

# Security Engineering

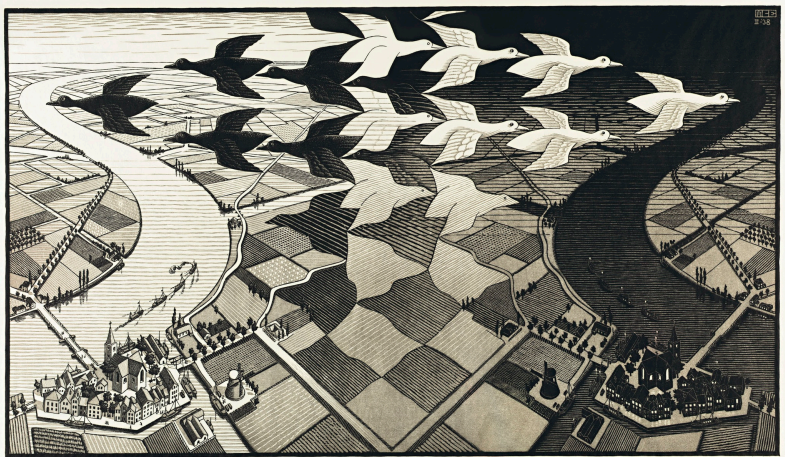
Email: christian.urban at kcl.ac.uk

Office: S1.27 (1st floor Strand Building)

Slides: KEATS (also homework is there)

Imagine you have an completely innocent email message, like birthday wishes to your grandmother? Why should you still encrypt this message and your grandmother take the effort to decrypt it?

- (Hint: The answer has nothing to do with preserving the privacy of your grandmother and nothing to do with keeping her birthday wishes super-secret. Also nothing to do with you and grandmother testing the latest encryption technology, nor just for the sake of it.)



M.C.Escher, Amazing World (from Gödel, Escher, Bach by D.Hofstadter)

# Interlock Protocol

A Protocol between a car  $C$  and a key transponder  $T$ :

- 1  $C$  generates a random number  $N$
- 2  $C$  calculates  $(F, G) = \{N\}_K$
- 3  $C \rightarrow T: N, F$
- 4  $T$  calculates  $(F', G') = \{N\}_K$
- 5  $T$  checks that  $F = F'$
- 6  $T \rightarrow C: N, G'$
- 7  $C$  checks that  $G = G'$

# Zero-Knowledge Proofs

- Essentially every NP-problem can be used for ZKPs
- modular logarithms: Alice chooses public  $A$ ,  $B$ ,  $p$ ; and private  $x$

$$A^x \equiv B \pmod{p}$$

# Modular Arithmetic

It is easy to calculate

$$? \equiv 46 \pmod{12}$$

# Modular Arithmetic

It is easy to calculate

$$10 \equiv 46 \pmod{12}$$

A: 10

# Modular Logarithm

Ordinary, *non*-modular logarithms:

$$10^? = 17$$



# Modular Logarithm

Ordinary, *non*-modular logarithms:

$$10^? = 17$$

$$\Rightarrow \log_{10} 17 = 1.2304489 \dots$$

# Modular Logarithm

Ordinary, *non*-modular logarithms:

$$10^? = 17$$

$$\Rightarrow \log_{10} 17 = 1.2304489 \dots$$

$$\Rightarrow 10^{1.2304489} = 16.999999$$

# Modular Logarithm

Ordinary, *non*-modular logarithms:

$$10^? = 17$$

$$\Rightarrow \log_{10} 17 = 1.2304489 \dots$$

$$\Rightarrow 10^{1.2304489} = 16.999999$$

Conclusion:  $1.2304489$  is very close to the *true* solution, slightly low

# Modular Logarithm

In contrast, modular logarithms behave much differently:

$$2^? \equiv 88319671 \pmod{97330327}$$

# Modular Logarithm

In contrast, modular logarithms behave much differently:

$$2^? \equiv 88319671 \pmod{97330327}$$

Lets say I 'found' 28305819 and I try

$$2^{28305819} \equiv 88032151 \pmod{97330327}$$

# Modular Logarithm

In contrast, modular logarithms behave much differently:

$$2^? \equiv 88319671 \pmod{97330327}$$

Lets say I 'found' 28305819 and I try

$$2^{28305819} \equiv 88032151 \pmod{97330327}$$

Slightly lower. I might be tempted to try  
28305820...

