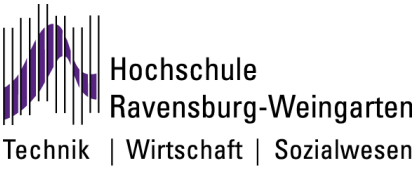


Dokumentation

zur Vorlesung Systemadministration
im Bachelor Studiengang Angewandte Informatik

Wintersemester 2016 / 2017
bei Herrn Prof. Dr. rer. nat. Eggendorfer

Umsetzung eines Honeypots

Michael Stroh
Matrikelnr. 24972

Daniel Schwenk
Matrikelnr. 24961

24. Oktober 2016

Inhaltsverzeichnis

Abkürzungsverzeichnis

DNS	Domain Name System
FTP	File Transfer Protocol
HIHP	high-interaction honeypot
HTTP	Hypertext Transfer Protocol
LIHP	low-interaction honeypot
LTS	Long Term Support
MIHP	mediium-interaction honeypot
SSH	Secure Shell
SQL	Structured Query Language

1 Einleitung

Das Internet und die Digitalisierung, die in alle Lebensbereiche Einzug hält, verändern Gesellschaft, Wirtschaft und Kultur. Egal ob im privaten oder beruflichen Umfeld, ständig sind wir von Computern in Form von Arbeitsgeräten, Smartphones oder anderen Geräten umgeben. Diese Verbreitung sowie die Vernetzung von Geräten untereinander wird in den nächsten Jahren im Zuge der „Internet-der-Dinge-Evolution“ weiter drastisch zunehmen.

Ein oft vernachlässigter Aspekt ist hierbei das Thema „IT-Sicherheit“. Keine Software ist frei von Fehlern und Sicherheitslücken. Falsch konfigurierte Dienste und Software, die nicht regelmäßig aktualisiert wird, sind ein leichtes Ziel für Angreifer. Durch die zunehmende Vernetzung wird das Thema IT-Sicherheit in Zukunft weiter an Bedeutung gewinnen.

Um eine Infrastruktur, egal ob im privaten oder geschäftlichen Bereich, vor möglichen Angriffen zu schützen bedarf es eines immer größeren Aufwandes.

1.1 Motivation

Die Gewährleistung der IT-Sicherheit ist mittlerweile eine immens wichtige, wenn nicht sogar die wichtigste Anforderung an eine intakte IT-Infrastruktur. Entsprechend sollte ein Systemadministrator über ein breites Spektrum an Wissen im Bereich der IT-Sicherheit besitzen sowie in der Lage sein, mögliche Angriffsszenarien frühzeitig zu erkennen.

Das Konzept eines Honeypots, also einen potentiellen Angreifer nicht nur vor eigentlich wichtigen System fernzuhalten, sondern auch noch von seinem Wissen zu profitieren, stellt dabei einen hochspannenden Ansatz dar. Dieser Ansatz soll dem Projektteam helfen, Wissen über mögliche Angriffsszenarien und Vorgehensweisen zu erlangen, um so die Sicherheit von bestehenden und zukünftigen Infrastrukturen gewährleisten zu können.

1.2 Ziele

Ziel dieser Arbeit ist es, ein System zu entwickeln, das als Honeypot dient. Dieser Honeypot soll eingesetzt werden, um Einblicke in die Vorgehensweise eines Angreifers zu bekommen. Das System stellt dazu ein vermeintlich leicht angreifbaren Webserver sowie SSH-Server dar.

Jegliche Zugriffe und Aktivitäten die ein Angriff hinterlässt werden protokolliert und ausgewertet. Das hierbei gewonnene Wissen soll in Form von IT-Sicherheitsmaßnahmen in bestehende und künftige IT-Infrastrukturen einfließen.

- Primärziel: einsatzfähige(r) Honeypot(s) - sicher, authentisch und lehrreich

1.3 Eigene Leistung

Der Hauptbestandteil dieses Projekts liegt in der Inbetriebnahme und Bereitstellung eines Honeypots, sowie die Integration desselben in eine für einen potentiellen Angreifer authentisch erscheinenden Umgebung. Dabei liegt das Hauptaugenmerk darauf, die Sicherheit des Systems zu gewährleisten. Die Dokumentation der Infrastruktur, der stattgefundener Angriffe sowie deren Auswertung stellen einen wichtigen Bestandteil dar. Dieses Wissen dient dem Projektteam zukünftig zur Absicherung von IT-Infrastruktur.

- Bereitstellung einer authentischen Umgebung für den Honeypot
- Gewährleistung der Sicherheit für das eigene und das globale Netz
- Inbetriebnahme des Honeypots selbst
- automatisierte Auswertung von Log-Files durch Skripte
- aus Auswertung erfolgt Erarbeitung und Ausweitung von Sicherheitsrichtlinien (Passwortrichtlinien, Firewallregeln)
- Dokumentation von Honeypot inklusive Umgebung, Angriffsphase und Ergebnisse

2 Grundlagen

2.1 Honeypot

Das allgemeine Ziel eines Honeypots ist es, einen Angreifer von einem eigentlichen Ziel abzulenken oder aber um Informationen von einem Angreifer und seiner Vorgehensweise zu sammeln [1]. Dazu wird von einem Honeypot ein oder mehrere Dienste, ein ganzes Rechnernetz oder das Verhalten von einem Anwender simuliert. Erfolgt ein Zugriff auf eine der simulierten Ressourcen, werden alle damit verbundenen Aktivitäten protokolliert und bei Bedarf Alarm ausgelöst. Das reale Netzwerk bleibt bleibt im Idealfall von Angriffen verschont, da es besser gesichert ist als der Honeypot. Der Ursprung der Bezeichnung Honeypot geht auf die Überlegung zurück, dass Bären mit einem Honigtopf sowohl abgelenkt als auch in eine Falle gelockt werden könnten [2].
Ein Honeypot kann je nach Eigenschaften oder Einsatzgebiet einer Klasse von Honeypots zugewiesen werden. Nawrocki et. al führen dazu in [1] eine Klassifizierung in „Produktions-“ und „Forschungshoneypots“, in Client- und Serverhoneypots sowie eine Unterscheidung nach physikalischem und virtuellem Honeypot auf. Zudem unterscheiden sie in Abhängigkeit der Interaktion:

- low-interaction honeypots (LIHP)
- medium-interaction honeypots (MIHP)
- high-interaction honeypots (HIHP)

Ein LIHP simuliert einen oder nur eine kleine Anzahl an Diensten wie SSH oder FTP und Antwortet dabei nur in sehr geringem Umfang, um beispielsweise Protokollhandshakes abzubilden. Die gewonnen Informationen dienen dabei oftmals nur zu statistischen Zwecken. MIHPs bilden einzelne Dienste erheblich genauer ab. Eine vollständige Kommunikation über den angeboten Dienst ist somit möglich. Da diese Typen jeweils nur einzelne Dienste und keine Funktionalität eines Betriebssystems abbilden, ist die Gefahr der Kompromittierung des Honeypots-System gering. HIHP sind in der Entwicklung, beim Ausrollen sowie bei der Wartung dagegen deutlich komplexer, ermöglichen jedoch auch den höchsten Grad der Erfassung von Angriffsmustern. Ein HIHP bildet ein komplettes Betriebssystem inklusive mehrerer Dienste ab. Der Fokus eines HIHPs liegt dabei nicht auf automatisierten Angriffen, sondern darauf, manuell ausgeführte Angriffe zu beobachten und protokollieren, um so neue Methoden der Angreifer rechtzeitig zu erkennen [2].
Der Einsatz von Honeypots bringt nicht nur Vorteile mit sich. Nawrocki et. al bemängeln in [1], dass ein Honeypot-System von einem Angreifer oftmals erkannt wird, da sich das

Verhalten der simulierten Dienste von einem realen Dienst unterscheidet. Zudem besteht jederzeit die Gefahr, dass ein Honeypot-System von einem Angreifer kompromittiert oder gar übernommen wird. Dies stellt ein erhebliches Risiko für die umgebende Infrastruktur dar.

