System Testing

Spring, 2023

Yi Xiang xiangyi@scut.edu.cn

Contents

- Non-Functional Testing
  - Usability Testing
  - Performance Testing
  - Security Testing

Cornered
by Mike Baldwin
11-18 © 2005 Mike Baldwin / Dist. by Universal Press Syndicate www.cornered.com
cornered@comic.com
SALE

"They're fake. Part of the new false sense of security system."

System Security

- The goal of computer security is to protect computer assets (*e.g.*, servers, applications, web pages, data) from:
  - corruption
  - unauthorized access
  - denial of authorized access
  - malicious software

- Security is strengthened by:
  - physically limiting the access of computers to trusted users
  - hardware mechanisms (*e.g.*, biometrics)
  - operating system mechanisms that impose rules on untrusted programs (*e.g.*, role-based access control)
  - anti-virus software to detect malware
  - secure coding techniques (*e.g.*, array bounds checking) to make code less vulnerable to security attacks.

Security Testing

- Security Testing is a type of Software Testing that uncovers vulnerabilities of the system and determines that the data and resources of the system are protected from possible intruders.

- Security testing of any system focuses on finding all possible loopholes and weaknesses of the system which might result into the loss of information or repute of the organization.

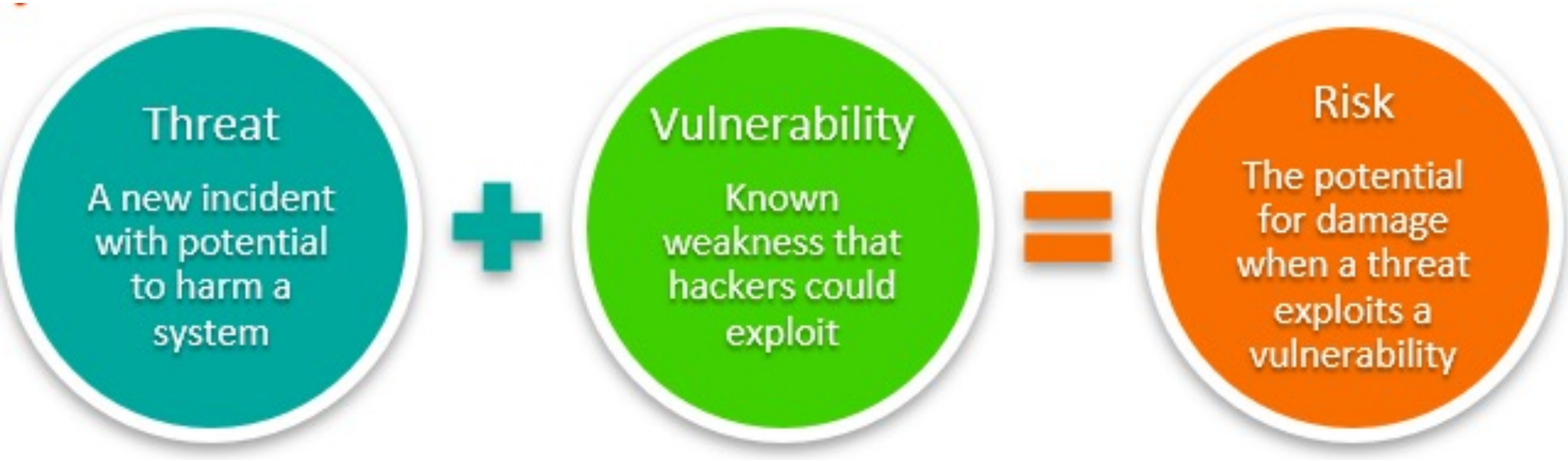
- The goal of security testing is to:

1. To identify the threats in the system.
2. To measure the potential vulnerabilities of the system.
3. To help in detecting every possible security risks in the system.
4. To help developers in fixing the security problems through coding

Security Testing - Terms

- Security Testing is a type of Software Testing that uncovers vulnerabilities of the system and determines that the data and resources of the system are protected from possible intruders.

  - Corrupted (loss of integrity)
  - Leaky(loss of confidentiality)
  - Unavailable or very slow (loss of availability)



Threat

A new incident with potential to harm a system

+

Vulnerability

Known weakness that hackers could exploit

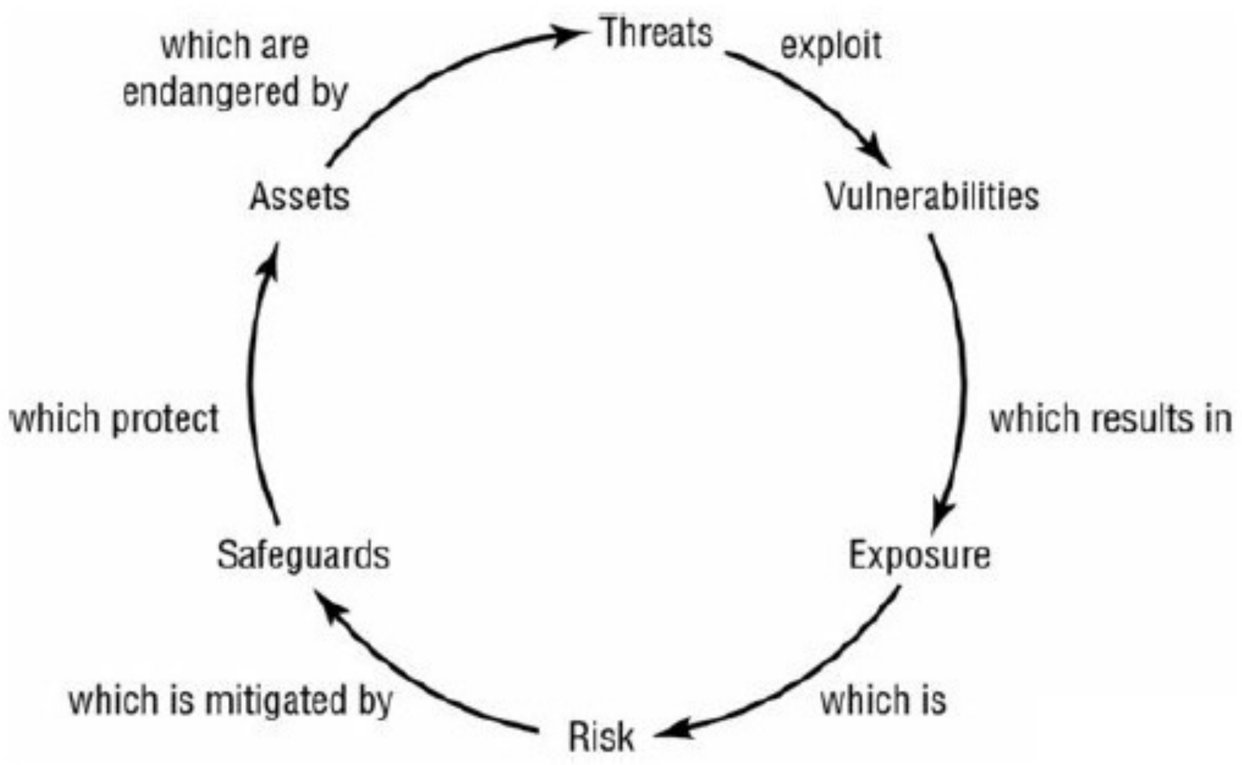
=

Risk

The potential for damage when a threat exploits a vulnerability

Approaches to Security Testing

1. Threat Modeling
2. Vulnerability Scanning
3. Penetration testing



1. Threat Modeling

- Threat modeling is a process that helps the architecture team:
  - Accurately determine the attack surface for the application
  - Assign risk to the various threats
  - Drive the vulnerability mitigation process

- The purpose of threat modeling
  - Provide defenders with a systematic analysis what controls or defenses need to be included , given the nature of the system , the probable attacker's profile , the most likely attack vectors , and the assets most desired by an attacker.

