Segurança de Sistemas e dados (MSI 2021/2022)

Aula 6

Rolando Martins
DCC – FCUP
Slides Adaptados do Prof. Manuel Eduardo Correia

Inference Control (De-identification)

Inference Control Example

* Suppose we query a database
  * Question: What is average salary of female CS professors at SJSU?
  * Answer: € 95,000
  * Question: How many female CS professors at SJSU?
  * Answer: 1
* Specific information has leaked from responses to general questions!

Inference Control and Research

* For example, medical records are private but valuable for research
* How to make info available for research and protect privacy?
* How to allow access to such data without leaking specific information?

Naïve Inference Control

* Remove names from medical records?
* Still may be easy to get specific info from such "anonymous" data
* Removing names is not enough
  * As seen in previous example
* What more can be done?

Less-naïve Inference Control

* Query set size control
  * Don't return an answer if set size is too small
* N-respondent, k% dominance rule
  * Do not release statistic if k% or more contributed by N or fewer
  * Example: Avg salary in Bill Gates' neighborhood
  * This approach used by US Census Bureau
* Randomization
  * Add small amount of random noise to data
* Many other methods — none satisfactory

Inference Control

* Robust inference control may be impossible
* Is weak inference control better than nothing?
  * **Yes**: Reduces amount of information that leaks
* Is weak covert channel protection better than nothing?
  * **Yes**: Reduces amount of information that leaks
* Is weak crypto better than no crypto?
  * **Probably not:** Encryption indicates important data
  * May be easier to filter encrypted data

CAPTCHA

Completely Automated Public Turing test to tell Computers and Humans Apart

Turing Test

* Proposed by Alan Turing in 1950
* Human asks questions to one human and one computer, without seeing either
* If questioner cannot distinguish human from computer, computer passes the test
* The **gold standard** in artificial intelligence
* No computer can pass this today
  * But see the Loebner Prize

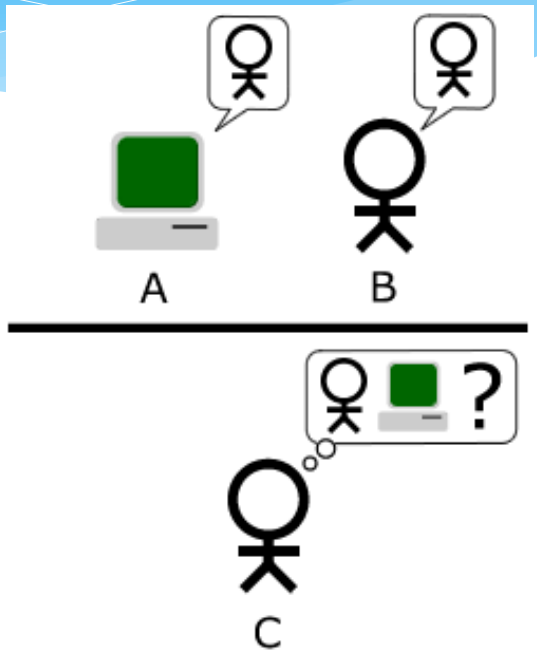


Image from wikipedia

CAPTCHA

* **CAPTCHA**
  * **C**ompletely **A**utomated **P**ublic **T**uring test to tell **C**omputers and **H**umans **A**part
* **A**utomated — test is generated and scored by a computer program
* **P**ublic — program and data are public
* **T**uring test to tell… — humans can pass the test, but machines cannot pass
  * Also known as **HIP** == **H**uman **I**nteractive **P**roof
* Like an inverse Turing test (well, sort of…)

CAPTCHA Paradox?

* "… CAPTCHA is a program that can generate and grade tests that it itself cannot pass…"
  * "… much like some professors…"
* Paradox — computer creates and scores test that it cannot pass!
* CAPTCHA used so that only humans can get access (i.e., no bots/computers)
* CAPTCHA is for **access control**

CAPTCHA Uses?

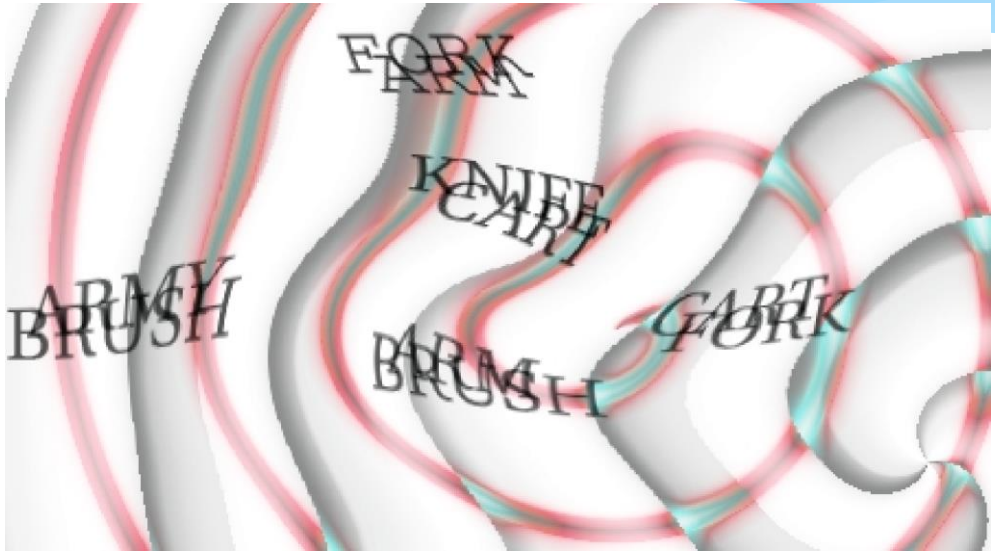
* Original motivation: automated bots stuffed ballot box in vote for best CS grad school
  * SJSU vs Stanford?
* Free email services — spammers like to use bots to sign up for 1000's of email accounts
  * CAPTCHA employed so only humans get accounts
* Sites that do not want to be automatically indexed by search engines
  * CAPTCHA would force human intervention

