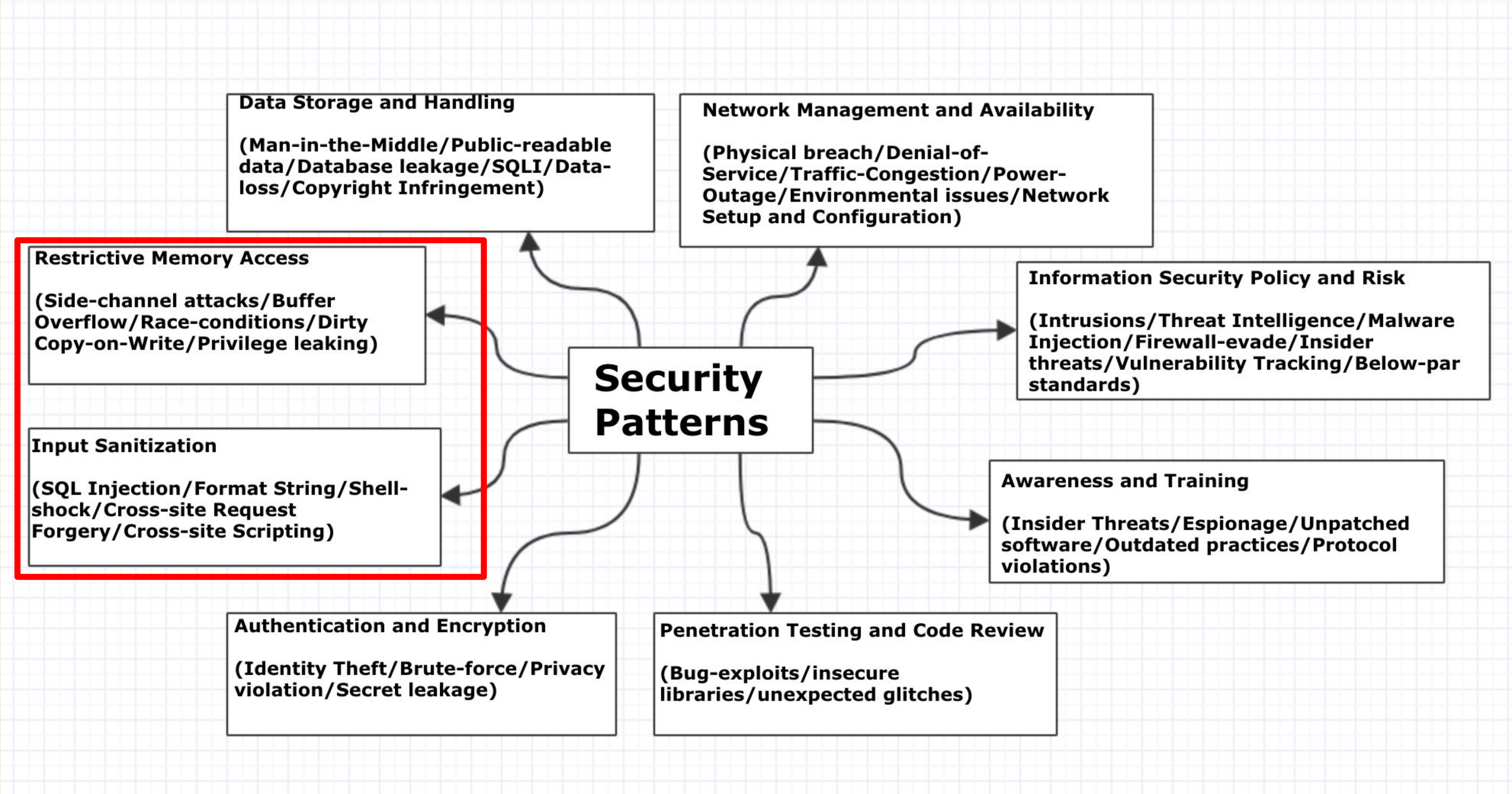
SECURITY PATTERNS

Vibhu A. Bharadwaj, Koushik Godbole, Lingyun Ke, Sai Vardhan Lella
CSE-776 (Design Patterns), Fall 2018
Syracuse University

Intent and Motivation

- Information and identity are valued much more than before.
- Lack of foresight in security implementation leaves the gates wide open for exploitation.
- Improvement in tech has made ‘Hacking’ easier than ever before with novel means being discovered each day.
- A security pattern is a solution that addresses a class of security problems/flaws.
- Security Patterns offer comprehensive solutions by treating Security as a Functional requirement in software design.
- Security patterns help achieve CIA (Confidentiality, Integrity and Availability) of information.

Classification of Security Patterns



Security Patterns

Data Storage and Handling

(Man-in-the-Middle/Public-readable data/Database leakage/SQLI/Data-loss/Copyright Infringement)

Network Management and Availability

(Physical breach/Denial-of-Service/Traffic-Congestion/Power-Outage/Environmental issues/Network Setup and Configuration)

Information Security Policy and Risk

(Intrusions/Threat Intelligence/Malware Injection/Firewall-evade/Insider threats/Vulnerability Tracking/Below-par standards)

Awareness and Training

(Insider Threats/Espionage/Unpatched software/Outdated practices/Protocol violations)

Penetration Testing and Code Review

(Bug-exploits/insecure libraries/unexpected glitches)

Authentication and Encryption

(Identity Theft/Brute-force/Privacy violation/Secret leakage)

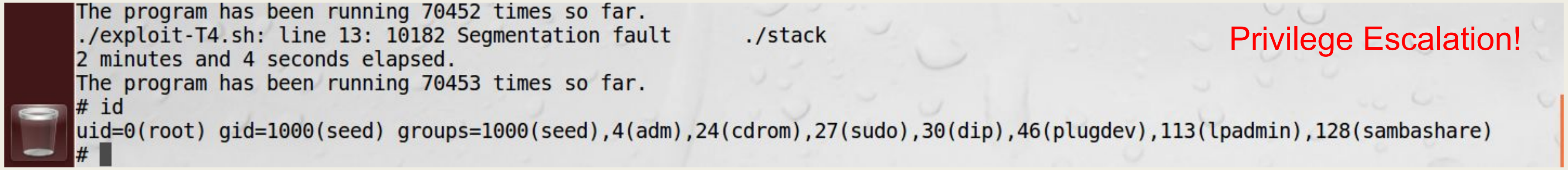
Input Sanitization

(SQL Injection/Format String/Shell-shock/Cross-site Request Forgery/Cross-site Scripting)

Restrictive Memory Access

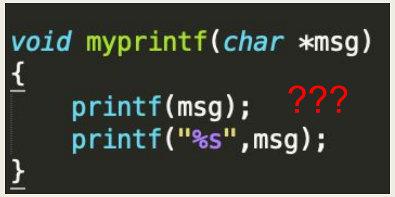
(Side-channel attacks/Buffer Overflow/Race-conditions/Dirty Copy-on-Write/Privilege leaking)

Some Examples of Security Flaws and exploitation



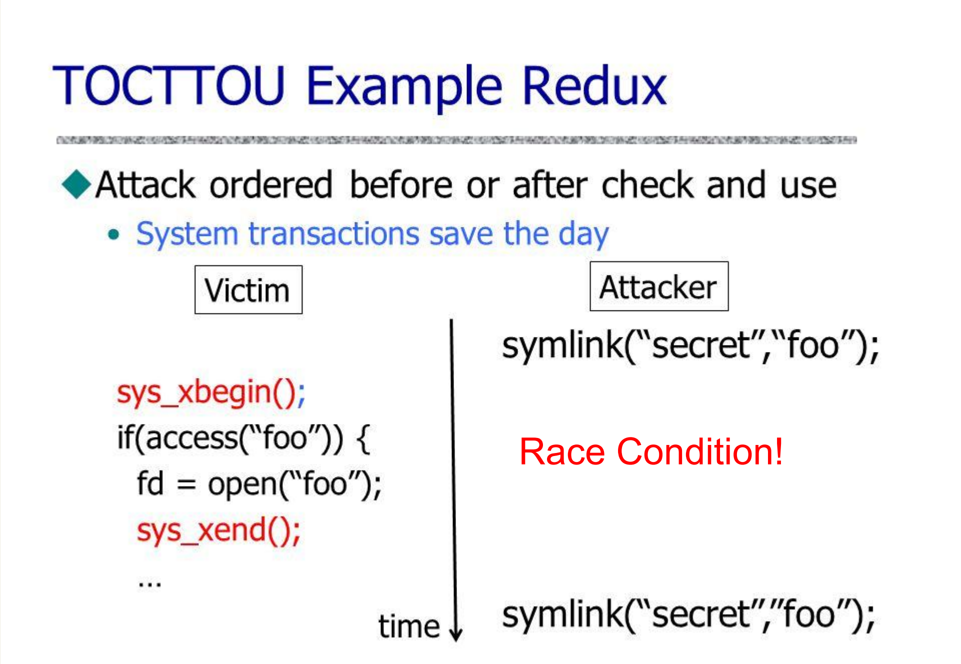
```
The program has been running 70452 times so far.
./exploit-T4.sh: line 13: 10182 Segmentation fault      ./stack
2 minutes and 4 seconds elapsed.
The program has been running 70453 times so far.
# id
uid=0(root) gid=1000(seed) groups=1000(seed),4(adm),24(cdrom),27(sudo),30(dip),46(plugdev),113(lpadmin),128(sambashare)
# 
```

Privilege Escalation!



```
void myprintf(char *msg)
{
    printf(msg);
    printf("%s",msg);
}
```

???



