DENNIS GUNAWAN

IF470 COMPUTER SECURITY

07 BUFFER OVERFLOW

REVIEW: PHYSICAL DATA LOSS

- Threat
  - Malware – Virus, Trojan Horse, and Worm
- Technical Details
  - Malicious Code
- Vulnerability
  - Voluntary Introduction
  - Unlimited Privilege
  - Stealthy Behavior – Hard to Detect and Characterize
- Countermeasure
  - Hygiene
  - Detection Tools
  - Error Detecting and Error Correcting Codes
  - Memory Separation
  - Basic Security Principles

COURSE SUB LEARNING OUTCOMES (SUB-CLO)

- Sub-CLO 7
  - Students are able to relate buffer overflow to real case in their daily life (C3)

OUTLINE

- Harm
  - Destruction of Code and Data
- Vulnerability
  - Off-by-One Error
  - Integer Overflow
  - Unterminated Null-Terminated String
  - Parameter Length and Number
  - Unsafe Utility Programs
- Countermeasure
  - Programmer Bounds Checking
  - Programming Language Support
  - Stack Protection / Tamper Detection
  - Hardware Protection of Executable Space
  - General Access Control

CONCEPT

Trying to put more than *n* bytes of data into a space big enough to hold only *n*

MEMORY ALLOCATION

Memory is

a scarce but flexible resource

- Operating systems jam one data element next to another, without regard for data type, size, content, or purpose
- Users and programmers seldom know, much less have any need to know, precisely which memory location a code or data item occupies

CODE & DATA

- Instructions, data, and everything else in memory are all binary strings
  - Strings of 0s and 1s

- `0x41` represents the letter A, the number 65, or the instruction to move the contents of register 1 to the stack pointer
- If you happen to put the data string "A" in the path of execution, it will be executed as if it were an instruction

CODE & DATA

- Each computer instruction determines how data values are interpreted
  - An Add instruction implies the data item is interpreted as a number
  - A Jump instruction assumes the target is an instruction
- At the machine level, nothing prevents a Jump instruction from transferring into a data field or an Add command operating on an instruction, although the results may be unpleasant

CODE & DATA

- Usually we do not treat code as data, or vice versa
  - We do not usually try to execute data values or perform arithmetic on instructions
  - Attackers sometimes do, especially in memory overflow attacks

- `0x1C`*pq* is the operation code for a Jump instruction
  - Go to the instruction *pq* bytes ahead of this instruction
- `0x1C0A`
  - Interpreted as jump forward 10 bytes
  - Represents the two-byte decimal integer 7178
  - Storing the number 7178 in a series of instructions is the same as having programmed a Jump

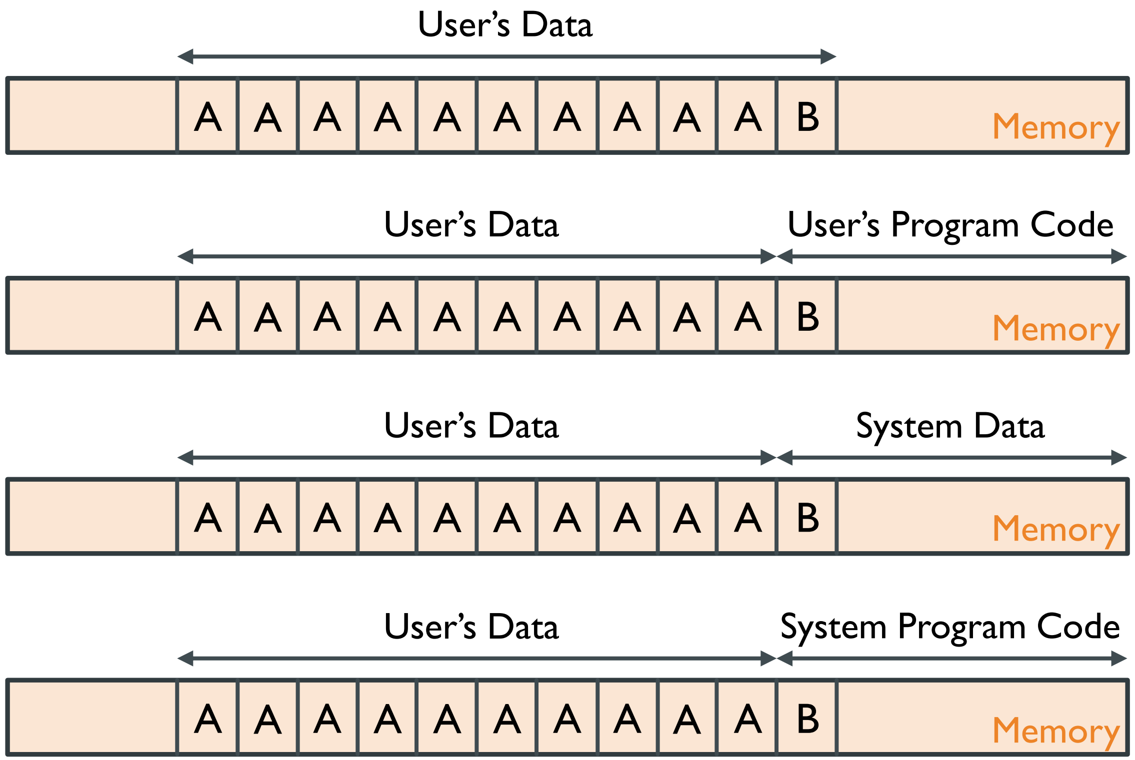
CODE & DATA

- The attacker's trick
  - To cause data to spill over into executable code
  - To select the data values such that they are interpreted as valid instructions to perform the attacker's goal
- Two-step goal
  - Cause the overflow
  - Experiment with the ensuing action to cause a desired, predictable result

IMPLICATIONS OF OVERWRITING MEMORY

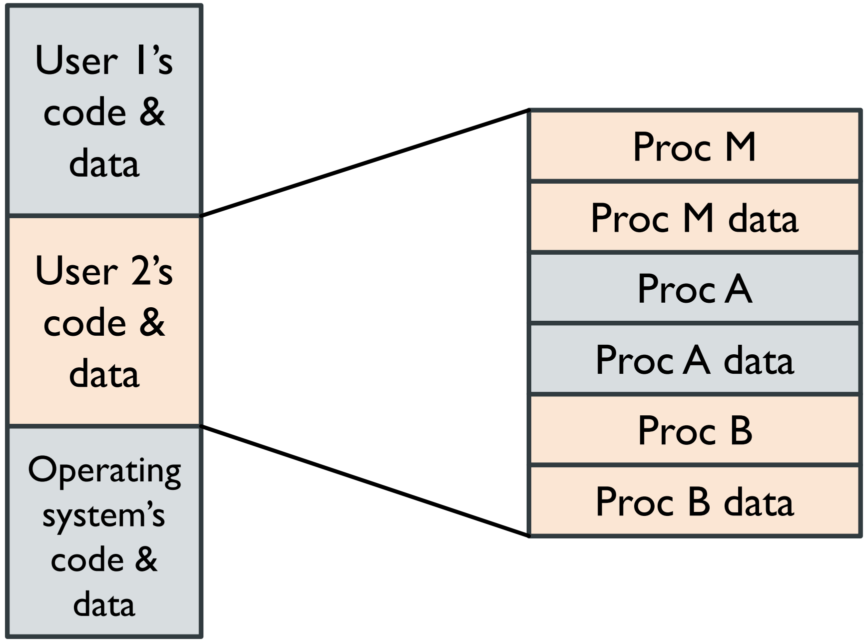
```
for(i=0; i<=9; i++)
   sample[i] = 'A';
sample[10] = 'B';
```

- The problem's occurrence depends on what is adjacent to the array sample
- All program and data elements are in memory during execution, sharing space with the operating system, other code, and resident routines

