Detektion von anonym abgerufenen rechtswidrigen Inhalten mit einem hashwertbasierten Datenscanner

Hannes Federrath, Wolfgang Pöppl
Universität Regensburg, Lehrstuhl Management der Informationssicherheit

Abstract: Benutzer von Anonymisierungsdiensten können strafrechtlich relevante Inhalte abrufen, ohne dass nachvollziehbar ist, wer die Inhalte heruntergeladen hat. Als Abrufer kann letztendlich immer nur der Anonymisierer ermittelt werden. Nachforschungen der Strafermittlungsbehörden laufen ins Leere und verursachen nur Arbeitsaufwand auf beiden Seiten. Daher soll die Möglichkeit, Anonymisierungsdienste zum Abruf strafrechtlich relevanter Inhalte verwenden zu können, reduziert werden, ohne die Anonymität des Abrufers zu gefährden. Das Papier diskutiert Möglichkeiten zur Realisierung eines Datenscanners und stellt die konkrete Implementierung eines solchen vor (im Folgenden AnonHash genannt). Über den Anonymisierungsdienst sollen weiterhin alle Informationen und Inhalte aus dem Internet abrufbar bleiben, abgesehen von Inhalten, deren Abruf und Besitz zweifelsfrei eine Straftat nach dem deutschen Strafgesetzbuch darstellt.

1 Einführung

Anonymisierungsdienste wie AN.ON (http://www.anon-online.de) oder Tor (http://tor.eff.org) ermöglichen anonymes und unbeobachtbares Surfen im Internet. Die Möglichkeiten, einen Datenscanner in einen solchen Anonymisierungsdienst zu integrieren, sollen am Beispiel von AN.ON diskutiert werden.

Die Architektur des AN.ON-Dienstes ist in Abbildung 1 dargestellt. Um den AN.ON-Dienst zu nutzen, wird der Web-Anonymizer JAP (Java Anon Proxy) verwendet. JAP [Anon06] ist ein lokal auf dem Rechner installiertes Programm (Proxy), das alle Internet-Anfragen (z.B. den Aufruf einer Webseite) verschlüsselt über eine Kaskade von mehreren unabhängigen Mixen [Chau81] leitet. Nur der letzte Mix kann die Anfrage im Klartext lesen und ausführen. Die zugehörige Antwort (also der Inhalt der Webseite) wird wieder verschlüsselt an den Benutzer zurückgeschickt. Die besuchte Webseite sieht als Absender der Anfrage nur den letzten Mix. Der Internet-Zugangsprovider bzw. jeder Lauscher auf den Verbindungen sieht nur, dass der Benutzer von AN.ON mit dem ersten Mix kommuniziert. Da diese Kommunikation verschlüsselt ist, kann er daraus das Ziel nicht ermitteln. Solange wenigstens einer der Mixe vertrauenswürdig ist und keine Verbindungsinformationen mit den anderen Mixbetreibern austauscht, kann der Benutzer des AN.ON Dienstes anonym und unbeobachtbar im Internet surfen. AN.ON unterstützt derzeit nur das HTTP-Protokoll.

AnonHash kann nicht sinnvoll in den JAP integriert werden, da JAP unter der alleinigen Kontrolle des Nutzers steht und der Scanvorgang vom Nutzer leicht deaktiviert werden

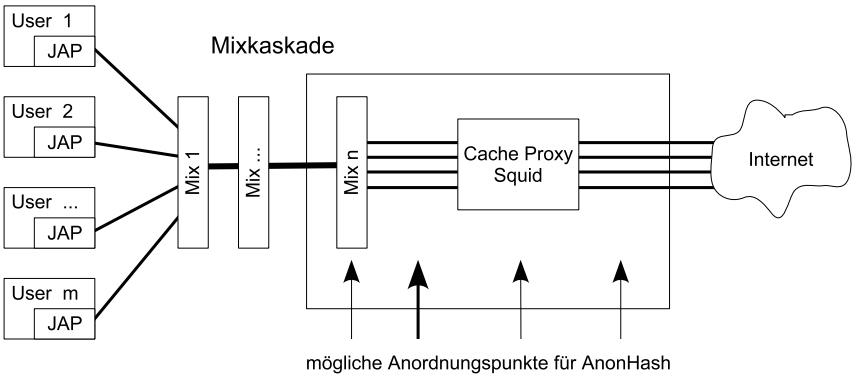

Abbildung 1: Architektur des AN.ON-Dienstes. Der letzte Mix leitet den gesamten Datenverkehr über einen Cache-Proxy ins Internet. Dazu kann ein beliebiger Cache-Proxy verwendet werden. Derzeit kommt hauptsächlich *Squid* (http://www.squid-cache.org) zum Einsatz. Üblicherweise laufen der letzte Mix und der Cache-Proxy auf demselben Rechner (angedeutet durch einen Kasten). Der **dickere Pfeil** zeigt, wo AnonHash zwischengeschaltet wird.