TOCTTOU Example Redux

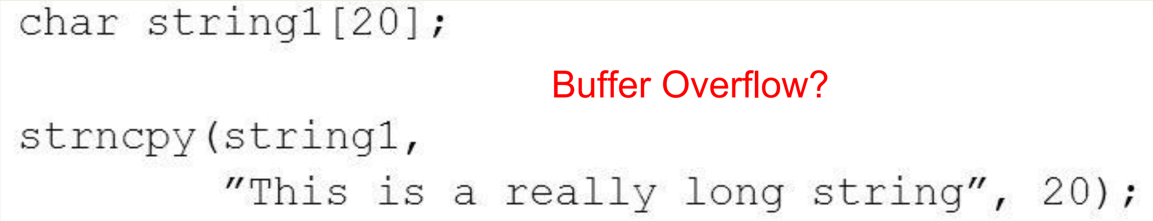
- Attack ordered before or after check and use
  - System transactions save the day

Victim	Attacker
	symlink("secret","foo");

sys_xbegin();
if(access("foo")) {
  fd = open("foo");
  sys_xend();
  …	

time	symlink("secret","foo");

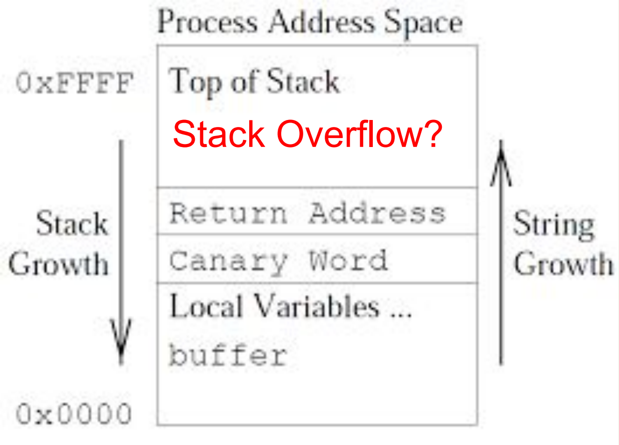
Race Condition!



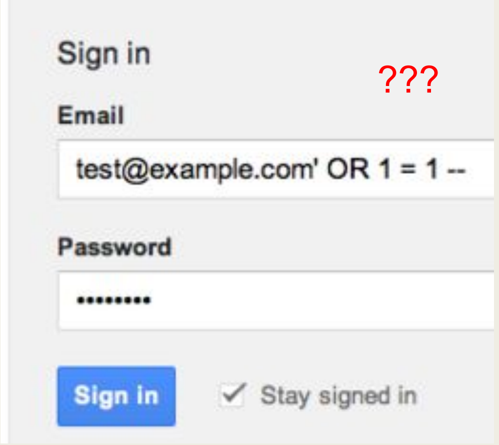
```
char string1[20];

strncpy(string1,
        "This is a really long string", 20);
```

Buffer Overflow?



Stack Overflow?



Sign in

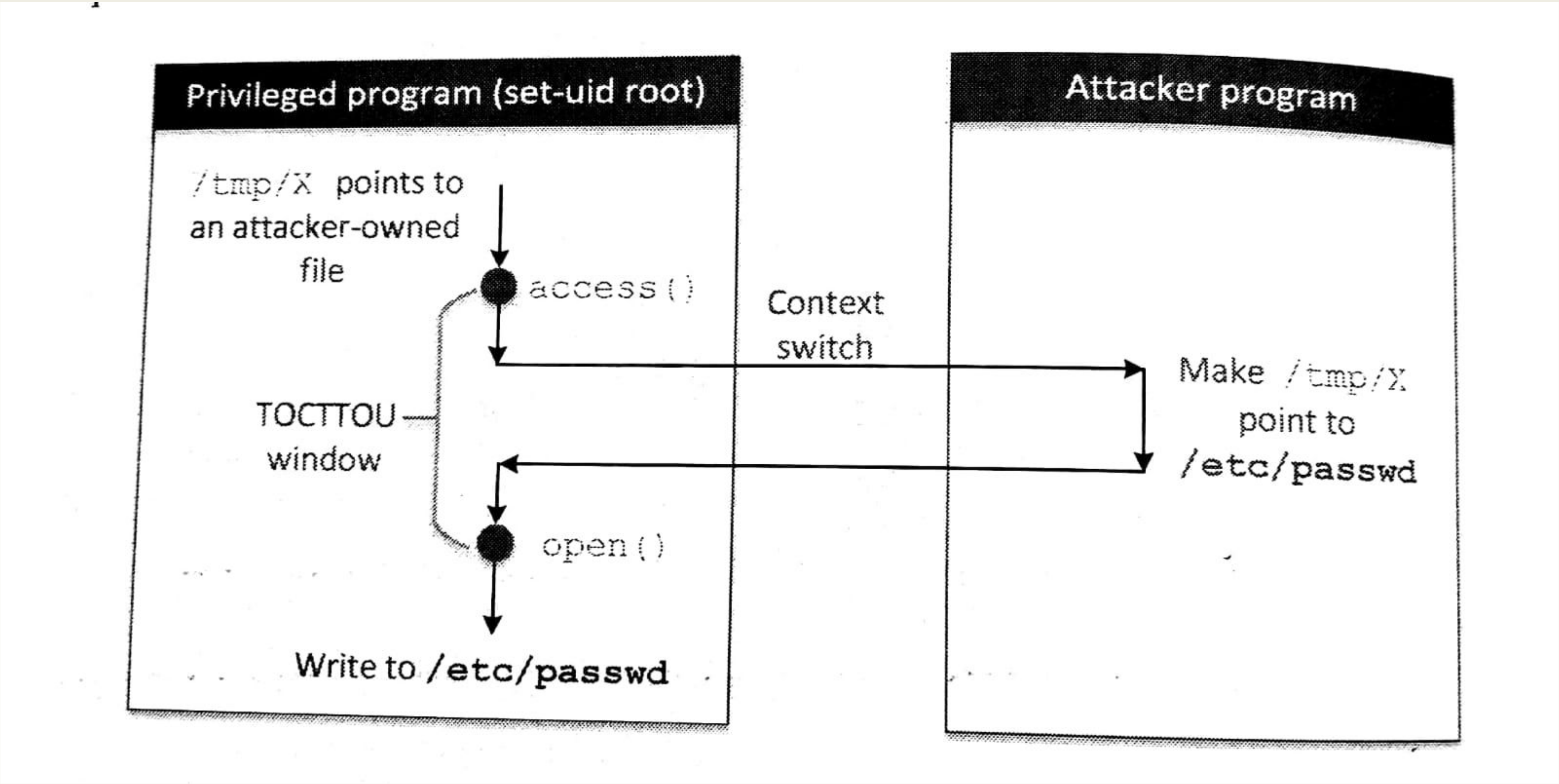
Email: test@example.com' OR 1 = 1 --

Password: ••••••••

???

