Universität Hamburg
Fachbereich Informatik

Bachelorarbeit

Schnittmengenangriffe auf DNS Range Queries

vorgelegt von

Max Jakob Maaß
geb. am 29. August 1990 in Hamburg
Matrikelnummer 6214480
Studiengang Informatik

eingereicht am 8. Oktober 2013

Betreuer: Dipl.-Wirtsch.-Inf. Dominik Herrmann
Erstgutachter: Prof. Dr.-Ing. Hannes Federrath
Zweitgutachter: Dr. Lars Braubach

Aufgabenstellung

Das Domain Name System (DNS) übernimmt im Internet die Umsetzung von Domainnamen in IP-Adressen. Da alle Nachrichten im DNS unverschlüsselt übertragen werden, erlangt ein rekursiver Nameserver (auch als DNS-Resolver bezeichnet) Kenntnis aller von einem Nutzer abgefragten Domainnamen. Da die Nutzer sich gegen diese Form der Überwachung kaum wehren können, stellt diese Praktik einen unerwünschten Eingriff in die Privatsphäre dar.

Zum Schutz vor Beobachtung durch DNS-Server wurde in der Literatur das Verfahren der „Range Queries“ vorgeschlagen [13]. Dabei verschleiert ein Nutzer die von ihm angefragten Domainnamen in einer großen Menge von bedeutungslosen Anfragen (Dummy-Traffic). Bei jeder echten Anfrage wird eine festgelegte Anzahl an zufällig ausgewählten anderen Anfragen an den Nameserver übermittelt. Bislang wurde die Sicherheit des Range-Query-Verfahrens lediglich bei isolierten DNS-Anfragen untersucht: die Wahrscheinlichkeit, bei $N-1$ zufällig gewählten Dummy-Anfragen pro echter Anfrage die wahre Anfrage korrekt zu erraten, beträgt demnach $\frac{1}{N}$.

In dieser Abschlussarbeit soll die Sicherheit des Range-Query-Verfahrens in einem komplexeren Szenario untersucht werden: beim Abruf von Webseiten, welche Bilder bzw. Daten von mehreren (teilweise über 100) Servern einbetten und daher in kurzer Zeit eine Vielzahl von DNS-Anfragen mit unterschiedlichen Domainnamen verursachen (DNS-Query-Bursts). Beispiel: Der Besuch der Webseite von CNN führt nicht nur zu einer DNS-Anfrage für www.cnn.com, sondern auch für i.cdn.turner.com u.a.

Für den Angriff wird unterstellt, dass ein neugieriger Nameserver die Menge der DNS-Anfragen, welche beim Abruf einer Webseite üblicherweise zu beobachten sind, für eine Vielzahl von Webseiten kennt. Gelingt es dem rekursiven Nameserver nun, in einer Abfolge von Range-Queries eine Sequenz von Domainnamen herauszufiltern, die in seiner Datenbank hinterlegt ist, kann er auf die Webseite, die diese Range Queries verursacht hat, schließen.

Im Rahmen der Abschlussarbeit soll ein analytisches bzw. heuristisches Verfahren entwickelt werden, welches den oben skizzierten semantischen Schnittmengenangriff auf Range Queries implementiert. Das Verfahren soll anhand einer im Rahmen der Arbeit durchzuführenden Datensammlung, welche die DNS-Anfragen populärer Webseiten enthält, evaluiert werden.

Zusammenfassung

Das Domain Name System (DNS) ist ein kritischer Bestandteil der Infrastruktur des modernen Internets. Es ermöglicht die Übersetzung von menschenlesbaren Internetadressen (Domains) wie *www.google.com* zu maschinenlesbaren IP-Adressen wie `173.194.70.102`, bietet dabei aber keine Möglichkeit zur Wahrung der Privatsphäre des Nutzers gegenüber neugierigen DNS-Servern.

Fangming Zhao, Yoshiaki Hori und Kouichi Sakurai schlugen in ihrem Artikel „Analysis of Privacy Disclosure in DNS“ [13] das Verfahren des DNS Range Query vor. Dabei wird eine Anfrage an den DNS-Server durch $N-1$ andere, zufällig aus einer möglichst großen Datenbasis gewählte Anfragen verschleiert. Im Rahmen dieser Bachelorarbeit soll ein Angriff auf dieses Verfahren entwickelt und simuliert werden, welcher auf den charakteristischen Mustern, die beim Aufruf einer Webseite mit Daten von mehr als einem Server entstehen, basiert. So führt z.B. ein Aufruf von *www.google.com* unter anderem auch zu einer Anfrage für *ssl.gstatic.com*.

Für diesen Angriff wird zuerst ein Datensatz erstellt, der über 92 000 solcher Muster von bekannten Webseiten enthält. Der Angriff wird zuerst theoretisch entwickelt und analysiert, um dann unter verschiedenen Grundannahmen in einem zu diesem Zwecke entwickelten Simulator getestet zu werden.

Die Simulationen zeigen, dass selbst bei hohen N genaue Rückschlüsse auf die besuchte Webseite möglich sind. Die Privatsphäre des Nutzers kann so mit vergleichsweise wenig technischem Aufwand stark beeinträchtigt werden, das Range Query-Verfahren reicht zur Wahrung der Privatsphäre der Nutzer nicht aus. Abschließend wird ein alternativer Range Query-Algorithmus, welcher auf der Verwendung der gefundenen Muster zur Verschleierung der besuchten Webseite basiert, entwickelt und evaluiert.

Inhaltsverzeichnis

1 Einleitung

Die Verwendung des Internets in seiner heutigen Form setzt fast zwingend die Verwendung des Domain Name Systems (DNS) voraus. Dieses System ist dafür verantwortlich, Domain-Namen wie *www.google.com* in IP-Adressen umzuwandeln, die z.B. für den Abruf einer Webseite nötig sind.

Als das Domain Name System entwickelt wurde, war die Wahrung der Privatsphäre der Nutzer gegenüber eines „neugierigen" DNS-Servers im Entwurf nicht vorgesehen. So ist es einem neugierigen Serverbetreiber problemlos möglich, herauszufinden, welche Internetseiten ein Nutzer besucht. Dies stellt einen starken Eingriff in die Privatsphäre des Nutzers dar, da der Besuch bestimmter Webseiten unter anderem Rückschlüsse auf politische Ausrichtung, Gesundheit oder Religion ermöglichen kann.

Um die Privatsphäre der Nutzer zu schützen, schlugen im Jahr 2007 die japanischen Wissenschaftler Fangming Zhao, Yoshiaki Hori und Kouichi Sakurai in einem Konferenzbeitrag das System des „Range Query" vor. Das System basiert darauf, bei jeder gesendeten DNS-Anfrage $N-1$ weitere, zufällig ausgewählte Anfragen zu versenden. Dadurch, so argumentieren Zhao et al., könne der DNS-Server-Betreiber die korrekte Anfrage nur noch mit einer Chance von $\frac{1}{N}$ erraten.

Beim Aufruf einer Internetseite entstehen charakteristische Anfragemuster. Ein Aufruf von *www.google.com* hat zum Beispiel eine DNS-Anfrage für *ssl.gstatic.com* zur Folge. Diese charakteristischen Muster werden vom so genannten *semantischen Schnittmengenangriff* (im folgenden Schnittmengenangriff genannt) ausgenutzt, der Thema dieser Bachelorarbeit ist.

Im Folgenden wollen wir zuerst eine Übersicht über die Funktionsweise des Domain Name Systems und die Probleme für die Privatsphäre des Nutzers geben. Anschließend stellen wir das DNS Range Query-Verfahren vor, gefolgt von dem Schnittmengenangriff. Darauf folgt eine Übersicht über die Programme, die zur Simulation des Schnittmengenangriffes implementiert wurden, sowie die implementierten Verfahren. Anschließend folgt eine Analyse und Interpretation der Ergebnisse der Simulation sowie eine Diskussion der Ergebnisse. Die Arbeit endet mit einer Übersicht über weitere Forschungsgebiete, die sich durch diese Ergebnisse eröffnet haben.

2 Das Domain Name System

Das Domain Name System (DNS) wird unter anderem zur Auflösung von Domain-Namen zu IP-Adressen verwendet und ist damit ein essentieller Bestandteil der Infrastruktur des Internets.

2.1 Struktur und Funktionsweise

Das DNS basiert auf einer hierarchischen Baumstruktur von Servern. Der oberste Teil der Hierarchie wird von den so genannten „Root-Servern" gebildet, die von der Internet Cooperation for Assigned Names and Numbers (ICANN) koordiniert werden.

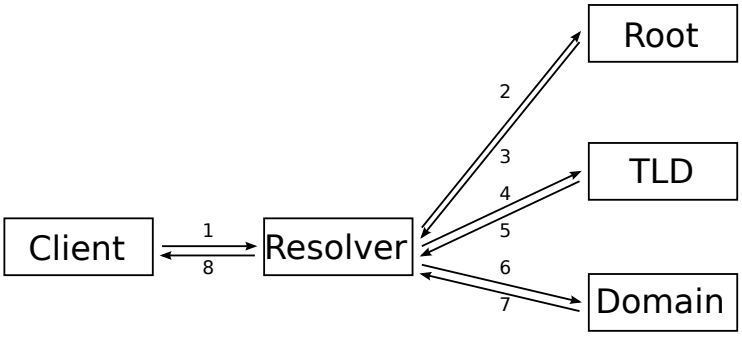


Abbildung 1: Ablauf einer rekursiven DNS-Anfrage

Eine Ebene unter den Root-Servern befinden sich die autoritativen DNS-Server der „Top-Level Domains“ (TLD) wie zum Beispiel *.com* und *.de*. Eine weitere Ebene tiefer befinden sich dann die Server, die für die tatsächlichen Domains, wie zum Beispiel *google.com* oder *web.de*, zuständig sind.

Außerhalb dieser Hierarchie stehen die so genannten „Resolver“. Diese nehmen DNS-Anfragen von Clients an und bestimmen über rekursive Anfragen an andere DNS-Server die IP-Adresse des gesuchten Servers. Die einzelnen Schritte bei der klassischen Namensauflösung (Typ-`A`-Anfragen) sind (siehe Abb. 1):

1. Der Client schickt eine Anfrage an den Resolver.
2. Der Resolver sendet die Anfrage an einen der Root-Server.
3. Der Root-Server antwortet mit der Adresse des zuständigen TLD-Servers.
4. Der Resolver sendet die Anfrage an den TLD-Server.
5. Der TLD-Server antwortet mit der Adresse des zuständigen Domain-Servers.
6. Der Resolver sendet die Anfrage an den Domain-Server.
7. Der Domain-Server antwortet mit dem DNS-Record der Domain, welcher u.a. die IP-Adresse des Zielservers enthält.
8. Der Resolver sendet diesen Record an den Client.

Um die Performance des Systems zu verbessern und die Last auf die autoritativen Server zu verringern, werden Antworten für eine bestimmte Zeit zwischengespeichert („caching“). Wenn innerhalb dieser Zeit eine weitere Anfrage nach dieser Adresse den Server erreicht, so gibt er die zwischengespeicherte Antwort zurück, ohne vorher die Server-Hierarchie zu durchlaufen.

2.2 Probleme für den Schutz der Privatsphäre

Aufgrund der kritischen Rolle, die das DNS in der Struktur des heutigen Internets spielt, gab es bereits viele Studien, die sich mit den Problemen des Systems beschäftigten. Wir beschränken

uns auf die Privacy-Probleme, die durch das DNS entstehen. Andere Probleme wie gefälschte Antworten auf DNS-Anfragen bleiben außen vor.

Die größte Problematik in dieser Hinsicht ist, dass ein neugieriger Resolver-Betreiber in der Lage ist, durch das Überwachen der ankommenden DNS-Anfragen zu erfahren, welche Webseiten seine Nutzer besuchen. Mit diesen Informationen ist es nicht nur möglich, die Interessen und Probleme eines Nutzers zu ermitteln (über den Vergleich der Anfragen mit einer Liste an bekannten und „interessanten“ Domain-Namen wie *www.anonyme-Alkoholiker.de*), sondern es ist auch möglich, über Nutzungsmuster die Nutzer nach dem Wechsel der IP-Adresse wiederzuerkennen, wie Herrmann, Banse und Federrath in ihrem Artikel „Behavior-based tracking: Exploiting characteristic patterns in DNS traffic“ [7] festgestellt haben.

In Artikeln wie „Towards Plugging Privacy Leaks in Domain Name System“ von Yanbin Lu und Gene Tsudik [10] und „Two-Servers PIR Based DNS Query Scheme with Privacy-Preserving“ von Yu Bo und Qu Luo [2] werden nicht nur diese Probleme näher beleuchtet, sondern auch neue DNS-Systeme vorgeschlagen, die laut den Autoren diese Problematiken beheben würden. Allerdings haben fast alle dieser Vorschläge das Problem, dass sie eine alternative DNS-Infrastruktur benötigen.

In dieser Arbeit soll insbesondere der Vorschlag des DNS Range Query von Zhao, Hori und Sakurai bearbeitet werden, der im nächsten Abschnitt behandelt wird.

3 Das DNS Range Query-Verfahren

Die Strategie des DNS Range Query wurde erstmals von Zhao, Hori und Sakurai in dem Artikel „Analysis of Privacy Disclosure in DNS Query“ [13] vorgeschlagen. Es soll, aufbauend auf der aktuellen DNS-Infrastruktur, die Privatsphäre von Internetnutzern gegenüber neugierigen DNS-Resolver-Betreibern sichern.

Prinzipiell ist diese Strategie für jede Form von DNS-Record (*A* für IPv4-Adressen, *AAAA* für IPv6, *MX* für Mailserver, ...) praktikabel. Für den Zweck dieser Arbeit werden wir uns aber auf die Übersetzung von Domain-Namen auf IPv4-Adressen (Record-Typ *A*) beschränken, da dies der normale Anwendungsfall für ein DNS Range Query ist.

Im Folgenden werden wir zuerst die Funktionsweise des Range Query kurz erläutern und die bisherige Forschung zu diesem Thema vorstellen. Ein Beispiel wird in Abschnitt 4.3 gezeigt.

3.1 Funktionsweise

Die Strategie des DNS Range Query sieht vor, dass für jede ausgehende DNS-Anfrage („Query“) noch eine gewisse Anzahl weitere, zufällig aus einer möglichst großen Menge an Domainnamen ausgewählte, Anfragen gestellt werden („Dummy-Anfragen“). Dadurch, so argumentieren die Autoren, hat ein neugieriger DNS-Server-Betreiber nur noch eine Chance von $\frac{1}{N}$, die besuchte Seite korrekt zu bestimmen. Dabei steht N für die Anzahl der verschickten Anfragen, und

wird im Folgenden als „Block-Größe“ bezeichnet. Wenn sämtliche Antworten des DNS-Servers eingetroffen sind, werden alle Dummy-Anfragen verworfen und die gesuchte Antwort an die Anwendung übermittelt, die die Anfrage gestellt hat.

3.2 Bisherige Forschung

Das DNS Range Query wurde schon von verschiedenen Gruppen von Forschern untersucht. Dabei wurden verschiedene Probleme mit dem Verfahren gefunden.

Die Wissenschaftler Castillo-Perez und Garcia-Alfaro beschäftigten sich in ihrer Arbeit „Anonymous resolution of DNS queries“ [3] mit einer angepassten Version des DNS Range Queries, die die Bandbreitennutzung verringern soll, aber neue Probleme in der Vertraulichkeit der Anfragen aufwirft. Die gleichen Wissenschaftler fanden in ihrem Artikel „Evaluation of Two Privacy-Preserving Protocols for the DNS“ [4] eine Schwachstelle, welche den Resolver durch das Unterdrücken von Antworten durch den DNS-Server und das Berechnen der Schnittmengen zwischen den dadurch erneut gestellten Anfragen in die Lage versetzte, die gesuchte Adresse herauszufiltern.

Das DNS Range Query wurde nicht nur auf seine Wirksamkeit, sondern auch hinsichtlich der Performanz untersucht. Dabei stellten Hannes Federrath, Karl-Peter Fuchs, Dominik Herrmann und Christopher Piosecny in einem Konferenzbeitrag mit dem Titel „Privacy-Preserving DNS: Analysis of Broadcast, Range Queries and Mix-based Protection Methods“ [5] fest, dass die Performanz bei der Nutzung von Range Queries stark eingeschränkt wird, mit durchschnittlichen Wartezeiten von über 500 ms bei einer Block-Größe von $N = 50$. Wenn man dies mit der Menge an Anfragen kombiniert, die für die Anzeige einer einzigen Internetseite benötigt werden, kann sich dadurch eine Verzögerung ergeben, die von dem Nutzer als störend empfunden werden kann.

Ein potentieller Angriff auf das DNS Range Query, welcher bisher nicht erforscht wurde, ist der so genannte „Schnittmengenangriff“, der im oben erwähnten Konferenzbeitrag von Federrath et al. [5] erstmals als „semantischer Schnittmengenangriff“ erwähnt wurde. Dieser Angriff ist das Thema dieser Bachelorarbeit und wird im folgenden Kapitel näher behandelt.

4 Der Schnittmengenangriff

Der Schnittmengenangriff macht sich eine Eigenschaft aktueller Webseiten zunutze. Wenn zum Beispiel die Webseite *http://www.google.com/* in einem regulären Web Browser geöffnet wird, erzeugt dies eine DNS-Anfrage für *www.google.com*, allerdings auch für *www.google.de* und *ssl.gstatic.com*.

Grund dafür ist, dass in vielen Webseiten verschiedene Inhalte von verschiedenen Subdomains, Content-Distribution-Networks (CDN) und externen Anbietern wie „Google Analytics“ oder „Google Adsense“ eingebunden werden, und für alle diese externen Inhalte eine eigene DNS-Anfrage benötigt wird.

Die Menge der abgefragten Hostnamen, in diesem Beispiel ⟨*www.google.com*, *www.google.de*, *ssl.gstatic.com*⟩, nennen wir ein **Anfragemuster**. Diese Muster macht sich der Schnittmengenangriff zunutze.

4.1 Angreifermodell

Wir gehen davon aus, dass der DNS-Resolver der Angreifer ist.[1] Der Angreifer im Schnittmengenangriff ist ein strategischer, passiver Angreifer, was bedeutet, dass er in der Lage ist, ankommende Anfragen zu speichern und Algorithmen anzuwenden. Er ist im Besitz einer Datenbank, die für jede Webseite, für die sich der Angreifer interessiert, das dabei zu beobachtende Anfragemuster enthält. Diese Datenbank kann der Angreifer selbst erzeugen. Des Weiteren ist die Datenbasis, aus der der Client seine Anfragen ziehen kann, eine Teilmenge von oder identisch zur Datenbasis des Angreifers, und er weiß, welche Generierungsstrategie der Client verwendet, kann sich also ideal darauf einstellen.[2]

Da es sich um einen passiven Angreifer handelt, ist er nicht in der Lage, Antworten auf Anfragen zu modifizieren oder Anfragen nicht zu beantworten. Des Weiteren ist er komplexitätstheoretisch begrenzt, kann also zum Beispiel Probleme, die nur in Exponentialzeit lösbar sind, nicht in Linearzeit lösen.

Je nach verwendetem Modus des Simulators sind verschiedene Annahmen über die Informationen, die der Angreifer über die Struktur der ankommenden Daten erhält, getroffen worden (vgl. Kapitel 6). Diese Annahmen sind als Teil des Angreifermodells für den jeweiligen Modus anzusehen.

4.2 Funktionsweise

Um die Funktionsweise des Schnittmengenangriffes zu erläutern, muss zuerst das Verfahren des DNS Range Query genau definiert werden. Dieses soll im Folgenden anhand von Abbildung 2 geschehen.

Wenn ein Aufruf von *http://www.google.com/* mit aktivem DNS Range Query erfolgt, wird für jede der drei **wahren** Anfragen, die für die Anzeige der Webseite benötigt werden, ein eigener **Block** erstellt (Schritt 1 in der Abbildung). Anschließend wird jeder dieser Blöcke mit $N-1$ zufällig und ohne Zurücklegen aus der Menge aller bekannten Domain-Namen Q gewählten **Dummy**-Anfragen ergänzt (Schritt 2, in Abbildung 2 war $N = 5$ gesetzt). Als letztes werden die Anfragen innerhalb der einzelnen Blöcke zufällig neu geordnet, damit aus der Position einer Anfrage innerhalb ihres Blocks keine Rückschlüsse gezogen werden können (Schritt 3).

[1] Prinzipiell könnte der Angriff auch von einer dritten Partei, die den Netzwerkverkehr abhört, durchgeführt werden (Man in the middle), es sollte sich kein Unterschied in den Ergebnissen ergeben.

[2] Diese Annahmen erleichtern die Implementation der Simulation erheblich. In Kapitel 9 wird diskutiert, wie realistisch sie sind und was die Folgen wären, sollten sie nicht gegeben sein.

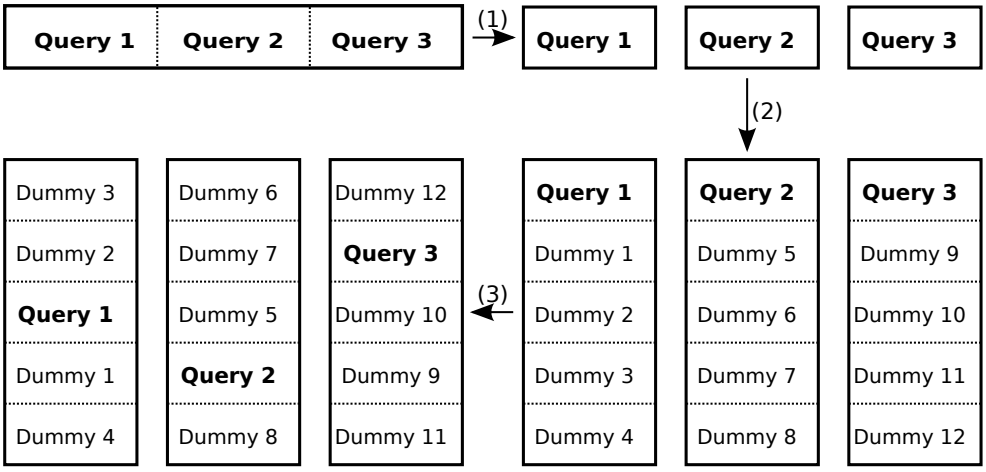


Abbildung 2: Beispiel für die Generierung eines Range Query mit Musterlänge 3, $N = 5$

Um nun die besuchte Webseite zu bestimmen, sucht der Angreifer in diesen drei Blöcken nach den Mustern, die er in seiner Datenbank gespeichert hat. Wenn der Angreifer also das Muster $\langle$*google.com*, *google.de*, *ssl.gstatic.com*$\rangle$ für die Domain *google.com* gespeichert hat, sucht er im einfachsten Fall im ersten Block nach *google.com*, im zweiten Block nach *google.de* und im dritten Block nach *ssl.gstatic.com*.

Dies ist nur der einfachste Fall des Schnittmengenangriffes, andere Strategien und Annahmen werden im Abschnitt 5 diskutiert und evaluiert.

Durch die zufällige Auswahl der Dummy-Anfragen können natürlich auch weitere Muster für andere Seiten entstanden sein. Die Wahrscheinlichkeit dafür sinkt aber mit der Länge der Muster, und die ursprüngliche Ratewahrscheinlichkeit von $p = \frac{1}{N}$ wird auf diese Weise fast nie erreicht werden können.

4.3 Erläuterung des Schnittmengenangriffs am Beispiel

Um das Prinzip zu verdeutlichen, zeigen wir es hier an einem Beispiel mit der Blockgröße $N = 5$. In der Realität sollte ein größeres N gewählt werden, für ein Beispiel würden die Muster dann aber zu unhandlich werden.

Im Folgenden werden Notationen verwendet, die hier einmal formal definiert werden sollen. Um eine Gruppe von Anfragen zu notieren, verwenden wir $\langle$ und $\rangle$ zur Abgrenzung. Eine derart abgegrenzte Gruppe verfügt über die gleichen Operationen wie eine reguläre Menge (es ist zum Beispiel möglich, die Vereinigung $\cup$ zweier solcher Gruppen zu bilden, oder die Kardinalität $|M|$ zu bestimmen). Elemente innerhalb dieser Gruppe sind einzigartig. Die Gruppe ist geordnet, einzelne Elemente können also über ein Subscript (M_n für Element n aus Gruppe M) indiziert werden, wobei der Index des ersten Elements 1 ist.

Wir kennen die Mengen Q als Menge aller bekannten Anfragen sowie P als Menge aller **Musteranfänge** (das erste Element eines Musters, welches die tatsächlich besuchte Webseite angibt,

in unserem Beispiel also *google.com*), kategorisiert nach der Länge des Musters. P_n ist dabei die Menge aller Musteranfänge von Mustern der Länge n.

Das folgende Beispiel lässt sich in zwei Phasen aufteilen: In der ersten Phase (Punkte 1 und 2) generiert der Client die Anfragen, die an den Server geschickt werden. In der zweiten Phase (Punkte 3 bis 6) versucht der neugierige DNS-Server (Angreifer), herauszufinden, welche Webseite besucht wurde. In diesem Beispiel wurde auf das zufällige Ordnen der Anfragen innerhalb der Blöcke verzichtet, da wir die Blöcke als Mengen modellieren, die generell ungeordnet sind. Die Reihenfolge der Anfragen spielt also so oder so keine Rolle.

Als Ziel verwenden wir weiterhin *http://www.google.com*, mit dem Muster $M = \langle$*google.com*, *google.de*, *ssl.gstatic.com*$\rangle$. Prinzipiell könnte aber ein beliebiges Muster aus P gewählt werden.

1. Der Client teilt das Muster M in $|M|$ Teile (in diesem Fall 3). Wir erhalten $S = \langle\langle$*google.com*$\rangle, \langle$*google.de*$\rangle, \langle$*ssl.gstatic.com*$\rangle\rangle$.
2. Für jede Anfrage in S wählt der Client ohne Zurücklegen $N-1$ zufällige Einträge aus Q und fügen sie dem Block hinzu. Wir erhalten $S =$
   $\langle\langle$*google.com*, *umbhululangat.blogspot.com*, *blogs.icemd.com*, *sitti.co.id*, *centrelink.gov.au*$\rangle$,
   $\langle$*google.de*, *img.priceangels.com*, *web.de*, *globeofblogs.com*, *tra.minireklam.com*$\rangle$,
   $\langle$*ssl.gstatic.com*, *stcm.nl*, *whichbudget.com*, *cupid.com*, *img.igl.net*$\rangle\rangle$.
3. Der Angreifer betrachtet die Anfragen aus dem ersten Block:
   - *google.com*: Ist Start eines Musters $\rightarrow$ als Option gespeichert
   - *umbhululangat.blogspot.com*: Ist kein Start eines Musters $\rightarrow$ wird ignoriert
   - *blogs.icemd.com*: Ist kein Start eines Musters $\rightarrow$ wird ignoriert
   - *sitti.co.id*: Ist Start eines Musters $\rightarrow$ als Option gespeichert
   - *centrelink.gov.au*: Ist Start eines Musters, $\rightarrow$ als Option gespeichert

   Es bleibt also die Gruppe an Optionen $MV = \langle$*google.com*, *sitti.co.id*, *centrelink.gov.au*$\rangle$.
4. Der Angreifer betrachtet die Elemente von MV und vergleicht die Längen der Muster der potentiellen Lösungen mit der Länge von S:
   - Das Muster der Option *google.com* besteht aus $A = \langle$*google.com*, *google.de*, *ssl.gstatic.com*$\rangle$. Der Vergleich der Musterlängen ergibt $|A| = 3 = |S|$.
     google.com bleibt also eine Option
   - Das Muster der Option *sitti.co.id* besteht aus $B = \langle$*sitti.co.id*, *google-analytics.com*, *youtube.com*, *s.ytimg.com,woopra.com*, *s.clicktale.net*, *static.woopra.com*$\rangle$. Der Vergleich der Musterlängen ergibt $|B| = 7 \neq 3 = |S|$.
     sitti.co.id ist also keine Option.
   - Das Muster der Option *centrelink.gov.au* besteht aus $C = \langle$*centrelink.gov.au*, *google-analytics.com*, *humanservices.gov.au*$\rangle$. Der Vergleich der Musterlängen ergibt $|C| = 3 = |S|$.
     centrelink.gov.au bleibt also eine Option.

