A=SIT Technology
E-Government Innovationszentrum

17. März 2016 Studie · Web & Cloud · Privatsphäre & Krypto · 18 Min. Lesezeit

Privatsphäre-Auswirkungen neuer Web-Technologien

Archiviert. Dieser Inhalt wird nicht mehr aktualisiert. Er bleibt zur Information verfügbar.

Quelle https://technology2.a-sit.at/veroeffentlichungen/privatsphaere-auswirkungen-neuer-web-technologien/

In dieser Kurzstudie werden bestehende Web-Tracking Technologien und Methoden aufgezeigt. Basierend auf diesen Methoden werden zwei neue Web-Technologien, Web Sockets und WebRTC analysiert, mit Hinblick auf Auswirkungen auf die Privatsphäre von Benutzerinnen und Benutzern. Unter Verwendung dieser neuen Web Technologien werden vier Szenarien entwickelt, die einerseits die Privatsphäre von Benutzerinnen und Benutzern gefährden und andererseits die unbemerkte Identifikation von einzelnen Benutzerinnen und Benutzern erheblich „verbessern" können.

Die Resultate dieser Kurzstudie zeigen, dass es möglich ist, basierend auf einem Satz von lokalen IP Adressen die zufällige Komponenten enthalten, einen Rechner zu identifizieren, egal welcher Browser verwendet wird. Die Identifikation eines Browsers kommt in einer mobil-affinen Welt der Identifikation einer Benutzerin bzw. eines Benutzers gleich. Außerdem wurden weitere Möglichkeiten beschrieben, wie WebSockets und im speziellen WebRTC ausgenutzt werden kann um DDoS Attacken durchzuführen.

1. Einleitung

Dieses Dokument beschreibt die Auswirkungen von neuen Web-Technologien auf die Privatsphäre von Benutzerinnen und Benutzern. Die Privatsphäre wird einerseits durch gezieltes Tracking von Benutzerinnen und Benutzern gefährdet, und anderseits von Kommunikationen die schwer von beispielsweise Firewalls ausgewertet werden können. Seit der Entwicklung des World Wide Webs haben sich die Möglichkeiten eine Besucherin bzw. einen Besucher einer Webseite zu verfolgen stetig verbessert. Abschnitt 2 gibt einen kurzen Überblick über bestehende Tracking-Methoden. Abschnitt 3 gibt einen Überblick über die WebRTC und WebSockets Technologien und deren Auswirkungen auf die Privatsphäre von Benutzerinnen und Benutzern. Weiteres wurde im Rahmen dieser Studie ein Framework entwickelt, dass die Ergebnisse bestehender Web-Fingerprinting Methoden *„verbessern"* kann. Die praktische Umsetzung und Evaluierung wird in Abschnitt 4 erläutert. Es wird gezeigt, dass Tracking mittels neuer Web-Technologien weiter *„verbessert"* werden kann. Außerdem werden weitere Implikationen der WebRTC und WebSocket Technologien auf die Privatsphäre von Benutzerinnen und Benutzern aufgezeigt.

2. Hintergrund

Für Angreiferinnen und Angreifer sind Webtechnologien durch die breite Verfügbarkeit ein attraktives Ziel, um die Privatsphäre bzw. sensible Daten von Benutzerinnen und Benutzern zu kompromittieren. Eine Möglichkeit dazu ist es Benutzerinnen bzw. Benutzer unbemerkt zu verfolgen und somit deren Surfverhalten bzw. die Häufigkeit und Art von Aufrufen zu analysieren. In den seltensten Fällen beschränkt sich diese Analyse auf eine Domain und wird oft über Domaingrenzen hinweg ausgeführt. In den Anfangszeiten des Webs konnten Besucherinnen und Besucher von Webseiten nur über ihre IP Adresse und über GET bzw. POST Parameter verfolgt werden, außerdem war der Workflow klar geregelt: Clients senden Anfragen, Server antworten auf diese Anfragen. Asynchrone Kommunikationsmöglichkeiten sind erst später hinzugekommen. 1994 wurden die Tracking-Möglichkeiten mit der Einführung von Cookies erweitert. Tracking Codes werden dabei in Cookies gespeichert und bei jedem Besuch der entsprechenden Webseite vom Browser wieder preisgegeben. Viele weitere und einfallsreiche Methoden wurden daraufhin entwickelt, die dazu verwendet werden können um Benutzerinnen und Benutzer zu verfolgen bzw. deren Privatsphäre zu gefährden. Eine Auflistung von relevanten Technologien ist folgend gegeben.

- 1994

o HTTP Cookies [1]

- 2000

o Einfügen von IDs in zwischengespeicherten Dokumenten [2] o Ladeperformance Tests [2] o DNS Zwischenspeicher [2]

- 2007

o HTTP Authentifizierungszwischenspeicher [3]

- 2008

o window.name DOM Attribut [4] o Clickjacking [5]

- 2009

o Flash Cookies [6][7]

- 2010

o Evercookies [8]

- 2011

o Flash LocalConnection Objekt [9] o ETag und Last-Modified HTTP-Headerfelder [10] o HTTP 301 Umleitungszwischenspeicher [11]

- 2012

o Cookie leaks/syncing [12] o Werbenetzwerke [12] o HTTP Strict Transport Security (HSTS) Zwischenspeicher [13] o Browser Identifizierung mittels Canvas [14]

- 2014

o Web SQL DB und HTML5 IndexedDB [15] o Hinzufügen von HTTP-Headerfeldern bei ausgehenden HTTP-Anfragen [16] Als besonders hartnäckig haben sich in diesem Zusammenhang *Evercookies* erwiesen. Evercookies speichern Daten nicht nur in normalen Browser Cookies oder Flash-Cookies, sondern verwenden eine Vielzahl an verschiedenen Speicherorten:

