SESSION IDENTIFIER ARE FOR NOW, PASSWORDS ARE FOREVER

XSS-BASED ABUSE OF BROWSER PASSWORD MANAGERS

Ben Stock, Martin Johns, Sebastian Lekies

Browser choices

About us

- Ben Stock, Martin Johns, Sebastian Lekies,
- Security Researchers at Uni Erlangen, SAP and Uni Bochum
- More and stuff at http://kittenpics.org

About this talk

- Results of an analysis of different browser's password managers
- Study on password fields on the Web
- After all the scary parts: a potential solution to the problem

Cross-Site Scripting

a.k.a. XSS (duh)

The Same-Origin Policy

- Question: why can't attacker.org read the visitors emails from GMail?

- Answer: the Same-Origin Policy is "in the way"
  - Only resources with matching protocol, domain and port may gain access

- That makes for a sad attacker (and his kitten)

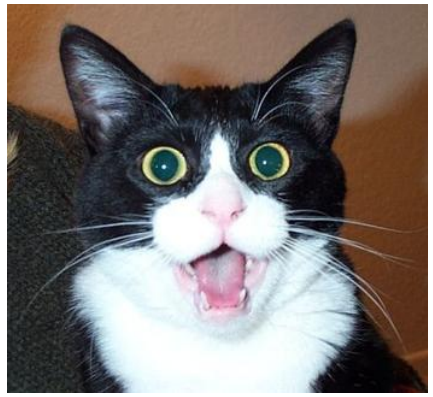
XSS – the underlying problem

- Web Apps process **data**
  - Which was provided by the user
  - POST, GET, headers, ….
- **Data** might be stored, or echoed back directly
- **Data** `<script>alert(1)</script>` is actually **Code**
- **…** interpreted by the victim's browser, executed in the origin of vulnerable application
- Attack method
  - Find flaw in Web application that allows injection of CODE, not just DATA
  - Make victim visit that site

➔ We can read your GMails ☺

XSS – what an attacker can do

- Open an alert box!



- Hijack a session
  - Oldest trick in the book: steal their cookies
  - Force victim to “click” a link (or post something about this talk on Twitter)
- Alter content
  - Display fake content
  - Spoof login forms
- **.. Steal your password manager’s passwords**

Types of XSS

	Reflected	Stored
Server	`<?php echo "Hello ".$_GET['name']; ?>`	`<?php $res = mysql_query("INSERT…".$_GET['message']); […] $res = mysql_query("SELECT…"); $row = mysql_fetch_assoc($res); echo $row['message']; ?>`
Client	`<script> var name = location.hash.slice(1)); document.write("Hello " + name); </script>`	`<script> var html= location.hash.slice(1); localStorage.setItem("message", html); […] var message = localStorage.getItem("message"); document.write(message); </script>`

Isn't XSS so 2010?

- **Our CCS paper from 2013**
  - ~10% of all Web sites carry DOM-based XSS
- **Our BlackHat US talk on bypassing the XSSAuditor**
  - 80% of all domains could still be exploited

Key findings include:

From a vulnerability class perspective, the research team made these discoveries:

- Cross-Site Scripting regains the number one spot after being overtaken by Information Leakage last year in all but one language. .Net has Information Leakage as the number one vulnerability, followed by Cross-Site Scripting.

