vmware

codechix

Secure Coding

By: Security Ninjas

- Bhakti Bohara — Arizona State University
- Deepanjana Gupta — Carnegie Mellon University
- Deepika Peringanji — Stony Brook University
- Shivani Sharma — North Carolina State University

Motivation

- Developers NEED to code securely
- Security mishaps have lead to huge losses
- Better be safe than sorry

ARIANE 5 FAILURE

- BACKGROUND:-
- European space agency's re-useable launch vehicle.
- Ariane-4 was a major success
- Ariane -5 was developed for the larger payloads

- LAUNCHED:-on June 4 1996
- MISSION was to delivered $500 million payloads to the orbit
- THE MAIDEN FLIGHT OF THE ARIANE 5 ENDED IN A FAILURE.
- ONLY AFTER 40 SECONDS THE FLIGHT VEERED OFF ITS PATH AND BROKE UP AND EXPLODED
- CAUSE: Unhandled floating point exception in code
- ENGINEERS FROM THE ARIANE PROJECT STARTED TO INVESTIGATE THE CAUSES OF LAUNCH FAILURE.

Source: Google Images

Brief Description of Curriculum

- The curriculum teaches about common vulnerabilities and helps in adapting to best security measures for applications

- Hands On
  - ➔ Discuss security threat
  - ➔ Ask student to attack given code snippet
  - ➔ Ask student to secure it

Prerequisites

Mac / Linux - or virtual box for terminal access

Basic working knowledge of SQL, C/C++ , Javascript or any other scripting language

Takeaways

- Coding anti-patterns in backend and frontend code leading to these vulnerabilities
- Security vulnerabilities arising from these bad practices
- Solutions and tips on how to avoid these practices

Index

Memory Management

Specific to stack smashing. Will touch upon buffer overflow and integer overflow.

Will contain hands on exercises on finding vulnerabilities in C code and fixing them.

Duration : 60 minutes

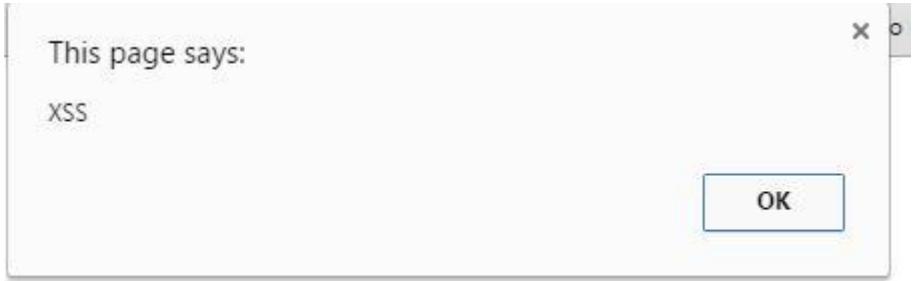
```
Enter the string to be stored : aasdjaskvlsajvblashvb;asgvjad
*** stack smashing detected ***: ./memAttack terminated
Aborted (core dumped)
```

```
Enter a string whose length is less than 128: aaaaaaaaaahhhhhhhhhhhhhhhhhhyyyyyyyyyyyyyyyyyyyyyyyyffffffffffffffffffffffffffffffffffffffffffffffffrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnn
nnnnnnnnnnnnnnnn

Enter a string whose length is less than 128: aaaaaaaaaahhhhhhhhhhhhhhhhhhyyyyyyyyyyyyyyyyyyyyyyyyffffffffffffffffffffffffffffffffffffffffffffffffrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnn
nnnnnnnnnnnnnnnn

Enter the lengths of the 2 strings whose sum should be less than 256 (in hex): 0xfffffffa 0x106
Segmentation fault (core dumped)
```

Input Validation

The front end should have a centralized input validation routine for untrusted data sources which rejects the input if any validation check fails.

- Specifying character set and canonicalize input before validation
- Check for hazardous characters, dot-dot-slash, null bytes
- Duration : 15 minutes



Database Security

SQL scripts interfacing with the database should be protected from SQL injection attacks.

Hands-on exercise will include a SQL script with queries susceptible to SQL injection and stacked queries which must be identified and fixed (convert simple query to parameterized query) to make it safe.

