

Theorie

Application Mapping

Marc Ruef

Schwierigkeitsgrad

Bei einem Portscan lassen sich die angebotenen Dienste eines Systems erkennen. Was für eine Funktionalität sich wirklich hinter einem Port verbirgt, muss aber zuerst herausgefunden werden. Durch das Application Mapping kann die Art des Dienstes sowie dessen Protokoll identifiziert werden. In diesem Artikel wird beschrieben, wie sich eine solche Auswertung effizient umsetzen und abwehren lässt.

Das Auswerten der Port-Stati eines Hosts ist ein wichtiger Bestandteil in der netzwerkbasierten Sicherheitsanalyse eines solchen. Der Prozess der Determinierung des angebotenen Dienstes und das durch ihn zur Verfügung gestellten Anwendungsprotokoll wird aber noch immer gerne übergangen. Dabei stellt das Application Mapping eine solide Methode zur Verfügung, um eben diese Informationen frühzeitig einzuholen und dadurch weitere Zugriffe effizient umzusetzen.

Eine wahrlich klassische Disziplin der Computersicherheit ist Portscanning. Durch automatisierte Verbindungsversuche und der Auswertung der Reaktionen des Zielsystems können die Portstati ermittelt werden. Eine Vielzahl an leicht zu bedienenden und über die Jahre gereiften Tools – zum Beispiel nmap oder SuperScan - macht es sehr einfach, diesen Schritt effizient sowie elegant zu durchlaufen [Ruef 1999, Ruef et al. 2002, McClure et al. 2005]. Bei netzwerkbasierten Angriffen und Attacken auf Netzwerkanwendungen hat nämlich eine grundlegende Information enorme Wichtigkeit. Denn offene Ports und die darüber angebotenen Dienste stellen die Grundlage der Zugriffe dar – Sie definieren zu einem erheblichen Mass die Angriffsfläche eines Hosts.

Das Ermitteln der Port-Stati mittels Portscanning ist jedoch oftmals nicht genug, um einen Angriff auf eine Netzwerkapplikation

In diesem Artikel erfahren Sie...

- was Application Mapping ist und wann es seine Anwendung findet;
- auf welche Techniken dabei zurückgegriffen wird;
- wie eine Automatisierung mit amap aussieht;
- wie eine eigene Implementierung eines Tools umgesetzt werden kann;
- wie eine solche Auswertung erschwert werden kann.

Was Sie vorher wissen/können sollten...

- was ein Port ist;
- wie TCP/IP sowie das Client/Server-Prinzip funktioniert.

einzuleiten. Direkt nach dem Portscanning gilt es nämlich direkt zu ermitteln, welches Anwendungsprotokoll überhaupt auf dem Port zum Einsatz kommt. Zwar pflegen nach wie vor viele Administratoren die durch die IANA empfohlenen Standard-Ports für Dienste zu nutzen, jedoch kann als Auditor nicht immer darauf vertraut werden. Webserver finden sich also nicht nur immer zwingend auf dem TCP-Port 80 (http), sondern werden ebenfalls gerne auf alternativen Port-Bereichen wie 81 oder 8000 eingesetzt. Aber auch exotische Varianten wie 12345 und 55555 können vorkommen [Ruef 2004, 2006].

Um das eingesetzte Anwendungsprotokoll ermitteln zu können, wird auf eine Technik zurückgegriffen, die gemeinhin Application Mapping genannt wird [Ruef 2003]. Im Grunde müsste sie Application Protocol Mapping bzw. Protocol Mapping heißen, denn es geht in erster Linie nicht darum, die eigentliche Implementierung einer Netzwerkapplikation zu ermitteln, sondern lediglich das genutzte Anwendungs-Protokoll zu bestimmen [Packetwatch 2004]. Der Name Application Mapping hat sich vor allem durch das Tool amap (Application Mapper) etabliert, das einige der Techniken, die wir im Folgenden besprechen wollen, zu automatisieren in der Lage ist.

Automatische Begrüßung

Grundlage des Application Mappings ist im Client/Server-Prinzip gegeben. Ein Server bietet eine Netzwerkanwendung an, die er auf einen TCP- oder UDP-Port bindet. Der Port befindet sich sodann im LISTENING-Modus (dt. Abhören). Ein Client, der nun diesen Netzwerkdienst nutzen will, verbindet sich auf den jeweiligen Zielport des Servers [Stevens 1994]. Das mitunter populärste Beispiel ist ein Webserver, der sich nach den Empfehlungen der IANA an den TCP-Port 80 (HTTP) gebunden wird. Ein Webbrowser, in diesem Fall der Client, pflegt nun mit einem Aufruf des Ziels samt Zielport die Webdokumente abzurufen. Genauso geschieht es bei Mailserver und –client sowie den vielen anderen Client/Server-basierten Netzwerkdiensten.

Einige textbasierte Netzwerkapplikationen sind mit Hinsicht auf mögliche Interaktivität entwickelt worden. Diese begrüssen beispielsweise nach etablierter Verbindung den Client mit einem Willkommens-Banner, der Informationen über die Implementierung und die weiteren technischen Möglichkeiten enthalten kann. Dafür bekannte klassische Anwendungen sind SMTP (tcp/25) und FTP (tcp/21). Mit einem Telnet-Client lassen sich derlei Verbindungen sehr einfach initiieren und damit die jeweiligen Willkommens-Banner abgreifen. Dies wird in der englischsprachigen Literatur auch Banner-Grabbing genannt [Ruef 1999, Ruef et al. 2002, McClure et al. 2005].

Bei den zeilenbasierten Implementierungen, wie sie sich bei allen populären Betriebssystemen finden lassen, kann dies durch die Eingabe von *telnet <zielhost> <zielport>* umgesetzt werden [Ruef 2004, 2006]. Um beispielsweise eine Verbindung mit dem Server mail.computec.ch auf dem TCP-Port 25 (smtp) umzusetzen, genügt die Eingabe von „telnet mail.computec.ch 25“. In der folgenden Ausgabe dieser exemplarischen Telnet-Sitzung können wir sehr schön sehen, wie nach der Etablierung der Verbindung (auf der TCP-Ebene) der Banner des Mailservers ausgegeben wird. In diesem findet sich der Name „Sendmail“ und die Versionsnummer 8.12.6 des SMTP-Servers.

```
C:\Documents and Settings\mruef>
telnet mail.computec.ch 25
220 mail.computec.ch ESMTP
Sendmail 8.12.6/8.12.6/taifun-1.0;
Wed, 30 Nov 2005 15
:54:54 +0100
```

Das automatische Begrüssen des Clients mit einem Banner ist eine Charakteristika, die einige Netzwerkapplikationen mit sich bringen. Dies ist also ein Indiz auf einige bekannte Implementierungen. So deutet dies auf populäre Dienste wie SMTP oder FTP hin. Andererseits spricht eine automatische Rückantwort eher gegen den Einsatz eines HTTP-Webservers. Dieses kleine Indiz reicht jedoch noch lange nicht aus, um die wirklich zum Einsatz kommende Netzwerkanwendung ausmachen zu können. Dazu müssen noch weitere Details, die wir nun genauer besprechen wollen, betrachtet werden.