- Standard HTTP Cookies
- HTTP Strict Transport Security (HSTS) Pinning
- Local Shared Objects (Flash Cookies)
- Silverlight Isolated Storage
- Automatisch generierte PNG Bilder
- Speichern von Cookies in der Web History
- Speichern von Cookies in HTTP ETags
- Speichern von Cookies im Web cache
    - Caching von window.name
    - Internet Explorer spezifischer userData Speicher
    - HTML5 Session Speicher
    - HTML5 Lokaler Speicher
    - HTML5 Globaler Speicher
    - HTML5 WebSQL
    - HTML5 IndexedDB
    - Java JNLP PersistenceService

- Java CVE-2013-0422 Exploit Wird von der *Evercookie* Implementierung erkannt, dass beispielsweise die Browsercookies geleert wurden, so wird dieses aus einem anderen Speicherbereich wiederhergestellt. Diese Technologien sind davon abhängig von einem Server einen Tracking Code zu erhalten, diesen zu speichern und in weiterer Folge wieder zurück zu liefern. Das *fingerprintjs2 [1]* Framework verfolgt hier einen anderen Weg, und generiert im Browser einer Benutzerin bzw. eines Benutzers einen Identifier, der abhängig von der Konfiguration und den Komponenten des jeweiligen Geräts bzw. Betriebssystems ist. Es wird hierbei die Existenz von verschiedenen Plattform Features abgefragt bzw. werden die assoziierten Werte verwendet um mit Hilfe einer Hash Funktion einen eindeutigen Identifier für den jeweiligen Browser zu generieren. Dabei ist es oftmals unerheblich ob sich der Browser im privaten oder normalen Modus befindet. *Fingerprintjs2* verwendet zur Berechnung eines möglichst eindeutigen Fingerabdrucks des Browsers folgende Quellen:
- User Agent
- Sprache
- Farbtiefe
- Auflösung
- Zeitzone
- Verfügbarkeit eines Session Speichers im Browser
- Verfügbarkeit des Lokalen Speichers im Browser
- Verfügbarkeit der indexedDB
- Verfügbarkeit von Internet Explorer spezifischen Methoden/Eigenschaften
- Verfügbarkeit von open DB
- CPU Informationen
- Verwendete Plattform
- DoNotTrack aktiviert
- Installierte Schriftarten die über Flash abgerufen werden können
- Installierte Schriftarten die über JS/CSS abgerufen werden können (bis zu 500)
- Canvas Fingerprinting
- WebGL Fingerprinting
- Installierte Plugins (inclusive AdBlock)
- Informationen über veränderte User Agent und Sprachwerte

- Touch Screen Verfügbarkeit und Fähigkeiten Ein Defizit dieses Verfahrens ist, dass Browser identifiziert werden, aber nicht Geräte oder Benutzerinnen bzw. Benutzer. Mehreren Browsern auf demselben Gerät werden somit unterschiedliche Identifier zugeordnet. Das Problem kann durch Server basierte Lösungen ebenfalls nicht gelöst werden, die Hürde kann aber durch den zusätzlichen Einsatz von neuen Webtechnologien niedriger gesetzt werden.

3. Neue Web-Kommunikationstechnologien

3.1. WebSockets

Bei WebSocket Verbindungen handelt es sich um normale HTTP Verbindungen, die nach dem Aufbau der Verbindung zu einer WebSocket Verbindung hochgestuft werden. Wurde die Verbindung aufgebaut, so können Rohdaten zwischen dem Client (Webbrowser) und Server in 1 https://github.com/Valve/fingerprintjs2 beide Richtungen gesendet werden, im Unterschied zu normalen HTTP Anforderungen die einem strikten Request-Response Modell folgen. Die relevanten Felder des HTTP Requests, *Codeblock 1 - HTTP Upgrade Request*

```
Sec-WebSocket-Version: 13
Sec-WebSocket-Key: 9clAt83+pmfGQxWl8v4Ktw==
Connection: Upgrade
Upgrade: websocket
```

Codeblock 2 - HTTP Upgrade Response

```
HTTP/1.1 101 WebSocket Protocol Handshake
Connection: Upgrade
Upgrade: websocket
```

um eine Verbindung zu einer WebSocket Verbindung aufzuwerten, ist in Codeblock 1 dargestellt. Analog dazu werden in Codeblock 2 die relevanten Felder der Upgrade Response dargestellt. Sicherheitstechnisch sind WebSockets nicht unumstritten, da sie mehrere Sicherheitsprobleme verursachen können. Firewalls beispielsweise sehen nur eine einzige Verbindung und können zwischen unterschiedlichen Paketen nicht unterscheiden, da keine standardisierten Header zum Einsatz kommen. Erkkilä [17] hat eine Sicherheitsanalyse der WebSocket-Technologie durchgeführt und hat folgende Hauptsicherheitsrelevanten Probleme identifiziert:

```
- _Same Origin Policy:_ Die Same Origin Policy untersagt den Zugriff auf Objekte
anderer Herkunft. Eine Webseite _http://web.site_ kann also beispielsweise per
XMLHttpRequest nicht mit der Seite _http://evil.site_ kommunizieren. Saiedian und
Broyle [18] beschreiben dennoch Möglichkeiten, diese zu umgehen, und zeigen
Probleme mit heutigen Webanwendungen. WebSocket Verbindungen werden von der Same
Origin Policy nicht eingeschränkt. Sie verfolgen einen _verified origin_ [19]
Ansatz bei dem die Entscheidung beim Zielserver liegt, eine Verbindung anzunehmen
oder abzulehnen. Es können somit Verbindungen von einer Webseite zu beliebigen
Server aufgebaut werden.

- _Proxies und Firewalls:_ Proxy Server und Firewalls erwarten HTTP Verbindungen
auf Port 80 bzw. 443. WebSockets können transparent auf beliebigen TCP Ports
betrieben werden.

- _Bösartige Services:_ Sobald eine bösartige Webseite im Browser ausgeführt wird,
kann diese Verbindungen zu beliebigen WebSocket Servern aufbauen. Eine bösartige
Seite könnte beispielsweise eine für Cross-Site-Scripting anfällige Webseite sein.
```

WebSockets haben keinen direkten Einfluss auf die Effizienz von Browser Fingerprinting Frameworks, können aber dazu verwendet werden, um Informationen an Firewalls vorbei zu übertragen.