Es bleiben also die Optionen $MV = \langle google.com, centrelink.gov.au \rangle$.

5. Der Angreifer vergleicht die einzelnen Bestandteile der Muster der Elemente von MV mit den Bestandteilen von S. Dabei ist die Reihenfolge nicht entscheidend, es geht nur darum, festzustellen, ob jeder Block von S exakt ein Element des überprüften Musters aus MV enthält.

   - $A_1 \in S_1$,
     $A_2 \in S_2$,
     $A_3 \in S_3 \rightarrow A$ ist komplett in S enthalten, *google.com* ist eine Option.
   - $C_1 \in S_1$,
     $C_2 \notin S_2 \wedge C_2 \notin S_3$,
     $C_3 \notin S_2 \wedge C_3 \notin S_3 \rightarrow C$ ist nicht komplett in S enthalten (sowohl C_2 als auch C_3 sind nicht in S enthalten), *centrelink.gov.au* ist also keine Option mehr.

6. Die Menge der verbleibenden Optionen MV (in diesem Fall $MV = \langle google.com \rangle$) stellen mögliche Ziele dar. Da nur noch ein Ziel verbleibt, können wir mit hoher Sicherheit sagen, dass der Nutzer dieses Ziel besucht hat. Wir haben die Sicherheit unserer Auswahl also von $\frac{1}{N}$ auf $\frac{1}{|MV|}$, in diesem Fall also $\frac{1}{1} = 1$ erhöht (unter der Annahme, dass wir sämtliche Muster der aktiven Domain-Namen des Internets kennen).

Um die Genauigkeit und Erfolgsraten dieser Strategie an einer großen Menge von existierenden Domain-Namen mit verschiedenen Generationsstrategien und Angriffsmethoden zu bestimmen, wurde als Teil der Bachelorarbeit ein Simulator entwickelt, der im Folgenden vorgestellt werden soll.

5 Simulation des Schnittmengenangriffs

Das Simulations-System besteht aus zwei separaten Teilen, die verschiedene Zwecke erfüllen. Beide Programme sind in der Programmiersprache „Python“ [6] geschrieben, und werden in diesem Abschnitt vorgestellt. Dabei beschränken wir uns auf eine oberflächliche Betrachtung der Programme, ohne zu sehr auf die genaue Implementierung einzugehen.

Der Quelltext beider Programme kann unter der URL *https://github.com/Semantic-IA/* abgerufen werden.

Die erste Komponente des Systems ist der **Anfragemuster-Sammler** (`DNSPatternFinder.py`). Dieser ruft eine Liste von Webseiten ab, die ihm als Datei übergeben werden, und speichert die Anfragemuster, die dabei entstehen, in eine andere Datei ab. Er wird in Abschnitt 5.1 genauer beschrieben

Die zweite Komponente ist der tatsächliche **Simulator** (`DRQPatternAttack.py`), der in Abschnitt 5.2 genauer behandelt wird. Er liest die Datei, die der Sammler erstellt hat, und simuliert anschließend das Generieren von Range Queries und die verschiedenen Generations- und Angriffsstrategien, die in den Abschnitten 6 und 7 vorgestellt werden.

5.1 Sammlung der Anfragemuster

Im Folgenden werden wir unsere Lösung zum Problem der Sammlung der Anfragemuster erläutern. Wir beginnen mit der Problemstellung und dem trivialen Ansatz, um dann die tatsächlich gewählte Methode vorzustellen. Des Weiteren behandeln wir Einschränkungen der gewählten Strategie, sowie die verwendeten Eingabedaten. Wir schließen diesen Abschnitt mit einer Analyse der Ergebnisse, die einige Charakteristiken des Datensatzes behandelt.

5.1.1 Problemstellung

Wir benötigen eine Software, die in der Lage ist, Internetseiten aufzurufen und die verwendeten DNS-Anfragen zu registrieren und zu speichern.

Die Software sollte möglichst performant sein und keine graphische Oberfläche benötigen, reguläre Web-Browser sind also keine Option. Außerdem sollte die Software sich möglichst leicht automatisiert betreiben lassen, ein weiterer Grund, der gegen einen regulären Web-Browser spricht.

Ein einfaches Herunterladen des Quellcodes der Webseite mit anschließendem Auslesen aller eingebundenen URLs würde zwar einen großen Anteil der eingebundenen Inhalte abdecken, würde allerdings in speziellen Fällen wie zum Beispiel Internetseiten, die Frames verwenden, nicht ausreichen, da dadurch nur die URL des Frame-Inhaltes, nicht aber die URLs, die im Inhalt des Frames nachgeladen werden, registriert werden würden. Dieser Fall ließe sich zwar explizit einprogrammieren, aber aufgrund der vielen verschiedenen Wege, durch die Inhalte dynamisch nachgeladen werden können, ist ein solcher Ansatz zwar technisch möglich, praktisch allerdings zu aufwendig.

Stattdessen wurde eine andere Lösung gewählt, die im Folgenden vorgestellt werden soll.

5.1.2 Lösungsansatz

Die Lösung, die gewählt wurde, besteht aus zwei Teilen: einem Programm, welches die Internetseiten aufruft und die dabei entstehenden Anfragen bestimmt, und einem zweiten Programm, welches das erste Programm steuert und dessen Ausgabewerte interpretiert.

Der erste Bestandteil ist das Open Source Headless-Webkit-Projekt „PhantomJS“ [8], ein System, welches die Webkit-Browser-Engine ohne graphische Benutzeroberfläche und über JavaScript automatisierbar zur Verfügung stellt. Normalerweise wird dieses Programm zum automatischen Testen von Abläufen auf Internetseiten verwendet, da es aber die von uns benötigten Funktionen bereitstellte, konnte es für unsere Zwecke genutzt werden.

PhantomJS ermöglicht es, eine Funktion einzustellen, die bei jeder geladenen Ressource aufgerufen wird. Die von uns verwendete Funktion extrahiert aus dem JSON-Objekt, welches die Informationen über die Ressource enthält, die Quell-URL. Diese wird dann auf die Domain reduziert (*http://www.example.com/picture.jpg* $\rightarrow$ *www.example.com*).

Außerdem wird ein eventuelles *www.* aus dem Domain-Namen entfernt, bevor es in ein Set von Ergebnissen gespeichert wird. Dies hat den Hintergrund, dass es bei einem Großteil der Internetseiten keinen Unterschied macht, ob ein *www.* vor der URL steht. Durch das Entfernen vermeiden wir gleichzeitig das Problem, dass eine Domain mehr als einmal in ihrem eigenen Muster auftritt (einmal mit *www.*, einmal ohne), was im Simulator im praktischen Teil zu unerwartetem Verhalten führte. Wir haben uns daher entschieden, lieber das Muster um ein Element zu verkürzen als den Code des Simulators unnötig zu verkomplizieren. Es wird sich zeigen, dass eine Verkürzung des Musters in fast allen Fällen zu einer vernachlässigbaren Verschlechterung der Erkennungsleistung des Angreifers führt.

Wenn die Seite fertig geladen ist, wird das Set der verwendeten Domains ausgegeben und das Programm terminiert.

Der zweite Bestandteil des entwickelten Sammlers ist ein Wrapper, der in der Programmiersprache Python geschrieben ist. Dieser dient nur dazu, eine Eingabeliste mit Domain-Namen einzulesen und der Reihe nach dem PhantomJS-Skript zu übergeben. Dabei werden die Informationen, die das Skript liefert, interpretiert und in einem Format in eine Datei geschrieben, welches von dem eigentlichen Simulator verwendet werden kann.

Das Programm ist lizensiert unter der BSD-Lizenz[3] und ist unter *https://github.com/Semantic-IA/DNSPatternFinder* abrufbar.

5.1.3 Einschränkungen

Diese Methode hat einige Einschränkungen. Die wichtigste davon ist, dass PhantomJS keine Plugins wie Adobe Flash oder Java simuliert. Wenn also ein Flash-Objekt normalerweise noch eine weitere Verbindung aufbauen würde, so wird diese von dem Skript nicht erkannt.

Das gleiche gilt für Verbindungen mittels HTML 5. Experimente haben ergeben, dass Verbindungen durch WebSockets nicht erkannt werden, andere Bestandteile von HTML 5 wie Video oder Audio wurden nicht getestet, aber vermutlich ebenfalls nicht erkannt. Da diese Technologien allerdings heutzutage erst von wenigen Webseiten verwendet werden, sollte der Einfluss vernachlässigbar sein.

Moderne Webbrowser verwenden eine Technik namens „DNS-Prefetching“[4], bei der Adressen, auf die die aufgerufene Seite einen Hyperlink gesetzt hat, bereits vorsorglich über das Domain Name System aufgelöst werden. Auch diese Funktionalität ist in PhantomJS nicht enthalten.

Die letzte Einschränkung ist, dass PhantomJS nicht ganz fehlerfrei ist. PhantomJS bietet so genannte Handler für verschiedene Fehlertypen an. Dies sind Funktionen, die aufgerufen werden, wenn ein Fehler auftritt, um darauf reagieren zu können. Es existieren allerdings einige Fehlertypen, die von keinem der von PhantomJS bereitgestellten Handler abgefangen werden. Diese

[3] *http://opensource.org/licenses/BSD-2-Clause*

[4] Mehr Informationen zu dieser Technik sind unter anderem in der Developer Documentation des Chromium-Projekts [11] und in dem Artikel „DNS prefetching and its privacy implications: When good things go bad“ von Krishnan et al. [9] zu finden.

Fehlermeldungen werden daraufhin auf der Konsole ausgegeben und haben dadurch ihren Weg in den Datensatz gefunden. Sie mussten von Hand entfernt werden, und wir können uns nicht sicher sein, ob diese Fehler das Abrufen von weiteren Inhalten verhindert haben.

Es wird sich jedoch herausstellen, dass die Ergebnisse des Simulators mit steigender Länge des Anfragemusters in fast allen Fällen besser werden. Daher sorgen eventuelle fehlenden Anfragen in den meisten Fällen eher für schlechtere als für bessere Genauigkeiten bei dem Schnittmengenangriff.

5.1.4 Verwendete Eingabedaten

Wir haben uns entschieden, als Datenbasis die ersten 100 000 Seiten der Alexa Toplist [1] zu verwenden. Diese Liste wurde schon in verschiedenen anderen Untersuchungen verwendet, zum Beispiel dem Artikel „On the Incoherencies in Web Browser Access Control Policies“ von Singh et al. [12]. Die Toplist wurde am 22. März 2013 abgerufen, Veränderungen nach diesem Datum beeinflussen unsere Daten also nicht mehr.

Diese Liste wurde anschließend modifiziert, um die Ranglisten-Platzierung und eventuelle Unterverzeichnisse aus der URL zu entfernen (*www.youtube.com/UniTVHamburg* wurde also zu *www.youtube.com* verkürzt), um Duplikate zu vermeiden, wenn von einer Seite mehrere Unterverzeichnisse in der Liste vorhanden waren.

Wir haben diese Quelle gewählt, weil es eine offen zugängliche Liste an Internetseiten ist, die ihrer von Alexa gemessenen Popularität nach geordnet werden. Da die Platzierung verschiedener Seiten in der Rangliste in unserem Anwendungsfall nicht entscheidend ist, können berechtigte Zweifel an der Richtigkeit der Rangordnung unbeachtet bleiben.

Die genaue Liste kann aus lizenzrechtlichen Gründen nicht veröffentlicht werden.

5.1.5 Ergebnisse

Der Sammler wurde auf einem Server mit Gigabit-Anbindung und Dual Core-Prozessor ausgeführt und benötigte etwas über eine Woche. Der Hauptgrund für diese lange Zeit war, dass aus einem nicht näher bestimmbaren Grund PhantomJS in manchen Fällen nicht reproduzierbare Deadlocks zeigte, wodurch nach und nach alle 10 parallelen Prozesse die Arbeit einstellten.

Aus uns nicht näher bekannten Gründen war es dem Sammler außerdem nicht möglich, Muster für 7 111 verschiedene Internetseiten zu bestimmen. Diese Adressen wurden nach wiederholten erfolglosen Versuchen aus dem Datensatz entfernt, wodurch 92 889 Muster im Datensatz verblieben.

Dieser Datensatz kann unter *https://github.com/Semantic-IA/data/tree/master/Pattern%20database* heruntergeladen werden. Die ersten 5 Zeilen des Datensatzes wurden in Abbildung 3 abgedruckt, wobei einige Zeilenumbrüche zur besseren Lesbarkeit eingefügt wurden.

Im Folgenden wollen wir noch eine kurze Analyse des Datensatzes vornehmen, bevor wir uns dem Simulator widmen.

```
google.com:google.de,google.com,ssl.gstatic.com
facebook.com:facebook.com,fbstatic-a.akamaihd.net,
fbexternal-a.akamaihd.net,fbcdn-dragon-a.akamaihd.net
youtube.com:i1.ytimg.com,s0.2mdn.net,clients1.google.com,
i4.ytimg.com,youtube.com,s.ytimg.com,csi.gstatic.com,
pagead2.googlesyndication.com,i3.ytimg.com,accounts.google.com,
i2.ytimg.com,ad-g.doubleclick.net,gstatic.com,
ad-emea.doubleclick.net
yahoo.com:yahoo.com,bs.serving-sys.com,yahoo.ivwbox.de,
l1.yimg.com,csc.beap.bc.yahoo.com,ad.yieldmanager.com,
cm.g.doubleclick.net,cct.o2online.de,ads.yimg.com,
cookex.amp.yahoo.com,l.yimg.com,b.scorecardresearch.com,
servedby.flashtalking.com,row.bc.yahoo.com,de.yahoo.com,
portal.o2online.de
baidu.com:baidu.com
```

Abbildung 3: Die ersten 5 Zeilen des Datensatzes

5.1.6 Analyse des Datensatzes

Der Datensatz enthält $|P| = 92\,889$ Muster und $|Q| = 216\,925$ Anfragen (wobei $P \subset Q$ gilt). Dabei beträgt die durchschnittliche Musterlänge („mean value“[5]) 13.02093, gerundet auf ganze Zahlen also 13 (was wir später in der Analyse der Algorithmen benötigen werden). Dabei berechnet sie sich *nicht* aus $\frac{|Q|}{|P|}$, da Elemente aus Q in mehr als einem Muster auftreten können. Stattdessen werden die einzelnen Musterlängen addiert und durch $|P|$ geteilt.

Die Varianz[6] der Musterlängen beträgt 204.0364, die Standardabweichung[7] beträgt 14.2841. Dies bedeutet, dass ein großer Teil der Muster eine Länge zwischen 1 und 27 hat, ein Schluss, der durch die Verteilung der Musterlängen, die in Abb. 4 und 5 visualisiert wurde, unterstützt wird. Dabei weist das kürzeste Muster eine Länge von 1 auf, während das längste Muster eine Länge von 315 erreicht.

Das Verhältnis zwischen der Anzahl der *Muster* und der Anzahl der *Anfragen* beträgt $v = \frac{|P|}{|Q|} = \frac{92\,889}{216\,925} \approx 0.4282$. Eine zufällig aus Q ausgewählte Anfrage ist also mit einer Wahrscheinlichkeit von etwa 42.82% der Beginn eines Musters.

Eine weitere interessante Statistik ist der Jaccard-Koeffizient. Er ist definiert als $J(A,B) := \frac{|A \cap B|}{|A \cup B|}$, teilt also die Kardinalität der Schnittmenge zweier Mengen durch die Kardinalität der Vereinigung. Dadurch erhält man eine Kenngröße für die Ähnlichkeit zweier Mengen (Wie viele Ele-

[5] $E(X) := \frac{\sum_{x \in X}(x)}{|X|}$

[6] $Var(X) = \sigma^2 := \sum_{x \in X} (x - E(X))^2$

[7] $stdev(X) = \sigma := \sqrt{\sigma^2} = \sqrt{Var(X)}$

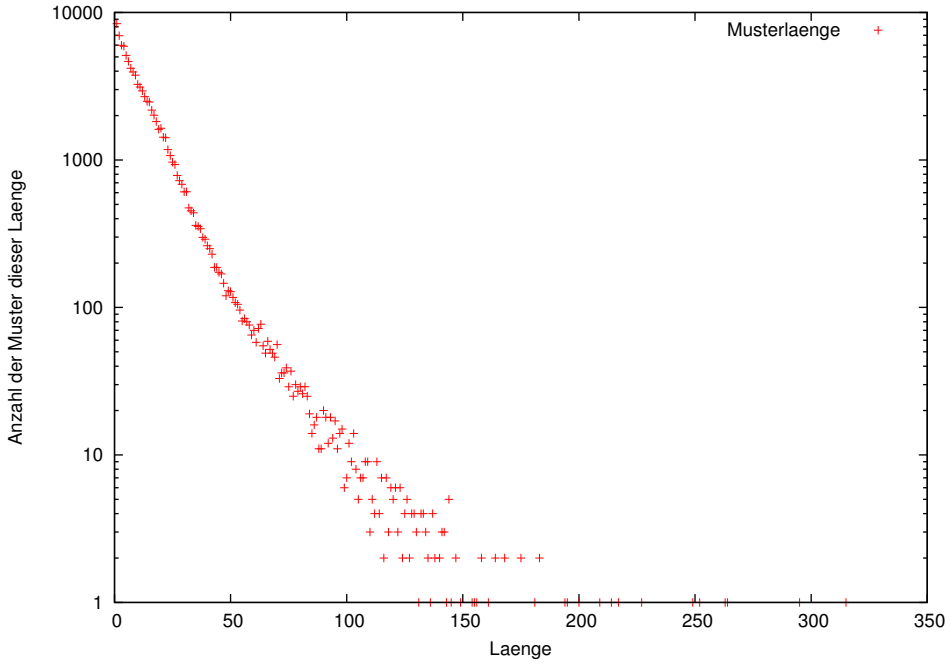


Abbildung 4: Verteilung der Musterlängen (auf logarithmischer Y-Achse)

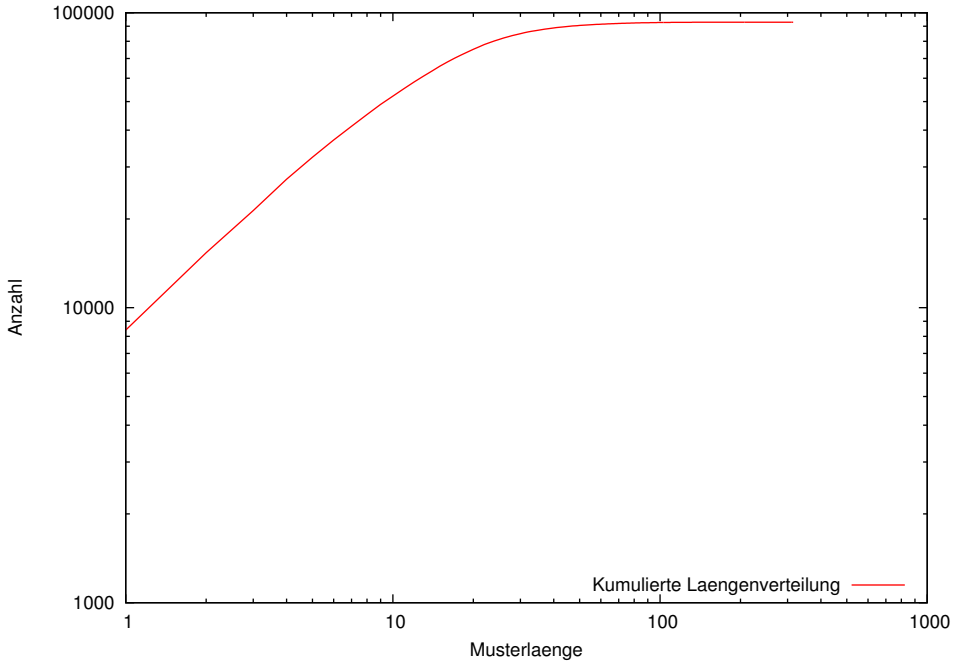


Abbildung 5: Kumulierte Verteilungsfunktion der Musterlängen (auf doppelt logarithmierten Achsen)

mente zweier Mengen übereinstimmen). Die kleinstmögliche Übereinstimmung ist 0, die größtmögliche ist 1.

Der Jaccard-Koeffizient ist eine interessante Messgröße, da eine hohe Ähnlichkeit von Mustern untereinander (also eine hohe Übereinstimmung der enthaltenen Anfragen) die Wahrscheinlichkeit erhöht, dass zufällig weitere Muster auftreten. Wenn also z.B. *google-analytics.com* als zufällige Anfrage gezogen wird, kann diese Anfrage bis zu 61 273 Muster (die Anzahl der Muster, die diese Anfrage enthalten) vervollständigen.[8]

Für die Analyse des Datensatzes haben wir den Jaccard-Koeffizienten von allen Mustern mit jedem anderen Muster berechnet. Da dies zu einer recht großen Menge an Daten führt ($|P|^2 = 8\,628\,366\,321$)[9], wurden die Ergebnisse doppelt aggregiert: Zuerst wurden alle Jaccard-Koeffizienten eines Musters zusammengefasst und einige Kenngrößen von dieser Menge an Ergebnissen berechnet. Anschließend wurden diese aggregierten Kenngrößen für alle Muster der gleichen Länge erneut zusammengefasst und aggregiert. Der Wert der Ausgabe an der Stelle 30 ist also die aggregierte Form der Kenngrößen aller Muster der Länge 30, welche wiederum die aggregierten Werte der Jaccard-Koeffizienten dieser Muster mit allen anderen Mustern sind.

Als Ergebnis erhalten wir einen Datensatz, der *Maximum*, *mean* (Durchschnitt), *median* (Mittelwert), sowie das *1.* und *3. Quartil* (*Q1, Q3*) angibt. Des Weiteren wird die Standardabweichung dieser Werte innerhalb der zusammengefassten Datensätze der gleichen Länge angegeben. Das Minimum wurde nicht beachtet, nachdem sich herausstellte, dass es durchgehend den Wert Null annahm.

Um die Übersichtlichkeit zu gewährleisten, wurde nur ein Teil dieser Daten hier abgedruckt, die vollen Ergebnisse können unter *https://github.com/Semantic-IA/data/blob/master/Pattern%20database/jaccard.txt* abgerufen werden.

Die Visualisierung dieser Daten stellte sich als schwieriger als erwartet heraus. Die ursprünglich geplante Darstellung von mean, Q1 und Q3 mit Standardabweichung als Fehlerbalken stellte sich als zu unübersichtlich heraus, weswegen eine Darstellung ohne Fehlerbalken gewählt wurde. Diese Darstellung ist in Abb. 6 zu sehen.

Wie in der Abbildung zu sehen ist, liegt das Maximum der Jaccard-Koeffizienten bei einer Musterlänge von 20-30, mit einem mean von ca. 0.055 und einem 3. Quartil von ca. 0.081. Das unruhige Funktionsbild im Bereich von 50 bis 150 lässt sich auf die vergleichsweise geringe Anzahl an Datensätzen zurückführen, da bei einer kleinen Anzahl an Datenpunkten einzelne Ausreißer große Auswirkungen auf den Durchschnitt und die Quartile haben können.

Ab einer Musterlänge von 150 sind für viele Musterlängen überhaupt keine Muster im Datensatz vorhanden (vergleiche auch Abb. 4), daher wird das Bild der Verteilung wieder ruhiger.

Der sinkende Jaccard-Koeffizient bei Musterlängen von über 20-30 hat mehrere Gründe. Zum einen ist der Jaccard-Koeffizient eines langen Musters (oder allgemeiner, einer großen Menge)

[8] Dies bedeutet übrigens, dass 61 273 der 92 889 Webseiten im Datensatz Google Analytics verwenden. Zum Vergleich: Facebook-Buttons enthalten nur 27 341 der Webseiten.

[9] Die Anzahl der zu berechnenden Jaccard-Koeffizienten ist aufgrund der Symmetrie ($J(A,B) = J(B,A)$) geringer, da allerdings $J(A,B)$ sowohl für A als auch für B relevant ist, müssen trotzdem $|P|^2$ Werte beachtet werden.

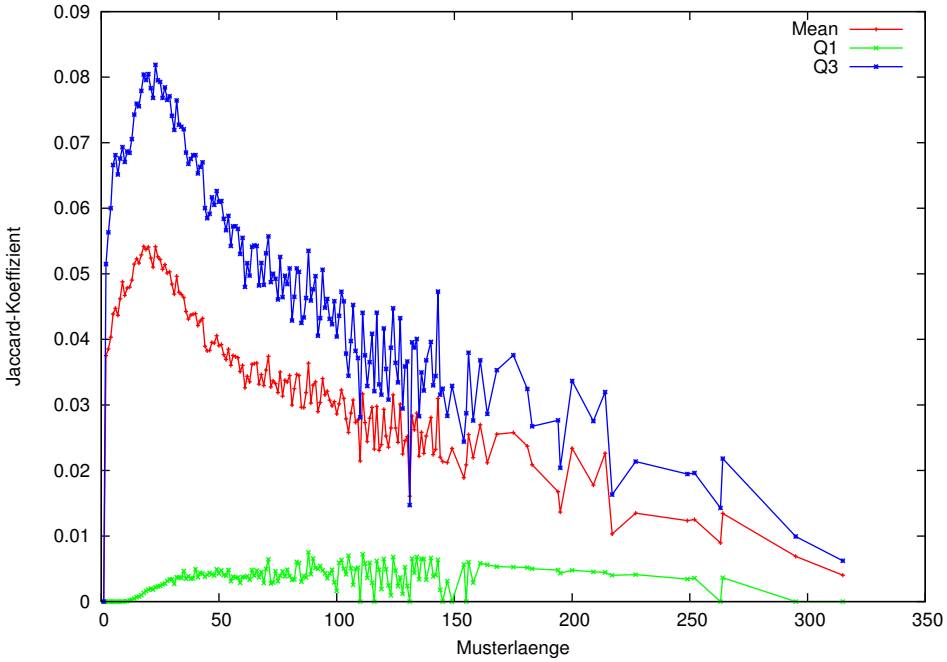


Abbildung 6: Einige Kenngrößen der Jaccard-Koeffizienten

mit einem kurzen Muster (respektive einer kleinen Menge) allein durch den Größenunterschied bereits limitiert (da die Kardinalität einer Schnittmenge maximal die Kardinalität der kleineren Menge haben kann). Eine hohe Übereinstimmung der Muster (Mengen) ist also nur bei einer ähnlichen Länge möglich. Da die Anzahl der Muster mit hohen Musterlängen allerdings sehr klein ist (siehe Abb. 4), kann der aggregierte Jaccard-Koeffizient dieser Muster mit *allen* Mustern nicht sehr groß sein. Die kleinen aggregierten Koeffizienten müssen also nicht bedeuten, dass es keine Muster mit großen Übereinstimmungen gibt.

