Umgehung von Firewalls auf Applikationsebene

Steffen Ullrich[1]

Kurzfassung:
Der größte Teil der heutigen Angriffe gegen Computersysteme findet über Mail und Web statt. Klassische Paketfilterfirewalls sind nicht in der Lage, diese Applikationsprotokolle zu analysieren und entsprechende Angriffe zu erkennen und abzuwehren. Mit Analysen auf Applikationsebene, wie Deep-Packet-Inspection oder Application-Level-Gateways, versprechen die Hersteller hingegen eine zuverlässige Abwehr derartiger Angriffe.
Wir haben untersucht, wie zuverlässig diese Analysen in der Realität sind. Dabei haben wir herausgefunden, dass die meisten Analysesysteme nur die typischerweise genutzten Teile der Applikationsprotokolle verstehen. Wenn hingegen seltene oder ungültige Protokollbestandteile benutzt werden, unterscheidet sich die Interpretation zwischen Client und Analysesystem. Damit war es uns möglich, bekannte Malware unbemerkt und zuverlässig durch existierende Analysesysteme zu schleusen.
Um möglichst viele Systeme zu untersuchen, haben wir zusätzlich zur manuellen Verhaltens- und Sourcecodeanalyse ausgewählter Systeme eine öffentlich verfügbare Testumgebung geschaffen, die einen Test der lokalen Firewall mittels eines einfachen Webbrowsers ermöglicht. Die Auswertung der dabei auf dem Testserver anfallenden Logdaten gab uns Einblick in eine Vielzahl von Schwächen der in der Praxis genutzten Systeme vieler Hersteller. Während einige dieser Probleme durch eine gründlichere Filterung behoben werden könnten, stellen andere prinzipielle Schwächen der genutzten Analysetechniken dar.

Stichworte: HTTP, MIME, Firewall, DPI, IDS, IPS, NGFW, ALG, Semantic Gap, Evasion, AET

1. Einleitung

Klassische Firewalls erlauben im Wesentlichen nur eine Filterung des Datenverkehrs basierend auf Quelle und Ziel, also IP-Adressen und Portnummern. Die meisten heutigen Angriffe finden jedoch auf Applikationsebene statt, typischerweise durch die Übertragung von Malware über Mail oder im Web. So wurden laut Angaben der Sicherheitsfirma Palo Alto Networks 2013 25% der Malware über Mail und 68% über das Web übertragen [1]. Moderne Next-Generation-Firewalls (NGFW), Unified-Threat-Managment-Lösungen (UTM), Intrusion-Detection-Systeme (IDS) oder Breach-Detection-Systeme (BDS) sind daher in der Lage, mit Techniken wie zum Beispiel Deep-Packet-Inspection (DPI) oder als Application-Level-Gateways (ALG), eine Analyse auf Applikationsebene durchzuführen. Diese soll es möglich machen, Malware und andere Angriffe in übertragenen E-Mails und Webzugriffen zu erkennen. Oftmals ist dies sogar möglich, wenn diese Übertragungen über eigentlich verschlüsselte Verbindungen wie HTTPS stattfinden.

Man könnte annehmen, dass sich sowohl die Analysesysteme (Firewall, IDS) wie auch die Endpunkte der Übertragung (Browser, Server, Mail-Programme) an die

1 genua GmbH, Kirchheim

existierenden Standards halten, sodass Abweichungen vom Standard als ungültig eingestuft werden und zum Abbruch der Übertragung führen. In der Praxis ist jedoch sowohl bei den Sendern, Analysesystemen und Empfängern der Daten ein eher informeller Umgang mit den Standards zu beobachten. Typischerweise werden nur die häufig genutzten Teile der Standards korrekt implementiert. Selten genutzte Bestandteile oder gar Abweichungen vom Standard werden meist trotzdem akzeptiert, aber oft auf unterschiedliche Weise in den beteiligten Systemen interpretiert.

Durch gezielte Nutzung derartiger Interpretationsunterschiede ist es einem Sender möglich, eine andere Auslegung der Daten im Analysesystem verglichen zum Empfänger zu bewirken. Dieses führt zum Beispiel dazu, dass der Virenscanner auf der Firewall nur Datenmüll zu Analyse bekommt, während im Browser oder Mailprogramm auf dem Endsystem die Daten in der vom Angreifer gewünschten Weise extrahiert werden. So ist es uns auf diese Weise gelungen, bekannte und unveränderte Malware von den meisten Firewalls unbemerkt erfolgreich zum Empfänger zu transportieren.

Nachfolgend werden Umgehungstechniken gezeigt, welche gegen den größten Teil der Enterprise-Firewalls, IDS und Antivirus-Produkte wirksam sind. Der erste Schwerpunkt liegt dabei auf der unentdeckten Auslieferung von Malware im Web durch Variationen des Applikationsprotokolls HTTP. Der zweite Schwerpunkt untersucht die Umgehung der Analyse von E-Mails durch die Manipulation des MIME-Formats. Nach Beschreibung dieser Umgehungsmöglichkeiten wird diskutiert, welche Möglichkeiten Analysesysteme haben, um diesen Angriffen zu begegnen, und wo durch die eingesetzte Technologie oder dem Wunsch nach hoher Performance Grenzen bei der adäquaten Behandlung dieser Angriffe existieren.

2. Stand der Forschung

Umgehungen der Analyse auf Ebene der Applikationsprotokolle wurden bisher nur wenig untersucht. Vergleichbare Angriffe auf Basis unterschiedlicher Interpretation von Daten waren hingegen bereits öfter Thema von Untersuchungen. So wurde 2013 mit der Evader-Software gezeigt, wie einfach vorhandene Firewalls durch Ausnutzen von Interpretationsunterschieden auf der Netzwerk- und Transportebene (und eingeschränkt auch auf Applikationsebene) umgangen werden können [2]. Zwar wurden nachfolgend viele Firewalls gegen diese sogenannten Advanced-Evasion-Techniques (AET) gehärtet. Unsere Untersuchungen zeigen jedoch, dass diese Systeme auf Ebene der Applikationsprotokolle weiterhin umgangen werden können.

Umgehungsmöglichkeiten innerhalb der Nutzdaten selbst wurden ebenfalls untersucht. So kann mit auf verschiedene Weisen interpretierbaren Dateien (z.B. Polyglot-Dokumenten) dafür gesorgt werden, dass ein Antivirus-Produkt die Daten anders als das Zielsystem interpretiert [3][4][5][6] und Schadcode daher nicht erkennt.

