Miscellaneous Topics

Insu Yun

So far we learned

- Command injection
- PATH injection
- (Stack) buffer overflow
- Format string bug
- A very small part of vulnerabilities

Common Weakness Enumeration (CWE)

- Software: **https://cwe.mitre.org/data/definitions/699.html**
    - 458 CWEs!

So far we learned (CWE)

- Command injection
    - CWE-78: Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')
- PATH injection
    - CWE-426: Untrusted Search Path
- (Stack) buffer overflow
    - CWE-121: Stack-based buffer overflow
- Format string bug
    - CWE-134: Use of Externally-Controlled Format String

CWE-457: Use of Uninitialized Variable

- The code uses a variable that has not been initialized, leading to unpredictable or unintended results.

CWE-457: Use of Uninitialized Variable

```
char *test_string;
if (i != err_val)
{
    test_string = "Hello World!";
}
printf("%s", test_string);
```

CWE-367: Time-of-check Time-of-use (TOCTOU) Race Condition

- The software checks the state of a resource before using that resource, but the resource's state can change between the check and the use in a way that invalidates the results of the check. This can cause the software to perform invalid actions when the resource is in an unexpected state.

CWE-363: Race Condition Enabling Link Following

- The software checks the status of a file or directory before accessing it, which produces a race condition in which the file can be replaced with a link before the access is performed, causing the software to access the wrong file.

CWE-363: Race Condition Enabling Link Following

```
// assume that it is running by $admin
function readFile($filename){
    $user = getCurrentUser();

    //resolve file if its a symbolic link
    if(is_link($filename)){
        $filename = readlink($filename);
    }

    if(fileowner($filename) == $user){
        echo file_get_contents($realFile);
        return;
    }
    else{
        echo 'Access denied';
        return false;
    }
}
```

CWE-363: Race Condition Enabling Link Following

- Exploitation
    - Process1
        - Create a empty file of $user
        - Delete it
        - Make a symbolic link for $admin
    - Process2
        - Keep running the program

CWE-502: Deserialization of Untrusted Data

- The application deserializes untrusted data without sufficiently verifying that the resulting data will be valid.

CWE-502: Deserialization of Untrusted Data

```
class ExampleProtocol(protocol.Protocol):
    def dataReceived(self, data):
        # Code that would be here would parse the incoming data
        # After receiving headers, call confirmAuth() to
authenticate

    def confirmAuth(self, headers):
        try:
            token =
cPickle.loads(base64.b64decode(headers['AuthToken']))
            if not check_hmac(token['signature'], token['data'],
getSecretKey()):
                raise AuthFail
            self.secure_data = token['data']
        except:
            raise AuthFail
```

CWE-502: Deserialization of Untrusted Data

- Python allows to define how to unpickle objects using __reduce__
  - i.e., arbitrary code execution
- Exploitation

```
import cPickle
import subprocess
import base64

class RunBinSh(object):
  def __reduce__(self):
    return (subprocess.Popen, (('/bin/sh',),))

print base64.b64encode(cPickle.dumps(RunBinSh()))
```

- Similar things also happen in Java, Ruby, others
  - How to prevent?

CWE-195: Signed to Unsigned Conversion Error

- The software uses a signed primitive and performs a cast to an unsigned primitive, which can produce an unexpected value if the value of the signed primitive can not be represented using an unsigned primitive.

CWE-195: Signed to Unsigned Conversion Error

```
DataPacket *packet;
int numHeaders;
PacketHeader *headers;

sock=AcceptSocketConnection();
ReadPacket(packet, sock);
numHeaders = packet->headers;

if (numHeaders > 100) {
    ExitError("too many headers!");
}
headers = malloc(numHeaders * sizeof(PacketHeader);
ParsePacketHeaders(packet, headers);
```

CWE-758: Reliance on Undefined, Unspecified, or Implementation-Defined Behavior

- The software uses an API function, data structure, or other entity in a way that relies on properties that are not always guaranteed to hold for that entity.

CWE-758: Reliance on Undefined, Unspecified, or Implementation-Defined Behavior

```
char *buf = ...;
char *buf_end = ...;
unsigned int len = ...;
if (buf + len >= buf_end)
    return; /* len too large */
if (buf + len < buf)
    return; /* overflow, buf+len wrapped around */
/* write to buf[0..len-1] */
```

CWE-758: Reliance on Undefined, Unspecified, or Implementation-Defined Behavior

```
class A {
    public:
        void read(int x) {
            int *addr = internalRead(x);
            printf("0x%x₩n", *addr);
        }

    private:
        int* internalRead(int x) {
            if (x < 0 || x >= 100){ return nullptr;}
            return array+x;
        }
        int flag = 0xdeadbeef;
        int array[100] = {0};
    };

int main() {
    A a;
    a.read(-1);
    return 1;
}
```

CWE-758: Reliance on Undefined, Unspecified, or Implementation-Defined Behavior

```
$ clang++ -o demo demo.cpp
$ ./demo
Segmentation fault   (core   dumped)
```

```
$ clang++ -O3 -o demo demo.cpp
$ ./demo
0xdeadbeef
```

CWE-252: Unchecked Return Value

- The software does not check the return value from a method or function, which can prevent it from detecting unexpected states and conditions.

CWE-252: Unchecked Return Value

```
char buf[10], cp_buf[10];
fgets(buf, 10, stdin);
strcpy(cp_buf, buf);
```

CWE-843: Access of Resource Using Incompatible Type ('Type Confusion')

- The program allocates or initializes a resource such as a pointer, object, or variable using one type, but it later accesses that resource using a type that is incompatible with the original type.

CWE-843: Access of Resource Using Incompatible Type ('Type Confusion')

```
class Parent {
    int p_data;
    virtual void print (void) {};
}

class Child : pulbic Parent {
    int c_data;
    void print(void) override {};
}

Parent *Pptr = new Parent;
Child *Cptr = static_cast<Child>(Pptr);
Cptr->c_data = 0x12345678;
```

CWE-840: Business Logic Errors

- Business logic vulnerabilities are flaws in the design and implementation of an application that allow an attacker to elicit unintended behavior.

CWE-840: Business Logic Errors

```
int pid = client->pid;
if (security_check(action, pid)) {
 perform_action(client);
}
```

- This is a real vulnerability in MacOS :)

Heap-related CWEs

- CWE-122: Heap-based Buffer Overflow
- CWE-416: Use After Free
- CWE-415: Double Free
- CWE-763: Release of Invalid Pointer or Reference

Other vulnerabilities

- Web
- Cryptography
- Smart contract
- Machine learning

Reference

- https://i.blackhat.com/eu-20/Wednesday/eu-20-Wu-Finding-Bugs-Compiler-Knows-But-Does-Not-Tell-You-Dissecting-Undefined-Behavior-Optimizations-In-LLVM.pdf
- https://cwe.mitre.org/
- https://portswigger.net/web-security/logic-flaws
- https://saelo.github.io/presentations/warcon18_dont_trust_the_pid.pd