

Chapter 20: Vulnerability Analysis

- What is a vulnerability?
- Example vulnerabilities
- Penetration studies
 - Flaw Hypothesis Methodology
 - Examples
- Vulnerability classification schemes

Definitions

- *Vulnerability (security flaw)*: failure of security policies, procedures, and controls that allow a subject to commit an action that violates the security policy
 - Subject is called an *attacker*
 - Using the failure to violate the policy is *exploiting the vulnerability* or *breaking in*

What is an Exploit?

- An **exploit** is any **input** (i.e., a piece of software, an argument string, or sequence of commands) that takes advantage of a bug, glitch or vulnerability in order to cause an attack
- An **attack** is an unintended or unanticipated behavior that occurs on computer software, hardware, or something electronic and that brings an advantage to the attacker

Incomplete List of Vulnerabilities – Latest Version

- Buffer overflows/memory safety errors
- Integer overflow, underflow, and sign conversion errors
- Null pointer errors
- Race conditions, atomicity violations, TOCTTOU (time of check to time of use) vulnerabilities
- Resource drains that may be subject to attack (e.g., memory or file handle leaks)
- Other memory errors (double frees, use after free, uninitialized memory use)
- Insufficient access control checks/incomplete mediation
- Tainted data/input validation errors (e.g., format string vulnerabilities, lack of checking on command strings sent to OS, insufficient restrictions on input file names)
- Unhandled exceptions or returned error status codes
- Leaks of confidential information

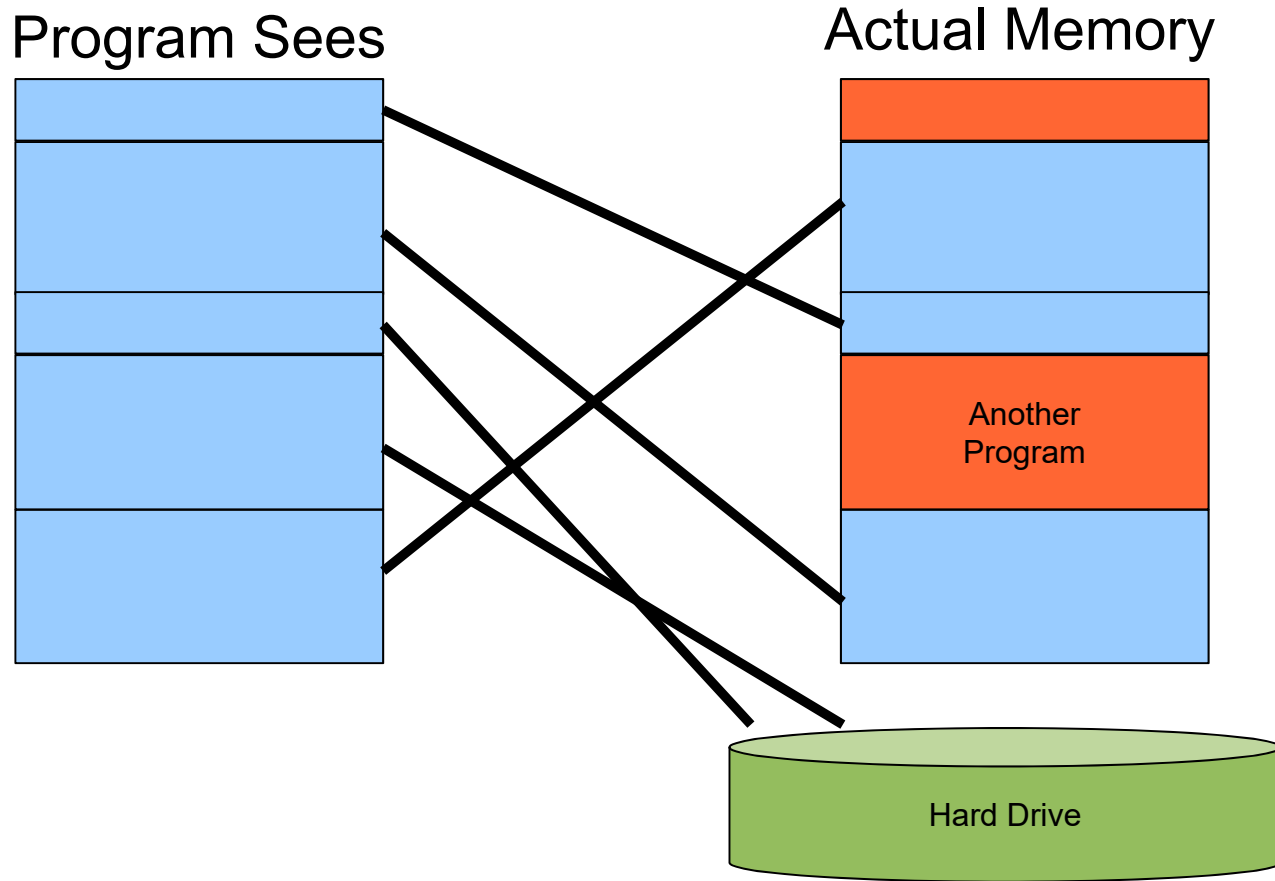
Buffer Overflow Attack

- One of the most common OS bugs is a **buffer overflow**
 - The developer fails to include code that checks whether an input string fits into its buffer array
 - An input to the running process exceeds the length of the buffer
 - The input string overwrites a portion of the memory of the process
 - Causes the application to behave improperly and unexpectedly
- Effect of a buffer overflow
 - The process can operate on malicious data or execute malicious code passed in by the attacker
 - If the process is executed as root, the malicious code will be executing with root privileges

Address Space

- Every program needs to access memory in order to run
- For simplicity sake, it would be nice to allow each process (i.e., each executing program) to act as if it owns all of memory
- The address space model is used to accomplish this
- Each process can allocate space anywhere it wants in memory
- Most kernels manage each process' allocation of memory through the **virtual memory** model
- How the memory is managed is irrelevant to the process

