APPLICATION VULNERABILITIES

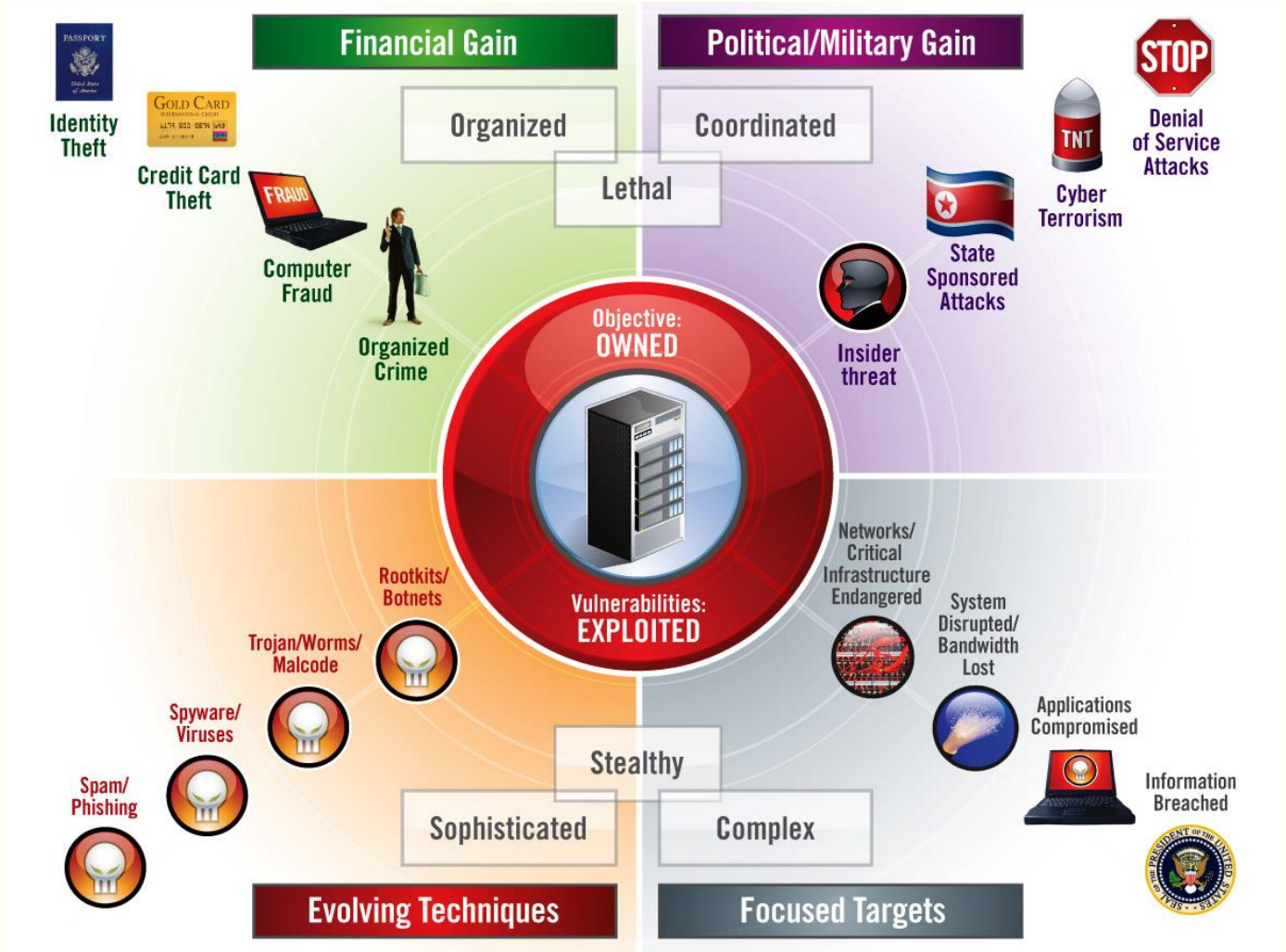
Issues, Exploits, Mitigation and Practices...

