Security
Operating Systems – EDA093/DIT401

Vincenzo Gulisano
vincenzo.gulisano@chalmers.se

UNIVERSITY OF
GOTHENBURG

Reading instructions

- page 593 -- page 612 (including the part of 9.4.1 included on the last page) page 626 (section 9.6) -- page 645 (NOT Code Reuse attacks) Section 9.9.3 (page 674-676) The Sony Rootkit (page 683 - 684)

Agenda

- Introduction/Overview
- Protection Domains
- Access Control Lists
- Capabilities
- Authentication
- Exploiting software bugs: buffer overflow attacks

Agenda

- Introduction/Overview
- Protection Domains
- Access Control Lists
- Capabilities
- Authentication
- Exploiting software bugs: buffer overflow attacks

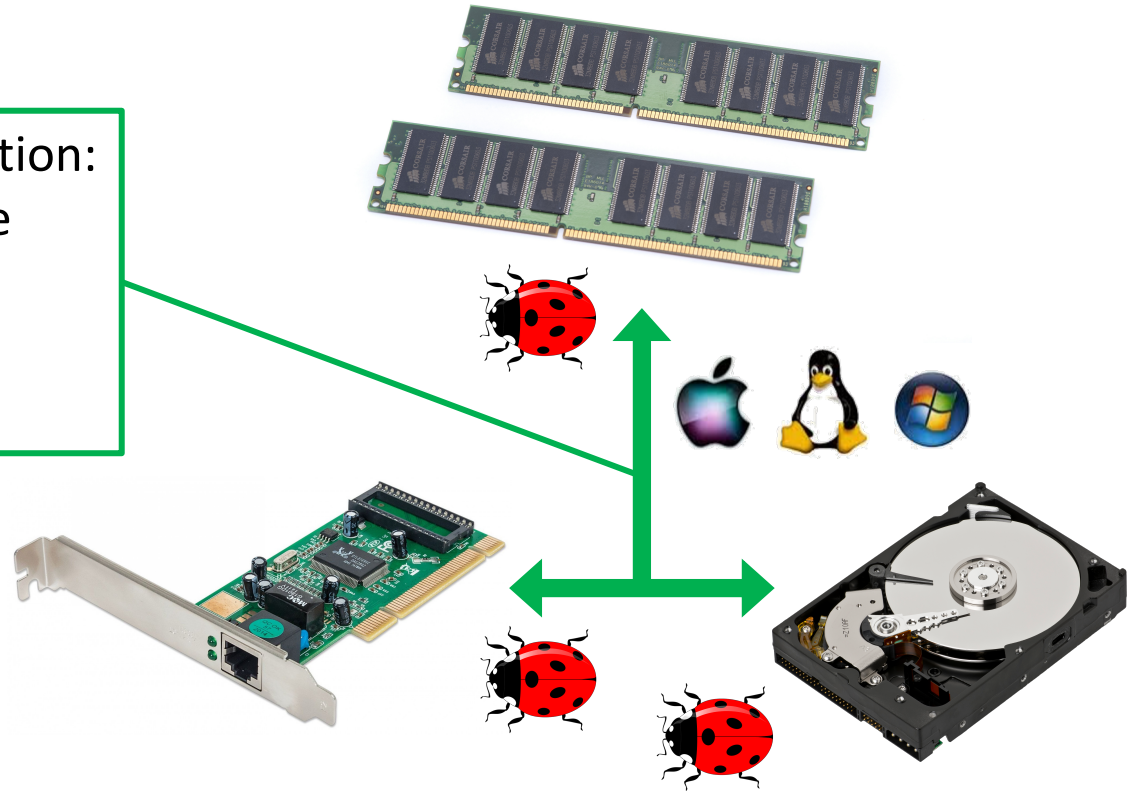
Important information:
- your private one
- commercial
- legal

...

Attacker might want to:
- read
- modify/delete
- prevent owner from accessing it

...

bugs are a way to reach such goals
Vulnerability = security bug
Exploit = bug-triggering input



Features

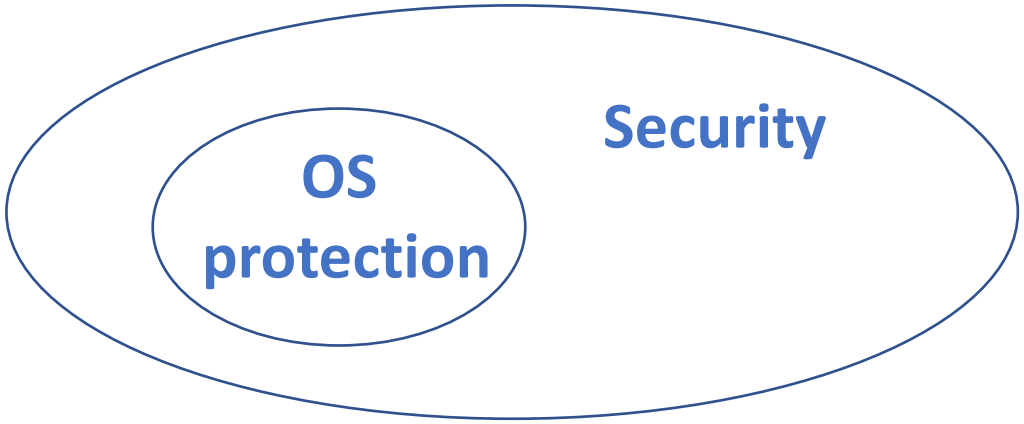
email in which you can attach images and files

