Improving Web Vulnerability Scanning

Daniel Zulla

Introduction

Hey!

- Hi there!
- I’m Dan. This is my first year at DEFCON.
- I do programming and security start-ups.
- I do some penetration testing as well

More Introduction

- Today I'm going to talk about vulnerability scanning
- Primary on the web
- "The cloud" is involved as well
- Network security too
- I'll show some things, so there is plenty of demo time
- Have fun, thanks for being here!

Some Facts

- There are a lot of web vulnerability scanners, fuzzers and penetration testing tools out there already
- Some of them work, some of them do not
- But basically all of them have one thing in common:
  They actually don't attack web applications on the application layer
- They mostly fuzz HTTP and sometimes perform injection attacks

Some more facts

- The fundamental design of web scanners has not changed in over a decade
- But: The web has changed.
- So there seems to be a problem.

Software Architecture

What web vulnerability scanners and fuzzers look like

A pentesters point of view

- Javascript/Ajax rich applications are still not supported
- Authenticated scanning is still incredibly challenging / not reliable
- Exploitation techniques are mostly poor
- “I don’t know which scanner will work for foo.com and which one for bar.com, so I use toolchains”

A developers point of view

- Javascript/Ajax rich applications are still not supported
- Authenticated scanning is still incredibly challenging / not reliable
- Exploitation techniques are mostly poor
- “I don’t know which scanner will work for foo.com and which one for bar.com, so I use toolchains”

- HTTP Libraries don’t support JS - Scanners are based on an HTTP Libraries
- Web Logins are not standarized - So how should they be detected
- No time for exploits
  (Already spent 100000 lines [and nights] of code making the crawler immune to encoding issues, malformed HTML, redirects and binary content!)
- A false positive is better than a false negative

How I see it

- Both of them are right.
- The web is a mess. Nobody cares about RFCs anymore. (Especially these SEO guys!)
- 10 years ago, you would have expected a Query String at the end of a URL like https://foo.com/xxx/yyy?foo=bar
- Nowadays, https://foo.com/something.ext/foo/bar is good practice
- The result: It's incredibly hard for scanner developers to figure out the dynamic components of an HTTP request. Because of that, we feel overhelmed and fuzz nearly everything.
- Header Keys, Header Values, VHost, Cookie, Method, Path, Version, ...

