

---

# Software Security

*Overflows, overruns and (some) confusions*

---

## Today's Lecture:

- Malware Taxonomy
- Characters and Numbers
- Canonical Representations
- Memory Management
- Race Conditions
- Defences

---

## Introduction:

Bad software is everywhere:

- NASA mars Lander (\$165 million) – Conversion of units
- [Ariane-5](#) rocket (\$500 million) – Integer overflow
- National Cancer Institute (8 patients die) –  
Drug Dosage Miscalculation
- Search for “*software horror stories*” or “*worst software bugs*” in Google

---

## Why is that?:

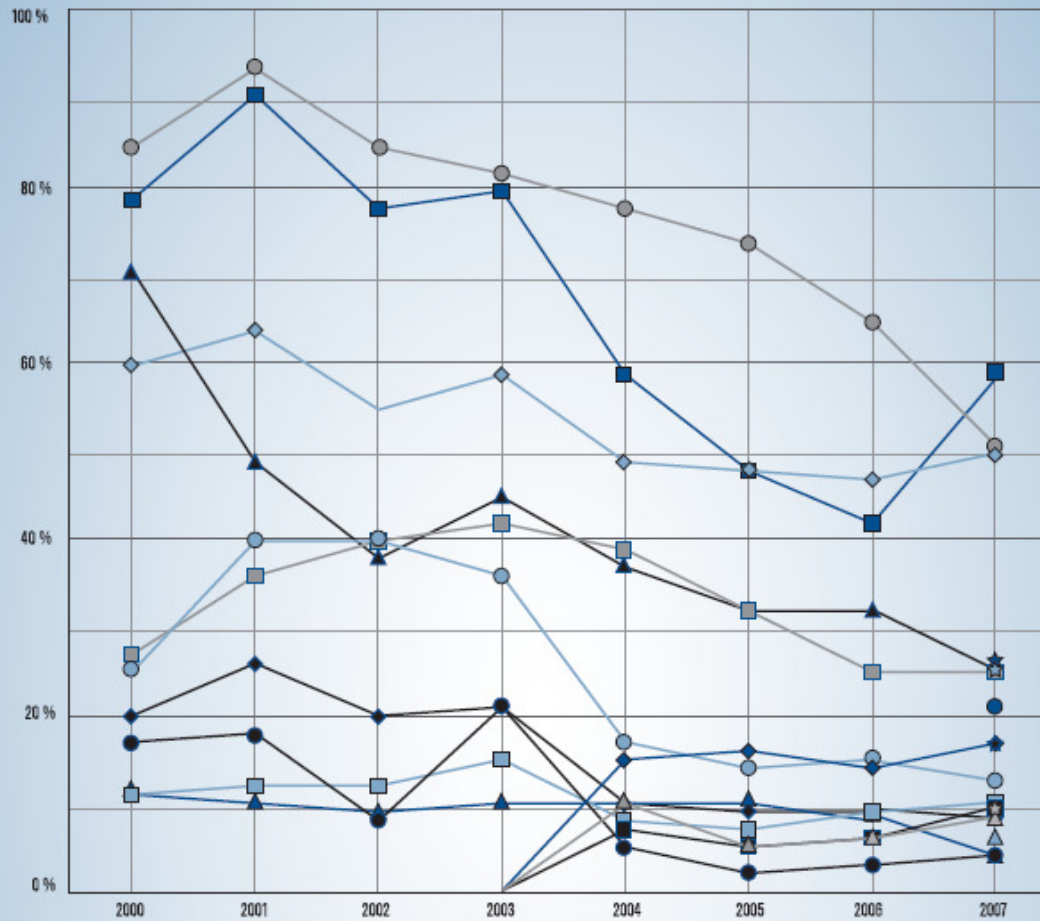
Complexity:

<u>System</u>	<u>Lines of Code:</u>
<i>Netscape</i>	– 17,000,000
<i>Windows</i>	– 40,000,000
<i>Space Shuttle</i>	– 10,000,000
<i>Boeing 777</i>	– 7,000,000

