Systems Security Group | Lancaster University

Operating Systems Security

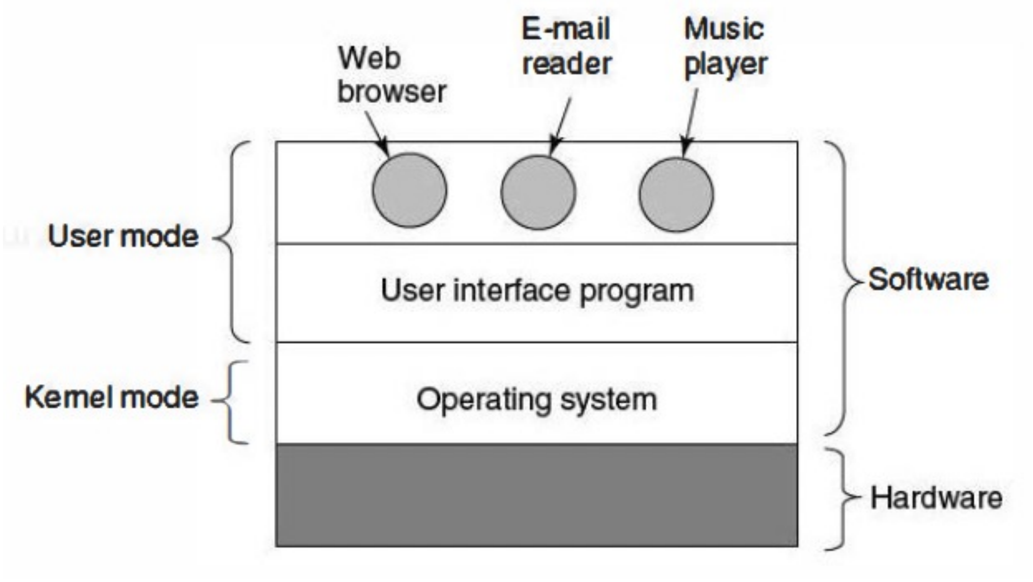
Learning Objectives

- Understand about the security threats in operating systems

- Learn about ways to protect

What is an operating system?

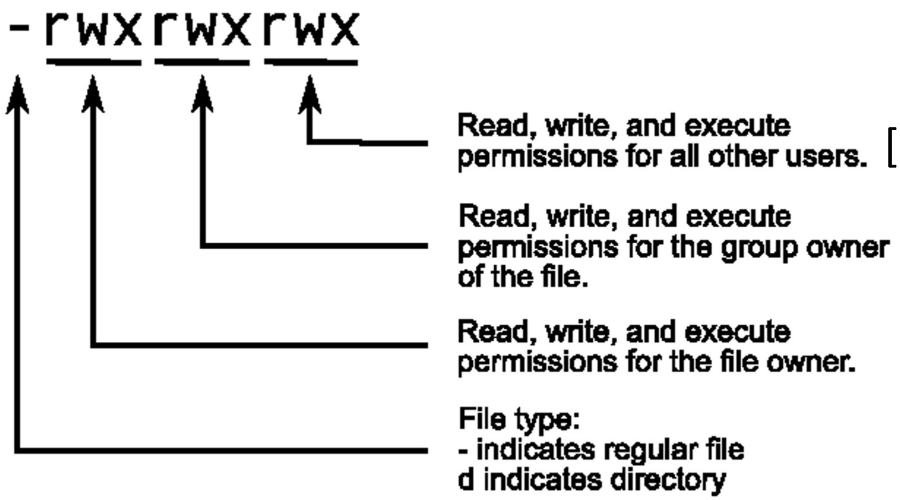
- A layer of software
  - Provide user programmes with a simpler model of a computer
  - Handle the managing of resources



[source: 1]

Why do we need protection?

- Confidentiality
  - Large amount of data is contained in systems
- Security issues
  - Human and nonhuman



-rwxrwxrwx

Read, write, and execute permissions for all other users.

Read, write, and execute permissions for the group owner of the file.

Read, write, and execute permissions for the file owner.

