Per-Circuit TCP-over-IPsec Transport for Anonymous Communication Overlay Networks

Manuel Schneider
ms476@pluto.uni-freiburg.de

Seminar am Lehrstuhl für Kommunikationssysteme
Albert-Ludwigs-Universität Freiburg

1 Einleitung

Das Internet ist zu einem integralen Bestandteil unserer heutigen Gesellschaft herangewachsen. Es bietet die Möglichkeit, beinahe unabhängig von der Distanz, in Millisekunden Informationen auszutauschen. Viele Dienste die durch das Internet zur Verfügung gestellt werden sind nahezu unentbehrlich geworden. Doch die internationale Vernetzung birgt auch Schattenseiten. Die Kommunikation läuft über eine öffentliche Infrastuktur und die zugrundeliegenden Protokolle wurden nicht mit dem Ziel der Anonymität entworfen.

Die Lösung für diese Problematik findet sich in Anonymisierungsdiensten. In diesem Artikel handelt es sich um Anonymisierungsnetzwerke, die als Overlay-Network, implementiert in der Anwendungsschicht, über dem traditionellen Transportprotokoll TCP/IP aufsetzen. Eine Alternative sind Anonymisierungsdienste, die auf VPN basieren. Bei diesen Diensten Verbindet sich der Nutzer per VPN zu einem vertrauenswürdigen Dienstleister und hält somit seine Verbindungsdaten vor den Hosts, zu denen er sich danach verbindet, verborgen. Das Problem bei diesen Diensten ist, dass man vor dem Dienstleister nicht anonym ist. Hingegen ist im Design eines Anonymisierungsnetzwerkes wie zum Beispiel Tor vorgesehen, dass keine Partei die kompletten Verbindungsdaten, bestehend aus Ziel- und Quelladresse kennt.

Tor ist das am weitesten verbreitete Netzwerk zur Anonymisierung der Verbingungsdaten. Es wurde 2003 gestartet und seither ständig weiter entwickelt. Das Netzwerk umfasst mittlerweile ca 6500 Knoten und eine kumulierte Bandbreite von ca 6 GBps[1]. Die Anonymität im Internet, die durch die Verwendung von Tor gewonnen wird, hat allerdings seinen Preis. Das hohe Verhältnis von Nutzern zu Knotenpunkten im Netzwerk treibt die Netzwerkauslastung an ihre Grenzen. Das ist für den Nutzer spürbar durch eine niedrige Bandbreite und hohe Latenzzeiten.

Diese Probleme rühren von einer Schwachstelle im Design, die Joel Reardon und Ian Goldberg in ihrem Artikel "Improving Tor using a TCP-over-DTLS Tunnel" addressiert haben. Mit der Idee des TCP-over-DTLS Tunnel stellen die Autoren einen Ansatz vor diese Probleme zu beheben. Die Lösung hat allerdings einen Haken: Die Autoren verwenden eine Userspace TCP Implementierung, deren Lizenz nicht mit der von Tor vereinbar ist. Mit dem Ansatz "Per-Circuit TCP-over-IPsec Transport" haben es Joel Reardon und Ian Goldberg geschafft die Performanceprobleme im Tornetzwerk unter Kontrolle zu bekommen, ohne auf die Userspace TCP Implementierung angewiesen zu sein.

In dieser Ausarbeitung wird auf das Design und die Evaluierung des "Per-Circuit TCP-over-IPsec Transport" eingegangen und zum Schluss dessen Einflüsse auf die Sicherheit des Tor-Netzewerkes diskutiert.

In Abschnitt 2 werden Grundlagen zur Architektur des Tor-Netzwerkes vermittelt. Der Fokus liegt dabei auf der Kommunikation, speziell dem Verbindungsaufbau im Netzwerk. In Abschnitt 3 wird das von Mashael AlSabah und Ian Goldberg vorgeschlagene Verfahren "Per-Circuit TCP-over-IPsec Transport" detailliert beschreiben. Zusätzlich werden nötigen Kenntnisse zu IPSec erläutert, die für die Verwendung im PCTCP relevant sind. In Abschnitt 4 werden die Veränderungen im PCTCP, bezüglich der Sicherheit kritisch betrachtet und diskutiert.

[1] `http://torstatus.blutmagie.de/#Stats`, Abgerufen am 18.12.2014

2 TOR

Tor ist eine Implementierung des Onion Routings und dient der Anonymisierung seiner Nutzer. Es ist als Overlay Network konzipiert, das heißt, dass die Tor-Infrastuktur auf dem weltweit bestehenden TCP/IP Netzwerk aufbaut. Die Knoten im Netzwerk, im Folgenden Onion Routers genannt, sind von Freiwilligen betriebene Maschinen, auf denen die Tor-Software läuft, welche als Relay konfiguriert wurde. Diese Menge der Onion Router bildet das Tor-Netzwerk. Die Nutzer des Tor-Netzwerkes nutzen die selbe, als Client konfigurierte Software. Im folgenden werden diese Clients Onion Proxies genannt.

Im Tor Netzwerk spielen die Directory Server eine zentrale Rolle. Directory Server sind die Server, über die die Teilnehmner des Netzwerks Informationen über den aktuellen Zustand des Tor-Netzwerkes austauschen. Die Details der Kommunikation zwischen den Directory Servern und den Onion Routern beziehungsweise Onion Proxies werden in dieser Ausarbeitung ausgelassen, da sie für das behandelte Problem nicht relevant sind. Es wird angenommen, dass die Directory Server korrekte und vollständige Informationen aller Onion Router liefern.