ACKNOWLEDGEMENT

I would like to express my deepest respect and gratitude to my guide **Dr. Sandeep Shukla,** for giving me this invaluable opportunity to work under him and his unwavering support, and my mentor, **Mr. Saurabh Kumar,** for teaching me everything and for trusting in me. Both of whom have inspired me to learn more and educate myself to become worthy to be known as their disciple.

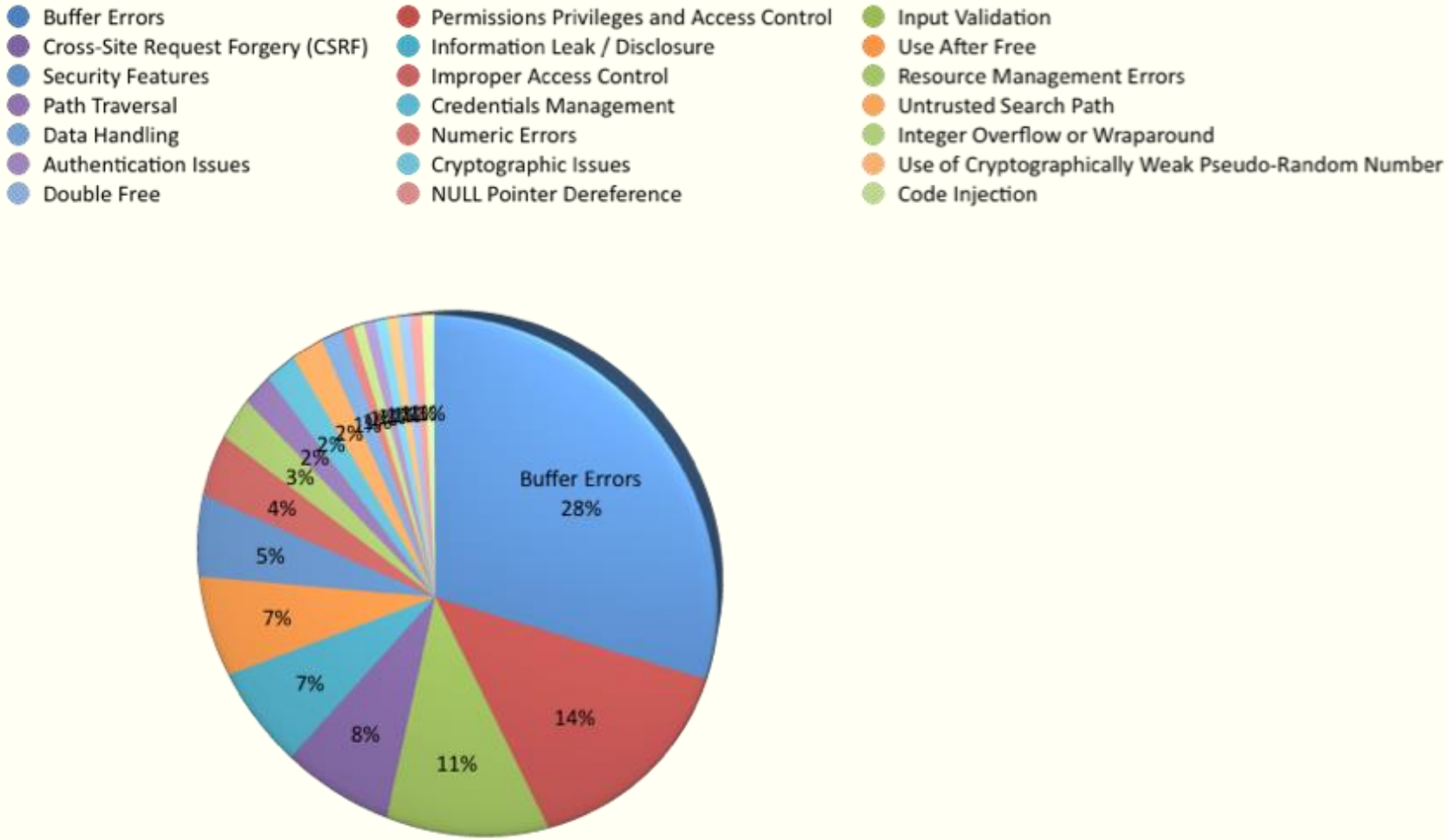
I would also like to thank my family and friends for their love and support.