Simplicity and security

email offering only plain ASCII text

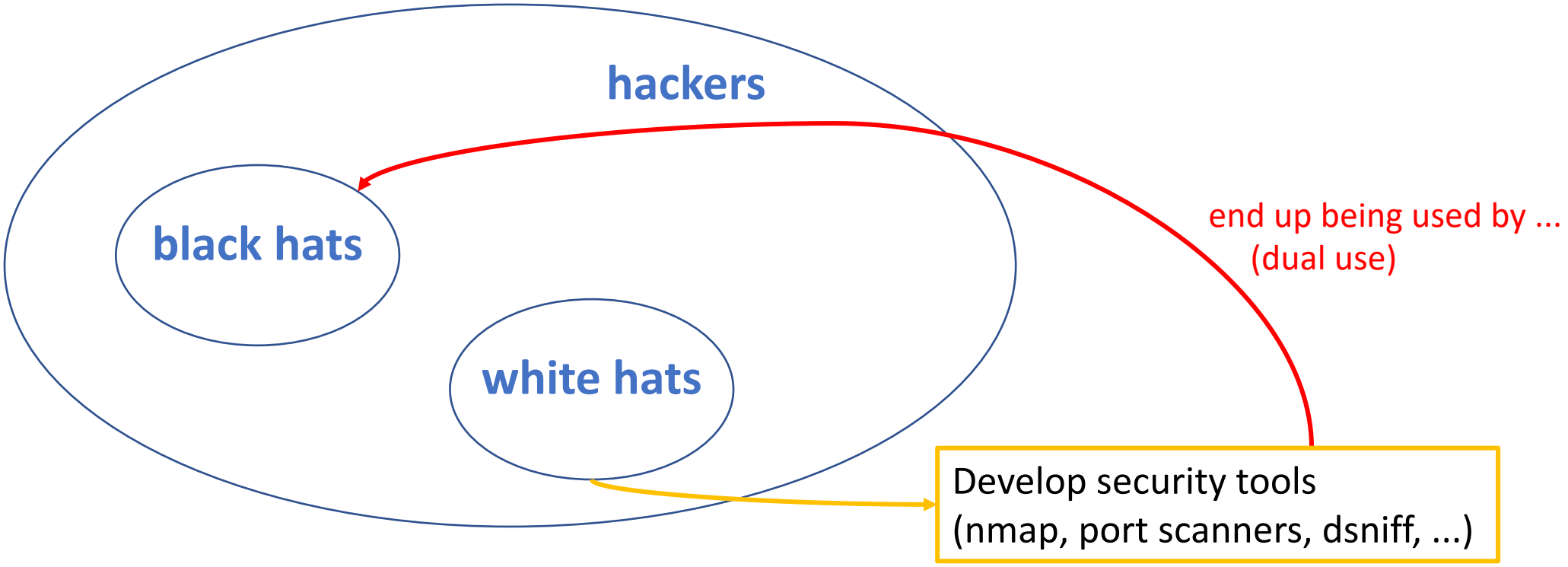
It’s complicated…

- Attacks
  - Worm: freestanding program, propagates without user intervention
  - Virus: attaches to another file/program, propagates through user interaction
  - Trojan horse: malware disguised as legitimate software
  - DoS, DDoS, ...
- Being victim of a security incident does not imply the attacker is after you/your data (e.g., botnets)
- Focusing on the ones related to the OS



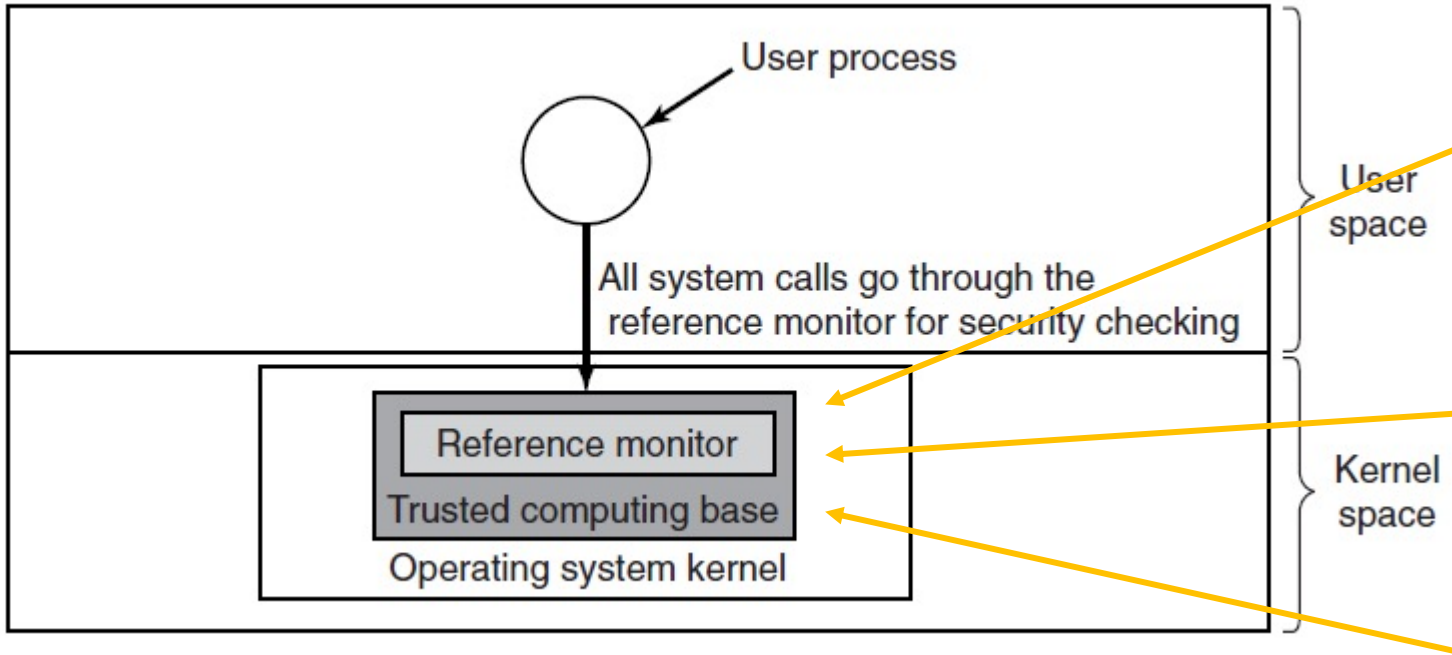
Goal	Threat
Confidentiality	Exposure of data
Integrity	Tampering with data
Availability	Denial of service

