The Shepherd Project

— Automated security audits of web login processes

Benjamin Krumnow

- Employee at the TH Köln
- External PhD student (50%) at the OU (~2 years)
  - H. Jonker, M. Van Eekelen, H. Vranken, S. Karsch
  - Joined the Shepherd project in Feb/ Mar 2017

- Karate, surfing, hiking & caving
- Vegetarian
- Fascinated by information security and privacy

Technology
Arts Sciences
TH Köln

Project Members

Marc Sleegers

- Initial Project "Shepherd" [1]
- B.Sc. in 2017

Hugo Jonker

- Supervision in all projects

Jelmer Kalkman

- Bachelor project
- Single Sign On and refactoring

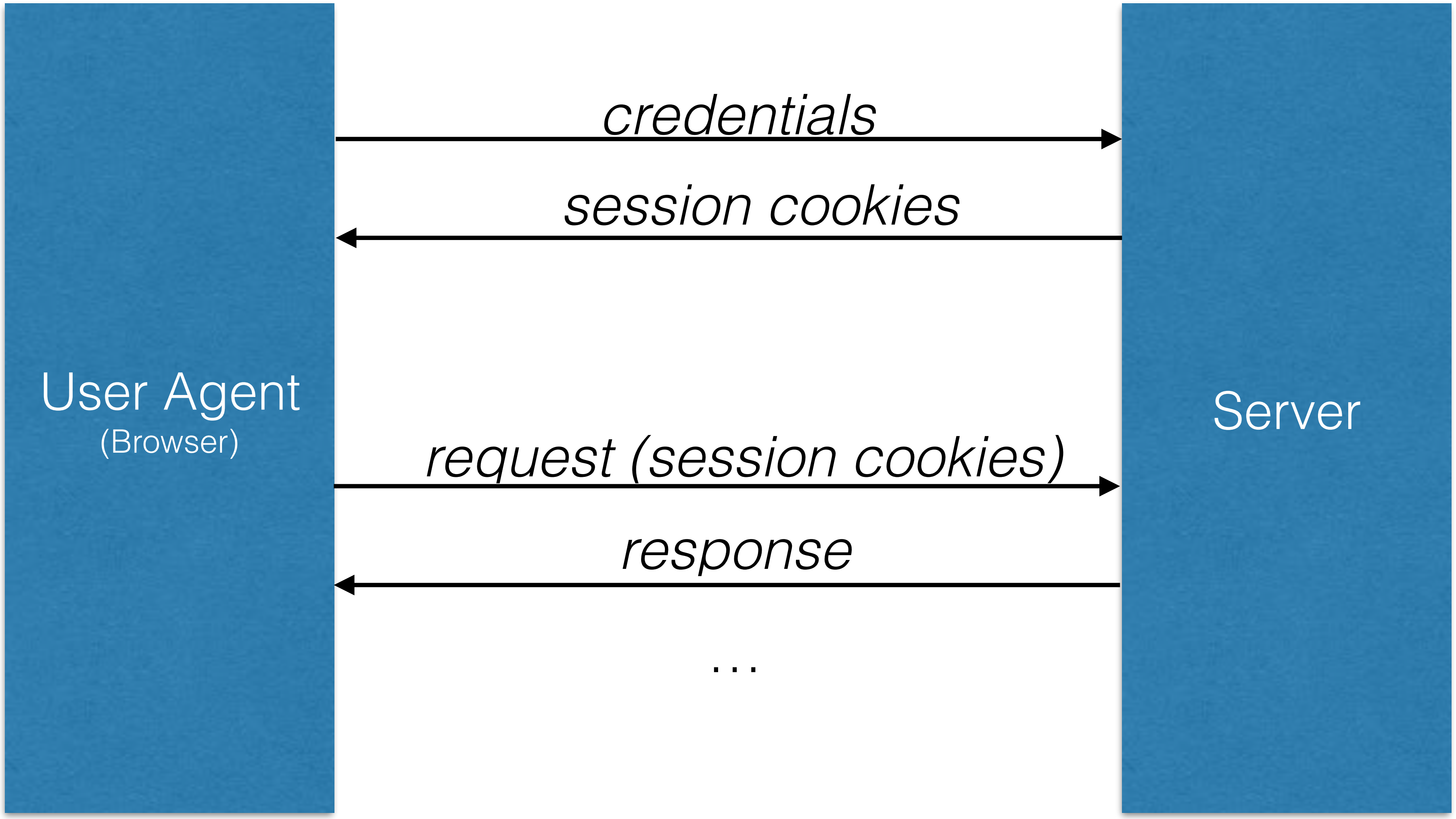
Alan Verresen

- Bachelor project
- Single Sign On and refactoring

Background: Login Process

Background: Login Process

request/response



User Agent
(Browser)

credentials

session cookies

request (session cookies)

