Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Ausarbeitung Anwendungen I
WiSe 2010/2011

Theodor Nolte

Mobile Honeypot

Fakultät Technik und Informatik
Department Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Inhaltsverzeichnis

Abstract

A mobile honeypot is an appliance which appears to an attacker as a *normal* mobile device, but is not in normal use. Its only purpose is to be attacked and to document this attack.

In general, the dedication of a honeypot is to record and detect (novel) attacks and also to understand objectives and behaviors of the attacker. However, the context and scenarios of mobile devices differ significantly from conventional end user systems: Hardware resources are rigidly limited, and in many occasions several wireless interfaces of different technologies are active. As personal assistants, a mobile often store sensitive data of its user.

Mobile honeypots can simulate a mobile device (low interaction honeypot) or are implemented as a real mobile device (high interaction honeypot). Currently, only one framework for implementing virtual honeyclients for mobile devices exists: honeyd. Honeyd uses a normal PC and simulates mobile devices.

In this presentation, we introduce objectives and the current state of mobile honeypots. Also, we consider how low interaction honeypots running on real mobile devices can be realized in the future.

1 Einleitung

1.1 Motivation

Die Zielvorgabe des SKIMS-Projekts [6] ist die Konzeption, Entwicklung und Analyse einer schichtenübergreifenden kooperativen Sicherheits-Umgebung für mobile Geräte. Daher liegt der Fokus von SKIMS auf die passive und aktive Verteidigung gegen Angriffe, zunächst jedoch ihrer Detektion. Neben der Analyse des Netzwerk-Verkehrs etwa mittels Entropie-Auswertungen stellen *Mobile Honeypots* ein Instrument dar, (unbekannte) Angriffe zu erkennen.

Das Szenario mobiler Endgeräte unterscheidet sich signifikant von konventionellen Endbenuter-Systemen:

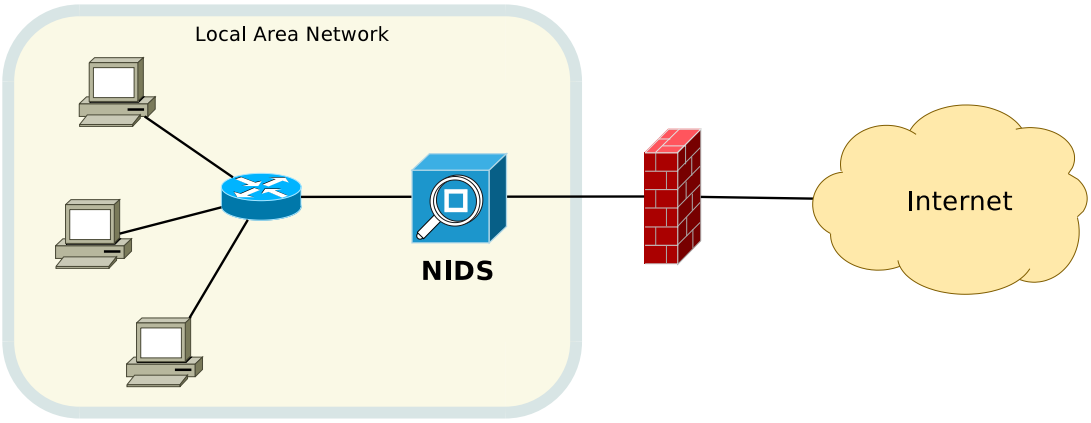
- Die Ressourcen-Verfügbarkeit sowie die Rechenkapazitäten von mobilen Endgeräten sind beschränkt.
- Die Tatsache, dass mobile Endgeräte als ein *Single Spot* privater Daten verwendet werden, macht sie interessant für Angriffe.
- Unterschiedliche Funk-Schnittstellen ermöglichen neue Angriffsformen: Der *„Äther“* ist ein inhärent allen zugängliches Übertragungsmedium, zugleich sind jedoch Angriffe über dieses Medium lokal begrenzt.

Ein *Mobile Honeypot* muss diesen Besonderheiten Rechnung tragen.

1.2 Inhaltlicher Aufbau der Ausarbeitung

In Abschnitt 2 werden der Begriff *Honeypot* und seine Grundlagen erörtert. In Abschnitt 3 werden die Randbedingungen, Anforderungen und die Problemstellung an einen *Mobile Honeypot* behandelt. Ausserdem wird honeyM, ein Framework virtueller Client Honeypots, vorgestellt. In Abschnitt 4 werden die Risiken erörtert. Und in Abschnitt 5 wird diese Ausarbeitung mit einer Zusammenfassung und einem Ausblick abgeschlossen.

