CH 9

Sunday, January 6, 2019 1:36 PM

Vulnerability Classes:

RCE	**Remote Code Execution:** Running arbitrary code w/app that implements protocol ▪ Hijacking logic of app/influencing cli subprocesses created in normal op **<u>Allows attacker to compromise sys of app executing</u>** ▪ Access to anything app can access: Maybe hosting network compromised too
DoS	**Denial of Service:** Causes crash/unresponsiveness • Denies usr access to app/service **<u>Categorized as:</u>** **1. Persistent:** Perm prevents usr from accessing service **2. Non-persistent:** As long as attacker attacks
Info Disclosure	Exists if there's a way to get an app to provide info it wasn't designed to ▪ Contents of mem/fs paths/auth creds
Auth Bypass	**Authentication Bypass:** Way to auth to app w/out providing all creds ▪ Incorrectly checking for a password/brute force/SQLi ▪ Allows to auth as a specific usr
Autho Bypass	**Authorization Bypass:** Can gain rights/access to resources not priv to access ▪ Allows attacker to access resource from incorrect auth

Memory Corruption Vulns: Mem-Safe vs. Mem-Unsafe Languages:

Memory safe languages:

- Java/C#/Python/Ruby don't normally req dev to deal w/low-level mem mgmt
- Can provide libs/constructs to perform unsafe ops: C# unsafe keyword
- Bounds checking for in-mem buffer access to prevent out-of-bounds reads/writes
- Not completely immune to mem corruption
- More likely to be a bug in the runtime

Memory-Unsafe languages:

- C/C++ perform little mem access verification/lack robust mechs for auto managing mem
- Many types of mem corruption can occur
- How exploitable? Depends on OS/compiler used/how app structured

Buffer Overflows: When app tries to put more data into region of mem than designed to hold

Can occur for 2 reasons:

1. **Fixed-length:** Incorrect input buffer fitting into allocated buffer
2. **Variable-length**: Size of allocated buffer incorrectly calc

Fixed-Length	**App incorrectly checks length of external data value** ▪ Relative to fixed-length buffer in mem ▪ Might be in stack/on a heap/exist as global buffer defined at compile time ▪ Mem length determined prior to knowledge of actual data length

```
def read_string() {
    byte str[32];
    int i = 0;

    do {
    str[i] = read_byte();
    i = i + 1;
    }
while(str[i-1] != 0);
printf("Read String: %s\n", str);
}
```

1. Allocates buffer where it will store string on stack: 32 bytes
   ▫ Loop reads byte from network/stores into incrementing index in buffer
2. Loop exits when last byte read from network eq to 0: Indicates value sent
3. <u>Mistake:</u> Loop doesn't verify length/reads as much data as avail from network

Unsafe String Functions: C doesn't define str type: Uses ptrs to list of char types

- End of str indicated by 0-value char

strcpy: Function copies strings: Takes only 2 args

1. **Ptr to source string**
2. **Ptr to destination mem buffer to store copy**
    - Nothing indicates length destination mem buffer
    - Recent C compilers added more sec vers of these
        - **strcpy_s**: adds a destination length arg

off-by-one error: Shift in index positions (screwing up arrays)

Var-Length

Possible for app to allocate buffer of correct size for data being stored
- If incorrectly calcs buffer size var-length b0f could happen
- Issue if calc induces undefined behavior by lang/platform

```
def read_uint32_array() {
    uint32 len;
    uint32[] buf;

    // Read number of words from network
    len = read_uint32();

    // Allocate mem buffer
    buf = malloc(len* sizeof(uint32));

    // Read values
    for(uint32 i = 0; i < len; ++i) {
        buf[i] = read_uint32();
    }
        printf("Read in %d uint32 values\n", len);
}
```

1. Buffer dynamically allocated at runtime to contain total size of input
    - 32-bit int: Uses to determine num of next 32-bit value
    - Determines total allocation size/allocates buffer corresponding size
2. Loop reads each value from protocol into allocated buffer

Int Overflows

modulo arithmetic: At processor instruction lvl: int math ops
- Allows values to wrap if they go above certain value: **modulus**
- Processor uses modulo if supports certain native int such as 32/64 bits
- Result of any op must be w/in ranged allowed for fixed-size int values

<u>Example</u>: 8 bit int: Can only take values bet 0-255
- Multiplying a value by 4 on 32-bit ints like 65 x 4 = 0x104 or 260
- Processor drops the overflowed bit

Out-of-bounds Buffer Indexing

Sometimes vuln occurs bc size of buffer incorrect
- Instead of incorrectly specifying size of value
- Some control over position in buffer
- If incorrect bounds checking on access position: Vuln exists

Selective mem corruption: Can be exploited to write data outside buffer:
- Exploited reading value outside buffer: Info disclosure/RCE

<u>Doesn't just have to involve writing</u>:
- Works when values read from buffer w/incorrect index
- If index used to read value/ret to client: Simple info disclosure
- Vuln could occur if index used to ID functions w/in app to run

Data Expansion Attack

Modern high-speed networks compress data to reduce num of raw octets
- At some point data must be decompressed
- If compression done by app: Data expansion possible