3 Anforderungen

Die Anforderungen an das Honeypot-System werden in Muss-, Kann- und Soll-Kriterien unterteilt.

3.1 Muss-Kriterien

- Honeypot ist über das Internet erreichbar
- Honeypot darf keinerlei Gefahr für andere Systeme darstellen
- Honeypot muss jederzeit deaktivierbar sein
- Angreifer darf keinerlei Möglichkeit zur Interaktion mit dem Host-Betriebssystem haben
- Angreifer darf keine bzw. nur gefälschte Antworten auf Anfragen erhalten
- Honeypot muss mindestens einen, besser jedoch mehrere Dienste, wie beispielsweise HTTP, SSH oder FTP, simulieren/anbieten
- Honeypot muss ein realistisch wirkendes Angriffsziel darstellen
- Angriffe werden geloggt
- Ein- und ausgehender Netzwerkverkehr muss (überwacht und) geloggt werden
- Protokollierte Daten dürfen durch Angreifer nicht verändert werden können

3.2 Soll-Kriterien

- Automatische Benachrichtigung, wenn System angegriffen wird
- Protokollierung und ggf. Forwarding von Log-Files dürfen für den Angreifer nicht sichtbar sein
- Automatisierte Auswertung von Logdaten
- Logdateien / ausgewertete Daten werden automatisch separat gespeichert (extra System, Cloud-Speicher)

3.3 Kann-Kriterien

- Geringer Stromverbrauch von Honeypot-System
- Kostengünstiger Versuchsaufbau
- Reverse DNS-Lookup von Angreifer-IP-Adresse(n)
- Simulation weiterer Geräte (Router, Firewall, PC)
- Honeypot simuliert offenes WLAN-Netz / Fake-Access-Point

4 Marktanalyse

Eine Marktanalyse zeigt, dass eine Vielzahl an verschiedenen Honeypot-Paketen, Skripten und Konfigurationen mit sehr unterschiedlichen Eigenschaften verfügbar sind. Darunter befinden sich sowohl kommerzielle als auch freie Lösungen.

4.1 HoneyDrive

Mit HoneyDrive existiert eine Honeypot-Linux-Distribution auf Basis von Xubuntu Desktop 12.04.04 LTS. Diese Linux-Distribution bringt 10 vorinstallierte und vorkonfigurierte Honeypot-Pakete wie Kippo SSH Honeypot, Glastopf Web Honeypot oder Amun Malware Honeypot mit. Eine ausführliche Auflistung ist unter [3] gegeben. Die Distribution wird als OVA-Datei angeboten und kann so unter einer Virtualisierungssoftware ausgeführt werden. Neben den vorinstallierten Honeypot-Paketen sind des weiteren unter anderem ein Web- und Datenbankserver sowie wie PHPMyAdmin vorinstalliert. Diese Linux-Distribution ermöglicht so einfach und schnell ein Honeypot-System aufzusetzen.

Das große Manko ist hier der veraltete Software-Stand. Die letzte Aktualisierung fand im Jahre 2014 statt. Dies, sowie der Overhead an vorinstallierten und vorkonfigurierten Diensten, birgt die Gefahr, dass ein potenzieller Angreifer über eine Lücke das Hostsystem kompromittieren oder übernehmen kann.

4.2 Kippo

Kippo ist ein SSH-Honeypot, entworfen um Bruteforce-Attacken sowie die komplette Interaktion des Angreifers mit der Shell zu protokollieren. Konnte sich ein Angreifer durch Eingabe der vorbestimmten Kombination aus Benutzer und Passwort einloggen, wird ihm von Kippo ein virtuelles System offengelegt. In diesem System kann der Angreifer wie gewohnt agieren [4].
Um der Anforderung der automatischen Benachrichtigung des Projektteams bei einem Angriff gerecht zu werden, gilt es Logdateien automatisiert zu analysieren und auszuwerten. Wird ein Bruteforce-Angriff erkannt, wird das Projektteam via Email benachrichtigt. Diese Anforderung kann Kippo ohne Anpassungen nicht leisten.

Ein wesentlicher Nachteil von Kippo ist die Tatsache, dass sich Tools zu seiner Erkennung im Umlauf befinden. Entsprechend versierte Angreifer werden Kippo daher frühzeitig erkennen.

4.3 Honeyd

Honeyd wird von unix-artigen Betriebssystemen unterstützt. Er ist ein Daemon, der virtuelle Hosts in einem Netzwerk erzeugt. Diese Hosts können das Vorhandensein spezieller Betriebssysteme und Services simulieren, indem sie mit authentischen Antwortpaketen auf etwaige Anfragen, insbesondere Fingerprint-Pakete reagieren. Honeyd eignet sich besonders für die Ablenkung eines Angreifers und die Verschleierung der wirklichen Infrastruktur. Ebenso dient er als Warnsystem, da jeder Zugriffsversuch auf einem der durch Honeyd erzeugten Hosts ein Hinweis auf unerwünschte Aktivitäten innerhalb der Infrastruktur darstellt [5].

Honeyd eignet sich nicht ohne Weiteres für die Aufzeichnung komplexerer Angriffe, insbesondere solcher, die auf dem System selbst stattfinden, bietet jedoch die Möglichkeit weitere Geräte zu simulieren und somit zur Authentizität der Infrastruktur des Projektteams beizutragen.

4.4 Glastopf

Glastopf ist ein als Webserver getarnter Honeypot. Dieses System nutzt den Umstand, dass viele Angreifer unter Zuhilfenahme von Suchmaschinen nach Schwachstellen auf Webservern suchen, indem es sich selbst bei den gängigsten Vertretern registriert. Dabei wird Fläche für gängige, webbasierte Angriffe wie SQL-Injections, Remote-Code-Execution, File-Inclusion et cetera, geboten. Von einem Angreifer eingeschleuster Code, wird in einer Sandbox ausgeführt. Alle Verbindungen und Angriffsversuche werden geloggt und in einer Datenbank protokolliert [6].

Durch die Mithilfe von Webcrawlern ist es mit Glastopf möglich für die eigenen Schwachstellen Reklame zu betreiben und somit in kürzerer Zeit eine größere Menge an Angriffen auf das System zu lenken. Die optische Aufmachung der Glastopf-Startseite trägt allerdings dazu bei, dass Angriffe in sehr hohem Anteil nur automatisiert und kaum oder gar nicht in individuell gezielter Form stattfinden werden.