# Modular Logarithm

In contrast, modular logarithms behave much differently:

$$2^? \equiv 88319671 \pmod{97330327}$$

Lets say I 'found' 28305819 and I try

$$2^{28305819} \equiv 88032151 \pmod{97330327}$$

Slightly lower. I might be tempted to try 28305820...but the real answer is 12314.

# Commitment Stage

- 1 Alice generates  $z$  random numbers  $r_1, \dots, r_z$ , all less than  $p - 1$ .

- 2 Alice sends Bob for all  $1..z$

$$b_i = A^{r_i} \text{ mod } p$$

- 3 Bob generates random bits  $b_1, \dots, b_z$  by flipping a coin

- 4 For each bit  $b_i$ , Alice sends Bob an  $s_i$  where

$$b_i = 0: s_i = r_i$$

$$b_i = 1: s_i = (r_i - r_j) \text{ mod } (p - 1)$$

where  $r_j$  is the lowest  $j$  with  $b_j = 1$



# Commitment Stage

- 1 Alice generates  $z$  random numbers  $r_1, \dots, r_z$ , all less than  $p - 1$ .

- 2 Alice sends Bob for all  $i$

$$b_i = A_i$$

Alice $r_i$ :	4	9	1	3
Bob $b_j$ :	0	1	0	1
		↑		
		$j$		

- 3 Bob generates random bits  $b_1, \dots, b_z$  by flipping a coin
- 4 For each bit  $b_i$ , Alice sends Bob an  $s_i$  where

$$b_i = 0: s_i = r_i$$

$$b_i = 1: s_i = (r_i - r_j) \bmod (p - 1)$$

where  $r_j$  is the lowest  $j$  with  $b_j = 1$

# Confirmation Stage

- For each  $b_i$  Bob checks whether  $s_i$  conforms to the protocol

$$b_i = 0: A^{s_i} \equiv b_i \pmod{p}$$

$$b_i = 1: A^{s_i} \equiv b_i * b_j^{-1} \pmod{p}$$

Bob was sent

$$b_1, \dots, b_z,$$

$$r_1 - r_j, r_2 - r_j, \dots, r_z - r_j \pmod{p - 1}$$

where the corresponding bits were 1; Bob does not know  $r_j$ , he does not know any  $r_i$  where the bit was 1

# Confirmation Step

- For each  $b_i$  Bob checks the protocol

$$\begin{aligned}A^{s_i} &= A^{r_i - r_j} \\ &= A^{r_i} * A^{-r_j} \\ &= b_{r_i} * b_{r_j}^{-1} \text{ mod } p\end{aligned}$$

$$b_i = 0: A^{s_i} \equiv b_i \text{ mod } p$$

$$b_i = 1: A^{s_i} \equiv b_i * b_j^{-1} \text{ mod } p$$

Bob was sent

$$b_1, \dots, b_z,$$

$$r_1 - r_j, r_2 - r_j, \dots, r_z - r_j \text{ mod } p - 1$$

where the corresponding bits were 1; Bob does not know  $r_j$ , he does not know any  $r_i$  where the bit was 1

# Proving Stage

- 1 Alice proves she knows  $x$ , the discrete log of  $B$  she sends

$$s_{z+1} = (x - r_j)$$

- 2 Bob confirms

$$A^{s_{z+1}} \equiv B * b_j^{-1} \text{ mod } p$$

# Proving Stage

- 1 Alice proves she knows  $x$ , the discrete log of  $B$  she sends

$$s_{z+1} = (x - r_j)$$

- 2 Bob confirms

$$A^{s_{z+1}} \equiv B * b_j^{-1} \pmod{p}$$

In order to cheat, Alice has to guess all bits in advance. She has only  $\frac{1}{2^z}$  chance of doing so.