Demo of Buffer Overflow:

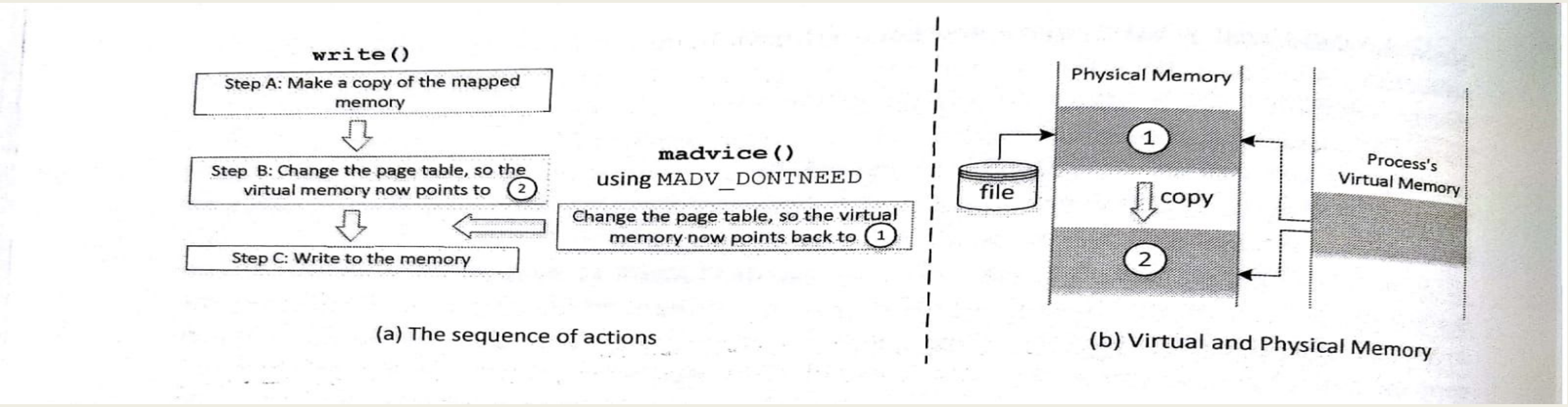
Race Condition example


Privileged program (set-uid root)
/tmp/X points to an attacker-owned file
access()
TOCTTOU window
open()
Write to /etc/passwd
Context switch
Attacker program
Make /tmp/X point to /etc/passwd

Restrictive Memory Access Pattern

The possible exploitation: -

- Heap Buffer Overflow
- Shellshock-BashCGI
- Side Channel Attacks
- Dirty Copy-on-Write



write()
Step A: Make a copy of the mapped memory
Step B: Change the page table, so the virtual memory now points to 2
Step C: Write to the memory
madvice()
using MADV_DONTNEED
Change the page table, so the virtual memory now points back to 1
Physical Memory
1
copy
2
file
Process's Virtual Memory

(a) The sequence of actions

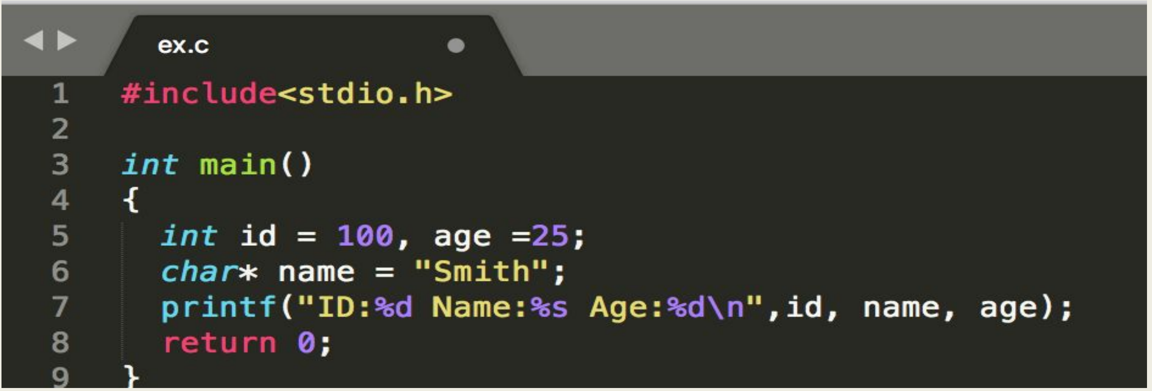
(b) Virtual and Physical Memory

(Taken as per Linux context)

Implementation of Restrictive Memory Pattern

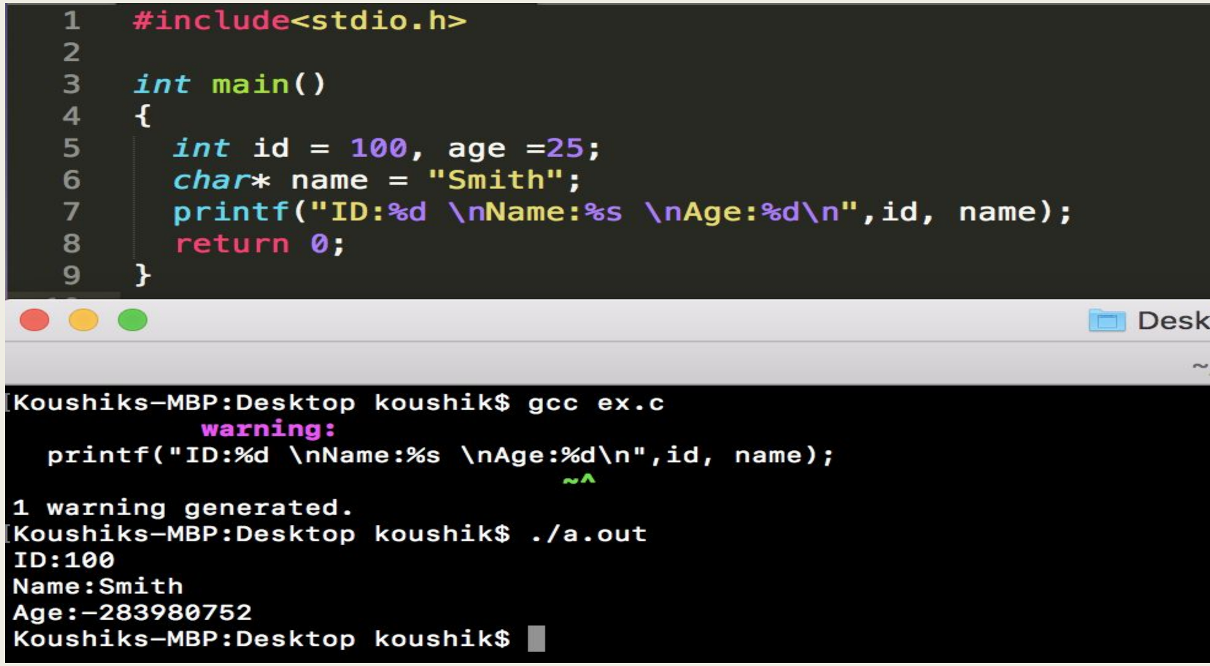
1. Eliminate Racing or make winning odds unfavorable for attacker
2. Use secure libraries/frameworks
3. Least Privilege Principle and service privilege levels
4. Sandboxing/clear memory boundaries
5. Update systems and applications

Formatted Strings



```c
#include<stdio.h>

int main()
{
  int id = 100, age =25;
  char* name = "Smith";
  printf("ID:%d Name:%s Age:%d\n",id, name, age);
  return 0;
}
```