There exists of course more: authenticity, accountability, nonrepudiability, privacy, ...


hackers
black hats
white hats
end up being used by ...
(dual use)
Develop security tools
(nmap, port scanners, dsniff, ...)

- Attacks
  - passive (e.g., sniff network traffic and try to break encryption)
  - Active (take control of a browser and use it to execute code)
- Protection mechanisms
  - Cryptography
  - Software hardening
  - And more... (e.g., Intel SGX, a set of security related instructions that allows user processes to have encrypted / isolated parts of memory, that cannot be access by other processes nor by the OS)

Trusted Computing Base (TCB)



hardware (except I/O that does not affect security), portion of kernel (process creation, process switching, memory management, file system, I/O) and user processes with root level

should be as small as possible

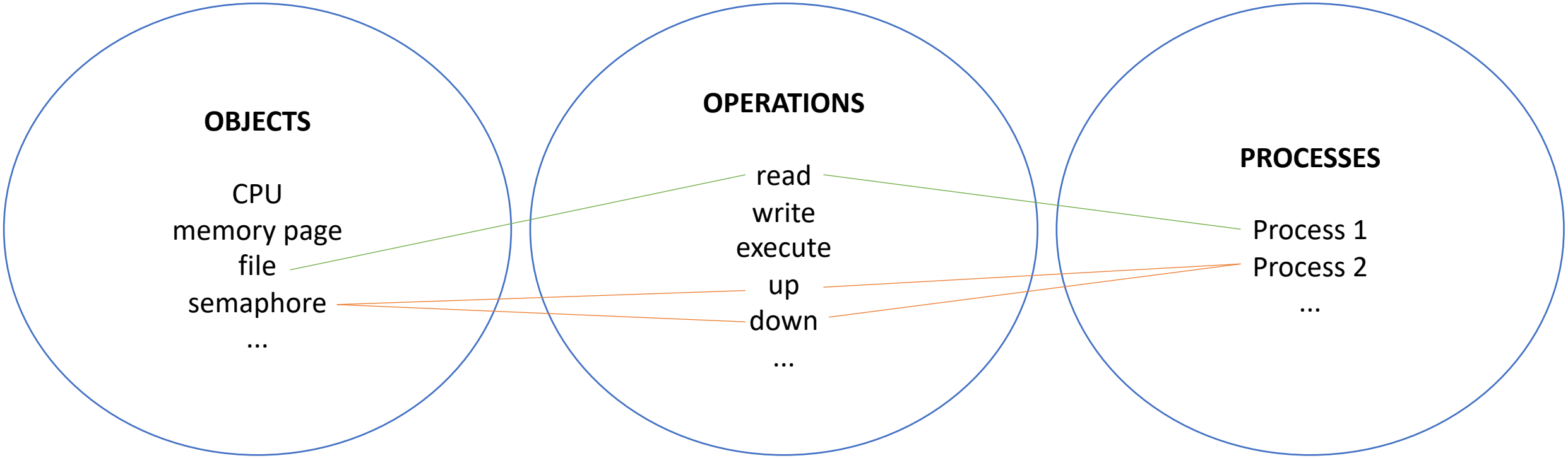
You always need a model:

- what should be protected?
- who can do what?
- what is the power of the adversary?

Agenda

- Introduction/Overview
- Protection Domains
- Access Control Lists
- Capabilities
- Authentication
- Exploiting software bus: buffer overflow attacks