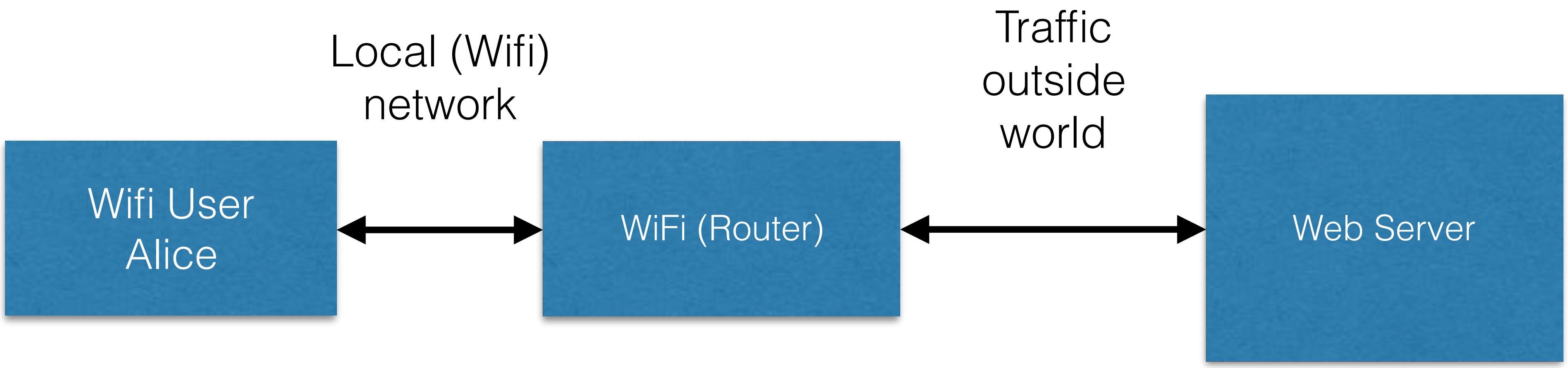
response

…

Server

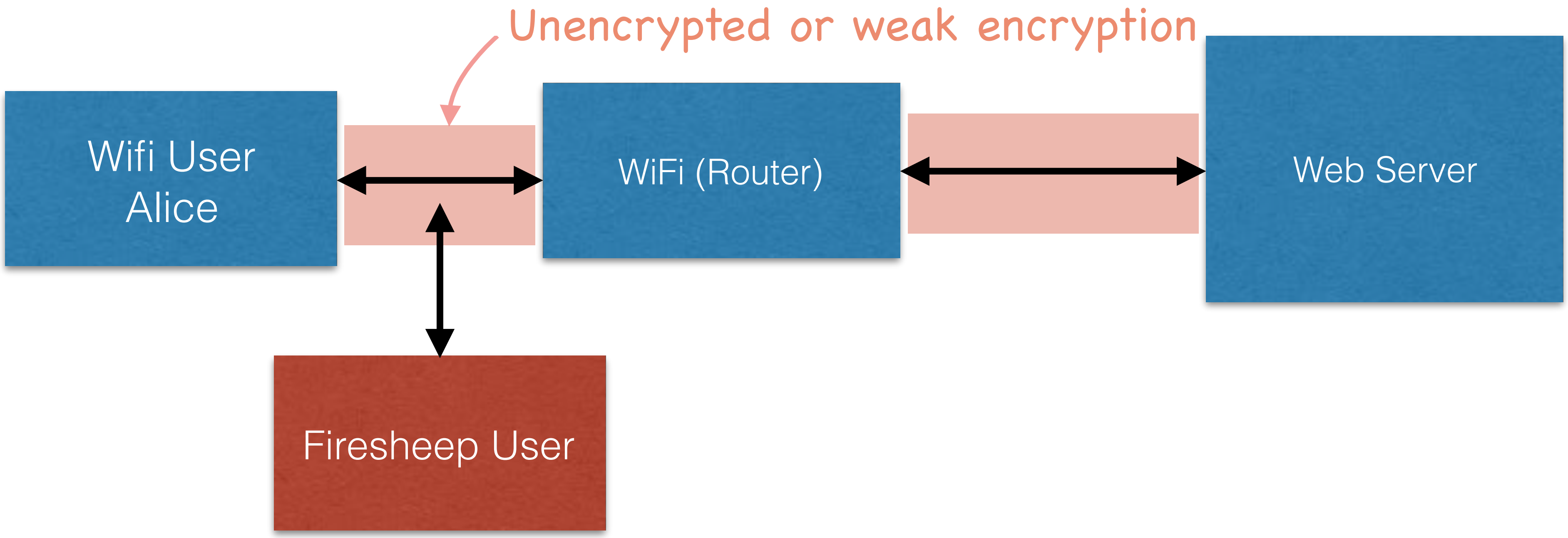
Motivation: Firesheep 2010 [2]

- Login process via an unencrypted channel
  - session can be hijacked or accounts stolen



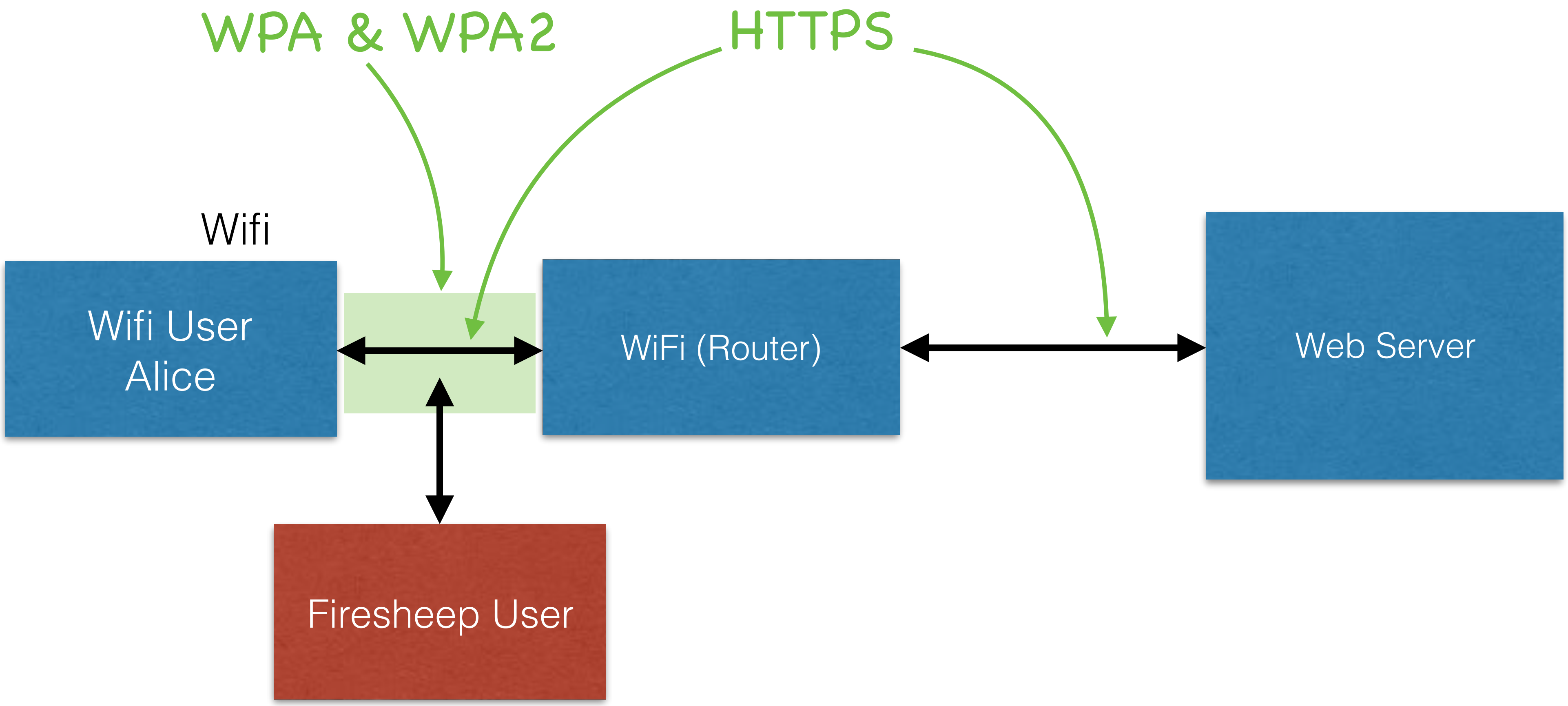
Motivation: Firesheep 2010 [2]