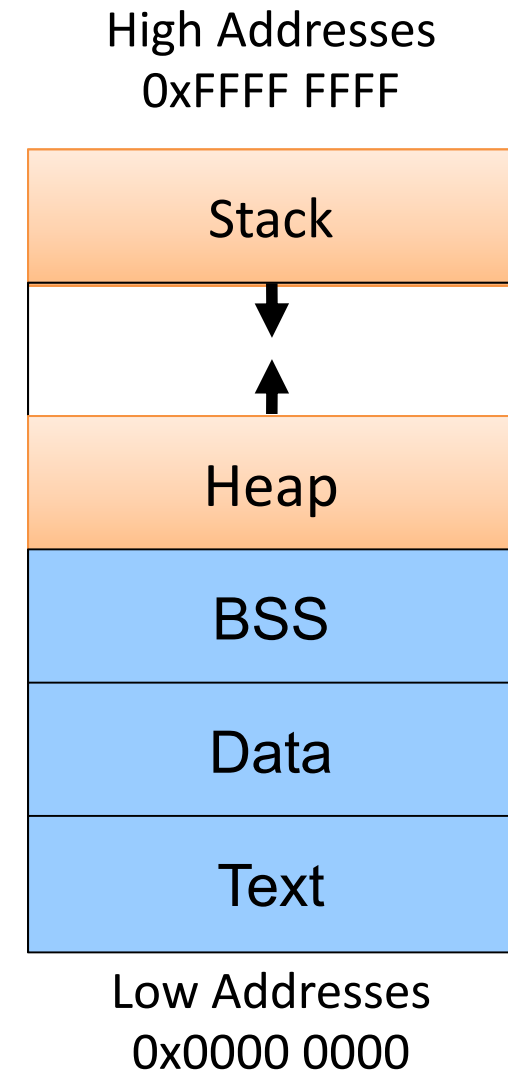
Virtual Memory



Mapping virtual addresses to real addresses

Unix Address Space

- **Text**: machine code of the program, compiled from the source code
- **Data**: static program variables initialized in the source code prior to execution
- **BSS** (block started by symbol): static variables that are uninitialized
- **Heap**: data dynamically generated during the execution of a process
- **Stack**: structure that grows downwards and keeps track of the activated method calls, their arguments and local variables



Vulnerabilities and Attack Method

- Vulnerability scenarios
 - The program has **root** privileges (**setuid**) and is launched from a shell
 - The program is part of a web application
- Typical attack method
 1. Find vulnerability
 2. Reverse engineer the program
 3. Build the exploit

Buffer Overflow Attack in a Nutshell

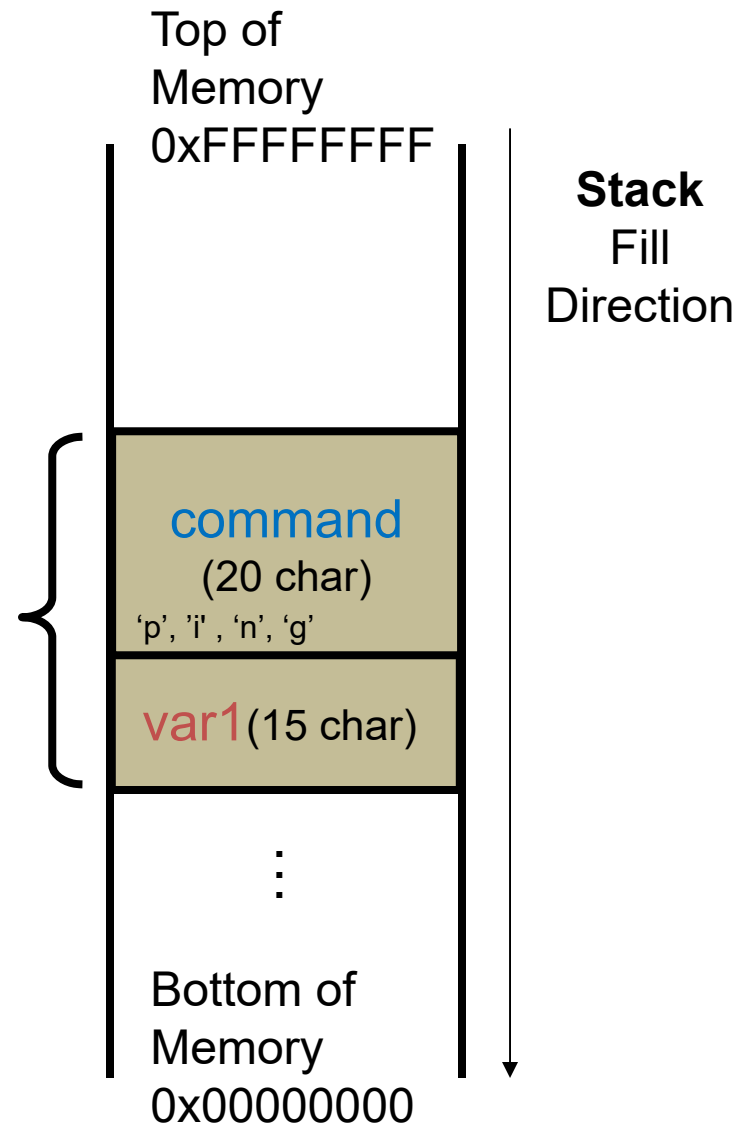
- First described in
 - Aleph One. Smashing The Stack For Fun And Profit. e-zine [www.Phrack.org](http://www.phrack.org) #49, 1996
- The attacker exploits an unchecked buffer to perform a buffer overflow attack
- The ultimate goal for the attacker is getting a shell that allows to execute arbitrary commands with high privileges
- Kinds of buffer overflow attacks:
 - Heap smashing
 - Stack smashing

Buffer Overflow

mybing.c

```
main(int argc, char *argv[ ])
/* get user_input */
{
    char command[20];
    char var1[15];
    strcpy(command, "ping ");
    strcat(command, argv[1]);
    strcpy(var1, argv[1]);
    printf("The result of myping %s is:\n",
    var1);
    system(command);
}
```

