Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Masterarbeit

André Harms

Cyber-Resilience-Analyse mittels Ausbreitungssimulation von Schadsoftware

Fakultät Technik und Informatik
Studiendepartment Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

André Harms

Cyber-Resilience-Analyse mittels Ausbreitungssimulation von Schadsoftware

Masterarbeit eingereicht im Rahmen der Masterprüfung

im Studiengang Master of Science Angewandte Informatik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. Thiel-Clemen
Zweitgutachter: Prof. Dr.-Ing. Hübner

Eingereicht am: 1. April 2014

André Harms

Thema der Arbeit
Cyber-Resilience-Analyse mittels Ausbreitungssimulation von Schadsoftware

Stichworte
APT, Schadsoftware, Simulation, Widerstandsfähigkeit, Graphen, Multiagentensystem

Kurzzusammenfassung
Ziel dieser Arbeit ist der Machbarkeitsnachweis eines Frameworks zur Ausbreitungssimulation von Schadsoftware. Kernaspekt ist die Konzeption dieses Frameworks und seine Implementierung. In einer Fallstudie wird mit Hilfe der entwickelten Basis ein Angriffsszenario simuliert und die gewonnenen Ergebnisse kritisch beleuchtet. Auf Grundlage dieser Erkenntnisse wird ermittelt, welche Faktoren sich für eine Resilience-Analyse eignen oder auf welche Besonderheiten geachtet werden muss.

André Harms

Title of the paper
Cyber resilience analysis via malware propagation simulation

Keywords
APT, Malware, Simulation, Resilience, Graph, Multi-Agent-System

Abstract
This document aims towards to show the feasibility of a framework for simulating malware propagation. The core aspects are the conception and the implementation of such a framework. A case study on an attack scenario utilizes the implementation. The results are then discussed and are used to determine which measurements are suitable for a cyber resilience analysis or which other factors may be relevant.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Listings

1. Einleitung

1.1. Motivation

Seitdem John Cohen 1984 die ersten praktischen Experimente mit Computerviren gemacht hat (Cohen, 1984), haben diese einen starken Wandel vollzogen. Waren sie anfangs noch in ihren Möglichkeiten beschränkt und wurden ungezielt verbreitet, werden sie heute bei zielgerichteten Angriffen auf IT-Systeme eingesetzt. Dies hat zum Beispiel der Angriff mit Stuxnet auf das iranische Atomprogramm (Symantec Corporation, 2011) oder auch Red October, welches mittels Spear-Phishing Attacken initial verteilt wurde (Kaspersky Lab, 2012), veranschaulicht. Die Angriffsmöglichkeiten sind dabei vielfältig. Um eine Attacke zu realisieren, werden häufig mehrere Angriffsvektoren verwendet und miteinander kombiniert. Oft werden initiale (sekundäre) Angriffe durchgeführt, um an das primäre Ziel zu gelangen. Hierzu werden ausgewählte Systeme infiziert, um anschließend eine Schadsoftware in einem gesicherten Bereich, oder auch in nicht vernetzte Systeme, zu verbreiten. Dies ist unter anderem auch bei Stuxnet geschehen (Symantec Corporation, 2011). Die Verwendung mobiler Geräte und eng vernetzter Systeme zum Datenaustausch - wie Cloud-Storage (DropBox, Google Drive, ...) oder Netzwerkshares - eröffnen dabei neue Verbreitungswege und werden aktiv zur Ausbreitung von Malware genutzt (Symantec Corporation, 2011, 2012; Wang u. Stavrou, 2010).
Möchte man bestehende Systeme auf ihre Anfälligkeit von Angriffen testen, und empfehlenswerte Handlungsweisen im Falle eines Angriffs herleiten, besteht die Möglichkeit Penetrationtests und Vulnerability-Assessments zur Analyse durchzuführen. Solche Tests kommen echten - aber abgesprochenen - Angriffen gleich. Somit besteht auch die Gefahr, ein System unbeabsichtigt bei so einem Test zu beschädigen und einen Ausfall hervorzurufen. Diese Tatsache birgt vor allem beim Testen von kritischer Infrastruktur - aufgrund ihrer Bedeutung - Gefahren. Auch ist solch ein Test im Kontext der Schadsoftwareausbreitung nicht geeignet, weil infizierte Systeme anschließend wieder bereinigt werden müssten. Daher bieten sich Simulationen von Angriffen an, um Resilience-Analysen durchzuführen und Erkenntnisse zu empfehlenswerte Gegenmaßnahmen zu erlangen.
Bestehende mikroskopische Simulationen, die sich mit der Ausbreitung von Schadsoftware

beschäftigen, greifen auf physische Replikation der zu testenden Umgebung zurück (vgl. Leszczyna et al. (2008)) oder benötigen manuelle Eingriffe. Dadurch ergeben sich Tests, die in ihrer Vorbereitung oder Durchführung zeit- und ressourcenintensiv sind. Hinzu kommt, dass die Dynamik eines Netzwerkes, die durch mobile Geräte erzeugt wird, nicht beachtet wird. Gerade diese Geräteklasse wird aber häufig von Institutionen als potentielle Gefahrenquelle - auch im Kontext von Bring your own Device (BYOD) - angesehen (Deloitte, 2013) und für Angriffe instrumentalisiert. Um eine detaillierte Betrachtung des Ausbreitungsverhaltens zu ermöglichen und die Dynamik zu berücksichtigen, ist eine manuelle oder physische Simulation nicht geeignet.