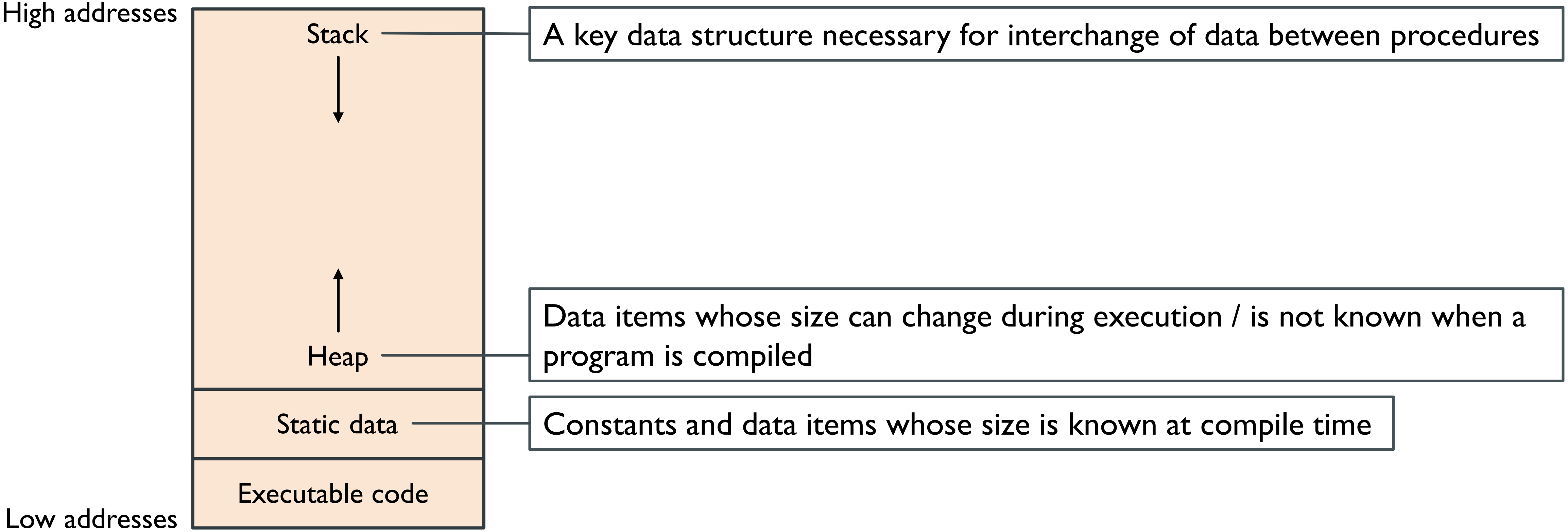


IMPLICATIONS OF OVERWRITING MEMORY

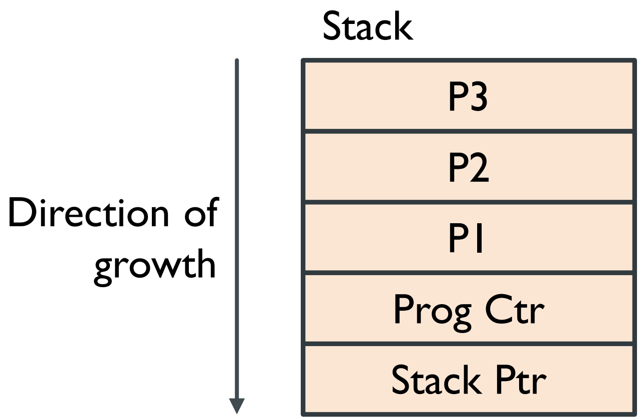
- The data end up on top of one of
  - Another piece of your data
  - An instruction of yours
  - Data or code belonging to another user
  - Data or code belonging to the operating system



THE STACK & THE HEAP

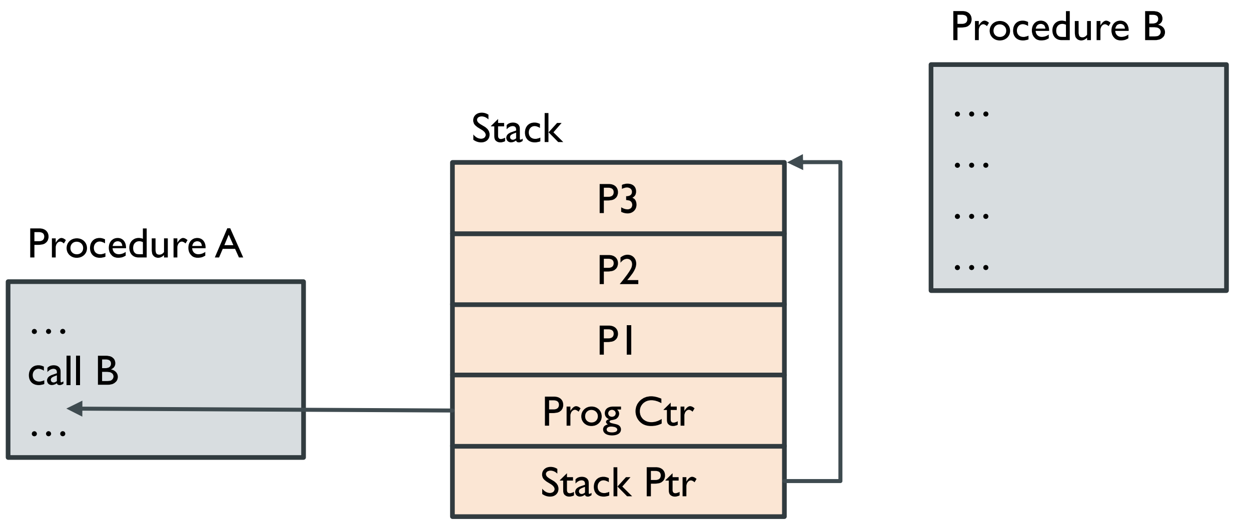


THE STACK & THE HEAP



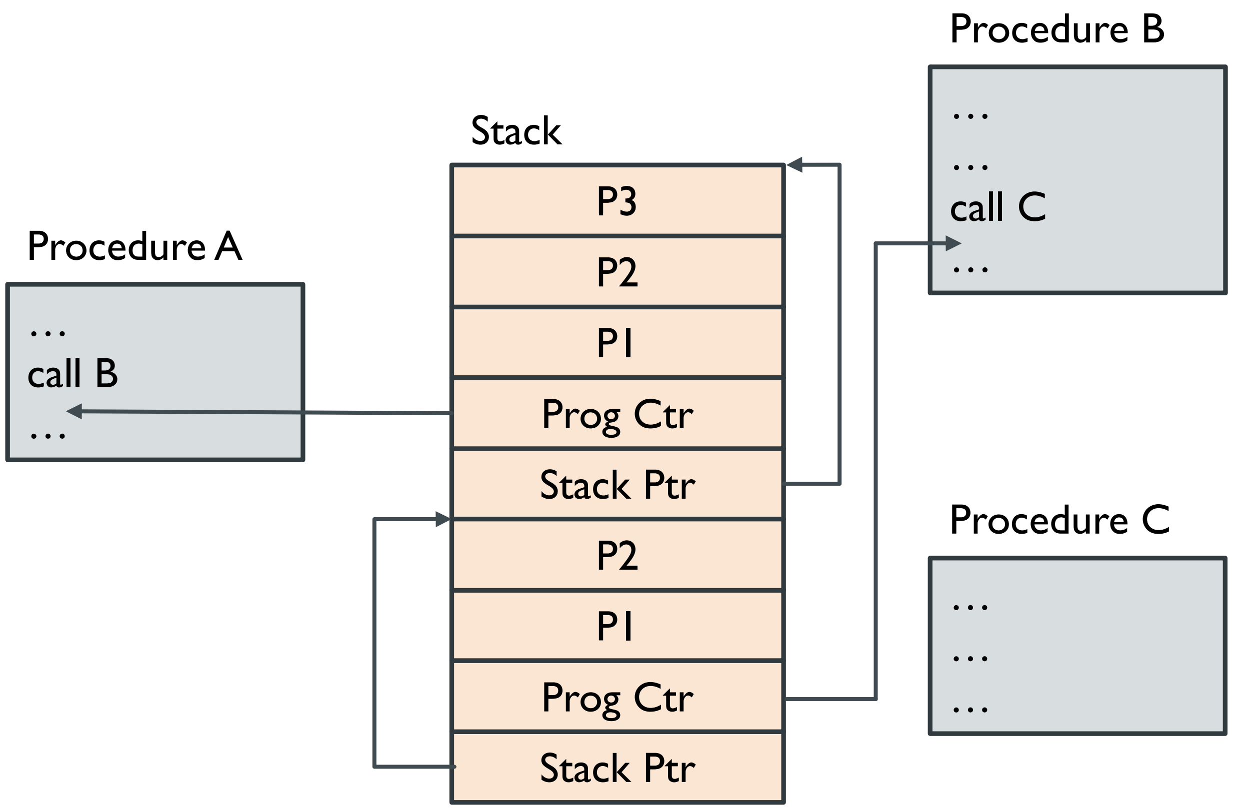
- Stack frame
  - Parameters
  - Program counter
    - Next instruction / return address (the address at which execution should resume when a procedure exits)
  - Stack pointer
    - Pointer to the logical bottom of a program’s section of the stack (to the point just before where a procedure pushed values onto the stack)
    - To help unwind stack data tangled because of a program that fails during execution