Ein Angriff auf Protokollebene hingegen ist das HTTP-Response-Splitting, welches jedoch weniger aus der Umgehung von Analysen als für Cache-Poisoning bekannt ist [7]. Ebenso können zum Beispiel über Manipulation des Host-Headers bei HTTP Zugriffsbeschränkungen einiger Firewalls umgangen werden [8].

3. Generelle Ursachen für die Umgehungsmöglichkeiten

Sowohl HTTP als auch MIME sind Standards, die mit den Zielen der Lesbarkeit durch den Menschen, Flexibilität und einfachen Erweiterbarkeit entwickelt wurden. Dabei wurden jedoch unerwünschte Nebeneffekte dieser Designentscheidung nicht gezielt adressiert. Das führt unter anderem dazu, dass

- Implementierungen sich unterscheiden in der Akzeptanz verschiedener Arten, Menge oder Positionen von Leerzeichen und Zeilenenden sowie der Groß-/ Kleinschreibung von Buchstaben, da diesbezügliche Unterschiede für einen menschlichen Leser meist irrelevant sind.
- die im Standard enthaltene Flexibilität unterschiedlich ausgeprägt implementiert wird, da sie in der Praxis nicht in vollem Umfang benötigt wird.
- es möglich ist, für die Interpretation der Inhalte wichtige Meta-Informationen auf verschiedene Weise, mehrfach und auch widersprüchlich anzugeben und so je nach Implementation unterschiedliche Verarbeitung zu bewirken.

Dazu kommt, dass Implementierungen in Clients überwiegend davon ausgehen, korrekte Inhalte zu bekommen. Zwar werden aus Gründen der Robustheit vereinzelt bestimmte inkorrekte Daten gezielt akzeptiert. Die meisten ungültigen Inhalte werden jedoch unabsichtlich akzeptiert und in der Annahme verarbeitet, dass es sich um valide Daten handelt. Das Ergebnis ist in diesen Fällen ein unbeabsichtigter Nebeneffekt der jeweiligen Implementation, statt einer sauberen Fehlerbehandlung. Im Gegensatz zu der, in den meisten Fällen implementierungsübergreifend einheitlichen, bewussten Robustheit gegenüber bestimmten Protokollfehlern, führt diese unabsichtliche Toleranz in vielen Fällen zu unterschiedlicher Interpretation der gleichen Inhalte.

4. Web: Umgehungen über Modifikation des HTTP-Protokolls

Das Hypertext-Transfer-Protocol (HTTP) ist das Applikationsprotokoll, welches zwischen Browser und Webserver zur Übertragung von Webseiten, Medien und Downloads eingesetzt wird. Die früheste Variante HTTP 0.9 [9] wurde 1991 eingesetzt und ca. 1996 von der deutlich komplexeren Variante HTTP/1.0 [10] abgelöst. 1999 wurde diese weiter erweitert zu HTTP/1.1 [11], welches das derzeit meistgenutzte Protokoll ist. Die aktuellste Entwicklung ist das grundlegend überarbeitete und erneut deutlich komplexere Protokoll HTTP/2 [12], welches zwar von vielen Browsern, aber bisher nur vergleichsweise wenigen Webseiten unterstützt wird. Bei Nutzung von Firewalls wird technisch bedingt zumeist automatisch HTTP/1.1 benutzt, sodass die fehlende Unterstützung für HTTP/2 in den meisten Firewalls kein Sicherheitsproblem darstellt. Im Folgenden wird HTTP/2 daher nicht weiter betrachtet. Der typischer Einsatz einer Firewall zur Analyse von Webverkehr ist in Abbildung 1 gezeigt.

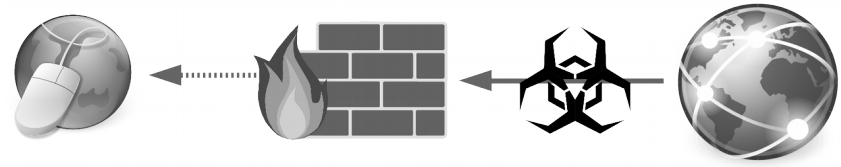

Abbildung 1: Perimeterfirewall für Web - die Firewall blockiert in der Antwort des Servers enthaltene Malware

4.1 Testaufbau

Um eine große Anzahl an Systemen möglichst einfach zu testen, wurde ein Framework zur Generierung von Tests geschaffen, bei dem man sich mit einem normalen Browser über eine Firewall hinweg zu einem Testserver verbindet. Dieser liefert den ungefährlichen Testvirus EICAR [13] aus, welcher explizit für derartige Tests gedacht ist und von praktisch allen Antivirus-Produkten erkannt wird. Nachdem initial verifiziert wurde, dass die Firewall tatsächlich diesen Testvirus erkennt, wird im weiteren Verlauf des Tests der Testvirus automatisch nacheinander in verschiedenen Varianten abgerufen und überprüft, ob der Virus unverändert die Firewall passieren konnte. Derartige erfolgreiche Umgehungsversuche der Analyse werden dem Nutzer im Browser angezeigt (siehe Abbildung 2) sowie an den Testserver zurückgemeldet. Da die Interpretation der Varianten auch abhängig vom Browser ist, sollte der Test entsprechend mit verschiedenen Browsern wiederholt werden.