Estimated 5 bugs/1000 loc

---

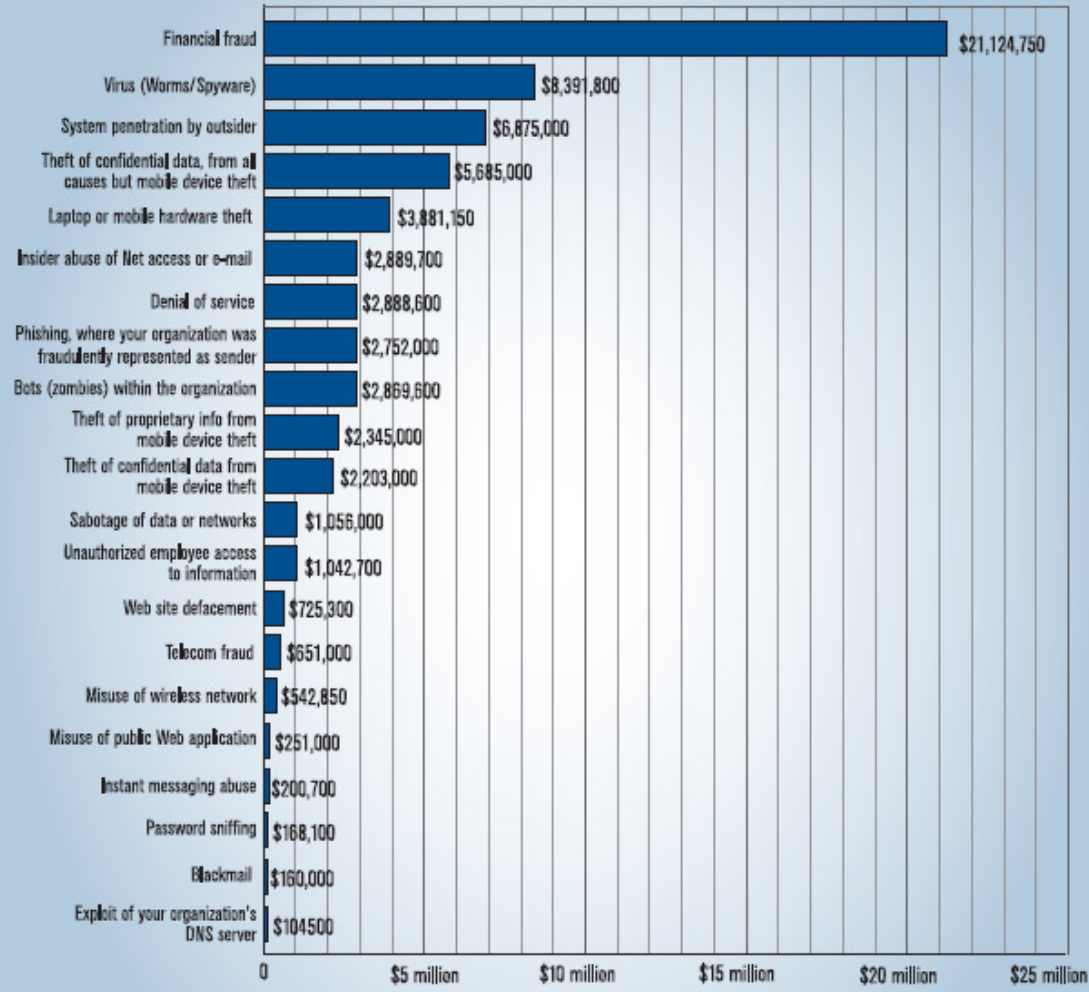
**Figure 14. Types of Attacks or Misuse Detected in the Last 12 Months**  
By Percent of Respondents