1.2. Fragestellung

Aufgrund der beschränkten und einschränkenden Möglichkeiten, die existierende Ansätze bieten, soll diese Masterarbeit folgende Fragestellungen untersuchen:

Machbarkeitsnachweis

Zunächst soll die Machbarkeit eines agentenbasierten Modells untersucht werden, welches die einschränkenden Eigenschaften der physischen Replikation einer Testumgebung vermeidet, und mobile Endgeräte berücksichtigt. Die Arbeit soll hierzu anschaulich zeigen, wie eine mögliche Modellierung und deren Implementierung aussehen kann. Ebenso soll beleuchtet werden, welche Nachteile und Einschränkungen gegebenenfalls entstehen und welche Hilfestellungen zur Resilience-Analyse mittels des Modells möglich sind. Die Arbeit soll somit ein grundlegendes Framework für weitere Forschungsarbeiten schaffen, um weiterführende Fragestellungen bearbeiten zu können.

Einfluss topologischer Eigenschaften auf das Ausbreitungsverhalten

Mit dem erstellten Modell und seiner Implementierung soll untersucht werden, ob topologische Eigenschaften eines untersuchten Netzes generell Einfluss auf die Ausbreitung von Schadsoftware ausüben, oder ob eventuell andere Faktoren größere Auswirkung haben. Hier soll vor allem geklärt werden, ob die technische Relevanz eines Systems direkten Bezug auf die Relevanz bei der Propagation der Malware hat. Es wir davon ausgegangen, dass dies nicht zwingend der Fall ist. So wird angenommen, dass auch Netzknoten mit einer weniger hohen Relevanz für ein Netzwerk überproportional an der Ausbreitung beteiligt sein können. Bewahrheitet sich diese Annahme, so ließen sich mit den Ergebnissen Awareness-Maßnahmen zielgerichteter

positionieren. Sollten dennoch zwischen der Netzwerk- und der Propagationsrelevanz Korrelationen festgestellt werden, könnten die Erkenntnisse zum Herleiten genereller Aussagen über bestimmte technische Strukturen verwendet werden, um Handlungsempfehlungen im Falle einer detektierten Systeminfektion geben zu können.

1.3. Abgrenzungen

Da zunächst die Machbarkeit eines Frameworks zur Ausbreitungssimulation nachgewiesen werden soll, sind ausgiebige Performancetests nicht Teil dieser Arbeit. Dies bedeutet auch, dass Leistungsoptimierungen nicht zum Umfang der Arbeit gehören.

Da die Simulationsergebnisse unterschiedlich interpretiert werden können und das Spektrum der Interpretationsziele groß ist, ist eine universelle Aufbereitung von Simulationsergebnissen nicht Bestandteil der Arbeit. Vielmehr wird die Auswertung der gewonnen Daten spezifisch für das Ermitteln, der zur Lösung der Fragestellung nötigen Informationen, durchgeführt. Somit ist eine komplexe Auswertungskomponente weder Teil des Frameworks noch der Arbeit.

1.4. Inhalt

In Kapitel 2 werden die, für das Verständnis der weiteren Arbeit, nötigen Grundlagen erläutert. Hierzu gehören neben fachliche auch technische Begriffe und Konzepte.

Daran anschließend werden in Kapitel 3 verwandte Arbeiten kurz vorgestellt und die eigene Arbeit eingeordnet.

In Kapitel 4 wird nach einer vorausgehenden Analyse der Anforderungen an ein Framework zur Ausbreitungssimulation von Schadsoftware eine Spezifikation erstellt, die als Ansatz für eine Realisierung dient. Danach wird ein Entwurf präsentiert, der als Vorlage für eine konkrete Implementierung verwendet wird. Auf die wichtigsten Teile dieser, und die hierfür verwendeten Werkzeuge, wird in diesem Kapitel ebenfalls eingegangen.

Kapitel 5 beschreibt eine durchgeführte Fallstudie, die das entwickelte Simulationsframework verwendet. Insbesondere wird hier das Szenario beschrieben und die Umsetzung mit ihren Parametern erläutert.

Im anschließenden Kapitel 6 folgt eine Ergebnispräsentation, die die Messwerte der Fallstudie in aufbereiteter Form enthält. Zudem werden die Messergebnisse auf ihre Plausibilität hin überprüft, damit sie für weiterführende Analysen und Bewertungen herangezogen werden können.

Eine Diskussion und kritische Betrachtung der Ergebnisse, die auch diese Analyse beinhaltet, findet in Kapitel 7 statt. Darüber hinaus wird hier neben der kritischen Betrachtung der Ergebnisse auch die der Implementierung des Simulationsframesworks durchgeführt. Aus der Bewertung der Ergebnisse heraus wird dann eine Empfehlung abgeleitet.

Kapitel 8 fasst abschließend die Arbeit und die daraus resultierenden Ergebnisse zusammen und gibt einen Ausblick auf mögliche Weiterentwicklungen und Forschungsansätze.

2. Grundlagen

2.1. Fachliche Grundlagen

2.1.1. Malware

Malware ist die Kurzform für den englischen Begriff „malicious software“, also bösartige Software, welche auch als Schadsoftware bezeichnet wird. Malware lässt sich in verschiedene Kategorien einteilen, die sich nahezu beliebig granular definieren lassen. Daher werden hier nur einige Gruppen von Malware vorgestellt, die für das Verständnis dieser Arbeit relevant sind:

Viren verbreiten sich eigenständig, indem sie sich an Programme oder Dokumente heften und diese als Wirt benutzen. Bei deren Ausführung oder Öffnung repliziert sich der Virus. Bei Computerviren wird meist die männliche Form verwendet, was in dieser Arbeit ebenso gemacht wird.

Würmer sind eigenständige Programme, die sich selbstständig, zum Beispiel durch Selbstversand mittels E-Mail, verbreiten. Hierzu wird kein Wirt verwendet, was sie von Viren unterscheidet.

Trojaner werden für den Zweck der Spionage entwickelt und bedienen sich dabei unterschiedlicher Verbreitungsmechanismen.