Simples Pattern-Matching

Die von ihren Grundzügen einfachste Technik ist im Nutzen von Pattern-Matching innerhalb der Rückmeldungen gegeben. Das englische Wort Pattern-Matching ist der Name einer Technik, bei der in einer Zeichenkette nach einem bestimmten Muster (engl. pattern) gesucht wird (engl. to match). Dies kann sehr simpel mit dem Suchen nach explizit linearen Zeichenketten umgesetzt werden. Ich möchte zur Verdeutlichung dieses elementaren Prinzips des Application Mappings ein umfassendes Beispiel darlegen. Nehmen wir eine Datei mit dem Namen data.txt, die die folgenden Zeilen gespeichert hat (Die Zeilennummern sind absichtlich Inhalt der Datei, um die Beispiele besser illustrieren zu können):

```
01 Dies ist die erste Zeile
02 Dies ist die zweite Zeile
03 Dies ist die dritte Zeile
04 Dies ist das Ende der Datei
```

Mit dem grep-Kommando können wir unter Unix eine Datei auf ihren Inhalt hin untersuchen lassen. Dazu benutzen wir den regulären Syntax, bestehend aus „grep <pattern> <datei>“. In folgendem Beispiel suchen wir in der besagten Datei data.txt nach der Zeichenkette „Dies“. Da das Pattern auf allen vier Zeilen greift, werden allesamt ausgegeben:

```
mruef@debian:~$ grep data.txt „Dies”
01 Dies ist die erste Zeile
```

```
02 Dies ist die zweite Zeile
03 Dies ist die dritte Zeile
04 Dies ist das Ende der Datei
```

Würden wir nun nach einer spezifischeren Zeichenkette suchen, die nur in einigen Zeilen vorkommt, werden entsprechend nur diejenigen als Treffer ausgegeben. Beispiele sind Muster wie *Zeile* (Zeilen 01, 02 und 03) oder *Ende* (nur Zeile 04). Vom Prinzip her ähnlich lässt sich dieser Zugriff auf einem Windows-System mit dem find-Kommando umsetzen, das den gleichen Grund-Synthax zu nutzen pflegt:

```
C:\Documents and Settings\mruef>
find data.txt "Ende"
04 Dies ist das Ende der Datei
```

Wenden wir nun die Technik des Pattern-Matchings auf eine Rückantwort einer etablierten Verbindung an. Folgend die Ausgabe eines Verbindungsversuchs, der auf den TCP-Port 21 des Hosts ftp.computec.ch umgesetzt wurde:

```
C:\Documents and Settings\mruef>
telnet ftp.computec.ch 21
220 ProFTPD 1.2.10 Server
(ftp.computec.ch) [80.74.129.35]
```

Eine einfache Methode des Pattern-Matchings zwecks Application Mapping ist im Heranziehen des Protokoll-Namens gegeben [Ruef 2004, 2006]. Beispielsweise deutet die Zeichenkette „FTP“ darauf hin, dass es sich beim Zielport um einen FTP-Dienst (File Transfer Protocol) handelt. Gleich verhält es sich bei den Abkürzungen SMTP für Mail-Relays und HTTP bei Webserver-Implementierungen. Benutzen wir nun diese drei genannten Akronyme als Pattern unserer Überprüfung, dann sehen wir, dass wir bei FTP eine Übereinstimmung haben und bei SMTP sowie HTTP nicht. Innerhalb der zuvor etablierten Verbindung wird also voraussichtlich FTP (File Transfer Protocol) genutzt. Diese Vermutung deckt sich ebenfalls mit dem genutzten Zielport, da von der IANA für eben solche Kommunikationen der TCP-Port 21 (ftp-control) vorgesehen ist.

Tabelle 1. *Einfache Pattern für das Erkennen der Dienste*

Name	Proto	Port	Trigger
Cisco PIX Firewall SMTP	tcp	25	^220.*********
Dict	tcp	2628	^220 .* dictd
FTP	tcp	21	^220.*\n || ^220.*FTP
ftp-darwin	tcp	21	^220 Inactivity timer
Glftp	tcp	21	^220.*SSH
Hylafax	tcp	4559	^220 .*hylafax
NNTP	tcp	119	^200.*NNTP
Oracle HTTPS	tcp	1526	^220- ora
POP3	tcp	110	^220.*poppassd || 500 Password
SMTP	tcp	25	^220.*\n250 || ^220.*SMTP
TrendMicro InterScan SMTP	tcp	25	^220.*InterScan
vmware-authd	tcp	902	^220 VMware Authentication

False Positives bei Pattern-Matching

Die Technologie des Pattern-Matching wird in vielen Bereichen eingesetzt. Und bei allen kämpft man mit zwei grundlegenden Problemen der Methode:

- `[Datenbank]` es können nur Muster entdeckt werden, die zuvor schon als solche bekannt sind;
- `[Abweichungen]` je nachdem wie strikt ein Muster angelegt wird, übersieht es Übereinstimmungen (false negatives) oder weist solche fälschlicherweise als gegeben aus (false positives).

Das Problem der schon vorgängig bekannten Datenbank ist in Bezug auf das Application Mapping, im Gegensatz zu Antivirus und der elektronischen Einbruchserkennung, eher unwichtigerer Natur. Auf Grund der Spezifikationen und strikten Einhaltung von Standards sind die Vorgaben einfach zu definieren. Zu größeren Problemen kann es im Umgang mit der Zielgenauigkeit von Muster-Ausdrücken (engl. expressions) kommen.

Nehmen wir als Beispiel, dass wir in einer Rückgabe die Zeichenkette „SMTP“ als Hinweis für das Simple Mail Transfer Protocol (RFC 821) erachten. Dies ist normalerweise der Fall, weil sich viele der jeweiligen SMTP-Implementierung mit ihrem Namen und dem gewünschten Anwendungsprotokoll identifizieren. Was passiert aber nun, wenn wir ein SMTP-Mapping auf einen Webserver umsetzen. Irgendwie bringen wir diesen dazu, dass er uns Daten – nämlich ein Web-Dokument – zurückschickt. In diesem ist jedoch nun ebenfalls die Zeichenkette SMTP vorhanden, da es per Zufall genau über Mail-Verbindungen oder den Standard referiert. Das Pattern-Matching wird in diesem Falle fälschlicherweise SMTP als potentiell eingesetztes Anwendungsprotokoll ausweisen.

Gerade wenn das Application Mapping zu großen Teilen automatisiert wurde, kann dies zu verfälschten Resultaten führen. Um dies zu verhindern, sollten komplexere Methoden innerhalb des Pattern-Matchings eingeführt werden. Mittels regulären Ausdrücken lassen sich klare Abgrenzungen umsetzen, so dass nicht mehr nur das Vorhandensein einer linearen Zeichenkette

Tabelle 2. *Die von populären Anwendungsprotokollen unterstützten Kommandos*

	FTP	SMTP	POP3	NNTP
HELP	X	X		X
USER	X		X	
PASS			X	
PORT	X			
STOR	X			
HELO		X		
VRFY		X		
EXPN		X		
RSET		X		
NOOP	X	X		
QUIT	X	X	X	X

zu einer Übereinstimmung führt. (Die Implementierung von amap versucht diesem Problem zusätzlich mit der Analyse der Länge einer Rückgabe Herr zu werden. Nähere Details im entsprechenden Absatz zur Funktionsweise von amap.)

Innerhalb des Application Mappings für SMTP ließe sich die Suche beispielsweise auf die ersten Zeilen der Rückgabe limitieren. Ein SMTP-Header wird nämlich nach etablierter Sitzung als erstes den Begrüßungs-Banner, der die gewünschte Information enthält, zurückliefern (ca. die ersten 100 Bytes). Innerhalb einer HTTP-Verbindung mit einem Webserver, der ein Dokument mit der Zeichenkette „SMTP“ als Inhalt bereitstellt, wird das Vorkommen dessen erst nach dem HTTP-Header gegeben sein (nach ca. 250 Bytes). Eine Komplikation ist also somit schon mal ausgeschlossen oder wenigstens erheblich minimiert.

