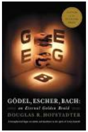
Security Engineering

Email: christian.urban at kcl.ac.uk
Office: S1.27 (1st floor Strand Building)
Slides: KEATS (also homework is there)

Imagine you have a 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 Protocols

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. $C \to T$: N, F

4. T calculates $(F', G') = \{N\}_K$
5. T checks that $F = F'$
6. $T \to C$: 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

$$A^x \equiv B \; mod \; p$$

Modular Arithmetic

It is easy to calculate

$$? \equiv 46 \; mod \; 12$$

Modular Arithmetic

It is easy to calculate

$$10 \equiv 46 \; mod \; 12$$

A: 10

Modular Logarithm

Ordinary, *non*-modular logarithms:

$$10^{?} = 17$$

Modular Logarithm

Ordinary, *non*-modular logarithms:

$$10^{?} = 17$$

$$\Rightarrow \quad log_{10}17 = 1.2304489\ldots$$

Modular Logarithm

Ordinary, *non*-modular logarithms:

$$10^{?} = 17$$

$$\begin{aligned} \Rightarrow \quad & log_{10} 17 = 1.2304489\ldots \\ \Rightarrow \quad & 10^{1.2304489} = 16.999999 \end{aligned}$$

Modular Logarithm

Ordinary, *non*-modular logarithms:

$$10^{?} = 17$$

$$\begin{aligned} \Rightarrow \quad & log_{10} 17 = 1.2304489\ldots \\ \Rightarrow \quad & 10^{1.2304489} = 16.999999 \end{aligned}$$

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

Modular Logarithm

In contrast, modular logarithms behave much differently:

$$2^{?} \equiv 88319671 \;\; mod \;\; 97330327$$

Modular Logarithm

In contrast, modular logarithms behave much differently:

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

Lets say I 'found' 28305819 and I try

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

Modular Logarithm

In contrast, modular logarithms behave much differently:

$$2^{?} \equiv 88319671 \;\; mod \;\; 97330327$$

Lets say I 'found' 28305819 and I try

$$2^{28305819} \equiv 88032151 \;\; mod \;\; 97330327$$

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

Modular Logarithm

In contrast, modular logarithms behave much differently:

$$2^{?} \equiv 88319671 \;\; mod \;\; 97330327$$

Lets say I 'found' 28305819 and I try

$$2^{28305819} \equiv 88032151 \;\; mod \;\; 97330327$$

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

Commitment Stage

1. Alice generates z random numbers $r_1, ..., r_z$, all less than $p - 1$.
2. Alice sends Bob for all $1..z$
$$h_i = A^{r_i} \; mod \; p$$
3. Bob generates random bits $b_1, ..., b_z$ by flipping a coin
4. For each bit b_i, Alice sends Bob an s_i where
$$\begin{array}{ll} b_i = 0: & s_i = r_i \\ b_i = 1: & s_i = (r_i - r_j) \; mod \; (p - 1) \end{array}$$

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

Commitment Stage

1. Alice generates z random numbers $r_1, \ldots, r_z$, all less than $p - 1$.
2. Alice sends Bob for all 1

$$b_i = A$$

Alice r_i:	4	9	1	3
Bob b_i:	0	1	0	1
		$\uparrow$		
		j		

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

$$\begin{array}{ll} b_i = 0: & s_i = r_i \\ b_i = 1: & s_i = (r_i - r_j)\ mod\ (p - 1) \end{array}$$

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

Confirmation Stage

1. For each b_i Bob checks whether s_i conforms to the protocol

$$b_i = 0: \quad A^{s_i} \equiv h_i \; mod\; p$$
$$b_i = 1: \quad A^{s_i} \equiv h_i * h_j^{-1} \; mod\; p$$

Bob was sent

$$h_1, \ldots, h_z,$$
$$r_1 - r_j, r_2 - r_j, ..., r_z - r_j \;\; mod\; p - 1$$

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

Confirmat

$$A^{s_i} = A^{r_i - r_j}$$
$$= A^{r_i} * A^{-r_j}$$
$$= h_{r_i} * h_{r_j}^{-1} \; mod \; p$$

- For each b_i Bob checks v
  the protocol

$$b_i = 0: \quad A^{s_i} \equiv h_i \; mod \; p$$
$$b_i = 1: \quad A^{s_i} \equiv h_i * h_j^{-1} \; mod \; p$$

Bob was sent

$$h_1, \ldots, h_z,$$
$$r_1 - r_j, r_2 - r_j, \ldots, r_z - r_j \quad mod \; p - 1$$

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

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

Proving Stage

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

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

2. Bob confirms

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

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 h_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 h_i (in step 2), s_i (in step 4) and s_{z+1} (in step 6).