5 Lösungsansätze

In den Anforderungen wurde definiert, dass das Honeypotsystem einen, oder besser mehrere Dienste anbieten soll. Um ein realistisches Angriffsziel abzugeben und die Dienste im Internet bereit zu stellen, wird ein Hostsystem mit öffentlicher IP-Adresse auf der Infrastruktur des Rechenzentrums aufgesetzt.
Aufgrund der begrenzten Ressourcen ist die Eigenentwicklung von Honeypotdiensten nicht realisierbar. Ein Lösungsansatz für die Bereitstellung von einem SSH-Honeypot ist der Einsatz von Kippo. Damit wird ein SSH-Dienst, der nach erfolgreichem Login ein virtuelles System simuliert, bereitgestellt. Die von Kippo erzeugten Logfiles gilt es automatisiert auszuwerten. Dies wird mit Hilfe von einem Bash-Skript bewerkstelligt, dass IP-Adressen, Benutzernamen und Passwörter extrahiert. Aus den extrahierten IP-Adressen werden zyklisch Firewallregeln für iptables erzeugt, um zukünftige Angriffe von dieser IP zu blockieren.
Zur Bereitstellung von Diensten wie FTP oder HTTP wird ein separates Honeypot-Werkzeug eingesetzt, dass über eine Evaluation einer Anzahl an in Frage kommenden Systeme ausgewählt wird. Um Logfiles und ausgewertete Daten zu archivieren, werden diese automatisch komprimiert und auf einem Cloud-Speicher wie GoogleDrive abgelegt. Eine Auswertung, aber auch Aufbereitung von Systemweiten Logfiles erfolgt mit Werkzeugen wie Logwatch oder Graylog. Eine Benachrichtigung im Falle eines Zugriffs auf die Honeypotdienste kann mit dem Werkzeug “inotifywait” umgesetzt werden. Damit lassen sich Logfiles oder anderen Daten und Verzeichnisse auf Änderungen prüfen, um daraufhin eine Aktion wie den Versand einer Benachrichtigungsemail anzustoßen.
Um das System bestmöglich abzusichern werden nicht benötigte Dienste deaktiviert. Der Zugriff auf das Hostsystem für das Projektteam erfolgt via Public-Key-Authentifizierung über SSH. Dieser SSH-Dienst ist unabhängig vom Honeypot-SSH-Dienst.

6 Implementierung

6.1 Hostsystem

6.1.1 Grundkonfiguration

Als Hostsystem kommt ein Debian Jessy (Version 8.6) 64-Bit zum Einsatz. Dieses System ist ein virtuelles System, welches auch als vServer bezeichnet wird. Bereitgestellt wird dieser vServer von providerdienste.de[1] mit folgenden Eigenschaften:

- 2 CPU-Kerne a 2,67 GHz
- 2 GB Arbeitsspeicher
- 50 GB Festplatte
- 1 IPv4-Adresse

Das System wurde mit diesen Eigenschaften gewählt, um sicherzustellen, dass für diesen Versuchsaufbau ausreichend Rechenleistung und Speicher zu Verfügung steht. Zudem ist die öffentliche IPv4-Adresse zu nennen, die den Teammitgliedern in einem Versuchsaufbau in einem privaten Heimnetzwerk aus technischen Gründen nicht zur Verfügung gestanden hat. Die Verwaltung des Servers erfolgt zunächst über SSH über einen Root-Zugang, der von providerdienste.de bereitgestellt wird. Alternative ist eine Verwaltung über eine sogenannte Remote-Konsole möglich. Über diese Konsole kann das System jederzeit gestartet und gestoppt werden oder auch das root-Passwort neu gesetzt werden, selbst dann wenn kein Zugriff via SSH zu Verfügung steht. Zudem kann eine Neuinstallation ausgeführt werden.

Da das System als installiertes System mit funktionsfähiger Netzwehrkonfiguration übergeben wurde, ist der erste Schritt eine Aktualisierung der installierten Pakete. Dazu werden die Paketlisten neu eingelesen und neue Paketversionen installiert:

```
apt-get update && apt-get dist-upgrade
```

Um nicht ausschließlich mit root-Rechten zu arbeiten wird für jedes Teammitglied ein eigener Benutzer mit Homeverzeichnis eingerichtet und ein vordefiniertes Passwort gesetzt:

[1] `https://www.providerdienste.de/`

```
useradd -d /home/mstroh -s /bin/bash -m mstroh
passwd mstroh
useradd -d /home/dschwenk -s /bin/bash -m dschwenk
passwd dschwenk
```

Wird bei der Installation von Debian ein root-Passwort angegeben, wird das Programm *sudo* nicht installiert. Dies ist bei dem uns vorliegen vServer der Fall. Um den Benutzern ohne den Wechsel zum Benutzer root die Ausführung von Programmen mit privilegierten Berechtigungen zu ermöglichen, wird das Programm *sudo* über *apt-get install sudo* installiert. Die Benutzer werden in die Gruppe *sudo* aufgenommen:

```
adduser mstroh sudo
adduser dschwenk sudo
```

Zudem müssen die Benutzer in die Datei /etc/sudoers aufgenommen werden:

```
dschwenk ALL=(ALL:ALL) ALL
mstroh ALL=(ALL:ALL) ALL
```

Die Benutzer können somit über *sudo* Aktionen, für die privligierte Berechtigungen notwendig sind, durchführen. Damit ist die Grundkonfiguration des vServers abgeschlossen.

6.1.2 Konfiguration Serverdienste

Das System wurde von providerdienste.de mit einem aktiven SSH-Dienst ausgeliefert. Da der Login eines root-Benutzers über SSH aus Sicherheitsgründen vermieden werden sollte, wird nachfolgend eine Public-Key-Authentifizierung für die Benutzeraccounts der Teammitglieder konfiguriert.

Damit eine Public-Key-Authentifizieurng stattfinden kann, muss im jeweiligen Benutzerverzeichnis auf dem Server ein *.ssh*-Verzeichnis mit entsprechenden Berechtigungen angelegt werden.

```
mkdir ~/.ssh
chmod 700 ~/.ssh
```

Auf dem lokalen System der Teammitglieder wird jeweils das Schlüsselpaar erzeugt:

```
ssh-keygen -t rsa -C "Kommentar"
```

In das oben erzeugte *.ssh*-Verzeichnis wird jeweils der öffentliche Schlüssel in die Datei *authorized_keys* abegelegt:

```
cat id_dsa.pub | ssh username@server 'cat >> .ssh/authorized_keys'
```

Die Berechtigungen auf die Datei *authorized_keys* wird jeweils über *chmod 600 /.ssh/authorized_keys* angepasst.

Um die Authentifizierung via Passwort zu deaktivieren wird die die *sshd_config* unter */etc/ssh/* wie folgt angepasst:

```
ChallengeResponseAuthentication no
PasswordAuthentication no
UsePAM no
```

Ebenfalls wird der Parameter *PermitRootLogin* auf *no* gesetzt, um einen zukünftigen Login des root-Benutzers zu deaktivieren. Abschließend wird der SSH-Dienst mit einem

```
/etc/init.d/ssh restart
```

neu gestartet. Von nun an ist nur noch eine Anmeldung über die Benutzeraccounts der Teammitglieder in Kombination mit einer Public-Key-Authentifizierung möglich.

