Analysis of Security Vulnerabilities

R. Iyer, Z Kalbarczyk, J. Xu, S. Chen

Center for Reliable and High-PerformanceComputing
Coordinated Science Laboratory
University of Illinois at Urbana-Champaign

http://www.crhc.uiuc.edu/DEPEND

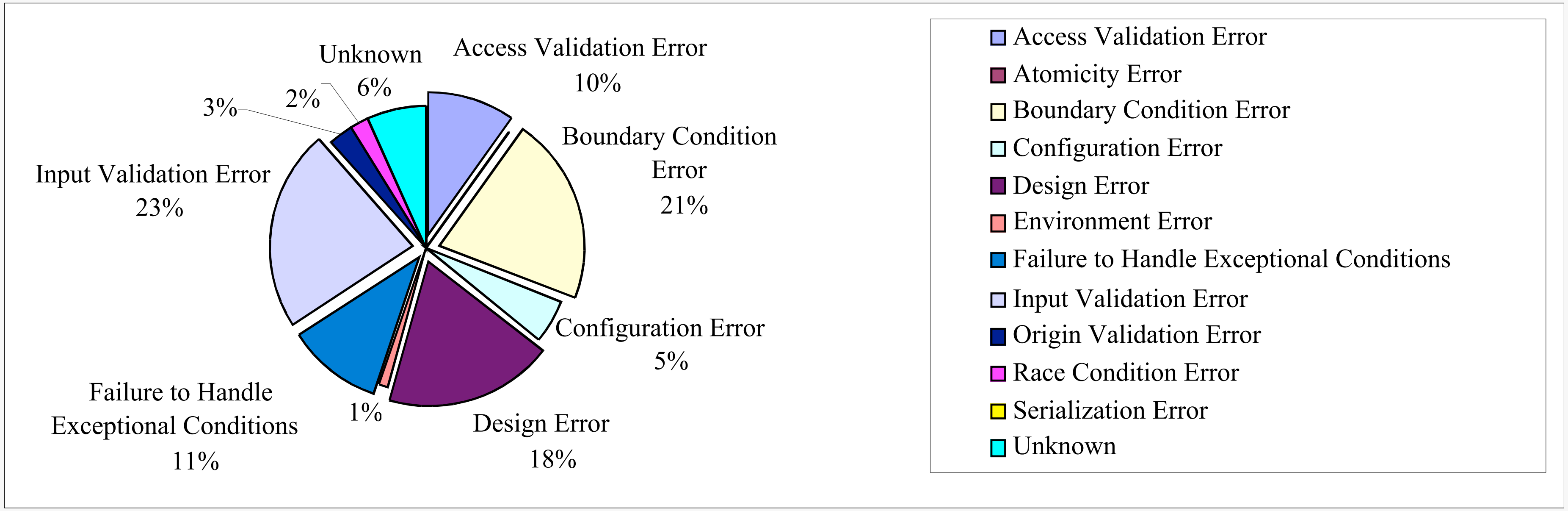
Approach

- Analyze data on security attacks to:
  - identify current vulnerabilities and to classify the attacks according to attack causes
  - understand potential inconsistencies in application/system specifications resulting in security vulnerabilities of an actual application/system implementation
- Generate measurement-based security attacks models depicting the attack process
- Investigate and propose software (e.g., compiler-based) and hardware (e.g., processor embedded) intrusion detection/prevention techniques

A Finite State Machine Methodology for Analyzing Security Vulnerabilities

- Used the *Bugtraq* database and application source code to analyze reported vulnerabilities.
- Developed finite state machine (FSM) models to depict the vulnerabilities and associated exploits.
- Only three primitive FSMs (pFSM) are required to describe at least 22% (out of 6000) of *Bugtraq* vulnerabilities.
- Discovered a new remotely exploitable heap overflow vulnerability, which is now published in *Bugtraq*.

Breakdown of Vulnerabilities (*Bugtraq*)



- *Access Validation Error* : an operation on an object outside its access domain.
- *Atomicity Error* : code terminated with data only partially modified as part of a defined operation.
- *Boundary Condition Error* : an overflow of a static -sized data structure: a classic buffer overflow condition.
- *Configuration Error* : a system utility installed with incorrect setup parameters.
- *Environment Error* : an interaction in a specific environment between functionally correct modules.
- *Failure to Handle Exceptional Conditions* : system failure to handle an exceptional condition generated by a functional module, device, or user input.
- *Input Validation Error* : failure to recognize syntactically incorrect input.
- *Race Condition Error* : an error during a timing window between two operations.
- *Serialization Error* : inadequate or improper serialization of operations.
- *Design Error* and, *Origin Validation Error* : Not defined.

Bugtraq database included 5925 reports on software related vulnerabilities
(as of Nov.30 2002)

Vulnerability Distributions Across Operating Systems

	Total Number of Vulnerabilities	Access Validation Error	Boundary Condition Error	Failure to Handle Exceptional Conditions	Input Validation Error	Design Error
RedHat Linux	822	10%	27%	7%	22%	15%
Windows	1856	9%	23%	15%	19%	23%
Solaris	453	10%	35%	5%	18%	11%

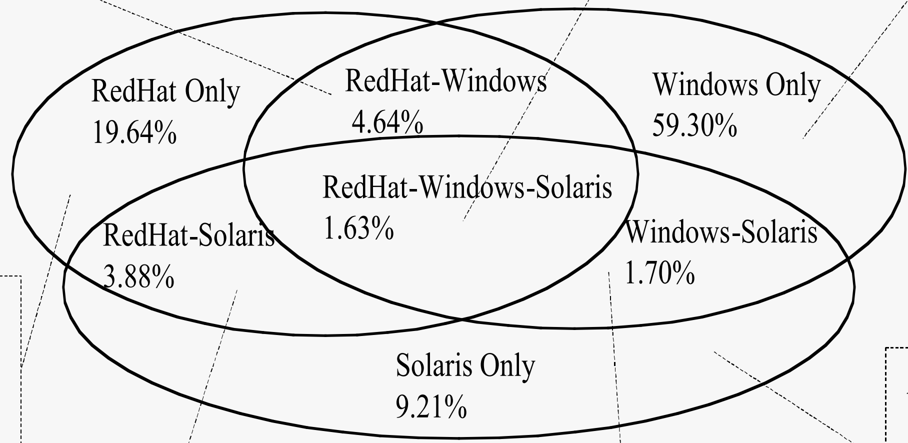
- Locations of observed vulnerabilities
  - Majority of the vulnerabilities occurred in the executing applications rather than in libraries or the OS kernels:
  - 78.9% for RedHat Linux (all versions), 77.3% for Windows 2000, and 90.5% for Solaris 2.6, i.e., between 10% and 22% of reported vulnerabilities are present in the underlying operating systems