THE STACK & THE HEAP



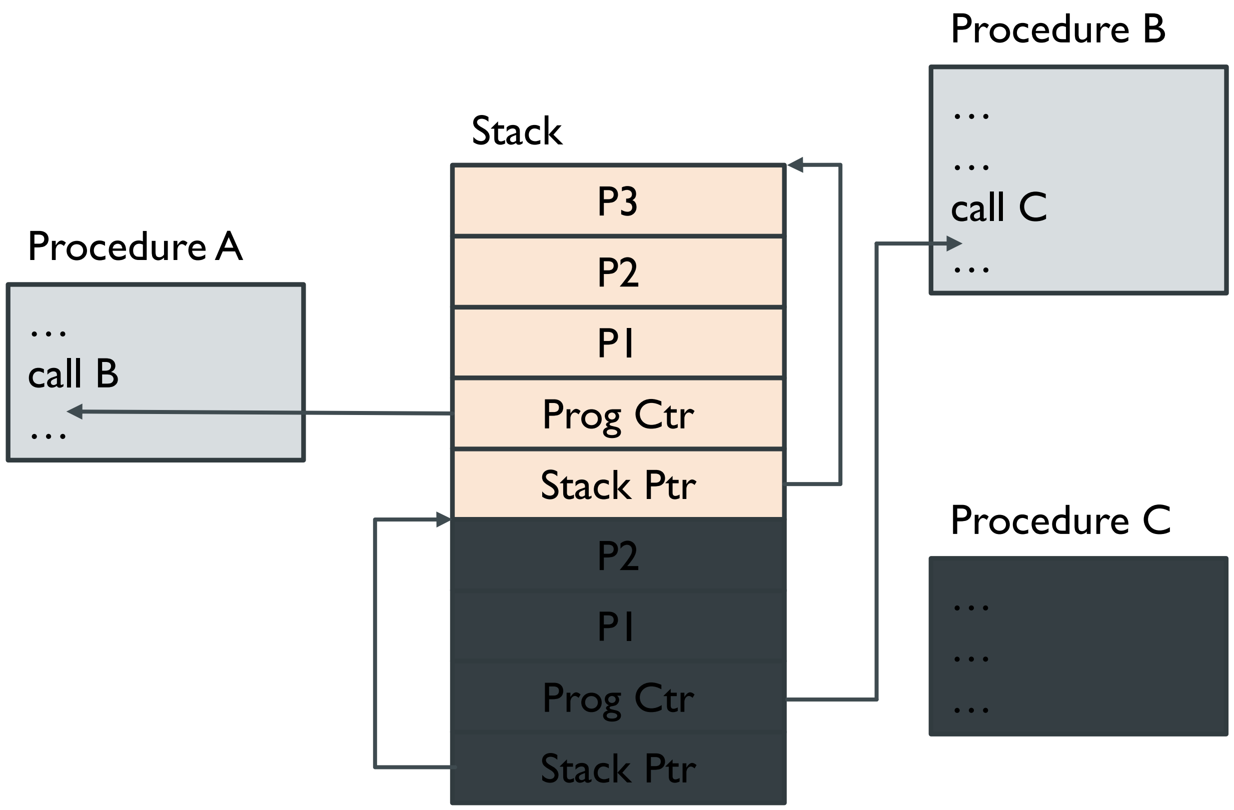
- When one procedure calls another, the stack frame is pushed onto the stack to allow the two procedures to exchange data and transfer control

THE STACK & THE HEAP



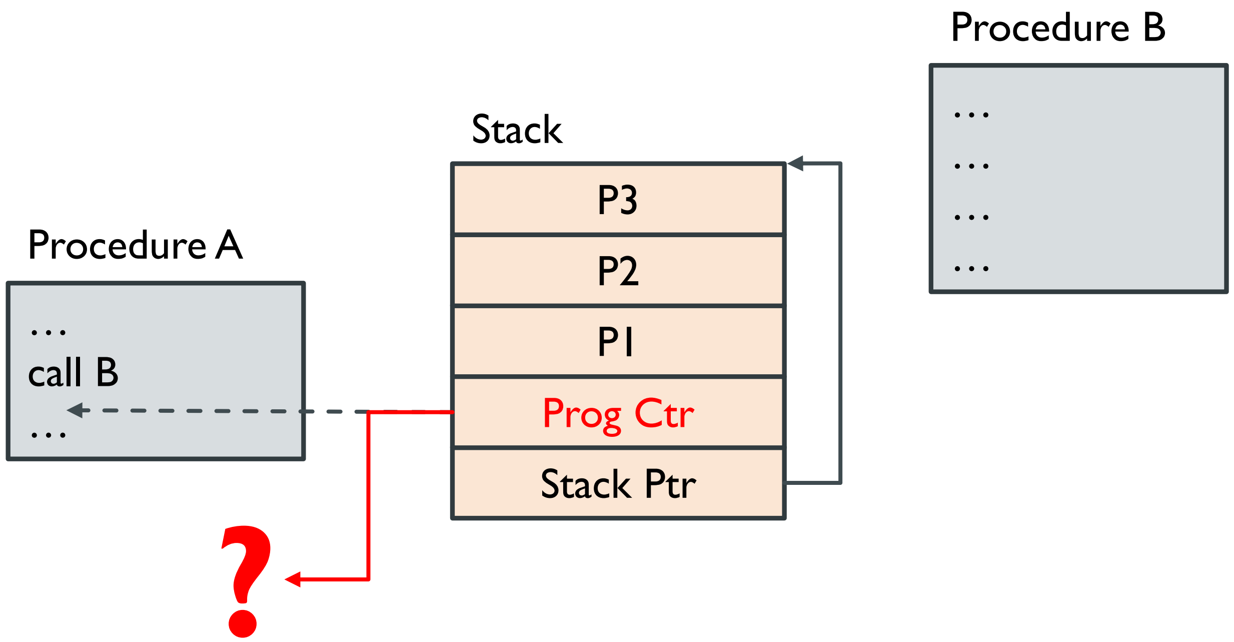
- Procedure A calls B that in turn calls C

THE STACK & THE HEAP



- After procedure C returns to B, the second stack frame is popped off the stack

THE STACK & THE HEAP



- If the attacker can overwrite the program counter, doing so will redirect program execution after the procedure returns

THE STACK & THE HEAP

Stack smashing

The attacker wants to overwrite stack memory in a usable manner

- Arbitrary data in the wrong place causes strange behavior
- Particular data in a predictable location causes a planned impact

- Overwrite the program counter stored in the stack
  - When this routine exits, control transfers to the address pointed at by the modified program counter address
- Overwrite part of the code in low memory
  - Substituting the attacker's instructions for previous program statements
- Overwrite the program counter and data in the stack
  - The program counter now points into the stack, causing the data overwritten into the stack to be executed

OFF-BY-ONE ERROR

- Miscalculating the condition to end a loop
  - Repeat while i <= n or i < n?
  - Repeat until i = n or i > n?
- Forgetting that an array of A[0] through A[n] contains n+1 elements

When the one-character ellipsis symbol "…" available in some fonts is converted by a word processor into 3 successive periods to account for more limited fonts

These unanticipated changes in size can cause changed data no longer to fit in the space where it was originally stored

INTEGER OVERFLOW

- A storage location is of fixed, finite size and therefore can contain only integers up to a certain limit

- The overflow depends on whether the data values are signed

Word Size	Signed Values	Unsigned Values
8 bits	-128 to +127	0 to 255 (2^8-1)
16 bits	-32,768 to +32,767	0 to 65,535 (2^{16}-1)
32 bits	-2,147,483,648 to +2,147,483,647	0 to 4,294,967,296 (2^{32}-1)

With 8-bit unsigned integers

$$255 + 1 = 0$$

UNTERMINATED NULL–TERMINATED STRING

- Variable length character (text) strings are delimited in 3 ways
  - Separate length – Basic and Java

H	E	L	L	O

Max. len.	Curr. len.
20	5

  - Length precedes string – Pascal

5	H	E	L	L	O

  - Null-terminated (string ends with null) – C

H	E	L	L	O	Ø

Suppose an erroneous process happens to overwrite the end of the string and its terminating null character

The application reading the string will continue reading memory until a null byte happens to appear (from some other data value), at any distance beyond the end of the string