3.2. WebRTC

Die WebRTC [20] Technologie geht einen Schritt weiter als WebSockets und ermöglicht die direkte Kommunikation von zwei Browser Instanzen ohne dazwischenliegenden Server, sobald Verbindungsdaten über einen separaten Kanal ausgetauscht wurden. Primär wurde WebRTC zum Video und Audio Streaming zwischen Endgeräten entwickelt, und bietet daher eine große Auswahl an Real-Time Transport Protocol (RTP) Media APIs. Außerdem stellt die API Rohdaten Kanäle bereit, um beliebige Daten kommunizieren zu können. Eine Architekturübersicht von WebRTC ist in **Fehler! Verweisquelle konnte nicht gefunden erden.** dargestellt. Um eine direkte Verbindung aufzubauen, müssen über einen abgesicherten Kanal und Rendezvous Mechanismus im ersten Schritt Verbindungsdaten ausgetauscht werden. WebRTC verwendet hierfür das Session Description Protocol (SDP) [21]. Hierbei wird über den Rendezvous Mechanismus ein *Offer* und eine *Answer* vom Empfänger ausgetauscht, die die benötigten Daten zum Verbindungsaufbau beinhaltet. Um eine Verbindung auch zwischen Rechnern hinter Firewalls zu ermöglichen, werden STUN [22] und TURN [23] Server zur Unterstützung verwendet. Der Rendezvous Mechanismus ist laut WebRTC Spezifikation nicht näher definiert und es bleibt der Entwicklerin bzw. dem Entwickler überlassen einen geeigneten Kanal zu wählen (beispielsweise XMLHTTPRequest oder WebSockets). WebRTC wird außerdem von verschiedensten Publikationen [24]–[26] als grundlegende Verbindungsstruktur für Peer-to-Peer Netzwerke im Browser verwendet.

2 Die in diesem Bild verwendeten Grafiken unterliegen der Creative Commons 3.0 Lizenz und wurden von folgenden Adressen bezogen: https://www.iconfinder.com/icons/104885/browser_chrome_icon, https://www.iconfinder.com/icons/104886/browser_firefox_icon, https://www.iconfinder.com/icons/370088/cloud_cloudy_weather_winter_icon, https://www.iconfinder.com/icons/111017/server_icon

