Universität Konstanz
Fachbereich für Informatik und Informationswissenschaft
Lehrstuhl für verteilte Systeme

Wissenschaftliche Arbeit
zur Erlangung des Grades eines Bachelor of Science (B.Sc.)
im Fachbereich Informatik & Informationswissenschaft der Universität Konstanz

Ein verbesserter PAM basierter Ansatz um "brute force" Passwort-Angriffe auf den "secure shell" Service zu erkennen und zu verhindern

Verfasser:

Jürgen Kollek

November 13, 2013

Gutachter:

Prof. Dr. Marcel Waldvogel

Prof. Dr. Marc H. Scholl

Universität Konstanz
Fachbereich für Informatik und Informationswissenschaft
D–78457 Konstanz
Germany

Abstrakt

SSH Verbindungen werden häufig verwendet. Ein Beispiel hierfür wäre die Wartung eines Servers durch einen Administrator, oder um Dateien zwischen zwei Computern zu kopieren. Es ist also attraktiv diese Verbindung auszunutzen und zu hacken.

Eine der häufigsten genutzten Methoden um SSH Verbindungen zu hacken ist die Brute-Force Methode. In meinem Bachelor-Projekt und meiner Bachelor-Thesis habe ich ein Programm implementiert, welches SSH-Server vor Brute-Force Angriffen schützt. Der Schutz basiert nicht nur auf dem zählen der fehlgeschlagenen Login-Versuche, sondern geht darüber hinaus. Wie der Schutz genau funktioniert, werde ich in dieser Thesis beschreiben.

Danksagung

Einen besonderen Dank möchte ich an meinen Tutor Manuel Knitza richten, da er mir bei meinem Projekt als auch bei meiner Thesis mit Rat und Tat zur Seite stand. Ohne ihn wäre ich an manchen Problemen nicht weiter gekommen. Ebenfalls einen Dank möchte ich an meinen Professor Dr. Marcel Waldvogel richten, der beim Wechsel meines Tutors half und ein offenes Ohr hatte. Zuletzt möchte ich Prof. Dr. Marc H. Scholl danken, dass er die zweit Korrektur meiner Bachelorarbeit übernommen hat.

Contents

List of Figures

List of Algorithms

List of Tables

Chapter 1

Einleitung

Es gibt viele Gründe für die Benutzung von Secure Shell (kurz SSH). Der wichtigste ist sicherlich die Kontrolle, die man über einen Server erhält. Dies kommt daher, dass man ein lokales Terminal bekommt, das zu dem entfernten Server gehört. Mit Hilfe dieses Terminals kann man so agieren, als würde man sich direkt vor dem Server befinden. Man kann alle Funktionen verwenden, die auch ein lokales Terminal erlaubt. Daher ist es für Hacker besonders attraktiv diese Verbindungen zu hacken und die Kontrolle über den Server zu übernehmen um eventuell geheime oder private Dateien auszuspähen. Deswegen ist es wichtig Server gegen solche Angriffe besonders zu schützen.

Bisher gibt es einige Ansätze um Secure Shell Verbindungen sicherer zu machen. Einer dieser Ansätze ist DenyHosts[1], welches ursprünglich für ein Linux Betriebssystem implementiert wurde. Es ist aber auch möglich dies auf einem Windows Betriebssystem zu nutzen. DenyHosts liest hierfür die "/log/auth.log" des Linux Betriebssystems aus und ermittelt die fehlgeschlagenen Login-Versuche. Die "auth.log" Datei protokolliert alle Anmeldeversuche am System mit. Wird eine bestimmte Schwelle überschritten, wird die IP-Adresse des Hosts in die Datei "/etc/hosts.deny" eingetragen. Durch diese Datei werden Zugriffe auf dem Server gesperrt. Der Nachteil von DenyHosts ist, dass es die "auth.log" Datei auswertet. Erfolgt ein Angriff, versucht der Angreifer manchmal die "auth.log" Datei zu manipulieren und so den Angriff zu verstecken.

Ein anderer Ansatz ist Fail2Ban[2]. Hierbei wird ebenfalls die "/log/auth.log" ausgewertet. Der Unterschied zu DenyHosts besteht darin, dass Fail2Ban die IP-Adresse des Hosts in die "iptables" einfügt und ihnen so den Zugriff verwehrt. Die "iptables" ist die integrierte Linux Firewall, welche eintreffenden als auch ausgehenden Verkehr vom System überwacht. Fail2Ban kann z.b. für Apache, qmail und Postfix verwendet werden. Der Nachteil von Fail2Ban besteht darin, dass es die "auth.log" Datei auswertet. Das Programm läuft dauerhaft im Hintergrund und muss die Log Datei immer wieder erneut untersuchen um mögliche Angriffe aufzudecken. Außerdem können Angreifer durch Manipulation, wie zum Beispiel "log injection", den Angriff verstecken. Bei der

[1] Homepage DenyHosts `http://denyhosts.sourceforge.net/`

[2] Homepage Fail2Ban `http://www.fail2ban.org/wiki/index.php/Main_Page`

"log injection" werden fehlerhafte Informationen in die Log-Datei geschrieben und somit können sich Angreifer verstecken.

Ein anderer Ansatz wird von PAM_abl[3] verfolgt. Dieser verwendet als Grundlage ein PAM-Modul, welches bei der Verbindung mit dem Server aufgerufen wird. PAM steht für "Pluggable Authentication Modul"[4] und ist eine Softwarebibliothek, welche eine Programmierschnittstelle für Authentifizierungsdienste bereitstellt. Wenn der Host eine Schwelle überschreitet, wird er wie in Fail2Ban in die "iptables" eingetragen. Der Nachteil von PAM_abl ist, dass es keine Möglichkeit gibt mit GeoLocations, Passwortanalyse oder DNSBL-Datenbanken zu arbeiten. Der Vorteil mit geobasierten Daten zu arbeiten, besteht darin, dass es Länder gibt, welche eine erhöhte Auffälligkeit im Netz haben als andere und man somit auffälligen Ländern weniger Login Versuche ermöglicht. Der Vorteil von Passwortanalyse ist, dass Brute Force Angriffe meistens komplett unterschiedliche Passwörter verwenden als "normale" Benutzer. Ein normaler Benutzer wird oft ähnliche Passwörter zum richtigen testen. Also kann man auch hier erkennen, ob eine Auffälligkeit vorliegt oder nicht. Wenn man auf schon vorhandene DNSBL-Datenbanken zurückgreift, dann werden viele schon auffällige IP-Adressen am Anfang abgefangen und diesen keinen Zugriff auf den Server erlaubt.

Natürlich gibt es noch viel mehr Programme, aber diese zählen mit unter zu den bekanntesten Vertretern.

Das Programm "PAM_abfp" (PAM_automatic brute force prevention)[5], welches während meiner Bachelorthesis erstellt wurde ist als PAM-Modul implementiert. Ebenso wurde dieses mit der Auswertung von geografischen Daten erweitert. Zusätzlich wurde das Programm noch an eine DNSBL-Datenbank angebunden und es wurde eine Passwortanalyse hinzugefügt.

Eine kleine Übersicht über die Eigenschaften der genannten Programme findet man in Tabelle 1.1:

Eigenschaften	**DenyHosts**	**Fail2Ban**	**pam_abl**	**pam_abfp**
Loginauswertung	"auth.log"	"auth.log"	pam basiert	pam basiert
mehrere Dienste	-	✓	✓	✓
iptables Eintrag	-	✓	✓	✓
GeoLocations	-	-	-	✓
Passwortanalyse	-	-	-	✓
DNSBL-Datenbank	-	-	-	✓

Table 1.1: Eigenschaften der einzelnen Programme

Die Thesis gliedert sich wie folgt: Die Grundlagen des Programms werden in Kapitel 2 dargelegt. Hier werden die Grundbegriffe erklärt welche nötig sind um die Thesis zu verstehen.

[3]Download-Link: `http://sourceforge.net/projects/pam-abl/`

[4]PAM-Entwicklung: [Gei08]

[5]zu finden auf: `https://github.com/KollekJ/PAM_abfp`

Weiter werde ich im Kapitel 3 mein Bachelor-Projekt erklären und erläutern.

Im nächsten Kapitel 4 werden die Veränderungen und Erweiterungen zum Bachelor-Projekt darlegen.

In Kapitel 5 wird das fertige Programm erläutert und seine Funktionsweise erklärt. Ebenso werden die Vor- und Nachteile, als auch weitere mögliche Erweiterungen aufgezeigt.

Eine Zusammenfassung im Kapitel 6 beschließen diese Thesis.

Chapter 2

Grundlagen

Ziel dieses Kapitels ist die Einführung in die Thematik von Secure Shell und Brute Force Angriffen.

2.1 Abkürzungen

SSH - Secure Shell
API - Application Programming Interface (Programmierschnittstelle)
X11 - Windows Fenster zum Darstellen von Oberflächen über ein Netzwerk
Overhead - Zusatzarbeit für das Programm durch Verwendung von mehr Speicher, Rechenleistung und Zeit
Malware - bösartige Software
Exploits - Schwachstelle ausnutzen
GeoLocation - Zuordnung von IP-Adressen zu ihrer geografischen Herkunft
Terminal - Benutzerendgerät zur Eingabe und Anzeige von Daten
iptables - Linux-Firewall
log-injection - Methode um Log-Dateien zu verändern
DNSBL - abfragbare schwarze Liste
Performance - Zeitverhalten von Programmen
Debugger - Werkzeug zum Diagnostizieren und Auffinden von Fehlern in Programmen
Grep - Programm zur Suche und Filterung von Dateien
Tail - Programm zur Ausgabe der letzten Zeilen
Parameter - Übergabewert der bei der Verarbeitung berücksichtigt wird
Sturctur - Objekt zur Organisierung und Speicherung von Daten
Memory Leak - Fehler bei welchem Speicherplatz belegt wird ohne diesen wieder freizugeben
Array Index out of Bounds - Zugriff auf ein Element, welches die Kapazität des Arrays übersteigt
Array - Datenstruktur
"shadow" Datei - Datei die verschlüsselte Passwörter der Benutzer enthält

Session - Verbindung zwischen Client und Server
CRON - Programm zur zeitlichen Ausführung von Programmen
Apache - Webserver Programm
qmail - Mailserver
Postfix - Mail Transfer Agent, nimmt E-mails entgegen und sendet diese
root - Benutzer mit vollen Zugriffsrechten
shared secret - geteiltest Geheimniss

2.2 Secure Shell

Bei Secure Shell (kurz SSH) handelt es sich um ein Netzwerkprotokoll, sowie die entsprechenden Programme um eine Verbindung zwischen zwei Geräten herzustellen. Es ermöglicht eine verschlüsselte und authentifizierte Verbindung zwischen zwei Computern herzustellen. Diese Verbindung ist über ein ungesichertes Netzwerk möglich.

Die Methode wird häufig genutzt um sich eine entfernte Kommandozeile auf den lokalen Rechner zu holen. Dies bedeutet, dass man auf der lokalen Kommandozeile die Ausgaben der entfernten Konsole erhält und die Tastatureingaben der lokalen Konsole an den entfernten Rechner gesendet werden. Seit Version 2, SSH-2, ist man nicht mehr nur auf Terminals beschränkt. Es ist sogar möglich eine X11 Umgebung zu laden, also eine grafische Oberfläche zu erhalten. Ebenfalls möglich ist es entfernte Dateisysteme zu mounten, also lokal einzubinden. Damit erhält man eine einfache Möglichkeit die Dateien des Dateisystems zu durchsuchen. In Kombination mit der X11 Umgebung ist es, als würde man sich vor einem lokalen Computer befinden.

Andere Anwendungsgebiete sind "Tunneln" und "Portweiterleitung". Unter "Tunneln" versteht man die Konvertierung und Übertragung eines Kommunikationsprotokolls. Wie man in Figure 2.1 [1] sehen kann, nimmt der Tunnelpartner die Verbindung auf und leitet die Anfrage an den richtigen Dienst weiter. Bei "Portweiterleitungen" handelt es sich um Weiterleitungen einer Verbindung, welche auf einen bestimmten Port eingeht, zu einem anderen Computer. Man kann also sagen, dass man durch SSH Zugriff auf andere Geräte erhält, als ob man sich direkt vor dem entfernten Gerät befindet.

[1] Wikimedia SSH-Tunnel[Wik13]

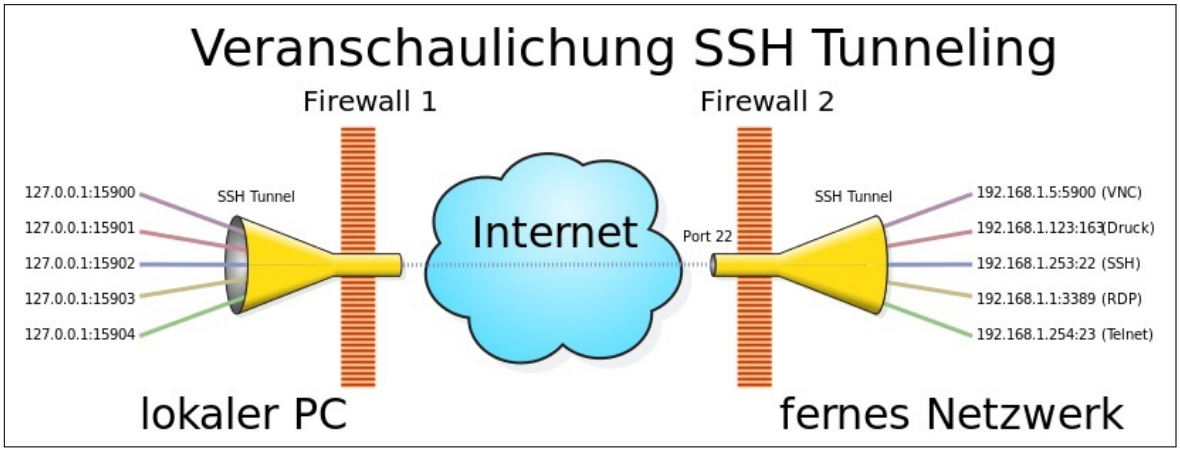

Figure 2.1: SSH-Tunnel

2.3 Brute-Force-Methode

Die Brute-Force-Methode gehört zu einer der meist genutzten Vorgehensweisen, um Probleme in der Informatik zu lösen[2]. Dies kommt sicher mitunter daher, da diese leicht zu implementieren ist und der Algorithmus sehr einfach ist. Sie probiert dabei alle potenziell möglichen Lösungswege durch, bis eine gefunden wurde. Theoretisch hat sie eine Erfolgswahrscheinlichkeit von fast 100%, jedoch ist dabei zu beachten, dass der Aufwand an Rechenoperationen proportional zu den probierenden Lösungen ist. Weiterhin ist zu beachten, dass mit steigendem Umfang die Anzahl an Lösungen exponentiell steigt. Dies bedeutet, je mehr Lösungen und je länger diese sind, desto länger dauert es, das richtige Passwort zu erraten.[3]

Eine Erweiterung ist das verwenden von Wörterbüchern. Dadurch kann die Brute-Force-Methode beschleunigt werden, da man nur auf vordefinierte Lösungen zurückgreift und diese nicht noch während der Laufzeit berechnet werden müssen. Das Problem dabei ist, dass sich die Lösung in dem Wörterbuch befinden muss, sonst wird sie nicht gefunden. Solche Angriffe heißen "Wörterbuchangriffe".