Common Vulnerabilities on Multiple Operating Systems

Access Validation Errors
Netscape
Boundary Condition Errors
Netscape
Design Errors
Netscape
Failure to Handle Exceptional Conditions
Netscape
Input Validation Errors
Netscape

Boundary Condition Errors
Apache, Netscape, Realplayer
Input Validation Errors
Apache

Access Validation Errors
Office, IE,IIS,SQL Server
Boundary Condition Errors
Adobe, AOL Messenger, IE, IIS, Office, SQL Server, TELNETd, Realplayer
Design Errors
Adobe, AOL Messenger, IE, IIS, Office, Apache, Netscape
Failure to Handle Exceptional Conditions
AOL Messenger, IE, IIS, Office, TELNETd
Input Validation Errors
AOL Messenger, Apache, IE, IIS, Office, SQL Server, Netscape



Access Validation Errors
Apache, man
Boundary Condition Errors
Man, FTPd, openSSH, RPM, Sendmail, TCPDUMP
Design Errors
Apache
Failure to Handle Exceptional Conditions
Apache, Sendmail, IPTables, openSSH
Input Validation Errors
Sendmail, Libc, TELNETd, RPC, Netscape

Access Validation Errors
Adobe, Apache, openSSH
Boundary Condition Errors
Apache, TELNETd, openSSH
Design Errors
openSSH
Input Validation Errors
Apache, openSSH

Input Validation Errors
Netscape

Access Validation Errors
Sendmail, RPC
Boundary Condition Errors
rlogin, PRC
Design Errors
RPC
Failure to Handle Exceptional Conditions
RPC
Input Validation Errors
Netscape

Abbreviations:
IE – Internet Explorer; IIS – Internet Information Services; RedHat – RedHat Linux

Common Vulnerabilities on Multiple Operating Systems

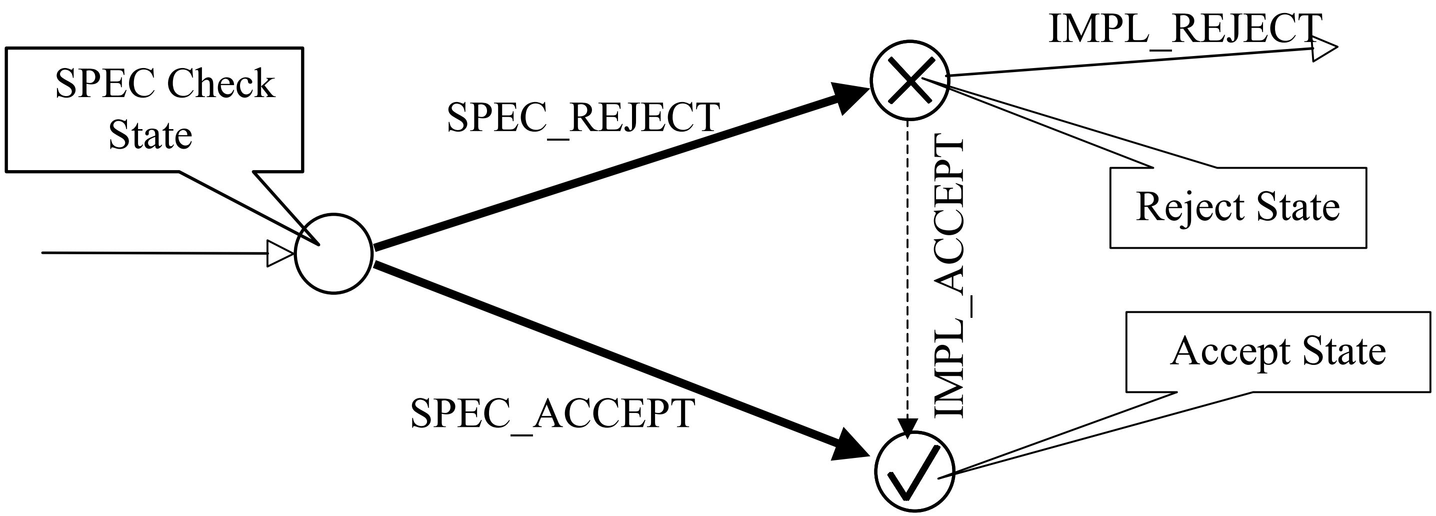
- Solaris is least vulnerable: Solaris applications and the applications that overlap between
  Solaris and other platforms contribute the smallest fraction
- RedHat Linux is second in terms of both its own contribution (OS and applications) and the overlapping applications.
- Windows (and Windows-exclusive applications) contributes nearly 60% of the reported vulnerabilities.
  - The overlap percentage between Windows and other applications is also the largest.

Observations from Vulnerability Analysis

- Exploiting a vulnerability involves multiple vulnerable **operations** on several objects.
- Exploits must pass through multiple **elementary activities**, each providing an opportunity for performing a security check.
- For each elementary activity, the vulnerability data and corresponding code inspections allow us to define a **predicate**, which if violated, naturally results in a security vulnerability.

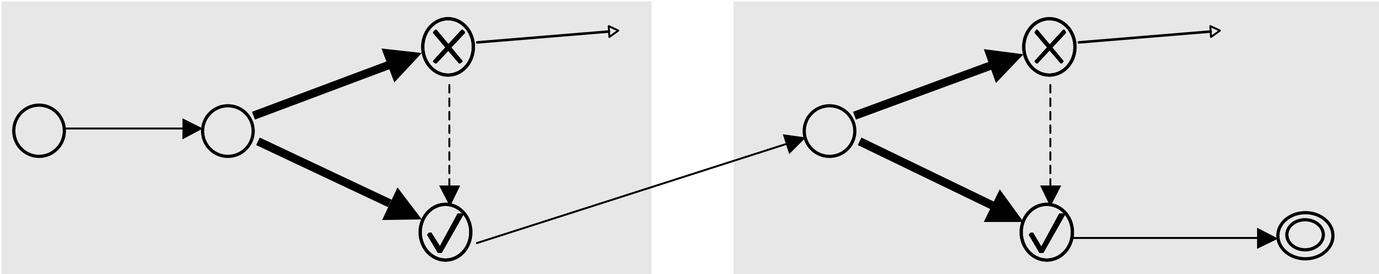
Primitive FSM

- We define <u>*Primitive FSM (pFSM)*</u> to depict an elementary activity, which specifies a predicate (SPEC) that should be guaranteed in order to ensure security.