Es gibt jedoch Situationen, in denen weitaus komplexere reguläre Ausdrücke zum Einsatz kommen müssen. Beispielsweise dann, wenn mit dynamischen Zeichenketten gerechnet werden muss, die jedoch von ihrer Grundstruktur immer in etwa ähnlich ausfallen. Ein gutes Beispiel ist der daytime-Dienst, der sowohl auf TCP als auch UDP angeboten werden kann und dem IANA den Port 13 zugeteilt hat. Daytime funktioniert so, dass der Server nach etablierter Sitzung eine Zeichenkette an den Client zurückliefert, die über die aktuelle Uhrzeit samt Datum informiert. Die Darstellung ist dabei so ausgerichtet, dass sie auch für Menschen lesbar bleibt. Handelt es sich beim Server um ein Unix-System, sieht diese Rückgabe folgendermassen aus (Zeile 02):

```
01 C:\Documents and Settings\
   mruef>telnet 192.168.0.10 13
02 Mon Nov 28 13:42:23 2005
```

In amap, eine der ersten automatisierten Implementierungen für Application Mapping, kommt in der Response-Datei appdefs.resp die folgende Zeile zum Einsatz, die genau mittels regulärem Ausdruck in der Lage ist die besagte Ausgabe festzuhalten:

```
daytime-unix:::26:^[A-Z].*
   [A-Z].* [0-3].
[0-9][0-9]:[0-9][0-9]:
   [0-9][0-9] 200.\rn
```

Eigentlich sehr exakt, aber dennoch nicht perfekt, ist dieses Pattern auf die daytime-Ausgabe in Unix-Umgebungen abgestimmt. Die Optimierungsmöglichkeiten sind gering. In der Ausgabe von daytime-unix wird als erstes der Wochentag als dreistellige Zeichenkette (z.B. Mon für Monday und Tue für Thuesday) ausgegeben. Dies wird zwar vom regulären Ausdruck erkannt, jedoch ist in diesem die Länge dieser Zeichenkette nicht vermerkt. Theoretisch würden also auch erweiterte Ausgaben in der Form von „Monday Nov 28 15:03:23 2005“ erkannt werden. Selbiges gilt für die Darstellung des Monats mittels drei Buchstaben (z.B. Nov für November und Dec für December). Da die Unix-Implementierungen in der Hinsicht jedoch sehr standardisiert ausfallen, wird es in diesen Bereichen sehr selten zu fehlerhaften Erkennungen führen. Eine Einschränkung von größerer Problematik ist jedoch, dass die Jahreszahl

Tabelle 3. *Das typische Timeout-Verhalten verschiedener Dienste*

Timeout-Verhalten	Beispiele
Kein Timeout	Telnet, SSH, NNTP, Echo, Discard, Chargen
Verbindungsabbruch nach Timeout	FTP, netbios-ssn (ca. 30 Sek)
Verbindungsabbruch nach irgendeinem Befehl(en)	HTTP, HTTPS/SSL, Finger, Time
Verbindungsabbruch nach letztem Befehl(en)	HTTP, HTTPS/SSL, Finger, Time
Verbindungsabbruch nach falschem Befehl(en)	einige SMTP-Implementierungen, einige Terminal-Authentisierungen (z.B. Telnet oder SSH), Proxies mit Access Lists
NOOP-Kommando nach Timeout	FTP

auf den Zeitraum von 2000 bis 2009 limitiert ist. Ein zeitlich falsch konfiguriertes System (z.B. im Jahr 1984) würde mit diesem Ausdruck nicht richtig überprüft und erkannt werden können.

Interessant an dieser spezifischen Analyse-Routine ist zudem, dass allein schon anhand der Rückantworten unscheinbarer Dienste wie daytime, in denen keine direkten Hinweise auf das eingesetzte Betriebssystem gegeben sind, Rückschlüsse auf eben dieses gezogen werden können. Wiederum die Fingerprint-Datenbank von amap führt fünf verschiedene Einträge für daytime, wobei zwei explizit auf Unix (Zeilen 01 und 03) und einer auf Windows (Zeile 02) appliziert werden kann. Die anderen sind generisch gehalten.

```
01 daytime-unix:::26:^[A-Z].* [A-Z]
   [0-3].* [0-9][0-9]:[0-9][0-9]:
   [0-9][0-9] 200.\rn
02 daytime-windows:::26-50:^
   [A-Z][a-z]+, [A-Z][a-z]+ [0-9]+,
   200[0-9] [0-9]+:[0-9]+:[0-9]+
   \x0a\x00
```

Tabelle 4. *Die komplexen Pattern von amap*

Name	Proto	Länge	Pattern
Dante	tcp	2	\x05\x02
Daytime-unix	tcp/udp	26	^[A-Z].* [A-Z].* [0-3].* [0-9][0-9]
Daytime-windows	tcp/udp	26-50	^[A-Z][a-z]+, [A-Z][a-z]+ [0-9]+, 200[0-9] [0-9]+
Daytime-unix	tcp/udp	20-36	^[A-Z][a-z]+ [A-Z][a-z]+ [0-9][0-9] [0-9]+
Daytime	tcp/udp	25-30	^[0-9][0-9] [A-Z][A-Z][A-Z] 200[0-9] [0-9][0-9]
Daytime	tcp/udp	26-45	^[A-Z][a-z][a-z]*, [A-Z][a-z][a-z]* [0-9]+, 200
Dhcp3d-isc	tcp	8	^\x00\x00\x00\x64\x00\x00\x00\ x18
Dns-pdnsd	tcp/udp	2	^\x00\x0c
Finger	tcp	1	\x66
http-compaqin-sightmanager	tcp	6	^HTTP/1
http-iis	tcp	34	^ Bad Request .Invalid URL.
Mailhurdle	udp	10	\x00\x08\x06\x9f\x7a\x06\x00\ x00\x00\x00
Mldonkey	tcp	1	\x31
ms-distribution-transport	tcp(/udp)	6	^..\x0a\x00
Ms-dtc	tcp/udp	3	..\n
Netmeeting	tcp	4	^\x03\x00\x00\x11
NTP	udp	48	^....\x00\x00..\x00\x00
QOTD	tcp/udp	5-1000	^"[A-Z].* .* .*[!?.]"\rn[A-Za-z].*\rn
SNMP	tcp	3	\x41\x01\x02
Spamd	tcp	1	\x32
SSL	tcp	1	\n
Time	tcp/udp	4	^\xc[2-5]

```
03 daytime-unix:::20-36:^
   [A-Z][a-z]+ [A-Z][a-z]+
   [0-9 ][0-9] [0-9]+:[0-9]+:
   [0-9]+ 200[0-9]\x0d\x0a
04 daytime:::25-30:^
   [0-9][0-9] [A-Z][A-Z][A-Z]
   200[0-9] [0-9][0-9]:
   [0-9][0-9]:[0-9][0-9] .*
05 daytime:::26-45:^
   [A-Z][a-z][a-z]*,
   [A-Z][a-z][a-z]* [0-9]+, 200
```

Dreistellige Statuscodes

Viele Netzwerkanwendungen, die zum Zweck der direkten Benutzer-Interaktivität entwickelt wurden, greifen auf das Prinzip der Statuscodes zurück. Dies funktioniert so, dass jede Ein- und Ausgabe zuerst mit einem nummerischen Statuscode, in der Regel ist dieser dreistellig, versehen ist. Beispielsweise nutzen allesamt FTP, SMTP sowie andere klassische Protokolle den Wert 220 für die Bestätigung der erfolgreichen Verarbeitung einer Anfrage. Die Hersteller halten sich dabei an das Prinzip, dass die erste Ziffer des dreistelligen Statuscodes die Kategorie bestimmt (z.B. Erfolgreich), die zweite die Meldungsart (z.B. Ressource gefunden) und die letzte den differentierten Beschrieb (z.B. angeforderte Ressource zurückgeschickt). Der Grund für den Einsatz von Integer-Werten als Statuscodes ist jener, dass sich diese einfacher durch Clients verarbeiten lassen. Diese müssen lediglich die ersten drei Bytes einer Rückgabe einlesen, um den Stand der Dinge erkennen zu können (andernfalls wäre eine komplexe Wort-Erkennung erforderlich).