Diese Vorgehensweise ist häufig erfolgreich, da Benutzer oft kurze und einfache Passwörter verwenden, welche nur aus Zeichen des Alphabets bestehen. Dies schränkt die Anzahl an möglichen Kombinationen ein und erleichtert das "erraten" des Passwortes. Mit einem normalen Computer ist es möglich etwa 15 bis 25 Millionen Passwörter pro Sekunde auszuprobieren. In Figure 2.2[4] ist der Zusammenhang zwischen Passwortlänge und verwendeter Zeichen veranschaulicht. Wie man sieht hat die Länge und die Variation der Zeichen enorme Auswirkungen auf die Dauer der Entschlüsselung. Zu der Entschlüsselung des Passwortes kommt auch noch die Dauer hinzu, welche man benötigt

[2]Siehe: [Jam08]
[3]Brute-Force [wik]
[4]Zusammenhang Länge und Zeichen [pd]

Passwort besteht aus	Mögliche Kombinationen	Benötigte Zeit zum Entschlüsseln
5 Zeichen (3 Kleinbuchstaben, 2 Zahlen)	36^5= 60.466.176	60.466.176 / 2.000.000.000 = **0,03 Sekunden**
7 Zeichen (1 Großbuchstabe, 6 Kleinbuchstaben)	52^7= 1.028.071.702.528	1.028.071.702.528 / 2.000.000.000 = 514 Sekunden = **ca. 9 Minuten**
8 Zeichen (4 Kleinbuchstaben, 2 Sonderzeichen, 2 Zahlen)	68^8= 457.163.239.653.376	457.163.239.653.376 / 2.000.000.000 = 228.581 Sekunden = **ca. 2,6 Tage**
9 Zeichen (2 Großbuchstaben, 3 Kleinbuchstaben, 2 Zahlen, 2 Sonderzeichen)	94^9= 572.994.802.228.616.704	572.994.802.228.616.704 / 2.000.000.000 = 286.497.401 Sekunden = **ca. 9,1 Jahre**
12 Zeichen (3 Großbuchstaben, 4 Kleinbuchstaben, 3 Sonderzeichen, 2 Zahlen)	94^{12}= 475.920.314.814.253.376.475.136	475.920.314.814.253.376.475.136 / 2.000.000.000 = 237.960.157.407.127 Sekunden = **ca. 7,5 Millionen Jahre**

Figure 2.2: Zeit um bestimmte Längen eines Passwortes zu entschlüsseln

um einen passenden Benutzernamen zu finden. Oft beschränkt sich der Angriff auf Benutzernamen, welche auf jedem System vorhanden sind, wie etwa "root"[5].

Um den Angreifer den Angriff zu erschweren ist es notwendig längere und unterschiedlichere Passwörter zu verwenden.

2.4 Botnet-Angriffe

Bei der Brute-Force-Methode ist es relativ einfach einen Angreifer ausfindig zu machen. Man kann den Angreifer daran erkennen, dass er vermehrt Anfragen innerhalb eines gewissen Zeitraumes an den Server sendet. Hierbei kann man davon ausgehen, dass es sich nicht um einen echten Nutzer handelt, sondern um einen Angriff. Aus Schutz wird der Server den Angreifer (genauer seine IP-Adresse) für eine gewisse Zeit sperren. Dies verhindert, dass der Angreifer eine Verbindung zum Server herstellen kann. Als Folge wird der Angriff für eine gewisse Zeit ausgesetzt.

Mithilfe von Botnets versucht man diesen Schutzmechanismus der Server zu umge-

[5]Siehe: [Chr06]

hen. Ein Botnet besteht aus mehreren verknüpften Computern[Ref.[6]]. Der Betreiber des Botnets kann dann mit den verschiedenen Computern jeweils kleinere Teilanfragen (3-5 Anfragen pro Server)[7] verschicken und somit die Wörterliste für einen Server abarbeiten. Da die Angreifer immer nur kleine Anfragen an den Server senden bleiben diese unter der Bannschwelle des Servers und somit ist dessen Schutz außer Kraft gesetzt. Weiterhin bleibt der gesamte Angriff unentdeckt, da die einzelnen Angreifer verschiedene IP-Adressen besitzen. Botnets können noch für viele andere Aufgaben missbraucht werden, wie zum Beispiel zum versenden von SPAM-Nachrichten oder zum verbreiten anderer Schadsoftware.

In Figure 2.3 ist ein Botnet veranschaulicht[Quelle [8]]. In dieser sieht man auch, wie ein Botnet entsteht. Oft wird ein noch nicht an das Botnet angebundener Computer mithilfe einer Schadsoftware infiziert. Wenn die Infizierung erfolgt ist, wird der Computer ins Botnet aufgenommen und stellt seine Ressourcen dem Betreiber des Botnets zur Verfügung. Die Infizierung geschieht meist für den Benutzer unsichtbar. Für das Eingliedern in das Botnet gibt es verschiedene Möglichkeiten, wie zum Beispiel durch Malware, Downloads oder Exploits. Bei Malware handelt es sich oft um Programme, die installiert werden und sich unter anderem im Anhang von E-Mails befinden. In Downloads kann sich ebenfalls Schadsoftware befinden. Bei Exploits handelt es sich um die Ausnutzung von Sicherheitslücken im Betriebssystem, Browsern oder anderer Software. Die Bot-Schadsoftware verfügt ebenfalls oft über eine automatische Weiterverbreitung, wodurch neue Bots hinzukommen. Wenn die Infizierung entdeckt und gelöscht wird, fällt der Computer aus dem Botnet.

2.5 Schutz vor Botnets

Eine Möglichkeit Schutz vor Botnet-Angriffen zu erhalten ist, indem man "SSH-Keys"[9] verwendet. Im Gegensatz zum normalen "Benutzername - Passwort" Schema verwendet "SSH-Keys" ein Paar von Schlüsseln (Keys). Dabei handelt es sich um den öffentlichen und den privaten Schlüssel. Der private Schlüssel ist nur dem Benutzer bekannt. Der öffentliche Schlüssel liegt auf dem Server bereit. Möchte sich ein Benutzer nun am Server anmelden erzeugt dieser mithilfe des privaten Schlüssels eine Signatur, welche dem Server übermittelt wird. Dieser verifiziert nun mit dem öffentlichen Schlüssel die Signatur des Benutzers, woraufhin eine Verbindung hergestellt wird. Die Errechnung des privaten Schlüssels aus dem öffentlichen Schlüssel ist praktisch unmöglich. Möchte man seinen privaten Schlüssel noch weiter schützen, empfiehlt es sich diesen mit Hilfe eines Passwortes zu verschlüsseln. In Figure 2.4 ist der Ablauf eines Verbindungsaufbaus mit SSH-Keys veranschaulicht[Quelle [10]].

[6] Microsoft [Mic]
[7] siehe: [YNS11]
[8] Bild-Quelle[hei]
[9] siehe: [MAvO11]
[10] Bild-Qulle [dev]

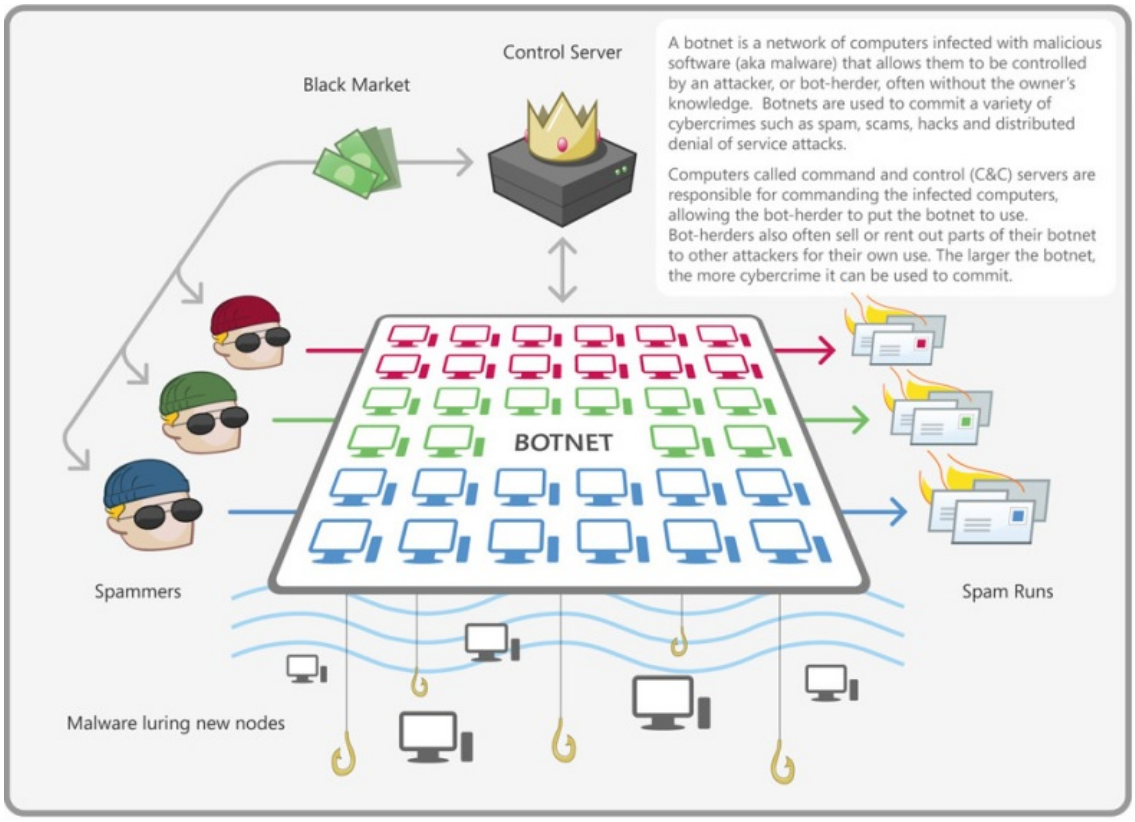

Figure 2.3: Botnet

Eine weitere Möglichkeit ist die Verwendung von "white lists". Eine "weiße Liste" ist eine Liste, die auf dem Server hinterlegt ist. In dieser stehen IP-Adressen, welche als vertrauenswürdig betrachtet werden und deshalb nicht nach einer gewissen Anzahl an Versuchen gesperrt werden. Man könnte auch nur Benutzern aus der Liste eine Verbindung zum Server herstellen lassen, sodass der Administrator des Servers den Benutzer explizit in die Liste eintragen muss. Die Folge wäre, dass Angreifer sich gar nicht mit dem Server verbinden können, sondern nur ausgewählte Benutzer.

Eine andere Variante ist die Möglichkeit mehrere Server, die einen Schutz vor der Brute-Force-Methode installiert haben, zu vernetzen[11]. Ein Angreifer, der bei den einzelnen Servern nicht entdeckt werden würde, aufgrund der Unterschreitung der Bannschwelle kann mit Hilfe von mehreren Servern entdeckt werden. Wenn eine IP-Adresse bei mehreren Servern wiederholt auftritt, kann davon ausgegangen werden, dass es sich um ein Botnet-Angreifer handelt. Die einzelnen Server sind dann in der Lage die Angreifer IP-Adresse zu sperren und die anderen Server, die noch nicht angegriffen worden sind, können die IP-Adresse ebenfalls sperren um einen Präventivschutz zu erhalten. Um dies zu erreichen muss man mehrere Server vernetzen und die Log-Dateien bzw. die Login-Versuche untereinander teilen.

Ein ähnlicher Ansatz ist das Abfragen der IP-Adresse bei einem "Master-Server". Wenn ein Angreifer versucht sich mit dem Server zu verbinden, überprüft dieser ob die IP-Adresse des Angreifers bei dem Master-Server gelistet ist oder nicht. Wenn eine IP-Adresse bei einem einzelnen Server geblockt wird, dann kann er es dem Master-Server mitteilen. Wenn die IP-Adresse von mehreren Servern geblockt wird, kann der Master-Server sie für alle anderen Server sperren.

Eine andere Art des Master-Servers wäre, dass dieser die einzelnen Log-Dateien der Server auswertet. Dafür sendet jeder Server, jeden einzelnen fehlgeschlagenen Login-Versuch zu dem Master-Server. Wenn der Angreifer bei dem einzelnen Server nicht entdeckt wird, würde er durch das Teilen der Login-Versuche entdeckt werden. Wenn der Master-Server einen Angreifer entdeckt hat, kann er diesen den einzelnen Servern mitteilen, woraufhin die den Angreifer sperren.

[11] siehe: [Ram03]

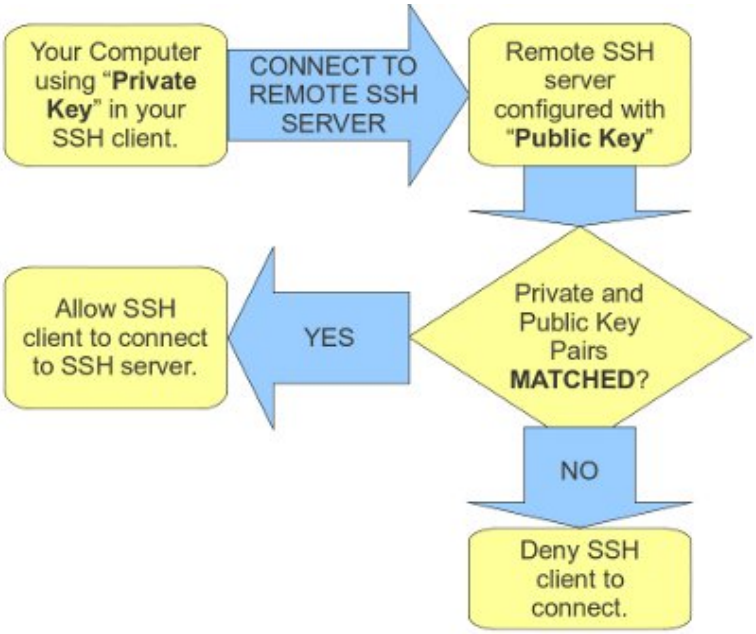

Figure 2.4: SSH-Keys Authentication

Chapter 3

Projekt

3.1 Einleitung Projekt

Die Aufgabe des Projekts bestand darin Secure Shell Verbindungen (kurz SSH) sicherer zu machen und falls Schwachstellen vorhanden sind diese zu beheben.
Um das zu erreichen sollte ich versuchen die SSH-Verbindung zu hacken. Genauer gesagt, sollte ich die Brute-Force-Methode für Angriffe auf einen SSH-Service verwenden.

Mit Hilfe dieser Methode, versuchte ich Benutzernamen und deren Passwörter zu erraten.

Anschließend sollte ich ein Programm erstellen, welches Brute-Force-Angriffe abwehrt.

3.2 Verwendete Programme

Da SSH in C implementiert ist bot es sich an das Projekt ebenfalls in C zu implementieren.