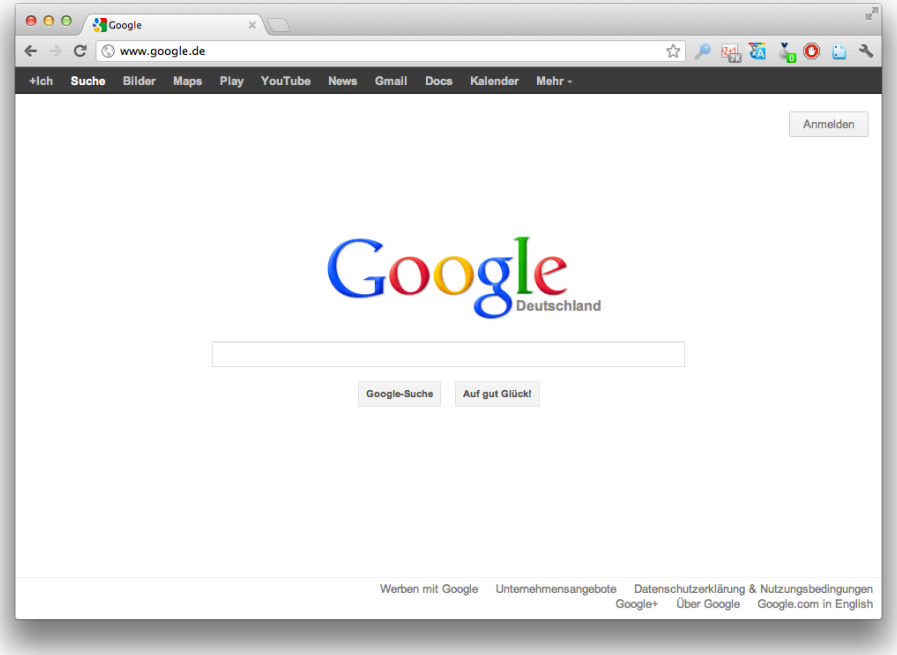
How I see it

- Fuzzing HTTP is incredibly important. You never know if you are talking to an apache2, nginx or some hidden application server upstream
- But it has nothing to-do with web vulnerability scanning
- So - developers are struggling with websites because they use HTTP to crawl and attack them. Things like flash, images, javascript seems to be an unsolveable problem
- Redirects are hard to handle sometimes (wait there is more)
- Javascript redirects (after 10 seconds!) and of course: onmouseover, onclick, onfocus, ...
- Flash isn't helpful either

Web 2.0

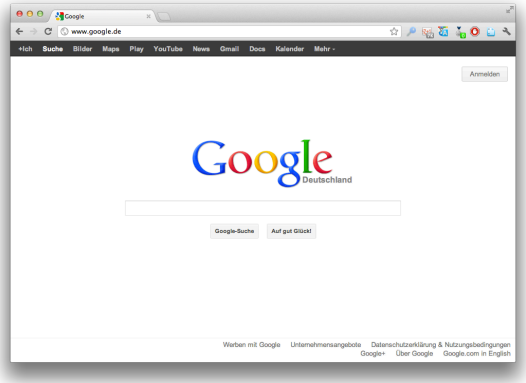
- But - WE DO SECURITY
- Is it really *our* job to make sure that our software executed all the JS and grabbed all the links?
- When we spend 100 hours on the crawler, and 5 hours on the actual payloads (that's how it looks right now) something, somewhere, went terribly wrong
- So - Is there a (open source?) piece of software that we could use instead of the HTTP library? Something that has prooven its mastery in handling unpredictably broken web content already? There is.

Webkit!


Google
www.google.de
+Ich Suche Bilder Maps Play YouTube News Gmail Docs Kalender Mehr
Anmelden
Google
Deutschland
Google-Suche
Auf gut Glück!
Werben mit Google Unternehmensangebote Datenschutzerklärung & Nutzungsbedingungen
Google+ Über Google Google.com in English