Passwords & Password Managers

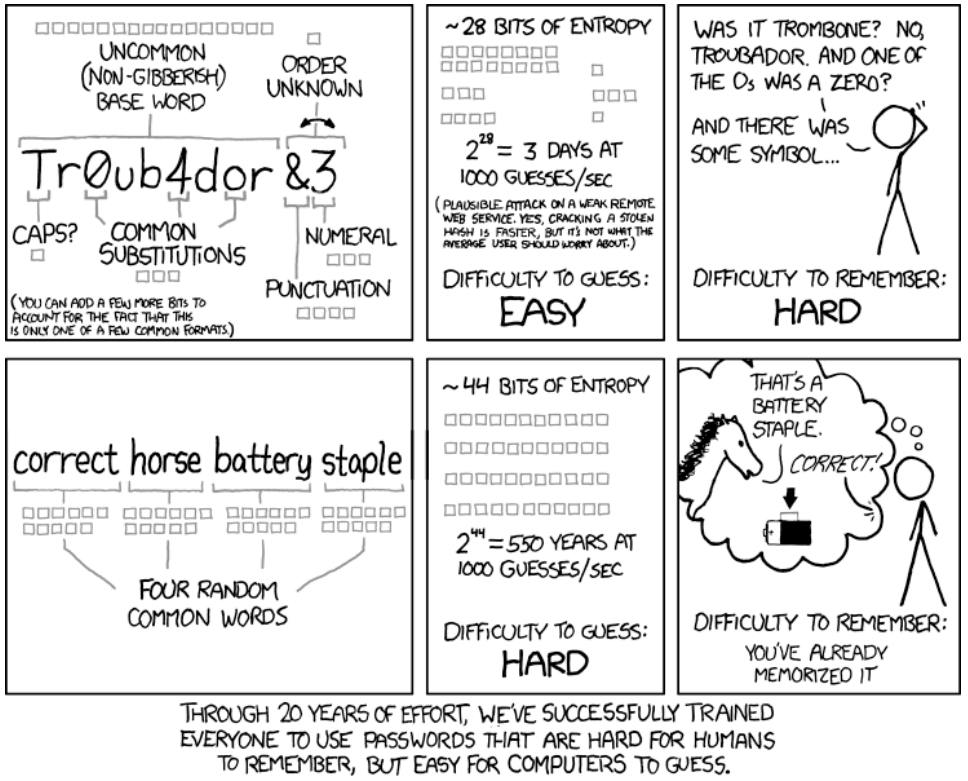
Passwords on the Web

- **Still first (and most of the time, only) form of authentication**
- **In a (semi) perfect world, we would have one password per site**
  - sufficiently complex to avoid brute forcing
  - reality….

Here are some interesting facts gleaned from my most recent data:

- 4.7% of users have the password *password;*
- 8.5% have the passwords *password* or *123456*;
- 9.8% have the passwords *password, 123456* or *12345678;*
- 14% have a password from the top 10 passwords
- 40% have a password from the top 100 passwords
- 79% have a password from the top 500 passwords
- 91% have a password from the top 1000 passwords

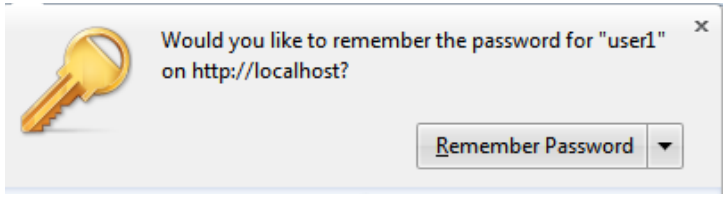
Source: https://xato.net/password/more-top-worst-passwords/



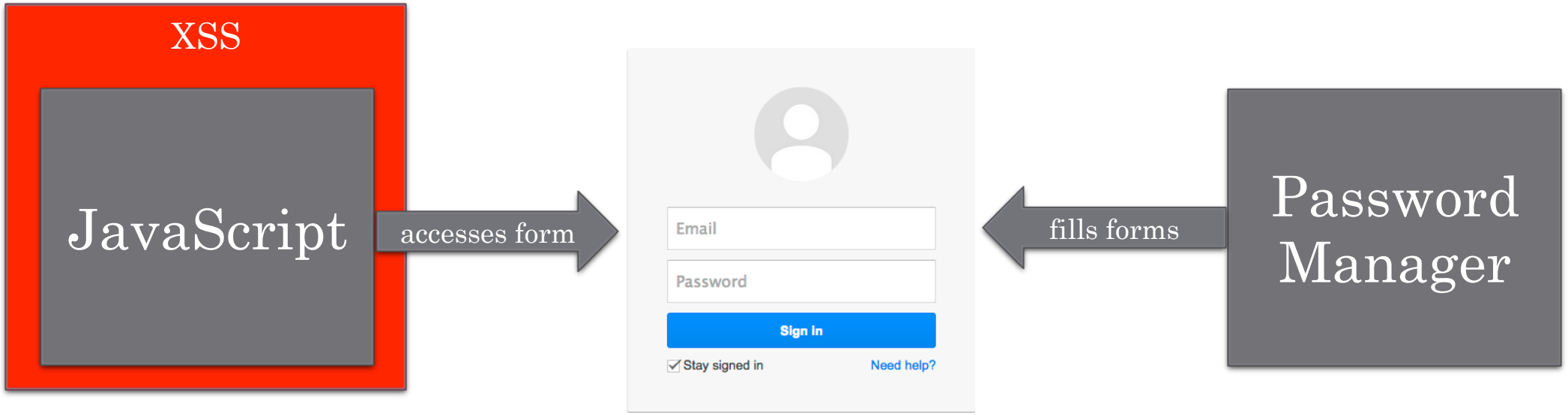
Solution: A Password Manager

- **Takes burden of remembering passwords off the user**
  - shipped with every browser these days

- **Two functionalities:**
  - **1.) Store credentials on first submission**
  - **2.) Insert credentials on subsequent visits**



Password Managers and XSS



DEMO: Passwords inserted via password managers are accessible via JavaScript.