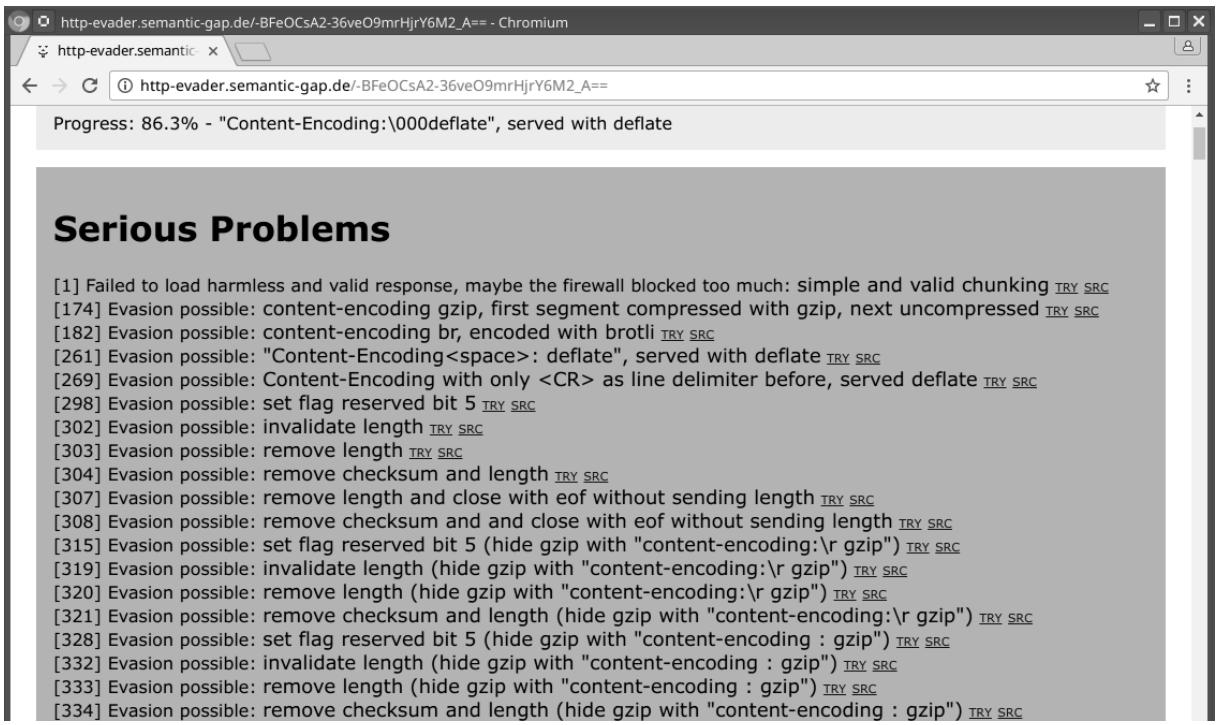

Abbildung 2: Bildschirmfoto vom Browser während eines Testlaufes

Im September 2015 wurde ein öffentlicher Testserver aufgesetzt und bekannt gemacht [14]. Die zahlreichen Zugriffe aus aller Welt mit unterschiedlichen Firewalls erlaubten einen tiefen Einblick in die Fähigkeiten und Probleme verschiedener Analysesysteme und führten zu einer kontinuierlichen Erweiterung der Tests auf derzeit über 750 Einzeltests, die automatisiert innerhalb weniger Minuten im Browser ausgeführt werden können. Weiterhin wurden Quelltextanalysen des HTTP-Stacks in den Browsern Firefox und Chromium sowie der IDS Snort und Suricata vorgenommen und darauf basierend Tests für weitere Umgehungen hinzugefügt.

Die Rückmeldungen der Browser während der Analyse erlauben in vielen Fällen eine Zuordnung zu bestimmten Firewallprodukten oder Herstellern, zum Beispiel anhand der gelieferten Fehlermeldung beim erfolgreichen Blockieren des Testvirus oder auch weil die ausgehende IP-Adresse eindeutig einem bestimmten Firewallhersteller zugeordnet werden kann. Unter den auf diese Weise erkannten Systemen fanden sich auch 12 Produkte, welche von Gartner 2015 zu den besten Firewalls für Enterprise, UTM beziehungsweise Cloud-Security gezählt wurden, im weiteren als Top-Firewalls bezeichnet.

4.2 Einführung in HTTP

Sowohl die Anfragen des Clients, wie auch die Antwort des Servers, bestehen bei HTTP/1.x aus dem *HTTP-Header* und dem *HTTP-Body*, welche durch eine Leerzeile getrennt sind. In seiner Anfrage spezifiziert der Client unter anderem, auf welche Ressource er zugreifen möchte und welche Kompressionsalgorithmen er unterstützt. Wir haben uns in der Forschung auf die Umgehung der Analyse durch passende Antworten des Servers konzentriert und gehen daher nicht tiefer auf die Anfragen des Clients ein. Entsprechende Untersuchungen wären jedoch bei der Analyse von Web-Application-Firewalls (WAF) interessant.

Eine typische (vereinfachte) Serverantwort sieht wie folgt aus:

```
HTTP/1.1 200 ok\r\n
Content-Encoding: gzip\r\n
Content-Type: text/html\r\n
Content-Length: 200\r\n
\r\n
... mit gzip komprimierter Inhalt ...
```

Die Antwort enthält die Protokollversion (HTTP/1.1). Der Statuscode 200 bedeutet, dass die Anfrage erfolgreich beantwortet wurde. Über *Content-Encoding* gibt der Server an, dass der Inhalt mit *gzip* komprimiert ist und mittels *Content-Length* wird die Länge des auf die Leerzeile folgenden HTTP-Body angegeben.

An diesem einfachen Beispiel werden bereits viele Probleme erkennbar, die in einer erfolgreichen Umgehung des Analyse-Systems münden können. So lassen sich

- ungültige Protokollversionen wie `http/1.2` angeben,
- fehlerhafte Statuscodes wie `0200` benutzen,
- Zeilenenden von „`\n`“ oder „`\r`“ statt „`\r\n`“ benutzen,

- einzelne Leerzeichen mit Tabulator oder mehreren Leerzeichen ersetzen,
- die Content-Length mehrfach angeben,
- ein Leerzeichen mit in die trennende Leerzeile geben und vieles mehr.

Die Browser sind hier im allgemeinen recht flexibel, verhalten sich aber in kleinen, jedoch entscheidenden Details, oft anders als die Analysesysteme. Das kann dazu führen, dass eine Firewall übertragene Daten anders interpretiert als der Browser, was insbesondere bei der Übertragung von Malware ein Problem darstellt. Im Rahmen unserer Untersuchungen wurde eine Vielfalt solcher Umgehungen der Analyse gefunden, von denen wir einige im Folgenden genauer beschreiben.