- A simple wrapper around ping

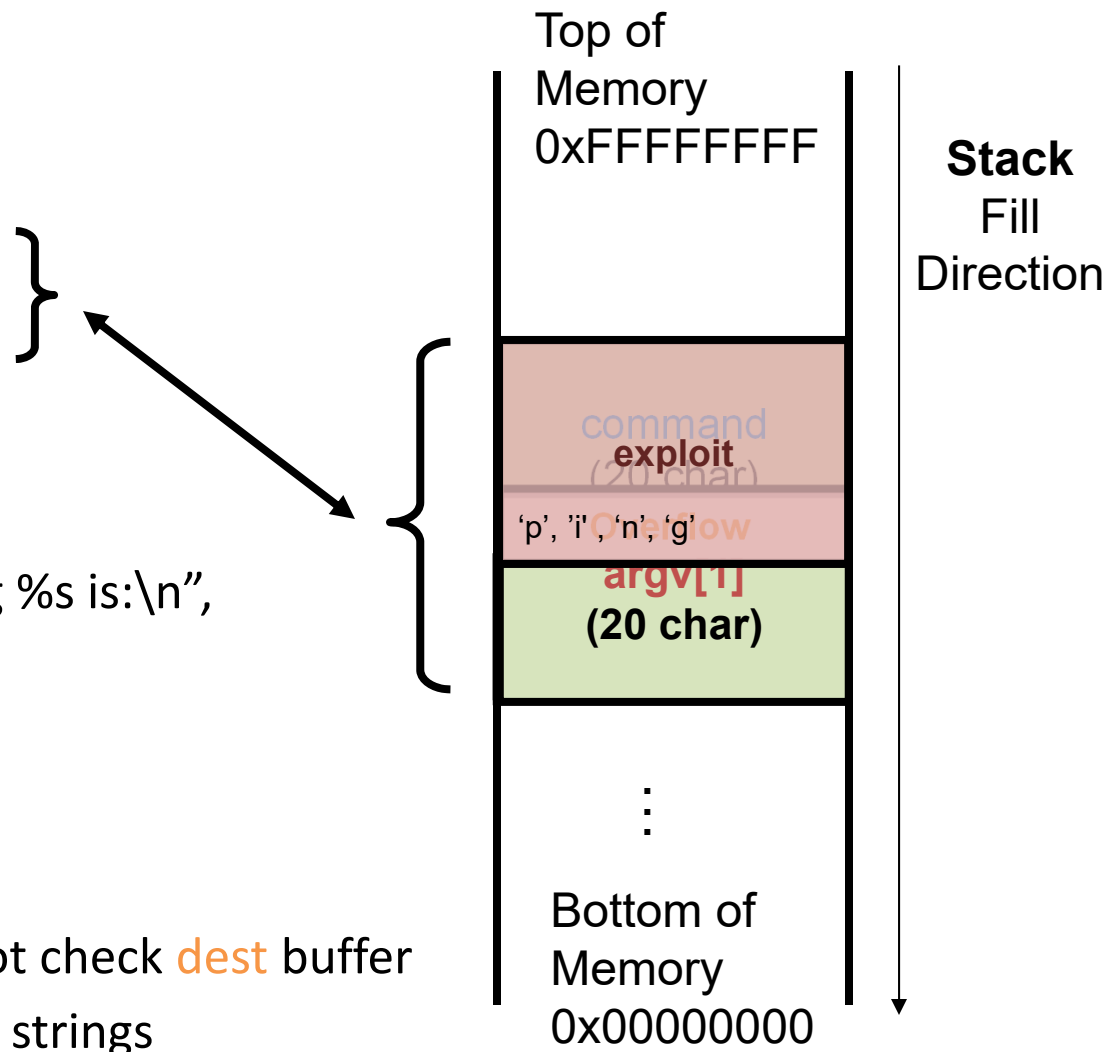


strcpy() Vulnerability

myping.c

```
main(int argc, char *argv[ ])
/* get user_input */
{
    char command[20];
    char var1[15];
    strcpy(command, "ping ");
    strcat(command, argv[1]);
    strcpy(var1, argv[1]);
    printf("The result of myping %s is:\n",
    var1);
    system(command);
}
```

- `argv[1]` is the user input
- `strcpy(dest, src)` does not check `dest` buffer
- `strcat(d, s)` concatenates strings