Je nach Einsatzzweck einer Malware verschwimmen die Grenzen auch; eine Zuordnung in die genannten Gruppen ist dann nicht mehr möglich. Somit ist Malware der Sammelbegriff für jede Software, die schadhaftes Verhalten vorspielt oder aufweist. Häufig wird vereinfacht auch der Begriff Virus synonym für alle Kategorien von Schadsoftware verwandt.

2.1.2. Cyberangriffe

Das BSI definiert Cyberangriffe wie folgt:

> „Ein Cyber-Angriff ist ein IT-Angriff im Cyber-Raum, der sich gegen einen oder mehrere andere IT-Systeme richtet und zum Ziel hat, die IT-Sicherheit zu brechen. Die Ziele der IT-Sicherheit, Vertraulichkeit, Integrität und Verfügbarkeit können dabei als Teil oder Ganzes verletzt sein.“ (BSI, 2011) [S. 14]

Dieser Definition wird sich hier weitgehend angeschlossen. Sie soll allerdings noch etwas ausgeführt und differenzierter betrachtet werden.

Je nach Ziel des Angreifers können verschiedene Mittel eingesetzt werden, um einen Angriff durchzuführen. Diese können technischer aber auch soziologischer Natur sein; es kann, muss aber keine Schadsoftware eingesetzt werden. Die Intention eines Angreifers bestimmt maßgeblich sein Vorgehen und die eingesetzten Mittel.

Möchte ein Angreifer an persönliche Informationen - wie Bankdaten - gelangen, um sich finanziell zu bereichern, so wird er wahrscheinlich eine groß angelegte Phishing-Attacke verwenden. Soll publikumswirksam gegen oder für eine Sache demonstriert werden, so kommen häufig DoS-Angriffe zum Einsatz, wie in der Vergangenheit schon häufiger von Interessengruppen praktiziert. Dies war beispielsweise im Jahr 2007 in Estland der Fall, als russische Aktivisten estländische Infrastruktur lähmten (John J. Kelly, 2008). Gemein haben beide hier angebrachten Beispiele, dass sie relativ auffällig und in ihrer Ausführungs- sowie Wirkungszeit stark begrenzt sind. Anders sieht dies bei hoch ausgeklügelten Angriffen aus, die folgend in Abschnitt 2.1.3 erläutert werden.

2.1.3. Advanced Persistent Threat

Advanced Persistent Threats (APTs) können als eigene Klasse von Angriffen angesehen werden. Als Abgrenzung zu herkömmlichen Angriffen, zielen APTs auf ein bestimmtes Opfer/Ziel ab und sind hochentwickelte Attacken oder sogar Angriffs-Kampagnen, die sowohl technische als auch soziologische Angriffsvektoren verwenden. Die verwendete Schadsoftware infiziert dabei meist nur eine beschränkte Anzahl von Systemen; wohingegen bei konventionellen Angriffen die Infektionen nicht zielgerichtet vonstatten gehen, und somit auch eine große Anzahl von Geräten befallen werden. Eine initiale Infektion ist bei einer APT meist nur das Sprungbrett, um in ein fremdes Netzwerk zu gelangen und um dort das eigentliche Ziel attackieren zu können. Ziel ist meist das Ausspähen von Informationen oder auch unauffällige Sabotage über einen längeren Zeitraum. In ihren technischen Ausprägungen können APTs sehr komplex sein, verwenden Zero-Day-Lücken oder auch schon allgemein bekannte Schwachstellen, und kombinieren diese häufig auf ausgeklügelte Weise. Zudem wird bei APTs häufig mit Techniken aus dem Social Engineering gearbeitet, um grundlegende Informationen zu erlangen und

Schadsoftware für eine initiale Infektion zu platzieren. Dies geschieht zum Beispiel durch Phishing (Mandiant, 2013). Durch die Zielsetzung des Angreifers möglichst über einen langen Zeitraum zu agieren und daher unbemerkt zu bleiben, sowie zielgerichtet anzugreifen, handelt es sich bei der Malware, die für eine APT herangezogen wird, meist um Individuallösungen. Diese werden nicht durch konventionelle Virenscanner oder Angriffserkennungssysteme (auch Intrusion Detection System (IDS)) erkannt, da sie zu speziell konstruiert sind. Durch den durchaus erhöhten Aufwand, der bei einer APT im Vergleich zu konventionellen Angriffen betrieben wird, lässt sich eine APT eher als eine Art Kampagne als nur als simpler Angriff beschreiben. Bekannte APTs sind zum Beispiel „Red October“ und „Stuxnet“, sowie das jüngere und noch ausgefeiltere „Careto“ - auch als „Mask“ bekannt (Kaspersky Lab, 2014).

2.1.4. Arten von Simulationsmodellen zur Schadsoftwareausbreitung

Es existieren verschiedene Ansätze, um die Propagation einer Schadsoftware innerhalb eines Netzwerks zu simulieren. Hierbei kommen verschiedene Simulationsmodelle mit unterschiedlichen Eigenschaften, sowie Vor- und Nachteilen zum Einsatz. Einige dieser Ansätze werden im Folgenden vorgestellt.

Mathematische Modelle (Homogeneous Mixing Models)

Mathematische Modelle zum Vorhersagen des Ausbreitungsverhaltens von Schadsoftware verfolgen ähnliche Ansätze, wie es auch mathematisch epidemiologische Modelle der Medizin machen, um Krankheitsausbreitungen vorherzusagen. Hierbei werden gewöhnliche Differentialgleichungen verwendet, um die Infektionshäufigkeit innerhalb eines informationstechnischen Systems abzuschätzen. Dafür werden Annahmen über die Schadsoftware und die Systemlandschaft getroffen:

1. Die Wahrscheinlichkeit, dass ein System ein anderes infiziert, ist für alle Systeme identisch.
2. Alle Rechnerknoten haben innerhalb eines bestimmten Zeitraumes die gleiche Anzahl von Kontakten.
3. Die Systeme sind vollständig durchmischt, so dass empfängliche Systeme Kontakt zu einem anderen System aus der Gesamtheit der Systeme bekommen kann.

