RUB

RUHR-UNIVERSITÄT BOCHUM

GET /out: Automated Discovery of Application-Layer Censorship Evasion Strategies

Henry Schulte
Studiengang: IT-Sicherheit
Matrikelnummer: 108020219383

Exposee – 25. Januar 2023.
Lehrstuhl für Netz- und Datensicherheit.

Supervisor: Prof. Dr. Jörg Schwenk
Betreuer: Dr.-Ing. Vladislav Mladenov

hgi Lehrstuhl für Netz- und Datensicherheit

1 Einleitung

In dieser Seminararbeit wird, basierend auf dem Paper *GET /out: Automated Discovery of Application-Layer Censorship Evasion Strategies* [1] und dem zugehörigen Source Code [2], eine bündige Übersicht der grundlegenden technischen Möglichkeiten der automatisierten Erkennung von Anwendungsschicht basierten Zensur Umgehungsstrategien gegeben. Des Weiteren werden in dieser Seminararbeit die Unterschiede und Vorteile im Vergleich zu Transportschicht basierten Ansätzen herausgestellt. Ein Konzept, welches vorgestellt wird, ist Fuzzer Design, speziell im Umfeld HTTP und DNS. Abschließend werden die Ergebnisse der im Paper beschriebenen empirischen Studie evaluiert und der Nutzen dieser Form der automatisierten Erkennung für reale Anwendungszwecke eingeordnet.

1.1 Motivation

Mit der Verbreitung des Internets entstand in manchen Nationen der Welt ein andauernder technischer Wettbewerb zwischen Zensur des Internets durch den Staat und dem Versuch von Forschern und Aktivisten Umgehungsstrategien dieser Zensurmechanismen zu finden, um ein freies Internet zu ermöglichen [12]. Durch die Möglichkeit diese Umgehungsstrategien neuerdings automatisiert zu entdecken, verschiebt sich dieser Wettbewerb aktuell zugunsten der Forscher und Aktivisten.

Problematisch ist jedoch, dass es bei den gängigen Ansätzen zur automatisierten Entdeckung Schwierigkeiten in der Bereitstellung und Umsetzbarkeit der gefundenen Umgehungsstrategien für Endbenutzer gibt, da diese allesamt Transportschicht orientiert sind. Konkret ist ein Hauptproblem, dass es auf vielen Plattformen nicht möglich ist, TCP/IP Header zu manipulieren, worauf diese Umgehungsstrategien jedoch basieren. Wie sich zeigen wird, lassen sich diese Probleme durch den in dieser Seminararbeit und dem zugrunde liegenden Paper beschriebenen Ansatz vermeiden, da nun nach Strategien gesucht wird, bei denen Anfragen ins Internet nicht mehr in ihrer Paketsequenz also auf dem Level der Transportschicht manipuliert werden, sondern schon in einem Protokoll der Anwendungsschicht, also HTTP oder DNS umstrukturiert werden.

Es mag die Frage aufkommen, warum sowohl etablierte Transportschicht basierte Ansätze als auch der hier behandelte Anwendungsschicht basierte Ansatz überhaupt eine Relevanz für Real-world Szenarien haben, wenn gerade in den letzten

Jahren vermehrt verschlüsselte Protokolle wie HTTPS im Internet eingesetzt werden [13].

Die Verschlüsselung macht gezielte Zensur für Dritte im Netzwerk prinzipiell unmöglich und somit das gezielte Suchen von Zensur Umgehungsstrategien überfällig. Nichtsdestotrotz ist die hier beschriebene Arbeit von hoher praktischer Relevanz, da Staaten in denen Nationale Zensur stattfindet, ein hohes Interesse daran haben, dass weiterhin unverschlüsselte Protokolle im Internet verwendet werden.

So antworten laut der Citizenlab's censorship Test Liste [6] beispielsweise 52% der China-spezifischen Websites nicht auf HTTPS. Zudem wird in diesen Staaten oftmals aktiv versucht verschlüsselte Protokolle zu blockieren oder diese durch Man-in-the-Middle Angriffe [7, 8, 9] zu unterwandern.

1.2 Ähnliche Arbeiten

Es gibt eine Reihe relevanter Publikationen und mit diesen einhergehende Software Projekte mit Bezug zu dem in dieser Seminararbeit erörterten Konzept. Insbesondere sind die gängigen Transportschicht basierten Ansätze zur automatisierten Entdeckung von Zensur Umgehungsstrategien zu erwähnen. Es gibt derzeit drei verschiedene Ansätze: Geneva [3], SymTCP [5] und Alembic [4].