File type:
- indicates regular file
d indicates directory

[source: linuxcommand.org]

Threats

- Exposure of data
  - Threats data confidentiality
- Tampering of data
  - Threats data integrity
- Denial of service
  - Threats system’s availability
- System infected by viruses
  - Threats the goal of excluding outsiders

Cryptography

- Cryptography may be used to ensure confidentiality and integrity
  - Symmetric cryptography
  - Asymmetric cryptography
  - Hash functions

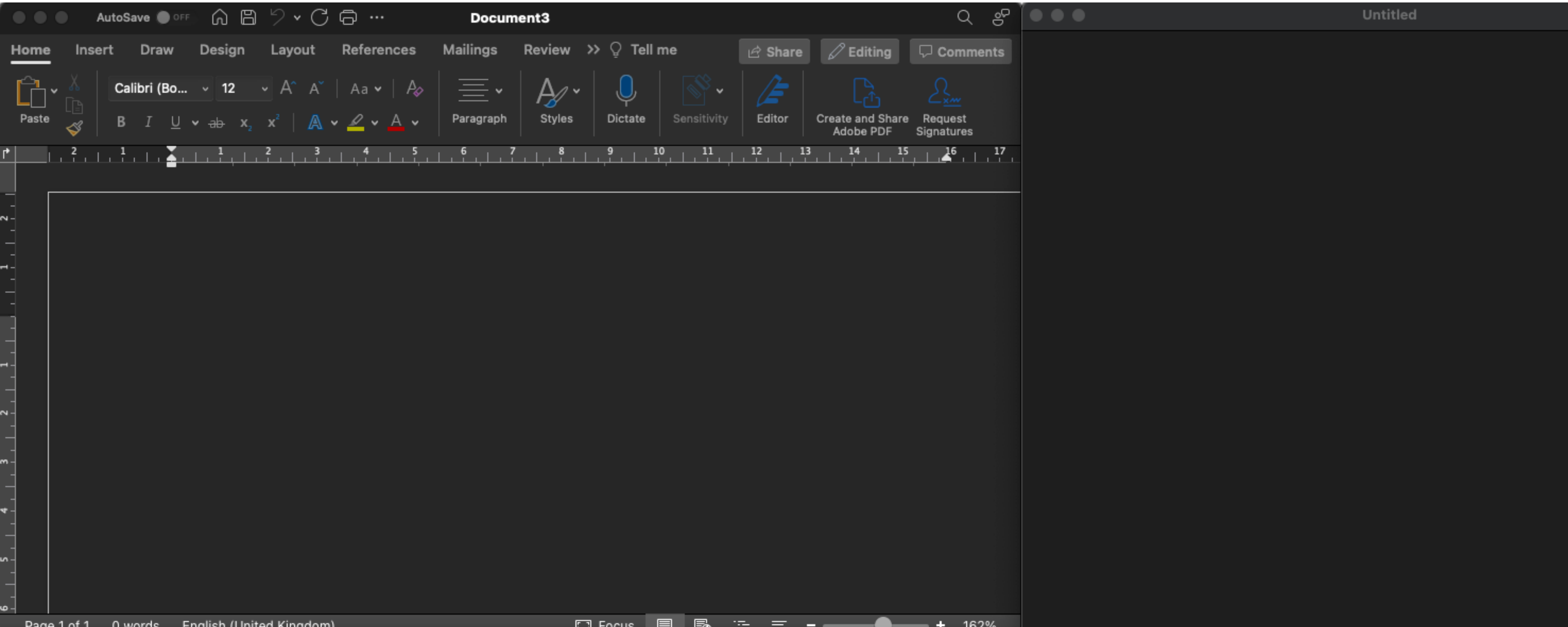
- What if the keys are compromised?

