
# Browser-History-Stealing
# Ein Angriff auf die Privatsphäre

Bachelorarbeit

zur Erlangung des akademischen Grades
Bachelor of Science

**HUMBOLDT-UNIVERSITÄT ZU BERLIN**
**MATHEMATISCH-NATURWISSENSCHAFTLICHE FAKULTÄT II**
**INSTITUT FÜR INFORMATIK**

eingereicht von: Rüdian, Sylvio
geboren am: 10.04.1991
in: Berlin

Gutachter(innen): Prof. Redlich
Prof. Fischer

eingereicht am: 01.07.2014 verteidigt am:

# Inhaltsverzeichnis

# Einführung

Der Wunsch nach Wahrung der Privatsphäre ist in den letzten Jahren zunehmend in den Fokus öffentlicher Diskussionen getreten. Da sich viele Bereiche des öffentlichen Lebens in die virtuellen Welten des Internets verlagern, nimmt die Bedeutung der Privatsphäre auch dort zu. Gestützt wird das Bewusstsein für die eigene Privatsphäre in Folge der NSA-Affäre. Es besteht der Wunsch nach Privatheit trotz der Nutzung einer Vielzahl von Onlinediensten.

Einerseits vertrauen Nutzer darauf, dass ihre Spuren, die sie im Internet hinterlassen, minimal sind und dass diese nur von autorisierten Personen genutzt werden können. Andererseits ist vielen Nutzern nicht bewusst, dass sie Spuren im Internet hinterlassen. Jede Webseite, die ein Nutzer aufruft und auch jede Suchanfrage, die ein Nutzer stellt, ermöglicht tiefe Einblicke in deren Persönlichkeit.

In dieser Arbeit wird ein zentraler Angriff auf die Privatsphäre detailliert behandelt. Aus verschiedenen technischen Perspektiven wird beschrieben, wie ein potentieller Angreifer an Informationen Dritter gelangen kann. Im Speziellen geht es darum zu erfahren, welche Webseiten ein Nutzer in der Vergangenheit aufgerufen hat. Als Basis hierfür dienen zwei Techniken, die von allen gängigen Browsern seit Jahren unterstützt werden. Zum einen wird mit jedem Aufruf einer Webseite ein Eintrag in einer History erstellt. Diese Einträge können indirekt ausgelesen werden. Zum anderen werden Zwischenspeicher betrachtet, welche temporär Ressourcen speichern, um ein mehrfaches Empfangen identischer Daten zu vermeiden. Auch diese können indirekt ausgelesen werden, um zu erfahren, welche Webseiten in der Vergangenheit aufgerufen wurden.

Einige der vorgestellten Angriffe sind mit heutigen Browsern nicht mehr durchführbar, da die vorhandenen Sicherheitslücken weitgehend geschlossen wurden. Andere Angriffe sind aufgrund der Kombination verschiedener Technologien so komplex, dass diese Sicherheitslücken bis heute nicht geschlossen werden konnten. Ein effektiver Schutz ist bei komplexen Angriffspfaden häufig nicht möglich, solange auf die Verwendung der gebotenen Features nicht

verzichtet wird. Eine Auswertung der Nutzung der Sicherheitslücken in der Zeit, bevor diese geschlossen wurden, wird zeigen, wie intensiv Firmen an Informationen über bereits besuchte Webseiten interessiert sind.

Die gesamte Arbeit behandelt demnach unterschiedliche Angriffspfade, die gegenwärtig bekannt sind, um zu erfahren, welche Webseiten ein Nutzer zuvor besucht hat. In jedem der vorgestellten Verfahren muss ein Nutzer eine präparierte Webseite öffnen. Wurde diese aufgerufen, können die Informationen entweder automatisch oder durch Nutzereingaben gesammelt werden. Muss ein Nutzer mit einer Webseite interagieren, sind die zu bedienenden Elemente so konstruiert, dass keinerlei Auffälligkeiten existieren, welche den Nutzer von einer Eingabe abhalten könnten.

Zusätzlich zur Beschreibung der möglichen Angriffspfade wird die Effizienz einzelner Angriffe diskutiert und ausgewertet.

# Grundlagen

In den vergangenen zwei Jahrzehnten haben sich Browser zu einem standardisierten Programm entwickelt, welches die Darstellung und Navigation durch Webseiten ermöglicht. Diese sind nahezu auf jedem privat oder gewerblich genutzten Endgerät verfügbar und werden zahlreich zum Surfen durch das Internet genutzt.

Die Kombination von grundlegenden Techniken wie HTML und HTTP und die Nutzung von URLs ermöglicht die Entwicklung moderner Webseiten mit zahlreichen Gestaltungsoptionen durch CSS.

"HTML stands for HyperText Markup Language" (Bellis, 2014). Die Darstellung von Webseiten wird durch HTML in ihrem Grundgerüst beschrieben. Somit ist eine eindeutige Definition der Struktur und des Layouts von Webseiten möglich. Basis für diese Beschreibung sind HTML-Tags, welche Elemente wie Überschriften oder Absätze verschachtelt enthalten können.

"The Hypertext Transfer Protocol (HTTP) is an application-level protocol for distributed [...] systems" (R. Fielding J. G.-L., 1999). Das Netzwerkprotokoll ermöglicht die Übertragung von "Hypertext", also von Webseiten, dessen inhaltlicher Aufbau durch HTML beschrieben wird.

Allgemein werden Verweise durch URLs definiert. In der Verwendung von URLs in Browsern kennzeichnen URLs eindeutige Ressourcen wie Webseiten, Dokumente oder Bilder. Ein Verweis beschreibt die Verknüpfung unterschiedlicher Ressourcen, durch welche ein Nutzer navigieren kann.

Verweise haben in HTML einen "einheitlichen Aufbau" (SELFHTML, Verweise definieren und gestalten, 2005) und werden mit dem HTML-Tag `<a>` begonnen. "Damit [...] ein Verweis aus diesem Element wird, ist das Attribut `href` erforderlich (*href = hyper reference = Hypertext-Referenz*)" (SELFHTML, Verweise definieren und gestalten, 2005). Diesem Attribut wird das gewünschte Verweisziel zugeordnet. Abgeschlossen wird der definierte Verweis mit dem HTML-Tag `</a>`, wobei zwischen dem öffnenden und

schließenden HTML-Tag der für den Nutzer sichtbare Text steht, der auf eine URL verweist. Der Text kann auch durch Bilder ersetzt werden. Im folgenden wird als Link stets das Objekt bezeichnet, das für den Nutzer auf einer Webseite als Text oder Bild mit einem Verweis zu einer URL dargestellt wird.

Das äußere Erscheinungsbild von Webseiten wird durch Cascading Style Sheets (CSS) seit 1990 erweitert (Zachary Weinberg, 2011). "Cascading Style Sheets (CSS) is a stylesheet language used to describe the presentation of a document written in a markup language, such as HTML" (Markus Jakobsson, 2007). CSS ermöglicht die Verwendung von Pseudoklassen, die unter anderem für Links verwendet werden können. "[Sowohl] das Erscheinungsbild von Verweisen [(Links)] zu noch nicht besuchten Seiten [und] zu bereits besuchten Seiten [kann bestimmt werden]"(SELFHTML, Pseudoelemente und Pseudoklassen, 2001). Die Pseudoklasse ":link" beschreibt allgemein den Stil zur Darstellung von Links. "`:visited`" beschreibt die Darstellung bereits besuchter Links und hat eine höhere Wertigkeit als die Darstellung von nicht besuchten Links. Wurde eine URL, auf welche ein Link verweist, bereits besucht, wird der Link zunächst mit den Stil-Eigenschaften der Pseudoklasse "`:link`" dargestellt, ergänzt um die Stil-Eigenschaften der Pseudoklasse "`:visited`", wobei bei sich überschneidenden Eigenschaften ausschließlich die Eigenschaftenwerte der Pseudoklasse "`:visited`" verwendet werden.

Somit ergibt sich eine Kontrolle über alle Elemente einer Webseite, einschließlich darüber wie besuchte und nicht besuchte Links gerendert und für den Nutzer dargestellt werden (Zachary Weinberg, 2011).

# History-Stealing

Der Begriff des History-Stealing, auch Browser- oder Chronik-Sniffing genannt, wird grundlegend durch zwei Konzepte unterschieden. Zum einen werden Tracking-Cookies verwendet, um detailliert zu analysieren, welche Webseiten ein Nutzer wann und wie lange besucht hat. Die Tracking-Cookies werden in eine Vielzahl von Webseiten integriert und von einer zentralen Stelle ausgewertet, um das Nutzerverhalten zu extrahieren.

Zum anderen beschreibt das History-Stealing ein Verfahren, welches durch die Verwendung von Seitenkanälen einen indirekten Zugriff auf die Browser-History oder auf den Cache ermöglicht. Somit kann ein Angreifer Kenntnisse darüber erhalten, welche Webseiten ein Nutzer zuvor besucht hat, ohne dass die Verwendung von Tracking-Cookies notwendig ist. Innerhalb dieser Arbeit beschäftige ich mich mit dem zweiten Verfahren.

Für einen potentiellen Angreifer ergeben sich aus dem Zugriff auf die Browser-History zahlreiche Möglichkeiten, Interessen von Nutzern detailliert auszuwerten. Somit können ohne die Verwendung von Tracking-Cookies Profile von Nutzern erzeugt werden, die gezielt in der Werbeindustrie zur Personalisierung von Werbung zur Anwendung kommen können. Da Nutzer den indirekten Zugriff auf ihre Browser-History nicht ohne Einschränkungen deaktivieren können, öffnen sich neue Wege für Angreifer, wodurch sensible private Daten zur Nutzung von Webseiten abgefragt werden können.

# Das Angreifermodell

Die Anwendung des History-Stealing ermöglicht es einer präparierten Webseite manuelle oder automatisierte Anfragen zu stellen, ob ein Link bereits besucht wurde oder nicht. "Es gibt keinen bekannten Weg, einen direkten Zugang zur vollständigen Browser-History zu erhalten" (Zachary Weinberg, 2011).

Es gibt bereits zahlreiche Schutzmechanismen, welche das History-Stealing unterbinden. Dennoch treten immer wieder neue Seitenkanäle hervor, mit deren Hilfe Wissen über zuvor besuchte Webseiten erlangt werden kann. Diese sind das zentrale Thema dieser Arbeit.

Ein Seitenkanal tritt genau dann auf, wenn ein System oder eine gezielte Kombination von Komponenten Informationen preisgeben, welche nicht zur ursprünglichen Funktion des Systems bzw. zu den einzelnen Komponenten gehören. In der Regel sind solche Seitenkanäle durch die Komplexität von Software nur schwer zu finden.

Seit den ersten Versionen werden in Internetbrowsern besuchte und nicht besuchte Links farblich unterschieden. Bereits einer der ersten grafischen Internetbrowser, NCSA Mosaic, stellt nicht besuchte Links blau und wenn diese in der Vergangenheit besucht wurden, violett dar (Nielsen, 1995). Dieses Konzept wurde bis heute beibehalten.

Das Angreifermodell besteht darin, entweder Seitenkanäle des Konzepts zur farblichen Unterscheidung von Links auszunutzen oder Seitenkanäle anzuwenden, die auf zeitlichen Messungen während der Generierung oder dem Laden von Webseiten basieren.

# Automatisierte Angriffe auf Stilelemente

Mit der Erfindung von clientseitigen Sprachen wie Javascript wurde es möglich, Programme direkt im Browser des Nutzers auszuführen. Prinzipiell sollte es dabei nicht möglich sein, automatisch zu erkennen, welche dritten Webseiten ein Nutzer besucht hat, da dies durch die "same-origin policy" unterbunden werden sollte (Nielsen, 1995). Demnach kann ein Javascript-Programm nur Daten des Servers sehen, von welchem es geladen wurde. Kombiniert man das allgemein statische CSS und Javascript, ergibt sich trotz der "same-origin policy" die Möglichkeit zu ermitteln, ob ein Nutzer eine Webseite zuvor besucht hat.

## Javascript - getComputedStyle()

Bis zum Jahr 2010 war es möglich, automatisierte Anfragen mithilfe von Javascript durchzuführen, um zu erfahren, ob eine Webseite bereits besucht wurde. Grundlage hierfür ist der erwähnte Mechanismus, dass bereits besuchte und nicht besuchte Links beim Rendern einer Webseite farblich unterschieden werden. "Unterstützt wird diese Methode durch 'Cascading Style Sheets' (CSS), welche zum Ende des Jahres 1990 zeitgleich wie die Sprache JavaScript entwickelt wurden" (Zachary Weinberg, 2011).

Das 'Document Object Model' (DOM) ermöglicht als Standard-API den Zugriff auf die aktuelle Webseite und bietet Funktionen, diese mithilfe der Sprache Javascript zu verändern (Zachary Weinberg, 2011). "The Document Object Model is an [...] interface that will allow programs and scripts to dynamically access and update the [...] style of documents" (Hégaret, 2005).

Nachdem das DOM im Browser erstellt wurde, erlaubt dies einen Zugriff auf den dargestellten Stil jedes HTML-Objektes durch ein Javascript-Programm. Der Javascript-Funktion "getComputedStyle" wird als Parameter das Element und die Pseudoklasse übergeben, für welche der zugewiesene Wert der Pseu-

doklasse zurückgegeben wird. Somit bestand bislang die Möglichkeit, auch auf den Stil von Links zuzugreifen.