Geneva [3] unterscheidet sich hierbei von den anderen Ansätzen, da es Zensur- und Serverinfrastruktur als Blackbox sieht und im Gegensatz zu SymTCP [5] und Alembic [4] keinen Einblick in die Implementierungsdetails dieser benötigt. Geneva [3] ist insbesondere wichtig, da der hier beschriebene Ansatz auf Geneva [3] aufbaut, beziehungsweise Geneva [3] als Framework nutzt.

Inhaltsverzeichnis

2 Aufbau Zensurinfrastruktur

In diesem Kapitel werde ich grundlegende technische Aspekte des Aufbaus von Zensurinfrastruktur insbesondere im Hinblick auf die in China, Indien und Kasachstan eingesetzten Techniken erläutern.

Während Themen wie Desinformationskampagnen oder das Blockieren jeglicher Internationaler Verbindungen durchaus eine Relevanz im Bezug zu Zensurinfrastrukturen haben, werde ich auf diese jedoch nicht weiter eingehen, da die in diesem Paper vorgestellten Umgehungsstrategien keine Anwendung bei dieser Art der Zensur haben.

Der Fokus liegt hier auf der weit verbreiteten Form der Zensur durch im Netzwerk integrierte Firewalls, sogenannten Middleboxes, welche Netzwerk Pakete nach festgelegten nicht erlaubten Schlüsselwörtern durchsuchen, um dann bei einem Treffer die zugehörige Verbindung zu blockieren [14].

2.1 Übersicht Aufbau Zensurinfrastruktur

Zensur auf staatlicher Ebene wird durch eine Vielzahl an technischen Methoden realisiert. Eine weit verbreitete Form der HTTP und DNS Zensur bilden dabei im Netzwerk eingebettete Firewalls, welche Anfragen per Deep Packet Inspection (DPI) nach verbotenen IP-Adressen und/ oder Schlüsselwörtern in den Header Feldern durchsuchen [1], [11]. Deutlich seltener wird auch die zugehörige Antwort auf Schlüsselwörter überprüft [11], [16].

An dieser Stelle sei gesagt, dass die in der Seminararbeit und dem zugrundeliegenden Paper [1] vorgestellten Strategien nur Filter umgehen können, welche nur Anfragen und nicht Antworten vom Server inspizieren. Solche Filter sind jedoch ohnehin am weitesten verbreitet[1], [11], [16].

Das liegt daran, dass die Strategie vom Client umgesetzt wird und demnach auch nur die Anfrage manipuliert werden kann. Es ist jedoch vorstellbar, wenn nötig, die durch die automatisierte Entdeckung gefundenen Strategien auch auf einem Server zu implementieren, um so gegebenenfalls auch Zensurmechanismen, welche Antworten betrachten, zu umgehen.

Es gibt verschiedene Verfahren, welche zum Einsatz kommen um die Verbindung zu blockieren, wenn ein Filter anschlägt. Die Great Firewall of China injeziert TCP-RST Packete in HTTP Verbindungen, um diese zu beenden [1]. In Indien eingesetzte, staatlich kontrollierte Internet Service Provider senden eine Block-Page als Antwort um HTTP Verbindungen zu beenden [11]. In Kasachstan werden die Pakete der HTTP Verbindung ohne eine Antwort verworfen [1]. Für zu blockierende DNS Anfragen, wird einfach eine falsche IP-Adresse zurückgeliefert [1].

Während es in der Umsetzung des Blockierens der Verbindungen viele Unterschiede gibt, gibt es jedoch eine Gemeinsamkeit im Verhalten der Filter. Schlägt das Parsen, also das Interpretieren, einer Anfrage im Filter fehl, also wird die Anfrage einfach gesagt nicht verstanden, dann wird nicht zensiert und die Anfrage wird durchgelassen. Dieses Prinzip nennt sich Fail-Open. Diese Tatsache werden sich viele der automatisiert gefundenen Umgehungsstrategien zunutze machen.

Ziel ist es also Anfragen zu erstellen, welche vom Filter nicht verstanden werden, aber vom angefragten Server schon, so dass dieser eine Antwort liefern kann.

2.2 Besondere Hürden

Eine besondere Hürde stellt das sogenannte Residual Censorship [17], welches unter anderem in China eingesetzt wird, dar. Damit gemeint ist das Konzept, dass wenn ein Nutzer eine verbotene Anfrage stellt, diese blockiert wird und zusätzlich für einen festgelegten Zeitraum auch jede weitere Anfrage an dasselbe Tripel aus Ziel IP, Server Port und Client Port, unabhängig davon ob der Filter anschlägt, von diesem Nutzer blockiert wird.