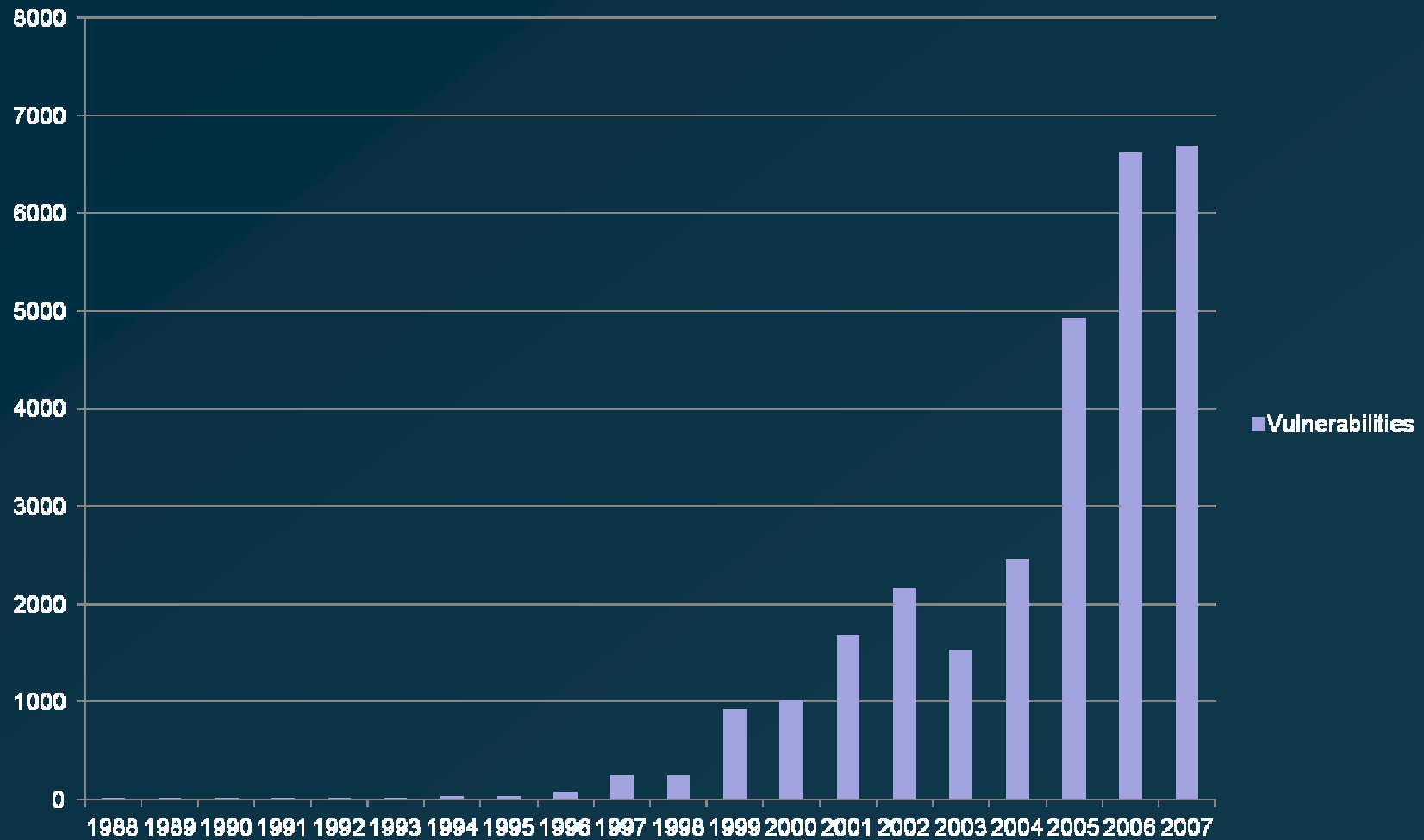
TYPE OF ATTACK	2007	TYPE OF ATTACK	2007
■ Insider abuse of Net access	59%	□ Financial fraud	12%
● Virus	52%	☆ Password sniffing**	10%
◇ Laptop / mobile device theft	50%	■ Web site defacement*	10%
★ Phishing where your organization was fraudulently represented as sender**	26%	▲ Misuse of public Web application*	9%
☆ Instant messaging misuse**	25%	◆ Theft of proprietary information (intellectual property)	8%
□ Denial of service	25%	▲ Exploit of the organization's DNS server**	6%
▲ Unauthorized access to information	25%	▲ Telecom fraud	5%
● Bots within the organization**	21%	● Sabotage	4%
★ Theft of customer / employee data**	17%		
◇ Abuse of wireless network*	17%		
○ System penetration	13%		

\*Added in 2004 survey  
\*\*Added in 2007 survey

**Figure 16. Dollar Amount Losses by Type of Attack**



## Vulnerability Growth



---

## Malware Taxonomy:

- Malware

Software that has malicious purpose

Types of malware:

- Viruses
- Worms
- Trojan Horse
- Backdoors
- Mobile Code
- Adware

---



---

## Malware Taxonomy:

- Viruses
  - self-replicating program with malicious intent
  - attaches themselves to executables
  - usually must be executed to propagate

First virus developed in 1982, called *Elk Cloner* for Apple DOS.