CAPTCHA: Rules of the Game

* Easy for most humans to pass
* Difficult or impossible for machines to pass
  * **Even with access to CAPTCHA software**
* From Trudy's perspective, the only unknown is a random number
  * Analogous to Kerckhoffs' Principle
* Desirable to have different CAPTCHAs in case some person cannot pass one type
  * Blind person could not pass visual test, etc.

CAPTCHAs Example

* Test: Find 2 words in the following



- Easy for most humans
- A (difficult?) OCR problem for computer
  - OCR == Optical Character Recognition

CAPTCHAs

* Current types of CAPTCHAs
  * Visual — like previous example
  * Audio — distorted words or music
* No text-based CAPTCHAs
  * Maybe this is impossible…

CAPTCHA's and AI

* OCR is a challenging AI problem
  * Hard part is the **segmentation problem**
  * Humans good at solving this problem
* Distorted sound makes good CAPTCHA
  * Humans also good at solving this
* Hackers who break CAPTCHA have solved a hard AI problem
  * So, putting hacker's effort to good use!
* Other ways to defeat CAPTCHAs???

The end

Access control

Software and Security

"If automobiles had followed the same development cycle as the computer, a Rolls-Royce would today cost $**100**, get a **million miles** per gallon, and **explode once a year**, killing everyone inside"

--- Robert X. Cringely

Why Software?

* Why is software as important to security as crypto, access control and protocols?
* Virtually all of information security is implemented in software
* If your software is subject to attack, your security is broken
  * Regardless of strength of crypto, access control or protocols
* Software is a poor foundation for security

Bad Software

* Bad software is everywhere!
* NASA Mars Lander (cost $165 million)
  * Crashed into Mars
  * Error in converting English and metric units of measure
* Denver airport
  * Buggy baggage handling system
  * Delayed airport opening by 11 months
  * Cost of delay exceeded $1 million/day
* MV-22 Osprey
  * Advanced military aircraft
  * Lives have been lost due to faulty software

Software Issues

"Normal" users

- Find bugs and flaws by accident
- Hate bad software…
- …but must learn to live with it
- Must make bad software work

Attackers

* Actively look for bugs and flaws
* Like bad software…
* … and try to make it misbehave
* Attack systems thru bad software

Complexity

- "**Complexity is the enemy of security**", Paul Kocher, Cryptography Research, Inc.

system	Lines of code (LOC)
Netscape	17,000,000
Space shuttle	10,000,000
Linux	1,500,000
Windows XP	40,000,000
Boeing 777	7,000,000

- A new car contains several orders of magnitude more LOC than was required to land the Apollo astronauts on the moon