Tor bietet abgesehen von der Anonymisierung der Nutzer auch die Anonymisierung der Server. Dieses Konzept wird Hidden Services genannt und ermöglicht mit sogenannten Rendevous-Points im Tor-Netzwerk, dass auch die Verbindungsdaten des Servers unbekannt bleiben können. Doch auch dieser Teil der Tor Software ist irrelevant für PCTCP und soll lediglich zur Vollständigkeit erwähnt werden.

2.1 Design

Wenn ein Onion Proxy eine anonyme Verbindung ins Internet öffnen möchte, benutzt er das Tor-Netzwerk als Proxy. Der Onion Proxy wählt eine Reihe von Onion Routern aus Über die inkrementell ein Pfad, der sogenannte Circuit, erstellt wird. Auf auf diesem Pfad kennen die Onion Routers lediglich ihren Vorgänger und Nachfolger. Der Datenverkehr wird vom Onion Proxy mehrfach verschlüsselt und entlang diesem Pfad gesendet. Die einzelnen Onion Router entschlüsseln, wie die Hüllen einer Zwiebel, nach und nach die Nutzlast der Pakete und geben sie an den nächsten Onion Router bis zur Exit Node weiter. Die Exit Node erstellt nun eine Verbindung zum gewünschten Host. Der Host zu dem die Verbindung erstellt wurde kann nur die Adresse der Exit Node sehen und kein Onion Router auf dem Pfad kennt zugleich Absender- und Empfängeradresse. In den folgenden Abschnitten soll dieser Vorgang im Detail erklärt werden.

2.2 Cells

Die Kommunikation im Tor-Netzwerk wird durch Tor-Datenpakete, sogenannte Cells, erledigt. Es gibt zwei Arten von Cells.

Control Cells werden zur Kommunikation zwischen den Onion Routern verwendet. Sie werden direkt vom Onion Router interpretiert, der sie empfängt und dienen dem Auf- und Abbau der Circuits. Mit den Relay Cells können Streamdaten weitergeleitet, oder das Versenden von Contol Cells in entfernten Onion Routern initialisiert werden.

Relay Cells haben einen zusätzlichen Header, den Relay Header, der Informationen über die weitergeleiteten Daten enthält. Beide Cells haben einen Circuit Identifier, der sie einem Circuit zuordnet, und einen Command im Header. Auf die Circuits wird im folgenden Abschnitt eingegangen. Einige notwendige Cells werden im Abschnitt **??** erläutert, jedoch nicht alle sind für diese Ausarbeitung relevant. Die vollständige Liste der Cells und den genauen Headeraufbau kann in [1] nachgesehen werden.

2.3 Circuits

Konzeptuell sind Circuits die Pfade im Netzwerk entlang welcher die Cells gesendet werden, bevor sie das Tor Netzwerk verlassen. Aus technischer Sicht ist der Circuit eine auf den Onion Routern des Pfades hinterlegte Menge von Regeln die den Fluss der Cells steuern.

Beim inkrementellen Ausbauen des Circuits wird mit jedem Onion Router OR_i ein gemeinsamer symmetrischer Schlüssel K_i ausgehandelt. Zusätzlich bekommt die neue Verbindung einen Circuit

Identifier, welcher auf dieser Verbindung eindeutig ist. Der Onion Router setzt den ausgehandelten Schlüssel mit dem Circuit Identifier in Verbidung, damit er in Zukunft weiß, mit welchem Schlüssel er Pakete die auf diesem Circuit ankommen zu entschlüsseln hat. Wenn ein Circuit um eine Onion Router erweitert wird, wird der dabei entstandene Circuit Identifier wiederum auch mit dem obigen Tupel verbunden. Somit weiß der Onion Router, mit welchem Circuit auf der anderen Verbindung die eingehende Cell gekoppelt ist. Somit handelt es sich hier um Source Routing und keiner der Knotenpunkte auf diesem Pfad kennt gleichzeitig Absender- und Empfängerdaten einer Verbindung die über diesen Pfad läuft.

Üblicherweise werden in der Praxis drei Onion Router für einen Circuit verwendet. Wenn nun eine Cell entlang des Circuits gesendet werden soll erstellt der Onion Proxy eine Relay Cell mit dem passenden Circuit Identifier und verschlüsselt den Relay Header und Nutzlast nacheinander mit den Schlüsseln der Onion Router auf dem Circuit bei der Exit Node beginnend. Der Onion Proxy sendet das Paket anschließend an OR_1. OR_1 empfängt das Paket und entschlüsselt die erste "Zwiebelschicht" des Relay Headers und Nutzlast mit dem Schlüssel der zu diesem Circuit hinterlegt ist. Daraufhin vergleicht er den im Relay Header hintelegten Hashwert der Daten mit dem aktuellen Hashwert der Daten. Wenn der Hashwert übereinstimmt ist die Nutzlast komplett entschlüsselt und wird vom Onion Router verarbeitet. Wenn er nicht übereinstimmt ist die Nutzlast noch verschlüsselt und der Onion Router leitet das Paket an den Cricuit weiter, der mit dem Eingangs Circuit gekoppelt ist. Stimmt der Hashwert am Ende des Circuits nicht überein ist ein fehler aufgetreten und der Circuit wird geschlossen.

Mit dieser Technik haben die Entwickler eine Leaky-Pipe Topology erstellt, die es ermöglicht den Circuit an jedem Knoten zu verlassen. Somit können die Exit Nodes entsprechend den Exit Policies gewählt werden.

Da das Erstellen der Circuits lange dauern kann, werden Circuits auf Vorrat erstellt. Die Tor-Spezifikation sieht auch vor, dass jede Minute ein neuer Circuit verwendet wird, um neue Streams durch das Netzwerk zu routen. Alte Circuits werden geschlossen wenn keine aktiven Streams mehr darüber geleitet werden.