Dies erschwert die automatisierte Suche nach Umgehungsstrategien und muss in der Auswertung der Strategie Erfolge mit einkalkuliert werden.

3 Hintergrund von Geneva

In diesem Kapitel betrachten wir den technischen Hintergrund von Geneva. In Geneva wird ein Fuzzer basierter Ansatz genutzt, um variierende Anfragen gegen echte Zensurinfrastruktur zu testen, um so Zensur Umgehungsstrategien automatisiert zu entdecken und zu optimieren. In dem in diesem Kapitel beschriebenen ursprünglichen Geneva Konzept findet die Manipulation der Anfragen nur auf der Ebene der Transportschicht statt, was so viel bedeutet wie, dass einzelne Pakete der Protokolle der Transportschicht in ihrer Reihenfolge getauscht, dupliziert, gelöscht oder anderweitig manipuliert werden. Im Kapitel 4 wird dann beschrieben, wie dieses Konzept angepasst wurde, um Manipulationen auf der Anwendungsschicht zu ermöglichen.

3.1 Konzept Fuzzer

In diesem Abschnitt soll das Prinzip eines Fuzzers beschrieben werden. Fuzzing bezeichnet eine häufig zum Testen von Software eingesetzte Technik, um das Verhalten eines technischen Systems zu prüfen, wenn Eingabewerte deterministisch verändert werden. Simpler ausgedrückt bedeutet dies, dass Eingaben nach einem festgelegten Schema leicht variiert werden und dann geprüft wird, ob ein Fehler, beziehungsweise eine Abweichung von dem erwarteten Verhalten auftritt.

Das Schema, nach welchem Eingabewerte verändert werden, wird Grammatik genannt. Diese ist vor dem Fuzzing Test zu definieren. Zusätzlich muss definiert werden, welches Verhalten erwartet wird, um Abweichungen erkennen zu können.

In Geneva ist die Eingabe, die im Fuzzer verändert wird, der Block von Paketen einer Anfrage. Die Operationen, mit welchen die Pakete verändert werden können, werden in Geneva als Manipulations Primitive bezeichnet. Wie diese aussehen wird in dem nächsten Abschnitt beschrieben.

3.2 Manipulations Primitive

In Geneva gibt es fünf Manipulations Primitive mit denen die einer Anfrage zugehörigen Pakete verändert werden können. Jedes dieser Primitive wird immer auf ein einzelnes Paket angewandt.

Es gibt die Möglichkeit, ein Paket mit *duplicate* zu klonen. Weiterhin kann ein Paket mittels *tamper* verändert werden und mit *fragment* in kleinere Pakete aufgeteilt werden. Für jedes Paket besteht dann die Möglichkeit es mittels *drop* zu verwerfen oder mit *send* abzuschicken.

Diese Operationen benötigen immer Parameter, welche beschreiben, wie das Paket verändert werden soll. Der Parameter {TCP:chksum:corrupt} für das Primitiv *tamper* gibt beispielsweise an, dass die Prüfsumme des TCP Pakets verändert werden soll [2].

Welche Primitive in welcher Reihenfolge auf ein Paket angewandt werden sollen wird in einem sogenannten Action Tree festgehalten. Ein beispielhafter Action Tree ist in Abbildung 3.1, welche ich aus der Github Dokumentation zu Geneva [18] übernommen habe, dargestellt. Die Primitive *duplicate* und *fragment* haben dabei immer zwei Kinder, *tamper* genau eines und *send* und *drop* keines. Zugehörig zu jedem Action Tree ist ein Trigger, dessen Auslösung die im Baum beschriebenen Manipulationen startet.

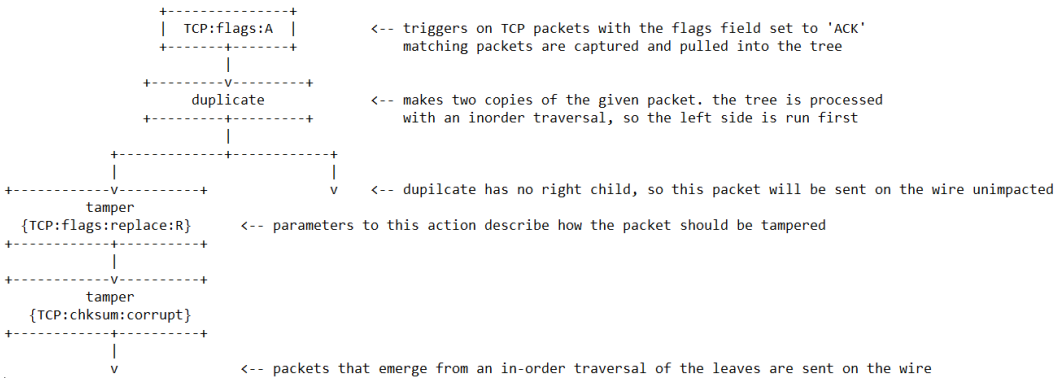

Abbildung 3.1: Action Tree [18]

3.3 Iteratives Konzept

Um den Erfolg der Manipulationen auszuwerten, wird ein iterativer Ansatz in Verbindung mit einem Fitness Score für jede Manipulationsstrategie eingesetzt.