PARAMETER LENGTH AND NUMBER

- Too many parameters
- Wrong output type or size
- Too-long string

- If the caller provides space for a 2-byte integer but the called routine produces a 4-byte result, those extra 2 bytes will go somewhere
- A caller may expect a date result as a number of days after January 1, 1970 but the result produced is a string of the form “dd-mmm-yyyy”

UNSAFE UTILITY PROGRAMS

In C the function strcpy(dest, src) copies a string from *src* to *dest*, stopping on a null, with the potential to overrun allocated memory

A safer function is strncpy(dest, src, max), which copies up to the null delimiter or max characters, whichever comes first

PROGRAMMER BOUNDS CHECKING

- Check lengths before writing
- Confirm that array subscripts are within limits
- Double-check boundary condition code to catch possible off-by-one errors
- Monitor input and accept only as many characters as can be handled
- Use string utilities that transfer only a bounded amount of data
- Be suspicious of procedures that might overrun their space

PROGRAMMING LANGUAGE SUPPORT

- The choice of programming language has an impact on security
- Choosing a language depends on many factors, of which security is often a minor consideration
  - Some languages are better for certain types of problems
  - A programmer may feel more comfortable with certain language

PROGRAMMING LANGUAGE SUPPORT

Safe Languages

If the language prevents overflow situations, it is called a safe language

- Memory safety
  - More protective languages prevent direct access to addresses through pointer variables and address functions
- Type safety
  - Transferring arbitrary data into an area of executable code violates the principle of type safety

Safe Compilers

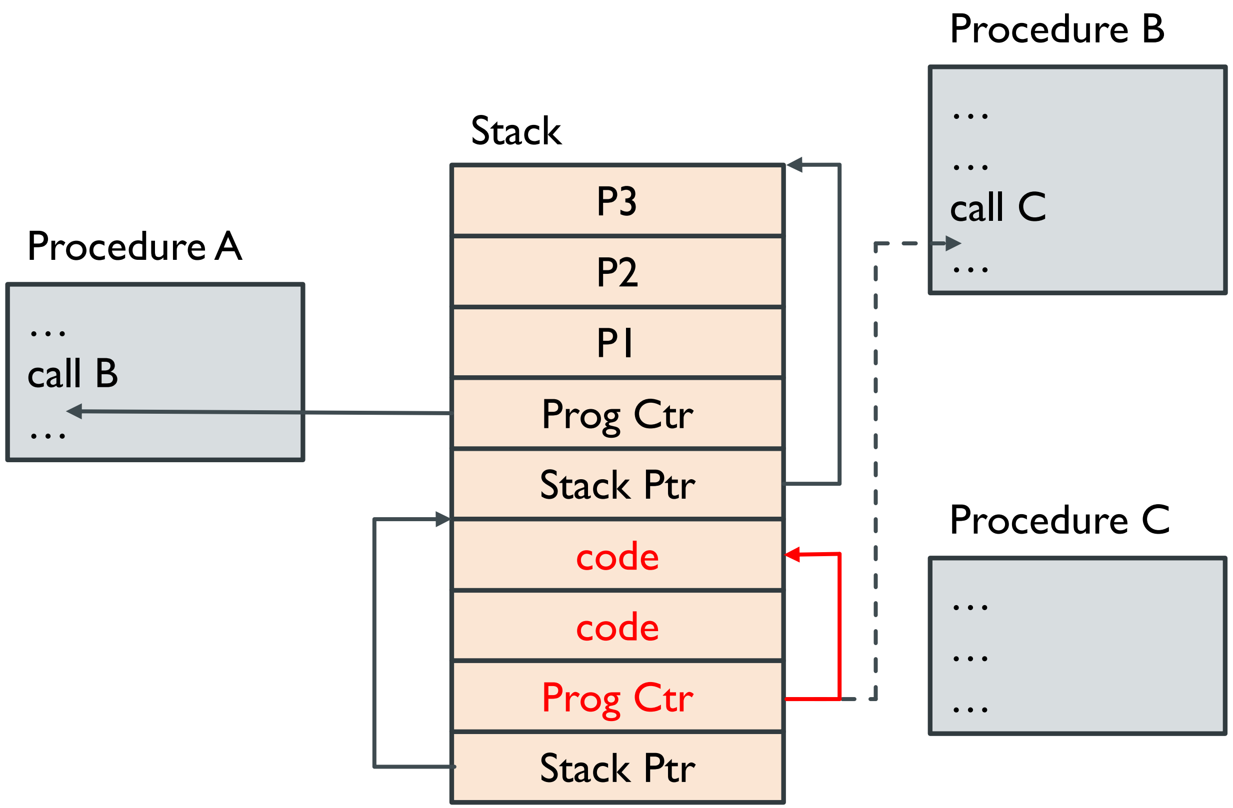
Compilers for less safe languages can generate code that automatically checks for sizes and bounds as a program executes

STACK PROTECTION / TAMPER DETECTION

Wrapping each stack frame
in a protective layer

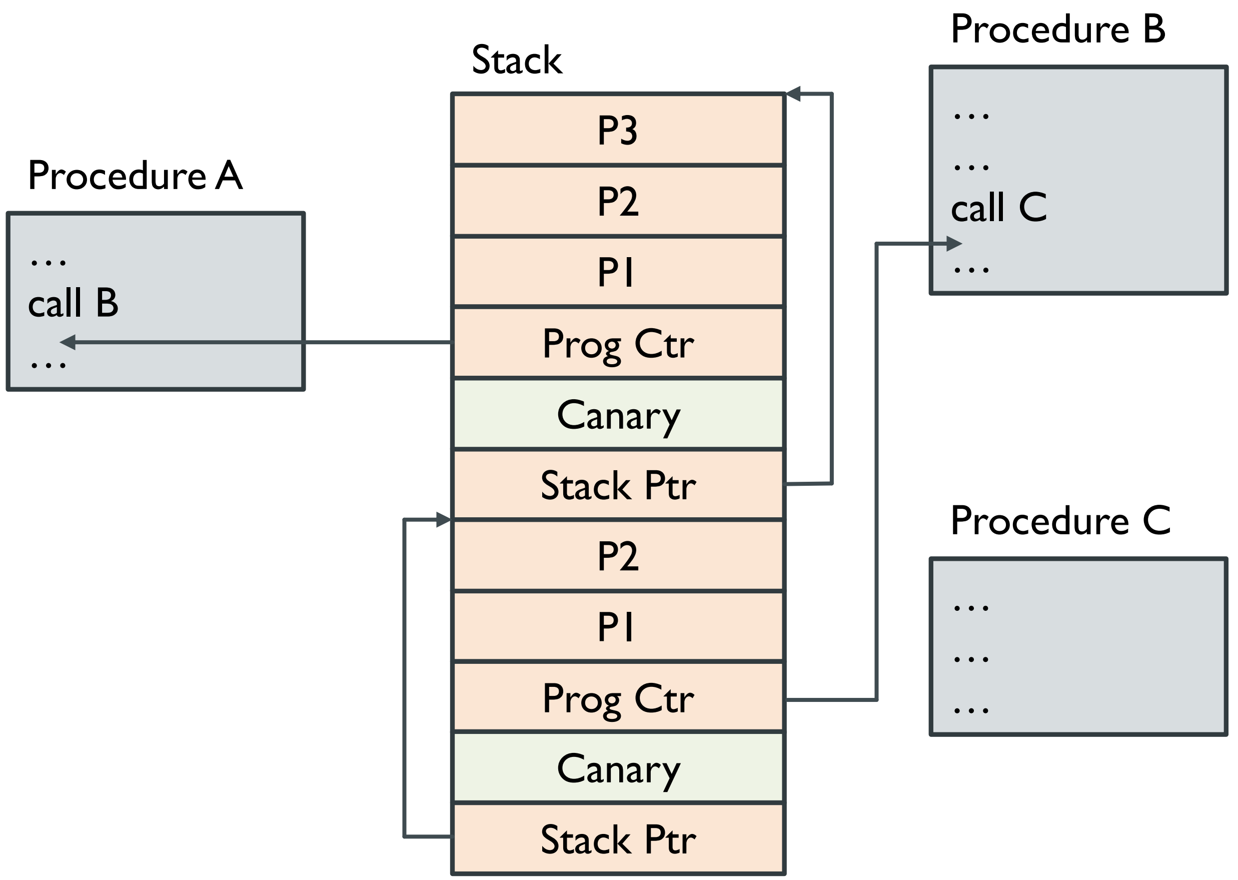
Canary

STACK PROTECTION / TAMPER DETECTION



- In a common buffer overflow stack modification, the program counter is reset to point into the stack to the attack code that has overwritten stack data