Danach folgt auf der gleichen Zeile, getrennt mit einem Leerzeichen, der ausgeschriebene Text des Status. Dieser wird oftmals deswegen angeboten, wenn der Dienst ohne dazugehörigen Client genutzt werden will, und auf eine automatisierte Interpretation des dreistelligen Statuscodes verzichtet wird (z.B. beim Nutzen einer interaktiven Telnet-Sitzung). Beispielsweise könnte also in einer erfolgreichen

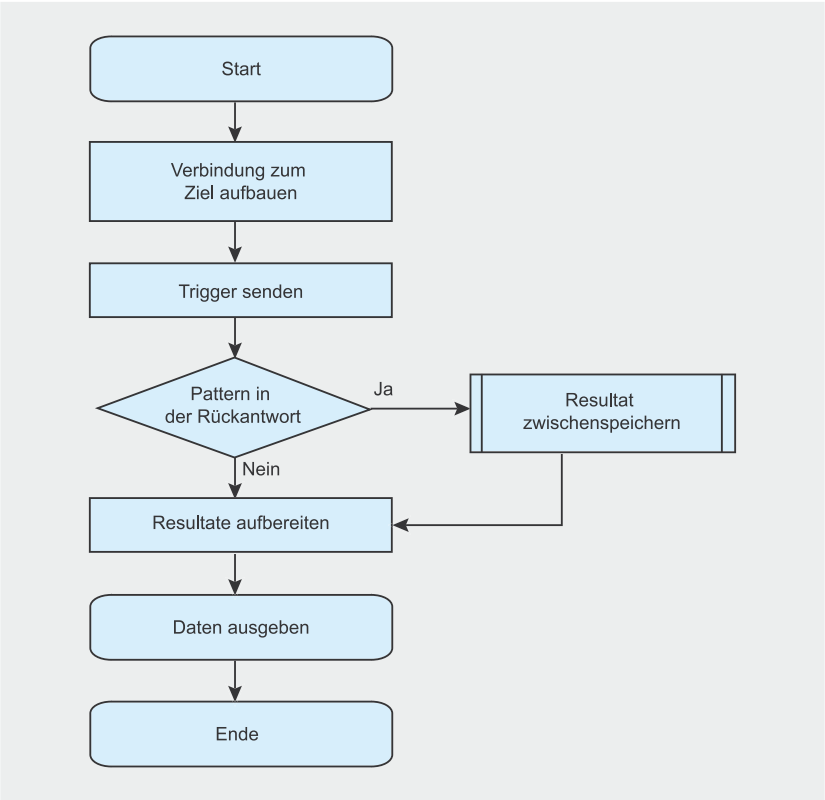


Abbildung 1. *Ablauf eines Application Mappings mittels Reiz/Reaktion*

interaktiven Kommunikation mit FTP die erste Zeile „220 Found“ lauten. Andere Statuscodes verweisen auf server- oder client-seitige Fehler bzw. fehlende Implementierungen. Betrachten wir ein kleines Beispiel einer FTP-Sitzung:

```
01 C:\Documents and Settings\mruef>
   ftp 192.168.0.10
02 Connected to 192.168.0.10.
03 220 debian.computec.ch FTP server
   (Version 6.4/OpenBSD/
   Linux-ftpd-0.17) ready.
04 User (192.168.0.10:(none)): mruef
05 331 Password required for mruef.
06 Password: xxxxxxxx
07 230- Linux debian 2.6.12-1-386 #
   1 Tue Sep 27 11:02:18 JST
   2005 i686 GNU/Linux
08 230 User maru logged in.
09 ftp>
```

Als erstes stellen wir von unserem Windows-Rechner eine FTP-Verbindung zum Zielsystem mit der IP-Adresse 192.168.0.10 her (Zeile 01). Unser FTP-Client zeigt uns an, dass ein Verbindungsaufbau erfolgreich stattgefunden hat (Zeile 02). In der ersten Rückgabe des FTP-Servers (Zeile 03), wird der Willkommens-Banner angezeigt. Diesem vorangesetzt ist der dreistellige Statuscode 220, der auf einen erfolgreichen Zugriff hindeutet. Nach der Eingabe des Benutzernamens mruef (Zeile 04), wird zur Eingabe des dazugehörigen Passworts aufgefordert (Zeile 05). Hier sehen wir den Statuscode 331, der für die Erforderlichkeit einer Authentisierung steht.

Gerade weil verschiedene Systeme auf das Prinzip der (dreistelligen) Statuscodes zurückgreifen, kann dieses Verhalten für das Application Mapping von Nutzen sein. Derlei semantische Gegebenheiten deuten typischerweise auf interaktive ASCII NVT Protokolle wie FTP, SMTP oder POP3 hin [Stevens 1994, Ruef et al. 2002]. Tabelle 1 zeigt bekannte Protokolle der Anwendungsschicht, die auf das klassische Statuscode-Prinzip zurückgreifen. In der letzten Spalte wird der Trigger angegeben, anhand dessen die jeweilige Protokoll-Art identifiziert werden kann. Diese sind hierbei so simpel wie möglich gehalten, um das Verständnis für die Sache festigen zu können. Es besteht durchaus die Möglichkeit, dass sich diese noch optimieren und damit Falschmeldungen minimieren lassen.

Zuerst eine formelle Anfrage stellen

Es gibt einige Dienste, die nicht so freizügig sind und auf das umfängliche Begrüßen des Clients nach der Etablierung der Verbindung verzichten. Derartige Implementierungen könnte man sinngemäß also auch als unhöflich oder unfreundlich – jedenfalls in Bezug auf das Verhalten eines Gastgebers - bezeichnen. Eine solche Kommunikation sieht meistens so aus, dass nach etablierter Verbindung der Bildschirm auf Seiten des Clients erstmal leer bleibt. Der Server (die Netzwerkapplikation) wartet hierbei auf erste Anfragen, bevor er selbst reagiert. Erhält er innerhalb einiger Sekunden oder Minuten keine Anfrage, kann dies mit einem Timeout zu einem automatisierten Verbindungsabbruch führen, womit der unnötige Ressourcen-Verbrauch persistenter Verbindungen verhindert wird.

Die wohl bekannteste Implementierung eines Anwendungsprotokolls, das zuerst eine formelle Anfrage erwartet, ist in HTTP (Hyper Text Trabsfer Protocol) gegeben. Schauen wir uns dies an einem Beispiel an, bei dem wir mit unserem Telnet-Client eine manuelle Verbindung zum Host mit der IP-Adresse 192.168.0.10 auf dem TCP-Zielport 80 aufbauen:

```
01 C:\Documents and Settings\mruef>
   telnet 192.168.0.1 80
02 HEAD / HTTP/1.0
03
04 HTTP/1.1 200 OK
05 Date: Mon, 28 Nov
   2005 16:39:07 GMT
06 Server: Apache/
   1.3.34 (Debian)
07 Connection: close
08 Content-Type: text/
   html; charset=iso-8859-1
09
10
11
12 Connection to host lost.
```

Nachdem die Verbindung mittels Telnet initiiert wurde (Zeile 01), muss der Client eine Anfrage absetzen. Da wir in diesem Beispiel schon wussten, dass es sich um die Implementierung eines HTTP-Servers handelt, wurde sogleich auf den richtigen Syntax zurückgegriffen. Mit der Anfrage von „HEAD / HTTP/1.0“ (abgeschlossen mit zwei Newline-Zeichen; Zeilen 02 und 03) konnte die reguläre Rückantwort (Zeilen 04 bis 11) provoziert werden. In dieser können wir die HTTP-typischen Merkmale (z.B. Beginnen der Rückantwort mit der Zeichenkette HTTP und eine Zeile, die mit „Server: “ anfängt) ausgemacht werden. Ohne das Absetzen unseres initialen Reizes mittels HTTP HEAD-Anfrage wäre dies in der beschriebenen Form nicht möglich gewesen.