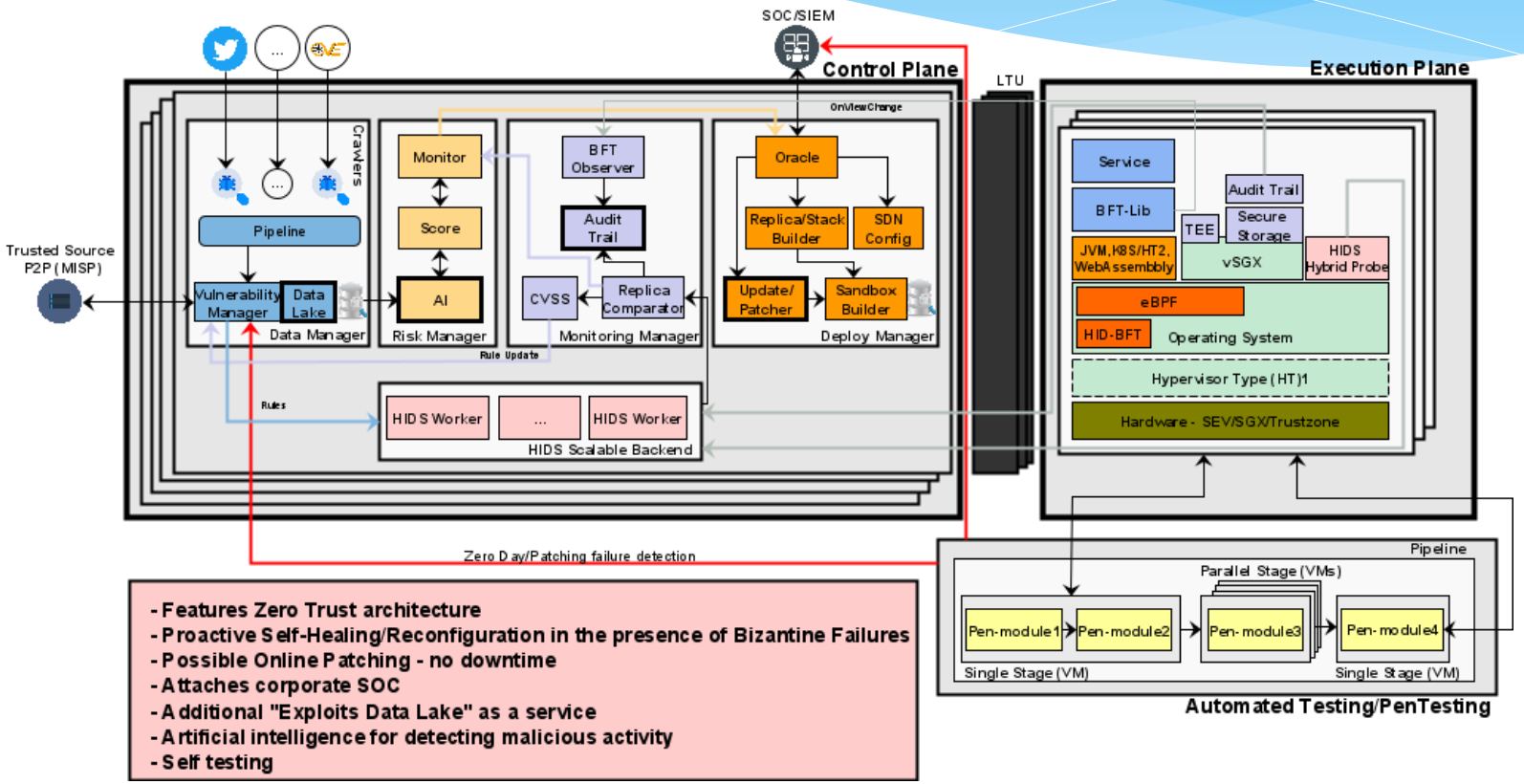
Lines of Code and Bugs

* Conservative estimate: 5 bugs/1000 LOC
* **Do the math**
  * Typical computer: 3,000 exe's of 10K LOC each
  * Conservative estimate of 50 bugs/exe
  * About 150k bugs per computer
  * 30,000 node network has 4.5 billion bugs
  * Suppose that only 10% of bugs security-critical and only 10% of those remotely exploitable
  * Then **"only" 4.5 million critical security flaws**!

Counter-Measurements: Skynet

* Fault Intrusion Tolerance
* **Features**
  * Zero-day detection
    * Risk Analysis
    * Graph mining
  * **Degradation under intrusion but maintains correctness**
  * Self-Testing
    * Introspection
    * Secure Enclaves as secure anchors
  * Self-healing

Counter-Measurements: Skynet's Architecture



Software Security Topics

* Program flaws (unintentional)
  * Buffer overflow
  * Incomplete mediation
  * Race conditions
* Malicious software (intentional)
  * Viruses
  * Worms
  * Other breeds of malware

Program Flaws

* An **error** is a programming mistake
  * To err is human
* An error may lead to incorrect state: **fault** (**defeito**)
  * A fault is internal to the program – not externally observable.
* A fault may lead to a **failure** (**falha**), where a system departs from its expected behavior
  * A failure is externally observable

Example

```
char array[10];
for(i = 0; i < 10; ++i)
     array[i] = `A`;
array[10] = `B`;
```

- This program has an **error** **(erro)**
- This error might cause a **fault** **(defeito)**
  - Incorrect internal state
- If a fault occurs, it might lead to a **failure** **(falha)**
  - Program behaves incorrectly (external)
- Informally we end up using the term **flaw** **(falha)** for all of the above.

Secure Software

* In software engineering, **try to** insure that a program does what is intended
* Secure software engineering requires that the software **does what is intended…**
* **… and nothing more**
* Absolutely secure software is impossible
  * Absolute security is almost never possible!
* How can we manage the risks associated with the inevitable software flaws?

Program Flaws

* Program flaws are unintentional
  * But still create security risks
* We'll consider 3 types of flaws
  * Buffer overflow (smashing the stack)
  * Incomplete mediation
  * Race conditions
* Many other flaws can occur
* These are the most common

Buffer Overflow

Typical Attack Scenario

* Users enter data into a Web form
* Web form is sent to server
* Server writes data to buffer, without checking length of input data
* Data overflows from buffer
* Sometimes, overflow can enable an attack
* Web form attack could be carried out by anyone with an Internet connection

