Computer Security

- Evaluation
  - Surprise Quiz → 20%
  - mid-term → 30%
  - end-term → 50%

- Security.
  → Adversary. ⇒ Hackers / Bad people.
  ↳ Achieving a goal on adversary, known as security.

1. Define policy. → something we want to achieve.
   ↳ * CS grade files can be accessed by class TAs
   1. ↳ Confidentiality. (unauthorized access / to the resources.)
   2. Integrity (unauthorized modification)
   3. Availability. (it should be available whenever needed by the authorized users)
      ↳ eg. the denial Request (when no. of users ↑↑↑↑ @ time t

- Threat Models. → assumptions about adversary.
  → No system is 100% secure
  → we have to make some assumptions
  → for a system to be secured we must have some assumptions.

- Assumption: To set password by giving the required character set.

→ Encryption & Decryption.

* Mechanism: all the implementations done and also testing them. SW / HW / System.
  provided that adversary does not have access to advanced system.

Teacher's Signature.....

Hritik Dusad
16UCS078
CSE

Notes

- What may go wrong?

→ 1) Policy.

Reset pass. → Recovery Questions.

→ Inter-operatability of different policies.

- Threat model.

→ This is the weakest part.

→ since we simply make assumption on the adversary which may not be true.

1. Human factor

→ any user who can access.

— Bad password

— connecting to public Network which may not be secured.

2. Time-Based. (in 1980s)

→ Kerberos → system → secured system

DES → encryption algorithm → with 56 bit.

∴ Total combination of available keys

$= 2^{56}$

→ But Those many keys can be predicted in one day, Hence are not secured at this time.

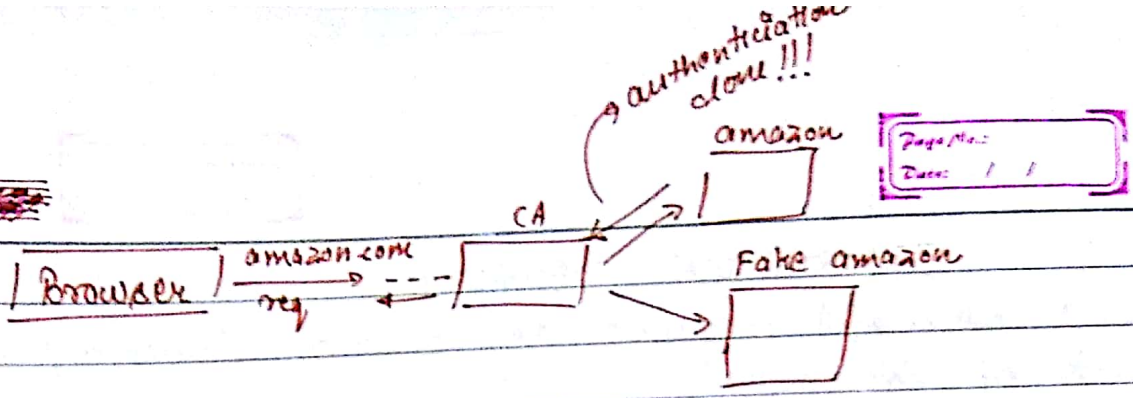
3. machine-Based

→ assuming that due to some problem one system/network can be compromised.

4. SSL/TLS certificate

→ CAs (Certificate authorities.)

Teacher's Signature.....



5) DARPA → Defence agency.

Red Team → attackers

Blue Team → defenders

* mechanism (if not implemented properly).

↳ Files → iCloud ← Find my phone → no limited trials.

↑

Photos

Recovery password : (limited no. of try)

2. Citibank login / Pwd

link: citibank.com / acct ? id = 1234

→ session login was required.

Applications

- Android bitcoins → crypto currency → electronic currency
  - ↳ some theft was done using this.
  - for login: Owner's private coin
  - using Java Secure API. (PRNG (Seed))

- Encoding representation

SSL/TLS ——— 10 amazon.com ———

Browser amazon.com/0.

→ we can insert \0 so that the other websites can be opened.

Teacher's Signature.....

Notes

Page No.:
Date: / /

INFORMATION SECURITY

- Information : meaningful sequence of characters / numbers / symbols.
  - Aspects of information
    1) Accurate.
    2) Timely.
    3) Relevant / contextualize.
    4. Purposeful.
    5. Valuable.

Inf Security : It is the practise of defending information from unauthorized access / usage, disclosure, modification and inspection, recording and destruction.

- Cyber security : Computing Based discipline, involving people information and process to enable assured operations of an organization. It involves the creation, operation analysis and testing of secure computer systems. It is an interdisciplinary course of study including aspects of law, policy, human factor ethics and risks management in the content adversevirons.