Trusted platform module (TPM)

- Cryptoprocessor with some non-volatile storage for the keys
- Can perform cryptographic operations in main memory
- Can verify digital signatures
- Since implemented in hardware, it’s fast too

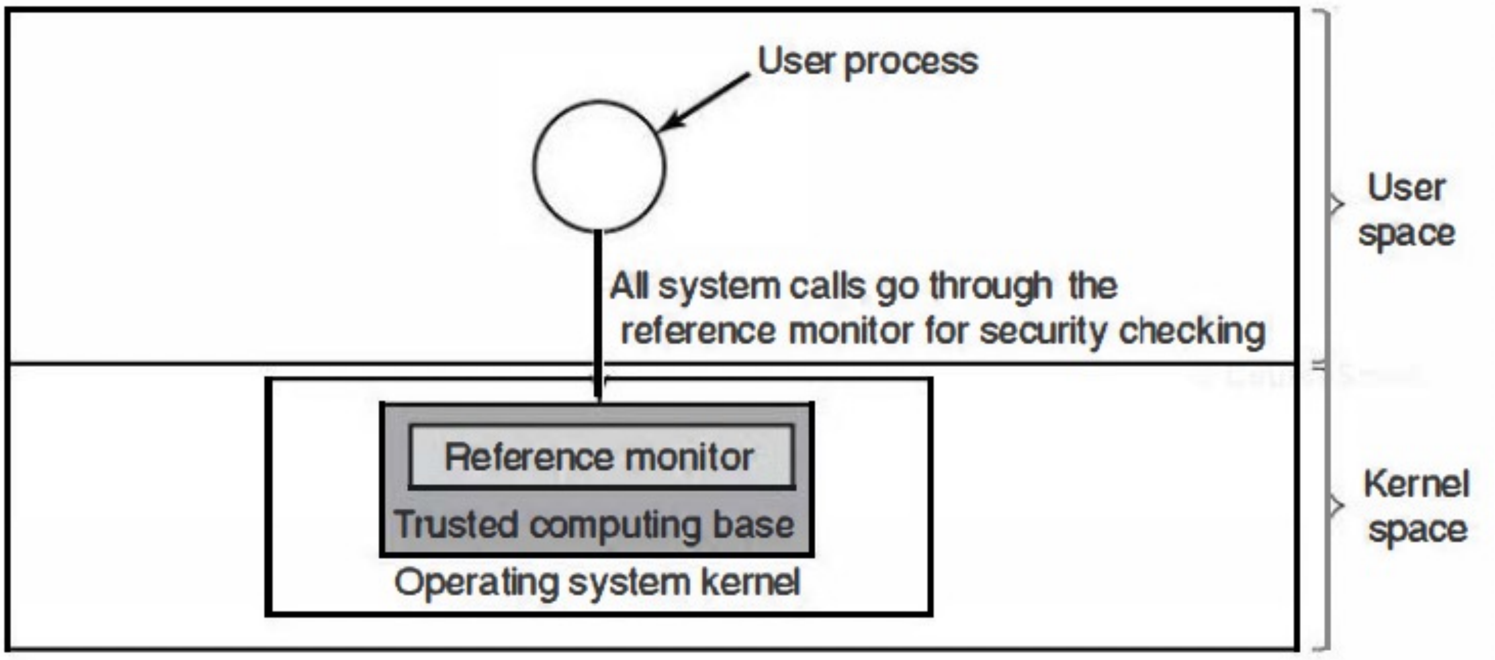
Trusted systems

- Can you build a secure computer system?
- Simplicity vs features



Trusted computing base

- Combination of hardware and software for enforcing security rules



[source: 1]

What else do we need?

