|  |  |
| --- | --- |
| Informationstechnologie | |
| Dokumentenart | Arbeitsunterlage |
| Titel | **Apache-Schulung: Lektion 5** |
| Nummer | 5 |
| Autor/-in | Christian Folini |
| Kontaktangaben | +41 79 220 23 76  christian.folini@post.ch |
| Ausgabestelle | IT222 |
| Geltungsbereich | Internet / WWW |
| Klassifizierung | intern |
| Version | X01.00 |
| Ausgabedatum | 6. November 2015 |
| Ersetzt Ausgabe vom | 1. Januar 1970 |

Apache Schulungszentrale: <https://bullnix.post.ch/foswiki/bin/view/BasicServices/ApacheSchulungen>

Quellen:  
<https://github.com/Apache-Labor/labor/blob/master/labor-04/README.md>

<http://www.netnea.com/cms/apache-tutorial-5-zugriffslog-ausbauen/>

## Das Zugriffslog Ausbauen und Auswerten

### **Was machen wir?**

Wir definieren ein stark erweitertes Logformat, um den Verkehr besser überwachen zu können.

### **Warum tun wir das?**

In der gebräuchlichen Konfiguration des Apache Webservers wird ein Logformat eingesetzt, das nur die nötigsten Informationen zu den Zugriffen der verschiedenen Clients mitschreibt. In der Praxis werden oft zusätzliche Informationen benötigt, die sich leicht im Zugriffslog (Accesslog) des Servers notieren lassen.

### **Voraussetzungen**

* Ein Apache Webserver, idealerweise mit einem File-Layout wie bei [Anleitung 1 (Kompilieren eines Apache Servers)](https://www.netnea.com/cms/apache_tutorial_1_apache_compilieren/) erstellt.
* Verständnis der minimalen Konfiguration in [Anleitung 2 (Apache minimal konfigurieren)](https://www.netnea.com/cms/apache_tutorial_2_apache_minimal_konfigurieren/).
* Ein Apache Webserver mit SSL-/TLS-Unterstützung wie in [Anleitung 4 (Konfigurieren eines SSL Servers)](https://www.netnea.com/cms/apache-tutorial-4-ssl-server-konfigurieren)

### **Schritt 1: Logformat Common verstehen**

Das Logformat Common ist ein sehr simples Format, das nurmehr sehr selten eingesetzt wird. Es hat den Vorteil, sehr platzsparend zu sein und kaum überflüssige Informationen zu schreiben.

LogFormat "%h %l %u %t \"%r\" %>s %b" common

...

CustomLog logs/access.log common

Mit der Direktiven LogFormat definieren wir ein Format und geben ihm einen Namen; hier common.

Diesen Namen rufen wir in der Definition des Logfiles mit der Direktiven CustomLog ab. Diese beiden Befehle können wir in der Konfiguration mehrmals einsetzen. Es lassen sich also mehrere Logformate mit mehreren Namenskürzeln nebeneinander definieren und mehrere Logfiles in verschiedenen Formaten schreiben. Es bietet sich etwa an, für verschiedene Dienste auf demselben Server separate Logfiles zu schreiben.

Die einzelnen Elemente des Logformats common lauten wie folgt:

%h bezeichnet den Remote host, normalerweise die IP-Adresse des Clients, der den Request abgesetzt hat. Falls dieser Client aber hinter einem Proxy-Server steht, dann sehen wir hier die IP-Adresse dieses Proxy-Servers. Wenn sich deshalb mehrere Clients den Proxy Server teilen, dann werden sie hier alle denselben Remote host-Eintrag besitzen. Zudem ist es möglich auf unserem Server die IP-Adressen durch einen DNS Reverse Lookup rückzuübersetzen. Falls wir dies konfigurieren (was nicht zu empfehlen ist), dann stünde hier der eruierte Hostname des Clients.

%l stellt den Remote logname dar. Er ist in aller Regel leer und wird durch einen Strich ("-") wiedergegeben. Tatsächlich ginge es dabei um die Identifizierung eines Clients durch einen ident-Zugriff auf den Client. Dies wird von den Clients kaum unterstützt und führt zu grossen Performance-Engpässen, weshalb es sich beim %l um ein Artefakt aus den frühen 1990er Jahren handelt.

%u ist gebräuchlicher und bezeichnet den Usernamen eines authentifizierten Users. Der Name wird durch ein Authentifizierungsmodul gesetzt und bleibt leer (also wiederum "-"), solange ein Zugriff ohne Authentifizierung auf dem Server auskommt.

%t meint die Uhrzeit des Zugriffes. Bei grossen, langsamen Requests meint es die Uhrzeit, in dem Moment in dem der Server die sogenannte Request-Zeile empfangen hat. Da Apache einen Request erst nach Abschluss der Antwort in das Logfile schreibt, kann es vorkommen, dass ein langsamer Request mit einer früheren Uhrzeit mehrere Einträge tiefer als ein kurzer Request steht, der später gestartet ist. Beim Lesen des Logfiles führt das bisweilen zu Verwirrung.

Die Uhrzeit wird standardmässig zwischen rechteckigen Klammern ausgegeben. Es wird normalerweise in der lokalen Zeit inklusive der Abweichung von der Standardzeit geschrieben. Zum Beispiel:

[25/Nov/2014:08:51:22 +0100]

Hier meint es also den 25. November 2014, 8 Uhr 51, 1 Stunde vor der Standardzeit. Das Format der Uhrzeit selbst lässt sich gegebenenfalls anpassen. Dies geschieht nach dem Muster %{format}t, wobei format der Spezifikation von strftime(3) folgt. Wir haben von dieser Möglichkeit in der 2. Anleitung bereis Gebrauch gemacht. Sehen wir es uns aber in einem Beispiel genauer an:

%{[%Y-%m-%d %H:%M:%S %z (%s)]}t

In diesem Beispiel bringen wir das Datum in die Reihenfolge Jahr-Monat-Tag, um eine bessere Sortierbarkeit zu erreichen. Und nach der Abweichung von der Standardzeit fügen wir die Zeit in Sekunden seit dem Start der Unixepoche im Januar 1970 ein. Dies ist ein durch ein Skript leichter les- und interpretierbares Format.

Dieses Beispiel bringt uns Einträgen nach folgendem Muster:

[2014-11-25 09:34:33 +0100 (1322210073)]

Soweit zu %t. Damit kommen wir zu %r und zur Request-Zeile. Hierbei handelt es sich um die erste Zeile des HTTP-Requests, wie er vom Client an den Server gesendet wurde. Streng genommen gehört die Requestzeile nicht in die Gruppe der Request-Header; in aller Regel subsummiert man sie aber zusammen mit letzteren. Wie dem auch sei, auf der Request-Zeile übermittelt der Client dem Server die Identifikation der Ressource, die er verlangt.