Abbildung 1: Network Intrusion Detection System



2 Honeypot

Wie schon einleitend erwähnt dient ein Honeypot dem Ziel, Angriffe auf ein Local Area Network (LAN) oder auf ein einzelnen Rechner zu erkennen und diese gegebenenfalls zu analysieren, um auf den so gewonnenen Informationen aufbauend, wirksame Gegenmaßnahmen ergreifen zu können. Ein Honeypot ist also ein Instrumentarium zur Verteidigung vor Netzwerkangriffen.

Bevor das Konzept des Honeypots etabliert wurde, wurden in lokalen Netzwerken *Network Intrusion Detection Systems* eingesetzt, um Netzwerkangriffe zu erkennen. Ein *Network Intrusion Detection System (NIDS)* ist ein System, welches an zentraler Stelle den Netzwerkverkehr in Echtzeit überwacht und analysiert. In Abbildung 1 ist ein NIDS so positioniert, dass es sämtlichen Netzwerkverkehr vom und zum lokalen Netzwerk (Local Area Network, kurz: LAN) verfolgen kann. Das NIDS zeichnet (möglichst) sämtliche Netzwerk-Pakete auf und analysiert sie anschließend. Klassifiziert es Pakete bzw. das dahinter stehende Kommunikationsmuster als verdächtig, so wird dies geloggt und es wird ein entsprechender Alarm gegeben (z.B. per E-Mail oder Nagios[1]). Eine beispielhafte Realisierung für ein NIDS ist Bro [2].

Jedoch weist ein NIDS konzeptuell bedingt signifikante Nachteile auf:

- Mit zunehmender Bandbreite des Netzwerkverkehrs kann möglicherweise die Performance des NIDS nicht ausreichen, was zu Folge hat, dass nicht alle Pakete analysiert werden und somit keine lückenlose Überwachung erfolgt.
- Netzwerkpakete mit verschlüsseltem Datenteil können nicht analysiert werden. Dies ist erst an einem Endpunkt der verschlüsselten Kommunikation möglich.

[1] Offizielle Webseite von Nagios: http://www.nagios.org

- Die Tatsache, dass die Netzwerkkommunikation eigentlich für produktive Zwecke eingesetzt wird, birgt die gefahr von *False Positives*, d.h. das nicht bösartige Netzwerkkommunikation fälschlicherweise vom NIDS als bösartig klassifiziert wird.
- Ein NIDS benötigt ein Vorwissen, um die Netzwerkkommunikation zu bewerten. Neue, bisher unbekannte Angriffe, die bisher bekannte Kommunikationsmuster vermeiden, werden nicht als bösartig erkannt.

Die Ursache für diese Nachteile liegt darin, dass das Ende-zu-Ende Argument [4] von einem NIDS verletzt wird. Das Ende-zu-Ende Argument lautet im originalen Wortlaut:

> *"The function in question can completely and correctly be implemented only with the knowledge and help of the application standing at the endpoints of the communication system. Therefore, providing that questioned function as a feature of the communication system itself is not possible. (Sometimes an incomplete version of the function provided by the communication system may be useful as a performance enhancement.)"*