First PC Virus called (c)Brain in 1986

---

## Malware Taxonomy:

- Worms
  - malicious programs that use the internet to spread
  - also self-replicates
  - does not need human intervention to duplicate
  - can spread uncontrollably in a very brief period of time
  - usually uses a software exploit or via e-mail

Examples: Morris Worm, CodeRed, SQL Slammer

---

## Malware Taxonomy:

- Trojan Horses
  - seemingly innocent application that contains malicious code
  - usually useful programs that have unnoticeable but harmful side effects

Example: pretending a file is innocent, possible due to Windows hiding extensions.

---

## Malware Taxonomy:

- Backdoors
  - Malware that creates covert access for attacker
  - Used for connecting, spying, collecting data etc...
  - Can be embedded in normal programs
  - Sometimes planted by rogue software developers

Example: Windows encryption software, BT HomeHub

---

## Malware Taxonomy:

- Mobile Code
  - A class of benign programs that are mobile, executed on a large number of systems, not explicitly installed by end-user
  - Usually for purpose of interactive content  
e.g. ActiveX, Java applets
  - Can be misused by rogue developers
  - Exploit user's inability to distinguish between good and rogue sites

---

## Malware Taxonomy:

- Adware
  - Program forcing unsolicited advertising on end users
  - Very popular
  - Usually bundled with free software, funded by the advertising
  - Gathers statistics about internet usage
  - Information used to display targeted information

---

## Malware Taxonomy:

- Sticky Software
  - Software implementing methods that prevent its deletion (e.g. no uninstall program)

### Future of Malware:

- Bios, Firmware, Sub-OS, etc...
- Polymorphic and Metamorphic Viruses

## Characters and Numbers:

- Many software flaws due to broken abstractions

### Characters:

- Different ways of representing characters exist
- If input validation does not exist or not complete - > trouble!

Example: *MS IIS vulnerability*

{IPAddress}/scripts/..%c0%af../winnt/system32/

%c0%af in UTF-8 encoding = /

thus -> C:\winnt\system32\



## Characters and Numbers:

### Integer overflows:

- when arithmetic operation attempts to create an integer larger than can be represented in available memory

### e.g. Ariane-5:

- 64 bit floating point number converted to 16-bit signed integer
- Number larger than 32,767 thus conversion failed
- Rocket exploded

---

## Characters and Numbers:

### Array overflows:

- Computing array indices uses integer arithmetic
- If no check exists, memory outside of allocated space gets accessed.
- upper and lower bounds of arrays should be checked
- some languages deal with this issue better than others (e.g. C vs. Java)

---

## Canonical Representations:

- Alternative names for various parts of systems

e.g. DNS, relative file paths

- Translation of names to their correct representations can be done in more than one way
- If not kept in mind, problems can arise

e.g. case sensitivity of files `pwd` vs. `Pwd`

Lesson – never trust your inputs

---

## Memory Management:

- In C and C++ programmer performs memory management
- Flexible, fast but dangerous
  - Buffer Overruns
  - Stack Overruns
  - Heap Overruns
  - Type Confusion

---

## Memory Management:

Buffer overrun:

- When program is executed, memory sections (buffers) are allocated to variables
- If such value exceeds size of buffer, an overflow occurs
- adjacent memory locations are overwritten

## Memory Management:

Stack overrun:

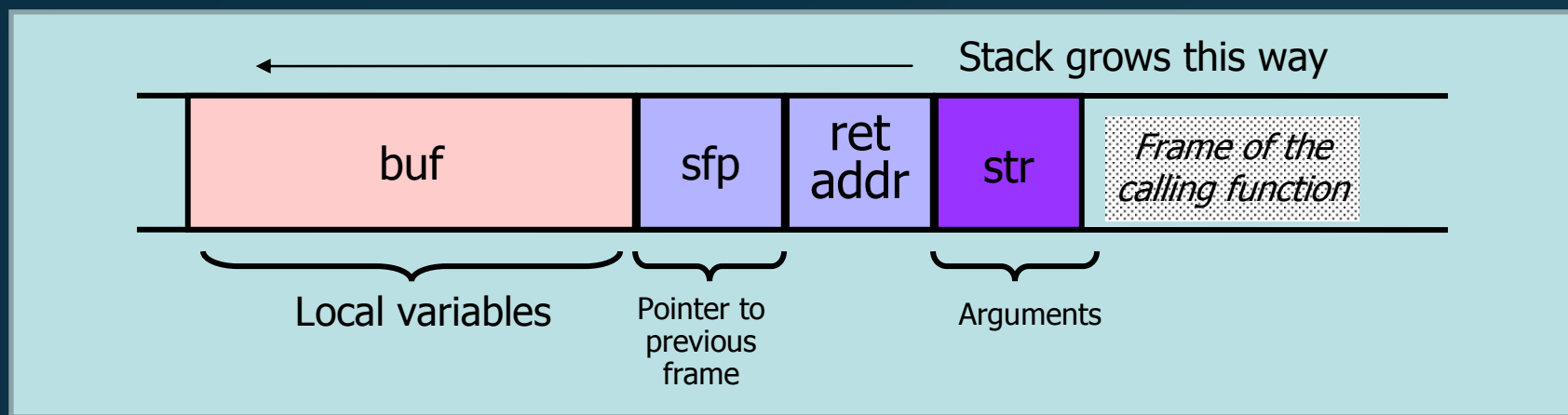
- Suppose Web server contains this function

```
void func(char *str) {  
    char buf[126];  
    strcpy(buf, str);  
}
```