Konkret folgt die Zeile diesem Muster:

Methode URI Protokoll

In der Praxis lautet ein einfaches Beispiel also wie folgt:

GET /index.html HTTP/1.1

Es wird also die GET-Methode angewendet. Dann folgt ein Leerschlag, dann der absolute Pfad auf die Ressource auf dem Server. Hier die Index-Datei. Optional kann der Client bekanntlich noch einen Query-String an den Pfad anhängen. Dieser Query-String wird in der Regel mit einem Fragezeichen eingeleitet und bringt verschiedene Parameter-Wert-Paare. Der Query-String wird im Logformat auch wiedergegeben. Schliesslich das Protokoll, das in aller Regel HTTP in der Version 1.1 lautet. Zum Teil wird aber gerade von Agents, also automatisierten Skripten, nach wie vor die Version 1.0 verwendet. Das neue Protokoll HTTP/2 wird in der Request-Zeile des ersten Requests noch nicht vorkommen. Vielmehr findet in HTTP/2 während des Requests ein Update von HTTP/1.1 auf HTTP/2 statt. Der Start folgt also obenstehendem Muster.

Das folgende Format-Element folgt einem etwas anderen Muster: %>s. Dies meint den Status der Antwort, also etwa 200 für einen erfolgreich abgeschlossenen Request. Die spitze Klammer weist darauf hin, dass wir uns für den finalen Status interessieren. Es kann vorkommen, dass eine Anfrage innerhalb des Servers weitergereicht wird. In diesem Fall interessiert uns nicht der Status, der das Weiterreichen ausgelöst hat, sondern den Status der Antwort des finalen internen Requests.

Ein typisches Beispiel wäre ein Aufruf, der auf dem Server zu einem Fehler führt (also Status 500). Wenn dann aber die zugehörige Fehlerseite nicht vorhanden ist, dann ergibt die interne Weiterreichung einen Status 404. Mit der spitzen Klammer erreichen wir, dass in diesem Fall das 404 in das Logfile geschrieben wird. Setzen wir die spitze Klammer in umgekehrter Richtung, dann würde der Status 500 geloggt. Um ganz sicher zu gehen, könnte es angebracht sein, beide Werte zu loggen, also folgender - in der Praxis ebenfalls unüblicher - Eintrag:

%<s %>s

%b ist das letzte Element des Logformats common. Es gibt die Zahl der im Content-Length Response Headers angekündigten Bytes wieder. Bei einer Anfrage an [*http://www.example.com/index.html*](http://www.example.com/index.html) entspricht dieser Wert der Grösse der Datei index.html. Die ebenfalls übertragenen Response-Header werden nicht mitgezählt. Dazu kommt, dass diese Zahl nur eine Ankündigung wiedergibt und keine Garantie darstellt, dass diese Daten auch wirklich übertragen wurden.

### **Schritt 2: Logformat Combined verstehen**

Das am weitesten verbreitete Logformat combined baut auf dem Logformat common auf und erweitert es um zwei Elemente.

LogFormat "%h %l %u %t \"%r\" %>s %b \"%{Referer}i\" \"%{User-agent}i\"" combined

...

CustomLog logs/access.log combined

Das Element "%{Referer}i" bezeichnet den Referrer. Er wird in Anführungszeichen wiedergegeben. Der Referrer meint diejenige Ressource, von der aus ursprünglich der jetzt erfolgte Request ausgelöst wurde. Diese komplizierte Umschreibung lässt sich an einem Beispiel besser illustrieren. Wenn man in einer Suchmaschine einen Link anklickt, um auf [*www.example.com*](http://www.example.com) zu gelangen und dort automatisch auf shop.example.com weitergeleitet wird, dann wird der Logeintrag auf dem Server shop.example.com den Referrer auf die Suchmaschine tragen und nicht den Verweis auf [*www.example.com*](http://www.example.com). Wenn dann aber von shop.example.com eine abhängige CSS-Datei geladen wird, dann geht der Referrer normalerweise auf shop.example.com. Bei alledem ist aber zu beachten, dass der Referrer ein Teil der Anfrage des Clients ist. Der Client ist angehalten, sich an das Protokoll und die Konventionen zu halten, tatsächlich kann er aber beliebige Informationen senden, weshalb man sich in Sicherheitsfragen nicht auf Header wie diesen verlassen darf.

"%{User-agent}i" schliesslich meint den sogenannten User-Agent des Clients, der wiederum in Anführungszeichen gesetzt wird. Auch dies ist wieder ein Wert, der durch den Client kontrolliert wird und auf den wir uns nicht zu sehr verlassen sollten. Mit dem User-Agent ist die Browser-Software des Clients gemeint; normalerweise angereichert um die Version, die Rendering Engine, verschiedene Kompatibilitätsangaben mit anderen Browsern und diverse installierte Plugins. Das führt zu sehr langen User-Agent-Einträgen und kann im Einzelfall so viele Informationen enthalten, dass ein individueller Client sich darüber eindeutig identifizieren lässt, weil er eine besondere Kombination von verschiedenen Zusatzmodulen in bestimmten Versionen besitzt.

### **Schritt 3: Modul Logio aktivieren**

Mit dem combined Format haben wir das am weitesten verbreitete Apache Logformat kennengelernt. Um die alltägliche Arbeit zu erleichtern, reichen die dargestellten Werte aber nicht. Weitere Informationen werden deshalb mit Vorteil im Logfile mitgeschrieben.

Es bietet sich an, auf sämtlichen Servern dasselbe Logformat zu verwenden. Anstatt jetzt also ein, zwei weitere Werte zu propagieren, beschreibt diese Anleitung ein sehr umfassendes Logformat, das sich in der Praxis in verschiedenen Szenarien bewährt hat.

Um das in der Folge beschriebene Logformat konfigurieren zu können, muss aber zunächst das Modul Logio aktiviert werden.

Wenn der Server wie in der Anleitung 1 beschrieben kompiliert wurde, dann ist das Modul bereits vorhanden und muss nur noch in der Liste der zu ladenden Module in der Konfigurationsdatei des Servers ergänzt werden.

LoadModule logio\_module modules/mod\_logio.so

Wir benötigen dieses Modul um zwei Werte mitschreiben zu können. IO-In und IO-Out. Also die totale Zahl der Bytes des HTTP-Requests inklusive Header-Zeilen und die totale Zahl der Bytes in der Antwort, wiederum inklusive Header-Zeilen.

### **Schritt 4: Neues Logformat Extended konfigurieren**

Damit sind wir bereit für ein neues, sehr umfassendes Logformat. Das Format umfasst auch Werte, die der Server mit den bis hierhin definierten Modulen noch nicht kennt. Er wird sie leer lassen, respektive durch einen Strich "-" darstellen. Nur mit dem eben aktivierten Modul Logio klappt das nicht. Wenn wir dessen Werte ansprechen, ohne dass sie vorhanden wären, stürzt der Server ab.

In den folgenden Anleitungen werden wir diese Werte dann nach und nach füllen. Weil es aber wie oben erklärt sinnvoll ist, überall dasselbe Logformat zu verwenden, greifen wir hier mit der nun folgenden Konfiguration bereits etwas vor.

