Protecting users against XSS-based password manager abuse

AsiaCCS 2014, Kyoto
Ben Stock, Martin Johns

FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG

TECHNISCHE FAKULTÄT

Agenda

- Basics on Password Managers
  - Intention & Implementation
  - Automating attacks on Password Managers
- Analysis of browsers and applications
  - Analysis of built-in password managers
  - Study of password fields on the Web
- Our proposal to a more secure password manager
- Summary & Conclusion

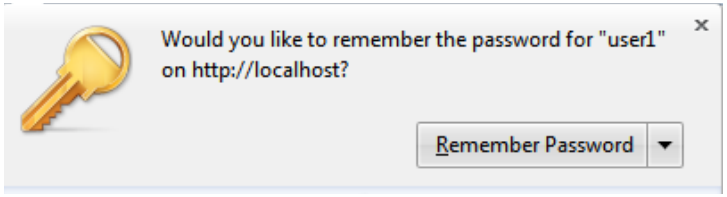
Password managers and attacks against them

FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG

TECHNISCHE FAKULTÄT

Password managers

- In a perfect world... one password per site
  - hard task to remember multiple complex passwords
- Solution: password managers
  - take burden to remember these passwords off the user
- Implementation in two locations
  1. submitting a form ➔ ask user to save password
  2. loading a form ➔ fill in user and password



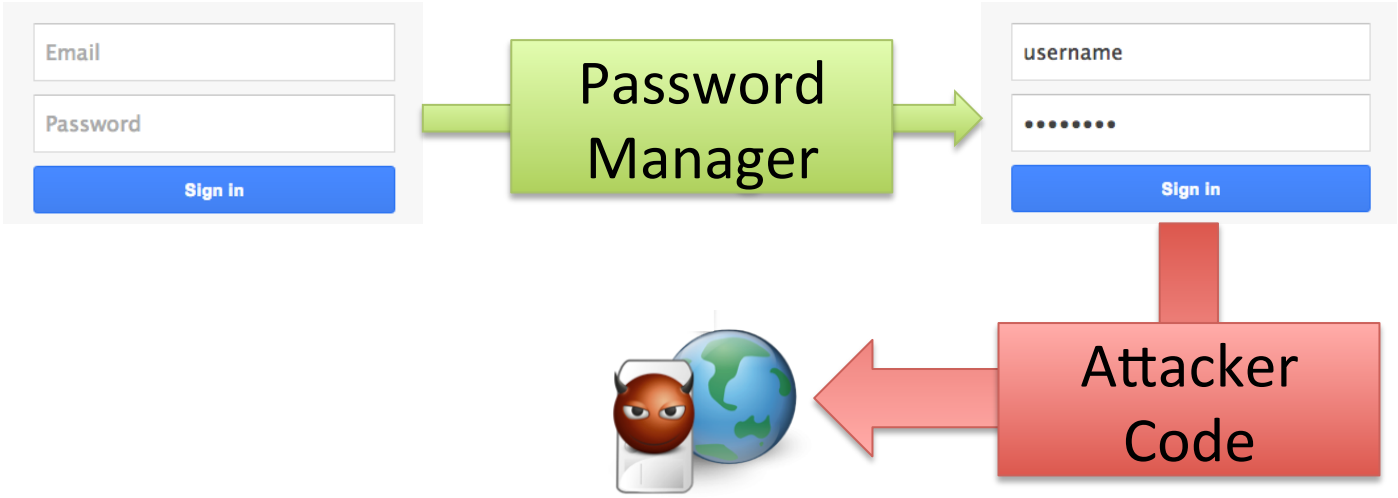
Password manager security issues

- Password manager detects form
  - looks up stored credentials
  - inserts them into form (and thus the DOM)
- DOM is accessible by JavaScript
  - both benign code and XSS payloads
- Attacker code may access field data (in clear text)

