Security and Cryptography 2021

Mirosław Kutyłowski

grading criteria:

- up to 50 points from lecture (exam), up to 50 points from dr Kubiak (project...)
- the lecture at least 30% ot of 50 points must be earned to pass
- sum of points ⇒ the final grade, 3.0: ≥40 points , 5.0 ≥ 80 points
- exam requires problem solving, memorizing facts is unnecessary

skills to be learned: developing end-to-end security systems, flawless in the real sense!

presence: obligatory during the lectures

exam date and form: subject to the situation

place: 11:15-13 Wednesday, 11:15-13 Friday, MS Teams

adjustments possible in order to ease logistics problems

grading system used last year:

- for each “chapter” consisting of a specific topic some verification of skills of the students
- possible verification forms:
  - i an assignment by myself (some concrete task/problem to be solved at home and returned within e.g. 1 week)
  - ii written exam with ePortal

    answer a problem to be typed in or solved on paper, jpeg to be uploaded within, say, 15 minutes
- IPR taken very seriously

Online materials:

- available on my webpage

  https://kutylowski.im.pwr.wroc.pl/lehre/cs21/

- ePortal will be used for 1-1 communiaction with the students (as it keeps a history of each conversation),

- tests – no decision about the platform yet. Subject to: stability and reliability of these tools

Contact:

- during the lecture: mute yourself, when having a question please unmute and switch on your video (the group is small enough to do it)
- email: yes, but assignments etc over ePortal
- the phone at Politechnika – no!
- MS Teams for conf calls
- Signal as second independent channel?

I. FAILURE EXAMPLES TO LEARN FROM

I.1. PKI for Signing Digital Documents

PKI - Public Key Infrastructure

- strong authentication of digital documents with digital signatures seems to be possible
- in fact we get an evidence that the holder of a private key has created a signature
- who holds the key? PKI has to provide a certified answer to this question
- PKI is not a cryptographic solution - it is an organizational framework (using some crypto tools)

PKI, X.509 standard

- a certificate binds a public key with an ID of its alleged owner,
- a couple of other fields, like validity date, key usage, certification policy, ...
- certificate signed by CA (Certification Authority)
- tree of CA's (or a directed acyclic graph), with roots as "roots of trust"
- **status of a certificate may change - revocation**
- checking status methods: CRL, OCSP

reasons for PKI failure:

a nice concept of digital signatures but

1. big infrastructure required:
   - –. substantial cost and effort
   - –. long time planning needed (so possible in China, but not in Europe)
   - –. unclear financial return
2. scope of necessary coordination,
   - –. in order to work must be designed at least for the Common Market
   - –. example of killing the concept: link to certification policy in Polish

3. lack of interoperability (sometimes as business goal)
    - companies make efforts to eliminate competition
    - standarization may be focused on securing market shares
    - a long process ...
4. necessary trust in roots
    - how do you know that the root is honest?
5. registration: single point of fraud, (e.g. with fake breeding documents)
    - once you get a certificate you may forge signatures
6. responsibility of CA
    - fiancial risk – based on risk or responsibility
7. cost - who will pay? For the end user the initial cost is too high.
    - certificates are too expensive for just a few signatures (at least initially)
8. legal strength of signatures
    - if scheme broken or signing devices turn out to be insecure you are anyway responsible for the signatures. After revocation only the new signatures invalid

9. unsolved problem of revocation: possible to check the status in the past but not now

reason: mismatch of requirements and interests with the designed solution

"…but there nothing one can do about it." – this is false

- Smart-ID project, Estonia (clever RSA-like solution, mediated signatures, no CRL, OCSP needed)
- SPKI idea (source centric certification), *suicide notes, certificates of health*

before Smart-ID in Estonia

- personal ID smart cards, implements RSA signature of the owner
- certificate of BSI for Infineon chip and software
- Czech colleagues from Brno found that the RSA keys generated so that the old attacks work
- an implementation bug or a trapdoor
- all smart cards had to be updated

Smart-ID

1. RSA:

   –. "RSA" where n is a product of two RSA numbers

   –. the same algebra – no difference seen unless you factorize n

   –. but secret keys distributed between the card and a mediator server

   –. nobody has full knowledge of the secret keys

2. links between consecutive signatures (to be checked by the mediator server)

3. revocation by blacklisting on the server

I.2. Clickjacking on Android

Overlay mechanism:

- apps are separated in their sandboxes - security design mechanism
- all apps display informations on the screen at the same time - they are overlays
- overlays:
  - require Android permissions
  - clickable or paththrough
  - opaque or transparent
  - combining: parameter α defines weights: for each color of RGB the new pixel value is

$$\text{old} \cdot \alpha + \text{new} \cdot (1 - \alpha)$$

- basic clickjacking:
  - on top an opaque overlay with something innocent (game...)
  - button is in fact a batton but the overlay is paththrough at this place
  - below is an unvisible button of an attacked app

  defense: Google's "obscure flag" - an app checks if at the moment of clicking there is an overlay above it

- context-hiding clickjacking: overlay covers everything but not the button

  defense: Google's "hide overlays" (applicable only in case of settings etc as the users like overlays)

- examples of attacks:
  - → Google play: after installing the app it asks for "open app", but acceptance is for installing and opening something else
  - → Browser: cover the context and make the user click (e-voting?)
  - → gmail: prepare a message, cover it with overlay and ask for accepting "send" button
  - → whatsapp, wechat: send messages, send SMS to chosen destinations (and learn attacked SIM subscriber number)
  - → Google Authenticator: "long click" copies a token to clipboard (and makes it available to other apps)
  - → Facebook, Tweeter: unprotected, possibilities to insert likes, send tweets, ...
  - → Lookout Mobile Security: 3 clicks and anti-virus protection disabled

- remedy?
  - no effective protection mechanism known
  - architecture separates apps so it is impossible to ask what the other apps are showing
  - some overlays must be tolerated due to expectations of the users
  - pressure to run security critical apps (banking...)