Wir gehen vom Combined-Format aus und erweitern es nach rechts. Dies hat den Vorteil, dass die erweiterten Logfiles in vielen Standard-Tools weiterhin lesbar sind, denn die zusätzlichen Werte werden einfach ignoriert. Ferner ist es sehr leicht, die erweiterten Logfiles in die Basis zurückzuübersetzen um dann eben wieder ein Logformat combined vor sich zu haben.

Wir definieren das Logformat wie folgt:

LogFormat "%h %{GEOIP\_COUNTRY\_CODE}e %u [%{%Y-%m-%d %H:%M:%S}t.%{usec\_frac}t] \"%r\" %>s %b \"%{Referer}i\" \"%{User-Agent}i\" %v %A %p %R %{BALANCER\_WORKER\_ROUTE}e \"%{cookie}n\" %{UNIQUE\_ID}e %{SSL\_PROTOCOL}x %{SSL\_CIPHER}x %I %O %{ratio}n%% %D %{ModSecTimeIn}e %{ApplicationTime}e %{ModSecTimeOut}e %{ModSecAnomalyScoreIn}e %{ModSecAnomalyScoreOut}e" extended

...

CustomLog logs/access.log extended

### **Schritt 5: Neues Logformat Extended verstehen**

Das neue Logformat erweitert die Zugriffsprotokolle um 19 Werte. Das sieht auf den ersten Blick übertrieben aus, tatsächlich hat das aber alles seine Berechtigung und im täglichen Einsatz ist man um jeden dieser Werte froh, gerade wenn man einem Fehler auf der Spur ist.

Schauen wir uns die Werte nacheinander an.

In der Erklärung zum Log-Format common haben wir gesehen, dass der zweite Wert, der Eintrag logname gleich nach der IP-Adresse des Clients ein unbenutztes Artefakt darstellt. Wir ersetzen diese Position im Logfile mit dem Country-Code der IP-Adresse des Clients. Das macht Sinn, weil dieser Country-Code eine IP-Adresse stark charakterisiert. (In vielen Fällen macht es einen grossen Unterschied, ob die Anfrage aus dem Inland oder aus der Südsee stammt). Praktisch ist es nun, sie gleich neben die IP-Adresse zu stellen und der nichtssagenden Nummer ein Mehr an Information zuzugesellen.

Danach das bereits in der Anleitung 2 definierte Zeitformat, das sich am Zeitformat des Error-Logs orientiert und nun mit diesem kongruent ist. Wir bilden die Mikrosekunden auch ab und erhalten so sehr präzise Timing-Informationen. Die nächsten Werte sind bekannt.

%v bezeichnet den kanonischen Servernamen, der den Request bearbeitet hat. Falls wir den Server mittels eines Alias ansprechen, wird hier nicht dieses Alias geschrieben, sondern der eigentliche Name des Servers. In einem Virtual-Host Setup sind die Servernamen der Virtual-Hosts auch kanonisch. Sie werden hier also auftauchen und wir können sie in der Logdatei unterscheiden.

Mit %A folgt die IP-Adresse mit welcher der Server die Anfrage empfangen hat. Dieser Wert hilft uns, die Server auseinander zu halten, wenn mehrere Logfiles zusammengefügt werden oder mehrere Server in dasselbe Logfile schreiben.

Dann beschreibt %p die Portnummer, auf welcher der Request empfangen wurde. Auch dies ist wichtig, um verschiedene Einträge auseinanderhalten zu können, wenn wir verschiedene Logfiles (etwa diejenigen für Port 80 und diejenigen für Port 443) zusammenfügen.

%R gibt den Handler wieder, der die Antwort auf einen Request generiert hat. Dieser Wert kann leer sein (also "-") wenn eine statische Datei ausgeliefert wurde. Oder aber er bezeichnet mit proxy, dass der Request an einen anderen Server weitergeleitet worden ist.

%{BALANCER\_WORKER\_ROUTE}e hat auch mit dem Weitergeben von Anfragen zu tun. Wenn wir zwischen mehreren Zielserver abwechseln, belegt dieser Wert, wohin die Anfrage geschickt wurde.

%X gibt den Status der TCP-Verbindung nach Abschluss der Anfrage wieder. Es sind drei Werte möglich: Die Verbindung ist geschlossen (-), die Verbindung wird mittels Keep-Alive offen gehalten (+) oder aber die Verbindung wurde abgebrochen bevor der Request abgeschlossen werden konnte (X).

Mit "%{cookie}n" folgt ein Wert, der dem User-Tracking dient. Damit können wir einen Client mittels eines Cookies identifizieren und ihn zu einem späteren Zeitpunkt wiedererkennen - sofern er das Cookie immer noch trägt. Wenn wir das Cookie domänenweit setzen, also auf example.com und nicht beschränkt auf [www.example.com](http://www.example.com), dann können wir einem Client sogar über mehrere Hosts hinweg folgen. Im Idealfall wäre dies aufgrund der IP-Adresse des Clients ebenfalls möglich, aber sie kann im Laufe einer Session gewechselt werden und es kann auch sein, dass sich mehrere Clients eine IP-Adresse teilen.

Der Wert %{UNIQUE\_ID}e ist ein sehr hilfreicher Wert. Für jeden Request wird damit auf dem Server eine eindeutige Identifizierung kreiert. Wenn wir den Wert etwa auf einer Fehlerseite ausgeben, dann lässt sich ein Request im Logfile aufgrund eines Screenshots bequem identifizieren - und im Idealfall die gesamte Session aufgrund des User-Tracking-Cookies nachvollziehen.

Nun folgen zwei Werte, die von mod\_ssl bekannt gegeben werden. Das Verschlüsselungsmodul gibt dem Logmodul Werte in einem eigenen Namensraum bekannt, der mit dem Kürzel x bezeichnet wird. In der Dokumentation von mod\_ssl werden die verschiedenen Werte einzeln erklärt. Für den Betrieb eines Servers sind vor allem das verwendete Protokoll und die verwendete Verschlüsselung von Interesse. Diese beiden Werte, mittels %{SSL\_PROTOCOL}x und %{SSL\_CIPHER}x referenziert, helfen uns dabei, einen Überblick über den Einsatz der Verschlüsselung zu erhalten. Früher oder später sind wir soweit, dass wir das TLSv1 Protokoll abschalten werden. Zuerst möchten wir aber sicher sein, dass es in der Praxis keine nennenswerte Rolle mehr spielt. Das Logfile wird uns dabei helfen. Analog der Verschlüsselungsalgorithmus, der uns die tatsächlich verwendeten Ciphers mitteilt und uns hilft eine Aussage zu machen, welche Ciphers nicht mehr länger verwendet werden. Die Informationen sind wichtig. Wenn zum Beispiel Schwächen in einzelnen Protokollversionen oder einzelnen Verschlüsselungsverfahren bekannt werden, dann können wir den Effekt unseres Massnahmen anhand des Logfiles abschätzen. So war etwa die folgende Aussage im Frühjahr 2015 Gold wert: "Das sofortige Abschalten des SSLv3 Protokolls als Reaktion auf die POODLE Schwachstelle wird bei ca. 0.8% der Zugriffe zu einem Fehler führen. Hochgerechnet auf unsere Kundenbasis werden soundsoviele Kunden betroffen sein." Mit diesen Zahlen wurde das Risiko und der Effekt der Massnahme voraussagbar.