Sowohl besuchte, als auch nicht besuchte Links einer Webseite können mithilfe von CSS mit einer unterschiedlichen Farbe versehen werden. Durch die Anwendung der Javascript-Funktion "getComputedStyle" war es möglich, diese farbliche Markierung auszugelesen. Damit ergibt sich ein Seitenkanal. Je nachdem, welchen Farbwert die Funktion als Rückgabewert liefert, wurde ein Link zuvor besucht oder nicht, da besuchte Links durch CSS einen anderen Farbwert zurückgeben als besuchte. Die Anwendung des Seitenkanals ist nur möglich, wenn in einer präparierten Webseite besuchte und nicht besuchte Links mit unterschiedlichen Farben durch CSS definiert wurden, da die ausgelesenen unterschiedlichen Farben indirekt der Erkennung von besuchten und nicht besuchten Links dienen. Je nach Leistung des Browsers können durch die Javascript-Funktion "je Minute bis zu 210.000 Links" (Blizzard, 2010) darauf überprüft werden, ob diese zuvor besucht wurden. Bekannt ist diese Technik seit 2002.

Um zu erfahren, ob diese Technik tatsächlich in gängigen Webseiten angewandt wird, wurde durch die Universität in Kalifornien im Februar 2010 eine empirische Studie erstellt. Nach dem Alexa-Ranking wurden die 50.000 am häufigsten besuchten Webseiten analysiert. "485 of them inspect style properties [...], 63 [of the 485] are reported as transferring the browser's history to the network [...]" (Dongseok Jang, 2010). Zur am häufigsten besuchten Webseite, welche erhaltene Informationen über die Browser-History über das Internet überträgt, zählt die Webseite von "youporn", die zum Zeitpunkt der Studie einen Alexa-Rank von 61 hatte. Auffallend häufig wurden neben URLs von Konkurrenten die URLs von sozialen Netzwerken wie Facebook und Twitter, zudem von Verkaufswebseiten wie Amazon und eBay, aber auch von dem in Deutschland weniger bekannten KFZ-Portal "Edmunds" darauf überprüft, ob diese bereits besucht wurden.

"Many of these websites seem to try to obfuscate what they are doing" (Dongseok Jang, 2010). Ein Teil der Webseiten hat zudem vorbereitete Bibliotheken verwendet, die speziell entwickelt wurden, das History-Stealing aktiv

durchzuführen. Eine Firma, die heutzutage weiterhin die sehr detaillierte Analyse von Nutzern durchführt, ist interclick.com, welche gegenwärtig zu Yahoo gehört und nach eigenen Werbeaussagen ein nahezu vollständiges Abbild der Verbraucher erstellt (Abbildung A1). Die Technik zur Durchführung von History-Stealing-Angriffen wurde also bewusst im großen Stil angewandt. Folglich gab es einen dringenden Handlungsbedarf, diese Technik zu unterbinden.

"It's a bug that's been discussed in developer circles for over a decade" ('Technocrat', 2010). Im Jahr 2010 wurde durch den Mozilla - Entwickler David Baron ein Vorschlag zur Entwicklung eines Schutzmechanismus eingeführt, welcher alle bekannten automatischen Anfragen, die auf der Unterscheidung der Farben basieren, unterbinden soll (Zachary Weinberg, 2011). "These fixes have since been implemented in all major browsers" (Stone, 2013). Sowohl in Firefox 4, Google Chrome 8, dem Internet Explorer und in Safari 5 wurde Barons Schutzmechanismus eingeführt ('Technocrat', 2010). Somit ist die direkte Abfrage, ob ein Link zuvor besucht wurde, durch die Javascript-Funktion "getComputedStyle" gegenwärtig nicht mehr möglich, da diese fortan identische Werte für Link-Objekte zurück gibt.

## Angriff durch Hintergrundbilder

Mit CSS können Hintergrundbilder in darzustellende Objekte geladen werden. Mit dieser grundlegenden Funktion ist es möglich, Webseiten bei der Entwicklung mithilfe von Hintergrundbildern zu verzieren. Dies erleichtert einem Entwickler die Einbindung von Hintergrundbildern zu Objekten, da keine manuelle Überlagerung eines Hintergrundbildes mit einem weiteren Objekt, welches direkt über dem Hintergrundbild positioniert wird, notwendig ist.

Prinzipiell kann jedem Objekt einer Webseite ein Hintergrundbild zugeordnet werden. Die Zuweisung eines Hintergrundbildes zu einem Objekt erfolgt über die Eigenschaft `"background-image"`, welcher als Attribut direkt eine Bild-URL zugewiesen wird.

Bei der Generierung der Webseite innerhalb des Browsers wird jedem Link zunächst der Stil der CSS-Pseudoklasse `a:link` zugewiesen. Als Stil kann

ebenso ein Hintergrundbild übergeben werden. Wurde die verweisende URL eines Links bereits besucht, wird dieser mit dem CSS-Stil der Pseudoklasse a:visited versehen und die Attributwerte der Pseudoklasse "link" werden überschrieben.

Wird der Eigenschaft "background-image" der Pseudoklasse "visited" eine Bild-URL zugewiesen, wird diese lediglich aufgerufen, wenn der Browser beim Generieren der Webseite das zu ladende Bild benötigt.

Eine aufzurufende Bild-URL kann so präpariert werden, dass beispielsweise eine 1x1-Pixel-Grafik zurückgegeben wird. Zeitgleich kann eine Protokollierung des Aufrufes durch eine serverseitige Sprache wie php erfolgen. "You can detect that the user has visited a page and [...] perform an action" ('id', 2007).

Wurde demnach durch den Nutzer ein Bild der Pseudoklasse "visited" aufgerufen, wusste der Angreifer eindeutig, ob die verweisende URL des präparierten Links vom Nutzer bereits aufgerufen wurde.

Bis zum Jahr 2010 bestand die Möglichkeit, Hintergrundbilder der CSS-Pseudoklasse "visited" zu definieren. Der vorgestellte Seitenkanal ist ohne Nutzung von Javascript möglich. Dies zeigt, dass auch Nutzer, die Programme wie NoScript zum Blockieren von Skripten in Ihren Browsern bewusst nutzen, nicht geschützt werden konnten. Lediglich Nutzer, die auf die Anzeige von Bildern verzichtet haben, waren vor diesem Angriff geschützt.

## Redraw-Events

Ein Browser muss entscheiden, ob ein Link mit dem mittels CSS definierten Stil als besucht oder nicht besucht dargestellt wird. "Every browser has a database of previously visited urls" (Stone, 2013). Hierzu wird beim Öffnen einer Webseite bei jedem Link überprüft, ob dieser zuvor besucht wurde, indem eine Abfrage an eine browserinterne Datenbank gestellt wird, in welcher die bereits besuchten Webseiten aufgelistet sind. Diese Datenbank wird durch das Aufrufen von Webseiten, die noch nicht besucht wurden, kontinuierlich durch neue Einträge erweitert.

Die Anfrage an die Datenbank erfolgt in den Browsern Firefox und dem Internet Explorer asynchron. Das bedeutet, dass ein Link auf dem Bildschirm zunächst als nicht besucht dargestellt wird. Wenn als Antwort der Datenbank eintrifft, dass eine angeforderte URL in der Datenbank existiert und diese demnach in der Vergangenheit bereits aufgerufen wurde, dann wird der CSS-Stil für besuchte Links beim nächsten Generieren der Webseite, im folgenden Frame genannt, angewandt und für den Nutzer sichtbar. "If this redraw can be deteced by a webpage, then it could tell if the link had been previously visited" (Stone, 2013).

Andere Browser wie Chrome führen eine synchrone Anfrage an die Datenbank durch und warten die Antwort der Datenbank ab, bis die gesamte Webseite generiert wird.

Wird mittels Javascript ein neues Ziel für ein Link-Element festgelegt, wird genau wie beim initialen Aufrufen einer Webseite eine Anfrage an die Datenbank erstellt und der CSS-Stil wird je nach Antwort für besuchte und nicht besuchte Links angewandt. Versuche von Paul Stone haben ergeben, dass das dynamische Zuweisen von neuen Link-Zielen sowohl in Firefox, als auch in Chrome zu einem Redraw führt, also zum neuen Generieren der Webseite auf dem Bildschirm.

Es gibt also die Möglichkeit, einen Redraw sowohl in Firefox, als auch in Chrome zu erzwingen, nämlich genau dann, wenn die Datenbankabfrage, die überprüft, ob ein Link bereits besucht wurde, eine positive Antwort zurück gibt.

In der Vergangenheit konnte Firefox Redraws direkt erkennen und mit dem Aufruf der Funktion "mozAfterPaint" verarbeiten. Eingeführt wurde diese Erweiterung, um Benchmarks direkt im Browser durchführen zu können. Ein potentieller Angreifer konnte demnach einen Link erstellen und die verweisende URL beliebig oft durch Javascript dynamisch verändern. Die Funktion "mozAfterPaint" wurde immer genau dann aufgerufen, wenn der Browser den Redraw einer Webseite abgeschlossen hatte. Mit einem einfachen Zähler konnte der Angreifer ermitteln, wie oft nach der dynamischen Veränderung

der Links ein Redraw stattgefunden hat und konnte schlussfolgern, ob die Webseite, auf welche die URL des Links verwiesen hat, zuvor besucht wurde. Aufgrund des sich ergebenden Seitenkanals, der das History-Stealing ermöglichte, wurde diese Funktion in Firefox ab der Beta-Version 10 deaktiviert (Stone, 2013).

## RequestAnimationFrame

Eine Neuerung des HTML5 ist die Möglichkeit, innerhalb eines Browsers flüssige Animationen zu erzeugen. Grundlage ist hierzu in Javascript die Funktion `"requestAnimationFrame"`, welcher eine Funktion als Parameter übergeben wird, die genau dann aufgerufen wird, bevor das Generieren des nächsten Frames auf dem Bildschirm erfolgt. Dieser Mechanismus erlaubt Entwicklern die Kontrolle über jeden Frame, der auf dem Bildschirm dargestellt wird. "These include executing JavaScript code, calculating the position of new and updates elements [...]" (Stone, 2013). Je nach Komplexität des auszuführenden Programmes wird eine unterschiedliche Zeit benötigt, bis die Funktion vollständig abgearbeitet ist und ein neuer Frame auf dem Bildschirm generiert werden kann. Wird die Funktion `"requestAnimationFrame"` wiederholt aufgerufen, können bis zu 60 Frames je Sekunde auf dem Bildschirm generiert werden, also ein Frame je 16ms (Stone, 2013).

Das Generieren von Frames erfolgt in drei Schritten. Zunächst wird das Javascript-Programm ausgeführt, welches der Funktion `"requestAnimationFrame"` als Parameter übergeben wurde. Danach werden die einzelnen Elemente mit ihrer definierten Größe und Position justiert und anschließend mit Inhalten wie Texten und Farben gefüllt und auf dem Bildschirm als Frame ausgegeben. Die Funktion `"requestAnimationFrame"` wird durch die als Parameter übergebene Funktion rekursiv wiederum aufgerufen, damit diese beim Generieren des folgenden Frames erneut zur Verfügung steht. Dauert die Ausführung der drei Schritte länger als 16ms, wird die Generierung des folgenden Frames verzögert. "This delay will be measurable by requestAnimationFrame [...]" (Stone, 2013). Demnach kann die Framerate gemessen und kontrolliert werden. "Mo-

dern computers' clocks provide enough precision [...]" (Zachary Weinberg, 2011). Grundlage für die Kontrollierbarkeit ist das Einfügen von komplexen Stilelementen wie Schatten oder Transparenzen, die zu einer bewussten Verzögerung führen, bis ein neuer Frame generiert wird.

Bei einer asynchronen Abfrage an die Datenbank wie unter Firefox oder dem Internet Explorer können bereits besuchte Webseiten erkannt werden, indem zunächst eine Webseite mit zahlreichen Links, welche alle auf dieselbe URL verweisen, erstellt wird. Alle Links erhalten zudem einen Schatten, wodurch die Generierung von Frames verzögert wird. Mithilfe der Funktion `"requestAnimationFrame"` werden nun die Zeiten der folgenden Frames ermittelt, um herauszufinden, ob ein Redraw erfolgt.

Das beschriebene Verfahren wurde im Versuch 1 im Browser Firefox demonstriert. In diesem Experiment werden als Basis 20 Links generiert, deren URLs identisch sind. Zum Beginn des Experiments verweisen die Links auf eine nicht besuchte URL, zum Beispiel auf eine URL, die nicht existiert. Die Generierung des ersten Frames dauert bekanntermaßen bei allen Links relativ lange, da die Generierung von mittels CSS definierten komplexen Schatten viel Zeit in Anspruch nimmt. Daher beginnt erst nach dem Setup eine Zeitmessung mithilfe der Funktion `"requestAnimationFrame"`, welche die benötigte Zeit zur Generierung des letzten Frames ermittelt und ausgibt. Nun wird die URL jedes Links neu gesetzt, nämlich genau so, dass nach der Veränderung alle Links auf dieselbe zu testende URL verweisen. Wurde die zu testende URL zuvor aufgerufen, findet ein Redraw statt, da die vorbereiteten URLs zuvor nicht besucht wurden und die Links in einem Frame neu generiert werden müssen. Ein Screenshot des Versuchs ist in Abbildung A2 zu finden. Umgekehrt findet kein Redraw statt, wenn die zu testende URL zuvor nicht aufgerufen wurde, da keine sichtbare Veränderung stattfindet.