2.4 Streams

Wenn der Circuitaufbau abgeschlossen ist, kann er von den Anwendungen verwendet werden. Wie eingangs erwähnt ist das Netzwerk ein Proxy für den Nutzer. Der Onion Proxy, also die Tor-Software auf dem PC, wird auch so verwendet. Um Tor zu verwenden muss der Onion Proxy als SOCKS Proxy [2] eingerichtet werden.

Um eine Verbindung zu öffnen, fordert die Anwendung den SOCKS Proxy an eine Verbindung zu öffnen. Der Onion Proxy nimmt einen vorrätigen Circuit oder erstellt notfalls einen neuen und sucht sich auf dem Circuit, ensprechend der Exit Policies, einen Onion Router als Exit Node aus. Der Onion Proxy öffnet den Stream, indem er eine Relay Begin Cell, die die Kontaktinformationen enthält, an die Exit Node sendet. Die Exit Node erstellt eine TCP-Verbindung zum gewünschten Host und antwortet mit einer Relay Connected Cell. (Vgl. Abb. 1) Der Onion Proxy berichtet der Anwendung über den erfolgreich erstellten TCP Stream und die Anwedung kann die TCP-Verbindung nun nutzen.

Um den Stream zu schließen kann der Stream analog zu TCP mit einem Handshake geschlossen werden. Dazu sendet eine Partei eine End Relay Cell und die andere Partei bestätigt mit selbiger. Im Falle eines Fehlers kann Der Stream mit einer Teardown Relay Cell sofort geschlossen werden.

2.5 Ciruit construction

Das technische Design einer Verbindung in Tor wird nun anhand des Aufbaus eines sogenannten Cuircuits gezeigt. Bevor eine Verbindung aufgebaut werden kann, muss sich der Onion Proxy zuerst die Informationen über das Tor-Netzwerk bei einem Directory Server holen. Die Details zu diesem Vorgang werden ausgelassen und es wird angenommen, dass der Onion Proxy korrekte und vollständige Informationen über das Tor-Netzwerk hat.

Der Onion Proxy wählt einen Onion Router aus, durch den das Tor-Netzwerk verlassen werden soll. Dieser Onion Router wird auch Exit Node genannt. Die Wahl des Exit Nodes wird Anhand der Exit Policies entschieden. Exit Policies werden über die Directory Server verteilt und sind

Richtlinien, die angeben zu welchen Ports, Hosts oder Netzwerke über die jeweilige Exit Node verbunden werden darf. Von der Exit Node aus sucht sich der Onion Proxy eine Reihe von Onion Routern aus, so dass keiner der Onion Router mehr als ein mal vorkommt.

Circuits werden inkrementell erstellt, das heißt, dass bei der Erstellung dem Circuit ein Onion Router nach dem anderen hinzugefügt wird. Um sich mit dem ersten Knoten zu verbinden, sendet der Onion Proxy eine Control Cell an den ersten Onion Router OR_1. Der Onion Proxy wählt einen Circuit Identifier, der auf dieser Verbindung noch nicht gewählt wurde. Diese Control Cell enthält das Commando Create und zusätzlich die erste Hälfte eines Diffie-Hellman Handshakes als Nutzlast, welche mit dem Public Key des Onion Router verschlüsselt ist. Den Public Key des Onion Routers erhält der Onion Proxy aus den Router Descriptor, welcher er anfangs vom Directory Server erhalten hat. OR_1 verbindet den Circuit mit dem abgeleiteten Schlüssel und bestätigt mit einer Created Control Cell, welche die zweite Hälfe des Diffie-Hellman Handshakes und den gehashten symmetrischen Schlüssel enthält. Mit dem Handshake kann sich der Onion Proxy den symmetrischen Schlüssel ableiten und mit dem gehashten Schlüssel kann letzterer verifiziert werden.

Auf das Diffie-Hellman Schlüsselaustauschverfahren [3] wird hier nicht weiter eingegangen. Es wird angenommen, dass die beiden Kommunikationspartner auf sicherem Wege einen symmetrischen Schlüssel K_1 zur weiteren verschlüsselten Kommunikation ausgehandelt haben. Der Onion Proxy und OR_1 können nun mit K_1 verschlüsselte Relay Cells austauschen.

Um den Circuit um einen weiteren Onion Router OR_2 zu erweitern, sendet der Onion Proxy eine Extend Relay Cell an OR_1. Die Extend Relay Cell enthält die Adresse des neuen Onion Routers OR_2 und die erste Hälfte eines Diffie-Hellman Handshakes als Nutzlast. Wenn schon eine Verbindung besteht, weil schon ein Circuit existiert, der über die beiden Onion Router läuft, wird diese verwendet. Wenn keine Verbindung besteht, erstellt OR_1 eine neue TLS [4] gesicherte TCP-Verbindung zu OR_2. OR_1 packt die Hälfte des Diffie-Hellman Handshakes von OR_1 in eine Create Control Cell und sendet diese an OR_2. Wie im ersten Schritt bekommt OR_1 eine Created Control Cell von OR_2 zurück. OR_1 verpackt die Antwort in eine Extended Relay Cell und leitet sie weiter an den Onion Proxy.