Mit %I und %O folgen die beiden Werte, welche durch das Modul Logio definiert werden. Es ist die gesamte Zahl der Bytes im Request und die gesamte Zahl der Bytes in der Response. Wir kennen bereits %b für die Summe der Bytes im Response-Body. %O ist hier etwas genauer und hilft zu erkennen, wenn die Anfrage oder ihre Antwort entsprechende Grössen-Limiten verletzte.

%{ratio}n%% bedeutet die Prozentzahl der Kompression der übermittelten Daten, welche durch die Anwendung des Modules Deflate erreicht werden konnte. Dies ist für den Moment noch ohne Belang, bringt uns in Zukunft aber interessante Performance-Daten.

%D gibt die komplette Dauer des Requests in Mikrosekunden wieder. Gemessen wird vom Erhalt der Request-Zeile bis zum Moment, wenn der letzte Teil der Antwort den Server verlässt.

Wir fahren weiter mit Performance-Daten. Wir werden in Zukunft die Stoppuhr ansetzen und die Anfrage auf dem Weg in den Server hinein, bei der Applikation und während der Verarbeitung der Antwort separat messen. Die entsprechenden Werte werden wir in den Environment Variablen ModSecTimeIn, ApplicationTime sowie ModSecTimeOut ablegen.

Und zu guter Letzt noch zwei weitere Werte, die uns ModSecurity in einer späteren Anleitung zur Verfügung stellen wird, nämlich die Anomalie-Punktezahl der Anfrage und der Antwort. Was es damit auf sich hat, ist für den Moment noch unwichtig. Wichtig ist, dass wir mit diesem stark erweiterten Logformat eine Basis gelegt haben, auf die wir zukünftig aufbauen können, ohne das Logformat nochmals anpassen zu müssen.

### **Schritt 6: Weitere Request- und Response-Header in zusätzlichem Logfile mitschreiben**

Im Arbeitsalltag ist man oft nach bestimmten Requests auf der Suche oder man ist sich nicht sicher, welche Requests einen Fehler verursachen. Da erweist es sich oft als hilfreich, wenn man bestimmte zusätzliche Werte mit ins Logfile schreiben kann. Beliebige Request- und Response-Header sowie Umgebungs-Variabeln lassen sich sehr leicht mitschreiben. Unser Logformat macht davon rege Gebrauch.

Bei den Werten \"%{Referer}i\" sowie \"%{User-Agent}i\" handelt es sich um Request-Header-Felder. Bei der Balancer-Route %{BALANCER\_WORKER\_ROUTE}e haben wir es mit einer Umgebungs-Variablen zu tun. Das Muster wird deutlich: %{Header/Variable}. Request-Header sind der Domäne i zugeordnet. Environment-Variabeln der Domäne e, die Response-Header der Domäne o und die Variablen des SSL-Moduls der Domäne x.

Schreiben wir zu Debug-Zwecken also ein zusätzliches Logfile. Wir benützen nicht mehr die LogFormat-Direktive, sondern definieren das Format zusammen mit dem File auf einer Zeile. Dies ist ein Shortcut, wenn man ein bestimmtes Format nur ein Mal verwenden möchte.

CustomLog logs/access-debug.log "[%{%Y-%m-%d %H:%M:%S}t.%{usec\_frac}t] %{UNIQUE\_ID}e \"%r\" %{Accept}i %{Content-Type}o"

Mit diesem zusätzlichen Logfile sehen wir, welche Wünsche der Client in Bezug auf die Content-Types äusserte und was unser Server tatsächlich lieferte. Normalerweise klappt dieses Zusammenspiel zwischen Client und Server sehr gut. Aber in der Praxis gibt es da schon mal Unstimmigkeiten; da ist ein zusätzliches Logfile dieser Art hilfreich bei der Fehlersuche. Das Resultat könnte dann etwa wie folgt aussehen:

$> cat logs/access-debug.log

2015-09-02 11:58:35.654011 VebITcCoAwcAADRophsAAAAX "GET / HTTP/1.1" \*/\* text/html

2015-09-02 11:58:37.486603 VebIT8CoAwcAADRophwAAAAX "GET /cms/feed/ HTTP/1.1" text/html,application/xhtml+xml,application/xml;q=0.9,\*/\*;q=0.8 text/html

2015-09-02 11:58:39.253209 VebIUMCoAwcAADRoph0AAAAX "GET /cms/2014/04/17/ubuntu-14-04/ HTTP/1.1" \*/\* text/html

2015-09-02 11:58:40.893992 VebIU8CoAwcAADRbdGkAAAAD "GET /cms/2014/05/13/download-softfiles HTTP/1.1" \*/\* text/html

2015-09-02 11:58:43.558478 VebIVcCoAwcAADRbdGoAAAAD "GET /cms/2014/08/25/netcapture-sshargs HTTP/1.1" \*/\* text/html

...

So lassen sich Logfiles in Apache also sehr frei definieren. Interessanter ist aber die Auswertung der Daten. Dazu benötigen wir erst einige Daten.

### **Schritt 7: Ausprobieren und Logdatei füllen**

Konfigurieren wir das erweiterte Zugriffslog im Format extended wie oben beschrieben und beschäftigen wir den Server etwas!

Wir könnten dazu Apache Bench wie in der zweiten Anleitung zwei beschrieben verwenden, aber das würde ein sehr einförmiges Logfile ergeben. Mit den folgenden beiden Einzeilern bringen wir etwas Abwechslung hinein.

$> for N in {1..100}; do curl --silent http://localhost/index.html?n=${N}a >/dev/null; done

$> for N in {1..100}; do PAYLOAD=$(uuid -n $N | xargs); curl --silent --data "payload=$PAYLOAD" http://localhost/index.html?n=${N}b >/dev/null; done

Auf der ersten Zeile setzen wir einfach hundert Requests ab, wobei wir sie im Query-String nummerieren. Auf der zweiten Zeile dann die interessantere Idee: Wieder setzen wir hundert Anfragen ab. Dieses Mal möchten wir aber Daten mit Hilfe eines POST-Requests im Body-Teil der Anfrage mitschicken. Diesen sogenannen Payload generieren wir dynamisch und zwar so, dass er mit jedem Aufruf grösser wird. Die benötigten Daten generieren wir mittels uuidgen. Dabei handelt es sich um einen Befehl, der eine ascii-ID generiert. Aneinandergehängt erhalten wir eine Menge Daten. (Falls es zu einer Fehlermeldung kommt, könnte es sein, dass der Befehl uuidgen nicht vorhanden ist. In diesem Fall wäre das Paket uuid zu installieren).