Im Normalfall wird es während des Application Mappings jedoch nicht so sein, dass einem der auf dem Zielport angebotene Dienst schon vorgängig bekannt ist. So können wir nicht beim ersten Versuch die in diesem Beispiel genutzte und konforme Anfrage nutzen. In solchen Fällen lohnt sich als initiale Handlung das mehrmalige Drücken der Eingabetaste. In den meisten Fällen wird dies eine Fehlermeldung provozieren (z.B. „Not supported“ oder „Syntax unknown“), die dann ebenfalls eine Vielzahl für die Protokoll-Analyse nützlichen Informationen bereithält. Statistisch gesehen kommen aber auf Grund ihrer Einfachheit der Implementierung und Nutzung sehr viele HTTP-Implementierungen zum Tragen, selbst wenn es sich um vermeintlich proprietäre Entwicklungen handelt. Damit können also auch HTTP-typische Eingaben wie „GET / HTTP/1.0“ gute Resultate einbringen.

Listing 1 *Eine einfache Implementierung eines Application Mappings*

```
01  #!/bin/sh
02
03  echo "pmap v1.0"
04
05  if [ $# == 2 ]; then
06      echo "Starting protocol mapping ..."
07      RESPONSE=`echo -e "QUIT\n" | nc -vv -w 3 $1 $2`
08
09      echo -n "Testing for trigger ftp ... "
10      MATCH=`echo "$RESPONSE" | grep ftp`
11      if [ "$MATCH" ]; then
12          echo "Found!"
13      else
14          echo "Not found."
15      fi
16  else
17      echo "Please use the following syntax:"
18      echo "pmap.sh <dhost> <dport>"
19  fi
```

Tabelle 5. *Gewichtung der Indizien während der Erkennung*

Name	Gewicht	Beschreibung
Portnummer	5 %	Falls auf dem getesteten Port ein Dienst angeboten wird, für den IANA diesen Port als Standard empfiehlt.
Transport-protokoll	5 %	Falls jenes Transportprotokoll genutzt wird, das für die Standard-Implementierungen zum Tragen kommt (z.B. TCP für HTTP und UDP für TFTP).
Begrü-ßungs-Ver-halten	5 %	Falls das vermeintliche Transportprotokoll eine automatische Banner-Begrüßung umsetzt oder nicht.
Timeout-Verhalten	5 %	Falls die Anwendung nach einer gewissen Zeit zu einem automatischen Timeout führt.
Rückantwort ohne Reiz	35 %	Darstellung des automatisch nach Etablierung der Verbindung zurückgeschickten Willkommen-Banners.
Rückantwort mit Reiz	45 %	Darstellung der durch einen Reiz provozierten Rückantwort.

Reize und Reaktionen

Wir haben gesehen, dass einige auf Interaktivität ausgelegte Applikationen den Client unmittelbar nach der Etablierung der Verbindung charakteristisch begrüßen. Ebenso haben wir aber besprochen, dass es einige eher unhöfliche Dienste wie HTTP gibt, die sich zuerst um die Herausgabe von Daten bitten lassen. Genau letzteres wollen wir nun im Detail behandeln. Und zwar nicht mal unbedingt im Zusammenhang mit den besagten unfreundlichen Implementierungen. Wir wollen viel eher diskutieren, wie das Application Mapping noch effizienter gestaltet werden kann, indem sich möglichst eindeutige Reaktionen provozieren lassen.

Eine klassische Möglichkeit, die nicht unbedingt für ihre Performance bekannt ist, ist das Nutzen einer vermeintlichen Hilfe-Funktion. Vor allem interaktive Dienste wie FTP oder SMTP bieten in der Regel das HELP-Kommando an, mit dem eine Auflistung der unterstützten Befehle ausgegeben wird. Anhand dieser HELP-Ausgabe bzw. den darin vermerkten Kommandos lassen sich oftmals sehr konkret das genutzte

Anwendungsprotokoll eruieren. Tabelle 2 gibt Aufschluss darüber, in welchen populären Anwendungsprotokollen welche Kommandos am ehesten anzutreffen sind. Diese Auflistung ist weder aus Protokoll- noch aus Kommando-Sicht komplett.

Alleine anhand der Fehlermeldung, ob nun ein Kommando existiert oder nicht, lassen sich gewisse Ableitungen auf das eingesetzte Protokoll machen. Vorsicht ist jedoch in der Hinsicht geboten, da nicht zwingend alle Server-Implementierungen auch alle Kommandos anbieten müssen. Beispielsweise verzichten viele moderne SMTP-Server aus Sicherheitsgründen darauf, klassische Kommandos wie VRFY, EXPN, DEBUG oder gar HELP einzusetzen. Fehlermeldungen, dass ein derartiger Zugriff administrativ verweigert wird oder das Kommando gar nicht erst implementiert wurde, sind dann keine Seltenheit. Dies kann natürlich, wenn man sich nun zu sehr auf eine derartige Charakteristika verlässt, zu fehlerhaften Resultaten führen.

Bieten mehrere Protokolle der Anwendungsschicht die gleichen Kommandos an, muss eine differenzierte Analyse vorgenommen werden. Dies kann einerseits damit geschehen, dass man die jeweiligen Implementierungen auf den Umgang mit Parametern überprüft. Ebenso können die regulären Rückantworten je nach Dienst anders ausfallen. Viele FTP-Server erlauben beispielsweise einen Parameter für das HELP-Kommando. Wird nun „HELP PORT“ eingegeben, folgt die spezifische Hilfe-Information für den FTP-Befehl PORT. SMTP verhält sich da oftmals ähnlich.

Das Timeout-Verhalten

Wir haben zuvor gesehen, dass das grundlegende Begrüßungs-Verhalten eine Charakteristika für eine Netzwerkapplikation und das durch sie bereitgestellte Anwendungsprotokoll darstellen kann. Jenachdem, ob und inwiefern sie den Client mit einem automatisierten Willkommens-Banner begrüßt, kann dies auf eine interaktive (z.B. SMTP) oder eher weniger interaktive Applikation (z.B. HTTP) hindeuten.

Eine ähnliche Charakteristika, die sich ebenfalls als Indiz für die Bestimmung eines Dienstes einsetzen lässt, ist das sogenannte Timeout-Verhalten. Ein Timeout tritt normalerweise dann innerhalb einer Applikation auf, wenn über längere Zeit kein regulärer Datenaustausch stattgefunden hat. Die Zeitdauer für das Einsetzen von Timeouts kann sich vom niedrigen Millisekunden-Bereich bis hin zu Stunden erstrecken. Typischerweise werden aber gerade bei semi-interaktiven Applikationen, die auf Sitzungen ausgelegt sind, im Bereich von mehreren Sekunden (z.B. 10 Sekunden) oder wenigen Minuten (z.B. 2 Minuten) gearbeitet.

Es gibt verschiedene Stati, die innerhalb einer Timeout-Analyse aufgezeigt werden können. Dabei gilt es dringlichst anzumerken, dass sich dieses nicht zwingend am Anwendungsprotokoll selbst orientiert, sondern in höchstem Masse von der jeweiligen Implementierung und ihrer Konfiguration abhängig sein kann. Viele Dienste lassen nämlich komfortabel oder wenigstens in einer Konfigurations-Datei versteckt das Bestimmen der Timeout-Werte zu. Dabei können wir grob zwischen den folgenden sechs Stati unterscheiden:

Die innerhalb des Application Mappings undankbarste Variante ist dann gegeben, wenn kein oder ein sehr langes Timeout gesetzt wird. Genau wie in der Herleitung eines logischen Beweises ist das Fehlen eines Zustands schwieriger zu bestimmen, weder die Existenz eines solchen. Die Verifikation des Status eines fehlenden Timeouts ist sehr schwierig und auf Grund seiner Natur entsprechend langwierig. Dies ist eine der Ursachen, warum von einer derartigen Implementierung abgesehen werden muss. Und falls eine solche doch eingesetzt werden will, müsste für den praktischen Betrieb ein kurzer Zeitwert für die Verifizierung eingesetzt werden (z.B. maximal 120 Sekunden). Mögliche Beispiele für Anwendungen, die auf ein Timeout weitestgehend verzichten sind klassische Dienste wie Echo, Discard und Chargen. Aber auch Terminal-Emulationen wie Telnet oder SSH werden selten automatisch gekappt.