Duration : 30 minutes

```
Enter your username:ccAdmin
Enter your password:pwdAdmin
Hello 101 ccAdmin
Enter your account type (SAVING/CHECKING):SAVING
Your balance is 20000
```

```
Enter your username:ccAdmin' UNION SELECT uid,username FROM users --
Enter your password:any password
Hello 101 ccAdmin 201 ccMod 301 ccStudent
Enter your account type (SAVING/CHECKING)::SAVING' UNION SELECT balance FROM balances WHERE uid = 201 OR uid = 301 --
Your balance is 4000 5000
```

Cryptographic Practices

Two hands-on problems

- Identifying the correct order/way to use the crypto libraries to get intended effect
- Emphasize on random number generation/ key size
- Duration : 15 minutes

```
def encrypt(plain_text, key):
    iv = "DeadBeef"
    des = DES.new(key, DES.MODE_CFB, iv)
    cipher_text = des.encrypt(plain_text)
    return cipher_text

def decrypt(cipher_text, key):
    iv = "DeadBeef"
    des = DES.new(key, DES.MODE_CFB, iv)
    decrypted_text = des.decrypt(cipher_text)
    return decrypted_text

if __name__ == "__main__":
    plain_text = "This is a secret message"
    key = "Good key"
    cipher_text = encrypt(plain_text, key)
    decrypted_text = decrypt(cipher_text, key)
```

```
def encrypt(plain_text, key):
    iv = Random.get_random_bytes(8)
    des = DES.new(key, DES.MODE_CFB, iv)
    cipher_text = des.encrypt(plain_text)
    return cipher_text

def decrypt(cipher_text, key):
    iv = Random.get_random_bytes(8)
    des = DES.new(key, DES.MODE_CFB, iv)
    decrypted_text = des.decrypt(cipher_text)
    return decrypted_text

if __name__ == "__main__":
    plain_text = "This is a secret message"
    key = "Good key"
    cipher_text = encrypt(plain_text, key)
    decrypted_text = decrypt(cipher_text, key)
```

Error Handling

Error handling is often neglected! Handling errors gracefully and leaving the system consistent is important

Never reveal sensitive information about the underlying application, system, server etc.,

Hands-on exercises - Releasing memory, cleaning up practices during erroneous exit
Duration : 10 minutes

```
HEAP SUMMARY:
    in use at exit: 40 bytes in 1 blocks
  total heap usage: 1 allocs, 0 frees, 40 bytes allocated

Searching for pointers to 1 not-freed blocks
Checked 77,256 bytes

40 bytes in 1 blocks are definitely lost in loss record 1 of 1
   at 0x4C2AB80: malloc (in /usr/lib/valgrind/vgpreload_memcheck-amd64-linux.so)
   by 0x4005AB: main (in /home/deepika/codehouse/secureCoding/error_handling/error)

LEAK SUMMARY:
   definitely lost: 40 bytes in 1 blocks
   indirectly lost: 0 bytes in 0 blocks
     possibly lost: 0 bytes in 0 blocks
   still reachable: 0 bytes in 0 blocks
        suppressed: 0 bytes in 0 blocks
```

```
filp = fopen(argv[1], "ab+");
if (!filp) {
    return -1;
}

if (b != 0) {
    fprintf (filp,"%d divided by %d is %d",a,b,a/b);
    return 0;
}
else {
    printf ("Divide by zero. Exiting");
    return -1;
}
```

Session Management

Managing Cookies, Sessions.

Hands-on:Simple script to add cookie via javascript.

Tasks :

- Restrict cookie through Path, Domain and Max Age
- Use URL encoding to eliminate whitespaces
- Use HTTPOnly.
- Set session id as cookie and send it from server
- Invalidate the session on logout

Duration : 60 minutes

```
<script>
    function setcookie(){

    var d = new Date();
    d.setTime(d.getTime() + (24*60*60*1000));
    var expires = "expires=" + d.toGMTString();
    document.cookie = document.getElementById('form').name.
        value + "=" + document.getElementById('form').pwd.
        value +"; " + expires;
    alert(document.cookie);
}

    function getcookie(){
        var x = document.cookie;
        alert(x);
    }
</script>
```

Other Topics …

Output Encoding

Password Management and Authentication

Access Control

Data Protection

Duration : 40 minutes

Future Scope

- More hands on exercises
- References to external exercises
- Vulnerabilities in other stacks e.g MEAN stack

Demo Time!

References

1. https://www.owasp.org/images/0/08/OWASP_SCP_Quick_Reference_Guide_v2.pdf
2. http://www.w3schools.com/
3. http://www.opinionatedgeek.com/Errors/Unknown.aspx?aspxerrorpath=/dotnet/tools/htmlencode/encode.aspx

Thank You!