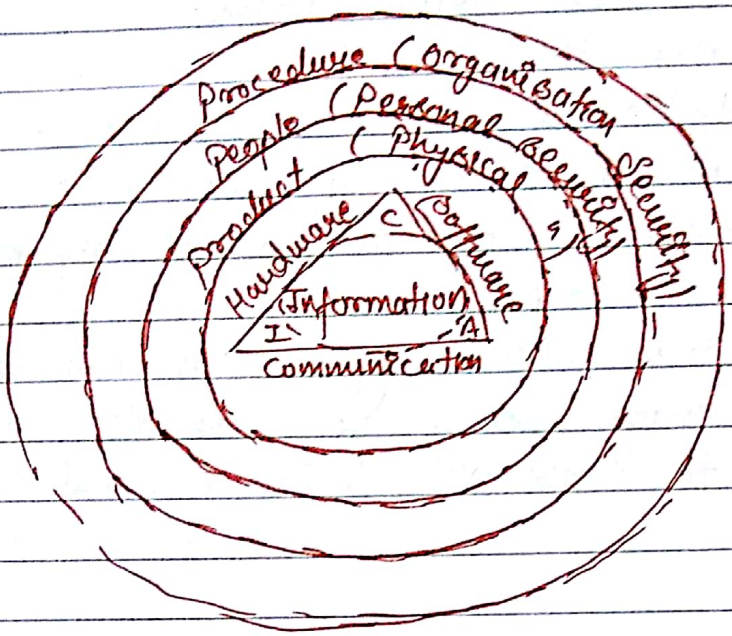
* ~~CIA~~ - Triads
  - CIA (confiden. – Integ. – availability)
  * confidentiality → only authorized users can access information
  * Integrity → ensure completeness accuracy and an absence of unauthorized modification we can determine that a piece of Informa. is provided as requested and verify its providance.

Teacher's Signature.....

Notes

Page No.:
Date: / /

- CIA Treads → model



* Availability: A system and all the system components are available and operational when required by any authorized users.

- RMIS (Reference model for information Assurance and Security)

* Information Taxonomy
  - Form (paper/online verbal)
  - location
  - Stages (Created/designed/stored/Transmitted/Destroyed)
  - Sensitivity (confidential/Highly conf.)

* Security Goals → C, I, A
  - Availability
  - Accountability
  - Authenticity and Trustworthy.
  - non-Repudiation
  - privacy.

Teacher's Signature.....

Notes

Page No.:
Date: / /

- Accountability: It is the ability of a system to hold user responsible for their actions.
- Authenticity: verifying the identity.
- Non-Repudiation: Ability of a system to prove the occurance/non-occurance of an event.
- Auditibility:- Ability to monitor actions by machines/peoples.
- Privacy. : System where organization can have their privacy policy and enable individual to control their personal info.

* Security counter measures
  - Technical
    → Cryptography, encryption, Authentication, Authorization, Organization → Strategies, Policies, Roles
  - Logical.
    → law, contracts, agreement
  - Human-factor.
    → Training., Ethics.

* Security Developments life cycles-
  1. Security Requirement Engineering
  2. Security Design
  3. Security Countermeasures
  4. Security management & monitoring

  (cyclic.)

Teacher's Signature.....

Page No.:
Date: / /

Notes

• CRYPTOGRAPHY

→ nuts and bolts for attaining security mechanism.

→ It is a toolkit of mechanism that can provide different types of security in digital world.

* Why we need cryptography?

Assuming no Internet and nothing
Information can be accessed from
1. Verbal
2. written.

** SECURITY MECHANISM.

1. Secrecy / Confidentiality.
2. not Change. / Integrity.
3. Identifying a person

When the information is digital
we need cryptography

→ We again required the same goals.

2. Data Integrity — conf → encryption codes.
→ Hash functions
→ message Authentication codes (MAC's)
→ Digital Signatures

3. Data-Origin-Authentication
→ Data Origin Authentication not only assures that data has not been changed but also provides some kind of assurance as to who send a data.
can be achieved by (MAC's) or power digital signature.

Teacher's Signature.....

* Authentication of source → In data integrity.

Notes

→ assurance / non-assurance of data.

4. Non-repudiation
   → Digital signature
5. Entity authentication
   → may be person, place / thing.
   → password. / Biometrics
   → smart-Tokens.

- Relationship b/w Security mechanism

(1.) Data-Origin Authentication is a stronger notion than Data Integrity.

Data-Origin Authentication ⇒ Data Integrity
⇍

eg: A sends a message to B.

** NOTES → This can be achieved iff Data integrity is provided of a source

(2) Non-repudiation is a stronger notion than Data-Origin authentication.

→ it also requires data integrity
Data Origin authentication is different from entity authentication

** entity authentication is real time.
we authenticate the entity.
we are authenticating the source of data in DOA.

Teacher's Signature.....

Notes ✍ 3/8/2018 "Friday"

Page No.:
Date: / /

- Cryptography toolkit
→ encryption
→ Digital signatures
→ Hash function
→ MAC (message authentication code)

Symmetric key crypto. → same key for encryption and decryption aka secrete key encryption.