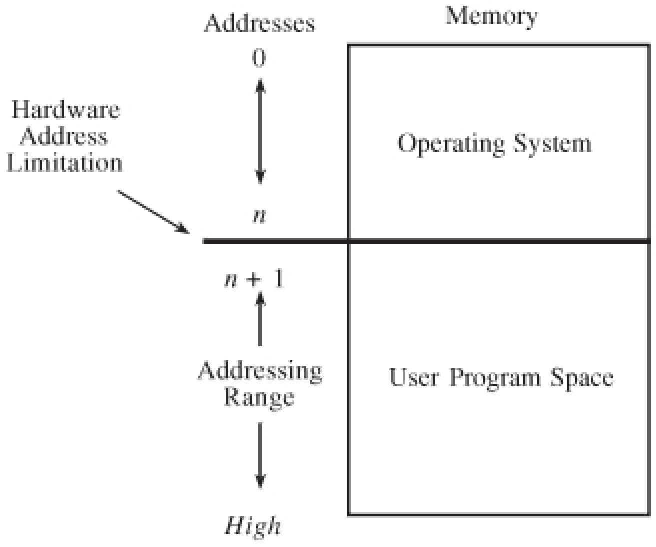
STACK PROTECTION / TAMPER DETECTION



Just below the program counter, StackGuard inserts a canary value to signal modification

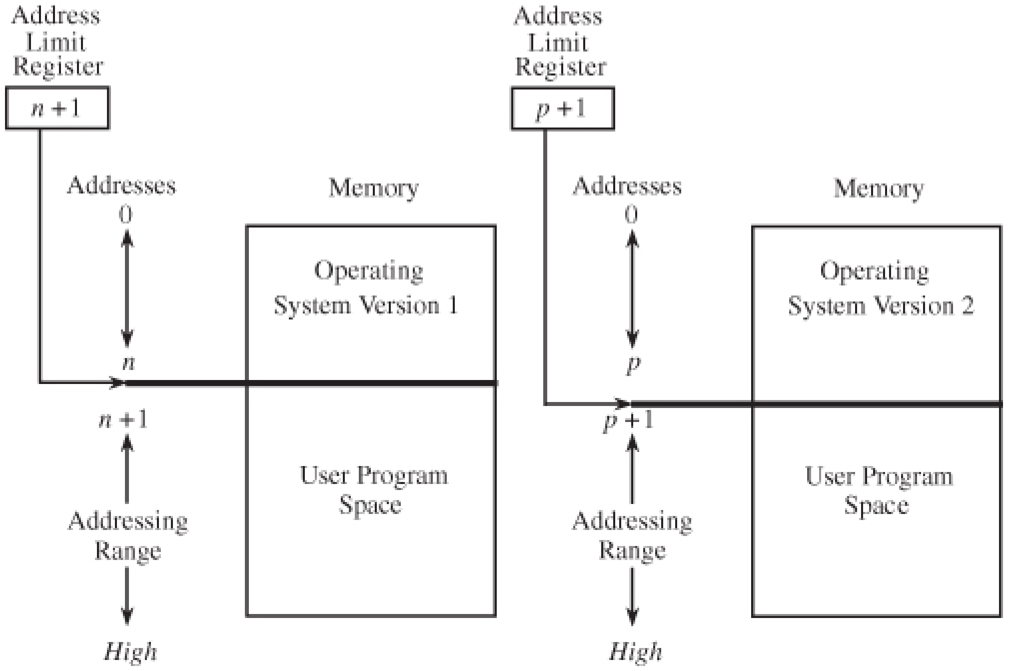
- The attacker usually cannot tell exactly where the saved program counter is in the stack
- The attacker also has to rewrite some words around it
- This uncertainty to the attacker allows StackGuard to detect likely changes to the program counter

HARDWARE PROTECTION OF EXECUTABLE SPACE: FENCE



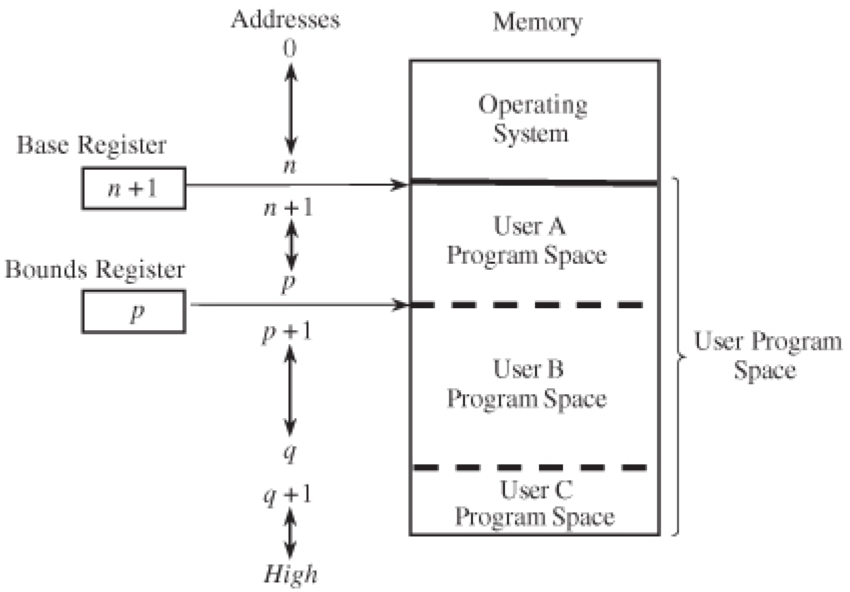
- A method to confine users to one side of a boundary
- To prevent a faulty user program from destroying part of the resident portion of the operating system
- Predefined memory address
  - A predefined amount of space was always reserved for the operating system, whether the space was needed or not

HARDWARE PROTECTION OF EXECUTABLE SPACE: FENCE REGISTER



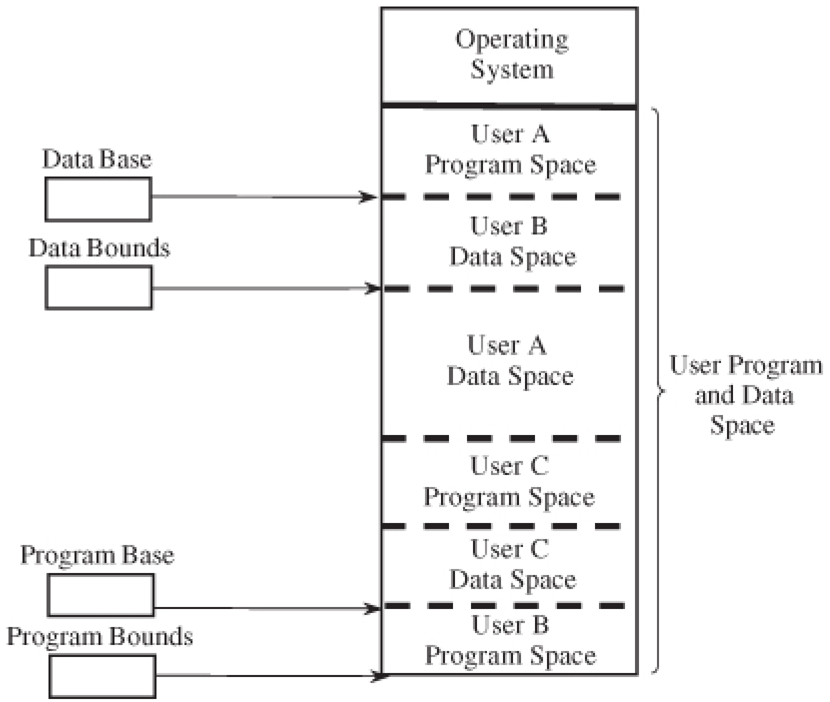
- A hardware register containing the address of the end of the operating system
- The location of the fence could be changed
- An operating system can be protected from a single user, but the fence cannot protect one user from another user

HARDWARE PROTECTION OF EXECUTABLE SPACE: BASE/BOUNDS REGISTERS



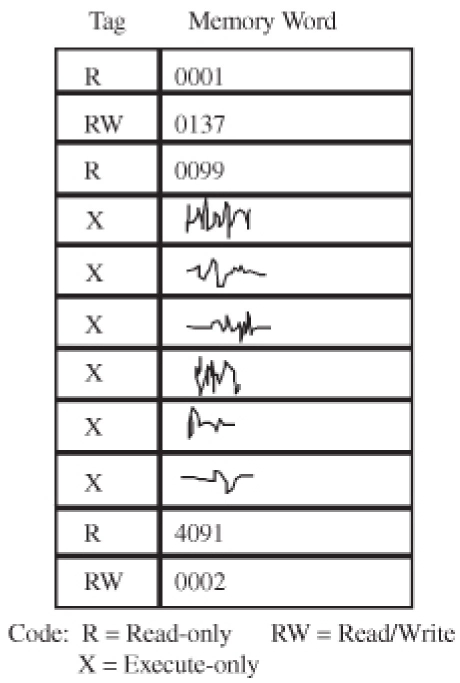
- A variable fence register is generally known as a base register: a lower address limit
- A bounds register: an upper address limit
- When execution changes from one user's program to another's, the operating system must change the contents of the base and bounds registers
- Guarantee only that each address is inside the user's address space
  - Data vs instruction?