Clickshield:

- requires minor changes in the Android framework
- concentrates on the central region (as on margin no critical buttons observed)
- attempts to check whether between the pixel of a target app and the screen final render pixel there is a nonmalicious relationship
- α render: $\mathrm{fr} = \mathrm{round}(\alpha \cdot \mathrm{ov} + (1 - \alpha) \cdot \mathrm{ta})$ where ta=target app pixel value, ov=overlay..., fr=final render pixel value

- given fr and ta we do not know α and ov, but
  - choose two points and assume that the overlay is uniform. Then solve for α:

    $$\mathrm{fr}_1 = \alpha \cdot \mathrm{ov} + (1 - \alpha) \cdot \mathrm{ta}_1$$

    $$\mathrm{fr}_2 = \alpha \cdot \mathrm{ov} + (1 - \alpha) \cdot \mathrm{ta}_2$$

  - use this fixed α to estimate ov over the whole screen. the outcome should be uniform in case of uniform verlays or those informing on margin (notifications)

    Otherwise expect a context switching.

I.3. Easy Fishing on Android

- mobile password managers:
  - associate app (package name) with a domain name
  - for a domain name associated with the app insert the user's credentials when the app accesses this URL
  - credentials are related to the domain name (facebook.com, etc) and not to the app
  - in fact: improves protection against fishing (difference between facebook and faceb00k detected, a human may make a mistake)

- Instant Apps: instead of downloading the whole app fetch only a small app that emulates the full version with accessing an URL
  - gets a full control over the screen – e.g. it may hide browser's security information
- limitations: at most one app with the same package name on an Android device, and at most one in Play Store

- attack:
  - create an app
  - choose the package name so that the mapping points to the attacked domain
  - include developer's URL for Instant App purposes (it is not checked by the Play Store)
  - lure the user to run Instant App
    - → the credentials will be included by the Password Manager
    - → the information will go to the developer's URL
    - → Instant App will show something different on the screen (information from password manager need not to be visible, browser information will be covered as well)
    - → the user should dislike the app and consequently the app will not be downloaded (no traces of forgery)

- mappings:
  - most important associations on a kind of whitelist
  - one-to-one would be more secure but frequently many apps to one domain
  - (insecure) heuristics:
    1. Keeper: finds a corresponding entry on play.google.com and takes "app developer website field" , it autosuggests this name to the user,

       attacks: write malicious app "developer website field"
    2. Dashlane: hardcopied 81 mappings, rest: autosuggestion heuristic based on at least 3 matching characters: xxx.face.yyy will be mapped to facebook.com

       attack: use similar package names
    3. Lastpass: translates directly (aaa.bbb.ccc to bbb.aaa), if not existing then consult from crowdsourced mapping (distributed database created by users – where it is easy to inject something)
    4. 1Password: does not provide mapping but presents suggestions and enables the user – a human to choose (and confuse FACEB00K with FACEBOOK)
    5. Google Smart Lock – burden of mapping to the developer (at the time not automated process) based on Digital Asset Links (on the website a list of permitted apps, authenticated with hashes of the signing key)

I.4. Buying a system

Problem: somebody has to deploy a secure IT system, how to purchase it?

- problematic requirements according to BSI guide:

  i **incomplete** – forgetting some threats is common

  ii **not embedded:** not corresponding really to the environment where the product has to be deployed

  iii **implicit:** customer has in mind but the developer might be unaware of them

  iv **not testable**: ambiguous, source of legal disputes, ...

  v **too detailed:** unnecessary details make it harder to adjust the design

  vi **unspecified meaning:** e.g. "*protect privacy*"

  vii **inconsistent:** e.g. ignoring trade-offs

- *specification-based purchasing process* versus *selection-based purchasing process*
- the user is not capable of determining the properties of the product himself: too complicated, too specialized knowledge required, a single error makes the product useless
- specifications of concrete products might be useless for the customers – hard to understand and compare the products
- informal specifications and descriptions, no access to crucial data

I.5. Blind Trust

Idea - our dream situation:

a security solution should work even if

- $\longrightarrow$ the designer is lazy, stupid, malicious, ...
- $\longrightarrow$ the components are malicious, faulty,...
- $\longrightarrow$ crypto in fact has been broken by bad guys
- $\longrightarrow$ there are trapdoors

Today

we are focused on

- $\longrightarrow$ security assumptions (probably invalid)
- $\longrightarrow$ trust to ...
- $\longrightarrow$ standard situations
- $\longrightarrow$ trusting AI products based on ML

Catacrypt don't wait for quantum computer, catastrophy is already there due to other reason