CODASPY 2015

Password Meters and Generators on the Web: From Large-Scale Empirical Study to Getting It Right

Steven Van Acker, Daniel Hausknecht, Andrei Sabelfeld

CHALMERS UNIVERSITY OF TECHNOLOGY
The Question

Are they securely designed?
How should they be securely designed?
The Question

Are they securely designed?
How should they be securely designed?
What makes a good password?
The Network Attacker

passive network attacker intercepts password

1. password sent over network in plaintext

2. result is rendered
The Second Party Attacker
The Third Party Attacker

- The password is read and leaked to the attacker.
- Malicious JavaScript code is loaded and executed.

Diagram shows a web page with a password input field and a database, with arrows indicating the flow of information and code execution.
The Large-scale Empirical Study

104 stand-alone password meters in set of 5900 URLs

392 password generators found in 8150 URLs

86 of the 250 top Alexa domains with password meter on registration page
Stand-alone Password Meters

- 94.2% include JavaScript, 89.8% over HTTP
- 77.9% include 3rd party JavaScript, 90.1% over HTTP
- 16.4% send password to server, 76.5% over HTTP
- 1 sends hashed password to server (MD5 and SHA256)
Password Meters on Registration Pages

- 100% include JavaScript, 24.4% of them over HTTP
- 96.5% include 3rd party JavaScript, 25.3% of them over HTTP
- 59.3% send password to server, 3.9% of them over HTTP
- 0 sends hashed password (MD5 and SHA256) to server
Stand-alone Password Generators

- 89.5% include JavaScript
  - 94.6% of them over HTTP

- 76.8% include 3rd party JavaScript
  - 94.0% of them over HTTP

- 25.5% server-side generation
  - 96.0% of them over HTTP

- 3 send password back to server over HTTP
- 3 send password to 3rd party server over HTTP
GETTING IT RIGHT?

CHALLENGE ACCEPTED
The Ingredients

- reviewable JavaScript
- sandboxed: isolation from main page
- iframe: separation of included code
- Content Security Policy: blocking all network traffic
The SandPass Framework

1. Third party code is loaded into sandbox
2. Password read locally
3. Framework consults modules
4. Result is rendered

Password can not leak from sandbox
The Case Study

Password meter provided by MSB

- Secure integration of untrusted code
- Safe metering of real passwords
- Immediate response
- Easy adoption
The Conclusion

State of the Art: many services vulnerable to
- network attacker
- second party attacker
- third party attacker

SandPass
- modular
- secure integration of untrusted code

Check out SandPass online:
www.cse.chalmers.se/~andrei/SandPass/