könnte. Da die Kommunikation zwischen JAP und letztem Mix verschlüsselt stattfindet, kann AnonHash nur sinnvoll auf der Kommunikationsstrecke ab einschließlich dem letzten Mix und dem Internet platziert werden. Mögliche Anordnungspunkte sind in Abbildung 1 durch Pfeile gekennzeichnet. Eine Anordnung zwischen Cache-Proxy und Internet hätte den Vorteil, dass auf dem Cache-Proxy nur Dateien gespeichert werden, die zuvor von AnonHash gescannt wurden. Eine Auslieferung der Daten aus dem Cache würde somit auch keinen erneuten Scanvorgang verursachen. In diesem Fall müssten jedoch die gesamten Modifikationen, die das HTTP-Protokoll von einem Proxy fordert, in AnonHash implementiert werden. Da diese Funktionalität bereits in dem vorhandenen Cache-Proxy enthalten ist, ist es nicht sinnvoll, diese Funktionalität erneut zu implementieren. Eine Platzierung vor dem Cache-Proxy (genauer: zwischen letztem Mix und Cache-Proxy) ermöglicht eine Realisierung, bei der prinzipiell keine Modifikationen an den Daten notwendig sind und der Datenverkehr unverändert durchgeleitet werden kann.

2 Datenscanner

AnonHash soll in der Lage sein, bekannte strafrechtlich relevante Inhalte im Downloadstrom des AN.ON-Dienstes erkennen zu können, um deren Abruf zu blockieren. Die den Strafverfolgungsbehörden (z.B. BKA) vorliegenden, eindeutig strafrechtlich relevanten Inhalte (z.B. Kinderpornographie im PERKEO-System, nicht jedoch auch legal herunterladbare Dateien wie z.B. urheberrechtlich geschützte Musik) werden im Weiteren als Rohdaten bezeichnet. Der Datenscanner nimmt selbst keine Einstufung in strafrechtlich relevante bzw. nicht strafrechtlich relevante Inhalte vor. Es wird lediglich versucht, identische Dateien, die auch in den Rohdaten enthalten sind, zu erkennen und zu blockieren. Dazu werden kryptographische Hashwerte der Daten verwendet.

Ist der Hashwert (z.B. MD5 oder SHA1) einer Datei identisch zu dem Hashwert einer anderen Datei, so sind mit sehr hoher Wahrscheinlichkeit die Inhalte der beiden Dateien ebenfalls identisch. Diese Methode besitzt entscheidende Vorteile gegenüber dem direkten Vergleich zweier Dateien. Die Rohdaten, die sehr umfangreich sein können, müssen (und dürfen auch aus rechtlichen Gründen) beim Überprüfen einer Datei nicht im Scanner vorliegen. Beim Scannvorgang mit der Hashwertmethode werden lediglich die Hashwerte der Rohdaten benötigt. Diese benötigen in der Regel nur einen Bruchteil des Speicherbedarfs der Ursprungsdaten.

Theoretisch besteht bei diesem Verfahren jedoch die Möglichkeit, dass es Dateien mit unterschiedlichem Inhalt aber identischem Hashwert gibt. Ein zufälliges Auftreten von Kollisionen bei Multimedia-Dateien ist entsprechend der Länge des Hashwertes (bei MD5 128 Bit, bei SHA1 160 Bit) jedoch sehr unwahrscheinlich.

Die Hashwertmethode kann performant implementiert werden. Sie ist prinzipiell unabhängig von Dateityp und Inhalt und kann auf jede Datei angewandt werden. Die Hashwerte von strafrechtlich relevanten Inhalten sind selbst strafrechtlich unbedenklich.

Zu beachten ist, dass es in der Natur von Hashalgorithmen liegt, dass die Veränderung von nur einem Bit in der Datei zu einem vollkommen anderen Hashwert führt. Diese Eigenschaft führt zwar zu einer sehr geringen Fehlerrate, führt aber auch dazu, dass nach der kleinsten Veränderung ein Inhalt nicht wiedererkannt wird.

Verfahren, bei denen optisch identische oder ähnliche Bilder gesucht werden, sind robust gegen kleine Veränderungen der Bildinhalte sowie gegen Änderung der Auflösungen, des Kontrastes oder ähnlichen Bildmerkmalen. Die gewünschte Übereinstimmungsgenauigkeit lässt sich dabei meist regeln, jedoch entstehen selbst bei Einstellung der höchstmöglichen Übereinstimmungsgenauigkeit höhere Fehlertoleranzen, als bei der Hashwertmethode. Die Ähnlichkeitserkennung stellt auch hohe Anforderungen an die Rechenleistung eines Scanners. Bei einer Suche in Echtzeit könnte es daher zu Verzögerungen kommen. Da das Verfahren außerdem auf bestimmte Medientypen beschränkt ist, wird es nicht weiter betrachtet.

3 Wiedererkennung von Inhalten in einem HTTP-Datenstrom

Das Ziel von AnonHash ist die Analyse des HTTP-Datenverkehrs eines Anonymisierungsdienstes in Echtzeit, um strafrechtlich relevante Inhalte zu blockieren. Hierzu wird ein hashwertbasiertes Verfahren verwendet. Das Bilden von Hashwerten und das Blockieren von Dateien in einem HTTP-Datenstrom führt zu zusätzlichen Problemen gegenüber dem Scannen eines Offline-Datenbestands. Die folgenden Abschnitte beschreiben diese Probleme. Dabei wird die Verwendung der HTTP-Version 1.1 angenommen, die zu früheren Versionen abwärtskompatibel ist.