Sendmail Debugging Function Signed Integer Overflow (Bugtraq #3163)

Op 1: Write an integer to an array location

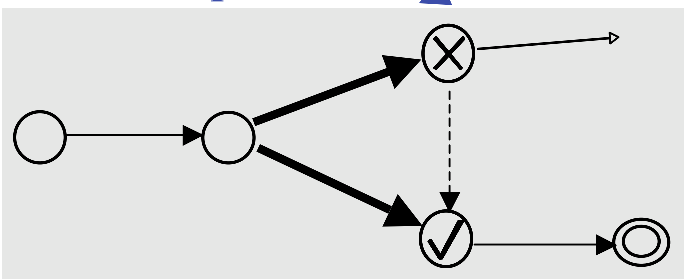


Elementary activity 1

Elementary activity 2

A function pointer can be overwritten

Op 2: Manipulate the function pointer



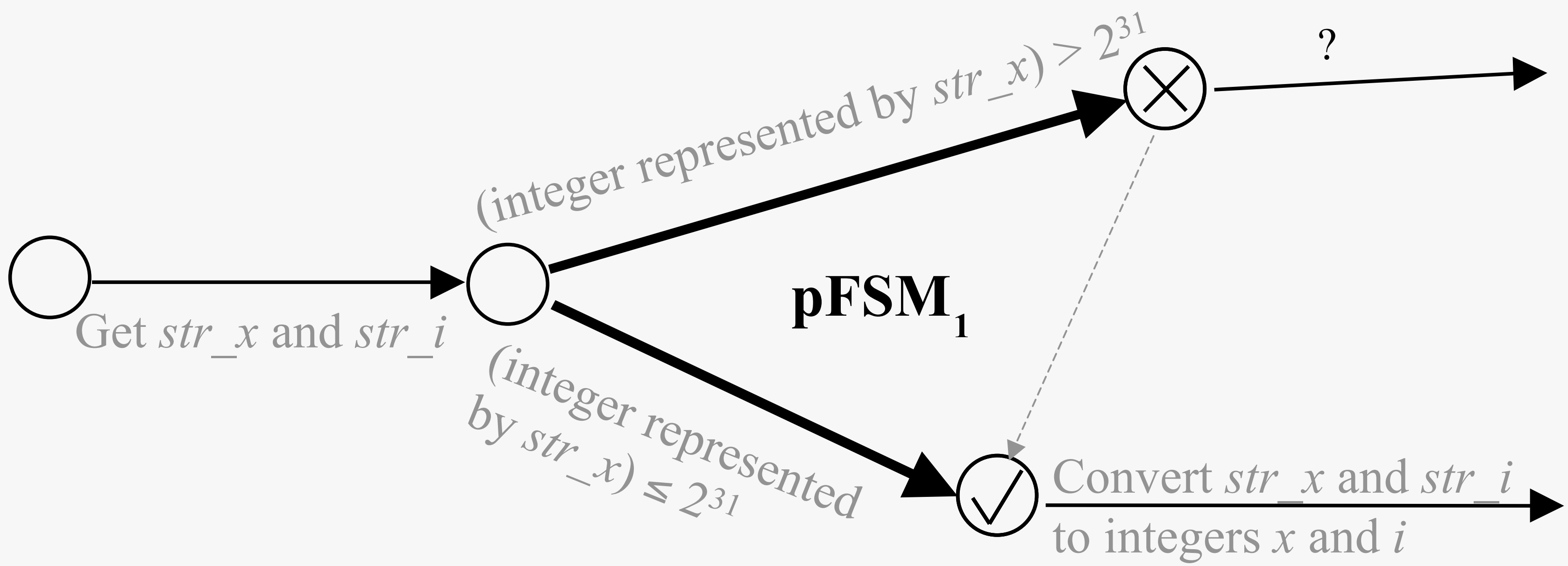
Elementary activity 3

Attacker's malicious code is executed

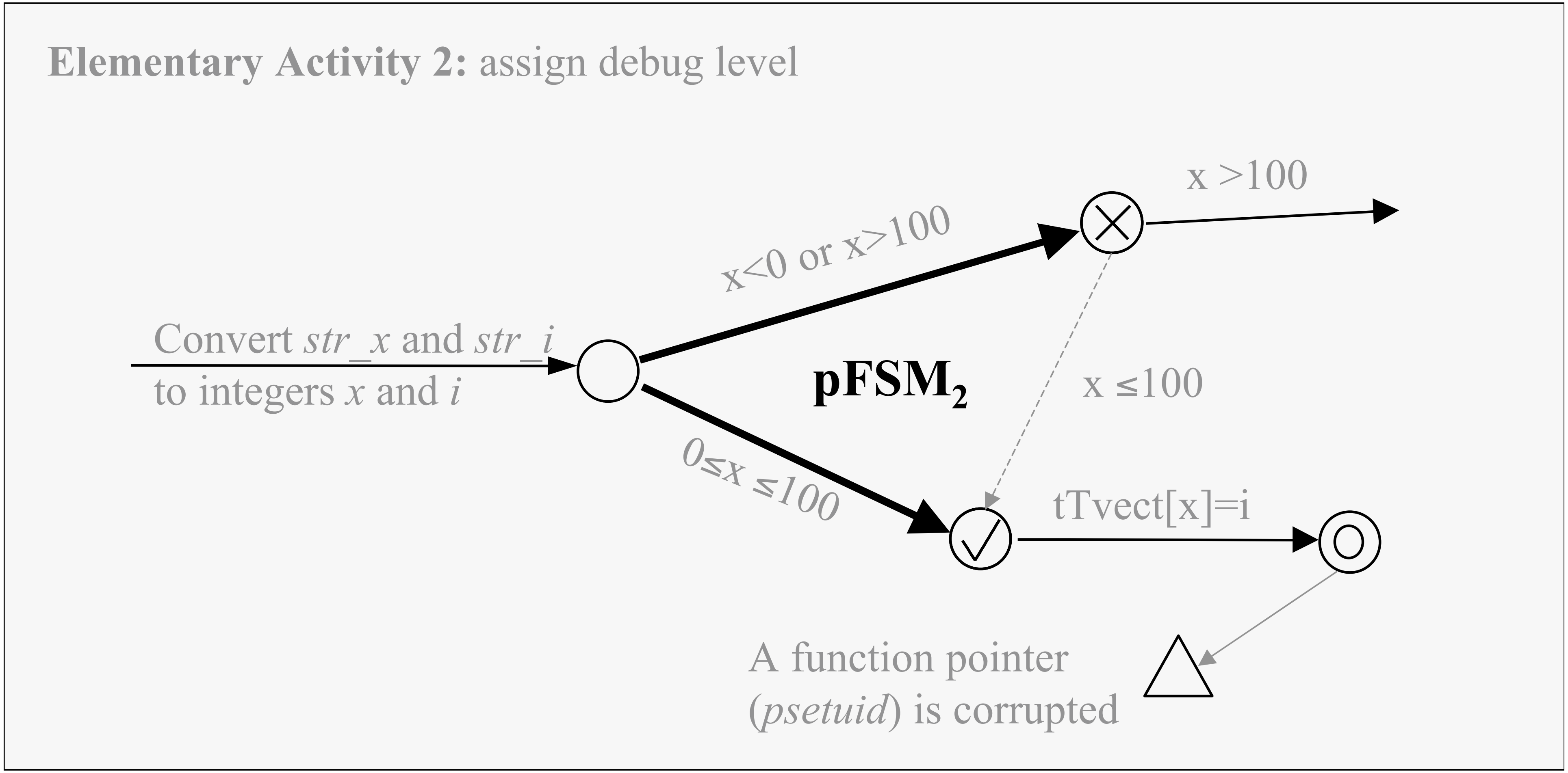
Elementary Activity 1 of *Sendmail* Vulnerability