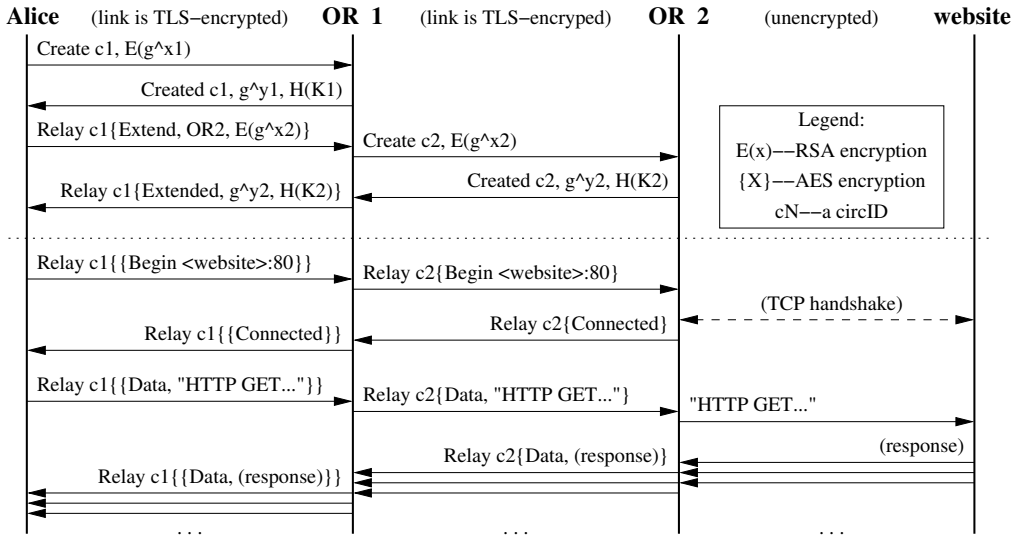


Abbildung 1: Darstellung des Aufbaus und der Verwendung eines Circuits. Quelle: [1]

Wie in Abbildung 1 nachvollzogen werden kann, hat sich der Extend Vorgang nicht geändert. Der Vorgang wurde lediglich vom Onion Proxy entfernt durch Relay Cells erledigt. Wichtig ist, dass der Schlüsselaustausch hier zwischen dem Onion Proxy und OR_2 stattgefunden hat, OR_1 hat die Daten lediglich weitergeleitet. Auch konnte OR_1 keine Informationen gewinnen, da eine Hälfte des Schlüsselaustausches verschlüsselt wurde. Der Circuit ist nun um OR_2 erweitert und der Onion

Proxy und OR_2 können nun mit K_2 wiederum verschlüsselte Relay Cells austauschen. Auf diese Weise können beliebig viele weitere Onion Router zu dem Circuit hinzugefügt werden.

2.6 Cross Circuit Interference Problem.

Unter den vielen Problemen die Tor bis heute noch hat, ist das Cross Circuit Interference Problem eines der konzeptuell schwerwiegensten [5]. Tor leidet unter gewissen Umständen unter hohen Netzwerklatenzen. Für alle Circuits, die über die Verbindung zwischen zwei Onion Router laufen, wird in Tor genau eine TCP-Verbindung verwendet. Das führt dazu, dass alle Circuits dem selben TCP-Mechanismus zugrundeliegen. Joel Reardon fand heraus, dass die Latenzen weder beim Empfangen noch vom Verarbeiten der Daten enstehen. Er stellte fest, dass die Daten die meiste Zeit in den Output Buffern der TCP Implementierung warten und dass die Ursache für die Latenzen die TCP Congestion Control ist, welche Circuits mit niedriger Bandbreite im Vergleich zu Circuits mit hohem Durchsatz benachteiligt [6,7].

TCP garantiert eine fehlerfreie und Ordnung erhaltende Datenübertragung. Dazu wird jedes Paket, das vom Sender versendet wird, vom Empfänger bestätigt. Um unnötige Latenz zu vermeiden, wird dieser Prozess teils parallelisiert, das heißt, der Sender kann Pakete versenden bevor Bestätigungen der vorgehenden Pakete eingegangen sind. Dies muss jedoch begrenzt werden, da unter den bisherigen Bedingungen alle anstehenden Pakete versendet werden könnten, was das Netzwerk temporär überlasten würde.

Um die Last auf dem Netzwerk zu reduzieren wurde beim Sender ein sogenanntes *Congestion Window* (CWND) zur Staukontrolle eingeführt. Das Congestion Window ist ein fiktives Fenster um die Reihe der zu versendenden Pakete, die ohne Bestätigung der Vorgänger versendet werden und somit parallel unterweges sein dürfen.

Teil des Congestion Control Mechanismus ist auch das Verwerfen von Paketen bei zu hoher Auslastung des Netzwerks. Nach einem festgelegten Timeout werden Pakete vom Sender als verworfen angenommen und neu versendet. An dieser Stelle wird nicht tiefer in die Details des Transmission Control Protocols eingegangen, mehr Informationen zum Handling des Congestion Windows und TCP generell kann in den relevanten RFCs gefunden werden [8,9].

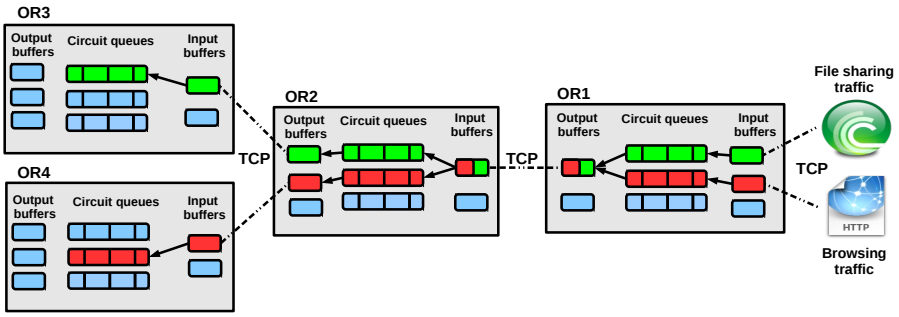


Abbildung 2: Beispielszenario in dem das Cross Circuit Interference Problem auftreten kann. Quelle: [10]