- for s_{z+1} she solves the easy

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

for y.

How can Alice cheat?

- Alice needs to coordinate what she sends as h_i (in step 2), s_i (in step 4) and s_{z+1} (in step 6).
- for s_{z+1} she solves the easy
$$A^{s_{z+1}} \equiv B * y \; mod \; p$$
for y.
- if she can guess j (first 1) then she sends y as h_j and 0 as s_j.

How can Alice cheat?

- Alice needs to coordinate what she sends as h_i (in step 2), s_i (in step 4) and s_{z+1} (in step 6).
- for s_{z+1} she solves the easy
$$A^{s_{z+1}} \equiv B * y \; mod \; p$$
for y.
- if she can guess j (first 1) then she sends y as h_j and 0 as s_j.
- however she does not know r_j because she would need to solve
$$A^{r_j} \equiv y \; mod \; p$$

How can Alice cheat?

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

$$A^{s_i} \stackrel{?}{\equiv} h_i * h_j^{-1} \; mod \; p$$

How can Alice cheat?

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

$$A^{s_i} \stackrel{?}{\equiv} h_i * h_j^{-1} \; mod \; p$$

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

$$A^{s_i} \equiv z * h_j^{-1} \; mod \; p$$

for z.

How can Alice cheat?

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

$$A^{s_i} \stackrel{?}{\equiv} h_i * h_j^{-1} \; mod \; p$$

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

$$A^{s_i} \equiv z * h_j^{-1} \; mod \; p$$

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

How can Alice cheat?

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

How can Alice cheat?

- Alice has to produce bogus h_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 h_i and h_j and now must find a correct s_i (which she chose at random at first)

$$A^{s_i} \equiv h_i * h_j^{-1} \; mod \; p$$

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 h_i for all bits that are going to be 1 in advance.

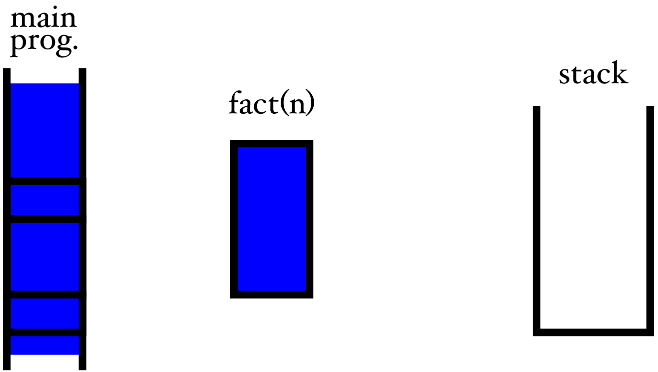
- Lets say $b_i = 0$ where Alice guessed 1: She has to send an s_i so that