- Threat modeling answers questions like:

"Where am I most vulnerable to attack?"
"What are the most relevant threats?"
"What do I need to do to safeguard against these threats?"

Threat Modeling process

1) Understand the security requirements
  - Use Scenarios –what are the boundaries of the security problem
  - Identify external dependencies –OS, web server, network, …
  - Define security assumptions –what can you expect with regard to security; will the DB encrypt columns? Is there a key manager? What are the limitations you are working with.

2) Create an activity matrix (actor-asset-action matrix)
  - Identify assets
  - Identify roles
  - Their interaction

3) Create Trust Boundaries
  - Identify threats that put assets at risk
  - Identify attacks that can be used to realize each threat
    - Threat Trees
    - Abuse Cases
  - Determine the risk for each attack and prioritize (if needed)
  - Define the conditions required for each attack to be successful

4) Plan and implement your mitigations

Threat Modeling Example - Online Library System Threat Model

1) Use Scenarios

- Students can search the database(s)
- Students can put holds on some items for checkout
- Staff can search the database(s)
- Staff can place some items on reserve for up to 15 weeks
- Librarians can do anything students or staff can do
- Librarians can place items on an invisible list
- Librarians can access limited account information

Threat Modeling Example - Online Library System Threat Model

2) External Dependencies

- Server type will be Linux
- System will have to be off-campus accessible
- MySQL database
- Database server will be the existing library server
- Private network between web server and dbserver
- Both servers must be behind the campus firewall
- All communications over TLS

Threat Modeling Example - Online Library System Threat Model

3) Roles

- *Anonymous user* –connected, but not yet authenticated
- *Invalid user* –attempted to authenticate and failed
- *Student* –authenticated student
- *Staff* –authenticated staff
- *Librarian* –authenticated librarian
- *System admin* –authenticated site administrator with configuration privileges
- *DB admin* –authenticated database administrator with full db privileges
- *Web server user* –user/process id of web server
- *Database read user* –dbuser for accessing the database with read-only access
- *Database write user* –dbuser for accessing the database with read-write access

Threat Modeling Example - Online Library System Threat Model

4) Assets

- Library users and librarian
- User credentials
- Librarian credentials
- User personal information
- Web site system
- DB system
- Availability of the web server
- Availability of the DB server
- User code execution on web site
- User DB read access
- Librarian/Admin code execution on the web site
- Librarian/Admin DB read/write access
- Ability to create users
- Ability to audit system events

Threat Modeling Example - Online Library System Threat Model

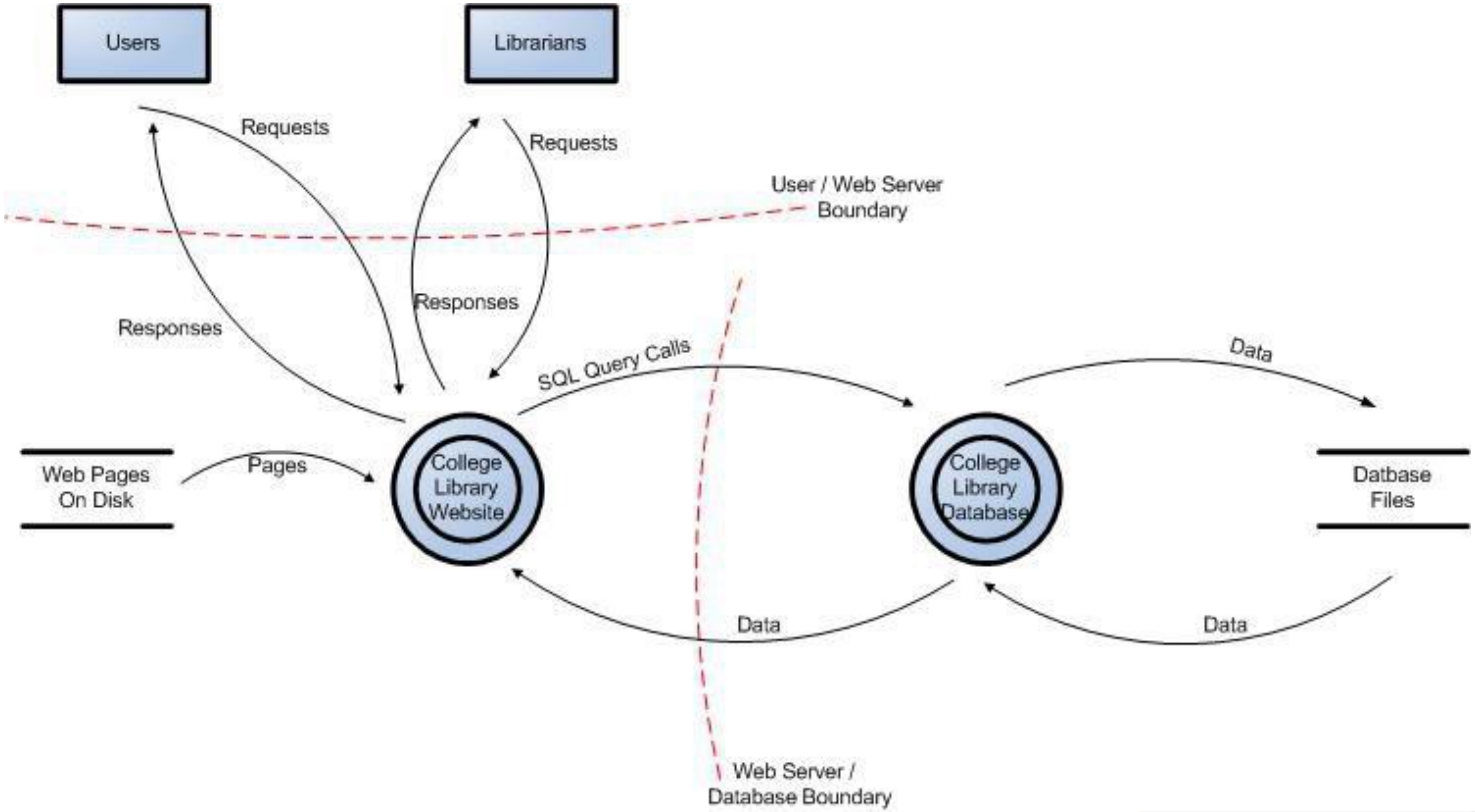
5) Activity Matrix

Asset/Role	Anonymous				Invalid				Student				Faculty				Librarian				Admin			
	C	R	U	D	C	R	U	D	C	R	U	D	C	R	U	D	C	R	U	D	C	R	U	D
Users	A	-	-	-	A	-	-	-	-	-	-	-	-	-	-	-	X	X	X	-				
Librarians	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-				
Personal info	-	-	-	-	-	-	-	-	B	B	B	-	B	B	-	-	-	-	-	-				
Web site	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	C	-	-	X	X	X	X

- A = Create if valid name, id, pin provided
- B = Only for their own profile information
- C = Must be limited to specific files, tables. No access to web site files.