4.3 Umgehung über HTTP 0.9

HTTP 0.9 ist das zuerst im Web benutzte Übertragungsprotokoll. Es ist deutlich einfacher als HTTP/1.x: es gibt keine Auftrennung in den HTTP-Header mit Meta-Informationen und den HTTP-Body sondern die Antwort des Webservers besteht einzig aus dem eigentlichen Dokument welches mit dem Schließen der TCP-Verbindung endet. Obwohl HTTP 0.9 bereits im Jahre 1996 durch HTTP/1.0 abgelöst wurde, unterstützten Ende 2016 noch fast alle Webbrowser dieses alte Protokoll, auch wenn die Unterstützung langsam abnimmt [15]. Unsere Untersuchungen zeigten, dass Ende 2015 die Malwareanalyse bei ca. 60% der Top-Firewalls umgangen werden konnte, indem der Webserver des Angreifers mit HTTP 0.9 geantwortet hat [16].

4.4 Umgehung durch Mehrfachkompression

Als Beispiel für eine in der Praxis nicht benötigte Flexibilität unterstützt der HTTP-Standard das beliebige Verschachteln von Kompressionsmethoden, sowohl durch Angabe mehrere *Content-Encoding-Header,* wie auch durch Angabe mehrere Methoden innerhalb des gleichen Headers. So bedeutet der folgende HTTP-Header, dass der Body zuerst mit *gzip*, dann mit *deflate* und schließlich erneut mit *gzip* komprimiert wurde:

```
HTTP/1.1 200 ok\r\n
Content-Encoding: gzip\r\n
Content-Encoding: deflate, gzip\r\n
\r\n
```

Das Verhalten von Browsern angesichts dieses Features ist unterschiedlich. Während Firefox, Chrome und Opera eine dreifache Kompression entsprechend dem Standard erwarten, interpretiert Safari nur den ersten Header und erwartet gzip. Microsoft Internet-Explorer (MSIE) und Microsoft Edge (MS Edge) ignorieren die Mehrfachangabe schlichtweg und erwarten den Inhalt daher unkomprimiert. Analysesysteme verhalten sich ebenfalls unterschiedlich und in vielen Fällen anders als die Browser: obwohl diese Problematik bereits 2013 veröffentlicht wurde [17], zeigten unsere Tests Ende 2015, dass 50% der Top-Firewalls auf diese Weise versteckte Malware ungehindert passieren ließen [18].

4.5 Umgehung mittels modifizierter gzip-Kompression

Ebenfalls überflüssige Flexibilität beinhaltet die Verwendung von gzip zur Kompression in HTTP [19]. Im Gegensatz zur ebenfalls von den Browsern unterstützten Kompressionsmethode *deflate* werden bei bei gzip die komprimierten Daten in einen für die Nutzung in HTTP unnötig flexiblen Container verpackt. Dieser beginnt mit einem Header, welcher unter anderem aus diversen reservierten Bits und mehreren optionalen Features wie Dateinamen, Kommentar und einer Checksumme über den Header besteht (siehe Abbildung 3). Nach dem Header folgen die komprimierten Daten und am Ende des Containers ein Trailer mit Angaben zur originalen Gesamtgröße und einer Checksumme über den Inhalt.

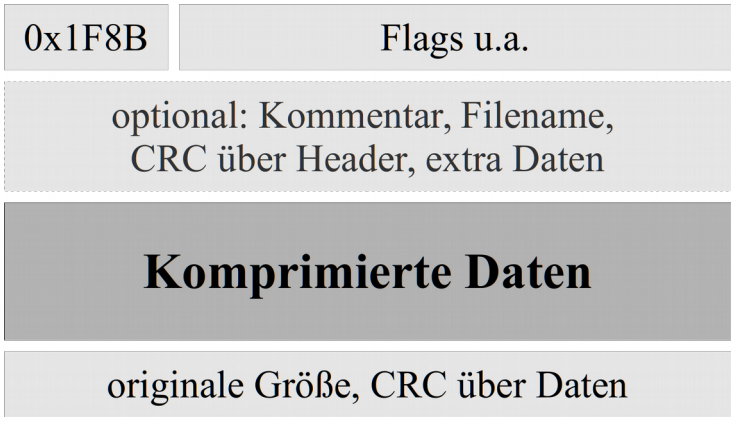

Abbildung 3: Aufbau des Kompressionsformates gzip

Keine der Informationen aus Header und Trailer, sondern ausschließlich die komprimierten Daten, werden für die Nutzung in HTTP gebraucht. Entsprechend ignorieren Chrome und Opera die Werte von reservierten Bits, Chrome, Opera und Firefox eine ungültige Checksumme über den Header und sämtliche Browser fehlerhafte oder fehlende Angaben im Trailer zur Gesamtlänge oder Checksumme.

Analysesysteme hingegen verwenden oftmals gängige gzip-Implementierungen, welche Wert auf die Korrektheit dieser, für HTTP unnötigen, Informationen legen. Damit scheitern sie bei dem Entpacken der Daten zur Analyse. Statt jedoch in diesem Fall die Inhalte sicherheitshalber zu blockieren, werden sie zumeist weitergeleitet. So haben unseren Untersuchungen zufolge Ende 2015 60% der Top-Firewalls mit modifiziertem gzip komprimierte Malware ungehindert weitergeleitet [20].

Das die meisten Firewalls nicht verarbeitbare Inhalte lieber weitergeleiten statt zu blockieren, wurde von uns bei vielen Umgehungen beobachtet. Wir vermuten als wesentliches Ziel dieser Vorgehensweise die Vermeidung von Fehlalarmen, das heisst das ungewollte Blockieren ungefährlicher aber von der Firewall nicht verstandener Inhalte.

4.6 Umgehungen durch Variationen von Leerzeichen und Zeilenenden

Die Konzeption von HTTP als ein vom Menschen lesbares Protokoll führt bei der Implementierung des Protokolls in Browsern und Analysesystemen zu einer großen, aber uneinheitlichen, Flexibilität gegenüber Variationen von Leerzeichen und Zeilenenden [21]. So sehen die ursprünglichen HTTP/1.x Standards vor, dass lange Header-Zeilen in der nächsten Zeile weitergeführt werden können, obwohl HTTP keinerlei Limitierungen der Zeilenlänge im HTTP-Header besitzt. Demzufolge ist die Unterstützung dafür unterschiedlich ausgeprägt: MSIE und MS Edge unterstützen dies, im Gegensatz zu anderen Browsern, nicht. Anderseits erlauben MSIE und MS Edge konträr zum Standard Leerzeichen am Zeilenanfang. Das führt dazu, dass in folgender Antwort der für die Interpretation des Inhalts essentielle Header Content-Encoding von einigen Browsern und Analysesystemen gesehen wird und von anderen nicht, da letztere es als Weiterführung der vorhergehenden Zeile betrachten:

```
HTTP/1.1 200 ok\r\n
X-Foo: bar\r\n
<space>Content-Encoding: deflate\r\n
\r\n
```