# How can Alice cheat?

- Alice needs to coordinate what she sends as  $b_i$  (in step 2),  $s_i$  (in step 4) and  $s_{z+1}$  (in step 6).

# How can Alice cheat?

- Alice needs to coordinate what she sends as  $b_i$  (in step 2),  $s_i$  (in step 4) and  $s_{z+1}$  (in step 6).
- for  $s_{z+1}$  she solves the easy

$$A^{s_{z+1}} \equiv B * y \text{ mod } p$$

for  $y$ .

# How can Alice cheat?

- Alice needs to coordinate what she sends as  $b_i$  (in step 2),  $s_i$  (in step 4) and  $s_{z+1}$  (in step 6).
- for  $s_{z+1}$  she solves the easy

$$A^{s_{z+1}} \equiv B * y \text{ mod } p$$

for  $y$ .

- if she can guess  $j$  (first 1) then she sends  $y$  as  $b_j$  and 0 as  $s_j$ .



# How can Alice cheat?

- Alice needs to coordinate what she sends as  $b_i$  (in step 2),  $s_i$  (in step 4) and  $s_{z+1}$  (in step 6).
- for  $s_{z+1}$  she solves the easy

$$A^{s_{z+1}} \equiv B * y \text{ mod } p$$

for  $y$ .

- if she can guess  $j$  (first 1) then she sends  $y$  as  $b_j$  and 0 as  $s_j$ .
- however she does not know  $r_j$  because she would need to solve

$$A^{r_j} \equiv y \text{ mod } p$$

# How can Alice cheat?

- Alice still needs to decide on the other  $b_i$  and  $s_i$ .  
They have to satisfy the test:

$$A^{s_i} \stackrel{?}{\equiv} b_i * b_j^{-1} \text{ mod } p$$

# How can Alice cheat?

- Alice still needs to decide on the other  $b_i$  and  $s_i$ . They have to satisfy the test:

$$A^{s_i} \stackrel{?}{\equiv} b_i * b_j^{-1} \text{ mod } p$$

- Lets say she choses the  $s_i$  at random, then she needs to solve

$$A^{s_i} \equiv z * b_j^{-1} \text{ mod } p$$

for  $z$ .

# How can Alice cheat?

- Alice still needs to decide on the other  $b_i$  and  $s_i$ . They have to satisfy the test:

$$A^{s_i} \stackrel{?}{\equiv} b_i * b_j^{-1} \text{ mod } p$$

- Lets say she choses the  $s_i$  at random, then she needs to solve

$$A^{s_i} \equiv z * b_j^{-1} \text{ mod } p$$

for  $z$ . It still does not allow us to find out the  $r_i$ . Let us call an  $b_i$  calculated in this way as **bogus**.

# How can Alice cheat?

- Alice has to produce bogus  $b_i$  for all bits that are going to be **1** in advance.

# How can Alice cheat?

- Alice has to produce bogus  $b_i$  for all bits that are going to be 1 in advance.
- Lets say  $b_i = 1$  where Alice guessed 0: She already has sent  $b_i$  and  $b_j$  and now must find a correct  $s_i$  (which she chose at random at first)

$$A^{s_i} \equiv b_i * b_j^{-1} \text{ mod } p$$

If she knew  $r_i$  and  $r_j$ , then easy:  $s_i = r_i - r_j$ . But she does not. So she will be found out.