Da die Links durch CSS mit komplexen Schatten versehen wurden, führt die Generierung von Links in einem neuen Frame zu einer zeitlich messbaren Verzögerung. Im Browser Firefox wurde in dem Versuch ein Frame je 16ms generiert. Bei einem Redraw entsteht eine messbare Verzögerung von ca. 35 bis 70ms.

Das Verfahren wird durch das Programm automatisch zwanzigfach durchgeführt, um Messfehler zu korrigieren. Bei einem Versuch habe ich das Experiment zwölffach sowohl mit einer nicht besuchten URL, als auch mit einer besuchten URL durchgeführt. Die erhaltenen Daten sind in der Abbildung A3 sichtbar. Trotz ermittelten Abweichungen während der Messungen wurde eine Treffergenauigkeit von 100% erreicht, mit einer Fehlerrate von 0%.

Das Experiment wurde auch von Paul Stone durchgeführt, der das beschriebene Konzept im Internet Explorer angewandt hat, wobei lediglich abweichende Zeitwerte bei der Generierung von Frames erzeugt werden. Ursache für die verschiedenen Zeitwerte ist unter anderem die Verwendung abweichender Hard- und Software mit unterschiedlicher Auslastung.

Da Chrome eine synchrone Anfrage an die Datenbank stellt, um zu ermitteln, ob die vorhandenen Links bereits besucht wurden und die gesamte Webseite im Browser erst generiert, wenn alle Informationen über bereits besuchte Links vorhanden sind, können keine Redraws beim initialen Generieren einer Webseite erkannt werden. Dennoch kann auch beim Browser Chrome eine Technik angewandt werden, um Redraws zu erkennen. Wird eine Vielzahl von Links generiert und werden diese dynamisch während der Laufzeit durch Javascript zu einer zu testenden URL verändert, erfolgt eine weitere synchrone Abfrage an die Datenbank. Ein Redraw erfolgt dann direkt im folgenden Frame, da die Webseite auf die vollständige Antwort der Datenbank wartet. Da der Redraw auch hier durch die Anwendung der Schatten eine relativ lange Zeit benötigt, kann dieser erkannt werden. "A simple test of 1000 URLs with 10 visited links completed in 17 seconds, a speed of 58 URLs tested per second" (Stone, 2013).

Jeder zeitbasierte Angriff ist von der Genauigkeit der Messungen abhängig. Andere Programme, parallel ausgeführte Anwendungen oder die Verwendung unterschiedlicher Browser können das Ergebnis verfälschen, da diese auch zur Verzögerung der Generierung von Frames führen können. Auf der Basis der zeitlichen Messung aufeinander folgender `"requestAnimationFrames"` kann ein potentieller Angreifer sogar auf die Auslastung des Betriebssystems schließen. Während einer Kalibrierungsphase kann ein Angreifer solange war-

ten, bis sich eine relativ konstante Auslastung des Betriebssystems eingestellt hat. Dann kann der Angriff mit einer geringen Fehlerrate bei der Messung durchgeführt werden. Es bestehen demnach Möglichkeiten, Seitenkanäle, die auf zeitlichen Messungen basieren, auszunutzen, um zeitliche Verzögerungen bei der Generierung von Frames zu ermitteln, wodurch das History-Stealing erneut möglich wird.

## Schutz vor automatisierten Angriffen

David Baron hat zum Schutz der Privatsphäre zwei Klassen von Techniken eingeführt, die das Auslesen, ob eine Webseite besucht wurde, unterbinden sollen. Zum einen werden für den Stil von Links alle Techniken unterbunden, welche besuchte Links in ihrer Größe oder Position verändern. Somit wird vermieden, dass weitere Seitenkanäle entstehen, die wiederum ein Auslesen ermöglichen. Zum andern wird das Laden verschiedener Hintergrundbilder unterbunden. Eigenschaften wie Transparenzen, Schatten oder Farbverläufe können bei besuchten Links weder hinzugefügt, noch entfernt werden. Zusammenfassend werden demnach nicht besuchte Links genau so dargestellt wie besuchte Links, wobei lediglich die Textfarbe für bereits besuchte Links verändert werden kann. Daraus resultiert, dass direkte Angriffe über CSS-Style-Eigenschaften nicht mehr wie zuvor möglich sind. Zudem werden zusätzliche Frames im Browser so generiert, dass das Rendern der besuchten und nicht besuchten Links die gleiche Zeit benötigt, was einen zeitbasierten Angriff unmöglich machen soll (Zachary Weinberg, 2011).

Links können jedoch weiterhin mit komplexen Stil-Elementen wie Schatten versehen werden, was bei einem Redraw trotz der eingefügten Frames zu messbaren Verzögerungen führt. Die Browser haben gar nicht die Möglichkeit, bei einer asynchronen Datenbankabfrage die Zeit zum Generieren der Frames vorausschauend zu verzögern. Zeitliche Angriffe sind also trotz der Generierung zusätzlicher Frames im Browser weiterhin durchführbar, solange ein Redraw stattfindet. Die von David Baron eingeführte Technik, welche lediglich besuchte Links betrachtet, ist nicht ausreichend, um einen wirkungsvollen Schutz gegen History-Stealing-Angriffe zu bieten.

# Interaktive Angriffe

Das Wissen darüber, welche Webseiten ein Nutzer besucht hat, kann auch direkt durch den Nutzer geliefert werden. Dabei wird ein Nutzer hintergangen, indem er mit für ihn sichtbaren Links interagiert. Bei dieser Art von Angriffen kann lediglich eine begrenzte Anzahl von Links getestet werden, dennoch ist dieser Angriff möglich. "Auch wenn interaktive Techniken langsamer sind [...], gibt es keinen praktikablen Weg, sich davor zu schützen" (Zachary Weinberg, 2011). Ein Nutzer muss dabei die Bereitschaft haben, mit einer präparierten Webseite zu interagieren, andernfalls ist ein interaktiver Angriff nicht möglich. Die Besonderheit eines interaktiven Angriffs besteht darin, lediglich die Veränderung der Farben von besuchten Links zu verwenden.

Ein gewöhnlicher Test zum Erkennen, ob es sich bei einem Besucher um einen Menschen handelt, Captcha[1] genannt, genügt der Anforderung, nicht als ungewöhnlich erkannt zu werden. Das Captcha ist ein Test, beim welchem dem Nutzer eine zufällige Folge von Wörtern oder Zeichen, meist verzerrt, in einem Bild dargestellt wird. Aufgabe ist, die dargestellten Wörter oder Zeichen manuell in ein Eingabefeld zu übertragen (Zachary Weinberg, 2011). Oftmals kann diese Aufgabe ohne große Aufwände durch einen Menschen erledigt werden, für Software ist diese Aufgabe jedoch sehr komplex. Somit kann automatisiert ermittelt werden, ob ein Nutzer tatsächlich ein Mensch ist, vorausgesetzt, dass dieser keine Spezialprogramme zum bildbasierten Lösen der Captchas besitzt.

Ein auf Webseiten eingebundenes Captcha gilt als gewohntes Element und erzeugt zudem keinen intuitiven Gedanken an potentielle Angriffe.

Der interaktive Angriff wird so durchgeführt, dass auf der Webseite besuchte und nicht besuchte Links unterschiedlich eingefärbt werden, wobei beispielsweise besuchte Links dieselbe Farbe erhalten wie der Hintergrund. Demnach sind diese Links für den Nutzer nicht mehr direkt sichtbar.

---

[1] Captcha ist die Abkürzung für 'Completely Automated Public Turing test to tell Computers and Humans Apart'.

## Wort-Captchas

Ein Nutzer kann aufgefordert werden, sichtbare Wörter eines Captchas einzugeben. Einem Link wird hierbei genau ein Wort zugeordnet. Alle Wörter werden in dem sogenannten Pseudo-Captcha dargestellt. Je nachdem, welche Wörter für ihn sichtbar sind, kann anhand der Auswertung ermittelt werden, welche Links bereits besucht wurden. Diese Angriffsart ermöglicht das Testen weniger Links, da die Bereitschaft, das Pseudo-Captcha zu lösen, mit zunehmender Wortanzahl sinkt.

Die Eigenschaft, dass der Text eines Captchas verzerrt ist, kann in dem beschriebenen Pseudo-Captcha durch die Anwendung von SVG-Filtern umgesetzt werden. Weiterhin können für die einzelnen Wörter unterschiedliche Schriftarten verwendet werden, wodurch ein Pseudo-Captcha entsteht, welches dem Google-Captcha sehr ähnlich ist (Abbildung A4).

Da die Möglichkeit besteht, dass ein Nutzer alle URLs, auf welche die Links verweisen, zuvor besucht hat, würde das bisher genannte Pseudo-Captcha eine farblich monotone Fläche generieren, die keine Wörter enthält. Daher muss mindestens ein Wort sichtbar sein, welches keinen Verweis auf eine URL enthält und immer sichtbar ist. Dieses Wort könnte zudem in dem Pseudo-Captcha der ursprünglichen Aufgabe dienen, zu erkennen, ob es sich um eine automatisierte Anfrage handelt. Dennoch reduziert dieses Wort die Anzahl der zu testenden Links.

Während der Durchführung des interaktiven Angriffes wird angenommen, dass ein Nutzer die dargestellten Wörter nicht als Links erkennt, da andernfalls der Nutzer skeptisch werden könnte.

Bewegt ein Nutzer zufällig die Maus über einen nicht sichtbaren Link, wird das Ziel des Links zum Beispiel in der Statusleiste des Browsers angezeigt, auch innerhalb des Pseudo-Captchas. Eine einfache Lösung des Problems ist, eine transparente Ebene über alle Links des Pseudo-Captchas zu legen, wodurch das Bewegen der Maus über einen Link zu keiner Interaktion und somit nicht zur Anzeige des Links in der Statusleiste führt.

Eine weitere Möglichkeit zur Erkennung der Links ist die Verwendung der Tabulator-Taste. Diese bietet die Funktion, nacheinander alle vorhandenen Links einer Webseite zu fokussieren. Mit dieser Funktion können also auch die Links des Captchas fokussiert und somit als Links erkannt werden. Durch die Verwendung von Javascript kann ein Erkennen der Links bei eingeschaltetem Javascript weitgehend vermieden werden, indem die Funktion der Tabulator-Taste durch Javascript überschrieben wird.

Die Verwendung eines Pseudo-Captchas ist praktisch möglich. Bei dem Test ganzer Wörter können jedoch nur wenige Links darauf überprüft werden, ob diese zuvor besucht wurden, da ein Link genau durch ein Wort repräsentiert wird, wobei ein zusätzliches Wort integriert werden muss, welches als einziges sichtbar ist, wenn alle anderen mit Verweisen präparierten Links dieselbe Schrift- und Hintergrundfarbe aufweisen und folglich nicht sichtbar sind.

Sei n die Anzahl der zu testenden Links und seien O die Objekte, die jeweils genau einen Link repräsentieren. Dann folgt, dass jedes O in der vorherigen Ausführung genau einem Wort entspricht: $|O| = n \Leftrightarrow |\text{Wörter}| = n$, wobei die Notation $|x|$ die Anzahl von x darstellt. Seien m die Objekte, die dem Nutzer präsentiert werden. Da es möglich ist, dass ein Nutzer alle Links bereits besucht hat und demnach keiner der Links sichtbar ist, ist ein Prüfwort notwendig, welches mindestens  präsentiert werden muss, damit das Pseudo-Captcha nicht ohne sichtbare Inhalte generiert wird:

$\Leftrightarrow |m| := |O| + 1$

$\Leftrightarrow |m| = n + 1$

$\Leftrightarrow |m| - 1 = n$

Dem Nutzer werden $|m|$ Objekte angezeigt, wobei nach der Eingabe der Wörter lediglich $|m| - 1$ Links getestet werden können.

Gegenwärtig akzeptiert der Nutzer offenbar die Eingabe von zwei Wörtern, da dies der Standard von Google-Captchas (Mai 2014) ist. Die Verwendung mehrerer kurzer Wörter ist möglich, die Bereitschaft der Nutzer zur Eingabe würde

jedoch mit zunehmender Wortanzahl stark sinken. Mit der Nachbildung des standardisierten Google-Captchas kann nur ein Link getestet werden. Um die Eingabe zu vereinfachen, werden durch Google Wörter der englischen Sprache verwendet, welche in dem Captcha angezeigt werden. Eine Fortführung, um mehr Links mit dem Lösen eines Captchas zu testen, ist möglich, indem beispielsweise das erste Wort aus einem Substantiv in der Einzahl besteht und dieses dem Prüfwort entspricht. Das folgende "Wort", das einem Link entspricht, besteht lediglich aus einem "s", welches häufig der Mehrzahl der Substantive in der englischen Sprache entspricht. Das gesamte erste Wort kann also auch zum Test eines Links verwendet werden, dennoch ist weiterhin die Anzahl zu testender Links enorm begrenzt.

## Zeichen-Captchas

Durch die Zuordnung einzelner Wörter oder Teile von Wörtern zu einzelnen Links ist die Anzahl der zu testenden Links beschränkt. Daher werden im folgenden Abschnitt Zeichen-Captchas behandelt, mit deren Hilfe deutlich mehr Links überprüft werden können.

Die Aufgabe des Nutzers ähnelt der von Wort-Captchas, nur dass hierbei nicht die Eingabe von Wörtern erfolgt, sondern von zufälligen Zeichenketten, bestehend aus Buchstaben, Zahlen und Sonderzeichen. Um den Angriff zu ermöglichen, wird eine LCD-Schriftart benötigt, die aus 7 Segmenten besteht.[2] Als Hintergrund dient eine weiße Fläche. Nicht besuchte Links werden mit einer schwarzen Textfarbe versehen, besuchte Links erscheinen in Weiß. Grundlage ist, dass einzelne Zeichen, die jeweils einen Link repräsentieren, überlagert werden.