Die Manipulation des Trenners nach dem HTTP-Header kann ebenfalls eine unterschiedliche Interpretation bewirken. Laut Standard ist der Trenner zwischen Header und Body eine Leerzeile, genau gesagt „`\r\n`“, wobei alle Browser auch ein einzelnes „`\n`“ akzeptieren. MSIE und MS Edge jedoch akzeptieren auch Leerzeichen in der Zeile und Safari zumindest auch zusätzliche „`\r`“. Somit beginnt für diese Browser und viele Analysesysteme im folgenden Beispiel der Header vor der Angabe des Content-Encoding, für alle anderen erst danach mit dem finalen „`\r\n`“. Dieses führt zu einer unterschiedlichen Auslegung der nachfolgenden Daten: die Firewall sieht eventuell nur unverständlichen Datenmüll während der Browser die mit deflate komprimierte Malware sieht.

```
HTTP/1.1 200 ok\r\n
\r\r\n
Content-Encoding: deflate\r\n
\r\n
... mit deflate komprimierte Malware ...
\r\n
```

Unsere Untersuchungen zeigen, dass mit diesen und ähnlichen Variationen die Analyse in praktisch allen untersuchten Firewalls umgangen werden kann.

4.7 Zusammenfassung der Testergebnisse

Die durch mit unserem öffentlichen Testserver gesammelten Informationen zeigen, dass bei nahezu allen untersuchten Systemen die Analyse umgangen werden konnte. Erschreckend dabei ist, dass selbst die laut Gartner besten Firewalls für Enterprise, UTM beziehungsweise Cloud-Security auf die beschriebene einfache Weise umgangen werden können. Die betroffenen Hersteller wurden informiert und einige verbesserten

in der Folge ihre Produkte [22][23][24][25]. Allerdings gibt es auch nach diesen Verbesserungen weiterhin nur wenige Produkte, welche in der Standardkonfiguration sämtliche Umgehungsversuche vereiteln.

5. E-Mail: Umgehungen durch Manipulation des MIME-Formats

Die Übertragung von E-Mails ist deutlich älter als die Übertragung von Webseiten und die vor mehr als 30 Jahren notwendigen Beschränkungen haben in vieler Hinsicht bis heute Bestand. So bestand zu damaliger Zeit eine E-Mail nur aus ASCII-Zeichen und die Zeilenlänge war auf weniger als 1000 Zeichen beschränkt (empfohlen waren maximal 78 Zeichen). Um unter diesen Bedingungen beliebige Zeichen und flexible Zeilenlängen verwenden und auch binäre Anhänge übertragen zu können, wurde die Multipurpose Internet Mail Extension (MIME) [26] entwickelt, welche zum Einen ein textbasiertes Containerformat beschreibt, um beliebige Inhalte mit den bisherigen Restriktionen abbilden zu können und zum Anderen definiert, wie z.B. Umlaute abwärtskompatibel im Mail-Header für die Nutzung in Absendernamen, Betreffzeile usw. benutzt werden können. Da jedoch MIME ebenso wie HTTP mit dem Ziel der Flexibilität und Lesbarkeit entwickelt wurde, gibt es ebenfalls viele Möglichkeiten, Inhalte so zu manipulieren, dass sie von verschiedenen Systemen unterschiedlich interpretiert werden und auf diese Weise Analysen umgangen werden können. Im Rahmen der durchgeführten Untersuchungen im in Abbildung 4 gezeigten Szenario wurde eine Vielzahl derartiger Umgehungen entdeckt, von denen im Folgenden nur einige detaillierter beschrieben werden.

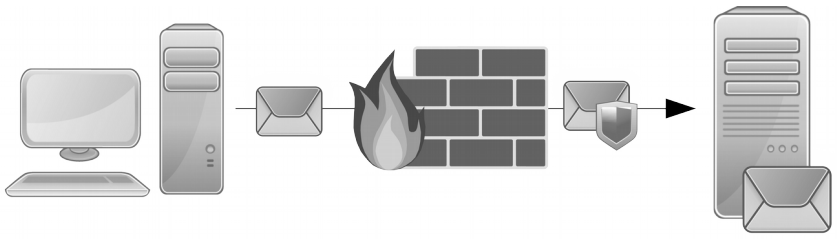

Abbildung 4: vom Client gesendete E-Mails werden von der Firewall von Malware gesäubert an den Mailserver weitergeleitet

5.1 Testaufbau

Es wurde ein Testframework entwickelt, welches über 300 Varianten einer den EICAR-Testvirus transportierenden Mail zum Testen erstellt. Dazu kommen noch über 100 Varianten, den Dateinamen zu deklarieren, um damit das in einigen Firewalls implementierte generische Blockieren von Anhängen basierend auf dem Dateinamenssuffix zu testen. Die Ideen für die Tests basieren dabei primär auf der Analyse der Standards sowie auf der Quelltextanalyse des Mailprogrammes Thunderbird, des IDS Snort sowie von MIME-Bibliotheken verschiedener Programmiersprachen.

Leider können übliche Mail-Clients nicht in der gleichen Weise für automatische Tests instrumentiert werden wie ein Webbrowser, sodass kein öffentliches Testsystem erstellt wurde. Die Analyse beschränkte sich daher bisher auf eigene Tests mit freien

Systemen wie dem IDS Snort, dem Virenscanner ClamAV und dem Mailfilter amavisd, sowie Testversionen einzelner kommerzieller Systeme.

In ersten Schritt wurde manuell das Verhalten von häufig genutzten Mailprogrammen wie Thunderbird, Microsoft Outlook und Apple-Mail für alle Testvarianten einer Mail ermittelt. Anschließend wurden Untersuchungen mit Firewalls, IDS, Virenscannern und Mailfiltern gemacht, um so Unterschiede in der Interpretation zwischen Analysesystem und Mailprogramm zu ermitteln. Zum Testen von Firewalls wurden die generierten Testmails durch eigene Programme automatisch über eine manuell konfigurierte Firewall verschickt und am anderen Ende empfangen. Zum Testen von Mailfiltern oder Virenscannern wurden die Testmails direkt an diese übergeben. Eine Reihe eigener Tools unterstützten bei der Analyse, ob die Malware erkannt oder entschärft wurde, also ob die Mail blockiert, die Malware entfernt oder aber die Mail so normalisiert wurde, dass die Malware nicht mehr aktiv werden kann.