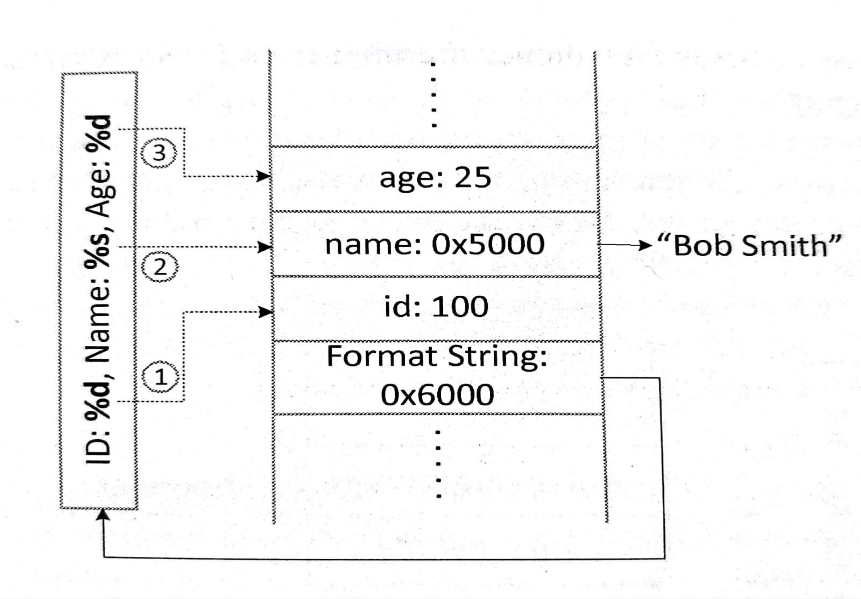


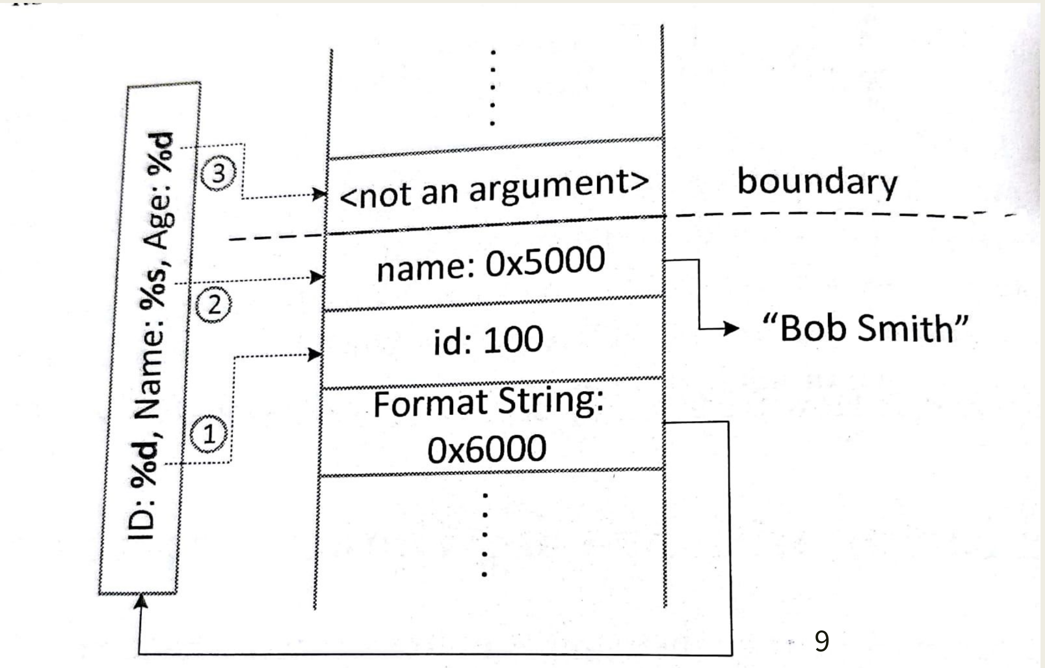
```c
#include<stdio.h>

int main()
{
  int id = 100, age =25;
  char* name = "Smith";
  printf("ID:%d \nName:%s \nAge:%d\n",id, name);
  return 0;
}
```

```
Koushiks-MBP:Desktop koushik$ gcc ex.c
           warning:
  printf("ID:%d \nName:%s \nAge:%d\n",id, name);
                               ~^

1 warning generated.
Koushiks-MBP:Desktop koushik$ ./a.out
ID:100
Name:Smith
Age:-283980752
Koushiks-MBP:Desktop koushik$
```





Counter Measures

1. Developers must have a good practice to not use user inputs as an part of a format string.

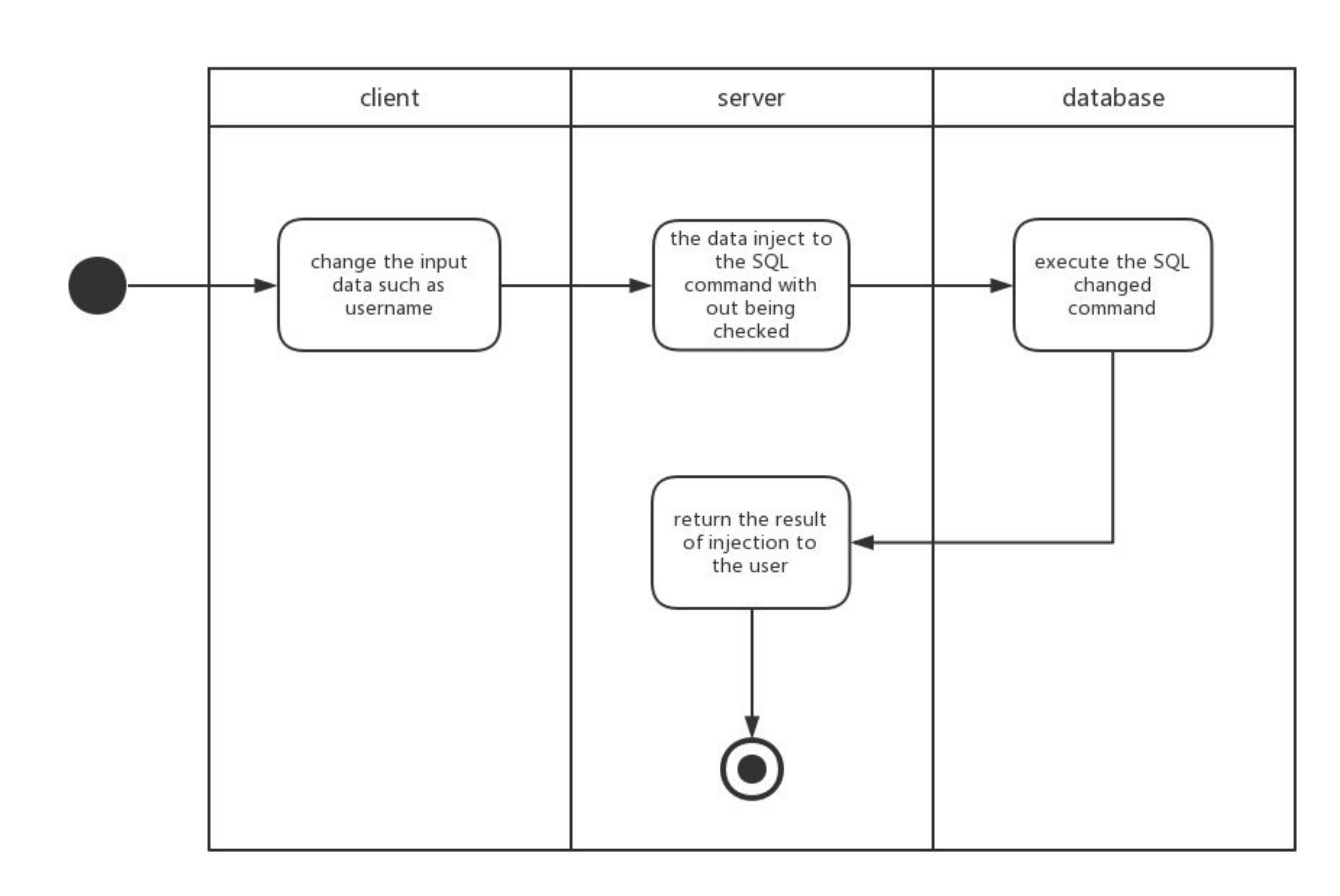
2. Compilers these days have built-in counter measures for detecting potential format string vulnerabilities.

3. Address Randomization

SQL injection

- One of the most common attacks on web applications.
- Sql is a code injection technique.
- Exploits vulnerabilities between web applications and database servers.
- Occurs when user inputs are not properly checked.

Activity Diagram


client
server
database
change the input data such as username
the data inject to the SQL command with out being checked
execute the SQL changed command
return the result of injection to the user