6.2 SSH-Honeypot

6.2.1 Installation und Konfiguration Kippo

Dieses Kapitel beschreibt die Vorgehensweise zur Installation und Konfiguration von einem SSH-Honeypot auf Basis von Kippo. Dieser SSH-Honeypot soll wie für SSH üblich auf Port 22 eingerichtet werden. Da aktuell der standard SSH-Dienst auf Port 22 läuft, muss dieser zuvor angepasst werden. Dazu wird der Port in der Konfigurationsdatei *etc/ssh/sshd_config* auf Port 10022 abgeändert und der Dienst anschließend neu gestartet.

Damit Kippo lauffähig ist, sind einige zusätzliche Pakete notwendig[2]. Diese werden, ebenso wie der git-Client für einen einfachen Download des Kippo-Projekts, installiert:

```
apt-get install python-dev openssl python-openssl python-pyasn1
  python-twisted git
```

Einer Ausführung von Befehlen oder Diensten mit root-Rechten sollte stets wohl bedacht sein und nach Möglichkeit vermieden werden. Die Ausführung des Honeypot-SSH-Dienstes mit root-Rechten oder auch unter einem unserer Benutzer wäre höchst sicherheitskritisch. Ein Angreifer könnte darüber volle Kontrolle über das Hostsystem erlangen. Um diese Gefahr möglichst gering zu halten wird ein separater Benutzer eingerichtet:

```
useradd -d /home/kippo -s /bin/bash -m kippo -g sudo
```

[2] Kippo Abhängigkeiten: `https://github.com/desaster/kippo#requirements`

Um auf einem Linux-System einen Port kleiner 1024 („well known ports“) zu verwenden sind root-Rechte erforderlich. Genau dies soll für den SSH-Honeypot-Dienst wie oben beschrieben vermieden werden. Um auch einem normalen Benutzer die Verwendung eines Ports kleiner 1024 zu ermöglichen, wird auf das Programm *AuthBind*[3] zurückgegriffen. Die Installation von Authbind erfolgt via:

```
apt-get install authbind
```

Die Verwendung von Port 22 wird über die Erstellung einer Datei unter */etc/authbind/byport/* sowie die Anpassung der Berechtigungen für den Kippo-Benutzer auf diese Datei ermöglicht:

```
touch /etc/authbind/byport/22
chown kippo /etc/authbind/byport/22
chmod 777 /etc/authbind/byport/22
```

Der Download von Kippo erfolgt direkt von der Projektseite auf Github[4]:

```
git clone https://github.com/desaster/kippo.git
```

Im Kippo-Verzeichnis befindet sich eine Datei, die eine Standardkonfiguration enthält. In dieser wird der voreingestellte Port auf Port 22 abgeändert. Zudem muss die Konfigurationsdatei in *kippo.cfg* umbenannt werden:

```
mv kippo.cfg.dist kippo.cfg
```

Damit Kippo mit Hilfe von AuthBind ausgeführt wird, muss das „Kippo-Start-Skript“ angepasst werden. Dazu wird der Befehl *authbind* in das Skript aufgenommen:

```
authbind --deep twistd -y kippo.tac -l log/kippo.log --pidfile kippo.pid
```

Der Parameter *–deep* sorgt dafür, dass nicht nur das direkt folgende Programm, sondern auch alle Programme die folge dieses Aufrufs sind, unter *authbind* ausgeführt werden. twistd, xy, wird zudem eine Logdatei sowie ein Pidfile übergeben. In diesem wird die Prozess-ID abgelegt.
Parameter erklären ...

Nach Ausführung des Kippo-Startskript *start.sh* läuft der Prozess im Hintergrund. In Folge dessen wird auch das Kippo-Logfile angelegt, in dem Zugriffe auf den SSH-Honeypot-Dienst dokumentiert werden. Änderungen in diesem Logfile können über

```
tail -f /home/kippo/kippo/log/
```

[3] *AuthBind*: `http://man.cx/authbind(1)`
[4] *Kippo-Projekt auf Github*: `https://github.com/desaster/kippo`

direkt verfolgt werden. Von nun an kann auch eine Verbindung auf Port 22 aufgebaut werden. Nicht zu vergessen ist, dass der SSH-Dienst auf Port 10022, der die Verbindung der Projektmitarbeiter ermöglicht, durch einen Portscanner wie NMap aufgespürt werden kann.

