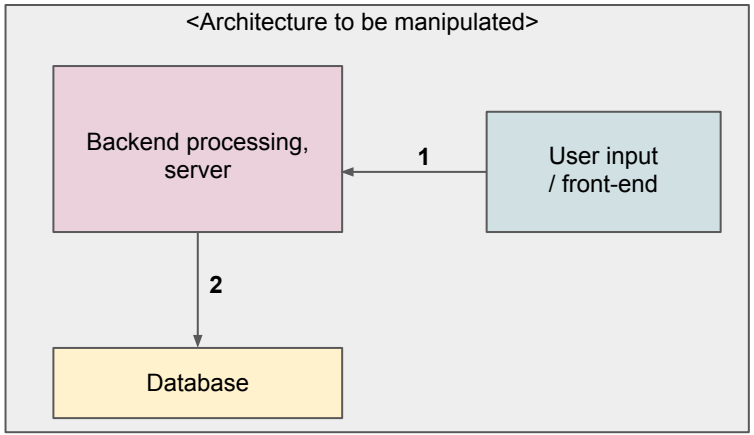
Computer Security Tutorial
(MT20 Wk 4)

Siddhartha Datta
siddhartha.datta@cs.ox.ac.uk

table of contents

sql injection attacks



Crafting SQL injection queries:

Close the input variable + *Bypass condition / desired command* + *Comment out rest of line*

Other possibilities:

- At username box:

```
SELECT * FROM users WHERE email = 'user585@new-bank.com' OR email
<> '' -- ' AND password = '' limit 1
```

- Drop tables:

```
SELECT * FROM users WHERE email = 'user585@new-bank.com' AND
password = ''; DROP TABLE users; --' limit 1"
```

query	Objective: Bypass login (Exercise 1, Question 8)
1	`' OR 1=1 --`
2	send_query("`SELECT * FROM users WHERE email = 'user585@new-bank.com' AND password = '' OR 1=1 --' limit 1`")

sql injection defenses

Parameterized queries
Escaping strings
White list input validation
Black list input validation?
Read-only database access?

symmetric key encryption | DES

Intuition for Question 4(i)

- When master key K is series of 0, generated round keys K_i are also series of 0 → XOR of these keys and message snippet return no permutation (c=m)
- S-boxes and other permutation tables* throughout DES help scramble the message (c!=m)

Prove whether message is scrambled:

Hex $K_i \in K = (0^{64})^{¼}$, $K_i = (0^{56})^{½}$
$L_1 = R_0$
$R_1 = L_0$ XOR P($\{S_i(E(R_0)$ XOR $K_1)\}^8_{i=1}$) = L_0 XOR P($\{S_i(E(R_0))\}^8_{i=1}$)
…
⇒ Scrambled by P($\{S_i(\cdot)\}^8_{i=1}$)

*Helpful illustration of what permutations occur in DES

symmetric key encryption | DES

Intuition for Question 4(ii)

- Decryption for DES is simply reversing the round keys. As the round keys are all identical, the second encryption is identical to decryption, i.e. return plaintext.

Return result of double encryption:

$$\text{Round}_1(\cdot) = \text{Round}_2(\cdot) = \ldots = \text{Round}_{16}(\cdot)$$
$$\text{Round}_{1\ldots16}(\text{Round}_{1\ldots16}(m)) = \text{Round}_{16\ldots1}(\text{Round}_{1\ldots16}(m)) = m$$

*Helpful illustration of what permutations occur in DES

symmetric key encryption | block ciphers

Question 5:

I. Given there are *(k-n)* extra bits in a key (which can be used to formulate 2^{k-n} more keys per *m:c* pair), more than 1 *m:c* pairs are needed to deduce the key.

II. Assume we ignore candidate pair storage: enumeration list length 2^k, *(k+n)* bits per *{k, $E_k(m)$}* pair → 2^k *(k+n)* bits of storage required. Substitute {k=56, n=64} to return DES storage requirement $120*2^{56}$.