# How can Alice cheat?

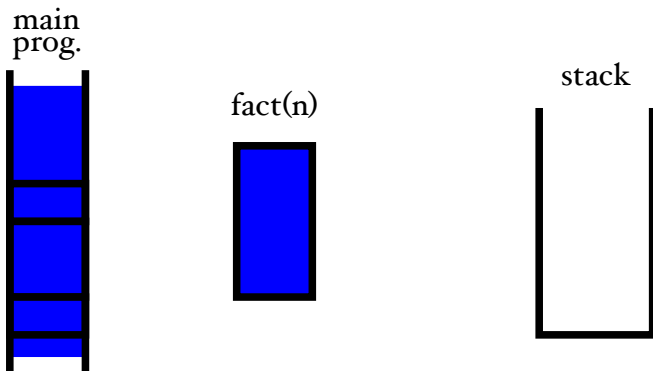
- Alice has to produce bogus  $b_i$  for all bits that are going to be 1 in advance.
- Lets say  $b_i = 0$  where Alice guessed 1: She has to send an  $s_i$  so that

$$A^{s_i} \equiv b_i \pmod{p}$$

She does not know  $r_i$ . So this is too hard and she will be found out.

# Buffer Overflow Attacks

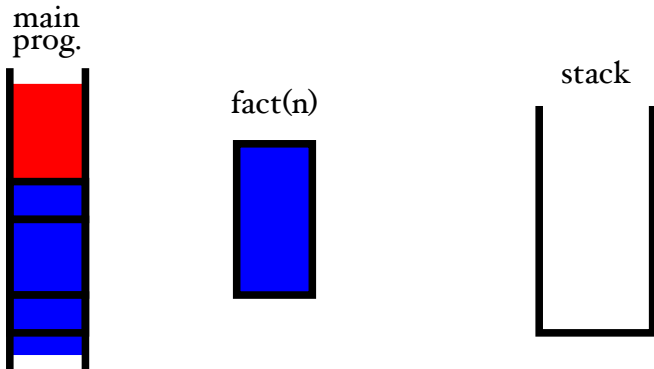
- the problem arises from the way C/C++ organises its function calls