Zusammengefasst lässt sich sagen, dass der Datensatz im Durchschnitt sehr heterogen zu sein scheint. Wenn das Maximum der durchschnittlichen Übereinstimmung 5.5% bei einer Länge von 30 beträgt, bedeutet das, dass sich diese Muster durchschnittlich $30 * 0.055 = 1.65$ Anfragen teilen. Gleichzeitig ist der Jaccard-Koeffizient aus den weiter oben beschriebenen Gründen bei einem so unterschiedlichen Datensatz, wie er hier vorliegt, in seiner Größe begrenzt. Eine hohe Übereinstimmung ist also zwischen einzelnen Mustern durchaus möglich, sie geht nur durch die doppelte Aggregation verloren.

Doch auch bei einem sehr kleinen Jaccard-Koeffizienten kann es noch zu zufällig auftretenden Mustern kommen. Ein Muster der Länge 1 hat, bedingt durch die weiter oben beschriebenen Begrenzungen des Jaccard-Koeffizienten, einen sehr kleinen Jaccard-Koeffizienten mit einem Muster der Länge 100. In diesem Fall bedeutet allerdings ein Jaccard-Koeffizient ungleich null, dass das kurze Muster komplett im langen Muster enthalten ist (da ansonsten die Schnittmenge leer und der Jaccard-Koeffizient null wäre). Zufällig auftretende Muster sind also nicht ausgeschlossen.

5.2 Der Simulator

Nachdem das Problem der Sammlung der Muster gelöst ist, kann mit der Entwicklung des eigentlichen Simulators begonnen werden. Dabei beschäftigen wir uns zuerst mit der Problemstellung, um anschließend den gewählten Lösungsansatz zu behandeln.

5.2.1 Problemstellung

Wir benötigen eine Software, die in der Lage ist, aus den gesammelten Eingabedaten nach verschiedenen Strategien Range-Query-Anfragen zu generieren und verschiedene Angriffe darauf zu simulieren.

Einige wichtige Eigenschaften, die diese Software erfüllen sollte, waren:

- **Performanz**: Da unser Datensatz etwa 93 000 Muster umfasst, die in verschiedenen Modi simuliert werden, muss das Programm auf einem aktuellen PC / Server eine akzeptable Laufzeit aufweisen.
- **Erweiterbarkeit**: Da am Anfang die Anzahl der Generations- und Angriffsstrategien nicht feststeht, muss das Programm problemlos erweiterbar sein.
- **Portabilität**: Das Programm sollte auf so vielen Betriebssystemen wie möglich lauffähig sein.
- **Lesbarkeit**: Der Quellcode des Programms sollte möglichst einfach lesbar und gut dokumentiert sein, um die Nutzung und Veränderung durch andere zu erleichtern.

5.2.2 Lösungsansatz

Als Programmiersprache für den Simulator wurde erneut Python [6] gewählt. Dies hat den Hintergrund, dass Python größtenteils plattformunabhängig ist, also unter Windows, Linux und Mac OS funktioniert. Des Weiteren erzwingt Python korrekte Einrückung des Codes, was die Lesbarkeit verbessert. Python bietet außerdem ein gutes Modul für multiprocessing (parallele Berechnung in mehreren Prozessen) an, was die Performance erheblich erhöht, sowie einfach les- und verwendbare Datenstrukturen.

Das Programm steht unter der BSD-Lizenz[10] und ist unter *https://github.com/Semantic-IA/DRQPatternAttack* abrufbar.

5.2.3 Nutzung des Simulators

Das Programm wird von der Kommandozeile aus bedient, und liefert seine Ausgaben ebenfalls auf die Kommandozeile. Es beinhaltet eine Kurzhilfe, die mit dem Parameter `-h` abgerufen werden kann.

[10] *http://opensource.org/licenses/BSD-2-Clause*

Die wichtigsten Einstellungen des Programms sind wohl die Modi, die mit dem Parameter `-m` eingestellt werden und in Kapitel 6 genauer behandelt werden, sowie die Größe, auf die die Anfrageblöcke der einzelnen wahren Anfragen mit Dummies ergänzt werden sollen. Diese wird mit dem Parameter `-s` eingestellt und ist, wenn keine Eingabe erfolgt, auf einen Standardwert von 50 eingestellt, was bedeutet, dass für jede normale Anfrage 49 Dummy-Anfragen ergänzt werden.

Des Weiteren kann die Datenbasis des Clients (aus der der Generator seine Dummy-Anfragen wählt) variiert werden. Der Angreifer arbeitet immer auf dem vollen Datensatz.[11] Die Auswahl der Anfragen für den Client erfolgt deterministisch, um eine Vergleichbarkeit der Ergebnisse verschiedener Modi zu gewährleisten. Der Algorithmus versucht, eine Kombination von Mustern zu ziehen, sodass die Zahl der einzigartigen Anfragen über alle Muster der gewünschten Anzahl von Anfragen entspricht. Dabei ist nicht garantiert, dass ein kleinerer Datensatz immer eine Teilmenge eines größeren ist. Die Anzahl der gewünschten Anfragen wird mit dem Parameter `-p` übergeben, wobei ein Wert von -1 dem gesamten Datensatz entspricht.

Wenn der Parameter `--stat` angegeben wird, generiert das Programm eine Reihe von Statistiken, die anschließend in einem für GnuPlot geeignetem Format gespeichert werden. Dabei wird im Ordner `_output` eine Ordnerstruktur nach dem Muster `Modus/Blockgröße/Datenbasis-Größe/` erstellt. In dem erstellten Ordner befinden sich dann eine Reihe von Dateien, die die *k*-Eindeutigkeit[12] der Ergebnisse angeben, einmal aufgeschlüsselt nach Musterlänge (Dateiname `M-[Musterlänge].txt`) und einmal aggregiert über alle Musterlängen (`M-ALL.txt`).

Ein weiterer Parameter ist `-c`, der die Anzahl der zufällig ausgewählten Webseitenaufrufe (im Folgenden „Muster“ genannt) bestimmt, die simuliert werden sollen. Auch dieser ist normalerweise auf 50 eingestellt. Wenn nicht nur ein zufälliger Teil, sondern die gesammte Datenbasis simuliert werden soll, sollte der Parameter `--all` verwendet werden. Dieser impliziert den Parameter `--stat`.

Vor allem wenn große Mengen an Daten simuliert werden sollen, sollte mit dem Parameter `-t` die Zahl der parallel laufenden Prozesse angegeben werden. Für eine ideale Auslastung ist die Anzahl der Prozessorkerne als Parameter zweckmäßig. Standardmäßig verwendet der Simulator nur einen Prozess.

Der einzige Parameter, der angegeben werden muss, ist der Pfad zur Datei, die die Muster enthält. Ein vollständiger Aufruf könnte unter Linux zum Beispiel so aussehen: `./DRQPatternAttack.py -t 4 -m 3 -s 25 --all patterns.txt`. Dieser Aufruf würde also mit vier parallelen Prozessen im Modus 3 mit einer Blockgröße von 25 sämtliche Muster aus der Datei patterns.txt simulieren.

Das Programm würde nun beginnen, die Eingabedatei einzulesen und anschließend die Simulation durchführen. Dabei wird jeweils ein Fortschrittsbalken angezeigt.

[11] Die Gründe dafür werden in Kapitel 8 erläutert.

[12] *k*-Eindeutigkeit: *k* Muster sind mögliche Ergebnisse des Angriffs. 1-Eindeutig bedeutet also, dass nur eine Webseite im Datensatz gefunden wurde, welches der Client besucht haben könnte.

6 Implementierte Generationsstrategien

Da eine Reihe von verschiedenen Strategien möglich sind, um die Dummy-Anfragen, durch die die wahre Anfrage getarnt wird, auszuwählen, wurden zwei dieser Strategien implementiert und evaluiert. Einige weitere Strategien, die nicht implementiert wurden, werden in Kapitel 10.1 behandelt.

Wir untersuchen die folgenden zwei Strategien:

1. **Zufällige Auswahl der Anfragen**: Dies ist die Strategie, die von Zhao et al. beim Entwurf des Range Query-Algorithmus vorgeschlagen wird. Diese Strategie wird in Abschnitt 6.1 vorgestellt und analysiert.
2. **Musterbasierte Auswahl der Anfragen**: Diese Strategie wurde im Rahmen dieser Bachelorarbeit entwickelt und sollte eine bessere Privatsphäre bieten. Diese Strategie wird in Abschnitt 6.2 vorgestellt und analysiert.

Außerdem sind verschiedene Annahmen darüber möglich, in welcher Form der Angreifer die Anfragen erhält. Für jeden der Generationsmodi wurden daher drei verschiedene Varianten implementiert, die sich einen Generator teilen, aber seine Ergebnisse unterschiedlich repräsentieren. Die drei Varianten sind:

1. **Vollständig ununterscheidbare Blöcke**: In dieser Variante ist der Angreifer nicht in der Lage, die Blöcke voneinander zu unterscheiden. Aus den im nächsten Punkt genannten Gründen ist diese Variante unrealistisch, wurde allerdings als geeigneter „worst case“ untersucht.
2. **Erster Block unterscheidbar, alle weiteren ununterscheidbar**: In dieser Variante kann der Angreifer den ersten Block von Anfragen sicher bestimmen, alle weiteren sind allerdings nicht eindeutig einem Block zuzuordnen. Diese Annahme ist vermutlich die realistischste, da der erste Block in jedem Fall bestimmbar ist, da die Antwort auf die erste Anfrage bekannt sein muss, um die Ziele weiterer Anfragen zu bestimmen (da die Seite aufgerufen werden muss, um die externen Inhalte zu bestimmen und die Anfragen für diese Domains zu stellen).
3. **Vollständig unterscheidbare Blöcke**: In dieser Variante kann der Angreifer alle Anfragen zweifelsfrei den jeweiligen Anfrageblöcken zuordnen, in denen sie enthalten sind. Diese Annahme ist nicht immer realistisch, da zwei Blöcke unter Umständen parallel an den Server gesendet werden, was eine Zuordnung erschwert. Sie stellt allerdings einen guten „best case“ dar.

Die drei Varianten sind in Abb. 7 visualisiert.

Generell gilt, dass bei allen Strategien die Grundannahme getroffen wurde, dass die Reihenfolge der Anfragen nicht erhalten bleibt (mit der Ausnahme der ersten Anfrage, die natürlich an erster Stelle stehen muss, da sie für die Bestimmung aller weiteren Anfragen benötigt wird). Das Muster $\langle A, B, C \rangle$ kann also auch als $\langle A, C, B \rangle$ beim Angreifer ankommen.

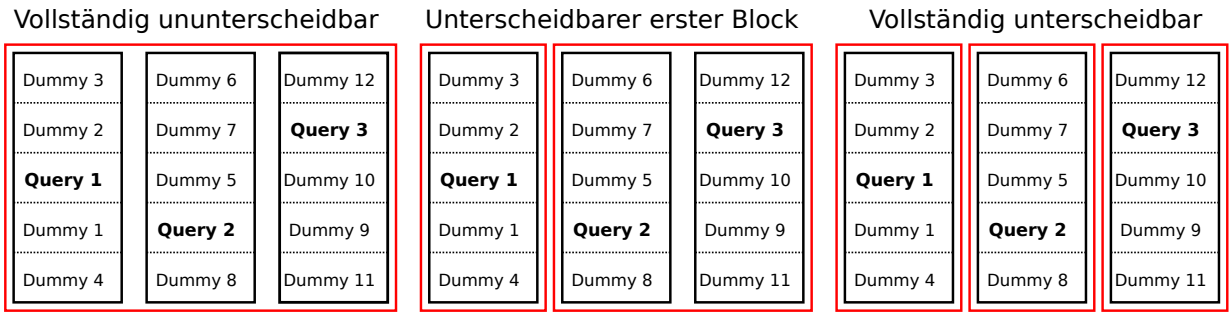


Abbildung 7: Die Aufteilung der Blöcke in den verschiedenen Modi

Diese Annahme ist realistisch, da sich der Aufbau einer Webseite verändern kann. Außerdem verwendet die DNS-Infrastruktur für fast alle Anfragen UDP-Pakete, bei denen im Gegensatz zu TCP-Verbindungen nicht garantiert ist, dass sie in der Reihenfolge ankommen, in der sie gesendet wurden. Die Annahme macht unser System daher robuster und realistischer.

6.1 Zufällige Auswahl der Anfragen

Bei der zufälligen Auswahl der Dummy-Anfragen werden Dummy-Anfragen zufällig und ohne Zurücklegen aus der Menge Q aller bekannten Domain-Namen gezogen. Dies ist die Strategie, die von Zhao et al. in [13] vorgeschlagen wurde.

Die vorgeschlagene Strategie war: Für jede „echte“ Anfrage M_n werden $N-1$ Anfragen zufällig und ohne Zurücklegen aus Q gewählt und gemeinsam mit M_n an den Resolver geschickt. Dabei kann maximal ein Duplikat innerhalb eines Blockes entstehen (wenn M_n bei der Auswahl aus Q gewählt wird). Sollte ein solches Duplikat entstehen, wird es entfernt, die Blockgröße sinkt auf $N-1$. Über mehrere Blöcke hinweg kann es in dieser Strategie allerdings durchaus zu Duplikaten kommen (wenn in zwei separaten Blöcken dieselbe Anfrage aus Q gewählt wird).

Die implementierte Strategie nutzt aus Performancegrünnden eine leicht modifizierter Variante: Anstatt $|M|$ separate Auswahlen aus Q zu treffen, werden direkt am Anfang $|M| \cdot (N-1)$ Anfragen aus Q gewählt. Diese werden dann auf die einzelnen Blöcke verteilt. Die Laufzeit konnte dadurch erheblich verringert werden, allerdings schließt diese Strategie blockübergreifende Duplikate aus, da das Ziehen der Anfragen weiterhin ohne Zurücklegen geschieht. Da ein solches Duplikat allerdings nur mit einer geringen Wahrscheinlichkeit auftritt, verändert dies das Ergebnis der Simulation nur in sehr wenigen Fällen. Es ist weiterhin möglich, bis zu ein Duplikat pro Block zu erhalten (wenn das entsprechende Element von M in „seinem“ Block gezogen wird).

Die ersten drei Modi des Simulators funktionieren mit dieser Strategie. Die Auswahl erfolgt über den Paramter `-m`:

- `-m 1`: Zufällige Auswahl, vollständig ununterscheidbare Blöcke.
- `-m 2`: Zufällige Auswahl, erster Block unterscheidbar.

- `-m 3`: Zufällige Auswahl, vollständig unterscheidbare Blöcke.

Bei dieser Strategie kann noch an zwei Stellen zwischen verschiedenen Varianten gewählt werden, namentlich bei der Strategie beim Ziehen der Dummy-Anfragen und beim Verhalten bei Duplikaten. Im Folgenden werden andere Varianten kurz vorgestellt. In Abschnitt 6.1.2 wird dann an den relevanten Stellen kurz erläutert, was die jeweiligen Varianten für Auswirkungen auf die erwarteten Ergebnisse hätten. Implementiert wurde jedoch nur die oben beschriebene Variante.

Ziehen der Dummy-Anfragen In der Beschreibung weiter oben ziehen wir die Dummy-Anfragen *ohne* Zurücklegen. Natürlich wäre auch ein ziehen *mit* Zurücklegen möglich. Dies würde die Chance von Duplikaten erhöhen, da nun dieselbe Dummy-Anfrage mehrmals gezogen werden könnte.

Verhalten bei Duplikaten In dieser Version des Range Query werden Duplikate entfernt und nicht durch neue Dummy-Anfragen ersetzt. Natürlich wäre es auch möglich, Duplikate durch neue, zufällig ausgewählte Anfragen zu ersetzen. Dadurch wäre sichergestellt, dass wir immer genau $|M| \cdot N$ Anfragen erhalten, was intuitiv zu einer schlechtere Erkennungsleistung für den Angreifer führen sollte.

6.1.1 Beispiel

Ein Beispiel für diese Generationsstrategie wurde bereits in Abschnitt 4.3 gegeben, daher wird es hier nicht wiederholt.

6.1.2 Erwartete Ergebnisse

Prinzipiell sind je nach gewähltem Modus eine unterschiedliche Erkennungsleistung zu erwarten. Daher werden die Modi auch hier unterschieden. Im Folgenden soll jeder Modus einzeln untersucht werden. Dabei liegt der Fokus auf eventuellen Schwachstellen der Strategie, die von dem Angreifer ausgenutzt werden können, um die Menge der Ergebnisse einzuschränken, sowie auf einer Abschätzung, mit wie vielen Ergebnissen wir bei verschiedenen Musterlängen durchschnittlich rechnen können.

Es werden alle drei Modi untersucht, allerdings liegt der Fokus auf dem zweiten Modus (unterscheidbarer erster Block), da dieser die realistischsten Annahmen darstellt und eine gute Leistung hier am wichtigsten ist.

Generell gilt es herauszufinden, welche Informationen der Angreifer aus den Daten, die er erhält, ziehen kann. Dabei sind vor allem die Informationen von Interesse, die dem Angreifer erlauben, die Menge der als Lösung in Frage kommenden Muster zu reduzieren. Dies kann er unter anderem, wenn ihm der Wert von $|M|$ bekannt ist (oder zumindest ein Zahlenbereich, in dem dieser

Wert liegen muss), da er dann nur noch Muster der Länge $|M|$ (oder einer Länge aus dem gefundenen Zahlenbereich) betrachten muss. Intuitiv gilt, dass die Genauigkeit steigt, je weniger Optionen betrachtet werden müssen, solange garantiert ist, dass sich das richtige Ergebnis unter diesen Optionen befindet.

Vollständig ununterscheidbare Blöcke Bei vollständig ununterscheidbaren Blöcken ist generell die schlechteste Erkennungsleistung zu erwarten, da die Anzahl der möglichen Muster bis zu $|M| \cdot N$ beträgt (jedes Element ist potentiell das erste Element des gesuchten Musters, da keine Aussagen über die Kausalität der Anfragen gemacht werden kann).

Da N dem Angreifer nicht bekannt ist, kann er auch keine Aussagen über $|M|$ treffen, er kann seine möglichen Ergebnisse also nicht nach Länge filtern. Und selbst wenn dem Angreifer N bekannt wäre (etwa weil es ein fest gesetzter Wert in einer Software ist), ist nicht einmal garantiert, dass $\frac{|S|}{N}$ eine ganze Zahl ist, da durch das Entfernen von Duplikaten $|S| \neq |M| \cdot N$ gelten kann. Entsprechend sind überhaupt keine sicheren Aussagen über $|M|$ möglich (mit der Ausnahme von $|M| \leq |S|$, was bei realistisch gewählten N allerdings keine brauchbare Information liefert). Eine Reduzierung von $|MV|$ ist also auf diese Art nicht möglich.

Es gelingt uns also auf keine Art, die Anzahl der in Frage kommenden Muster zu reduzieren. Das bedeutet, dass alle in S gefundenen Muster potentiell das gesuchte Muster sind, und keine der gefundenen Muster anhand ihrer Länge ausgeschlossen werden können.

In diesem Modus sollte es keinen großen Unterschied machen, ob Duplikate durch neue Anfragen ersetzt werden und ob das Ziehen von Dummies mit oder ohne Zurücklegen geschieht. Das Ersetzen von Duplikaten würde die Chance auf zufällig entstehende Muster leicht erhöhen, das Ziehen mit Zurücklegen und ohne Ersetzen der Duplikate würde sie leicht verringern, aber in keinem Fall ist eine große Änderung zu erwarten.

Die anderen beiden Modi werden stochastisch analysiert, um einen Erwartungswert für die Genauigkeit des Algorithmus zu erhalten. Bei diesem Modus wurde auf eine solche Analyse verzichtet, da er aus den in Kapitel 6 angegebenen Gründen in der realen Welt nicht realistisch ist. Prinzipiell ließe er sich allerdings über die in den folgenden Abschnitten verwendeten Strategien ebenfalls analysieren.

Erster Block unterscheidbar In diesem Modus erhalten wir zwei Blöcke, S_1 und S_2. Dabei enthält S_1 das erste Element des Anfragemusters M (zusammen mit bis zu $N-1$ anderen Anfragen), und S_2 enthält den Rest des Musters M mit insgesamt bis zu $(M-1) \cdot (N-1)$ anderen, zufällig gewählten Anfragen.

Die Erkennungsleistung bei einem unterscheidbaren ersten Block sollte generell höher liegen als bei vollständig ununterscheidbaren Blöcken. Dies hat den Grund, dass wir nun nicht mehr $|S|$ verschiedene potentielle Muster beachten müssen, sondern nur noch bis zu N, da sich der Beginn des Musters im ersten Block S_1 befinden muss. Dieser enthält entweder N oder $N-1$ (wenn der Musteranfang auch bei der zufälligen Ziehung der Dummy-Anfragen gezogen und in den

ersten Block einsortiert wurde) Anfragen. Weniger als $N-1$ Anfragen sind nicht möglich, da das Ziehen von Dummies ohne Zurücklegen geschieht, und somit das erste Element von M das einzige Duplikat ist, das auftreten kann.

Des Weiteren können wir in diesem Modus eine Vermutung über $|M|$ anstellen: Wir wissen, dass für jedes Element von M insgesamt $N-1$ Dummy-Anfragen gezogen werden, wodurch $|S_1|+|S_2| = |M| \cdot N$, also $|M| = \frac{|S_1|+|S_2|}{N}$ gilt, sofern bei der zufälligen Ziehung von Dummy-Anfragen keine Duplikate entstanden sind. Ohne Duplikate könnten wir außerdem $|S_1| = N$ festlegen, da der erste Block exakt eine Anfrage mit $N-1$ Dummy-Anfragen enthalten sollte. Da wir aber nicht immer davon ausgehen können, keine Duplikate zu erhalten, können wir nicht immer von $|S_1| = N$ ausgehen.

Wir wissen jedoch, dass N entweder $|S_1|$ oder $|S_1|+1$ sein muss, da es zu maximal einem Duplikat kommen kann (wenn M_1 in den $N-1$ zufällig ausgewählten Anfragen enthalten ist, wodurch durch das Entfernen des Duplikates $|S_1|$ um eins sinken würde).

Wir benennen unsere Vermutung über den Wert von N im Folgenden mit n:

$$n = |S_1| \tag{1}$$

Mit diesem Wert können wir nun eine erste Abschätzung darüber erhalten, welchen Wert $|M|$ haben könnte. Unter der Annahme, dass es zu keinen Duplikaten gekommen ist, würde, wie bereits weiter oben festgestellt wurde, die folgende Gleichung gelten:

$$|M| = \frac{|S_1|+|S_2|}{n} \tag{2}$$

Allerdings muss bei großen $|M|$ und N mit Duplikaten gerechnet werden, weshalb diese Formel nicht allgemeingültig ist. Zum einen könnte n aus den weiter oben beschriebenen Gründen um eins zu klein sein, zum anderen kann es auch in S_1 und S_2 zu Duplikaten (und damit verringerten Blockgrößen) kommen.

Um diese Probleme zu lösen, bestimmen wir eine obere und untere Grenze, zwischen denen $|M|$ liegen muss. Dazu definieren wir die Untergrenze m_1 sowie die Obergrenze m_2, wodurch $m_1 \leq |M| \leq m_2$ gelten muss.

Die Untergrenze m_1 können wir direkt definieren: Die Werte von $|S_1|$ und $|S_2|$ können zu keinem Zeitpunkt nach der initialen Berechnung mehr wachsen, nur noch durch das Entfernen von Duplikaten schrumpfen, wodurch unsere $|S_1|$ und $|S_2|$ Minimalwerte darstellen. N kann maximal $n+1$ betragen. Dies bedeutet, dass die Untergrenze m_1 mit der folgenden Formel bestimmt werden kann:

$$m_1 = \left\lceil \left(\frac{|S_1|+|S_2|}{n+1} \right) \right\rceil \tag{3}$$

Die Obergrenze ist schwerer zu bestimmen. Die intuitive Lösung wäre, in der Formel (2) das $|M|$ durch ein m_2 zu ersetzen und um den Bruch eine obere Gaußklammer zu setzen. Solange wir weniger als n Duplikate erhalten haben, sollte die obere Gaußklammer die Obergrenze ausreichend erhöhen, um das gesamte Intervall möglicher Werte von $|M|$ abzudecken.

$$m_2 \stackrel{?}{=} \left\lceil \left(\frac{|S_1| + |S_2|}{n} \right) \right\rceil \tag{4}$$

Allerdings funktioniert diese Lösung nicht in allen Fällen, denn sobald es zu mehr als N Duplikaten gekommen ist, kommt die Rechnung zu zu kleinen Ergebnissen. Wir benötigen also noch eine Abschätzung dafür, zu wie vielen Duplikaten es maximal gekommen sein kann.

Die initiale Vermutung wäre, dass die maximale Anzahl d der Duplikate, die aufgetreten sein können, durch die folgende Formel bestimmt werden kann, wobei m_2 den Wert von Formel (4) darstellt.

$$d = \left\lceil \left(\frac{m_2}{n} \right) \right\rceil \tag{5}$$

Damit ließe sich dann m_2 durch ein $m_2 + d$ so erhöhen, dass die Obergrenze die potentiellen Duplikate beachtet. Allerdings existieren für Formel (5) einige Randfälle, in denen sie eine zu kleine Zahl ergibt. Experimente haben gezeigt, dass das Verdoppeln von d in jedem Fall alle möglichen Werte enthält, auch wenn die Obergrenze dadurch teilweise höher als unbedingt nötig ist.

$$m_2 = \left\lceil \left(\frac{|S_1| + |S_2|}{n} \right) \right\rceil + 2 * d \tag{6}$$

Im Folgenden möchten wir nun bestimmen, mit wie vielen **erkannten Mustern** wir bei verschiedenen Musterlängen rechnen müssen. Dazu benötigen wir die Wahrscheinlichkeit, bei der zufälligen Ziehung der Dummy-Anfragen ein weiteres Muster der richtigen Länge zu ziehen. Generell lässt sich die Wahrscheinlichkeit, dass ein Muster einer bestimmten Länge n in S enthalten ist, durch eine Reihe von hypergeometrischen Verteilungen modellieren. Allgemein gibt eine hypergeometrische Verteilung die Wahrscheinlichkeit an, beim Ziehen von n Elementen aus einer Grundmenge N genau k Elemente mit einer bestimmten Eigenschaft zu ziehen. Dabei gibt M die Anzahl der Elemente aus N mit der gewünschten Eigenschaft an. Die Formel einer hypergeometrischen Verteilung lautet im allgemeinen:

$$h(k|N;M;n) := \frac{\binom{M}{k}\binom{N-M}{n-k}}{\binom{N}{n}} \tag{7}$$

Dabei bezeichnet $\binom{n}{k}$ wie üblich den Binomialkoeffizienten.

Die Variablen der hypergeometrischen Verteilung überschneiden sich teilweise mit den von uns definierten, daher ersetzen wir sie im Folgenden durch ihre Entsprechungen in unserem Benennungssystem. In unserem Fall ist die Grundmenge N die Menge aller Anfragen Q. Die Anzahl der günstigen Ergebnisse M der hypergeometrischen Verteilung, um einen weiteren Musteranfang der korrekten Länge zu erhalten, ist $|P_n|$, wobei P_n wie üblich die Menge aller Musteranfänge der Länge n beschreibt. Das n der hypergeometrischen Verteilung lautete in diesem Fall $N-1$, da wir N Anfragen pro echter Anfrage aus M verschicken, und eines dieser Elemente immer die entsprechende Anfrage aus M sein wird. Damit verbleiben $N-1$ zufällig gewählte Anfragen aus Q.

