

SECURITY PATTERNS

Harika Bandaru, Vishal Parekh, Honey Shah, Weiheng Chai
CSE-776 (Design Patterns), Fall 2018
Syracuse University

Why Security Pattern

- Recent

- [ht](#)

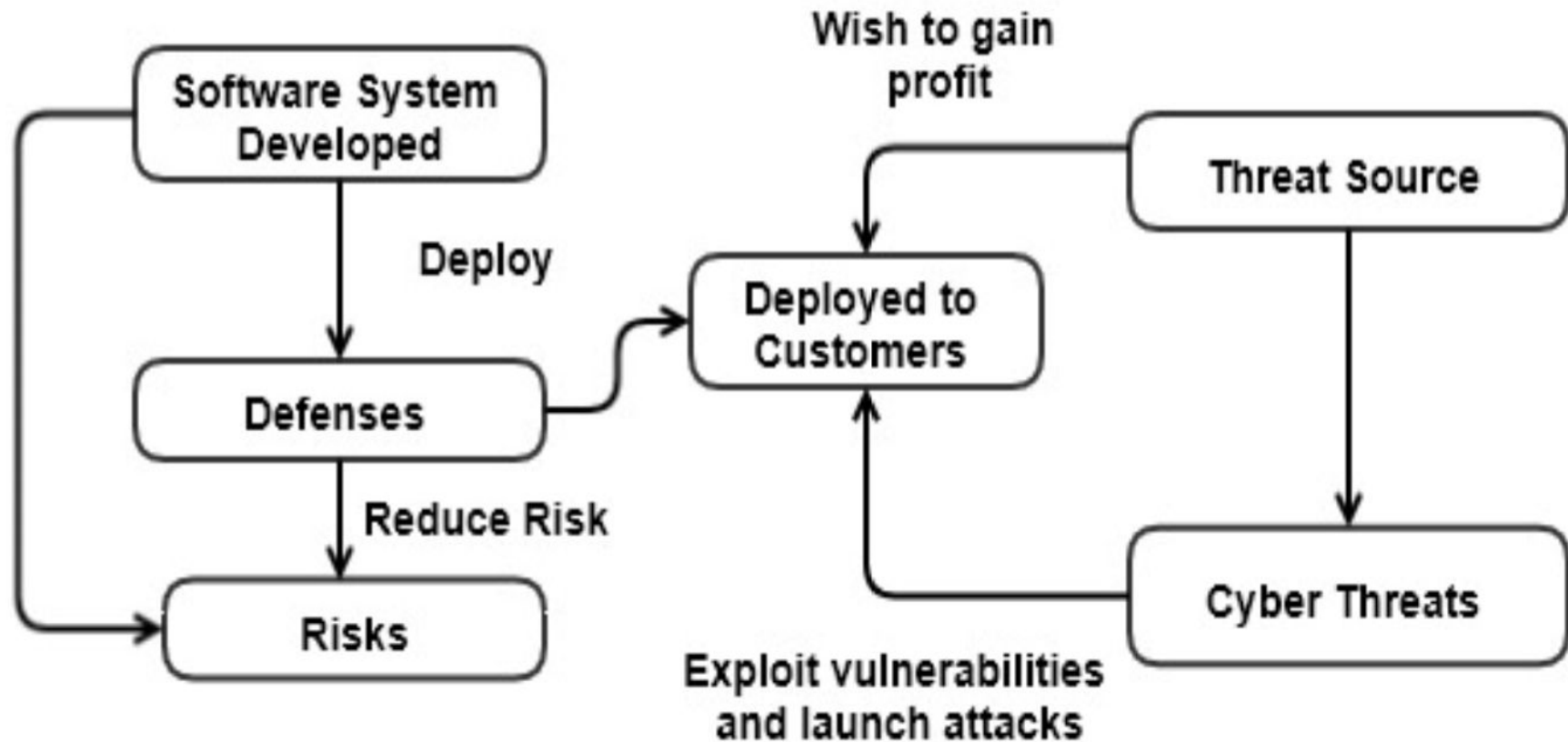
- *Eq*

- [ht](#)

- [20](#)

- [ht](#)

- [big](#)



Intent, Motivation

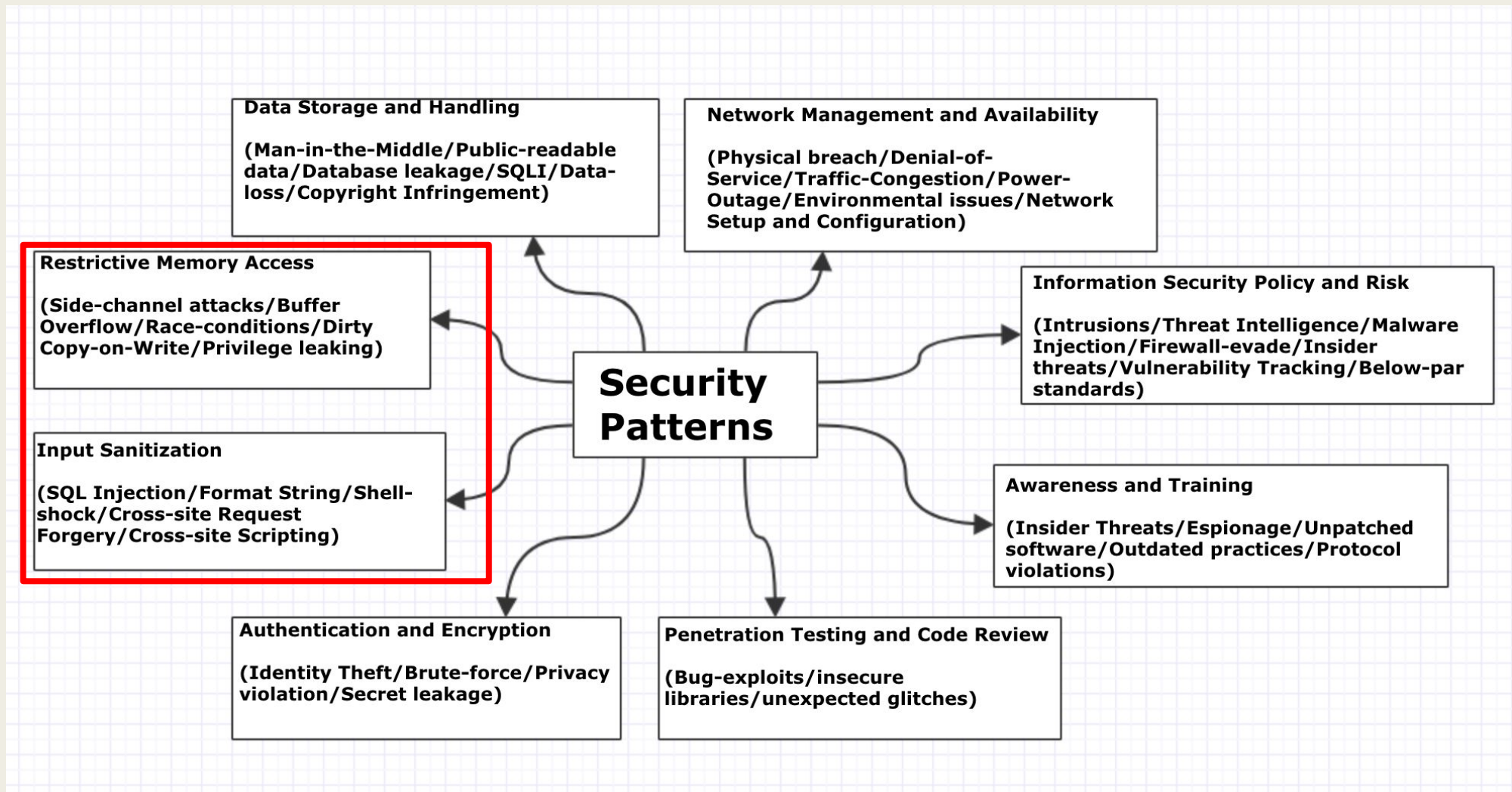
- Any pattern is a optimal re-usable solution to common problems.
- Initially security was considered as a Non-Functional-Requirement for system implementation.
- Which resulted in many breaches.
- Improvement in tech has made 'Hacking' easier than ever before with novel means being discovered each day.
- Complexity Vs Security or Performance Vs Security