Evolving threats with advancement into digital world...



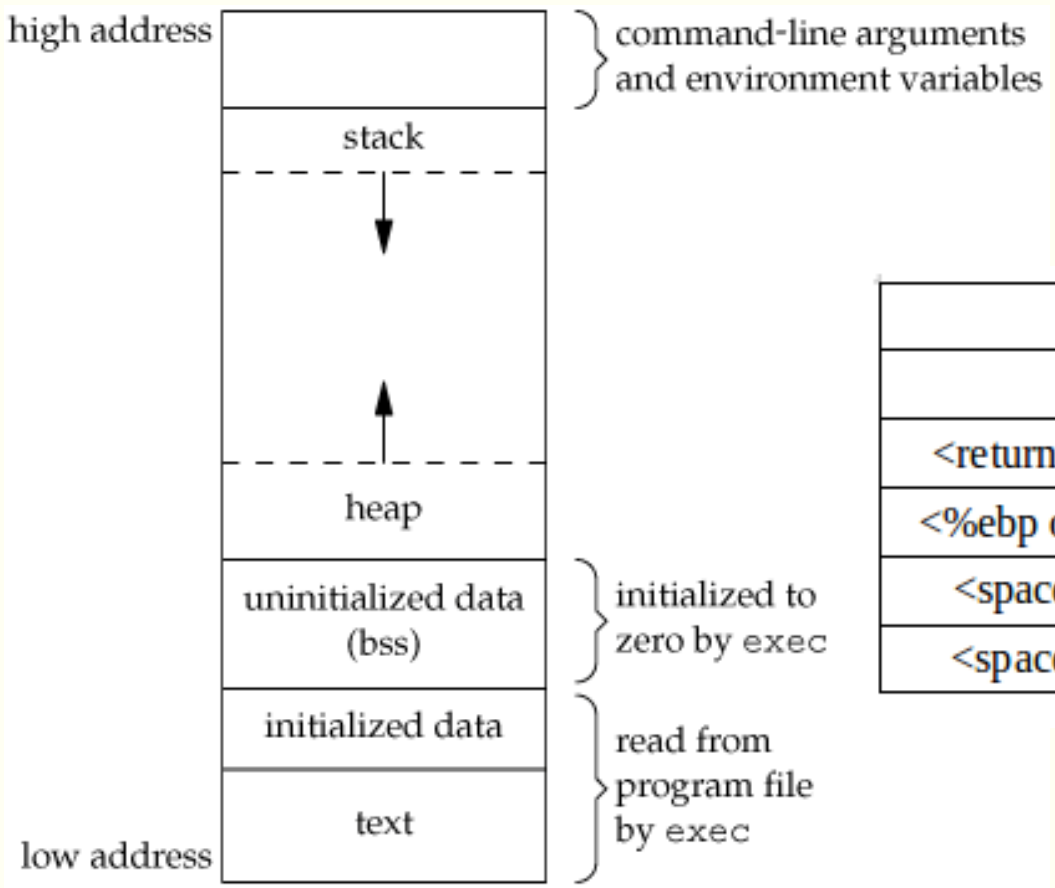
Introduction to some very common vulnerabilities...

- Buffer Overflow
- SQL injection
- String Format Vulnerability
- Integer Overflow
- Double free
- Use-after-free
- Cross-site-scripting
- HTTP header spoofing
- And many more...



Some Basics about Stack...

1. Command Line arguments and environment variables
2. Stack
3. Heap
4. BSS segment (uninitialized)
5. Data Segment (initialized)
6. Text (the program instructions)



Buffer Overflow...