Elementary Activity 1: get user input

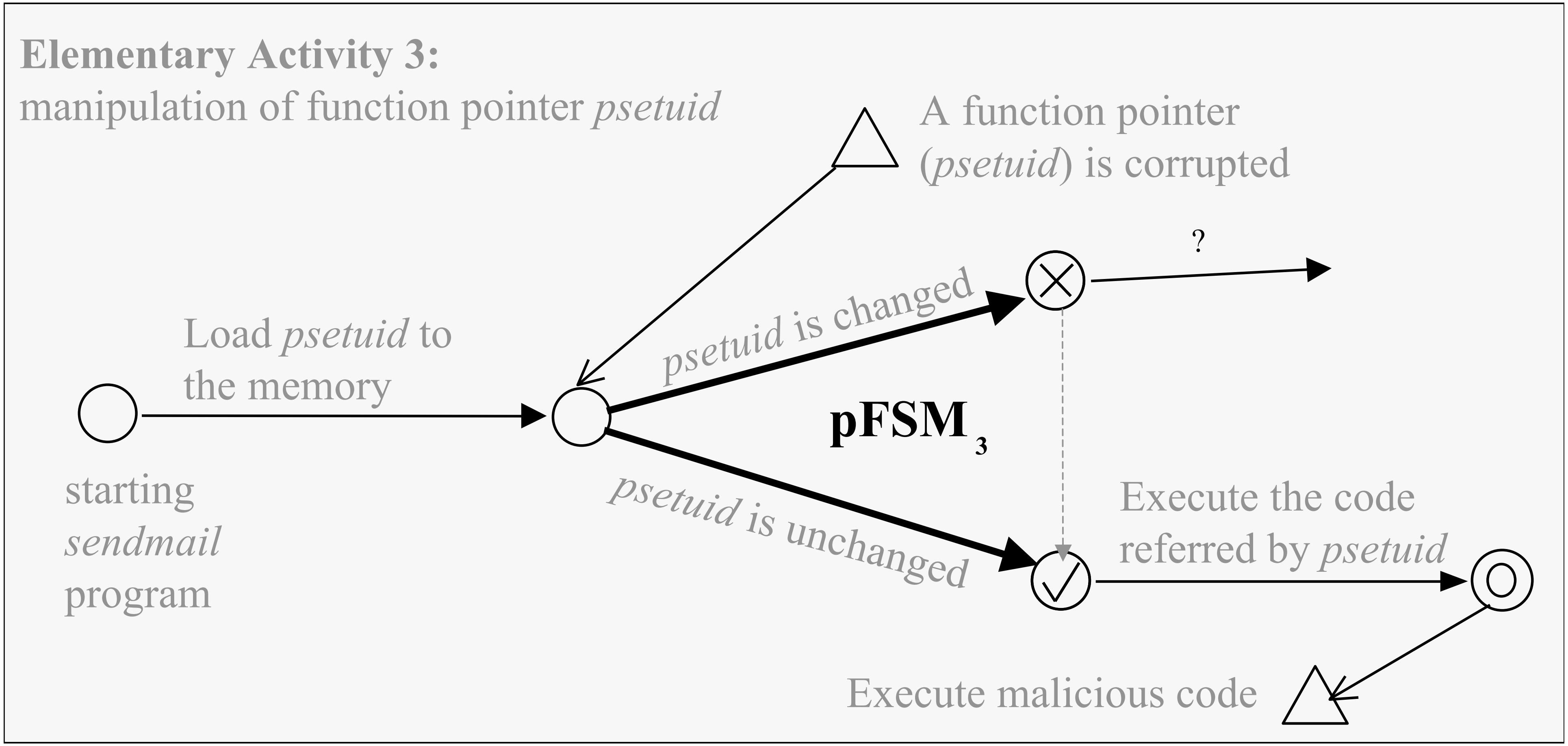
Get strings str_x and str_i, convert them to integers x and i



Elementary Activity 2 of *Sendmail* Vulnerability



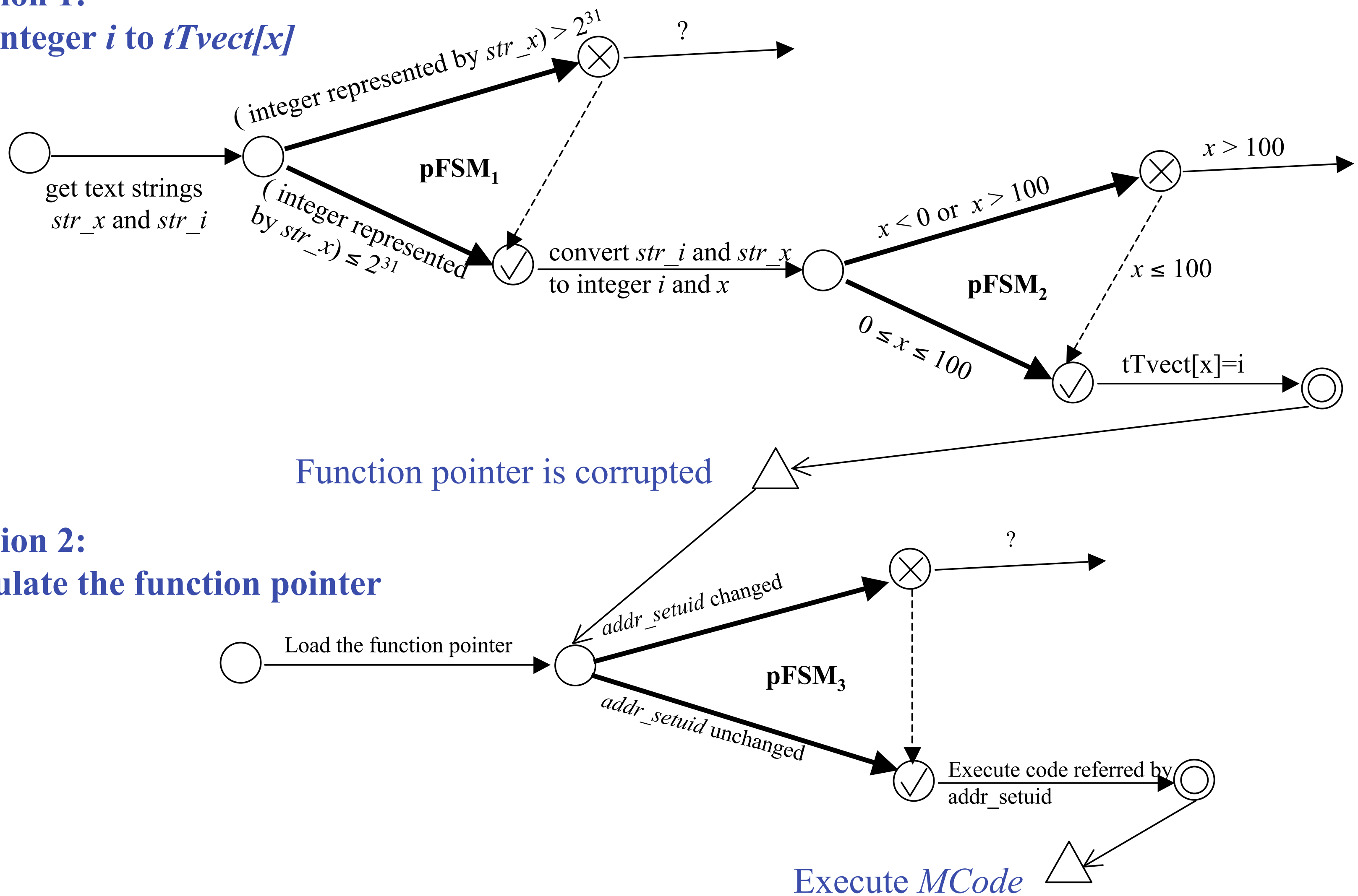
Elementary Activity 3 of *Sendmail* Vulnerability