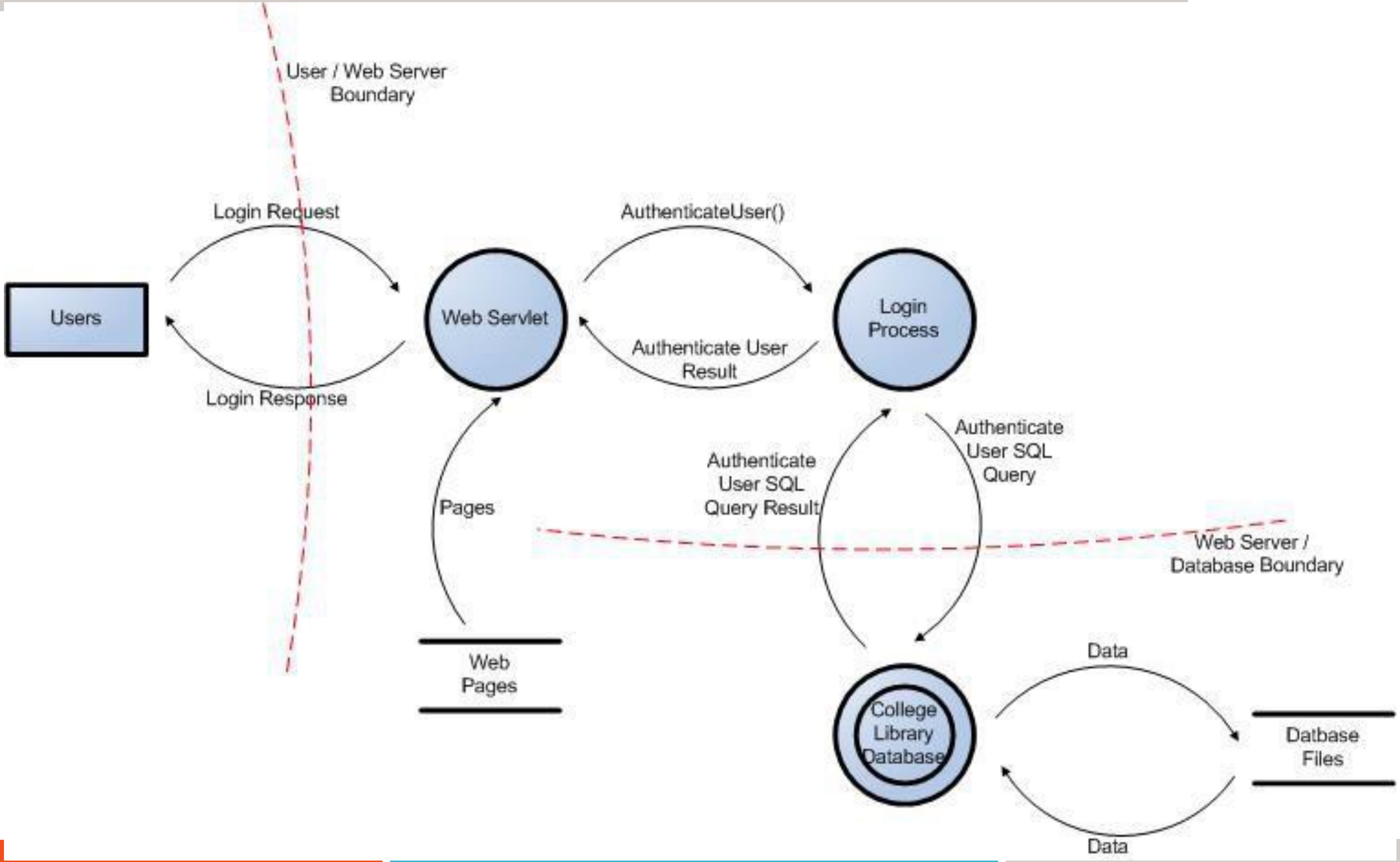
Threat Modeling Example - Online Library System Threat Model

6) Trust Boundaries



Threat Modeling Example - Online Library System Threat Model

7) Login DFD



Threat Modeling Example - Online Library System Threat Model

8) Threats

- Anonymous user *evades the authentication system*
- Anonymous user *gathers information from the authentication* system
- Anonymous user can forcefully browse to pages
- Librarian has access to web site pages on the server
- Student or Staff can *modify privilege level*
- Student or Staff can forcefully *browse to restricted pages*
- Any user can tamper with *critical data* on the client
- Student/Staff/Anonymous can *inject SQL* into the database
- Student/Staff/Anonymous can *inject JavaScript* into an HTML page
- SSL version is vulnerable or allows *vulnerable algorithms*
- …..

It is fine to use STRIDE and think about every place where Spoofing, Tampering, …. can be used

Microsoft STRIDE Threat Model

Identify Threats

- Experts can brainstorm
- How to do this without being an expert?
  - Use STRIDE to step through the diagram elements
  - Get specific about threat manifestation

Threat	Property we want
Spoofing	Authentication
Tampering	Integrity
Repudiation	Nonrepudiation
Information Disclosure	Confidentiality
Denial of Service	Availability
Elevation of Privilege	Authorization

Threat	Spoofing
Property	Authentication
Definition	Impersonating something or someone else
Example	Pretending to be any of billg, microsoft.com, or ntdll.dll

Threat	Repudiation
Property	Non-Repudiation
Definition	Claiming to have not performed an action
Example	“I didn’t send that email,” “I didn’t modify that file,” “I certainly didn’t visit that Web site, dear!”

Threat	Denial of Service
Property	Availability
Definition	Deny or degrade service to users
Example	Crashing Windows or a Web site, sending a packet and absorbing seconds of CPU time, or routing packets into a black hole

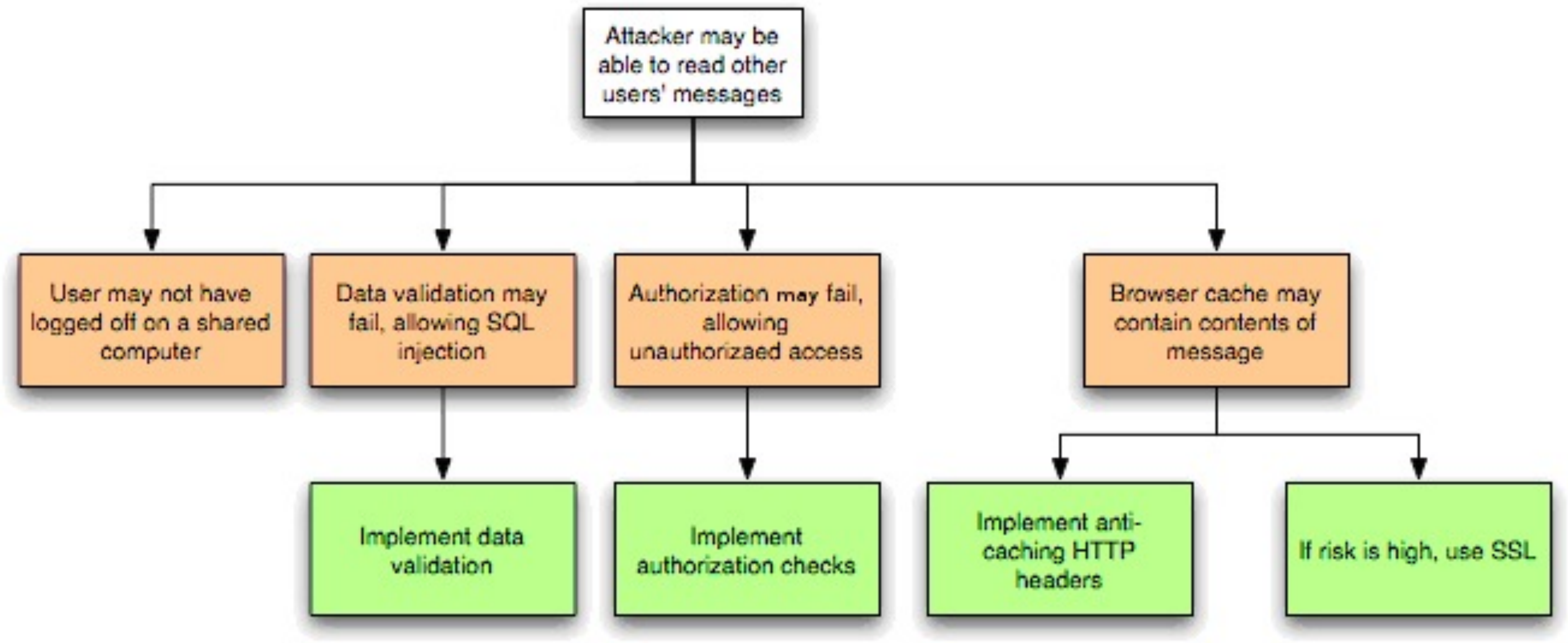
Threat	Tampering
Property	Integrity
Definition	Modifying data or code
Example	Modifying a DLL on disk or DVD, or a packet as it traverses the LAN