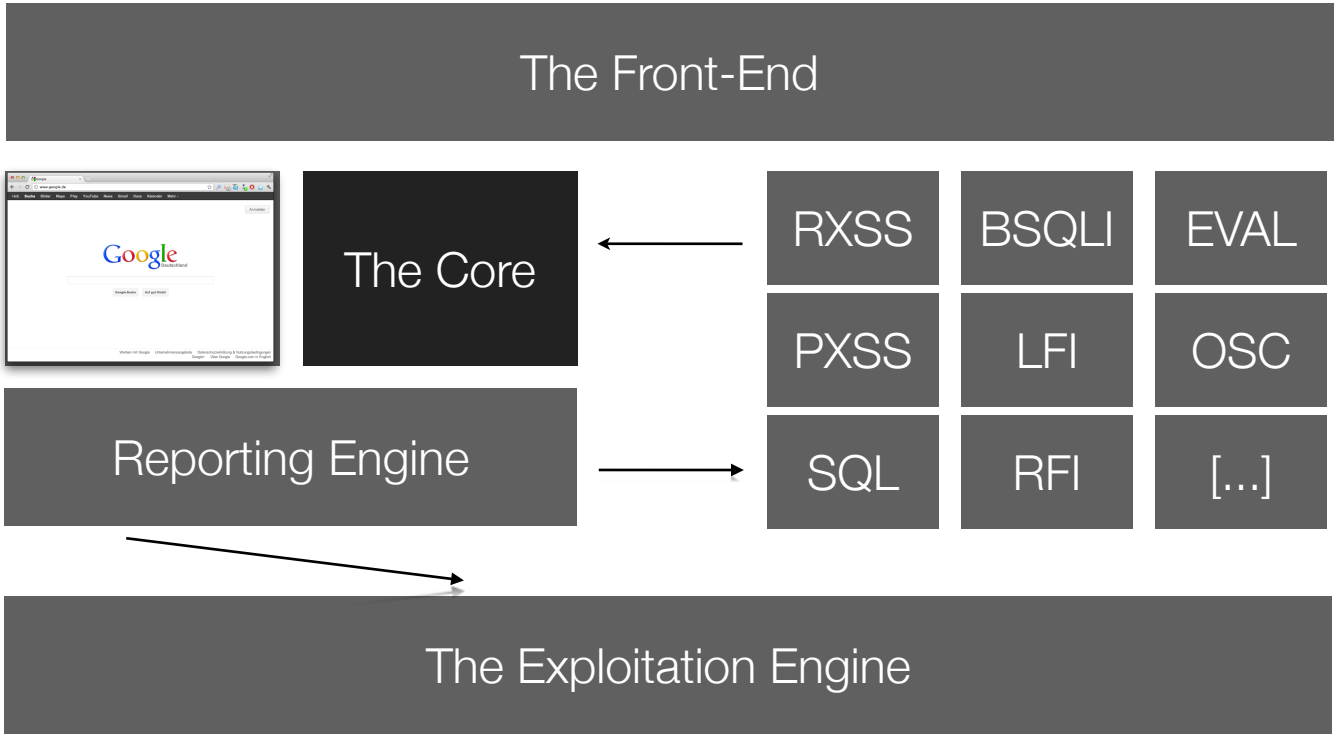
Webkit knows

- Javascript
- Javascript events
- Redirects
- Flash
- Images
- Websockets
- WebGL
- CSS Rendering
- Binary Downloads
- Broken HTML
- Broken CSS
- Performance
- Forking / Multiprocessing
- [...]



Software Architecture

What it should look like



Changes? Improvments?

- Replacing the HTTP library by a Webkit Engine
- Less code (A **lot** less code)
- 100% support for JS/Ajax/Broken HTML/JS Events/Crazy Redirects and all kinds of things
- The ability to simulate human user behaviour
- CSS Renderings (Two text fields beside each other: 10px - one of them is a input[type=password]) - May be a login!

Making it scale (heavily)

- Webkit is **slow** (Website rendering, Executing JS, ... - compared to - Speaking Plaintext HTTP)
- Downloading Images is slow
- Waiting for delayed JS events is slow
- Flash is even slower

Making it scale (heavily)

Bad news: Qt / PyQt / PySide

- QtWebkit does not support multithreading
- It tends to SEGFAULT from time to time :(
- Multiple QApplication instances are almost impossible to handle in one Python namespace

Making it scale (heavily)

Good news: Building a preforking TCP Server

- Spawning a pool of processes works quite well (one QApplication +one Browser instance per Process)
- Simultaneous downloads
- Better accessibility inside the scanner (multiprocessing insides loops to increase performance)

Missing pieces

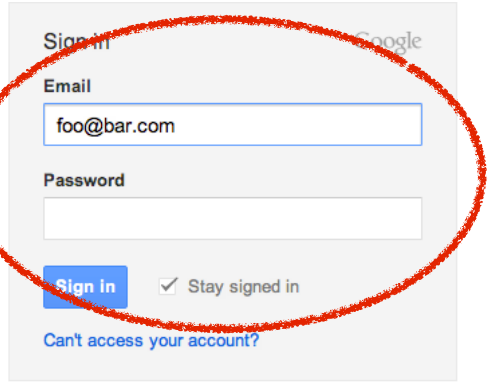
- Mastering Authentication
- Exploitation & Privilege Escalation
- Geographically distributed scanning: Using the cloud
- Reporting

Mastering Authentication

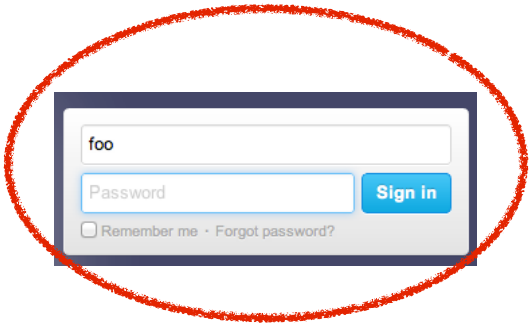
- There is no such thing as a standarized web login
- Basically, everybody develops access control on the web slightly differently
- You can try to detect them by the name/id of the attributes, but that is not reliable
- But in the end, Web logins generally have a few things in common that makes them easily detectable. At least, for our browser engine