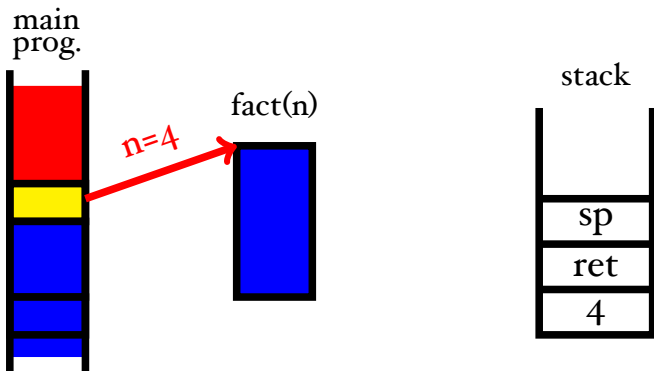
# Buffer Overflow Attacks

- the problem arises from the way C/C++ organises its function calls



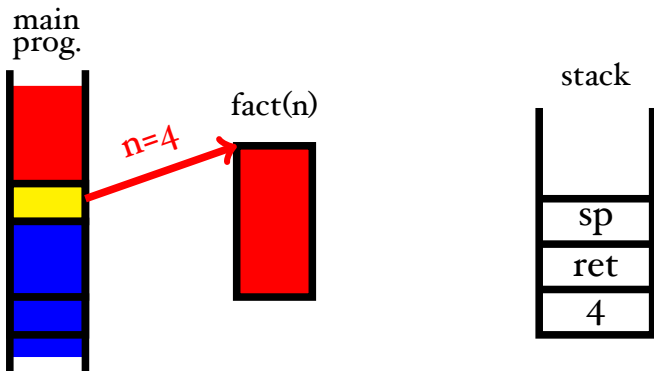
# Buffer Overflow Attacks

- the problem arises from the way C/C++ organises its function calls



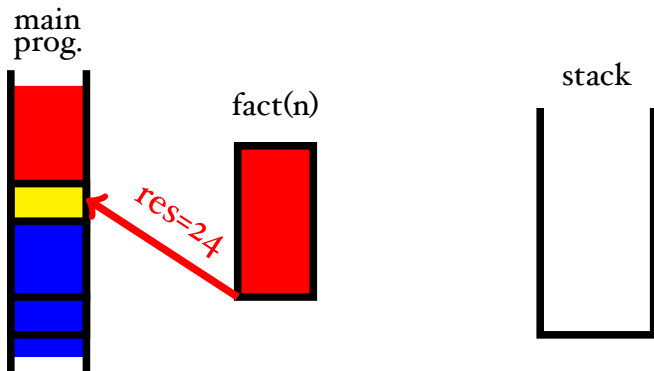
# Buffer Overflow Attacks

- the problem arises from the way C/C++ organises its function calls



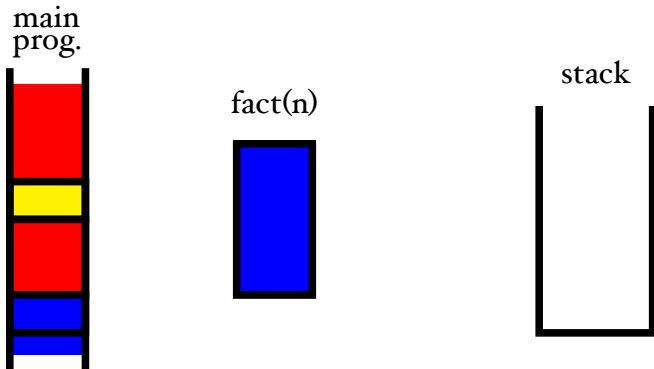
# Buffer Overflow Attacks

- the problem arises from the way C/C++ organises its function calls



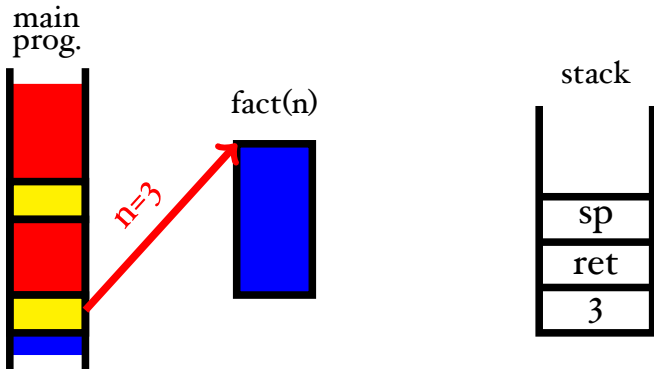
# Buffer Overflow Attacks

- the problem arises from the way C/C++ organises its function calls



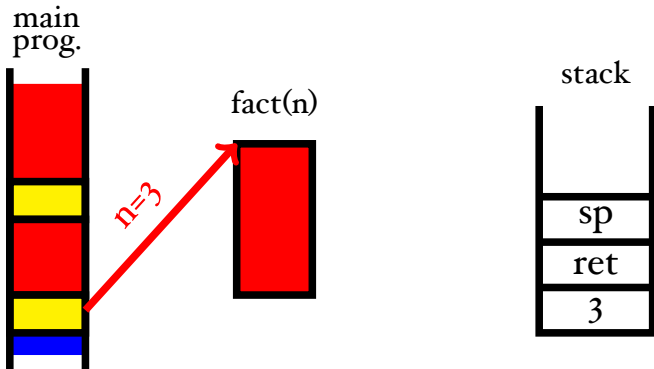