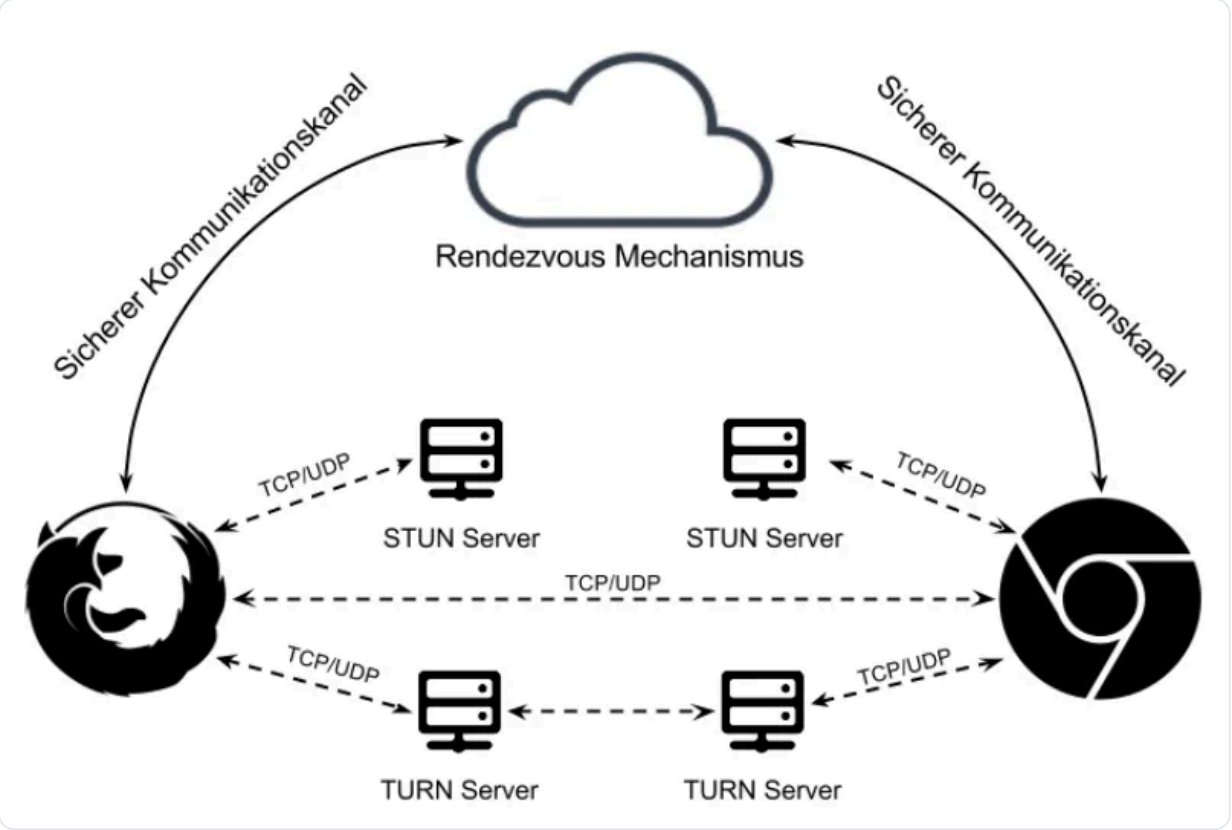


Abbildung 1: - WebRTC Architektur [2] Die Verbindungssicherheit von WebRTC basiert auf erprobten Methoden: Datagram Transport Layer Security (DTLS) [27] um Datenverbindungen abzusichern, Secure Real-Time Transport Protocol (SRTP) [28] um Medienverbindungen abzusichern. Die WebRTC Spezifikation erlaubt es nicht, unverschlüsselte Pakete über eine WebRTC Verbindung zu schicken, es muss immer die entsprechende Technik (DTLS bzw. SRTP) zum Einsatz kommen. Der Schlüsselaustausch bzw. das Herstellen über DTLS, wie in Abbildung 2 erläutert, funktioniert ähnlich wie das Standard TLS Protokoll. Im WebRTC Fall generieren beide Endpunkte für jede Verbindung ein neues selbstsigniertes Zertifikat, tauschen die Zertifikatsfingerabdrücke über den Rendezvousmechanismus aus und starten dann die Herstellung der direkten Verbindung laut dem DTLS Protokoll. Dadurch wird ein gesicherter und authentifizierter Kanal (sofern der Rendezvousmechanismus vertrauenswürdig ist) sichergestellt. Das SDP Offer beinhaltet sicherheitsrelevante Parameter wie in Codeblock 3 dargestellt. Die Parameter ice-ufrag und ice-pwd werden dazu verwendet um die entsprechende Gegenstelle im Verbindungsprozess wiederzuerkennen, der fingerprint Parameter wird dazu verwendet, um das Zertifikat der Gegenstelle abzugleichen.

Codeblock 3 - Sicherheitsrelevante Parameter im SDP Offer

```
a=ice-ufrag:EHWMPYXH4P3h0cmb
a=ice-pwd:eEkNiCOtWLbyrfA9anakJbeq
a=fingerprint:sha-256
30:AE:32:2E:4E:2E:7D:3B:E9:35:6C:7F:84:CA:23:75:F0:34:FB:E9:4
2:C9:2B:D9:66:86:D1:16:2F:FA:EE:32
```

Abbildung 2 - WebRTC Verbindungsherstellungsprozess Basierend auf dem detaillierten Verständnis der Funktionsweise der WebRTC Technologie wurden mehrere Schwachstellen identifiziert die in Kapitel 4 evaluiert werden.

4. Evaluierung

Es wurden in den besprochenen Technologien mehrere Schwachstellen identifiziert und in folgenden Kapiteln evaluiert:

```
- Problem beim Umgang mit nicht vertrauenswürdigen Rendezvous Mechanismen

- Privatsphäre Verlust durch Erforschung des lokalen Netzwerks

- Benutzung von WebRTC für groß angelegte DDos Attacken

- Unterstützung von Browser-Fingerprinting Mechanismen beim Identifizieren von
Benutzerinnen und Benutzern, auch über mehrere Browser hinweg.
```

4.1. Nicht vertrauenswürdiger Rendezvous Mechanismus

Für WebRTC gibt es prinzipiell zwei unterschiedliche Anwendungsszenarien:

```
- Webapplikationen die direkt im Browser ausgeführt werden.
```

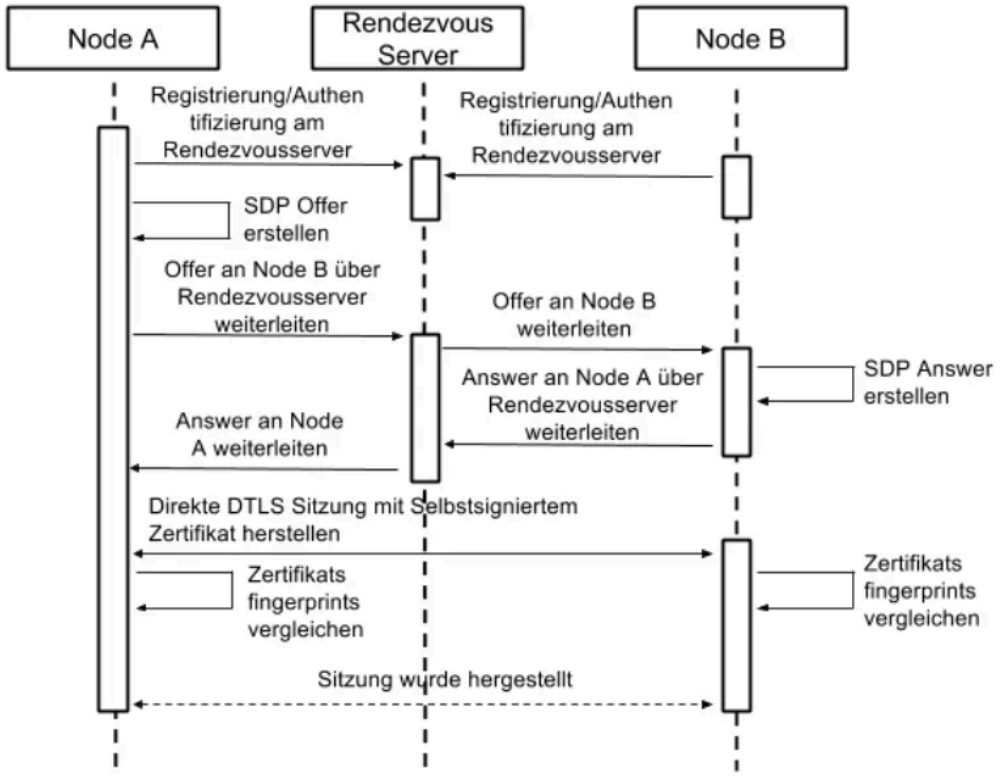


Abbildung 3: - Man-in-the-Middle-Angriff über kompromittierten Rendezvousserver Alice der Meinung mit Bob verbunden zu sein, hat aber tatsächlich eine Verbindung zu Eve, welche wiederrum eine Verbindung zu Bob hat. In diesem Szenario wurden die Rendezvousserver Uplinks als gesichert angesehen. Sollte der Uplink nicht gesichert sein so ergibt sich ein anderes Bild, bei dem der Rendezvousserver nicht kompromittiert sein muss. Dieser Fall ist in Abbildung 4 dargestellt.