Man verwendet die Kombinationen der Zeichen -, 4, 5 und F. Das Zeichen "-" dient wie in dem Wort-Captcha als Prüfzeichen, dieses soll stets sichtbar sein, damit mindestens dieses Zeichen angezeigt wird. "4, 5 und F" repräsentieren jeweils einen Link. Werden diese Zeichen überlagert, ergeben sich fol-

[2] Innerhalb der Arbeit verwende ich für den Zeichen-Captcha-Angriff die Schriftart "Digital 7 Mono": http://www.myfont.de/fonts/infos/4774-Digital-7-Mono.html

gende Kombinationen, wobei "+" ein Operator zum Überlagern der Zeichen sei:

4 + 5 = 9; 4 + F = A, 5 + F = 6; 4 + 5 + F = 8 (Zachary Weinberg, 2011)

Ein besuchter Link kann nach Baron nicht transparent erscheinen. Somit würden in Weiß erscheinende Zeichen die schwarzen Zeichen vollständig überlagern. Werden jedoch alle Zeichen, die sowohl auf nicht besuchte, als auch besuchte URLs verweisen, transparent dargestellt, können die durch die Überlagerung entstehenden Zeichen gut erkannt werden. Dies ist nach Baron weiterhin erlaubt.

Anders als bei dem Wort-Captcha ist beim Zeichen-Captcha die Eingabe aller Zeichen in der richtigen Reihenfolge notwendig. Beim Wort-Captcha genügt die Eingabe der unterschiedlichen Wörter. An welcher Stelle diese eingegeben wurden, ist irrelevant, da bei der Verarbeitung der Eingabe weiterhin ermittelt werden kann, welches Wort eingegeben wurde, da die Wörter eindeutig und nicht gleich sind. Beim Zeichen-Captcha können die resultierenden Zeichen gleich sein, somit ist die Reihenfolge der Eingabe von Bedeutung. Folglich muss für jede Stelle genau ein Zeichen sichtbar sein, dieses wird durch das Prüfzeichen "-" realisiert.

Die Abbildung #1 zeigt, welche Zeichen durch die Überlagerung einzelner Zeichen entstehen und welche URLs bei welchen Zeichenkombinationen besucht wurden. Das Zeichen "4" repräsentiert die URL1, "5" repräsentiert die URL2 und das "F" die URL3. Diese Zeichen wurden auf den Koordinatenachsen mit einem Kreis markiert. Dies sind die Schnittpunkte mit den Ebenen, welche jeweils eine besuchte URL darstellen. Jede der Ebenen wird senkrecht zur Koordinatenachse gesetzt, welche das mit dem Kreis markierte Zeichen repräsentiert.

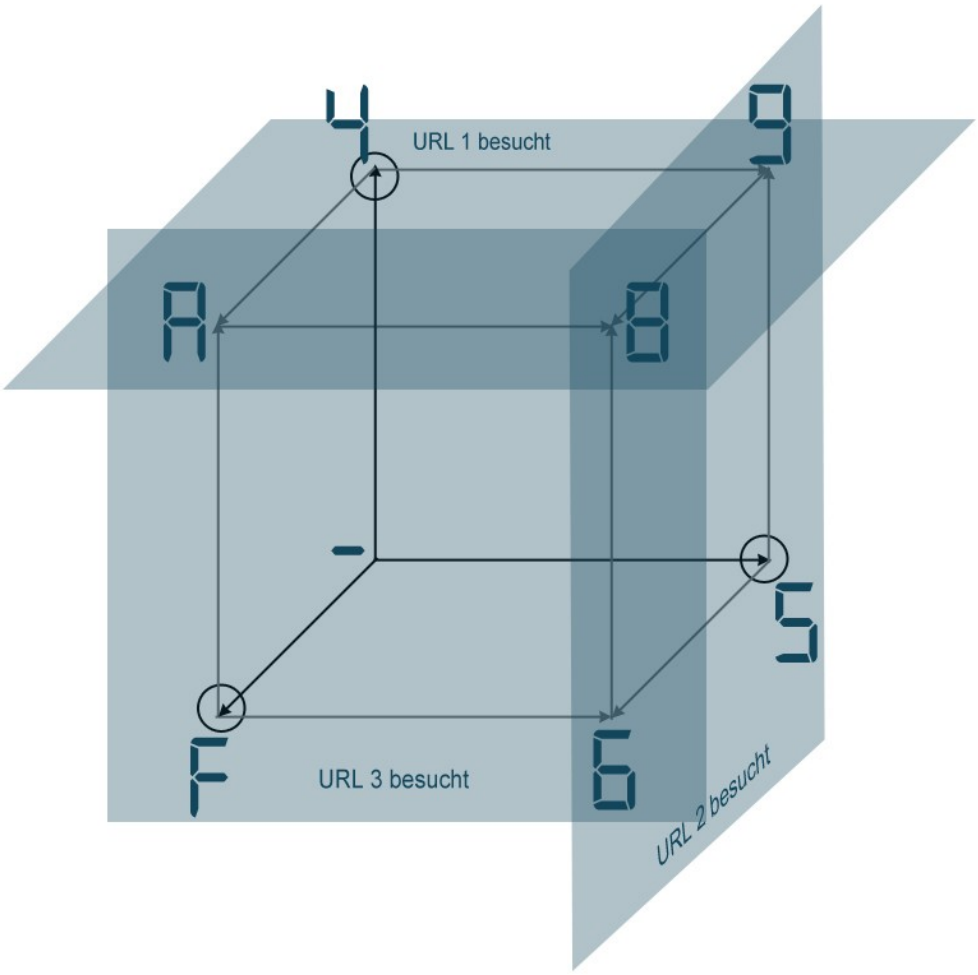


Abbildung #1 - Überlagerung der Zeichen

Jeder Pfeil beschreibt in der Abbildung eine Überlagerung mit allen vorherigen Zeichen, wobei die Überlagerung am Koordinatenursprung mit dem Zeichen "-" beginnt. Am Pfeilende ist das durch die Überlagerung entstehende Zeichen sichtbar. Durch Betrachtung der sich ergebenden Schnittpunkte mit den Ebenen, welche die besuchten URLs repräsentieren, kann nun einfach ermittelt werden, welche Links bereits aufgerufen wurden. Mit jedem Zeichen können zusammenfassend drei Links überprüft werden.

Sei n die Anzahl der zu testenden Links und seien O die Objekte, die jeweils genau einen Link repräsentieren. Dann folgt, gemäß der Wort-Captchas, wobei Wörter durch Zeichen ersetzt wurden: |O| = n <=> |Zeichen| = n, wobei die Notation |x| die Anzahl von x darstellt. Seien m die Objekte, die dem Nutzer präsentiert werden. Nach der vorherigen Ausführung gilt:

Für |m| = 1 folgt: |O| = n => n = 3

=> |m| = 3 * n

=> |m| / 3 = n

Im Vergleich zu den Wort-Captchas (|m|-1 = n) folgt:

für alle m mit |m| > 1: |m|-1 > |m| / 3 und demnach folgt:

für alle n > 0: n +1 < 3*n

Je dargestelltes Objekt können mit den Zeichen-Captchas und der angewandten Überlagerung dreimal so viele Links überprüft werden.

Die Anwendung des Zeichen-Captchas erfolgt im Versuch 2. Dort werden die vorgestellten Zeichen überlagert und diese ergeben neue Zeichen in der Abhängigkeit besuchter URLs. Ein Screenshot des Zeichen-Captchas ist in der Abbildung #2 zu sehen. Nach der Eingabe der sichtbaren Zeichen erfolgt eine serverseitige Auswertung der Rückkoppelung.

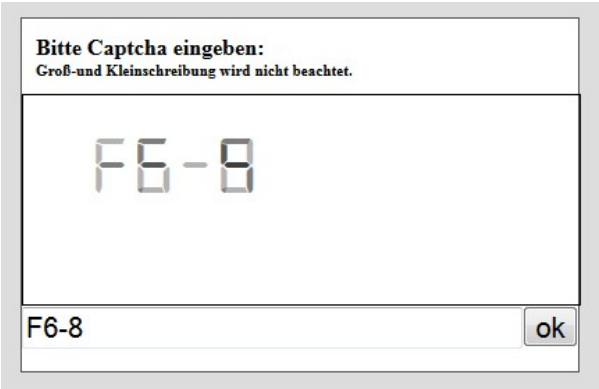


Abbildung #2 Zeichen-Captcha aus dem Versuch 2

## Visual Matching

Die zuvor behandelten Captchas basieren auf der Eingabe von Zeichen durch die Tastatur des Nutzers. Mit der Eingabe eines Zeichens können hierbei maximal drei URLs gleichzeitig überprüft werden. Die Anzahl zu testender URLs pro einzugebendem Zeichen ist auf drei beschränkt, da es keine andere Kombination von überlagerten Zeichen gibt, die in Ihrem Ergebnis vollständig disjunkt ist.

Das Visual Matching wird ebenso als Captcha eingesetzt und basiert auf Nutzereingaben durch die Maus, nicht durch die Tastatur. Im Gegensatz zu den Zeichen-Captchas ist die Basis die Überlagerung von Bildern und nicht von Zeichen. Die Entstehung von unbekannten Zeichen, die durch Überlagerungen von Buchstaben oder Zahlen entstehen können, stellen demnach keine obere Schranke mehr dar. Dem Nutzer wird ein Testbild und zwei Reihen von Bildern präsentiert. Der Nutzer soll die zwei Bilder auswählen, die überlagert das präsentierte Testbild ergeben. Dabei soll ein Bild aus der ersten und ein Bild aus der zweiten Reihe stammen. Dieses Verfahren wird auch "Pattern Matching Puzzle" genannt.

Als Grundlage dient im folgenden ein 4x4-Raster als zusammengesetztes Bild. Zur Vereinfachung werden ausschließlich Schwarz und Weiß als Farben verwendet. Da Hintergrundfarben von Bereichen in Webseiten nicht abhängig von besuchten oder nicht besuchten Links festgelegt werden können, wird im folgenden ein Trick angewandt.

Wenn die Textfarbe eines besuchten und nicht besuchten Links verändert werden kann, besteht die Möglichkeit, einen Bereich zu konstruieren, in welchem sich ein großes Zeichen befindet und in welchem Überlappungen ausgeblendet werden. Wird das Zeichen so platziert, dass die Farbe des Zeichens die gesamte Fläche ausfüllt, kann die Hintergrundfarbe eines Objektes verändert werden, je nachdem, ob ein Link zuvor besucht wurde. Dass dieses Vorgehens möglich ist, wurde im Versuch 3 gezeigt.

Kann ein Hintergrund eines Bereiches nun davon abhängig eingefärbt werden, ob ein hinterlegter Link bereits besucht wurde, kann aus mehreren dieser

Bereiche ein Bild erzeugt werden. Die Verwendung von SVG-Grafiken ist demnach nicht mehr notwendig und ein Angreifer kann grundlegende HTML- und CSS-Funktionen verwenden.

Die zwei Bildreihen, aus welchen der Nutzer jeweils ein Bild auswählt, werden so konstruiert, dass diese unabhängig voneinander sind. Ein Feld des Rasters aus der oberen Reihe, welches mindestens in einem Bild der oberen Reihe ausgefüllt ist, darf in den Bildern der unteren Reihe nicht ausgefüllt sein. Gleiches gilt umgekehrt. Nur so wird gewährleistet, dass sich ausgefüllte Bereiche beider Reihen nicht überlagern. In der Publikation "I Still Know What You Visited Last Summer" (Zachary Weinberg, 2011) wurden die beiden Reihen des "Pattern Matching Puzzle" wie in der folgenden Abbildung #3 beispielhaft konstruiert:

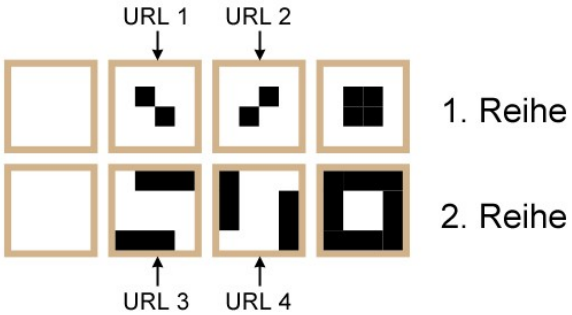


Abbildung #3 Visual Matching Beispiel (Zachary Weinberg, 2011)

Die erste Reihe beschränkt sich auf die inneren vier Flächen, die untere Reihe verwendet ausschließlich die verbleibenden 12 Flächen des Randes. In dem konkreten Beispiel repräsentiert in der ersten Reihe das zweite und dritte Bild jeweils eine URL. Wurde demnach die erste URL bereits besucht, werden zwei Flächen der Diagonale des Testbildes schwarz dargestellt. Gleiches gilt für die URL2, wobei die inneren Segmente der verbleibenden Diagonale schwarz eingefärbt werden, wenn die URL2 zuvor besucht wurde. Alle Bilder, die eine URL repräsentieren, müssen in ihren schwarz eingefärbten Flächen disjunkt sein. Nur so ergibt sich dem Angreifer die Möglichkeit, eine eindeutige Aussage über bereits besuchte URLs zu treffen, nachdem der Nutzer aus jeder Reihe genau ein Bild ausgewählt hat, dessen Überlagerung das Testbild ergibt. Zu-

dem müssen in jeder Reihe alle Bildkombinationen vertreten sein, die auftreten, wenn andere oder keine der URLs derselben Reihe besucht wurden.
Die dem Nutzer dargestellten Bildreihen werden von einem Angreifer als Bilder erstellt. Lediglich das sich aus der Überlagerung der Bilder aus der ersten und zweiten Reihe ergebende neue Testbild wird manipuliert. Die Einzelflächen im Raster des Testbildes werden jeweils einer URL zugewiesen. Für das vorgestellte Beispiel ergibt sich das Raster aus Abbildung #4.

| | | | |
|---|---|---|---|
| URL 4 | URL 3 | URL 3 | URL 3 |
| URL 4 | URL 1 | URL 2 | URL 4 |
| URL 4 | URL 2 | URL 1 | URL 4 |
| URL 3 | URL 3 | URL 3 | URL 4 |

Abbildung #4 Visual Matching, Zuordnung Felder - URLs

Ähnlich wie bei den Zeichen-Captchas kann der Angreifer durch die Interaktion des Nutzers nun genau ermitteln, welche der vier URLs zuvor besucht wurden. Das "Pattern Matching Puzzle" kann theoretisch beliebig erweitert werden. Mit zunehmender Anzahl von Feldern steigt jedoch auch die Anzahl von Möglichkeiten, die sich aus überlagerten Bildern ergeben können, wenn mehrere URLs einer präsentieren Zeile besucht wurden. Das erhöht die Komplexität und erfordert eine höhere Aufmerksamkeit des Nutzers, damit das Captcha auch tatsächlich korrekt gelöst wird.