SQL Injection Demo

1. User:Admin Pass:seedadmin
2. User:Alice Pass:seedalice
3. User:Admin’#

SQL Injection preventive measures

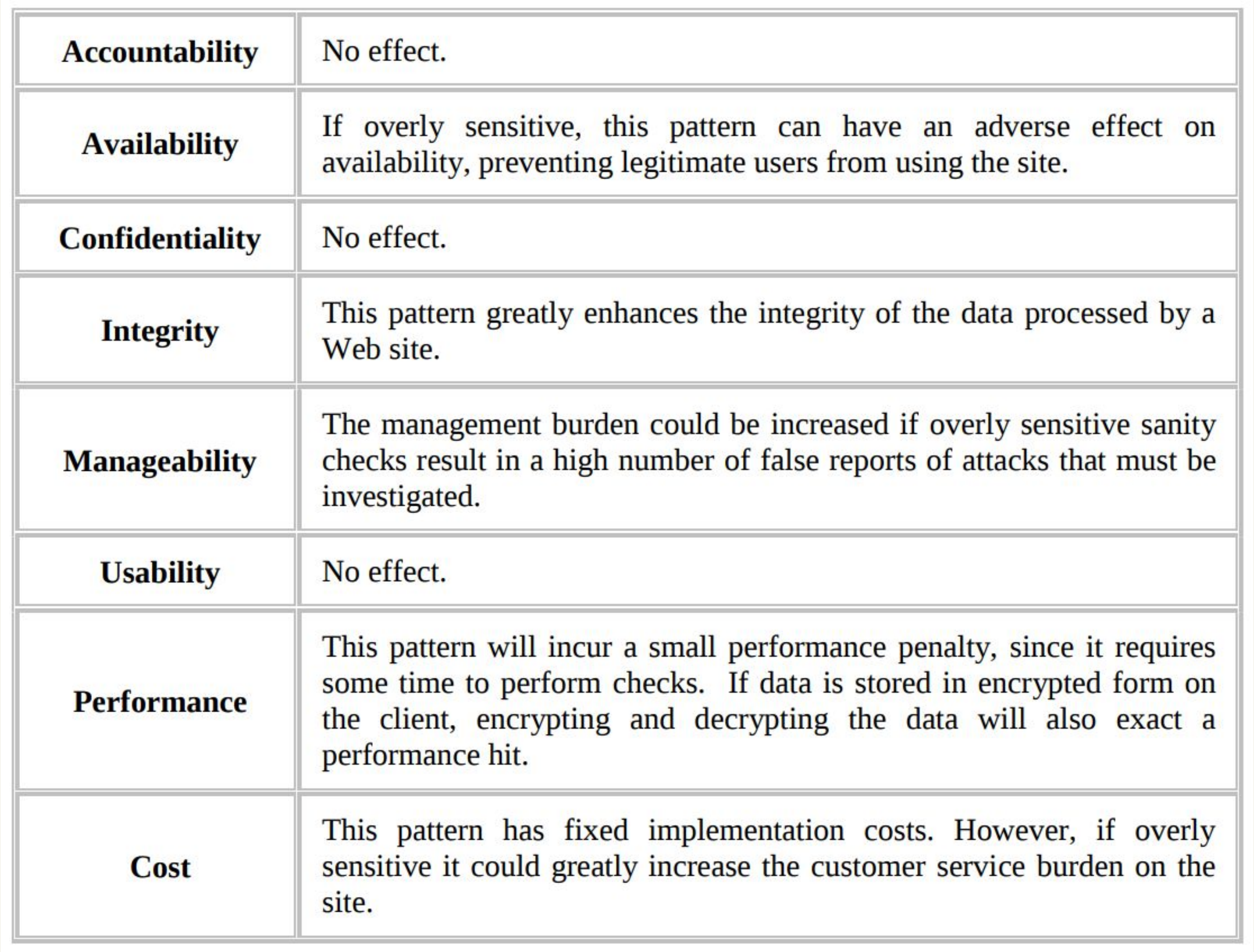
- Do Some validation checks at client.
- Usage of Prepared statements.

Client Input Filter Pattern (Sanitize i/p)

Ignore the client validation. Do the validation at the server once again.

- Data validity checks
- Sensitive information from the client should be kept in a encrypted, tamper-proof form.
- Discard request that are obviously questionable.
- Filter the data submitted from the client.
- Remove script tags.

Trade offs



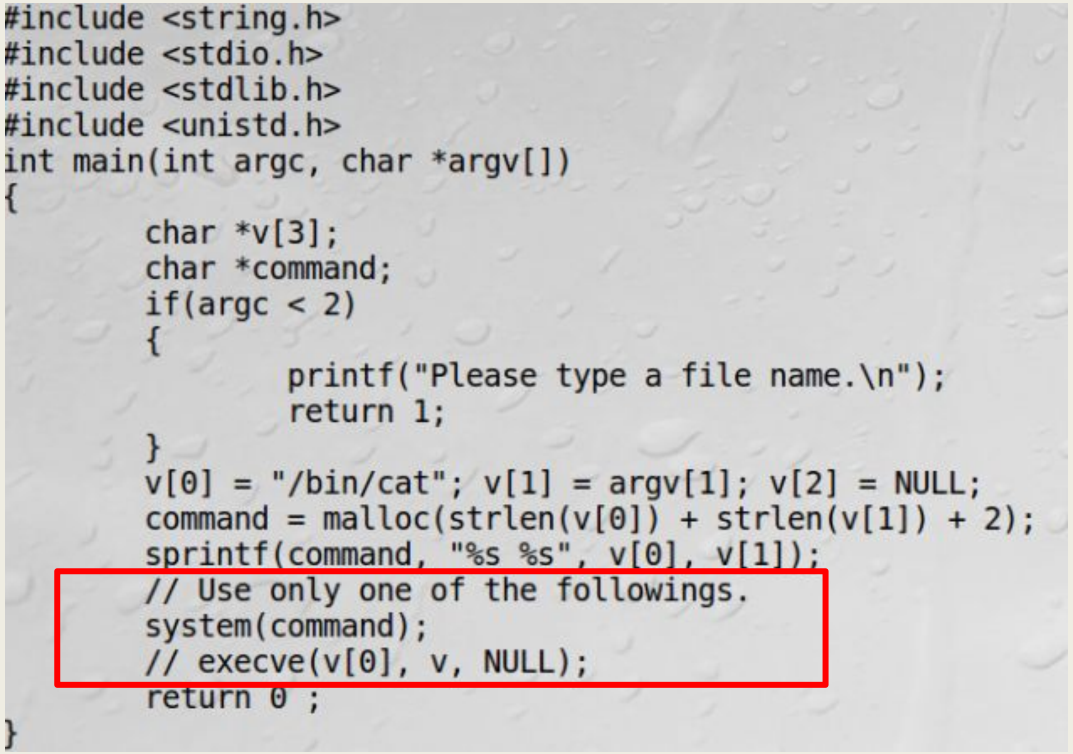
Accountability	No effect.
Availability	If overly sensitive, this pattern can have an adverse effect on availability, preventing legitimate users from using the site.
Confidentiality	No effect.
Integrity	This pattern greatly enhances the integrity of the data processed by a Web site.
Manageability	The management burden could be increased if overly sensitive sanity checks result in a high number of false reports of attacks that must be investigated.
Usability	No effect.
Performance	This pattern will incur a small performance penalty, since it requires some time to perform checks. If data is stored in encrypted form on the client, encrypting and decrypting the data will also exact a performance hit.
Cost	This pattern has fixed implementation costs. However, if overly sensitive it could greatly increase the customer service burden on the site.

More examples: Lack of Input Sanitization

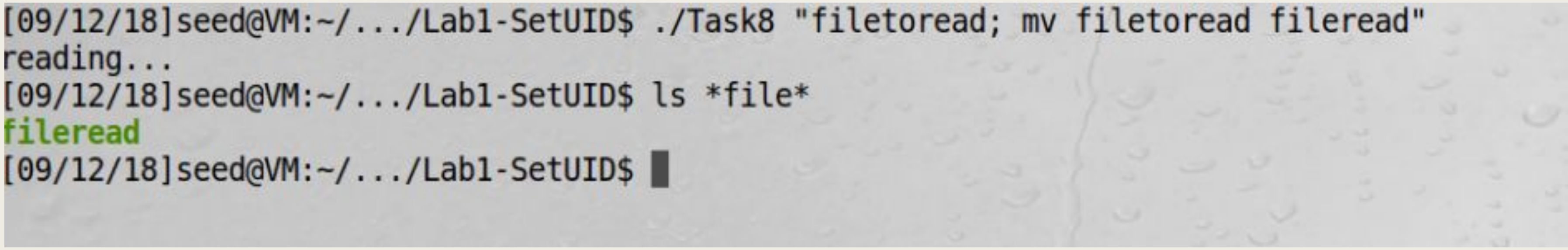
- System() call
- Shell-Shock – Command Injection
- XSS (Cross Site Scripting)
- Kernel Memory Access using Loadable Kernel Module

System()

- A C function in stdlib.h
- Execute a shell command.
- Treats the argument as shell command.