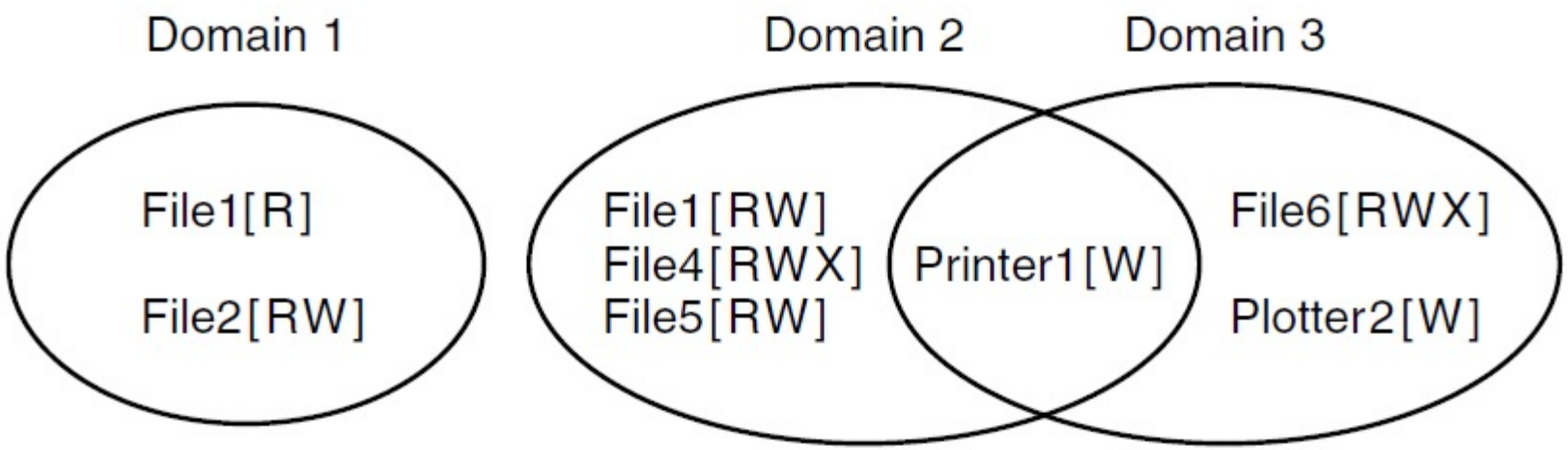
Protection domains



Protection domains

- Domain: set of <object, rights> pairs
- Usually, a domain refers to 1 user, but it can be more general than that (e.g., groups)
- Security works best when each domain has the minimum objects / privileges to do its work (POLA, Principle of Least Authority)

Protection domains



- At each point in time, each process runs in some protection domain
- Each process can in principle switch domain (depending on the system running it, of course...)
- Unix: each process has UID and GID

Domain	Object: File1	File2	File3	File4	File5	File6	Printer1	Plotter2
1	Read	Read Write						
2			Read	Read Write Execute	Read Write		Write	
3						Read Write Execute	Write	Write

Figure 9-4. A protection matrix.

Access Control Lists

Domain	Object: File1	File2	File3	File4	File5	File6	Printer1	Plotter2	Domain1	Domain2	Domain3
1	Read	Read Write								Enter	
2			Read	Read Write Execute	Read Write		Write				
3						Read Write Execute	Write	Write			

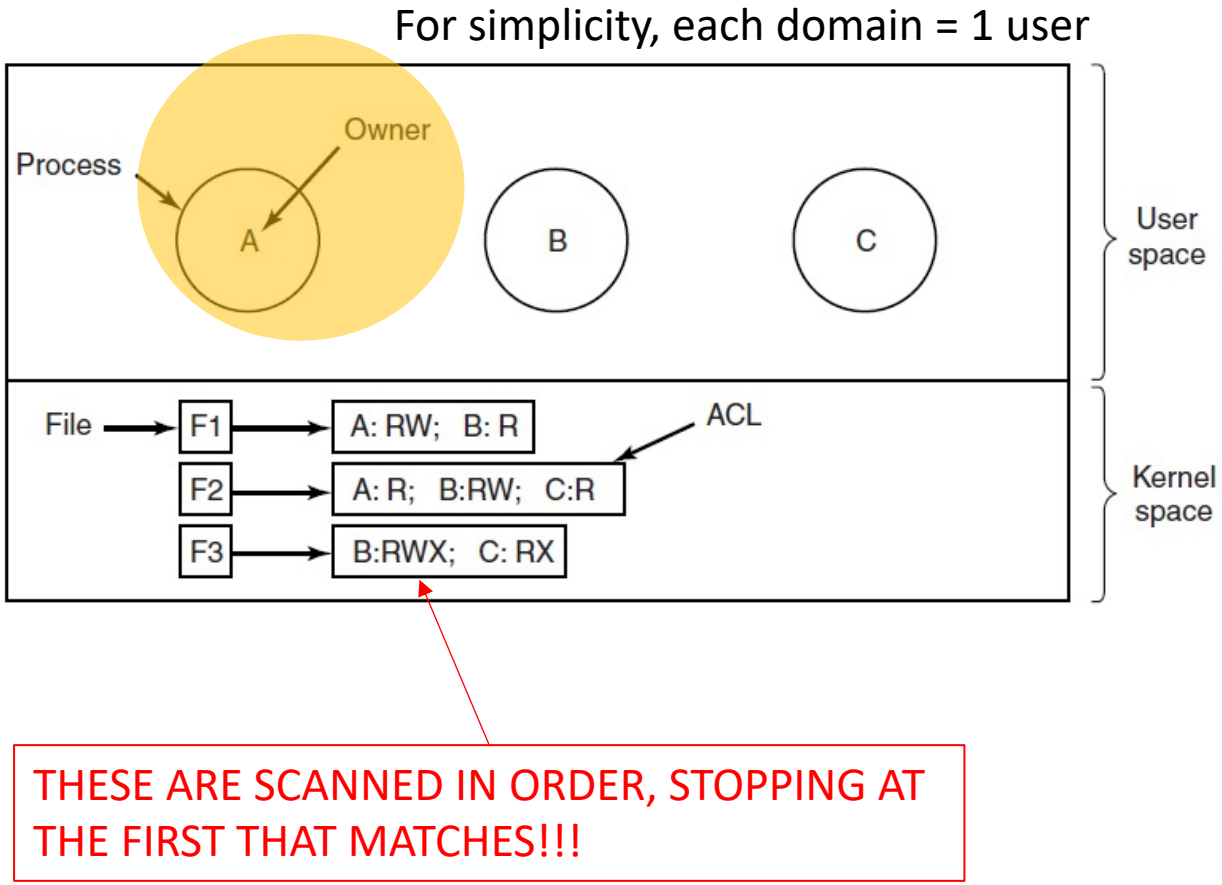
Capabilities

Figure 9-5. A protection matrix with domains as objects.