Security Considerations

- World without password managers:
  - **XSS attacks only work when the user is authenticated**
  - **Attacks are limited to the user‘s session**

- World with password manager:
  - **Passwords can be accessed, no matter whether the user is authenticated or not**
  - **Attacks are limited to the life time of the password**
  - **Potentially affects other services as well (password reuse)**

➢ **Current password managers make XSS attacks more severe**

Security mechanisms in Password Managers

and how we can steal your credentials anyways ☺

Five key features of PW Managers

- **URL Matching**
- **Form Matching**
- **User Interaction**
- **No auto filling in frames**
- **Autocomplete attribute**

DEMO: URL and form matching

DEMO: User interaction required

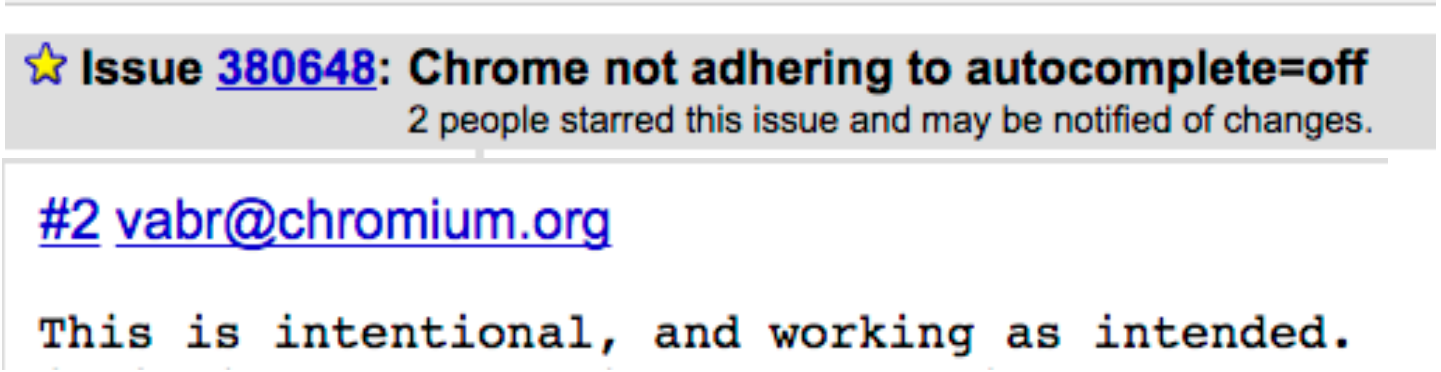
HTML5 autocomplete

The **"off"** keyword indicates either that the control's input data is **particularly sensitive (for example the activation code for a nuclear weapon)**; or that it is a value that will never be reused (for example a one-time-key for a bank login) and the user **will therefore have to explicitly enter the data each time**, instead of being able to rely on the UA to prefill the value for him; or that the document provides its own autocomplete mechanism and does not want the user agent to provide autocompletion values.

DEMO: Adheres to autocomplete

Our notion vs. Google's notion

- **Our notion**
  - if programmer puts `autocomplete=off` to any part of the login field, at the very least that part should not be stored
  - better even, if nothing is stored for that form
- **We opened a bug report**
  - while in clear violation of the spec, Google wants it that way



Trade-off:

Usability vs. Security

The State of the Web

its passwords, that is

What are login forms like out there?

- **Gain better understanding how sites handle forms**
  - Using autocomplete=off to opt out of password managers
  - X-Frame-Options (now deprecated) to ensure no framing

- **We implemented a simple crawler**
  - Scans entry pages to domains for
    - login fields
    - indicative keywords ("login", "sign in", ...)
  - Wraps access to these fields to notice JavaScript access

Analysis of Web password fields

- **Crawl of Alexa Top 4000**

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

Similar attacker model

Man-in-the-Middle attacks

Similar attacker model

- **Work by Silver et al. (USENIX Sec '14)**
- **Provide free WiFi in cafés**
- **User loads Google, multiple hidden iframes with target login pages + injected XSS payload**
  - Password manager autofills, credentials can be stolen
- **According to authors, quick to automate**