Need for Security Patterns:

- Ensure Information Security by maintaining CIA (Confidentiality, Integrity and Availability)
- A security pattern is a solution that addresses a class of security problems/flaws.
- Examples of security patterns: User input sanitization and Restrictive memory access



A Map of Security Patterns



Some Examples of Security Flaws and exploitation

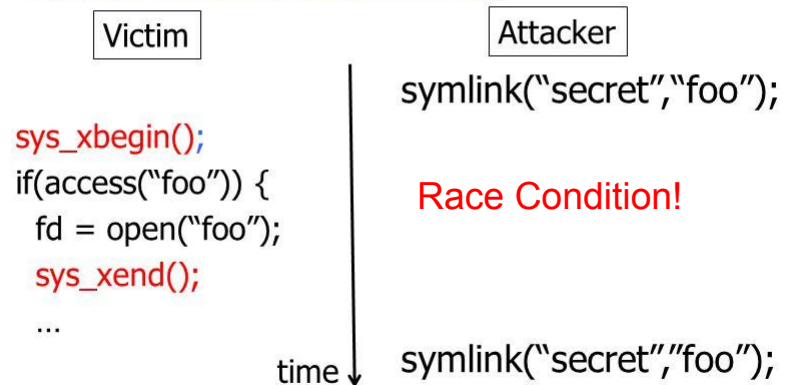
```
The program has been running 70452 times so far.  
./exploit-T4.sh: line 13: 10182 Segmentation fault      ./stack  
2 minutes and 4 seconds elapsed.  
The program has been running 70453 times so far.  
# id  
uid=0(root) gid=1000(seed) groups=1000(seed),4(adm),24(cdrom),27(sudo),30(dip),46(plugdev),113(lpadmin),128(sambashare)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

Privilege Escalation!

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

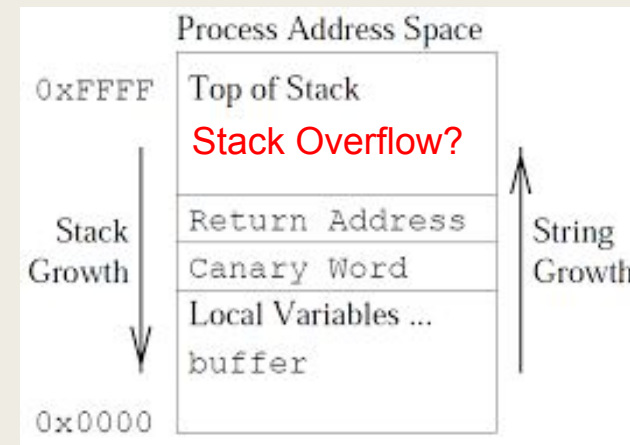
TOCTTOU Example Redux

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



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

Buffer Overflow?



Sign in

Email

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

Password

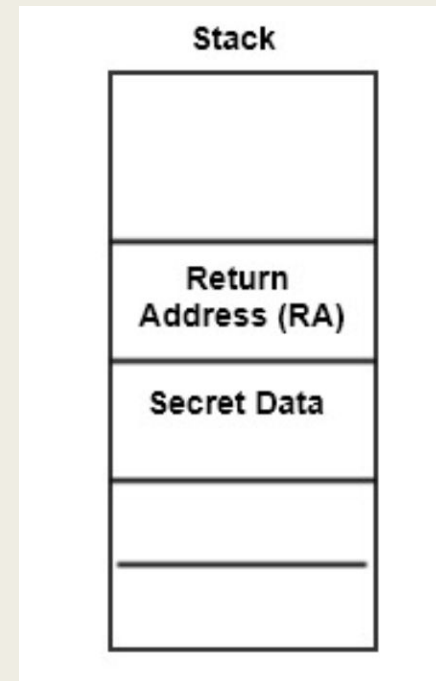
.....

Sign in ☒ Stay signed in

Demo of Buffer Overflow:

Statistics:

https://nvd.nist.gov/vuln/search/results?form_type=Basic&results_type=overview&query=Buffer+Overflow&search_type=last3months

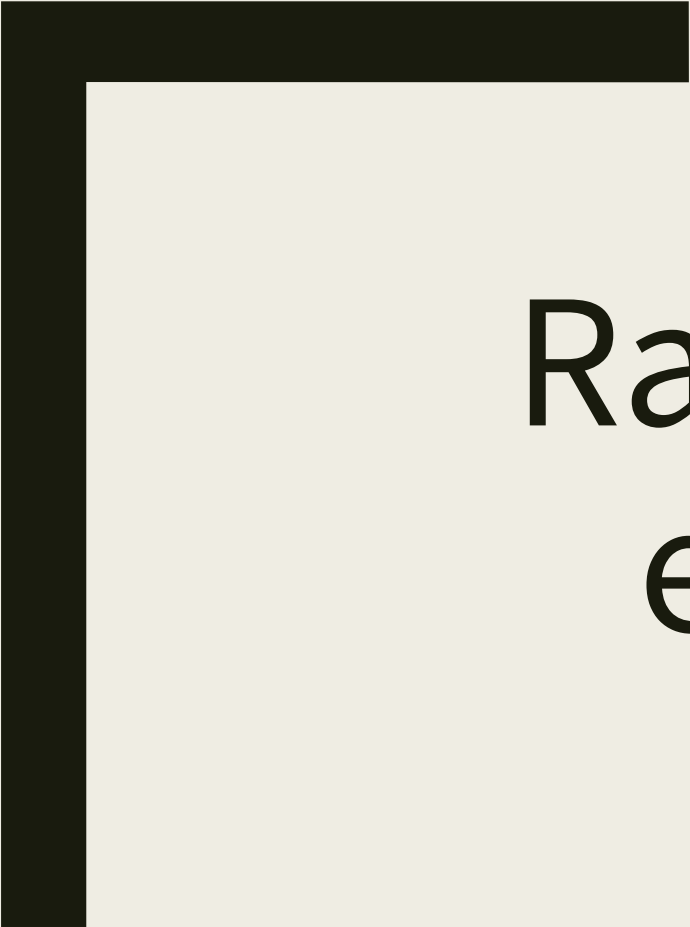


Trusted zone

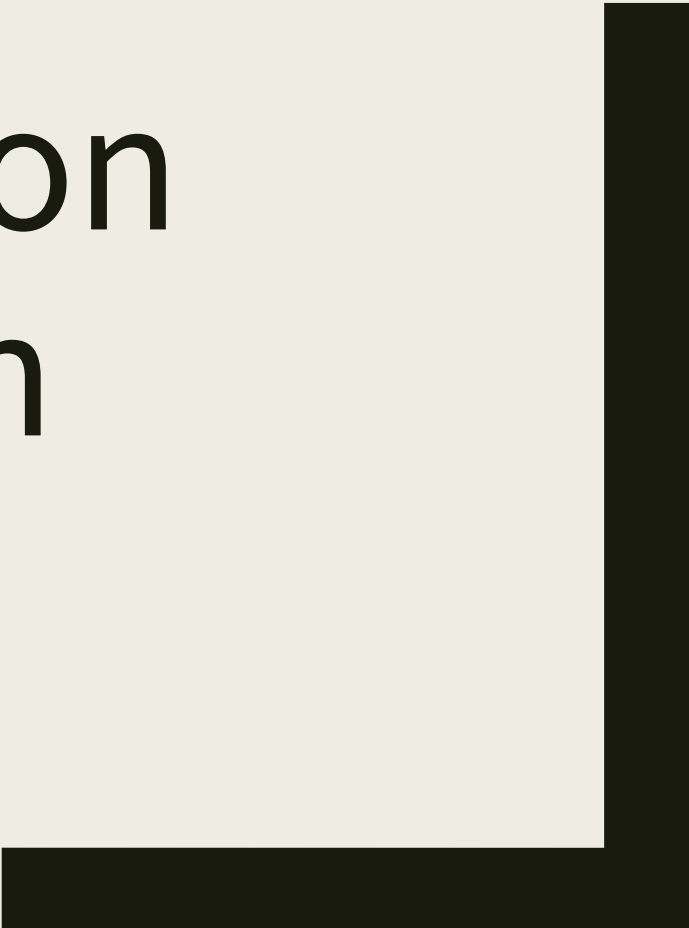
- Separation of user mode from OS which is more trusted.
- Privilege separation. Leverage the support of hardware.
- Ring architecture.
- System api for user program to talk with OS.
- Process memory separation.

Defense Measures

- Use type safe languages like C++,java,etc that does boundary checking automatically,
- Why does this vulnerabilities exists,though we have type safe languages?
 - Flexibility : Low level languages
 - Performance : Avoiding type safety checks etc.
- Use secure library functions.
- Validate user-input.
- Source code analysis tools
 - [https://www.owasp.org/index.php/Source Code Analysis Tools](https://www.owasp.org/index.php/Source_Code_Analysis_Tools)



Race Condition exploitation