- Cross-Platform oder native mobile bzw. Desktop Applikationen, die direkt am Gerät installiert sind. Im ersteren Fall scheint es sinnvoll zu sein, dass der Rendezvousserver auch gleichzeitig der Webserver ist. Für eine Angreiferin bzw. einen Angreifer ist es sicher mit weniger Aufwand verbunden, direkt die Webapplikation zu verändern als den Rendezvousserver. Im letzteren Fall ist der Rendezvousserver unabhängig vom Rest der Applikation. Hat ein Angreifer den Rendezvousserver unter Kontrolle ist es ein Leichtes Man-in-the-Middle-Angriffe durchzuführen, ohne das beteiligte Nodes diese erkennen können. Der Prozess ist in Abbildung 3 dargestellt. Die Ausgangssituation ist, dass Alice mit Bob direkt kommunizieren möchte. Alice erzeugt also ein _Offer_ und beabsichtigt es an Bob über den Rendezvousserver zu senden. Der kompromittierte Rendezvousserver leitet das _Offer_ aber an Eve weiter. Eve erstellt nun ein zweites _Offer_ für Bob und leitet es an Bob über den Rendezvousserver weiter. Bob kann zu diesem Zeitpunkt nicht unterscheiden ob ein _Offer_ von Eve oder von Alice stammt und muss auf die Angaben des Rendezvousserver vertrauen. Bob erstellt also eine _Answer_ und glaubt sie an Alice zurückzusenden. Sie wird aber vom Rendezvousserver an Eve weitergeleitet, welche wiederrum eine _Answer_ auf das ursprüngliche _Offer_ von Alice erstellt. Zu diesem Zeitpunkt ist

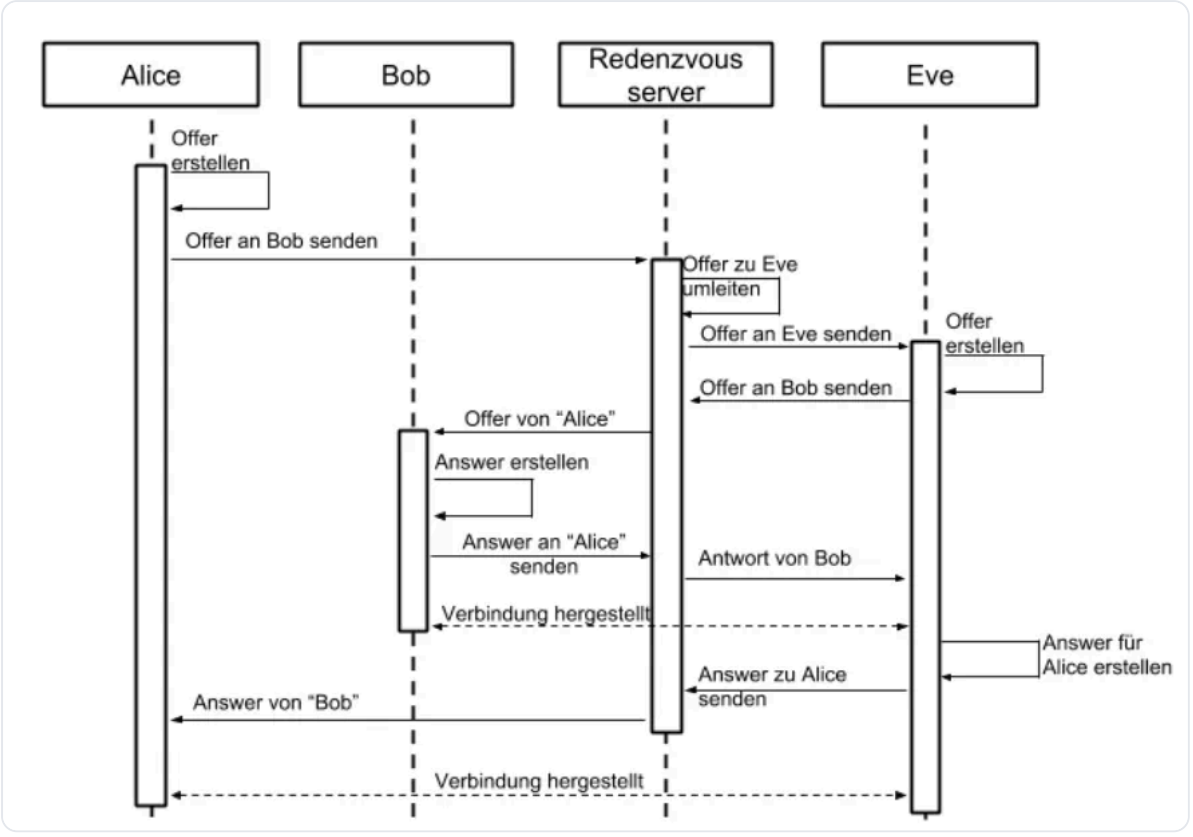
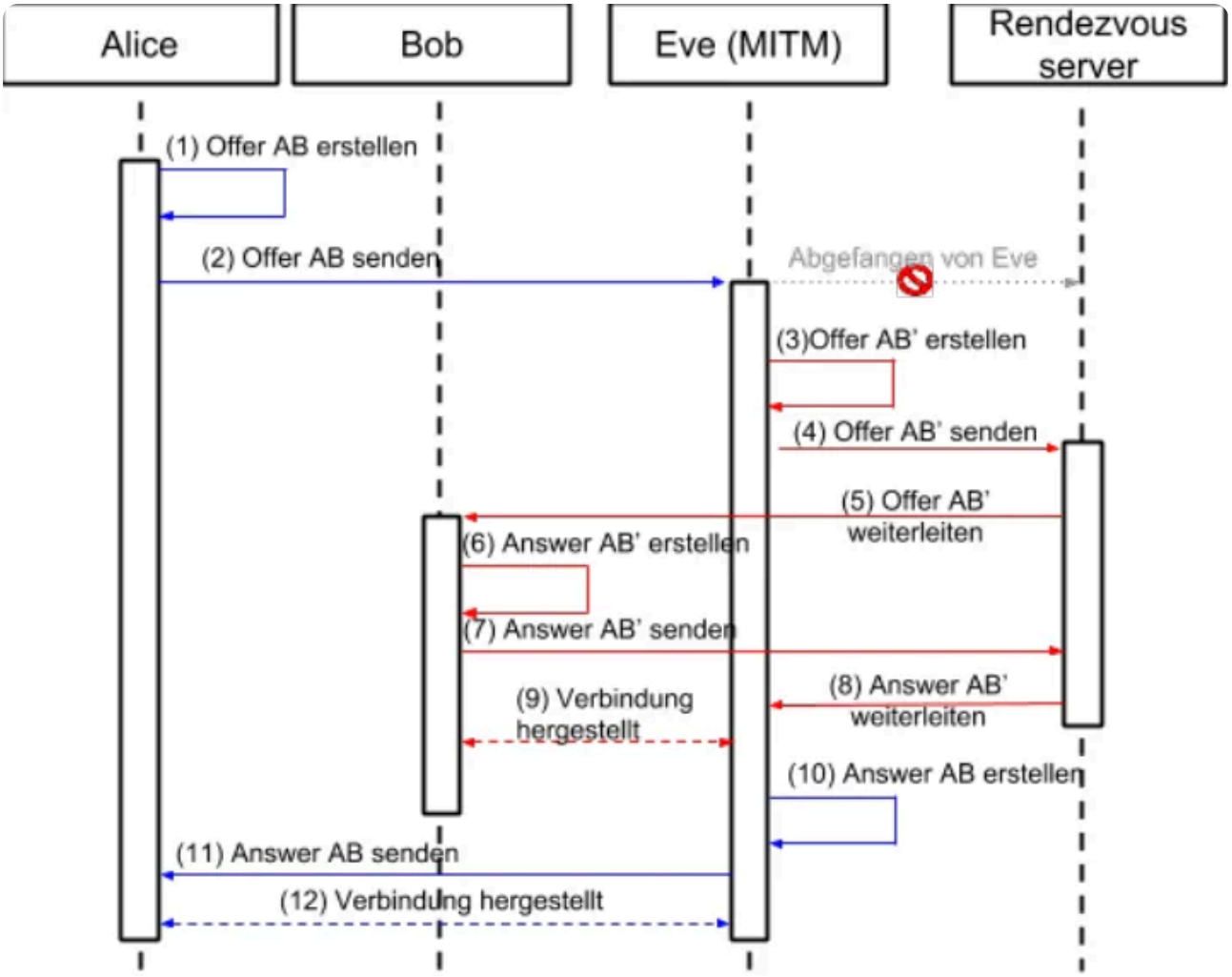