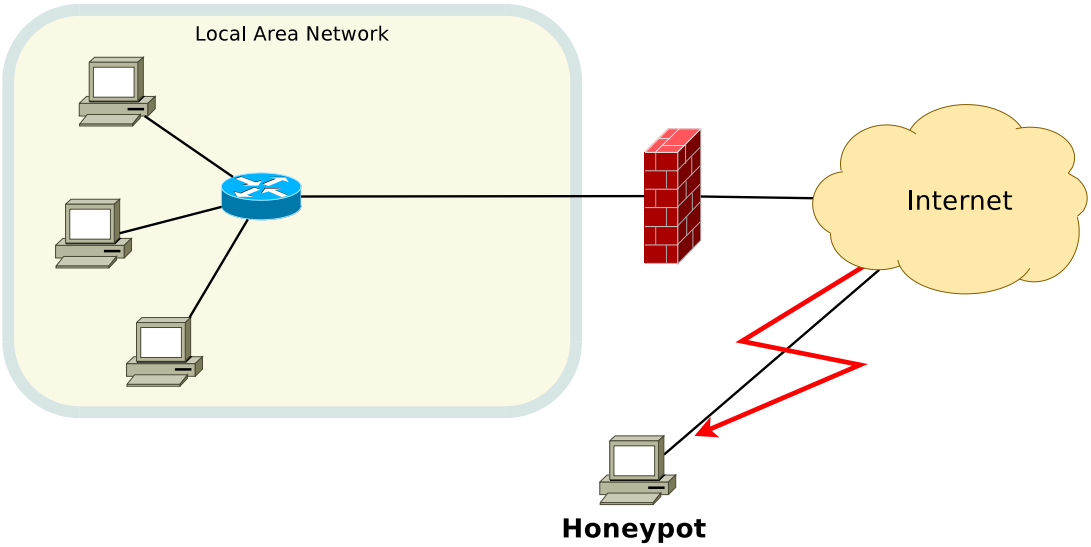
Es besagt, dass eine Funktionalität innerhalb der Netzwerk-Kommunikation nur an ihren Endpunkten implementiert werden kann, weil nur dort die für diese Funktionalität benötigten Informationen vollständig vorliegen; in manchen Fällen mag es sich jedoch aus Gründen der Performanz lohnen, den Kompromiss einzugehen, eine unvollständige Funktionalität bereits innerhalb des Kommunikationsnetzes, d.h. im Protokoll-Stack (OSI) auf Höhe der Netzwerk-Ebene (oder niedriger) anzusiedeln. Auf das Problem des Erkennens von bösartiger Netzwerkkommunikation angewandt bedeutet das Ende-zu-Ende Argument, dass nur an einem Endknoten der Kommunikation alle Informationen (im Klartext) vorliegen. Genau dieser Umstand ist bei einem Honeypot im Gegensatz zu einem NIDS gewährleistet.

Im Buch *Virtual Honeypots* von Niels Provos und Thorsten Holz [3] wird ein Honeypot beschrieben als ein (End-)System, dass nicht produktiv genutzt wird. Abgesehen von den Paketen, die durch die Bereitschaft üblicher Dienste entstehen (z.B. telnet, ssh, http), ist jedwede Netzwerkkommunikation mit dem Honeypot *verdächtig* und kann a-priori als *bösartig* eingestuft werden. Somit ermöglicht ein Honeypot einen ungetrübten Blick auf das Verhalten eines Angreifers – der Angriff liegt „im Klartext“ vor. In Abbildung 2 ist ein Honeypot dargestellt. Durch die direkte Anbindung an das Internet kann er viele Angriffe „ernten“, da die überwiegende Mehrheit von Angriffen über das Internet erfolgt.

Honeypots können unterschieden werden anhand unterschiedlicher Kriterien. Ein **physikalischer Honeypot** wird durch einen realen Rechner repräsentiert, also mit eigener Hardware, einem eigenen Prozessor und einer eigenen Netzwerkschnittstelle mit einer eigenen Netzwerkadresse. Hingegen stellt ein **virtueller Honeypot** ein logisch eigenständiges System dar, welches durch einen anderen Rechner in einer virtuellen Umgebung wie VMware[2] oder User Mode Linux (UML)[3] simuliert wird.

[2] Offizielle Webseite von VMware: http://www.vmware.com
[3] Offizielle Webseite von UML: http://user-mode-linux.sourceforge.net

Abbildung 2: Honeypot



Ein **Client Honeypot** verhält sich wie ein normaler Client-Rechner und besucht beispielsweise selbsttätig Webseiten, um von schadhaften Webservern angegriffen zu werden. Ein **Server Honeypot** dagegen bietet Services an, wie sie ein üblicher produktiv genutzter Server anbietet, etwa einen Webserver oder einen Terminalzugang mittels ssh.

Die für uns hier wichtigste Unterscheidung ist die zwischen einem *Low-Interaction Honeypot* und einem *High-Interaction Honeypot*. Ein **High-Interaction Honeypot** ist ein echtes System mit echten Diensten bzw. echter Client-Funktionalität. Solch ein Honeypot kann daher auch „richtig“ angegriffen werden. Und solch ein Angriff kann dann auch ausführlich und vollständig analysiert werden. Allerdings ist der Betrieb eines *High-Interaction Honeypots* aufwendig. Denn man muss ihn ständig überwachen und sollte einen Angriff schließlich beenden (indem man den Honypot vom Netz trennt) bevor der Honypot selber als Basis für weitergehende Angriffe dient. Und nach einem Angriff muss der High-Interaction Honeypot wieder in ein unkompromittiertes System zurückgeführt werden sprich, es bedarf einer Neuinstallation. Der Betrieb eines *High-Interaction Honeypots* birgt zudem die Gefahr, dass ein Angreifer ihn als einen solchen erkennt und umgehend den Angriff abbricht oder gar einen Scheinangriff weiterführt, um den Betreiber des Honeypots mit Falschinformationen zu verwirren.