In dem ersten Durchlauf, also der ersten Iteration, werden festgelegte oder zufällig gewählte Strategien gegen die im Netz eingesetzte Zensur getestet. Jeder Strategie wird dann ein Fitness Score basierend auf dem Erfolg der Zensur Umgehung, dem Netzwerk Overhead und der Komplexität der Manipulationsstrategie zugeordnet, wobei Overhead und Komplexität möglichst gering sein sollen, um einen hohen Fitness Score zu erhalten.

Die Strategien mit dem höchsten Fitness Score werden dann als Eingabe für die nächste Iteration genutzt. Sie werden dann gemäß des Fuzzings leicht abgeändert und erneut getestet. Es wird wieder ein Fitness Score zugeordnet und die Besten werden erneut weitergegeben.

So lassen sich letztlich effiziente und erfolgreich gegen echte Zensur getestete Strategien automatisiert entdecken.

4 Anwendungsschicht basierter Ansatz

Wie wir in Kapitel 3 festgestellt haben, basiert das Konzept von Geneva auf der Manipulation einzelner Pakete. Um diesen Ansatz nun auf die Manipulation der Daten der Anwendungsschicht abzuwandeln, werden im Anwendungsschicht basierten Ansatz die einzelnen Bestandteile einer HTTP beziehungsweise DNS Anfrage als Äquivalent zum Paket aufgefasst. Die Bestandteile lassen sich unmittelbar aus der Grammatik von HTTP und DNS herleiten. Weiterhin sind die Manipulations Primitive und das iterative Konzept dem Anwendungsschicht basierten Ansatz angepasst.

4.1 HTTP und DNS Grammatik

Eine HTTP (Version 1.0 und 1.1) [20] Anfrage besteht aus einigen wenigen konstanten Feldern und einer Reihe variabler Header Felder. Die Zeile mit den konstanten Feldern und die einzelnen Header Felder sind durch \r\n voneinander getrennt. Hinter dem letzten Header steht ein doppeltes \r\n, welches den Start des HTTP Body markiert. Die konstanten Felder in der ersten Zeile sind die Methode, beispielsweise GET oder PUT, die Pfadkomponenten, also Pfad, Parameter und Fragment, sowie die HTTP Version. Diese sind durch ein Leerzeichen separiert.

Der Anwendungsschicht basierte Ansatz benutzt einen Parser, welcher die Konstanten Felder und die einzelnen Header eines Requests aufteilt und in einer Liste speichert. Diese fungieren dann als Äquivalent zu den Paketen. Es können nun also Manipulationen auf jedem einzelnen Feld vorgenommen werden.

DNS Anfragen bestehen aus einer Reihe konstanter Header, gefolgt von den zu ermittelnden DNS Records. Während der Standard, spezifiziert in RFC 1035 [19], mehrere angefragte Records in einer Anfrage erlaubt, wird in der Praxis pro Anfrage meist jedoch nur ein Record angefragt [1].

Der Parser des Anwendungsschicht basierten Ansatzes, teilt jede DNS Anfrage in die einzelnen Header-Werte, sowie die einzelnen angefragten Records auf, so dass Manipulationen auf diese Elemente angewandt werden können.

4.2 Veränderte Manipulations Primitive

Da die im Transportschicht basierten Ansatz von Geneva definierten Manipulations Primitive auf der Struktur von Paketen aus Protokollen wie TCP arbeiten, müssen für den Anwendungsschicht basierten Ansatz neue Möglichkeiten zur Manipulation der Daten einer Anfrage konzipiert werden.

Die Primitive *duplicate* und *drop* können in ihrer Funktionalität übernommen werden. Anders sieht es bei dem Primitiv *fragment* aus, welches weg fällt, da eine Unterteilung der Elemente, wie HTTP Header, in kleinere Elemente in der Regel nicht möglich ist. Während die Grundidee des Veränderns von Paketen, welches im ursprünglichen Geneva Ansatz mittels *tamper* realisiert ist, erhalten bleibt, muss die Methodik, mit der Elemente verändert werden können jedoch angepasst werden. Grund dafür ist, dass die HTTP und DNS Anfragen keine klar festgelegte Struktur aufweisen, wie es bei beispielsweise TCP-Paketen der Fall ist. Das Ändern muss also flexibler sein.