Dynamic Memory Allocation Failures: System memory finite: When mem pool runs dry:
- Dynamic mem allocation pool handles situations where app needs more
- Results in error value being ret from allocation functions (NULL ptr)

<u>Possible vulns may arise from not correctly handling dynamic mem allocation failure:</u> DoS/app crash

Default/Hardcoded Creds

Default creds commonly added as part of installation process
- Usually default usrname/passwd associated w
- Problem if they aren't changed

User Enumeration

Most usr-facing auth use usernames to control access to resources
- Typically name combined with token
- User ID doesn't have to be a secret: Often publicly avail emails
- More likely you could brute force passwds by valid accts

Incorrect Resource Access: Protocols provide access to resources (HTTP)/file-sharing/ID for resource

- Identifier could be file path/unique: App must resolve identifier in order to access target resource
- Many vulns can affect such protocols when processing resource identifiers

Canonicalization	If resource identifier hierarchical list of resources/dirs: Referred to as path ▪ OS defines way to specify relative path info using .. (parent dir) ▪ Before a file can be accessed: OS must find it using this path info **Naïve remote file protocol:** Pass directly to OS ▪ Could take path supplied by remote user: Concatenate it w/base dir
Verbose Errors	When app tries to retrieve resource/isn't found: Returns error info ▪ Simple as error code w/full description of what doesn't exist ▪ Shouldn't disclose any more info than required
Mem Exhaustion	Resources of sys on which app runs finite: Exhausting them ▪ Allocating mem dynamically based on absolute value transmitted in protocol
CPU Exhaustion	**CPU's can only do certain # of tasks a time** **<u>2 main ways:</u>** **1. Algorithmic complexity** **2. Identifying external controllable params to cryptographic systems** **<u>Algorithmic Complexity:</u>** ▪ All algs have associated computational cost ▪ How much work performed for particular input to get desired output ▪ More work alg needs? More time from processor ▪ Some algs become expansive as num of input params increase **<u>Example: Bubble Sort:</u>** Inspects each value pair in a buffer/swaps them ▪ If left value of pair greater than right ▪ Bubbling higher values at end of buffer until buffer is sorted ▪ Amt of work alg req proportional to num of elements in buffer to sort **<u>Best case:</u>** Single pass through buffer req N iterations: All elements already sorted **<u>Worse case:</u>** Buffer sorted In reverse: Alg needs to repeat sort process N^2 times ▪ If attacker could specify a large num of reverse-sorted values ▫ Computational cost becomes significant ▫ Could consume 100% of CPU's processing time: DoS **Configurable Crypto:** ▪ Primitives processing: Hashing create significant amt of workload ▫ Authentication creds ▪ Passwds should always be hashed using digest alg before stored ▪ Converts pass into hash value: Impossible to reverse ▪ Someone could still guess pass/generate hash ▪ If guessed passwd matches when hashed: Original pass discovered <u>To mitigate:</u> Typical to run hashing op multiple times • Increase computational cost for app ▫ DoS: Long time bc of size/alg # of iterations specified externally

Format String Vulnerabilities: Most lang have mech to convert arbitrary data into str

- Common to define some fmting mech to specify output
- Attacker can supply str value to app used directly as fmt str
- **printf**/variants such as **sprintf** which print to str
  - Takes fmt str as first arg/list of values to fmt
- Specifies position/type of data using a %? syntax (? replaced by alphanumeric char)
  - Fmt specifier can also include fmt info: num of dec places in num
  - Attacker who can directly control fmt str could corrupt mem/disclose info

List of Commonly Exploitable printf Fmt Specifiers

Fmt Specifier	Description	Potential Vulns
%d, %p, %u, %x	Prints ints	Info disclosure from stack if ret to an attacker
%s	Prints 0 terminated str	Info disclosure from stack if ret to an attacker Cause invalid mem accesses to occur: DoS
%n	Writes current # of printed chars	Selective mem corruption/app crash

to ptr specified in args

Command Injection: Most OS: Set of utilities for various tasks

- Some decide easiest way to exe task is to exe an external app/os util
- Some data from network client inserted into cli to perform desired op

SQLi: Simplest app may need to persistently store/retrieve data: Relational DB

- SQL: Structured Query Language: Defines what data tables to read/how to filter them
- Can easily result in vuln like cmd injection:
- Instead of inserting untrusted data into CLI w/out appropriately escaping
- Attacker inserts data into SQL query: Executed on DB: Can mod op of query

Txt-Encoding Char Replacement: Some conversions bet txt encodings can't be round-tripped:

- Converting from 1 encoding to another loses impt info
  - If reverse applied original txt can't be restored
- Converting from wide char set (Unicode) to narrow (ASCII)
  - Impossible to encode entire Unicode char set in 7 bits

<u>Conversions handle this 2 ways:</u>

1. **Replaces char that can't be represented w/placeholder (?)**
   - Problem if data value refers to something where ? is delimiter/special char
2. **Best-fit mapping**: Used for chars for similar char in new encoding
   - Problem when converted txt processed by app

Implementation issue: App 1st verifies sec condition using 1 encoded form of a str

- Then uses other encoded form of str for specific action:
  - Reading resource/executing cmd