- Login process via an unencrypted channel
  - session can be hijacked or accounts stolen
- Automated capturing of session cookies
- Hijacking sessions by a “click”
- Popular services like Facebook, Google and co. fixed this issue!



It’s 2018! What has changed since then?

- Encryption
- Browser extensions and developments (Cookie flags, HSTS, HKPK)
- New possible login mechanisms (Single-Sign-On, HTTP bearer tokens)



Research questions:

How much have login process security measures been adapted?

How much have login process security measures been adapted?

1. Are these vulnerabilities still valid?

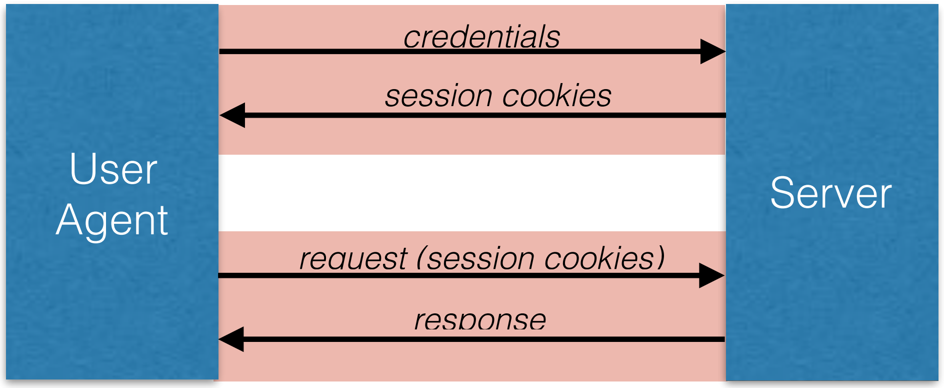
   —> Evaluate session stealing attacks in a lab and in the wild

   —> Evaluate attacks on Single-Sign-On based sessions

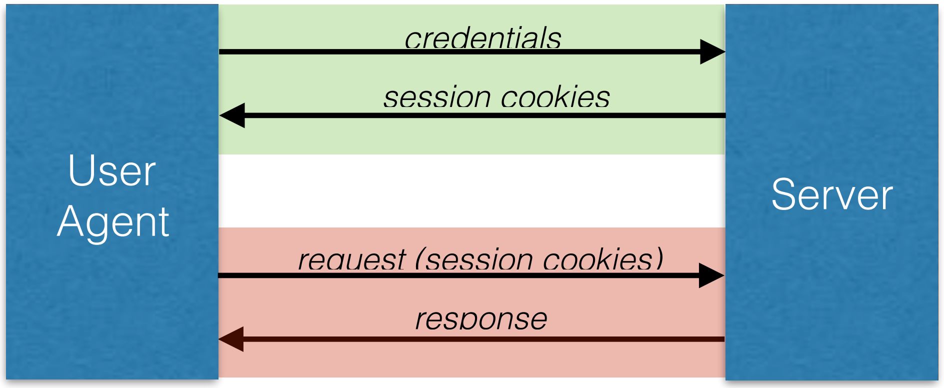
Evaluation of vulnerabilities

- Three kinds of vulnerabilities evaluated in a lab

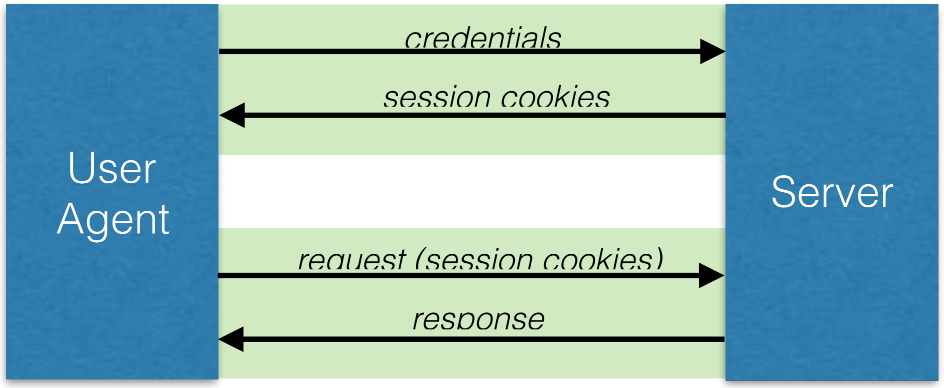
1. All over HTTP -> Leaks even credentials