Da die Annahmen, die getroffen werden, die Realität nicht genau abbilden und Aspekte wie systemspezifische Merkmale, Mobilität oder auch Dynamik nicht berücksichtigen, sind die

Ergebnisse meist ungenau. Zudem fehlen spatiale Informationen bezüglich der Ausbreitung, so dass Aussagen nur über die Ausbreitungshäufig- und Geschwindigkeit gemacht werden können, nicht jedoch über den Zustand von Netzknoten.

Eine Ausprägung eines solchen mathematischen Ansatzes ist zum Beispiel das SIS-Modell, welches in mehreren Arbeiten zur Abschätzung der Ausbreitung von Malware herangezogen wurde. Auf das SIS-Modell wird deshalb unter 2.1.5 genauer eingegangen.

Zelluläre Automaten (Non-Homogeneous Mixing Models)

Ein weiterer Ansatz zur Malware-Ausbreitungssimulation stellen zelluläre Automaten dar. Mit ihnen lassen sich die Beziehungen zwischen den Systemknoten modellieren, so dass spatiale Eigenschaften berücksichtigt werden (Song et al., 2008). Eine Zelle stellt dabei einen Netzknoten im Informationssystem dar. Diese können gemäß eines zugrunde liegenden Modells - zum Beispiel das SIS-Modell - verschiedene Zustände annehmen. Ein neuer Zustand einer Zelle errechnet sich aus ihrem aktuellen Zustand, sowie dem Zustand ihrer Nachbarn. Somit kann ein infektiöses System immer nur eine bestimmte Anzahl - basierend auf dem Grad seiner Nachbarschaft - anderer Systeme anstecken. Im Gegensatz zu den „Homogeneous Mixing Models“ berücksichtigt dieser Ansatz topologische Gegebenheiten des untersuchten Netzes und ist daher realitätsnäher. Allerdings werden die Dynamik eines Netzes durch mobile Geräte und spezielle Systemeigenschaften nicht einbezogen.

Agentenbasierte Modelle

Neben den beiden bereits erwähnten Ansätzen, gibt es Bemühungen, das Ausbreitungsverhalten von Schadsoftware mittels agentenbasierter Modelle zu simulieren. Im Gegensatz zu den mathematischen Modellen, die Formeln über die Gesamtheit der Rechner-Systeme aufstellen, erlauben agentenbasierte Modelle mikroskopische Betrachtungen und liefern somit Informationen über Einzelsysteme und Ausbreitungsgründe. Bei dieser Art der Modellierung wird eine Umgebung vorausgesetzt, in der die Agenten agieren können. Agenten können dabei miteinander kommunizieren und mit der Umgebung interagieren. Außerdem ist es möglich, verschiedene Agenten mit unterschiedlichen Eigenschaften zu erzeugen, so dass eine genauere Abbildung der Realität möglich ist. Heterogene Patchlevel, unterschiedliche Betriebssysteme, Dienste und Fähigkeiten lassen sich berücksichtigen; je nachdem wie das konkrete Modell aussieht.

Für die eigene Arbeit wird ein agentenbasiertes Modell aufgestellt, da sich durch die mikroskopische Betrachtung die Ausbreitung über ein Netzwerk und seiner Teilnehmer beurteilen

lässt. Dies ist mit den anderen beiden Ansätzen nicht möglich, zum Bearbeiten der eigenen Fragestellung aber nötig.

2.1.5. SIS-Modell

Da das SIS-Modell in mehreren Arbeiten zur Simulation der Ausbreitung von Schadsoftware herangezogen wird, und die Grundlage für Weiterentwicklungen ist, soll es an dieser Stelle genauer erläutert werden. Das SIS-Modell ist ein einfaches Modell zur Simulation von Infektionsausbreitungen in der medizinischen Epidemiologie, welches 1927 von Kermack und McKendrick vorgestellt und später auch für die Ausbreitungssimulation von Computerviren herangezogen wurde (Kephart u. White, 1991). Beim SIS-Modell werden die untersuchten Netzknoten (N) in zwei Gruppen von Systemen eingeteilt. Systeme, die nicht befallen sind, werden der Gruppe „Susceptible“ (S) zugeschrieben. Systeme, die befallen sind, und von denen aus andere Systeme infiziert werden könnten, gehören der Gruppe „Infectious“ (I) an. Somit ist $N = S + I$.

Das SIS-Modell basiert auf einer diskreten, endlichen Markov-Kette, bei der Zustandsübergänge mit einer gewissen Wahrscheinlichkeit eintreten können (Abb. 2.1). Dabei hängt der Folgezustand nur vom aktuellen Zustand ab und nicht vom bisherigen Infektionsverlauf.

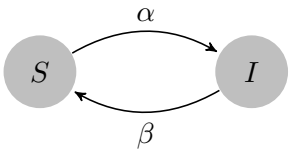


Abbildung 2.1.: Zustandsgraph, SIS-Modell

Ein System kann beim SIS-Modell mit einer Wahrscheinlichkeit α infiziert werden. α setzt sich dabei aus einer Kontaktwahrscheinlichkeit (K) und dem Kontagionsindex (q), welcher die Wahrscheinlichkeit einer Erkrankung bei Kontakt mit einem infizierten System angibt, zusammen ($alpha = K * q$). Kommt ein nicht infiziertes System in Kontakt zu einem infizierten und ist anfällig für die Schadsoftware, wechselt es zu I. Es wird außerdem berücksichtigt, dass ein System desinfiziert werden kann. Dies geschieht mit der Wahrscheinlichkeit β. Eine Immunisierung - z.B. durch Patchinstallation - wird hingegen nicht berücksichtigt; ein desinfiziertes System kann also wieder befallen werden.