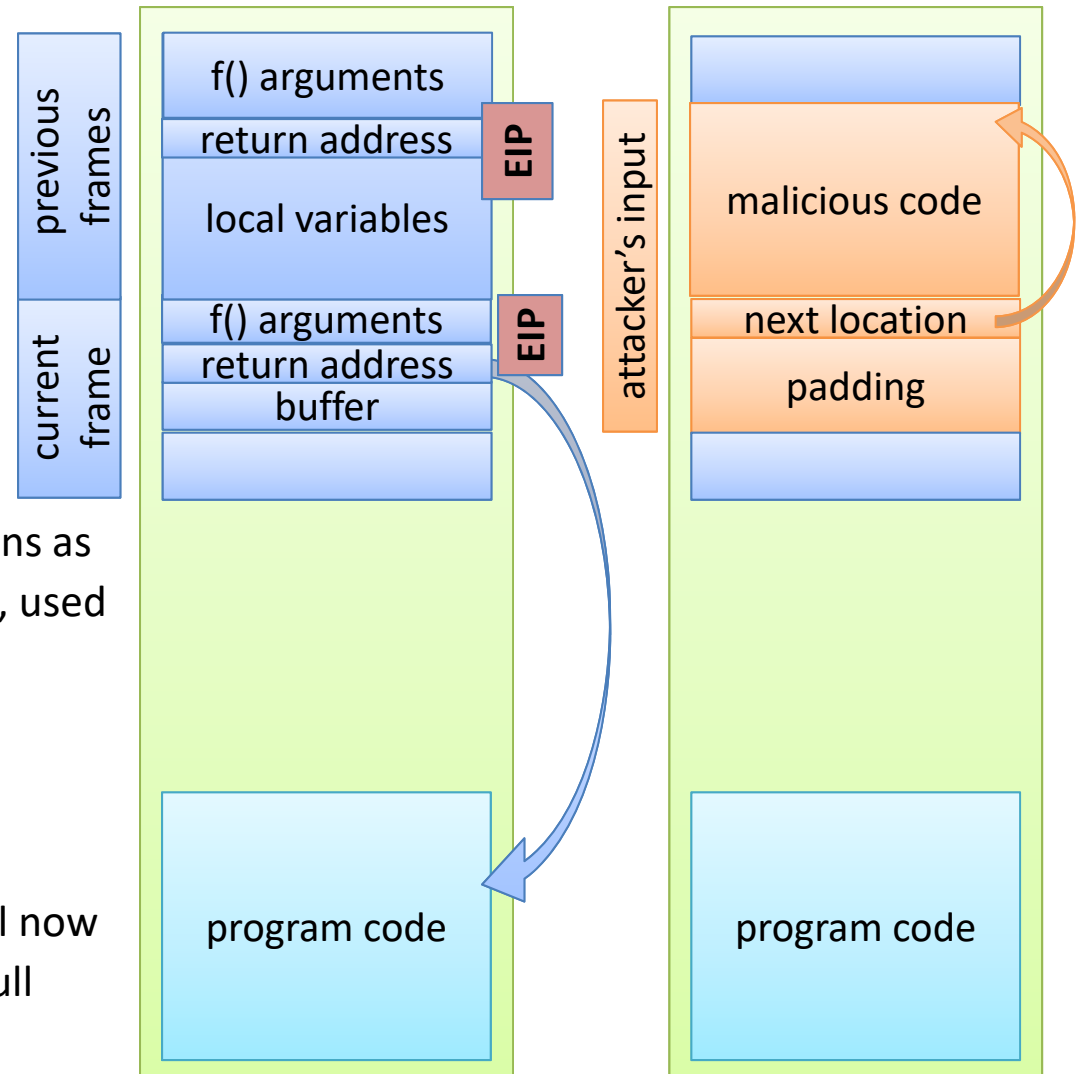
strcpy() vs. strncpy()

- Function `strcpy()` copies the string in the second argument into the first argument
 - e.g., `strcpy(dest, src)`
 - If source string > destination string, the overflow characters may occupy the memory space used by other variables
 - The **null character** is appended at the end automatically
- Function `strncpy()` copies the string by specifying the number `n` of characters to copy
 - e.g., `strncpy(dest, src, n); dest[n] = '\0'`
 - If source string is longer than the destination string, the overflow characters are discarded automatically
 - You have to place the **null character** manually

Return Address Smashing

```
void fingerd (...) {  
    char buf[80];  
    ...  
    get(buf);  
    ...  
}
```

- The Unix `fingerd()` system call, which runs as root (it needs to access sensitive files), used to be vulnerable to buffer overflow
- Write malicious code into buffer and overwrite return address to point to the malicious code
- When return address is reached, it will now execute the malicious code with the full rights and privileges of root



Unix Shell Command Substitution

- The Unix shell enables a command argument to be obtained from the standard output of another
- This feature is called **command substitution**
- When parsing command line, the shell replaces the output of a command between back quotes with the output of the command
- Example:
 - File **name.txt** contains string **farasi**
 - The following two commands are equivalent
 - **finger `cat name.txt`**
 - **finger farasi**

Shellcode Injection

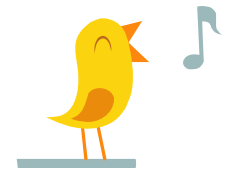
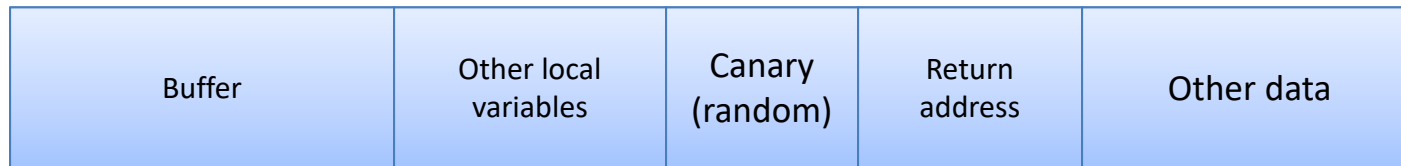
- An exploit takes control of attacked computer so injects code to “spawn a shell” or “shellcode”
- A shellcode is:
 - Code assembled in the CPU’s native instruction set (e.g. x86 , x86-64, arm, sparc, risc, etc.)
 - Injected as a part of the buffer that is overflowed.
- We inject the code directly into the buffer that we send for the attack
- A buffer containing shellcode is a “payload”

Buffer Overflow Mitigation

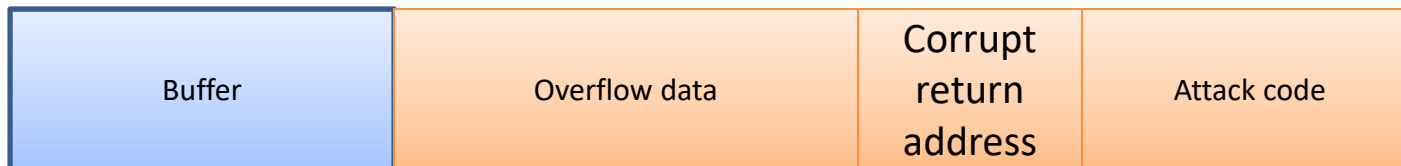
- We know **how** a buffer overflow happens, but **why** does it happen?
- This problem could not occur in Java; it is a C problem
 - In Java, objects are allocated dynamically on the heap (except ints, etc.)
 - Also cannot do pointer arithmetic in Java
 - In C, however, you can declare things directly on the stack
- One solution is to make the buffer dynamically allocated
- Another (OS) problem is that **fingerd** had to run as root
 - Just get rid of **fingerd**'s need for root access (solution eventually used)
 - The program needed access to a file that had sensitive information in it
 - A new world-readable file was created with the information required by **fingerd**

Stack-based buffer overflow detection using a random canary

Normal (safe) stack configuration:



Buffer overflow attack attempt:



- The canary is placed in the stack prior to the return address, so that any attempt to overwrite the return address also over-writes the canary.