3.1 Problembereich HTTP-Protokoll Eigenschaften

Wird der Hashwert einer Datei gebildet, die auf einem Datenträger gespeichert ist, so liegt die Datei bei Scannbeginn bereits komplett vor. Anfang und Ende der Datei sind genau bekannt und damit auch die exakte Länge des Inhalts.

Bei der Übermittlung von Dateien mittels HTTP ist das nicht der Fall. Hier wird den Inhalten ein HTTP-Header vorangestellt. Nach Übertragung des Headers folgt dann der HTTP-Body mit dem Dateiinhalt, sofern vorhanden. Um das Ende des Bodies bestimmen zu können, sieht HTTP/1.1 folgende Fälle vor [RFC2616]:

1. Alle Antworten auf einen HEAD-Request, sowie Responses mit den Statuscodes 1xx, 204 und 304, dürfen keinen Body enthalten. Dies gilt unabhängig von evtl. vorhandenen Headerfeldern.
2. Beim Vorliegen von *Transfer-Encoding* (chunked), kann die Länge des Bodies durch die angegebene chunk-size ermittelt werden oder durch Schließen der Verbindung (siehe Punkt 5).
3. Am Häufigsten wird das Body-Ende durch das Content-Length-Headerfeld beschrieben. Diese Angabe in Byte beschreibt die exakte Länge das nachfolgenden Inhalts.
4. Hat die Nachricht den Medientyp *multipart/byteranges* und ist die Länge der Nachricht nicht anderweitig zu ermitteln, so kann die Länge aus den Angaben dieses Medientyps errechnet werden.
5. Der Server kann das Ende der Response auch durch Schließen der Verbindung signalisieren. Der Client kann das Ende des Requests nicht auf diese Weise anzeigen, da der Server sonst keine Möglichkeit mehr hätte, eine Response zu schicken.

Es ist wichtig, das Ende des Bodies erkennen zu können, da nur auf diese Weise der Hashwert über dem gewünschten Dateninhalt gebildet werden kann. Außerdem ermöglicht HTTP/1.1 das sog. Pipelining. Dabei können in einer Verbindung mehrere Requests und Responses gesendet werden. Dies verringert den Aufwand für den Verbindungsauf- und abbau und ist häufig anzutreffen. Auch hier ist das Erkennen des Body-Endes sehr wichtig, um mit der Analyse des darauf folgenden Requests/Response an der richtigen Stelle beginnen zu können. Leider ist in der Praxis immer wieder nicht-Standard-konformes Verhalten anzutreffen. Beispiele hierfür sind Body-Daten bei einem Statuscode von 304, der keinen Body erlaubt oder ein Content-Length-Headerfeld, das eine falsche Längenangabe enthält.

Durch dieses Verhalten wird das Bilden von Hashwerten auf den richtigen Teilen der Pipeline enorm erschwert und zum Teil sogar unmöglich. In diesen Fällen muss entschieden werden, ob die betreffende Verbindung vom Scanner geschlossen wird, oder ob die nachfolgenden Daten in dieser Verbindung ohne Analyse übertragen werden. Bei Fehlern, die ein Abrufer absichtlich erzeugen kann, sollte die Verbindung geschlossen werden, da dies zur Umgehung des Scanners missbraucht werden könnte.

Zu beachten ist in diesem Zusammenhang auch, dass Header- und/oder Body-Daten absichtlich manipuliert werden können, um den Dienst zu beeinträchtigen oder um Daten ohne Analyse abrufen zu können. Diese Angriffe existieren beim Scannen von Offline-Daten nicht. Mögliche Angriffe auf AnonHash werden in Abschnitt 6 vertieft.

3.2 Speicherplatz und Latenzzeit

Ziel des Systems ist es, relevante Inhalte nicht nur zu erkennen, sondern auch zu blockieren. Das bedeutet, dass mit dem Ausliefern der Inhalte erst begonnen werden kann, wenn der Scannvorgang abgeschlossen ist und die Treffer-Entscheidung negativ war. Soll diese Entscheidung auf einem Hashwert basieren, der über der gesamten Datei gebildet wurde, so entstehen einige Schwierigkeiten: Erstens läuft das Zwischenspeichern der Daten für alle Verbindungen parallel ab. Bei sehr vielen Benutzern können die Speicherkapazitäten auf dem Scanner an ihre Grenzen stoßen. Zweitens muss die gesamte Datei auf dem Scanner zwischengespeichert werden. Solange der Scanner nun die Datei lädt und den Hashwert ermittelt, entstehen für den Abrufer höhere Latenzzeiten. Damit nicht der Eindruck entsteht, dass auf dieser Verbindung keine Daten übertragen werden bzw. um ein Timeout zu verhindern, könnten die ersten Bytes bereits vor Bekanntwerden des Scanergebnisses versandt werden. Diese Methode hat den Nachteil, dass das Analysieren und Ersetzen eines Inhalts aufwendiger wird, da berücksichtigt werden muss, dass bereits einige Bytes gesendet wurden.