Mastering Authentication

Not more than 2 visible (!) text fields

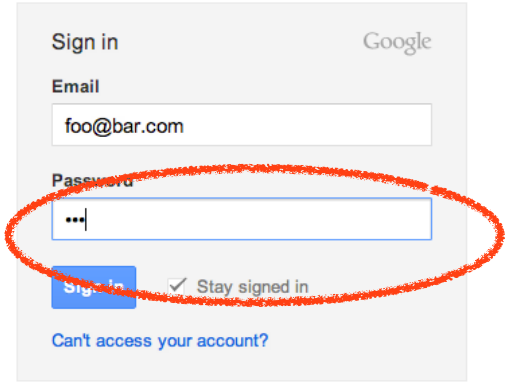


has_login_texts()

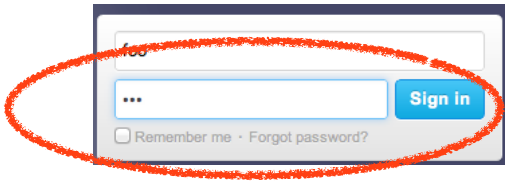


Mastering Authentication

Man-Behind-You Protection



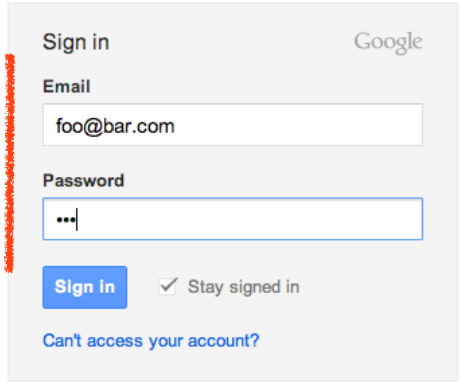
is_input_hidden()



Mastering Authentication

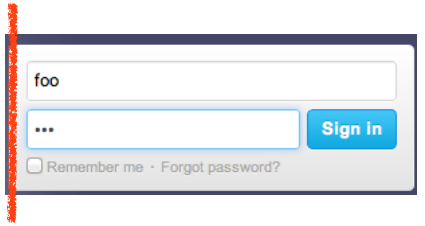
Geometry! Usually, the two visible text fields are under(), next_to() or at least near(radius=10px) each other

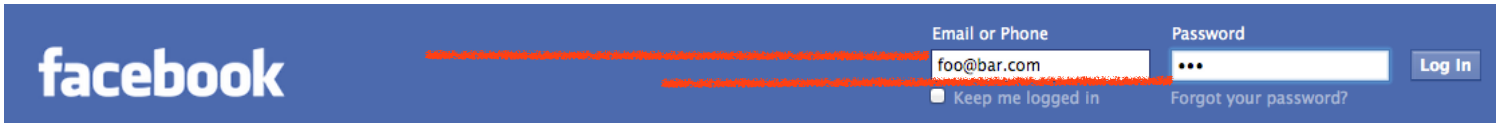
X1 = X2



!

X1 = X2





Y1 = Y2

Mastering Authentication

- That was easy!
- The common way to solve that problem, is to iterate through a wordlist (login, auth, signin, [...]) while checking the input[id], input[name] attributes
- That's not necessarily wrong or bad practice
- After putting the pieces together:
- .login("username", "password")

Mastering Authentication
Demo Time

- Proof Of Concept 1: Twitter (Some Javascript)
- Proof Of Concept 2: Facebook (More Javascript)
- Proof Of Concept 3: Google Plus (Most Javascript + Browser Hacks)

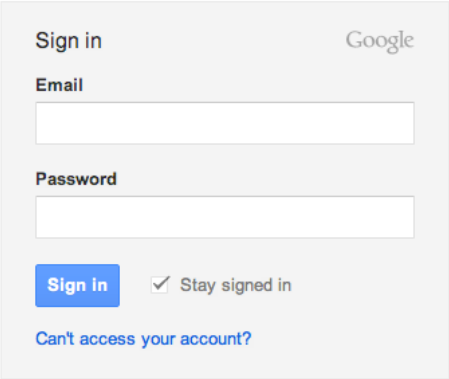
Mastering Authentication

When we are signed in

- New problems occur: How can we let the scanner check if we are indeed signed in?
- Common practive: Looking for a /logout/i String
- The problem: Inefficient. Likely to cause false positives
- There has to be a better way:
- Introduction “Strategies”