Zudem habe ich "Oracle VM VirtualBox"[1] mit einem installierten Ubuntu System verwendet. Zusätzlich benutzte ich "VMPlayer"[2], da in dieser virtuellen Umgebung Ubuntu stabiler und schneller läuft.

Außerdem verwendete ich einen Zusatz für SSH, da SSH normalerweise keine Konsolenaufrufe mit Passwort erlaubt. Der verwendete Zusatz heißt "sshpass"[3]. Dieser erlaubt eine SSH-Verbindung herzustellen, bei der man den Benutzernamen und das Passwort in einem Befehl an die Konsole übergibt.

Auf meinem Windows 7 Betriebssystem (das Hostsystem) habe ich den WinSSHD Dienst installiert. Dieser ermöglicht mir, mich von meinem Gastsystem mit meinem

[1] VirtualBox: `https://www.virtualbox.org/`
[2] VMPlayer: `http://www.vmware.com/de/products/player/`
[3] webseite sshpass: `http://sourceforge.net/projects/sshpass/`

Hostsystem zu verbinden.
Im Laufe des Projekts wurde auf eine Linux Umgebung umgestellt, da im weiteren Verlauf des Projekts, eine mögliche Brute-Force-Abwehr implementiert werden sollte.

Um die Brute-Force-Methode anwenden zu können, habe ich ein C-Programm geschrieben. Dies erlaubt mir Wörterlisten (Benutzernamen und Passwörter) zu benutzen, um einen automatischen Angriff auf einen SSH Service durchzuführen. Bei erfolgreicher Verbindung wird der Benutzername und das passende Passwort in eine extra Datei geschrieben.
Prof. Dr. Waldvogel hat mir hierfür ebenfalls ein Programm zur Verfügung gestellt, welches in Perl geschrieben ist. Dieses Programm hat den Vorteil, dass es die Wörterlisten schneller abarbeiten kann, da es mehrere Prozesse (und somit Verbindungen) auf einmal starten kann. Dies ist vor allem bei großen Wörterlisten von Vorteil, da man hier der Problematik einer langen Laufzeit der Brute-Force-Methode entgegenwirken kann.

Die verwendeten Wörterlisten von cracklib[4] habe ich aus dem Internet heruntergeladen. Diese werden normalerweise dafür verwendet, um die Stärke von Passwörtern herauszufinden. In meinem Fall verwende ich sie, um die passenden Passwörter zu den Benutzernamen herauszufinden.

Des Weiteren habe ich die BerkeleyDB[5] verwendet um Angreifer in einer Datenbank zu speichern. Der Vorteil einer Datenbank besteht darin, dass mehrere Operationen gleichzeitig darauf ausgeführt werden können. Ebenso vereinfacht sie das Suchen nach bestimmten Einträgen.

Um das Projekt im Internet zu speichern, wurde das bereitgestellte svn-repository der Universität Konstanz verwendet.

3.3 SSH-Block

Das Programm das ich erstellt habe, ist ein Service für einen Linux Server. Nach einer gewissen Anzahl an fehlgeschlagenen Verbindungsversuchen sperrt das Programm die IP-Adresse des Angreifers, sowie den verwendeten Benutzernamen. Es stellt sicher, dass ein möglicher Brute-Force-Angriff abgewehrt wird, da es dem Angreifer keine Verbindung zum Server erlaubt.

3.3.1 Wie wird der Angriff abgewehrt?

Der Brute-Force-Angriff wird abgewehrt, indem man die "auth.log" (welche im Ordner "/var/log/" des Linux Betriebssystems zu finden ist) ausliest. In dieser Datei loggt das Betriebssystem alle Aktionen, die vorgenommen werden. Dies schließt auch erfolgreiche und erfolglose Anmeldungen am SSH-Server mit ein.

[4] cracklib webseite: `http://cracklib.sourceforge.net/`

[5] Berkeley DB: `http://www.oracle.com/technetwork/products/berkeleydb/overview/index.html`, `http://sepp.oetiker.ch/subversion-1.6.4-rp/gsg/C/BerkeleyDB-Core-C-GSG.pdf`

Um diese Datei auszulesen, wird die Datei „gegreppt“(siehe Algrotihmus 1) und man liest alles was mit dem SSH-Server zu tun hat, heraus. Der Befehl "grep" ist ein Programm welches der Suche und Filterung definierter Zeichenketten aus Dateien dient[6]. Genauer gesagt, wird nicht alles ausgelesen, sondern es wird sich dabei auf fehlgeschlagene Login-Versuche beschränkt. Um den Overhead zu reduzieren werden nur die letzten 500 Zeilen ausgewertet, die mit dem SSH-Server zu tun haben. Ich habe die letzten 500 Zeilen gewählt, da man davon ausgehen kann, dass auch bei hoher Auslastung alle Login-Versuche untersucht werden. Ein kleinerer Wert wäre dennoch möglich. Wenn ein Benutzer und eine IP-Adresse 10 Mal oder mehr innerhalb einer gewissen Zeitspanne vorkommen, wird die IP-Adresse in die "iptables" eingetragen. Der Benutzer wird in eine andere Benutzergruppe verschoben und zwar in die Gruppe "blocked", die beim SSH-Server als "DenyGroup" eingetragen ist. Gruppen, die in die "DenyGroup" eingetragen werden, ist es nicht erlaubt sich am Server anzumelden. Diese Funktion wird von openSSH bereitgestellt.

Der gesperrte Benutzer und die IP-Adresse werden nach Ablauf einer gewissen Zeit automatisch freigegeben. Dies erfolgt durch Austragung der IP-Adresse aus der "iptables" und durch Verschiebung des Benutzers in die "unblock" Gruppe, welche beim SSH-Server als "AllowGroup" eingetragen ist.

Algorithm 1 Grep-Funktion

```
1: procedure GREP_LOG
2:     snprintf("grep -niE 'sshd.*failed password|failed password.*sshd|sshd.*failed
   none|failed none.*sshd' /var/log/auth.log |tail 500 >/var/log/ssh_block_grep.txt");
3:
4:     system(grep);
```

3.3.2 Welche Probleme traten beim programmieren auf?

Beim programmieren bin ich auf diverse Schwierigkeiten gestoßen. Im folgendem Abschnitt werde ich diese Probleme spezifizieren und aufzeigen wie sie gelöst wurden.

Das erste Problem, auf das ich gestoßen bin, war das Auslesen der "auth.log" Datei. Das Problem bestand darin, dass ich die Log-Datei auf den wesentlichen Inhalt reduzieren musste. Erfolgreiche Anmeldungen können ignoriert werden, weil sich Benutzer höchst wahrscheinlich nicht abmelden und erneut (allerdings falsch) anmelden. Wenn das dennoch passiert, dann ist es sehr wahrscheinlich, dass die gesetzte Zeit (10 Minuten) abgelaufen ist. In dieser Zeit bleiben Benutzer und IP-Adresse gespeichert. Dieses Problem habe ich gelöst, indem ich die Datei "greppe" und gewisse Parameter übergeben habe. Die Parameter müssen übereinstimmen, damit das Ergebnis angezeigt wird. Diese sind in Algorithmus 1 in Zeile 2 zu sehen. Anschließend wird das Ergebnis an "tail" (tail – gibt nur die letzten "n" Zeilen zurück) übergeben. Das Übergeben an "tail" hat den

[6]grep: [GNU13]

Zweck das Auslesen der Datei zu verringern und somit den "Overhead" zu reduzieren. Das Ergebnis der beiden Befehle wird in einer Datei gespeichert. Diese Datei enthält nun alle fehlgeschlagenen Anmeldungen am SSH-Server, die auf die letzten 500 Anmeldungen begrenzt sind.

Ein weiteres Problem was auftrat war das Auslesen des Datums aus der erzeugten Datei. Dies ist wichtig, da man die Zeit des letzten fehlgeschlagenen Versuchs mit der aktuellen Zeit vergleichen muss. Wenn die Differenz der beiden Zeiten kleiner als 10 Minuten ist, wird die Zeilennummer, der Benutzername, die IP-Adresse und die Zeit in einer Liste gespeichert. Das Problem war, dass das Zeitformat in der Datei als String vorlag. Da ich die Funktion difftime (time_t end, time_t beginning) verwenden wollte, musste ich also den String in das Format "time_t" umwandeln. Nachdem ich den String in einer "struct tm" gespeichert habe, konnte ich diese in das Format "time_t" mit der passenden Funktion umwandeln. Nun konnte ich die aktuelle Zeit mit der Zeit des fehlgeschlagenen Login-Versuches vergleichen und ermitteln, ob dieser älter als 10 Minuten ist.

Anschließend habe ich festgestellt, dass ich eine Struktur benötige, um den Benutzernamen, die IP-Adresse, die zuletzt gelesene Zeit, die Zeilennummer und den Zähler zu speichern. Um dies zu realisieren, habe ich eine einfach verkettete Liste programmiert. Diese Liste enthält einzelne Knoten mit den vorgenannten Informationen. Wenn ich nun die "auth.log" (bzw. die oben erzeugte Datei) auslese, dann speichere ich die ausgelesen Daten in dieser Liste. Falls der Benutzername oder die IP-Adresse schon vorhanden ist, aktualisiere ich die Zeilennummer sowie die Zeit und erhöhe den Zähler um eins. Falls der Benutzername oder die IP-Adresse nicht vorhanden ist, erzeuge ich einen neuen Knoten und füge diesen ans Ende der Liste. Die Aktualisierung bzw. das Hinzufügen geschieht jedoch nur, wenn die ausgelesene Zeit weniger als 10 Minuten von der aktuellen Zeit entfernt ist. Ansonsten werden die Daten verworfen. Wenn der Zähler größer als 10 wird, dann schreibe ich die IP-Adresse in die "iptables" bzw. verschiebe den Benutzer in die Gruppe "blocked". Das Verschieben und das Eintragen in die "iptables" erfolgt mit der Funktion "system()". Die Funktion "system()" führt Terminal Befehle aus. Ebenfalls lege ich in die BerkeleyDB einen entsprechenden Eintrag ab. Wenn die gespeicherte Zeit größer als 10 Minuten wird, dann trage ich die IP-Adresse aus der "iptables" aus bzw. verschiebe den Benutzernamen in die "unblock" Gruppe. Dies geschieht ebenfalls mit der Funktion "system()".

Ein anderes Problem war, dass urpsrünglich die blockierten Benutzernamen und IP-Adressen in einer ".txt" Datei abgespeichert wurden. Später habe ich diese Speicherung umgestellt. Die Benutzernamen und die IP-Adressen werden nun in einer BerkeleyDB abgespeichert.

Eine weitere Schwierigkeit war, die IP-Adressen und die Benutzernamen im System zu sperren. Hierfür hat sich die "iptables" angeboten. Da openSSH Gruppen zum blockieren unterstützt, war das Sperren der Benutzernamen möglich. Um Linux Befehle in einem C Program zu verwenden, stelle ich den passenden String zusammen und übergebe diesen an die Funktion "system()". Der String, welcher für die Verschiebung von Benutzern

```
IP-Address:
snprintf(block, sizeof(block)," iptables -A INPUT -s %s -j DROP"
                                                        ,host);
ret = system(block);

User:
snprintf(block, sizeof(block)," usermod -g blocked %s",host);
ret = system(block);
```

Figure 3.1: String zum Verschieben des Benutzers und eintragen in die iptables

und die Eintragung in die "iptables" verwendet wird, ist in Figure 3.1 zu sehen.

Gegen Ende des Projekts war es einfacher das Programm als ein System-Service zu implementieren, als es komplett abzuändern damit es ein PAM-Modul wird. In Rücksprache mit meinem Tutor wurde das Programm als System-Service implementiert.

Das neu gesetzte Ziel hatte zur Folge, dass ich nun ein "init"-script programmieren musste. Dieses brauche ich, um das Programm als System Service anzumelden und als solches auch zu starten. Daher habe ich ein „init"-Script programmiert, welches für diesen Zweck ausreicht.

Am Ende des Projektes habe ich versucht, das Programm mit Valgrind[7] zu optimieren. Valgrind ist eine Werkzeugsammlung zum Debuggen und zur Fehleranalyse von Programmen. Valgrind hilft einem Benutzer, die Schwachstellen des Programms ausfindig zu machen. Mithilfe von Valgrind war es mir möglich, diverse "Array Index out of Bounds" Fehler und "Memory Leaks" zu beheben. "Memory Leaks" sind Fehler, die zur Folge haben, dass ein Prozess Speicherplatz belegt, diesen aber nicht nutzt oder freigibt. "Array Index out of Bounds" sind Fehler, die verursacht werden, wenn man versucht auf ein Element zuzugreifen, das die Kapazität des Arrays übersteigt.

3.3.3 Weitere Anpassungen und Erweiterungen

Man könnte das Programm noch so verändern, dass es perfomanter läuft. Das bedeutet die Auswertung der erzeugten Datei zu verbessern, welche die fehlgeschlagenen Login Versuche enthält. Eine Überlegung hierfür war, alles auf ein PAM-Modul umzustellen. Dadurch würde man der Problematik der Auswertung entgehen, da man alle relevanten Daten vom PAM-service bereitgestellt bekommt.
Zudem wäre eine Speicherung, die mehr auf die BerkeleyDB ausgelegt ist, sicherlich sinnvoll. Dadurch könnte man auf eine Liste verzichten und alles in mehreren einzelnen Datenbanken abspeichern. Das würde die Performance des Programmes weiter steigern, weil hierdurch eine höhere Parallelität erzielt werden kann. Falls man das Programm

[7]Valgrind: `http://valgrind.org/`

auf eine PAM-basis umstellt, können zeitgleiche Anmeldungen zu Problemenführen. Der Grund ist, dass eine Liste nicht parallel arbeiten kann. Deswegen wäre eine Umstellung auf mehrere BerkeleyDB sinnvoll.

Eine Erweiterung des Programmes wäre die Auswertung der verwendeten Passwörter, natürlich auf verschlüsselter Ebene. Da Brute-Force-Angriffe oft komplett unterschiedliche Passwörter verwenden, was zu einer unterschiedlichen Verschlüsselung führt, kann man die Toleranz der fehlgeschlagenen Verbindungsversuche für Angreifer absenken. Für Benutzer kann man eine Höhere erzielen, also mehrere Versuche zu lassen. Dies würde dazu führen, dass Benutzer, nicht so schnell blockiert werden würden, wenn sie ihr Passwort vergessen haben. Da "normale" Nutzer in der Regel sehr ähnliche Passwörter versuchen (einen Buchstaben weglassen bzw. Einen hinzufügen oder Einen ersetzen), führt dies zu einer ähnlichen Verschlüsselung und man würde die Schwelle der fehlgeschlagenen Verbindungsversuche beibehalten oder unter Umständen sogar etwas erhöhen.

Ein anderer Ansatz für eine Erweiterung wäre, dass man "GeoLocations" berücksichtigt. Da es Länder mit verschiedenen häufigen Auffälligkeiten gibt[8], wie zum Beispiel erhöhtem SPAM-Mail Versand, kann man diesen oder genauer gesagt Ihren IP-Adressen eine geringere Bannschwelle beimessen. Solche IP-Adressen würden für einen fehlgeschlagenen Login-Versuch stärker bestraft werden, als eine IP-Adresse die nicht auffällig ist.