Bei einem **Low-Interaction Honeypot** hingegen wird die Funktionalität eines produktiven Systems nur nachgebildet (was bedeutet, dass ein *Low-Interaction Honeypot* virtuell ist). So wird beispielweise bei einem ssh-Terminal nur die Drei-Wege-Handshake Funktionalität nachgebildet, jedoch kommt anschließend keine verschlüsselte Kommunikation zustande. Die meisten Angriffe erfolgen automatisiert und sind in ihrer Funktionalität beschränkt – sie zielen ab

auf die „Low Hanging Fruits“. Dafür reicht es aus, diese Angriffe mit gerade ausreichender Funktionalität auf der Gegenseite zu täuschen. Ein *Low-Interaction Honeypot* birgt somit ein geringeres Risiko als ein *High-Interaction Honeypot*. (Allerdings muss man bedenken, dass das Host-System, welches den *Low-Interaction Honeypot* betreibt, selber aufgrund von Schwachstellen angegriffen werden kann.) *Low-Interaction Honeypots* lassen sich einfacher automatisiert betreiben, als das sie wieder in ihren Ausgangszustand zurückgeführt werden können, nachdem eine (bösartige) Kommunikation beendet wurde, ohne das eine Neuinstallation oder das Rückspielen eine Images notwendig wird. Aufgrund des automatischen Betriebs werden oft mehrere *Low-Interaction Honeypots* zu einem **Honeynet** zusammengefasst. Solch ein *Honeynet* kann als ein *Sensornetz* betrieben werden, um die Aktivitäten von automatisierten Angriffen, wie sie mittels Bots erfolgen, zu monitoren. Diese Art des Einsatzes stellt auch die größte Verbreitungsform von Honeypots dar.

Die „White-Hats“ und „Black-Hats“ befinden sich in einem ständigen Wettstreit miteinander. So wissen die „Black-Hats“ natürlich, dass es Honeypots gibt, und versuchen diese mit neuen Angriffen auszutricksen. Dies führt wiederum dazu, dass die „White-Hats“ ihre *Low-Interaction Honeypots* in ihrer simulierten Funktionalität erweitern, um auch diese Angriffe wieder erfolgreich nachverfolgen zu können. Ab einem bestimmten Grad der Fülle an nachgebildeten Funktionalitäten spricht man dann von einem **Mid-Interaction Honeypot**.

Wie schon mit dem *Mid-Interaction Honeypot* angedeutet, sind die aufgeführten Unterscheidungen nicht strikt, sie schließen sich auch nicht gegenseitig aus.[4] Man kann hieran erkennen, dass das Fachgebiet Computersicherheit im Ursprung keine akademische Wissenschaft darstellt, sondern aus dem praktischen Alltag der ständigen Abwehr von Computerangriffen gewachsen ist, eine exakte Klassifizierung mitunter *schwierig* und mitunter nicht praktikabel ist.

3 Anforderungen und Problemstellung an einen Mobile Honeypot

Nachdem wir im vorigen Abschnitt uns allgemein mit dem Begriff Honeypot beschäftigt haben, erörtern wir die Randbedingungen mobiler Geräte. Anschließend wird honeyM vorgestellt. Schließlich leiten wir daraus die Anforderungen und die Problemstellung an einen *Mobile Honeypot* ab.

3.1 Randbedingungen mobiler Endgeräte

Ein mobiles Endgerät wie z.B. ein Android-Smartphone[5] unterscheidet sich in vielen Punkten von einem normalen Rechner. Zunächst sei erwähnt, dass die Ressourcen-Verfügbarkeit und

[4] So könnte z.B. ein vormals physikalischer Honeypot, der sowohl einen ssh-Service anbietet als auch als Client Honeypot Webseiten aufsucht in eine simulierte Umgebung als virtueller Honeypot migrieren.