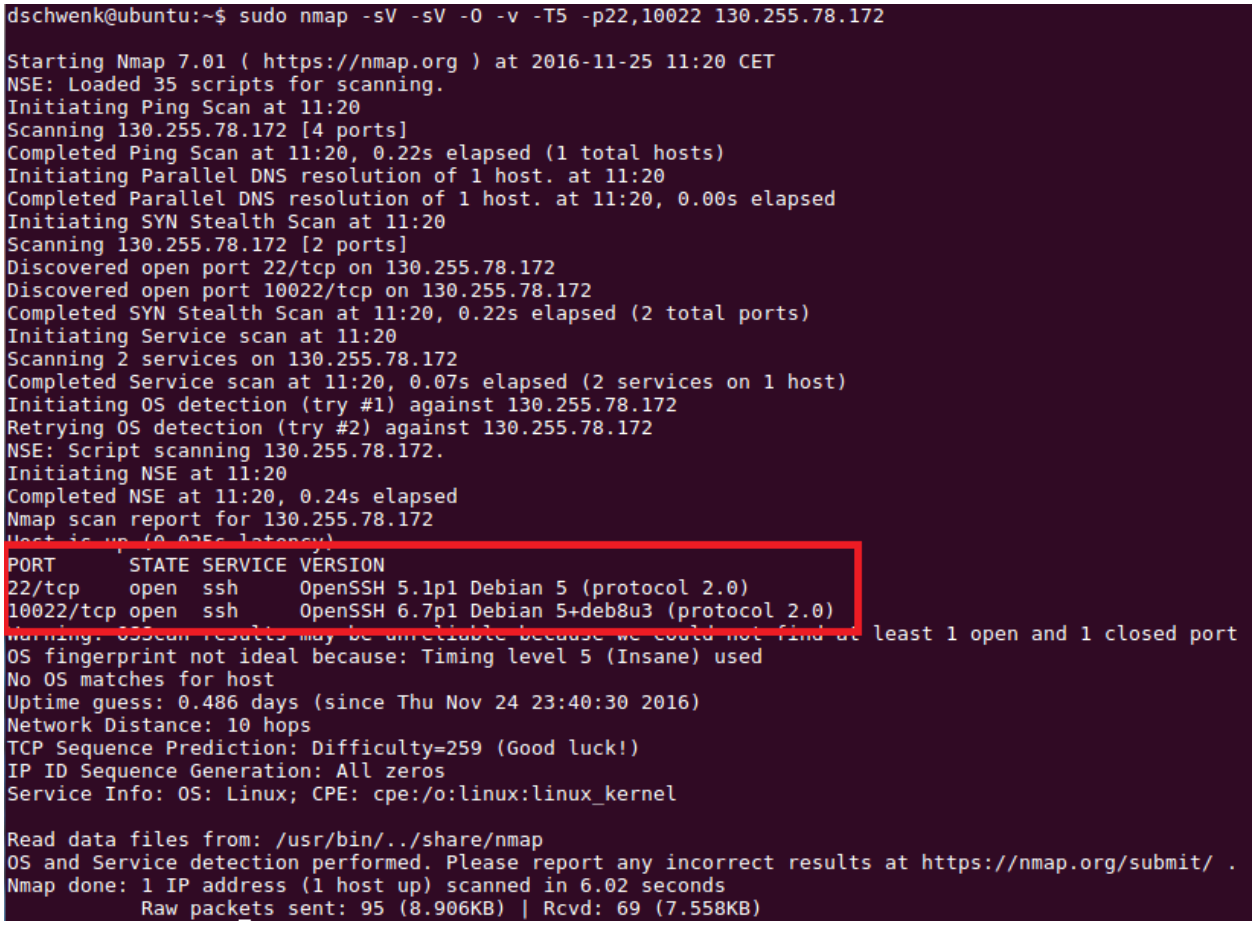


Abbildung 6.1: Ausgabe Portscanner Nmap gegen das Honeypot-System

Das Kippo-SSH-Banner, welches in Abbildung 6.1 zu erkennen ist, kann über die Kippo-Konfigurationsdatei *kipp.cfg* angepasst werden. In dieser Konfigurationsdatei können ebenfalls folgende Einstellungen vorgenommen werden:

- Verzeichnis, in die Kippo eine Logdatei mit allen Aktivitäten schreibt. Alternativ kann eine Datenbank zur Protokollierung verwendet werden. Aus Zeit- und Performancegründen entscheidet sich das Projektteam für die Logdatei.
- x,y ergänzen, siehe
-

Die Kombinationen an Benutzernamen und Passwörter, mit denen ein Login über den SSH-Honeypot möglich ist, kann in der Datei *userdb.txt* spezifiziert werden. Standardmäßig ist

hier der Benutzername *root* mit dem Passwort *123456* hinterlegt. Um einem Angreifer eine größere Angriffsfläche zu bieten, wird diese List um folgende Einträge ergänzt:

```
root:123456
root:root
root:r00t
admin:123456
admin:admin
admin:password
```

Damit ist die Installation und grundlegende Konfiguration des SSH-Honeypots abgeschlossen.

6.2.2 Kippo-Logfile auswerten

Die IP-Adressen von Angreifern werden im Kippo.log-Logfile neben zahlreichen Informationen wie eingegeben Benutzernamen, Passwörter und Befehle gespeichert. Ein Auszug aus einem Kippo-Logfile ist im Anhang unter x zu finden. Aus diesen Informationen sollen Statistiken zu Benutzernamen und Passwörter sowie automatisiert Firewallregeln erstellt werden, um Angriffe von diesen IPs zu verhindern.
Wie dem Kippo-Logfile unter x.y zu entnehmen ist, werden Benutzernamen und Passwörter als Teile von Zeichenketten im Logfile abgelegt. Für eine Auswertung müssen diese aus dem Logfile extrahiert werden. Dies erfolgt mit Hilfe der Werkzeuge *grep* und *awk*. Duplikate werden durch *sort* und *uniq* entfernt.

```
grep ' login attempt ' kippo.log |
  awk '{print ($9)}' |
  sort |
  uniq |
  sed -r 's/]|\[//g' > user.txt
```

Hierdurch erhalten wir eine Liste von Kombinationen aus Benutzernamen und Passwörter in der Form *username/passwort*. Passwörter werden zudem separat ohne Benutzernamen extrahiert. Passwortduplikate werden hierbei nicht entfernt, um daraus aussagekräftige Statistiken generieren zu können.

```
grep ' login attempt ' kippo.log |
  awk '{print ($9)}' |
  sed "s|^.*/||g" |
  sed "s|]||g" > pw.txt
```

Die Ausführung dieser Befehle wird über ein Bash-Skript realisiert. Das vollständige Skript ist im Anhang unter x.y zu finden. Die Auswertung der Passwörter erfolgt über *Pipal Password Analyzer*[5]. Dieses open source Werkzeug erstellt Statistiken über die am häufigsten

[5] *Pipal Password Analyzer*: `https://github.com/digininja/pipal`

eingegeben Passwörter, über Zusammensetzung der Passwörter aus verschiedenen Zeichenklassen sowie Passwortlänge. Zudem erstellt es dazu „Text-Grafiken“. Der *Pipal Password Analyzer* basiert auf *ruby*, was durch *apt-get install ruby* installiert wird. Der Download des Werkzeugs selbst erfolgt von der Projektseite auf Github:

```
git clone https://github.com/digininja/pipal.git
```

Anschließend können Statistiken via

```
pipal.rb /pfad/zur/passwortdatei Ausgabedatei
```

erzeugt werden. Eine dieser Statistiken ist im Anhang unter x.y zu finden.

6.2.3 Firewallregeln erstellen

Ziel unseres System ist es, einen Angreifer zu beobachten und aus seinem Vorgehen zu lernen. Anschließend soll der Angreifer von der Infrastruktur fern gehalten werden, um die Sicherheit anderer Systeme zu wahren. Um einen Angreifer wirkungsvoll von einer Infrastruktur fernzuhalten, besteht die Möglichkeit den Datenverkehr des Angreifers mit einer Firewall, die dieser Infrastruktur vorgelagert ist, zu blockieren. Da in dem vorliegenden Versuchsaufbau keine weiterreichende Infrastruktur mit einer vorgelagerten Firewall vorhanden ist, wird hier exemplarisch auf dem Honeypotsystem selbst die Abwehr der Datenpakete des Angreifers mit Hilfe von *iptables* vorgenommen. Die Wahl fällt auf *iptables*, da hiermit Firwallregeln über sogenannte Ketten von Regeln erstellt werden können. Zudem ist *iptables* standardmäßig unter Debian verfügbar und kann über ein Bash-Skript automatisiert werden.
Um Firewallregeln generieren zu können, müssen die IP-Adressen der Angreifer aus dem Kippo-Logfile extrahiert werden. Dies geschieht wie bereits unter 6.2.2 beschrieben mit Hilfe der Werkzeuge *grep*, *sort* und *uniq*. Dazu wird an *grep* ein regulären Ausdruck übergeben, der IP-Adressen filtert. Damit keine identischen Firewallregeln erzeugt werden, werden doppelte IP-Adressen entfernt.

```
cat logfile.log |
  grep -o '[0-9]\{1,3\}\.[0-9]\{1,3\}\.[0-9]\{1,3\}\.[0-9]\{1,3\}' |
  sort |
  uniq > unique-ips.txt
```

Die Generierung der Firwallregeln geschieht über nachfolgendes Skript:

```
#!/bin/bash
#
# set variables
FW="/sbin/iptables"
IPFILE="/home/dschwenk/kippo-data/ips.txt"
BACKUPFILE="/home/dschwenk/firewall/iptables-backup.txt"
```

```
# backup current rules
iptables-save > $BACKUPFILE

# delete existing rules and chains
$FW -F
$FW -X

# set standard rules
$FW -P INPUT   ACCEPT
$FW -P FORWARD ACCEPT
$FW -P OUTPUT  ACCEPT

while read IP; do
  $FW -A INPUT -s $IP -j DROP
done < $IPFILE
```

Das Skript fertig zuerst über iptabales-save ein Backup der bestehenden Regeln an, welches wie unter **??** beschrieben archiviert wird. Eine beispielhafte Ausgabe von iptables-save ist im Anhang unter x.y zu finden. Anschließend werden alle existierenden Regeln gelöscht. Dies geschieht explizit, um mögliche false positives, die durch dynamische IP-Adressen zustanden kommen können, nicht dauerhaft zu blockieren. Ebenfalls explizit werden über die Regeln standardmäßig alle Verbindungen akzeptiert. Damit lassen sich aus den Verbindungs-Logfiles beispielsweise Portscans nachweisen.
Dieses Skript wird über einen cronjob einmal täglich ausgeführt.