Eine ebenfalls mögliche Erweiterung wäre zum Beispiel die Anbindung an eine DNSBL (Domain Name System Block List). Dies wäre sinnvoll, da man hier gewisse IP-Adressen sperrt, die für gewisse Auffälligkeiten bekannt sind. Die Folge wäre eine Entlastung des Server und ein vorzeitiges Blocken von auffälligen IP-Adressen.

[8] siehe: [Ken10]

Chapter 4

Erweiterung Projekt

Dieses Kapitel wird sich mit der Erweiterung des Bachelor-Projekts befassen. Wie in Kapitel 3.3.3 "Weiterer Anpassungen und Erweiterungen" bereits angesprochen habe ich das Programm perfomanter gestaltet. Die einzelnen Erweiterungen werden im Kapitel 5 "Bachelor-Thesis" genauer erklärt.

Eine höhere Performance habe ich erzielt, indem ich das gesamte Projekt auf "Pluggable Authentication Modul" (kurz PAM) umgestellt habe. PAM ist eine Softwarebibliothek, die eine Programmierschnittstelle für Authentifizierungsdienste bereitstellt. Dies ermöglicht Programmen, unabhängig von dem darunterliegenden Authentifizierungsschema implementiert zu werden. Das Programm läuft performanter, da nicht mehr dauerhaft die "auth.log" Datei des Betriebssystems ausgewertet wird. Über PAM erhält man direkte Informationen über den verwendeten Benutzer und dessen IP-Adresse. Durch die Umstellung läuft es "schneller", weil es nur aufgerufen wird, wenn es auch notwendig ist. Dies geschieht, sobald ein Benutzer versucht sich am System anzumelden. Der Unterschied von meiner Bachelor-Thesis zu meinem Bachelor-Projekt besteht darin, dass das Programm dauerhaft im Hintergrund läuft und nur verwendet wird, wenn es gebraucht wird. Dadurch werden die Resourcen des Servers weniger belastet.

Weiterhin habe ich die verwendeten Passwörter des Benutzers auf verschlüsselter Basis ausgewertet. Das bedeutet, wenn ein Benutzer sein Passwort eingibt wird es verschlüsselt und anschließend mit der "shadow" Datei abgeglichen. Wenn die Verschlüsselung übereinstimmt, dann wird dem Benutzer Zugriff gewährt. Falls das Passwort nicht übereinstimmt, wird das Programm durchlaufen - näheres in Kapitel 5 "Bachelor-Thesis".

Ein weiterer Ansatz den ich verfolgt habe, ist die Verwendung von "Geolocation" Daten. Das kommt daher, dass es Länder bzw. IP-Adressen gibt, die eine erhöhte Auffälligkeit besitzen. Das bedeutet zum Beispiel das Bereitstellen von vielen Bot Netzwerken oder das Versenden von SPAM-Mails. Aus dem Sicherheitsreport von Symantec geht zum Beispiel hervor, dass sich 2006 26% aller Bots in China befanden[1]. Durch die Auswertung des Standortes des Benutzers, kann eine höhere oder niedrigere Bannschwelle

[1] Symantec-Report [Gmb]

erreicht werden, wodurch der Server besser geschützt wird.

Das Programm wurde an "SPAMHAUS" angebunden. Wenn ein Benutzer versucht sich anzumelden, wird überprüft ob seine IP-Adresse schon bei SPAMHAUS gemeldet ist oder nicht. Falls sie gemeldet wurde, wird dem Benutzer kein Zugriff zu dem Server gewährt.

Spamhaus ist eine DNSBL (Domain Name System Block List), also eine Datenbank, welche IP-Adressen listet, die dabei beobachtet wurden SPAM-Mails zu verschicken, Webseiten von Spammer zu hosten, Dienste an Spammer anzubieten oder ähnliches. Manchmal werden sie auch RBLs (Realtime Blackhole Lists), "blocklists" oder "blacklists" genannt.

Eine zusätzliche Erweiterung ist die Anbindung an einen Master-Server, den ich implementiert habe. Er besitzt eine Datenbank, die alle bisher geblockten IP-Adressen der einzelnen Server enthält. Jeder einzelne Server sendet bei einer Sperrung einen kurzen Befehl an den Master-Server. Dadurch wird die gesperrte IP-Adresse in die Datenbank des Master-Servers aufgenommen. Versucht sich nun ein Benutzer an einem Server anzumelden, sendet dieser eine Anfrage an den Master-Server, ob die IP-Adresse bei diesem gelistet ist, oder nicht. Falls diese gelistet ist, wird die Verbindung zum Benutzer getrennt. Das erhöht die Sicherheit aller Server, welche an den Master-Server angeschlossen sind. Angriffe auf andere Server wurden bereits gelistet und nicht jeder einzelne Server muss den Angreifer blockieren.

Chapter 5

Bachelor-Thesis

Dieses Kapitel beschäftigt sich mit dem Programm, das ich in meinem Bachelor-Projekt erstellt habe. Dabei werde ich auf die verschiedenen Veränderungen und Erweiterungen eingehen und die einzelnen Teile des Programms näher erläutern.

5.1 Hauptfunktion Authentifizierung

Zuerst gehe ich auf den Hauptteil des Programms ein, welches in zwei Hauptfunktionen aufgeteilt ist. Eine, die sich um die Authentifizierung kümmert, und die Andere, die verwendet wird, um die Session zu erstellen. In diesem Kapitel gehe ich auf die erste Hauptfunktion ein. Sie wird mit der Funktion "PAM_EXTERN int pam_sm_authenticate(pam_handle_t *pamh, int flags, int argc, const char **argv)" aufgerufen. Die Zweite wird in Kapitel 5.2 erläutert. Die Funktion pam_sm_authenticate ist eine von der PAM-API bereitgestellte Funktion. Die Funktionalität wird durch den Benutzer bestimmt, daher wurde sie in meinem Programm entsprechend angepasst. Der Funktionsparameter pam_handle_t *pamh ist eine leere Struktur. Das Programm kann bzw. muss über Zugriffsfunktionen, also "Getter- und Setter-Funktionen", auf den Inhalt dieser Struktur zugreifen.

In der pam_sm_authenticate Funktion werden zuerst alle Variablen initialisiert. Dazu gehören die einzelnen Datenbanken, die Struktur "info" und diverse Hilfsvariablen. Anschließend werden alle Datenbanken geöffnet. Dabei handelt es sich um verschiedene Datenbanken. Die Erste, die den Status der Benutzer bzw. IP-Adressen enthält, die Zweite die die Zeit des letzten fehlgeschlagenen Login-Versuches für den Benutzer bzw. IP-Adresse enthält, und die Dritte Datenbank enthält den Zähler für die fehlgeschlagenen Login-Versuche. Es kommen noch zwei weitere Datenbanken dazu. Eine Datenbank die alle Länder mit einem Wert bereit hält und eine, die die Zeit der letzten fehlgeschlagenen Login-Versuche speichert. Sollten diese nicht vorhanden sein, werden diese neu erstellt und anschließend geöffnet. Die Struktur "info" habe ich erzeugt, damit ich alle wichtigen Informationen, wie Host (IP-Adresse), Benutzer, die Datenbanken und das Passwort nicht einzeln übergeben muss, sondern alle gebündelt übergeben kann.

Um den Inhalt der pam_handle_t Struktur auszulesen werden nach dem Öffnen der Datenbanken die einzelnen Zugriffsfunktionen aufgerufen. Die einzelnen Variablen werden in die Struktur "info" geschrieben, damit sie dauerhaft vorhanden sind. Dadurch muss nicht jedes mal erneut die Zugriffsfunktion aufgerufen werden. In Algorithmus 2 erkennt man den Pseudoalgorithmus für das Öffnen der Datenbanken und die Aufrufe der einzelnen Zugriffsfunktionen.

Algorithm 2 Hauptfunktion authenticate Teil 1

```
1: procedure SM_AUTHENTICATE_SM(pam_handle_t *pamh, int flags, int argc, const
   char **argv)
2:     repeat
3:         dbopen(name)
4:     until all databases are open
5:
6:     if pam_get_item(PAM_USER) ≠ PAM_SUCCESS then
7:         goto fail
8:     else if pam_get_item(PAM_SERVICE) ≠ PAM_SUCCESS then
9:         goto fail
10:    else if pam_get_item(PAM_RHOST) ≠ PAM_SUCCESS then
11:        goto fail
12:    else if pam_get_item(PAM_AUTHTOK) ≠ PAM_SUCCESS then
13:        goto fail
14:
15:    parse_conf(abpf_info)
16:    ret = check_server(abpf_info)
17:    if ret == 0 then
18:        command(host,"add")
19:        return PAM_AUTH_ERR
20:
21:    go on part 2
```

Danach wird überprüft, ob die IP-Adresse beim Master-Server schon gelistet ist. Falls sie gelistet ist, wird die IP-Adresse übernommen und in die "iptables" eingetragen. Dadurch ist es der IP-Adresse nicht mehr möglich, sich mit dem Server zu verbinden - zu sehen im Algorithmus 4. Die Funktionen "simple_xor()" und "simple_decrypt()" sind Verschlüsselungs- bzw. Entschlüsselungsfunktionen. Um diese Funktionen nutzen zu können ist ein Diffie-Hellman-Schlüsselaustausch[1] notwendig, um ein "shared secret" auf dem Server und dem Master-Server zu erzeugen. Der Schlüsselaustausch erfolgt dabei über einen unsicheren Kanal, bei dem sich die beiden Partner Nachrichten zuschicken. Aus diesen werden das "shared secret" berechnet. Mit Hilfe des "shared secret" wird

[1]Diffie-Hellman-Schlüsselaustausch: `http://de.wikipedia.org/wiki/Diffie-Hellman-Schl%C3%BCsselaustausch`

Algorithm 3 Hauptfunktion authenticate Teil 2

```
1: procedure SM_AUTHENTICATE_SM(pam_handle_t *pamh, int flags, int argc, const
   char **argv)
2:     proceeding part 1
3:
4:     ret = get_pw(abpf_info)
5:     if ret == 0 then
6:         ret = encrypt_pw(abpf_info)
7:     if ret == 2 then
8:         return PAM_SUCCESS
9:
10:    ret = check_rbls(abpf_info)
11:    if ret == 1 then
12:        commandhost,"add"
13:        return PAM_AUTH_ERR
14:
15:    check_unbans(abpf_info)
16:    parse_list(abpf_info)
17:    err = check_attempt(abpf_info)
18:
19:    repeat
20:        dbclose(name)
21:
22:    until all databases are closed
23:    if err == -1 then
24:        return PAM_AUTH_ERR
25:    else
26:        return PAM_SUCCESS
```

eine exklusive XOR-Bitoperation durchgeführt, um die IP-Adresse zu verschlüsseln. Die Entschlüsselungsfunktion ist also das Umkehren der Verschlüsselung, ebenfalls mit einer exklusiven XOR-Bitoperation. Algorithmus 5 zeigt die Verschlüsselungsfunktion.

Algorithm 4 Abfragen des Master-Servers

```
 1: procedure CHECK_SERVER(info *info)
 2:     read_config
 3:     get p,q and mod
 4:     calculate d, with a = rand()
 5:     a = fmod(a,mod+1)
 6:     d = pow(g,a)
 7:     d = fmod(d,p)
 8:
 9:     create Socket
10:     set serv_addr
11:
12:     connect(serv_addr)
13:
14:     send(p,g,d)
15:     receive answer b
16:     c = pow(b,a)
17:     c = fmod(c,p)
18:
19:     encrypted = simple_xor()
20:     send(encrypted)
21:
22:     receive answer
23:     decrypt = simple_decrypt(answer)
24:
25:     if strstr(decrypt, "NOT") != NULL then
26:         return 2
27:     return 0
```

Wenn die IP-Adresse des Benutzers noch nicht gelistet ist, wird überprüft wie ähnlich das eingegebene Passwort mit dem richtigen Passwort ist. Dabei werden die ähnlichen Zeichen ersetzt und verschlüsselt, zum Beispiel eine "3" anstatt ein "e". Das verschlüsselte richtige Passwort des Benutzers erhält man mit der Funktion "getspnam_r()". Diese liest dafür die "shadow"-Datei des Betriebssystems aus und speichert, falls es diesen Benutzer gibt, das Ergebnis. Aus dem Ergebnis wird nun die ID und der Salt ermittelt. Für die Verschlüsselung des ähnlichen Passworts wird die Funktion "crypt_r(const char *key, const char *salt)" verwendet, siehe Algorithmus 6. Der Unterschied dieser Funktion zu der normalen "crypt()" Funktion besteht darin, dass diese threadsicher ist. Das verschlüsselte Passwort wird je nach ID, welche sich im "Salt" befindet, verschlüsselt.

Algorithm 5 Verschlüsselung mit XOR

```
1: procedure SIMPLE_XOR(const char *msg, int key)
2:     size_t msglen = strlen(msg)
3:     size_t keylen = sizeof(key)
4:     char * encrypted
5:     int i
6:
7:     for i = 0, i <msglen, i++ do
8:         encrypted[i] = msg[i] ^key
9:     encrypted[msglen] = '\0'
10:    return encrypted
```

Es kann dabei in MD5 oder SHA-512 verschlüsselt werden. Das verschlüsselte Passwort wird nun mit dem verschlüsselten Passwort in der "shadow" Datei abgeglichen. Wenn das Passwort stimmt, wird das Programm beendet und der Zähler wird nicht erhöht. Hier könnte man auch eine andere Variante wählen, wie zum Beispiel eine nicht ganz so hohe Erhöhung des Zählers, um den Faktor 0.5. Damit würde sichergestellt werden, dass es ein Fehlversuch ist und dieser auch "bestraft" wird. In dieser Funktion wird nicht überprüft, ob das Passwort richtig ist, sondern nur nach seiner Ähnlichkeit. Das bedeutet also, wenn das Passwort richtig ist wird in dieser Funktion das Programm nie beendet. Stimmt das Passwort nicht überein, wird das Programm weiter abgearbeitet. Die Überprüfung des Passwortes wurde in das Modul "pam_unix.so" ausgelagert. Dieses Modul wird nach der Abarbeitung des Programms aufgerufen und überprüft, ob das Passwort richtig ist. Wenn es richtig ist, wird die zweite Hauptfunktion 5.2 aufgerufen.