Assymetric key crypto. → diff. keys for encry. and decrypt. One key is private and other is public aka public key encryption.

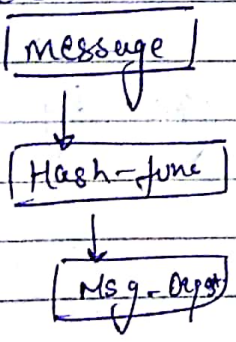
→ continue after 2 pages

* Digital Signatures :-
→ A mark which is made by person P who wants to sign M, so a signature is created as S(P,M)
should follow-

1. Non-Forgeable - [M, S(P,M)] → should be unique.

2. Authenticate - uniquely identify the sender.

Sender



(abstract from ~~memory~~ of message)

Teacher's Signature.....

Notes

Page No.
Date: / /

- encrypt a message using private key and the output will create unique signature

→ if something is encrypted using your private key it can be decrypted using public key and vice-versa.

(3×7) mod 10 = 1

Private ↙ ↘ Public.

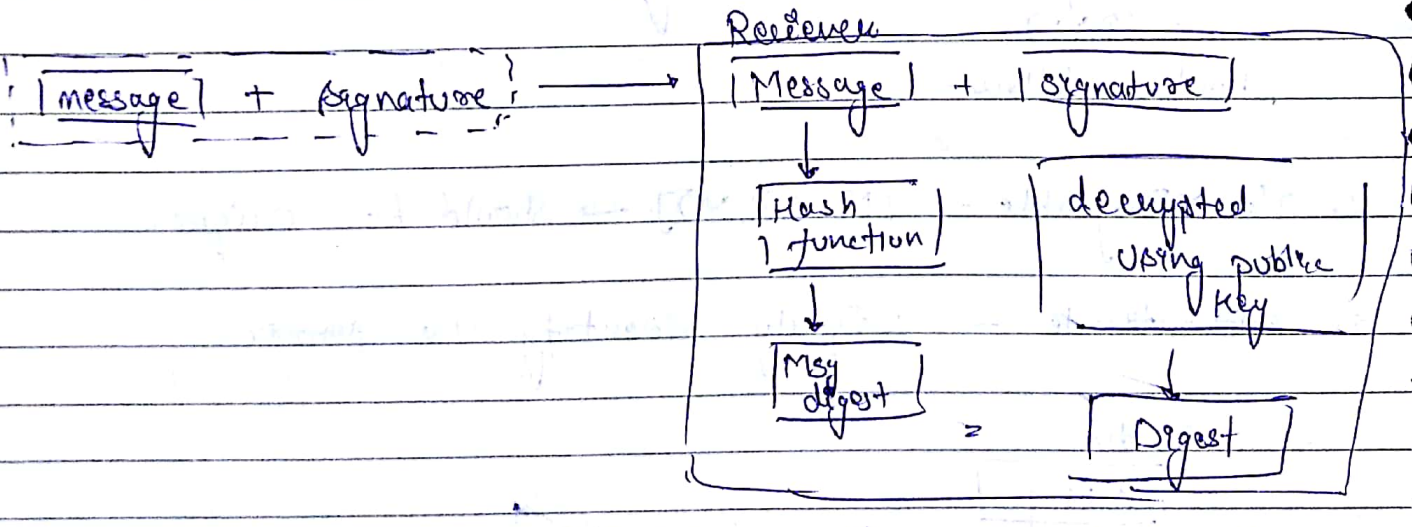
Plain text = 4 (encrypt. using public key)

(4×7) mod 10 = 8 (cipher text)

(8×3) mod 10 = ④ (decryp using private key)

encryp. using private key (4×3) mod 10 = 2

decryp. " public " (2×7) mod 10 = 4



⇒ Message is signed by sender

Teacher's Signature.....

Notes

Page No.:
Date: / /

- Hash func → digital fingerprint of any document.
- Msg digest → to make the system efficient.

When some hash value is generated, if it is encrypted by a secrete key → MAC.

Q.

→ Diving into
- lower security. Buffer Overflow attack by Robert morris.

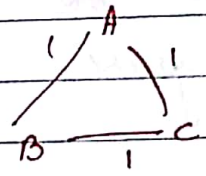
If buffer overflowing, system crashes a string in which a self replicating code will put hold and attack other users also.

Continued. →

1. Stream Cipher (bit/byte) A ——→ B

→ Block cipher
simple, fast
key management
key exchange

eg. DES, AES, Blowfish



Parties	key eq^n
2	1
3	3
4	6
5	10
6	$^6C_2$

Teacher's Signature.....

Notes

Page No.:
Date: / /

22. Public Private key pair.
One way func.
Eg. n

RSA algo (multiplicative Inverse) → Private
$\frac{7 \times 1}{7} = 1$ $(7 \times 3) \bmod 10 =$
↙ if no fraction should be used.
Public

Eg. RSA
$(4 \times 7) \bmod 10 = 8$
$(8 \times 3) \bmod 10 = 4$

→ Key management is pample, each user will keep only a key (public, private)

→ Inspite of cipher, we ~~dtd~~ don't achieve confidentiality. because of fixed systems.
Today we have to tell what security method we are using, so 0 3rd party can guess cipher.