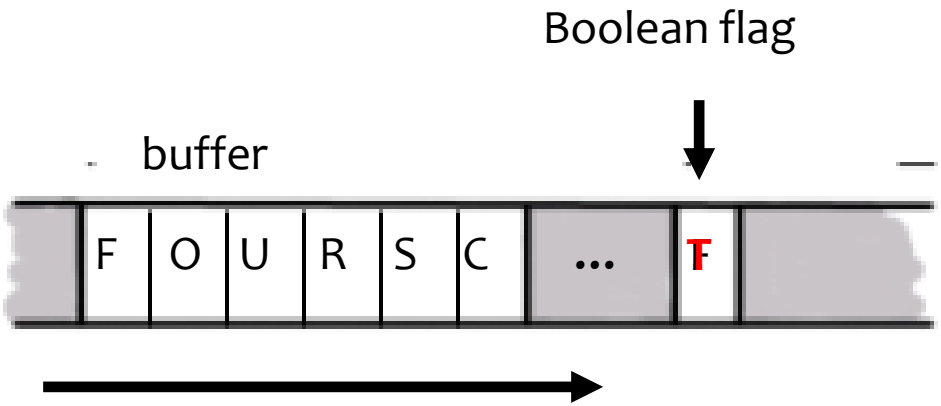
Buffer Overflow

```
int main(){
    int buffer[10];
    buffer[20] = 37;}
```

* **Q:** What happens when this is executed?
* **A:** Depending on what resides in memory at location "buffer[20]"
  * Might overwrite **user** data or code
  * Might overwrite **system** data or code

Simple Buffer Overflow

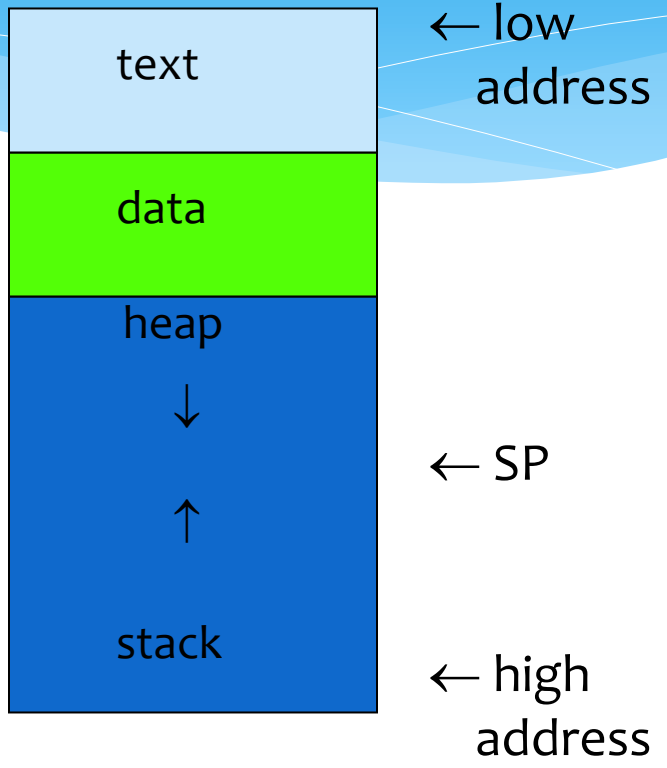
* Consider boolean flag for authentication
* Buffer overflow could overwrite flag allowing anyone to authenticate!



- In some cases, attacker need not be so lucky as to have overflow overwrite flag