```

[10/01/18]seed@VM:~/.../RaceCondition$ gedit vulp.c
[10/01/18]seed@VM:~/.../RaceCondition$ gcc -o vulp vulp.c
[10/01/18]seed@VM:~/.../RaceCondition$ sudo sysctl -w fs.protected_symlinks=0
fs.protected_symlinks = 0
[10/01/18]seed@VM:~/.../RaceCondition$ cat vulp.c
/* vulp.c */

```

```

#include <stdio.h>
#include <unistd.h>
#include <string.h>

```

```

int main()
{
    char * fn = "/tmp/XYZ";
    char buffer[60];
    FILE *fp;

```

```

/* get user input */
scanf("%50s", buffer );

```

```

if(!access(fn, W_OK)){

```

```

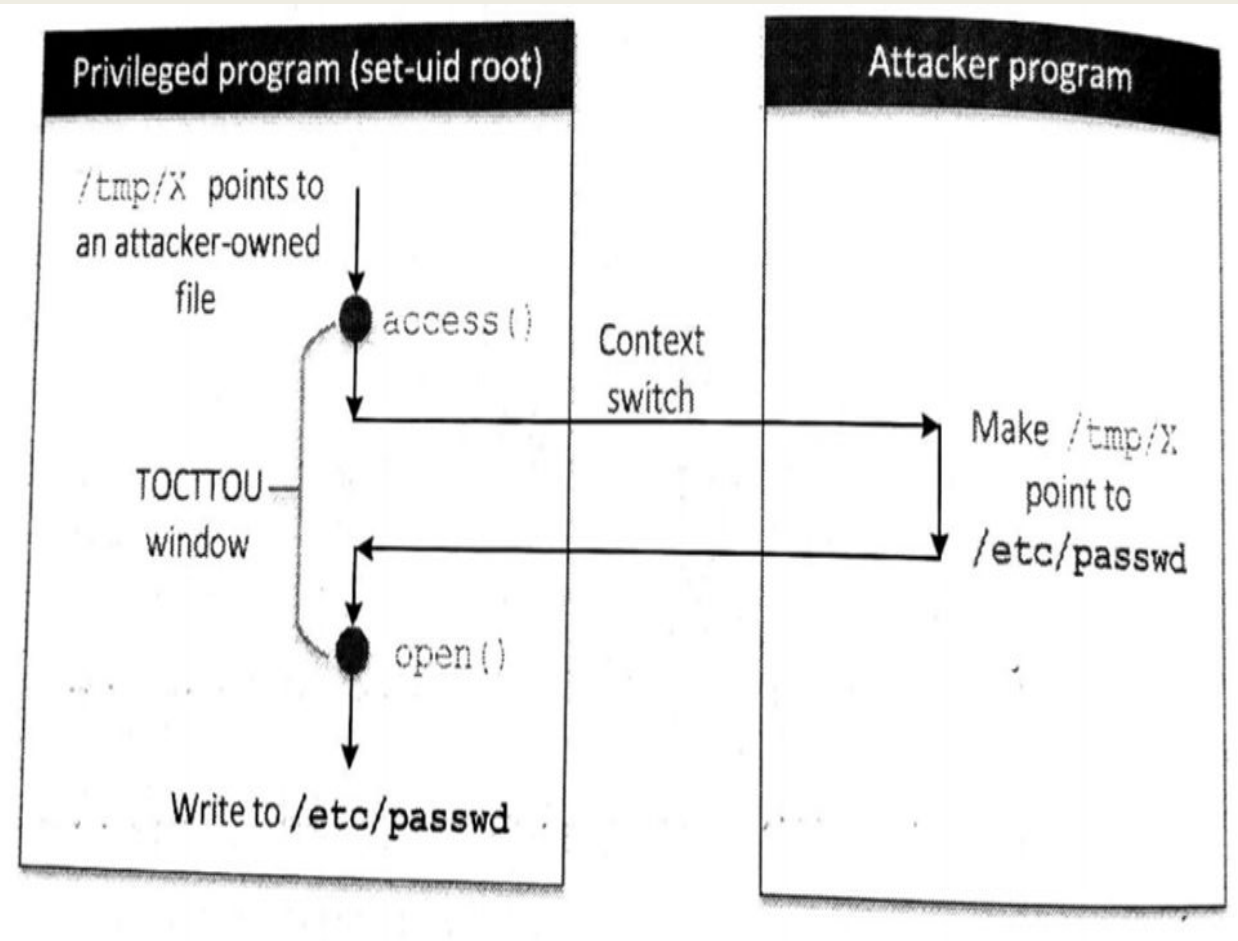
    fp = fopen(fn, "a+");
    fwrite("\n", sizeof(char), 1, fp);
    fwrite(buffer, sizeof(char), strlen(buffer), fp);
    fclose(fp);

```