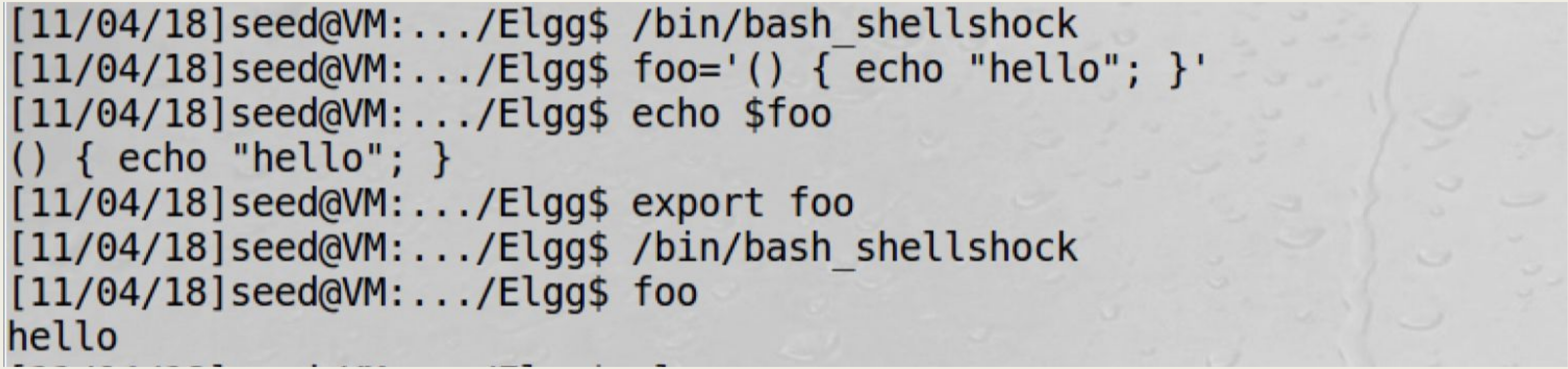


```
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
int main(int argc, char *argv[])
{
        char *v[3];
        char *command;
        if(argc < 2)
        {
                printf("Please type a file name.\n");
                return 1;
        }
        v[0] = "/bin/cat"; v[1] = argv[1]; v[2] = NULL;
        command = malloc(strlen(v[0]) + strlen(v[1]) + 2);
        sprintf(command, "%s %s", v[0], v[1]);
        // Use only one of the followings.
        system(command);
        // execve(v[0], v, NULL);
        return 0 ;
}
```

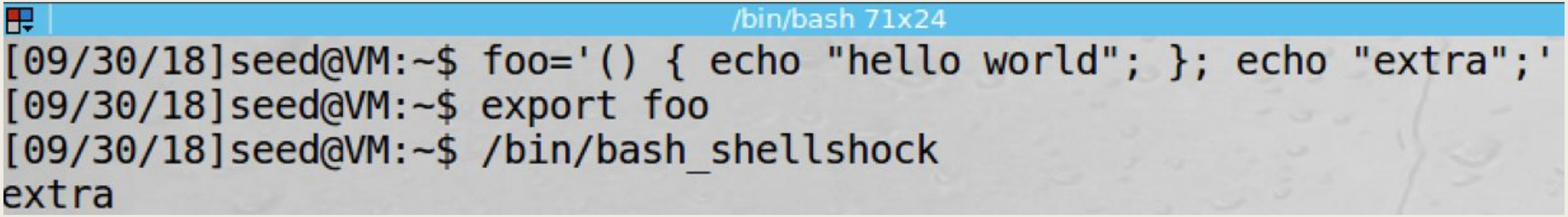


```
[09/12/18]seed@VM:~/.../Lab1-SetUID$ ./Task8 "filetoread; mv filetoread fileread"
reading...
[09/12/18]seed@VM:~/.../Lab1-SetUID$ ls *file*
fileread
[09/12/18]seed@VM:~/.../Lab1-SetUID$ 
```

Shell-Shock – Command Injection

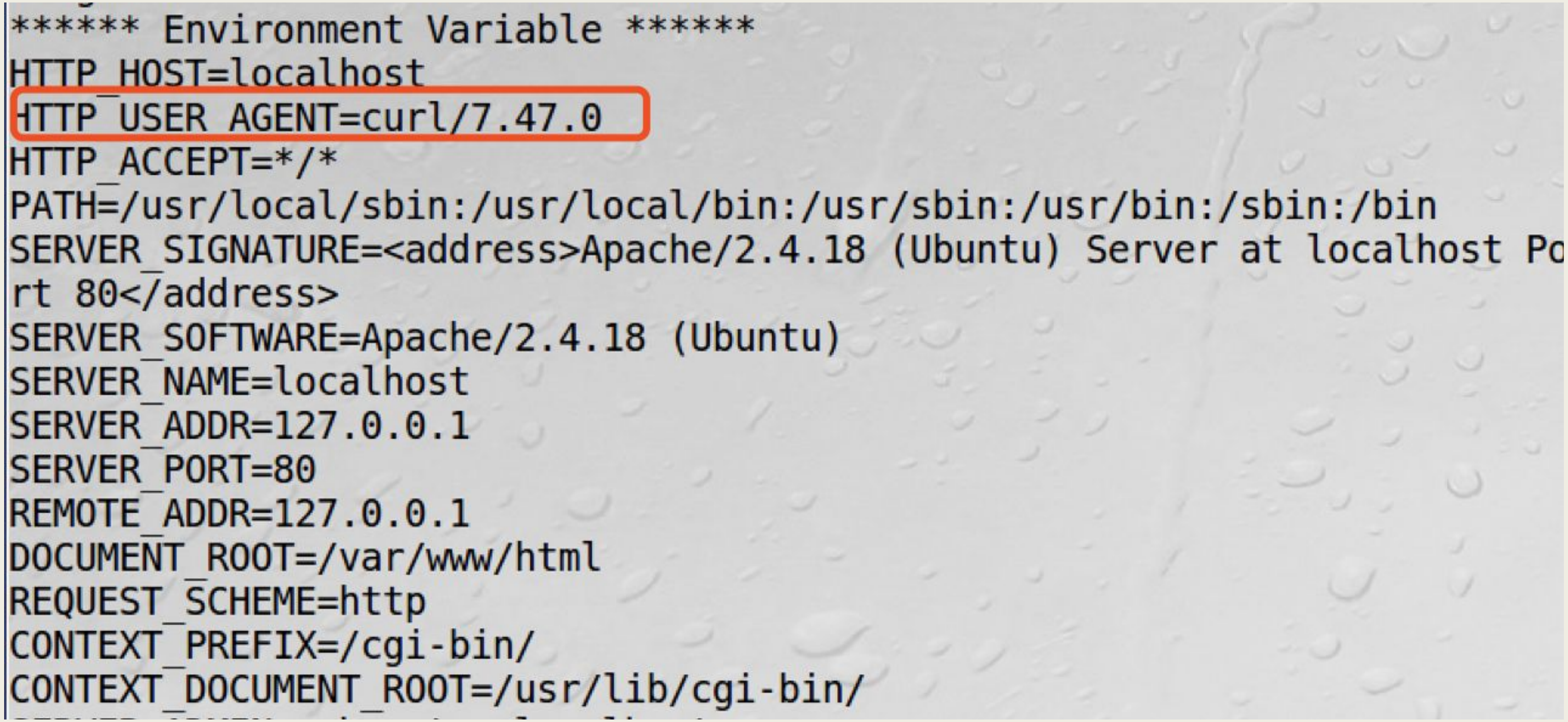


```
[11/04/18]seed@VM:.../Elgg$ /bin/bash_shellshock
[11/04/18]seed@VM:.../Elgg$ foo='() { echo "hello"; }'
[11/04/18]seed@VM:.../Elgg$ echo $foo
() { echo "hello"; }
[11/04/18]seed@VM:.../Elgg$ export foo
[11/04/18]seed@VM:.../Elgg$ /bin/bash_shellshock
[11/04/18]seed@VM:.../Elgg$ foo
hello
```



```
[09/30/18]seed@VM:~$ foo='() { echo "hello world"; }; echo "extra";'
[09/30/18]seed@VM:~$ export foo
[09/30/18]seed@VM:~$ /bin/bash_shellshock
extra
```