Abbildung 4: - Ungesicherter Uplink zum Rendezvousserver In diesem Szenario kann Eve die Verbindung zum Rendezvousserver unterbrechen und die Anfrage umleiten. Das Endergebnis ist im Wesentlichen das Gleiche wie für den ersteren Fall: Alice denkt eine direkte Verbindung zu Bob zu haben, hat aber eine Verbindung zu Eve die alle Pakete an Bob weiterleitet.



4.2. Erforschen des Lokalen Netzwerkes

WebRTC versucht, ungeachtet der vorherrschenden Netzwerkinfrastruktur, eine direkte Verbindung aufzubauen. Befinden sich zwei Nodes also beispielsweise im selben privaten Netzwerk, so wird zuerst versucht über dieses private Netzwerk eine Verbindung aufzubauen bevor öffentliche Adressen versucht werden oder öffentliche TURN Server ins Spiel kommen. Dies hat zur Folge, dass WebRTC alle lokalen IP Adressen (inklusive der privaten Adressen) per JavaScript zugänglich macht. Zur Ermittlung von öffentlichen Adressen werden externe STUN Server verwendet. Diese Antworten auf Anfragen mit der IP Adresse der anfragenden Node. Auf diesem Wege können Nodes hinter beispielsweise NAT Gateways ihre öffentliche IP Adresse ermitteln.

Codeblock 4 - JavaScript Code Snippet um lokale IP Adressen abzurufen

```
var candidateRegEx = new RegExp(".*typ (host|srflx|relay).*");
var pc = new RTCPeerConnection({iceServers:
[{urls: "stun:stun.l.google.com:19302"}]},
{optional: [{RtpDataChannels: true}]});
pc.createDataChannel("");
pc.createOffer(function(result){
pc.setLocalDescription(result, function(){}, function(){});
}, function(){});
pc.onicecandidate = function(iceCandidate){
if(iceCandidate.candidate){
var c = iceCandidate.candidate.candidate;
console.log("Detected IP " + c.split(" ")[4] + " as " +
candidateRegEx.exec(c)[1]);
}
};
```

Codeblock 4 zeigt ein Codebeispiel in JavaScript wie lokale IP Adressen mittels JavaScript abgerufen werden können. Alles was dazu benötigt wird ist ein *RTCPeerConnection* Objekt, wobei keine Gegenstelle notwendig ist. Mit der Erstellung eines Datenkanals wird das *Offer- Answer* Protokoll gestartet und die IP Adresskandidaten werden gesammelt. Es handelt sich hierbei um sensible Informationen. Angreifer können beispielsweise Abbildungen des lokalen Netzwerks anfertigen mittels *jslanscanner [3]* ohne dabei lokale Adressbereiche erraten zu müssen. Außerdem können diese Informationen dazu verwendet werden um beispielsweise das Browser Exploitation Framework (BeEF) Project [4] zu verbessern. Das Framework kann unter Anderem einzelne Geräte im lokalen Netzwerk identifizieren.