[5] Offizielle Webseite von Android: http://www.android.com/

die Rechenkapazität im Vergleich zu normalen Rechnern stark beschränkt ist. Auch wenn diese in den letzten Jahren stark zugenommen hat, so muss man bei der Implementierung von Software berücksichtigen, dass der Verbrauch an Ressourcen möglichst minimal ist. Auch die Zeit stellt insofern ein Ressource dar, als dass beispielsweise unter Android Prozesse sogenannter *Apps* verdrängt oder gar vorzeitig beendet werden, so dass die dadurch freiwerdenden Rechenkapazitäten etwa für die Bewältigung eingehender Anrufe zur verfügung stehen. Für weitere Informationen hierzu verweise ich auf die Arbeit „Scala on Android“ [5], in der auch das Java-Framework zur Programmierung von *Apps* für das Smartphone-Betriebssystem Android beschrieben ist. Eine herausragend wichtige Ressource in mobilen Systemen stellt Energie dar. Verbraucht eine Software viel Energie, etwa weil sie sich aufgrund fehlerhafter Programmierung nicht „schlafen legt“ wenn sie gerade im Hintergrund befindlich nicht genutzt wird, so leert dies vorzeitig den Akku des mobilen Geräts (ein gerade bei Android-Smartphones aktuelles Problem), so dass das Gerät zunächst aufgeladen werden muss, bevor es wieder eingesetzt werden kann. Üblicherweise verfügen mobile Endgeräte über mehrere unterschiedliche drahtlose Netzwerkschnittstellen. Zum einen gibt es die GSM- und UMTS-Schnittstellen. Daneben verfügt ein mobiles Gerät in der Regel über ein WLAN-Interface, ein Bluetooth-Interface sowie oft auch über ein GPS-Interface.

3.2 honeyM

Die zurzeit einzige Realisierung eines *Mobile Honeypot* ist honeyM. HoneyM [1] ist ein Framework, um virtuelle Honeyclients mobiler Geräte einsetzten zu können. Es bietet Bibliotheken von *Fingerprints* mobiler Geräte, um diese simulieren zu können. Hierbei emuliert honeyM WLAN-, Bluetooth- und GPS-Schnittstellen. Das Framework zielt darauf ab, Angriffe auf neue, unbekannte Schwachstellen aufzudecken. HoneyM trägt dem Umstand Rechnung, dass Angriffsvektoren mehrere Funkschnittstellen umfassen können, d.h. ein Angriff beispielsweise sowohl Schwachstellen im Bluetooth- als auch im WLAN-Stack ausnutzt. Einen großen Aufwand wird betrieben bei der Nachahmung des Verhaltens konkreter mobiler Geräte *(Fingerprinting)* wie dem IPhone von Apple. Realisiert wird honeyM durch einen PC, der mit mehreren WLAN-Schnittstellen, Bluetooth-Schnittstellen und einem GPS-Empfänger ausgestattet ist. So können mehrere mobile Geräte zugleich simuliert werden.

3.3 Anforderungen und Problemstellung

Im Rahmen des SKIMS-Projekts sollen *Mobile Honeypots* als Sensoren eingesetzt werden, die als virtuelle mobile Honeypots von dem eigentlich (hauptsächlich) produktiv genutzten Endgerät simuliert werden. Beispielsweise betreibt ein Android-Smartphone einen virtuellen mobilen Client-Honeypot, der vorgibt, ebenfalls ein Android-Smartphone zu sein. Werden Angriffe detektiert, kann das mobile Gerät Maßnahmen zum eigenen Schutz ergreifen und über geeignete Kommunikationsmechanismen mobile Geräte in seiner unmittelbaren Umgebung warnen, so

dass auch diese Geräte sich vor dem Angreifer Schützen. Im Idealfall ist der Angreifer damit isoliert und kann keinen Schaden anrichten.

Der Sensor muss möglichst wenig Netzverkehr und möglichst wenig Rechenlast erzeugen, damit einerseits dem mobilen Endgerät genügend Ressourcen für seinen hauptsächlichen Einsatzzweck verbleiben, und andererseits der Akku nicht verfrüht erschöpft.

4 Abschätzung der Risiken bei der Umsetzung eines Mobile Honeypot

Allgemein sind die Randbedingungen für die Entwicklung von Anwendungen (den *Apps*) für mobile Geräte komplizierter als für normale Rechner, da die Anwendungen viel sorgsamer mit den Ressourcen umgehen müssen und eine Anwendung unvermittelt beendet werden kann (vom Betriebssystem erzwungen oder schlicht weil der Akku alle ist). Somit ist das Risiko, das die Entwicklung einer Software scheitert, allgemein höher.