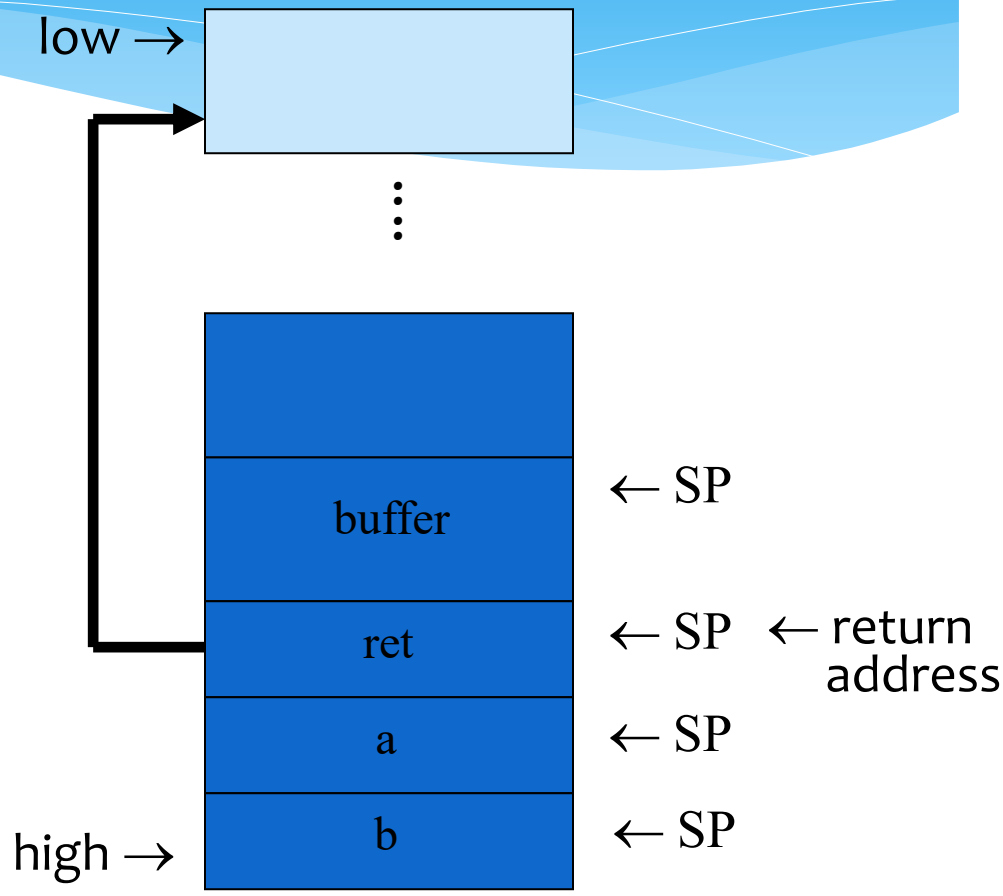
Memory Organization

* **Text** == code
* **Data** == static variables
* **Heap** == dynamic data
* **Stack** == "scratch paper"
  * Dynamic local variables
  * Parameters to functions
  * Return address



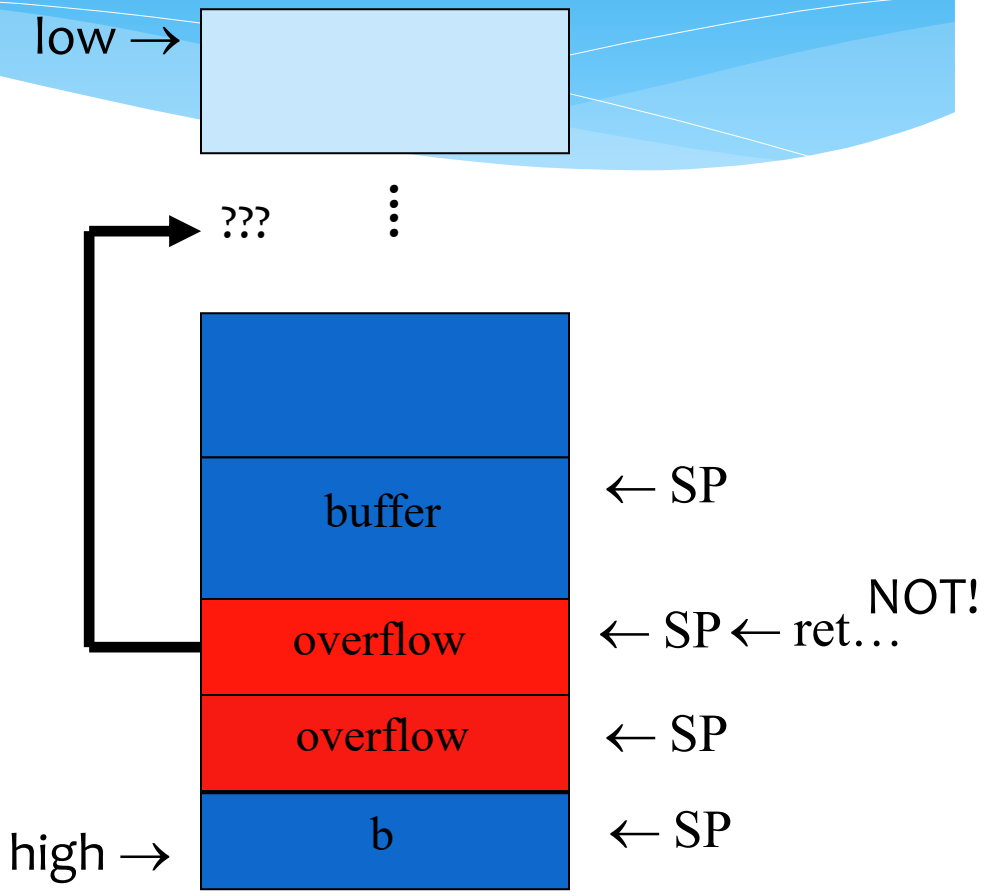
Simplified Stack Example

```
void func(int a, int b){
   char buffer[10];
}
void main(){
   func(1, 2);
}
```