Dazu wurden die drei neuen Manipulations Primitive *insert*, *replace*, und *changecase* eingeführt.

Mittels *insert* können neue Bytes in ein Feld eingefügt werden. Dabei kann durch Parameter festgelegt werden, welche Bytes, an welcher Position (Start, Mitte, Ende, zufällig), in welcher Komponente, also zum Beispiel in einem HTTP Header Namen oder Wert, und wie oft diese Bytes eingefügt werden sollen.

Das Primitiv *replace* ersetzt Text an einer definierten Stelle mit den angegebenen Bytes. Die Parameter spezifizieren hier, welche Bytes eingesetzt werden, in welcher Komponente der Text ersetzt werden soll und wie oft die neuen Bytes eingefügt werden sollen.

Zudem kann mittels *changecase* die Groß- und Kleinschreibung gesetzt werden. Ein Parameter legt fest, ob alles groß, alles klein oder jedes Zeichen zufällig groß oder klein geschrieben werden soll.

4.3 Anpassungen am iterativen Konzept

Der Anwendungsschicht basierte Ansatz benutzt weiterhin den genetischen Algorithmus von Geneva. Auch hier wird also wie in Kapitel 3.3 beschrieben, ein Training über mehrere Generationen basierend auf einem Fitness Score, mit kleinen Anpassungen in jeder Generation genutzt. Unterschiedlich ist lediglich die Auswertung einer Strategie in einer Iteration.

Um HTTP Strategien auszuwerten, wird eine Anfrage mit einem verbotenen Host-Header oder einem verbotenen Schlüsselwort in der Anfrage an einen kontrollierten

Server gesendet. Erfolgreich ist eine Strategie dann, wenn sie die angefragte Ressource zurück liefert. Um das bereits angesprochene Problem des Residual Censorship [17] zu lösen, wird nach Misserfolg einer Anfrage ein anderer Server Port in den nächsten Anfragen genutzt, welcher auf dem kontrollierten Zielserver auf den Port unter dem der Server läuft, weiterleitet.

Da bei DNS Anfragen in China die Anfrage nicht blockiert wird, sondern mit einem falschen DNS Eintrag geantwortet wird, muss die Auswertung von DNS Strategien anders erfolgen. Dazu werden Strategien zuerst außerhalb von China an einen erreichbaren DNS Server geschickt. Wenn die Strategie eine Antwort erhält, ist diese gültig und erhält einen höheren Fitness Score. Die gültigen Strategien werden dann in China an einen Server gesendet auf dem kein DNS Server läuft und sollten demnach auch keine Antwort erhalten. Erhalten sie dennoch eine Antwort, so ist klar, dass diese vom Zensor kommt und dieser angeschlagen ist. Der Fitness Score wird dann herab gesetzt.

Zudem erhalten sowohl bei HTTP, als auch bei DNS Strategien, welche gültig sind aber vom Zensor erfasst werden einen niedrigen, aber höheren Fitness Score als Strategien mit ungültigen Anfragen. So wird der genetische Algorithmus dahingehend gelenkt gültige Anfragen zu verfolgen.

4.4 Methodik der Durchführung

Zur einfacheren Benutzung der Strategien wurde ein Proxy im Source Code [3] integriert, mit dem diese ausgeführt werden können.

Als HTTP Server wurden Apache und Nginx genutzt, da diese am weitesten verbreitet sind [21]. Weiterhin wurden eine Reihe von weit verbreiteten DNS Resolvern wie Cloudflare und Google genutzt.

Die Anfragen, umstrukturiert durch die verschiedenen Strategien, wurden aus den Ländern China, Indien und Kasachstan, in denen Zensur aktiv ist, gestellt. Die kontrollierten Server wurden für das Training in Ländern in Europa, in Japan und in den USA aufgestellt, wo keine Zensur aktiv ist.

Zum Aufspüren von HTTP Strategien wurden insgesamt 160 Trainingsdurchläufe mit je 500 Strategie Kandidaten und 50 Iterationen für die verschiedenen Server und Zensur Versionen durchgeführt. Dabei wurden die Server so konfiguriert, dass der verbotene Header, beziehungsweise das verbotene Schlüsselwort in der Anfrage bei dem Server enthalten sein muss, damit Strategien, welche diesen verbotenen Wert einfach entfernen oder verändern nicht als erfolgreich gewertet werden.

Für DNS wurden 40 Trainingsdurchläufe für die verschiedenen DNS Resolver durchgeführt.