Die Verwendung von "Pattern Matching Puzzles" führt zusammenfassend genau wie bei den Wort- und Zeichen-Captchas zu einer Rückkoppelung von auf dem Bildschirm dargestellten Farbwerten. Ein Angreifer hat die Möglichkeit, diese gezielt auszuwerten und weiß, wenn das "Pattern Matching Puzzle"

korrekt gelöst wurde, welche der getesteten URLs zuvor besucht wurden. Es besteht demnach auch hierbei ein erhöhtes Missbrauchspotential.

## Captchas - Zusammenfassung

Interaktive Angriffe sind mit Captchas auch heute weiterhin möglich. Die Anzahl zu testender Links ist bei Wort- und Zeichen-Captchas enorm beschränkt, mit zunehmender Interaktion eines Nutzers steigt aber auch die Anzahl der erhaltenen Information über bereits besuchte Links linear, wenn mit jedem Captcha-Aufruf neue URLs getestet werden.

Mit zunehmender Komplexität des zu lösenden Captchas nimmt die Bereitschaft dieses zu lösen ab und auch die Fehlerrate steigt an. Daher können "Pattern Matching Puzzles" nicht beliebig komplex gestaltet werden.

Bei den präsentierten Pseudo-Captchas habe ich auf die Verzerrung der Zeichen, wie dies bei gewöhnlichen Captchas üblich ist, verzichtet. "An attacker [...] could use SVG transformations to distort the text [...]" (Zachary Weinberg, 2011). Es gibt zahlreiche Möglichkeiten, die Pseudo-Captchas so zu erstellen, dass eine Unterscheidung zu echten Captchas ohne den Quelltext zu analysieren unmöglich wird. Solange die Möglichkeit besteht, besuchte und nicht besuchte Links farblich zu unterscheiden, wird stets eine Rückkoppelung konstruierbar sein, die es wie bei den Captchas ermöglicht, die farblichen Informationen direkt vom Nutzer zu erhalten.

## Browsergames

Jedes Szenario, welches eine Rückkoppelung durch den Nutzer verursacht, kann als Basis für einen History-Stealing-Angriff verwendet werden. Durch das vorherige Beispiel wurde sichtbar, dass ein Hintergrund eines Bereiches einer Webseite davon abhängig eingefärbt werden kann, ob eine darauf verweisende URL bereits besucht wurde. Wird das Zeichen, das auf eine URL verweist, so positioniert, dass nicht die gesamte Fläche eines Bereiches ausgefüllt wird, sondern nur partiell, können komplexe Formen entstehen.

Im Versuch 4 wurde durch die Verschiebung des Buchstabens "K" ein Dreieck erzeugt, welches nur sichtbar ist, wenn die verweisende URL des Links bereits besucht wurde. Das Ergebnis ist in der Abbildung A5 sichtbar.

CSS3 ermöglicht, Transformationen auf Objekten durchzuführen, unter anderem auch auf das erstellte Dreieck aus dem Versuch 4. "The transform property applies a 2D or 3D transformation to an element. This property allows you to rotate, scale, move, skew, etc., elements" (Refsnes Data). Werden wie zuvor beschrieben mehrere Dreiecke erstellt und transformiert, ist die Konstruktion komplexer Formen möglich. Im Versuch 5 wurde als Objekt ein Stern auf der Basis von fünf Dreiecken erzeugt (Abbildung A6). Über dieses kann nun eine Ebene positioniert werden, welche Interaktionen des Nutzers mit dem Stern auswertet.

Durch die Verschiebung der Zeichen können als sichtbare Bereiche nicht nur Dreiecke erzeugt werden. Aufgrund der Existenz unzähliger Schriftarten, ist ebenso die Konstruktion einer Vielzahl von Kurven möglich.

Theoretisch ergibt sich die Kontrolle jedes Pixels eines Objektes. Zudem können beliebig komplexe Formen erzeugt werden. In einem Browsergame kann ein Nutzer dazu animiert werden, mit den für ihn sichtbaren Elementen zu interagieren. "Pro Monat spielen rund 290 Millionen Menschen auf Facebook [...]. [Das führt zu einer Spielzeit von] 927.000.000 Stunden [...] pro Monat" (Thöing, 2010).

Ein Browser-Stealing-Angriff durch die Konstruktion von Browsergames ist demnach realistisch, da ein großes Publikum existiert, welches Browsergames intensiv nutzt.

## Videochats

Dass eine Rückkoppelung durch den Nutzer auch in weniger offensichtlichen Konstruktionen möglich ist, zeigt folgendes Beispiel, welches auf der Nutzung von Webcams basiert. Moderne soziale Netzwerke ermöglichen heutzutage die Verwendung von Videochats, in welchen die Nutzer das per Webcam des an-

deren Nutzers aufgezeichnete Video mit einer geringen Verzögerung sehen können. "Computer screens are backlit" (Zachary Weinberg, 2011), das Display eines Computers leuchtet demnach. Trifft das vom Display strahlende Licht auf ein Objekt, zum Beispiel auf das Gesicht des Nutzers, der sich vor dem Bildschirm befindet, ist diese Abweichung der Farben auch für eine Webcam wahrnehmbar.

Wird neben dem Videochat eine Fläche mit einer Hintergrundfarbe, beispielsweise eine Werbeanzeige platziert, kann nun ein Seitenkanal ausgenutzt werden. Die Hintergrundfarbe wird wie beim Visual Matching konstruiert. Dadurch erhält der Hintergrund eine unterschiedliche Farbe, davon abhängig ob ein Link zuvor besucht wurde.

Um die Aussage zu überprüfen, ob eine Webcam tatsächlich Reflektionen von Bildschirmfarben aufnimmt, habe ich den Versuch 6 durchgeführt. Der grundlegende Aufbau ist in der Abbildung #5 dargestellt.

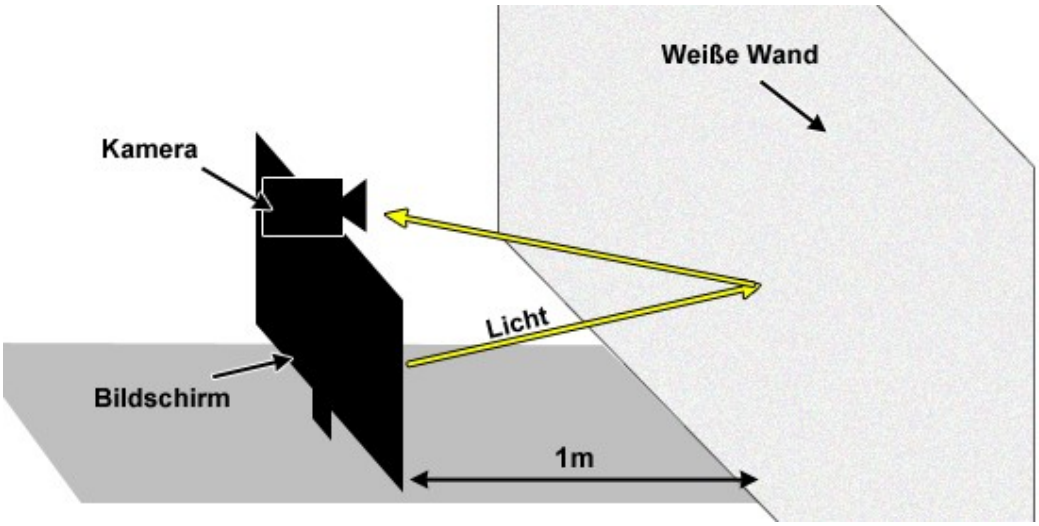


Abbildung #5 Versuch 6

Ein Bildschirm wurde parallel zur Wand, mit einem Abstand von 1m positioniert. Oberhalb des Bildschirms wurde eine Kamera fixiert. Dies erzeugt ein

einfaches Szenario, in welchem physische Bewegungen des Nutzers nicht relevant sind. Auf dem Bildschirm wurden nun nacheinander drei Hintergrundfarben eingeblendet, erst Schwarz, dann Blau und anschließend Gelb. Die Farbflächen haben 50% des Bildschirms ausgefüllt. Während der Einblendung der unterschiedlichen Hintergrundfarben wurde mit der Kamera ein Video aufgenommen.

Abbildung A7 enthält das Ergebnis der reflektierten Farben, welche durch die Kamera aufgenommen wurden. Der Versuch zeigt eine farbliche Veränderung des Bildes in der Kamera, sobald die Farben auf dem Bildschirm verändert wurden. Der Seitenkanal besteht nun darin, dem Nutzer, während die Kamera aktiv ist, nacheinander verschiedene Werbeanzeigen zu präsentieren, die eine unterschiedliche Hintergrundfarbe aufweisen. Die Webcam kann diese Änderungen nun durch Bildanalysen erkennen. Verändert sich der Inhalt der Werbeanzeige nicht auffällig häufig, wird dieser Angriff nicht ohne Weiteres durch den Nutzer erkannt, da ihm die Einblendung von Werbeanzeigen als gewöhnlich erscheint.

Während eines Versuchs an der Carnegie Mellon University wurde genau die dargestellte Rückkoppelung der Informationen durch eine Webcam in weiteren präparierten Umgebungen simuliert, sowohl in einem würfelförmigen Raum, als auch in einem Schlaf- und einem Wohnzimmer. Vorausgesetzt, dass der Raum ausreichend gleichbleibend beleuchtet war, wurde eine Erkennungsgenauigkeit der Änderungen, die durch die Webcam ermittelt wurden, von 100% erreicht (Zachary Weinberg, 2011).

Da Bewegungen der Nutzer das Ergebnis beeinträchtigen, könnte mittels Bildanalysen ermittelt werden, in welchen Phasen die Webcam des Nutzers aktiv, dieser jedoch nicht anwesend ist. In dieser Zeit könnte eine Vielzahl von Links automatisch überprüft werden.

# Zeitbasierte Angriffe auf den Cache

Alle bisherigen Erläuterungen basieren auf dem CSS-Feature, dass besuchte und nicht besuchte Links visuell unterschieden werden können. Es gibt jedoch weitere Seitenkanäle zur Durchführung des History-Stealing-Angriffs. Grundlage hierfür ist zum einen der lokale Cache des Browsers, auch Page-Cache genannt, und zum anderen der DNS-Cache.

## Browser-Cache

Der Browser-Cache ist ein lokaler Zwischenspeicher, der Inhalte von Webseiten temporär speichert, um ein wiederholtes Empfangen identischer Daten zu vermeiden. "They save copies of recently-accessed files [...]" (Edward W. Felten, 2000). Schätzungsweise 46% der HTTP-Anfragen (Souders, 2012) werden im Browser-Cache zwischengespeichert, was zu einer erhöhten Geschwindigkeit bei der Abfrage von im Browser-Cache vorhandenen Webseiten führt, da identische Daten nicht mehrfach empfangen werden müssen.

Innerhalb des Browsers ist der Cache global. Werden demnach Inhalte einer externen Webseite in eine präparierte Webseite eingebunden, werden diese, wenn Sie im Cache vorkommen, aus dem Cache geladen. Mithilfe von Javascript kann die Zeit gemessen werden, wie lange das Laden der Inhalte dauert.

Das folgende Beispiel demonstriert den Angriff. Ein potentieller Angreifer möchte ermitteln, ob die Webseite http://b.com von einem Nutzer bereits besucht wurde. Dazu wurde die Webseite http://a.com vom Angreifer folgendermaßen manipuliert: Wenn der Nutzer die URL http://b.com bereits besucht hat, wurden durch ihn auch die sichtbaren Elemente aufgerufen, beispielsweise ein Bild, das unter http://b.com/bild.jpg erreichbar ist. Die Webseite http://a.com des Angreifers kann nun ermitteln, ob sich das Bild http://b.com/bild.jpg im Cache des Nutzers befindet, wenn der Nutzer die

Webseite http://a.com besucht. Ist das Resultat positiv, wurde die Webseite http://b.com durch den Nutzer in der Vergangenheit aufgerufen.