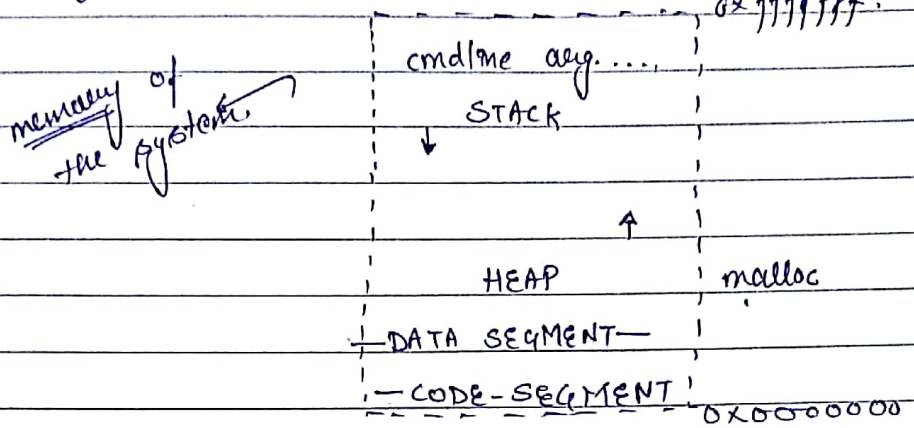
Teacher's Signature.....

Page No.:
Date: / /

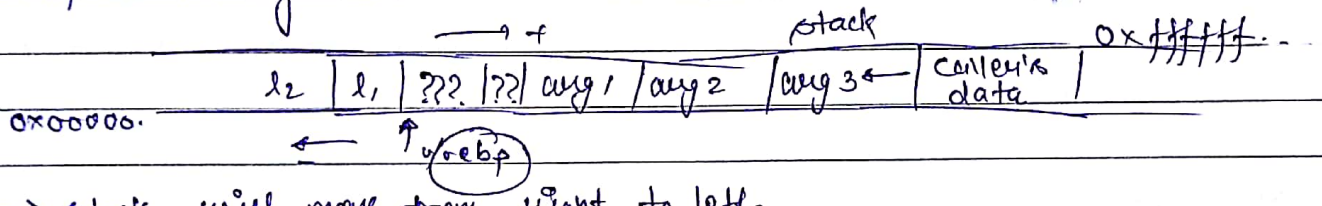
Notes

* System Security

→ for a 32 bit system, we have $2^{32}-1$ addresses in the memory.



If the memory is



→ stack will move from right to left.

→ compiler will push arguments (if any) onto the stack in reverse order.

→ In C the first func°. that is called as "main"

eg.
```
func ( arg1, arg2, arg3) {
    Int l1;
    Int l2;
    l2++;
}
```

** main() { → 1st frame
func (1,2,3); → 2nd frame
func1 (); → 3rd frame for this call
}
func1() { → 4th frame
func();
}

Teacher's Signature.....

Notes

scope of func' and variables.

Page No.:
Date: / /

- **Access variables.**

→ compiler does not know the absolute pointer/address of a variable. It is calculated in response to frame pointer.

→ every time a part/funct code is executed the OS everytime a new frame is created.
The pointer to this frame is known as "frame pointer".

→ EBP register gives the frame pointer. which is newly created.

EBP = %ebp → which is Relative address.

now to get the value of diff. variables we use ebp considering all the data is int.

→ from the stack above ← corresp. to x86

$l_1 = -4(\%ebp)$
$l_2 = -8(\%ebp)$
$arg1 = +8(\%ebp)$
$arg3 = +16(\%ebp)$

NOTE:- we cannot access the absolute address.

→ as soon as the func returns something the corresponding frame is destroyed.

→ As the func is called the current frame is changed to the defination of the func which is called.

- Return from fun:

→ we have to store the caller function's (%ebp). so that when the func returns we can again start from where we left.

Teacher's Signature.....

% ebp → for frame.
% eip → for instruction.
% esp → stack pointer

Notes

Page No.
Date: / /

- Resume previous functions.
  % EIP → stores current value of instruction pointer.
  → This is used as to store where we will resume the execution of the fun. (this gives the idea about instruction in that frame).

**
→ push argument
→ Then % esp
→ Then % ebp
→ Then all local variables
→ all change ~~the~~ will be reflected in Registers

We have 8 Registers but till now only three are of use.
Ebp → frame pointer
Eip → instruction pointer
Esp → stack pointer.

* every time we push
→ first increase esp.
→ Then push.

→ we will set the value of ebp to be equal to esp.

- 1. Caller func
  2. Calling func
  3. Return from fun.