6.3 Web-Honeypot

...

6.4 Archivierung und Automatisierung

- passende überschriften finden
- archivieren von logfiles
- benachrichtigung bie angriff
- reverse lookup ip -adressen (+ darstellung auf karte? geo-ip?

6.4.1 Archivierung von Daten auf Cloud-Speicher

Um der Soll-Anforderung nach der Archivierung von Logdaten sowie Analyseergebnisse zu entsprechen, werden diese Daten automatisiert auf einem Cloud-Speicher abgelegt. Dieses Archivierung stellt sicher, dass auf diese Daten selbst dann zurückgegriffen werden kann, wenn das System kompromittiert wurde oder nicht weiter zur Verfügung steht. Eine Marktanalyse ergibt eine Vielzahl an verfügbaren Cloud-Speichern. Das Projektteam entscheidet sich auf Grund der Verfügbarkeit eines Linux-Clients, der einfachen Installation und Konfiguration dieses Clients sowie dessen Möglichkeit zur automatischen Ausführung via Kommandozeile für Google Drive[6]. Anzumerken ist, dass dieser Client nicht von Google selbst, sondern von Petter Rasmussen in einem open source Projekt entwickelt wird[7]. Der Download der aktuellen Version 2.1 des 64-Bit-Clients für Linux erfolgt durch:

```
wget https://docs.google.com/uc?id=0B3X9GlR6EmbnQ0FtZmJJUXEyRTA&export=
    download
```

Durch eine Umbennung wird der kryptische Dateiname lesbar gemacht. Zudem wird die Datei als ausführbar markiert:

```
mv 0B3X9GlR6EmbnQ0FtZmJJUXEyRTA&export gdrive
chmod +x gdrive
```

Die Installation erfolgt via:

```
sudo install gdrive /usr/local/bin/gdrive
```

Nach der erfolgreichen Installation des Clients müssen diesem Berechtigungen zum Zugriff auf einen Google Drive Account eingerichtet werden. Dazu wird der Client mit einem beliebigen Parameter aufgerufen:

```
gdrive list
```

Infolge dessen wird eine Aufforderung zum Besuch der Google-Drive-Website zur Authentifizierung ausgegeben. Somit ist eine Verbindung zwischen dem Client und dem Google Drive-Dienst hergestellt. Das Backup selbst wird über das Skript unter 6.4.2 erstellt und hochgeladen.

6.4.2 Automatisierung

In den vorherigen Kapitel wurde die Vorgehensweise zur Extraktion, Auswertung und Weiterverarbeitung von Daten erläutert. Für einzelne Aufgaben wurden dazu Skripte erstellt. Um diese Skripte nun nicht regelmäßig von Hand ausführen zu müssen, werden diese ein-

[6] Google Drive: `https://www.google.com/intl/de_de/drive/`
[7] Peter Rasmussen gDrive command line tool: `https://github.com/prasmussen/gdrive`

zelne Skripte über ein zentrales Skript gesteuert und täglich ausgeführt. Dieses Skript führt folgende Aufgaben aus:

- Daten aus Kippo-Logdatei extrahieren
- extrahierte Daten auswerten
- bestehende Firewallregeln sichern sowie neue Regeln erstellen
- x.y
- Archivierung von Logdaten sowie Analyseergebnissen

In die Archivierung fließen sämtliche Log- und Analysedaten sowie Konfigurationsdateien ein. Dazu gehören:

- Kippo-Konfiguration und Logdatei
- Auswertung der Benutzer- und Passwortdaten
- Auswertung der IP-Adressen
- Konfigurationsdateien von iptables
- x.y

Für einen effizienten Upload werden die oben genannten Dateien alle 24 Stunden zu einem komprimierten zip-Archiv zusammengefasst und auf den Cloud-Speicher hochgeladen. Um diesen Vorgang zu automatisieren wird folgendes Bash-Skript angelegt:

```
#!/bin/bash
#
# set variables
DATE=‘date +%Y%m%d_%H-%M‘
BACKUP_FILENAME="/home/dschwenk/backup/"$DATE"_backup.tar.gz"

BACKUP_KIPPO_LOG="/home/kippo/kippo/log/kippo.log"
BACKUP_KIPPO_CONF="/home/kippo/kippo/kippo.cfg"
BACKUP_KIPPO_DATA="/home/dschwenk/kippo-data"

PIPAL_OUTPUT="/home/dschwenk/pipal/pipal-stats.txt"

BACKUP_FIREWALL="/home/dschwenk/firewall/iptables-backup.txt"


# start Kippo logfile data extract
/home/dschwenk/kippo-data/extract-logifile-data.sh
```

```
# do Pipal password analysis
/home/dschwenk/pipal/pipal.rb --output $PIPAL_OUTPUT /home/
   dschwenk/kippo-data/pw.txt

# do firewall stuff
/home/dschwenk/firewall/create-iptable-rules.sh

# do backup stuff
# create zip
tar czPf $BACKUP_FILENAME $BACKUP_KIPPO_LOG $BACKUP_KIPPO_CONF
   $BACKUP_KIPPO_DATA $PIPAL_OUTPUT $BACKUP_FIREWALL

# upload backup
gdrive upload $BACKUP_FILENAME
```

Listing 6.1: Skript zur Extraktion von Benutzernamen und Passwörter aus Kippo-Logfile

Dieses Skript soll täglich ausgeführt werden. Dies kann über einen *cronjob* erreicht werden. Da für die Konfiguration von Firewallregeln via *iptbles* root-Berechtigungen notwendig sind, muss dieses Skript mit privilegierten Berechtigungen ausgeführt werden. Über *sudo crontab -e* wird ein *cronjob* für den root-Benutzer eingerichtet:

```
@daily /home/dschwenk/do_all_stuff.sh
```

Damit wird das Skript einmal täglich ausgeführt.

7 Evaluation

Die unter 3.1 aufgeführten Muss-Kriterien wurden vollständig erfüllt. In diesem Projekt wurde ein Honeypotsystem auf einem von providerdienste.de bereitgestellten vServer umgesetzt. Dieser vServer ist über eine öffentliche IPv4-Adresse im Internet erreichbar und implementiert ein SSH- und Webhoneypot. Durch die von providerdienste.de bereitgestellte Konsole ist es möglich, das System selbst dann, wenn keine Verbindung via SSH möglich ist oder der Server kompromittiert oder gar übernommen wurde, zu konfigurieren und darauf zuzugreifen. Durch die Aktualisierung aller Pakete und der Limitierung der SSH-Authentifzierung auf ein Public-Key-Verfahren wurde das Honeypotsystem bestmöglich abgesichert. Jeglicher Datenverkehr wird geloggt.