Durch gezielte Abfragen kann ermittelt werden, ob eine Ressource im lokalen Browser-Cache bereits existiert. Handelt es sich um eine eindeutige Ressource einer Webseite, wurde die Webseite zuvor besucht, wenn die angeforderte Ressource nicht in externe Webseiten eingebunden wurde.

Um den Angriff durchzuführen, muss erstens der Nutzer eine präparierte Webseite aufrufen und zweitens muss ein Verfahren existieren, welches ermittelt, ob sich eine Ressource bereits im Cache eines Nutzers befindet. Ersteres wird nicht weiter thematisiert. Das Verfahren, um deterministisch entscheiden zu können, ob sich eine Ressource im Cache eines Nutzers befindet, wird im folgenden beschrieben.

"The [...] goal [...] will be to determine how accurately an attacker can distinguish cache hits from misses" (Edward W. Felten, 2000). Es wird angenommen, dass einem Angreifer keine Informationen während der Ausführung des Angriffs vorliegen, da dieses Szenario der Realität sehr nahe liegt. Der Angreifer benötigt Messungen zu Ressourcen, die im Cache (Hits) und nicht im Cache (Misses) liegen. Hierzu wird die Zeit von einer Anfrage bis zum vollständigen Eintreffen einer Ressource ermittelt.

Dazu werden vorab zwei unterschiedliche Ressourcen, zum Beispiel Bilder, automatisch nacheinander geladen, wobei angenommen wird, dass der Nutzer diese zuvor nicht aufgerufen hat. Zeitgleich findet eine Messung statt, um die vergangene Zeit von der Abfrage bis zum vollständigen Eintreffen, relativ zur Größe der Ressourcen zu ermitteln. Die erhaltenen Werte seien $M_1$ und $M_2$, wobei 1 und 2 jeweils eine unterschiedliche Ressource repräsentieren. Die beiden Ressourcen werden automatisch im Browser-Cache zwischengespeichert, wenn dieser beim Nutzer nicht deaktiviert wurde. Somit kann der Angreifer messen, wie lange eine erneute Anfrage dieser Ressourcen, relativ zur Dateigröße dauert. Die neuen Werte werden durch $H_1$ und $H_2$ präsentiert.

Nun liegen für zwei Ressourcen die Abfragezeiten relativ zur Dateigröße vor. Diese stammen einmal nicht aus dem Cache ($M_1$, $M_2$) und einmal direkt aus dem Cache ($H_1$, $H_2$). Ein Angreifer kann nun ermitteln, ob der Cache deaktiviert wurde, indem überprüft wird, ob ($M_i < H_i$) oder ($M_i - H_i \leq \varepsilon$ mit $i \in \{1;2\}$ und $\varepsilon \to 0$). Sind beide Bedingungen nicht erfüllt, kann folgende Kalibrierung verwendet werden:

"Choose T as mean of $\max(H_1,H_2)$ and $\min(M_1,M_2)$" (Edward W. Felten, 2000). Für den Wert T ergibt sich demnach eine Schwelle, mit der entschieden werden kann, ob eine Ressource im Cache liegt. Das Verfahren ist als "Four-Sample-Method" bekannt. Die ermittelten Zeiten für Hits liegen nahezu immer deutlich unter den Misses. Wird bei der Anfrage zu einer neuen Ressource ein Wert ermittelt, der kleiner als T ist, wurde die Ressource aus dem Browser-Cache geladen. Ist der Wert größer als T, stammt die Ressource nicht aus dem Cache. Wird der seltene Fall angenommen, dass die gemessene Zeit der Schwelle T entspricht, kann keine eindeutige Aussage getroffen werden. Bei der Anwendung der "Four-Sample-Method" wurde durch Edward W. Felten and Michael A. Schneider eine Treffergenauigkeit von 97,7% ermittelt (Edward W. Felten, 2000).

Für die beschriebene Ermittlung, ob sich eine Ressource bereits im Cache des Browsers befindet, ist eine clientseitige Sprache wie Javascript notwendig. Würden alle Ressourcen nacheinander vollständig geladen werden, könnte folgendes Verfahren serverseitig ermitteln, ob sich eine Ressource im Browser-Cache des Nutzers befindet:

Eine Testdatei könnte beim Aufruf einer manipulierten Webseite durch den Nutzer heruntergeladen werden. Bei diesem Aufruf kann der Server des Angreifers einen Zeitstempel temporär speichern. Dann lädt der Nutzer die zu überprüfende Ressource herunter und daraufhin wiederum eine präparierte Testdatei auf dem Server des Angreifers. Dieser kann nun die vergangene Zeit zur ersten heruntergeladenen Testdatei messen und schlussfolgern, ob eine Ressource im Browser-Cache des Nutzers liegt. Das Verfahren wurde durch wurde durch Edward W. Felten and Michael A. Schneider beschrieben und zeigt, wie die Messung ohne die Verwendung von Javascript funktioniert

(Edward W. Felten, 2000). Mir ist jedoch kein aktueller Browser bekannt, der Ressourcen nacheinander überträgt. Dies erfolgt in der Regel parallel.

Zusammenfassend kann ein potentieller Angreifer mit einer präparierten Webseite und der Verwendung von Javascript mit einer hohen Genauigkeit ermitteln, ob sich eine angefragte Ressource im Cache des Nutzers befindet. Ein Schutz, der das Ermitteln des Vorhandenseins von Ressourcen im Browser-Cache unterbindet, ist zum einen durch die Deaktivierung des Browser-Caches gewährleistet. Dies würde jedoch zu einer erhöhten Datenredundanz im Netzwerk führen, wodurch enorme Geschwindigkeitsverluste entstehen. Zum anderen könnte der Angriff unterbunden werden, wenn Javascript kein Zugang zu einer präzisen Systemzeit ermöglicht wird. Damit wären die notwendigen detaillierten Messungen nicht mehr möglich.

## DNS-Cache

Direkt auf dem Computer werden bei jeder neuen Verbindung zu einer Webseite DNS-Anfragen gestellt, um eine Zuordnung von Domains und Subdomains zu IP-Adressen zu erhalten. Der DNS-Mechanismus ermöglicht einem Nutzer die Verwendung von für ihn lesbaren Domains, welche einer IP-Adresse zugeordnet sind. Die erhaltene Information, welche eine Zuordnung von Domains bzw. Subdomains zu verweisenden IPs enthält, wird durch den Computer direkt oder durch einen angebundenen Router zwischengespeichert. Dies ist der DNS-Cache.

Ein Angreifer kann ähnlich zum Angriff auf den Page-Cache eine Zeitmessung während eines DNS-Lookup durchführen. Wird der DNS-Lookup zweifach durchgeführt und ist die erste Zeitmessung mit der zweiten identisch, kann geschlussfolgert werden, dass der DNS-Lookup lediglich innerhalb des DNS-Caches erfolgt ist, wenn der DNS-Lookup ausreichend schnell erfolgte und der DNS-Cache aktiv ist. Dabei wird davon ausgegangen, dass der zweite DNS-Lookup aus dem Cache erfolgt, da die Zuordnung der angeforderten Domain bzw. Subdomain auf die zu verweisende IP nach der ersten Anfrage temporär im DNS-Cache gespeichert wird. Benötigt der zweite DNS-Lookup

deutlich weniger Zeit als der erste Lookup, ist davon auszugehen, dass die Zuordnung beim ersten Lookup nicht im DNS-Cache vorhanden war und folglich wurde die Webseite, auf welche die Domain bzw. Subdomain verweist, nicht besucht, seitdem der DNS-Cache geleert oder die DNS-Zuordnung entfernt wurde.

Ein Problem liegt in der Vorbereitung zur Definition der zeitlichen Schwelle, welche DNS-Lookups aus dem Cache von Anfragen, die nicht aus dem Cache stammen, unterscheiden. Demnach müssen Zeitmessungen erfolgen, welche die zeitliche Schwelle bzw. einen Schwellenbereich ermitteln. Die Zeitmessung erweist sich als schwierig, da DNS-Lookups je nach Entfernung und Auslastung der DNS-Server zu unterschiedlichen Zeiten führen.

Der DNS-Lookup eröffnet eine weitere Angriffsoption und schränkt gleichzeitig die Eindeutigkeit zur Ermittlung bereits besuchter Webseite ein. "[...] it can also reveal search queries [...], because some browsers [...] prefetch DNS entries [...]" (Zachary Weinberg, 2011). Das DNS-Prefetching ist ein Mechanismus, welcher DNS-Lookups zu Domains und damit zu verweisenden Webseiten erstellt, die wahrscheinlich besucht werden. Vor allem bei mobilen Browsern konnte mit dieser Technik eine gesteigerte Geschwindigkeit zum Laden von Webseiten in Höhe von 5% erreicht werden (Sheppy, 2014).

Durch das DNS-Prefetching werden auch bei Suchanfragen bei bekannten Suchmaschinen DNS-Mappings geladen, bevor die Webseiten besucht wurden. Somit ergibt sich bei dem beschriebenen Angriff auf den DNS-Cache keine Eindeutigkeit, ob eine Webseite tatsächlich besucht wurde oder ob ein DNS-Mapping durch das DNS-Prefetching, beispielsweise durch Suchmaschinen, erfolgte.

Zur eindeutigen Ermittlung, ob eine Webseite durch einen Nutzer tatsächlich zuvor besucht wurde, führt nur der Angriff auf den Page-Cache oder eine der in dieser Arbeit vorgestellten Angriffspfade. Ein anderes Problem besteht darin, dass Caches nur temporär existieren und meist nach kurzer Zeit wieder verworfen werden. Der Angreifer hat also lediglich die Möglichkeit, für einen

kurzen und beschränkten Zeitraum Informationen über die Browser-History zu erhalten.

In der Regel ist ein DNS-Cache nicht nur genau einem Nutzer zugeordnet. Vielmehr existiert häufig eine Cache-Hierarchie, "Multi-Level Cache" genannt. Diese entsteht durch die Nutzung von Proxys und leistungsfähigen Routern, die ebenso DNS-Caches erzeugen. "[...] a client PC's DNS cache is typically backed by another cache that is shared at the departmental level" (Edward W. Felten, 2000). Somit kann die Zugriffszeit bei einem DNS-Lookup reduziert werden, wenn sich Nutzer verschiedener PCs einen Multi-Level Cache teilen und auf dieselben DNS-Ressourcen zugreifen. Teilen sich Mitarbeiter von Abteilungen oder gar ganze Gebäude einen Multi-Level Cache, kann ein Angreifer eine Vielzahl aufgerufener Webseiten ermitteln, die durch die DNS-Abfragen aller Nutzer innerhalb des Multi-Level Caches entstanden sind.

Ein Angreifer kann gezielt testen, ob ein Multi-Level Cache existiert. Ein Nutzer wird kontrolliert durch den Angreifer dazu gebracht, einen eindeutigen DNS-Lookup, dessen Anfragen durch den Angreifer protokolliert werden, durchzuführen. Daraufhin wird ein weiterer Nutzer dazu gebracht, denselben DNS-Lookup durchzuführen und lässt zeitgleich eine Ressource durch den Nutzer anfordern. Erreicht den Angreifer der neue DNS-Lookup nicht und wird dennoch eine angeforderte Ressource durch den weiteren Nutzer empfangen, weiß der Angreifer, dass sich beide Nutzer einen Multi-Level Cache teilen, da andernfalls die angeforderte Ressource des Nutzers nicht eingetroffen wäre.

Dies ergibt ein hohes Gefahrenpotential, da mit dieser Technik zahlreiche Informationen von Firmen oder deren Abteilungen preisgegeben werden. Der Test gemeinsamer Multi-Level Caches kann sogar soweit führen, dass komplette Firmenstrukturen und zusammenarbeitende Nutzer systematisch analysiert werden können.

# Angriffe durch Status Feedbacks

Auf zahlreichen interaktiven Webseiten wie Facebook oder Twitter erfolgt ein Login oftmals über längere Perioden, was ein häufiges erneutes Einloggen des Nutzers vermeidet. Häufig sehen Nutzer, welche aktiv zum Beispiel in einem sozialen Netzwerk eingeloggt sind, andere Informationen als Gäste, die zum aktuellen Zeitpunkt keinen gültigen Login besitzen. Greift ein Gast auf eine URL zu, die Informationen enthält, welche nur für eingeloggte Nutzer abrufbar sind, wird ein Fehler zurückgegeben. Dieser entstehende Fehler kann durch Javascript mit dem "onerror"-Event als "nicht einkalkuliertes Anwenderverhalten" (SELFHTML, Fehlerbehandlung mit dem Event-Handler onerror, 2005) verarbeitet werden. Eine Einschränkung muss hierbei beachtet werden. Ausschließlich Webseiten, welche "Cross-site HTTP requests" unterstützen, gleichzeitig auf HTTP-Status-Codes in einer Javascript-Funktion reagieren und je nach Status-Code unterschiedliche Events starten, liefern externen Programmen eine Rückmeldung.

"Cross-site HTTP requests are HTTP requests for resources from a different domain [...]" (tanob, 2014). Werden demnach Bilder oder auch Javascript-Programme und CSS-Style-Informationen von unterschiedlichen Domains geladen, kommt genau diese Technik zur Anwendung. Alle modernen Browser unterstützen diese Technik und wenige moderne Webseiten machen keinen Gebrauch davon.