→ Push

Teacher's Signature .....

Notes

Page No.:
Date: / /

1. → Push arguments of calling func. onto stack in reverse order.

2. → Push the return address i.e the address of instruction you want to run after control returns to you
→ Jump to func address.

2. Calling func
→ Put the old frame pointer onto the stack.
→ Set frame pointer (ebp) to where the end of the stack is right now.
→ Push local variables onto the stack

eg.
```
main () {
1.  func (x, y, z);
2
3
4   func1 (z);
5
6
}

4.0 func1 (z) {
4.2   loc1;
4.3   func (4);
      }
```

ret	r.ebp	4.4	x	y	z	loc1	main r.ebp	5	z	main's data

r.ebp

Teacher's Signature.....

Page No.:
Date: / /

3. Return from function.

→ Reset the previous stack pointer

→ Jump back to instruction pointer.

%esp = %ebp

%ebp = (caller's %ebp)

%eip = +4(%esp).

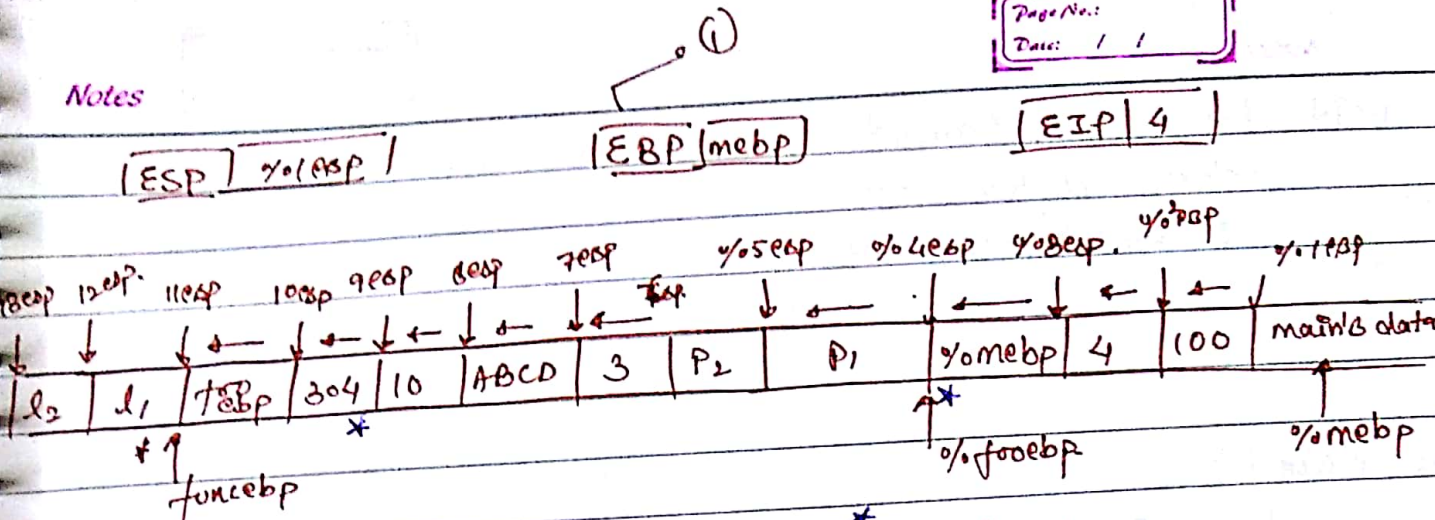
```
* func (arg1, arg2, arg3) {
  2.1 l1;
  2.2 l2;
  2.3 l2+1;
  2.4 ----
}
  foo (arg1) {
    3.1 p1;
    3.2 p2;
    3.3 func (10, "ABCD", 3);
  }

  int main() {
  ① q1;
  ② func (8, "XYZ", -10);
→ ③   foo (100);
  ④ q1++;
}
```

| %ebp | 4 | 140 | main's data

write the memory stack for ans. ③

Teacher's Signature.....

Notes



update in current ebp @ %0esp = fooebp

update in esp eip = 304

update in ebp = funcebp

assuming that 304 is returning statement

Then following changes.

~~%0esp = %0funcebp.~~

when returning

%0esp = (%0ebp) current

%0ebp = (%0esp)

ebp = foo's ebp.