2. HTTPS for the login and fallback to HTTP afterwards

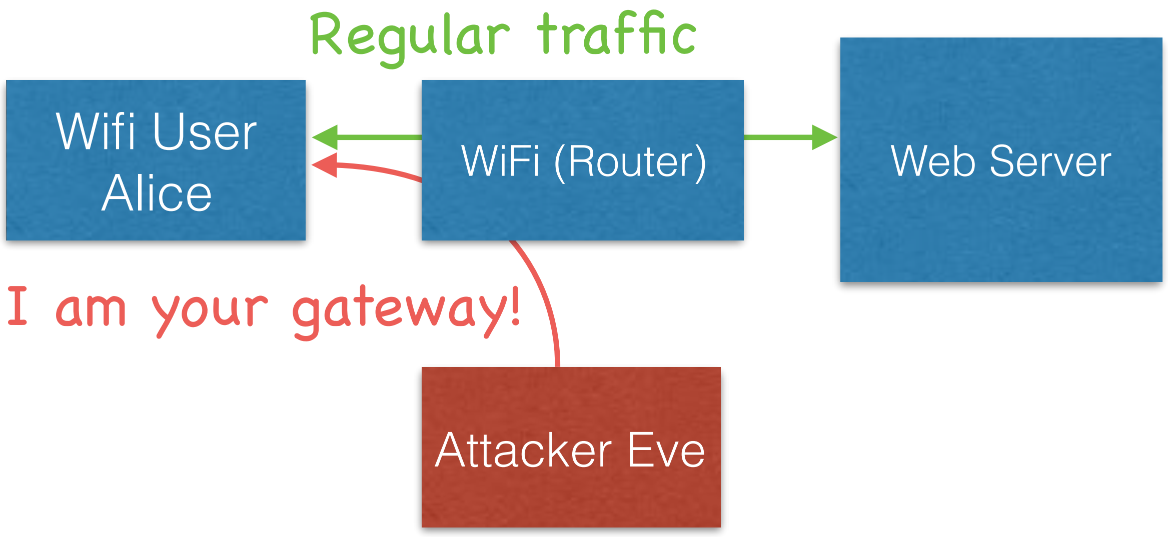


3. All over HTTPS, but misses the secure flag. Single HTTP request sufficient for attack

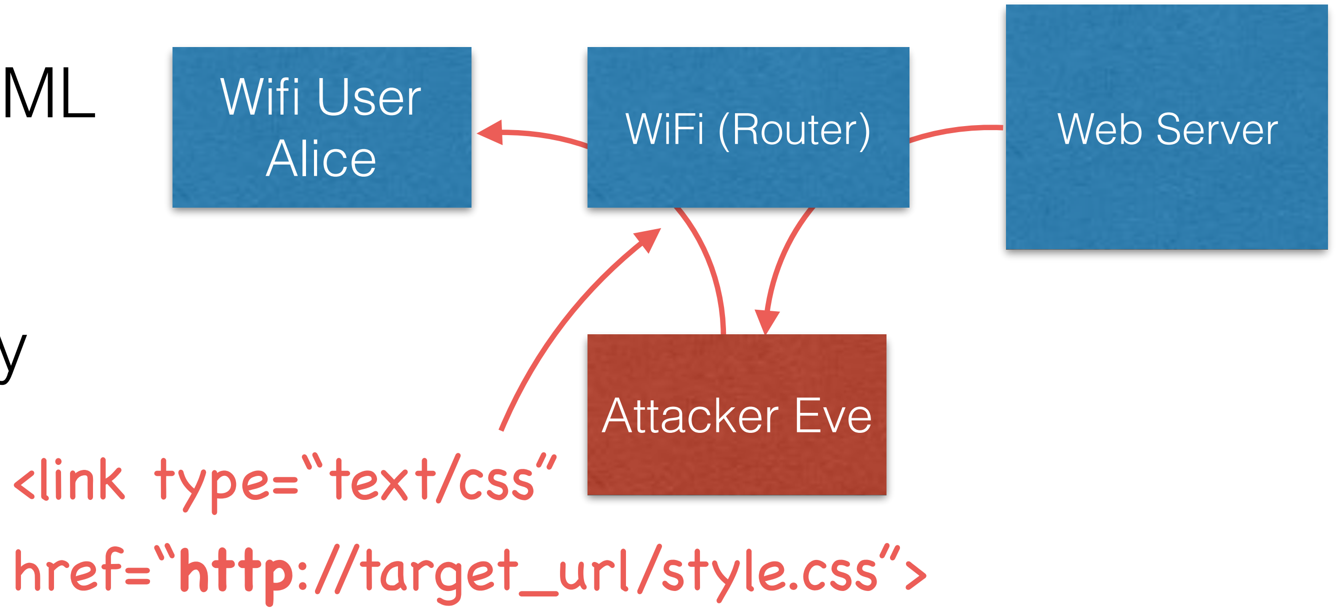


Automatic attack

1. Become a MITM on the network layer
   - ARP spoofing attack to re-route traffic (IPv4 only!)
   - Modify package IP addresses
   - See [10] for more MITM attacks