Zur Demonstration, wie einfach ermittelt werden kann, ob ein Nutzer bei dem sozialen Netzwerk Facebook angemeldet ist, wurde der Versuch 7 erstellt. Das Javascript-Programm ruft eine persönliche Facebook-Seite[3] auf, welche nur für Nutzer sichtbar ist, die aktuell bei Facebook eingeloggt sind. Nun wird durch Javascript der erhaltene HTTP-Status ausgewertet. Im Erfolgsfall wird der Status-Code 200 zurückgegeben, dieser signalisiert: "The request has succeeded" (R. Fielding J. G.-L., 1999). In diesem Fall bedeutet dies, dass die Seite fehlerfrei geladen wurde. Durch den Status-Code 200 wird die Javascript-Funktion "onload" aufgerufen und es ist möglich, die Information zu verarbeiten, dass

[3] https://www.facebook.com/ruedian

der Nutzer bei Facebook aktuell eingeloggt ist. Das Ergebnis ist in Abbildung A8 sichtbar.

Ist ein Nutzer gegenwärtig nicht bei Facebook eingeloggt, wird der Status-Code 404 erzeugt: "The server has not found [...] the Request-URI" (R. Fielding J. G.-L., 1999). Das Ergebnis dieses Szenarios ist in der Abbildung A9 dargestellt. Dem Nutzer wird demnach ein Fehler zurückgegeben, dass die angeforderte URL nicht aufgerufen werden konnte. Folglich ist der User zum aktuellen Zeitpunkt nicht bei Facebook eingeloggt und auch diese Information kann durch Javascript verarbeitet werden.

"The hack works for all versions of Firefox, Chrome and Safari" (Lübberstedt, unbekannt). Da in diesen Browsern weiterhin die Status-Codes zur Ausführung von Funktionen führen, ist die Anwendung der vorgestellten Technik nach wie vor möglich. Jeremiah Grossman zeigt, dass bereits bei der Abfrage von Bildern in Gmail unterschiedliche HTTP Status-Codes erzeugt werden, abhängig davon, ob ein Nutzer aktuell in der Weboberfläche eingeloggt ist (Grossman, 2008).

Der vorgestellte Seitenkanal kann automatisch eine Vielzahl von Anbietern testen und ermitteln, ob der Nutzer ein potentielles Mitglied ist. Wurde ermittelt, dass ein Nutzer bei der getesteten Webseite eingeloggt ist, dann ist er definitiv auch ein Nutzer der Webseite. Konnte kein aktiver Login ermittelt werden, kann jedoch nicht automatisch geschlussfolgert werden, dass der Nutzer kein Mitglied der Webseite ist, da dieser trotz Mitgliedschaft nicht automatisch aktiv bei der Webseite eingeloggt sein muss. Ist ein Nutzer gegenwärtig bei Facebook eingeloggt, wird die aufgerufene Webseite von Facebook teilweise mit ihren Inhalten geladen. Die erhaltene Antwort enthält etwa 320 KB HTML-Daten. Ist ein Nutzer nicht eingeloggt, wird einer Fehlerseite zurückgegeben, die etwa 24 KB HTML-Daten enthält. Das Empfangen der HTML-Daten benötigt eine unterschiedliche Zeit, die nun also auch gemessen werden kann. Die benötigten Zeiten sind in den Abbildungen A8 und A9 sichtbar. Bei einem aktiven Login dauert das Empfangen der Daten etwa viermal solange wie bei keinem Login. Somit ergibt sich ein weiterer Seitenkanal.

# Zusammenfassung

In dieser Arbeit wurden Angriffe zum indirekten Auslesen der Browser-History vorgestellt. Hierzu wurden zahlreiche Seitenkanäle aufgezeigt, die zwei ursprünglich unabhängige Techniken nutzen. Zum einen wird die Eigenschaft benutzt, dass URLs, die auf bereits besuchte Webseiten verweisen, farblich markiert und durch CSS in ihrem Erscheinungsbild kontrolliert werden können. Zum anderen kann mittels Javascript eine Analyse während des Ladens von Ressourcen wie Bildern durchgeführt werden.

Bis zum Jahr 2010 konnten in hohem Maß Anfragen zu bereits besuchten Webseiten automatisiert durch die Kombination von CSS und Javascript durchgeführt werden, was durch David Baron weitgehend unterbunden wurde. David Baron hat die kritische Vielzahl der automatisierten Angriffe unterbunden, auf Kosten der Funktionalität zur Verwendung von Stils und Effekten bei besuchten Links. Trotzdem existieren weiterhin Seitenkanäle, die durch die Kombination von HTML5, CSS3 und Javascript weiterhin automatisierte Anfragen stellen können.

"Attacks that involve the user [...] [are] outside of browser's control" (Zachary Weinberg, 2011). Solange der Browser besuchte Links anders darstellt, als besuchte, ist immer eine Rückkoppelung durch den Nutzer möglich, zum Beispiel durch die Pseudo-Captchas oder auch durch die Verwendung der Webcam, die visuelle Änderungen des Bildschirms wahrnimmt und zur Webseite zurückführt. Diese Angriffspfade sind trotz der Unterbindung durch David Baron nach wie vor möglich. Auch wenn mit den interaktiven Methoden nur ein Bruchteil von Anfragen erstellt werden kann, ist die Tatsache, dass weiterhin eine sensible Komponente wie die bereits besuchten Webseiten weiterhin analysiert werden kann, beunruhigend, da dies weiterhin den Eingriff in die Privatsphäre ermöglicht.

Einige der präsentierten Seitenkanäle können geschlossen werden, jedoch immer auf Kosten der Funktionalität. Der Browser kann im privaten Modus die aufgezeigten Seitenkanäle einschränken, da nicht zwischen besuchten und nicht besuchten Webseiten unterschieden wird, weil keine Chronik und dem-

nach keine History angelegt wird. Dies führt dazu, dass weder die automatisierten Angriffe zum Auslesen des Stils, noch interaktive Angriffe wie die durch Pseudo-Captchas möglich sind. Angriffe zur Analyse des Caches sind jedoch auch hierbei möglich. Zudem sollte der Browser auf dem aktuellsten Stand gehalten werden, denn nur so können bekannte Angriffspfade, die durch Sicherheitslücken entstehen, vermieden werden.

Dass kritische Sicherheitslücken häufig erst geschlossen werden, wenn von ihnen eine Gefahr ausgeht, zeigte die Studie von Dongseok Jang, Ranjit Jhala, Sorin Lerner und Hovav Shacham aus der Universität in Kalifornien. Die dargestellte kritische Sicherheitslücke war seit 2002 bekannt und wurde erst im Jahr 2010 geschlossen, nachdem zahlreiche Firmen diese ausgenutzt hatten. Auch bei den heute bekannten Sicherheitslücken wird es einen ähnlichen Verlauf geben. "I think if it remains unfixed in the major browsers, unscrupulous sites will start to take advantage of it like they did with the old CSS history hole" (Goodin, 2014).

Offenbar ist die potentiell bestehende Gefahr kein Grund, Änderungen seitens der Browserhersteller durchzuführen. "I did report the vulnerability to Microsoft last year... they replied that they're not planning on fixing it in IE11" (Goodin, 2014).

Auch wenn die negativen Anwendungsmöglichkeiten deutlich überwiegen, gibt es auch einige Fallbeispiele, in denen sich für den Nutzer Vorteile bei einem Browser-History-Angriff ergeben. Eine Bank könnte beispielsweise nach dem Aufrufen einer Onlinebanking-Seite überprüfen, ob bekannte Phishing-Seiten in der Vergangenheit aufgerufen wurden und könnte den Nutzer gezielt warnen (Nielsen, 1995).

Da kontinuierlich neue Features wie HTML5 und CSS3 entwickelt werden, ohne dass betrachtet wird, dass Kombinationen daraus möglicherweise neue Seitenkanäle hervorrufen, werden History-Stealing-Angriffe immer möglich sein, solange die Browser-History oder auch der Cache existieren.