Die Wahrscheinlichkeit, genau einen Musteranfang eines Musters der Länge n zu ziehen, lässt sich durch das Einsetzen der entsprechenden Werte in (7) folgendermaßen modellieren:

$$p(n) = \frac{\binom{|P_n|}{1}\binom{|Q|-|P_n|}{(N-1)-1}}{\binom{|Q|}{N-1}} = \frac{|P_n| \cdot \binom{|Q|-|P_n|}{N-2}}{\binom{|Q|}{N-1}} \tag{8}$$

Wenn wir in Gleichung (8) nun unsere Werte für $|Q|$ und unseren Grundwert von $N = 50$ einsetzen, erhalten wir:

$$p(n) = \frac{|P_n| \cdot \binom{216925-|P_n|}{50-2}}{\binom{216925}{50-1}} \tag{9}$$

Wenn wir nun von einem Muster der durchschnittlichen Musterlänge 13 ausgehen, erhalten wir:

$$p(13) = \frac{|P_{13}| \cdot \binom{216925-|P_{13}|}{50-2}}{\binom{216925}{50-1}} = \frac{2685 \cdot \binom{216925-2685}{50-2}}{\binom{216925}{50-1}} \approx 0.3336 \tag{10}$$

Dazu kommt nun noch die Wahrscheinlichkeit, in S_2 die restlichen Elemente des Musters hineinzuziehen. Dazu interessiert uns die Wahrscheinlichkeit, beim Ziehen von $N-1$ Dummies für jedes der restlichen $|M|-1$ Elemente des Muster M die restlichen $n-1$ Elemente des zufällig auftretenden Musters der Länge n zu ziehen. Auch diese Wahrscheinlichkeit, im Folgenden als Funktion q mit dem Parameter n bezeichnet, lässt sich ebenfalls mit einer hypergeometrischen Verteilung bestimmen:

$$q(n) = \frac{\binom{n-1}{n-1} \cdot \binom{|Q|-(n-1)}{(|M|-1)*(N-1)-(n-1)}}{\binom{|Q|}{(|M|-1)*(N-1)}} = \frac{\binom{216925-(n-1)}{(|M|-1)*(50-1)-(n-1)}}{\binom{216925}{(|M|-1)*(50-1)}} \tag{11}$$

Wenden wir Formel (11) nun auf $n = 13$, $|M| = 13$, $N = 50$ an, erhalten wir:

$$q(n) = \frac{\binom{216925-(13-1)}{(13-1)*(50-1)-(13-1)}}{\binom{216925}{(13-1)*(50-1)}} \approx 1.4056 \times 10^{-31} \tag{12}$$

Da, damit das Muster komplett auftritt, sowohl das erste Element als auch die restlichen Elemente auftreten müssen, berechnet sich die Wahrscheinlichkeit, dass das Muster auftritt, aus dem Produkt von $p(n)$ und $q(n)$. In diesem Fall erhalten wir also $p(13) \cdot q(13) = 0.3336 * 1.4056 \times 10^{-31} = 4.6892 \times 10^{32}$. Die Wahrscheinlichkeit, genau ein komplettes Muster der Länge $n = 13$ in den Dummies eines Musters der Länge $|M| = 13$ bei Blockgröße $N = 50$ zu ziehen, ist also sehr gering.

Unsere Formeln beachten momentan allerdings nur den Fall, *genau* ein Muster einer speziellen Länge zu ziehen. Was passiert aber, wenn wir *mindestens* ein Muster dieser Länge ziehen wollen? Für diesen Fall müssen wir die Formeln etwas modifizieren.

Beginnen wir mit der Formel für den Musteranfang im ersten Block. Wenn wir nicht mehr 1, sondern k Musteranfänge von Mustern der Länge n ziehen wollen, erweitern wir die Formel (8) folgendermaßen:

$$p(n,k) = \frac{\binom{|P_n|}{k}\binom{|Q|-|P_n|}{(N-1)-k}}{\binom{|Q|}{N-1}} \tag{13}$$

Des Weiteren müssen wir die Formel (11) anpassen, nicht mehr $n-1$ passende Anfragen, sondern $(n-1)*k$ passende Anfragen zu ziehen. Dafür versehen wir den ersten Binomialkoeffizienten mit einem Exponenten von k, und passen anschließend noch den zweiten Binomialkoeffizienten an, um die veränderte Anzahl an verbleibenden Mustern, die aus der Restmenge gezogen werden sollen, wiederzuspiegeln:

$$q(n,k) = \frac{\binom{n-1}{n-1}^k \cdot \binom{|Q|-(n-1)*k}{(|M|-1)*(N-1)-(n-1)*k}}{\binom{|Q|}{(|M|-1)*(N-1)}} = \frac{\binom{216925-(n-1)*k}{(|M|-1)*(50-1)-(n-1)*k}}{\binom{216925}{(|M|-1)*(50-1)}} \tag{14}$$

Für unsere Analyse ist besonders der Erwartungswert $E(X)$ interessant. Er gibt den durchschnittlichen Wert der Zufallsvariable X an, in unserem Fall also die durchschnittliche Anzahl an Mustern, die erkannt werden wird. Der Erwartungswert ist definiert als die Summe über alle Einzelwahrscheinlichkeiten multipliziert mit den jeweilig der Wahrscheinlichkeit zugehörigen Werten, in unserem Fall also der Anzahl der Muster, k:

$$E(X) = \sum_{i \in I}(x_i p_i) \tag{15}$$

In unserem Fall ist $x_i = k$, da k die Anzahl der gefundenen Muster darstellt, und $p_i = p(n,k) \cdot q(n,k)$, da sich die Wahrscheinlichkeit, k Muster zu ziehen, aus den beiden Einzelwahrscheinlichkeiten $p(n,k)$ und $q(n,k)$ zusammensetzt. Die Summe geht von $k = 1$ bis $N-1$, da wir weiterhin bis zu $N-1$ Muster ziehen können. Als letztes addieren wir noch eine Eins zum Ergebnis, da das Muster M in jedem Fall vorhanden sein wird.

Es ergibt sich die folgende Formel:

$$E(n) = 1 + \sum_{k=1}^{N-1} (p(n,k) \cdot q(n,k) \cdot k) \tag{16}$$

Diese Formel gilt für alle $n > 1$. Ein Spezialfall ist $n = 1$. Da bei einer Musterlänge von $n = 1$ keine weiteren Elemente in S_2 gezogen werden müssen, entfällt der Anteil $q(n,k)$ der Formel (16). Und anstatt eine Summe zu berechnen, kann man in dem Fall auch die Formel für den Erwartungswert einer hypergeometrischen Verteilung verwenden:

$$E(n = 1) = 1 + (N-1)\frac{|P_n|}{|Q|} = 1 + (50-1)\frac{8391}{216925} \approx 2.8953 \tag{17}$$

Nun können wir die Erwartungswerte für einzelne n berechnen. Unsere bisherigen Erkenntnisse erlauben und allerdings nur, die Länge des Musters innerhalb des Intervalls $[m_1, m_2]$ festzulegen. Um den Erwartungswert für ein solches Intervall zu approximieren, erweitern wir die Formel (16) um eine weitere Summe:

$$E(n \in [m_1, m_2]) = 1 + \sum_{n=m_1}^{m_2} \sum_{k=1}^{N-1} (p(n,k) \cdot q(n,k) \cdot k) \tag{18}$$

Diese Formel ist nicht ganz korrekt, da die Wahrscheinlichkeit, weitere Muster zu finden, mit jedem gefundenen Muster sinkt (da einige Anfragen schon zur Bildung des ersten Musters gezogen wurden, stehen sie nicht mehr zum Bilden eines weiteren Musters zur Verfügung). Dies zu modellieren wäre allerdings für diese Arbeit zu aufwendig, und der Unterschied sollte gering ausfallen.

Wenn wir diese Rechnung nun für einige $|M|$ durchführen wollen, müssen wir für diese $|M|$ die Grenzen m_1 und m_2 bestimmen. Die Formeln (3) und (6), die diese Werte bestimmen, ergeben größtmögliche Intervall zwischen diesen beiden Werten, wenn keine Duplikate aufgetreten sind, also $S_1 = N$ und $S_2 = (N-1) * (|M|-1)$ gilt. Wir erhalten:

$$m_1 = \left\lceil \left(\frac{N + (N-1) * (|M|-1)}{N+1} \right) \right\rceil \tag{19}$$

$$m_2 = \left\lceil \left(\frac{N + (N-1) * (|M|-1)}{N} \right) \right\rceil + 2 * \left\lceil \left(\frac{\left\lceil \left(\frac{N+(N-1)*(|M|-1)}{N} \right) \right\rceil}{N} \right) \right\rceil \tag{20}$$

In Formel (20) repräsentiert der Term hinter dem Additionszeichen das hinzufügen von $2 * d$, wie es weiter oben in diesem Abschnitt definiert wurde.

Um nun den Erwartungswert für eine Musterlänge zu bestimmen, berechnen wir nach den oben angegebenen Formeln m_1 und m_2 und setzen sie als Grenzwerte in die erste Summe von Formel (18) ein. Die Ergebnisse für verschiedene Musterlängen sind in Abb. 8 zu sehen.

$\lvert M\rvert$	1	2	3	4+
$E(\lvert M\rvert)$	2.895397027	1.000074207	1.000000071	1

Abbildung 8: Erwartungswerte für verschiedene Musterlängen

Wie man in der Tabelle sehen kann, sinkt der Erwartungswert an erkannten Mustern sehr schnell auf 1 und verbleibt dort. Das bedeutet, dass ein Angreifer, der unsere Optimierungen nutzt, Muster ab einer Länge von 2 bei einer Blockgröße von $N = 50$ bereits fast immer eindeutig identifizieren kann.

Hier, wie auch bei der nächsten stochastischen Analyse im nächsten Abschnitt, muss allerdings noch beachtet werden, dass unsere stochastische Abschätzung momentan nicht den Fall beachtet, dass Muster teilweise oder sogar komplett in anderen Mustern enthalten sein könnten. Die erhöht die Wahrscheinlichkeit, dass das enthaltene Muster bei der Generation von Range Queries für das andere Muster zufällig auftreten. Diese Eigenschaft war stochastisch nicht modellierbar und konnte daher nicht beachtet werden, sie dürfte die Erwartungswerte allerdings voraussichtlich etwas erhöhen.

Die Schwierigkeiten bei der Bestimmung von $|M|$ und N wären nicht aufgetreten, wenn wir gefundene Duplikate durch neue, zufällig ausgewählte Anfragen ersetzen würden. In dem Fall würde sich $|M|$ durch ein simples $1 + \frac{|S_2|}{|S_1|}$ bestimmen lassen, da auf jede Anfrage garantiert exakt $N-1$ Dummies entfallen. Dadurch wäre $|S_1| = N$ garantiert, da S_1 nur die erste Anfrage sowie ihre Dummies enthält. Um in diesem Fall den Erwartungswert zu berechnen, können wir wieder Formel (16) nutzen.

Würden wir die Dummy-Anfragen hingegen mit Zurücklegen ziehen, so wäre nicht mehr garantiert, dass $|S_1|$ entweder N oder $N-1$ beträgt, da mehr als ein Duplikat entstehen könnten. Auch unsere Abschätzung über das Intervall, in dem $|M|$ sich befinden muss, wären nicht mehr zwangsweise richtig, da mehr als $N-1$ Duplikate in S_2 auftreten könnten. Die optimierte Strategie würde weiterhin in den *meisten* Fällen funktionieren, aber nicht mehr zuverlässig *immer*.

Vollständig unterscheidbare Blöcke Von allen vorgestellten Modi sollte dieser die beste Erkennungsleistung bieten. Wie beim unterscheidbaren ersten Block müssen wir maximal N potentielle Muster untersuchen, jedoch wissen wir zusätzlich, dass $|M| = |S|$ sein muss (da jedes Element aus M durch einen eigenen Block in S dargestellt wird). Dies bedeutet, dass die in S_1 gefundenen Musteranfänge auch noch nach der Länge der zugehörigen Muster gefiltert werden können. Wenn also das potentielle Muster x eine Länge von $|x| \neq |S|$ hat, kann x ausgeschlossen werden. Dies reduziert die Menge der potentiellen Ergebnisse MV erheblich.

Auch dieses Szenario lässt sich mit Hilfe von hypergeometrischen Verteilungen modellieren. Die Wahrscheinlichkeit, in S_1 das erste Element eines Musters der Länge n zu ziehen, wird unverändert durch die Formel (9) modelliert.

Dazu kommt nun noch die Wahrscheinlichkeit, sämtliche weiteren Musterelemente in die jeweiligen anderen Blöcke zu ziehen. Dies bedeutet, dass wir in den zweiten Block von S exakt

ein Element des Musters ziehen möchten, in den dritten Block ebenfalls exakt ein Element (allerdings keines derer, die bereits gezogen wurden), bis zum $|M|$-ten Block, in dem das letzte verbleibende Element des zweiten Musters vorhanden sein soll.

Da eine Reihe von stochastischen Bedingungen, die alle eintreten müssen, über eine Multiplikation modelliert wird, können wir diese Einschränkung über das Produkt von $n-1$ hypergeometrischen Verteilungen modellieren. Dabei müssen wir beachten, dass kein Element des Musters doppelt gezogen wird, die Menge der günstigen Ergebnisse also immer weiter schrumpft, bis sie beim letzten Glied des Produktes bei 1 angekommen ist.

$$q(n) = \prod_{j=1}^{n-1} \frac{\binom{n-j}{1}\binom{|Q|-(n-j)}{(N-1)-1}}{\binom{|Q|}{N-1}} = \prod_{j=1}^{n-1} \frac{\binom{n-j}{1}\binom{216925-(n-j)}{50-2}}{\binom{216925}{50-1}} \tag{21}$$

Wenn wir (21) nun auf $n = 13$ anwenden, erhalten wir:

$$q(13) = \prod_{j=1}^{13-1} \frac{\binom{13-j}{1}\binom{216925-(13-j)}{50-2}}{\binom{216925}{50-1}} \approx 8.3297 \times 10^{-36} \tag{22}$$

Die Chance, dass genau ein komplettes zweites Muster der Länge n auftritt, beträgt $p(n) \cdot q(n)$ (das Ziehen eines Musteranfangs, gefolgt vom Ziehen aller Musterelemente), was im Falle von $n = 13$ einen Wert von $0.3336 \cdot 8.3297 \times 10^{-36} \approx 4.68924 \times 10^{-32}$ annimmt. Von den 2685 Mustern der Länge 13 wird also nur in $4.68924 \times 10^{-32} \cdot 2685 \approx 1.25906 \times 10^{-28}$ exakt ein weiteres Muster auftreten.

Diese Rechnung beachtet dabei nicht, dass auch mehr als ein weiteres Muster auftreten kann. Dies wird zwar mit zunehmender Musterlänge immer unwahrscheinlicher, kann aber bei einer kurzen Länge durchaus vorkommen.

Damit $q(n)$ das Ziehen von mehr als einem Muster modellieren kann, nutzen wir die Eigenschaft hypergeometrischer Verteilungen, dass wir im Zähler mehrere Binomialkoeffizienten multiplizieren können, um mehrere Ereignisse eintreten zu lassen. Da die Musterlänge aller erwünschten Muster gleich ist, können wir dafür den ersten Binomialkoeffizienten mit einem Exponenten von k versehen. Dadurch fordern wir, aus k Gruppen mit einer gewissen Anzahl an Elementen jeweils exakt ein Element zu ziehen. Dann müssen wir noch den zweiten Binomialkoeffizienten anpassen, um festzulegen, dass jetzt nicht nur ein, sondern k Elemente gezogen werden. Setzen wir diese Änderungen um, erhalten wir:

$$q(n,k) = \prod_{j=1}^{n-1} \frac{\binom{n-j}{1}^{k}\binom{|Q|-(n-j)\cdot k}{N-1-k}}{\binom{|Q|}{N-1}} \tag{23}$$

Wenn man den Erwartungswert (siehe Formel (16)) dieser Funktionen für eine Reihe von $n = |M|$ betrachtet, wird schnell klar, dass der Wert innerhalb weniger Schritte bereits auf annähernd eins fällt und dort verbleibt (Siehe Tabelle in Abb. 9). Interessant ist ebenfalls der Vergleich mit den

$\|M\|$	1	2	3	4+
$E(\|M\|)$	2.895313482	1.000074189	1.000000036	1

Abbildung 9: Die Erwartungswerte für eine Reihe von $|M|$ bei $N = 50$

Resultaten aus dem vorherigen Modus (Vgl. Abb. 8). Die Ergebnisse stimmen fast komplett überein, nur bei $|M| = 2$ entsteht noch ein geringer Unterschied.

Allerdings gilt auch in diesem Modus die Einschränkung, dass der Erwartungswert in der Realität höher ist, da auch hier die Teile eines Musters bereits vorhanden sein können, die in M enthalten sind.

Zusammengefasst lässt sich sagen, dass wir in diesem Modus mit sehr wenigen zufällig auftretenden Mustern rechnen müssen. Dabei unterscheidet sich die Leistung nur sehr wenig von der Leistung im vorherigen Modus (vgl. Tabelle in Abb. 8). Dies bedeutet, dass ein Angreifer nicht einmal in der Lage sein muss, einzelne Blöcke zu unterscheiden. Solange er den ersten Block erkennen kann und die im vorherigen Abschnitt genannten Vorbedingungen erfüllt sind, kann er eine annähernd mit dem Idealfall identische Erkennungsleistung erreichen.

In diesem Modus macht es keinen großen Unterschied, ob mit oder ohne Zurücklegen gezogen wird und ob Duplikate nur entfernt oder auch ersetzt werden. Nicht ersetzte Duplikate verringern die Anzahl an Anfragen, die zufällige Muster ergeben könnten, erhöhen die Erkennungsleistung also minimal. Ziehen mit Zurücklegen würde zu marginal mehr Duplikaten führen, die, je nach gewählter Methode, entweder ersetzt oder entfernt werden würden. Die Leistung wird sich nicht verschlechtern und voraussichtlich nicht stark verbessern, egal, welche der Strategien verwendet wird.

6.2 Musterbasierte Auswahl der Anfragen

Bei der musterbasierten Auswahl der Dummy-Anfragen versucht der Client, das Muster der besuchten Webseite mit möglichst vielen weiteren Mustern derselben Länge zu verschleiern. Dadurch soll der Angreifer insgesamt mindestens N Muster der gleichen Länge erkennen, ohne eine Information darüber erhalten zu können, welches davon die besuchte Webseite darstellt. Im Folgenden soll das Verfahren dafür anhand von Abbildung 10 erläutert werden.

Auch in diesem Modus wird für jede wahre Anfrage ein eigener Block erstellt (Schritt 1 in der Abbildung). Anschließend wird nach $N-1$ Mustern der Länge $|M|$ gesucht (Schritt 2), wobei das zu verschleiernde Muster M nicht in Frage kommt. Wenn nicht genug Muster dieser Länge vorhanden sind, werden zwei zufällige Zahlen x und y mit $x+y = |M|$ gewählt und es wird je ein Muster der Länge x und eines der Menge y gewählt, die konkateniert werden, um ein kombiniertes Muster der Länge $|M|$ zu erzeugen (unterste Zeile in der Abbildung). Existieren keine Muster der Länge x und/oder y, so werden x und y neu gewählt, bis Werte gefunden wurden, für die

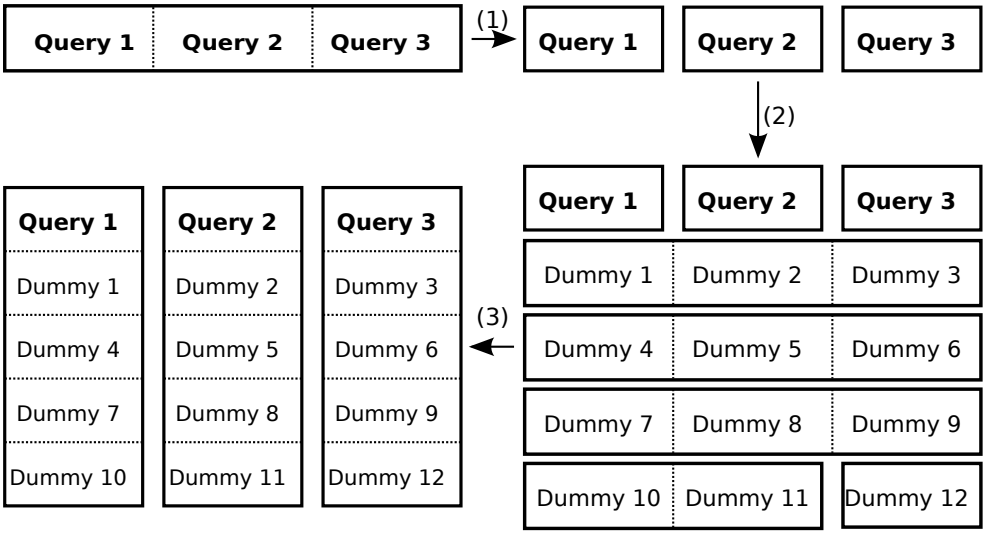


Abbildung 10: Beispiel für die Generierung eines Range Query mit musterbasierter Auswahl, Musterlänge 3, $N = 5$

Muster existieren. Dieser Prozess wird wiederholt, bis insgesamt $N-1$ Muster oder kombinierte Muster gewählt wurden.[13]

Die gewählten Muster werden nun in ihre einzelnen Anfragen zerlegt und diese jeweils zu Blöcken kombiniert (Schritt 3). Dabei bildet das erste Element jedes Musters den ersten Block, das zweite Element den zweiten Block, und so weiter.

Sollte eine Anfrage in mehreren Mustern an derselben Position stehen, werden alle bis auf ein Duplikat entfernt. Die Blockgröße dieses Blockes sinkt entsprechend.

In der Realität ist diese Strategie schwierig umzusetzen, da $|M|$ nicht zwangsweise zu Beginn der Anfrage bekannt ist, was die Auswahl passender Muster zur Verschleierung erschwert. Allerdings sollte es für die meisten Webseiten möglich sein, eine solche Aussage zu treffen, wenn eine Datenbank der Muster angelegt und aktuell gehalten wird, da viele Seiten ihre Inhalte nicht oft ändern. Eine Ausnahme bilden zum Beispiel Seiten mit Benutzerinhalten (Soziale Netzwerke, Linksammlungen wie Reddit, ...). Diese Problematik wurde für die Implementation nicht beachtet, der Simulator geht davon aus, dass die Musterlängen bekannt sind.

Diese Strategie stellt die Modi 4 bis 6 des Simulators:

- `-m 4`: Musterbasierte Auswahl, vollständig ununterscheidbare Blöcke.
- `-m 5`: Musterbasierte Auswahl, erster Block unterscheidbar.
- `-m 6`: Musterbasierte Auswahl, vollständig unterscheidbare Blöcke

[13] In der Implementation im Simulator wurde dieser Algorithmus leicht modifiziert, da er, wenn nicht ausreichend Muster zur Verfügung standen, nicht terminierte. Stattdessen wird jetzt geordnet durch alle Zahlenpaare, die sich auf die korrekte Länge summieren, iteriert. Diese Änderung sorgt für keine Veränderung der Erkennungsleistung, sollte aber in der Realität vermieden werden, da es die Musterauswahl vorhersehbar und damit anfällig für Heuristiken macht.

6.2.1 Beispiel

Für unser Beispiel verwenden wir weiterhin das Muster $M = \langle$*google.com*, *google.de*, *ssl.gstatic.com*$\rangle$.

1. Auch hier beginnt der Client damit, das Muster in seine einzelnen Anfragen zu zerlegen. Wir erhalten wieder $S = \langle\langle$*google.com*$\rangle, \langle$*google.de*$\rangle, \langle$*ssl.gstatic.com*$\rangle\rangle$.
2. Nun gilt es, $N-1$ zufällige Muster der Länge $|M| = 3$ zu ziehen, also aus der Menge $P_3 \backslash M$.[14] Mit $N = 5$ erhält der Client nun zum Beispiel die Dummy-Muster $D =$
   $\langle\langle$*joomlapolis.com*, *stat.joomlapolis.com*, *scanalert.com* $\rangle$,
   $\langle$*211expert.com*, *kaiyuan.hudong.com*, *211brand.com*$\rangle$,
   $\langle$*gebruikershandleiding.com*, *google-analytics.com*, *gstatic.com* $\rangle$,
   $\langle$*frameweb.com*, *google-analytics.com*, *ajax.googleapis.com*$\rangle\rangle$.

   In dem Fall, dass nicht genug Muster der Länge $|M|$ verfügbar sind, wählt das System für jedes fehlende Muster zwei Muster A und B, sodass $|A| + |B| = |M|$ gilt, und verwendet diese als Ersatz.
3. Die einzelnen Anfragen dieser Muster werden jetzt auf die ihrer Position entsprechenden Blöcke von S verteilt. Der Musteranfang wird in S_1 einsortiert, die zweite Anfrage in S_2, und so weiter. Wir erhalten $S = \langle\langle$*google.com*, *joomlapolis.com*, *211expert.com*, *gebruikershandleiding.com*, *frameweb.com*$\rangle$,
   $\langle$*google.de*, *stat.joomlapolis.com*, *kaiyuan.hudong.com*, *google-analytics.com*$\rangle$[15],
   $\langle$*ssl.gstatic.com*, *scanalert.com*, *211brand.com*, *gstatic.com*, *ajax.googleapis.com*$\rangle\rangle$.
4. Der Angreifer verfährt nun analog zum Verfahren in Abschnitt 4.3.

6.2.2 Erwartete Ergebnisse

Auch hier sind je nach gewähltem Modus verschiedene Erkennungsleistungen zu erwarten, daher werden auch hier die einzelnen Modi unterschieden. Im Gegensatz zur Analyse der zufälligen Auswahl soll hier allerdings auf eine stochastische Analyse verzichtet werden, da sich das Verhalten der Modi argumentativ einfacher erläutern lässt.

Vollständig ununterscheidbare Blöcke Im Modus der vollständig ununterscheidbaren Blöcke ist generell nach wie vor die schlechteste Erkennungsleistung zu erwarten. Die intuitive Erwartung wäre, dass immer genau N Ergebnisse erkannt werden, da das Originalmuster mit $N-1$ weiteren Mustern ergänzt wurde. Tatsächlich sind jedoch noch größere Ergebnismengen zu erwarten. Dafür gibt es zwei Gründe: Zum einen werden bei größeren Musterlängen, von denen weniger als N Muster bekannt wird, zwei Muster für jedes fehlende Muster eingefügt, wodurch

[14] $A \backslash B$ bezeichnet wie immer die Menge A ohne die Elemente der Menge B.
[15] *google-analytics.com* kommt in diesem Block zwei mal vor, das Duplikat wurde allerdings sofort entfernt.

der Angreifer mehr als N Muster erkennen wird. Zum anderen kann es auch zufällig zu weiteren Mustern kommen, wenn alle Bestandteile eines nicht gewählten Musters in verschiedenen gewählten Mustern auftreten. Ein auf diese Art entstandenes Muster ist für den Angreifer nicht von einem normalen Muster zu unterscheiden und wird daher ebenfalls gezählt.

Prinzipiell ließe sich durch die Analyse der Verteilung der Musterlängen die Anzahl der Ergebnisse noch stark reduzieren (es ist zum Beispiel anzunehmen, dass *in den meisten Fällen* die am öftesten erkannte Musterlänge die Länge des gesuchten Musters ist), aber eine solche Analyse ist fehleranfällig, und viele solche Einschränkungen könnten durch geschickte Auswahl der Dummy-Muster ausgehebelt werden. Solche Heuristiken werden wir später in Kapitel 10 behandeln.

