# Position in the line of attack



# Basic System-Architecture

Basically, 4 different basic system architectures were proposed and evaluated:

1. **Client Side Filters**
   1. **Local client service**

The system runs as a privileged Service on the client – this would allow a very efficient implementation but also makes the system a high-value target for attacks. End-to-end encryption would make it hard or even impossible to detect attacks in this kind of architecture

* 1. **Browser Extension**

This architecture requires a huge amount of development effort as we would have to develop and maintain a different system for each available browser. Yet, encryption would not have to be considered at all as this topic would be transparently handled by the browser.

1. **Middleware**
   1. **Forward Proxy**

Forward proxies are well supported on all major plattforms and generally implementing the system as a middleware allows central management and configuration of the system. Encryption would have to be passed by a man-in-the-middle-attack

* 1. **Suffix Proxy**

A suffix proxy is a special kind of proxy that allows the client to selectivly enabling the filter by using the Domain Name System. Instead of e.g. requesting the page untrusted.com the client should request untrusted.com.suffix.com. While encryption does not concern this kind of architecture, the selectivity of the filter might be a problem, especially when using it in a company environment.

Based on the analysis and evaluation of the possible architectures and on the evaluation of trends in the area of future web-technologies we concluded that:

* A local client service will not be feasible as in the future– especially with the upcoming http2.0 standard – encryption will be used much more frequently
* A browser extension will not be feasible as they impose a high development and maintenance overhead and are not available for many platforms (e.g. browsers on mobile devices)
* A forward proxy might be a well-suited solution, yet technology trends like „certificate pinning“ will make it hard to access encrypted communication with a man-in-the-middle-attack.
* A suffix proxy is an almost perfect solution, but the ability to enable and disable the filter might very likely contradict company security policies.

Considering these conclusions, we propose the following basic architecture, that combines the advantages of a forward- and a suffix-proxy:



**CLIENT**

can be any browser or application on any device, that allows the user to provide custom proxy settings.

**PROXY**

A simple proxy server that redirects all requests to the suffix-proxy. This proxy server will also have to perform user authentication.

**SUFFIX-PROXY**

The suffix-proxy will contain the major part of our application:

* It will receive the request from the client
* Forward the request to the web-resource and:
  + has to be able to handle different protocols (http(s), spdy, quic, WebRTC, push-technologies (WebSocket, SSE),… -> see Paper „Technologies for Web and Cloud Service Interaction“ by Harald Lampesberger)
  + has to be able to handle sessions,…
* Receive the response from the web-resource and:
  + Inspect the web-resource for attack-vectors
    - Detect the content-type of the resource
    - Perform various detection-mechanisms depending on the content-type
  + Perform url-rewriting
  + Either forward the response to the client if it is considered to be secure or warn the client otherwise

A very simple communication between these components might look like shown in the following diagram:



# Technologies

As scalability will be of great importance for our application we will develop our application using the actor based system AKKA. This actor-framework allows to build applications that are almost indefinitely scalable.

Preferably we will use Scala as our programming language of choice, yet Java is still an alternative.

After superficially evaluating different Libraries, I think that the Servlet Container Jetty might offer the most sophisticated library for building an application that has to handle various different protocols and also supports future technologies like SPDY and QUIC. Yet an in depth analysis of Jetty and similar Servlet Containers and Libraries will have to be done.

# TODO:

## Evaluieren

ScalaTest

<http://square.github.io/okhttp/>

<https://httpd.apache.org/>

<http://www.eclipse.org/jetty/documentation/current/spdy.html>

<http://www.eclipse.org/jetty/documentation/current/spdy-configuring-proxy.html>

<http://www.eclipse.org/jetty/documentation/current/proxy-servlet.html>

<https://gist.github.com/jponge/1752767>

<http://de.slideshare.net/Codemotion/jetty-9-the-next-generation-servlet-container>

## Jetty + Scala

<http://developian.blogspot.co.at/2010/06/using-jetty-and-jersey-with-scala.html>

<http://blog.magpiebrain.com/2009/05/16/a-stub-webserver-for-scala-using-jetty/>

<https://devcenter.heroku.com/articles/scaling-out-with-scala-and-akka>

<http://www.eclipse.org/jetty/documentation/current/embedding-jetty.html>

Eclipse + SBT

http://jowisoftware.de/blog/archives/39-Quick-Guide-Setup-SBT-for-Eclipse-including-SBT-cygwin-Setup-Updated.html