Allocate local buffer  
(126 bytes reserved on stack)

Copy argument into local buffer

- When this function is invoked, a new frame with local variables is pushed onto the stack



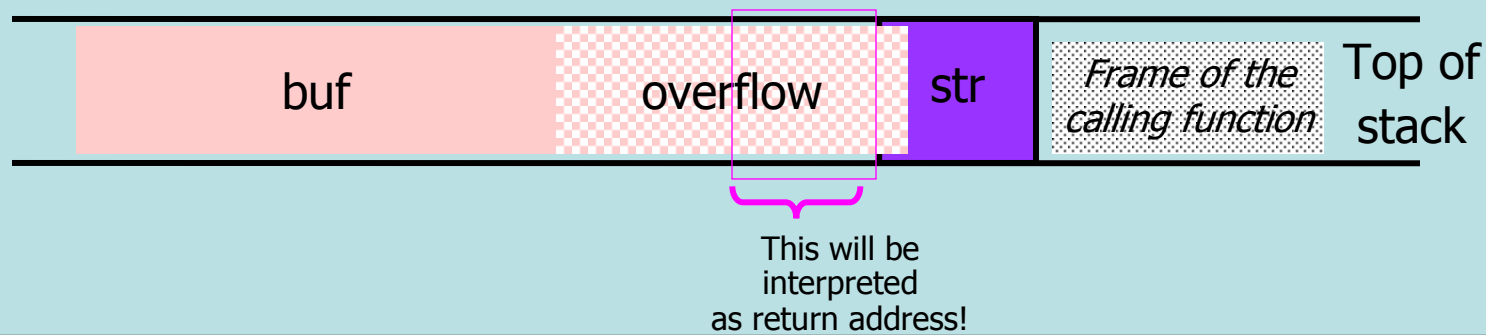
## Memory Management:

- In case str is overstuffed
- Memory pointed to by str is copied onto stack...

```
void func(char *str) {  
    char buf[126];  
    strcpy(buf, str);  
}
```

strcpy does NOT check whether the string at \*str contains fewer than 126 characters

- If a string longer than 126 bytes is copied into buffer, it will overwrite adjacent stack locations



---

## Memory Management:

### Heap overrun:

- Occurs on the heap
- Heap – memory area dynamically allocated by application
- contains for example pointers to (temporary) files and functions
- attacker redirects pointer from tmp file to target file using a buffer overrun, e.g. to a password file and overwrites it with his own data



---

## Memory Management:

### Type Confusion:

- manipulation of pointer structure so that two pointers, one with a wrong class tag, point to the same objects
- thus object can be manipulated by parties that have access to the 'wrong' type.
  
- Java and .NET attempt to prevent this

---

## Race Conditions:

- Occurs when multiple computations access shared data in a way that their results depend on the sequence of access.
- Known in security literature as TOCTTOU
- Traditional issues of concurrency

## Defences:

- *Hardware* – specialised hardware
  - *Type Safety* – type safe languages
  - *Safer Functions* – using safer coding practices
- Prevention
- *Code Inspection* – Fuzzing, Valgrind, etc...
  - *Testing*
- Detection
- *Least Privilege*
- Mitigation

---

## Summary:

- Malware Taxonomy
- Overflows
- Defences