```

}
else printf("No permission \n");
}

```



```

unlink("/tmp/XYZ");
symlink("/etc/passwd", "/tmp/XYZ");
usleep(10000);

```

Solution

```
    struct stat statBefore, statAfter;

1:  lstat("/tmp/X", &statBefore);

2:  if (!access("/tmp/X", O_RDWR)) {
    /* the real UID has access right */
3:    f = open("/tmp/X", O_RDWR);
4:    lstat("/tmp/X", &statAfter);

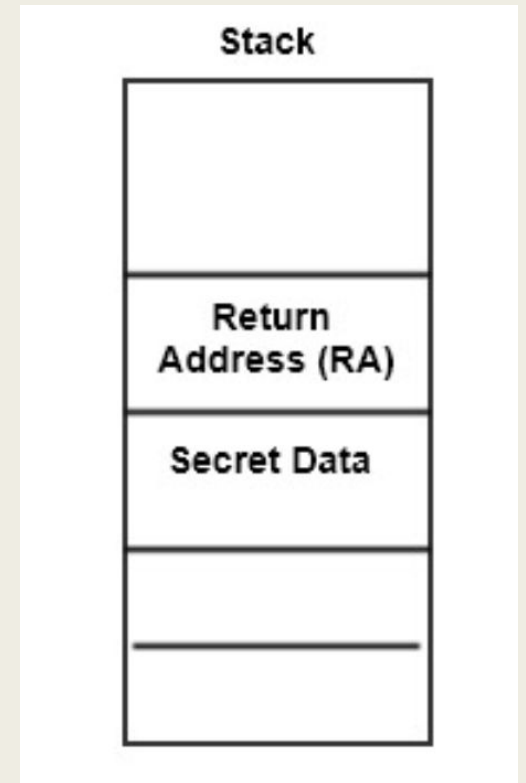
5:    if (statAfter.st_ino == statBefore.st_ino)
6:    { /* the I-node is still the same */
7:      Write_to_file(f)
8:    }
9:    else perror("Race Condition Attacks!");
10: }
11: else fprintf(stderr, "Permission denied\n");
```

Restrictive Memory Access Pattern

■ The possible exploitation:

1. Heap Buffer Overflow
2. Shellshock-BashCGI
3. Side Channel Attacks
4. Dirty Copy-on-Write
5. OpenSSL Heartbleed

(Taken as per Linux context)



Solution for memory access

1. **Eliminate Racing or create more hurdles to minimise the attacker to win.**
2. **Use secure libraries/frameworks**
3. **Least Privilege Principle and service privilege levels**
4. **Sandboxing/clear memory boundaries**
5. **Update systems and applications**

Lack of Input Sanitization

- **Validation:** Validation is the process of ensuring that **input** data falls within the expected domain of valid program **input**.
- **Sanitization** is the process of removing sensitive information from a document or other message
- SQL Injection
- Format String
- System()
- Shell-Shock – Command Injection
- XSS (Cross Site Scripting)
- Kernel Memory Access using Loadable Kernel Module

SQL injection

SQL injection is a code injection technique, used to attack data-driven applications, in which nefarious SQL statements are inserted into an entry field for execution.

- 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.

Demo of SQL Injection

```
//author Weiheng
//for cse776 security pattern presentation
//SQL injection demo
//2018/11/4
using System;
using MySql.Data.MySqlClient;

namespace SQL_inj_SAMPLE
{
    class MyWeb
    {
        public bool Login()
        {
            Console.WriteLine("please input the username:");
            string username = Console.ReadLine();
            Console.WriteLine("please input the password:");
            string password = Console.ReadLine();
            String connetStr = "server=127.0.0.1;port=3306;user=root;password=123456; database=mydb;";
            MySqlConnection conn = new MySqlConnection(connetStr);
            try
            {
                conn.Open();//open channel
                Console.WriteLine("connect successfully");
            }
            catch (MySqlException ex)
            {
                Console.WriteLine(ex.Message);
            }

            string mysqlcmd = "select * from user where username=" + username + " and Password=" + password + " ";
            MySqlCommand cmd = new MySqlCommand(mysqlcmd, conn);
            MySqlDataReader reader = cmd.ExecuteReader();
            if (!reader.Read()) Console.WriteLine("wrong username or password");
            else
            {
                Console.WriteLine("login success!");
            }
            conn.Close();
            return true;
        }
    }
}
```

main function

```
static void Main(string[] args)
{
    MyWeb web = new MyWeb();
    int i = 0;
    while (i < 10) {
        web.Login();
        i++;
    }
}
```

initialize the database:

	username	Password
▶	Weiheng	qwertyu
	Weiheng1	qwertyu1
	Weiheng2	qwertyu2
	Weiheng3	qwertyu3
	Weiheng5	qwertyu5
•	NULL	NULL

Run the code

```
C:\Windows\system32\cmd.exe
} please input the username:
s Weiheng3
M please input the password
M qwertyu3
i connect successfully
e login success!
} please input the username:
}
c
r
```

```
login success.
} please input the username:
} test1
c please input the password
r qwertyu
} connect successfully
ti wrong username or password
} please input the username:
M
i
```

Login using SQL injection