Automating attacks

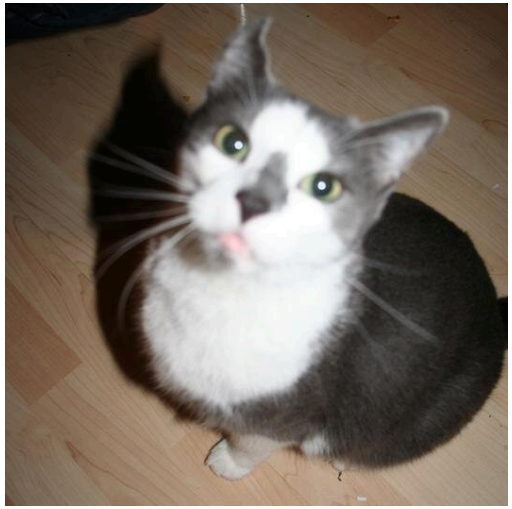
- Consider XSS attacker (similarly for network attacker)
  - capable of injecting HTML markup (and JavaScript code)

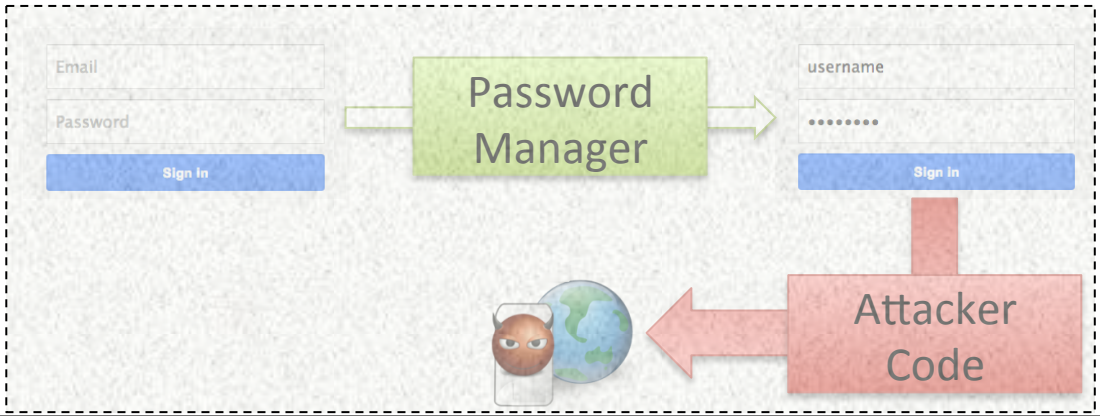
1. inject form with user/password fields
2. wait for the Password Manager to fill out form
3. retrieve credentials and send them to the attacker



Ninja exploitation

http://kittenpics.org





Factors for a successful attack

- **URL matching**
  - is XSS on any document on the domain sufficent?
- **Form matching**
  - can we use a minimal form to easily automate attacks?
- **Autofilling in frames**
  - can we exploit multiple domains the same time?
- **User interaction**
  - can we fully automate the attack?
- **Autocomplete attribute**
  - can a Web site opt out of password storage?

Analysis of browsers and applications

FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG

TECHNISCHE FAKULTÄT

What we learned thus far...

- Automated attacks are dependent on
  - password manager implementations
  - Web application implementation
    - `autocomplete=off`
    - (delivery over HTTPS)

- Two analyses to conduct
  - Password manager implementations
    - Chrome, Firefox, IE, Safari, Opera and Maxthon
  - Analysis of password fields on the Web
    - Secure delivery, autocomplete, ...

Highlights of built-in password managers*

- **URL matching**
  - only IE stores the URL, Maxthon does not even store protocol or port
- **Form matching**
  - no browser stored structure/target URL
- **Autofilling in frames**
  - only IE refuses to insert credentials into frames
- **User interaction**
  - only IE requires user interaction
- **Autocomplete attribute**
  - Chrome, Opera and Maxthon do not adhere to autocomplete

*refer to the paper for the complete analysis

Analysis of password fields on the Web

- Crawl of Alexa Top 4000
  - natural languages matching to detect login forms
  - wrapping getter for password fields to detect access

Characteristic	# Sites	% rel.	% abs.
Password field	2,143	100%	53,6%
Form on HTTPS page	821	38,3%	20,5%
Action on HTTPS page	1,197	55,9%	29,9%
Autocomplete off	293	13,6%	7,3%
JavaScript access	325	15,1%	8,1%

Summarizing our findings

- Password managers are quite relaxed in matching criteria
  - XSS on same-domain is sufficient for all but IE
  - Chrome, Opera and Maxthon don't adhere to autocomplete

- Password fields are meant to work with managers
  - only 13,6% opt-out with `autocomplete=off`