In Abbildung 2 wird ein Szenario dargestellt in dem der Fluss der Daten zweier Circuits über eine TCP-Verbindung vereint wird. Ein Circuit wird verwendet um eine Verbindung zu einer Webseite herzustellen und der andere Circuit wird verwedet um große Datenmengen mit der Filesharing Anwendung Bittorrent zu übertragen. Beide Circuits laufen Über OR1 und OR2. Bei der aktuellen Tor-Architektur bedeutet das, dass die beiden Circuits die TCP-Verbindung zwischen OR1 und OR2 und speziell die Aus- und Eingangspuffer im Kernelspace teilen. Unter Last kann das dazu führen, dass Circuits die relativ wenig Daten fördern, wie der Circuit zum Besuch der Webseite in dem obigen Szenario, verhältnismäßig große Latenzen haben.

Grund dafür ist das Head-on-line Blocking, welches entsteht wenn Pakete verloren gehen beziehungsweise bei Überlastung des Netzwerks verworfen werden. In diesem Fall setzt der Mechanismus der Staukontrolle ein. Der Sender erhält keine Bestätigung und die Pakete werden nach dem festgesetzten Timeout neu übertragen. Wie in Abbildung 3 zu sehen, werden Daten, die längst in Input Buffer zum lesen bereit sind, nicht freigegeben, weil auf ein bestimmtes, in der Sequenz vorhergehendes Paket gewartet wird. Da TCP für das erhalten der Ordnung des Datenstroms zuständig ist, dürfen die Pakete nicht an dei Anwendung weitergegeben werden. Der TCP Spezifikation entsprechend ist das erforderlich, für die Circuits in Tor ist das problematisch und nicht notwendig, da die Ordung schon auf der Streamebene erhalten wird.

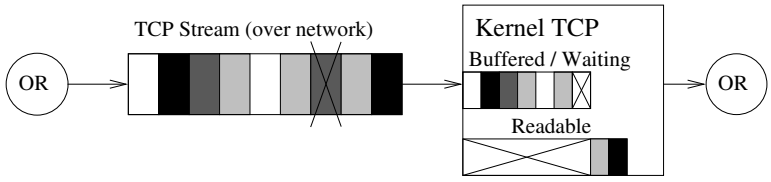


Abbildung 3: Die unterschiedlich schattierten Rechtecke stellen Cells unterschiedlicher Circuits dar. Die letzten zwei Cells sind bereit zur Weitergabe an die Anwendung werden aber zurückgehalten, weil auf das verlorene Paket gewartet wird. Quelle: [7]

3 PCTCP

In dem Artikel in dem Reardon auf diese Schwachstelle hinweist, schlagen die Autoren eine Lösungsansatz vor, in dem das Design dahingehend verändert wird, dass die für Tor unnötige Garantie die Ordung der Datenpakete zu erhalten umgangen wird. Das wir erreicht, indem für die Verbindungen zwischen den Onion Routern UDP-Verbindungen verwendet werden, worauf ein TCP Stack im Userspace aufsetzt. [7]. Die Autoren nennen diesen Ansatz TCP-over-DTLS.

Im Detail bedeutet das, dass für die Verbindungen der einzelnen Circuits jeweils eine eigene TCP-Verbindung aufgebaut wird. Dadurch bleiben alle positiven Eigenschaften der vorherigen TCP-Verbindung erhalten und das Cross Circuit Interference Problem wird gelöst. Grund dafür ist, dass es nicht mehr vorkommen kann, dass mehrere Circuits dem Mechanismus der Staukontrolle einer einzigen TCP-Verbindung unterliegen. Somit besteht keine Möglichkeit der Interferenz mehr, da es in der selben TCP-Verbindung keine Cicuits mehr gibt auf die Einfluss genommen werden kann. Diese TCP-Verbindungen werden aber nicht vom Betriebsystem geöffnet. Nach dem Design des TCP-over-DTLS ist der TCP Stack Teil der Tor-Software. Die Pakete werden im Userspace erstellt und erst dann an das Betriebssystem weiter gegeben um per UDP beziehungsweise DTLS versendet zu werden. Datagram Transport Layer Security ist das UDP Äquivalent zu TLS für TCP [11]. Es sorgt für die Vertraulichkeit und Authentizität auf der Strecke zwischen den einzelnen Onion Routern. Da im Vergleich zu TCP Sockets der UDP Socket nicht gebunden wird, reicht für den TCP-over-DTLS Ansatz genau ein Socket für beliebig vielen Verbindungen zu anderen Onion Routern aus.

Wenn ein Paket verworfen wird, dann wird lediglich die Staukontrolle eines bestimmten Circuits aktiviert, da die Circuits eigenes Congestion Window haben. Alle anderen bleiben unberührt und es kommt zu keiner Cross Circuit Interference. Siehe dazu die graphische Veranschaulichung in Abbildung 4. Des weiteren werden die Pakete, wenn sie verworfen werden, gleichverteilt über den Datenstrom verworfen. Daraus folgt, dass die Wahrscheinlichkeit dass ein Paket verworfen wird, proportional zum Bandbreitenbedarf des Circuits auf der Verbindung steigt und somit die Fairness der Staukontrolle wieder hergestellt ist.