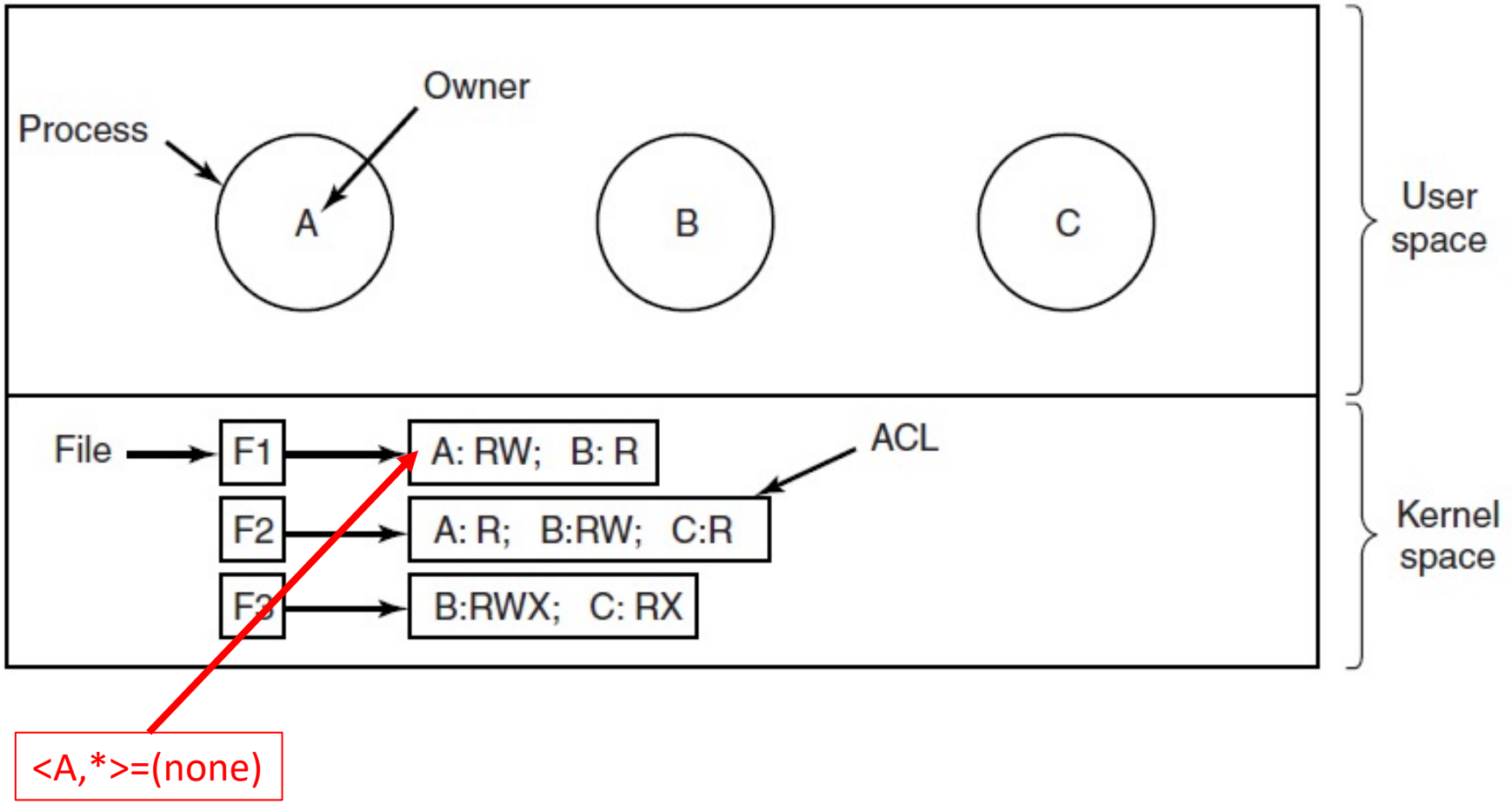
Agenda

- Introduction/Overview
- Protection Domains
- Access Control Lists
- Capabilities
- Authentication
- Exploiting software bugs: buffer overflow attacks