%eip = 4(%0esp)

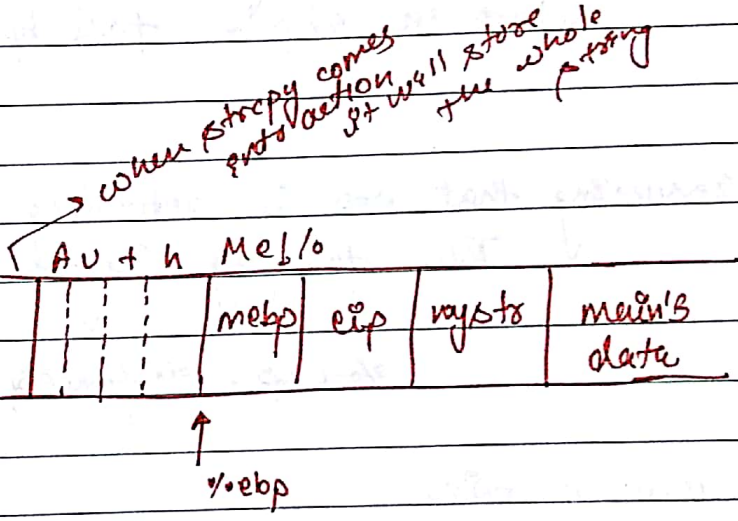
- Buffer: It is any contigious memory that is associated with the variable on the frame.
- Overflow: put more into the buffer than it can hold.

Teacher's Signature.....

Notes

```
void foo (char *arg) {
    char buffer[4];
    strcpy (buffer, arg1);
}

int main () {
    char *mystr = "AuthMe!"
    foo (mystr);
}
```

Corresponding stack



NOTE:- C, C++ are not type safe language

→ now since the strcpy will copy the string in the mebp part, but the mebp stored will be overwritten and the ASCII values of the "Me!/0" will be stored thus forming a new address, so when we are returning again to the old frame, we cannot reach there as the address will be changed.

Hence producing "Segmentation fault"!

** Hence Buffer overflow is problem

Teacher's Signature.....

Notes

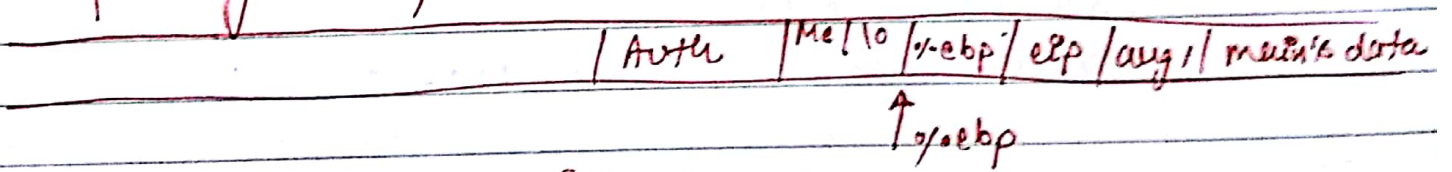
Page No.:
Date: / /

The security threat due to this/above.

A new function:

```
void foo (char *arg1){
    int authenticated = 0;
    char buffer[4];
    strcpy(buffer, arg1);
    if (authenticated){
        access secrete
            information
    }
}

int main(){
    char *mystr = "AuthMe!"
    foo(mystr);
}
```

Corresponding Stack.



→ now since copying will take place
Hence authenticated = 1
Hence the user can access the information
↳ when the case of overflow.

→ This attack is known as "buffer overflow attack."
confidentiality compromised.

Teacher's Signature.....

Page No.:
Date: / /

→ When system crashed → "Availability compromised"
→ The attacker can get remote access to the user, this is known as "code injection attack"

• CODE INJECTION ATTACK.
→ One way to achieve is by buffer overflow.
1. loading of code
→ if want to load, the code should be in machine language / compiled form.
→ The code should not contain zeros. (compiled) → NULL characters
How to generate such codes?
→ open a shell in the user's system.

* shell code
→ a code to open a shell code.
**

```
#include <stdio.h>
int main(){
    char *name[2];
    name[0] = "/bin/sh";
    name[1] = NULL;
    execve(name[0], name, NULL);
}
```

The corresponding machine level code & assembly code

```
XOR %.eax %eax   |  0x31 0xc0
push1 %eax       |  0x50
                 |   :
```

→ Assembly code → machine language

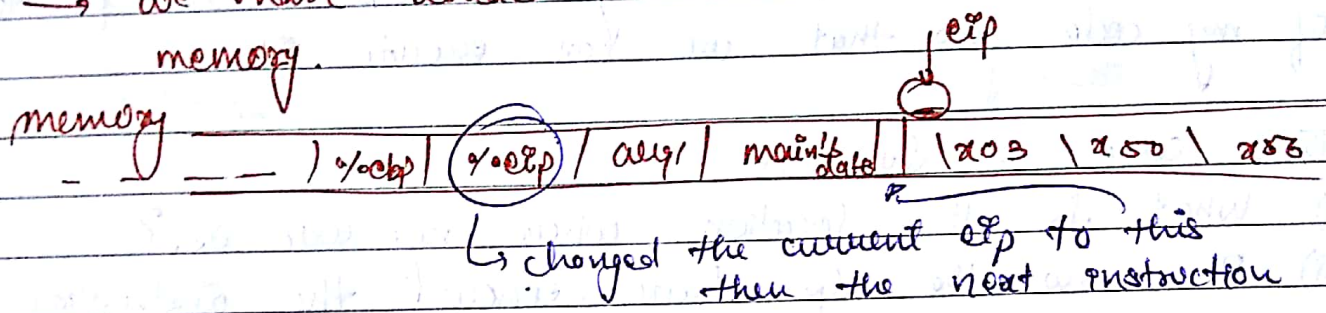
Teacher's Signature.....