Im SKIMS-Projekt wird Android für die Prototypen-Implementierung verwendet werden. Das Android-Framework kapselt eine *App* vor dem darunter liegenden Linux-Betriebssytem unter anderem aus Sicherheitsgründen ab (siehe [5]). Für einen *Mobile Honeypot* – auch für einen virtualisierten – werden jedoch Funktionalitäten auf Betriebssystem-Ebene benötigt, die über das Framework nicht bereitestellt werden. Um dies zu erreichen sind Modifizierungen am Android-Betriebssystem notwendig. Dies widerspricht jedoch dem Betrieb eines Honeypots, als das nun die Schwierigkeit besteht, zwischen der für den Honeypot-Betrieb notwendigen Veränderungen und den durch eine denkbare Kompromittierung erfolgten Änderungen zu unterscheiden. Zudem können die notwendigen Veränderungen selbst neue Angriffsmöglichkeiten schaffen. Dieses Problem stufe ich als ziemlich kritisch und schwer zu lösen ein, da es konzeptuell bedingt ist.

5 Zusammenfassung und Ausblick

5.1 Zusammenfassung

Ein *Mobile Honeypot* ist eine Vorrichtung, die einem Angreifer als ein normales mobiles Gerät erscheint, jedoch nicht als ein solches verwendet wird. Der einzige Zweck ist es, angegriffen zu werden und solche Angriffe zu dokumentieren.

Im Allgemeinen ist es die Aufgabe eines Honeypots, (neue) Angriffe zu entdecken und aufzuzeichnen sowie die Ziele und das Verhalten eines Angreifers zu verstehen. Jedoch unterscheiden sich der Kontext und die Szenarien mobiler Geräte entscheidend von konventionellen Endbenutzer-Systemen: Die Hardware-Ressourcen sind stärker beschränkt, und es gibt mehrere unterschiedlich genutzte Funk-Schnittstellen verschiedener Technologien. Als ein *Smarphone* hält ein mobiles Gerät of vertrauliche Daten des Anwenders vor.

Mobile Honeypots können ein mobiles Gerät simulieren *(Low-Interaction Honeypot)* oder als ein echtes mobiles Gerät realisiert sein *(High-Interaction Honeypot)*. Derzeit gibt es nur ein Framework virtualisierter *Mobile Honeypots*: honeyM. HoneyM basiert auf einem normalen PC und simuliert mobile Geräte.

5.2 Ausblick

Neben dem oben beschriebenen Ansatz, virtuelle Honeypots zu betreiben, die vorgeben, ein mobiles Gerät (beispielsweise ein Smartphone) zu sein, kann ein mobiler Honeypot auch für die Aufgabe als ein „klassischer“ Sensor für Angriffe auf Funktionalitäten normaler Rechner interessant sein. Mit den Ortswechseln sind die Sensoren in unterschiedlichen Netzwerken eingebunden und werden so Angriffen ausgesetzt, denen sie mit einer statischen Netzanbindung nicht ausgesetzt wären. Denn ein Angreifer der weiß, in welchen (Dark-)Netzbereichen *Honeynets* als Sensoren Angriffe detektieren, kann diese meiden. Adressen mobiler Sensoren hingegen sind ungemein schwieriger vorherzusagen.

Literatur

[1] T. J. O'Connor and Ben Sangster. honeym: a framework for implementing virtual honeyclients for mobile devices. In *Proceedings of the third ACM conference on Wireless network security*, WiSec '10, pages 129–138, New York, NY, USA, 2010. ACM.

[2] Vern Paxson. Bro: a system for detecting network intruders in real-time. *Comput. Netw.*, 31:2435–2463, December 1999.

[3] Niels Provos and Thorsten Holz. *Virtual honeypots: from botnet tracking to intrusion detection*. Addison-Wesley Professional, first edition, 2007.

[4] J. H. Saltzer, D. P. Reed, and D. D. Clark. End-to-end arguments in system design. *ACM Trans. Comput. Syst.*, 2:277–288, November 1984.

[5] Theodor Nolte. Scala on Android. https://pub.informatik.haw-hamburg.de/home/pub/n/nolte_theodor/ScalaOnAndroid.pdf, 31. Mai 2010.

[6] Thomas C. Schmidt. SKIMS - A Cooperative Autonomous Immune System for Mobile Devices. http://www.realmv6.org/skims.html, 28. Februar 2011.