Zusätzlich wurden manuelle Tests mit ausgewählten Webmailern gemacht, welche eingehende Mails auf Malware scannen. Ziel war es dabei zu ermitteln, ob Interpretationsunterschiede zwischen dem Virenscanner und der Darstellung der Mail im Browser ausnutzbar sind.

5.2 Einführung in MIME

Der grundlegende Aufbau einer MIME-Nachricht mit einem Text und einem Anhang sieht wie folgt aus:

```
 1 From: me@example.com
 2 To: you@example.com
 3 Subject: Viele =?UTF-8?Q?Gr=C3=BC=C3=9Fe?=
 4 Content-type: multipart/mixed; boundary=trenner
 5
 6 --trenner
 7 Content-type: text/plain; charset=UTF-8
 8 Content-Transfer-Encoding: quoted-printable
 9
10 Viele Gr=C3=BC=C3=9Fe von mir.
11 --trenner
12 Content-type: application/zip; name=test.zip
13 Content-Transfer-Encoding: base64
14
15 UEsDBAoAAAAAAO6Tn0m2hH8nEwAAABMAAAAIA ....
   ...
18 AQABAE4AAABVAAAAAAA=
19 --trenner--
```

In Zeile 4 wird im zentralen Mail-Header ein *Content-Type* der Art „multipart“ mit einem dazugehörigen Trenner (Boundary) definiert. Dieser wird in Zeilen 6 und 11 genutzt, um die folgenden Daten in einzelne Abschnitte zu unterteilen sowie in Zeile 19 das Ende des Multiparts zu markieren. Jeder der Abschnitte hat einen eigenen

Header, in welchem mit Content-Type die Art und wenn nötig mit *Content-Transfer-Encoding* die Kodierung beschrieben ist. Eine Kodierung ist notwendig, um Zeichen außerhalb von ASCII in der Mail darzustellen. Der MIME-Standard definiert dazu die Kodierungen *Base64* für binäre Daten und *Quoted-Printable* für Daten, die primär ASCII-Zeichen enthalten. Base64 wird im Beispiel in Zeilen 15..18 für das binäre Attachment verwendet, während Quoted-Printable sowohl in Zeile 10 für den Text der Mail wie auch in Zeile 3 zur Einbettung von Umlauten in die Betreffzeile benutzt wird.

5.3 Umgehung mit manipuliertem Boundary

Die korrekte Erkennung des Boundary bei einer Mulitpart-Mail ist essentiell, um die einzelnen Teile richtig zu extrahieren. Werden wie im folgenden Beispiel mehrere Trenner deklariert, so kann dieses erneut zu Interpretationsdifferenzen zwischen Mail-Client und Analysesystem führen [27]:

```
 1 Content-type: multipart/mixed; boundary=bb; boundary=##
 2
 3 --bb
 4 Content-type: application/zip; name=virus.zip
 5 Content-Transfer-Encoding: base64
 6
 7 --##
 8
 9 UEsDBAoAAAAAAO6Tn0m2hH8nEwAAAB....

12 AQABAE4AAABVAAAAAAA=
13 --##--
14 --bb--
```

Ein Client, der den ersten Trenner benutzt, wird die Zeilen 4 und 5 als Header des Abschnitts ansehen, der bis zur Zeile 13 geht. Da die meisten Clients nicht zu Base64 gehörende Zeichen wie „-“ und „#“ ignorieren, wird das Attachment korrekt extrahiert. Ein Analysesystem, welches hingegen den zweiten Trenner benutzt, sieht nur die Daten von Zeile 8 bis 12 und erkennt nicht, dass es sich hier um Base64 handelt.

5.4 Umgehung durch Manipulation des Content-Transfer-Encoding-Headers

Ähnlich wie beim Boundary, lassen sich auch mehrere Deklarationen des Content-Transfer-Encoding vornehmen. Welche davon berücksichtigt wird, ist wiederum abhängig von der jeweiligen Implementation [28]. Zusätzlich akzeptieren Mail-Clients eine große Bandbreite an Variationen der Definition, wovon zumeist nur ein Bruchteil vom Analysesystem ebenfalls identisch interpretiert wird. In der Praxis akzeptierte Variationen für „base64“ zeigt das folgende Beispiel:

```
Content-Transfer-Encoding: ,base64
Content-Transfer-Encoding: \rbase64
Content-Transfer-Encoding: xx base64
Content-Transfer-Encoding: =?UTF-8?B?YmFzZTY0?=
```

Die letzte Zeile ist ein typisches Beispiel für die Verwendung eines richtigen Standards an der falschen Stelle. Hier wird vom Mail-Client eine Kodierung entsprechend RFC 2047 akzeptiert, welche für die Darstellung von Umlauten u.ä. in der Betreffzeile und ähnlichen Mail-Headern gedacht ist, aber nicht für Content-Transfer-Encoding oder bei der Definition des Multipart-Boundary. MIME hat mehrere derartige Regeln, wie bestimmte Umformungen nur an spezifischen Stellen benutzt werden dürfen oder wann Semikolon und wann Komma als Trenner dienen. Auf Grund dieser oftmals unnötigen Komplexität ist es wenig verwunderlich, dass einzelne Implementationen dies fehlerhaft implementieren.

5.5 Umgehung mittels manipuliertem Base64

Da historisch bedingt in E-Mails nur ASCII-Zeichen benutzt werden können, wurde mit Base64 eine Kodierung von Binär nach ASCII definiert, bei der jeweils drei beliebige Oktetts in vier ASCII-Zeichen im Bereich „`0-9A-Za-z+/`“ umgewandelt werden:

\x2A								\x7B								\xF3							
0	0	1	0	1	0	1	0	0	1	1	1	1	0	1	1	1	1	1	1	0	0	1	1
K								n								v							
z																							

Werden weniger als drei Zeichen kodiert, so wird für die Kodierung angenommen, dass es sich um „`\x00`“ handelt und im ASCII-Ergebnis entsprechend viele Zeichen am Ende mit „=“ ersetzt. Aus „`\x2a\x7b\xf3`“ wird wie in der Tabelle gezeigt „`Knvz`“, aus „`\x2\x7b`“ hingegen „`Kns=`“, und aus „`\x2`“ wird „`Kg==`“. Daten dürfen nur komplett kodiert werden, womit ein „=“ höchstens am Ende der kodierten Daten auftreten kann.