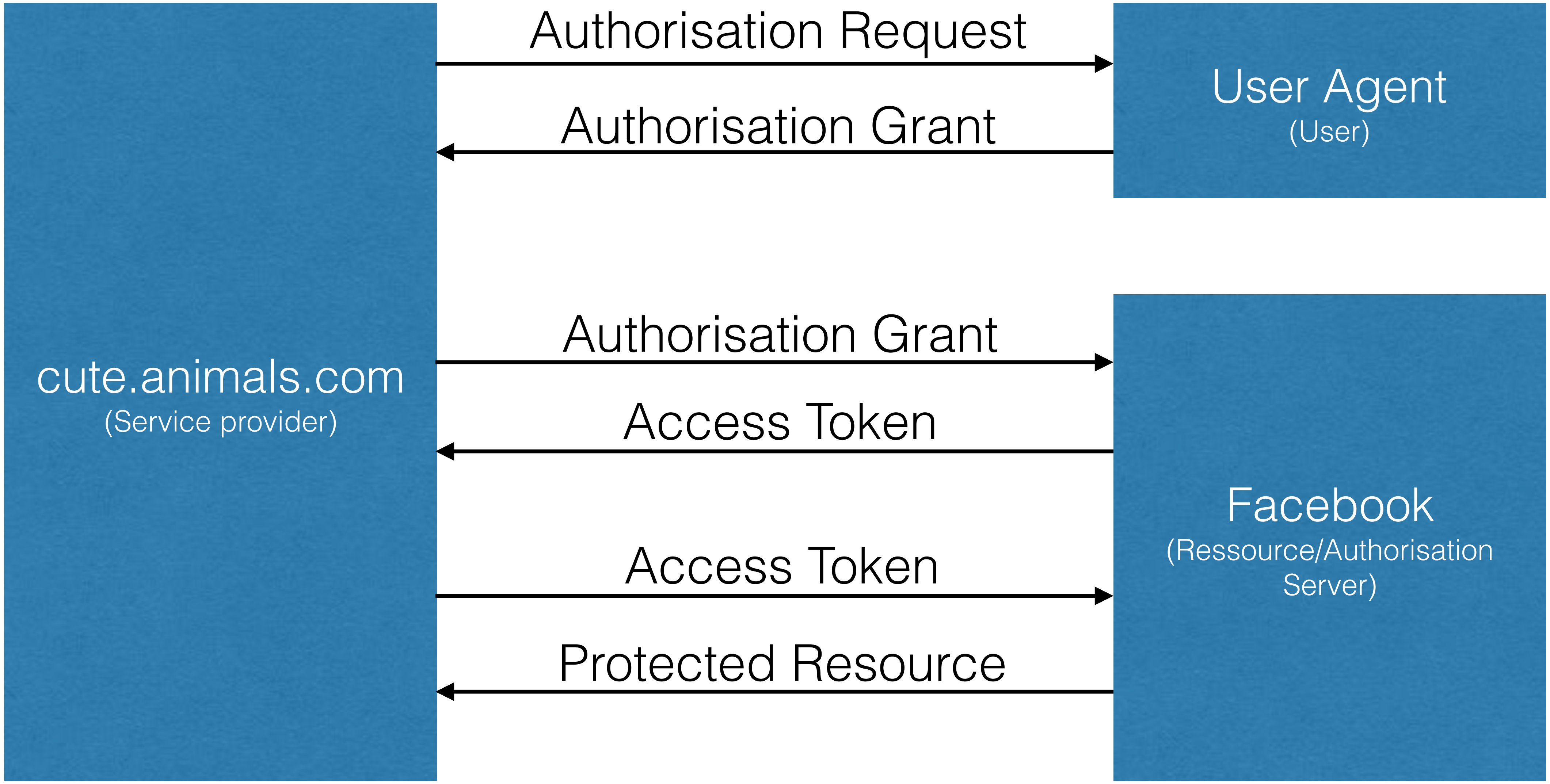
2. CSRF attack with modifying HTML sent over HTTP
   - Injecting elements in HTTP response within a HTML body
   - (Capture cookies)


Wifi User Alice
WiFi (Router)
Web Server
Attacker Eve
<link type="text/css" href="**http**://target_url/style.css">

Does that work for Single-Sign-On

Attacking Sessions established with OAuth

- Example OAuth flow



How much have login process security measures been adapted?

How much have login process security measures been adapted?

1. Are the vulnerabilities still valid?

   —> Evaluate session stealing attacks in a lab and in the wild

   —> Evaluate attacks on Single-Sign-On based sessions

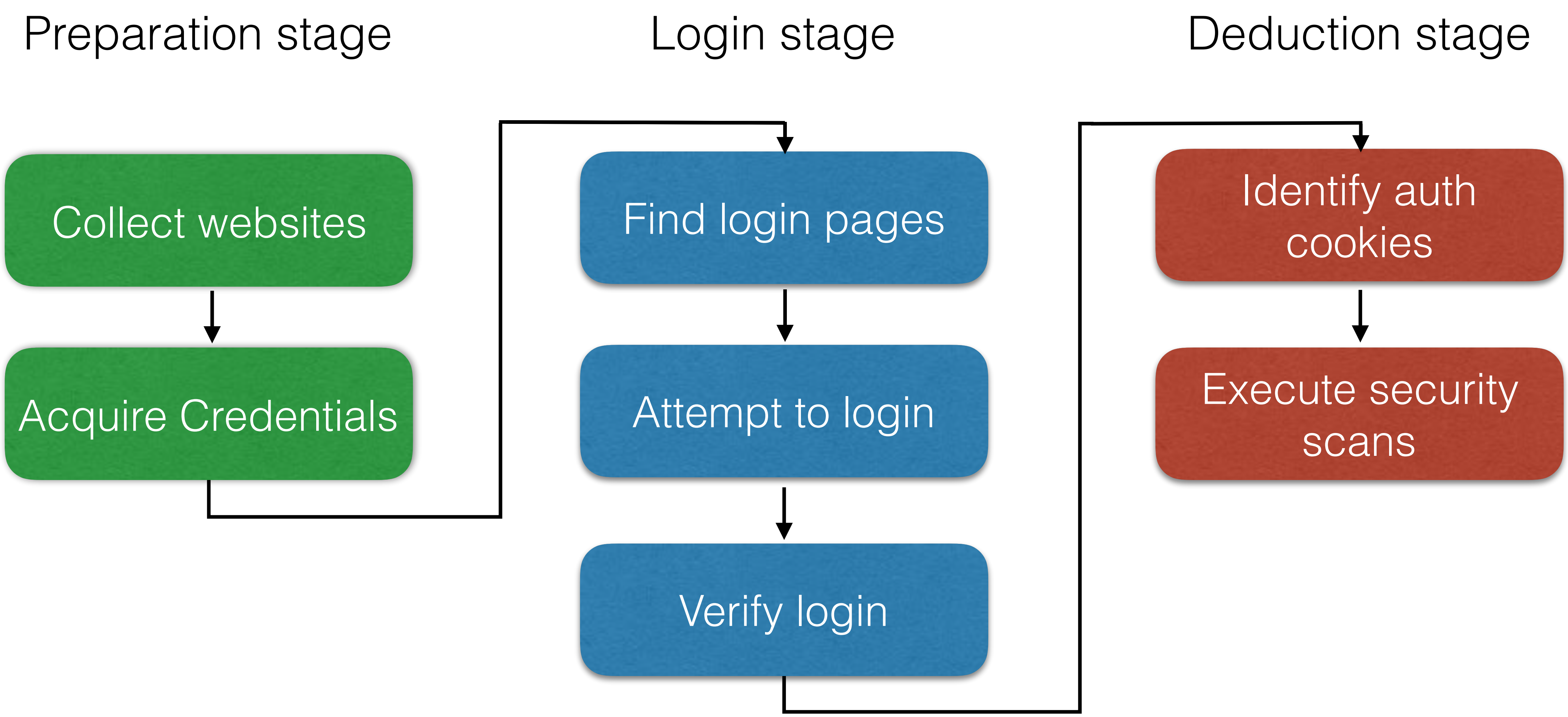
2. How many sites are still vulnerable to such attacks?