Strategy.Authentication
Step 1: Identification

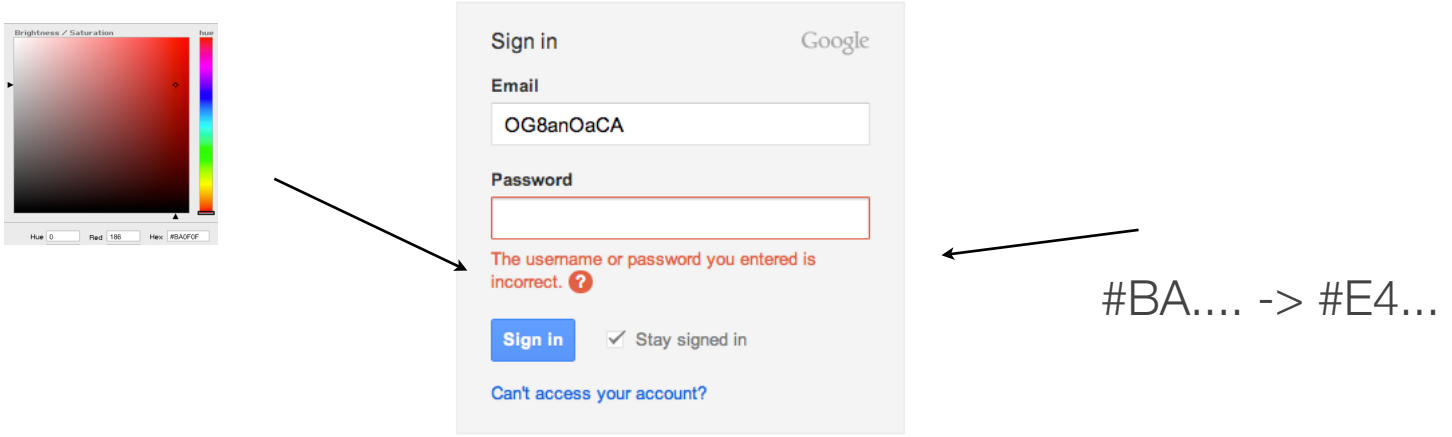
- Identifying a login form (3-way approach, input[type=password], geometry, [...])



Strategy.Authentication

Step 2: Error messages (Why a browser engines rocks)

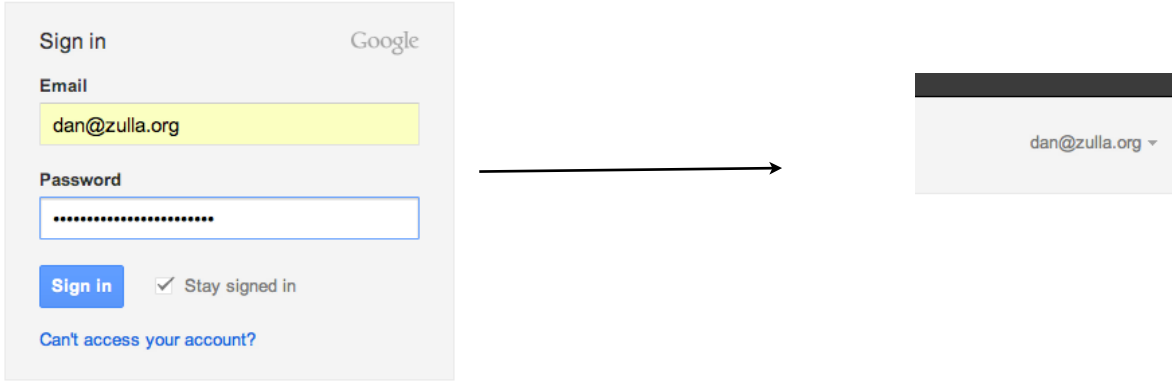
- Verifying **wrong** credentials - Random strings - Failed login



Strategy.Authentication

Step 3: Going in: .login(“..”, “..”)