$$A^{s_i} \equiv h_i \; mod \; p$$

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

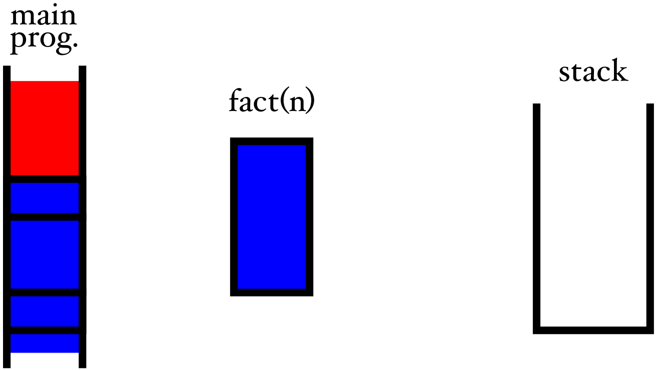
Buffer Overflow Attacks

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



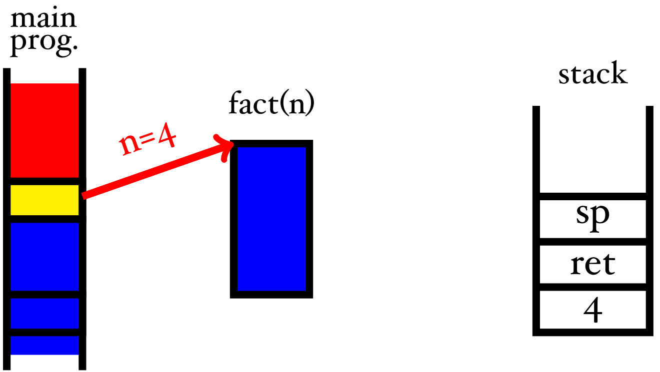
Buffer Overflow Attacks

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



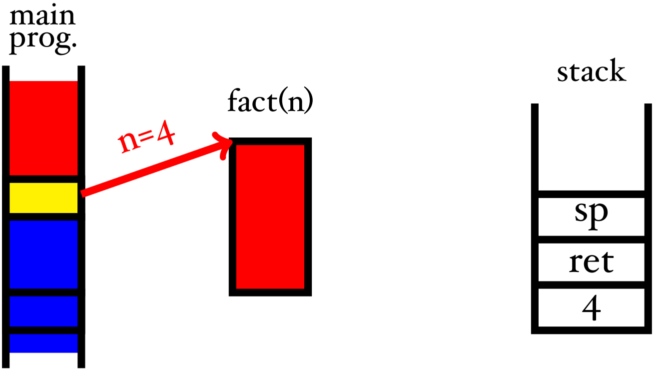
Buffer Overflow Attacks

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



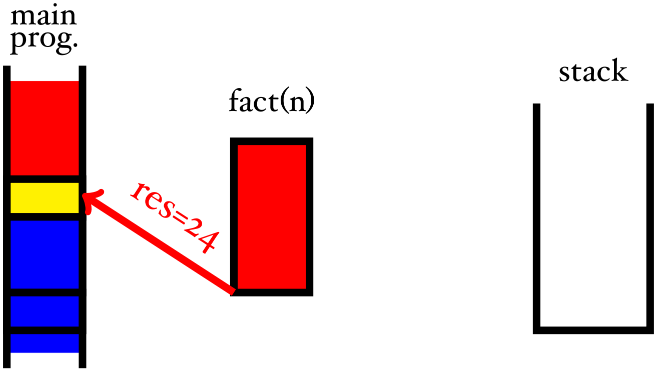
Buffer Overflow Attacks

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



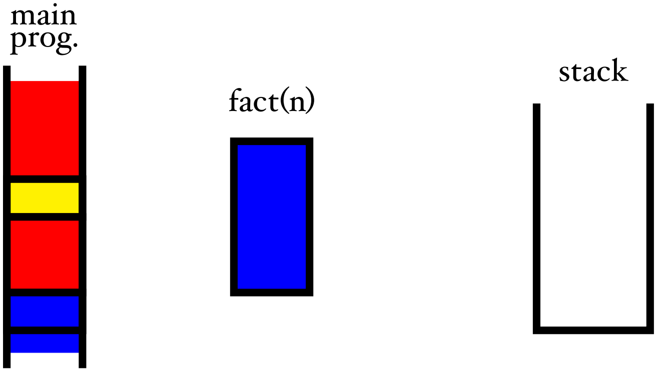
Buffer Overflow Attacks