# Buffer Overflow Attacks

- the problem arises from the way C/C++ organises its function calls



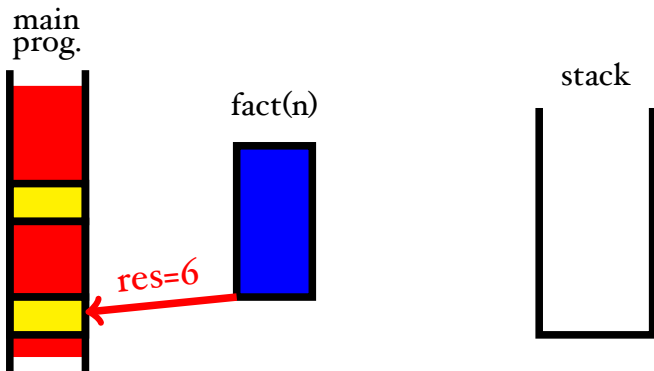
# Buffer Overflow Attacks

- the problem arises from the way C/C++ organises its function calls



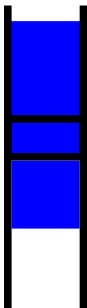
# Buffer Overflow Attacks

- the problem arises from the way C/C++ organises its function calls

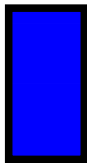




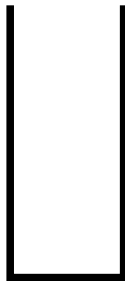
main  
prog.



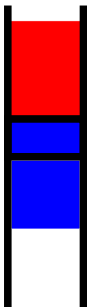
fact(n)



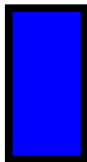
stack



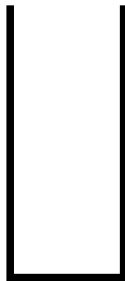
main  
prog.

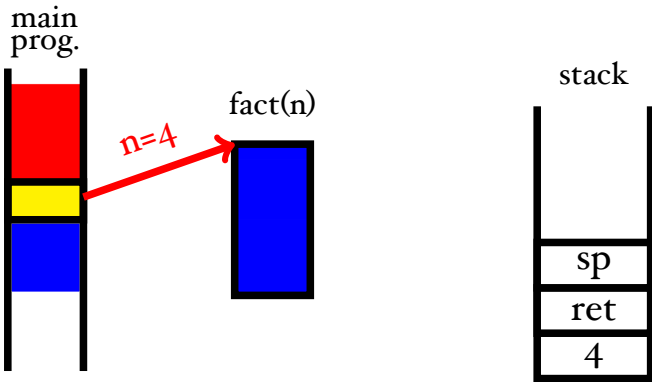


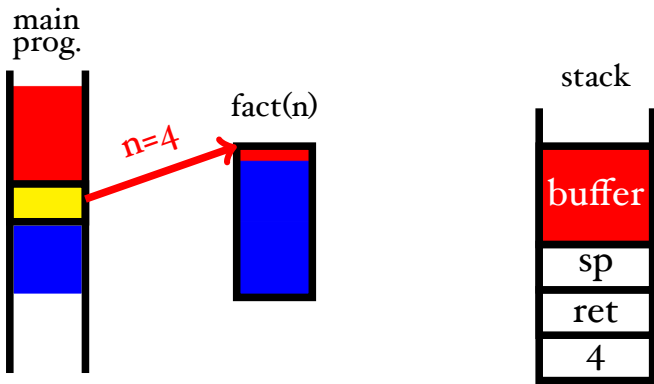
fact(n)