Die Bearbeitung dieser Zeile dürfte einen Moment dauern. Als Resultat sehen wir folgendes im Logfile:

127.0.0.1 - - [2015-10-03 05:54:09.090117] "GET /index.html?n=1a HTTP/1.1" 200 45 "-" "curl/7.35.0" www.example.com 127.0.0.1 443 - - "-" - TLSv1.2 ECDHE-RSA-AES256-GCM-SHA384 534 1485 -% 446 - - - - -

127.0.0.1 - - [2015-10-03 05:54:09.133625] "GET /index.html?n=2a HTTP/1.1" 200 45 "-" "curl/7.35.0" www.example.com 127.0.0.1 443 - - "-" - TLSv1.2 ECDHE-RSA-AES256-GCM-SHA384 534 1485 -% 436 - - - - -

127.0.0.1 - - [2015-10-03 05:54:09.179561] "GET /index.html?n=3a HTTP/1.1" 200 45 "-" "curl/7.35.0" www.example.com 127.0.0.1 443 - - "-" - TLSv1.2 ECDHE-RSA-AES256-GCM-SHA384 534 1485 -% 411 - - - - -

...

127.0.0.1 - - [2015-10-03 05:55:13.347215] "POST /index.html?n=98b HTTP/1.1" 200 45 "-" "curl/7.35.0" www.example.com 127.0.0.1 443 - - "-" - TLSv1.2 ECDHE-RSA-AES256-GCM-SHA384 4291 1539 -% 866 - - - - -

127.0.0.1 - - [2015-10-03 05:55:13.400345] "POST /index.html?n=99b HTTP/1.1" 200 45 "-" "curl/7.35.0" www.example.com 127.0.0.1 443 - - "-" - TLSv1.2 ECDHE-RSA-AES256-GCM-SHA384 4328 1539 -% 878 - - - - -

127.0.0.1 - - [2015-10-03 05:55:13.453047] "POST /index.html?n=100b HTTP/1.1" 200 45 "-" "curl/7.35.0" www.example.com 127.0.0.1 443 - - "-" - TLSv1.2 ECDHE-RSA-AES256-GCM-SHA384 4366 1539 -% 910 - - - - -

Wie oben vorhergesagt sind noch sehr viele Werte leer oder durch - gekennzeichnet. Aber wir sehen, dass wir den Server [*www.example.com*](http://www.example.com) auf Port 443 angesprochen haben und dass die Grösse des Requests mit jedem POST-Request zunahm, wobei sie zuletzt beinahe 4K, also 4096 Bytes, betrug. Mit diesem einfachen Logfile lassen sich bereits einfache Auswertungen durchführen.

### **Schritt 8: Einfache Auswertungen mit dem Logformat Extended durchführen**

Wer das Beispiel-Logfile genau ansieht, wird erkennen, dass die Dauer der Requests nicht ganz sauber verteilt ist. Es gibt zwei Ausreisser. Wir können das wie folgt identifizieren:

$> egrep -o "\% [0-9]+ " logs/access.log | cut -b3- | tr -d " " | sort -n

Mit diesem Einzeiler schneiden wir den Wert, der die Dauer eines Requests angibt, aus dem Logfile heraus. Wir benützen das Prozentzeichen des Deflate-Wertes als Anker für eine einfache Regular Expression und nehmen die darauf folgende Zahl. egrep bietet sich an, weil wir mit RegEx arbeiten wollen, die Option -o führt dazu, dass nicht die gesamte Zeile, sondern nur der Treffer selbst ausgegeben wird. Das ist sehr hilfreich. Ein Detail, das uns zukünftige Fehler verhindern hilft, ist das Leerzeichen nach dem Pluszeichen. Es nimmt nur diejenigen Werte, die auf die Zahl ein Leerzeichen folgen lassen. Das Problem ist der User-Agent, der in unserem Logformat ja auch vorkommt und der bisweilen auch Prozentzeichen enthält. Wir gehen hier davon aus, dass zwar im User-Agent Prozentzeichen gefolgt von Leerschlag und einer Ganzzahl folgen können. Dass danach aber kein weiterer Leerschlag folgt und diese Kombination nur im hinteren Teil der Logzeile nach dem Prozentzeichen der Deflate-Platzeinsparungen vorkommt. Dann schneiden wir mittels cut so, dass nur das dritte und die folgenden Zeichen ausgegeben werden und schliesslich trennen wir noch mit tr das abschliessende Leerzeichen (siehe Regex) ab. Dann sind wir bereit für das numerische Sortieren. Das liefert folgendes Resultat:

...

925

932

935

939

958

1048

1093

1393

3937

In unserem Beispiel stechen vier Werte mit einer Dauer von über 1000 Mikrosekunden, also mehr als einer Millisekunde heraus, drei davon sind noch im Rahmen, aber einer ist mit 4 Millisekunden klar ein statistischer Ausreisser und steht den anderen Werten gegenüber klar im Abseits.

Wir wissen, dass wir je 100 GET und 100 POST Requests gestellt haben. Aber zählen wir sie übungshalber dennoch einmal aus:

$> egrep -c "\"GET " logs/access.log

Dies sollte 100 GET Requests ergeben:

100

Wir können GET und POST auch einander gegenüber stellen. Wir tun dies folgendermassen:

$> egrep -o '"(GET|POST)' logs/access.log | cut -b2- | sort | uniq -c

Hier filtern wir die GET und die POST Requests anhand der Methode, die auf ein Anführungszeichen folgt, heraus. Dann schneiden wir das Anführungszeichen ab, sortieren und zählen gruppiert aus:

100 GET

100 POST

Soweit zu diesen ersten Fingerübungen. Auf der Basis dieses selbst abgefüllten Logfiles ist das leider noch nicht sehr spanned. Nehmen wir uns also ein richtiges Logfile von einem Produktionsserver vor.

### **Schritt 9: Tiefer gehende Auswertungen auf einem Beispiel-Logfile**

Spannender werden die Auswertungen mit einem richtigen Logfile von einem produktiven Server. Hier ist eines, mit 10'000 Anfragen:

<https://github.com/Apache-Labor/labor/blob/master/labor-04/labor-04-example-access.log>

$> head labor-04-example-access.log

75.249.65.145 US - [2015-09-02 10:42:51.003372] "GET /cms/tina-access-editor-for-download/ HTTP/1.1" 200 7113 "-" "Mozilla/5.0 (compatible; Googlebot/2.1; +http://www.google.com/bot.html)" www.example.com 124.165.3.7 443 redirect-handler - + "-" Vea2i8CoAwcAADevXAgAAAAB TLSv1.2 ECDHE-RSA-AES128-GCM-SHA256 701 12118 -% 88871 803 0 0 0 0

71.180.228.107 US - [2015-09-02 11:14:02.800605] "GET /cms/application\_3\_applikationsserver\_aufsetzen/?q=application\_2\_tina\_minimal\_konfigurieren HTTP/1.1" 200 12962 "-" "Mozilla/5.0 (compatible; Yahoo! Slurp; http://help.yahoo.com/help/us/ysearch/slurp)" www.example.com 124.165.3.7 443 redirect-handler - + "-" Vea92sCoAwcAADRophUAAAAX TLSv1.2 ECDHE-RSA-AES256-GCM-SHA384 700 17946 -% 77038 1669 0 0 0 0