Die Lösung des TCP-over-DTLS scheint bestens geeignet, um die Latenzen im Tor-Netzwerk zu vermeiden. Die Autoren verwenden in ihrem Proof-of-concept allerdings die bereits vorhandene Userpace TCP Implementierung Daytona [12], aber in der Praxis gibt es leider keine Userpace

TCP Implementierung, deren Lizenz mit der des Tor-Projekts vereinbar ist. Daher entschied sich einer der Autoren, Ian Goldberg, einen alternativen Ansatz zu verfolgen, der auf dem selben "Ein-Circuit-pro-TCP-Verbindung" Prinzip des TCP-over-DTLS aufbaut: das Per-Circuit TCP-over-IPsec.

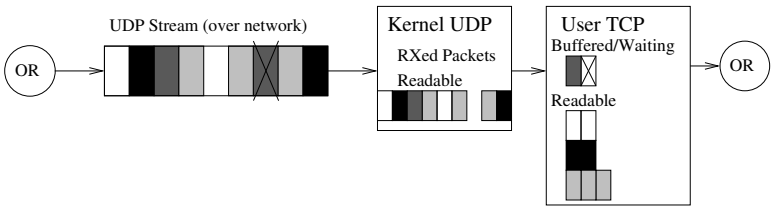


Abbildung 4: Die unterschiedlich schattierten Rechtecke stellen Cells unterschiedlicher Circuits dar. Alle empfangenen Cells werden nun an die Anwendung weitergegeben. Vergleiche Abbildung 3. Quelle: [7]

3.1 Design

PCTCP ist dem TCP-over-DTLS sehr ähnlich. Die Idee wie das Cross Circuit Interference Problem umgangen werden kann ist die selbe. Es wird ganz einfach für jeden Circuit eine neue TCP-Verbindung verwendet, damit die Staukontrolle nicht auf der Ebene der Verbindung zwischen je zwei Onion Router, sondern auf der Ebene der einzelnen Circuits läuft. Somit haben die Circuits keine Möglichkeit sich geegenseitig zu beeinflussen. Konzeptuell wurde das Multiplexing der Circuits also von der Transportschicht auf Vermittlungsschicht verschoben. Im Vergleich zum TCP-over-DTLS wird im PCTCP keine Userspace TCP-Verbindung, sondern eine übliche TCP-Verbindungen im Kernelspace, erstellt. Die Autoren sprechen hier von einer Kernel-Mode per-circuit TCP-Verbindung.

Wenn ein Angereifer einen Onion Router überwacht könnte er so die einzelnen TCP-Verbindungen und somit Circuits zählen und den Datenverkehr analysieren, da die TCP-Header nicht verschlüsselt werden (Vgl. Abbildung 5b). Daher wird die Onion Router zu Onion Router Verbindung per IPSec abgesichert.

Die Verbindung vom Onion Proxy zum ersten Onion Router wird wie gehabt zu erhalten. Es bleibt somit eine einzige TLS gesicherte Verbindung, über die alle Circuits gemutliplext werden, bestehen. Das bedeutet das die Cross Circuit Interference nicht ganz beseitigt wurde, hat aber den Vorteil, dass für die Umsetzung am Quelltext des Clients nichts geändert werden muss. Die Änderung des Designs findet lediglich im Tor-Netzwerk zwischen den Onion Routern statt. Die Netzwerkschnittstelle des PCTCP ist rückwartskompatibel zu Standard Tor Onion Routern. Das ermöglicht PCTCP inkrementell zu verteilen, da der Bertrieb eines herterogeneren Tor-Netzwerkes möglich ist.

3.2 IPSec

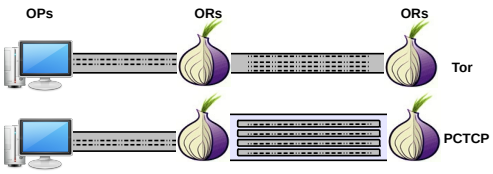
Internet Protocol Security (IPSec) ist eine Protollfamilie die Verbindungen über ein potentiell unsicheres Netzwerk sichert. IPsec soll in einem solchen Netzwerk die Schutzziele Vertraulichkeit, Authentizität und Integrität erhalten. Dazu bietet IPSec zwei Protokolle die in je zwei Modi betrieben werden können: IP Authentication Header [13] und IP Encapsulating Security Payload [14].

Der IP Authentication Header (AH) erlaubt es den kommunizierenden Hosts die übertagenen Daten zu authentifizieren und die Integrität zu prüfen. Optional bietet es Features um Replayattacken zu verhindern. Allerdings kann es nicht für die Vertraulichkeit der Daten sorgen.

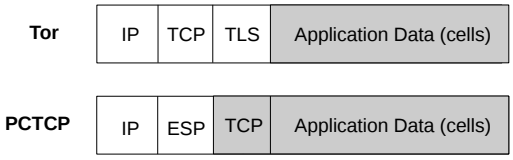
IP Encapsulating Security Payload (ESP) bietet ebenfalls Authentisierung und Integrität der übertagenen Daten. Zusätzlich können mit dem ESP die übertragenen Daten verschlüsselt werden, nachdem mit einem der Schlüsselaustauschverfahren, die auch Teil der IPsec Familie sind, ein gemeinsamer Schlüssel ausgehandelt wurde.

Beide Protokolle können in zwei Modi betrieben werden: Dem Transport- und Tunnelmodus. Im Transportmodus wird die Verbindung zwischen zwei Hosts abgesichert. In diesem Modus sind die beiden Hosts die Endpunkte bezüglich der Kommunikation und Verschlüsselung. Das bedeutet das der Sender und Empfänger selbst für die Verschlüsselung und beziehungsweise oder Authentisierung der Daten zuständig sind. Da der Transport der Daten direkt von Host zu Host läuft, wird der IP Header weiterhin für das Routing benötigt. Der Rest, also ab der Transportschicht wird als Nutzlast des IPSec gefördert.