```
C:\Windows\system32\cmd.exe
please input the username:
test
please input the password:
'OR 1=1;#
connect successfully
login success!
please input the username:
```

30
31

```
select * from user where username=' test' and Password=' 'OR 1=1;#
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content: I

username	Password
testsqlinjection	test
Weiheng	qwertyu
Weiheng2	qwertyu2
Weiheng3	qwertyu3
Weiheng5	qwertyu5
NULL	NULL

`select * from user where username=' test' and Password=' 'OR 1=1';#`

Add new users using SQL injection

```
please input the username:
;insert into user(username,Password)values(' testsqlinjection',' test');#
please input the password
123
connect successfully
wrong username or password
please input the username:
```

username	Password
testsqlinj...	test
Weiheng	qwertyu
Weiheng1	qwertyu1
Weiheng2	qwertyu2
Weiheng3	qwertyu3
Weiheng5	qwertyu5

```
please input the username:
testsqlinjection
please input the password
test
connect successfully
login success!
please input the username:
```

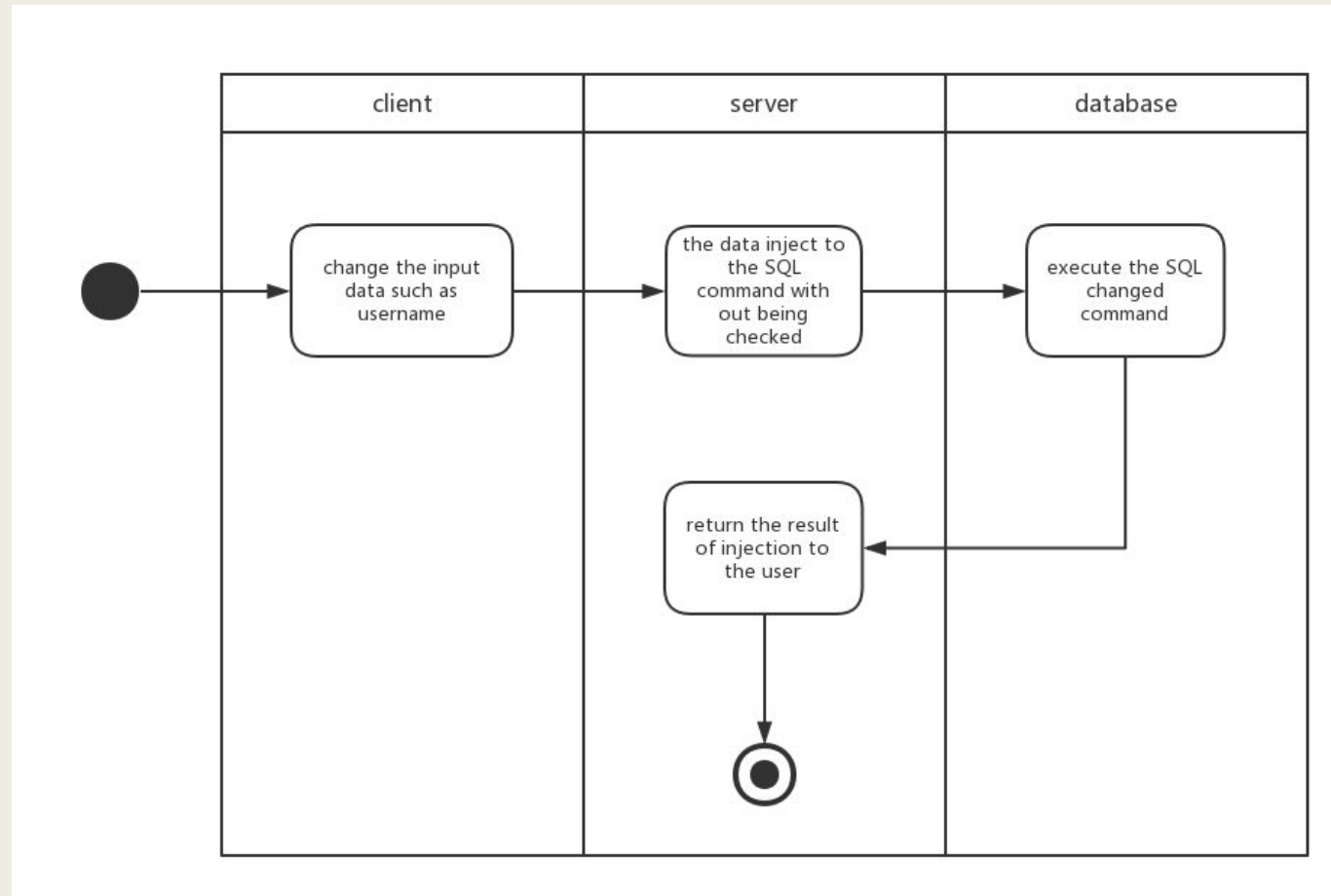
`select * from user where username="" ;insert into user() values();#and Password="" + password + "";`

Delete existing users using SQL injection

```
please input the username:
';delete from user where username='Weiheng1';#
please input the password
123
connect successfully
wrong username or password
please input the username:
```

	username	Password
▶	testsqlinj...	test
	Weiheng	qwertyu
	Weiheng2	qwertyu2
	Weiheng3	qwertyu3
	Weiheng5	qwertyu5
*	NULL	NULL

Activity diagram



Some common SQL statements for SQL injection

■ And 1=1/And 1=2

■ OR 1=1/OR 1=2

■ +、-、*、/、>、<、<=、>=

■ 1 like 1/1 like 2

■

■ And '1' = '1/And
'1' = '2

■ OR '1' = '1/OR
'1' = '2' #

■ +' /+' 1、-' 0/-' 1、>、<、
<=、>=

■ 1' like '1/1' like '2

■

SQL Injection prevention

- **Option 1:** Use of Prepared Statements
- **Option 2:** Do Some validation checks at client

Prepared Statements demo

```
class MyWeb
{
    public bool Login()
    {
        Console.WriteLine("please input the username:");
        string username = Console.ReadLine();
        Console.WriteLine("please input the password");
        string password = Console.ReadLine();
        String connetStr = "server=127.0.0.1;port=3306;user=root;password=123456; database=mydb;";
        MySqlConnection conn = new MySqlConnection(connetStr);
        try
        {
            conn.Open();//open channel
            Console.WriteLine("connect successfully");
        }
        catch (MySqlException ex)
        {
            Console.WriteLine(ex.Message);
        }
        string mysqlcmd = "select * from user where username=@val1 and Password=@val2";

        MySqlCommand cmd = new MySqlCommand(mysqlcmd, conn);
        cmd.Parameters.AddWithValue("@val1", username);
        cmd.Parameters.AddWithValue("@val2", password);
        MySqlDataReader reader = cmd.ExecuteReader();
        if (!reader.Read()) Console.WriteLine("wrong username or password");
        else
        {
            Console.WriteLine("login success!");
        }
        conn.Close();
        return true;
    }
}
```

```
please input the username:
' OR 1=1;#
please input the password
qwe
connect successfully
wrong username or password
please input the username:
```