HARDWARE PROTECTION OF EXECUTABLE SPACE: BASE/BOUNDS REGISTERS



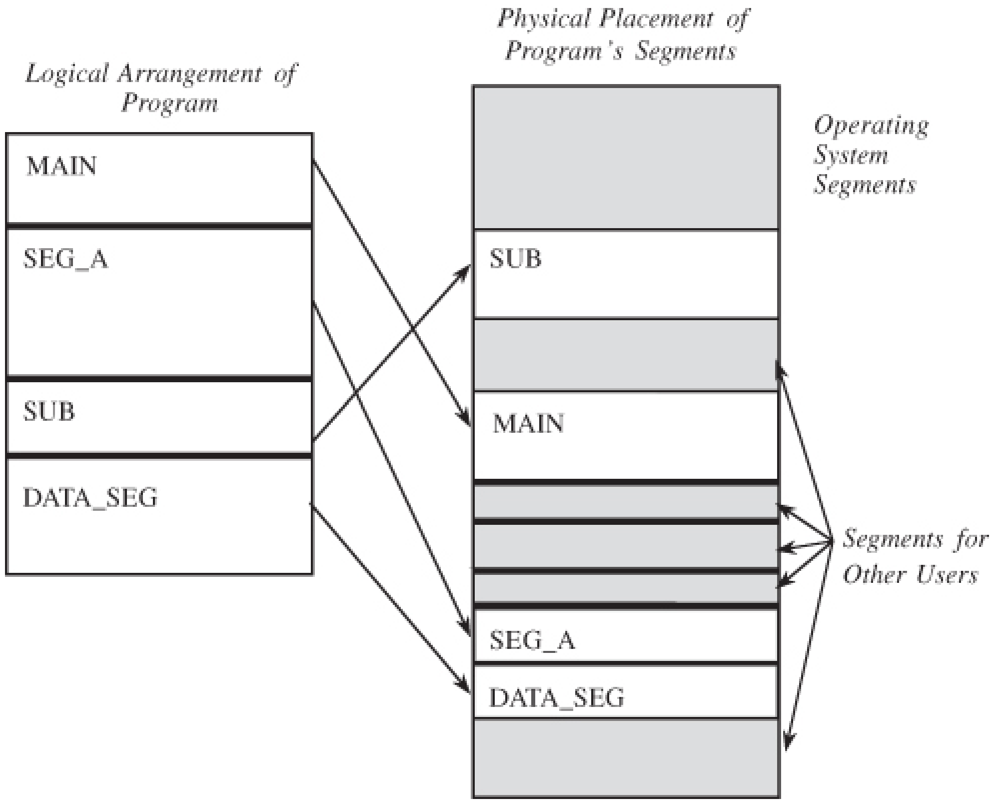
- Using another pair of base/bounds registers
  - One for the instructions (code) of the program
  - A second for the data space
- Contiguous nature
  - Each pair of registers confines accesses to a consecutive range of addresses
- Base/bounds registers create an all-or-nothing situation for sharing
  - Either a program makes all its data available to be accessed and modified or it prohibits access to all

HARDWARE PROTECTION OF EXECUTABLE SPACE: TAGGED ARCHITECTURE



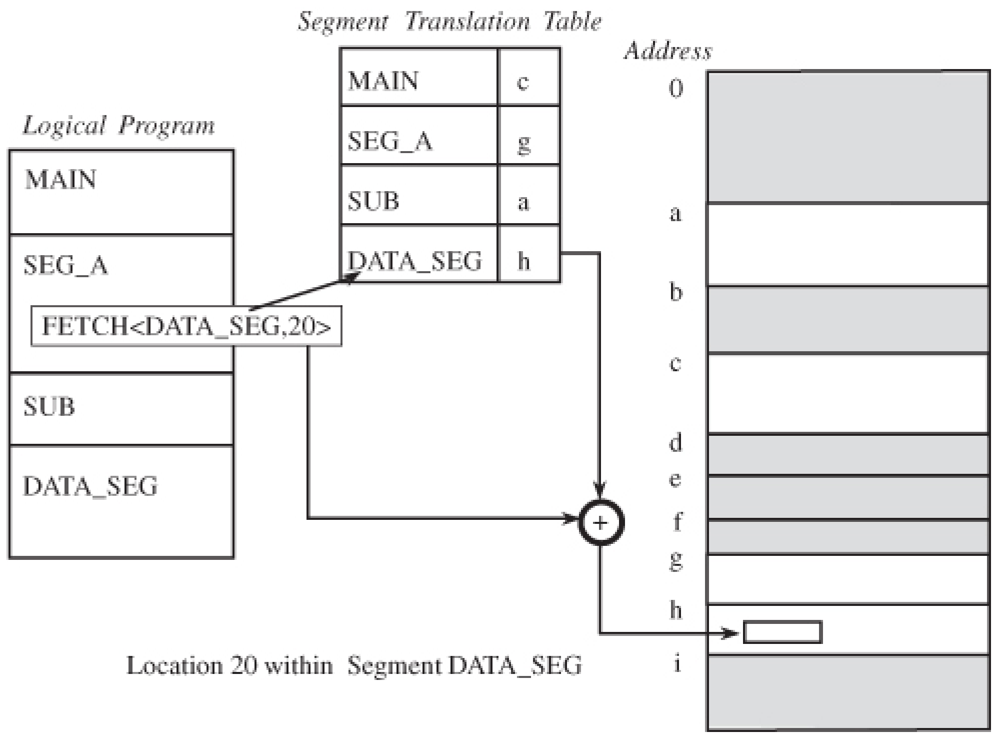
- Every word of machine memory has one or more extra bits to identify the access rights to that word
- Two adjacent locations can have different access rights

HARDWARE PROTECTION OF EXECUTABLE SPACE: SEGMENTATION



- Involves the simple notion of dividing a program into separate pieces
  - Each piece has a logical unity
- Logically, the programmer pictures a program as a long collection of segments
- Segments can be separately relocated, allowing any segment to be placed in any available memory locations

HARDWARE PROTECTION OF EXECUTABLE SPACE: SEGMENTATION



- A code or data item within a segment is addressed as the pair <name, offset>
- A user's program does not know what true memory addresses it uses
- Secure implementation of segmentation requires the checking of a generated address to verify that it is not beyond the current end of the segment referenced
  - Reference <A,9999> looks perfectly valid, but in reality segment A may be only 200 bytes long