Der übliche automatische Verbindungsabbruch geschieht in der Regel nach einem zeitlich begrenzten Timeout. Ein gutes Beispiel hierzu sind FTP-Kommunikationen, die nach einer gewissen Zeitspanne (und dem mehrmaligen Nutzen des NOOP-Kommandos) die Verbindung serverseitig schließen.

Exotischer ist der semit-automatische Verbindungsabbruch nach der Eingabe eines Befehls und der Generierung der entsprechenden Rückantwort. Dies geschieht meistens in sehr sitzungsorientierten Verbindungen, bei denen möglichst wenige persistente Kommunikationskanäle offen gehalten werden sollen. Populärstes Beispiel ist HTTP, bei dem standardmässig nach der Anfrage des Clients und der Rückantwort des Servers die Verbindung von letzterem automatisch geschlossen wird.

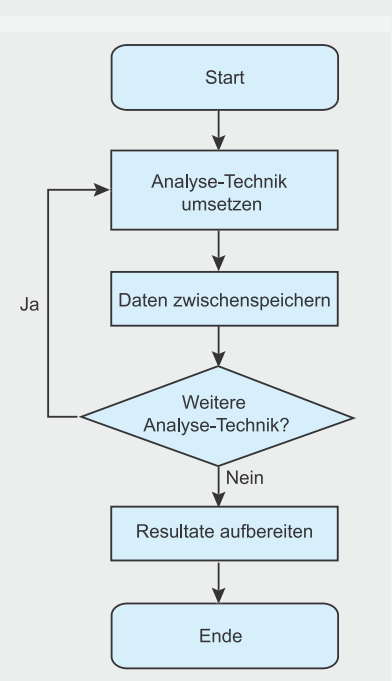


Abbildung 2. *Ablauf eines intelligenten Application Mappings*

Andere Implementierungen gleicher Art sind Daytime, Time und QOTD.

Einige auf Interaktivität hin entwickelte Dienste schliessen automatisch die Verbindung, falls vom Client gewisse Anforderungen nicht erfüllt werden können. Beispielsweise beenden viele Server den Verbindungsaufbau einer Terminal-Emulations Sitzung mit Telnet oder SSH, wenn das Passwort mehrmals falsch eingegeben wurde. Oder so mancher SMTP-Server beendet abrupt die Kommunikation, falls der Client einige Male hintereinander ein nicht implementiertes Kommando nutzen will. Dieses Verhalten ist aber in der Tat ebenfalls sehr vom Programmdesign der Applikation abhängig und bezieht sich damit nicht direkt auf das Anwendungsprotokoll ansich.

Eine letzte Verhaltens-Art in Bezug auf Timeouts ist im Nutzen von automatisch generierten NOOP-Kommandos gegeben. NOOP ist dabei eine Abkürzung für das englische „No Operation“. Dieses Kommando wird absichtlich periodisch eingesetzt, um eben den automatisierten Verbindungsabbruch eines Timeouts zu verhindern. Client oder Server halten so quasi die Leitung warm und damit den Kommunikationskanal aufrecht, in dem sie ab und an wenigstens kleine Daten austauschen. Oftmals nutzen interaktive Applikationen, wie zum Beispiel FTP, eine derartige Operation. Erst nach mehrmaligem Nutzen des NOOP-Kommandos über mehrere Minuten hinweg, wird zu einem serverseitigen und automatisierten Verbindungsabbruch gegriffen.

Die Analyse des Timeout-Verhaltens ist nicht gerade einfach, lässt sich dieses oftmals nur mit einem erhöhten Zeitaufwand, vielen differenzierten Tests oder gar nicht exakt bestimmen. Aus diesem Grund hat man Beispielsweise bei amap von der Implementierung einer solchen Charakteristika-Analyse abgesehen. Es ist auch nicht damit zu rechnen, dass eine derartige Erweiterung Einzug halten wird, da der Aufwand/Ertrag-Faktor einer derartigen Auswertung schlichtweg zu schlecht ausfällt.

Automatisierung mittels amap

Eine der ersten automatisierten Implementierungen zur Umsetzung des Application Mappings ist die Software amap. Der Name dieser geschah in Anlehnung an das populäre Scanning- und Auswertungs-Utility nmap von Fyodor. Amap unterliegt der General Public License (GPL) und wurde für UNIX/Linux von Van Hauser und DJ RevMoon der deutschen Gruppe THC (The Hackers Choice, *http://www.thc.org*) entwickelt.

Die grundlegende Bedienung der Software ähnelt der von nmap. Das erste Argument ist auch hier der Name oder die IP-Adresse des Zielsystems. Als zweiter Parameter muss nun aber zwingend die Angabe einer Portnummer im regulären Bereich von 1 bis 65'535 erfolgen. Wird auf das Heranziehen der Option –u verzichtet, findet standardmässig der Zugriff auf den TCP-Port statt. Andernfalls wird UDP als Transportprotokoll eingesetzt. Sollen lediglich dedizierte Anwendungsprotokolle überprüft werden, lohnt sich das Miteinbeziehen des Parameters *-p <protokoll>*, der die Spezifikation einzelner Protokolle zulässt (z.B. ftp oder smtp).

Besonders für Entwickler, professionelle Penetration Tester und während des Debuggings interessant ist der Schalter –d. Mit diesem kann amap angewiesen werden, dass die Rückgaben des Zielports in bester Debug-Manier für jeden Trigger auf stdout ausgegeben werden sollen. In hexadezimaler und ASCII-Ansicht werden die Rückantworten ersichtlich, so dass sich exakt bestimmen lässt, welcher Reiz nun welche Reaktion ausgelöst und damit innerhalb des Pattern-Matchings gegriffen hat. Ähnlich verhält es sich bei der Option –b, bei der sämtliche ASCII-Rückgaben der erfolgreichen Trigger ausgegeben werden.

Neben der binären Datei von amap, die dessen Herzstück darstellt, werden drei modulare Datenbank-Dateien eingesetzt. Diese als normale Text-Dateien abgespeicherten Daten finden sich standardmässig allesamt in /usr/share/amap. Von erhöhter Wichtigkeit ist dabei die Datei appdefs.trig, welche die Trigger für das Umsetzen der Reize bereitstellt. Beispielsweise wird für den Test von SMTP, dieser wird standardmässig auf den `Port tcp/25` angewandt, der Reiz `HELO AMAP\r\n` eingesetzt.

Ebenfalls von erhöhter Wichtigkeit ist die Datei appdefs.resp, die die möglichen Rückantworten der jeweiligen Dienste als reguläre Ausdrücke bereitstellt. Beispielsweise rechnet amap bei der Rückantwort eines FTP-Servers mit Zeichenketten wie *Login name*: oder `^220.*FTP`. Auf komplexe, verschachtelte und rekursive reguläre Ausdrücke wird verzichtet. Es finden sich sodann für verschiedene Dienste mehrere Zeilen, die sich dem potentiellen Erscheinungsbild annehmen können. Für die Analyse des Verhaltens von RPC-Ports wird die separate Datei appdefs.rpc eingesetzt, die die jeweiligen IDs der bekannten RPC-Dienste vermerkt.

Automatisierung mittels nmap-Ausgaben

Eine Besonderheit, die amap aufzuweisen hat und die die keine anderweitige Erwähnung findet (auch nicht beim Entwickeln einer eigenen Lösung) ist das Analysieren des Umfangs einer Reaktion. Und zwar wird in der vierten Spalte der Response-Datei appdefs.resp eine optionale mittels Integer-Werten (Absolut oder Bereich) die erwartete Länge der Rückantworten vermerkt. Dies ist sodann ein zusätzliches Indiz, um die Korrektheit einer Überprüfung sicherzustellen.