Access Control Lists



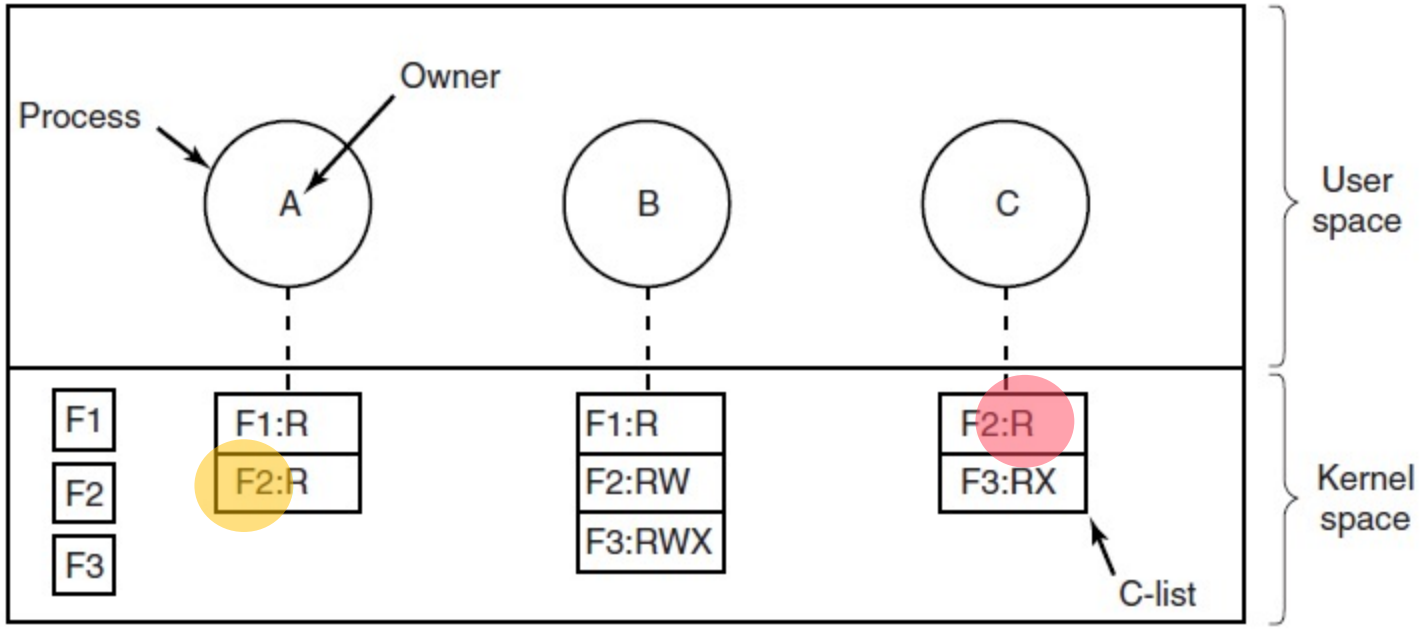
- RWX rights are common, but there could be more
- rights might be per group of users, not an individual user
  - Could be stored as pairs (UID1,GID1):... (UID2,GID2):... – this models *ROLES*
    - Could include wildcards (UID1,*)
  - could be stored individual UID1:... UID2:... GID1:...


Process
Owner
A
B
C
User space
File
F1
F2
F3
A: RW; B: R
A: R; B:RW; C:R
B:RWX; C: RX
ACL
Kernel space
<A,*>=(none)

Agenda

- Introduction/Overview
- Protection Domains
- Access Control Lists
- Capabilities
- Authentication
- Exploiting software bugs: buffer overflow attacks

Capabilities



There can be other rights too: copy / remove/ destroy ...

The object contained in a capability list could be a pointer to another capability list (to facilitate the sharing of subdomains)

Capabilities

- How to prevent a user from tampering/altering capabilities?
  - Tagged architecture: each word in memory has a special bit. If set, word contains a capability, and can only be changed in kernel mode
  - Keep C-lists in OS memory, user processes can point to them
  - Keep in user space and encrypt them so that users cannot modify them

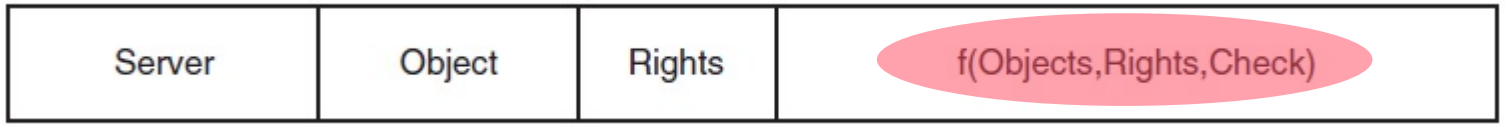


Figure 9-9. A cryptographically protected capability.

If the user alters the Rights, the f() computed by the server / OS and the one in the capability will differ!

Agenda

- Introduction/Overview
- Protection Domains
- Access Control Lists
- Capabilities
- Authentication
- Exploiting software bugs: buffer overflow attacks

Authentication

- A secure system requires authentication
  - Need to know who the user is before enforcing a certain policy (which is user-specific)
- Approaches:
  - something the user knows (password)
  - something the user has (a smart card)
  - something the user is (the owner of a certain fingerprint)

About passwords...

- VERY BAD IDEAS:
  - store <user id, password> to check if the password of a user is correct
  - do not have the BIOS password-protected
  - Naïve mechanisms:
    - complain about the login name being wrong
    - show * while users are typing characters
  - weak passwords
  - default login/password → STUXNET!!!!
  - re-used passwords

Unix authentication (alternative 1, not secure)

- passwords are not stored, what is stored is
  <user id, result of encrypting a fixed block of data with the user password>

- Easy to attack using dictionary attacks
  - Try all common passwords/combinations of words until one matches the encrypted set of characters associated with a user id → password found

Unix authentication + n-bit salt

Bobbie, 4238, e(Dog, 4238)
Tony, 2918, e(6%%TaeFF, 2918)
Laura, 6902, e(Shakespeare, 6902)
Mark, 1694, e(XaB#Bwcz, 1694)
Deborah, 1092, e(LordByron,1092)