Cross Site Scripting (XSS)

- Attacker injects scripting code into pages generated by a web application
 - Script could be malicious code
 - JavaScript (AJAX!), VBScript, ActiveX, HTML, or Flash
- Threats:
 - Phishing, hijacking, changing of user settings, cookie theft/poisoning, false advertising , execution of code on the client, ...

XSS Example

- Website allows posting of comments in a guestbook
- Server incorporates comments into page returned

```
<html>
<body>
<title>My Guestbook!</title>
Thanks for signing my guestbook!<br />
Here's what everyone else had to say:<br />
Joe: Hi! <br />
John: Hello, how are you? <br />
Jane: How does this guestbook work? <br />
</body>
```

- Attacker can post comment that includes malicious JavaScript

```
Evilguy: <script>alert("XSS Injection!");
</script> <br />
```

guestbook.html

```
<html>
<title>Sign My Guestbook!</title>
<body>
Sign my guestbook!
<form action="sign.php"
      method="POST">
  <input type="text" name="name">
  <input type="text" name="message"
        size="40">
  <input type="submit" value="Submit">
</form>
</body>
</html>
```

SQL Injection

- SQL injection occurs when user input is not filtered for escape characters and is then passed into an SQL statement

- `uName = getQueryString("username");`
`uPass = getQueryString("userpassword");`

`sql = 'SELECT * FROM Users WHERE Name = ' + uName + ' AND Pass = ' + uPass + ' '`

- Normal case:

- `SELECT * FROM Users WHERE Name = "John Doe" AND Pass = "myPass"`

- What if the user inputs User Name as `" or ""="` and Password as `" or ""="`

- We get

- `SELECT * FROM Users WHERE Name = "" or ""=" AND Pass = "" or ""="`

The condition is satisfied because `or ""="` is always TRUE

Definitions

- *Vulnerability (security flaw)*: failure of security policies, procedures, and controls that allow a subject to commit an action that violates the security policy
 - Subject is called an *attacker*
 - Using the failure to violate the policy is *exploiting the vulnerability* or *breaking in*

Incomplete List of Vulnerabilities – Latest Version

- Buffer overflows/memory safety errors
- Integer overflow, underflow, and sign conversion errors
- Null pointer errors
- Race conditions, atomicity violations, TOCTTOU (time of check to time of use) vulnerabilities
- Resource drains that may be subject to attack (e.g., memory or file handle leaks)
- Other memory errors (double frees, use after free, uninitialized memory use)
- Insufficient access control checks/incomplete mediation
- Tainted data/input validation errors (e.g., format string vulnerabilities, lack of checking on command strings sent to OS, insufficient restrictions on input file names)
- Unhandled exceptions or returned error status codes
- Leaks of confidential information

Classification Scheme by the Research Into Secure Operating Systems (RISOS) Project

- Incomplete parameter validation
- Inconsistent parameter validation
- Implicit sharing of privileged/confidential data
- Asynchronous validation/inadequate serialization
- Inadequate
identification/authentication/authorization
- Violable prohibition/limit
- Exploitable logic error

Incomplete Parameter Validation

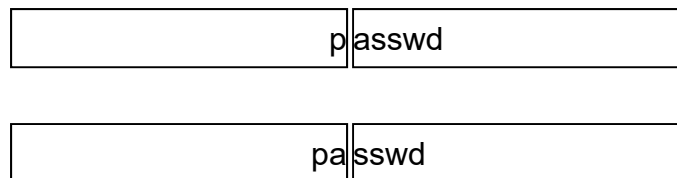
- Parameter not checked before use
- Example: emulating integer division in SunOS kernel (RISC chip involved)
 - Caller provided addresses for quotient, remainder
 - Quotient address checked to be sure it was in user's protection domain
 - Remainder address *not* checked
 - Set remainder address to address of process' level of privilege
 - Compute $25/5$ and you have level 0 (kernel) privileges
- Solution: check for type, format, range of values, access rights, presence (or absence)

Inconsistent Parameter Validation

- Each routine checks parameter is in proper format for that routine but the routines require different formats
- The inconsistency across interfaces causes this flaw
- Example: each database record is one line, with colons separating fields
 - One program accepts colons, newlines as part of data within fields
 - Another program reads them as field and record separators
 - This allows bogus records to be entered

Implicit Sharing of Privileged / Confidential Data

- OS does not isolate users / processes properly
- Example: attacking file password protection in TENEX
 - OS allows user to determine when paging occurs
 - Files protected by passwords
 - Passwords checked character by character; stops at first incorrect one
 - Attack:
 - (1) position guess for password so page boundary occurred between 1st and 2nd character
 - If no page fault, 1st char was wrong; if page fault, it was right
 - (2) Continue until password discovered



Asynchronous Validation / Inadequate Serialization

- Time of check to time of use (TOCTTOU) flaws, intermixing reads and writes to create inconsistencies
- Example: *vi* flaw discussed next

