Root Causes for Vulnerability

Overview

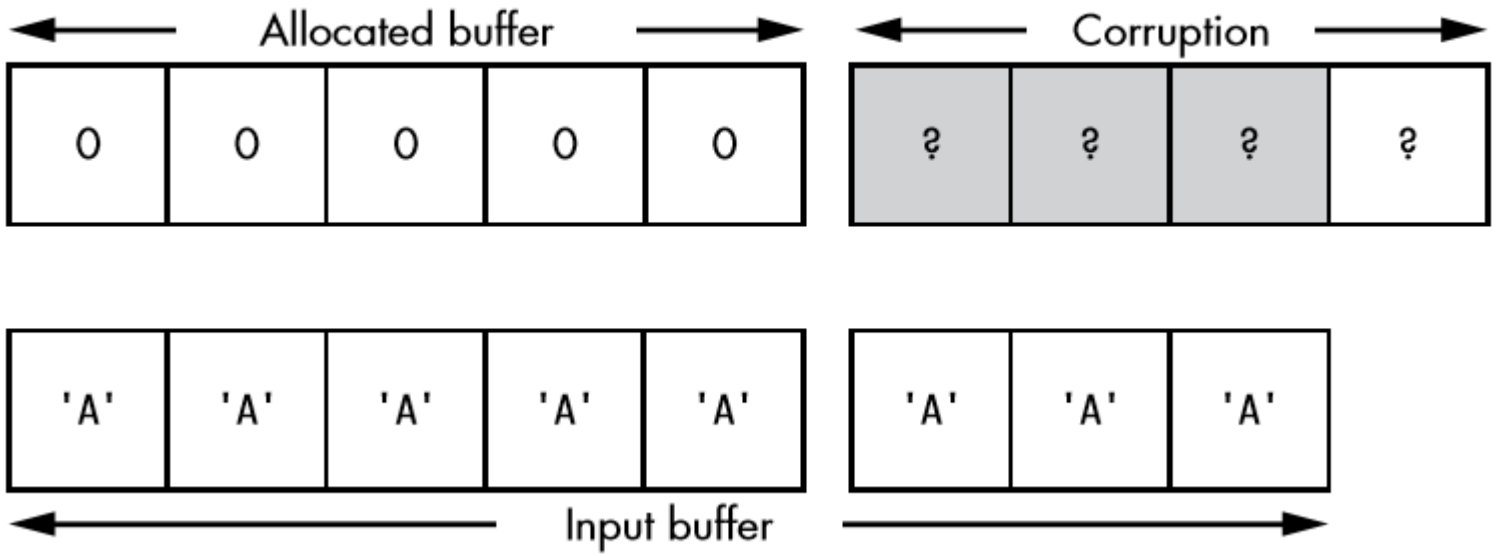
- Vulnerability classes
- Memory corruption vulnerabilities
- Default or hardcoded credentials
- User enumeration
- Incorrect resource access
- Memory exhaustion attacks
- Storage exhaustion attacks
- CPU exhaustion attacks
- Format string vulnerabilities
- Command Injection
- SQL Injection
- Text-encoding character replacement

Vulnerability classes

- **Remote Code Execution**
- **Denial-of-Service**
- **Information Disclosure**
- **Authentication Bypass**
- **Authorization Bypass**
  - Don’t confuse authorization bypass with authentication bypass vulnerabilities.

Memory corruption vulnerabilities

- **Memory-Safe vs. Memory-Unsafe Programming Languages**
- **Memory Buffer Overflows**



Memory Buffer Overflows

- **Fixed-Length Buffer Overflows**

```
...
char buf[BUFSIZE];
gets(buf);
...
```

```
...
char buf[BUFSIZE];
cin >> (buf);
...
```

```
char *lccopy(const char *str) {
    char buf[BUFSIZE];
    char *p;

    strcpy(buf, str);
    for (p = buf; *p; p++) {
        if (isupper(*p)) {
            *p = tolower(*p);
        }
    }
    return strdup(buf);
}
```

Memory Buffer Overflows

- **Variable-Length Buffer Overflows**

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

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

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

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

Integer Overflows

- **Module Arithmetic**
  - 8 bit integer example
    - 65 * 4 = 260

Integer Overflows

- **Out-of-Bounds Buffer Indexing**

```
❶ byte app_flags[32];

  def update_flag_value()
  {
❷   byte index = read_byte();
    byte value = read_byte();

    printf("Writing %d to index %d\n", value, index);

❸   app_flags[index] = value;
  }
```

- **Dynamic Memory Allocation Failures**

Default or hardcoded credentials

- **Default Credentials**
- **Hardcoded Credentials**

```
def process_authentication()
{
❶ string username = read_string();
  string password = read_string();

  // Check for debug user, don't forget to remove this before release
❷ if(username == "debug")
  {
    return true;
  }
  else
  {
 ❸ return check_user_password(username, password);
  }
}
```