Summarizing the FSM Model of the *Sendmail* Vulnerability

Operation 1:
Write integer *i* to *tTvect[x]*

Operation 2:
Manipulate the function pointer

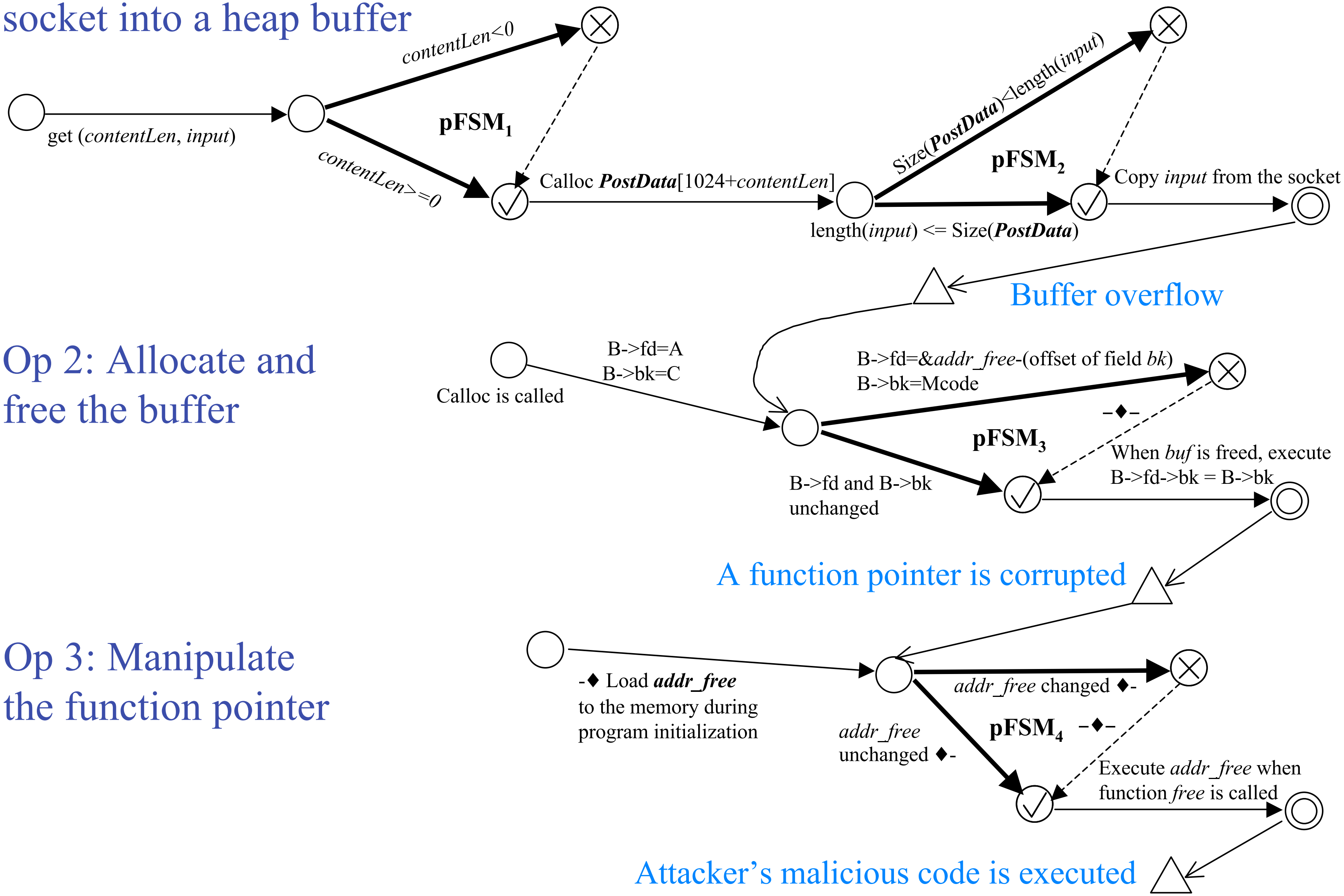


NULL HTTPD Heap Overflow Vulnerabilities (Bugtraq #5774, #6255)