Definition of TOCTTOU

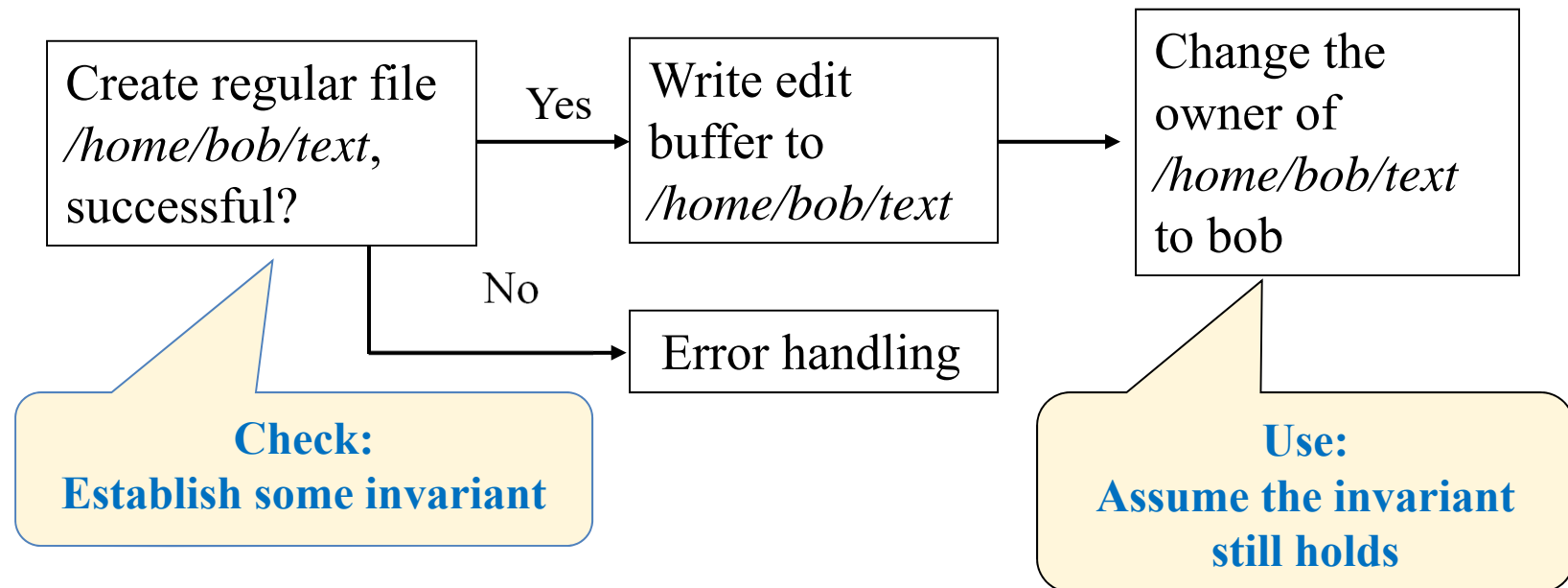


- Time-Of-Check-To-Time-Of-Use, a race condition in Unix-style file systems.
- Check - establishes some precondition (invariant) about a file.
- Use - Operates on the file assuming that the invariant is still valid.

TOCTTOU Call Pair: vi



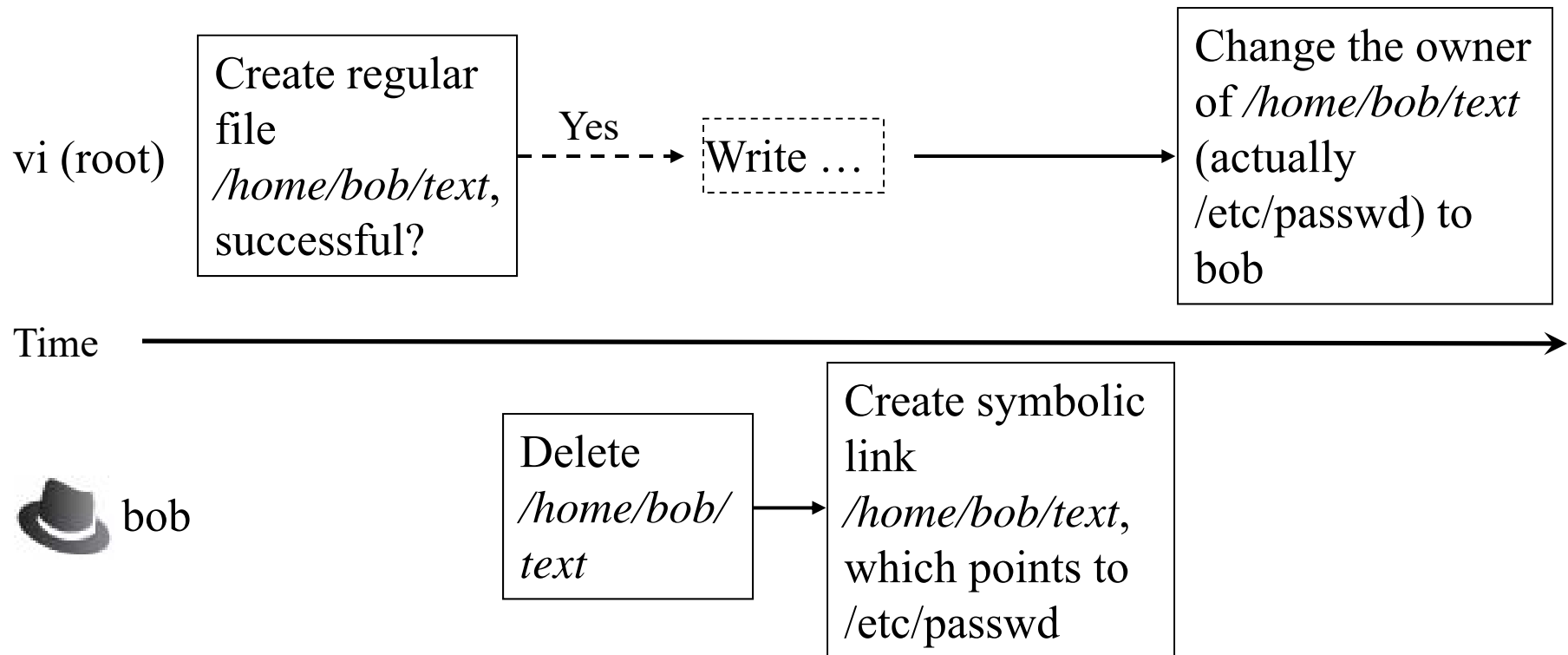
- Runs as root to edit a file owned by a normal user.
E.g., `vi /home/bob/text`, then saves the file.



TOCTTOU Attack Scenario: vi



Effect: The attacker may get unauthorized root access!