Es gibt einige Weiterentwicklungen des SIS-Modells, wie zum Beispiel das SIR-Modell, bei dem Resistenzen berücksichtigt werden oder auch das SIRS-Modell, welches Mutationen berücksichtigt. Hier werden resistente Systeme wieder infizierbar. Im Bezug auf die Ausbreitung von

Schadsoftware wurden basierend auf dem SIS-Modell erweiterte Ansätze erstellt, welche den Grad der Vernetzung eines Informationssystems und spatiale Eigenschaften berücksichtigen (Kephart u. White, 1991; Lora Billings, 2002; Song et al., 2008).

2.2. Technische Grundlagen

2.2.1. Agenten und Behaviours

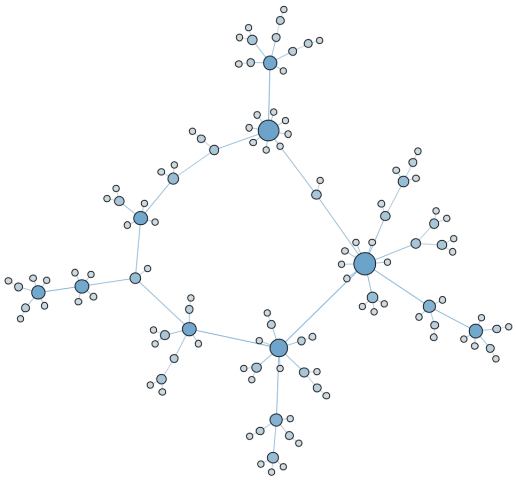
Für den Begriff des Agenten existiert keine universelle Definition. Allerdings herrscht allgemeiner Konzens darüber, dass Autonomie eine wichtige Eigenschaft eines Agenten ist. Dies führt dazu, dass ein Agent meist als eigenständige Anwendung konzipiert ist. Andere Eigenschaften, die ein Agent haben kann, hingegen unterscheiden sich in verschiedenen Fachbereichen, so dass diese nicht zwingend zu einem Agenten gehören müssen. So gibt es Anwendungsfälle in denen ein Agent in der Lage sein muss zu lernen, in anderen hingegen ist dies nicht nötig oder sogar unerwünscht (Wooldridge, 2002). Nach Wooldridge ist ein Agent wie folgt definiert:

> „An agent is a computer system that is situated in some environment, and that is capable of autonomous action in this environment in order to meet its design objectives.“ (Wooldridge, 2002) [S.5]

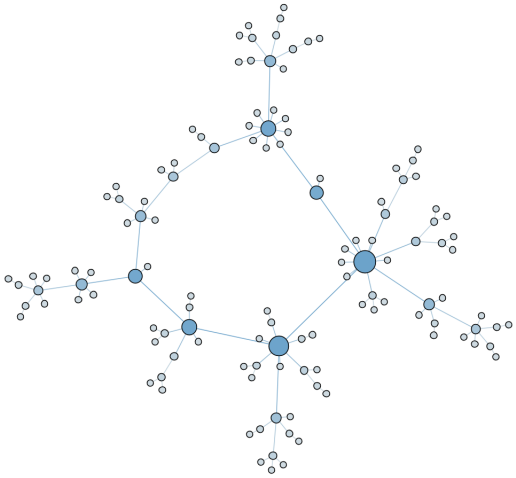
Agenten lassen sich bezüglich ihres Verhaltens (Behaviours) differenzieren. So kann ein Agent Entscheidungen unabhängig von seinen vergangen Erfahrungen und nur in Abhängigkeit von aktuellen Ereignissen treffen. Diese Agenten werden als rein reaktive Agenten bezeichnet. Auch kann ein Agent eigenständig aktiv werden, um seine Ziele zu erreichen und handelt somit pro aktiv (Wooldridge, 2002). Allerdings können Agenten auch mehrere Verhalten vereinen.

2.2.2. Zentralität (Graphenmetriken)

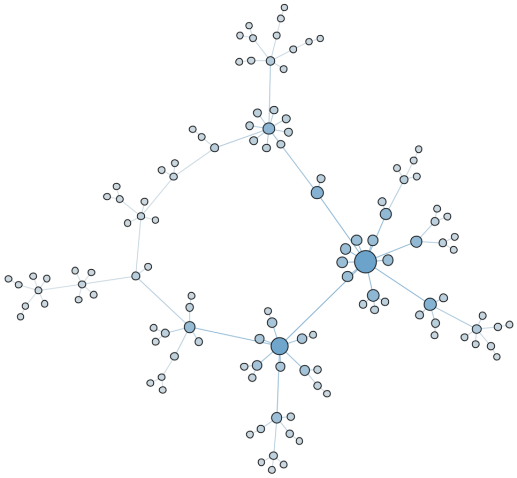
Die Zentralität beschreibt die Wichtigkeit eines Knotens innerhalb eines Netzwerks. Sie spielt häufig bei der Analyse von sozialen oder Computer-Netzen eine wichtige Rolle, aber auch bei der Bestimmung der Relevanz von Inhalten. So wird zum Beispiel der PageRank (eine Ausprägung der Eigenvektor-Zentralität) von Suchmaschinen eben hierfür verwendet. Folgend werden drei Zentralitäts-Metriken für Knoten vorgestellt, die intuitiv Einfluss auf das Ausbreitungsverhalten von Schadsoftware haben könnten. Diese werden grafisch in Abbildung 2.2 gegenübergestellt.