5.45.105.71 DE - [2015-09-02 11:32:39.371240] "GET /cms/feed/ HTTP/1.1" 200 35422 "-" "Tiny Tiny RSS/1.15.3 (http://tt-rss.org/)" www.example.com 124.165.3.7 443 redirect-handler - + "-" VebCN8CoAwcAADRcb14AAAAE TLSv1.2 ECDHE-RSA-AES256-GCM-SHA384 671 40343 -% 144443 791 0 0 0 0

155.80.44.115 IT - [2015-09-02 11:58:35.654011] "GET /robots.txt HTTP/1.0" 404 21023 "-" "Mozilla/5.0 (compatible; MJ12bot/v1.4.5; http://www.majestic12.co.uk/bot.php?+)" www.example.com 124.165.3.7 443 redirect-handler - - "-" VebIS8CoAwcAABx@Xo4AAAAJ TLSv1 AES256-SHA 894 25257 -% 68856 836 0 0 0 0

155.80.44.115 IT - [2015-09-02 11:58:37.486603] "GET /cms/2013/09/23/ HTTP/1.1" 200 22822 "-" "Mozilla/5.0 (compatible; MJ12bot/v1.4.5; http://www.majestic12.co.uk/bot.php?+)" www.example.com 124.165.3.7 443 redirect-handler - + "-" VebITcCoAwcAADRophsAAAAX TLSv1 AES256-SHA 627 23702 -% 75007 805 0 0 0 0

155.80.44.115 IT - [2015-09-02 11:58:39.253209] "GET /cms/2013/09/23/convert-from-splashid-safe-to-keepass-password-safe/ HTTP/1.1" 200 6450 "-" "Mozilla/5.0 (compatible; MJ12bot/v1.4.5; http://www.majestic12.co.uk/bot.php?+)" www.example.com 124.165.3.7 443 redirect-handler - + "-" VebIT8CoAwcAADRophwAAAAX TLSv1 AES256-SHA 485 6900 -% 79458 808 0 0 0 0

155.80.44.115 IT - [2015-09-02 11:58:40.893992] "GET /cms/2013/09/23/convert-from-splashid-safe-to-keepass-password-safe/feed/ HTTP/1.1" 200 463 "-" "Mozilla/5.0 (compatible; MJ12bot/v1.4.5; http://www.majestic12.co.uk/bot.php?+)" www.example.com 124.165.3.7 443 redirect-handler - + "-" VebIUMCoAwcAADRoph0AAAAX TLSv1 AES256-SHA 485 991 -% 25378 798 0 0 0 0

155.80.44.115 IT - [2015-09-02 11:58:43.558478] "GET /cms/2013/10/21/ HTTP/1.1" 200 6171 "-" "Mozilla/5.0 (compatible; MJ12bot/v1.4.5; http://www.majestic12.co.uk/bot.php?+)" www.example.com 124.165.3.7 443 redirect-handler - + "-" VebIU8CoAwcAADRbdGkAAAAD TLSv1 AES256-SHA 611 6702 -% 78686 816 0 0 0 0

155.80.44.115 IT - [2015-09-02 11:58:45.287565] "GET /cms/2013/10/21/nftables-to-replace-iptables-firewall-facility-in-upcoming-linux-kernel/ HTTP/1.1" 200 6492 "-" "Mozilla/5.0 (compatible; MJ12bot/v1.4.5; http://www.majestic12.co.uk/bot.php?+)" www.example.com 124.165.3.7 443 redirect-handler - + "-" VebIVcCoAwcAADRbdGoAAAAD TLSv1 AES256-SHA 501 6932 -% 82579 769 0 0 0 0

155.80.44.115 IT - [2015-09-02 11:58:49.801640] "GET /cms/2013/10/21/nftables-to-replace-iptables-firewall-facility-in-upcoming-linux-kernel/feed/ HTTP/1.1" 200 475 "-" "Mozilla/5.0 (compatible; MJ12bot/v1.4.5; http://www.majestic12.co.uk/bot.php?+)" www.example.com 124.165.3.7 443 redirect-handler - + "-" VebIWcCoAwcAADRbdGsAAAAD TLSv1 AES256-SHA 501 1007 -% 23735 833 0 0 0 0

Schauen wir uns hier mal die Verteilung der GET und POST Requests an:

$> cat labor-04-example-access.log | egrep -o '"(GET|POST)' | cut -b2- | sort | uniq -c

9781 GET

12 POST

Das ist ein eindeutiges Resultat. Sehen wir eigentlich viele Fehler? Also Anfragen, die mit einem HTTP Fehlercode beantwortet wurden?

$> cat labor-04-example-access.log | cut -d\" -f3 | cut -d\ -f2 | sort | uniq -c

9040 200

5 206

447 301

47 304

16 400

3 403

401 404

41 408

Da liegt neben den sechzehn Requests mit der HTTP Antwort "400 Bad Request" ein recht grosser Anteil an 404ern vor ("404 Not Found"). HTTP Status 400 bedeutet ein Protokoll-Fehler. 404 ist bekanntlich eine nicht gefundene Seite. Hier müsste man also mal nach dem Rechten sehen. Aber bevor wir weiter gehen, ein Hinweis auf die Anfrage mittels des Befehls cut. Wir haben die Logzeile mit dem Trennzeichen " unterteilt, das dritte Feld mit dieser Unterteilung extrahiert und dann den Inhalt wieder unterteilt, dieses Mal aber mit dem Leerschlag (man beachte das \-Zeichen) als Trennzeichen und das zweite Feld extrahiert, was nun dem Status entspricht. Danach wurde sortiert und die uniq-Funktion im Zählmodus angewendet. Wir werden sehen, dass diese Art des Zugriffs auf die Daten ein sich wiederholendes Muster ist. Schauen wir das Logfile noch etwas genauer an.

Weiter oben war die Rede von Verschlüsselungsprotokollen und wie ihre Auswertung eine Grundlage für einen Entscheid der adäquaten Reaktion auf die POODLE-Schwachstelle war. Welche Verschlüsselungsprotokolle kommen seither eigentlich auf dem Server in der Praxis vor:

$> cat labor-04-example-access.log | cut -d\" -f9 | cut -d\ -f4 | sort | uniq -c | sort -n

21 -

65 TLSv1.1

1764 TLSv1

8150 TLSv1.2

Es scheint vorzukommen, dass Apache kein Verschlüsselungsprotokoll notiert. Das ist etwas merkwürdig, da es aber ein sehr seltener Fall ist, gehen wir ihm für den Moment nicht nach. Wichtiger sind die Zahlenverhältnisse zwischen den TLS Protokollen. Hier dominiert nach dem Abschalten von SSLv3 das Protokoll TLSv1.2, neben einem substantiellen Anteil TLSv1.0. TLSv1.1 ist vernachlässigbar.