Nachdem das Passwort des Benutzers nicht ähnlich mit dem richtigen Passwort ist, wird die "list.txt" Datei abgearbeitet. Die Funktion wird nur aufgerufen wenn die Datenbank für die Liste nicht schon existiert. Möchte man die Datenbank aktualisieren muss man vorher die "List.db" Datei löschen. In Algorithmus 7 ist der Quellcode für die Abarbeitung zu finden. In dieser Datei befinden sich die Abkürzungen für die meisten Länder nach der ISO-3166-1-Kodierliste. Neben dem Land befindet sich zusätzlich ein "value". Dieser Wert bestimmt, wie viele fehlgeschlagene Login-Versuche eine IP-Adresse aus dem jeweiligen Land erhält. Zum Beispiel kann eine IP-Adresse mit dem Wert 1 maximal 9 Versuche durchführen, um sich am Server anzumelden. Danach ist die IP-Adresse gesperrt. Ein Land mit einem Wert von 5 hingegen hat maximal einen Versuch. Der Grund warum solche Länder dennoch einen Versuch bekommen liegt darin, dass man einen Zugriff auf seinen Server erhält, wenn man sich im Ausland aufhält. Würde man keinen Versuch zulassen, wäre es nicht möglich sich auf seinen Server einzuloggen. Besonders dann, wenn man sich in einem Land mit dem Wert 5 aufhält. Es ist zu beachten, dass nicht alle Benutzer aus einem Land mit einem hohen Wert eine Bedrohung für den Server darstellen. An dieser Stelle ist auch zu erwähnen, dass man nicht unterscheidet ob es eine geteilte IP-Adresse ist oder nicht. Wenn man sich in einem Hotel aufhält, haben alle Gäste die gleiche IP-Adresse. Versucht sich jetzt einer dieser Gäste auf dem

Algorithm 6 Verschlüsselung des Passwortes

```
 1: procedure ENCRPYT_PW(info *info)
 2:     getspnam_r(info-¿user,sp,pwdbuffer,sizeof(pwdbuffer),encrypt)
 3:     if encrypt is not NULL then
 4:         hashcopy = strdup(encrypt→sp_pwdp)
 5:         hash = strdup(encrypt→sp_pwdp);
 6:         nr = strtok(hash,"$")
 7:         salt = strtok(NULL, "$")
 8:         hashgen = strtok(NULL, "$")
 9:
10:         strcat(puffer, "$")
11:         strcat(puffer, nr)
12:         strcat(puffer, "$")
13:         strcat(puffer, salt)
14:         strcat(puffer, "$")
15:
16:         strncpy(buffer,info→pw,sizeof(buffer)-1);
17:         if buffer is not NULL then
18:             for i <strlen(buffer); i++ do
19:                 for j <sizeof (search); j++ do
20:                     if buffer[i] == search[j] then
21:                         buffer[i] = replace[j]
22:                         crypted = crypt_r(buffer,puffer,data);
23:                         if strcmp(crypted, hash) is 0 then
24:                             ret = 0
25:                             break
26:                         buffer[i] = search[j]
27:                     j = 0
28:                     if ret is 0 then
29:                         break
30:
31:         if ret == 0 then
32:             return 2
33:         else
34:             return 1
```

Server anzumelden und die Anmeldung schlägt fehl, werden alle Benutzer, die die gleiche IP-Adresse verwenden, ebenfalls gesperrt. Eine Möglichkeit dieses Problem teilweise zu umgehen wäre die Verwendung von "OS fingerprinting". Darunter versteht man die Erkennung von Betriebssystemen. Das Erkennen geschieht durch die unterschiedlichen TCP/IP-Protokollstapel-Implementierungen der Systeme und den verschiedenen Einstellungen im Header der Netzwerkpakete. Es gibt zwei verschiedene Methoden zur Bestimmung: aktiv und passiv. Bei der passiven Methode wird der ablaufende Datenverkehr bewertet und analysiert. Beispielsweise kann dadurch das Betriebssystem eines Besuchers beim öffnen einer Webseite bestimmt werden. Die aktive Methode zeichnet sich dadurch aus, dass man Datenpakete an das Zielsystem sendet und die Antwort analysiert. Durch die Verwendung von "fingerprinting" wäre es möglich eine genauere Abstufung von IP-Adressen zu erhalten und dem Problem von geteilten IP-Adressen entgegenzuwirken. Andererseits würde dies dazu führen, dass man Angreifern eine Möglichkeit bietet die Sicherheit des Servers teilweise zu umgehen und diesen mehrere Versuche erlaubt. Dadurch gefährdet man die Sicherheit des Servers.

Algorithm 7 Abarbeiten der list.txt

```
1: procedure PARSE_LIST(info *info)
2:     file = fopen("path to file","a+")
3:
4:     while fgets(buffer,sizeof(buffer),file) is not NULL do
5:         sscanf(buffer, "%[^'='']=%d", host, val)
6:         if val == 0 then
7:             val = 3
8:         db_addtolist(info, host, val)
9:         val = 0
```

Wurde die Liste abgearbeitet, sendet das Programm eine Anfrage an "SPAMHAUS", was in Algorithmus 8 zu sehen ist. Hierbei überprüft SPAMHAUS, ob die IP-Adresse bei ihnen schon gelistet ist. Wenn man möchte, kann man hier noch weitere DNSBLs hinzufügen, welche in der entsprechenden Config-Datei eingetragen werden können. Getestet wurden die xbl-, sbl- und pbl-Datenbanken. Die Abfrage der verschiedenen DNSBLs habe ich nicht selber implementiert. Hierfür habe ich den Teil, der für das Programm relevant ist, aus einem bereits existierenden Programm entnommen. Dieses Programm heißt "rblcheck" [2] und basiert auf der GNU-Lizenz. Das Programm wurde auf der "udns" Bibliothek aufgebaut und ist ursprünglich ein Kommandozeilenprogramm. Die "udns" Bibliothek ist eine DNS Bibliothek und implementiert einen "Stub DNS Resolver". Stub-Resolver arbeiten rekursiv. Dies heißt, dass sie ihren Nameserver anfragen. Wenn dieser die gewünschte Information nicht besitzt, fragt der Nameserver bei weiteren Servern nach der gewünschten Information nach. Stub-Resolver überlassen die Arbeit also dem Nameserver. Für mein Programm hat der Teil, der die DNSBL Datenbanken abfrägt ausgereicht. Daher habe ich auf den Rest des Programms verzichtet.

[2] Programm-URL `https://sourceforge.net/projects/rblcheck/`

Algorithm 8 Abfragen der SPAMHAUS Datenbank

```
1: db = pblSpam, xblSpam, sblSpam
2: PacketSize = 512
3: part ip into a.b.c.d
4:
5: for all db do
6:     snprintf(domain, d.c.b.a.db)
7:     res_init()
8:     len = res_query(domain, answer, PacketSize)
9:
10:    if len == -1 then
11:        return NULL
12:
13:    if len >PacketSize then
14:        len = res_query(domain,answer,len)
15:        if len == -1 then
16:            return NULL
17:
18:        result[0] = "\0"
19:        return result
```

Ist die IP-Adresse nicht bei einer DNSBL gelistet wird überprüft, ob es IP-Adressen und Benutzer gibt, die das Zeitlimit überschritten haben. Diese werden dann gegebenenfalls wieder freigegeben, so dass sie sich mit dem Server wieder verbinden können. Diese Art der Entsperrung reicht nicht aus. Das Problem hierbei ist, dass sie sich nicht selbst entsperren können. Das Programm wird aufgrund der Sperrung nicht ausgeführt. Es ist aber möglich, dass andere Benutzer oder IP-Adressen andere Benutzer und IP-Adressen entsperren. Das Freigeben von Benutzern und IP-Adressen ist in Algorithmus 9 zu sehen. Dabei wird die Datenbank der gesperrten Benutzer bzw. IP-Adressen durchlaufen und bei jedem Eintrag überprüft, ob die eingetragene Zeit länger als 10 Minuten von der aktuellen Zeit zurückliegt. Wenn die eingetragene Zeit älter ist, dann wird der entsprechenden Eintrag aus den Datenbanken gelöscht. Zu beachten ist, dass hierbei IP-Adressen nicht aus der "iptables" gelöscht bzw. Benutzer nicht in die Gruppe "unblock" verschoben werden . Diese Art der Entsperrung dient ausschließlich dazu, die Einträge aus den Datenbanken zu entfernen.

Wird ein Benutzer oder eine IP-Adresse gesperrt, wird dies zusätzlich in einer von 10 Dateien festgehalten. In welcher Datei gespeichert wird, hängt von der Bannzeit ab. Diese beträgt 10 Minuten. Die 10 Dateien repräsentieren dabei die Minuten der aktuellen Uhrzeit. Wenn ein Benutzer um "13:25" gesperrt werden würde, dann würde dies in der fünften Datei eingetragen werden. Dies wird in Figure 5.1 noch einmal verdeutlicht.

Für diese zehn Dateien habe ich zusätzlich ein Programm implementiert, welches in Kapitel 5.6 beschrieben ist. Dieses wird mit Hilfe des CRON-Jobs jede Minute aus-

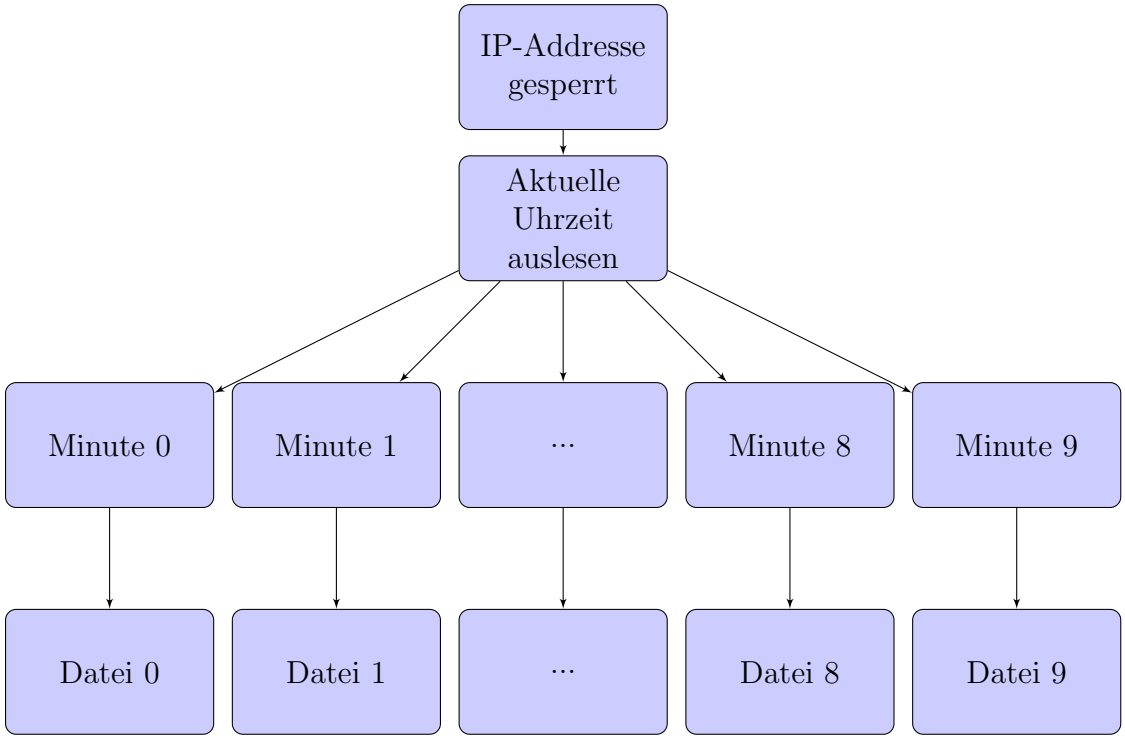

Figure 5.1: Auslesen der Uhrzeit und Eintragen in die Datei

geführt. Stehen Benutzer oder IP-Adressen in der entsprechenden Datei, werden diese entfernt und entsperrt. Um Overhead zu vermeiden, habe ich mich dazu entschieden, nicht alle Bannzeiten in eine Datei zu schreiben, sondern in mehrere kleinere Dateien. Dies hat den Vorteil, dass das Programm an Performance und Geschwindigkeit gewinnt. Die Aufteilung in zehn Dateien hat den Hintergrund, dass das Hauptprogramm eine Bannzeit von zehn Minuten besitzt.

Nachdem die Überprüfung der Entsperrungen erfolgt ist, wird der Login-Versuch ausgewertet (siehe dafür auch Figure 5.2). Wie in Algorithmus 10 zu sehen, wird dafür in der Hauptfunktion die Funktion "check_attempt" aufgerufen. In dieser Funktion wird mit Hilfe der Funktion "check_hostdb" der Login-Versuch ausgewertet. Hierfür wird überprüft, ob die IP-Adresse schon gelistet und geblockt ist. Dieser Fall dürfte niemals eintreffen. Falls der Benutzer oder die IP-Adresse gesperrt ist, wird das Programm nie ausgeführt. Ich habe mich dazu entschlossen diese Abfrage zu behalten. Falls das Programm dennoch ausgeführt wird bricht es an dieser Stelle ab. Das Programm könnte zum Beispiel ausgeführt werden, wenn es bei der Sperrung zu Problemen gekommen ist und der Benutzer bzw. die IP-Adresse in der Datenbank eingetragen aber im System nicht gesperrt ist. Die Abfrage kann also unter gewissen Umständen den Overhead des Programms reduzieren.