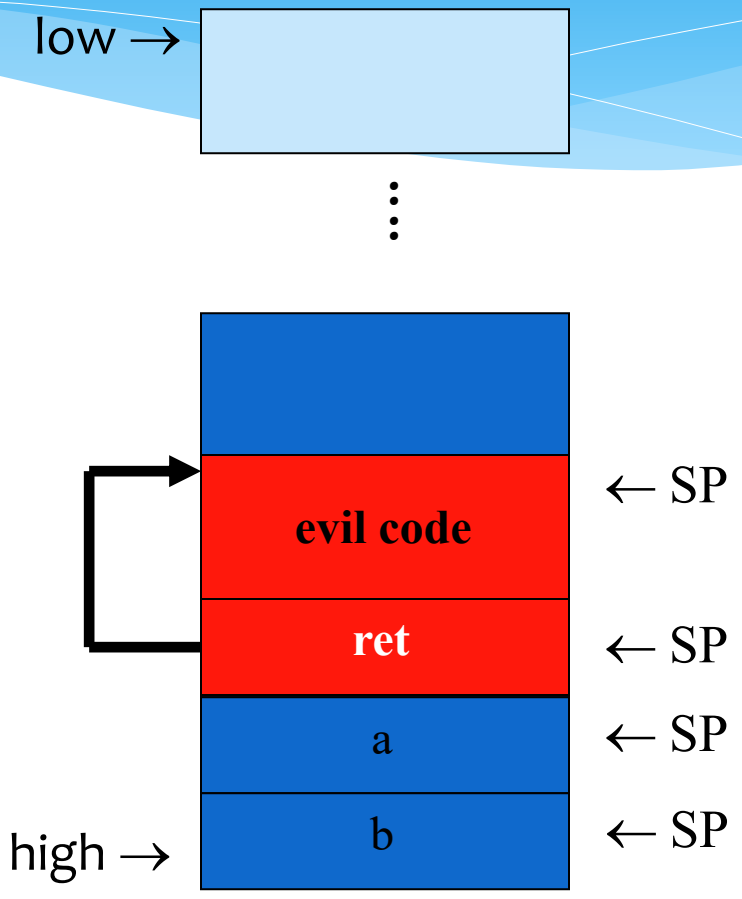
Smashing the Stack

- What happens if buffer overflows?
- Program "returns" to wrong location
- A crash is likely



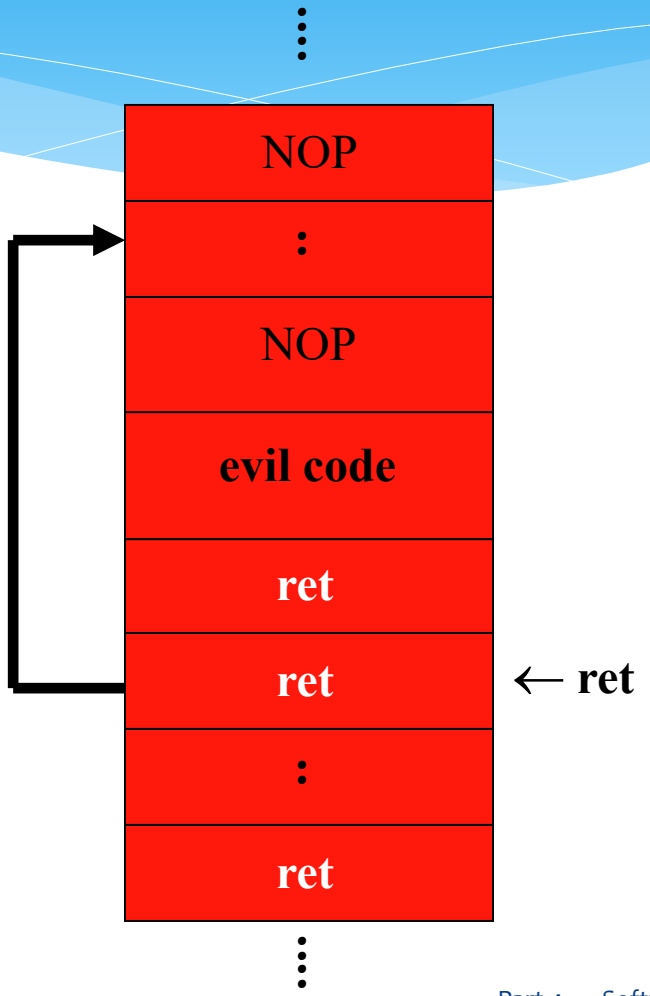
Smashing the Stack

- Trudy has a better idea…
- **Code injection**
- Trudy can run code of her choosing!



Smashing the Stack

- Trudy may not know
  - Address of evil code
  - Location of **ret** on stack
- Solutions
  - Precede evil code with NOP "landing pad"
  - Insert lots of new **ret**

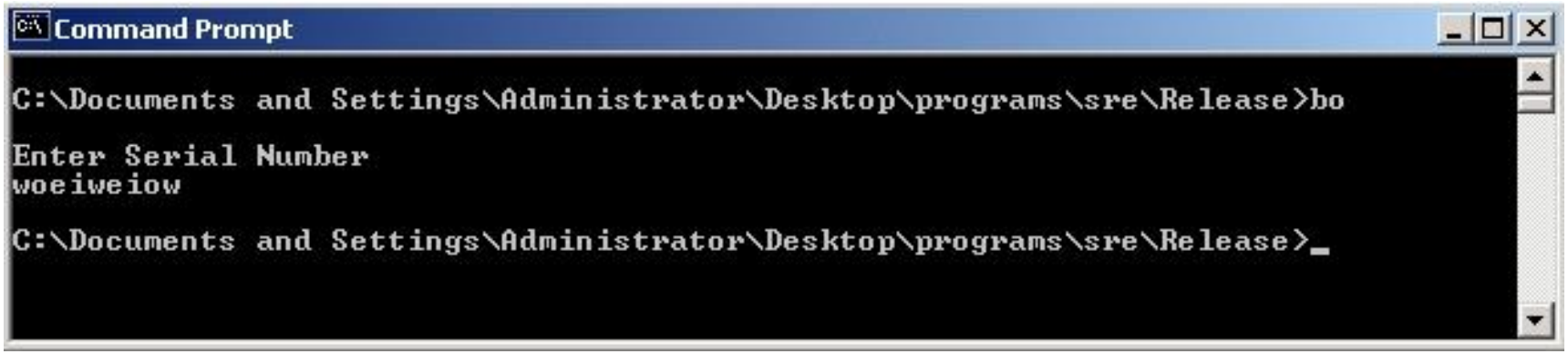


Stack Smashing Summary

* A buffer overflow must exist in the code
* Not all buffer overflows are exploitable
  * Things must line up just right
* If exploitable, attacker can **inject code**
* Trial and error likely required
  * Lots of help available online
  * Smashing the Stack for Fun and Profit, Aleph One
* Also heap overflow, integer overflow, etc.
* Stack smashing is "attack of the decade"