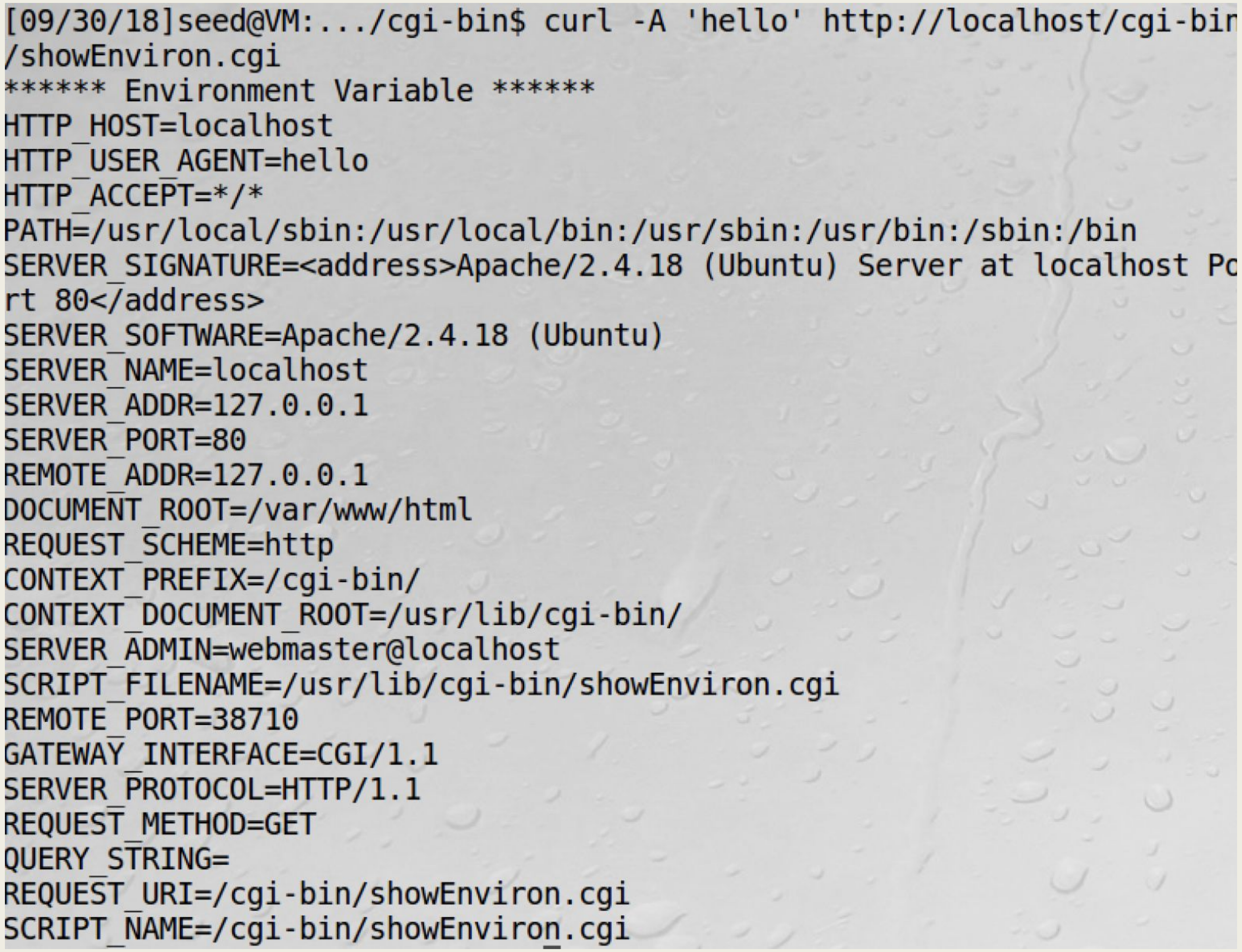
Shell-Shock – Command Injection



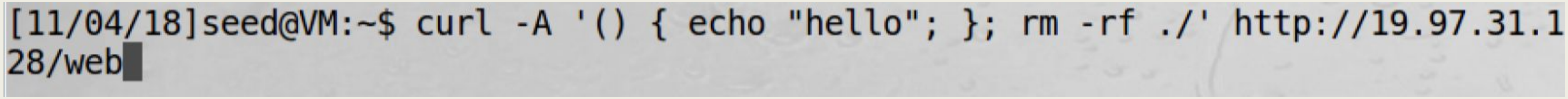
```
****** Environment Variable ******
HTTP_HOST=localhost
HTTP_USER_AGENT=curl/7.47.0
HTTP_ACCEPT=*/*
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
SERVER_SIGNATURE=<address>Apache/2.4.18 (Ubuntu) Server at localhost Po
rt 80</address>
SERVER_SOFTWARE=Apache/2.4.18 (Ubuntu)
SERVER_NAME=localhost
SERVER_ADDR=127.0.0.1
SERVER_PORT=80
REMOTE_ADDR=127.0.0.1
DOCUMENT_ROOT=/var/www/html
REQUEST_SCHEME=http
CONTEXT_PREFIX=/cgi-bin/
CONTEXT_DOCUMENT_ROOT=/usr/lib/cgi-bin/
```

curl -A, --user-agent <agent string>

Shell-Shock – Command Injection



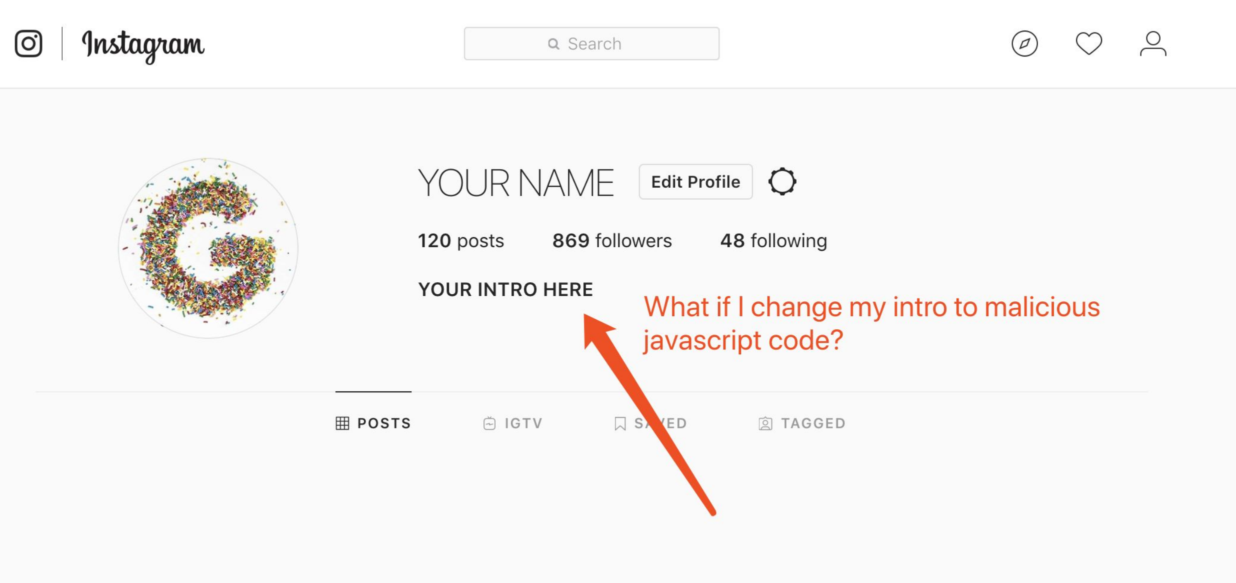
```
[09/30/18]seed@VM:.../cgi-bin$ curl -A 'hello' http://localhost/cgi-bin
/showEnviron.cgi
****** Environment Variable ******
HTTP_HOST=localhost
HTTP_USER_AGENT=hello
HTTP_ACCEPT=*/*
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
SERVER_SIGNATURE=<address>Apache/2.4.18 (Ubuntu) Server at localhost Po
rt 80</address>
SERVER_SOFTWARE=Apache/2.4.18 (Ubuntu)
SERVER_NAME=localhost
SERVER_ADDR=127.0.0.1
SERVER_PORT=80
REMOTE_ADDR=127.0.0.1
DOCUMENT_ROOT=/var/www/html
REQUEST_SCHEME=http
CONTEXT_PREFIX=/cgi-bin/
CONTEXT_DOCUMENT_ROOT=/usr/lib/cgi-bin/
SERVER_ADMIN=webmaster@localhost
SCRIPT_FILENAME=/usr/lib/cgi-bin/showEnviron.cgi
REMOTE_PORT=38710
GATEWAY_INTERFACE=CGI/1.1
SERVER_PROTOCOL=HTTP/1.1
REQUEST_METHOD=GET
QUERY_STRING=
REQUEST_URI=/cgi-bin/showEnviron.cgi
SCRIPT_NAME=/cgi-bin/showEnviron.cgi
```



```
[11/04/18]seed@VM:~$ curl -A '() { echo "hello"; }; rm -rf ./' http://19.97.31.1
28/web
```

XSS (Cross Site Scripting)

It is a security vulnerability attack for web applications, which is a kind of code injection. It allows malicious users to inject code into a web page, and other users are affected when they view the web page. This type of attack usually includes HTML and a client-side scripting language.

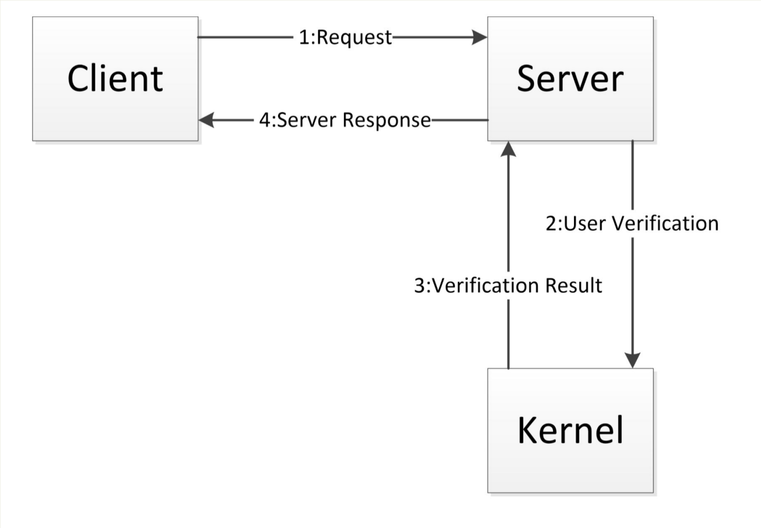


What if I change my intro to malicious javascript code?