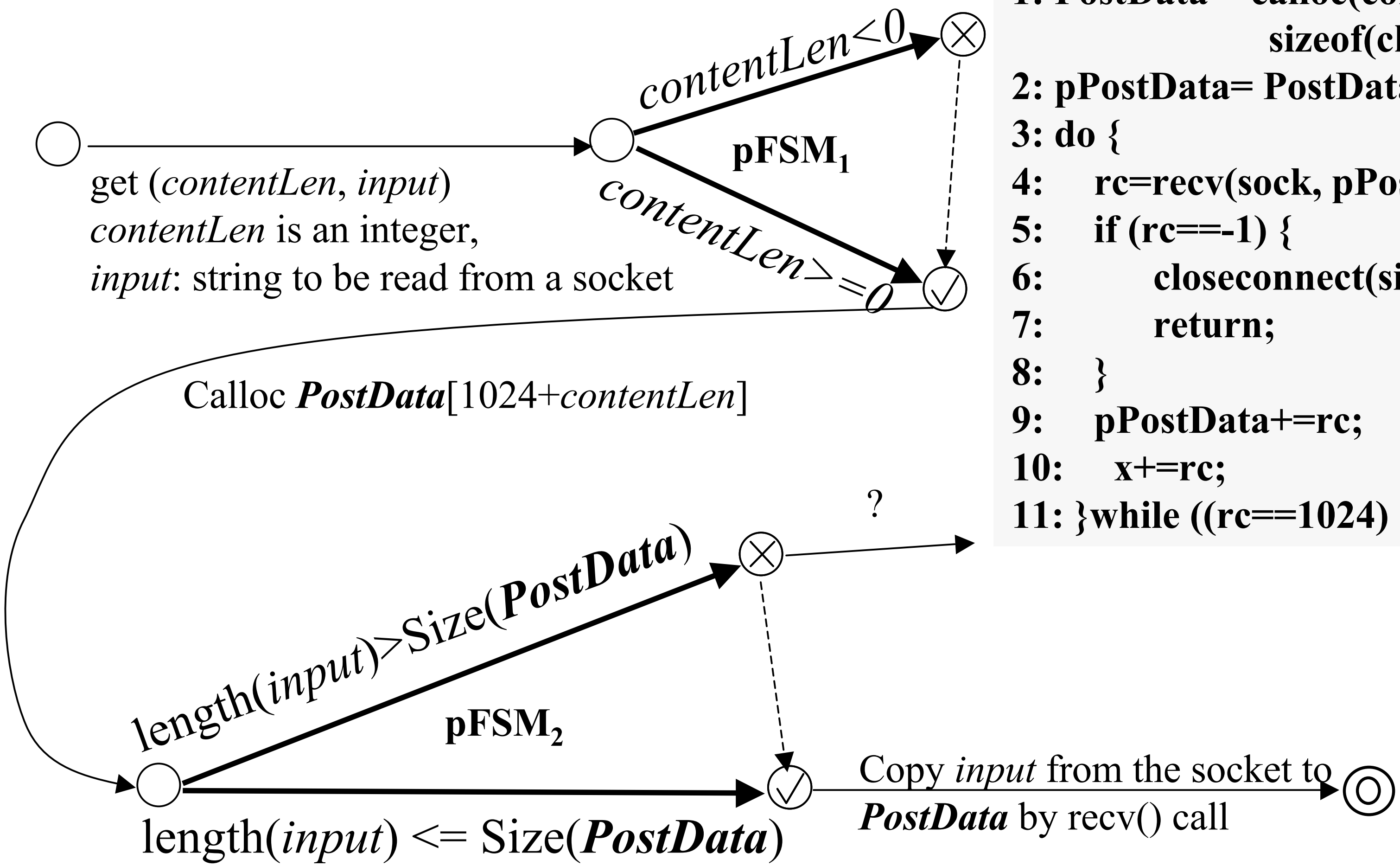
Op 1: Read user input from a socket into a heap buffer

Op 2: Allocate and free the buffer

Op 3: Manipulate the function pointer



Operation 1 of NULL HTTPD: Read postdata from socket to an allocated buffer *PostData*




```
Get contentLen   //Can be negative
PostData = calloc(contentLen  +1024,
                  sizeof(char));x=0; rc=0;
pPostData= PostData;
do {
    rc=recv(sock, pPostData, 1024, 0);
    if (rc==-1) {
        closeconnect(sid,1);
        return;
    }
    pPostData+=rc;
    x+=rc;
}while ((rc==1024) || (x<contentLen));
```

Modeled Vulnerabilities

- Signed Integer Overflow
- Heap Overflow
- Stack Overflow
- Format String Vulnerabilities
- File Race Conditions
- Some Input Validation Vulnerabilities

Common pFSM Types

- ***Object Type Check.*** to verify whether the input object is of the type that the operation is defined on.
- ***Content and Attribute Check.*** to verify whether the contents and the attributes of the object meet the security guarantee.
- ***Reference Consistency Check.*** to verify whether the binding between an object and its reference is preserved from the time when the object is checked to the time when the operation is applied on the object.

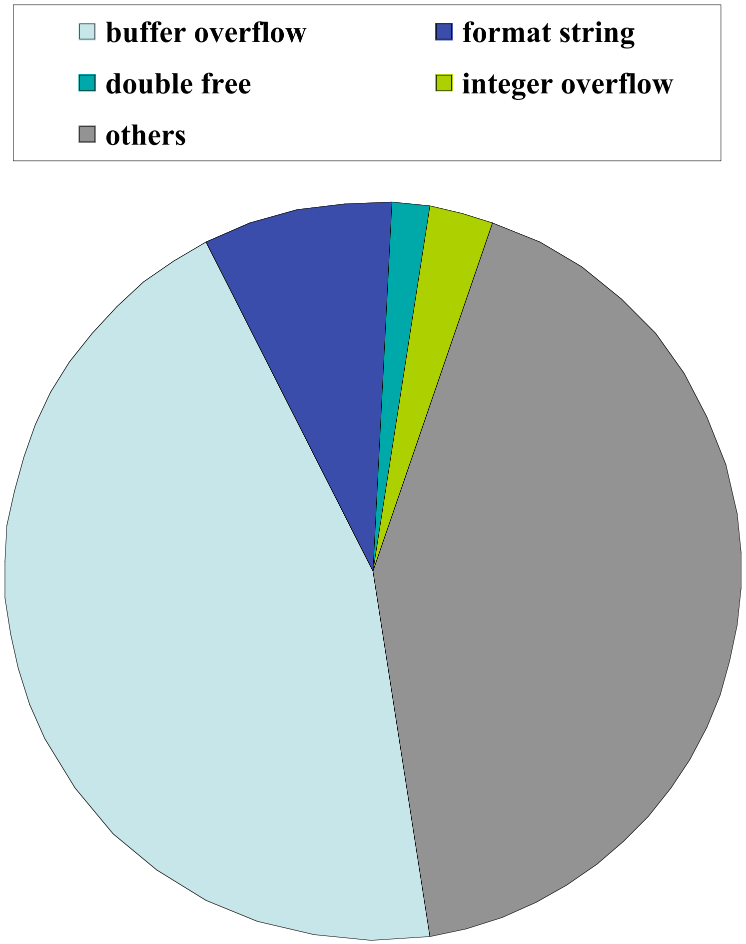
Common pFSM Types (cont.)