Threat	Information Disclosure
Property	Confidentiality
Definition	Exposing information to someone not authorized to see it
Example	Allowing someone to read the Windows source code; publishing a list of customers to a Web site

Threat	Elevation of Privilege (EoP)
Property	Authorization
Definition	Gain capabilities without proper authorization
Example	Allowing a remote Internet user to run commands is the classic example, but going from a “Limited User” to “Admin” is also EoP

Threat Modeling Example - Online Library System Threat Model

9) Understand the threats : Threat Tree



Attacker may be able to read other users' messages

User may not have logged off on a shared computer

Data validation may fail, allowing SQL injection

Authorization may fail, allowing unauthorizaed access

Browser cache may contain contents of message

Implement data validation

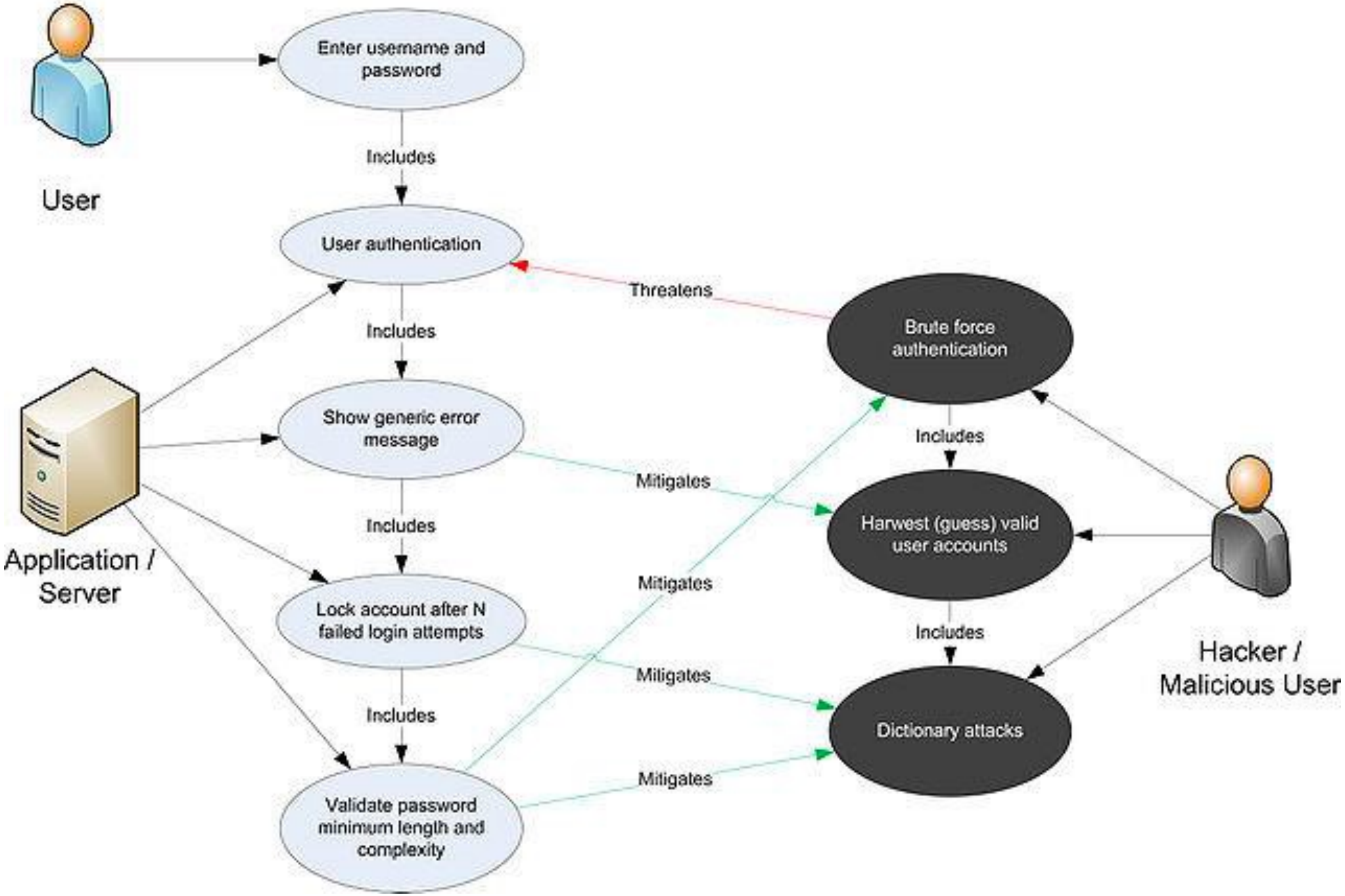
Implement authorization checks

Implement anti-caching HTTP headers

If risk is high, use SSL

Threat Modeling Example - Online Library System Threat Model

Abuse Case



User
Enter username and password
Includes
User authentication
Threatens
Brute force authentication
Includes
Show generic error message
Includes
Harvest (guess) valid user accounts
Mitigates
Application / Server
Includes
Mitigates
Lock account after N failed login attempts
Includes
Hacker / Malicious User
Miligates
Includes
Dictionary attacks
Mitigates
Validate password minimum length and complexity

Threat Modeling Example - Online Library System Threat Model

10) Plan your mitigations

- Authentication
  - ✓ All credentialed users require user name and password required for authentication
  - ✓ All pages check authentication
  - ✓ All credentials communicated only with secure channel
  - ✓ No backdoor accounts or default accounts can be left available
- Authorization
  - ✓ Use role-based authentication with unlimited levels, but including anonymous, user, staff, librarian, admin
  - ✓ All accesses will use least privilege and fail securely
- Cookie Management
- Data/Input Validation
- Error Handling
- Logging/Auditing
- Cryptography
- Secure Code Environment
- Session Management

2. Vulnerability Scanning

- **Vulnerability scanning** is the process of discovering, analyzing, and reporting on security flaws and vulnerabilities.

- **Vulnerability scans** are conducted via **automated vulnerability scanning** tools to identify potential risk exposures and attack vectors across an organization’s networks, hardware, software, and systems.