Das Soll-Kriterium der automatischen Benachrichtigung bei einem Angriff auf das Honeypotsystem stellte sich im Nachhinein als nicht sonderlich hilfreich dar. Speziell auf Port 22 und somit auf den SSH-Honeypot wurden täglich viele Verbindungen aufgebaut. Die Benachrichtigung darüber war für die Projektmitglieder aber wenig hilfreich, da jegliche Kommunikation und Angriffsversuche geloggt und automatisiert ausgewertet wurden. Zudem werden die auswerteten Daten und Konfigurationen automatisch über eine verschlüsselte Verbindung auf dem Cloud-Speicher Google Drive abgelegt. Die Benachrichtigung bietet so nur einen geringen Mehrwert. Bei der Archivierung der auswerteten Daten sowie dem Backup von Konfigurationsdateien ist anzumerken, dass hier ein separater Server oder ein anderes Backupmedium möglicherweise besser geeignet wäre, da es mit den Richtlinien von Google zu einem Datenschutzkonflikt kommen kann.

Die Kann-Kriterien unter 3.3 wurden nur teilweise erfolgreich umgesetzt. So war es dem Projektteam aus technischen Gründen nicht möglich, neben dem Honeypotsystem weitere Systeme wie einen Router, eine Firewall oder PCs in die Infrastruktur zu integrieren. Auch die Umsetzung eines Honeypots, der ein offenes WLAN oder einen Fake-Acces-Point bietet konnte nicht umgesetzt werden. Hier bietet sich Verbesserungspotential, da die durch ein Honeypot gewonnen Informationen speziell in einer größeren Umgebung wertvoll sind. So ist denkbar, aus den gesammelten IP-Adressen Firwallregeln zu erstellen, die auf einer der Infrastruktur vorgelagerten Firewall zur Wirkung kommen. Die komplette Umsetzung des Projekts fand auf einem gemieteten vServer statt. Die Kosten liegen hier mit 9,00 Euro je Monat in einem überschaubaren Rahmen. Die Anforderung nach einem geringen Stromverbrauch kann nicht exakt beurteilt werden, da über den Stromverbrauch keine Informationen vorliegen. Auf Grund dessen, dass es sich bei unserem System um ein virtuelles System handelt, welches sich ein physikalisches System mit anderen virtuellen Systemen teilt, kann jedoch von einer gewissen Energieeffizienz ausgegangen werden. Die Anforderung an einen

reverse DNS-Lookup wurde wie unter x.y beschrieben umgesetzt und zusätzlich durch eine Darstellung einer Geo-IP-Karte (siehe Anhang x.y) ergänzt

8 Fazit

- dieses Projekt hat uns deutlich gemacht, dass ein Server, der über das Internet erreichbar ist ein beliebtes Ziel ist
- viele Verbindunge in kurzer zeit
- innerhalb kürzester Zeit zig Angriffe auf den SSH-Honeypot
- keine Standardpasswörter
- Passwörter mit mindestens 8, besser 10 oder mehr Zeichen, sowie eine Kombination aus verschiedenen Zeichenklassen wie Groß- und Kleinbuchstaben, Zahlen und Sonderzeichen ist sicher.

- webhoneypot fazit?

- Honeypot ist mit fertigen Honeypot-Konfiguration schnell und einfach in eine bestehende Infrastruktur zu integrieren. Der daraus resultierende Wissensgewinn ist enorm wichtig, um eine Infrastruktur vor potentielle Angriffen abzusichern.

- über Bash-Skripte lassen sich Vorgänge schnell und einfach automatisieren.

Literaturverzeichnis

[1] Nawrocki, Marcin ; Wählisch, Matthias ; Schmidt, Thomas C. ; Keil, Christian ; Schönfelder, Jochen: A Survey on Honeypot Software and Data Analysis. In: *CoRR* abs/1608.06249 (2016). http://arxiv.org/abs/1608.06249

[2] Wikipedia: *Honeypot*. https://de.wikipedia.org/wiki/Honeypot, 2016. – abgerufen: 14.11.2016

[3] HoneyDrive: *Honeypots in a box!* http://bruteforce.gr/honeydrive, 2016. – abgerufen: 10.11.2016

[4] Kippo: *SSH Honeypot*. https://github.com/desaster/kippo, 2016. – abgerufen: 10.11.2016

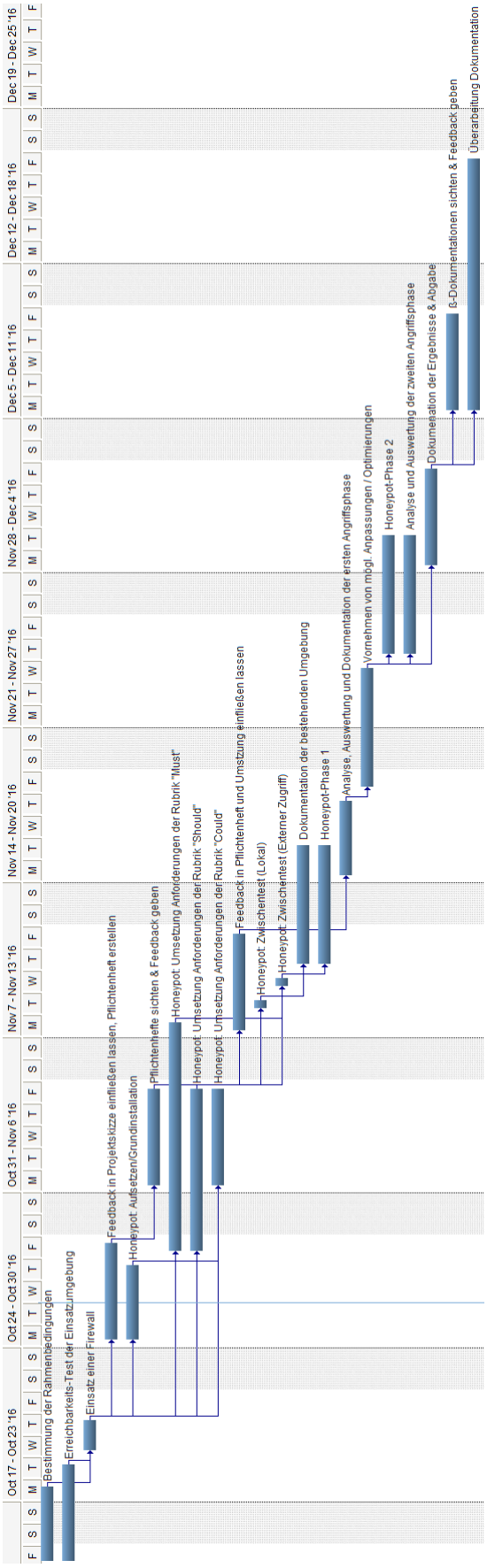
[5] Honeyd: *Developments of the Honeyd Virtual Honeypot*. http://www.honeyd.org/, 2016. – abgerufen: 10.11.2016

[6] Glastopf: *Web Application Honeypot*. http://glastopf.org, 2016. – abgerufen: 10.11.2016