<u>If we consider candidate pairs as well:</u>

- Assuming we continue with the assumption in I. that k>n
- Assume we only store (k, k') pairs when E(k, m) == D(k', c), i.e. bits stored per pair is 2k

Length of enumeration list 2^k with each item (k+n) bits
+ List of candidate pair list (on avg) 2^{k-n} of each item 2k bits
= $2^k[k*(1+2^{1-n})+n]$

symmetric key encryption | block ciphers

Question 6:

Possible padding includes 01, 02 02, 03 03 03, ….., 08 08 08 08 08 08 08 08 (i.e. empty message).
Delete last n bytes to unpad, where n is the last byte of the decrypted message.

2^8 possible values per byte, permutation position fixed.
P[last_byte = 01] = $1/2^8$
P[last_byte = 02 02] = $1/(2^8)^2$
…
P[last_byte = 08...08] = $1/(2^8)^8$

i.e. P{correct padding] = $\sum_{i=1}^{8} \frac{1}{2^{8i}}$

symmetric key encryption | block ciphers

Question 7:

$$d_i = c_i \text{ XOR } D_k(c_{i+1})$$
$$d_i = c_i \text{ XOR } D_k(E_k(m_i \text{ XOR } c_i))$$
$$d_i = c_i \text{ XOR } m_i \text{ XOR } c_i$$

All blocks can be successfully decoded to original message $d_i = m_i$ except for d_3,
which returns $d_3 = c_3$ XOR m_3‘ XOR c_3’
i.e. m_3 / m_3’ are non-recoverable, and the message is corrupted.

Integrity of the message/ciphertext not preserved.

special topics | “Hidden in plain sight”

Code obfuscation

the act of creating source code that is difficult for humans to understand

- ➔ To make it harder for attackers to understand how our code functions
- ➔ To preserve intellectual property
- ➔ May slow down the system

Original:

```
int i;
main(){
    for(;i["]<i;++i){--i;}"];read(i+++"hello, world!\n"));
}
read(i){
    write(1,i,1);
}
```

Obfuscated:

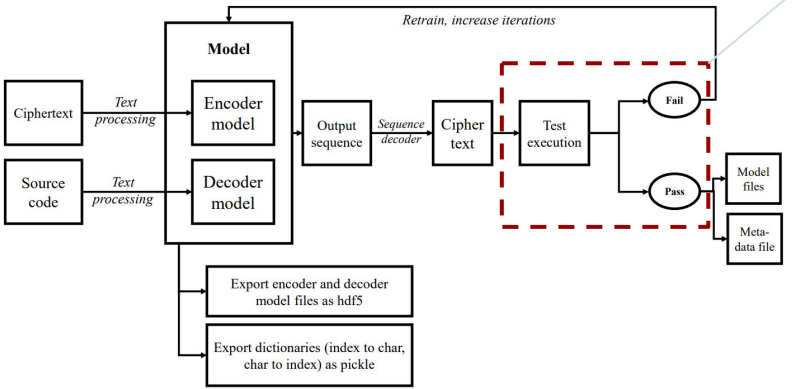
```
int i;main(){for(;i["]<i;++i){--i;}"];read('-'-'-',i+++"hell\
o, world!\n",'/'/'/'));}read(j,i,p){write(j/p+p,i---j,i/i);}
```

special topics | “Hidden in plain sight”

Common techniques:

- **Code flow obfuscation**
  - change the control flow in the code
  - the changed control flow must lead to the same results as the initial one, but the code is more difficult to be analyzed
  - Control ordering – change the order in which the program statements are executed
  - Control computation – change the control flow in the program a
    - Inserting dead code in the code execution flow to hide the true control flow; the code complexity is increased
    - Inserting alternative structures having a controlled evaluation of the condition to break a statement block into two statement sub-blocks at least
- **Name obfuscation**
  - replace identifiers with meaningless strings as new identifiers
- **Data obfuscation**
  - changes the ways in which data are stored in the memory