➔Password managers need to be protected against XSS attackers

Building a secure password manager

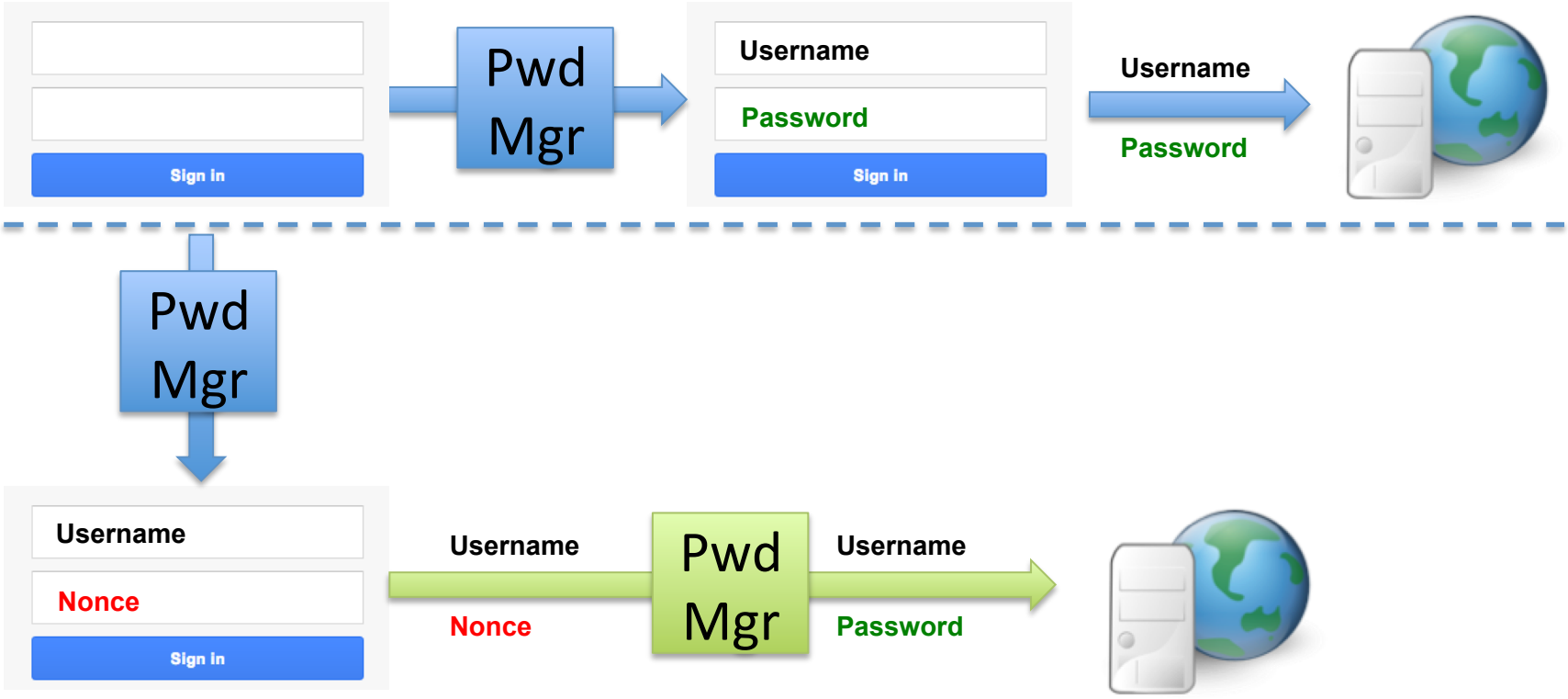
FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG

TECHNISCHE FAKULTÄT

Mismatch in notion/implementations

- Password Managers should aid in authentication
- **Authentication**: "Credentials are sent to the server"
- **Implementation**: "Credentials are inserted into forms and then sent to the server"
- **We propose to align implementation with notion**