Inadequate Identification / Authorization / Authentication

- Erroneously identifying user, assuming another's privilege, or tricking someone into executing program without authorization
- Example: on UNIVAC 1100, access to a file named "SYS\$*DLOC\$" meant process privileged
 - Check: can process access any file with qualifier name beginning with "SYS" and file name beginning with "DLO"?
 - There was an *ordinary* file named "SYSA*DLOC\$", which can be accessed by any process. Therefore, an attacker just needs to access this file and her process is privileged

Violable Prohibition / Limit

- Boundary conditions not handled properly
- Example: early versions of the TENEX OS kept in low memory, user process in high memory
 - Boundary was highest address of OS: `os_top`
 - All memory accesses by user apps checked against this:
if `addr > os_top`, okay
 - Memory addresses beyond end of high memory are reduced **modulo** memory size
 - So, process could access $(\text{memory size})+1$, which is greater than `os_top` and thus allowed
 - But actually the hardware accesses word 1, which is part of OS ...

Exploitable Logic Error

- Problems not falling into other classes
 - Incorrect error handling, unexpected side effects, incorrect resource allocation, etc.
- Example: unchecked return from monitor
 - Monitor adds 1 to address in user's return word, returns to that address
 - Index bit (indicating indirection) is a bit in word
 - Attack: set return word to be -1 ; adding 1 overflows, changes index bit, so return is to location stored in register 1
 - Arrange for this to point to bootstrap program (of the attacker) stored in other registers
 - On return, program executes with system privileges

Which of the following is not a type of vulnerability?

A. Buffer overflow B. Race C. Denial of Service D. Unhandled exception E. Double free

Start the presentation to activate live content

If you see this message in presentation mode, install the add-in or get help at PollEv.com/app

Total Results: 0

How to Detect Vulnerabilities?

- Formal verification
- Penetration testing

Formal Verification

- Mathematically verifying that a system satisfies certain **constraints**
- *Preconditions* state assumptions about the system
- *Postconditions* are result of applying system operations to preconditions, inputs
- Required: postconditions satisfy constraints

Penetration Testing

- Testing to verify that a system satisfies certain **constraints**
- Hypothesis stating system characteristics, environment, and state relevant to vulnerability
- Result is compromised system state
- Apply tests to try to move system from state in hypothesis to compromised system state (**constraints** violated)

Penetration Studies

- Test for evaluating the strengths and effectiveness of all security controls on system
 - Also called *tiger team attack* or *red team attack*
 - Goal: violate site security policy
 - Not a replacement for careful design, implementation, and structured testing
 - Tests system *in toto*, once it is in place
 - Includes procedural, operational controls as well as technological ones

Flaw Hypothesis Methodology

1. Information gathering
 - Become familiar with system's functioning
2. Flaw hypothesis
 - Draw on knowledge to hypothesize vulnerabilities
3. Flaw testing
 - Test them out
4. Flaw generalization
 - Generalize vulnerability to find others like it
5. (*maybe*) Flaw elimination
 - Testers eliminate the flaw (usually *not* included)

Metasploit Framework

- Network Scanning and Host Fingerprinting:
 - Find active hosts on the network (IP address)
 - Find vulnerable hosts (OS, services, open ports, etc)
 - Nmap
- Testing whether a target host has a known vulnerability
 - E.g., CVE-2008-4250

Common Vulnerabilities and Exposures

- CVE (<http://cve.mitre.org/>) is a dictionary of publicly known information security vulnerabilities and exposures

The screenshot shows the CVE website homepage. At the top, there's a navigation bar with links for CVE LIST, COMPATIBILITY, NEWS — MARCH 21, 2013, and SEARCH. Below this is a banner with the CVE logo and the text "Common Vulnerabilities and Exposures" and "The Standard for Information Security Vulnerabilities". A green bar indicates "TOTAL CVEs: 55084". The main content area is titled "CVE List Main Page" and describes the CVE® as a publicly available and free to use list or dictionary of standardized identifiers for common computer vulnerabilities and exposures. It features two main sections: "National Vulnerability Database" and "CVE List Master Copy". The "National Vulnerability Database" section states that full database functionality for the CVE List is provided through MITRE's partnership with the U.S. National Vulnerability Database (NVD) and lists links for "CVE Search on NVD", "CVE Fix Information", and "CVE SCAP Mappings". The "CVE List Master Copy" section states that the master copy of the CVE List is maintained for the community by MITRE on this public CVE Web site and lists links for "Search Master Copy of CVE", "Download CVE List", and "View CVE List". A sidebar on the left contains links for "About CVE", "Terminology", "Documents", "FAQs", "CVE List", "About CVE Identifiers", "Search CVE", "Search NVD", "Updates & RSS Feeds", "Request a CVE-ID", "CVE In Use", "CVE-Compatible Products", "NVD for CVE Fix Information", "CVE Numbering Authorities", "News & Events", "Calendar", "Free Newsletter", "Community", "CVE Editorial Board", "Sponsor", "Contact Us", "Search the Site", and "Site Map". A sidebar on the right contains links for "CVE List", "About CVE Identifiers", "Editorial Policies", "Data Sources", "Reference Materials", "Search Tips", "Updates & RSS Feeds", "Request a CVE Identifier", "ITEMS OF INTEREST", "Terminology", and "NVD".

HOME > CVE LIST

About CVE
Terminology
Documents
FAQs

CVE List
About CVE Identifiers
Search CVE
Search NVD
Updates & RSS Feeds
Request a CVE-ID

CVE In Use
CVE-Compatible Products
NVD for CVE Fix Information
CVE Numbering Authorities

News & Events
Calendar
Free Newsletter

Community
CVE Editorial Board
Sponsor
Contact Us

Search the Site
Site Map

CVE List Main Page

CVE® is a publicly available and free to use list or dictionary of standardized identifiers for common computer vulnerabilities and exposures.

National Vulnerability Database
Full database functionality for the CVE List is provided through MITRE's partnership with the U.S. [National Vulnerability Database \(NVD\)](#).