(a) Degree-Zentralität



(b) Betweenness-Zentralität



(c) Eigenvektor-Zentralität

Abbildung 2.2.: Gegenüberstellung Zentralitäts-Metriken (je größer und dunkler ein Knoten, desto höher ist der Messwert)

Grad-Zentralität

Der Grad - auch Valenz - eines Knotens ist eine einfache Metrik der Graphentheorie. Er beschreibt die Summe der mit dem Knoten verbundenen Kanten. Handelt es sich um einen gerichteten Graphen, so wird zwischen Eingangsgrad - die Anzahl eintreffender Kanten eines

Knotens - und Ausgangsgrad - die Anzahl ausgehender Kanten eines Knoten - unterschieden. Da nur die Anzahl der verbundenen Kanten eines Knoten berücksichtigt werden, aber nicht ihre Bedeutung im Gesamtkontext eines Netzes, handelt es sich um eine lokale Metrik.

Betweenness-Zentralität

Diese Metrik erfasst, wie häufig ein Knoten Teil eines kürzesten Weges zwischen zwei anderen Knoten ist. Sie drückt somit die Wichtigkeit eines Knotens in Bezug auf den Informationsaustausch aus. Befindet sich ein Knoten also am Rand eines Graphen, so besitzt er naturgemäß eine niedrige Betweenness-Zentralität; wohingegen er unter Betrachtung anderer Metriken eine höheren Messwert aufweisen kann, wie zum Beispiel ein Vergleich von Abb. 2.2 (a) und 2.2 (b) zeigt. Es existiert auch eine Kanten-Betweenness, die hier aber nicht behandelt wird.

Eigenvektor-Zentralität

Die Eigenvektor-Zentralität bestimmt die Wichtigkeit eines Knoten anhand der Bedeutsamkeit seiner Nachbarn. Ist ein Knoten mit einem anderen - wichtigen - Knoten verbunden, so erhöht dies die eigene Relevanz mehr, als wäre er mit einem niedrig bewertetem Knoten verbunden. Somit wird nicht nur die lokale Relevanz eines Knoten ermittelt, sondern seine Relevanz über das gesamte Netzwerk. Varianten dieser Zentralität sind der PageRank und die Katz-Zentralität.

3. Verwandte Arbeiten und Einordnung

Es existieren viele Arbeiten, die sich mittels mathematischer Modelle dem Thema Ausbreitungssimulation von Malware annehmen. Dabei werden häufig spezielle Aspekte beachtet. Cheng et al. (2011) erstellen in ihrer Arbeit ein mathematisches Modell, welches hybride Ausbreitungswege von Malware für Smartphones berücksichtigt. Sie betrachten dabei den Verbreitungsweg über MMS und per Bluetooth-Verbindungen. Zugrunde liegt das SI-Modell, welches eine Genesung infizierter Systeme nicht vorsieht. Bei der Verbreitung mittels Bluetooth berücksichtigen Cheng et al. (2011) die nötige spatiale Nähe von Geräten, damit eine Infektion stattfinden kann, welches durch entsprechende Infektionswahrscheinlichkeiten für jeden Ausbreitungsweg geschieht. Das resultierende Modell ist mittels gewöhnlicher Differentialgleichungen implementiert.

Song et al. (2008) hingegen verwenden zelluläre Automaten, um die Ausbreitung von Schadsoftware zu simulieren. Es werden je ein zelluläres Automatenmodell auf Basis des stochastischen SIS und des stochastischen SIR Modells erstellt, die vorgeben, welche Zustände ein System annehmen kann. Die beiden resultierenden SIS-CA[1] und SIR-CA werden anhand generierter skalenfreier Netze (Barabasi-Albert Algorithmus), und Zufallsgraphen (Erdos-Renyi Algorithmus) getestet. Durch die Verwendung zellulärer Automaten können Yurong Song et al. den spatial-temporalen Verlauf einer Schadsoftwareausbreitung bestimmen und den Einfluss topologischer Gegebenheiten berücksichtigen. Eigenschaften von Systemen, wie Mobilität oder ihre Funktionen innerhalb des Netzwerkes, werden hingegen nicht berücksichtigt.

Ein agentenbasierter Ansatz wird von Leszczyna et al. (2008) beschrieben. Sie modellieren die Schadsoftware selbst als Agenten, der sich innerhalb eines Netzwerkes ausbreitet. Hierzu wird das zu testende Netz physisch im Labor repliziert und auf den entsprechenden Endgeräten eine Laufzeitumgebung für die Agenten installiert. Diese stellt eine Art Sandbox zur Verfügung, mit der Systemänderungen durch die Schadsoftware simuliert werden. Die Malware-Agenten können sich dann innerhalb dieser Umgebung bewegen, replizieren und ihr Verhalten ausüben.

[1] CA = hier Cellular Automata

Durch die physische Replikation der Testumgebung ergibt sich - je nach Umfang dieser - ein hoher Aufwand. Gedacht ist dieser Ansatz vor allem zum Testen von kritischer Infrastruktur oder Teilen davon.

Die eigene Arbeit zielt auf die Ausbreitungssimulation von Schadsoftware innerhalb beliebiger Netzwerke ab. Anders als bei Leszczyna et al. (2008) wird daher auf eine physische Replikation verzichtet und auf eine rein virtuelle Nachbildung der zu testenden Umgebung gesetzt. Besonderer Fokus wird dabei auf die Kommunikationswege innerhalb des Netzes gelegt, so dass topologische Eigenschaften berücksichtigt und untersucht werden können. Systemeigenschaften wie z.B. installierte Software, Patchlevel und Firewalleinstellungen, sowie die Portabilität einiger Systeme werden ebenso mit einbezogen werden. Um dies realisieren zu können wird im Gegensatz zu Song et al. (2008) ein agentenbasierter Ansatz gewählt. Da keine Replikation der Umgebung stattfindet, wird im Gegensatz zu Leszczyna et al. (2008) nicht die Malware als Agent modelliert, sondern die einzelnen Systeme mit ihren Eigenschaften und Verhalten.