Comparing the attacks

XSS attacker

- needs XSS vuln on target
- works regardless of HTTPS

Network attacker

- does not work with HTTPS
- Can eavesdrop on posted credentials

Attacks against password managers can easily be automated

Bottom line

- **Default password managers in browser are not that secure**
  - issues related to
    - auto filling
    - ignoring autocomplete
    - missing form verification
- **Can we come up with something better?**
  - For attacks against untrained users

On a positive note

We have a solution…

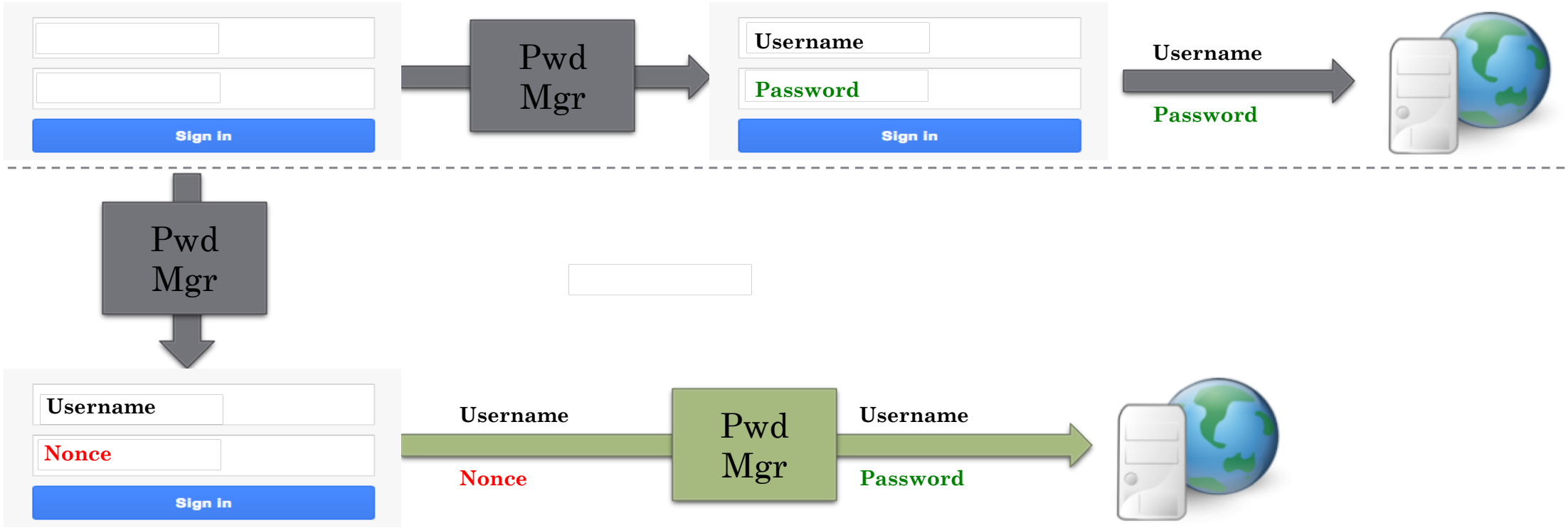
Mismatch in notion/implementations

- 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

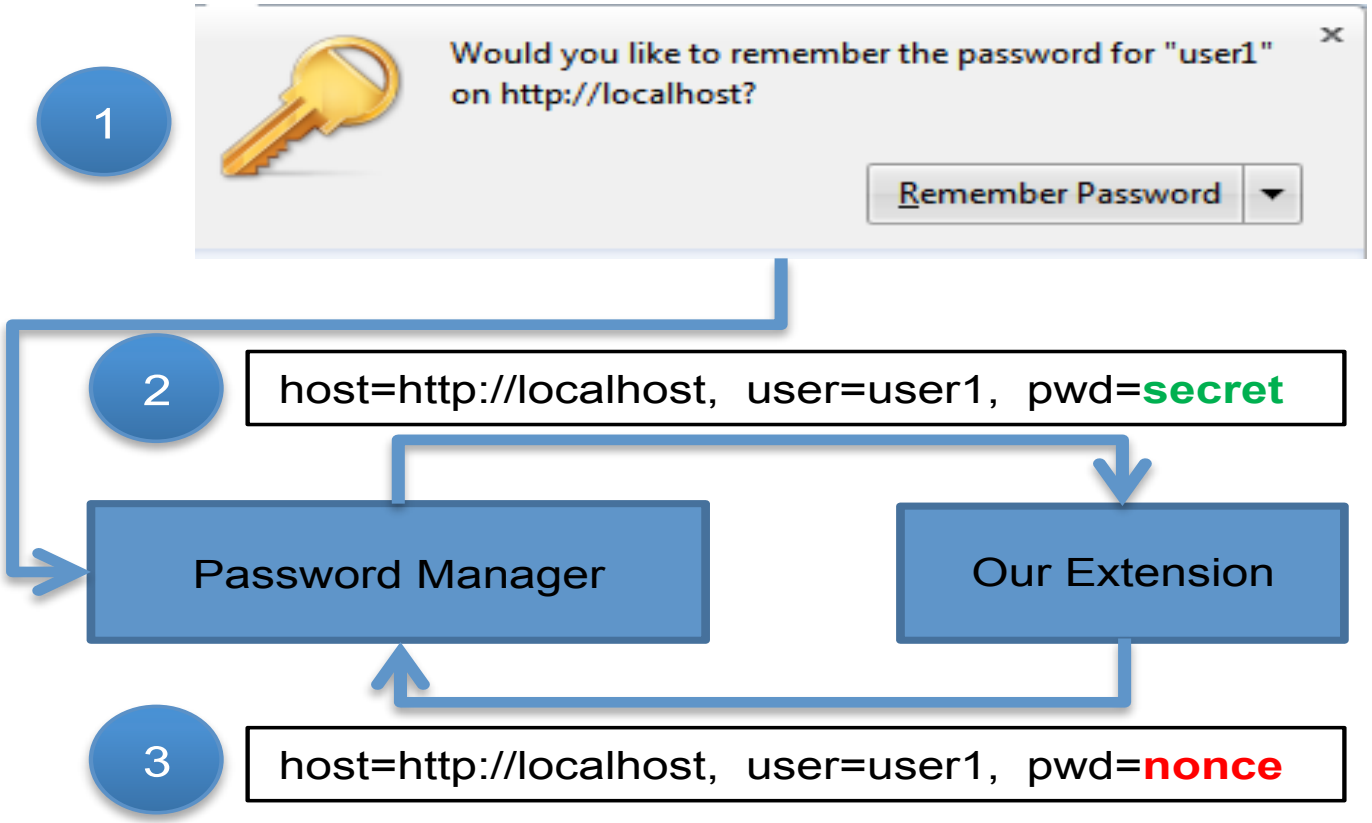