HARDWARE PROTECTION OF EXECUTABLE SPACE: PAGING

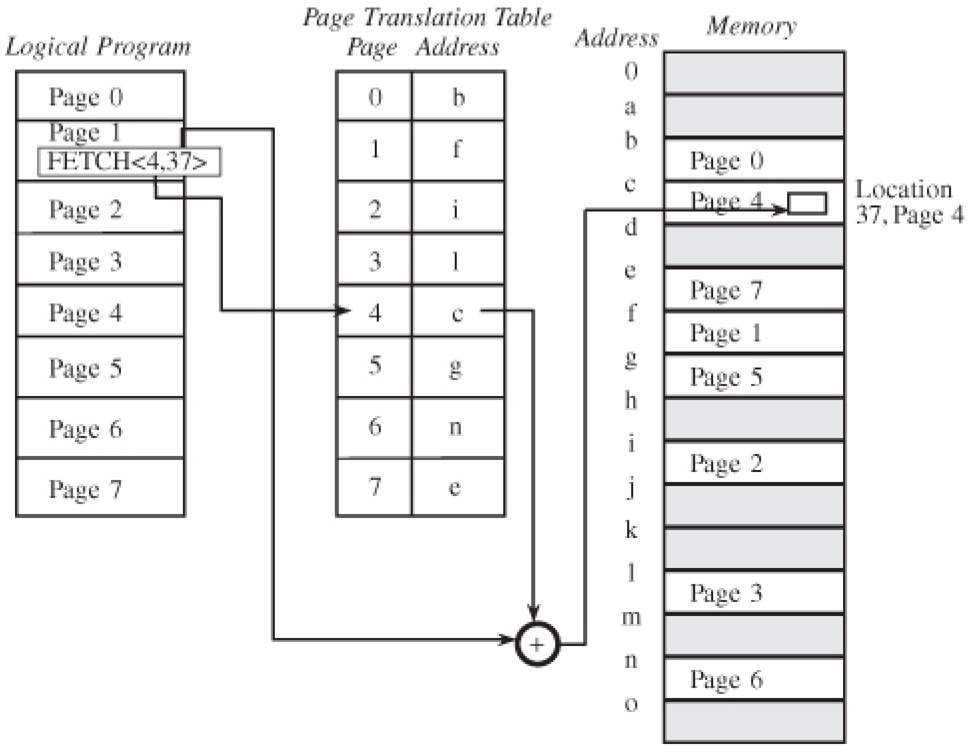
- The program is divided into equal-sized pieces called pages, and memory is divided into equal-sized units called page frames
  - Fragmentation is not a problem
  - Each page can fit in any available page in memory

- For implementation reasons, the page size is usually chosen to be a power of 2 between 512 and 4096 bytes

- There is no logical unity to a page
  - A page is simply the next 2^n bytes of the program
  - There is no way to establish that all values on a page should be protected at the same level

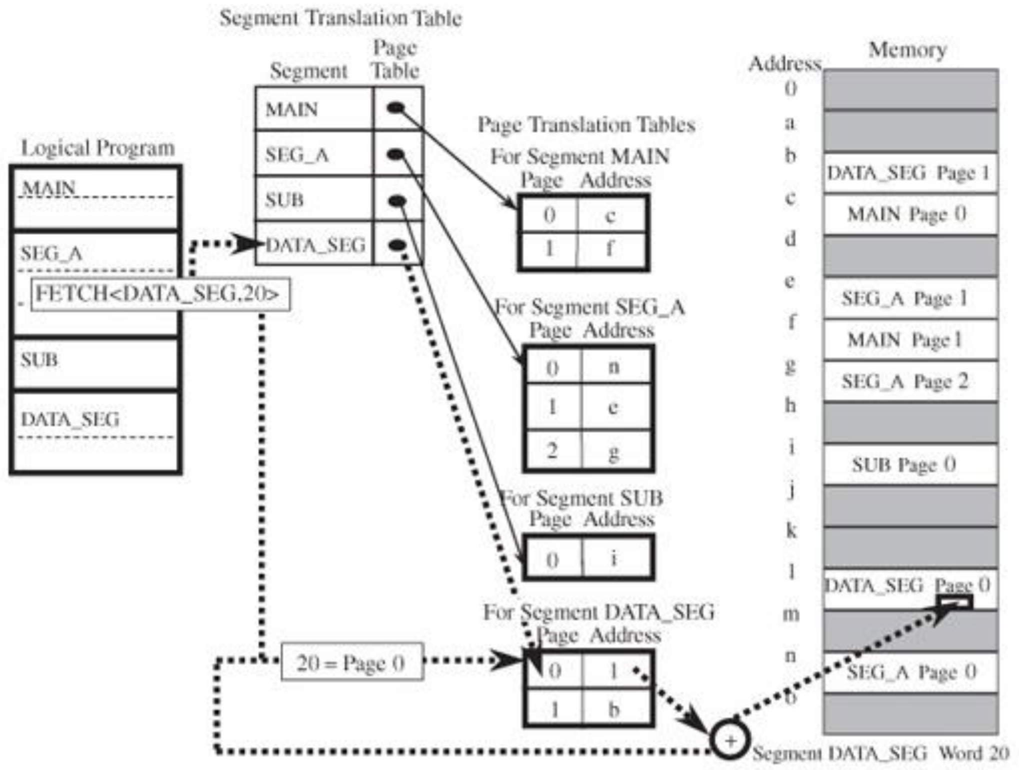
- The entire mechanism of paging and address translation is hidden from the programmer

HARDWARE PROTECTION OF EXECUTABLE SPACE: PAGING



- Each address in a paging scheme is a two-part object, consisting of <page, offset>
- Consider a page size of 1024 bytes ($1024 = 2^{10}$), where 10 bits are allocated for the offset portion of each address
  - A program cannot generate an offset value larger than 1023 in 10 bits
  - Moving to the next location after <x,1023> causes a carry into the page portion, thereby moving translation to the next page
- During the translation, the paging process checks to verify that a <page, offset> reference does not exceed the maximum number of pages the process has defined

HARDWARE PROTECTION OF EXECUTABLE SPACE: COMBINED PAGING WITH SEGMENTATION



- Paging offers implementation efficiency, while segmentation offers logical protection characteristics
- The programmer could divide a program into logical segments
- Each segment was then broken into fixed-size pages

GENERAL ACCESS CONTROL

- Protecting objects involves several complementary goals
  - Check every access
    - In some situations, we may want to prevent further access immediately after we revoke authorization
  - Enforce least privilege
    - A subject should have access to the smallest number of objects necessary to perform some task
  - Verify acceptable usage
    - Check that the activity to be performed on an object is appropriate

GENERAL ACCESS CONTROL: ACCESS CONTROL MATRIX

- A table
  - Each row represents a subject
  - Each column represents an object
  - Each entry is the set of access rights for that subject to that object

objects

subjects

	File A	Printer	System Clock
User W	Read Write Own	Write	Read
Admin		Write Control	Control

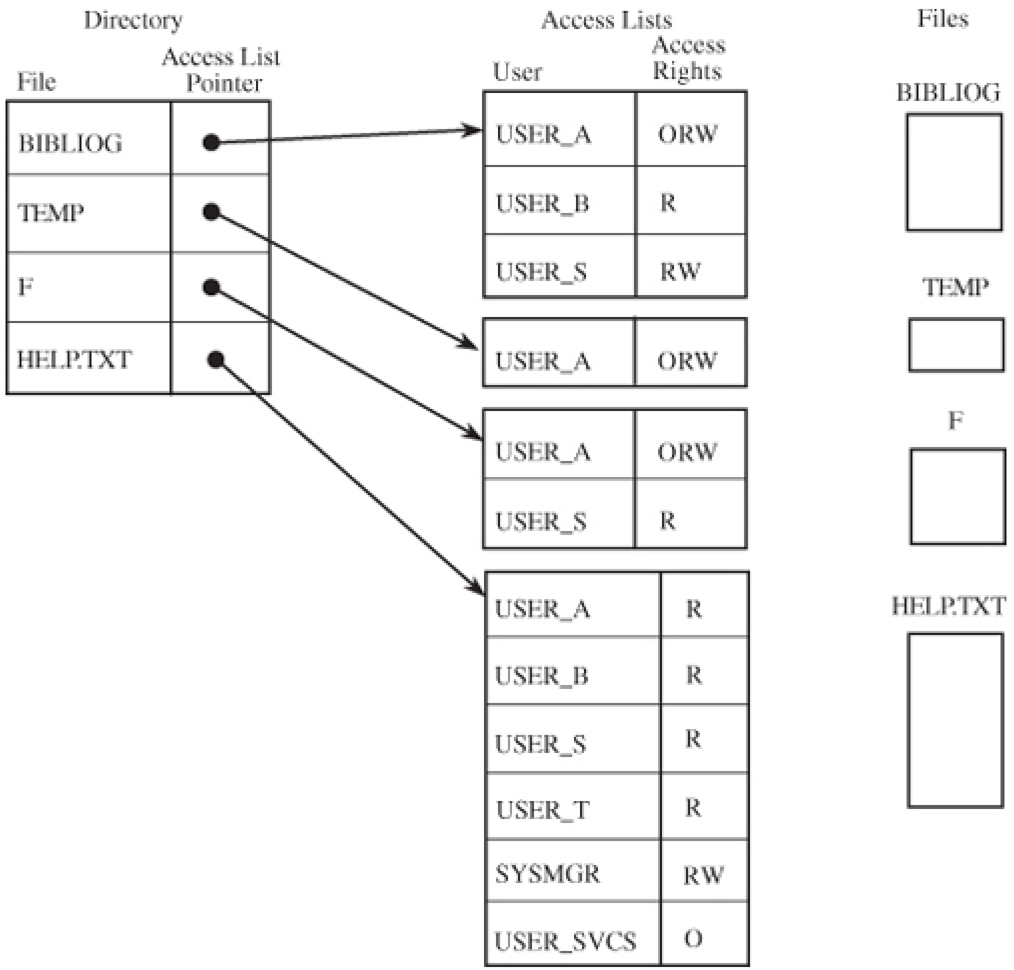
- Most cells are empty
  - Most subjects do not have access rights to most objects

	Bibliog	Temp	F	Help.txt	C_Comp	Linker	Clock	Printer
USER A	ORW	ORW	ORW	R	X	X	R	W
USER B	R	–	–	R	X	X	R	W
USER S	RW	–	R	R	X	X	R	W
USER T	–	–	–	R	X	X	R	W
SYS MGR	–	–	–	RW	OX	OX	ORW	O
USER SVCS	–	–	–	O	X	X	R	W

- Can be represented as a list of triples, each having the form <subject, object, rights>

Subject	Object	Right
USER A	Bibliog	ORW
USER B	Bibliog	R
USER S	Bibliog	RW
USER A	Temp	O
USER A	F	ORW
USER S	F	R
etc.		