# Abbildungen


Abbildung A1
Werbevideo - Screenshot
Quelle: http://www.genomeplatform.com/

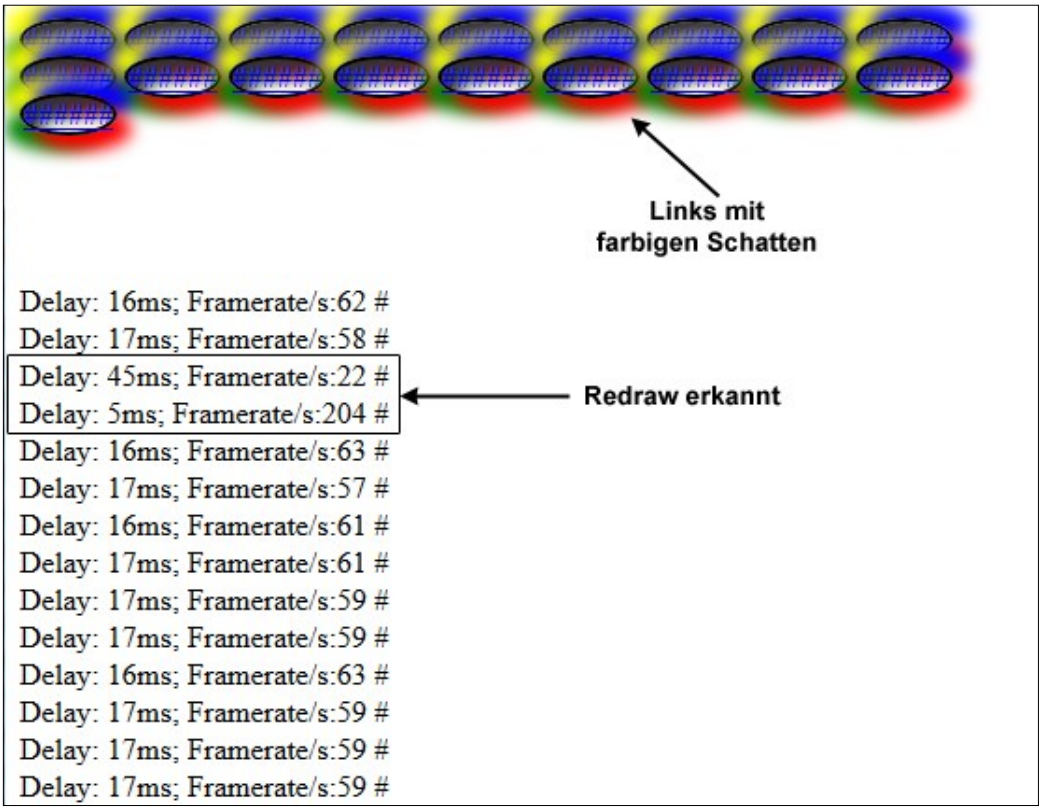


Abbildung A2
Versuch 1 Screenshot

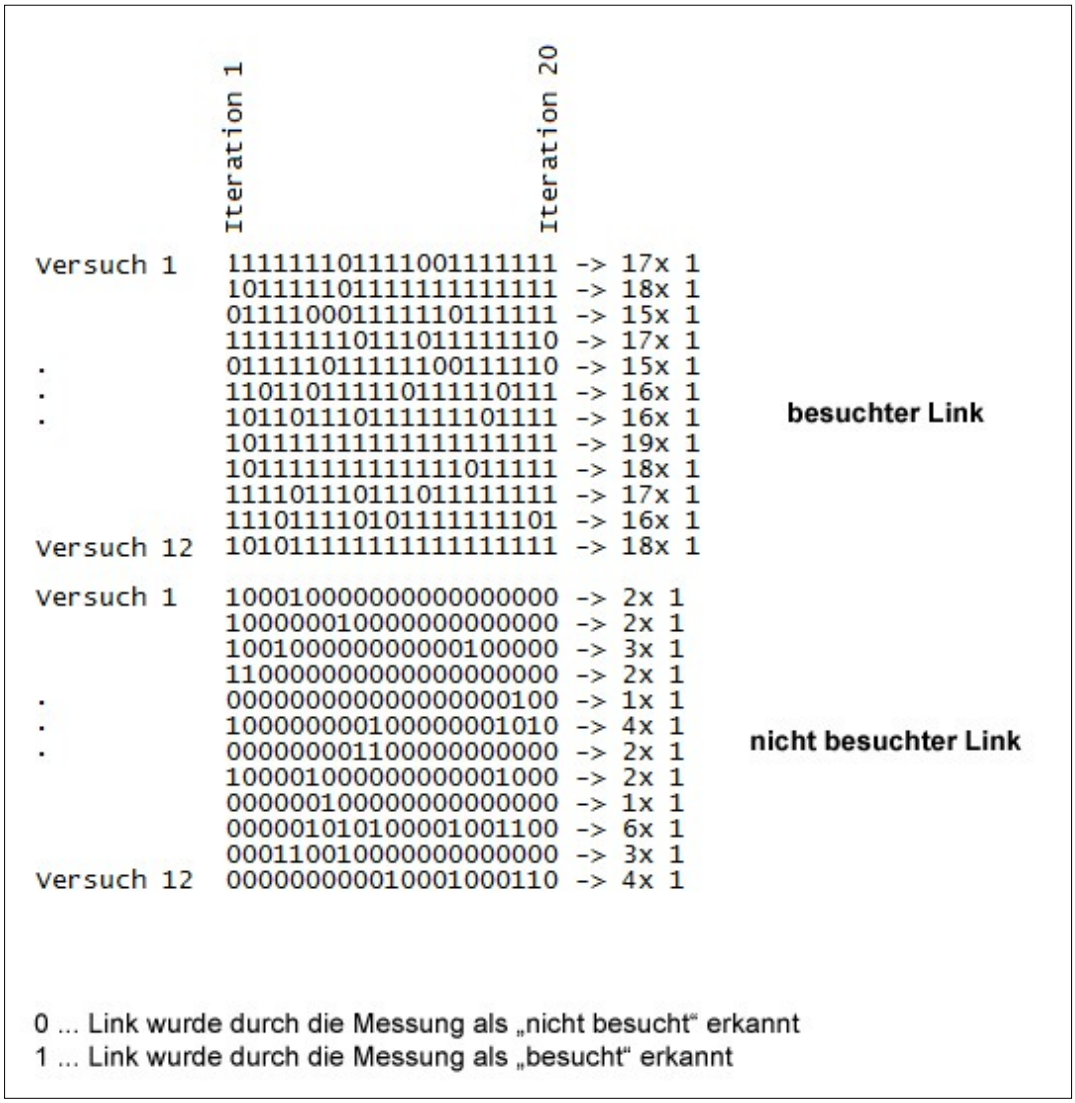


Abbildung A3
Versuch 1 Ergebnis


Abbildung A4
Google-Captcha Mai 2014
Quelle: https://www.google.com/recaptcha


Abbildung A5
Versuch 4 - Erstellung eines Dreiecks

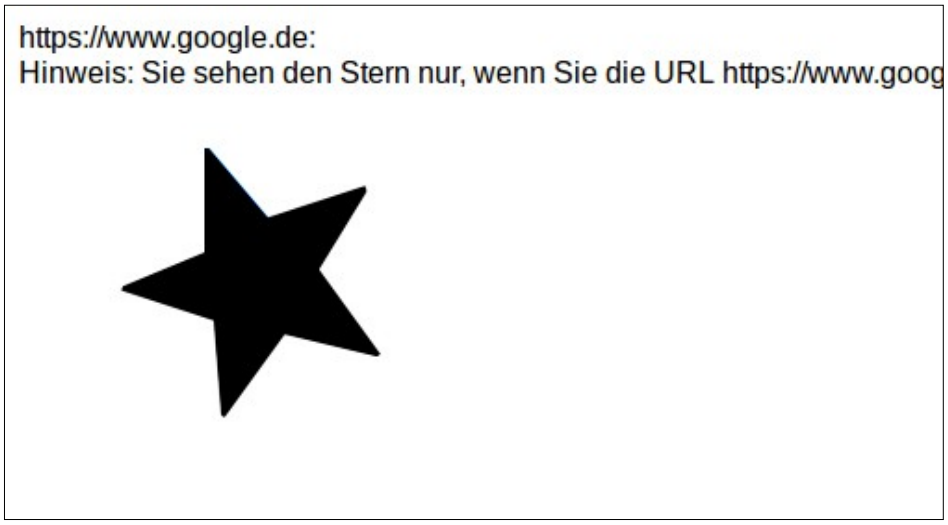


Abbildung A6
Versuch 5 - Erstellung komplexer Figuren

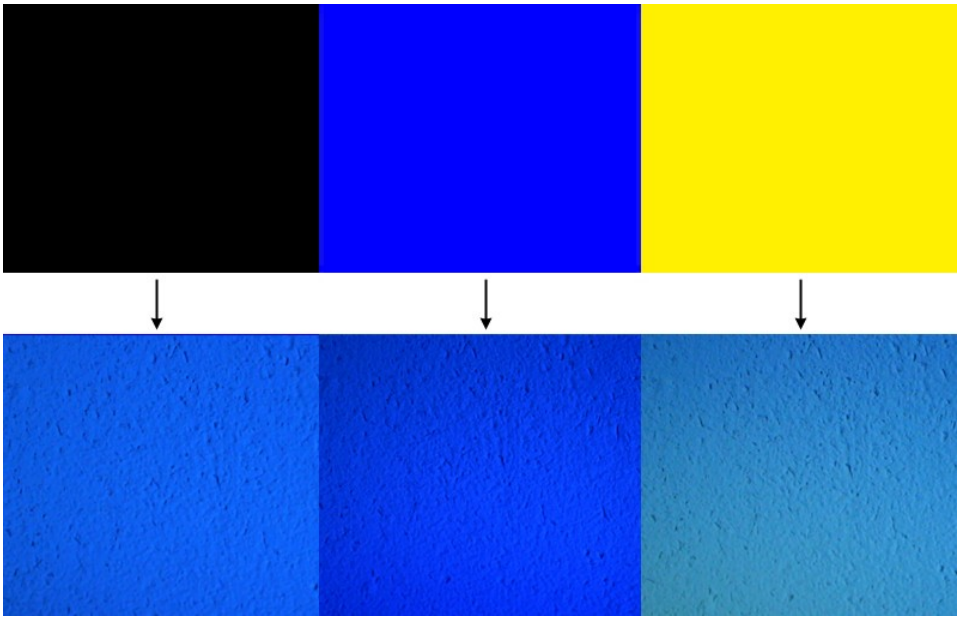

Abbildung A7
Versuch 6 - Ergebnis

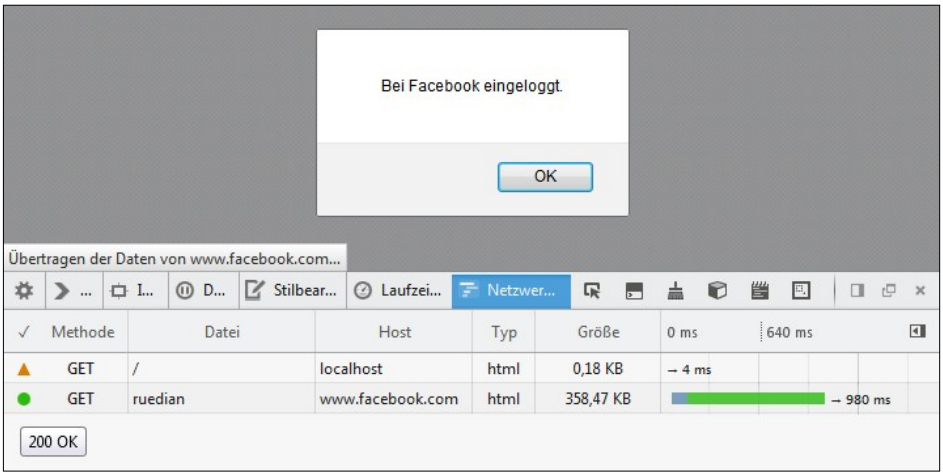


Abbildung A8
Versuch 7 - HTTP-Status 200, Zeit: 980ms

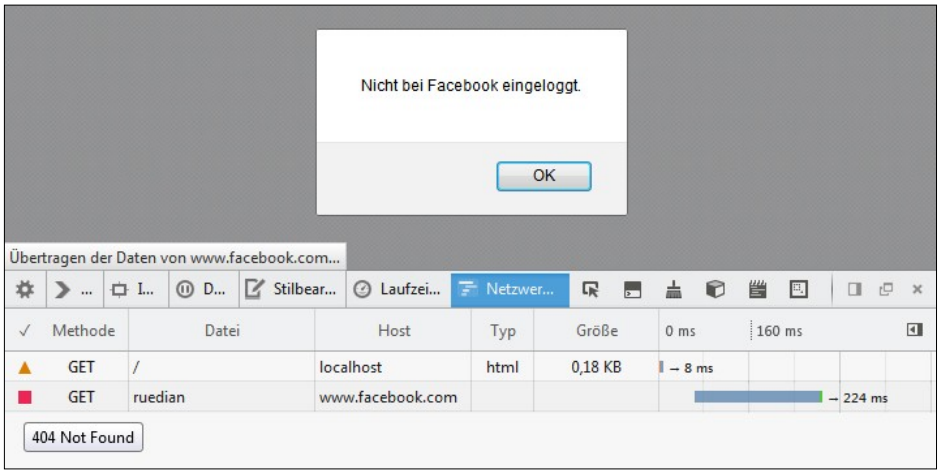


Abbildung A9
Versuch 7 - HTTP-Status 404, Zeit: 224ms

# Literaturverzeichnis

**Artur Janc, L. O. (2010).** *Web Browser History Detection as a Real-World.* Berlin, Heidelberg: Springer-Verlag.

**Bellis, M. (16. 05 2014).** *The History of HTML*. Abgerufen am 16. 06 2014 von http://inventors.about.com/od/computersoftware/a/html.htm

**Blizzard, C. (31. 03 2010).** *privacy-related changes coming to CSS :visited*. Abgerufen am 20. 06 2014 von https://hacks.mozilla.org/2010/03/privacy-related-changes-coming-to-css-vistited/

**Dongseok Jang, R. J. (2010).** An empirical study of privacy-violating information flows in JavaScript web applications. In *CCS '10 Proceedings of the 17th ACM conference on Computer and communications security* (S. 270-283). University of California, San Diego, USA.

**Edward W. Felten, M. A. (2000).** Timing Attacks on Web Privacy. In *CCS'00* (S. 25-32). Princeton: Association for Computing Machinery.

**Goodin, D. (05. 06 2014).** *They're ba-ack: Browser-sniffing ghosts return to haunt Chrome, IE, Firefox (Promoted Comments)*. Abgerufen am 06. 06 2014 von http://arstechnica.com/security/2014/06/theyre-ba-ack-browser-sniffing-ghosts-return-to-haunt-chrome-ie-firefox/

**Grossman, J. (11. 03 2008).** *Login Detection, whose problem is it?* . Abgerufen am 05. 06 2014 von http://jeremiahgrossman.blogspot.de/2008/03/login-detection-whose-problem-is-it.html

**Hégaret, P. L. (19. 01 2005).** *Document Object Model (DOM)*. Abgerufen am 26. 05 2014 von http://www.w3.org/DOM/

**'id', anonym. (28. 02 2007).** *Steal Browser History Without JavaScript*. Abgerufen am 09. 06 2014 von http://ha.ckers.org/blog/20070228/steal-browser-history-without-javascript/

**Lübberstedt, J. (unbekannt).** *Use HTTP status code to check Facebook login status*. Abgerufen am 05. 06 2014 von http://webdevwonders.com/use-http-status-code-to-check-facebook-login-status/

**Markus Jakobsson, T. N. (2007).** *Inferring Context Using Cascading Style Sheets and Browser History*. Abgerufen am 09. 06 2014 von https://www.indiana.edu/~phishing/browser-recon/

**Nielsen, J. (1995).** *Multimedia and hypertext: the internet and beyond.* San Francisco: Vieweg.

**R. Fielding, J. G.-L. (06 1999).** *10 Status Code Definitions*. Abgerufen am 05. 06 2014 von http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html

**R. Fielding, J. G.-L. (06 1999).** *Hypertext Transfer Protocol -- HTTP/1.1*. Abgerufen am 16. 06 2014 von http://www.ietf.org/rfc/rfc2616.txt

**Refsnes Data. (unbekannt).** *CSS3 transform Property*. Abgerufen am 14. 06 2014 von http://www.w3schools.com/cssref/css3_pr_transform.asp

**SELFHTML, R. (22. 08 2005).** *Fehlerbehandlung mit dem Event-Handler onerror*. Abgerufen am 05. 06 2014 von http://de.selfhtml.org/javascript/beispiele/fehlerbehandlung.htm

**SELFHTML, R. (27. 10 2001).** *Pseudoelemente und Pseudoklassen*. Abgerufen am 09. 06 2014 von http://de.selfhtml.org/css/eigenschaften/pseudoformate.htm#link_visited_focus_hover_active

**SELFHTML, R. (31. 10 2005).** *Verweise definieren und gestalten*. Abgerufen am 09. 06 2014 von http://de.selfhtml.org/html/verweise/definieren.htm

**Sheppy, Z. t.-M. (26. 05 2014).** *Controlling DNS prefetching*. Abgerufen am 2014. 06 05 von https://developer.mozilla.org/en-US/docs/Web/HTTP/Controlling_DNS_prefetching

**Souders, S. (22. 03 2012).** *stevesouders.com*. Abgerufen am 02. 06 2014 von Cache them if you can: http://www.stevesouders.com/blog/2012/03/22/cache-them-if-you-can/

**Stone, P. (2013).** Pixel Perfect Timing Attacks with HTML5. *Context Information Security*, (S. 29). London.

**tanob, c. j. (18. 04 2014).** *HTTP access control (CORS)*. Abgerufen am 2014. 06 05 von https://developer.mozilla.org/en-US/docs/Web/HTTP/Access_control_CORS

**'Technocrat', a. B. (04. 12 2010).** *Thoughts of a Technocrat*. Abgerufen am 09. 06 2014 von http://djtechnocrat.blogspot.de/2010/12/browser-privacy-css-history-sniffing-in.html

**Thöing, S. (21. 10 2010).** *Facebook: 927.000.000 Stunden Spielzeit pro Monat - Social-Gaming-Statistik erschienen*. Abgerufen am 15. 06 2014 von http://www.pcgames.de/Facebook-Firma-215528/News/Facebook-927000000-Stunden-Spielzeit-pro-Monat-Social-Gaming-Statistik-erschienen-795027/

**Zachary Weinberg, E. Y. (2011).** *I Still Know What You Visited Last Summer.* Mellon University.

## Selbstständigkeitserklärung

Ich erkläre hiermit, dass ich die vorliegende Arbeit selbstständig verfasst und nur unter Verwendung der angegebenen Quellen und Hilfsmittel angefertigt habe. Weiterhin erkläre ich, eine Bachelorarbeit in diesem Studiengang erstmalig einzureichen.

Berlin, den 01.07.2014 .....