Notes

Page No.:
Date: / /

→ To copy we will pass the machine level code as string in the stack, then machine level code is executed, but we have to be careful about the pointers or "Segmentation fault" will occur.

2. Executing the loaded code.
→ we have loaded the code somewhere in the memory.



↳ changed the current eip to this then the next instruction executed will be our code.

* void func (char* arg) {
    char buffer[4];
    strcpy (buff, arg);
  }

• CODE INJECTION THROUGH BUFFER OVERFLOW.

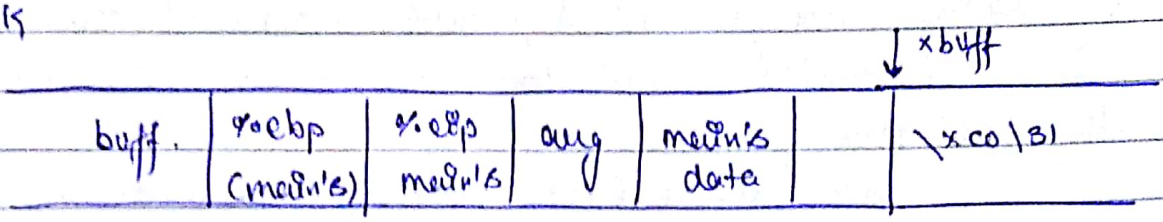
1. Load the Code
→ Pre-compiled (in accordance with the architecture of function)
→ No-strong terminations.
→ self-contained.

Teacher's Signature.....

Page No.:
Date: / /

- How we can decad the code ??

STACK



↳ It shows what is the next instruction to be executed

If somehow I am able to change the eip to the pointer of my code so that we can execute it.

There come 2 questions:

i) What is the location where my code is.?

ii) To know the eip from where the instruction to be executed ?

Possible ways:

→ If we have source code we can easily find the eip and change it
(but it is not possible ∵ source code mostly is not available).

NOTE: we cannot traverse all the available addresses.
eg. if 32 bit → $2^{32}-1$ address.

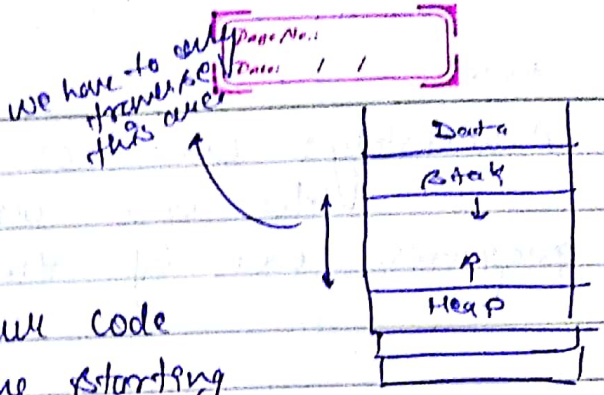
Therefore, to make it space efficient by searching only the stack part until Heap,
but we can not have the starting address of Heap, but still it's better than to make assumptions on $2^{32}-1$ add.

Teacher's Signature.....

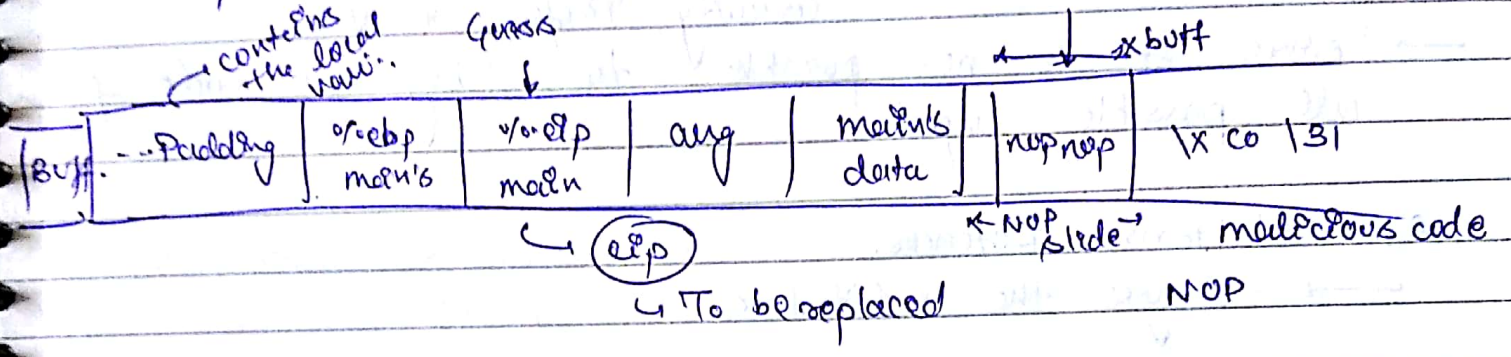
Notes

→ NOP → To access / execute the next instruction.