Stack Smashing Example

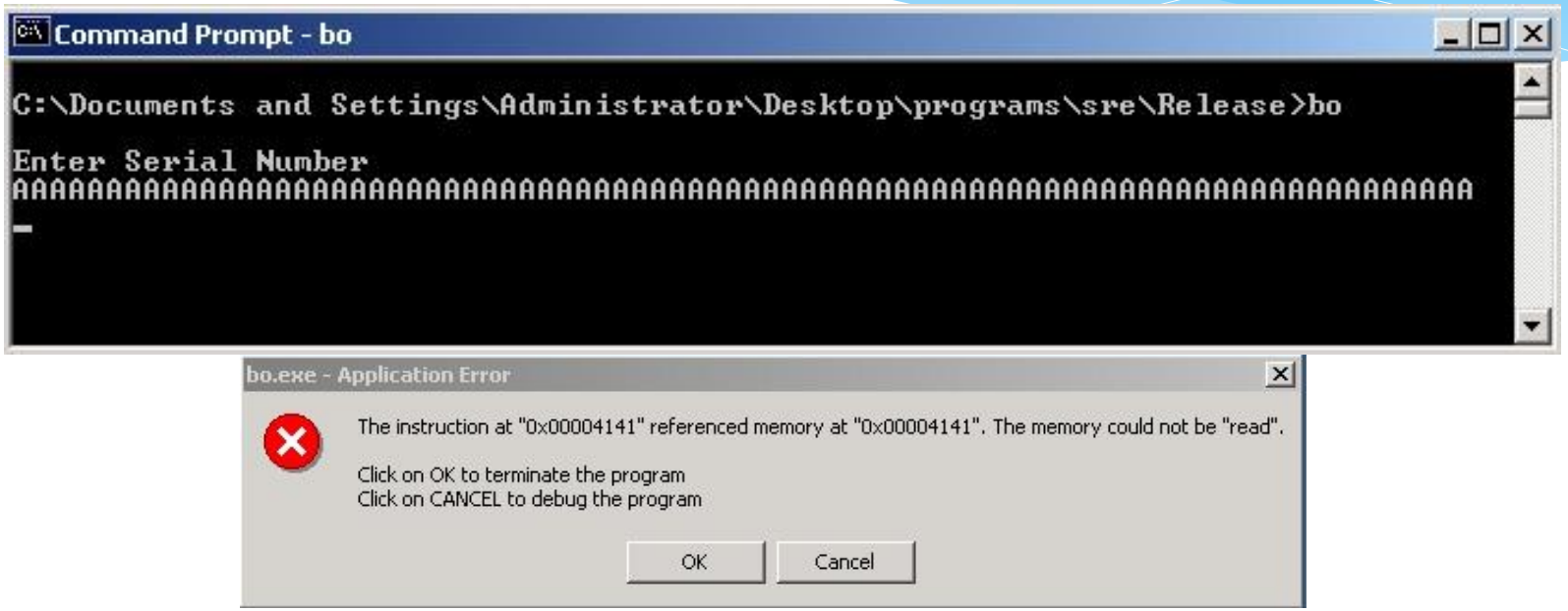
* Program asks for a serial number that the attacker does not know
* Attacker does **not** have source code
* Attacker does have the executable (exe)



- Program quits on incorrect serial number

Example

- By trial and error, attacker discovers an apparent buffer overflow



- Note that 0x41 is "A"
- Looks like **ret** overwritten by 2 bytes!