Zum gewünschten Resultat sind wir wieder über eine Reihe von cut-Befehlen gelangt. Eigentlich wäre es doch angebracht, diese Befehle vorzumerken, da wir sie immer wieder brauchen. Das wäre dann eine Alias-Liste wie die folgende:

alias alip='cut -d\ -f1'

alias alcountry='cut -d\ -f2'

alias aluser='cut -d\ -f3'

alias altimestamp='cut -d\ -f4,5 | tr -d "[]"'

alias alrequestline='cut -d\" -f2'

alias almethod='cut -d\" -f2 | cut -d\ -f1 | sed "s/^-$/\*\*NONE\*\*/"'

alias aluri='cut -d\" -f2 | cut -d\ -f2 | sed "s/^-$/\*\*NONE\*\*/"'

alias alprotocol='cut -d\" -f2 | cut -d\ -f3 | sed "s/^-$/\*\*NONE\*\*/"'

alias alstatus='cut -d\" -f3 | cut -d\ -f2'

alias alresponsebodysize='cut -d\" -f3 | cut -d\ -f3'

alias alreferer='cut -d\" -f4 | sed "s/^-$/\*\*NONE\*\*/"'

alias alreferrer='cut -d\" -f4 | sed "s/^-$/\*\*NONE\*\*/"'

alias aluseragent='cut -d\" -f6 | sed "s/^-$/\*\*NONE\*\*/"'

alias alservername='cut -d\" -f7 | cut -d\ -f2'

alias alservername='cut -d\" -f7 | cut -d\ -f2'

alias allocalip='cut -d\" -f7 | cut -d\ -f3'

alias alcanonicalport='cut -d\" -f7 | cut -d\ -f4'

alias alport='cut -d\" -f7 | cut -d\ -f4'

alias alhandler='cut -d\" -f7 | cut -d\ -f5'

alias albalroute='cut -d\" -f7 | cut -d\ -f6'

alias alconnstatus='cut -d\" -f7 | cut -d\ -f7'

alias altrkcookie='cut -d\" -f8'

alias alreqid='cut -d\" -f9 | cut -d\ -f2'

alias alsslprotocol='cut -d\" -f9 | cut -d\ -f3'

alias alsslcipher='cut -d\" -f9 | cut -d\ -f4'

alias alioin='cut -d\" -f9 | cut -d\ -f5'

alias alioout='cut -d\" -f9 | cut -d\ -f6'

alias aldeflateratio='cut -d\" -f9 | cut -d\ -f7 | tr -d %'

alias alduration='cut -d\" -f9 | cut -d\ -f8'

alias aldurationin='cut -d\" -f9 | cut -d\ -f9'

alias aldurationapp='cut -d\" -f9 | cut -d\ -f10'

alias aldurationout='cut -d\" -f9 | cut -d\ -f11'

alias alscorein='cut -d\" -f9 | cut -d\ -f12 | tr "-" "0"'

alias alscoreout='cut -d\" -f9 | cut -d\ -f13 | tr "-" "0"'

alias alscores='cut -d\" -f9 | cut -d\ -f12,13 | tr " " ";" | tr "-" "0"'

Die Aliase beginnen alle mit al. Dies steht für ApacheLog oder AccessLog. Darauf folgt der Feldnahme. Die einzelnen Aliase sind nicht alphabethisch geordnet. Sie folgen vielmehr der Reihenfolge der Felder im Format des Logfiles.

Diese Liste mit Alias-Definitionen befindet sich in der Datei <https://github.com/Apache-Labor/labor/blob/master/bin/.apache-modsec.alias>. Dort liegt sie gemeinsam mit einigen weiteren Aliasen, die wir in späteren Anleitungen definieren werden. Wenn man öfter mit Apache und seinen Logfiles arbeitet, dann bietet es sich an, diese Alias-Definitionen im Heim-Verzeichnis abzulegen und sie beim Einloggen zu laden. Also mittels folgendem Eintrag in der .bashrc-Datei oder über einen verwandten Mechanismus.

test -e ~/.apache-modsec.alias && . ~/.apache-modsec.alias

Wenden wir den neuen Alias das also gleich mal an:

$> cat labor-04-example-access.log | alsslprotocol | sort | uniq -c | sort -n

21 -

65 TLSv1.1

1764 TLSv1

8150 TLSv1.2

Das geht schon etwas leichter. Aber das wiederholte Tippen von sort gefolgt von uniq -c und dann nochmals ein numerisches sort ist mühselig. Da es sich erneut um ein wiederkehrendes Muster handelt, lohnt sich auch hier ein Alias, der sich als sucs abkürzen lässt: ein Zusammenzug der Anfangsbuchstaben und des c von uniq -c.

$> alias sucs='sort | uniq -c | sort -n'

Das erlaubt uns dann:

$> cat labor-04-example-access.log | alsslprotocol | sucs

21 -

65 TLSv1.1

1764 TLSv1

8150 TLSv1.2

Das ist nun ein einfacher Aufruf, den man sich gut merken kann und der leicht zu schreiben ist. Wir blicken nun auf ein Zahlenverhältnis von 1764 zu 8150. Total haben wier hier genau 10'000 Anfragen vor uns; die Prozentwerte sind von Auge abzuleiten. In der Praxis dürften die Logfiles aber kaum so schön aufgehen, wir benötigen also Hilfe beim Ausrechnen der Prozentzahlen.

### **Schritt 10: Auswertungen mit Prozentzahlen und einfache Statistik**

Was uns fehlt ist ein Befehl, der ähnlich wie der Alias sucs funktioniert, aber im selben Durchlauf die Zahlenwerte in Prozentzahlen verwandelt: sucspercent.

$> alias sucspercent='sort | uniq -c | sort -n | $HOME/bin/percent.awk'

Rasche Rechnungen erledigt man in Linux traditionellerweise mit awk. Dafür steht neben der oben gelinkten Alias-Datei, die sucspercent ebenfalls enthält, zusätzlich das awk-Skript <https://github.com/Apache-Labor/labor/blob/master/bin/percent.awk> zur Verfügung, das man idealerweise im Unterverzeichnis bin seines Heimverzeichnisses ablegt. Das obenstehene sucspercent Alias geht denn auch von diesem Setup aus.

$> cat labor-04-example-access.log | alsslprotocol | sucspercent

Entry Count Percent

---------------------------------------------------

- 21 0.21%

TLSv1.1 65 0.65%

TLSv1 1764 17.64%

TLSv1.2 8150 81.50%

---------------------------------------------------

Total 10000 100.00%

Wunderbar. Nun sind wir in der Lage für beliebige, sich wiederholende Werte die Zahlenverhältnisse auszugeben. Wie sieht es denn zum Beispiel mit den verwendeten Veschlüsslungsverfahren aus?