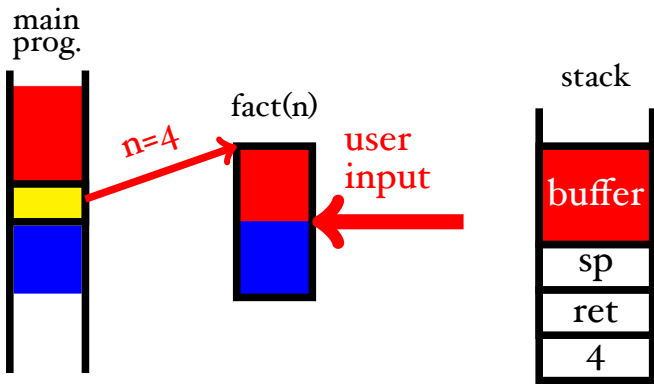


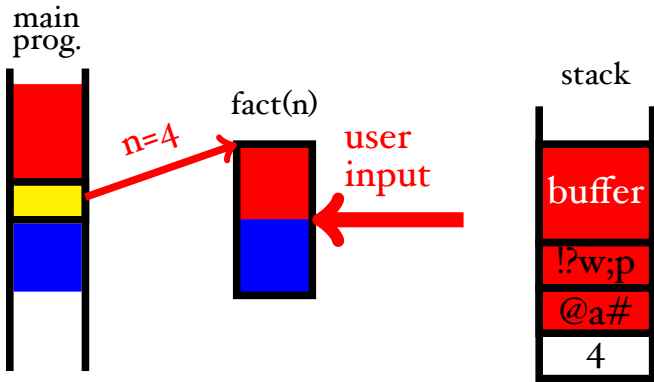
stack

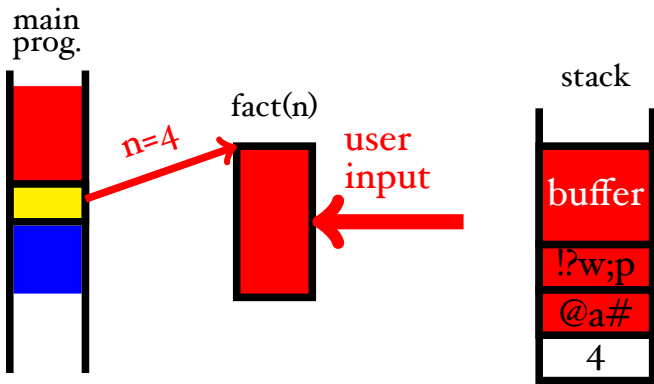


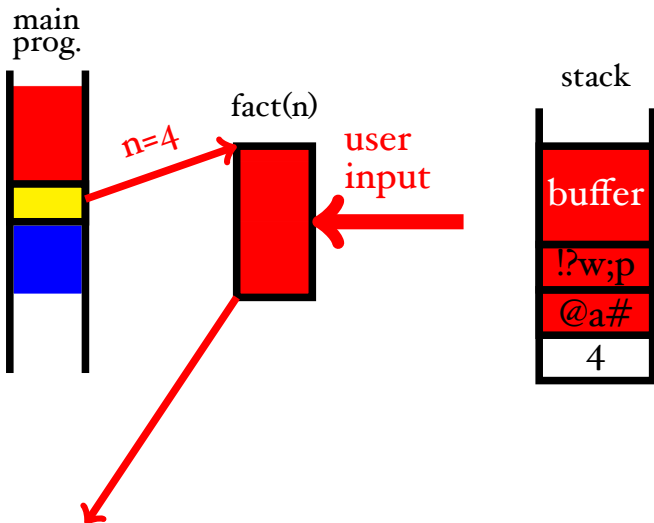














# Coming Back To...

Imagine you have an completely innocent email message, like birthday wishes to your grandmother? Why should you still encrypt this message and your grandmother take the effort to decrypt it?

# Coming Back To...

Imagine you have an completely innocent email message, like birthday wishes to your grandmother? Why should you still encrypt this message and your grandmother take the effort to decrypt it?

- Bruce Schneier  
NSA Surveillance and What To Do About It  
<https://www.youtube.com/watch?v=QXtS6UcdOMs>

Terrorists use encrypted mobile-messaging apps. The spy agencies argue that although they can follow the conversations on Twitter, they “go dark” on the encrypted message apps. To counter this “going-dark problem”, the spy agencies push for the implementation of back-doors in iMessage and Facebook and Skype and everything else UK or US-made, which they can use eavesdrop on conversations without the conversants’ knowledge or consent.

- What is the fallacy in the spy agencies going-dark argument?