   - We need to look at the cookies
   - Analysing websites with Single-Sign-On logins for "*homegrown*" sessions

   —> Build a scanner for websites to search for possible session attacks

Scanning the web for login process security

The scanner at a glance



Preparation stage

Collect websites

Acquire Credentials

Login stage

Find login pages

Attempt to login

Verify login

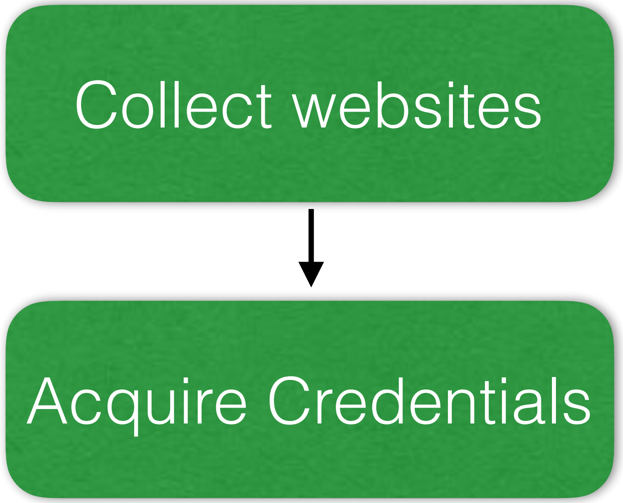
Deduction stage

Identify auth cookies

Execute security scans

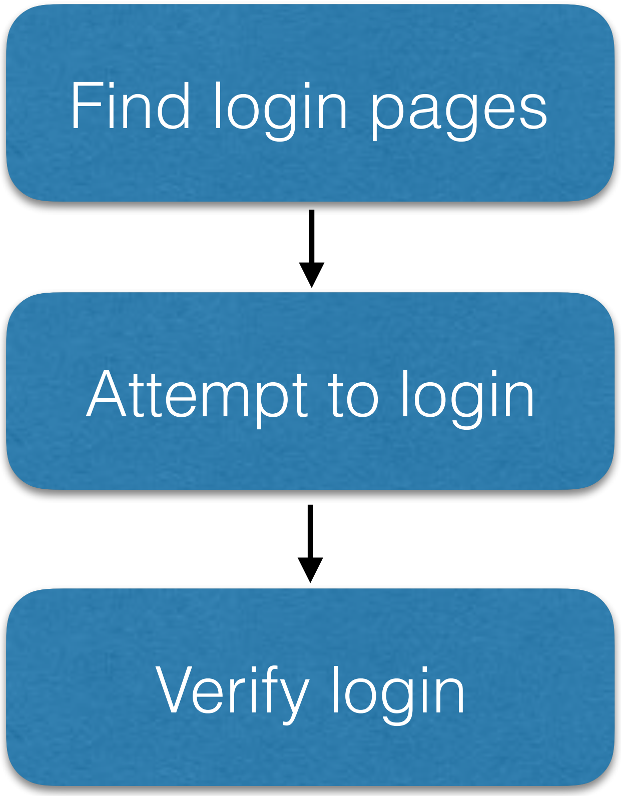
Preparation stage

- Alexa Top 1 Million web sites
- BugMeNot (BMN) - Service user-generated credentials
- Single-Sign-On (SSO) credentials
  - Importance: Unique criteria and study is not biased by relying on the BMN database



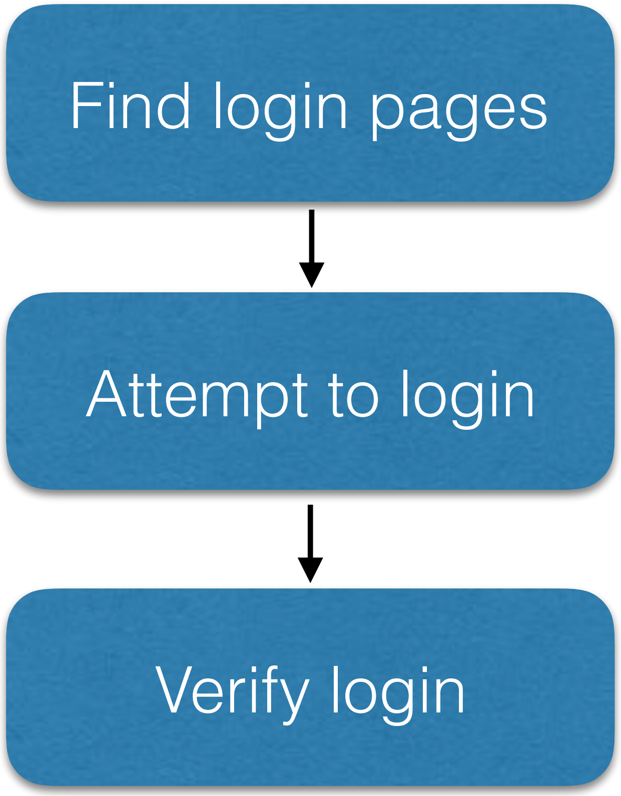
Login stage

1. Traverse web sites
    - Assumption: login page is reachable from landing page
    - Landing page, urls, clickable elements, brute force, urls 2nd level
2. Coverage of 4 login types