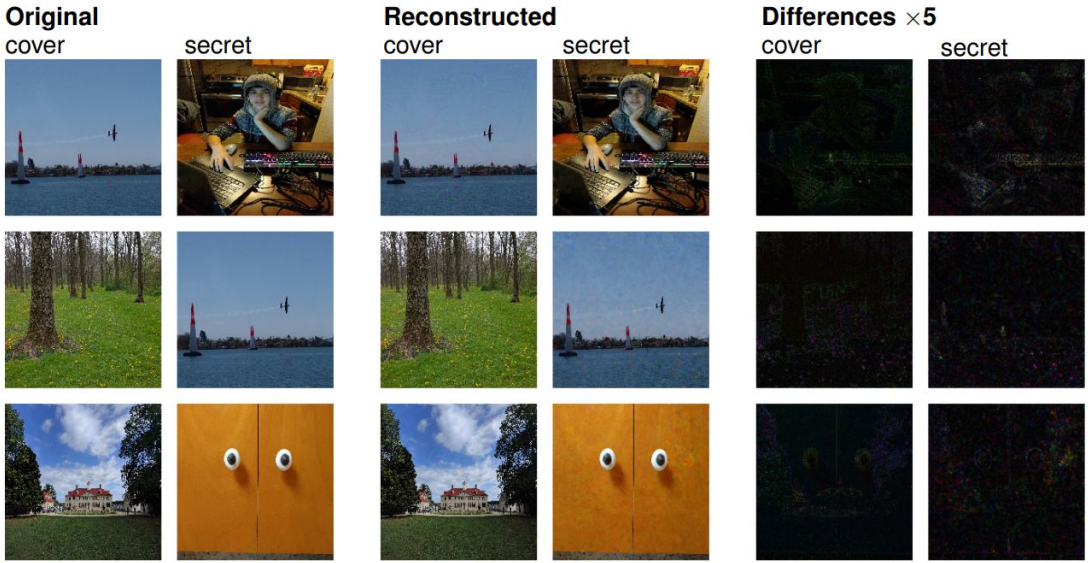
Verification of obfuscated code against original/ unobfuscated code



special topics | “Hidden in plain sight”

Steganography

concealing a file, message, image, or video within another file, message, image, or video.

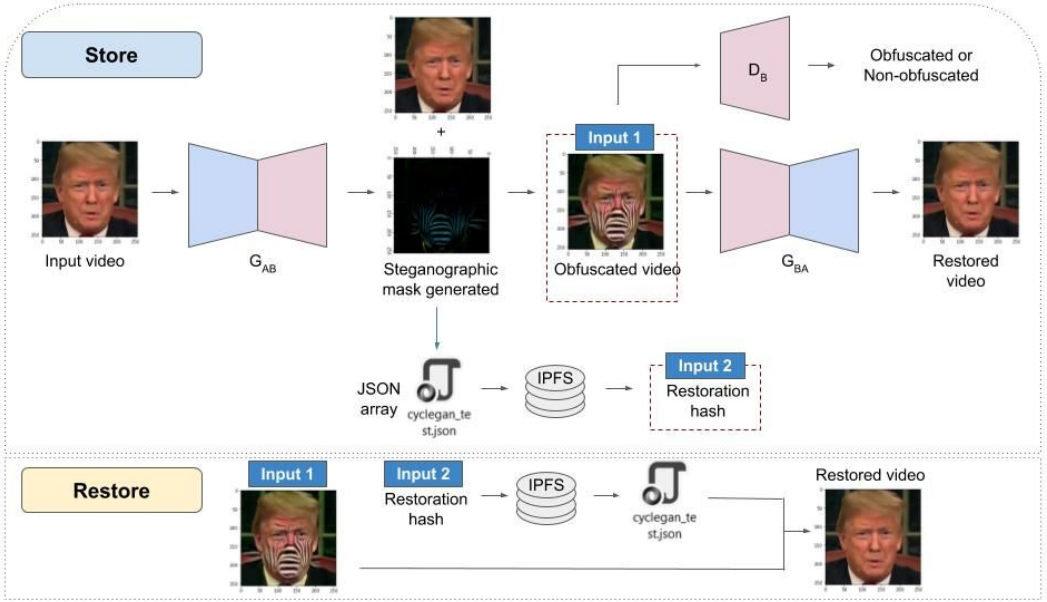


Using adversarial perturbations / style transfer techniques to underlay a message into public messages

special topics | “Hidden in plain sight”

→ Allows us to store information publicly while preserving privacy

→ Can be flipped into a system design question



Example of an architecture where inputs are a public message can contain private details, using CycleGANs and blockchain