Nach jedem Trainingsdurchlauf wurde eine manuelle Analyse durchgeführt, um die gefundenen Strategien zu analysieren und zu verstehen. Zusätzlich wurden, sofern dies der Fall war, Felder welche die Strategie Suche erheblich beeinflusst haben für weitere Trainingsdurchläufe deaktiviert. So wurde sichergestellt, dass die Suche nach Strategien verschiedene Ansätze verfolgt. Wenn in einem Trainingsdurchlauf beispielsweise immer das HTTP Version Feld verändert wurde um eine erfolgreiche Strategie zu entwickeln, sollen in weiteren Durchläufen Strategien gefunden werden, welche nicht das Version Feld manipulieren.

Zusätzlich wurden alle automatisch gefundenen Strategien manuell für verschiedene Server und Zensur Versionen getestet. Dabei wurde festgestellt, dass die Erfolgs Rate im Wesentlichen für unterschiedliche Konstellationen nicht geringer wurde.

5 Evaluation der Umgehungsstrategien

Die Auswertung der Trainingsdurchläufe ergab 77 Klassen von HTTP und 9 Klassen von DNS Strategien. Unter einer Klasse sind verschiedene Varianten von Strategien zu verstehen, welche denselben Ansatz verfolgen und nur in der Umsetzung abweichen. Von den gefundenen HTTP Strategien, umgehen 56 die Zensur in Indien korrekt, 29 die Zensur in Kasachstan und 22 die Schlüsselwort basierte Zensur und 27 die Host basierte Zensur in China.

Da es den Rahmen dieser Seminararbeit sprengen würde alle Strategien vorzustellen, beschränke ich mich auf die Vorstellung von drei HTTP und einer DNS Zensur Umgehungsstrategie. Abschließend ordne ich den vorgestellten Ansatz der automatisierten Zensurumgehung, die damit einhergehenden Forschungsergebnisse und dessen Relevanz ein.

5.1 Vorstellung gefundener Strategien

5.1.1 HTTP

Falscher Wert in dem HTTP Versionsfeld Diese Strategie funktioniert wie folgt. Es wird ein falscher Wert, also eine nicht existente HTTP Version, in das Feld der HTTP Version eingesetzt. Während laut Standard, RFC 7230 [20] solche Anfragen vom Server nicht beantwortet werden sollten, ist dies bei vielen Apache und Nginx Versionen jedoch kein Problem und die angefragte Ressource wird geliefert. In Indien und China umgeht diese Strategie die Zensurmechanismen. In Kasachstan gelingt diese Strategie nur, wenn in das Versionsfeld mindestens 1434 Bytes eingefügt werden. In Abbildung 5.1 ist die Strategie, umgesetzt in einer HTTP Anfrage an eine verbotene Domain, noch einmal bildlich dargestellt.

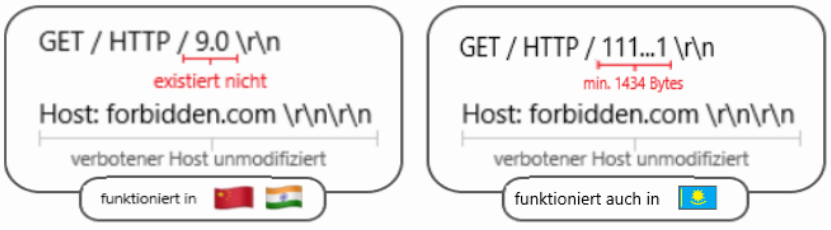

Abbildung 5.1: HTTP Versionsfeld verändert

Damit eine Zensurumgehung mittels dieser Strategie also erfolgreich durchgeführt werden kann, muss ein Anwender seine ausgehenden HTTP Anfragen also so verändern, dass der Wert des Versionsfeld durch einen falschen Wert ersetzt wird. Er kann dies, sofern er ein Linux OS benutzt, direkt über einen in Geneva integrierten Proxy automatisch für alle Anfragen umsetzten lassen. Es ist des Weiteren auch für alle anderen Strategien möglich, diese dem Proxy zur Umsetzung zu übergeben. Für Windows Nutzer werden keine geeigneten Tools zur Umsetzung vorgestellt.

Ändern der Groß- und Kleinschreibung Der Standard in RFC 7230 [20] spezifiziert, dass gewisse Felder in HTTP Anfragen Case Sensitiv seien müssen. Dazu zählt unter anderem die HTTP Methode. Ändert man eines dieser Felder in der Groß- und Kleinschreibung, oder den Namen des Host Headers, zu beispielsweise host (kleingeschrieben), lässt sich die Zensur in Indien umgehen. In Kasachstan und China funktioniert dies nicht.