Je größer die zu übertragenden Dateien sind, desto stärker fallen die genannten Schwierigkeiten ins Gewicht. Die Übertragung von mehreren Megabyte großen Dateien ist keine Seltenheit mehr, wodurch das Zwischenspeichern der Daten in der Praxis nur mit Modifikationen umsetzbar ist:

Variante 1: Hashwertbildung on-the-fly. Die Daten werden vom Scanner ohne Zwischenspeicherung durchgeleitet. Während des Durchleitens wird der Hashwert gebildet, der erst dann vorliegt, wenn das letzte Byte durchgeleitet wurde. Wird nun erkannt, dass es sich um einen strafrechtlich relevanten Inhalt gehandelt hat, so kann die betreffende URL für künftige Downloads gesperrt werden. Dazu müssen die betroffenen URLs gespeichert werden. Da bei dieser Modifikation keine Daten zwischengespeichert werden müssen, wird die Latenz kaum beeinflusst und es kann zu keinen Speicherproblemen kommen. Der große Nachteil bei dieser Modifikation ist, dass jeder bekannte strafrechtlich relevante Inhalt von jeder URL einmal abrufbar ist. Da der Scanner dezentral auf mehreren Mix-Kaskaden laufen wird, wäre dies jeweils pro Kaskade der Fall. Um das zu verhindern, müssten die Kaskaden untereinander ihre gesperrten URLs austauschen. An die Speicherung und den Austausch dieser URL-Listen müssten erhöhte Sicherheitsanforderungen gestellt werden, da eine Sammlung von URLs, die zu strafrechtlich relevanten Inhalten führen, nicht verbreitet werden sollte.

Das Speichern von URL-Listen bringt, v.a. bei Verwendung von dynamisch generierten Inhalten, weitere Probleme mit sich. So kann sich der Inhalt, auf den eine bestimmt URL verweist, dynamisch verändern. Dieselbe URL zeigt dann innerhalb kurzer Zeit auf ver-

schiedene Inhalte. Manche Seiten realisieren auf diese Weise eine Art „Picture of the day“. Dabei wird jeden Tag ein anderes Bild unter derselben URL veröffentlicht. Das hat zur Folge, dass u.U. nicht relevante Inhalte blockiert werden. Umgekehrt wird oft der gleiche Inhalt unter sich ständig ändernden URLs angeboten. So wird z.B. versucht, ein Deep-Linking zu verhindern. Dies führt dazu, dass der Inhalt nach jeder URL-Änderung wieder abrufbar ist. Bei adaptiven Inhalten, die für bestimmte Endgeräte und Übertragungsgeschwindigkeiten oder bestimmte Nutzer generiert werden, könnte das Anlegen von URL-Listen jedoch von Vorteil sein. Inhalte, die zu diesem Zweck angepasst oder verändert wurden, würden dann trotzdem blockiert werden können. Die zuvor genannten Probleme bestehen aber weiterhin.

Variante 2: Hashwertbildung nur auf Teilen des Inhalts. Die Hashwertbildung erfolgt nur auf einem bestimmten Anfangsteil und nicht mehr auf der kompletten Datei. Dadurch kann die Treffer-Entscheidung schon gefällt werden, sobald dieser Anfangsteil empfangen wurde. Ist die Entscheidung negativ, können die nachfolgenden Teile der Datei durchgeleitet werden. Durch diese Modifikation können Speicherbedarf und Latenz niedrig gehalten werden. Sie hängen natürlich entscheidend von der Größe des zu scannenden Anfangsteils ab. Hier muss ein Kompromiss gefunden werden. Je kleiner dieser Anfangsteil ist, umso weniger Daten müssen zwischengespeichert werden und umso schneller kann die Hashwertbildung erfolgen. Auf der anderen Seite muss der Anfangsteil auch groß genug gewählt werden, damit die Wahrscheinlichkeit, dass eine strafrechtlich nicht relevante Datei zufällig den gleichen Anfangsteil hat wie eine strafrechtlich relevante Datei, möglichst gering ist. Sollte es zu derartigen Kollisionen kommen, würde dies zu Fehlalarmen beim Scanner führen.

3.3 Designentscheidung

Auf Grund der genannten Nachteile, wird die Anwendung der Hashwertbildung on-the-fly für eine Implementierung ausgeschlossen. Bei der Wahl eines genügend großen Anfangsteils scheint die Hashwertbildung auf Teilen des Inhalts (Nachrichtenanfang) für den gewünschten Zweck am Besten geeignet zu sein. Eine sinnvolle Größe für den zu verwendenden Anfangsteil wurde in einer Analyse von jpg-Bildern ermittelt [Pöpp06]. Die ermittelten Grenzen für die Hashwertbildung sind verhältnismäßig klein, so dass bei den untersuchten 168.275 Realbildern bereits ab einer Anfangsteilgröße von 12 kB keine Kollisionen mehr auftraten. Dieser Wert lässt auch noch Spielraum für einen großzügigen Sicherheitsaufschlag.

