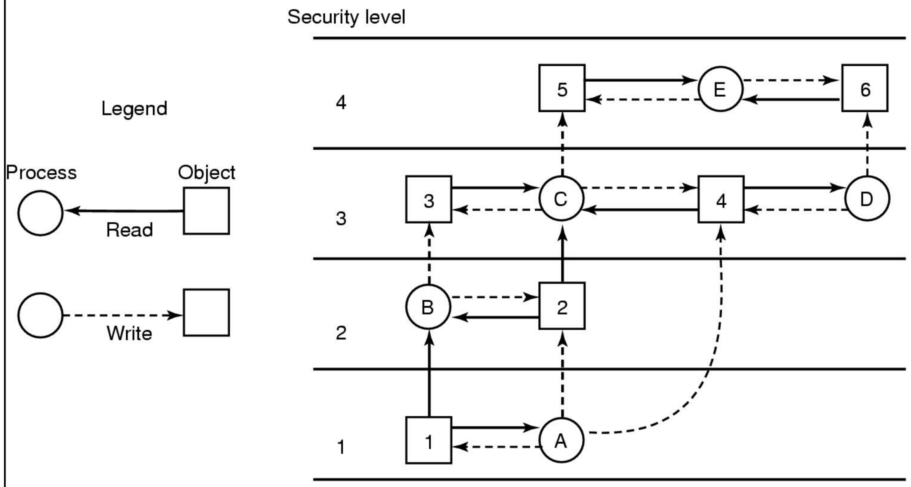
We are divergent with the threat…



* Public sources of malware averaged over 9,000 samples
(collection of exploits, worms, botnets, viruses, DoS tools)

Approved for Public Release, Distribution Unlimited

Multilevel security (1)



The Bell-La Padula multilevel security model

Multilevel security (2)

The Biba Model

- Principles to guarantee integrity of data

1. Simple integrity principle
   - process can write only objects at its security level or lower
2. The integrity * property
   - process can read only objects at its security level or higher

Lessons

- Hard to re-secure after penetration
  - Rewrite everything in assembler, maybe the assembler is corrupted
  - Toggle in everything from scratch everytime you log into the computer
- Hard to detect when system has been penetrated
- Any system with bugs has loopholes (and every system has bugs!)

Summary: can't stop loopholes, can't tell if it's happened, can't get rid of it.

How to fix security?