Wenn es keinen Eintrag in den Datenbanken gibt, dann wird der fehlgeschlagene

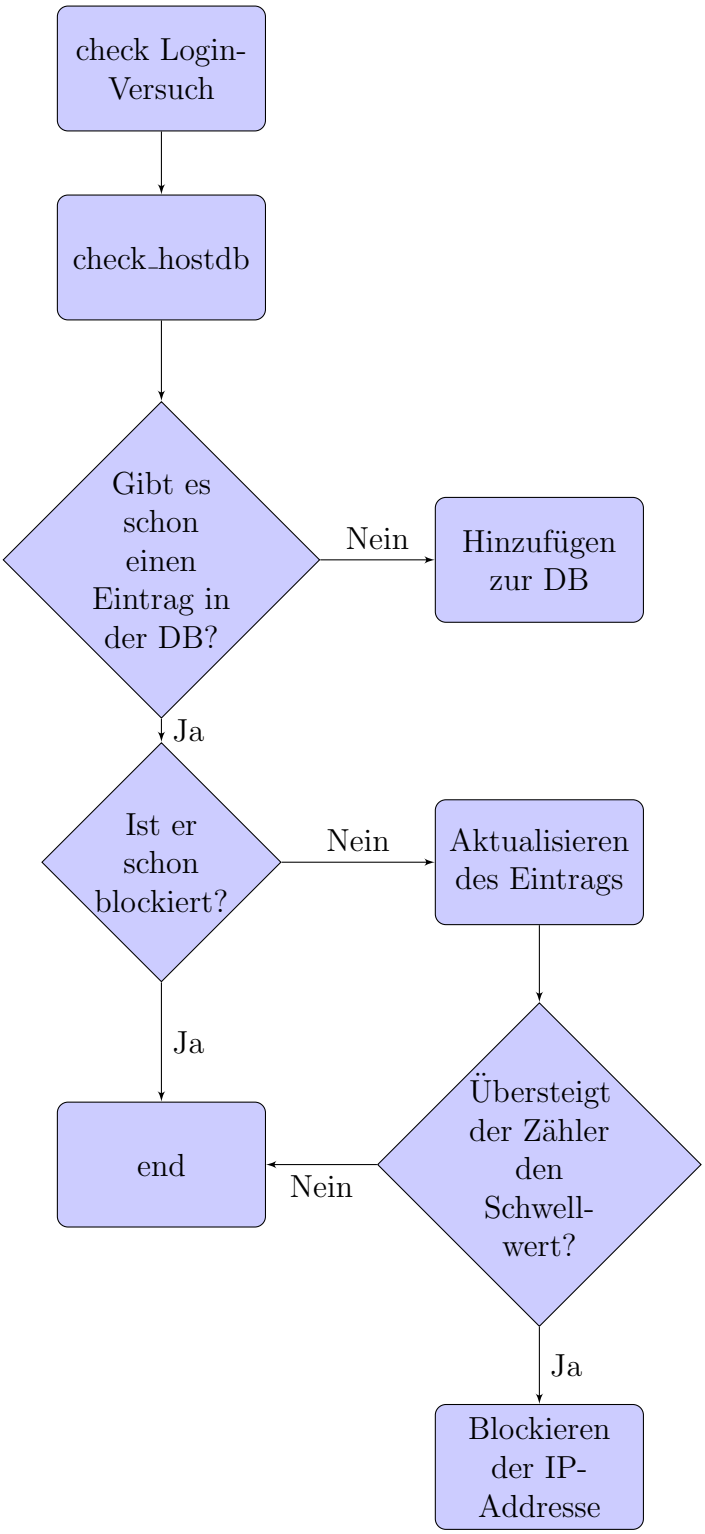

Figure 5.2: Überprüfen des Login-Versuches

Algorithm 9 Überprüfen der zu entsperrenden Benutzer bzw. Hosts

```
1: procedure CHECK_UNBANS(info *info, const char *isHost)
2:     if Host then
3:         db = host database
4:     else
5:         db = user database
6:
7:     tnow = time(NULL)
8:     db → cursor(db,NULL,dbc,0)
9:
10:    while ret = dbc → get(dbc, key, data, DB_NEXT) == 0 do
11:        tban = data.data
12:        Host = key.data
13:        diff = difftime(tnow, tban)
14:
15:        if diff ≥ 600 then
16:            db_removestate(info, Host, isHost)
17:            db_removetime(info, Host, isHost)
18:            db_removecount(info, Host, isHost)
```

Login-Versuch zu den entsprechenden Datenbanken hinzugefügt und das Programm beendet.

Falls uns die Funktion "check_hostdb" zurückgibt, dass es bereits einen Eintrag in den Datenbanken gibt und dieser noch den Status "entsperrt" hat, dann werden die entsprechenden Einträge in den Datenbanken aktualisiert. Beim Aktualisieren des Zählers wird ebenfalls festgestellt, ob der Benutzer oder die IP-Adresse gesperrt wird oder nicht. Wenn es zu einer Sperrung aufgrund der Überschreitung der Bannschwelle kommt, dann wird der Eintrag in den Datenbanken aktualisiert und die IP-Adresse in die "iptables" hinzugefügt. Der Benutzer wird in die Gruppe "blocked" verschoben. Der Zähler für einen Eintrag wird also immer erhöht, auch bei einem erfolgreichen Login-Versuch. Wenn das Modul "pam_unix.so" das Passwort als richtig identifiziert hat, dann wird der Zähler verringert und die Zeit mit der Zeit des letzten fehlgeschlagenen Login-Versuches überschrieben. Wurde der Benutzer bzw. die IP-Adresse durch das Überschreiten des Schwellwertes gesperrt wird dies wieder rückgängig gemacht. Das passiert in der zweiten Hauptfunktion des Programms. Wie das Sperren genau funktioniert, wird im Unterkapitel 5.3 "Sperrung von Benutzer bzw. Hosts" erläutert.

Abschließend werden in der Hauptfunktion die einzelnen Datenbanken geschlossen. Das muss durchgeführt werden, da sonst die Daten nicht permanent auf die Festplatte geschrieben werden. Das bedeutet somit ein Verlust der Daten.

Wenn alles in Ordnung ist, gibt das Programm "PAM_SUCCESS"[3] zurück. Das

[3]PAM-Tutorial [Pol13]

Algorithm 10 Auswertung des Login-Versuches

```
 1: procedure CHECK_ATTEMPT(info *info)
 2:     tm = time(NULL)
 3:
 4:     if user not NULL && service not NULL && host not NULL then
 5:         err = check_hostdb(info,"yes")
 6:
 7:         if err == -1 then
 8:             goto user
 9:         if err == 0 then
10:             dbadd(info, tm ,YES)
11:         if err == 1 then
12:             dbupdate(info, tm, YES)
13:         user:
14:         err = check_hostdb(info,"no")
15:
16:         if err == -1 then
17:             return -1
18:         if err == 0 then
19:             dbadd(info, tm, NO)
20:         if err == 1 then
21:             dbupdate(info, tm, NO)
```

Programm ist also erfolgreich beendet worden. Danach wird das Modul "pam_unix.so" aufgerufen. Dieses überprüft das eingegebene Passwort. War das Passwort falsch, endet der Login-Versuch an dieser Stelle. Ansonsten wird die zweite Hauptfunktion 5.2 des Programms aufgerufen.

5.2 Hauptfunktion Session

Die zweite Hauptfunktion arbeitet den Session Teil des PAM-Moduls ab und wird nur aufgerufen, wenn das "pam_unix.so" Modul erfolgreich beendet wurde. Da bekannt ist, dass es sich um einen erfolgreichen Login handelt, wird in dieser Funktion der Zähler entsprechend verringert und die letzte Login-Zeit in die Datenbank geschrieben. Dabei werden, wie in Algorithmus 11, die Datenbanken geöffnet und mit Hilfe der Zugriffsfunktionen die PAM-Items ausgelesen. Nach dem Auslesen wird die Funktion "check_count(info,isHost)" 12 aufgerufen. Diese Funktion verringert den Zähler für den spezifischen Benutzer und die IP-Adresse wieder um den entsprechenden Betrag. In Algorithmus 13 ist zu sehen, wie die Verringerung des Zählers von statten geht. Der Betrag, um welchen verringert wird, richtet sich dabei nach dem Land oder nach der Eingabe in der "user.conf" Datei. Ist kein Eintrag in der Datei vorhanden wird der Zähler um 1 verringert. Das Zurücksetzen der Zeit erfolgt in der Funktion "decreasetime(info,isHost)" und wird mit Hilfe einer Datenbank realisiert. Dabei wird in der Hauptfunktion des "authentication" Moduls bei der Erneuerung der Zeit die alte Zeit in eine Datenbank "Oldtime" geschrieben. Da es ein erfolgreicher Login ist, kann die Zeit aus der "Oldtime" Datenbank wieder in die "Zeit" Datenbank geschrieben werden und somit ist die letzte fehlgeschlagene Login Zeit wieder in der Datenbank vorhanden.

5.3 Sperrung von Benutzer bzw. Hosts

In diesem Teil gehe ich darauf ein, wie die Sperrung von Benutzern und IP-Adressen (Hosts) durchgeführt wird.

Wie in Abschnitt 5.1 bereits erklärt wurde findet eine Sperrung immer über eine Aktualisierung des Zählers statt. Das Aktualisieren des Zählers ist in Algorithmus 14, Zeile 11, zu sehen. Wenn der Zähler eines Benutzers oder einer IP-Adresse den festgelegten Schwellwert überschreitet, wird der Benutzer oder die IP-Adresse blockiert. Das Blockieren läuft dabei wie folgt ab: zuerst wird der Zähler des entsprechenden Eintrags erhöht (Algorithmus 16). Um wie viel der Zähler erhöht wird, ist dabei von der Herkunft der IP-Adresse abhängig. Bei Benutzern wird der Wert entsprechend der "user.conf" Datei durchgeführt. Die Herkunft der IP-Adresse wird dabei mit Hilfe der Funktion "geo_ip()" ermittelt. Da eine Liste mit Ländern vorhanden ist und diesen ein Wert vom Systemadministrator zugeordnet wurde, wird der Wert des Landes aus der Datenbank ausgelesen. Anschließend wird der Zähler der IP-Adresse mit der Formel $c = 10 * value + c$ erhöht. In der Formel steht "c" für den vorherigen Zählerwert und "value" für den Wert des Lan-

Algorithm 11 Zweite Hauptfunktion

```
1: procedure PAM_SM_OPEN_SESSION(pam_handle_t *pamh, int flags, int argc, const
   char **argv)
2:     repeat
3:         dbopen(name)
4:     until all databases are open
5:
6:     if pam_get_item(PAM_USER) ≠ PAM_SUCCESS then
7:         goto fail
8:     else if pam_get_item(PAM_SERVICE) ≠ PAM_SUCCESS then
9:         goto fail
10:    else if pam_get_item(PAM_RHOST) ≠ PAM_SUCCESS then
11:        goto fail
12:    else if pam_get_item(PAM_AUTHTOK) ≠ PAM_SUCCESS then
13:        goto fail
14:
15:    check_count(abfp_info)
16:    repeat
17:        dbclose(name)
18:    until all databases are closed
19:    return PAM_SUCCES
```

des. Die Zahl "10" ist nur ein Wert um beim Testen des Programms den Unterschied zwischen Benutzer und IP-Adresse besser zu sehen.

Nach dem der Zähler erhöht wurde, überprüft das Programm ob der Schwellwert überschritten wurde. Hierfür sind mehrere Abfragen verwendet worden. Bei den verschiedenen Abfragen wird mit der Funktion "getvalue()" der Wert des Herkunftslandes ermittelt. Dies ist notwendig, da die Anzahl der verfügbaren maximalen Versuche von der Herkunft abhängig ist. Ist der richtige Wert gefunden, überprüft das Programm ob der neue Zähler den Schwellwert erreicht oder überschreitet. Ist dies der Fall, wird überprüft ob der Login mehr als 10 Minuten zurückliegt. Dies sollte nie geschehen, da die letzte fehlgeschlagene Login Zeit vor der Zählererhöhung angepasst wurde. Die einzige Ausnahme ist, wenn der Computer zwischen der Erhöhung und dem Abfragen eine erhöhte Auslastung hat und es zu einer Verzögerung kommt. Liegen die Zeiten weniger als 10 Minuten auseinander, wird der Status des Benutzers bzw. der IP-Adresse von "nicht blockiert" auf "blockiert" geändert. Zeitgleich wird der Master-Server über die Sperrung der IP-Adresse benachrichtigt. Dieser wird daraufhin die IP-Adresse in seiner Datenbank aufnehmen, sodass diese auch bei anderen Servern gesperrt ist.

Das Ändern des Status von "nicht blockiert" auf "blockiert" wird mit der Funktion "system()"durchgeführt. Dabei wird zuerst überprüft, ob es sich um einen Benutzer oder eine IP-Adresse handelt. IP-Adressen werden in die "iptables" des Betriebssystems eingetragen und Benutzer werden von der Gruppe "unblock" in die Gruppe "blocked"

Algorithm 12 Verringerung des Zählers und zurücksetzen der Zeit

```
 1: procedure CHECK_COUNT(info,isHost)
 2:     initialize variables
 3:     if strcmp(isHost,YES)==0 then
 4:         set variables to Host
 5:     else
 6:         set variables to User
 7:     dbget(db,key,data)
 8:
 9:     value = getvalue(info)
10:     if value = 1 then
11:         count = data.data
12:         if count ≥ 90 then
13:             decreasecount(info,isHost)
14:             command(host, "delete", isHost, tnow)
15:             change_state(info, isHost)
16:             decreasetime(info, isHost)
17:             return
18:         else
19:             decreasecount(info, isHost)
20:             decreasetime(info, isHost)
        ⋮
21:     if User then
22:         count = data.data
23:         if count ≥ 9 then
24:             decreasecount(info, isHost)
25:             command(user, "delete", isHost, tnow)
26:             change_state(info, isHost)
27:             decreasetime(info, isHost)
28:         else
29:             decreasetime(info, isHost)
30:             decreasecount(info, isHost)
```

Algorithm 13 Verringerung des Zählers

```
procedure DECREASECOUNT(info,isHost)
    initialize variables and set to Host or User

    dbget(db,key,data)
    count = data.data

    Host-case:
    value = getvalue(info)
    count = count - (10 * value)
    delete entry from db
    add entry to db with new count

    User-case:
    open user.conf file
    while read and not the end of user.conf do
        user = strtok(buffer,"=")
        ucount = strtok(NULL,"\n")
        if strcmp(user,info.user) then
            usercount = uc
    count = count - usercount
    delete entry from db
    add entry to db with new count
```

Algorithm 14 Zähler update Part 1

```
1: procedure DBUPDATECOUNT(info *info, const char *isHost)
2:     if Host then
3:         db = host database
4:     else
5:         db = user database
6:
7:     db → (db,NULL,dbc,0)
8:
9:     Host-Case:
10:    while err = dbc →get(dbc, key, data, DB_NEXT) == 0 do
11:        if strcmp(host, key.data) == 0 then
12:            count = data.data
13:            if dbc→c_del(dbc,0) == 0 then
14:                count = dbcountadd(info, isHost, count)
15:                break
16:
17:    User-Case:
18:    while err = dbc →get(dbc, key, data, DB_NEXT) == 0 do
19:        if strcmp(user, key.data) == 0 then
20:            count = data.data
21:            conf = fopen("user.conf","a+")
22:            if conf is not NULL then
23:                while fgets(buffer,sizeof(buffer),conf != NULL do
24:                    if strcmp(buffer,"#",1)!=0 then
25:                        user = strtok(buffer, "=")
26:                        ucount = strtok(NULL,"\n")
27:                        uc = atoi(ucount)
28:                        if strcmp(user,info→user)==0 then
29:                            usercount = uc
30:            count = count + usercount
31:            if dbc→c_del(dbc,0)==0 then
32:                dbcountadd(info,isHost,count,0)
33:
34:    go on part 2
```

Algorithm 15 Zähler update Part 2

```
 1: procedure DBUPDATECOUNT(info *info, const char *isHost)
 2:
 3:     proceeding part 1
 4:
 5:     if Host then
 6:         value = getvalue(info)
 7:         if value == 1 then
 8:             if count ≥ 90 then
 9:                 tnow = time(NULL)
10:                 dbtime = dbgettime(info, isHost)
11:                 diff = difftime(tnow, dbtime)
12:                 if diff ≤ 600 then
13:                     send_server(info)
14:                     dbupdatestate(info, tnow, isHost)
15:                     return
16:             else
17:                 return
                                                    ⋮
18:     else if count ≥ 9 then
19:         tnow = time(NULL)
20:         dbtime = dbgettime(info, isHost)
21:         diff = difftime(tnow, dbtime)
22:         if diff ≤ 600 then
23:             dbupdatestate(info, tnow, isHost)
24:             return
25:     else
26:         return
```

Algorithm 16 Berechnung der Erhöhung des Zählers

```
 1: procedure CALC_COUNT(info *info, int count, const char *isHost)
 2:     returned = geo_ip(host)
 3:     db = list
 4:     create cursor dbc
 5:
 6:     if returned is not NULL then
 7:         while err = dbc→get(dbc, key, data, DB_NEXT) == 0 do
 8:             if strncmp(returned, key.data,2) == 0 then
 9:                 value = data.data
10:                 break
11:     c = 10 * value + c
```

verschoben. Das Blockieren ist in Algorithmus 17 zu sehen.