4.3. DDoS Attacken

Um mittels WebRTC eine Verbindung zwischen zwei Nodes herzustellen, verlangt das Protokoll dass Nodes ihre eigene IP Adresse, verwendetes Port und Protokoll an die Gegenstelle senden. Die Verbindungskandidaten des empfangenden und des sendenden Clients werden kombiniert und für den Verbindungsaufbau verwendet. Im Prozess des Verbindungsaufbaus werden bereits Daten übertragen. Wird nun ein kompromittierter Client verwendet, hat dieser die Möglichkeit die übertragenen Verbindungsdaten und Felder wie *ice-ufrag* und *ice-pwd* (siehe Codeblock 3) zu ändern. Ein kompromittierter Client einer Signalling-Domain kann also andere Clients derselben Domain dazu bringen Daten an beliebige IP-Adressen zu übertragen. In aktuellen WebRTC Versionen wurde die unterstütze Länge der Parameter auf 255 Bytes begrenzt, trotzdem bleiben mehrere Parameter die verändert werden können, um einen DDoS Angriff durchzuführen:

```
- Die Anzahl der Verbindungsaufforderungen (Offers) die zeitgleich an die
Gegenstelle gesendet werden.

- Die Anzahl an IP-Adressen (auch wenn mehrere identisch sind) die in jeder
Verbindungsaufforderung enthalten sind.

- Die Wartezeit zwischen versenden von Verbindungsaufforderungen. Die Wartezeit
wurde bei den Versuchen auf eine Sekunde fixiert. Niedrigere Werte haben eine
erhöhte Wahrscheinlichkeit gezeigt, dass die Gegenstelle nicht mehr reagiert hat.
Die anderen beiden Parameter wurden variiert. Die Anzahl der
Verbindungsaufforderungen wurde von 10 bis 30 variiert, die Anzahl an enthaltenen
IP Adressen wurde zwischen 10 und 2000 variiert. Die Tests wurden im Google Chrome
Browser Version 43 durchgeführt. Der Netzwerkverkehr wurde mittels Wireshark
aufgezeichnet. Das Ergebnis ist in Tabelle 1 ersichtlich. Es ist klar ersichtlich,
dass mit steigender Anzahl von Verbindungsanforderungen pro Zeiteinheit und mit
steigender Anzahl von gesendeten IP Adressen, die verwendete Bandbreite bis zu
einem gewissen Level angehoben werden kann. Werden die Werte weiter erhöht, geht
die Bandbreite zurück, was auf interne Browser Limitierungen zurückzuführen sein
dürfte. Von einer einzelnen Node aus können also 0,8 Mbit/s an Daten verschickt
werden.
```

3 https://code.google.com/p/jslanscanner 4 http://beefproject.com/ *Tabelle 1 - Ausgehendes Datenvolumen in kbit/s als Funktion der Anzahl der Verbindungsanforderungen pro Zeiteinheit*

4.4. WebRTC Fingerprinting

Browser Fingerprinting Frameworks versuchen einen Rechner (bzw. den darauf laufenden Browser) eindeutig zu identifizieren. Es kann somit über mehrere Webseiten und über mehrere Aufrufe hinweg, egal ob im privaten oder normalen Modus, eine Sitzung konkret einer Benutzerin bzw. einem Benutzer zugeordnet werden. Die Identifizierung einer Benutzerin bzw. eines Benutzers über mehrere Browser hinweg ist damit im Allgemeinen nicht möglich. Basierend auf Kapitel 4.2 wurde eine Technik entwickelt die auch eine computerweite Identifizierung ermöglicht. Rechner besitzen neben ihrer öffentlichen IP Adresse, die bei jedem Request dem Gegenüber bekannt ist, oftmals auch mehrere lokale IP Adressen. Diese können sein:

- IP Adresse für den Zugriff auf das Lokale Netzwerk, oftmals IPv4 und IPv6

- Zweite lokale IP Adresse, beispielsweise WLAN

- Adresse von virtuellen Adaptern für die Kommunikation mit virtuellen Rechnern (VMWare, VirtualBox,…)

- Adresse um mit angeschlossenen Geräten zu kommunizieren (z.B. Mobiltelefon o.ä.).

Diese Adressen sind einem Außenstehenden generell nicht bekannt, können aber wie in Kapitel 4.2 beschrieben, abgerufen werden. Einzelne Stellen der Adressen werden lokal oftmals zufällig vergeben, bspw. in einem Subnetz welches derzeit noch nicht belegt ist. Eine einzelne Adresse ist demnach kein Garant um einen Rechner eindeutig zu identifizieren. Bündelt man aber die Adressen und überprüft auf Gleichheit von einem Set von Adressen so erhöht sich die Zuordenbarkeit mit jeder weiteren Adresse erheblich. Mit diesem Verfahren können folgende Parameter identifiziert werden:

- Identifizierung eines Rechners über verschiedene Browser hinweg,

- sowie die Kategorisierung von verschiedenen identifizierten Geräten zu Clustern, die sich mit hoher Wahrscheinlichkeit im selben Netzwerk befinden basierend auf den Subnetzen der Privaten IP Adressen und der öffentlichen IP Adresse. Es wurden etwa 80 Samples gesammelt. Davon wurden etwa 97% der Samples korrekt erkannt und dem richtigen Netzwerk zugeordnet. Die restlichen Samples konnten nicht eindeutig zugeordnet werden da sie nur über sehr vereinzelte private IP Adressen verfügten. _Fingerprintjs2_ generierte insgesamt 21 verschiedene Fingerabdrücke für die 8 Benutzerinnen und Benutzer bzw. deren 11 Geräte. Mittels WebRTC Fingerprinting wurden 10 Geräte erkannt, zwei Rechner konnten nicht unterschieden werden, da sie dieselbe öffentliche und eine gleiche private IP Adresse hatten.

5. Schlussfolgerung

In diesem Dokument wurden verschiedenste Auswirkungen von neuen Webtechnologien wie WebRTC auf die Privatsphäre von Benutzerinnen und Benutzern beschrieben. Einerseits

Verbindungsanforderungen pro Zeitinheit

Anzahl an gesendeten IP Adressen	10	20	30
10	130,24	139,87	120,22
100	176,29	302,25	177,86
1000	525,58	***812,75***	794,03
1500	360,20	523,38	427,24
2000	261,35	320,59	278,61

können damit Browser Fingerprinting Frameworks *„verbessert“* und erweitert werden. Damit ist es möglich, basierend auf einem Satz von lokalen IP Adressen die zufällige Komponenten enthalten, einen Rechner zu identifizieren, egal welcher Browser verwendet wird. Die Identifikation eines Browsers kommt in einer mobil-affinen Welt der Identifikation einer Benutzerin bzw. eines Benutzers gleich. Außerdem wurden weitere Möglichkeiten beschrieben, wie WebSockets und im speziellen WebRTC ausgenutzt werden kann um DDoS Attacken durchzuführen. Im Allgemeinen ergeben sich durch neue Web Technologien immer neue und bessere Wege um plattformübergreifende Applikationen zu entwickeln, die einen ähnlichen Funktionsumfang wie native Applikationen aufweisen. Gerade wegen der plattformübergreifenden Unterstützung könnten diese Technologien besonders interessant für Angreifer sein und ein besonderes Augenmerk muss auf sicherheitsrelevante Maßnahmen gesetzt werden.