- The types of vulnerability scanners are:
  - Port Scanner
  - Web Application Vulnerability Scanner
  - Network Vulnerability Scanner
  - Host-based Vulnerability Scanner
  - Database Scanners
  - Source Code Vulnerability Scanner
  - Cloud Vulnerability Scanner

Vulnerability Scanner

- A tool that tell which host is vulnerable to what given a set of vulnerabilities (plugins)

- Original vulnerability scanner
  - It was called SATAN (Security Admin Tool for Analyzing Networks)
  - Written by Dan Farmer in 1995 employed by SGI at the time
  - Very controversial when released
  - It eventually resulted in SGI firing Dan Farmer

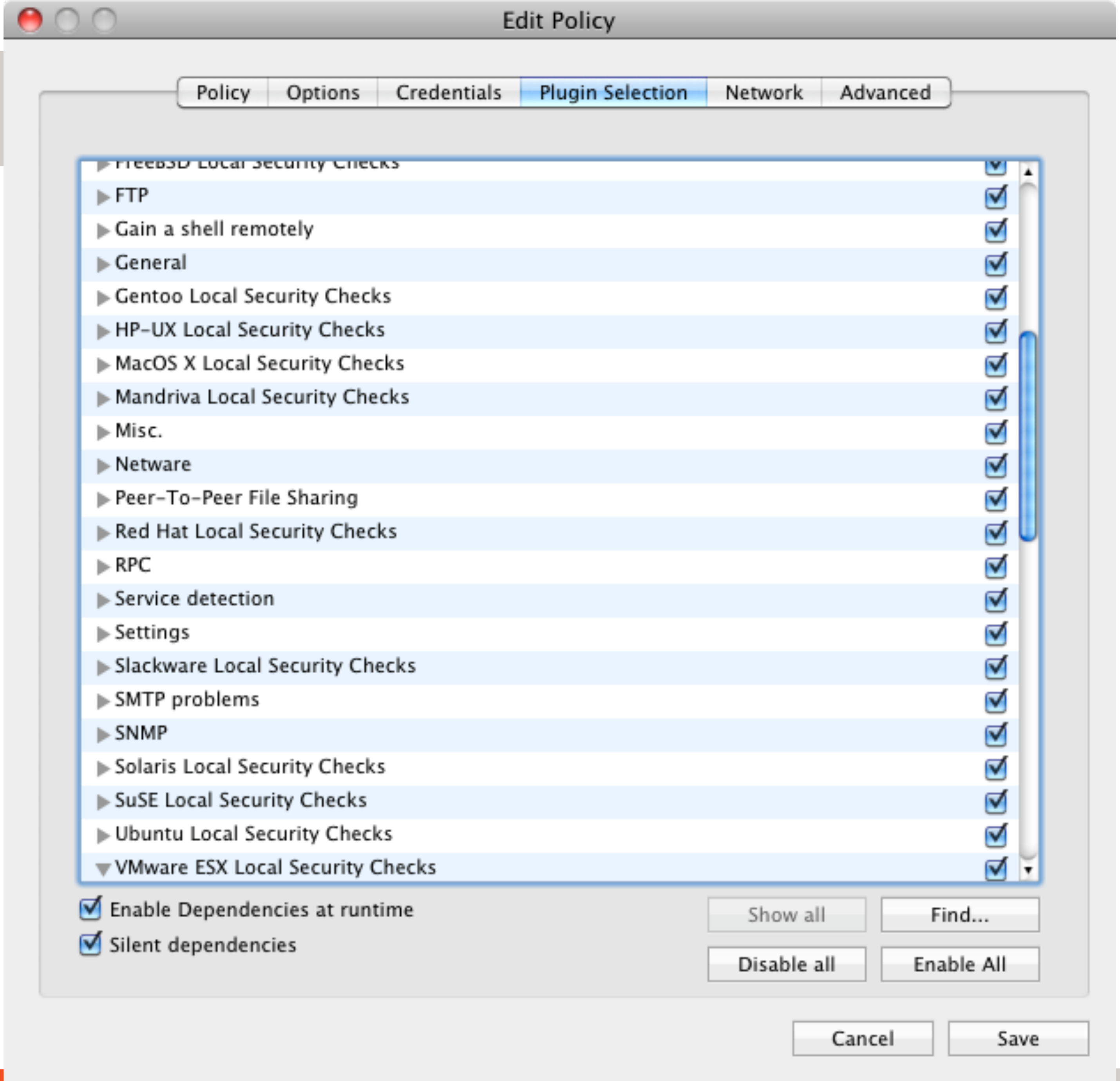
- **Tenable Nessus**
- Qualys Vulnerability Management
- Netsparker
- Amazon Inspector
- Acunetix Vulnerability Scanner
- **SAINT Security Suit**
- Metasploit
- Nmap
- …….

Vulnerability Scanner ------ Nessus

- Nessus project started by Renaud Deraison in 1998
- Very popular vulnerability scanner
- Oct 2005 founded Tenable security and changed to “closed source”
- Still free but with limited signature set
- OPEN-VAS is a fork of the original Nessus code
  - and is still open source at http://www.openvas.org

Nessus Plugin Selection

- Keep an updated list of vulnerabilities (database)
- Know the version of each service
- Is able to test without breaking the service