4 Optimierte erfolglose Suche

Sobald bei einem Scanvorgang ein Hashwert einer Datei gebildet wurde, muss möglichst schnell entschieden werden, ob dieser Inhalt blockiert werden muss oder ob er ausgeliefert werden kann. Dazu muss überprüft werden, ob sich dieser Hashwert in der Datenbank

befindet. Herkömmliche Suchverfahren (z.B. binäre Suche, aber auch Hash-Verfahren) optimieren das schnelle Auffinden eines Datensatzes in einer Datenbank. Vermutlich werden Abrufversuche von strafrechtlich relevanten Inhalten jedoch nur einen sehr geringen Bruchteil aller Abrufe darstellen. Daher werden die gebildeten Hashwerte in den meisten Fällen nicht in der Datenbank enthalten sein (erfolglose Suche).

4.1 Nicht-Finden unter Verwendung eines BitSet

Zur Optimierung der erfolglosen Suche wird der Hashwert als nummerischer Wert betrachtet und als Index eines BitSets verwendet. Ein BitSet entspricht einem Array aus boolean-Werten. Der Begriff wurde aus der Java Dokumentation [Java06] übernommen, ist aber auch in anderen Programmiersprachen üblich. Befindet sich der Hashwert in der Datenbank, so wird der boolean-Wert an dieser Stelle 1 (true) gesetzt, sonst auf 0 (false). Soll nun überprüft werden, ob sich ein Hashwert in der Datenbank befindet, so muss nur der Wert an der entsprechenden Stelle des BitSets überprüft werden. Diese Überprüfung kann mit konstanter Komplexität $O(1)$ geschehen.

Bei der Verwendung von sehr langen Hashwerten stößt man mit diesem Verfahren jedoch sehr schnell an die Grenzen des zur Verfügung stehenden Arbeitsspeichers. MD5-Hashwerte sind beispielsweise 128 Bit lang. Das BitSet, das benötigt würde, müsste 2^{128} Bit umfassen, was etwa 38 Mio. Mrd. Mrd. Terrabyte entspricht.

Um den Speicheraufwand für das BitSet zu reduzieren, werden deshalb nur die ersten n Bit des Hashwertes herangezogen. Die verbleibenden Stellen werden ignoriert. Sollten mehrere Hashwerte in der Datenbank enthalten sein, deren erste n Bit identisch sind, so bleibt eine bereits bestehende 1 im BitSet erhalten. Eine 1 im BitSet besagt also, dass es mindestens einen Hashwert in der Datenbank gibt, dessen erste n Bit dem BitSet-Index entsprechen. Ob sich der zu überprüfende Hashwert tatsächlich in der Datenbank befindet, kann damit noch nicht entschieden werden. In diesem Fall ist eine Suche in der Datenbank notwendig. Das Suchverfahren kann beliebig gewählt werden. Eine binäre Suche beispielsweise verursacht logarithmischen Aufwand. Abbildung 2 zeigt ein Beispiel der BitSet-Überprüfung für $n = 24$.

Auf diese Weise lassen sich Suchläufe in der Datenbank zwar nicht vollständig vermeiden, die Häufigkeit einer Suche lässt sich dadurch aber stark reduzieren. Je mehr Nullen das BitSet enthält, desto größer ist die Wahrscheinlichkeit, dass auf eine Suche in der Datenbank verzichtet werden kann. Die Anzahl der Nullen hängt von der Größe des BitSets, von der Anzahl der Hashwerte und von der Verteilung der Hashwerte ab. Da man die Anzahl und die Verteilung der Hashwerte nicht beeinflussen kann, bleibt allein die Größe des BitSets als wählbare Größe. Die Anzahl der Hashwerte in der Datenbank wird als nicht beeinflussbar angenommen, da sie aus der Menge der bekannten strafrechtlich relevanten Inhalte resultiert, auf die man keinen Einfluss hat. Das BitSet sollte so groß wie möglich gewählt werden, da dadurch die Anzahl der Suchläufe in der Datenbank am stärksten reduziert werden kann.

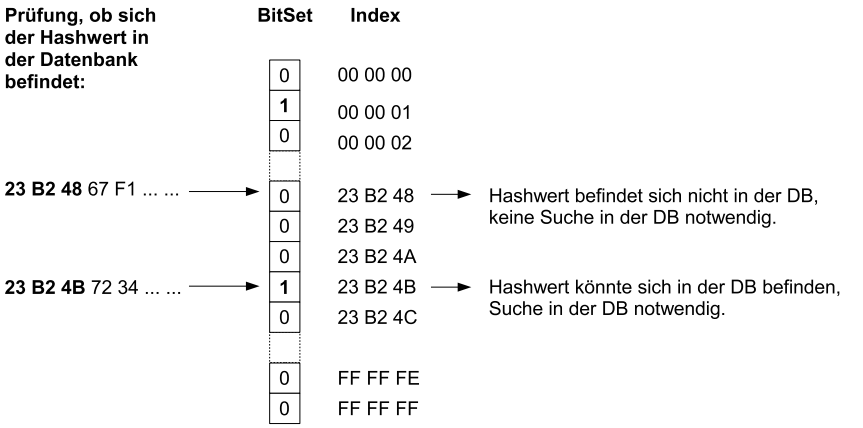

Abbildung 2: Suche mit BitSet, Überprüfen eines Hashwertes. Im BitSet werden 24 Bit gespeichert. Die restlichen Bits werden in einer Datenbank gespeichert.