- Verifying **valid** credentials - Behaviour should not be similiar to the behaviour of a **invalid** login



Strategy.Authentication

Step 4: Going out. .logout()

- Doing similiar work again for .logout() function seems obsolote
- But it really isn't.
- It is the basis to a .is_still_loggedin() function
- Which is really important to **stay** logged in during crawling
- And if the scanner logged itself out, it can simply .login() again
- That's cool. :-)

Exploitation and Privilege Escalation

- There is a whole universe besides injection vulnerabilities
- Usually, scanners don't detect them
- But they should
- And now they can: .login("user1", "..."); .logout(); .login("user2", "...")
- => Demo Time: Privilege Escalation, Multi-User Systems

Geographically distributed scanning: Using the cloud

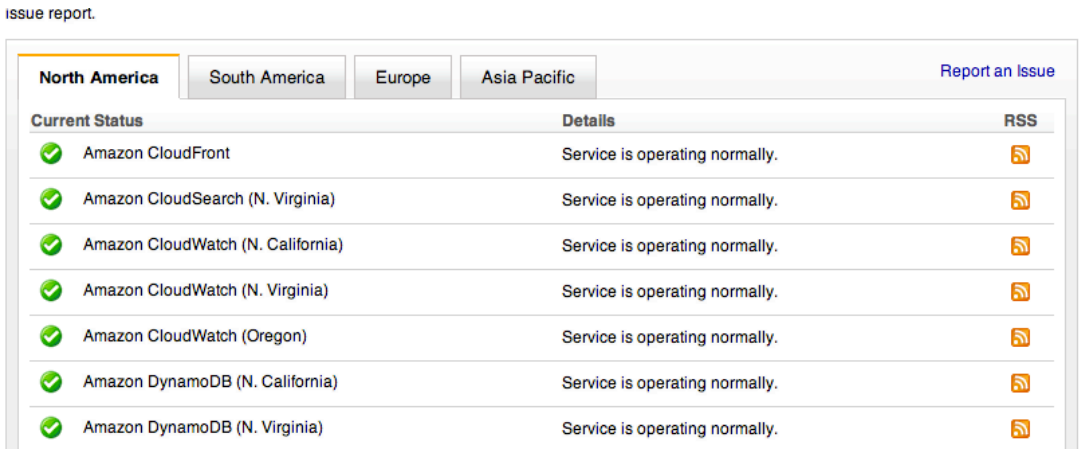
- When (injection) vulnerabilities are getting complicated:
- Scenario 1: The backend of a website creates a log entry for every new IP address. It logs the USERAGENT. The log entries are kept in a SQL database. The function that creates the log entries, is vulnerable. The User-Agent is injectable. The problem is:
- It only works once. As soon as the IP is in the database, the function won't be executed anymore :-(
- ==> SQLMap (and every other tool) will fail.

Geographically distributed scanning: Using the cloud

- But they shouldn't!
- The limitation is totally detectable
- And a new IP is just as far away as a single EC2 API call

Geographically distributed scanning: Using the cloud

- Indeed! The cloud is a good thing for security :)
- Demo Time: Introducing:
  sqlmap and w3af (on steroids)



Combining “Strategies” and the distributed scanning

- Introducing next generation vulnerability scanning
- Exploiting a really amazingly hard SQL Injection
- Demo Time

Further Research & Additional Ideas

- Country specific restrictions can be by-passed in a fully automatic manner
- (Error) messages can be parsed and interpreted: Wolfram Alpha
- Bloomfilters should be integrated
- Other “Strategies” should be implemented (the limitations are gone)

More Live Demos

- Demonstrating a logical layer beyond Authentication:
  .pay(“0000111122223333”, CVV=121, type=VISA)
  .search(“search query”)
  .sort(“DESC UNION SELECT [...]”)
- Interpreting error messages
- Pivoting on penetrated hosts - Spawning another scanner instance
- And finally: Reporting!

Thanks!