- [CVE Search on NVD](#)
- [CVE Fix Information](#)
- [CVE SCAP Mappings](#)

CVE List Master Copy
The master copy of the CVE List is maintained for the community by MITRE on this public CVE Web site.

- [Search Master Copy of CVE](#)
- [Download CVE List](#)
- [View CVE List](#)

CVE List
About CVE Identifiers
Editorial Policies
Data Sources
Reference Materials
Search Tips
Updates & RSS Feeds
Request a CVE Identifier

ITEMS OF INTEREST
Terminology
NVD

TOTAL CVEs: 55084

You may download the CVE List, copy it, redistribute it, reference it, and analyze it, provided you **do not modify** CVE itself as per our [Terms of Use](#). CVE and NVD are both sponsored by the [National Cyber Security Division](#) of the [U.S. Department of Homeland Security](#).

NVD (National Vulnerability Database)

<http://nvd.nist.gov/>

The screenshot shows the NVD website interface. The browser address bar displays `web.nvd.nist.gov/view/vuln/search-results?query=xss&search_type=all&cves=on`. The page header includes the NIST logo and the text "Sponsored by DHS National Cyber Security Division/US-CERT". The main title is "National Vulnerability Database" with the subtitle "automating vulnerability management, security measurement, and compliance checking". A navigation bar contains links for "Vulnerabilities", "Checklists", "800-53/800-53A", "Product Dictionary", "Impact Metrics", "Data Feeds", and "Statistics". Below this is a secondary navigation bar with "Home", "SCAP", "SCAP Validated Tools", "SCAP Events", "About", "Contact", and "Vendor Comments".

The left sidebar contains the following sections:

- Mission and Overview**: NVD is the U.S. government repository of standards based vulnerability management data. This data enables automation of vulnerability management, security measurement, and compliance (e.g. FISMA).
- Resource Status**:
 - NVD contains:**
 - 55603 [CVE Vulnerabilities](#)
 - 203 [Checklists](#)
 - 244 [US-CERT Alerts](#)
 - 2701 [US-CERT Vuln Notes](#)
 - 8140 [OVAL Queries](#)
 - 70436 [CPE Names](#)
 - Last updated:** Tue Mar 26 15:40:16 EDT 2013
 - CVE Publication rate:** 12.97
- Email List**: NVD provides four mailing lists to the public. For information and subscription instructions please visit [NVD Mailing Lists](#).

The main content area shows search results for "XSS". It indicates "There are 7,287 matching records. Displaying matches 1 through 20." and provides a pagination link: `1 2 3 4 5 6 7 8 9 10 11 > >>`. The first three results are displayed:

- CVE-2013-1833**
 - Summary:** Multiple cross-site scripting (XSS) vulnerabilities in the File Picker module in Moodle 2.x through 2.1.10, 2.2.x before 2.2.8, 2.3.x before 2.4.x before 2.4.2 allow remote authenticated users to inject arbitrary web script or HTML via a crafted filename.
 - Published:** 03/25/2013
 - CVSS Severity:** 3.5 (LOW)
- CVE-2013-1833**
 - Summary:** Multiple cross-site scripting (XSS) vulnerabilities in the File Picker module in Moodle 2.x through 2.1.10, 2.2.x before 2.2.8, 2.3.x before 2.4.x before 2.4.2 allow remote authenticated users to inject arbitrary web script or HTML via a crafted filename.
 - Published:** 03/25/2013
 - CVSS Severity:** 3.5 (LOW)
- CVE-2013-1833**
 - Summary:** Multiple cross-site scripting (XSS) vulnerabilities in the File Picker module in Moodle 2.x through 2.1.10, 2.2.x before 2.2.8, 2.3.x before 2.4.x before 2.4.2 allow remote authenticated users to inject arbitrary web script or HTML via a crafted filename.
 - Published:** 03/25/2013
 - CVSS Severity:** 3.5 (LOW)

The fourth result is partially visible:

- CVE-2013-2501**
 - Summary:** Cross-site scripting (XSS) vulnerability in the Terillion Reviews plugin before 1.2 for WordPress allows remote attackers to inject arbitrary or HTML via the ProfileId field.
 - Published:** 03/22/2013
 - CVSS Severity:** 4.3 (MEDIUM)

The fifth result is also partially visible:

- CVE-2013-2501**
 - Summary:** Cross-site scripting (XSS) vulnerability in the Terillion Reviews plugin before 1.2 for WordPress allows remote attackers to inject arbitrary

Secunia

secunia.com/vulnerability-review/vulnerability_update_top50.html

SECUNIA
VULNERABILITY REVIEW 2013 - THE HIGHLIGHTS

Secunia
Stay Secure

Vulnerability Update - All Vulnerability Update - Top 50 Time to Patch Vendor Update SCADA Security Browser Security Back to overview

Contact Secunia Download the report Get a free scan of your IT network

Find out how many vulnerabilities were discovered in the 50 most popular programs in 2012.

Identifying the 50 most popular programs (the Top 50 portfolio)

To assess how exposed endpoints are, we analyze the types of products typically found on an endpoint. For this analysis we use anonymous data gathered from scans throughout 2012 of the millions of private computers which have the Secunia Personal Software Inspector (PSI) installed.

The total number of vulnerabilities in the Top 50 most popular programs was 1,137 in 2012, showing a 98% increase in the 5 year trend.	Most of these were rated by Secunia as either 'Highly critical' (78.8%) or 'Extremely critical' (5.3%).	The 1,137 vulnerabilities were discovered in 18 products in the Top 50 portfolio - that's 63 vulnerabilities per vulnerable product on average.	Vulnerabilities were discovered in 18 products from 8 vendors, including Windows OS.	With a 91% share, the primary attack vector in the Top 50 portfolio was Remote Network.
--	---	---	--	---

Read more in the Secunia Vulnerability Review 2013. [Download it here.](#)

TOP 50 What is the Top 50 portfolio?

Twitter Facebook LinkedIn Google+

- <https://secuniaresearch.flexerasoftware.com/community/research/>