4.2 Suchstrategie in AnonHash

Um den Scanvorgang weiter zu optimieren, wird ein zweistufiger Prozess vorgeschlagen. Damit nicht bei jedem Scanvorgang der Hashwert über dem gesamten definierten Anfangsteil einer Datei gebildet werden muss, wird zunächst ein Hashwert über einem kleineren Anfangsteil gebildet. Der zweite Hashwert muss nur dann gebildet werden, wenn der erste Hashwert einen möglichen Treffer signalisiert. Wie die entsprechenden Hashwertdatenbanken gebildet werden und wie sie aufgebaut sind, ist in [Pöpp06] ausführlich beschrieben.

Da AnonHash aufgrund des zweistufigen Scanverfahrens zwei Datenbanken verwendet, muss für jede Datenbank entschieden werden, ob und in welcher Größe ein BitSet verwendet werden sollte. Die Prüfung der verschiedenen Möglichkeiten hat ergeben, dass es am günstigsten ist, die erste Datenbank mit einem möglichst großen BitSet auszustatten und die zweite Datenbank ohne BitSet zu betreiben.

Da die zweite Datenbank nur abgefragt wird, wenn in der ersten Datenbank ein Treffer angezeigt wurde, wird diese viel seltener mit Anfragen belastet werden. Daher ist es sinnvoll, auf das BitSet der zweiten Datenbank zu verzichten und den Speicherbedarf dem ersten BitSet zur Verfügung zu stellen. Auf diese Weise können die Suchläufe in der ersten Datenbank noch stärker reduziert werden. Abbildung 3 verdeutlicht diese Konstellation. Allerdings muss in diesem Fall für jede Anfrage an die zweite Datenbank ein Suchlauf (hier: eine binäre Suche) erfolgen. Ist die Anzahl der Anfragen an die zweite Datenbank jedoch gering, ergibt sich insgesamt ein Performancevorteil.

Im Beispiel (Abbildung 3) sind von acht Anfragen an die Datenbank 1 nur noch zwei Suchläufe in der Datenbank 1 notwendig. Für jede positive Suche (hier beide) muss nun eine Anfrage an die Datenbank 2 gestellt werden. Da Datenbank 2 kein BitSet enthält, muss für jede Anfrage eine binäre Suche durchgeführt werden.

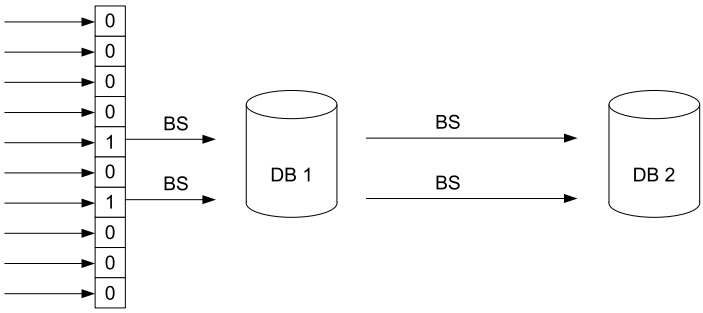

Abbildung 3: Nur Datenbank 1 mit BitSet

Eine weitere denkbare Optimierung wäre das (zeitweilige) Eintragen der URL des erkannten Inhalts in die Sperrliste des Cache-Proxys, sofern dieser über eine derartige Funktion zum dynamischen Hinzufügen von Sperreinträgen verfügt. Somit würde der Inhalt beim nächsten Abruf der URL bereits vom Cache-Proxy blockiert und belastet AnonHash nicht mehr (siehe auch Abbildung 1).

5 Performance

Erste Prototypen des Datenscanners wurden in Java entwickelt. Java hat den Ruf, für Serveranwendungen ungeeignet zu sein. Die I/O-Klassen ab Java 1.4 bieten jedoch die Möglichkeit, performante Serverprogramme zu entwickeln. Die Verwendung von Java als Programmiersprache erlaubt zudem einen plattformunabhängigen Einsatz von AnonHash.

In Zukunft werden auch Mixkaskaden mit einer Internetanbindung von mehr als 100 Mbit/s zur Verfügung stehen. Auf stark genutzten Mixkaskaden befinden sich in Stoßzeiten bis zu 2000 Benutzer. Um festzustellen, ob die AnonHash-Implementierung in Java den geforderten Datendurchsatz von 100 MBit/s erreichen kann, wurde ein Performancetest durchgeführt. In einer Testumgebung wurden zufallsgenerierte Dateien über AnonHash übertragen und die Übertragungsgeschwindigkeiten bzw. Übertragungsdauern protokolliert. Die Testläufe wurden sowohl mit sehr vielen kleinen (10.000×50 bzw. 500 kB), als auch mit einer sehr großen (500 MB) Datei durchgeführt.