Type of pFSM / Example Vulnerabilities	Object Type check	Content and Attribute Check	Reference Consistency check
Sendmail Signed Integer Overflow	**$pFSM_1$**	**$pFSM_2$**	**$pFSM_3$**
NULL HTTPD Heap Overflow		**$pFSM_1$** **$pFSM_2$**	**$pFSM_3$** **$pFSM_4$**
Rwall File Corruption	**$pFSM_2$**	**$pFSM_1$**	
IIS Filename Decoding Vulnerability		**$pFSM_1$**	
Xterm File Race Condition		**$pFSM_1$**	**$pFSM_2$**
GHTTPD Buffer overflow on Stack		**$pFSM_1$**	**$pFSM_2$**
rpc.statd format string vulnerability		**$pFSM_1$**	**$pFSM_2$**

Lessons Learned

- Conclusions
  - Extracted common characteristics of security vulnerabilities
  - Based on the characteristics, developed an FSM methodology to model vulnerabilities.
  - Only three pFSM types were required. Force rigorous reasoning. Indicate opportunities of security check.
- Future Directions
  - Automatically specify certain predicates in the FSMs
  - Runtime checks or formal proofs of the predicates.

Motivation

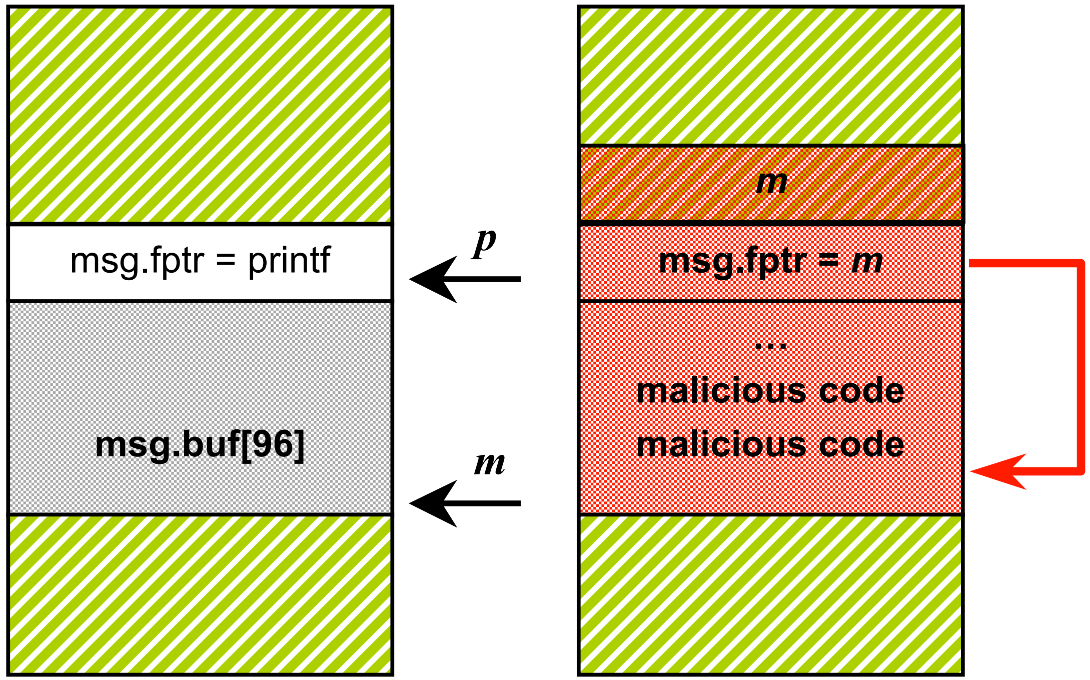


- 60% of all CERT security advisories (1999-2002)
  - Buffer overflow, format string, double free, integer overflow
- Observation:
  - Customized solutions exist for some subclasses vulnerabilities
  - Generic techniques needed for masking broad-range of vulnerabilities

How Attacks Work?

- Two conditions for a successful attack
  - Injecting malicious code/data at address *m* in app. memory
  - Changing control data at address *p* to point to *m*

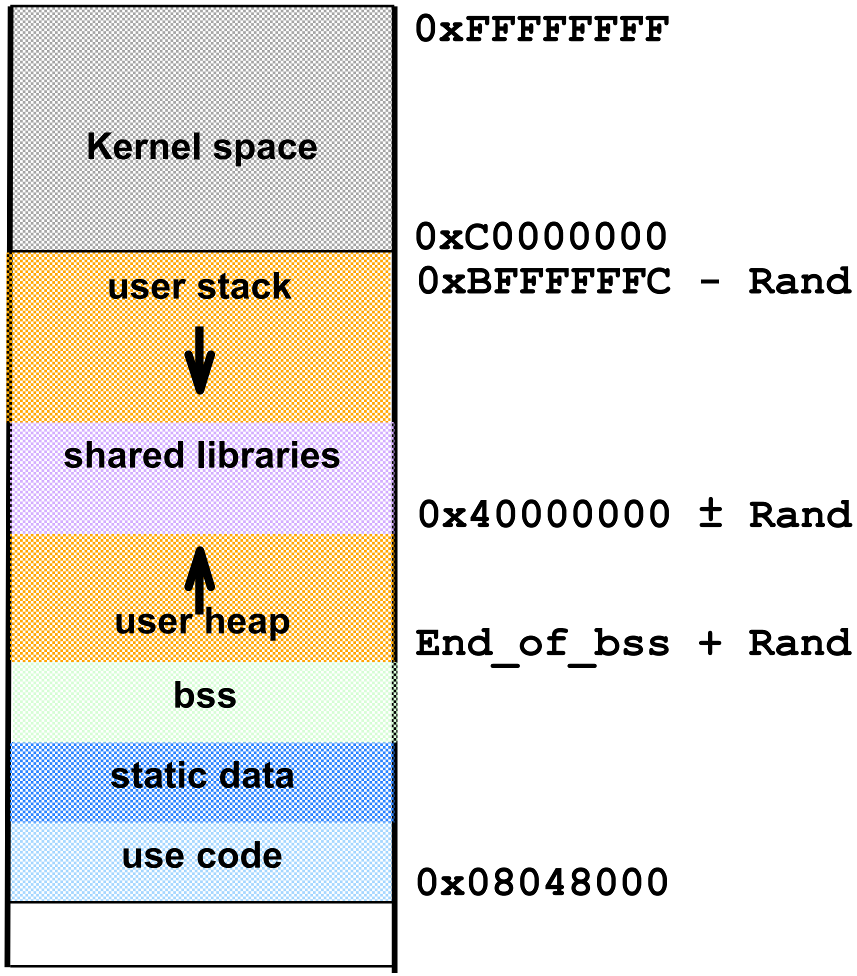
```
struct message {
  char buf[96];
  int (*fptr)(char*);
};
struct message msg;

int get_message(…){

  msg.fptr = printf;
  gets(msg.buf);
  msg.fptr(msg.buf);
}
```



