

# 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

<b>Abbkürzungsverzeichnis</b>	<b>ii</b>
<b>1 Einleitung</b>	<b>1</b>
1.1 Motivation . . . . .	1
1.2 Ziele . . . . .	2
1.3 Eigene Leistung . . . . .	2
<b>2 Grundlagen</b>	<b>3</b>
2.1 Honeypot . . . . .	3
<b>3 Anforderungen</b>	<b>5</b>
3.1 Muss-Kriterien . . . . .	5
3.2 Soll-Kriterien . . . . .	6
3.3 Kann-Kriterien . . . . .	6
<b>4 Marktanalyse</b>	<b>7</b>
4.1 HoneyDrive . . . . .	7
4.2 Kippo . . . . .	7
4.3 Honeyd . . . . .	8
4.4 Glastopf . . . . .	8
<b>5 Lösungsansätze</b>	<b>9</b>
<b>6 Implementierung</b>	<b>10</b>
6.1 Hostsystem . . . . .	10
6.1.1 Grundkonfiguration . . . . .	10
6.1.2 Konfiguration Serverdienste . . . . .	11
6.2 SSH-Honeypot . . . . .	11
6.2.1 Installation und Konfiguration Kippo . . . . .	11
6.2.2 Kippo-Logfile auswerten . . . . .	13
6.2.3 Firewallregeln erstellen . . . . .	14
6.3 Web-Honeypot . . . . .	15
6.4 sonstiges . . . . .	15
6.4.1 Archivierung von Daten . . . . .	15
<b>7 Evaluation</b>	<b>18</b>

<b>8 Fazit</b>	<b>19</b>
<b>Literaturverzeichnis</b>	<b>20</b>
<b>A Anhang</b>	<b>21</b>

# Abkürzungsverzeichnis

<b>DNS</b>	Domain Name System
<b>FTP</b>	File Transfer Protocol
<b>HIHP</b>	high-interaction honeypot
<b>HTTP</b>	Hypertext Transfer Protocol
<b>LIHP</b>	low-interaction honeypot
<b>LTS</b>	Long Term Support
<b>MIHP</b>	mediium-interaction honeypot
<b>SSH</b>	Secure Shell
<b>SQL</b>	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 stattgefundenen 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 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 gewonnenen Informationen dienen dabei oftmals nur zu statistischen Zwecken. MIHPs bilden einzelne Dienste erheblich genauer ab. Eine vollständige Kommunikation über den angebotenen 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 Brute-force-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 Brute-force-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 x.y 64-Bit zum Einsatz. Dieses System ist ein virtuelles System welches auch als vServer bezeichnet wird. Bereitgestellt wird dieser vServer von [providerdienste.de](https://www.providerdienste.de/)<sup>1</sup> mit folgenden Eigenschaften:

- 2 CPU-Kerne
- 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 zur 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 über einen Root-Zugang .. Alternative ist eine Verwaltung über eine sogenannte Remote-Konsole möglich. Über diese Konsole kann das System jederzeit gestartet und gestoppt werden, selbst dann wenn kein Zugriff via SSH zur Verfügung steht.

Um das System nach der Übergabe durch [providerdienste.de](https://www.providerdienste.de/) auf einen aktuellen Stand zu bringen, wird ein Update der installierten Pakete durchgeführt. Dazu werden die Paketlisten neu eingelesen und neue Paketversionen installiert:

```
|| apt-get update && apt-get upgrade
```

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

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

---

<sup>1</sup><https://www.providerdienste.de/>

Um diesen Benutzern die Ausführung von Kommandos mit root-Rechten dennoch zu ermöglichen, werden diese Benutzer in die Gruppe *sudo* aufgenommen

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

Damit ist die Grundkonfiguration des vServers abgeschlossen.

### 6.1.2 Konfiguration Serverdienste

- Konfiguration von Diensten wie SSH (Public Key Authentifizierung einrichten)
- beachten, dass Server-SSH-Dienst von Port 22 auf anderen Port gelegt wird (z.b 10022), da Honeypot-SSH-Dienst auf Port 22 laufen sollte
- vielleicht Hinweis, dass über port scanner der normale ssh dienst trotzdem gefunden wird

Das System wurde von providerdienste.de mit einem aktiven SSH-Dienst ausgeliefert. ..

Damit eine Authentifizierung stattfinden kann, muss im Benutzerverzeichnis der Teammitglieder auf dem Server ein *.ssh*-Verzeichnis angelegt werden. In diese Verzeichnisse wird jeweils der öffentliche Schlüssel in die Datei *authorized\_keys* abgelegt.

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

Um die Authentifizierung via Passwort zu deaktivieren wird 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

```
|| service ssh restart
```

neugestartet. 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 via *service sshd restart* neugestartet. Damit Kippo lauffähig ist, sind einige zusätzliche Pakete notwendig<sup>2</sup>. 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
```

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*<sup>3</sup> zurückgegriffen. AuthBind ermöglicht es auch Benutzern auf privilegierte Ports zuzugreifen. 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<sup>4</sup>:

```
|| 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:

---

<sup>2</sup> Kippo Abhängigkeiten: <https://github.com/desaster/kippo#requirements>

<sup>3</sup> *AuthBind*: [http://man.cx/authbind\(1\)](http://man.cx/authbind(1))

<sup>4</sup> *Kippo-Projekt auf Github*: <https://github.com/desaster/kippo>



```
|| 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-Start-Skript“ 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/
```

direkt verfolgt werden. Von nun an kann auch eine Verbindung auf Port 22 aufgebaut werden: Die Anpassung des Banners kann in der Konfigurationsdatei von Kippo vorgenommen 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.