Selbstverständlich verringert der Einsatz von AnonHash die Übertragungsgeschwindigkeit. Die Tests haben gezeigt, dass die Größe des Anfangsteils für die Hashwertbildung großen Einfluss auf die Übertragungsdauer hat. Betrachtet man die absoluten Werte, so erkennt man, dass AnonHash selbst auf „normaler“ Hardware (Testkonfiguration: Intel Pentium 4 mit 3 GHz, 1 GB RAM) noch sehr hohe Übertragungsraten erreichen kann: Bei der Übertragung einer großen Datei über Gigabit-Ethernet wurden je nach Scannereinstellung Übertragungsraten von ca. 600 MBit/s erreicht. Bei der Übertagung sehr vieler kleiner Dateien wurden immerhin die geforderten ca. 100 MBit/s erreicht. Abbildung 4 zeigt einen Ausschnitt der Performance-Test-Ergebnisse grafisch. Weitere Testergebnisse finden sich in [Pöpp06].

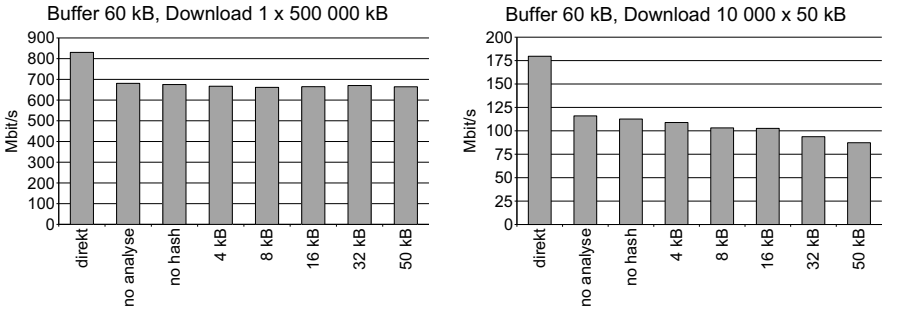

Abbildung 4: Performancetest-Ergebnisse grafisch [Pöpp06]. Die erste Säule gibt den Durchsatz ohne AnonHash an. Die zweite Säule entspricht dem Durchsatz von AnonHash bei abgeschalteter Analyse (Durchleitung aller Inhalte). Die dritte Säule gibt den Durchsatz bei aktivierter Analyse der Header, aber ohne Hashwertbildung an. Die letzten fünf Säulen zeigen die Durchsätze entsprechend der Blocklänge, auf der die Hashwerte gebildet werden.

6 Angriffe auf AnonHash

Für AnonHash wollen wir zusammenfassend von folgendem Angreifermodell ausgehen:

AnonHashbetreiber: AnonHashbetreiber haben grundsätzlich die volle Kontrolle über AnonHash. Da der Quellcode offen gelegt ist, kann die Funktionalität jederzeit eingeschränkt oder erweitert werden. Zusätzliche Angriffe auf die Anonymität der Nutzer sind auf Grund der Architektur von mixbasierten Anonymisierungsdiensten *nicht* möglich. AnonHashbetreiber werden als Angreifer auf das System ansonsten ausgeschlossen.

Inhalteanbieter: Der Inhalteanbieter verändert strafrechtlich relevante Inhalte *nicht*, d.h. aktive Angriffe (Manipulation der Inhalte zur Umgehung des Scanners) werden ausgeschlossen.

Abrufer von Inhalten: Der Nutzer (Abrufer) kann sein lokales System manipulieren, deswegen wird AnonHash zwischen letztem Mix und Cache-Proxy realisiert und nicht beim Nutzer. Ansonsten ist eine Umgehung des Filters durch Verzicht der Nutzung des Anonymisierers möglich.

Mixbetreiber: Es gilt das Angreifermodell des Mixnetzes. Durch den Einsatz des Scanners soll also die Anonymitätseigenschaft des Mix-Netzes unbedingt erhalten bleiben, selbst wenn ein abgerufener Inhalt blockiert wird.

Ein Ziel von Inhalteanbietern und Abrufern von Inhalten könnten gezielte (Denial-of-Service-Angriffe) auf AnonHash sein:

Alle Verbindungen in AnonHash werden unabhängig voneinander verarbeitet, so dass Verbindungen, in denen ein Fehler auftritt, von AnonHash geschlossen werden können, ohne die anderen Verbindungen zu beeinträchtigen. Auch durch das Senden sehr langer Request-Header, die die Kapazität des verwendeten Buffers übersteigen, ist kein erfolgreicher Angriff möglich. Die Verbindung wird in diesem Fall ebenfalls geschlossen. Werden nicht-Standard-konforme Daten gesendet, die nicht analysiert werden können, wird eben-

so verfahren. AnonHash-spezifische DoS-Angriffe durch Senden manipulierter Requests oder Responses sind somit nicht erfolgreich.

Würde es einem Angreifer gelingen, die Funktionalität von AnonHash zu beeinträchtigen, könnten die Betreiber eines Anonymisierungsdienstes wieder dazu übergehen, auf den Einsatz AnonHash zu verzichten. Unter dem definierten Angreifermodell existieren **verbleibende Angriffsmöglichkeiten** auf AnonHash:

AnonHash bildet für Dateien kleiner 1024 Byte keine Hashwerte, da davon ausgegangen wird, dass in so kleinen Dateien keine strafrechtlich relevanten Inhalte enthalten sind. Werden vom Webserver „Partielle Downloads“ angeboten, so kann ein Angreifer einen Anfangsteil einer Datei anfordern, der kleiner als 1024 Byte ist. Der Download der restlichen Datei ist ebenfalls ohne Erkennung möglich, da sich für den Datei-Rest kein Hashwert in der Datenbank befindet. AnonHash besitzt derzeit keine Funktion dies zu verhindern. Mögliche Lösungsansätze werden in [Pöpp06] beschrieben.