Models to restrict access

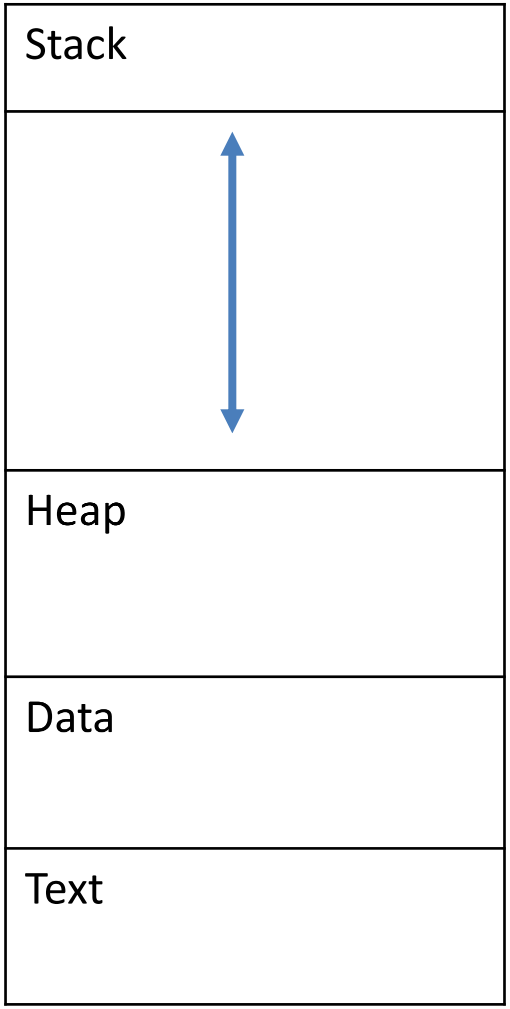
- Access matrix
- Multilevel security
  - Bell-La Padula model
  - Biba model

Are we secure now?

- Confinement problem [3]
  - Deals with preventing a process from transmitting information to any other program except its caller.
- Covert channels
  - Path of communication that was not designed to be used for communication.

Processes

- A program in execution
- **Stack**: temporal data e.g. local variables, return address
- **Heap**: dynamic allocated memory of the process at run time
- **Text**: current activity by the program counter and registers
- **Data**: contains global and static variables



Threads

- A process generates a thread when it requires to send something to the CPU for processing
- The thread includes an individual instruction set and data
- Generated dynamically
- Multithreaded applications are capable of running several different threads at the same time.
- Threads share the same resources of the process that created them

Memory leaks

- Every process is allocated with an amount of memory
- Memory should be released when the process finished with it
- Not the case for poorly written applications
- May be used for Denial of Service attacks and lead to memory starvation

Technical attacks

- Buffer overflow attacks
- String formatting attacks
- Integer overflow attacks
- Code injection attacks