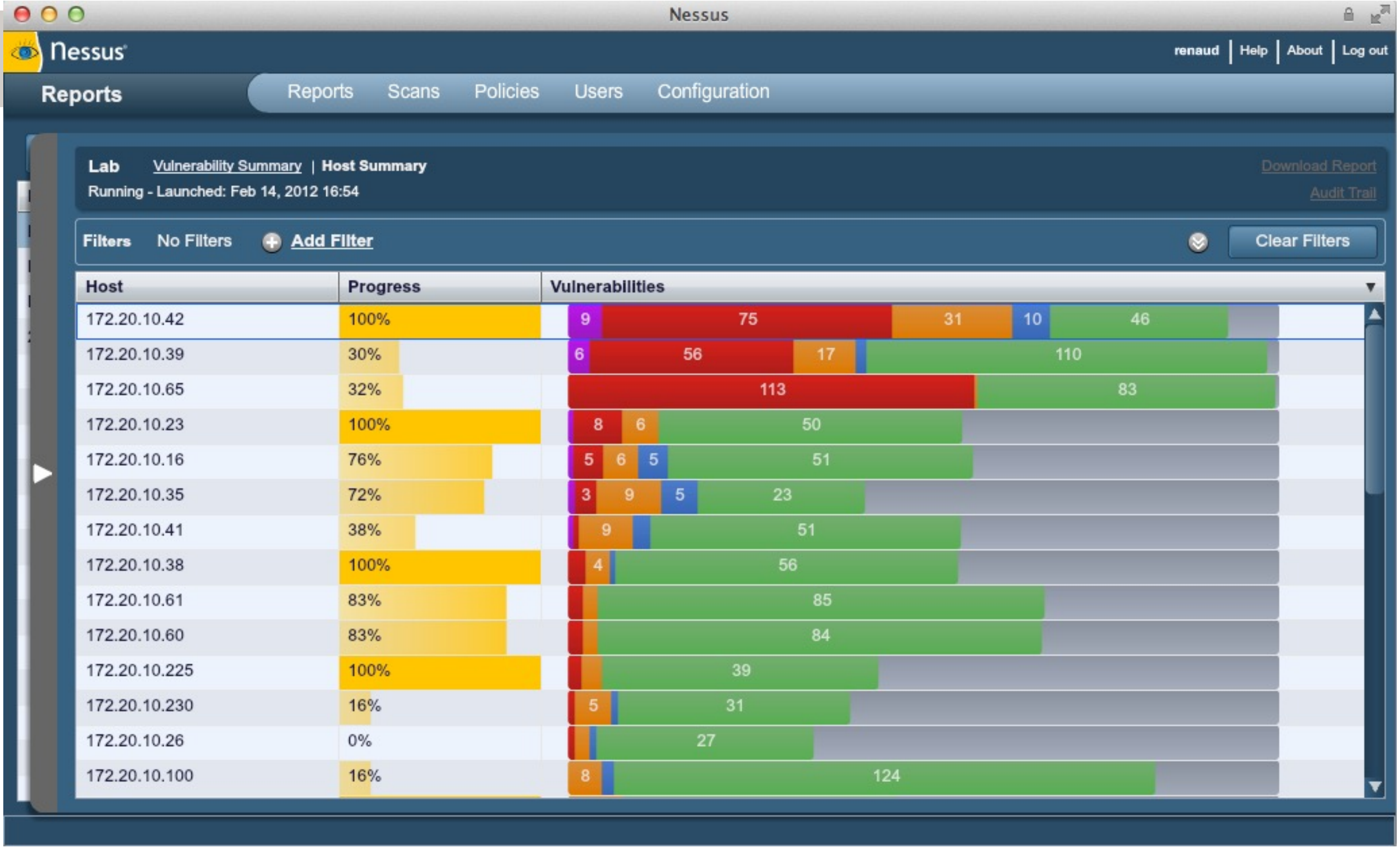


- ▶ FreeBSD Local Security Checks ☑
- ▶ FTP ☑
- ▶ Gain a shell remotely ☑
- ▶ General ☑
- ▶ Gentoo Local Security Checks ☑
- ▶ HP-UX Local Security Checks ☑
- ▶ MacOS X Local Security Checks ☑
- ▶ Mandriva Local Security Checks ☑
- ▶ Misc. ☑
- ▶ Netware ☑
- ▶ Peer-To-Peer File Sharing ☑
- ▶ Red Hat Local Security Checks ☑
- ▶ RPC ☑
- ▶ Service detection ☑
- ▶ Settings ☑
- ▶ Slackware Local Security Checks ☑
- ▶ SMTP problems ☑
- ▶ SNMP ☑
- ▶ Solaris Local Security Checks ☑
- ▶ SuSE Local Security Checks ☑
- ▶ Ubuntu Local Security Checks ☑
- ▼ VMware ESX Local Security Checks ☑

☑ Enable Dependencies at runtime

☑ Silent dependencies

Nessus Scan Results



Nessus
renaud | Help | About | Log out
Reports
Reports Scans Policies Users Configuration
Download Report
Audit Trail
Filters No Filters Add Filter
Clear Filters

Lab Vulnerability Summary | Host Summary

Running - Launched: Feb 14, 2012 16:54

Host	Progress	Vulnerabilities
172.20.10.42	100%	9 75 31 10 46
172.20.10.39	30%	6 56 17 110
172.20.10.65	32%	113 83
172.20.10.23	100%	8 6 50
172.20.10.16	76%	5 6 5 51
172.20.10.35	72%	3 9 5 23
172.20.10.41	38%	9 51
172.20.10.38	100%	4 56
172.20.10.61	83%	85
172.20.10.60	83%	84
172.20.10.225	100%	39
172.20.10.230	16%	5 31
172.20.10.26	0%	27
172.20.10.100	16%	8 124

Software Vulnerability

- A software vulnerability is an instance of a fault in the specification, development, or configuration of software such that its execution can violate the (implicit or explicit) security policy.

- Types of vulnerabilities
  - E.g., Buffer Overflows → The most common form of security vulnerability in the last 10 years
  - SQL Injection
  - Weak password
  - HTTP Trace

Buffer Overflow vulnerability, Exploits & Attacks

- A buffer overflow, or buffer overrun, is a common software coding mistake that an attacker could exploit to gain access to your system.
  - Reading or writing past the end of the buffer → overflow
  - As a result, any data that is allocated near the buffer can be read and potentially modified (overwritten)
    - ✓ A password flag can be modified to log in as someone else.
    - ✓ A return address can be overwritten so that it jumps to arbitrary code that the attacker injected (smash the stack) → attacker can control the host.
  - This vulnerability can cause a system crash or, worse, create an entry point for a cyberattack.
- C and C++ are more susceptible to buffer overflow.

Buffer Overflow Attacks : Two steps

1) Arrange for suitable code to be available in the program’s address space (buffer)
   - Inject the code
   - Use code that is already in the program

2) Overflow the buffer so that the program jumps to that code.

1) Code already in program

- Only need to parameterize the code and cause the program to jump to it.

- Example:

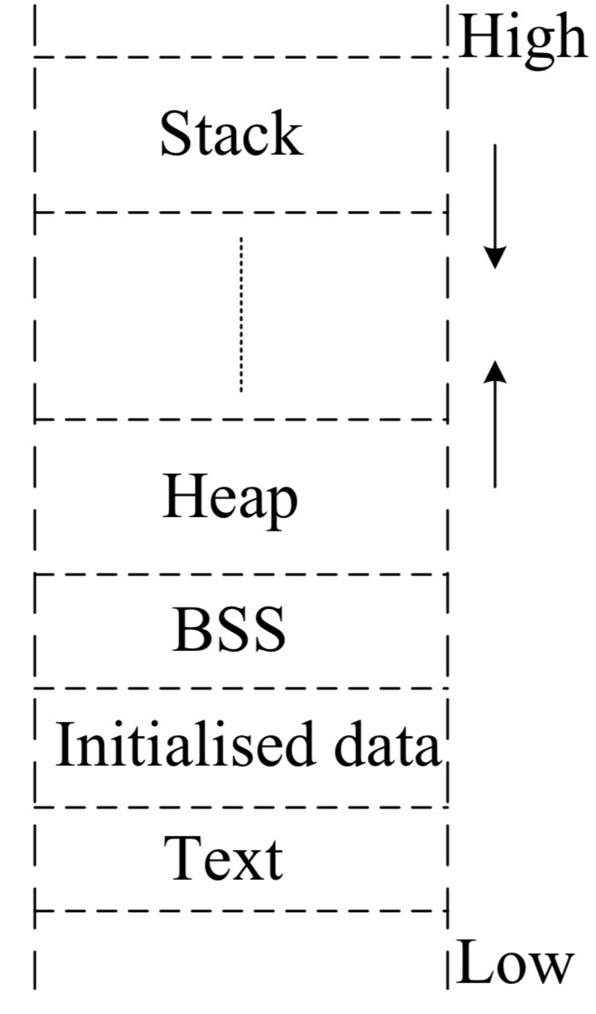
  Code in libc that executes “exec(arg)”, where arg is a string pointer argument, can be used to point to “/bin/sh” and jump to appropriate instructions in libc library.

Inject the code

- Use a string as input to the program which is then stored in a buffer.
- String contains bytes that are native CPU instructions for attacked platform.

- Buffer can be located on the stack, heap, or in static data area.