Algorithm 17 Blockieren von Hosts bzw. Benutzern

```
procedure COMMAND(*host, *status, *isHost, tnow)
    if strcmp(status, "delete) == 0 then
        if strcmp("yes",isHost) == 0 then
            snprintf(block, sizeof(block), "ipdables -D INPUT -s %s -j DROP", host)
            ret = system(block)
            write_Log(host, status, isHost)
            return
        else
            snprintf(block, sizeof(block), "usermod -g unblock %s", host)
            ret = system(block)
            write_Log(host, status, isHost)
            return
    else if strcmp("yes", isHost) == 0 then
        snprintf(block, sizeof(block), "iptables -A INPUT -s %s -j DROP", host)
        ret = system(block)
        addtoFile(YES, host, tnow)
        write_Log(host, status, isHost)
        return
    else
        snprintf(block, sizeof(block), "usermod -g blocke%s", host)
        ret = system(block)
        addtoFile(NO, host, tnow)
        write_Log(host, status, isHost)
        return
```

5.4 PAM-Anbindung

In diesem Teil der Thesis werde ich die Anbindung an PAM genauer erläutern.

5.4.1 Warum PAM?

Warum verwendet man überhaupt PAM? Würde jedes Programm sich selbst um die Authentifizierung von Benutzern kümmern, würde der Programmieraufwand enorm und unnötig steigen. Wenn es eine neue Authentifizierungsmethode gibt, müsste jedes Programm erneut angepasst und erweitert werden. Ebenfalls müssten Fehler bei jedem Programm einzeln behoben werden.

Der Vorteil von PAM besteht also darin, dass es die Authentifizierung in eine Bibliothek auslagert. Das Programm muss sich also nicht mehr um diese Aufgabe bemühen.

```
# PAM configuration for the Secure Shell service
auth            optional        pam_abfp.so
auth            required        pam_unix
session         optional        pam_abfp.so
```

Figure 5.3: Auszug aus der sshd-Datei

Kommen neue Authentifizierungsverfahren hinzu können diese eingebunden werden, ohne dass Programme sich verändern.[4]

5.4.2 Anbindung

Die Anbindung erfolgt durch die Eintragung in die Datei "etc/pam.d/sshd". Der Eintrag 5.3 ist dabei wie folgt zu lesen[5]: ein Eintrag enthält immer einen "type", "control" und "module-path". Der "type" sagt aus, um welchen Modultyp es sich handelt. Bei meinem Programm handelt es sich um den Modultyp "auth" und "session". Der "auth" Teil übernimmt die Verifikation des Benutzers und der "session" Teil stellt Einstellungen für die Sitzung ein. Beim "control" handelt es sich um "optional". Die beiden Hauptfunktion des Programms müssen nicht erfolgreich beendet worden sein um Zugang zum Server zu erhalten. An letzter Stelle steht der Pfad zum PAM-Modul, also zum Programm. Das Modul kann sich auch in dem "/lib/security" Ordner befinden, wobei dann der Modulname ausreicht.

5.5 Master-Server

In diesem Kapitel werde ich genauer auf den Master-Server eingehen. Grundsätzlich handelt es sich dabei um einen einfachen Server[6,7]. Die Hauptfunktion ist in Algorithmus 18 zu sehen. Dieser erzeugt zunächst einen "Socket". Ein Socket ist ein Software-Modul mit dem man Computer miteinander verbinden kann. Nach dem Erzeugen eines Sockets wird die IP-Adresse und der Port für den Server festgelegt, was mit der Funktion "bind()" geschieht. Durch den Aufruf dieser Funktion wird dem Socket eine Adresse zugewiesen.

Nach dem Binden des Sockets an eine Adresse, wird eine Warteschlange eingerichtet, für Verbindungen, welche am Socket eingehen. Das Einrichten erfolgt mit der Funktion "listen()". Das Programm wird dabei solange unterbrochen, bis eine Verbindung eintrifft. Die Funktion kann mehrere Wünsche gleichzeitig bearbeiten. Für den Server

[4]Linux-Magazin[vSW04]
[5]TU Chemnitz [Tec08]
[6]BinaryTides [Bin12]
[7]Galileo Computing [Wol09]

habe ich eine Warteschlange der Größe 10 gewählt, da diese ausreichen sollte um alle Verbindungen abzuarbeiten.

Nun wird das Programm in eine Endlosschleife gebracht, welche die Serverhauptschleife darstellt. In dieser Hauptschleife wird die Funktion "check_db()" aufgerufen. Diese Funktion überprüft die Datenbank des Servers auf IP-Adressen, die das Zeitlimit überschritten haben. Wenn es solche Adressen gibt, dann werden diese aus der Datenbank gelöscht und wieder für andere Server freigegeben.

Sind nun ein oder mehrere Verbindungswünsche eingetroffen, werden diese mit der Funktion "accept()" abgearbeitet. Dabei wird immer die nächste Verbindung aus der Warteschlange geholt. Ebenfalls blockiert die Funktion die Hauptschleife, bis ein Client unseren Server aufruft.

Versucht ein Client sich mit dem Server zu verbinden, wird ein neuer Prozess erzeugt. Dies macht man um mehrere Verbindungen gleichzeitig abarbeiten zu können, was auch den Master-Server erheblich beschleunigt. Nach dem Erzeugen eines neuen Prozesses, empfängt der Server Daten vom Client. Der Server unterscheidet nun, ob es sich um eine Abfrage oder um eine Sperrung handelt, das Protokoll hierfür ist in Figure 5.4[8] zu sehen. Dabei wird zuerst der Diffie-Hellman-Schlüsselaustausch vollzogen. Ist das "shared Secret" erzeugt, kann mit diesem die Nachricht des Server entschlüsselt werden, zu sehen in Algorithmus 19. Ist die Nachricht erfolgreich entschlüsselt worden, wird überprüft, ob es sich um eine Abfrage oder um eine Sperrung handelt. Das "shared Secret" ist dafür da, das Sperren von IP-Adressen für nicht autorisierte Benutzer zu erschweren.

Wenn es sich bei der Nachricht des Clients um eine Abfrage handelt, dann überprüft der Server mit der Funktion "read_db()" ob die IP-Adresse schon in der Datenbank vorhanden ist oder nicht. Anschließend wird die entsprechende Nachricht dem Client zurückgeschickt. Am Schluss wird der Prozess geschlossen und so die Ressourcen dem System wieder zur Verfügung gestellt.

5.6 Entsperrung über CRON-Job

In diesem Kapitel werde ich die Entsperrung über den CRON-Job genauer erläutern. Bei dieser Entsperrung handelt es sich um das eigentliche Entsperren.

Als erstes wird ein CRON-Job in der "crontab", welche sich im Ordner "/etc/" befindet, erstellt. Hierfür wird die Zeile verwendet, welche in Figure 5.5 zu sehen ist. Die fünf "*" stehen dabei für die Minute, Stunde, Tag, Monat und Wochentag. Das Programm "testdb" wird also jede Minute ausgeführt. An zweiter Stelle steht der Benutzer, der das Programm ausführen soll. In diesem Fall ist dies der Benutzer "root". Anschließend folgt die Pfadangabe zum Programm. An letzter Stelle steht die Ausgabe, welche in diesem Fall komplett verworfen wird. Es wird keine Ausgabe benötigt, da das Programm uns eine Log-Datei erstellt.

[8] erstellt mit `https://www.websequencediagrams.com/`

Algorithm 18 Hauptfunktion des Master-Servers

```
1:  procedure MAIN(argc, argv)
2:      listenfd = socket(AF_INET,SOCK_STREAM, 0)
3:      bind(listenfd, serv_addr, sizeof(serv_addr))
4:      listen(listendf, 10)
5:
6:      while 1 do
7:          check_db()
8:          connfd = accept(listenfc, client, c)
9:
10:         ⋮
11:
12:         if strstr(decrypted,block) != NULL then
13:             ret = db_add(db, ip)
14:         else
15:             read_db(db, clientmst)
16:             if ret == 0 then
17:                 listed
18:                 snprintf(sendBuff, sizeof(sendBuff),"listed")
19:             else
20:                 not listed
21:                 snprintf(sendBuff, sizeof(sendBuff),"not listed")
22:         write(connfd, sendBuff,strlen(sendBuff))
```

Algorithm 19 Entschlüsselung

```
1:  procedure SIMPLE_DECRYPT(char *msg, int key)
2:      for i=0; i<msglen; i++ do
3:          encrypted[i] = msg[i] ^key
4:      encrypted[messagelen] = "\0"
5:      return encrypted
```

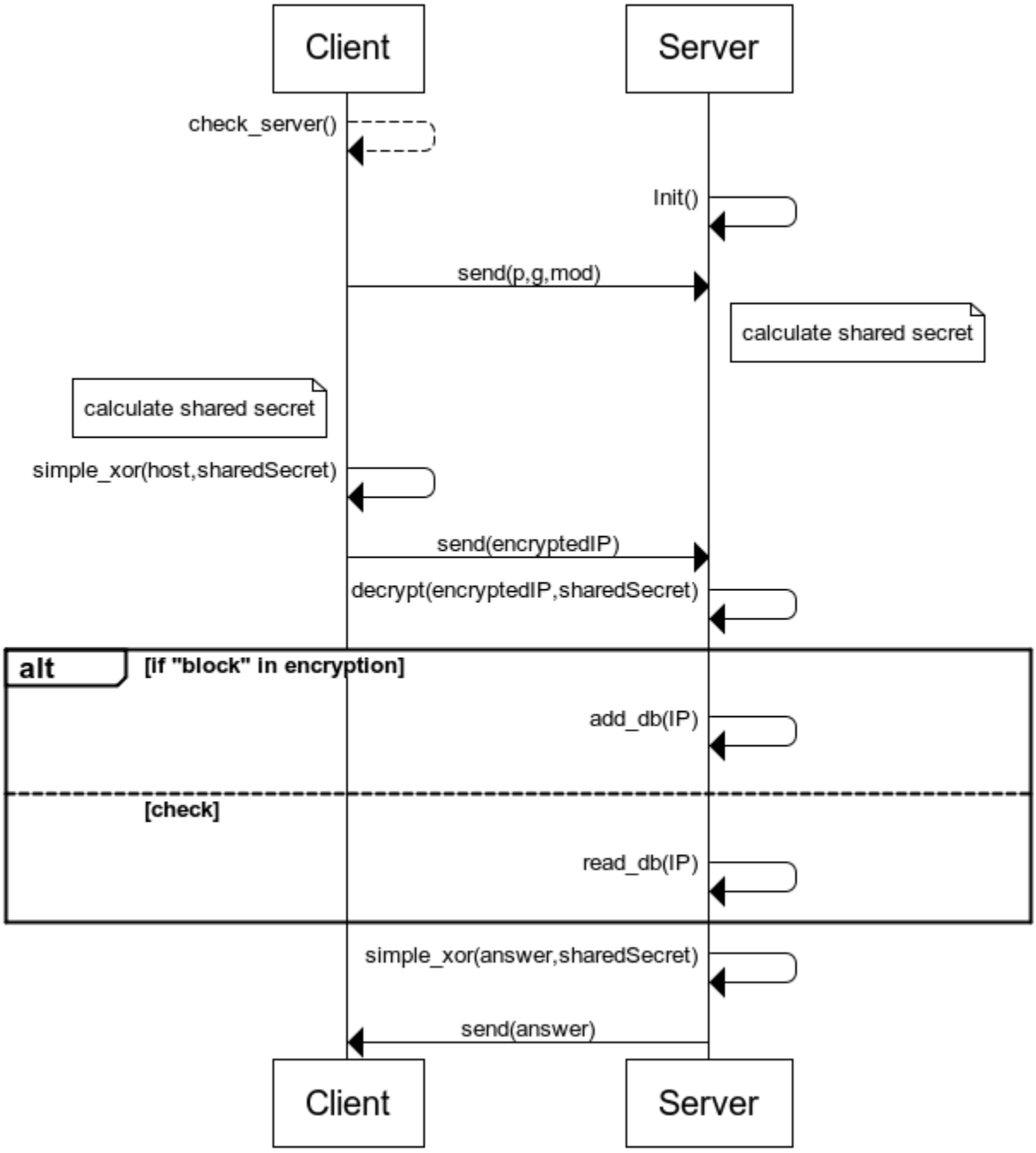

Figure 5.4: Protokoll zwischen Server und Client

```
* * * * * root /usr/pam_abfp/testdb > /dev/null 2>&1
```

Figure 5.5: Eintrag in der crontab

Im Algorithmus 20 sieht man nun die Hauptfunktion des Programms. Am Anfang wird die aktuelle Zeit ausgelesen, was in Zeile 2 bis 5 passiert. Dann wird die Funktion "my_itoa()" aufgerufen. Diese Funktion gibt die Minute der aktuellen Zeit zurück. Zum Beispiel, wenn es "10:53" ist, dann ist der Rückgabewert der Funktion 3. Diese Überprüfung wird gemacht, damit das Programm weiß, welche der zehn Dateien als nächstes abgearbeitet werden muss.

Ist die aktuelle Minutenzahl ermittelt, sucht das Programm die richtige Datei. Dies wird über mehrere Abfragen gemacht. Hat es nun die richtige Minute gefunden, dann wird der Pfad zu den Dateien gesetzt. Es handelt sich hierbei um zwei, da es jeweils eine File für die gesperrten Benutzer und eine für die gesperrten IP-Adressen gibt. Anschließend wird die Funktion "clearHost()" aufgerufen.

In Algorithmus 21 wird zuerst die Datei geöffnet. Danach wird diese so lange ausgelesen, bis das Ende der Datei erreicht ist. Dabei wird jede Zeile der Funktion "command()" übergeben. Die Funktion "command()" arbeitet ähnlich wie die Funktion in Algorithmus 17. Der Unterschied zwischen den beiden Funktionen besteht darin, dass bei der Zweiten der Teil, welcher für das Hinzufügen zuständig ist, weggelassen wurde. Am Ende der Funktion "clearHost()" wird noch die entsprechende Datei gelöscht und eine neue erzeugt.

Algorithm 20 Hauptfunktion des Programms zum Entsperren

```
 1: procedure MAIN(argc, argv)
 2:     tnow = time(tnow)
 3:     tmnow = localtime(tnow)
 4:     smin = my_itoa(tmnow.min)
 5:     min = atoi(smin)
 6:
 7:     if min == 0 then
 8:         fpath = "/usr/pam_abfp/ban/file0"
 9:         upath = "/usr/pam_abfp/ban/u0"
10:         clearHost(YES, fpath)
11:         clearHost(NO, upath)
12:         return(EXIT_SUCCESS)
13:     ⋮
14:     return EXIT_SUCCESS
```

Algorithm 21 Entsperrung der Benutzer und IP-Adressen

```
1: procedure clearHost(isHost, fpath)
2:     file = fopen(fpath,"a+")
3:
4:     while fgets(buffer,file) is not NULL do
5:         command(buffer,isHost)
6:
7:     clearFile(fpath)
8:     fclose(file)
```

5.7 Testen des Programms

In diesem Teil wird das Programm "pam_abfp" getestet. Mit Hilfe von "planet-lab"[9] habe ich versucht mich von verschiedenen Orten aus einzuloggen. Ich habe darauf geachtet, dass der Server das Modul besitzt. Dabei wird unter anderem gezeigt, das bei Überschreitung des Schwellwertes die Sperrung funktioniert.