Erster Block unterscheidbar Bei einem unterscheidbaren ersten Block steht zu erwarten, dass es zu maximal N möglichen Ergebnissen kommen wird. Der erste Block muss weiterhin den Beginn des Musters enthalten, enthält allerdings aus den Beginn von $N-1$ anderen Mustern (wenn ein fehlendes Muster durch zwei Muster kürzerer Länge ersetzt wird, kann sich nur ein Musteranfang im ersten Block befinden).

Die Ergebnisse sollten daher (sofern keine weiteren Heuristiken zur Identifikation des wahren Musters, wie z.B. eine Analyse der Längen, angewendet werden) immer bei exakt N erkannten Mustern liegen. Auch hier ließen sich die Ergebnisse über die weiter oben genannten Methoden reduzieren, diese bleiben allerdings fehleranfällig und sind weiterhin leicht durch Änderungen am Generator auszuhebeln, daher wurde für diese Bachelorarbeit auf solche Änderungen verzichtet.

Vollständig unterscheidbare Blöcke Bei vollständig unterscheidbaren Blöcken ist auch bei dieser Generationsstrategie die Länge des Original-Musters bekannt. Dies bedeutet, dass Dummy-Muster der falschen Länge direkt ausgeschlossen werden können. Entsprechend wird der Angreifer die vollen N Muster als Möglichkeiten erhalten, solange $N \leq |P_{|M|}|$ gilt (N also kleiner oder gleich der Anzahl der Muster der richtigen Länge in der Datenbasis des Clients ist). Sobald diese Ungleichung nicht mehr gilt, wird die Anzahl der Ergebnisse, die der Angreifer erhält, auf $|P_{|M|}|$ reduziert. Zusammengefasst erhält der Angreifer also $\texttt{min}(N, |P_{|M|}|)$ mögliche Ergebnisse.

Da die Länge des Musters bekannt ist, ist ein weiteres Einschränken der Ergebnismenge nicht mehr ohne weiteres möglich. In der realen Welt wäre es möglich, anhand des bisherigen Verhaltens des Nutzers darauf zu schließen, welche der zur Auswahl stehenden Webseiten wahrscheinlicher sind. Auch denkbar wäre die Auswahl des wahrscheinlichsten Ergebnisses anhand einer Analyse der Position der zur Auswahl stehenden Webseiten in der Alexa-Rangliste [1] oder ähnlicher Ranglisten, unter der Annahme, dass der Besuch einer populären Webseite wahrscheinlicher ist als der einer vergleichsweise unbekannten. Derartige Einschränkungen sind allerdings nicht als Teil dieser Bachelorarbeit analysiert worden, und ließen sich teilweise ebenfalls durch

verbesserte Generationsstrategien mit heuristischer Auswahl der Dummies umgehen. Einige Vorschläge, sowohl für Angreifer-Heuristiken als auch für Generationsstrategien, werden in Kapitel 10 aufgegriffen.

7 Implementierte Angriffsstrategien

Das Ziel des Angreifers ist, mit möglichst großer Sicherheit festzustellen, welche Webseite vom Client besucht wurde. Dazu gehört sowohl die Erkennung von Mustern in den Anfrageblöcken, die der Client generiert und der Angreifer erhält, als auch das Ausschließen von Mustern, die zwar auftreten, aber auf Grund verschiedener Ausschlusskriterien nicht das gesuchte Muster sein können (etwa weil die Länge des Musters nicht innerhalb eines speziellen Bereiches liegt).

Je nach Grundannahme über das Format, in dem der Angreifer die Daten des Clients enthält (ununterscheidbare Blöcke, unterscheidbarer erster Block, komplett unterscheidbare Blöcke, siehe Abschnitt 6) sind verschiedene Angriffsstrategien möglich. Daher wurde für jede dieser Grundannahmen mindestens ein Algorithmus für einen Angriff entworfen.

Die Algorithmen für ununterscheidbare und vollständig unterscheidbare Blöcke sind mit beiden Generationsstrategien (zufällige und musterbasierte Auswahl) kompatibel, nur bei dem unterscheidbaren ersten Block unterscheiden sich die verwendeten Algorithmen, da die bei der zufälligen Auswahl geltenden Einschränkungen der Anzahl der möglichen Duplikate bei der musterbasierten Ergänzung nicht mehr gelten.

Im Folgenden sollen alle implementierten Angriffsstrategien vorgestellt werden. Dafür wird unter anderem Pseudocode verwendet. Der verwendete Pseudocode nutzt dabei teilweise mathematische Symbole, insbesondere aus der Mengenlehre. Des Weiteren gelten alle bisher verwendeten Definitionen (wie z.B. Q als Menge aller Anfragen) als Variablen, die direkt genutzt werden können.

7.1 Vollständig ununterscheidbare Blöcke

Der Angriff auf vollständig ununterscheidbare Blöcke ist gleichzeitig der worst case unseres Simulators und der einfachste Algorithmus, da wir keinerlei Einschränkungen nutzen können, um die Menge der möglichen Ergebnisse zu verringern.

Der Algorithmus 1 iteriert durch alle Elemente der Eingabemenge und überprüft, ob das Element ein Musteranfang ist. Sollte dies der Fall sein, überprüft es, ob die Schnittmenge zwischen dem aktuell überprüften Muster von e und dem Anfrageblock B gleich dem Muster von e ist. Wenn dies der Fall ist (das Muster von e also komplett in B enthalten ist), ist e ein Kandidat für das gesuchte Muster und wird in die Liste der Resultate R gespeichert.

Sollte die Schnittmenge nicht dem Muster von e entsprechen, kann (unter unseren Annahmen) das Muster von e nicht das gesuchte Muster sein, da wir davon ausgehen, dass sämtliche Anfragen, die zur Auflösung des Musters benötigt werden, von uns empfangen werden.

Algorithm 1 Algorithmus für vollständig ununterscheidbare Blöcke

```
1:  function ATTACK(B)                          ▷ B ist die Menge der Anfragen, die generiert wurde
2:      R = [ ]                                 ▷ Initialisiere die Ergebnismenge als leere Liste
3:      for all e ∈ B do                        ▷ Iteriere durch alle Elemente von B
4:          if e ∈ P then                       ▷ Wenn e ein Musteranfang ist...
5:              if PATTERNOF(e) ∩B = PATTERNOF(e) then
6:                                              ▷ ...und das Muster von e komplett in B enthalten ist
7:                  R = R + e                   ▷ ...füge e der Liste der Ergebnisse hinzu.
8:              end if
9:          end if
10:     end for
11:     return R                                ▷ Gib die Liste der Ergebnisse zurück
12: end function
```

7.2 Unterscheidbarer erster Block

Der unterscheidbare erste Block ist der realistischte Fall für eine Anwendung in der realen Welt. Für diesen Modus betrachten wir zwei Algorithmen: Einen regulären Algorithmus (Algorithmus 2), der sowohl für die zufällige Auswahl der Dummies als auch für die musterbasierte Auswahl funktioniert, sowie einen auf die zufällige Auswahl von Anfragen optimierten Algorithmus (Algorithmus 3), der mit der musterbasierten Auswahl inkompatibel ist.

Zuerst betrachten wir den allgemeingültigen Algorithmus:

Algorithm 2 Algorithmus für unterscheidbaren ersten Block, Version 1

```
1:  function ATTACK(F, B)                       ▷ F: Erster Block, B: Restliche Anfragen
2:      R = [ ]                                 ▷ Initialisiere die Ergebnismenge als leere Liste
3:      B = B ∪ F      ▷ Setze B auf die Vereinigung von B und F (für spätere Schnittmengen)
4:      for all e ∈ F do                        ▷ Iteriere über alle Elemente aus F
5:          if e ∈ P then                       ▷ Wenn e ein Musterbeginn (aus P) ist...
6:              if PATTERNOF(e) ∩B = PATTERNOF(e) then
7:                                              ▷ ...und das Muster von e komplett in B enthalten ist...
8:                  R = R + e                   ▷ ...füge e zur Liste der Ergebnisse hinzu
9:              end if
10:         end if
11:     end for
12:     return R                                ▷ Gib die Liste der Ergebnisse zurück
13: end function
```

Als Parameter erhält der Algorithmus 2 den ersten Block und die restlichen Anfragen als zwei getrennte Mengen F und B. Zuerst werden B und F vereinigt, damit später das Bestimmen der Schnittmengen mit den infrage kommenden Mustern einfacher wird. Anschließend iteriert der

Algorithmus durch alle Elemente des ersten Blocks F und prüft für jedes, ob es ein Musteranfang ist.

Für jeden gefundenen Musteranfang wird anschließend wieder die Schnittmenge zwischen dem zugehörigen Muster und B (was nun $B \cup F$ enthält) bestimmt und mit dem Muster verglichen. Wenn die Schnittmenge und das Muster gleich sind, ist das Muster komplett in B enthalten und ist ein Kandidat für das gesuchte Muster. Es wird in die Liste der Resultate R gespeichert.

Diesen Algorithmus können wir allerdings noch verbessern: Wie wir in Abschnitt 6.1.2 im Absatz des unterscheidbaren ersten Blocks festgestellt haben, können wir die Genauigkeit über das Einführen einiger Einschränkungen verbessern. Im Folgenden soll dieser verbesserte Algorithmus 3 vorgestellt werden, der allerdings aufgrund der verwendeten Annahmen nur für die zufällige Auswahl der Dummy-Anfragen (`-m 2`) und nicht für die musterbasierte Auswahl (`-m 5`) funktioniert, da für diese die Annahmen über die maximale Anzahl an Duplikaten nicht mehr gelten.

Algorithm 3 Algorithmus für unterscheidbaren ersten Block, Version 2

```
1:  function ATTACK(F, B)                          ▷ F: Erster Block, B: Restliche Anfragen
2:      R = [ ]                                    ▷ Initialisiere die Ergebnismenge als leere Liste
3:      n = |F|                                    ▷ Setze n auf |F|, unsere Vermutung über den Wert von N
4:      B = B ∪ F          ▷ Setze B auf die Vereinigung von B und F (für spätere Schnittmengen)
5:      m2 = ⌈|B|/n⌉                               ▷ Setze m2 auf den erwarteten Höchstwert für M
6:      m2 = m2 + 2 * ⌈(m2/n)⌉     ▷ Erhöhe den oberen Grenzwert, um Duplikate auszugleichen
7:      m1 = ⌈|B|/(n+1)⌉                           ▷ Setze m1 auf den kleinsten möglichen Wert von M
8:      for all e ∈ F do                           ▷ Iteriere durch alle Elemente von F
9:          if (e ∈ P) ∧ (m1 ≤ |PATTERNOF(e)| ≤ m2) then
10:                          ▷ Wenn e ein Musterbeginn eines Musters einer erlaubten Länge ist...
11:             if PATTERNOF(e) ∩B = PATTERNOF(e) then
12:                                  ▷ ...und das Muster von e komplett in B enthalten ist...
13:                 R = R + e                      ▷ ...füge e zur Liste der Ergebnisse hinzu
14:             end if
15:         end if
16:     end for
17:     return R                                   ▷ Gib die Liste der Ergebnisse zurück
18: end function
```

Der optimierte Algorithmus 3 unterscheidet sich vom alten Algorithmus 2 dadurch, dass er die in Abschnitt 6.2.2 gefundenen Einschränkungen der Mustergröße verwendet.

Zuerst werden die Ober- und Untergrenze der möglichen Werte von M bestimmt. Anschließend iteriert der Algorithmus wie gehabt durch alle Elemente des ersten Blocks, allerdings werden hierbei nur die Musteranfänge beachtet, deren Länge im korrekten Intervall liegt.

Anschließend wird wie bisher verfahren: Die Schnittmenge zwischen dem Muster und den Eingabemengen wird gebildet und mit dem Muster verglichen, und wenn beide Mengen gleich sind, ist e ein Kandidat für das gesuchte Muster und wird in die Menge der Resultate R gespeichert.

Der große Vorteil dieses Verfahrens ist, dass, abhängig von den berechneten m_1 und m_2, ein großer Teil der in Frage kommenden Muster ausgeschlossen werden kann. Vor allem bei langen Mustern kann es sonst dazu kommen, dass sich irgendwo in der Eingabemenge noch ein Muster der Länge zwei oder drei versteckt, welches durch Zufall ausgewählt wurde. Diese Fälle werden nun als falsch erkannt und nicht beachtet.

Der Nachteil des Verfahrens ist, dass es nur funktioniert, solange unsere Annahme über die Auswahl der Dummy-Anfragen (zufällig und ohne Zurücklegen) stimmt und der Wert von N über die verschiedenen Blöcke konstant bleibt. Wir können aber immer den ersten Algorithmus verwenden, der in jedem Fall funktioniert, allerdings eine geringere Erkennungsleistung aufweisen dürfte.

7.3 Vollständig unterscheidbare Blöcke

Der Angriff auf vollständig unterscheidbare Blöcke stellt den best case unseres Simulators dar, und soll im Folgenden näher betrachtet werden.

Algorithm 4 Algorithmus für vollständig unterscheidbare Blöcke

```
 1: function ATTACK(B)                          ▷ B: Menge der Blöcke (Menge von Mengen)
 2:     R = [ ]                                 ▷ Initialisiere die Ergebnismenge als leere Liste
 3:     for all p ∈ B_1 do                      ▷ Iteriere durch alle Anfragen p aus dem ersten Block
 4:         if p ∈ P_|B| then                   ▷ Wenn p der Beginn eines Musters der Länge |B| ist...
 5:             done = [0]*|B|                  ▷ done ist ein Array mit |B| Elementen, auf 0 initialisiert
 6:             for all q ∈ PATTERNOF(p) do     ▷ Iteriere durch alle Anfragen im Muster von p
 7:                 for all i ∈ [1, |B|] do     ▷ Iteriere durch alle Indizes von B
 8:                     if q ∈ B_i then         ▷ Wenn q in B_i enthalten ist...
 9:                         done[i] = 1         ▷ ...setze done[i] auf 1...
10:                         break 1             ▷ ...und breche den inneren For-Loop ab
11:                     end if
12:                 end for
13:             end for
14:             if 0 ∉ done then                ▷ Wenn alle Elemente von done 1 sind...
15:                 R = R + p                   ▷ ...ist p ein mögliches Ergebnis
16:             end if
17:         end if
18:     end for
19:     return R                                ▷ Gib die Liste der Ergebnisse zurück
20: end function
```

Der Algorithmus 4 iteriert durch alle Anfragen p im ersten Anfrageblock (B_1) und überprüft, ob sie der Beginn eines Musters der Länge $|B|$ sind (da wir die Länge des gesuchten Musters aus der Anzahl der Blöcke in B eindeutig herleiten können, ist eine solche Überprüfung sinnvoll). Wenn p dieses Kriterium erfüllt, initialisieren wir die Variable *done* als ein Array mit $|B|$ Elementen

(dessen Index bei 1 beginnt und das zu Beginn mit Nullen gefüllt ist) und beginnen, durch alle Anfragen q des Musters von p zu iterieren.

Für jede Anfrage wird durch die einzelnen Blöcke iteriert und überprüft, ob die Anfrage im Block enthalten ist. Wenn dies der Fall ist, wird die Zahl im Array *done* an der Position des Blocks auf 1 gesetzt und der innere For-Loop (über die Blöcke) wird unterbrochen.

Nachdem alle Elemente des Musters von p durchgegangen wurden, wird überprüft, ob *done* noch eine Null enthält. Wenn dies nicht der Fall ist, waren sämtliche Elemente des Musters von p in den verschiedenen Blöcken von B enthalten. Dies impliziert, dass jeder Block ein Element des Musters enthält, was dieses Muster zu einem Kandidaten für die Lösung macht.

8 Analyse der Ergebnisse

Nachdem die Generations- und Angriffsstrategien vorgestellt wurden, wurde die Simulation für jeden der Modi[16] durchgeführt. Dabei wurden zwei Parameter variiert:

- **Blockgröße**: Die Blockgröße wurde variiert, da der bisherige Standardwert von 50 keine wissenschaftliche Grundlage hatte. Intuitiv sollte die Erkennungsleistung des Angreifers bei steigender Blockgröße proportional sinken. Gleichzeitig sorgt eine höhere Blockgröße allerdings für mehr Netzwerkverkehr und mehr Verzögerungen beim Warten auf die Antworten.[17] Daher muss ein guter Kompromiss zwischen Privatsphäre und Leistung gefunden werden.

  Die simulierten Werte für die Blockgröße waren 2, 5, 10, 20, 50, 100, 200 und 500. Damit wurde ein großer Bereich abgedeckt ohne die Simulationszeit unnötig zu erhöhen. Auch wenn der Abstand zwischen den einzelnen Schritten teilweise sehr groß ist, sollten allgemeine Tendenzen sichtbar sein.

- **Datenbasis des Clients**: Wir gehen davon aus, dass der Angreifer im allgemeinen mehr Wissen über die Anfragemuster der relevanten Seiten als der Client hat. Dies hat mehrere Gründe: Nichts hindert den Angreifer daran, die Software, die der Client nutzt, selber zu installieren und eine vorinstallierte Datenbasis auszulesen. Auch wenn die Datenbasis dynamisch erstellt wird, kann der Angreifer selbst die Anfragemuster aller Seiten, an denen er interessiert ist, abrufen und aktuell halten.

  Selbst wenn der Angreifer nicht nur an spezifischen Seiten, sondern an allen Seiten des Internets interessiert ist, ist es realistisch, anzunehmen, dass der Angreifer mehr Ressourcen zur Verfügung hat als der Client, er also problemlos in der Lage ist, mehr Muster als der Client zu sammeln.

  Die verwendeten Datenbasen für den Client waren der gesamte Datensatz (216925 Anfragen), sowie Partitionen von 100, 200, 500, 1000, 2000, 5000, 10000, 20000, 50000,

[16] Es wird die Modus-Nummerierung aus Abschnitt 6.1 bzw. 6.2 verwendet. Modus 1 ist also der Modus der vollständig ununterscheidbaren Blöcke bei zufälliger Auswahl der Dummies, etc.

[17] Zum Einfluss der Blockgröße auf die Geschwindigkeit sei erneut auf die Arbeit von Federrath et al. [5] verwiesen.

100 000 und 200 000 Anfragen. Die Methode zum Erstellen der Partitionen wurde bereits in Abschnitt 5.2.3 vorgestellt.

Im Folgenden sollen nun die Ergebnisse der Simulation analysiert werden. Dabei beginnen wir damit, den Einfluss der Blockgröße bei Verwendung des gesamten Datensatzes für alle Modi zu bestimmen und die Leistung zu vergleichen. Anschließend soll der Einfluss von verschiedenen Datenbasen analysiert werden.

Während der Simulation stellte sich heraus, dass bei sehr kleinen Client-Datenbasen die Vorbedingungen für den optimierten Algorithmus für den Modus des unterscheidbaren ersten Blocks nicht mehr zuverlässig erfüllt waren. Daher wurde für die Simulation mit variablen Client-Datenbasen der nicht-optimierte Algorithmus verwendet. Der optimierte Algorithmus wurde mit der vollen Datenbasis separat simuliert, auf die Ergebnisse und den Vergleich mit dem nicht optimierten Algorithmus werden wir im Folgenden ebenfalls kurz eingehen.

8.1 Variation der Blockgröße

Durch die Blockgröße wird vom Nutzer das Verhältnis zwischen Privatsphäre und Geschwindigkeit bestimmt. Eine höhere Blockgröße sollte, wie weiter oben erwähnt, zu einer besseren Wahrung der Privatsphäre führen, während eine kleinere Blockgröße zu einer höheren Geschwindigkeit führt. Die Blockgröße N enthält das zu verbergende Muster, $N = 2$ bedeutet also, dass ein Dummy pro Block gewählt wird.

Im Folgenden soll versucht werden, anhand der Ergebnisse der Simulation einen guten Mittelweg zwischen Leistung und Privatsphäre zu finden. Dafür werden die Modi getrennt untersucht. Speziell interessiert uns, wie die Erkennungsleistung des Angreifers relativ zur Veränderung der Blockgröße skaliert.

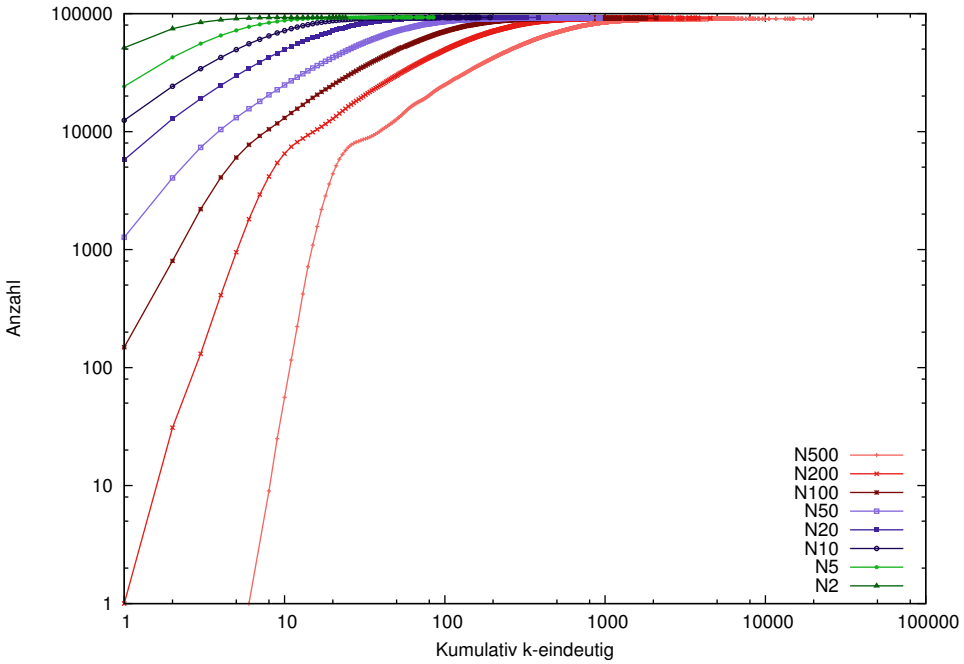
Die abgebildeten Statistiken stellen die kumulative k-Eindeutigkeit dar, also die Anzahl der Muster, die k-eindeutig oder eindeutiger (n-Eindeutig mit $n < k$) identifiziert werden konnten. 1-eindeutig bedeutet, dass nur ein Muster in Frage kam, 2-eindeutig, dass zwei Muster mögliche Lösungen waren, und so weiter.

Des Weiteren wird eine Tabelle angegeben, welche die Anzahl der 1-eindeutigen Muster, den Median[18] der k-Eindeutigkeiten und die maximale k-Eindeutigkeit pro Blockgröße angibt. Des Weiteren wird die Änderungsrate $\triangle$ der Anzahl der 1-eindeutigen Muster angegeben, welche als Differenz der Anzahl der 1-eindeutigen Muster geteilt durch den Unterschied in der Blockgröße definiert ist.

8.1.1 Vollständig ununterscheidbare Blöcke, zufällige Auswahl der Dummies

In diesem Modus ist gut sichtbar, dass die Genauigkeit des Angreifers stark abhängig von der gewählten Blockgröße ist. Abb. 11 zeigt die kumulative k-Eindeutigkeit bei verschiedenen Blockgrößen N auf einer doppelt logarithmierten Skala, simuliert mit dem vollen Datensatz.

[18] Median bedeutet in diesem Fall: Erstes k, bei dem mindestens 50% der Muster k-Eindeutig sind.

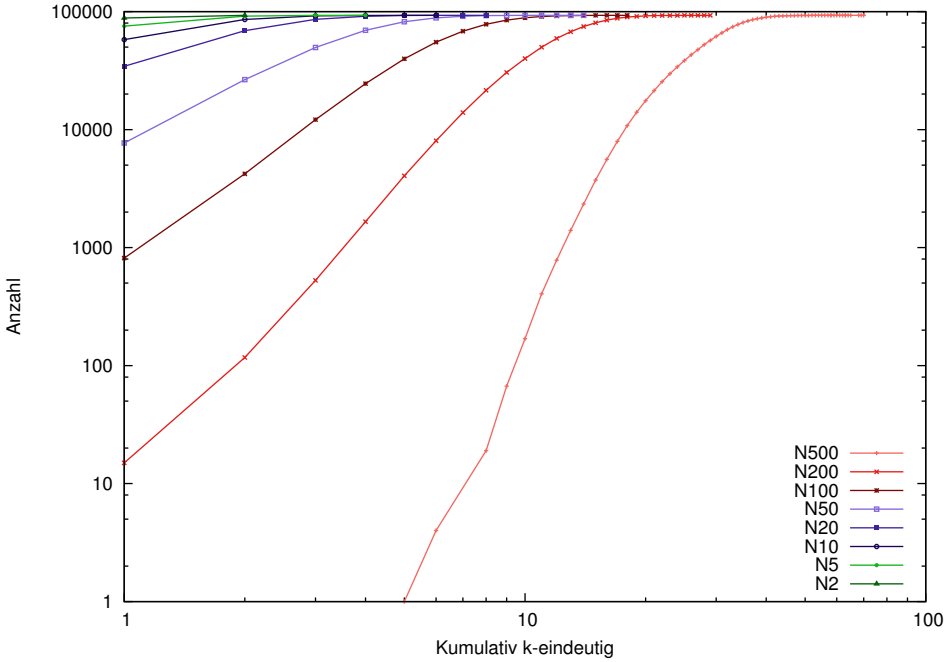


N	1-eindeutig	△	median(k)	max(k)
2	51 396	—	1	24
5	24 121	−9 091.6667	3	85
10	12 469	−2 330.4000	5	192
20	5 765	−670.4000	10	386
50	1 270	−149.8333	23	927
100	149	−22.4200	46	2 083
200	1	−1.4800	92	4 528
500	0	−0.0033	231	19 356

Abbildung 11: Kumulative k-Eindeutigkeit bei ununterscheidbaren Blöcken und zufälliger Auswahl der Dummies, mit verschiedenen Blockgrößen N.

Wie in der Abbildung zu sehen ist, skalieren die Genauigkeiten des Angreifers relativ zur gewählten Blockgröße N. Allerdings skaliert es nicht linear, wie ein Blick auf die Anzahl der 1-eindeutig erkennbaren Muster für die verschiedenen N in der Tabelle zeigt. Stattdessen sinkt die Genauigkeit bei den ersten Schritten von N sehr stark (von 51 396 1-eindeutig identifizierbaren Mustern bei $N = 2$ zu nur noch 1 270 1-eindeutig identifizierbaren Mustern bei $N = 50$), die letzten Schritte ergeben allerdings nur noch vergleichsweise kleine Veränderungen (um die letzten 1 270 Muster zu tarnen, müssen wir das N bis auf 200 erhöhen, und selbst dann existiert noch ein 1-eindeutig identifizierbares Muster). Interessanterweise liegt der median(k) immer in einem Bereich um $N/2$, der Median skaliert also anscheinend annähernd linear zu N.