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



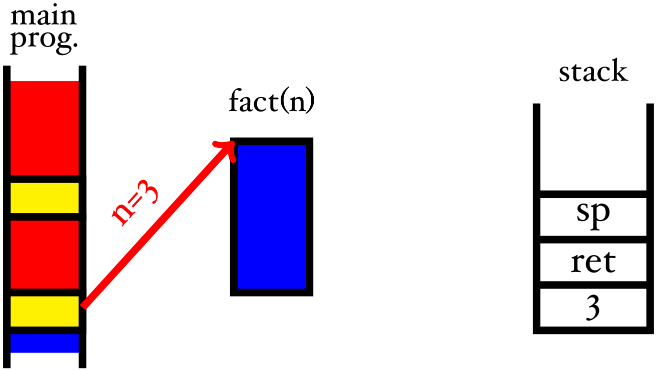
Buffer Overflow Attacks

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



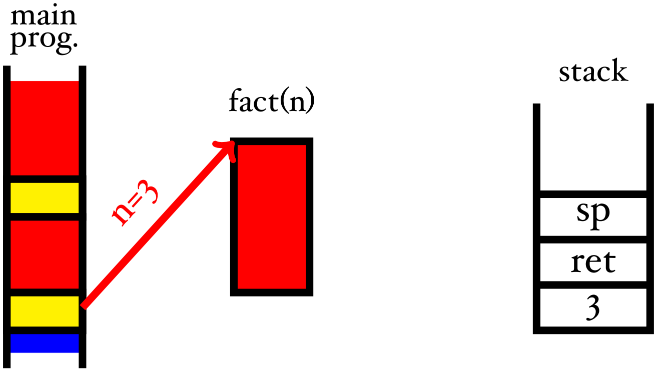
Buffer Overflow Attacks

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



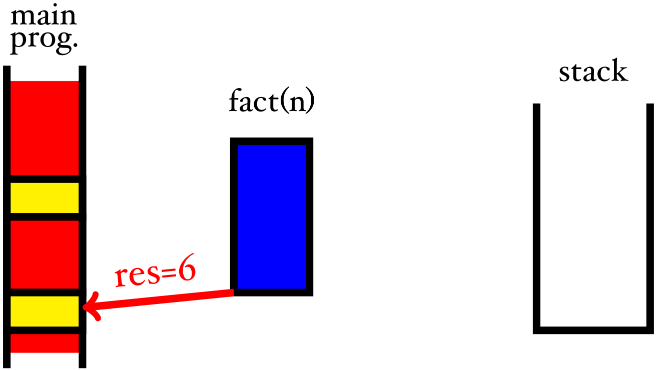
Buffer Overflow Attacks

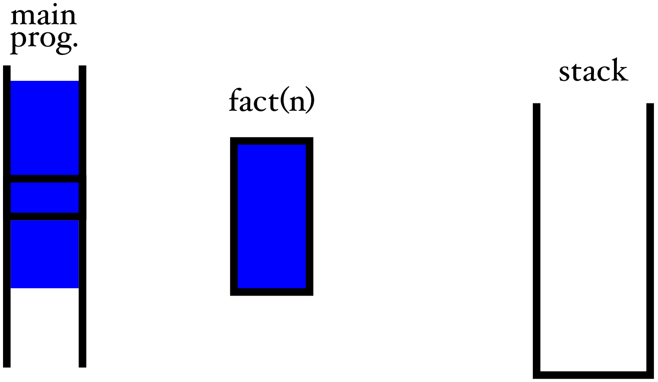
- the problem arises from the way C/C++ organises its function calls

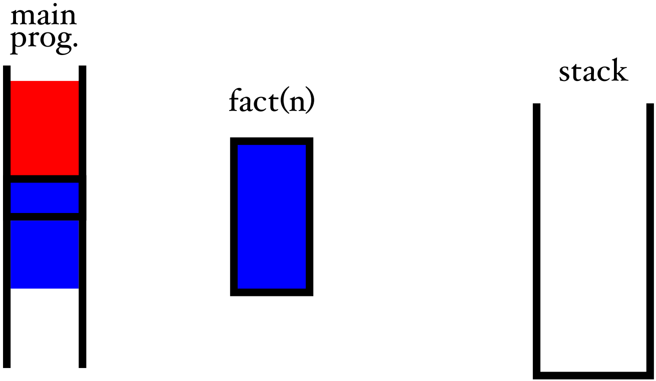


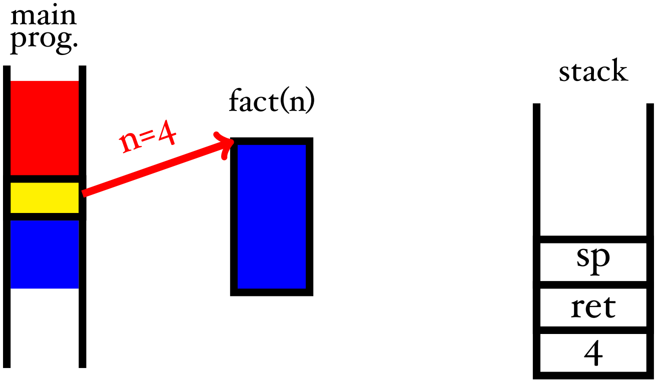
Buffer Overflow Attacks

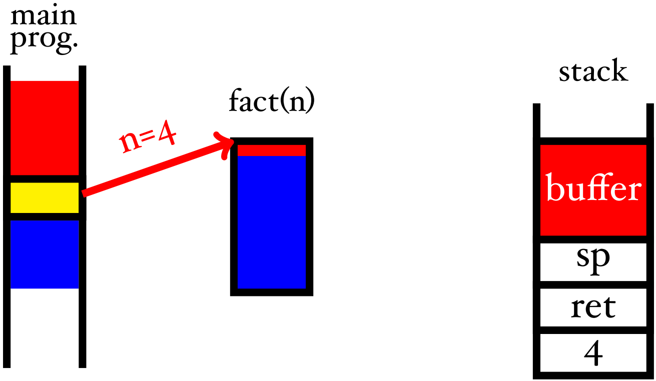
- the problem arises from the way C/C++ organises its function calls