GENERAL ACCESS CONTROL: ACCESS CONTROL LIST



- This representation corresponds to columns of the access control matrix

	File A	Printer	System Clock
User W	Read Write Own	Write	Read
Admin		Write Control	Control

- There is one such list for each object
  - The list shows all subjects who should have access to the object and what their access is

GENERAL ACCESS CONTROL: ACCESS CONTROL LIST

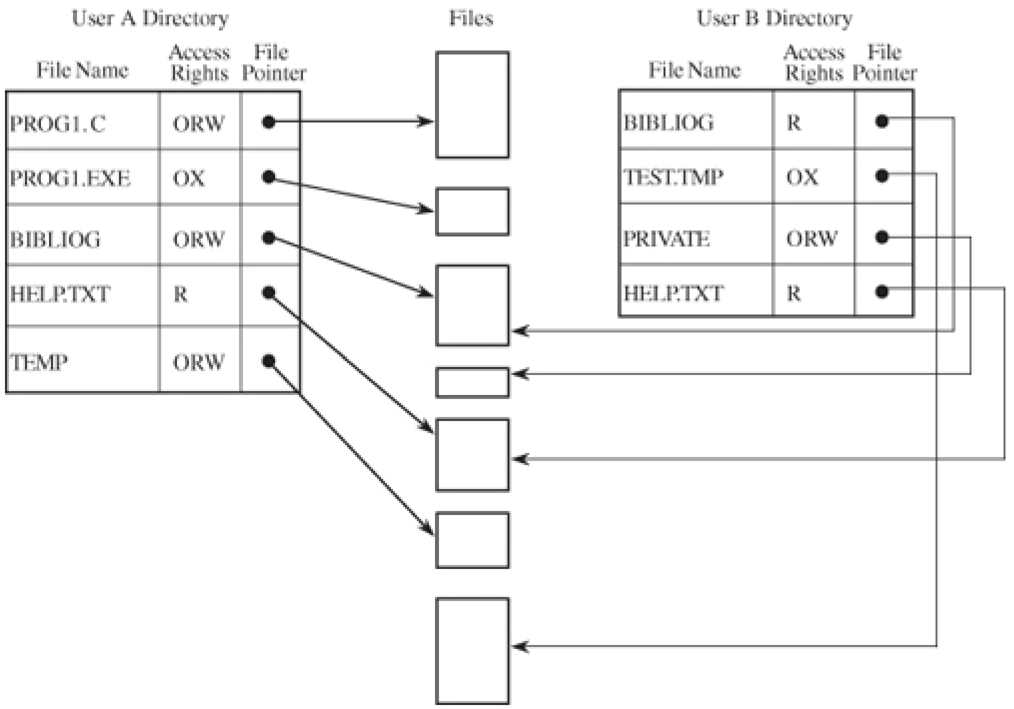
UNIX uses an approach with user-group-world permissions

- The access permissions for each object are a triple (u, g, w)
  - u is for the access rights of the user
  - g is for other members of the group
  - w is for all other users in the world

GENERAL ACCESS CONTROL: PRIVILEGE LIST (DIRECTORY)

- A row of the access matrix, showing all those privileges or access rights for a given subject

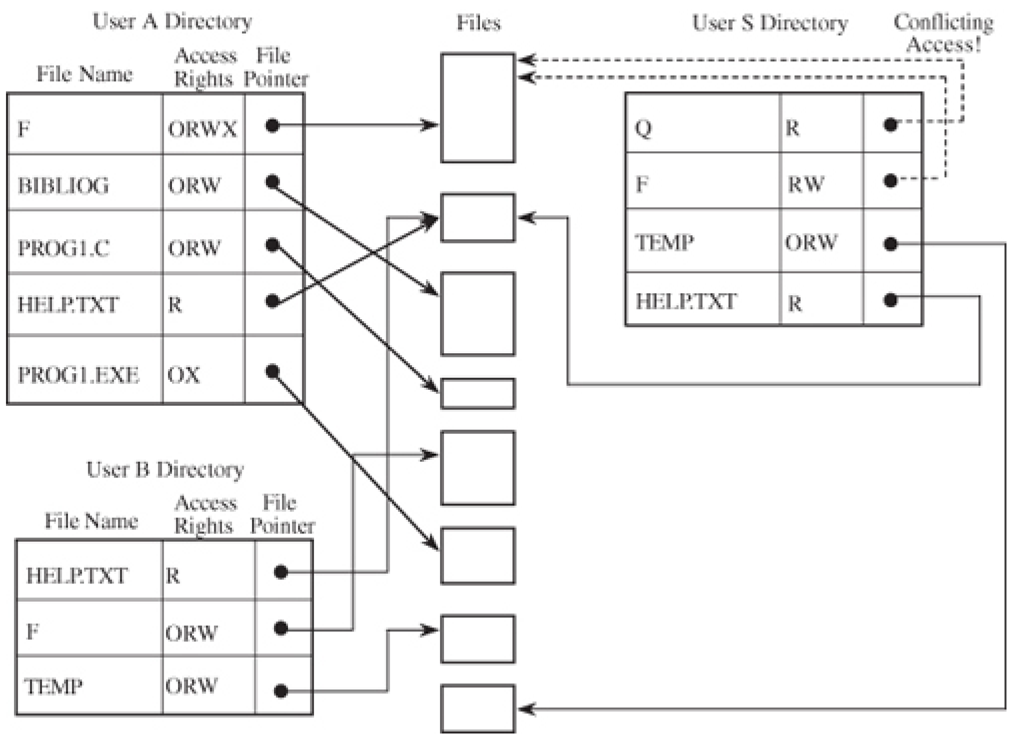
	File A	Printer	System Clock
User W	Read Write Own	Write	Read
Admin		Write Control	Control



- If a user is removed from the system, the privilege list shows all objects to which the user has access so that those rights can be removed from the object

GENERAL ACCESS CONTROL: PRIVILEGE LIST (DIRECTORY)

- Several difficulties can arise
  - The list becomes too large if many shared objects are accessible to all users
  - Revocation of access
    - What if A wants to remove the rights of everyone to access F?
    - Propagation of access rights
      - A has passed to user B the right to read file F
      - B may have passed the access right for F to another user C
      - A may not know that C's access exists and should be revoked
  - Pseudonyms



GENERAL ACCESS CONTROL: CAPABILITY

	File A	Printer	System Clock
User W	Read Write Own	Write	Read
Admin		Write Control	Control

- An unforgeable token that gives the possessor certain rights to an object
  - One way to make it unforgeable is to not give the ticket directly to the user
  - Instead, the operating system holds all tickets on behalf of the users
- You might think of a capability as a ticket giving permission to a subject to have a certain type of access to an object
- A capability is just one access control triple of a subject, object, and right

REFERENCES

- Pfleeger, Charles P. and Shari Lawrence Pfleeger (2012), *Analyzing Computer Security*, 1st Edition, Prentice Hall.

AS THE WORLD IS INCREASINGLY INTERCONNECTED, EVERYONE SHARES THE RESPONSIBILITY OF SECURING CYBERSPACE

Vision

To become an **outstanding** undergraduate Computer Science program that produces **international-minded** graduates who are **competent** in software engineering and have **entrepreneurial spirit** and **noble character**.

Mission

1. To conduct studies with the best technology and curriculum, supported by professional lecturer
2. To conduct research in Informatics to promote science and technology
3. To deliver science-and-technology-based society services to implement science and technology