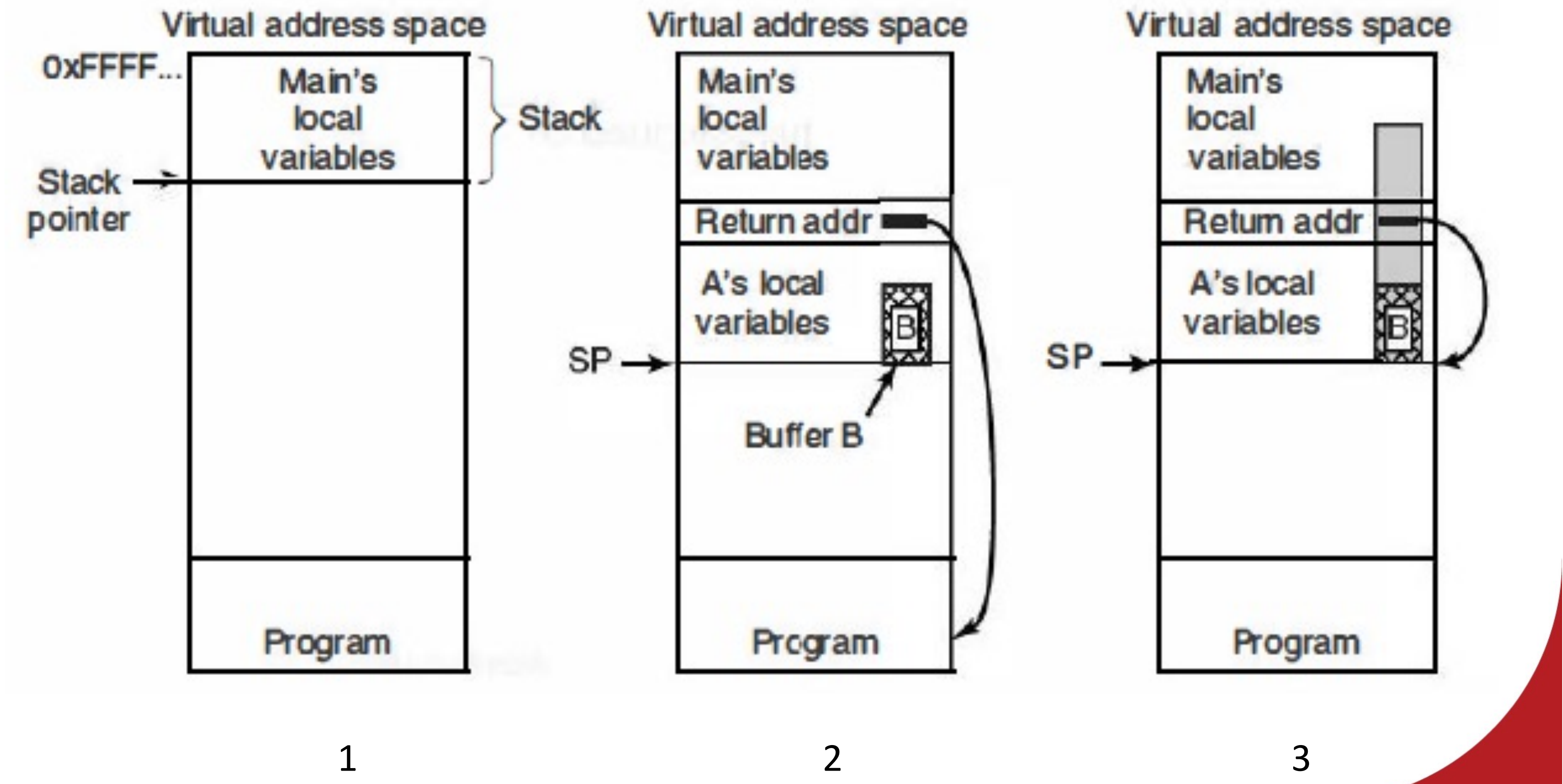
Buffer overflow

- C compiler does not do array bounds checking
- Suppose the following code

```
int i;
char ch[16];
i=32;
ch[i]=0;
```

Buffer overflow

[source: 1]



String formatting attacks

- Issues when valid formatting is not used with `printf`
- Statements such as `printf(buffer)` are still valid
- An attacker can pass a list of parameters that can be executed as a command
  - Execute code, read the stack, etc.

String formatting attacks

```
#include <stdio.h>
int main() {

char* s = "%x %x Hello World";
    printf("%s\n", s);  //%x %x Hello World
    printf(s);//aedda000 aebb49e0 Hello World

    return 0;
}
```

Integer overflow

- Integer arithmetic operations are commonly done using modulo arithmetic
- Modulo arithmetic allows values to wrap if they go above a certain value
- 8-bit integer will hold values between 0 and 255
- What if a higher number has to be stored there?

Code injection attacks

- Gets a program to execute code without realising that
- Mainly due to poor implementations

```
int main(void) {
        char command[1024], src[500], dst[500];
        strcpy(command, “cp “);
        printf(”Source: “); gets(src);
        printf(”Destination: “); gets(dst);
        strcpy(command, src); strcpy(command, “ “);
        strcpy(command, dst);
        system(command);
    return 0;}
```

Code injection attacks

- If a user adds
  - src = source.txt and dst = destination.txt
- It will execute
  - cp source.txt destination.txt
- How about
  - src = source.txt and
  - dst = destination.txt; rm –rf /
- It will execute
  - cp source.txt destination.txt; rm –rf /

References

[1] Andrew S. Tanenbaum, “Modern Operating systems”. Chapter 9

[2] Shon Harris, All in one CISSP, Chapter 5 on Security Architecture and Design.

[3] Lampson, B. W. (1973). A note on the confinement problem. *Communications of the ACM*, *16*(10), 613-615.