Dieses Ergebnis zeigt uns, dass eine Erhöhung der Blockgröße zwar die Genauigkeit des Angreifers senkt, allerdings nicht überall linear. Der Median skaliert zwar annähernd linear zu N, der in der Praxis interessantere Wert der Anzahl der 1-eindeutigen Muster allerdings nicht. Ab einem



N	1-eindeutig	$\triangle$	median(k)	max(k)
2	88 173	—	1	2
5	75 346	−4 275.6667	1	4
10	57 875	−3 494.2000	1	6
20	34 426	−2 344.9000	2	8
50	7 693	−891.1000	3	14
100	815	−137.5600	6	18
200	15	−8.0000	11	29
500	0	−0.0500	27	70

Abbildung 12: Kumulative k-Eindeutigkeit bei einem unterscheidbaren ersten Block und zufälligere Auswahl der Dummies, mit verschiedenen Blockgrößen N

bestimmten Wert lohnt sich das Erhöhen der Blockgröße nicht mehr, da die zusätzliche Sicherheit gegen einen Angreifer in keinem Verhältnis zur zusätzlichen Zeit, die die Verarbeitung der weiteren Anfragen dauert, steht. Wo genau diese Grenze liegt, muss durch abwägen zwischen dem Privatsphärebedürfnis des Nutzers und der gewünschten Geschwindigkeit beim Abrufen von Internetseiten bestimmt werden, vermutlich liegt sie aber zwischen $N = 20$ und $N = 100$.

8.1.2 Unterscheidbarer erster Block, zufällige Auswahl der Dummies - regulärer Algorithmus

Wir betrachten zuerst den regulären (nicht optimierten) Algorithmus für diesen Modus. Der optimierte Algorithmus wird in einem eigenen Abschnitt betrachtet.

Was bei einer Betrachtung des Graphen in Abb. 12 sofort auffällt: Das Maximum der kumulativen

k-Eindeutigkeit ist nicht mehr im Bereich von 11 000 wie bei Modus 1, stattdessen liegt es bei 70 ($N = 500$) bzw. 29 ($N = 200$). Dies ist ein dramatischer Unterschied zum vorherigen Modus, vor allem wenn man beachtet, dass dieser Modus der Realität vermutlich am nähesten kommt. Selbst bei realistisch hoch gewählten Blockgrößen wie $N = 50$ ist der Angreifer nach wie vor in der Lage, selbst ohne weitere Optimierungen alle Muster mindestens 14-eindeutig zu identifizieren, und Blockgrößen wie $N = 500$, die ein Maximum von 70-eindeutig erreichen, sind in der Realität unpraktikabel.

Wenn man die Tabelle betrachtet, stellt man fest, dass die Anzahl der 1-eindeutig identifizierbaren Muster weiterhin bei steigendem N sinkt. Allerdings sinkt die Zahl langsamer und beginnt bei einer höheren Anzahl als bei Modus 1 (vgl. Tabelle in Abb. 11).

Zusammengefasst kann man sagen, dass das Verfahren in diesem Modus selbst mit sehr großen N einen unverhältnismäßig kleinen Gewinn an Privatsphäre bietet.

8.1.3 Unterscheidbarer erster Block, zufällige Auswahl der Dummies - optimierter Algorithmus

Nachdem wir im letzten Abschnitt den regulären Algorithmus für den Modus 2 betrachtet haben, folgt nun eine Betrachtung des optimierten Algorithmus, der in Abschnitt 7.2 vorgestellt wurde. Wie bereits weiter oben erwähnt wurde, funktioniert dieser Algorithmus nicht für beliebig kleine Datensätze (oder, genauer gesagt, wenn $|M| * N > |Q|$ gilt, also mehr Dummy-Anfragen benötigt werden als vorhanden sind). Da dies aber bei einem realen System in der Regel nicht der Fall sein sollte, wird der Algorithmus trotzdem betrachtet.

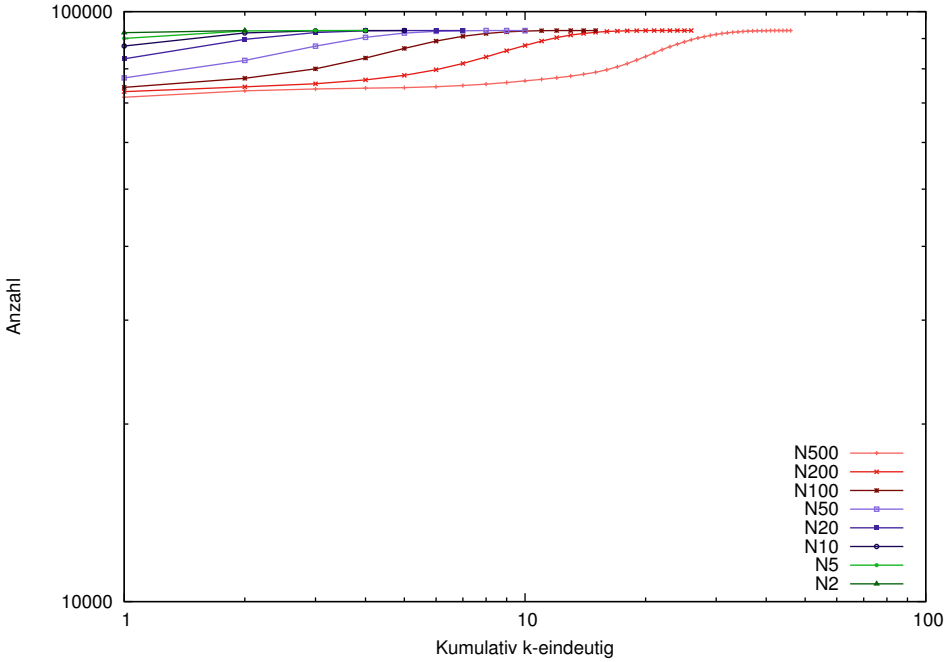
Eine Betrachtung von Abb. 13 zeigt, dass, egal welches N gewählt wird, mindestens 71 596 der 92 889 Muster sofort 1-eindeutig identifiziert werden können. Der Einfluss von N ist dabei vergleichsweise gering, wobei ein höheres N für einen höheren Maximalwert der k-Eindeutigkeit sorgt, der allerdings selbst bei $N = 500$ maximal 48 erreicht.

Diese Ergebnisse bedeuten, dass bei einem optimierten Angreifer selbst mit unpraktikabel hohen Blockgrößen wie $N = 500$ ein großer Anteil der aufgerufenen Webseiten 1-eindeutig vom Angreifer identifiziert werden kann. Im nächsten Abschnitt wird sich zeigen, dass dies beinahe der Leistung mit vollständig unterscheidbaren Blöcken entspricht.

8.1.4 Vollständig unterscheidbare Blöcke, zufällige Auswahl der Dummies

Auch in diesem Modus ist die Genauigkeit sehr hoch. Wie wir in Abb. 14 ablesen können, liegt die geringste Anzahl von 1-eindeutigen Mustern bei knapp unter 80 000 Mustern für $N = 500$, und noch höher für kleinere Werte von N. Auch hier unterscheiden sich die verschiedenen N vergleichsweise wenig, die maximale k-Eindeutigkeit beträgt 37 für $N = 500$.

Interessanterweise unterscheidet sich die Leistung nur wenig von der des optimierten Algorithmus für Modus 2. Die Leistung ist zwar besser (die minimale Anzahl von 1-eindeutigen Mustern liegt bei 79 397 anstatt den 71 596 vom optimierten Modus 2, vgl. Abb. 14 bzw. 13), aber



N	1-eindeutig	△	median(k)	max(k)
2	92089	—	1	2
5	90063	−675.3333	1	4
10	87422	−528.2000	1	5
20	83185	−423.7000	1	7
50	77222	−198.7667	1	10
100	74405	−56.3400	1	16
200	73183	−12.2200	1	26
500	71596	−5.2900	1	48

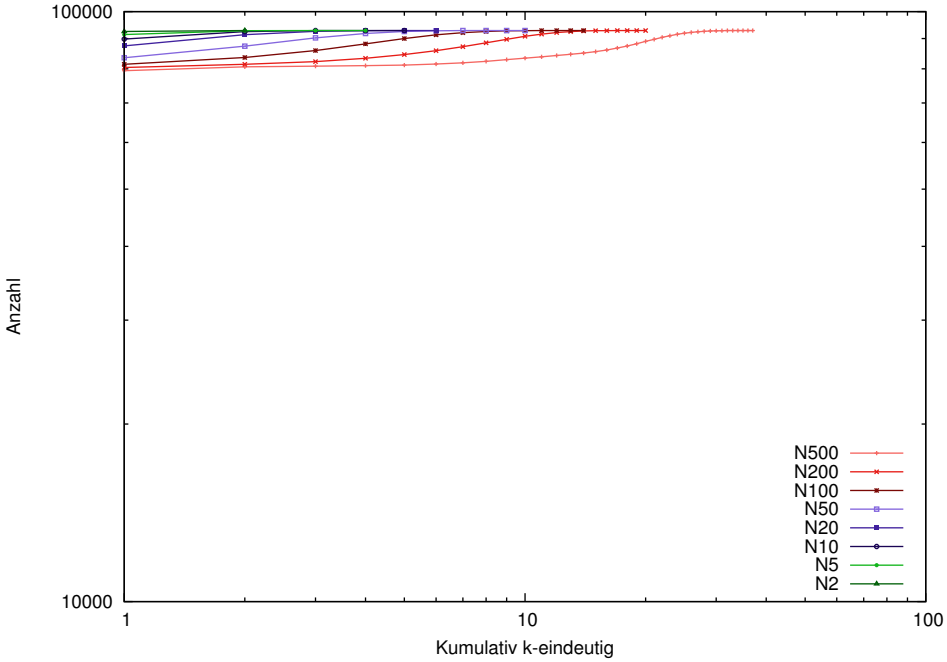
Abbildung 13: Kumulative k-Eindeutigkeit bei Verwendung des optimierten Algorithmus für unterscheidbare erste Blöcke bei zufälliger Auswahl der Dummies, mit verschiedenen Blockgrößen N.

im Vergleich zur Verbesserung gegenüber dem regulären Modus' 2 mit seinen null 1-eindeutig identifizierbaren Mustern bei $N = 500$ ist der Unterschied sehr gering.

Bei der Nutzung des regulären DNS Range Query in der Form, in der es von Zhao et al. vorgeschlagen wurde, macht es also fast keinen Unterschied, ob der Angreifer einzelne Blöcke auseinanderhalten kann oder nicht, solange er in der Lage ist, den ersten Block zu identifizieren. Dass dies technisch problemlos möglich sein sollte, hatten wir bereits in Kapitel 6 festgestellt.

8.1.5 Vollständig ununterscheidbare Blöcke, musterbasierte Auswahl der Dummies

Wir beginnen nun mit dem ersten Modus, der die musterbasierte Auswahl der Anfragen implementiert. Wie man in Abb. 15 sehen kann, ist der Erfolg sofort sichtbar. Kein einziges Muster



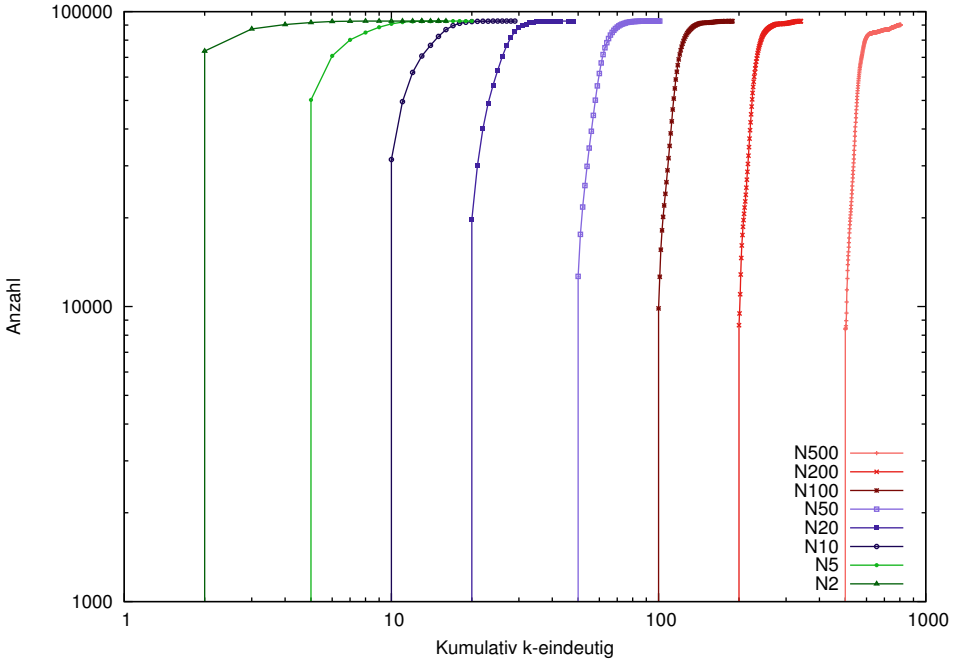
N	1-eindeutig	△	median(k)	max(k)
2	92518	—	1	2
5	91446	−357.3333	1	4
10	89853	−318.6000	1	5
20	87457	−239.6000	1	6
50	83524	−131.1000	1	10
100	81425	−41.9800	1	14
200	80396	−10.2900	1	22
500	79397	−3.3300	1	37

Abbildung 14: Kumulative k-Eindeutigkeit bei vollständig unterscheidbaren Blöcken und zufälliger Auswahl der Dummies, mit verschiedenen Blockgrößen N.

ist besser als N-eindeutig identifizierbar, egal, welches N gewählt wurde. Des Weiteren kommen weitere Muster hinzu, die zufällig entstehen, da sie Teilmengen der Vereinigung aller Dummy-Muster sind. Diese Muster wurden nicht explizit gezogen, sondern sind aufgrund ihrer Überschneidungen zu den anderen Mustern aufgetreten. Dies belegt die These, dass selbst mit den durchschnittlich sehr niedrigen Jaccard-Koeffizienten, die in Abschnitt 5.1.6 bestimmt wurden, das Auftreten weiterer Muster bei weitem nicht unmöglich ist.

Bei steigenden Werten für N ergeben sich erwartungsgemäß in immer mehr Fällen zufällig weitere Muster, da immer mehr Anfragen zur Verfügung stehen, in denen sie auftreten können. Das Erhöhen der Blockgröße N von 5 auf 50 ergibt also durchschnittlich *mehr* als $50-5=45$ weitere erkannte Muster.

Es lässt sich vorwegnehmen, dass dieser Modus die für den Nutzer beste Privatsphäre ergibt und am besten mit steigenden N skaliert. Leider ist es ebenfalls der unrealistischte Modus, da sowohl



N	1-eindeutig	△	median(k)	max(k)
2	0	—	2	16
5	0	0	5	20
10	0	0	11	30
20	0	0	23	49
50	0	0	58	104
100	0	0	113	189
200	0	0	223	341
500	0	0	549	806

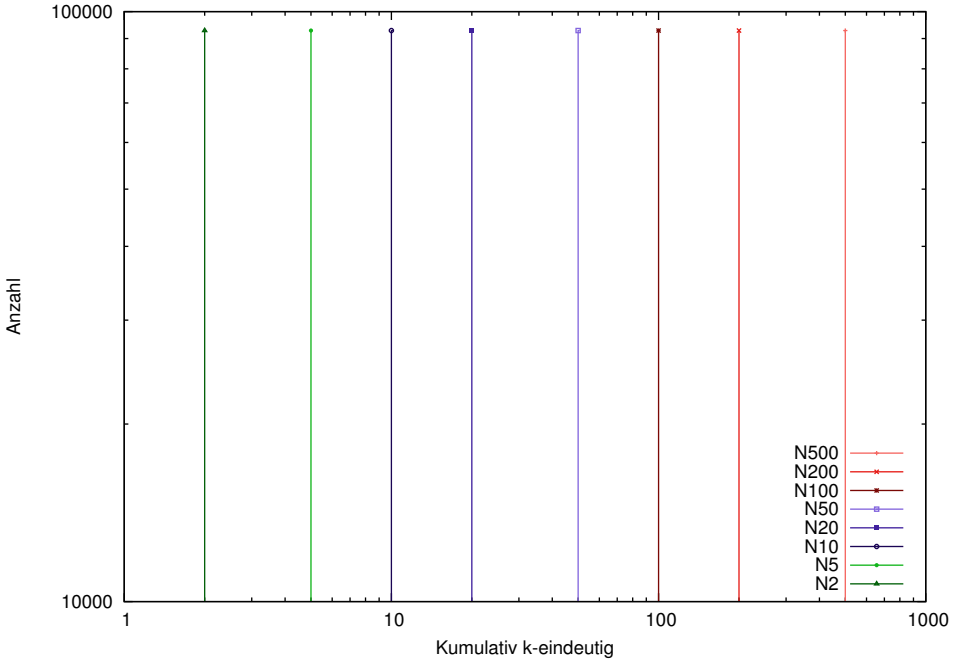
Abbildung 15: Kumulative k-Eindeutigkeit bei ununterscheidbaren Blöcken und musterbasierter Auswahl der Dummies, mit verschiedenen Blockgrößen N

die Annahme der komplett ununterscheidbaren Blöcke aus den zuvor beschriebenen Gründen nicht realistisch ist, als auch die Strategie der musterbasierten Auswahl der Dummies in der Praxis viele Probleme hat, die in Abschnitt 6.2 bereits erwähnt wurden.

8.1.6 Unterscheidbarer erster Block, musterbasierte Auswahl der Dummies

Auch in diesem Modus ist kein Muster besser als N-eindeutig identifizierbar. Wenn man allerdings Abb. 16 betrachtet, stellt man fest, dass auch kein Muster schlechter als N-eindeutig identifizierbar ist.

Dies liegt daran, dass der Musteranfang sich im ersten Block befinden muss. Die Größe des ersten Blocks ist immer genau N, entsprechend kann es maximal zu N erkannten Mustern kommen.



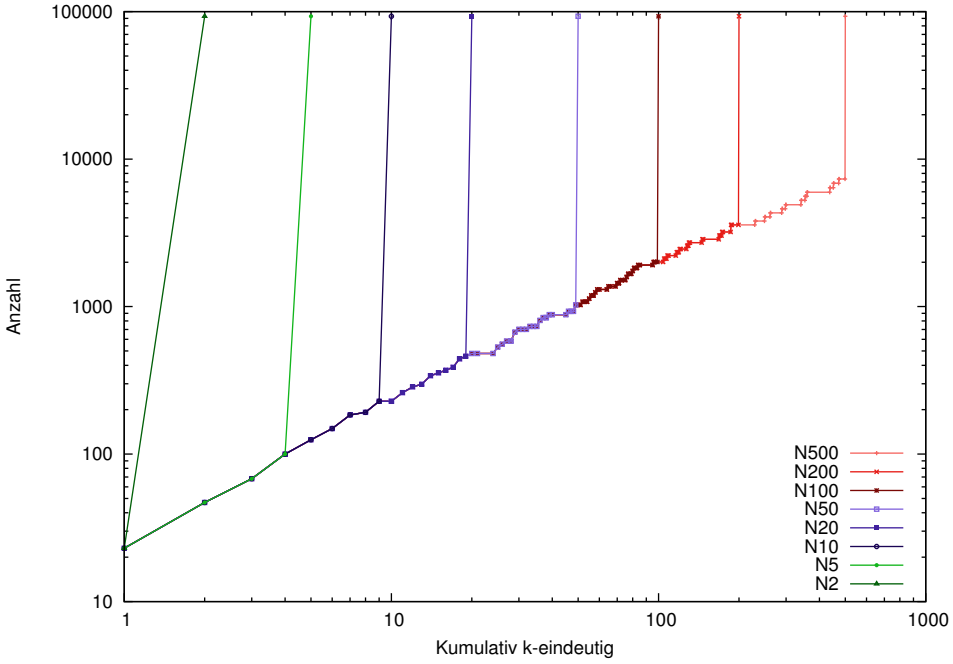
N	1-eindeutig	△	median(k)	max(k)
2	0	—	2	2
5	0	0	5	5
10	0	0	10	10
20	0	0	20	20
50	0	0	50	50
100	0	0	100	100
200	0	0	200	200
500	0	0	500	500

Abbildung 16: Kumulative k-Eindeutigkeit bei einem unterscheidbaren ersten Block und musterbasierter Auswahl der Dummies, mit verschiedenen Blockgrößen N.

Gleichzeitig hat der Angreifer allerdings keine Möglichkeit, festzustellen, welche Länge das korrekte Muster hat, daher können keine Muster ausgeschlossen werden. Da der Client die Blöcke mit Mustern auffüllt, wird der Angreifer also immer genau N Muster erkennen.

Ein Spezialfall ist, wenn keine N Muster der korrekten Länge in der Datenbasis vorhanden sind. In diesem Fall wählt der Client bekanntlich zwei Muster, deren Längen sich auf die korrekte Länge summieren, und konkateniert diese. Auch in diesem Fall befindet sich nur einer der Musteranfänge im ersten Block, es werden also weiterhin nur N Muster gefunden. Die einzige Müglichkeit, weniger als N Muster zu erhalten, ist, wenn auch von diesen Mustern nicht ausreichend zur Verfügung stehen, um die Blöcke auf N Anfragen aufzufüllen. Dies sollte jedoch bei ausreichend großen Datenbasen nicht geschehen.

Dadurch gilt, dass die Erkennungsleistung des Angreifers exakt mit dem gewählten N skaliert, es ergibt sich die von Zhao et al. angegebene Ratewahrscheinlichkeit für den Angreifer von $\frac{1}{N}$.



N	1-eindeutig	△	median(k)	max(k)
2	23	—	2	2
5	23	0	5	5
10	23	0	10	10
20	23	0	20	20
50	23	0	50	50
100	23	0	100	100
200	23	0	200	200
500	23	0	500	500

Abbildung 17: Kumulative k-Eindeutigkeit bei vollständig unterscheidbaren Blöcken und musterbasierter Auswahl der Dummies, mit verschiedenen Blockgrößen N.

Da allerdings die musterbasierte Auswahl weiterhin in der Praxis unrealistisch ist, und sich weitere Optimierungen[19] finden ließen, um die Anzahl der in Frage kommenden Muster zu senken, bedeutet dies nicht, dass das Range Query tatsächlich eine in der Realität praktikable Lösung ist.

8.1.7 Vollständig unterscheidbare Blöcke, musterbasierte Auswahl der Dummies

Auf den ersten Blick wirkt das Bild der kumulierten Verteilungsfunktion in Abb. 17 überraschend. Eine derartig lineare Funktion auf einer doppelt logarithmierten Skala, bei der sich alle

[19] Bei dem aktuell verwendeten Generator ließe sich z.B. ausnutzen, dass das korrekte Muster eines der Muster mit der maximalen Länge der gefundenen Muster sein muss. Dies wurde nicht verwendet, da es sich einfach durch eine alternative Generationsstrategie verhindern ließe und der daraus entstehende „Wettlauf“ zwischen Generationsstrategien und Angriffsstrategien den Umfang der Bachelorarbeit gesprengt hätte. Einige Ansätze werden in Kapitel 10 vorgestellt

Funktionen trotz unterschiedlicher Parameter exakt überlagern, scheint intuitiv unwahrscheinlich zu sein. Der Grund dafür wird allerdings offensichtlich, wenn man sich die Funktion des Angreifers und des Generators (Clients) vor Augen führt.

Der Generator versucht, möglichst viele Muster derselben Länge zu verwenden, um das korrekte Muster zu verschleiern. Wenn allerdings nicht genügend Muster der korrekten Länge verfügbar sind, füllt er die fehlenden Muster mit Kombinationen aus zwei kürzeren Mustern auf, die sich auf die korrekte Musterlänge addieren.

Der Angreifer kennt die korrekte Musterlänge, da er die Anzahl der Blöcke kennt. Entsprechend kann er alle derart ausgewählten Muster sofort ausschließen, da sie die falsche Länge haben. Bei den übrig bleibenden Mustern kann er allerdings das korrekte Muster nicht von den Dummies unterscheiden.

Entsprechend beträgt die k-Eindeutigkeit eines Musters, wie bereits in Abschnitt 6.2.2 beschrieben wurde, das Minimum von N und der Anzahl der Muster dieser Länge. Dies bedeutet folgerichtig, dass wir 23 1-eindeutige Muster erhalten, da es 23 Muster mit einmaligen Musterlängen gibt, wodurch diese 23 Muster 1-eindeutig identifizierbar sind. Die kumulative k-Eindeutigkeit steigt also mit der Anzahl der Musterlängen, von denen der Datensatz k oder weniger Muster dieser Länge enthält. Diese Steigung ist linear auf einer doppelt logarithmierten Skala, was das Bild der Funktion in Abb. 17 erklärt.

Diese Eigenschaft macht den Vergleich der k-Eindeutigkeiten bei verschiedenen N uninteressant, da die k-Eindeutigkeiten zweier N gleich sind, solange das k noch keines der beiden N überschritten hat. Sobald das k ein N überschreitet, wird es sofort den maximalen Wert von 92 889 für dieses N annehmen.

Entsprechend skaliert die Erkennungsleistung des Angreifers nur in ihrer Obergrenze (dem k, bei dem alle Muster k-eindeutig oder besser identifiziert werden können) durch Veränderung des N, welche, wie bei der vorherigen Funktion auch, „N-eindeutig“ beträgt.

8.1.8 Fazit zur Variation der Blockgröße

Unsere bisherigen Simulationen haben gezeigt, dass die von Zhao et al. angegebene Ratewahrscheinlichkeit für den Angreifer von $\frac{1}{N}$ nur in sehr spezifischen und unrealistischen Situationen eingehalten werden kann. Stattdessen waren wir in der Lage, selbst bei unrealistisch großen Blockgrößen unter bestimmten Voraussetzungen mindestens 85% der Muster 1-eindeutig zu identifizieren, und selbst unter nicht-idealen, aber realistischen Umständen noch mindestens 75%.

Die Privatsphäre könnte durch Änderungen an den Generatoren noch verbessert werden, aber auch die technischen Möglichkeiten des Angreifers sind noch nicht ausgeschöpft. Es lässt sich allerdings bereits jetzt sagen, dass ein Erreichen der angegebenen Ratewahrscheinlichkeiten mit dem von Zhao et al. vorgeschlagenen Verfahren höchst unrealistisch ist, und selbst mit realistisch umsetzbaren Erweiterungen des Verfahrens voraussichtlich nicht erreicht werden kann.

8.2 Variation der Datenbasis

Nachdem wir den Effekt der Blockgröße auf die verschiedenen Modi kennen, interessiert uns nun der Effekt der Größe der Datenbasis, die der Client zum Generieren eines Range Queries verwenden kann, auf die Genauigkeit des Angreifers. Auch für diese Analyse unterscheiden wir wieder die verschiedenen Modi, die wir schon in der vorherigen Analyse verwendet haben.

Um den Effekt der Datenbasis besser vergleichen zu können, betrachten wir drei verschiedene Datenbasis-Größen (im Folgenden als S bezeichnet): $S = 2\,000$, $S = 20\,000$ und $S = 200\,000$. Vergleiche haben gezeigt, dass die Werte mit $S = 216\,925$ (die gesamte Datenbasis) sich nicht stark von den Werten für $S = 200\,000$ unterscheiden, daher wurde dieser Fall außen vor gelassen, um die Übersichtlichkeit zu wahren.

Die Verwendung von unterschiedlichen Datenbasen stellt uns vor ein Problem bei der Darstellung: Wenn wir andere Datenbasen verwenden, ist auch die Anzahl der Muster, die darin enthalten sind, unterschiedlich.[20] Entsprechend wäre eine Darstellung der Anzahl der k-eindeutigen Muster zweier verschiedener Datenbasen nicht vergleichbar. Wir haben die Zahlen daher in Anteilswerte zwischen null und eins umgerechnet, um eine Vergleichbarkeit herzustellen. Die kumulierten Verteilungsfunktionen geben daher immer an, welcher Anteil der Muster k-eindeutig oder besser (n-eindeutig mit $n < k$) ist.