Transparent Runtime Randomization

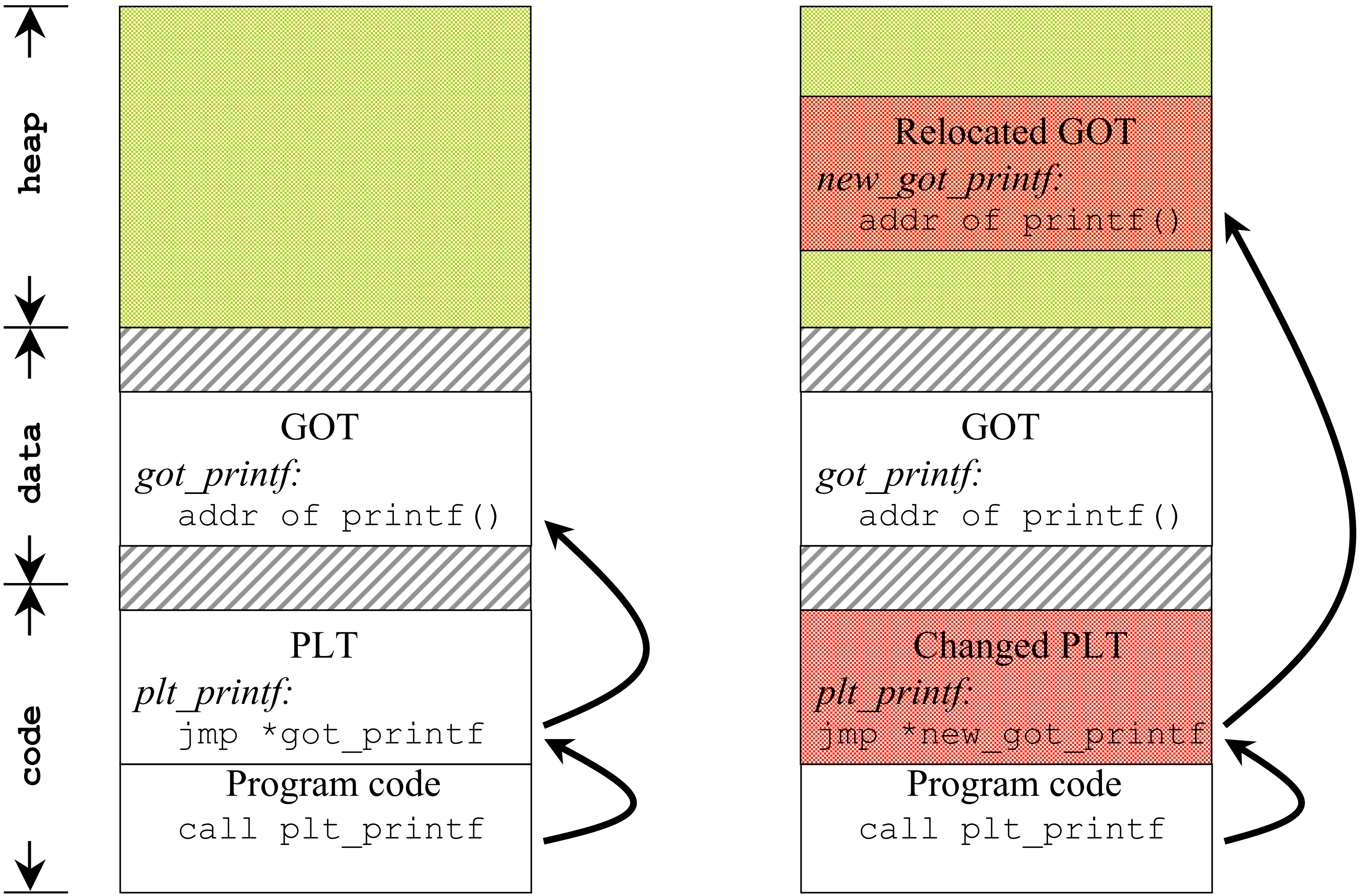
- Key to a successful attack
  - Correctly determine the runtime values of *m* or *p*
- Why runtime values of *m* or *p* can be determined?
  - Memory layout is fixed and addresses are highly predictable
  - Lack of diversity in modern systems
- Introduce diversity into a system
  - Dynamically randomize the memory layout of a program
  - Each invocation has a different layout
  - Defeating attacks
    - Breaks memory layout assumption
    - Make it hard to determine *m/p*
- Implementation – transparent to application
  - Modify dynamic program loader; develop programmable HW
  - Position independent regions: stack, heap, shared libraries
  - Position dependent regions: global offset table (GOT)

Position Independent Regions

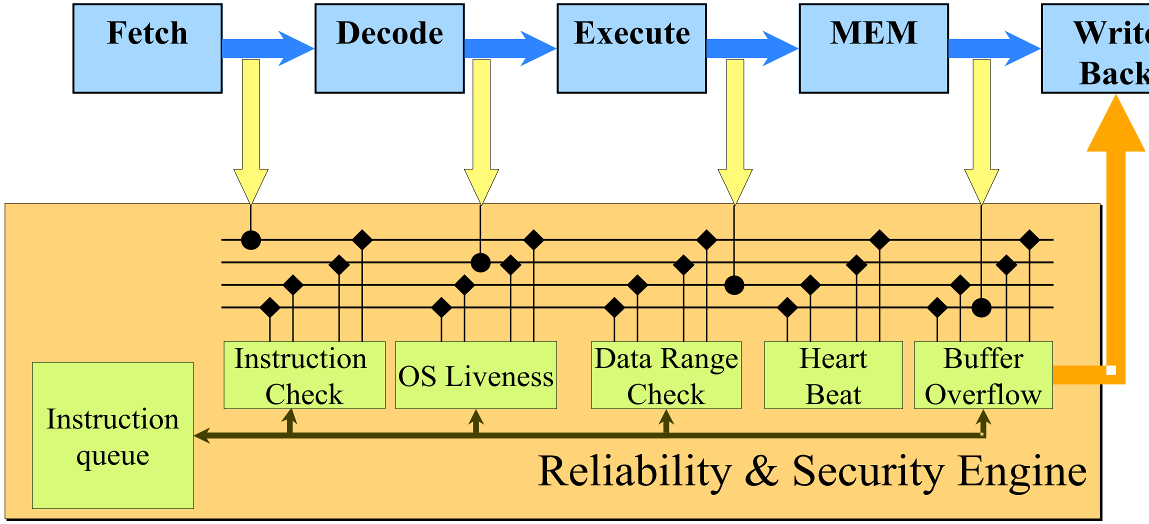


- Different sections at different fixed locations
- Change the loader
  - Part of the process initialization modules
  - Random offset is applied to different regions

Position Dependent Region



Reliability and Security Engine (RSE)



- Hardware framework to support checking modules
- Can be integrated with the processor or implemented as an FPGA-based engine
- Hardware execution blocks
  - homogenous
  - contain embedded hardware modules for
    - error detection and recovery
- Interface with the application through *CHECK* instructions
- Interface with the external system through generic I/O interface
- Modules are dynamically loadable and run-time re-configurable