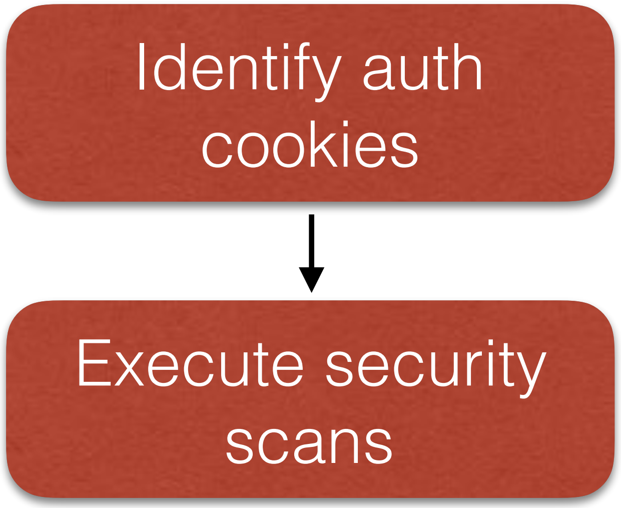
Login stage

3. Verify successful logins
   - Disappearing of the password field
     - Getting blocked, account is restricted, captchas, page switch
   - Presence of account details, keyword “logout” or login area



Deduction stage

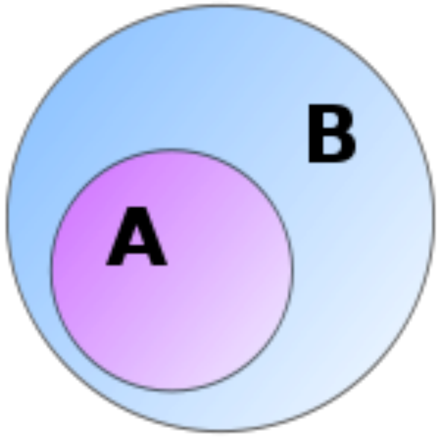
- Finding authentication cookies
  - Working verification function necessary
  - Eliminate cookies, which do not contribute to the login



Identify auth cookies

Execute security scans

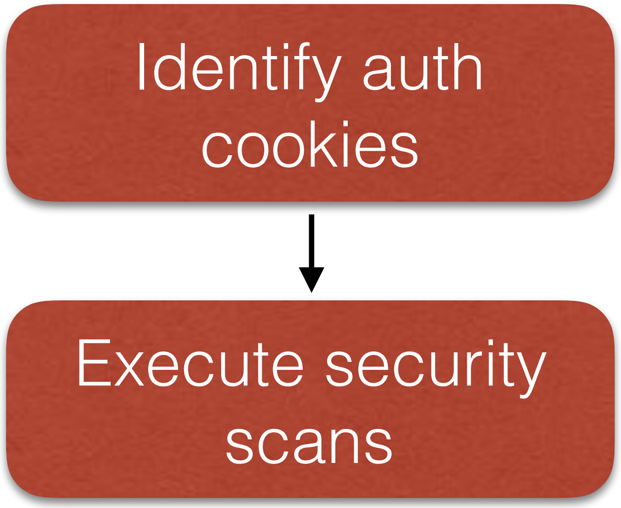
- Previous work as solution Mundada et al. (2016) and Calzavara et al. (2014) [7,8]
  - Large search space, because any subset is possible (2^n, exponential in n)
  - Fast reduction by removing supersets of A and all subsets (power set) of ¬A



B is a superset of A (B⊇A)[6]

Deduction stage

- Execute security scans
  - Cookie Flags: SameOrigin, Secure, HTTPOnly
  - HSTS and HKPK detection
  - Cookie fixation



Identify auth cookies

Execute security scans

Performing the study

The study

1. Build credential pool for logging in
    1.1. Creating fake Single Sign On (SSO) accounts
    1.2. Source credentials from BugMeNot with a static scanner

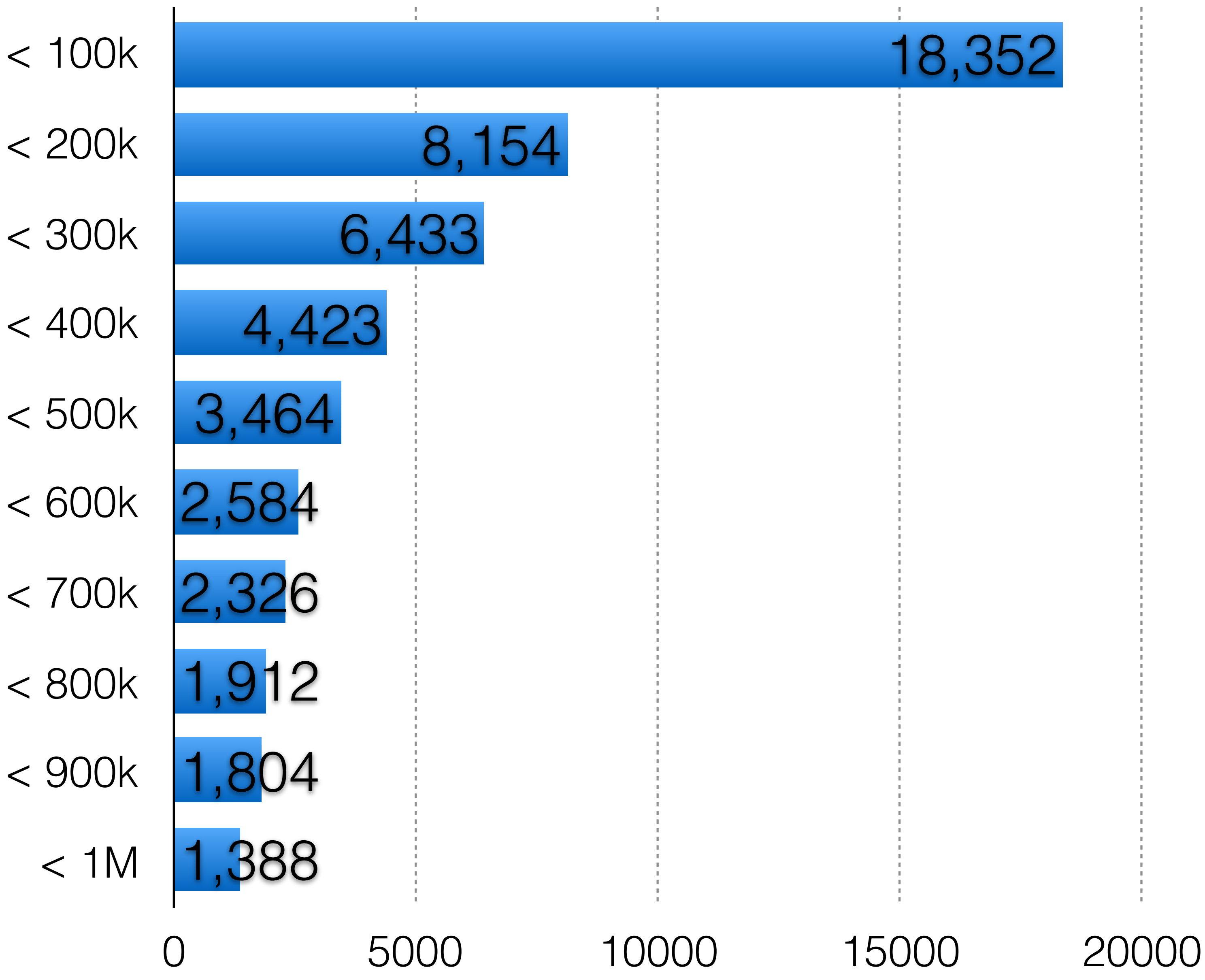
2. Scanning with a dynamic scanner (Selenium)
    2.1.~65K domains with BugMeNot credentials
    2.2. Alexa top 1 Million with SSO credentials