Um die Auswirkungen von verschiedenen Blockgrößen bei verschiedenen Datenbasen zu prüfen, verwenden wir außerdem drei Blockgrößen in den Darstellungen: $N = 20$, $N = 100$ und $N = 200$. Dies erlaubt uns, allgemeine Trends besser von einzelnen Ausreißern unterscheiden zu können.

Die intuitive These wäre, dass die Genauigkeit des Angreifers bei einer größeren Datenbasis immer geringer sein sollte als bei einer kleineren Datenbasis und gleicher Blockgröße. Ob dies korrekt ist, werden wir in der folgenden Analyse feststellen.

Da der optimierte Algorithmus aus den weiter oben genannten Gründen für den Angriff auf den unterscheidbaren ersten Block bei zufälliger Auswahl der Dummies für kleine Datenbasen nicht zuverlässig funktioniert, wird er bei dieser Analyse nicht beachtet.

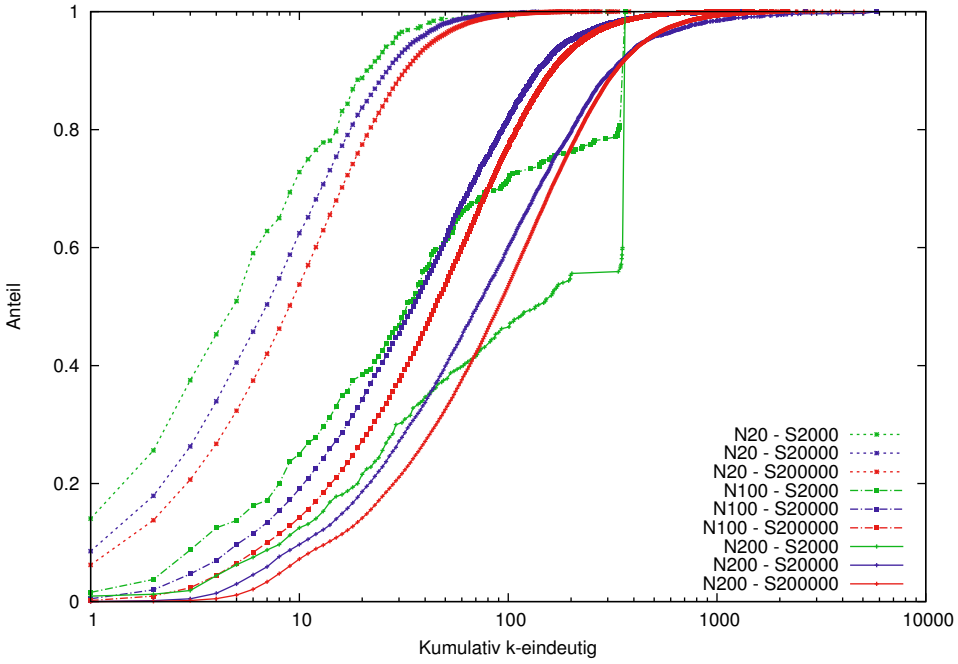
8.2.1 Vollständig ununterscheidbare Blöcke, zufällige Auswahl der Dummies

Wir beginnen wieder mit den vollständig ununterscheidbaren Blöcken. In Abb. 18 wurde das Verhalten bei verschiedenen Block- und Datenbankgrößen abgebildet.[21] Dabei gibt die Farbe die Größe der Datenbank an, die Linienart die Blockgröße. Des Weiteren befindet sich in der Abbildung noch eine Tabelle, die für die verschiedenen Parameter einige Kenngrößen (Anteil der 1-eindeutigen Muster, Median der k-Eindeutigkeit, maximale k-Eindeutigkeit) angibt.

Auf den ersten Blick sorgt eine kleinere Datenbasis für eine höhere Genauigkeit für den Angreifer. Dabei sind allerdings einige Anomalien zu beachten:

[20] Dazu sei hier erneut auf den Algorithmus zur Variation in Abschnitt 5.2.3 verwiesen.

[21] Hier, wie auch in den anderen Abbildungen dieser Analyse, ist die logarithmierte X-Achse zu beachten.



N	S	1-eindeutig	median(k)	max(k)
20	2 000	0.1406	5	299
20	20 000	0.0855	7	264
20	200 000	0.0622	9	379
100	2 000	0.0156	33	363
100	20 000	0.0046	35	5821
100	200 000	0.0018	45	2181
200	2 000	0.0094	135	363
200	20 000	0	71	5821
200	200 000	< 0.0000	90	5045

Abbildung 18: Verschiedene Datenbankgrößen S mit N als 20, 100, 200 in Modus 1

Bei der kleinsten Datenbasis von $S = 2\,000$ sinkt die Erkennungsleistung bei Blockgrößen von $N = 100$ und $N = 200$ bei k-Eindeutigkeiten von über 50 unter die Leistung der größeren Datenbasen, der Angreifer erhält also weniger Informationen als bei größeren Datenbasen. Dafür springt die kumulative k-Eindeutigkeit zwischen bei $k = 363$ plötzlich auf 1, unabhängig von der Blockgröße. Dies ist dadurch bedingt, dass bei dieser Kombination aus vergleichsweise hoher Blockgröße und kleiner Datenbasis die Menge der verfügbaren Anfragen ausgereizt wird. Es ist anzunehmen, dass das k, bei dem die maximale k-Eindeutigkeit erreicht wird, die Anzahl der Muster[22] in dieser Datenbasis darstellt. In diesem Fall hat dieses k für $S = 2\,000$ einen Wert von 363, für $S = 20\,000$ einen Wert von 5 821 (diese Werte wurden aus der Tabelle in Abb. 18 in der Spalte max(k) abgelesen). Die vorher niedrigere k-Eindeutigkeit könnte eine Anomalie der

[22] Dabei ist egal, ob diese Muster vom Algorithmus, der die Datenbasis variiert, gezogen wurden, oder ob sie durch zufällige Überschneidungen der anderen Muster entstehen.

Datenbasis sein[23], dies ließe sich nur durch weitere Tests mit anderen Datenbasen feststellen, die im Rahmen der Bachelorarbeit zeitlich nicht mehr möglich waren.

Des Weiteren ist bei Blockgrößen von 100 und 200 die Erkennungsleistung von $S = 200\,000$ auf den ersten Blick zu weiten Teilen schlechter als die von $S = 20\,000$, dies dreht sich allerdings bei einem k zwischen 300 und 400 um. Da das maximale k allerdings 5 821 beträgt (siehe max(k) in der Tabelle), bedeutet dies, dass die Erkennungsleistung des Angreifers für die *größere* Datenbasis tatsächlich auf dem größten Teil der Daten *höher* ist. Dieses Verhalten lässt sich nur durch eine unterschiedliche Verteilung der Musterlängen bzw. der Anzahl der Muster, die aus dieser Datenbasis zufällig entstehen können, erklären. Der Unterschied liegt allerdings im einstelligen Prozentbereich und würde sich somit nur in sehr wenigen Fällen auswirken.

Zusammengefasst lässt sich sagen, dass unsere These in diesem Modus experimentell nicht allgemein bestätigt werden konnte. Auch wenn sie für einen Teil der Daten zu gelten scheint, gibt es in diesem Modus zu viele Anomalien wie das unerwartete Verhalten bei $S = 2\,000$, um die These wirklich als bestätigt ansehen zu können. Ob dies allerdings an der Datenbasis lag oder einen allgemeinen Trend für beliebige Datenbasen darstellt, ließe sich nur durch zeitintensive weitere Versuchsreihen mit anderen Datenbasen bestimmen, die im Rahmen der Bachelorarbeit zeitlich nicht möglich waren.

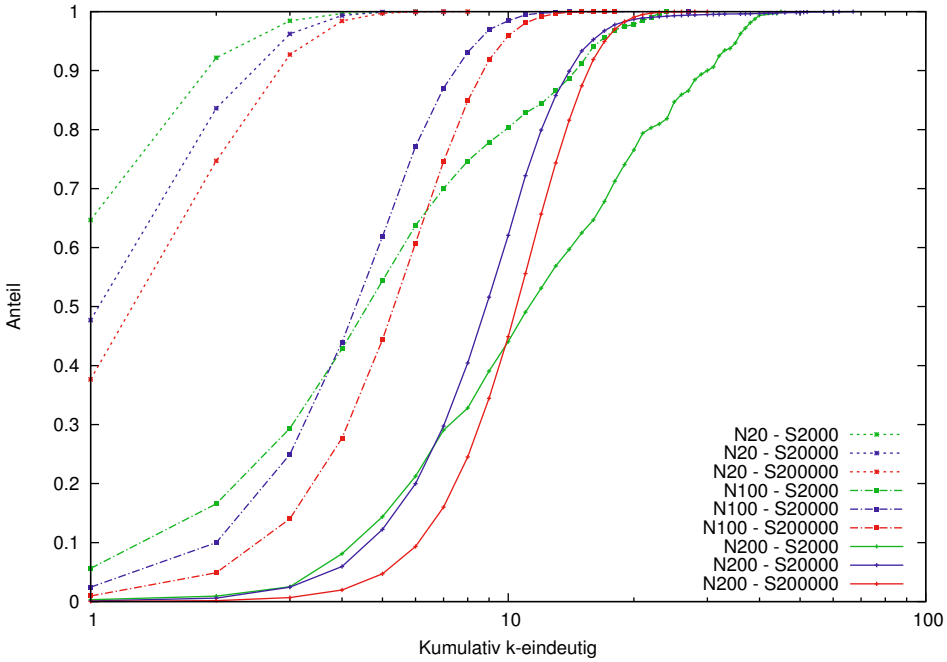
8.2.2 Unterscheidbarer erster Block, zufällige Auswahl der Dummies

Das Verhalten bei diesem Modus, abgebildet in Abb. 19, ist auf den ersten Blick in weiten Teilen ähnlich zu Modus 1. Der wichtigste Unterschied ist allerdings, dass die maximale k-Eindeutigkeit bei 67 (siehe Tabelle) anstatt der 5 821 des vorherigen Modus liegt. Auch in diesem Modus können wir wieder beobachten, wie bei einer Blockgröße von $N = 200$ die kleinere Datenbasis von $S = 20\,000$ eine höhere maximale k-Eindeutigkeit als die größere ($S = 200\,000$) hat, allerdings ist der Effekt bei weitem nicht so stark wie zuvor.

Auch das unerwartete Verhalten bei $S = 2\,000$ tritt wieder auf, der Verlauf der Daten ist allerdings etwas anders als zuvor. Es gibt keinen Punkt mehr, an dem die Funktion plötzlich massiv ansteigt, stattdessen steigt die Funktion kontinuierlich bis zu ihrem Maximum (welches dieses mal für unterschiedliche Blockgrößen nicht identisch ist), ohne dabei plötzliche Sprünge in ihrem Wert zu haben. Dies lässt sich dadurch begründen, dass sich der Musteranfang im ersten Block der Anfragen befinden muss. Daher kann die Funktion ihr oberes Limit von 363 Mustern nicht mehr erreichen, was der Grund für den Sprung in Modus 1 war.

Es lässt sich dasselbe Fazit wie zuvor ziehen: Eine größere Datenbasis scheint (in unseren Messungen) meistens, aber nicht zwangsweise immer, zu schlechteren Ergebnissen für den Angreifer zu führen. Doch auch hier gilt: da diesen Daten nur eine Messung zugrunde liegt, wäre für eine stichhaltige Analyse eine Messreihe mit verschiedenen, zufällig gewählten Datenbasen nötig, die im Rahmen der Bachelorarbeit zeitlich nicht möglich war.

[23] Die Datenbasis stimmt in allen Durchläufen mit der gleichen Datenbasis-Größe überein, daher können sich Anomalien auch über verschiedene Blockgrößen hinweg auswirken.



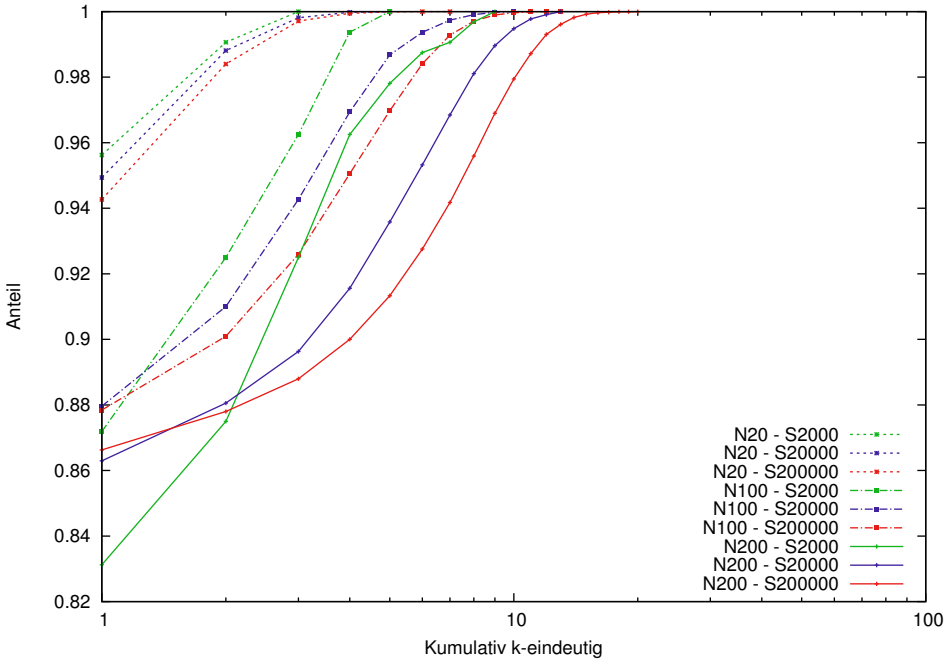
N	S	1-eindeutig	median(k)	max(k)
20	2000	0.6469	1	6
20	20000	0.4771	2	7
20	200000	0.3767	2	8
100	2000	0.0563	5	24
100	20000	0.0245	5	27
100	200000	0.0095	6	18
200	2000	0.0031	12	45
200	20000	0.0011	9	67
200	200000	0.0002	11	30

Abbildung 19: Verschiedene Datenbankgrößen S mit N als 20, 100, 200 in Modus 2.

8.2.3 Vollständig unterscheidbare Blöcke, zufällige Auswahl der Dummies

Auch in diesem Modus sieht man die Auswirkungen von verschiedenen Datenbasen, doch auch hier sind die Auswirkungen kleiner und unregelmäßiger als erwartet. Abb. 20 zeigt, dass die Anomalie bei der kleinsten Datenbasis auch in diesem Modus einen Effekt zeigt, die 1-Eindeutigkeit der Daten mit $S = 2000$ ist bei einer Blockgröße von 100 und 200 jeweils geringer als die der größeren Datensätze. Der sonstige Verlauf der Funktionen ist sehr ähnlich zu dem in anderen Modi beobachteten Verhalten.

Eine Anomalie im Vergleich zum bisher beobachteten Verhalten ist der Verlauf der Funktion mit einer großen Blockgröße von $N = 200$ und einer kleinen Datenbasis ($S = 2000$), in der Abbildung als durchgehende, grüne Linie dargestellt. Die Leistung mit diesen Parametern ist in Teilen schlechter als die einer größeren Datenbasis mit kleinerer Blockgröße ($N = 100$, $S =$

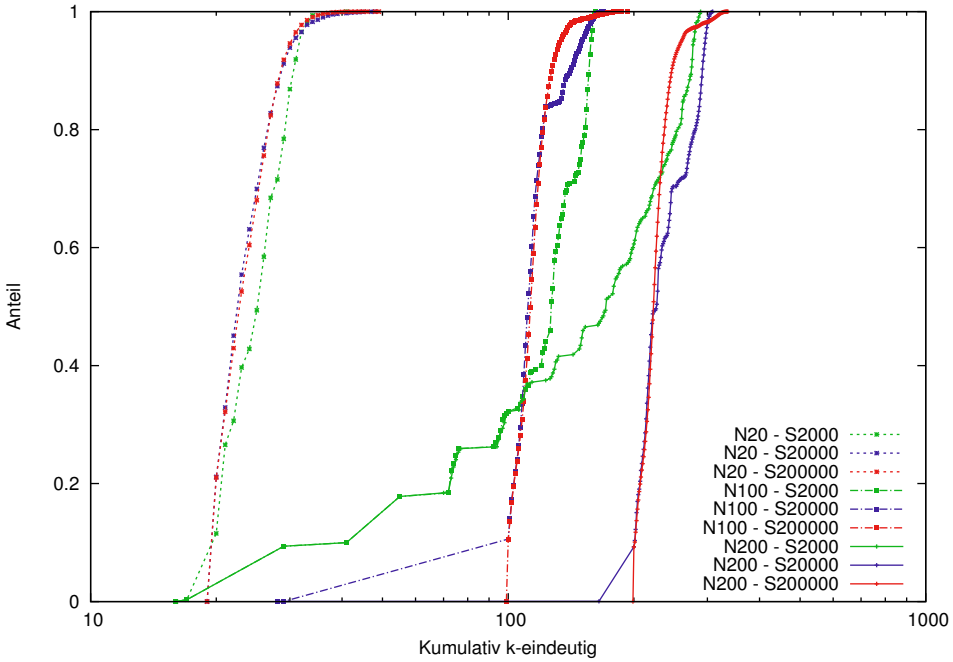


N	S	1-eindeutig	median(k)	max(k)
20	2000	0.9563	1	3
20	20000	0.9494	1	5
20	200000	0.9427	1	7
100	2000	0.8719	1	5
100	20000	0.8796	1	10
100	200000	0.8784	1	13
200	2000	0.8313	1	9
200	20000	0.8629	1	13
200	200000	0.8663	1	20

Abbildung 20: Verschiedene Datenbankgrößen S mit N als 20, 100, 200 in Modus 3.

200000, rot gestrichelte Linie). Eine größere Datenbasis mit kleinerer Blockgröße bietet also in diesem Intervall eine bessere Privatsphäre als eine kleine Datenbasis mit großer Blockgröße. Ob dies ein Zufall oder ein allgemeiner Trend ist, ließe sich nur mit einer größeren Basis an experimentellen Daten bestimmen.

Außerdem ist zu beachten, dass die Skala der Y-Achse erst bei 0.82 beginnt, die Unterschiede also, in absoluten Werten ausgedrückt, verhältnismäßig klein ausfallen. Mindestens 83% der Muster sind, unabhängig von Blockgröße und Datenbasis, sofort 1-eindeutig identifizierbar, wie wir schon in Abschnitt 8.1.4 festgestellt hatten (und was auch durch die Tabelle in Abb. 20 bestätigt wird). Alle Unterschiede durch Blockgröße und Datenbasis verändern also verhältnismäßig wenig am Endergebnis, dass der Angreifer in der Lage ist, einen großen Teil der Muster direkt 1-eindeutig zu identifizieren.



N	S	1-eindeutig	median(k)	max(k)
20	2000	0	26	41
20	20000	0	23	48
20	200000	0	23	49
100	2000	0	127	162
100	20000	0	112	169
100	200000	0	114	193
200	2000	0	172	289
200	20000	0	227	309
200	200000	0	223	336

Abbildung 21: Verschiedene Datenbankgrößen S mit N als 20, 100, 200 in Modus 4

Zusammengefasst lässt sich sagen, dass sich die Erkenntnisse der letzten Modi auch hier fortsetzen: Die These wird von den Daten nicht durchgehend unterstützt, und auch wenn große Teile der Daten die These in diesem Modus unterstützen (sogar so weit, dass der Effekt der Datenbasis den Effekt der Blockgröße in einem gewissen Intervall übertraf), so unterstützt doch der wichtigste Teil der Daten, die 1-Eindeutigkeit, die These in diesem Experiment nicht.

8.2.4 Vollständig ununterscheidbare Blöcke, musterbasierte Auswahl der Dummies

Eine Betrachtung der Ergebnisse dieses Modus', dargestellt in Abb. 21, zeigt uns einige interessante Eigenschaften der verschiedenen Datensätze. Die grünen Linien, die den Verlauf des kleinsten Datensatzes bei verschiedenen Blockgrößen darstellen, beginnen sehr früh, zu steigen, allerdings mit einer vergleichsweise kleinen Dichte an Messwerten.

Diese frühe Steigung lässt sich mit der Auswahlstrategie des musterbasierten Generators erklären: Wie bereits unter anderem in Abschnitt 8.1.7 beschrieben wurde, sucht der Generator, wenn nicht ausreichend viele Muster der korrekten Länge zur Verfügung stehen, nach zwei Mustern, deren Länge sich auf die korrekte Länge addiert. Wenn allerdings von diesen Mustern ebenfalls nicht ausreichend viele zur Verfügung stehen, bricht der Generator ab und verwendet die bisher gefundenen Muster, ohne weitere Anfragen hinzuzufügen.[24]

Durch dieses Verhalten können für manche Musterlängen bei kleinen Datenbasen nicht genug Dummy-Muster gefunden werden, die k-Eindeutigkeit sinkt entsprechend unter das eigentlich erwartete Minimum von N, welches wir in Abschnitt 8.1.5 bestimmt hatten, auf die Anzahl der verfügbaren passenden Muster (plus eventuelle zufällig durch die Vereinigung der den gewählten Mustern aufgetretene Muster). Diese Menge an Anfragen ist identisch für alle Muster der gleichen Länge (da sie ebenfalls alle zur Verfügung stehenden Muster verwenden werden), entsprechend zeigen alle Muster dieser Länge in der Simulation die gleiche k-Eindeutigkeit. Dies erklärt ebenfalls, warum sich die Linien der verschiedenen Blockgrößen überlappen, bis das jeweilige N erreicht wurde: Da der Datensatz identisch ist, haben alle Blockgrößen das Problem, dass nicht genug Muster zur Verfügung stehen, bis an einem bestimmten Punkt ausreichend viele Muster gefunden werden, um die Blöcke auf die Größe N aufzufüllen. Daher erhalten verschiedene Blockgrößen bis zu diesem Punkt die gleichen Ergebnisse für die betroffenen Muster. Ein ähnlicher Effekt, wenn auch weniger ausgeprägt, zeigt sich bei einer Datenbasis-Größe von $S = 20000$.

Interessanterweise zeigen die kleinen Datenbasen zwar ein schnelleres frühes Wachstum, zeigen allerdings bei $N = 20$ und $N = 100$ auf einem Abschnitt nachdem das jeweilige N überschritten wurde, eine generell bessere Leistung (also geringere k-Eindeutigkeit) als die größeren Datenbasen. Erst auf den letzten 10% sinkt ihre Leistung wieder unter die ihrer Gegenstücke mit größerer Datenbasis. Ein weiterer interessanter Effekt: für die beiden größeren Datenbasen liegt der median(k) durchgehend in einem Bereich von $1.1 * N$ bis $1.2 * N$ (vgl. Tabelle in Abb. 21).

Beide Effekte lassen sich dadurch erklären, dass dank der Annahme der ununterscheidbaren Blöcke kein erster Block existiert, in dem sich ein Musteranfang befinden müsste, damit sein Muster erkannt wird. Wenn also ein fehlendes Muster der korrekten Länge durch zwei andere Muster ersetzt wird, werden beide Muster erkannt (während in den anderen Modi nur das erste erkannt worden wäre). Da die kleineren Datenbasen durch die geringere Anzahl an Mustern öfter ein fehlendes Muster durch zwei andere ersetzen müssen, steigt ihre k-Eindeutigkeit entsprechend. Außerdem erleichtert die Abwesenheit unterscheidbarer Blöcke auch das zufällige Auftreten von Mustern, da die einzelnen Bestandteile nicht auf mehrere Blöcke aufgeteilt sein müssen.

Als Fazit lässt sich sagen, dass in diesem Modus die Größe der Datenbasis eine große Rolle spielt, bis ein gewisser Schwellenwert überschritten ist. Bei diesen Messungen liegt der Schwellenwert zwischen 2000 und 20000, da bei einer Datenbasis von 20000 Anfragen das frühe Ansteigen

[24] Es wäre natürlich möglich, in diesem Fall nach drei Mustern (oder, allgemein, bis zu $|M|$ Mustern), die sich auf die korrekte Länge addieren, zu suchen. Diese Erweiterung wurde nicht verwendet, um die Komplexität des Programmcodes gering zu halten, es hätte allerdings den in dieser Simulation beobachteten Effekt abgeschwächt.

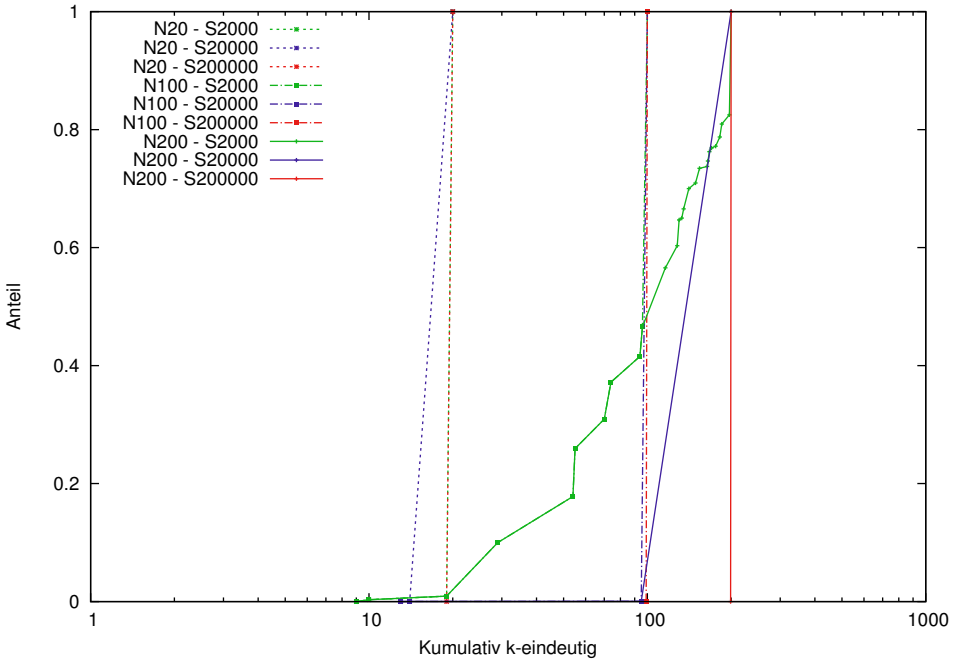
der Anteile mit geringer k-Eindeutigkeit nicht relevant stark auftritt.[25] Sobald dieser Schwellenwert überschritten ist, ist die Blockgröße das Kriterium, welches die durchschnittliche und maximale k-Eindeutigkeit am meisten beeinflusst, und eine höhere Blockgröße kann tatsächlich für eine geringere k-Eindeutigkeit sorgen. Dies liegt allerdings auch am verwendeten Generationsalgorithmus. Ein auf dieses Szenario angepasster Generationsalgorithmus könnte durch andere Auswahl der Dummy-Muster eine sehr viel höhere durchschnittliche k-Eindeutigkeit erreichen, genau wie ein angepasster Angreifer-Algorithmus ebenfalls durch weitere Heuristiken (wie z.B. die Beachtung der Musterlänge) die k-Eindeutigkeiten verringern könnte. Dies sind allerdings alles Untersuchungen, die den Rahmen einer Bachelorarbeit gesprengt hätten und daher in den Bereich der möglichen Gebiete für weitergehende Forschung in Kapitel 10 verschoben werden muss.