2） Jump to attack code ---- Memory regions



1. Code/text segment

- Static / Contains code (instructions) and read-only data
- Corresponds to text section of executable file
- If attempt to write to this region → segmentation violation

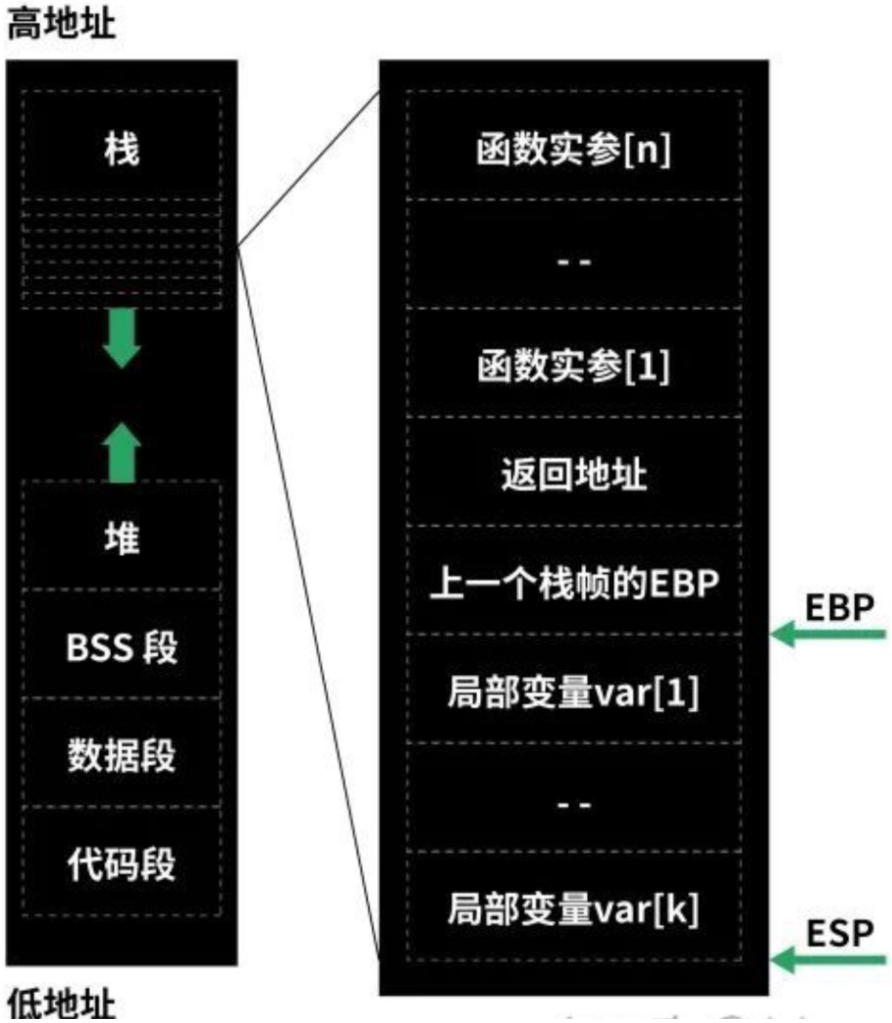
2. Data segment

- Permanent data with statically known size
- Both initiated and uninitiated variables
- Corresponds to the data-bss sections of the executable file （block started by symbol）

3. Heap

- Dynamic memory allocation
- malloc() in C and new in C++ → More flexibility
- More stable data storage – memory allocated in the heap remains in existence for the duration of a program

Memory regions



4. Stack

- Provides high-level abstraction
  - Allocates local variables when a function gets called (with known lifetime)
  - Passes parameters to functions
  - Returns values from functions

- Push/Pop operations (LIFO) – implemented by CPU

- Extended Stack Pointer (ESP) – TOP of stack (or next free available address)
- Fixed address – BOTTOM of stack
- Logical Stack Frame (SF) – contains parameters to functions, local variables, data to recover previous SF (e.g: instruction pointer at time of function call)
- Frame Pointer (FP)/Extended Base Pointer (EBP) Beginning of Activation Record (AR), used for referencing local variables and parameters (accessed as offsets from BP)

Accessing an activation record

- Contains all info local to a single invocation of a procedure
  - Return address
  - Arguments
  - Return value
  - Local variables
  - Temp data
  - Other control info

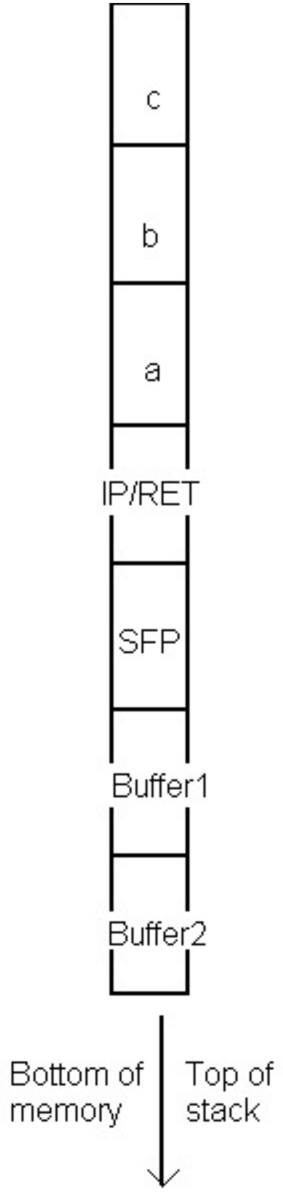
- When a procedure is called
  - Previous SF is saved
  - ESP is copied into EBP → new EBP
  - ESP advances to reserve space for local variables
  - Upon procedure exit, the stack is cleaned up

Buffer Overflow Example

```
void function(int a, int b, int c) {
   char buffer1[5];
   char buffer2[10];
}
void main() {
  function(1,2,3);
}
```

```
pushl $3
pushl $2
pushl $1
call function

pushl %ebp
movl %esp,%ebp
subl $20,%esp
```



Buffer overflow example

```
void main() {
  int x;

  x = 0;
  function(1,2,3);
  x = 1;
  printf("%d\n",x);
}
```

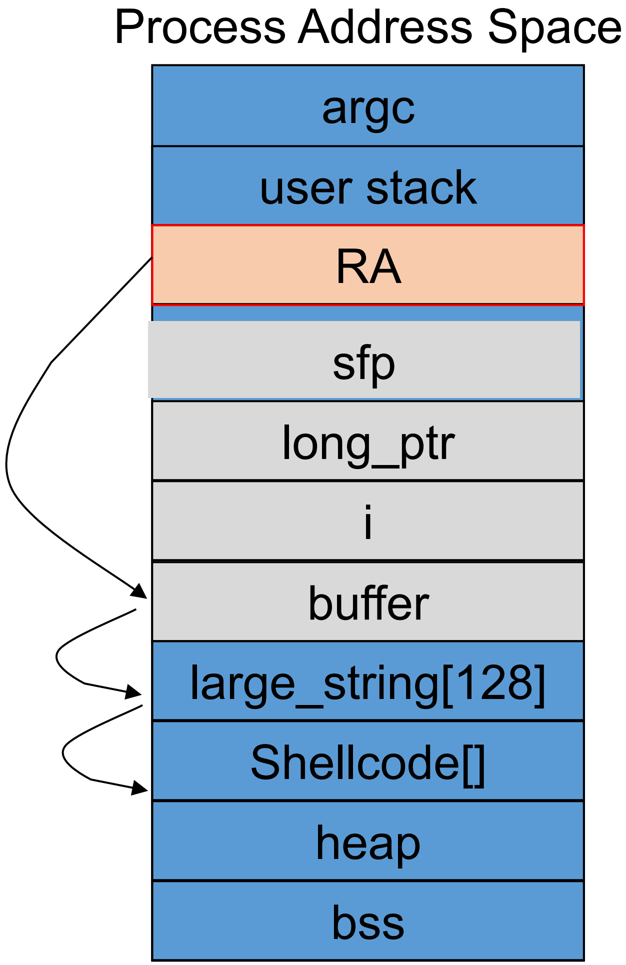
```
void function(int a, int b,
  int c) {
   char buffer1[5];
   char buffer2[10];
   int *ret;

   ret = buffer1 + 12;
   (*ret) += 8; }
```