- have unique random n-bit prefixes for each user
- Now 1 dictionary is not enough, the attacker needs 2^n dictionaries --> unfeasible

One-time passwords / one-way hash chain

- Use hash function f with input and output having the same length
- Choose a number n (how many passwords)
- $P_1 = \underbrace{f(f(f(f(s))))}_{n \text{ of them}}$
- $P_2 = \underbrace{f(f(f(s)))}_{n-1 \text{ of them}}$
- $P_i = f(P_{i+1})$ --> but you cannot find P_{i+1} from P_i

One-time passwords / one-way hash chain

- Server stores P_0 (which is $f(P_1)$) and value 1
- First login, sends 1 to user, gets P_1, compares $f(P_1)$ with P_0 and, if successful, stores P_1 and value 2

... can be repeated *n* times

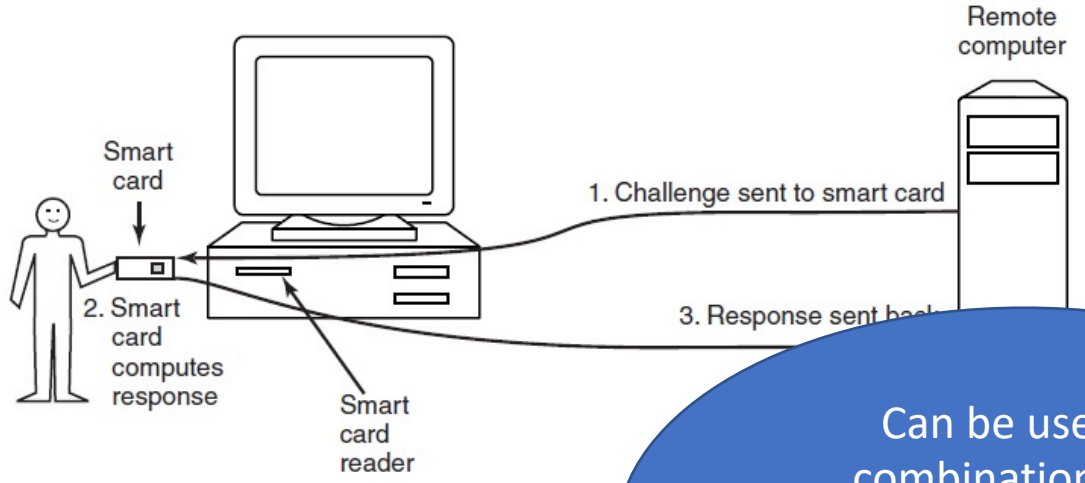
Challenge-Response authentication

- Chose from a long list of questions
  - The user knows the answer, no need to “write it down somewhere”
  - Storage on server must be secure of course
- Let the user choose an algorithm (e.g., x^2) and ask the user to compute the right answer
- If an initial password/key *k* can be shared between user and server,
  - server shares r
  - both user and client compute f(r,k) and compare results – f can be known

  --> with high computing power → complex f and large r/k → hard for the attacker to break

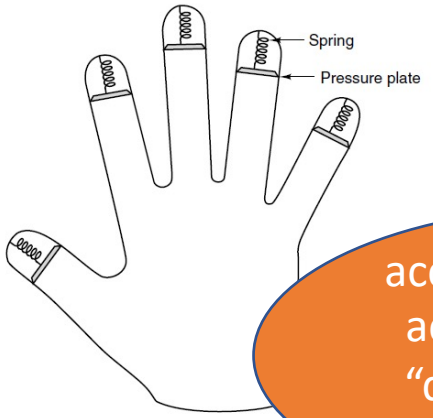
And more

- Authentication using a physical object



Can be used in combination with passwords, of course (multiple techniques can be used at the same time)

- Biometrics



account for variability account for enough “distance” between users

About cards

- Magnetic stripe cards
  - read only
  - can contain the user password
- Chip cards
  - stored value cards
    - value in them can be updated
- Smart cards
  - mount a small CPU
  - Can send messages/encrypting them in some cases
    - This can be useful to authorize payments when there’s no connectivity
  - Can run challenge-response authentication

Agenda

- Introduction/Overview
- Protection Domains
- Access Control Lists
- Capabilities
- Authentication
- Exploiting software bugs: buffer overflow attacks

Exploiting software

- Cracking passwords is only one way of course to gain control
- Alternatives
  - drive-by-download: infect browser, have user download malicious content, malicious content downloads malware, ...
  - **exploit bugs**
    - OS have many...