Falsches Interpretieren des Pfades Diese Strategie basiert darauf, Zeichen von gewisser Bedeutung an einer falschen Stelle im Pfad einer HTTP Anfrage einzufügen. Eine Variante dieser Strategie ist es beispielsweise ein Fragezeichen an den Beginn des Pfades zu setzten. Technisch gesehen müsste der Pfad dann als leer interpretiert werden, da das Fragezeichen den Beginn der Parameter anzeigt. In der Realität, liefern alle getesteten Apache Versionen jedoch dennoch die angefragte Ressource zurück. Zensur in China und Indien erfasst so manipulierte Anfragen nicht.

5.1.2 DNS

Zähler erhöhen Eine Möglichkeit DNS Zensur in China zu umgehen ist es, den Wert von Zählern in den dafür vorgesehenen Feldern zu erhöhen. Beispiele für solche Felder sind *qdcount*, welches die Anzahl der aufzulösenden Adressen enthält, oder *ancount*, welches die Anzahl der Antworten enthält. Den Wert des Felds *qdcount* auf 2 zu erhöhen, obwohl nur eine aufzulösende Adresse vorhanden ist, umgeht die Zensur in China beispielsweise. Allerdings antwortet auf diese Anfrage von den getesteten DNS Resolvern nur Cloudflare. Eine DNS Anfrage in der diese Strategie angewandt wurde ist in Abbildung 5.2 zu sehen.

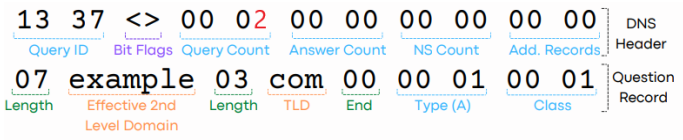

Abbildung 5.2: DNS Anfrage mit falschem Wert im Feld Query Count

6 Fazit und Ausblick

Die gefundenen Strategien zeigen auf, dass Server in der Regel korrumpierte Anfragen besser verstehen und akzeptieren als die eingesetzte Zensur Infrastruktur. Anders gesagt, sind die Zensur Mechanismen im Nachteil, welche nur besonders RFC konforme HTTP und DNS Anfragen verarbeiten können. Damit solche Strategien verhindert werden, müssen Nationen, welche Zensur einsetzen, Geld und Zeit investieren um die eingesetzten Zensurmechanismen auszubessern.

Ein weiteres Problem für diese Nationen, welches sich durch diese automatisiert gefundenen Strategien zeigt, ist es, dass Zensurinfrastruktur größere Buffer zum Speichern der einzelnen Teile einer Anfrage benötigen als die Zielserver, da sonst verbotene Keywords so weit verschoben werden können, dass sie nicht mehr von der Zensur erfasst werden können, aber vom Zielserver schon. Auch hier muss also von Nationen mit aktiver Zensur investiert werden.

Die Automation bietet nun den entscheidenden Vorteil, dass, selbst wenn Staaten die gefundenen Defizite in den Zensurmechanismen beheben, sofern dies überhaupt immer möglich ist, die Möglichkeit besteht, neue Strategien in kurzer Zeit zu entdecken. Wir sehen also, dass sich durch die Möglichkeit zur automatisierten Entdeckung von Zensurumgehung, welche nun auch auf der Ebene der Anwendungsschicht ausgeführt werden kann, der Kampf für ein freies Internet ein Stück weiter zugunsten der Aktivisten verschiebt.

Nun bleibt die Frage, wie auf dieser Entwicklung und den zugehörigen Forschungsergebnissen aufgebaut werden kann. Ich denke, dass nun weitere Schritte in Richtung der Auslieferbarkeit und der tatsächlichen Anwendbarkeit getätigt werden sollten, da dies als Grundidee genannt wurde, warum das ursprüngliche Transportschicht basierte Geneva System auf die Manipulation der Anwendungsschicht basierten Protokolle ausgeweitet werden sollte. Zwar existiert im Source Code [2] ein Proxy, mittels dem die Strategien ausgeführt werden können, dieser ist jedoch nur Linux kompatibel und besitzt kein graphisches Interface. Um eine weitere Verbreitung zu ermöglichen, sollte an diesen Punkten angesetzt werden. Denkbar wäre das Entwickeln einer Browser-Erweiterung, welche die automatisierte Suche nach Strategien selbstständig ausführt und dann gefundene erfolgreiche Strategien unmittelbar anwendet.