8.2.5 Unterscheidbarer erster Block, musterbasierte Auswahl der Dummies

Auch in diesem Modus tritt das Verhalten, welches wir bereits im vorherigen Modus beobachten konnten, auf. In Abb. 22 ist bei der kleinsten Datenbasis wieder der frühe Anstieg des Anteils k-eindeutiger Muster zu erkennen, lange bevor die Blockgröße erreicht wurde. Auch wenn es in der Abbildung schwer zu erkennen ist, überschneiden sich die Funktionswerte verschiedener Blockgrößen mit der gleichen Datenbasis wieder, bis das jeweilige N erreicht wurde. Der median(k) ist in diesem Fall allerdings niedriger (116 anstatt der 172 des vorherigen Modus für $S = 2\,000$). Dies liegt daran, dass nun ein erster Block existiert, in dem alle Musteranfänge liegen müssen. Entsprechend wird nun von zwei gezogenen Mustern, die sich auf die korrekte Länge addieren, nur noch eins gezählt, wie es schon in der Analyse der verschiedenen Blockgrößen der Fall war.

Im Gegensatz zum vorherigen Modus steigt die k-Eindeutigkeit nicht mehr über die jeweiligen Blockgrößen hinaus an, max(k) beträgt immer genau N (vgl. Tabelle). Diese Eigenschaft wurde bereits in der Analyse der variierten Blockgröße in Abschnitt 8.1.6 genauer behandelt. Im Gegensatz zu den Ergebnissen aus diesem Modus zeigt sich allerdings, dass es den Funktionen durchaus möglich ist, eine k-Eindeutigkeit von weniger als N anzunehmen. Der Grund dafür ist wieder der Generator, der, wenn er keine zwei Muster, die sich auf die korrekte Länge addieren, findet, keine weiteren Muster mehr hinzufügt. Der Effekt ist allerdings nur bei sehr kleinen Datenbasen wirklich sichtbar.

Auch in diesem Modus gilt also: Eine größere Datenbasis hat nur bis zu einem gewissen Schwellenwert zwischen 2 000 und 20 000 einen spürbaren Effekt[26], darüber hat nur die Wahl der Blockgröße einen Einfluss auf die k-Eindeutigkeit.



N	S	1-eindeutig	median(k)	max(k)
20	2000	0	20	20
20	20000	0	20	20
20	200000	0	20	20
100	2000	0	100	100
100	20000	0	100	100
100	200000	0	100	100
200	2000	0	116	200
200	20000	0	200	200
200	200000	0	200	200

Abbildung 22: Verschiedene Datenbankgrößen S mit N als 20, 100, 200 in Modus 5

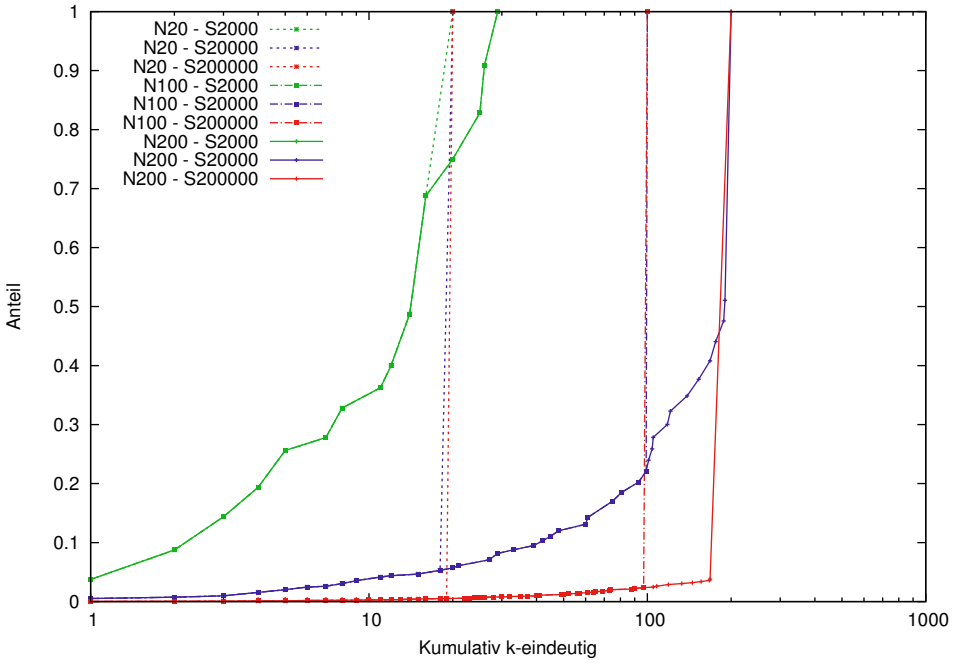
8.2.6 Vollständig unterscheidbare Blöcke, musterbasierte Auswahl der Dummies

Auch dieser Modus zeigt ein Verhalten ähnlich zu dem des vorherigen Modus', wie in Abb. 23 zu sehen ist. Da nun aber die Länge des korrekten Musters wieder durch die Anzahl der Blöcke bestimmt ist, können zusätzlich gezogene Muster bei einem Mangel an Mustern der korrekten Länge ausgeschlossen werden.

Dies hat bei verschiedenen Datenbasen verschiedene Auswirkungen. Da größere Datenbasen

[25] Es ist zwar ein kleiner Anstieg zu sehen, eine Prüfung der Ergebnisdaten hat allerdings ergeben, dass sich dieser auf lediglich zwei Muster beschränkt, bevor die N-Eindeutigkeit erreicht wird.

[26] Zumindest mit den aktuellen Angreifern. Generell sollte in jedem Fall eine möglichst umfangreiche Datenbasis angestrebt werden, um hier nicht implementierte Angriffe zu verhindern, die von einer kleineren Datenbasis profitieren. Einige Ansätze für solche Angriffe werden in Kapitel 10.2 angesprochen.



N	S	1-eindeutig	median(k)	max(k)
20	2 000	0.0375	16	20
20	20 000	0.0052	20	20
20	200 000	0.0003	20	20
100	2 000	0.0375	16	29
100	20 000	0.0052	100	100
100	200 000	0.0003	100	100
200	2 000	0.0375	16	29
200	20 000	0.0052	190	200
200	200 000	0.0003	200	200

Abbildung 23: Verschiedene Datenbankgrößen S mit N als 20, 100, 200 in Modus 6

im Schnitt mehr Muster einer gegebenen Länge haben, steigt auch die durchschnittliche k-Eindeutigkeit mit der Datenbasis. Dieser Effekt kann in Abb. 23 auch direkt an der Steigung der verschiedenen Funktionen abgelesen werden. Dies geht so weit, dass bei einer Datenbasis von 2 000 Anfragen kein Muster schlechter als 29-eindeutig identifizierbar ist (da im Datensatz die häufigste Musterlänge offensichtlich 29 mal vorkam).

In diesem Modus gilt also tatsächlich die These, dass eine größere Datenbasis bei gleicher Blockgröße zu einer durchschnittlich geringeren Präzision des Angreifers führt. Doch auch hier ist die k-Eindeutigkeit nach oben durch N begrenzt, wodurch eine große Datenbasis mit einer kleinen Blockgröße eine höhere Erkennungsleistung für den Angreifer zeigen wird als eine kleinere Datenbasis mit größerer Blockgröße, solange die Größe der Datenbasis noch einen gewissen Schwellenwert überschreitet (eine Größe von 2 000 ist offensichtlich nicht ausreichend).

8.2.7 Fazit zur Variation der Datenbasis

Die ursprüngliche These, dass eine größere Datenbasis immer eine bessere Privatsphäre (also höhere k-Eindeutigkeiten) als eine kleinere Datenbasis mit gleicher Blockgröße zeigen sollte, konnte in der Simulation nicht bestätigt werden, es gab zu viele Gegenbeispiele, um diese These als bestätigt ansehen zu können.

Wie viele dieser Gegenbeispiele aus den Abschnitten 8.2.1 bis 8.2.3 mit der speziellen Datenbasis zusammenhängen und wie viele tatsächlich durch das Verfahren bedingt sind, lässt sich ohne weitere Simulationen, die im Rahmen der Bachelorarbeit nicht mehr möglich waren, nicht bestimmen. Mindestens das Verhalten, das in den Abschnitten 8.2.4 bis 8.2.6 beschrieben wurde, ist allerdings durch das Verfahren bedingt und sollte mit beliebigen, zufällig gewählten Datenbasen replizierbar sein.

9 Diskussion der Ergebnisse

Das Experiment basiert auf einigen Vorannahmen, die in der Realität nicht zwangsweise zutreffen. Im Folgenden wollen wir diese Annahmen nun diskutieren und überprüfen, welche Auswirkungen es hätte, sollten sie nicht gelten. Teilweise wiederholen wir dabei Annahmen aus dem Angreifermodell aus Abschnitt 4.1.

9.1 Vollständigkeit der Datenbasis

Die erste Annahme, die wir getroffen haben, war, dass die Datenbasis des Angreifers *alle* Anfragen und Muster enthält, die der Client aufrufen kann. Diese Annahme ist problematisch, da der Client prinzipiell dazu in der Lage ist, das gesamte Internet aufzurufen, während es dem Angreifer schwer fallen würde, eine Datenbank der Musterlängen aller existierenden Internetseiten zu erstellen und aktuell zu halten. Entsprechend wäre es dem Client in der Realität möglich, eine Webseite aufzurufen, die nicht in der Datenbasis des Angreifers vorhanden ist.

Eine mögliche Lösung für dieses Problem wäre, dass der Angreifer ihm unbekannte Domains dynamisch abruft und ihre Muster in die Datenbasis abspeichert. Dadurch wächst die Datenbasis des Angreifers über die Zeit weiter an, und er ist, nachdem er alle unbekannten Domain-Namen abgerufen hat, dazu in der Lage, nachträglich mit dem vergrößerten Datensatz einen Angriff auf die Anfragen des Clients zu führen. Die Identifikation der besuchten Webseite wäre somit verzögert, aber nach wie vor möglich. Sollte diese Strategie nicht gewollt oder möglich sein, so kann der Angreifer nicht mehr sicher sagen, ob sein gefundenes Muster tatsächlich besucht wurde, da eventuell ein ihm unbekanntes Muster in den Anfragen enthalten war, welches tatsächlich besucht wurde.

9.2 Veränderlichkeit der Muster

Die zweite Annahme ist, dass sich die Muster nicht verändern. Ein Muster, welches in der Datenbank vorhanden ist, kann in der realen Welt allerdings binnen kürzester Zeit veraltet sein, vor allem bei großen Webseiten mit vielen Nutzerinhalten, die sich schnell ändern (*www.reddit.com* wäre ein Beispiel für diese Sorte von Internetseite). Da der Client immer das aktuellste Muster abruft (er lädt schließlich die aktuelle Version der Internetseite) stellt dies den Angreifer vor das Problem, seine Datenbasis aktuell zu halten.

Auch hier wäre wieder eine mögliche Lösung, dass der Angreifer für alle Musteranfänge, die er in den ankommenden Anfragen erkennen kann, die Datenbank aktualisiert. Er würde also dynamisch das aktuelle Muster von *reddit.com* bestimmen, wenn diese URL eine der ankommenden Anfragen wäre. Sollte ein solches Verhalten nicht erwünscht sein, so könnte er eventuell noch mit Toleranzen arbeiten („mindestens *reddit.com*, *redditstatic.com*, ..., plus bis zu 10 andere Seiten“). Dies würde allerdings die Wahrscheinlichkeit für inkorrekt identifizierte Muster erhöhen und damit die Genauigkeit des Angreifers verringern.

Allerdings hat auch der Client das Problem der veränderlichen Musterlängen, wenn er die musterbasierte Auswahl der Dummy-Anfragen nutzen möchte. Da sich Muster und damit auch Musterlängen ändern können, kann der Client nicht sicher sein, dass er die korrekte Musterlänge in seiner Datenbasis hat. Entsprechend fällt es ihm schwer, Muster der korrekten Länge als Dummies zu verwenden. Für den Client ist es außerdem unmöglich, in diesem Fall dynamisch erst einmal die Musterlänge festzustellen, da er dafür die Seite laden müsste, wofür er wieder Dummy-Muster der korrekten Länge bräuchte, und so weiter (Henne-Ei-Problem). Eine mögliche Lösung für diese Problematik wird in Abschnitt 10.1 vorgestellt.

9.3 Caching / TTL

Eine weitere Problematik ist, dass der Client Antworten auf DNS-Anfragen für eine bestimmte Zeit, die so genannte TTL, zwischenspeichert (caching). Wird innerhalb der TTL eine weitere Anfrage nach einer zwischengespeicherten Domain gestellt, wird diese sofort aus dem lokalen Cache beantwortet. Der Angreifer hat keine Möglichkeit, festzustellen, ob die Anfrage innerhalb der TTL erneut gestellt wurde. Diese Problematik wurde in der Simulation aufgrund der mit einer Implementation verbundenen Komplexität nicht beachtet. Eine mögliche Lösung für dieses Problem wird in Abschnitt 10.2 vorgestellt.

9.4 DNS-Prefetching

Ebenfalls nicht beachtet wurde der Effekt des DNS-Prefetching. Wie in Abschnitt 5.1.3 bereits erwähnt haben, enthält unsere Datenbasis nicht die Auswirkungen von DNS-Prefetching.[27]

[27] Für mehr Details zum Thema DNS-Prefetching sei hier erneut auf die Chromium Developer Documentation [11] und den Artikel von Krishnan et al. [9] verwiesen.

DNS-Prefetching hat sowohl positive als auch negative Auswirkungen für den Angreifer. Die amerikanischen Forscher Krishnan und Monrose haben in ihrem Artikel „DNS prefetching and its privacy implications: When good things go bad“ [9] festgestellt, dass es einem neugierigen DNS-Server möglich wäre, aus den Anfragen, die er durch DNS-Prefetching erhält, den gesuchten Term (oder zumindest das Thema einer Suche) in einer Suchmaschine wie Google abzuleiten.

Gleichzeitig kann prefetching das Problem der veränderlichen Muster verschiedener Webseiten verstärken. Ein Beispiel: Das Aufrufmuster von *reddit.com*, einer populären Webseite, welche unter anderem Links auf interessante Artikel auf anderen Webseiten aggregiert, würde sich bei aktivem DNS-Prefetching stündlich oder öfter verändern (immer, wenn ein Link auf eine neue Seite auf der Startseite erscheint oder ein alter Link verschwindet). Das Problem wird noch dadurch verstärkt, dass angemeldete Nutzer unterschiedliche Startseiten erhalten und entsprechend unterschiedliche Muster durch das Prefetching entstehen.

In diesem Fall wäre die einzige Lösung für den Angreifer, nur die Musterelemente, welche zur Webseite selber gehören, als Muster zu speichern, und die Webseite als mögliche Lösung anzusehen, sobald diese Elemente aufgetreten sind, selbst wenn die Musterlänge nicht mit der Anzahl der Blöcke (so diese denn unterscheidbar sind) übereinstimmt. Eine solche Strategie würde allerdings die Zahl der „false positives“, also der falsch erkannten Muster, erhöhen.

9.5 Datenbasis

Als methodisches Problem dieser Bachelorarbeit muss noch erwähnt werden, dass sämtliche Ergebnisse aus einem einzigen Durchlauf stammen. Für wissenschaftlich stichhaltige Argumentationen wären weitere Testläufe mit anderen Datenbasen nötig, um allgemeine Trends von Anomalien in der Datenbasis abgrenzen zu können. Dies war im Rahmen der Bachelorarbeit nicht mehr möglich.

10 Weitere Generations- und Angriffsstrategien

Im Verlauf der Bachelorarbeit sind einige weitere Ideen für Generationsstrategien und Angriffe aufgekommen, die nicht mehr implementiert und getestet werden konnten. Daher werden diese im Folgenden kurz vorgestellt, um Ansätze für eventuelle weitergehende Forschung in diesem Gebiet zu bieten.

10.1 Nicht implementierte Generationsstrategien

Eine erste Maßnahme, um zumindest den optimierten Algorithmus beim Modus des unterscheidbaren ersten Blocks zu verhindern, wäre eine **dynamische Blockgröße** N. Sobald die Blockgröße

nicht mehr konstant N beträgt[28], sind die mathematischen Vorbedingungen des optimierten Algorithmus nicht mehr erfüllt. Die Auswirkungen auf die anderen Modi dürften allerdings gering ausfallen.

Die Aufrufe von Webseiten sind in der realen Welt nicht gleichverteilt, populäre Webseiten werden sehr viel öfter als obskure Webseiten aufgerufen. Daher wäre ein Angreifer theoretisch dazu in der Lage, Vermutungen über die besuchte Webseite anzustellen, wenn sein Angriff mehr als ein Resultat ergibt. Dies könnte der Client verhindern, indem er das Ziehen von Dummy-Anfragen (oder -Mustern, je nach gewählter Generationsstrategie) **gewichtet nach Popularität** der Webseiten vornimmt. So wäre es wahrscheinlicher, *www.google.com* zu ziehen, als eine obskure Seite wie *www.blogdephp.com*. Dazu benötigt der Client eine Datenbank, die die relative Popularität der Dummies enthält.

Eine mögliche Erweiterung der musterbasierten Ergänzung wäre, die **Musterlänge** des Zielmusters auf das nächste Vielfache einer Zahl $x > 1$ **aufzurunden**. Ein Muster mit 9 Elementen würde also z.B. bei $x = 5$ auf 10 Elemente aufgerundet (das letzte Element wäre entweder eine zufällige Anfrage oder ein Muster der Länge 1). Nun könnten weitere Muster der Längen 6 - 10 genutzt werden, um die Blöcke aufzufüllen, was die Anzahl der zur Verfügung stehenden Muster erhöhen würde. Entscheidend ist dabei, dass immer der gesamte Raum an Musterlängen, der von dem Vielfachen von x abgedeckt wird (in diesem Fall die Längen 6 bis 10) als Datenbasis für weitere Muster verwendet wird, damit keine Rückschlüsse auf die wahre Länge des Musters möglich sind.[29]

Diese Strategie setzt ebenfalls voraus, dass die Länge eines Musters vor Beginn der Abfrage bekannt ist. Dass dies eine unrealistische Annahme ist, haben wir bereits in Abschnitt 9.2 festgestellt. Es wäre allerdings möglich, sich die Länge des Musters zu merken, sobald es einmal aufgerufen wurde, und beim nächsten Aufruf zu verwenden. Im schlimmsten Fall hat sich die Länge so geändert, dass ein anderes Vielfaches von x herauskommt, allerdings sollte eine solche Änderung bei den meisten Internetseiten eher die Ausnahme als die Regel darstellen. Beim ersten Aufruf kann ein Erfahrungswert (z.B. der Durchschnitt der bekannten Musterlängen) oder eine (hohe) Konstante gewählt werden.

10.2 Nicht implementierte Angriffstrategien

Wie wir aus Abschnitt 9.3 wissen, ist die Annahme, dass ein Client immer alle benötigten Domain-Namen abfragen muss, nicht korrekt, da alle modernen Systeme die Antworten des DNS-Servers für eine bestimmte Zeit (die so genannte **TTL**) zwischenspeichern (Caching). Wollen wir den Angriff also in der realen Welt durchführen, müssen wir diesen Fall beachten.

Es bieten sich zwei Lösungen an: Zum einen könnte der Angreifer Informationen über die Anfragen, die er dem Client beantwortet hat, **zwischenspeichern**, und sie bis zum Ablauf ihrer jeweiligen TTL in sämtliche Prüfungen von ankommenden Mustern von dem Client mit

[28] Oder, genauer gesagt, N oder $N-1$ im Falle eins Duplikates

[29] Würde stattdessen zum Beispiel ein Raum von $|M| \pm 2$ verwendet, könnte ein Angreifer daraus wieder Rückschlüsse auf $|M|$ ziehen.

einbeziehen. Weiß der Angreifer also, dass der Client noch eine gültige Adresse für *google-analytics.com* abgespeichert hat, so ist es möglich, dass das Muster der aufgerufenen Webseite *google-analytics.com* enthält, die Anfrage allerdings noch aus dem Cache beantwortet werden konnte. Wenn nun also ein Muster erkannt wurde, bei dem alle Elemente bis auf *google-analytics.com* in den ankommenden Anfragen enthalten sind, so ist dieses Muster ein Kandidat für das gesuchte Muster.

In unserem Angreifermodell ist der Angreifer passiv (siehe Abschnitt 4.1). Wenn wir von diesem Angreifermodell abrücken, könnte der Angreifer dem Client auch **falsche TTLs** vorgaukeln und somit die Zeit, die eine Anfrage zwischengespeichert wird, auf wenige Sekunden reduzieren. Dies könnte den Angreifer allerdings gegenüber einem misstrauischen Nutzer, der sich wundert, warum die TTLs ankommender DNS-Antworten nie länger als 5 Sekunden sind, verraten. Dies könnte (sollte) auch automatisiert in einem Range Query-Programm geschehen.

In Abschnitt 8.1.6 hatten wir bereits erwähnt, dass es im Modus des unterscheidbaren ersten Blocks bei den aktuellen Generatoren möglich wäre, einige der weiteren auftretenden Muster auszuschließen, da bei den aktuellen Generatoren sehr wahrscheinlich ist, dass das gesuchte Muster sich in der Menge der Muster mit der höchsten Länge befindet.[30] Eine Gegenmaßnahme dagegen wäre die im vorherigen Abschnitt vorgestellte Methode, die Musterlängen auf Vielfache einer Zahl **aufzurunden**.

Eine letzte Erweiterung, die die Erkennungsleistung in der realen Welt bei veränderlichen Mustern erhöhen könnte, wären **Wildcard-Domains** in den Patterns. Wenn zum Beispiel eine Seite Informationen aus einem Content Distribution Network (CDN) abruft, welches viele hundert Domains nach dem Muster *cdn1325.whatever.com* verwendet und bei jedem Aufruf auf eine andere dieser Domains verweist, könnte der Datenbankeintrag *cdn*.whatever.com* alle diese Fälle enthalten und somit für beliebige Domains des CDN gelten. Dies verringert das Problem der variablen Muster für große Seiten, auch wenn es nach wie vor nicht in der Lage ist, trotz großer Veränderungen im Muster einer Webseite (z.B. wenn das CDN gewechselt wird) weiterhin zu funktionieren.

11 Zusammenfassung der Erkenntnisse

Unsere Simulationen haben gezeigt, dass das Range Query-Verfahren unter realistischen Bedingungen die von Zhao et al. angegebene Ratewahrscheinlichkeit von $\frac{1}{N}$ für den Angreifer nicht einhalten kann. Ein neugieriger DNS-Server ist bei der Verwendung des vorgeschlagenen Range Query-Algorithmus auch bei unpraktikabel hohen Blockgrößen von über 100 noch in der Lage, bis zu 80% der abgerufenen Webseiten 1-eindeutig zu erkennen (vgl. Abb. 13, Abschnitt 8.1.3).

Weiterhin konnten wir zeigen, dass eine Erweiterung des Range Query-Verfahrens, welche auf der Auswahl ganzer Anfragemuster als Dummies zur Verschleierung der besuchten Internetseite

[30] Es ist allerdings nicht garantiert, da ein zufällig auftretendes Muster noch immer länger sein kann. Eine weitere Heuristik, die prüft, ob es einen Wert gibt, den mindestens ein Muster als Länge hat und auf den sich fast alle der anderen Muster in bestimmten Kombinationen summieren, könnte diesen Fall jedoch ausschließen.

basiert, in der Theorie die Privatsphäre des Nutzers schützen könnte. Da dieses Verfahren allerdings in seiner aktuellen Form nicht problemlos umsetzbar ist und nicht alle möglichen Angriffe auf das Verfahren getestet wurden, ist in diesem Bereich weitere Forschung nötig, bevor ein endgültiges Urteil über die Leistung und Praktikabilität des Verfahrens gefällt werden kann.

Literatur

[1] Alexa: Die top 1 000 000 Webseiten des Internets, gemessen durch Alexa. Abgerufen am 22. März, 2013. URL *http://www.alexa.com/topsites*.

[2] Yu Bo, Qi Luo: Two-Servers PIR Based DNS Query Scheme with Privacy-Preserving. In: The 2007 International Conference on Intelligent Pervasive Computing (IPC 2007). IEEE, Okt. 2007, 487–490.

[3] Sergio Castillo-Perez, Joaquin Garcia-Alfaro: Anonymous resolution of DNS queries. On the Move to Meaningful Internet .

[4] Sergio Castillo-Perez, Joaquin Garcia-Alfaro: Evaluation of Two Privacy-Preserving Protocols for the DNS, 2009.

[5] Hannes Federrath, Karl-Peter Fuchs, Dominik Herrmann, Christopher Piosecny: Privacy-Preserving DNS: Analysis of Broadcast, Range Queries and Mix-based Protection Methods. Proceedings of the 16th European conference on Research in computer security (2011) 665–683.

[6] Python Software Foundation: Die Homepage der Programmiersprache Python, abgerufen am 7. Oktober, 2013. URL *http://python.org/*.

[7] Dominik Herrmann, Christian Banse, Hannes Federrath: Behavior-based tracking: Exploiting characteristic patterns in DNS traffic. Computers & Security (2013) URL *http://dx.doi.org/10.1016/j.cose.2013.03.012*.

[8] Ariya Hidayat: Die Homepage des Headless Webkit-Projekts PhantomJS, abgerufen am 7. Oktober, 2013. URL *http://phantomjs.org/*.

[9] Srinivas Krishnan, Fabian Monrose: DNS prefetching and its privacy implications: when good things go bad. In: Proceedings of the 3rd USENIX conference on Large-scale exploits and emergent threats: botnets, spyware, worms, and more, LEET'10. USENIX Association, Berkeley, CA, USA, 2010, 10–10. URL *http://dl.acm.org/citation.cfm?id=1855686.1855696*.

[10] Yanbin Lu, Gene Tsudik: Towards Plugging Privacy Leaks in Domain Name System .

[11] Chromium Project: DNS Prefetching in den Design Documents des Chromium-Projekts, abgerufen am 5. Oktober, 2013. URL *http://www.chromium.org/developers/design-documents/dns-prefetching*.

[12] Kapil Singh, Alexander Moshchuk, Helen J. Wang, Wenke Lee: On the incoherencies in web browser access control policies. In: Security and Privacy (SP), 2010 IEEE Symposium on. 2010, 463–478.

[13] Fangming Zhao, Yoshiaki Hori, Kouichi Sakurai: Analysis of Privacy Disclosure in DNS Query. 2007 International Conference on Multimedia and Ubiquitous Engineering MUE07 (2007) 952–957.

Quelltexte und Ergebnisdaten

Die Quelltexte und Ergebnisdaten dieser Bachelorarbeit stehen unter der BSD 2-Clause Lizenz und können unter *https://github.com/Semantic-IA* abgerufen werden.

Sämtliche Dateien wurden unter Linux erstellt, entsprechend könnte die Anzeige der Zeilenumbrüche unter anderen Betriebssystemen abweichen.

Erklärung

Ich versichere, dass ich die Arbeit selbstständig verfasst und keine anderen als die angegebenen Hilfsmittel – insbesondere keine im Quellenverzeichnis nicht benannten Internetquellen – benutzt habe, die Arbeit vorher nicht in einem anderen Prüfungsverfahren eingereicht habe und die eingereichte schriftliche Fassung der auf dem elektronischen Speichermedium entspricht.

Ich bin mit der Einstellung der Arbeit in den Bestand der Bibliothek des Departments Informatik einverstanden.

Hamburg, den 8. Oktober 2013

Max Jakob Maaß