Während die meisten Implementierungen einfach sämtliche Zeichen ignorieren, die nicht in dem Ziel-Alphabet von Base64 enthalten sind, gibt es Unterschiede beim (nicht erlaubten) Auftreten eines „=“ innerhalb des kodierten Ergebnisses. Manche Implementierungen brechen hier ab und ignorieren den Rest der Daten, andere machen weiter. Bei diesen hängt das Ergebnis auch von der Stellung des „=“ab: ist es nicht am Ende eines kodierten Vierer-Tupels so wird es entweder ignoriert oder auf verschiedene Weise bei der Dekodierung mit einbezogen. Bei unseren Forschungen ist es gelungen auf diese Weise E-Mails mit einem Malware-Attachment zu generieren, die von den meisten Firewall-, Mailfilter- und Antiviren-Produkten nicht gefunden wurde, jedoch von weit verbreiteten Desktop- und Web-basierten Mail-Clients wie gewünscht interpretiert wurden.

5.6 Zusammenfassung der Testergebnisse

Die initiale Analyse üblicher Mailprogramme zeigte, dass diese eine Vielfalt von ungültigem MIME akzeptieren und auf unterschiedliche Weise interpretieren. Dabei waren wir in der Lage, die Analyse von jedem der Systeme auf vielfältige Weise zu umgehen. Zwar wurden verglichen mit HTTP bei MIME nur Tests mit wenigen

Produkten gemacht. Aufgrund der gemachten Erfahrungen aus den HTTP-Tests ist aber davon auszugehen, dass auch die bisher nicht untersuchten kommerziellen Produkte ähnliche Schwächen aufweisen.

Ebenso wurden bei einigen untersuchten Webmailern Wege gefunden, Malware in Mails so einzubetten, dass der eingebaute Virenscanner sie nicht entdeckt aber das Webmail-Interface im Browser diese unmodifiziert zum Download bereitstellt. Die betroffenen Betreiber wurden informiert [29][30].

6. Technologie- und performancebedingte Grenzen der Datennalyse

Die für die Analyse von Mail- und Webverkehr genutzten Systeme lassen sich grob nach passiver oder aktiver Inspektion unterscheiden. Im passiven Fall (Abbildung 5) wird der Datenstrom analysiert und dieser dann entweder blockiert oder überwiegend unmodifiziert weitergeleitet. Ein typischer Vertreter dieser Art sind Intrusion-Detection bzw. Intrusion-Prevention-Systeme (IDS/IPS) wie beispielsweise Snort, aber auch viele Next Generation Firewalls (NGFW) und Unified-Threat-Management-Systeme (UTM) arbeiten nach diesem Prinzip. Diese Art von Analyse schließt auch minimale Modifikationen am Datenstrom ein, wie zum Beispiel das Hinzufügen von Received-Headern am Anfang einer E-Mail oder das Entfernen bzw. Ersetzen unbekannter Kompressionsmethoden im Accept-Encoding-Header eines HTTP-Requests.

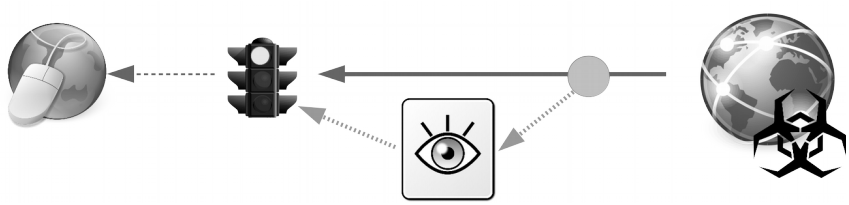

Abbildung 5: Passive Analyse - Firewall entscheidet nur über Blockieren oder Passieren

Bei aktiver Inspektion (Abbildung 6) hingegen kann das Analysesystem die Daten auf Applikationsebene grundlegend verändern. Auf diese Weise ist es möglich, die Daten zu normalisieren. So können zum Beispiel widersprüchliche MIME-Boundarys entfernt werden, um eine eindeutige Interpretation der Daten im zu schützenden System zu erzwingen. Ein typischer Vertreter dieser Analyse sind Application-Level-Gateways (ALG) sowie analysierende Web-Proxys oder Mail-Gateways.

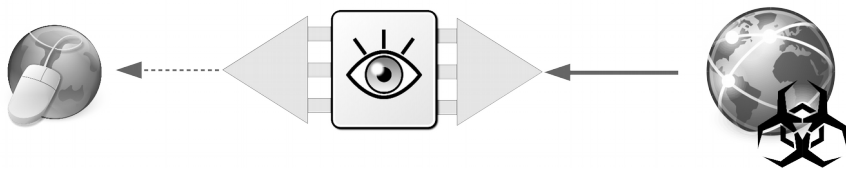

Abbildung 6: Aktive Analyse - Daten werden ein Protokollbestandteile zerlegt, analysiert und neu zusammen gesetzt

Indem eine solide aktive Analyse zuerst den Datenstrom auf Applikationsebene zerlegt, anschließend die extrahierten Daten analysiert und zuletzt den Datenstrom protokollkonform neu generiert, können Interpretationsdifferenzen zwischen Client und Analyse vermieden werden. Allerdings ist der Aufwand dafür deutlich höher als bei der typischen Heuristik- bzw. Signature-basierten passiven Analyse, bei welcher nur bereits vorab als problematisch bekannte Daten blockiert werden.