A Anhang

Gantt-Diagramm

	Name	Duration	Start	Finish	Predecessors
1	Bestimmung der Rahmenbedingungen	2d	14/10/2016	17/10/2016	
2	Erreichbarkeits-Test der Einsatzumgebung	3d	14/10/2016	18/10/2016	
3	Einsatz einer Firewall	2d	19/10/2016	20/10/2016	1,2
4	Feedback in Projektskizze einfließen lassen, Pflichtenheft erstellen	5d	24/10/2016	28/10/2016	3
5	Honeypot: Aufsetzen/Grundinstallation	4d	24/10/2016	27/10/2016	3
6	Pflichtenhefte sichten & Feedback geben	5d	31/10/2016	04/11/2016	4
7	Honeypot: Umsetzung Anforderungen der Rubrik "Must"	7d	28/10/2016	07/11/2016	3,5
8	Honeypot: Umsetzung Anforderungen der Rubrik "Should"	6d	28/10/2016	04/11/2016	3,5
9	Honeypot: Umsetzung Anforderungen der Rubrik "Could"	5d	31/10/2016	04/11/2016	3,5
10	Feedback in Pflichtenheft und Umstzung einfließen lassen	5d	07/11/2016	11/11/2016	6
11	Honeypot: Zwischentest (Lokal)	1d	08/11/2016	08/11/2016	7,8
12	Honeypot: Zwischentest (Externer Zugriff)	1d	09/11/2016	09/11/2016	7,8,11
13	Dokumentation der bestehenden Umgebung	4d	10/11/2016	15/11/2016	11,12
14	Honeypot-Phase 1	4d	10/11/2016	15/11/2016	12
15	Analyse, Auswertung und Dokumentation der ersten Angriffsphase	4d	14/11/2016	17/11/2016	10
16	Vornehmen von mögl. Anpassungen / Optimierungen	4d	18/11/2016	23/11/2016	15
17	Honeypot-Phase 2	4d	24/11/2016	29/11/2016	16
18	Analyse und Auswertung der zweiten Angriffsphase	4d	24/11/2016	29/11/2016	16
19	Dokumenation der Ergebnisse & Abgabe	5d	28/11/2016	02/12/2016	16
20	ß-Dokumentationen sichten & Feedback geben	5d	05/12/2016	09/12/2016	19
21	Überarbeitung Dokumentation & Abgabe	10d	05/12/2016	16/12/2016	19


Oct 17 - Oct 23 '16
Oct 24 - Oct 30 '16
Oct 31 - Nov 6 '16
Nov 7 - Nov 13 '16
Nov 14 - Nov 20 '16
Nov 21 - Nov 27 '16
Nov 28 - Dec 4 '16
Dec 5 - Dec 11 '16
Dec 12 - Dec 18 '16
Dec 19 - Dec 25 '16
Bestimmung der Rahmenbedingungen
Erreichbarkeits-Test der Einsatzumgebung
Einsatz einer Firewall
Feedback in Projektskizze einfließen lassen, Pflichtenheft erstellen
Honeypot Aufsetzen/Grundinstallation
Pflichtenhefte sichten & Feedback geben
Honeypot Umsetzung Anforderungen der Rubrik "Must"
Honeypot Umsetzung Anforderungen der Rubrik "Should"
Honeypot Umsetzung Anforderungen der Rubrik "Could"
Feedback in Pflichtenheft und Umsetzung einfließen lassen
Honeypot Zwischentest (Lokal)
Honeypot Zwischentest (Externer Zugriff)
Dokumentation der bestehenden Umgebung
Honeypot-Phase 1
Analyse, Auswertung und Dokumentation der ersten Angriffsphase
Vornehmen von mögl. Anpassungen / Optimierungen
Honeypot-Phase 2
Analyse und Auswertung der zweiten Angriffsphase
Dokumentation der Ergebnisse & Abgabe
ß-Dokumentationen sichten & Feedback geben
Überarbeitung Dokumentation

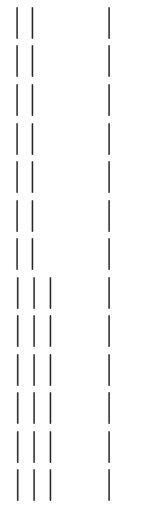
Benutzernamen und Passwörter extrahieren

```
#!/bin/bash
#
# set variables
KIPPO_LOG="/home/kippo/kippo/log/kippo.log"
USERNAME_FILE="/home/dschwenk/kippo-data/usernames.txt"
PW_FILE="/home/dschwenk/kippo-data/pw.txt"
IP_FILE="/home/dschwenk/kippo-data/ips.txt"

# extract usernames
grep ' login attempt ' $KIPPO_LOG |
  awk '{print ($9)}' |
  sort |
  uniq |
  sed -r 's/]|\[//g' > $USERNAME_FILE

# extract passwords
grep ' login attempt ' $KIPPO_LOG |
  awk '{print ($9)}' |
  sed "s|^.*/||g" |
  sed "s|]||g" > $PW_FILE

# extract IPs
cat $KIPPO_LOG |
  grep -o '
     [0-9]\{1,3\}\.[0-9]\{1,3\}\.[0-9]\{1,3\}\.[0-9]\{1,3\}' |
  sort |
  uniq > $IP_FILE
```

Listing A.1: Skript zur Extraktion von Benutzernamen und Passwörter aus Kippo-Logfile

Passwortstatistik

```
Basic Results

Total entries = 7
Total unique entries = 5

Top 10 passwords
Qwe+123- = 2 (28.57%)
asdf = 2 (28.57%)
sdf = 1 (14.29%)
qwe12 = 1 (14.29%)
qwe = 1 (14.29%)

Top 10 base words
asdf = 2 (28.57%)

Password length (length ordered)
3 = 2 (28.57%)
4 = 2 (28.57%)
5 = 1 (14.29%)
8 = 2 (28.57%)

Password length (count ordered)
8 = 2 (28.57%)
3 = 2 (28.57%)
4 = 2 (28.57%)
5 = 1 (14.29%)
```



```
    |||   |
    |||   |
|||||||||
012345678

One to six characters = 5 (71.43%)
One to eight characters = 7 (100.0'%)
More than eight characters = 0 (0.0%)

Only lowercase alpha = 4 (57.14%)
Only uppercase alpha = 0 (0.0%)
Only alpha = 4 (57.14%)
Only numeric = 0 (0.0%)

First capital last symbol = 2 (28.57%)
First capital last number = 0 (0.0%)

Single digit on the end = 0 (0.0%)
Two digits on the end = 1 (14.29%)
Three digits on the end = 0 (0.0%)

Last number
2 = 1 (14.29%)

   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
||||||||||
0123456789

Last digit
```

```
2 = 1 (14.29%)

Last 2 digits (Top 10)
12 = 1 (14.29%)

Character sets
loweralpha: 4 (57.14%)
loweralphanum: 1 (14.29%)

Character set ordering
allstring: 4 (57.14%)
othermask: 2 (28.57%)
stringdigit: 1 (14.29%)
```

Listing A.2: Passwortstatistik generiert durch Pipal Password Analzyer

Selbständigkeitserklärung

Ich versichere, dass ich die vorliegende Dokumentation selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Die Tabellen, Abbildungen oder Listings in dieser Arbeit sind von mir selbst erstellt worden oder mit einem entsprechenden Quellennachweis versehen.

Weingarten, den 02. Dezember 2016

Unterschrift: ....
Michael Stroh Daniel Schwenk