User enumeration

```
def process_authentication()
{
  string username = read_string();
  string password = read_string();

❶ if(user_exists(username) == false)
  {
 ❷ write_error("User " + username " doesn't exist");
  }
  else
  {
 ❸ if(check_user_password(username, password))
    {
      write_success("User OK");
    }
    else
    {
   ❹ write_error("User " + username " password incorrect");
    }
  }
}
```

Incorrect Resource Access

- **Canonicalization**

```
def send_file_to_client()
{
❶   string name = read_string();
    // Concatenate name from client with base path
❷   string fullPath = "/files" + name;

❸   int fd = open(fullPath, READONLY);

    // Read file to memory
❹   byte data[] read_to_end(fd);

    // Send to client
❺   write_bytes(data, len(data));
}
```

Incorrect Resource Access

- **Verbose Errors**

```
def send_file_to_client_with_error()
{
❶ string name = read_string();

  // Concatenate name from client with base path
❷ string fullPath = "/files" + name;

❸ if(!exist(fullPath))
  {
  ❹ write_error("File " + fullPath + " doesn't exist");
  }
  else
  {
  ❺ write_file_to_client(fullPath);
  }
}
```

Memory Exhaustion Attacks

```
def read_buffer()
{
  byte buf[];
  uint32 len;
  int i = 0;

  // Read the number of bytes from the network
❶ len = read_uint32();

  // Allocate memory buffer
❷ buf = malloc(len);

  // Allocate bytes from network
❸ read_bytes(buf, len);

  printf("Read in %d bytes\n", len);
}
```

Storage Exhaustion Attacks

- **Compact embedded systems**
- **Logging**

CPU Exhaustion Attacks

- **Algorithmic Complexity**

```
def bubble_sort(int[] buf)
{
  do
  {
    bool swapped = false;
    int N = len(buf);
    for(int i = 1; i < N - 1; ++i)
    {
      if(buf[i-1] > buf[i])
      {
        // Swap values
        swap( buf[i-1], buf[i] );
        swapped = true;
      }
    }
  } while(swapped == false);
}
```

Notation	Description
O(1)	Constant time; the algorithm always takes the same amount of time.
O(log *N*)	Logarithmic; the worst case is proportional to the logarithm of the number of inputs.
O(*N*)	Linear time; the worst case is proportional to the number of inputs.
$O(N^2)$	Quadratic; the worst case is proportional to the square of the number of inputs.
$O(2^N)$	Exponential; the worst case is proportional to 2 raised to the power *N*.

CPU Exhaustion Attacks

- **Configurable Cryptography**

```
def process_authentication()
{
❶   string username = read_string();
    string password = read_string();
❷   int iterations = read_int();

    for(int i = 0; i < interations; ++i)
    {
❸     password = hash_password(password);
    }

❹   return check_user_password(username, password);
}
```

Format String Vulnerabilities

```
def process_authentication()
{
      string username = read_string();
      string password = read_string();

      // Print username and password to terminal
      printf(username);
      printf(password);

      return check_user_password(username, password))
}
```

Format specifier	Description	Potential vulnerabilities
%d, %p, %u, %x	Prints integers	Can be used to disclose information from the stack if returned to an attacker
%s	Prints a zero terminated string	Can be used to disclose information from the stack if returned to an attacker or cause invalid memory accesses to occur, leading to denial-of-service
%n	Writes the current number of printed characters to a pointer specified in the arguments	Can be used to cause selective memory corruption or application crashes

Command Injection

```
def update_password(string username)
{
❶ string oldpassword = read_string();
  string newpassword = read_string();

  if(check_user_password(username, oldpassword))
  {
    // Invoke update_password command
  ❷ system("/sbin/update_password -u " + username + " -p " + newpassword);
  }
}
```

- ***password; xcalc***

SQL Injection

```
def process_authentication()
{
❶   string username = read_string();
    string password = read_string();

❷   string sql = "SELECT password FROM user_table WHERE user = '" + username "'";

❸   return run_query(sql) == password;
}
```

Text-Encoding Character Replacement

- **ASCII**
- **Unicode**

```
def add_user()
{
❶ string username = read_unicode_string();

  // Ensure username doesn't contain any single quotes
❷ if(username.contains("'") == false)
  {
    // Add user, need to convert to ASCII for the shell
  ❸ system("/sbin/add_user '" + username.toascii() + "'");
  }
}
```

Summary

- **Many possible root causes**
- **Vulnerabilities appear in most surprise places**
- **Identifying vulnerabilities is complex**
  - Network protocols used
  - Third party libraries
  - Languages