Buffer overflow attacks

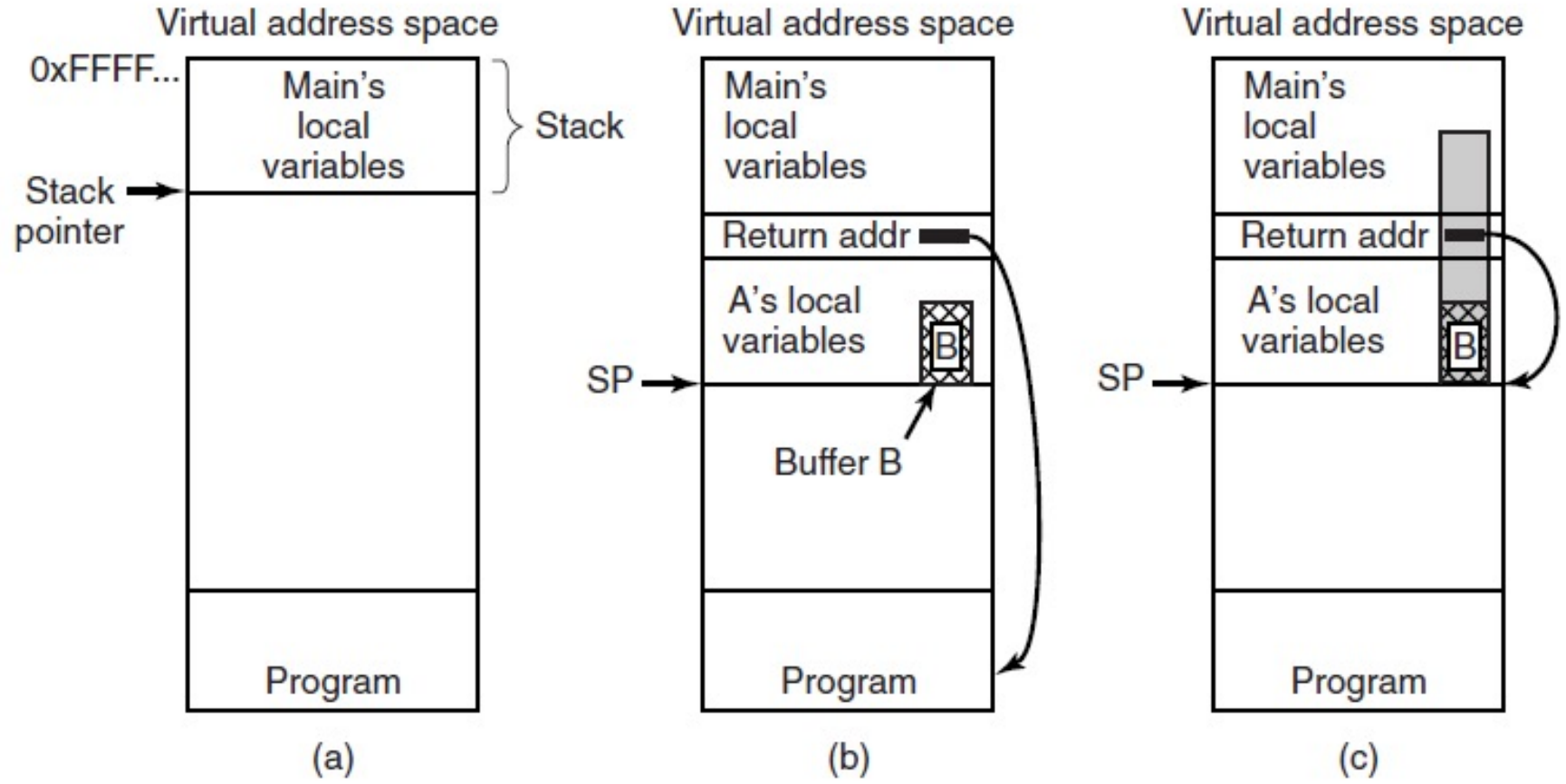
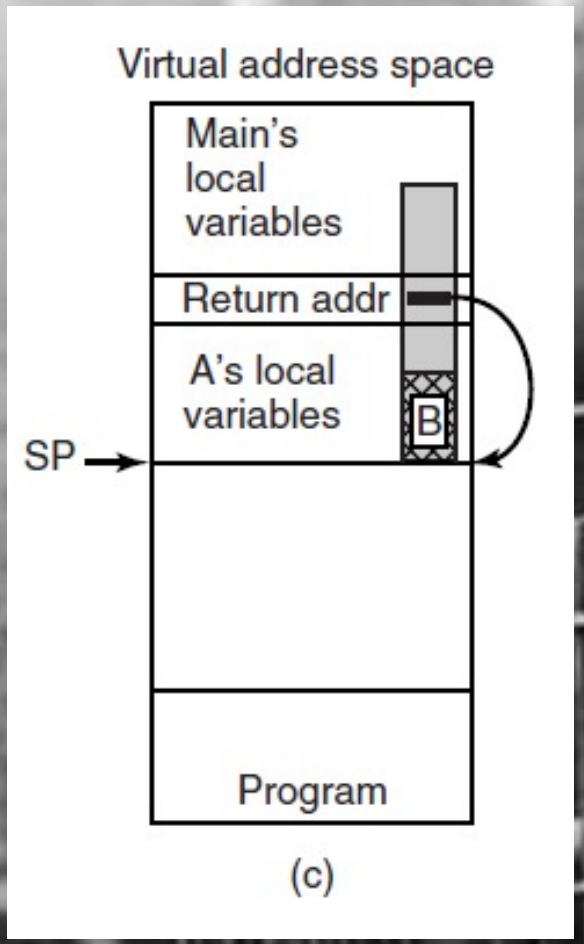


Figure 9-21. (a) Situation when the main program is running. (b) After the procedure A has been called. (c) Buffer overflow shown in gray.

Possible defense: Stack Canaries

- Just below the return address, add random canary value
- When compiling a program, insert code that, upon return from a function, check if the canary value is still correct
- Is it not? → return address has been modified WHP

- NOTICE: stack canaries do not always work, check ”Avoiding Stack Canaries” in section 9.7



Alternative defense strategy (not 100% secure either)

- No-execute/NX bit (hardware)
- hardware bit that ensures data segments are readable and modifiable, but not executable
- Only text section can be executed

- Code injection attacks do not *work*