Literaturverzeichnis

[1] GET /out: Automated Discovery of Application-Layer Censorship Evasion Strategies. Verfügbar unter: https://www.usenix.org/system/files/sec22-harrity.pdf

[2] Source Code zum im Paper vorgestellten Ansatz. Verfügbar unter: https://geneva.cs.umd.edu/

[3] K. Bock, G. Hughey, X. Qiang, und D. Levin. Geneva: Evolving Censorship Evasion Strategies. Auf der ACM Konferenz zu Computer and Communications Security (CCS), 2019.

[4] S.-J. Moon, J. Helt, Y. Yuan, Y. Bieri, S. Banerjee, V. Sekar, W. Wu, M. Yannakakis, und Y. Zhang. Alembic: Automated Model Inference for Stateful Network Functions. Auf dem Symposium zu Networked Systems Design and Implementation (NSDI), 2019.

[5] Z. Wang, S. Zhu, Y. Cao, Z. Qian, C. Song, S. V. Krishnamurthy, K. S. Chan, und T. D. Braun. SymTCP: Eluding Stateful Deep Packet Inspection with Automated Discrepancy Discovery. Auf dem Network and Distributed System Security Symposium (NDSS), 2020.

[6] CitizenLab. URL Test Liste zum Aufspüren von Website Zensur. https://github.com/citizenlab/test-lists/, 2022.

[7] R. S. Raman, L. Evdokimov, E. Wustrow, A. Halderman, und R. Ensafi. Kazakhstan's HTTPS Interception. https://censoredplanet.org/kazakhstan, 2019.

[8] R. S. Raman, L. Evdokimov, E. Wustrow, A. Halderman, und R. Ensafi. Investigating Large Scale HTTPS Interception in Kazakhstan. Auf der ACM Internet Measurement Konferenz (IMC), 2020.

[9] wkrp. HTTPS MITM auf eine Vielzahl von GitHub IP Adressen in China. https://github.com/net4people/bbs/issues/27, 2020.

[10] Phillipa Gill, Masashi Crete-Nishihata, Jakub Dalek, Sharon Goldberg, Adam Senft, und Greg Wiseman. 2015. Characterizing Web censorship worldwide: Another look at the OpenNet initiative data. ACM Trans. Web 9, 1, Article 4 (January 2015), 29 pages. DOI: http://dx.doi.org/10.1145/2700339

[11] Kushagra Singh, Gurshabad Grover, und Varun Bansal. 2020. How India Censors the Web. In 12th ACM Conference on Web Science (WebSci '20), July 6–10, 2020, Southampton, United Kingdom. ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3394231.3397891

[12] Sadia Afroz, David Fifield, Michael C. Tschantz, Vern Paxson, J. D. Tygar. Censorship Arms Race: Research vs. Practice. 2015. DOI: http://www.icir.org/vern/papers/censor-eval-hotpets2015.pdf

[13] Saptak Sengupta, Liran Tal, und Brian Clark. Web Almanac Chapter 14: Security. 2022. https://almanac.httparchive.org/en/2022/security

[14] F. Li, A. Razaghpanah, A. M. Kakhki, A. A. Niaki, D. Choffnes, P. Gill, und A. Mislove. lib.erate, (n): A library for exposing (traffic-classification) rules and avoiding them efficiently. ACM Internet Measurement Conference (IMC), 2017.

[15] Anonymous, A. A. Niaki, N. P. Hoang, P. Gill, und A. Houmansadr. Triplet Censors: Demystifying Great Firewall's DNS Censorship Behavior. Auf dem USENIX Workshop zu Free and Open Communications on the Internet (FOCI), 2020.

[16] Jakub Dalek, Bennett Haselton, Helmi Noman, Adam Senft, Masashi CreteNishihata, Phillipa Gill, und Ronald J. Deibert. 2013. A Method for Identifying and Confirming the Use of URL Filtering Products for Censorship. In Internet Measurement Conference. ACM. http://conferences.sigcomm.org/imc/2013/papers/imc112s-dalekA.pdf

[17] K. Bock, P. Bharadwaj, J. Singh, und D. Levin. Your Censor is My Censor: Weaponizing Censorship Infrastructure for Availability Attacks. In USENIX Workshop on Offensive Technologies (WOOT), 2021.

[18] Github Seite zu Geneva. Kkevsterrr. https://github.com/kkevsterrr/geneva

[19] P. Mockapetris. Domain names - implementation and specification. RFC 1035, 1987. https://datatracker.ietf.org/doc/html/rfc1035

[20] R. Fielding and J. Reschke. Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing. RFC 7230, 2014. https://www.rfc-editor.org/rfc/rfc7230.html.

[21] Usage statistics of web servers, 2022. https://w3techs.com/technologies/overview/web_server.