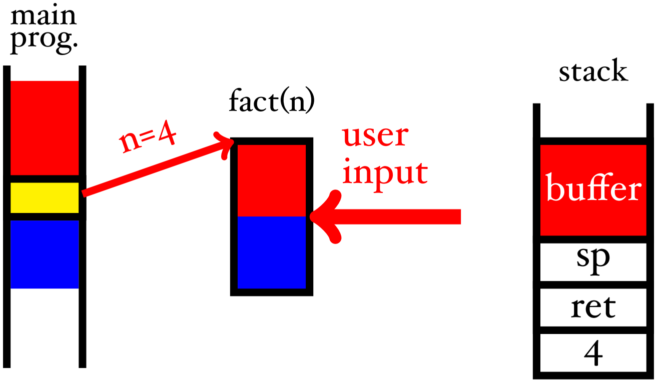


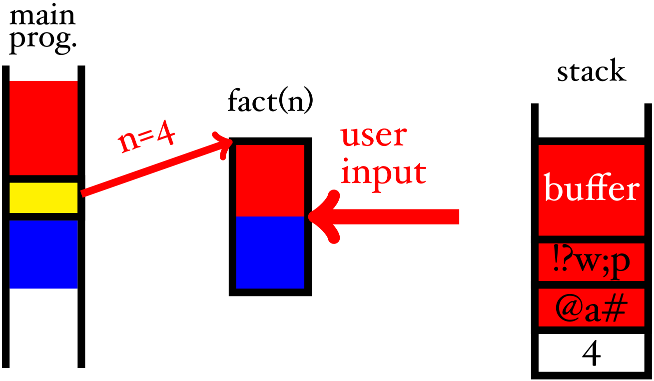

main
prog.
fact(n)
stack

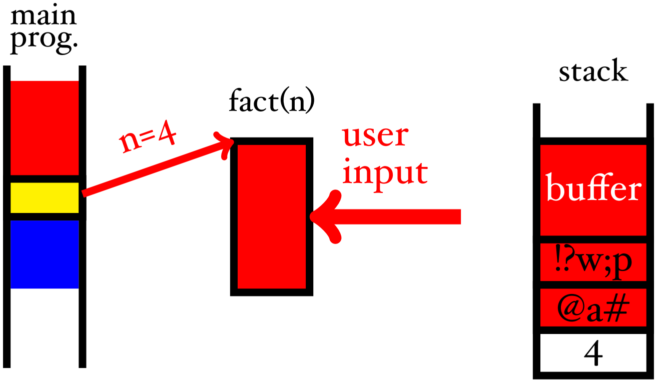

main
prog.
fact(n)
stack

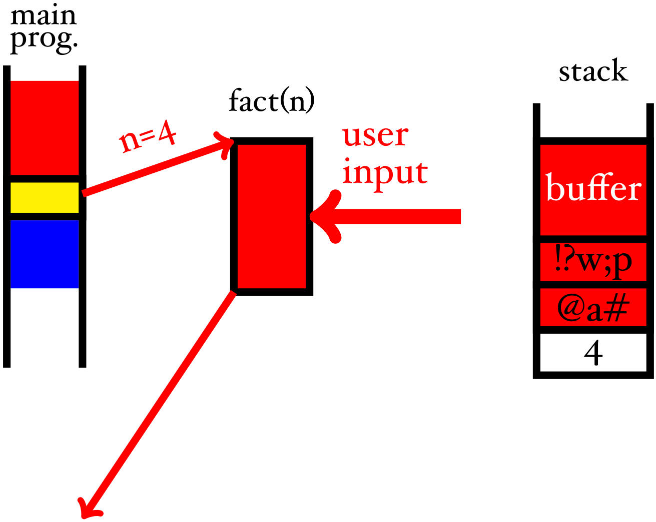

main
prog.
n=4
fact(n)
stack
sp
ret
4


main
prog.
n=4
fact(n)
stack
buffer
sp
ret
4


main prog.
n=4
fact(n)
user input
stack
buffer
sp
ret
4


main
prog.
n=4
fact(n)
user
input
stack
buffer
!?w;p
@a#
4


main prog.
n=4
fact(n)
user input
stack
buffer
!?w;p
@a#
4


main prog.
fact(n)
n=4
user input
stack
buffer
!?w;p
@a#
4

Coming Back To...

Imagine you have a 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 a 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?

- Any wild guesses?

Coming Back To...

> Imagine you have a 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?

- Any wild guesses?
- 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?

Even good passwords consisting of 8 characters, can be broken in around 50 days (obviously this time varies a lot and also gets shorter and shorter over time). Do you think it is good policy to require users to change their password every 3 months (as King's did until recently)?

Under which circumstance should users be required to change their password?