Andererseits müsste eine solide passive Analyse sämtliche Möglichkeiten der Interpretation untersuchen um das gleiche Schutzniveau wie die aktive Analyse zu bieten. Das aber setzt voraus, dass überhaupt sämtliche Interpretationsvarianten bekannt sind. Zusätzlich würde die Betrachtung aller Varianten und Kombinationen deutlich zu Lasten der Performance gehen. Alternativ müssten sämtliche Daten blockiert werden, bei denen eine alternative Interpretationen denkbar wäre. Nach unseren Erfahrungen ist jedoch ein nicht unbeträchtlicher Anteil des Web- und Mailverkehrs im Internet nicht vollständig standardkonform und erlaubt unterschiedliche Interpretationen, sodass ein derart aggressives Verhalten viele ungefährliche Inhalte blockieren würde [31]. Die Folge davon ist im Allgemeinen, dass der Endnutzer der Firewall die störenden Filter einfach abschaltet und so erneut die Angriffsfläche erhöht. Damit ist eine passive Analyse in der Praxis entweder unsicherer als eine aktive Analyse oder verursacht viele Fehlalarme.

Als Anwender derartiger Analysesysteme sollte man sich daher der technologiebedingten Grenzen bewusst sein und verstehen, dass man nicht gleichzeitig sowohl eine solide Analyse als auch eine hohe Performance bei geringen Kosten haben kann. In dieser Situation bietet unser öffentlicher Testserver Endnutzern und Testlaboren die Möglichkeit, die Qualität einer Firewall zumindest in einem wichtigen Teilaspekt genauer zu untersuchen und so das verbleibende Restrisiko bei der Wahl eines bestimmten Produkts besser einzuschätzen. Aber auch Hersteller nutzen den Testserver bereits zur Qualitätskontrolle ihrer Produkte, als auch um sich gegenüber der Konkurrenz zu profilieren.

Literaturhinweise

[1] Palo Alto Networks, Modern Malware Review, 2013

[2] Olli-Pekka Niemi, Antti Levomäki, Evading Deep Inspection for Fun and Shell, 2013

[3] Jasvir Nagra, GIF/Javascript Polyglots, 2009, http://www.thinkfu.com/blog/gifjavascript-polyglots

[4] Suman Jana and Vitaly Shmatikov, Abusing File Processing in Malware Detectors for Fun and Profit, 2012

[5] Jonas Magazinius, Billy K. Rios, Andrei Sabelfeld, Polyglots: Crossing Origins by Crossing Formats, 2013

[6] Ange Albertini, Funky File Formats, 2014, https://events.ccc.de/congress/2014/Fahrplan/system/attachments/2562/original/Funky_File_Formats.pdf

[7] OWASP, Cache Poisoning, , https://www.owasp.org/index.php/Cache_Poisoning

[8] Jianjun Chen, Jian Jiang, Haixin Duan, Nicholas Weaver, Tao Wan, Vern Paxson, Host of Troubles: Multiple Host Ambiguities in HTTP Implementations, 2016

[9] W3C, The Original HTTP as defined in 1991,

[10] T. Berners-Lee, R. Fielding, H. Frystyk, RFC 1945 - Hypertext Transfer Protocol -- HTTP/1.0, 1996
[11] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, T. Berners-Lee, RFC 2616 - Hypertext Transfer Protocol -- HTTP/1.1, 1999
[12] M. Belshe, R. Peon, M. Thomson, RFC 7540 - Hypertext Transfer Protocol Version 2 (HTTP/2), 2015
[13] EICAR - European Institute for Computer Anti-Virus Research e.V., EICAR: Intended Use, 2006, http://www.eicar.org/86-0-intended-use.html
[14] , http://http-evader.semantic-gap.de/
[15] , Reduce cases where HTTP/0.9 is supported, 2016, https://bugs.chromium.org/p/chromium/issues/detail?id=624462
[16] Steffen Ullrich, HTTP Evasions Explained - Part 1 - Evading Using HTTP 0.9, 2015, http://noxxi.de/research/http-evader-explained-1-http09.html
[17] Steffen Ullrich, Dubious HTTP II - Unusual HTTP Content-Encodings, 2013, http://noxxi.de/research/unusual-http-content-encoding.html
[18] Steffen Ullrich, HTTP Evasions Explained - Part 4 - Doubly Compressed Content, 2015, http://noxxi.de/research/http-evader-explained-4-double-encoding.html
[19] P. Deutsch, RFC 1952 - GZIP file format specification version 4.3, 1996
[20] Steffen Ullrich, HTTP Evasions Explained - Part 5 - GZip Compression, 2015, http://noxxi.de/research/http-evader-explained-5-gzip.html
[21] Steffen Ullrich, HTTP Evasions Explained - Part 6 - Attack of the White-Space, 2015, http://noxxi.de/research/http-evader-explained-6-whitespace.html
[22] Checkpoint, Check Point Response to HTTP Evader, 2015, https://supportcenter.checkpoint.com/supportcenter/portal?eventSubmit_doGoviewsolutiondetails=&solutionid=sk109113
[23] Junper, HTTP Evader: Automate Firewall and IDS Evasion Tests, Analyse Browser Behavior, 2016, https://forums.juniper.net/t5/Security-Incident-Response/HTTP-Evader-Automate-Firewall-and-IDS-Evasion-Tests-Analyse/ba-p/293098
[24] CISCO, Advanced Malware Evasion Techniques HTTP-Evader, 2016, http://blogs.cisco.com/security/advanced-malware-evasion-techniques-http-evader
[25] fire fail, Palo Alto Networks Bypass- HTTP EVADER, 2016, https://www.youtube.com/watch?v=Mo4LUh-5hYo
[26] N. Freed, N. Borenstein, K. Moore, J. Postel, RFC 2045-2048 Multipurpose Internet Mail Extensions, 1996
[27] Steffen Ullrich, Dubious MIME - Conflicting Multipart Boundaries, 2015, http://noxxi.de/research/mime-conflicting-boundary.html
[28] Steffen Ullrich, Dubious MIME - Conflicting Content-Transfer-Encoding Headers, 2014, http://noxxi.de/research/content-transfer-encoding.html
[29] Steffen Ullrich, Bypassing GMX Virus Scanning using Conflicting MIME Boundaries, 2015, http://noxxi.de/research/gmx-mime-conflicting-boundary.html
[30] Steffen Ullrich, Bypassing AOL Mail Virus Scanning with Conflicting Content-Transfer-Encoding Headers, 2015, http://noxxi.de/research/aol-mime-conflicting-cte.html
[31] BlueCoat, [..error message-..] after upgrading the ProxySG to SGOS 6.5.9.2 or later, 2016, http://bluecoat.force.com/knowledgebase/articles/Solution/000030509