Im Tunnelmodus werden zwei Netzwerke durch einen IPsec-Tunnel verbunden. Dazu verbinden sich die Gateways zweier Netzwerke miteinander und die Internetzwerk Pakete werden über die IPSec Gateways geleitet. Die Pakete des Datenverkehrs werden von den Gateways samt IP Header gekapselt und mit einem neuen IP Header versehen. Der zusätzliche IP Header ist für das Routing zum gegenüberliegenden Gateway zuständig. Letzteres entpackt das empfangene Paket und leitet es in das eigene Netzwerk weiter. In diesem Modus sind die Endpunkte bezüglich der Verschlüsselung und Kommunikation nicht zwangsweise die selben Instanzen. Für die Sicherheit sind die Gateways zuständig, während die Kommunikationsendpunkte beliebige Teilnehmner aus den beiden Netzwerken sein können.



(a) Vergleich des Designs. Quelle: [10]



(b) Vergleich der Inter Onion Router Pakete. Grau schattierte Teile sind verschlüsselt. Quelle: [10]

Abbildung 5: Vergleiche Zwischen Standard-Tor und PCTCP.

Da das ESP Protokoll alle Fähigkeiten des AH Protokolls und zusätzlich die für Tor wichtige Vertraulichkeit besitzt, wird für die Zwecke im PCTCP das ESP Protokoll verwendet. Zusätzlich würde das AH Protokoll ein praktisches Problem für Tor darstellen, da es nicht ohne weiteres mit NAT [2] vereinbar ist. Grund für die Inkompatibilität ist die Tatsache, dass das AH Protokoll die Integrität des IP Headers überwacht, welches aber während des Prozesses der Network Adress Translation geändert wird. Da Tor ein Netzwerk aus Freiwilligen ist und diese in den meisten Fällen vom Heimnetzwerk aus hinter einem Gateway mit NAT sind, kommt das AH Protokoll nicht in Frage und das ESP Protokoll wird für die Verbindungen verwendet.

Die Verbindungen die mit IPSec gesichert werden sollen, sind übliche Host-to-Host Verbindungen. Das bedeuet, dass die Hosts bezüglich der Verschlüsselung und Kommunikation Endpunkte sind. Folglich wird für die Zwecke des PCTCP das ESP Protokoll im Transport Mode verwendet.

Da IPSec im Transport Mode den TCP Header mitverschlüsselt, kann ein Überwacher eines Onion Routers die Verbindungen nicht mehr unterscheiden, und somit keine Verbindungen mehr zählen oder den Datenverkehr einzelner Verbindungen analysieren. Da das ESP Protokoll bereits für die Vertraulichkeit der Verbindungen sorgt, ist die Verschlüsselung der Verbindungen zwischen den Onion Routern durch SSL/TLS redundant und kann ausgelassen werden. Schliesslich vervollständigt sich das Bild des Designs, das die Autoren des PCTCP vorschlagen. Entsprechend der Namensgebung des Per-Circuit TCP-over-IPsec werden Verbindungen über Tor, die Circuits, in jeweils eigenen TCP-Verbindungen, welche über IPSec gesichert werden, erstellt. Eine graphische Darstellung des Designs befindet sich in Abbildung 5a.

[2] Network Adress Translation [15] ist ein Verfahren, das die Netzwerkadressen in den Headern der IP Pakete verändert, um mehrere Adressen auf eine abzubilden. Sinnvollerweise kommt dieser Mechanismus in Routern zum Einsatz und wird standardmäßig von den Internetgateways (Router) in Privathaushalten verwendet.

3.3 Ciruit construction

Wie auch im Standard Tor verbindet sich der Onion Proxy *OP* mit dem ersten Onion Router OR_1 im Tor-Netzwerk mit einer einzigen TLS-Verbindung. Die beiden Kommunikationspartner haben einen symmetrischen Schlüssel ausgehandelt und können verschlüsselte Relay Cells austauschen. Um den Circuit um einen Onion Router OR_2 zu erweitern, sendet *OP* eine Extend Relay Cell an OR_1. Die Extend Relay Cell enthält die selben Informationen wie im Standard-Tor. Bis hierher gleicht der Prozess exakt dem Standard-Tor Verbindungsaufbau. Um jetzt eine Verbindung zu OR_2 zu erstellen und somit den Circuit zu erweitern, erstellt OR_1 in jedem Fall eine neue, IPSec gesicherte TCP-Verbidung zu OR_2. Standard-Tor versucht hingegegen bestehende Verbindungen wieder zu verwenden. Das Verfahren läuft wie üblich weiter: OR_1 sendet eine Create Control Cell an OR_2, bekommt als Bestätigung eine Created Control Cell von OR_2 zurück und verpackt diese in eine Extended Relay Cell und leitet sie weiter an *OP*.

In Experimenten zeigten AlSabah und Goldberg, dass durch PCTCP eine 60 prozentige Verbesserung der Latenzen und circa 30 prozentige Verbesserung der Downloadzeiten erreicht werden konnten. Die einzelnen Experimente und die genauen Ergebnisse sind in [10] einzusehen.

4 Diskussion

Die Verwendung von Tor bietet keine allumfassende Anonymität. Der Nutzer muss auf einen bewussten Umgang mit Tor achten. Es ist wichtig, dass er genau weiß was Tor bietet und was es nicht bietet. So kann sich der Nutzer immernoch selbst durch sein Verhalten im Internet deanonymisieren. Tor bietet auch keine so genannte Protokoll Normalisierung, das heißt komplexe Protokolle aus höheren Schichten, wie zum Beispiel HTTP, können auch trotz der Verwendung von Tor Informationen über die Identität übertragen, denn Tor arbeitet lediglich auf der Netzwerkschicht. Für diesen Dienst muss auf andere Mittel, wie zum Beispiel den Prokollfilter Privoxy, zurückgegriffen werden.