Validation checks demo(using regular expression)

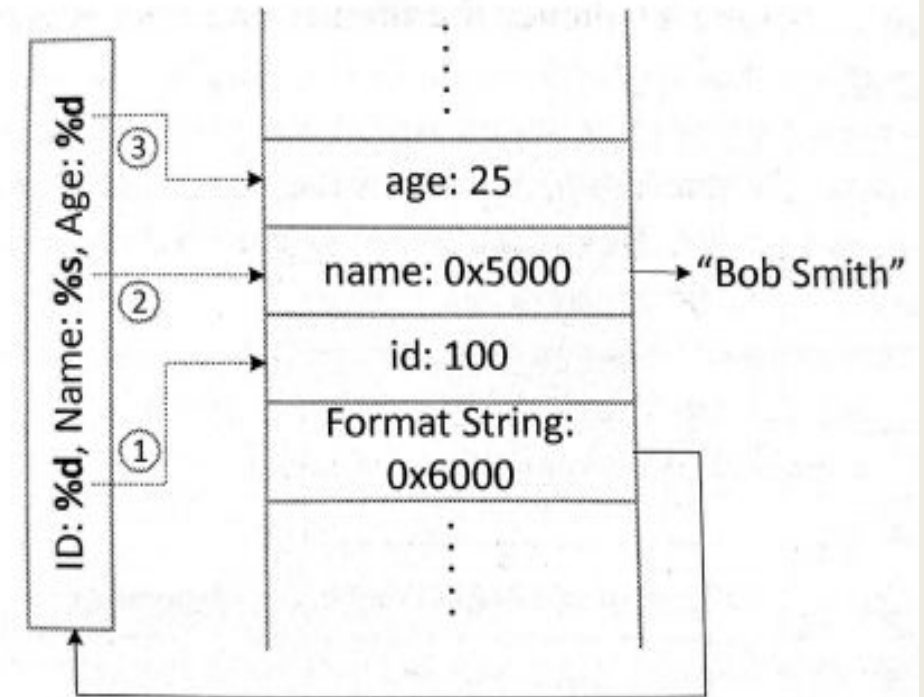
```
{
    public bool Login()
    {
        Console.WriteLine("please input the username:");
        string username = Console.ReadLine();
        Console.WriteLine("please input the password");
        string password = Console.ReadLine();
        Regex reg = new Regex(@"^\w+$");
        if (!reg.IsMatch(username) || !reg.IsMatch(password)) {
            Console.WriteLine("invalid username or password");
            return false; }
        String connetStr = "server=127.0.0.1;port=3306;user=root;password=123456; database=mydb:";
        MySqlConnection conn = new MySqlConnection(connetStr);
        try
        {
            conn.Open();//open channel
            Console.WriteLine("connect successfully");
        }
        catch (MySqlException ex)
        {
            Console.WriteLine(ex.Message);
        }
        string mysqlcmd = "select * from user where username=' " + username + "' and Password=' " + password + "'";
        MySqlCommand cmd = new MySqlCommand(mysqlcmd, conn);
        MySqlDataReader reader = cmd.ExecuteReader();
        if (!reader.Read()) Console.WriteLine("wrong username or password");
        else
        {
            Console.WriteLine("login success!");
        }
        conn.Close();
        return true;
    }
}
```

```
please input the username:
' OR 1=1;#
please input the password
123
invalid username or password
please input the username:
```

https://www.owasp.org/index.php/SQL_Injection_Prevention_Cheat_Sheet

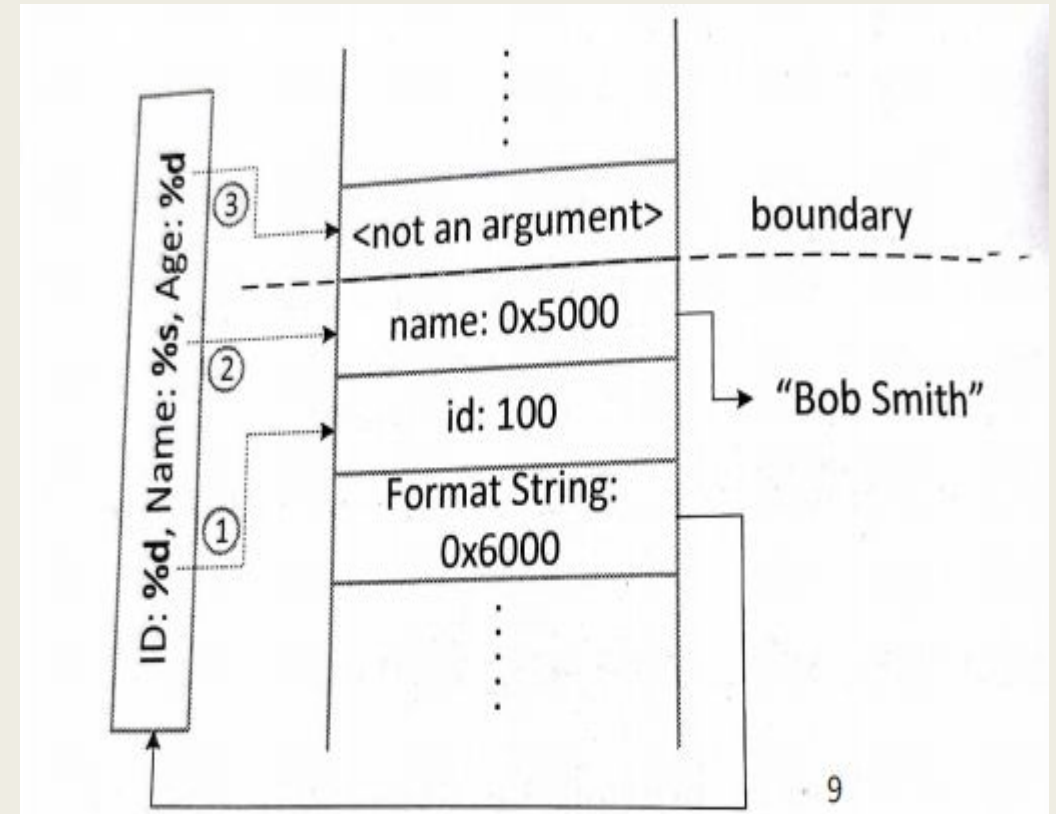
Format String

```
1  #include<stdio.h>
2
3  int main()
4  {
5      int id = 100, age =25;
6      char* name = "Smith";
7      printf("ID:%d Name:%s Age:%d\n",id, name, age);
8      return 0;
9  }
```