Constraints for this approach

- **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

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

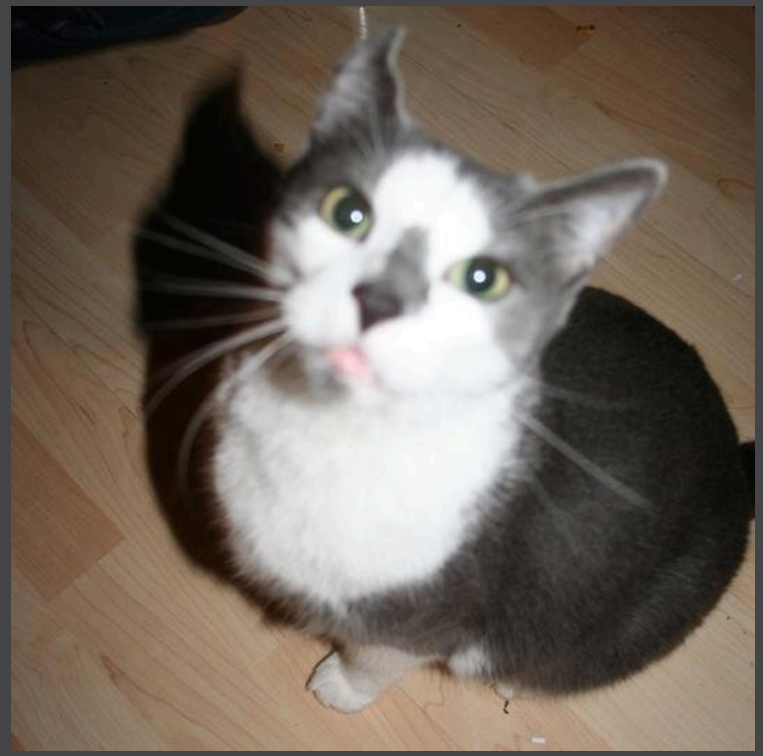
Conclusion

What to take away?

- **Current generation of browser-based password managers is quite vulnerable**
  - even if people "hate" IE, in this instance it is quite secure
- **Way too many credentials are posted over HTTP**
  - network attacker has an easy job
- **At least, against an XSS attacker, we can do something**
  - harder against network attacker, though
  - Possible solution: enforce https for requests carrying credentials

Thank you

visit us at kittenpics.org



Sebastian Lekies **Ben Stock**

@sebastianlekies **@kcotsneb**