Example

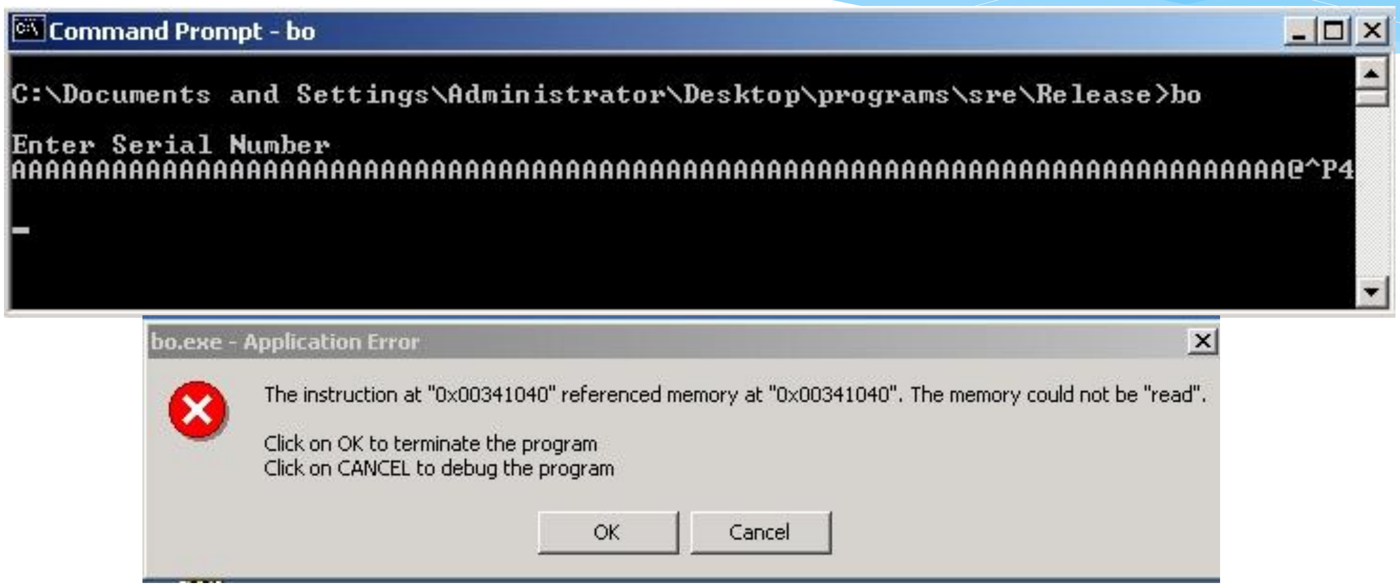
* Next, disassemble bo.exe to find

```
.text:00401000
.text:00401000                sub     esp, 1Ch
.text:00401003                push    offset aEnterSerialNum ; "\nEnter Serial Number\n"
.text:00401008                call    sub_40109F
.text:0040100D                lea     eax, [esp+20h+var_1C]
.text:00401011                push    eax
.text:00401012                push    offset aS       ; "%s"
.text:00401017                call    sub_401088
.text:0040101C                push    8
.text:0040101E                lea     ecx, [esp+2Ch+var_1C]
.text:00401022                push    offset aS123n456 ; "S123N456"
.text:00401027                push    ecx
.text:00401028                call    sub_401050
.text:0040102D                add     esp, 18h
.text:00401030                test    eax, eax
.text:00401032                jnz     short loc_401041
.text:00401034                push    offset aSerialNumberIs ; "Serial number is correct.\n"
.text:00401039                call    sub_40109F
.text:0040103E                add     esp, 4
```

- The goal is to exploit buffer overflow to jump to address 0x401034

Example

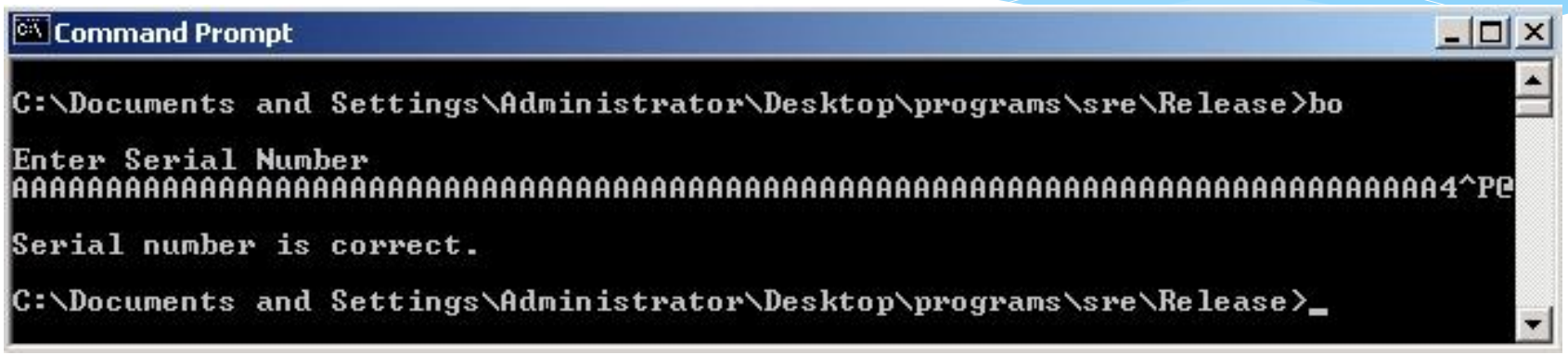
* Find that 0x401034 is "@^P4" in ASCII



- Byte order is reversed? Why?
- X86 processors are "little-endian"

Example

* Reverse the byte order to "4^P@" and…



- Success! We've bypassed serial number check by exploiting a buffer overflow
- Overwrote the return address on the stack

Example

* Attacker did not require access to the source code
* Only tool used was a disassembler to determine address to jump to
* Can find address by trial and error
  * Necessary if attacker does not have exe
  * For example, a remote attack

Example

* Source code of the buffer overflow

- Flaw easily found by attacker
- **Even without the source code!**

```
#include <stdio.h>
#include <string.h>

main()
{
    char in[75];

    printf("\nEnter Serial Number\n");

    scanf("%s", in);

    if(!strncmp(in, "S123N456", 8))
    {
       printf("Serial number is correct.\n");
    }
}
```