Entwicklung einer eigenen Lösung

Da wir nun im ersten Teil die konzeptionellen und technischen Grundlagen des Application Mappings gesehen und im zweiten Teil eine mögliche Implementierung

anhand des Vorzeige-Beispiels von THC amap diskutiert haben, wollen wir nun eine eigene Software-Umsetzung anstreben. Hierbei geht es lediglich darum, generische Beispiele aufzuzeigen, anhand derer wir unsere Design-Entscheidungen und die jeweiligen Implementierungen besprechen wollen. Perfektion der Umsetzung der Software geniesst in diesem Kapitel keinerlei Priorität. Ebenfalls wollen wir unsere Anstrengungen mit einem einfachen Shell-Skript implementieren. Dieses bietet zwar weder von der Entwicklungssicht in Bezug auf Performance noch in in Punkto Ergonomie ein angenehmes und anzustrebendes Mass. Jedoch ist mit einer einfachen und generischen Skripting-Sprache das Thema besser auch für ein größeres Publikum verständlich zu erklären. Die Anforderungen unseres kleinen Entwicklungs-Projekts sind die folgenden:

- `[Software-Nutzen]` Implementierung einer Anwendung zum semi-automatischen Umsetzen von Application Mapping;
- `[Umfang]` Umsetzung einer zeilenbasierten Lösung mittels einfachem Shell-Skripting;
- `[Pattern-Matching]` Lediglich das Umsetzen eines einfachen Pattern-Matchings innerhalb der Rückantworten.

Wie wir an dieser Anforderungsliste sehen können, werden wir auf das Implementieren einiger der möglichen Analyse-Techniken (z.B. das Timeout-Verhalten und Längen-Analysen) verzichten. Stattdessen geht es uns wirklich nur um das direkte Abgreifen von Rückmeldungen mittels Triggern, die wir sodann auf unsere gespeicherten Muster überprüfen wollen. Das folgende Flussdiagramm illustriert dieses einfache Verhalten (Abbildung 1.).

Wir überprüfen als erstes, ob sich der Zielport überhaupt im LISTENING-Modus befindet. Ist dem nicht so, wird dieser Zustand gespeichert und das Programm mit einer Fehlermeldung verlassen. Können wir eine Verbindung zum Zielport herstellen, schicken wir als erstes einen Trigger, mit dem wir eine Rückantwort provozieren wollen. Sodann analysieren wir diese auf das Vorhandensein eines Musters. Falls das Pattern in der Rückantwort enthalten ist, wird dieses Resultat zwischengespeichert und für die Ausgabe vorbereitet. Bevor die Anwendung dann verlassen wird, wird das Resultat des Scan-Vorgangs ausgegeben.

Einfache Shell-Skript Implementierung

In diesem Kapitel werden wir uns eine wirklich primitive Implementierung der zuvor beschriebenen Anforderungen eines zeilenorientierten Tools zur semi-automatischen Umsetzung von Application Mapping zusammenbasteln. Dazu bedienen wir uns eines simplen Shell-Skripts, das insgesamt 19 Zeilen umfasst (Listing 1.)

In der Zeile 03 geben wir den Titel unserer Applikation bei jedem Aufruf bekannt. Wir wollen unsere Beispiel-Implementierung pmap nennen, das in Anlehnung an nmap und amap für Protocol Mapping steht.

In der Zeile 05 führen wir mit der if-Abfrage eine Überprüfung der Anzahl der mit dem Skript-Aufruf übergebenen Argumente durch. Werden nicht 2 Argumente übergeben, fällt die Abfrage direkt in die else-Anweisung (Zeile 16). Diese gibt lediglich eine Fehlermeldung aus, erklärt kurz den erforderlichen Syntax für den Aufruf des Skripts und beendet dieses dann (Zeilen 17 bis 19).

Werden exakt 2 Argumente übergeben, startet die Anwendung das Protocol Mapping (Zeile 06). Die Zeile 07 kann als Herzstück der Anwendung betrachtet werden. Hierbei wird mit dem Netzwerk-Utility NetCat eine Verbindung zum Zielsystem (erstes Argument $1) und dem Zielport (zweites Argument $2) umgesetzt. Konnte eine Verbindung etabliert werden, wird über diesen Kanal ein Newline-Zeichen geschickt. Das Resultat dieses Zugriffs wird in der temporären Variable $RESPONSE zwischengespeichert.

Danach findet direkt eine Überprüfung des Vorhandenseins des Triggers `ftp` in der Variable `$RESPONSE` gespeicherten Rückgabe statt. Hierzu wird der grep-Befehl mit dem entsprechenden Trigger auf die Quelldaten angewendet und in die temporäre Variable `$MATCH` abgelegt (Zeile 10).

Das Gehirn von pmap prüft nun, ob die Variable $MATCH einen Inhalt hat, oder nicht (Zeilen 11 bis 15). Konnte die grep-Überprüfung nicht erfolgreich umgesetzt werden, hat diese nichts in die Variable gespeichert. Andernfalls findet sich die erfolgreich ermittelte Zeile in der besagten Variable. Jenachdem, ob nun eine Übereinstimmung gefunden werden konnte, wird der jeweilige Status ausgegeben (Zeilen 12 oder 14). Ein erfolgreicher (Zeilen 01 bis 04) und ein nicht erfolgreicher (Zeilen 06 bis 09) Durchlauf dieses kleinen Skripts sieht auf einem Linux-System folgendermassen aus:

```
01 mruef@debian:~$ ./pmap.sh ftp.
   computec.ch 21
02 pmap v1.0
03 Starting protocol mapping …
04 Testing for trigger
   ftp … Found!
05
06 mruef@debian:~$ ./
   pmap.sh smtp.computec.ch 25
07 pmap v1.0
08 Starting protocol mapping …
09 Testing for trigger
   ftp … Not found.
```

Wie gesagt ist diese Umsetzung von pmap sehr einfach gehalten, um das Prinzip einer solchen Implementierung didaktisch einfacher darlegen zu können. In einer professionellen Applikation wären eine Vielzahl an Fehler-Überprüfungen von Vorteil, zusätzliche Tests erforderlich und ein umfangreicheres Reporting anzustreben. Vom Prinzip her wird aber eigentlich kein Neuland mehr betreten.

Der Zuverlässigkeits-Algorithmus

Das Grundprinzip des Application Mappings ist schnell verstanden und eine einfache Implementierung dessen mit wenigen Zeilen umgesetzt. Meines Erachtens kann eine Anwendung wie amap noch deutlich verbessert werden, wenn zusätzliche Auswertungs-Mechanismen implementiert werden. Diese differentierte Analyse der Charakteristika eines Verbindungs- bzw. Service-Verhaltens muss sodann wiederum mit einem Algorithmus aufbereitet werden, der die logische Gewichtung sowie die Abhängigkeiten umfassend abdeckt. Tabelle 5 zeigt auf, welche möglichen Techniken zm Einsatz kommen können, welches logische Gewicht innerhalb der Analyse ihnen zuteil wird und auf welche Teile der Dienstanalyse sie genau abzielen.

Den möglichen Ablauf eines solchen Algorithmus möchte ich gerne exemplarisch Schritt für Schritt darstellen, so wie er in einer praktischen Implementierung sinnvoll scheint:

- falls der Zielport von IANA für einen Dienst R vorgesehen ist, wird dieser Wert R mit der logischen Gewichtung von 5 % zwischengespeichert. A = { R, 5 };
- falls für diese Verbindung das vorherrschende Transportprotokollgenutzt wird, wird dieser Wert mit der logischen Gewichtung von 5 % zwischengespeichert. B = { R, 5 };
- das Begrüßungsverhalten mittels Willkommens-Banner nach Etablierter Sitzung wird mit 5 % zwischengespeichert. C = { R, 5 };
- das Timeout-Verhalten wird mit 5 % zwischengespeichert. R = { R, 5 };
- die automatische Rückantwort nach etablierter Sitzung wird auf etwaige Muster hin untersucht und mögliche Resultate mit 35 % zwischengespeichert. E = { R, 35 };
- die durch Reize provozierten Rückantworten werden ebenfalls auf etwaige Pattern hin untersucht und mögliche Resultate mit 45 % gespeichert. F = { R, 45 };
- durch das Ermitteln derjenigen Dienste, die am meisten Gewichtung bekommen haben, können die am ehesten in Frage kommenden Resultate bestimmt werden. Z = { A, B, C, D, E, F } |.