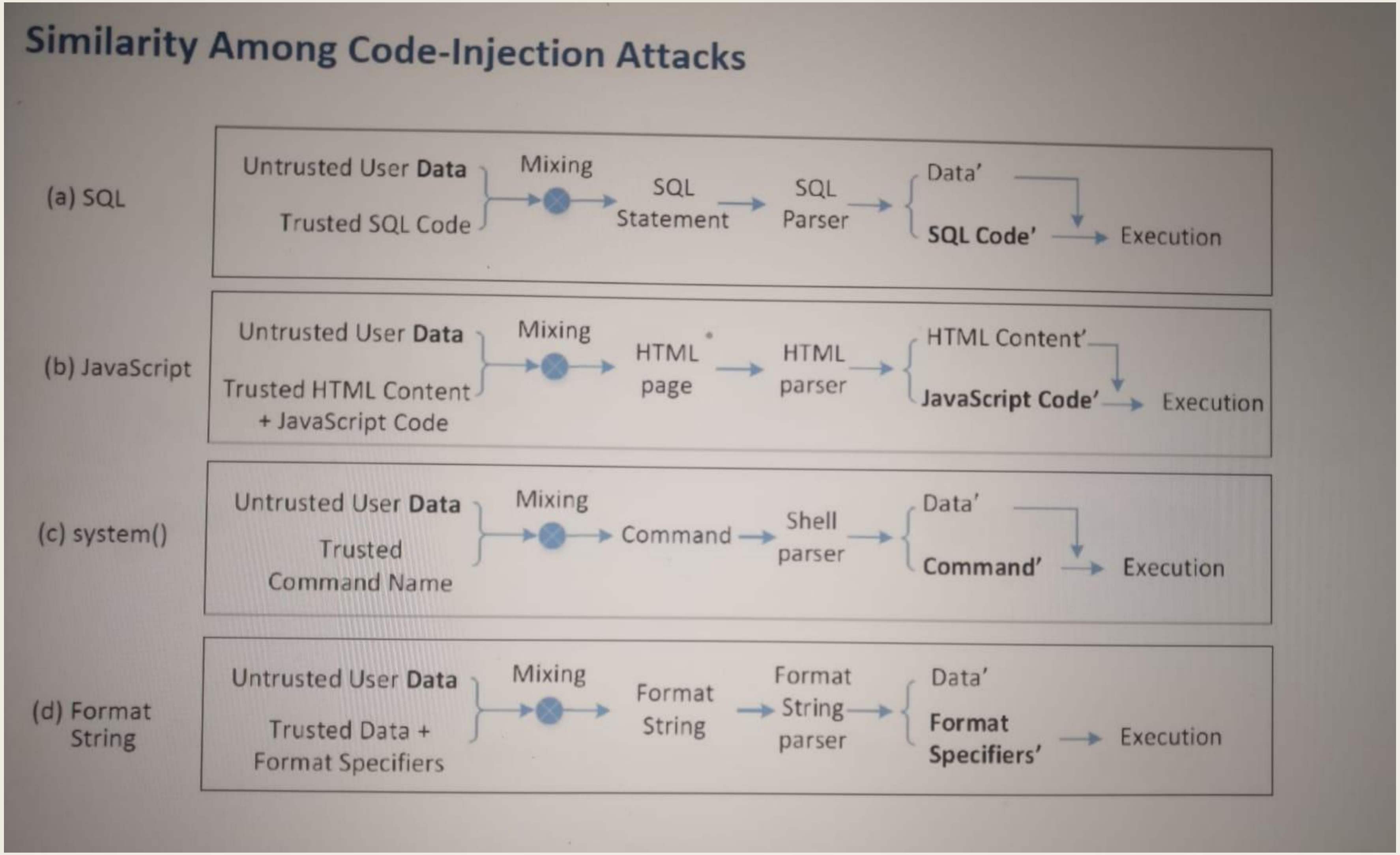
Kernel Memory Access

- Often when users are required to interact directly with kernel using features like Loadable Kernel Modules (LKM), we neglect that invalid usage might lead to the application crashing.

- While this is an built functionality in linux to prevent modification and access to protected memory, it is essential to note that these accesses to memory must be pre-defined and 'white-listed' while other exceptions must be handled so that they may not affect program functionality.



Similarity Among Code-Injection Attacks



Similarity Among Code-Injection Attacks

(a) SQL

Untrusted User **Data**
Trusted SQL Code
Mixing
SQL Statement
SQL Parser
Data'
SQL Code'
Execution

(b) JavaScript

Untrusted User **Data**
Trusted HTML Content + JavaScript Code
Mixing
HTML page
HTML parser
HTML Content'
JavaScript Code'
Execution

(c) system()

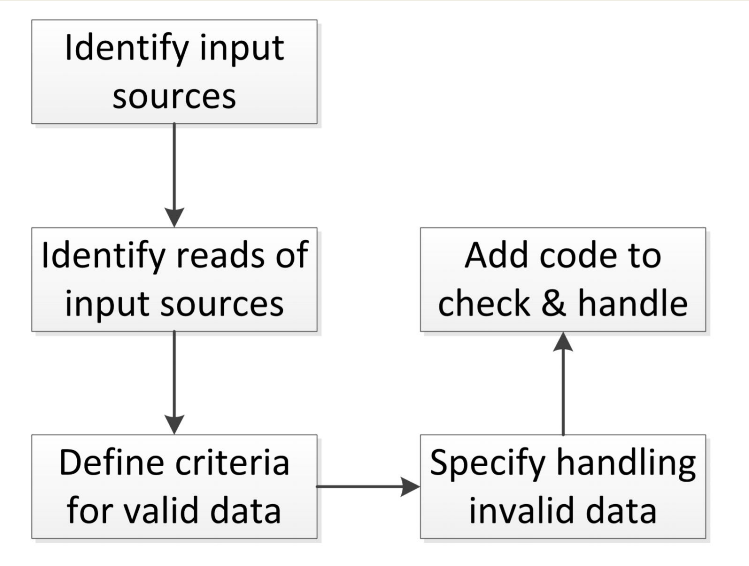
Untrusted User **Data**
Trusted Command Name
Mixing
Command
Shell parser
Data'
Command'
Execution

(d) Format String

Untrusted User **Data**
Trusted Data + Format Specifiers
Mixing
Format String
Format String parser
Data'
Format Specifiers'
Execution

Input Sanitization Pattern

- Input validation should be done at trusted server/client side
  - *Identify the source*
  - *Parts in software which reads the input*
  - *Define criteria for validation*
  - *Handling invalid cases*
  - *Code for validation and handling invalid cases.*


Identify input sources
Identify reads of input sources
Define criteria for valid data
Specify handling invalid data
Add code to check & handle

Advantages of Implementing Security Patterns :

- Secure coding techniques ensure greater system security.
- Security is viewed as functional requirement in Software Engg.
- The confidentiality & privacy of client will be improved.
- A small number of patterns would improve performance, like Client Data Storage pattern, etc.
- While cost of implementation is incurred, it is a better than the cost incurred when there a security flaw is exploited.

Disadvantages of implementing Security Patterns:

- Most of patterns would incur a performance penalty.
- Cost in terms of manpower, training, testing and infrastructure increases.
- Specific security solutions get outdated quickly and there is a constant need to be updated.

References:

1. Coursework and Labs : CSE 644 (Internet Security),SU
2. Coursework and Labs : CSE 643 (Computer Security),SU
3. Coursework and Labs : IST 704 (Applied Information Security),SU
4. Code Demonstrations : http://www.cis.syr.edu/~wedu/seed/labs.html
5. Security Patterns Repository v1.0 Darrell M. Kienzle et. al
6. SU IT Services-InfoSec (Information Security Policy) - https://its.syr.edu/about-us/departments/information-security/
7. Computer Security, A Hands-on Approach by Wenliang Du
8. Special Thanks - Chris Croad (CISO, ITS, SU), Dr Kevin Du (EECS, SU), Benson Poikayil (InfoSec Ops, ITS, SU)
9. Design Patterns, Erich Gamma et. al
10. http://www.informationisbeautiful.net/visualizations/worlds-biggest-data-breaches-hacks/