→ we insert many NOP's in our code so that we can reach the starting of our code.



No. of times NOP is inserted constitutes NOP slide



** → we assumed the esp's location and then we replace one of the address in the NOP slide with esp so that malicious code can be executed.

This is known as ~~stack smashing~~.

→ This whole particular procedure where we are working only on stack is called stack Smashing

• Control Hijacking attack

→ If the attack is successful then the control is shifted to the attacker this is known as CHA.

→ Integrity is violated, since we have inserted our code in the stack.

Teacher's Signature.....

Notes

Page No.:
Date: / /

- Availability :- Since as the attack is successful, the user will not be able access the programs, even if the attack is not successful.

- Defence Mechanism.

1. Fix the bug.
   → Audit softwares.
   ↳ Security, Prefix, Postfix.
   → Since it is not possible to debug the code for all possible inputs.

2. Platform Defences.
   → changes the architecture.

3. making memory non-executable (W^X) → executable
   ↳ writable
   → for code injection attack the defence is: making memory stack either writable or executable, but not both.

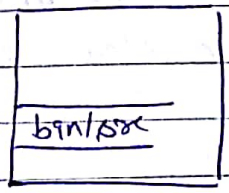
These days.
stack writable → non-executable (NX = 1)
→ genuine user will not write code on stack thus his/her code will be executable ∵ it is placed in "code segment"
→ whereas attacker will place the code in stack.

Teacher's Signature.....

Notes → libso attack. → memory is not executable.

Page No.:
Date: / /

* libc.so is library in "C" language.
→ return-to-libc
it will replace the eip to the current bin/sx address and thus doing the attack.



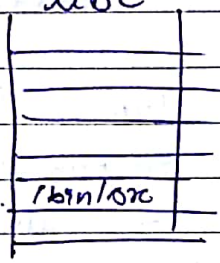
*

2. ASLR (address space layout randomization).
→ somehow we will randomize the addresses of instruction / libraries where they are placed.

* limitations on change in Hardware.
→ Heap need to be both writable and executable
→ Does not provide protection against return oriented programming exploits.

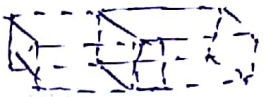
↳ eg.

1. * Return-to-libc.
↳ standard library of c.
replace the eip to the current instruction which the attacker wants to execute



2. ASLR
(Address space layout randomization).
→ maps shared libraries to random locations in process memory.

Teacher's Signature.....



Page No.:
Date: / /

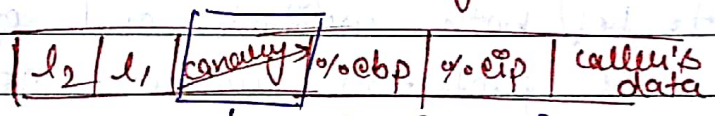
Notes

→ stack / Heap / .... → location randomized.
→ system calls
→ Instruction-set Randomization. (~~FR~~ ISR)
↳ system will encrypt

* steps → code → assembly language → encrypt → decrypted @ Runtime.
but this is very costly.

** DEFENCE MECHANISM

1. Stack guard
→ embed canaries after each stack frame.



↳ This will save the ebp and eip from attacker / it ~~acts~~ acts as a guard.

→ if the attackers change the data the value at canaries will change and thus we will exit from the program.

Types of canary

1. Random Canary.
→ if (canary old = canary new)
continue;
else
exit;

NOTE:- checks stack integrity. [for eip and ebp].

Teacher's Signature.....

Notes

STEPS FOLLOWED BY COMPILER → called canary.

1. Random string is chosen @ program startup.
2. Insert canary string into every stack-frame.
3. Verify the canary before returning from the function.
4. Exit the program if canary changed.

* Availability is violated
→ since as the value of canary is change the program exits.

→ The canary is same for a particular run whole the time and is inserted in each frame. Now the attacker can read the canary at the first and then exploit.

2. Terminator Canary.
→ We will generate a canary in such a way that it contains terminator character
eg. null, '\n', '\0' etc.

So now there is no need to check the canary for equality.
And thus DoS is not reached

→ We have to recompile all the present codes so as to insert canary.
↳ go do this.

Teacher's Signature.....

Notes

Page No.
Date: / /

* Analogy) lab → safe → vulnerable condition
  len(dest) >= len(source)

3. STACK SHIELD

→ We store "return address" and "stack frame pointer" in a safe place.

→ When attack has happened, then we check the integrity of these two values.

If (RET && SFP same) → as old
    continue;
Else
    update with the original RET and SFP. and continue;

4. Control flow integrity.
   If are making control flow graph of the program