The Format String exploit occurs when there is a mismatch between the format string and the actual arguments.

```
1  #include<stdio.h>
2
3  int main()
4  {
5      int id = 100, age =25;
6      char* name = "Smith";
7      printf("ID:%d \nName:%s \nAge:%d\n",id, name);
8      return 0;
9  }
```



Attacks on Format String Vulnerability

- **Crashing the program.**

```
printf ("%s%s%s%s%s%s%s%s%s%s%s%s");
```

- **Viewing the stack.**

```
printf ("%08x %08x %08x %08x %08x\n");
```

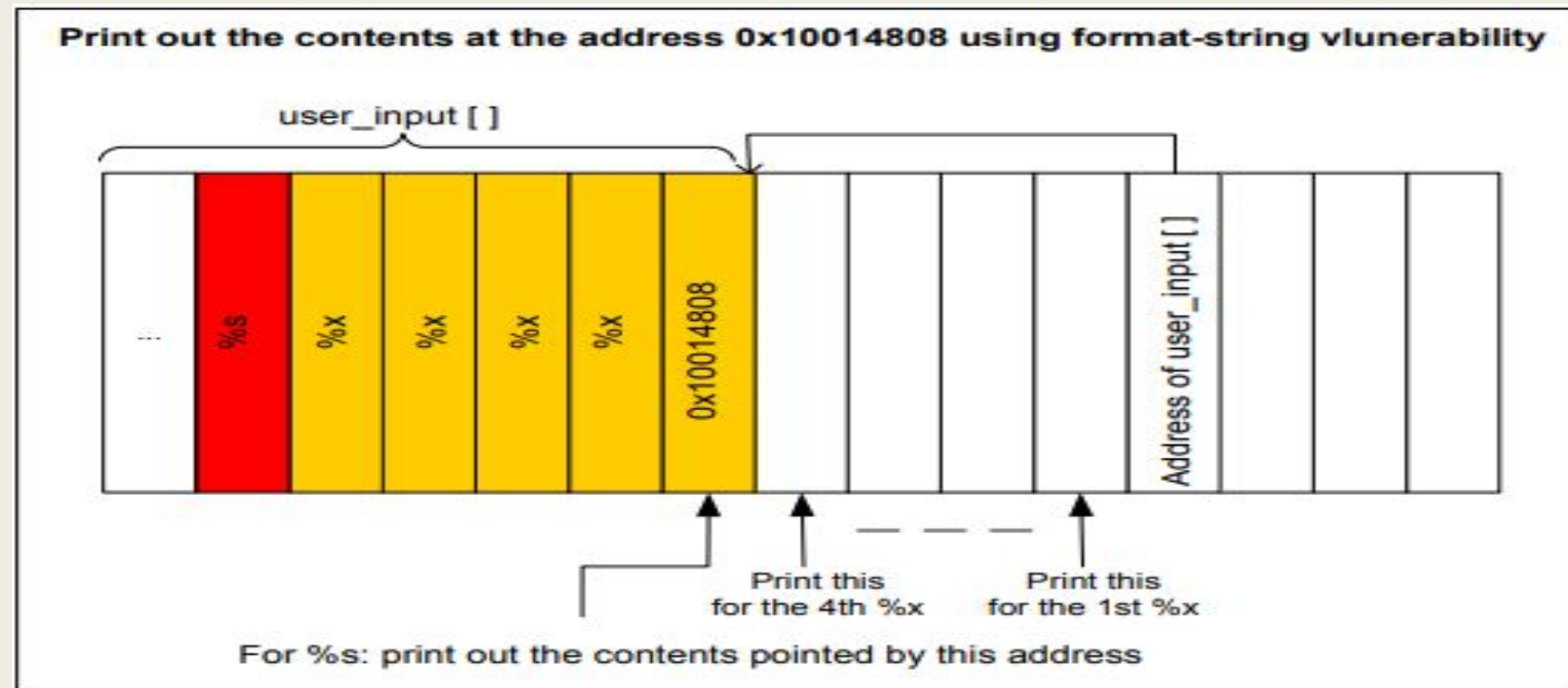
Output: 40012980 080628c4 bffffff7a4 00000005 08059c04

- Viewing memory at any location.

```
printf("\x10\x01\x48\x08 %x %x %x %x %s");
```

- Writing an integer to nearly any location in the process memory.

```
printf("12345%n", &i);
```



Counter Measures

- Developers must have a good practice to not use user inputs as an part of a format string.
- Compilers these days have built-in counter measures for detecting potential format string vulnerabilities.
- Address Randomization

System()

- A C function in `stdlib.h`
- Treats the argument as shell command and executes it.
- Rather than using `system` use `execev`