6. Referenzen

[1] “Giving Web a Memory Cost Its Users Privacy,” 2001. [Online]. Available: http://www.nytimes.com/2001/09/04/business/giving-web-a-memory-cost-its-usersprivacy.html.

[2] E. W. Felten and M. a. Schneider, “Timing Attacks on Web Privacy,” *CCS*, pp. 25–32, 2000.

[3] J. Grossman, “Tracking Users with Basic Auth,” 2007. [Online]. Available: http://jeremiahgrossman.blogspot.co.at/2007/04/tracking-users-without-cookies.html.

[4] T. Frank, “Session variables without cookies,” 2008. [Online]. Available: http://www.thomasfrank.se/sessionvars.html.

[5] R. Lemos, “You don’t know (click)jack,” 2008. [Online]. Available: http://www.securityfocus.com/news/11535.

[6] N. Mohamed, “You Deleted Your Cookies? Think Again,” *WIRED*, 2009. [Online]. Available: http://www.wired.com/2009/08/you-deleted-your-cookies-think-again/.

[7] A. Soltani, S. Canty, Q. Mayo, L. Thomas, and C. J. Hoofnagle, “Flash Cookies and Privacy,” pp. 1–8, 2009.

[8] S. Kamkar, “evercookie,” *2010* . [Online]. Available: http://samy.pl/evercookie/.

[9] “LocalConnection - AS3,” 2015. [Online]. Available: http://help.adobe.com/en_US/FlashPlatform/reference/actionscript/3/flash/net/LocalConnecti on.html.

[10] N. Cubrilovic, “Persistent and Unblockable Cookies Using HTTP Headers,” 2011. [Online].

Available: https://www.nikcub.com/posts/persistant-and-unblockable-cookies-using-httpheaders/.

[11] E. Bursztein, “Tracking users that block cookies with a HTTP redirect,” 2011. [Online].

Available: https://www.elie.net/blog/security/tracking-users-that-block-cookies-with-a-httpredirect.

[12] F. Roesner, T. Kohno, and D. Wetherall, “Detecting and defending against third-party tracking on the web,” *Proc. USENIX Conf. Networked Syst. Des. Implement.*, no. Nsdi, p. 12, 2012.

[13] M. Davidov, “The Double-Edged Sword of HSTS Persistence and Privacy,” 2012. [Online].

Available: http://www.leviathansecurity.com/blog/the-double-edged-sword-of-hstspersistence-and-privacy.

[14] K. Mowery and H. Shacham, “Pixel Perfect : Fingerprinting Canvas in HTML5,” *Web 2.0 Secur. Priv. 20*, pp. 1–12, 2012.

[15] G. Acar, C. Eubank, S. Englehardt, M. Juarez, A. Narayanan, and C. Diaz, “The Web Never Forgets: Persistent Tracking Mechanisms in the Wild,” *Proc. 2014 ACM SIGSAC Conf. Comput. Commun. Secur. - CCS ’14*, pp. 674–689, 2014.

[16] J. Mayer, “How Verizon’s Advertising Header Works,” 2014. [Online]. Available:

http://webpolicy.org/2014/10/24/how-verizons-advertising-header-works/.

[17] J. Erkkilä, “WebSocket Security Analysis.”

[18] H. Saiedian and D. Broyle, “Security vulnerabilities in the same-origin policy: Implications and alternatives,” *IEEE Comput. Soc.*, vol. 44, no. 9, pp. 29–36, 2011.

[19] L. Huang, E. Y. Chen, A. Barth, E. Rescorla, and C. Jackson, “Talking to Yourself for Fun and Profit,” *Proc. W2SP*, pp. 1–11, 2011.

[20] A. Bergkvist, D. C. Burnett, C. 20 Jennings, and A. Narayanan, “WebRTC 1.0: Real-time Communication Between Browsers,” 2015. [Online]. Available: http://www.w3.org/TR/webrtc/. [Accessed: 03-Jun-2015].

[21] J. Rosenberg, dynamicsoft, H. Schulzrinne, and Columbia U., “An Offer/Answer Model with the Session Description Protocol (SDP).” 2002.

[22] J. Rosenberg, R. Mahy, P. Matthews, and D. Wing, “Session Traversal Utilities for NAT (STUN).” 2008.

[23] Y. Mahy, P. Matthews, and J. Rosenberg, “Traversal Using Relays around NAT (TURN):

Relay Extensions to Session Traversal Utilities for NAT (STUN).” 2010.

[24] C. Vogt and M. J. Werner, “BOPlish – Distributed Content Communities Between Browsers based on WebRTC,” 2014.

[25] C. Vogt, M. J. Werner, and T. C. Schmidt, “Leveraging WebRTC for P2P content distribution in web browsers,” in *Network Protocols (ICNP), 2013 21st IEEE International Conference on*, 2013, pp. 1–2.

[26] A. Reiter and T. Zefferer, “POWER : A Cloud-Based Mobile Augmentation Approach for Web- and Cross-Platform Applications,” in *IEEE CloudNet*, 2015.

[27] E. Rescorla, I. RTFM, N. Modadugu, and I. Google, “RFC 6347 - Datagram Transport Layer Security Version 1.2,” 2012.

[28] D. McGrew, Cisco Systems, E. Rescorla, and R. Inc., “Datagram Transport Layer Security (DTLS) Extension to Establish Keys for the Secure Real-time Transport Protocol (SRTP).” 2010.