Overview BugMeNot

Sourcing Alexa 1 M (late Feb)

- No. of credentials: 131,034
- No. of sites : 50,840
    - refresh before scan
- No credentials for : ~949K
- Errors : 222
    - Error 404 - Bug

Sites with credentials within in the Alexa 1M



BugMeNot: old vs new set

Previous results (late Oct):

- Fresh Alexa Top 1M dataset
- gave us ~59K domains vs. ~50K
- 14,888 domains were missing in the new set
- 6,118 new sites

- Overall: 65,728 domains

Scanning

Runtime performance

- 2 Servers, 5 browser instances each: ~7.500 sites per machine a day
- Average scanning time: 61 seconds
- Average performance to find session cookies
  - Duration: 51 seconds
  - Executions: 11,7 (∅ 8 cookies)
  - Session cookies found: 1,5

- SSO scanner still under development:
  - Currently limited to Facebook
  - Today: Early results with 500 websites
  - Goal before the conference 100K

Performance of the scanner

Procedure	BMN 65728	%	SSO ~300	%
Login page detected	38421	58%	79	26%
Authenticated	11445	61K: 18% 38K: 29%	35	44%
Verified	LP: 4790 LA: 5858	41% 51%	7	20%
Session cookies found	6378 (7105)	89%	-	-
Failed scans	4449	6%	-	-
Captchas	2341	3%	-	-

Security Results

Detection		BMN 11445	%	Deducted (6379)	%
Header	HSTS[1]	1416	12%	5521	77%
	HKPK[2]	76	0,6%	43	0,6%
Cookies Flags	No SameSite	0	0%	0	0%
	No secure (but HSTS)	6086 (214)	53%	2693 (50)	42%
	No HTTPOnly	4907	42%	2639	41%
Cookies	Fixation	736	6,4%	175	2,7%

1) HTTP Strict Transport Security
2) HTTP Public Key Pinning

False-Positive and False-Negatives

- Chances for False-Positives and False-Negatives
    - Login page found, login success, verifying
        - Websites with credentials but no login
        - Password fields can disappear
        - Simple usernames

- Checking False-Positive
    - Reproducing runs is time consuming
    - Storage of pictures (Disk space, visible signs)

- Current solution: Confidence level

Practicability Challenges

- Runtime performance
    - Selenium API contains slow functions, which can become tricky to detect
    - Dynamic timeout estimation

- Optimisation page traversing
    - Heuristics vs. probability model
    - Scan and execute vs. first scan, then execute

- Stability
    - Selenium timeouts, running out of memory and browser crashes
    - Re-scanning vs. stage freezing [3]

Conclusions of the study

- Approach
  - Automatic logging into websites is a non-trivial task
  - Pattern-based approach with taking immediate actions has got limitations
  - Suitability of selenium for web scraping (also see [3])???
  - Comparison with [7,8,9]

- Vulnerabilities
  - HSTS still rarely used (same for SameSite flag and)
  - Secure flag missing for over 42 % with high certainty
    - Might be biased by BugMeNot database
  - Low HKPK usage <— Further investigation needed

Conclusions for the PhD project

- Improve the scanner
- Account for more countermeasures
- Classify websites
- Other login methods (Bearer tokens, OpenID,…)
- Transforming more functions to the core framework (usage in future projects)

References

[1] Counting Sheep - Analysing online authentication security

Marc Sleegers, March 2017

[2] FireSheep

Eric Butler, 2010, https://codebutler.github.io/firesheep/tc12/, last seen 23th of March 2017.

[3] A.: Online tracking: A 1-million-site measurement and analysis

Engelhardt, S., Narayanan. In: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, pp. 1388–1401 (2016)

[4] RFC6749 - The OAuth 2.0 Authorization Framework

Internet Engineering Task Force (IETF), 2012, https://tools.ietf.org/html/rfc6749

[5] BugMeNot

http://bugmenot.com/terms.php

[6] Subset

Wikipedia, https://en.wikipedia.org/wiki/Subset, last seen 22nd March 2018

References

[7] Half-Baked Cookies: Hardening Cookie-Based Authentication for the Modern Web
Yogesh Mundada, Nick Feamster, and Balachander Krishnamurthy. In Proc. 11th Asia Conference on Computer and Communications Security (ASIACCS), pages 675{685, 2016

[8] Quite a mess in my cookie jar! Leveraging machine learning to protect web authentication
Calzavara, Stefano, Gabriele Tolomei, Michele Bugliesi, and Salvatore Orlando. In Proceedings of the 23rd international conference on World wide web, pp. 189-200. ACM, 2014.

[9] Measuring login webpage security
Steven van Acker, Daniel Hausknecht, and Andrei Sabelfeld. The 32nd ACM SIGAPP Symposium On Applied Computing, 2017.

[10] A survey of man in the middle attacks
Mauro Conti, Nicola Dragoni, and Viktor Lesyk.. IEEE Communications Surveys & Tutorials, 18(3): 2027, 2016.

Questions