- Output: 0
- Return address has been modified and the flow of execution has been changed
- All we need to do is place the code that we are trying to execute in the buffer we are overflowing, and modify the return address so it points back to buffer

Buffer overflow example

```
char shellcode[ ] = “\xeb\x1f\x5e\x89\x76\x08\x31\xc0\x88\x46\x07\x89\x46\x0c\xb0\x0b”
    “\x89\xf3\x8d\x4e\x08\x8d\x56\x0c\xcd\x80\x31\xdb\x89\xd8\x40\xcd”
    “x80\xe8\xdc\xff\xff\xff/bin/sh”;
char large_string[128];
void main() {
    char buffer[96];
    int i;
    long *long_ptr = (long *) large_string; /* long_ptr takes the address of large_string */
    /* large_string are filled with the address of buffer */
    for (i = 0; i < 32; i++)
            *(long_ptr + i) = (int) buffer;
    /* copy the contents of shellcode into large_string */
    for (i = 0; i < strlen(shellcode); i++)
            large_string[ i ] = shellcode[ i ];
    /* buffer gets the shellcode and 32 pointers back to itself */
    strcpy(buffer, large_string); }
```



Process Address Space

argc

user stack

RA

sfp

long_ptr

i

buffer

large_string[128]

Shellcode[]

heap

bss

3） Buffer overflows defenses

- Writing correct code (good programming practices)
- Debugging Tools
- Non-executable buffers
- Array bounds checking
- Code pointer integrity checking (e.g., StackGuard)

Problems with C

- Some C functions are problematic
  - Static size buffers
  - Do not have built-in bounds checking

- While loops
  - Read one character at a time from user input until end of line or end of file
  - No explicit checks for overflows

Some problematic C functions

Function	Severity	Solution: Use
gets	Most Risky	fgets(buf, size, stdin)
strcpy, strcat	Very Risky	strncpy, strncat
sprintf, vsprintf	Very Risky	snprintf, vsnprintf or precision specifiers
scanf family	Very Risky	precision specifiers or do own parsing
realpath, syslog	Very Risky (depending on implementation)	Maxpathlen and manual checks
getopt, getopt_long, getpass	Very Risky (depending on implementation)	Truncate string inputs to reasonable size

Good programming practices – I
(useful to know for code inspections)

DO NOT USE:	Instead USE:

void main( ) {
char buf [40];
gets(buf);
}
	void main( ) {
	char buf [40];
	fgets(buf,40,stdin);
	}

Good programming practices – II

DO NOT USE:	Instead USE:

void main() {
char buf[4];
char src[8] = "rrrrr";
strcpy(buf,src);
}
	if (src_size >= buf_size) {
	cout<< "error";
	return(1);
	}
	else {
	strcpy(buf,src);
	}
	OR
	strncpy(buf,src,buf_size - 1);
	buf[buf_size - 1] = '\0';

Debugging tools

- More advanced debugging tools
  - Fault injection tools – inject deliberate buffer overflow faults at random to search for vulnerabilities
  - Static analysis tools – detect overflows

- Can only minimize the number of overflow vulnerabilities but cannot provide total assurance

Non-executable buffers

- Make data segment of program’s address space non-executable → attacker can’t execute code injected into input buffer (compromise between security and compatibility)

- If code already in program, attacks can bypass this defense method
- Kernel patches (Linux and Solaris) – make stack segment non-executable and preserve most program compatibility

Array bounds checking

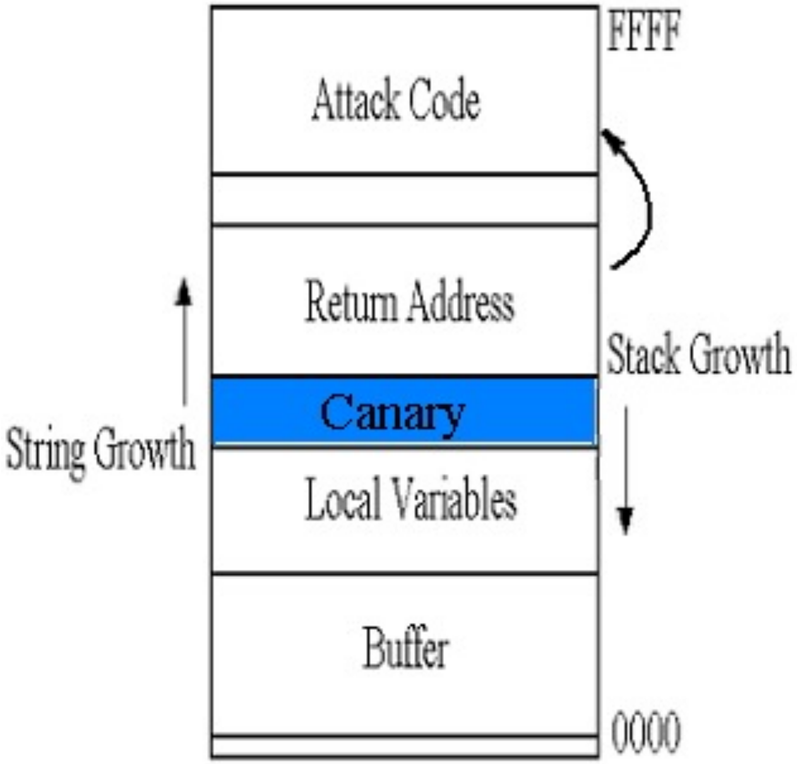
- Attempts to prevent overflow of code pointers
- All reads and writes to arrays need to be checked to make sure they are within bounds (check most array references)
  - Campaq C compiler
  - Jones & Kelly array bound checking
  - Purify memory access checking
  - Type-safe languages (e.g., Java)

Code pointer integrity checking

- Attempts to detect that a code pointer has been corrupted before it is de-referenced
- Overflows that affect program state components other than code pointer will succeed
- Offers advantages in performance, compatibility with existing code and implementation effort
  - Hand-coded stack introspection
  - StackGuard → PointGuard

StackGuard

- Compiler technique that provides protection by checking the return address in AR
- When detects an attack → causes app to exit, rather than yielding control to attacker
    - Terminator canary
    - Random canary



3. Penetration testing

- Penetration testing, or pen testing, is a threat assessment strategy that involves simulating real attacks to evaluate the risks associated with potential security breaches.

- It is a simulated cyberattack against your computer system to uncover potential vulnerabilities that could hamper the security of your system.

- Sometimes called ethical hacking, pen testing is intended to seek out exploitable vulnerabilities against an organization's security infrastructure.

Penetration testing

- External vs. Internal
  - Penetration Testing can be performed from the viewpoint of an external attacker or a malicious employee.

- Overt vs. Covert
  - Penetration Testing can be performed with or without the knowledge of the IT department of the company being tested.

Difference between Penetration Test and Vulnerability Scan

- Strategy
  - Vulnerability assessment checks for known weaknesses in a system and generates a report on risk exposure
  - Pen testing is meant to exploit weaknesses on a system or an entire IT infrastructure to uncover any threats to the system.

- Scope
  - Pen testing not only involves discovering vulnerabilities that could be used by attackers but also exploiting those vulnerabilities to assess what attackers can exploit after a breach. So, vulnerability assessment is one of the essential prerequisites for doing a pen test.

- Approach
  - A vulnerability assessment is an automated process performed with the help of automated tools to scan for new and existing threats that can harm your system.
  - Pen testing requires a well-planned, methodological approach and is performed by experienced individuals who understand all the facets of security posture.

Phases of Penetration Testing

- - Reconnaissance and Information Gathering
- - Network Enumeration and Scanning
- - Vulnerability Testing and Exploitation
- - Reporting