4. Umsetzung

4.1. Analyse

Um ein funktionierendes System zu erstellen, werden zuerst die Anforderungen ermittelt und festgehalten. Folgend werden diese nach fachlichem und technischen Aspekten getrennt voneinander aufgeführt. Eine Zusammenfassung der Anforderungen befindet sich im Anhang A.

4.1.1. Fachliche Anforderungen

Zum Erzeugen möglichst realistischer Simulationsergebnisse, ist es nötig, verschiedene fachliche Aspekte zu berücksichtigen. Diese werden nachstehend aufgeführt.

Grundlegend

Da die Möglichkeit gegeben sein soll, bestehende Infrastrukturen zu testen, und ein manuelles Erfassen der infrastrukturellen Daten als zu aufwendig erachtet wird, soll dieses automatisiert oder zumindest unterstützt werden. Optional sollen die automatisch erfassten Daten einfach kontrolliert werden können. Daneben soll auch auch die Möglichkeit geschaffen werden, eigene fiktive Infrastrukturen zu erstellen, um bestimmte Voraussetzungen modellieren und untersuchen zu können. Weiterhin sollen automatisch gesammelte Daten bearbeitet werden können. Dies soll geschehen, ohne die gesammelten Daten zu überschreiben, da sie eventuell Basis weiterer Untersuchungen sind. Um eine Ausbreitung präziser beobachten und abschätzen zu können, sollen die simulierten Systeme differenziert von einander betrachtet werden können.

Systemeigenschaften und Vulnerabilität

Ob ein System durch einen Schädling infiziert werden kann, hängt maßgeblich von seiner Gesamtkonfiguration ab. Hierzu gehören z.B. die installierte Software inklusive des Betriebssystems, die Einstellungen der Software und die verwandte Version. Bestimmte Versionen oder auch Konfigurationen besitzen unterschiedliche Vulnerabilitäten, die verschiedene Auswirkungen haben können. Somit gilt es, möglichst viele - für die Verwundbarkeit relevanten -

Eigenschaften zu berücksichtigen. Um eine Vulnerabilität zu identifizieren, kann auf Daten von Vulnerabilitätsdatenbanken zurück gegriffen werden.

Zentralität und Mobilität

Da immer mehr Daten zentral abgelegt werden (z.B. Cloud) und von überall zugreifbar sind, soll dieser Aspekt bedacht werden. Ebenso sind heutzutage mobile Endgeräte und Massenspeicher - wie Laptops, USB-Sticks, usw. - keine Seltenheit mehr. Eine wichtige Rolle bei der Verbreitung von Schadsoftware haben diese Faktoren bei Stuxnet gespielt (Symantec Corporation, 2011). Deshalb soll es generell möglich sein, diese zu modellieren.

Benutzer und Aktivitäten

Aktivitäten von Benutzern, die entsprechendes Systemverhalten auslösen, haben eine bestimmte Dauer. Sie werden zufällig ausgeführt und anhand von Handlungsmustern erzeugt. Darunter fällt auch das Ändern der Postion von Benutzern, wenn sie z.B. reisende Tätigkeiten unternehmen. Auf ein GIS kann dabei verzichtet werden, da es sich um eine prototypische Implementierung handelt, die zur Evaluierung des Ansatzes dienen soll. Ein GIS könnte in Betracht auf Mobilität und Reisedauer eine genauere Simulation ermöglichen, ist aber nicht zwingend erforderlich und wird für diese Machbarkeitsstudie nicht benötigt.

Kommunikationswege

In einem Netzwerk kommunizieren die Systeme meistens nicht direkt miteinander. Der Informationsaustausch erfolgt über Routen, welche anhand bestimmter Faktoren gewählt werden. Oft ist das Ziel, einen Kommunikationspfad zu finden, der einen möglichst schnellen Informationsaustausch ermöglicht. Dies geschieht in der Regel intuitiv über den kürzesten Pfad im Netzwerk, was auch im Modell berücksichtigt werden soll. Ausgefeiltes Routing, welches sich zum Beispiel nach Auslastung von Knoten richtet, soll zunächst nicht berücksichtigt werden. Zudem kommt es in einem Netzwerk vor, dass Nachrichten aufgrund ihrer Bestimmungen verworfen werden. Dies wird z.B. durch Firewalls durchgeführt, welche port- oder dienstbasierend entscheiden, ob eine Nachricht weiter vermittelt oder zurückgewiesen wird. Auch dieser Aspekt soll beachtet werden. Durch die gewünschten mobilen Knoten, müssen die Kommunikationswege zur Simulationszeit dynamisch geändert werden können.

Topologie

Um die Kommunikationswege adäquat simulieren zu können, muss die zugrunde liegende Topologie erfasst werden. Hierfür ist es nötig Geräte bis OSI-Layer 3 zu erfassen. Darunterliegende Schichten müssen nicht betrachtet werden, da sie für das Routing nicht relevant sind. Somit sind Switches, Hubs und Netzwerkdevices nicht zu berücksichtigen. Die automatische Erfassung der Topologie-Daten unterliegt dabei technischen Grenzen, so dass nicht alle Geräte und Routen bei einem automatischen Scan sichtbar sind. Diese Einschränkungen müssen bedacht werden.

Schadsoftware

Die Schadsoftware, die für die Fallstudie in Kapitel 5 benötigt wird, soll sich an den Verbreitungsmöglichkeiten von Stuxnet orientieren. Dies bedeutet, dass portable Medien und Geräte, aber vor allem auch Netzwerkspeicher, zur Verbreitung genutzt werden. Sie soll sich wie Stuxnet nebenläufig ausbreiten, ohne dass der Benutzer explizit infizierte Dateien öffnet und zur Ausführung bringt (Symantec Corporation, 2011).