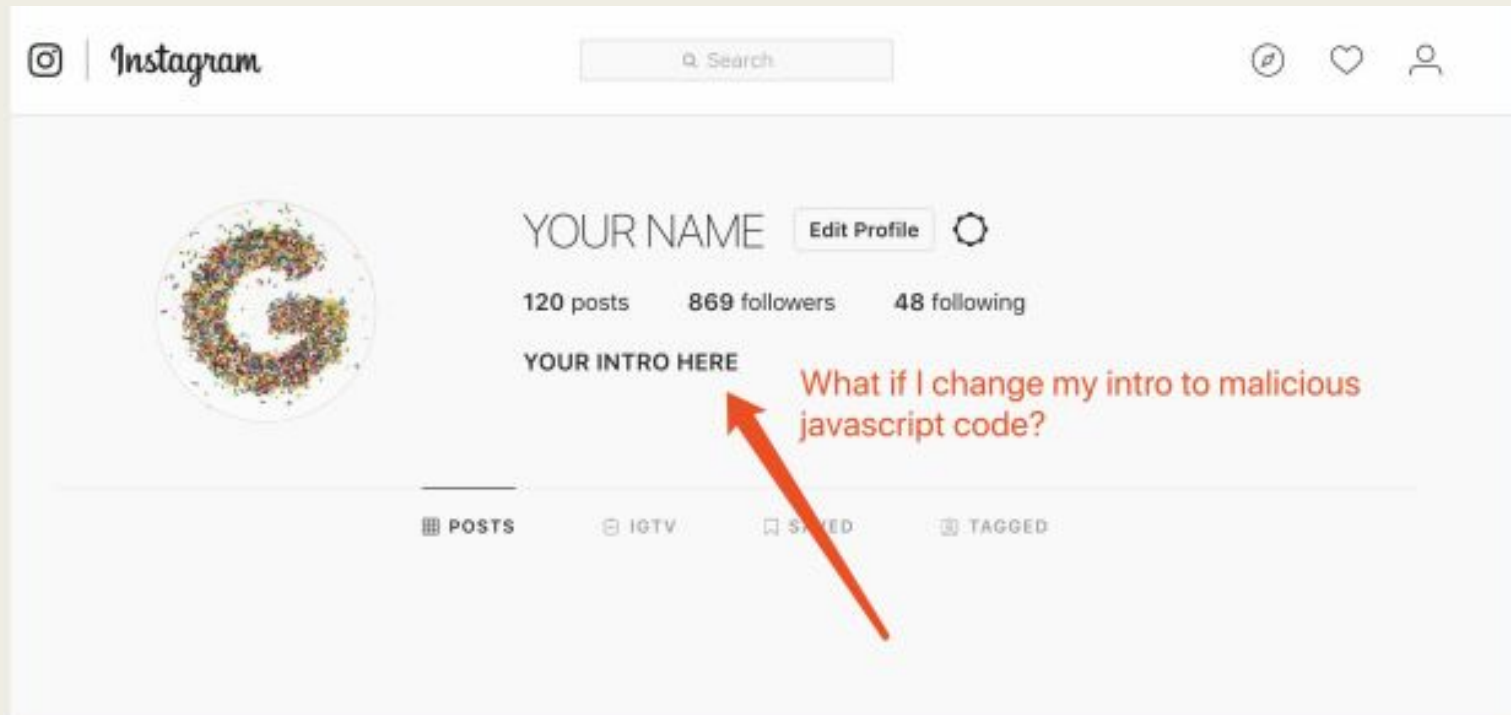
Example

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

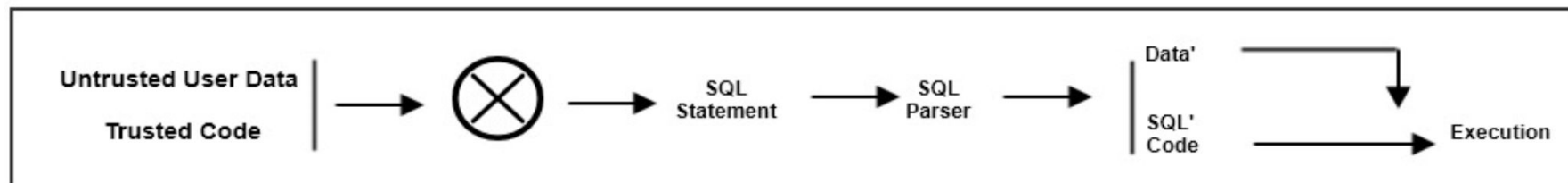
Cross Site Scripting (XSS)

- XSS is a code injection attack made possible through insecure handling of user input.
- A successful XSS attack allows an attacker to execute malicious JavaScript in a victim's browser.
- A successful XSS attack compromises the security of both the website and its users.



Similarity Among Code-Injection

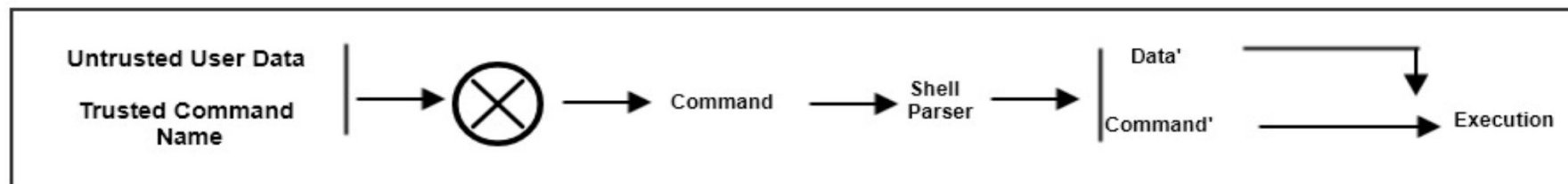
(a) SQL



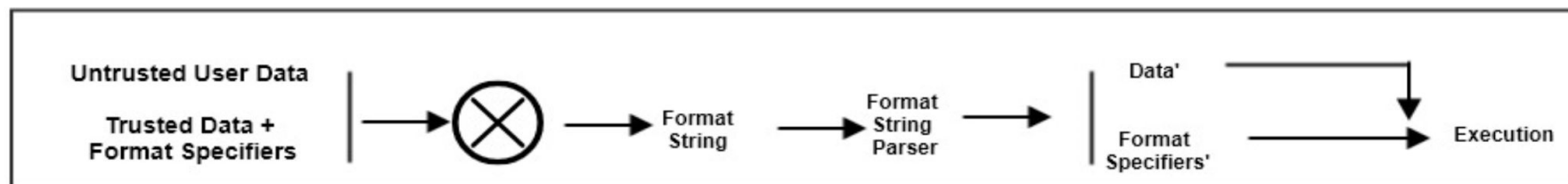
(b) JavaScript



(c) System()

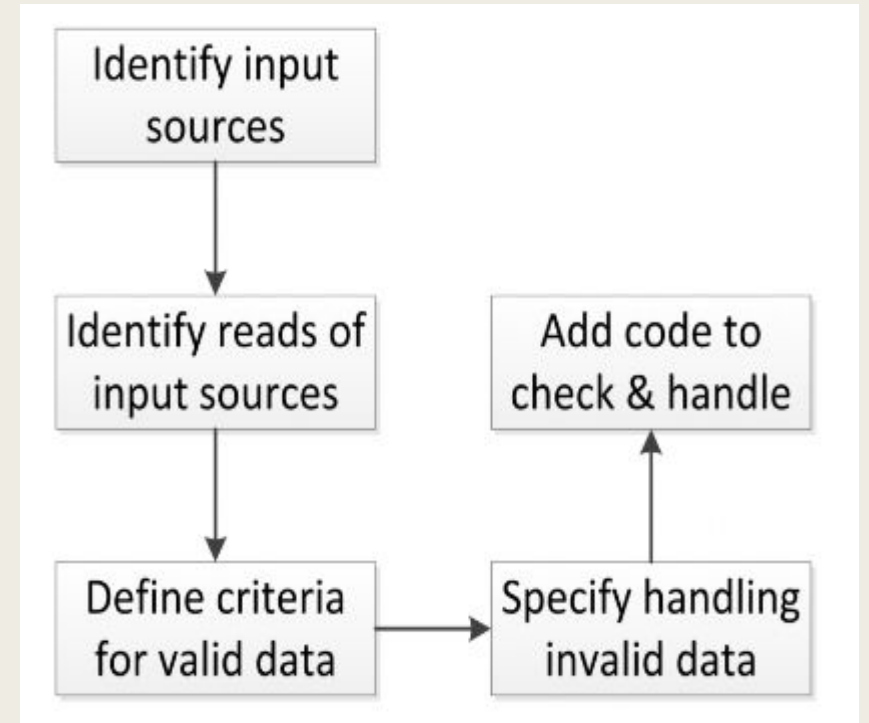


(d) Format String



Input Sanitisation pattern

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



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/>

Thank you.