Für das Umsetzen dieses Algorithmus bietet sich fast eine Datenbank an, die sich jedoch auch nur virtuell in Form von Arrays oder als flache Datei-Datenbank realisieren lässt. Durch simple arithmetische Operationen (Addition und Division) können Mittelwerte errechnet und Wahrscheinlichkeitsaussagen dank der logischen Gewichtungen getätigt werden.

Gegenmassnahmen

Nun zum Schluss stellt sich die entscheidende Frage, wie sich sicherheitsbewusste Administratoren denn nun vor automatisierten oder manuellen Auswertungen im Sinne eines Application Mappings (z.B. mit amap) schützen können. Grundsätzlich können die gleichen Techniken ihren Einsatz finden, wie sie schon seit Jahren durch raffinierte Viren-Programmierer oder hochspezialisierte Penetration Tester herangezogen wurden: Man versucht durch das Abändern des Verhaltens die mit Pattern-Datenbanken umgesetzten Fingerprint-Analysen ins Leere laufen zu lassen. Wobei Entwickler von Computerviren gegen die Antiviren-Lösungen und Penetration Tester gegen Intrusion Detection-Systeme kämpfen, ist unser Gegner in diesem Fall der Auswertungs-Prozess des Application Mappings.

Wir haben zwar gesehen, dass das Umsetzen effizienter und umfassender regulärer Ausdrücke eine hohe Kunst darstellt. Entsprechend können wir unter Umständen mit einfachen Anpassungen des Verhaltens bzw. der Ausgaben das Pattern-Matching verhindern. Aber genauso sind wir gewissen Regeln unterworfen, die das Anpassen der Protokolle oder gar das komplette Neuerfinden dieser verhindert. Viren-Programmierer können auch nicht einfachso nach Lust und Laune das durch die Antiviren-Lösung genutzte Byte X durch ein anderes ersetzen. Ein solcher Eingriff hat meistens eine verheehrende Auswirkung auf das

Im Internet

- *computec.ch, http://www.computec.ch*, deutschsprachiges Archiv mit freien Publikationen zum Thema Computersicherheit;
- *THC amap, http://thc.org/thc-amap/*, die populäre Implementierung eines Application Mapping Tools;
- *nmap, http://www.insecure.org/nmap/*, das populäre Scanning- und Auswertungs-Utility.

Über den Autor

Marc Ruef arbeitet als Security Consultant bei der schweizer Firma scip AG und ist dort Leiter des Bereichs Security Auditing und Penetration Testing. Im Oktober 2002 ist im Data Becker Verlag sein zweites Buch mit dem Titel *Hacking Intern* erschienen. Neben einer Vielzahl von Fachpublikationen zur theoretischen Informatik und IT-Security unterstützt er diverse internationale Projekte aus diesen Bereichen. Seit 1997 betreut er die Webseite *computec.ch*, die als grösstes Archiv deutschsprachiger Publikationen zum Thema Informationssicherheit gilt. Des Weiteren ist er Entwickler des Attack Tool Kit (ATK), einem open-source Exploiting Frame-work, das als Ergänzung zu Lösungen wie Nessus generische
Penetration Tests einfacher und transparenter macht.
Kontak mit dem Autor:
marc.ruef@computec.ch
http://www.computec.ch/

grundlegende Verhalten des Produkts selbst, was dann ebenfalls nicht im Interesse des Autors liegt.

Wenn wir die Muster-Datenbank von amap betrachten, dann sehen wir, dass viele der Überprüfungen auf dem Suchen des Namens des Protokolls in der Rückantwort umgesetzt werden. Beispielsweise deutet das Vorhandensein der Zeichenkette SMTP auf den Einsatz des Simple Mail Transfer Protocols hin. Genau gleich verhält es sich mit FTP, HTTP und vielen anderen Protokollen. Entsprechend muss ein Administrator darum bemüht sein, dass er auf das Herausgeben dieser grundlegenden Information verzichtet. Vor allem die Willkommens-Banner freizügiger Anwendungen müssen so angepasst werden, dass auf das Nennen der eigenen Funktion gänzlich verzichtet wird. Sodann kann der Angreifer nur noch durch gezielte Reize eine verräterische Reaktion provozieren. Vor allem bei automatisierten Lösungen wie amap wird dies aber viele Test-Zugriffe ins Leere laufen lassen. Alleine in der Version 4.8 von amap würden von 349 Patterns insgesamt ca. 78 erfolgreiche Zugriffsvarianten wegfallen. Darunter einige wirklich populäre Umsetzungen wie FTP oder SMTP.

Eine andere Möglichkeit liegt darin, dass wir mit einem Application Gateway durch einen dedizierten Proxy den zu schützenden Dienst entkoppeln [Ruef et al. 2002, Ruef 2004, 2006]. Zwar können wir so nicht direkt verhindern, dass eine erfolgreiche Ermittlung des angebotenen Anwendungsprotokolls durchgeführt werden kann. Jedoch können wir so in den meisten Fällen das weitere Bestimmen der spezifischen Implementierung (z.B. IIS oder Apache) enorm erschwert werden.

Zusammenfassung

Mit dem Application Mapping wird versucht anhand des Verhaltens des an einen offenen Port gebundenen Dienst das genutzte Protokoll zu ermitteln. Durch verschiedene Verfahren kann dabei zwischen den einzelnen Anwendungen und teilweise ihren charakteristischen Implementierungen unterschieden werden. Eine wichtige Rolle spielt dabei das Pattern-Matching innerhalb der Rückantworten des überprüften Dienstes.

Application Mapping ist zwar eine klassische Disziplin der Computersicherheit, der jedoch in der älteren Literatur nur wenig Aufmerksamkeit geschenkt wurde. Praktisch nie wurde der eigentliche Schritt der Protokoll-Erkennung als dedizierter Vorgang wahrgenommen. Erst mit amap - als erste populäre Implementierung einer automatisierten Lösung - könnte das Application Mapping einen festen Platz innerhalb des netzwerkbasierten Security Auditings einnehmen.

Im weiteren Verlauf haben wir die grundlegende Funktionsweise von amap kennengelernt, um uns dann selber an die Entwicklung einer ähnlichen Lösung zu machen. Diese haben wir sehr einfach umgesetzt, um das Prinzip einer solchen Implementierung besser diskutieren zu können. Desweiteren haben wir dennoch einige Möglichkeiten besprochen, wie unsere eigene Umsetzung und amap implemeniert und die Resultate noch zuverlässiger ermittelt werden können. Durch die Kombination zusätzlicher Techniken (z.B. Analyse der Port-Vorgaben durch IANA oder das Timeout-Verhalten) können sehr exakte Determinierungen angestrebt werden.

Und zum Schluss haben wir betrachtet, wie wir als sicherheitsbewusste Administratoren Vorkehrungen treffen können, um das automatisierte oder manuelle Application Mapping zu erschweren. Durch das Abändern des Verhaltens des angebotenen Dienstes können wir vor allem automatisierte Zugriffe, wie sie zum Beispiel durch amap initiiert werden, ins Leere laufen lassen. Der Prozess des Application Mappings wird so wieder zu einer zeitintensiven und mühseligen Aufgabe, die manuell bewältigt werden will. ●