Beim Vorliegen eines Transfer-Encodings oder bei manchen Multipart-Nachrichten [RFC2616], wird die Analyse für die Verbindung deaktiviert. Werden auf dieser Verbindung im Rahmen des HTTP-Pipelinings weitere Inhalte übertragen, werden diese nicht mehr analysiert. Ob und wieviele Requests über eine Pipeline übertragen werden, hängt auch vom Server ab. Der Abrufer kann also den Download nicht gescannter Inhalte keineswegs erzwingen. Dieser Angriff kann verhindert werden, wenn in AnonHash eine Funktion implementiert wird, die es erlaubt, das Ende von Multipart-Nachrichten zu bestimmen bzw. *chunks* zu verfolgen.

Neben den beschriebenen Anriffen gibt es auch eine Reihe von Angriffen, die außerhalb des Angreifermodells liegen. Es handelt sich dabei vorwiegend um Anbieterangriffe, die eine Erkennung der angebotenen Inhalte verhindern sollen. Einer der wichtigsten Angiffe ist das Verändern der Inhalte. Wird auch nur ein Bit in dem Bereich verändert, über den der Hashwert gebildet wird, so schlägt die Erkennung fehl. Weitere Angriffe außerhalb des Angreifermodells werden in [Pöpp06] beschrieben.

7 Fazit und Ausblick

Die Verbreitung von Kinderpornografie und anderer strafrechtlich relevanter Inhalte über das Internet ist ein ernstzunehmendes Problem. Anonymisierungsdienste wie AN.ON werden oft mit dem Vorwurf konfrontiert, Straftäter bei der Verbreitung solcher Inhalte zu unterstützen. Mit der Integration eines Datenscanners in den Anonymisierungsdienst besteht nun die Möglichkeit, aktiv das Herunterladen von bekannten strafrechtlich relevanten Inhalten einzuschränken.

AnonHash ermöglicht die Analyse von HTTP-Datenströmen. Viele in AnonHash verwendete Techniken sind jedoch protokollunabhängig und können auch auf andere Datenströme angewandt werden. Dazu zählen u.a. die in Kapitel 4 entwickelte *optimierte erfolglose Suche*, der Aufbau der Hashwertdatenbanken und die Kollisionsuntersuchungen (vgl. dazu [Pöpp06]).

Um die Erkennung strafrechtlich relevanter Inhalte weiter zu verbessern, könnte das Scannen innerhalb von Archiven implementiert werden. Auch der Ausschluss des Datei-Headers bei der Hashwertbildung kann die Erkennungsrate weiter verbessern, wenn Dateien nur am Header verändert wurden, z.B. durch Drehen von Bildern.

Da bereits die Veränderung eines Bits den Hashwert einer Datei verändert, kann ein hashwertbasiertes Verfahren nicht gegen Anbieterangriffe schützen. So kann auch AnonHash, vom Anbieter veränderte Dateien nicht blockieren. Die Abrufer hingegen haben in der Regel nicht die Möglichkeit, eine Erkennung durch AnonHash zu umgehen. Den Abruf von bekannten (unveränderten) Inhalten kann AnonHash somit wirksam blockieren. Um die verbleibenden Abruferangriffe zu verhindern, die in Abschnitt 6 beschrieben wurden, sollten noch fehlende Elemente des HTTP-Protokolls nachgebildet werden. Da manche Client- und Serveranwendung z.T. fehlerhafte Requests oder Responses schicken, könnte die HTTP-Behandlung in AnonHash toleranter gestaltet werden.

Danksagung

Wir danken Dominik Herrmann fürs Korrekturlesen und den Gutachtern für ihre konstruktiven Hinweise, die zur Verbesserung dieses Papiers beigetragen haben. Wir danken weiterhin dem Bundeskriminalamt für die inhaltliche Unterstützung der bei der Entwicklung von AnonHash.

Literatur

[Anon06] Java Anon Proxy. http://anon.inf.tu-dresden.de, 2006.

[Chau81] Chaum, David: Untraceable Electronic Mail, Return Addresses, and Digital Pseudonyms. Communications of the ACM 24/2 (1981) 84-88.

[Java06] Sun Microsystems: J2SE Development Kit Documentation. http://java.sun.com/javase/downloads/index.jsp, Abruf am 22.08.2006.

[LWdW05] Lenstra, Arjen; Wang, Xiaoyun; de Weger, Benne: Colliding X.509 Certificates, http://eprint.iacr.org/2005/067.pdf, 2005.

[Pöpp06] Pöppl, Wolfgang: Integration eines Datenscanners in den Anonymisierungsdienst AN.ON, Diplomarbeit, Universität Regensburg, Lehrstuhl Management der Informationssicherheit, 2006.

[RFC2616] RFC 2616 Hypertext Transfer Protocol – HTTP/1.1, http://www.faqs.org/rfcs/rfc2616.html, Abruf am 23.09.2006