Für die Arbeit an der Sicherheit definieren die Entwickler von Tor ein Bedrohungsmodell. Es wird angenommen, dass ein Angreifer Teile des Datenverkehrs im Netzwerk überwachen und Knotenpunkte kompromittieren oder selbst betreiben kann. Des weiteren ist er fähig Datenverkehr zu erstellen, modifizieren, verwerfen oder verzögern. In seinem Design bietet Tor keine Sicherheit gegen Ende-zu-Ende Attacken [1]. Unter diesen Annahmen gilt die Anonymität als aufgedeckt, wenn ein Angreifer es geschafft den Datenverkehr am Eingangs- und am Ausgangsknoten eines Circuits mitzuverfolgen. Diese Situation kann genutzt werden um mit Timing-Attacken und Datenverkehrsanalysen Verbindungen zu deanonymisieren.

Die Lösung der Performanceprobleme durch PCTCP, hat ihren Preis. Die Lösung des Head-online Problems wird erreicht, indem für jeden erstellte Circuit eine neue TCP-Verbindung geöffnet wird, was einen für Tor neuen Angriffsvektor einführt: die Socket Exhaustion Angriffe. Mit un Mit Umstaas er

doch auch Die oben genannte Ausgangssituation ist für einen Angreifer erstrebeswert

D

wird m Angreifer versuchen Dieser Umstand ist meist das Ziel Auf diesem Problem Dieses Problem l

...

Hier weiter blubbern, weil der sniper angriff genutzt werden kann um neue circuits zu planen. damit kann solang rerouting forciert werden biss alle connecctions übe rkomprommitierte onions laufen

Abgesehen vom Hauptziel seine Nutzer zu anonymisieren, unterliegt das Design von Tor weiteren Zielen. Tor sieht vor in der realen Welt auf eine einfache Weise eingesetzt werden zu können. Mögliche Hürden, zum Beispiel Kernelpatches oder vermeidbare manuelle Interkation, werden umgangen, so dass für den Benutzer die best mögliche Benutzerfreundlichkeit erreicht wird. Grund dafür ist die Tatsache, dass im Kontext von Tor die Benutzerfreundlichkeit in Korrelation zur Sicherheit steht. Die Benutzerfreundlichkeit beeinflusst direkt die Menge der Nutzer des Systems. In Tor werden die Benutzer hinter der Identität anderer Benutzer versteckt. Je weniger Nutzer also im Netzwerk teilmehmen, desto weniger Anonymität kann dem einzelnen Nutzer geboten werden. **Hier weiter PCTCP ist blöd, weil nicht nutzerfreundlich**

Literatur

1. Dingledine, R., Mathewson, N., Syverson, P.: Tor: The second-generation onion router. In: Proceedings of the 13th Conference on USENIX Security Symposium - Volume 13. SSYM'04, USENIX Association (2004)
2. Leech, M., Ganis, M., Lee, Y., Kuris, R., Koblas, D., Jones, L.: SOCKS Protocol Version 5. RFC 1928, Internet Engineering Task Force (1996)
3. Diffie, W., Hellman, M.: New Directions in Cryptography. IEEE Transactions on Information Theory **22**(6) (1976) 644–654
4. Dierks, T., Rescorla, E.: The Transport Layer Security (TLS) Protocol Version 1.2. RFC 5246, Internet Engineering Task Force (2008)
5. Ringledine, R., Murdoch, S.: Performance Improvements on Tor or, Why Tor is slow and what we're going to do about it. (2009)
6. Reardon, J.: Improving Tor using a TCP-over-DTLS Tunnel. Master's thesis, University of Waterloo, Waterloo, ON (2008)
7. Reardon, J., Goldberg, I.: Improving Tor Using a TCP-over-DTLS Tunnel. Technical report, Centre for Applied Cryptographic Research (CACR) at the University of Waterloo (2009)
8. Information Sciences Institute University of Southern California: Transmission Control Protocol. RFC 793, Internet Engineering Task Force (1981)
9. Allman, M., Paxson, V., Blanton, E.: TCP Congestion Control. RFC 5681, Internet Engineering Task Force (2009)
10. AlSabah, M., Goldberg, I.: PCTCP: Per-Circuit TCP-over-IPSec Transport for Anonymous Communication Overlay Networks. Technical report, Centre for Applied Cryptographic Research (CACR) at the University of Waterloo (2013)
11. Modadugu, N., Rescorla, E.: The Design and Implementation of Datagram TLS. In: NDSS. (2004)
12. Pradhan, P., Kandula, S., Xu, W., Shaikh, A., Nahum, E.: Daytona: A user-level tcp stack. URL http://nms. csail. mit. edu/% 7Ekandula/data/daytona. pdf (2002)
13. Kent, S.: IP Authentication Header. RFC 4302, Internet Engineering Task Force (2005)
14. Kent, S.: IP Encapsulating Security Payload. RFC 4303, Internet Engineering Task Force (2005)
15. Srisuresh, P., Egevang, K.: Traditional IP Network Address Translator (Traditional NAT). RFC 3022, Internet Engineering Task Force (2001)
16. Geddes, J., Jansen, R., Hopper, N.: IMUX: Managing Tor Connections from Two to Infinity, and Beyond. Technical report, Centre for Applied Cryptographic Research (CACR) at the University of Waterloo (2014)