- Buffer overflow is a vulnerability in low level codes of C and C++. An attacker can cause the program to crash, make data corrupt, steal some private information or run his/her own code.
- It basically means to access any buffer outside of it's alloted memory space. This happens quite frequently in the case of arrays. Now as the variables are stored together in stack/heap/etc. accessing any out of bound index can cause read/write of bytes of some other variable.

Example:

```
void main()
{
    char buffer[20];
    printf("Enter some text:\n");
    scanf("%s", buffer);
    printf("You entered: %s\n", buffer);
}
```

'buffer' will overflow in this case if the user inputs anything more than 20 characters.

Vulnerable functions: gets(), scanf(), sprintf(), and strcpy()

String Format Vulnerability...

- The Format String exploit occurs when the submitted data of an input string is evaluated as a command by the application.
- Simply put, the programmer presumes that the buffer is not controlled or is always simple string. Or maybe just unaware of the fact that it leads to a vulnerability.
- Vulnerable functions: fprintf(), printf(), sprintf(), snprintf(), vfprintf(), vprintf(), vsprintf(), vsnprintf().

Example:

```
void main()

{

   char buffer[20];

   printf("Enter some text:\n");

   scanf("%20s", buffer);

   printf(buffer);

}
```

If the input is : "%x"… then what?

The printf will look something like print("%x")

SQL Injection...

- SQL Injection (SQLi) refers to an injection attack wherein an attacker can execute malicious SQL statements that control a web application's database server.
- This leads to many severe flaws like enabling the attacker to access to the whole database, impersonating a specific user or bypass authorization.
- In worst case scenario this can even be used as initial attack vector by executing system calls.
- Mitigation: Parameterized Queries

Example:

```
$mysqli= new
mysqli($host,$dbuser,$dbpass,$dbname);

$id= $_POST{'id'};

# SQL query (dynamic)

$query = "SELECT * FROM cust WHERE id = $id";

$result = $mysqli->query($query);
```

It's fine as long as the id is genuine or just misses.
But what if the it is '<QRY> OR 1=1'

A Few Useful Tools

Demonstration of how a vulnerability could be exploited.

```
./auth
Enter the flag

%13$x%14$x%15$x%16$x%17$x%18$x%19$x%20$x%21$x%22$x
You said 67616c6661426c7b67686f334d756d4f71566d5a457333504368
4c53% which is incorrect!!
```