### 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 zeitweise 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 > user.txt
```

Ebenfalls werden Passwörter extrahiert:

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

Passwortduplikate werden hierbei nicht entfernt, um daraus aussagekräftige Statistiken generieren zu können.

- Auswertung der Passwörter (diese müssen zuerst geparkt werden)
- alternativ Kippo Graph

### 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 Firewallregeln 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 Firewallregeln geschieht automatisiert über nachfolgendes Skript:

```
#!/bin/bash
#
# set variables
FW="/sbin/iptables"
IPFILE="/home/dschwenk/ipfile.txt"
BACKUPFILE="/home/dschwenk/backupfile.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 fertigt zuerst über `iptables-save` ein Backup der bestehenden Regeln an, welches wie unter 6.4.1 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 sonstiges

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

### 6.4.1 Archivierung von Daten

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<sup>5</sup>. Anzumerken ist, dass dieser Client nicht von Google selbst, sondern

---

<sup>5</sup> Google Drive: [https://www.google.com/intl/de\\_de/drive/](https://www.google.com/intl/de_de/drive/)

von Petter Rasmussen in einem open source Projekt entwickelt wird<sup>6</sup>. Der Download der aktuellen Version 2.1 des 64-Bit-Clients für Linux erfolgt durch:

```
|| wget https://docs.google.com/uc?id=0B3X9G1R6EmbnQ0FtZmJJUXEyRTA&export=download
```

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

```
|| mv 0B3X9G1R6EmbnQ0FtZmJJUXEyRTA&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.

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

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

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 up variables
```

---

<sup>6</sup> Peter Rasmussen gDrive command line tool: <https://github.com/prasmussen/gdrive>

```
DATE='date +%Y%m%d_%H-%M'
FILENAME=$DATE_backup.tar.gz

# create zip archive
tar xyz $FILENAME

# upload zip archive
gdrive upload $FILENAME
```

Ausgeführt wird dieses Skript täglich um 6:00 Uhr mit Hilfe eines cronjobs. Die Übertragung der Daten erfolgt verschlüsselt. Ein möglicher Angreifer kann so den Datenstrom nicht nachvollziehen.

## 7 Evaluation

- Anforderungen erfüllt (ja / nein - warum) - möglicherweise Raum für Verbesserungen

## 8 Fazit

- abc

# Literaturverzeichnis

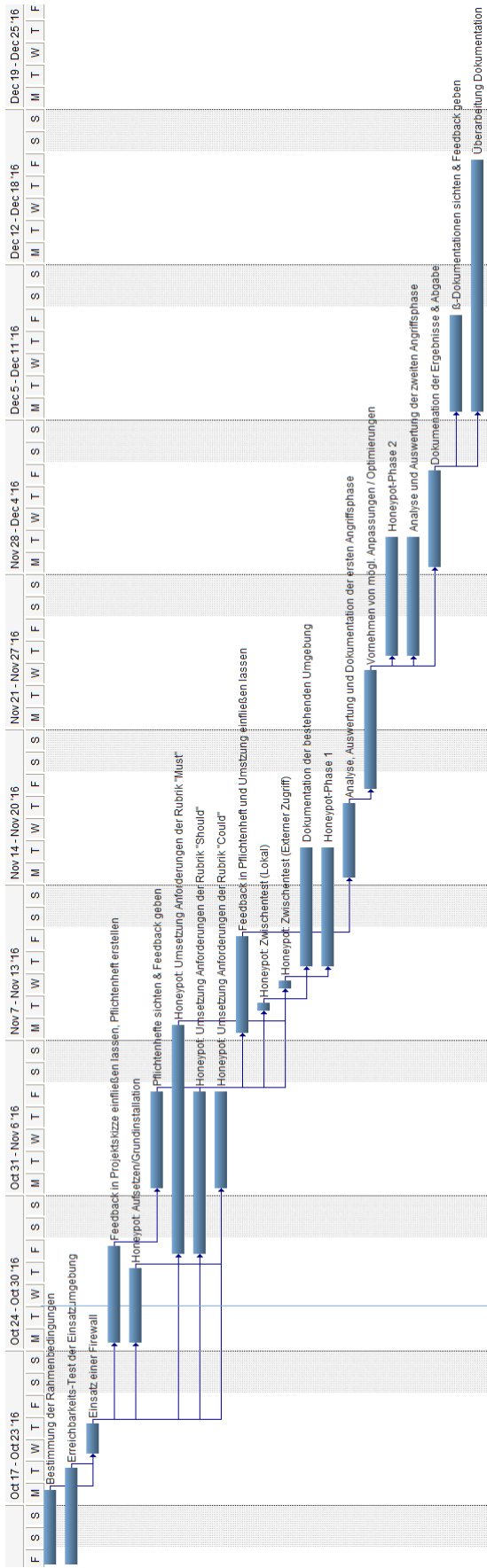
- [1] NAWROCKI, Marcin ; WÄHLISCH, Matthias ; SCHMIDT, Thomas C. ; KEIL, Christian ; SCHÖNFELDER, Jochen: A Survey on Honeypot Software and Data Analysis. In: *CoRR* abs/1608.06249 (2016). <http://arxiv.org/abs/1608.06249>
- [2] WIKIPEDIA: *Honeypot*. <https://de.wikipedia.org/wiki/Honeypot>, 2016. – 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 Umsetzung 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	Dokumentation der Ergebnisse & Abgabe	5d	28/11/2016	02/12/2016	16
20	β-Dokumentationen sichten & Feedback geben	5d	05/12/2016	09/12/2016	19
21	Überarbeitung Dokumentation & Abgabe	10d	05/12/2016	16/12/2016	19



# 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