Our proposed solution

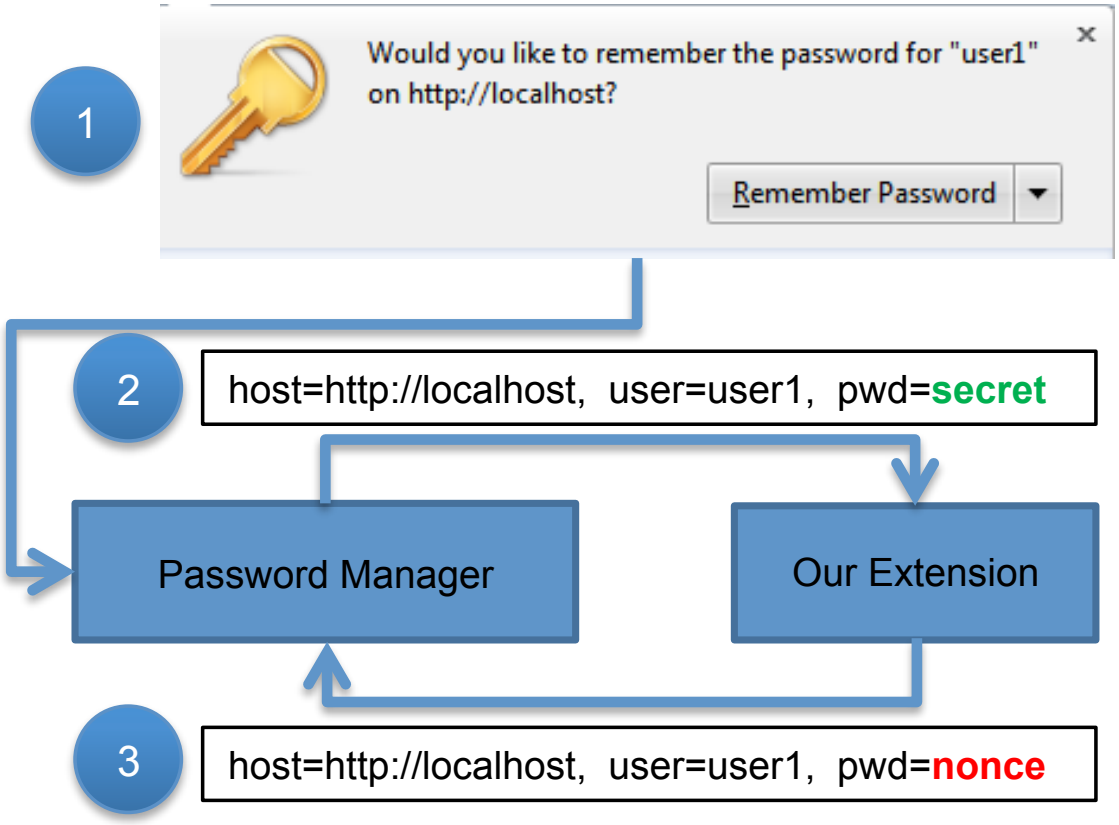


Constraints for this approach

- **Potential pitfalls**
  - Attacker changes a form's target
    - posting data to his own server / a page that reflects the content
  - Attacker changes method to GET
    - .. and subsequently reads the URL to which a frame was redirected

- **Proposed constraints**
  - strict checking of form target URL
  - exchanging nonce only in POST parameters

PoC Implementation



PoC Implementation

POST /login.php
Data: user=user1&pwd=**nonce**

POST /search.php
Data: user=user1&query=**nonce**

Our Extension

POST /login.php
Data: user=user1&pwd=**secret**

POST /search.php
Data: user=user1&query=**nonce**

Functional evaluation

- 325 domains used JavaScript to access password fields
  - 229 domains only check that field is not empty
  - 96 domains send password via XHR
    - 23 domains hash password before sending it out
    - 1 domain applies base64 encoding
    - 6 domains send password in GET parameters
- 30/2143 domains have issues with our solution
  - 98,6% of all domains we analyzed work just fine
  - storing passwords in XHRs is not currently supported by browsers

Summary and Conclusion

FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG

TECHNISCHE FAKULTÄT

Summary & Conclusion

- Most current implementations of password managers allow for automatic stealing of passwords

- Cause: passwords are inserted into forms
  - not into outgoing request

- We propose alignment of notion and implementation
  - PoC implemented as a Firefox extension
  - working with 98,6% of domains we analyzed

Thank you for your attention!

Special thanks to my student workers Eric Schmall and Armin Stock

@kcotsneb
http://kittenpics.org
ben.stock@fau.de

FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG

TECHNISCHE FAKULTÄT