Das verwendete Setup war ein lokaler Server, der ans Internet angebunden wurde und verschiedene planet-lab Orte um sich am Server anzumelden. Am Anfang wurde immer ein erfolgreicher Login durchgeführt, um sicherzustellen, dass die Anmeldung am Server funktioniert. Es wurden verschiedene Orte, also Länder, gewählt um zu zeigen, dass die Verwendung von GeoLocations funktioniert. Die Anbindung an SPAMHAUS kann aufgrund der Einbindung nicht gezeigt werden, da hierfür keine geeigneten IP-Adressen zur Verfügung standen. Die Passwortanalyse kann mit Hilfe eines kleinen Zusatzprogrammes gezeigt werden. Dieses liest die verschiedenen Datenbanken des Server aus.

5.7.1 GeoLocation-Test

Beim Testen der GeoLocation-Funktion wird mit einem spanischen Server angefangen. Hierfür erfolgt ein erfolgreicher Login um zu zeigen, dass das Einloggen am Server funktioniert. Der Output des Terminal ist dabei in Figure 5.6 zu sehen.

Den Wert für Spanien habe ich auf "1" gesetzt. IP-Adressen aus Spanien haben also maximal neun Login-Versuche. Nach dem Bestehen des erfolgreichen Logins wird mehrfach ein falsches Passwort eingegeben. In Figure 5.7 sieht man die Einträge in den Datenbanken.

Wenn der Wert für Spanien auf "3" gesetzt wird, sehen die Datenbankeinträge wie in Figure 5.8 aus. Dabei ist zu beachten, dass der Wert für den Benutzer unverändert geblieben ist. Setzt man den Benutzer "test" ebenfalls auf den Wert "3", dann sieht man in Figure 5.9, dass bei dem Benutzer ebenfalls der Zähler stärker gestiegen ist als um den Wert "1".

[9] Planetlab-url `http://www.planet-lab.eu/`

```
Welcome to Ubuntu 13.04 (GNU/Linux 3.8.0-31-generic i686)

 * Documentation:  https://help.ubuntu.com/

57 packages can be updated.
29 updates are security updates.

New release '13.10' available.
Run 'do-release-upgrade' to upgrade to it.

Last login: Sat Nov  9 00:42:15 2013 from planetlab1.upc.es

test@ubuntu: ~test@ubuntu:~$
```

Figure 5.6: Erfolgreicher Login mit spanischer IP-Adresse

```
Host/User: planetlab1.upc.es : Time: Sat Nov  9 02:43:07 2013
Host/User: planetlab1.upc.es : Count: 90
Host/User: planetlab1.upc.es : State: 1

Host/User: test : Time: Sat Nov  9 02:43:07 2013
Host/User: test : Count: 9
Host/User: test : State: 1
```

Figure 5.7: Gesperrte Datenbank mit spanischer IP-Adresse

```
Host/User: planetlab1.upc.es : Time: Sat Nov  9 03:26:50 2013
Host/User: planetlab1.upc.es : Count: 30
Host/User: planetlab1.upc.es : State: 0

Host/User: test : Time: Sat Nov  9 03:26:50 2013
Host/User: test : Count: 1
Host/User: test : State: 0
```

Figure 5.8: Wert auf 3 gesetzt für IP-Adresse

```
Host/User: planetlab1.upc.es : Time: Sat Nov  9 03:31:05 2013
Host/User: planetlab1.upc.es : Count: 60
Host/User: planetlab1.upc.es : State: 0

Host/User: test : Time: Sat Nov  9 03:31:05 2013
Host/User: test : Count: 4
Host/User: test : State: 0
```

Figure 5.9: Wert auf 3 gesetzt für Benutzer

```
Host/User: planetlab1.upc.es : Time: Sat Nov  9 02:43:10 2013
Host/User: planetlab1.upc.es : Count: 0
Host/User: planetlab1.upc.es : State: 0

Host/User: test : Time: Sat Nov  9 02:43:10 2013
Host/User: test : Count: 0
Host/User: test : State: 0
```

Figure 5.10: Entsperrte Einträge mit spanischer IP-Adresse

Nach Ablauf von zehn Minuten wird die IP-Adresse und der Benutzername wieder entsperrt. Dies sieht man anhand der Datenbankeinträge in Figure 5.10.

5.7.2 Passwort-Test

Das Testen der Passwortanalyse habe ich durch das Einloggen mit dem richtigen Passwort überprüft, dies ist bereits in Figure 5.6 zu sehen. Bei einem erfolgreichen Login wird der Zähler nicht erhöht. Wird ein ähnliches Passwort verwendet, wird der Zähler ebenfalls nicht erhöht, siehe Figure 5.11. Der hierfür verwendete Benutzer war "test" und das dazugehörige richtige Passwort war "test". Verwendet man anstatt des richtigen Passwortes folgendes Passwort "t3st", dann wird der Zähler nicht erhöht. Hierbei werden auch die Zeiten nicht erhöht, da das Programm diese nicht erneuert, sondern bei einem ähnlichen Passwort das Programm beenden.

5.7.3 Master-Server-Test

Das Funktionieren des Master-Server wird, anhand der Sperrung einer IP-Adresse gezeigt(siehe Figure 5.12). Wird die Adresse gesperrt, wird diese auch am Master-Server gesperrt. Dies wird über die Log-Datei des Servers verdeutlicht. Die Log-Datei ist dabei in Figure 5.13 zu sehen. Leider war es mir nicht möglich zu zeigen, dass die IP-Adresse für alle gesperrt ist, da mir nicht genügend Server zur Verfügung standen.

```
Host/User: planetlab1.upc.es : Time: Sat Nov  9 02:43:10 2013
Host/User: planetlab1.upc.es : Count: 0
Host/User: planetlab1.upc.es : State: 0

Host/User: test : Time: Sat Nov  9 02:43:10 2013
Host/User: test : Count: 0
Host/User: test : State: 0
```

Figure 5.11: Eingabe eines ähnlichen Passwortes

```
Host/User: planetlab1.upc.es : Time: Sun Nov 10 05:43:08 2013
Host/User: planetlab1.upc.es : Count: 150
Host/User: planetlab1.upc.es : State: 1

Host/User: test : Time: Sun Nov 10 05:42:59 2013
Host/User: test : Count: 9
Host/User: test : State: 1
```

Figure 5.12: gesperrte IP-Adresse und Log Datei des Servers

```
Adding planetlab1.upc.es
```

Figure 5.13: Log Datei des Servers

5.7.4 Analyse

In diesem Teil werde ich die Performance des Programms darlegen. Durch die Verwendung von PAM wird das Programm nur dann aufgerufen, wenn ein Benutzer versucht sich am Server anzumelden. PAM erlaubt es auf verschiedene Informationen direkt zuzugreifen und ermöglicht somit schnellen Zugriff auf die benötigten Informationen. Hierbei entsteht durch das Programm wenig Auslastung für das System.

Durch die Verwendung von Datenbanken ist es möglich ankommende Anfragen schnell abzuarbeiten. Die Abarbeitung bei Datenbanken beruht dabei in meinem Programm auf der Suche in einem "B-Baum". Ein "B-Baum" ist ein vollständig balancierter Baum, welcher nach Schlüsseln sortiert ist. Das Einfügen, Suchen und Löschen ist dabei in logarithmischer Laufzeit($T(n) = O(log_{\lceil m/2 \rceil}(n))$)[10] möglich. Da das Programm fast ausschließlich nur auf Datenbanken operiert, ist die Zeit welche benötigt wird um die passenden Einträge zu finden nur logarithmisch.

Das Abfragen einer DNSBL-Datenbank ist aufgrund der integrierten Funktionen von C sehr schnell möglich. Die Performance hängt dabei nicht von meinem Programm, sondern von der Performance der DNSBL selbst ab. Hat diese eine erhöhte Auslastung, dann wird das Abarbeiten der Anfrage etwas verzögert.

Das Abarbeiten der einzelnen Konfigurationsdateien wird schnell abgearbeitet, da es sich bei diesen um sehr kleine Dateien handelt. Dadurch, dass die Datei einmal vom Anfang bis zum Ende durchlaufen wird, hat es eine Laufzeit von $O(n)$, also lineare Laufzeit.

[10] B-Baum[Bit07]

Chapter 6

Zusammenfassung

Das Ergebnis dieser Thesis ist, dass es möglich ist SSH-Verbindungen sicherer gestalten zu können und somit Angreifern weniger Angriffsmöglichkeiten zu bieten. Anders als bereits vorhandene Programme bietet das Programm "PAM_abfp" die Möglichkeit die IP-Adressen der Angreifer genauer zu lokalisieren. Daher ist es möglich Benutzer in Gruppen aufzuteilen und diese bei einem fehlerhaften Login unterschiedlich zu bestrafen. Dies gewährleistet einen höheren Schutz gegenüber Angreifern.

Durch die Anbindung an eine DNSBL-Datenbank ist es möglich, bereits auffällige IP-Adressen im Voraus vom Server auszusperren. Durch diese Maßnahme wird sichergestellt, dass Angreifer gar nicht erst eine Verbindung zum Server aufbauen können und die Daten auf dem Server stärker geschützt sind.

Die Passwortanalyse gewährt den Benutzern eine Möglichkeit trotz eines fehlerhaften Login Versuches nicht direkt vom Server ausgesperrt zu werden. Den Benutzern ist es weiterhin möglich auf ihre Daten zuzugreifen.

Die Anbindung an einen Master-Server gewährt die Option bereits bei anderen gesperrten Servern IP-Adressen vom eigenen Server auszuschließen. Durch diesen Einsatz wird der Server zusätzlich vor Angreifern aus Botnets geschützt.

Die in Kapitel 1 vorgestellten Programme bieten diese zusätzlichen Schutzmechanismen nicht und deswegen erlaubt das Programm, welches in meiner Bachelor-Thesis erstellt wurde, gegenüber diesen eine höhere Sicherheit für Server zu schaffen.

Durch die Verwendung dieser Erweiterungen wird ein Server, welcher sensible Daten bereithält deutlich mehr geschützt. Angreifern wird das Eindringen in den Server erheblich erschwert. Aufgrund der Bannzeit und dem dadurch verbundenen aussperren des Angreifers steigt die Zeit für einen Brute-Force Angriff erheblich, weshalb es sich für Angreifer nicht lohnt den Server anzugreifen.

Bibliography

[Bin12] BinaryTides. Server and client example with C sockets on Linux. http://www.binarytides.com/server-client-example-c-sockets-linux/, 2012. Online, visited on September 13th 2013.

[Bit07] O. Bittel. Algorithmen und datenstrukturen - suchen, 2007. Available online at http://www-home.fh-konstanz.de/~bittel/aldaBac/Vorlesung/01cSuchen.pdf at 1-97, visited on November 10th 2013.

[Chr06] Christian Seifert. Malicious SSH Login Attempts, 2006.

[dev] devshed. sshprivatepublickey. Online at http://images.devshed.com/wh/stories/sshprivatepublickey1.jpg, visited on August 14th 2013.

[Gei08] Kenneth Geisshirt. Development with pluggable authentication modules. https://www.packtpub.com/article/development-with-pluggable-authentication-modules-pam, 2008. Online, visited on November 12th 2013.

[Gmb] Symantec GmbH. Symantec sicherheitsreport: Deutschland mit den meisten phishing-webseiten in europa. Online at http://www.presseportal.de/pm/6332/957528/symantec-sicherheitsreport-deutschland-mit-den-meisten-phishing-webseiten-in-europa, visited on September 13th 2013, to find China passage search for "China".

[GNU13] GNU. Grep. http://www.gnu.org/software/grep/, 2013. Online, visited on November 9th 2013.

[hei] heise. Botnet. Online at http://www.heise.de/imgs/18/4/8/7/1/7/5/nt-botnet-gr.jpg-16273f301b3c504e.jpeg, visited on August 24th 2013.

[Jam08] James P. Owens, Jr. A Study of Passwords and Methods Used in Brute-Force SSH Attacks. Master's thesis, CLARKSON UNIVERSITY, 2008.

[Ken10] Christopher Kenna. Analysis of and response to ssh brute force attacks, 2010. The College of William and Mary Computer Science Department.

[MAvO11] Mohammad Mannany Mansour Alsaleh and P.C. van Oorschot. Revisiting defenses against large-scale online password guessing attacks, 2011.

[Mic] Microsoft. So schützen Sie Ihren PC besser gegen das Botnet und Schadsoftware. Online at `http://www.microsoft.com/de-de/security/pc-security/botnet.aspx`, visited on September 9th 2013.

[pd] passwort depot. Brute-Force-Angriffe. Online at `http://www.password-depot.de/know-how/brute_force_angriffe.htm`, visited on August 24th 2013.

[Pol13] Wayne Pollock. PAM Tutorial, 2013. Available online at `http://content.hccfl.edu/pollock/AUnix2/PAM-Help.htm`, visited on August 12th 2013.

[Ram03] Ramaprabhu Janakiraman and Marcel Waldvogel and Qi Zhang. A peer-to-peer approach to network intrusion detection and prevention. In *Proceedings of IEEE WETICE 2003 Workshop on Enterprise Security*, Linz, Austria, 2003.

[Tec08] Technische Universität Chemnitz. *PAM: Konfiguration von Authentisierungsverfahren*, 2008. Available online at `http://www.tu-chemnitz.de/urz/kurse/unterlagen/pamkonf.html`, visited on August 23rd 2013,.

[vSW04] Dirk von Suchodoletz and Martin Walter. Die Pforten zur Linux Maschinerie: Pluggable Authentication Modules. *Linux-Magazin*, 2004. Online, visited on September 3rd 2013.

[wik] wikipedia. Brute-Force-Methode. Online at `http://de.wikipedia.org/wiki/Brute-Force-Methode`, visited on September 3rd 2013.

[Wik13] Wikipedia. SSH-Tunnel. `http://commons.wikimedia.org/wiki/File:SSH-tunnel.svg`, 2013. Online, visited on September 17th 2013.

[Wol09] Jürgen Wolf. *C von A bis Z - das umfassende Handbuch.* Galileo Press, Bonn, 3. aufl. edition, 2009.

[YNS11] Benjamin Jenghorng Wu Yen-Ning Su, Guang-Han Chung. Developing the upgrade detection and defense system of ssh dictionary-attack for multi-platform environment. Online at `(http://www.SciRP.org/journal/ib)`, 2011.

Selbstständigkeitserklärung

Hiermit erkläre ich, daß ich die vorliegende Arbeit selbstständig und nur mit erlaubten Hilfsmitteln angefertigt habe.

Konstanz, November 13, 2013

Jürgen Kollek