$> cat labor-04-example-access.log | alsslcipher | sucspercent

Entry Count Percent

---------------------------------------------------

DHE-RSA-AES256-SHA256 2 0.02%

ECDHE-RSA-DES-CBC3-SHA 5 0.05%

DES-CBC3-SHA 8 0.08%

- 21 0.21%

DHE-RSA-AES256-GCM-SHA384 43 0.43%

ECDHE-RSA-AES128-SHA256 86 0.86%

ECDHE-RSA-AES128-SHA 102 1.02%

DHE-RSA-AES256-SHA 169 1.69%

AES256-SHA 565 5.65%

ECDHE-RSA-AES256-SHA384 919 9.19%

ECDHE-RSA-AES256-SHA 1008 10.08%

ECDHE-RSA-AES256-GCM-SHA384 1176 11.76%

ECDHE-RSA-AES128-GCM-SHA256 5896 58.96%

---------------------------------------------------

Total 10000 100.00%

Ein guter Überblick auf die Schnelle. Damit können wir für den Moment zufrieden sein. Gibt es etwas zu den HTTP-Protokollversionen zu sagen?

$> cat labor-04-example-access.log | alprotocol | sucspercent

Entry Count Percent

---------------------------------------------------

quit 4 0.04%

\*\*NONE\*\* 41 0.41%

HTTP/1.0 70 0.70%

HTTP/1.1 9885 98.85%

---------------------------------------------------

Total 10000 100.00%

Das veraltete HTTP/1.0 kommt also durchaus noch vor, und bei 45 Anfragen scheint etwas schief gegangen zu sein. Konzentrieren wir uns in der Rechnung auf die erfolgreichen Requests mit einem gültigen Protokoll und sehen wir uns die Prozentzahlen nochmals an:

$> cat labor-04-example-access.log | alprotocol | grep HTTP | sucspercent

Entry Count Percent

---------------------------------------------------

HTTP/1.0 70 0.70%

HTTP/1.1 9885 99.30%

---------------------------------------------------

Total 9955 100.00%

Hier kommt zusätzlich noch ein grep zum Einsatz. Wir können das Muster "Alias-Feldextraktion -> sucs" also noch durch weitere Filteroperationen verfeinern.

Mit den verschiedenen Aliasen für die Extraktion von Werten aus dem Logfile und den beiden Aliasen sucs und sucspercent haben wir uns handliche Werkzeuge zurecht gelegt, um Fragen nach der relativen Häufigkeit von sich wiederholenden Werten einfach und mit demselben Muster von Befehlen beantworten zu können.

Bei den Messwerten, die sich nicht mehr wiederholen, also etwa der Dauer eines Requests, oder der Grösse der Antworten, nützen uns die Prozentzahlen aber wenig. Was wir brauchen ist eine einfache statistische Auswertung. Gefragt sind der Durchschnitt, vielleicht der Median, Informationen zu den Ausreissern und sinnvollerweise die Standardabweichung.

Auch ein solches Skript steht zum Download bereit: https://github.com/Apache-Labor/labor/blob/master/bin/basicstats.awk. Es bietet sich an, dieses Skript ähnlich wie percent.awk im privaten bin-Verzeichnis abzulegen.

$> cat labor-04-example-access.log | alioout | basicstats.awk

Num of values: 10000

Average: 15375

Median: 6646

Min: 0

Max: 340179

Range: 340179

Std deviation: 25913

Mit diesen Zahlen wird der Service rasch plastisch. Mit einer durchschnittlichen Antwortgrösse von 15KB und einem Median von 6.6 KB haben wir einen typischen Webservice vor uns. Der Median bedeutet ja konkret, dass die Hälfte der Antworten kleiner als 6.6 KB waren. Die grösste Antwort kam bei 340 KB zu stehen, die Standard-Abweichung von knapp 26 KB bedeutet, dass die grossen Werte ingesamt selten waren.

Wie sieht es mit der Dauer der Anfragen aus. Haben wir dort ein ähnlich homogenes Bild?

$> cat labor-04-example-access.log | alduration | basicstats.awk

Num of values: 10000

Average: 91306

Median: 2431

Min: 18

Max: 301455050

Range: 301455032

Std deviation: 3023884

Hier ist es zunächst wichtig, sich zu vergegenwärtigen, dass wir es mit Mikrosekunden zu tun haben. Der Median liegt bei 2400 Mikroekunden, das sind gut 2 Millisekunden. Der Durschschnitt ist mit 91 Millisekunden viel grösser, offensichtlich haben wir zahlreiche Ausreisser, welche den Schnitt in die Höhe gezogen haben. Tatsächlich haben wir einen Maximalwert von 301 Sekunden und wenig überraschend eine Standardabweichung von 3 Sekunden. Das Bild ist also weniger homogen und wir haben zumindest einige Requests, die wir untersuchen sollten. Das wird nun aber etwas komplizierter. Das vorgeschlagene Vorgehen ist nur ein mögliches und es steht hier als Vorschlag und als Inspiration für die weitere Arbeit mit dem Logfile:

$> cat labor-04-example-access.log | grep "\"GET " | aluri | cut -d\/ -f1,2,3 | sort | uniq | while read P; do AVG=$(grep "GET $P" labor-04-example-access.log | alduration | basicstats.awk | grep Average | sed 's/.\*: //'); echo "$AVG $P"; done | sort -n

...

97459 /cms/

97840 /cms/application-download-soft

98959 /cms/category

109910 /cms/technical-blog

115564 /cms/content

146096 /cms/feed

146881 /files/application-9-sshots-appl.png

860889 /cms/download-softfiles

Was passiert hier nacheinander? Wir filtern mittels grep nach GET-Requests. Wir ziehen die URI heraus und zerschneiden sie mittels cut. Uns interessieren nur die ersten Abschnitte des Pfades. Wir beschränken uns hier, um eine vernünftige Gruppierung zu erhalten, denn zu viele verschiedene Pfade bringen hier wenig Mehrwert. Die so erhaltene Pfadliste sortieren wir alphabetisch und reduzieren sie mittels uniq. Das ist die Hälfte der Arbeit.

Nun lesen wir die Pfade nacheinander in die Variable P und bauen darüber mit while eine Schleife. Innerhalb der Schleife berechnen wir für den in P abgespeicherten Pfad die Basisstatistiken und filtern die Ausgabe auf den Durschnitt, wobei wir mit sed so filtern, dass die Variable AVG nur die Zahl und nicht auch noch die Bezeichnung Average: enthält. Nun geben wir diesen Durchschnittswert und den Pfadnamen aus. Ende der Schleife. Zu guter letzt sortieren wir alles noch numerisch und erhalten damit eine Übersicht, welche Pfade zu Requests mit längeren Antwortzeiten geführt haben. Offenbar schiesst ein Pfad namens /cms/download-softfiles obenaus. Das Stichwort download lässt dies plausibel erscheinen.

Damit kommen wir zum Abschluss dieser Anleitung. Ziel war es ein erweitertes Logformat einzuführen und die Arbeit mit den Logfiles zu demonstrieren. Dabei kommen wiederkehrend eine Reihe von Aliasen und zwei awk-Skripts zum Einsatz, die sich sehr flexibel hintereinander reihen lassen. Mit diesen Werkzeugen und der nötigen Erfahrung in deren Handhabung ist man in der Lage, rasch auf die in den Logfiles zur Verfügung stehenden Informationen zuzugreifen.