Hexadecimal => 67616c6661426c7b67686f334d756d4f71566d5a4573335043684c53

When converted to String => galfaBl{gho3MumOqVmZEs3PChLS

On reversing every four characters we get=> **flag{lBaohgOmuMZmVqp3sESLhC**

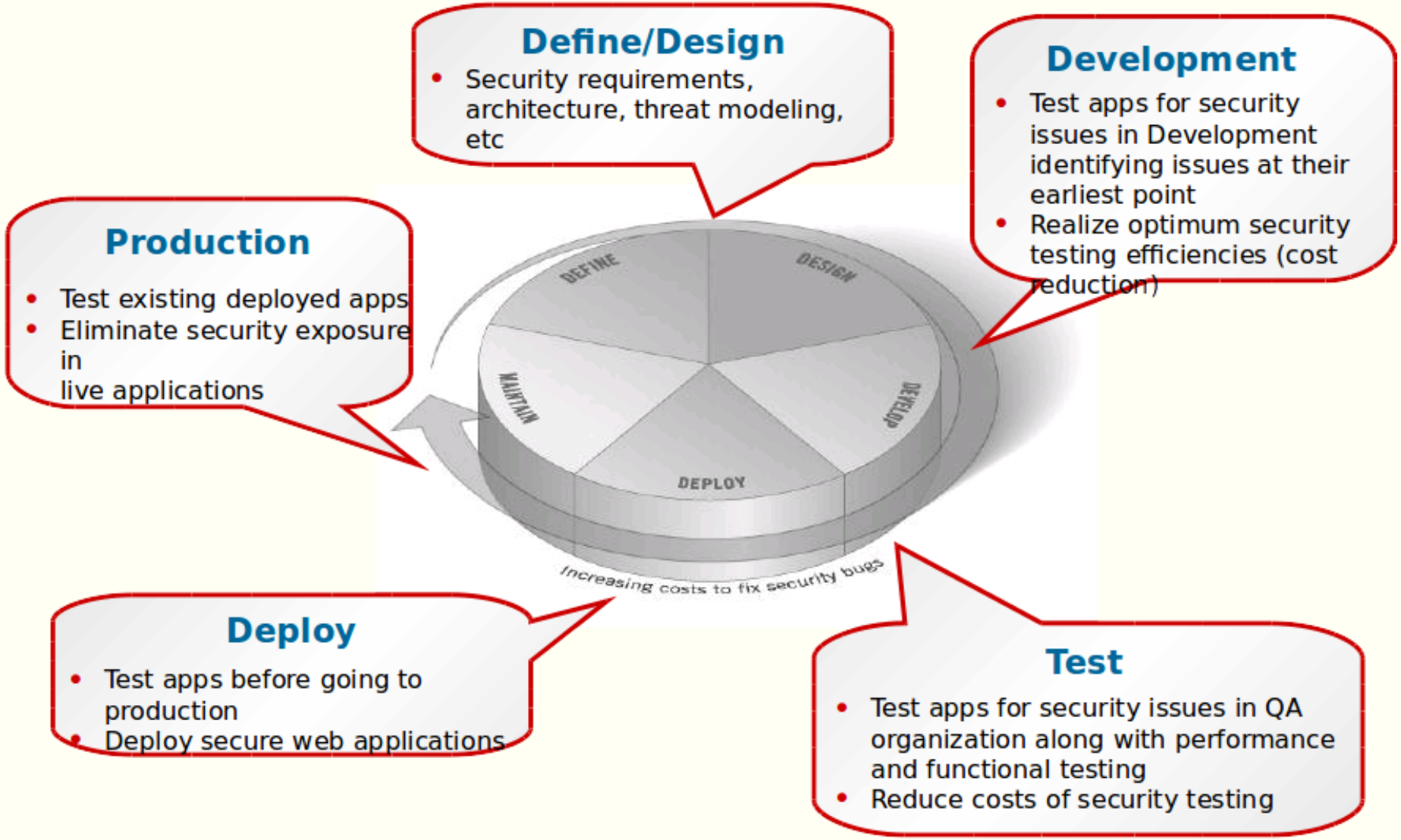
Flag Leaked due to Vulnerability

Disclaimer

Beneficial Practices...

- **Input validation and Input sanitization**. Validation is mostly just logical and program specific. Sanitization is stripping data of useless parts or tags unrelated to the required input.
- Always know and limit how much data you want your buffer to hold.
- 'Code as you go' approach is not safe. Planning before implementation can avoid many integration flaws that can otherwise be easily overseen.
- **VAPT** is a very efficient way to find flaws both logically and in implementation.
- Give access and control only as much as is necessary and no more.
- Thumb rule: Cross-check what is doubtful, and double check that is obvious. And never rely upon the users judgment.
- Go through a development process integrated with security standards and checks.

Building Security into Development Process



*Graphics from OWASP.com

It's a War and everything is fair...

Future works

Learning more about:

- Network Security
- Mobile Security
- Web Security

Understanding possibilities of attack vectors from related fields and finding their mitigation techniques.

ANY QUERIES ?

Mail me at : sagarknit7@gmail.com

THANK YOU..

Also Thanks to...

- Securitymooc- securitymooc.in
- OWASP- owasp.org
- Cyber Security News- cybersecuritynews.co.uk
- Wikipedia- wikipedia.org
- Geeks for Geeks- geeksforgeeks.org
- Wordpress Codex- codex.wordpress.org
- Hackstation.org- <indian law>
- Dhaval Kapil- dhavalkapil.com
- Image source- google.co.in