4.1.2. Technische Anforderungen

Allgemein

Bei der Implementierung des Simulationsmodells soll berücksichtigt werden, dass die Simulationssoftware ohne viel Aufwand auf unterschiedlichen Systemen laufen kann. Das bedeutet, dass zur Umsetzung eine plattformunabhängige Programmiersprache genutzt werden und auf plattformabhängige Datenbanksysteme zur Speicherung der Ergebnisse und Simulationsparameter verzichtet werden soll.

Ein weiterer zu beachtender Aspekt stellt die Performance dar, die zwar nicht expliziter Teil der Arbeit ist, aber dennoch vorausschauend bedacht werden soll. Die technische Grundlage der Realisierung sollte so gewählt werden, dass eine möglichst große Anzahl an Systemen simuliert werden kann, damit auch große Netzwerke betrachtet werden können. Dazu soll die Rechenleistung möglichst auf alle zur Verfügung stehenden Prozessoren und ihre Kerne verteilt werden. Eine Verteilung auf mehrere physische Systeme steht nicht im Fokus sollte aber grundsätzlich möglich sein.

Visualisierung

Die simulierte IT-Infrastruktur soll mittels einer GUI visualisiert werden. Um schon während der Simulation einen Eindruck vom Ausbreitungsverhalten zu bekommen, soll die GUI den Verlauf dynamisch anzeigen können. Außerdem soll die grafische Oberfläche dabei helfen, Kennzahlen innerhalb der Testumgebung zu visualisieren. Hierzu gehören neben den Graphen-Metriken auch Statistiken, die während der Simulation entstehen. Die Visualisierung der Daten soll bei der Ergebnisanalyse unterstützend sein.

4.2. Spezifikation

4.2.1. Arbeitsablauf

Durch den Wunsch, als Simulationsgrundlage existierende Netze zu verwenden, aber auch nicht existente Infrastrukturen untersuchen zu können, ergibt sich ein angestrebtes Verfahren, das aus vier Schritten besteht (Abb. 4.1):

- **Information Gathering**: Der Information Gathering Prozess erfasst Systemeigenschaften (Topologie, installierte Software/Dienste) des zu testenden Netzes weitgehend automatisch.
- **Prüfen, Bearbeiten, Erstellen von Informationen**: Das Prüfen und Nachbearbeiten der ermittelten Topologie etc. wird durch einen grafischen Editor unterstützt. Hiermit lassen sich auch Testumgebungen ohne Informationen ohne vorheriges Information Gathering manuell erstellen.
- **Simulation**: Hier wird die Propagation einer Malware innerhalb der durch die gegebenen Informationen definierten Umgebung simuliert.
- **Analyse**: Eine Analyse der Simulationsergebnisse wird während und nach der Simulation durch visuelle Hilfsmittel unterstützt.

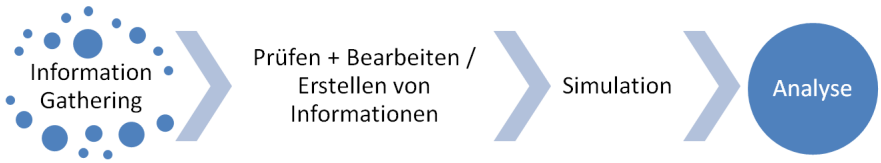


Abbildung 4.1.: Workflow

Soll ein fiktives Netz analysiert werden, kann auf das Information Gathering verzichtet werden und stattdessen durch Eingabe der Daten im Editor ersetzt werden. Auch ist der Schritt des Überprüfens eher fakultativ, wenn auch sicherlich sinnvoll. Somit sind beide Prozessteile so umzusetzen, dass sie unabhängig voneinander funktionieren. Deshalb lässt sich der gesamte Ablauf in vier grobe Komponenten unterteilen, die teilweise selbstständig funktionieren.

4.2.2. Ausgliederung von Teilprozessen

Da Planung und Umsetzung aller Prozesse den Rahmen dieser Arbeit überschreiten würden, wurden die Abläufe des Information Gatherings und der Editor ausgegliedert, worauf folgend genauer eingegangen wird.

Information Gathering

Das Information Gathering wurde von Robert Krauss im Rahmen seiner Bachelorarbeit umgesetzt (Krauß, 2012). Orientierend an den Anforderungen hat er eine Software realisiert, die für die Simulation notwendigen Informationen (FA-6-FA-8; FA-21) sammeln und aggregieren kann. Diese werden persistiert und den anderen Komponenten zur Verfügung gestellt. Die zum Persistieren verwendete Datenstruktur wurde mit den Anforderungen der Simulationskomponente abgestimmt.Da viele - für das Information Gathering nützliche - Tools nur unter unixoiden Betriebssystemen existieren, beschränkt sich die Umsetzung dieses Teilprozesses dabei auf Linux. Dadurch, dass das nötige Information Gathering automatisiert wurde, lassen sich bestehende Infrastrukturen einfacher zur Simulation nachbilden (FA-1).

Editor

Der Editor (Abb. 4.2) stellt eine grafische Möglichkeit zur Verfügung, die während des Information Gathering gesammelten Daten zu visualisieren und zu bearbeiten (FA-2, FA-3), oder aber ohne Grundlage vorhandener Daten diese manuell zu erfassen (FA-10). Somit kann das Ergebnis des Information Gatherings veranschaulicht und ggf. nachbearbeitet werden (FA-9).

Stephan Paulsen und Erhan Yilmaz haben den Editor umgesetzt und dabei auf die - vom Information Gathering vorgegebene - Datenstruktur geachtet. Um die Anforderung nach einer Versionierung (FA-4) zu erfüllen, wurde die Struktur geringfügig erweitert, so dass keine Kompatibilitätsprobleme entstehen, welche in der Dokumentation zum Editor festgehalten ist.

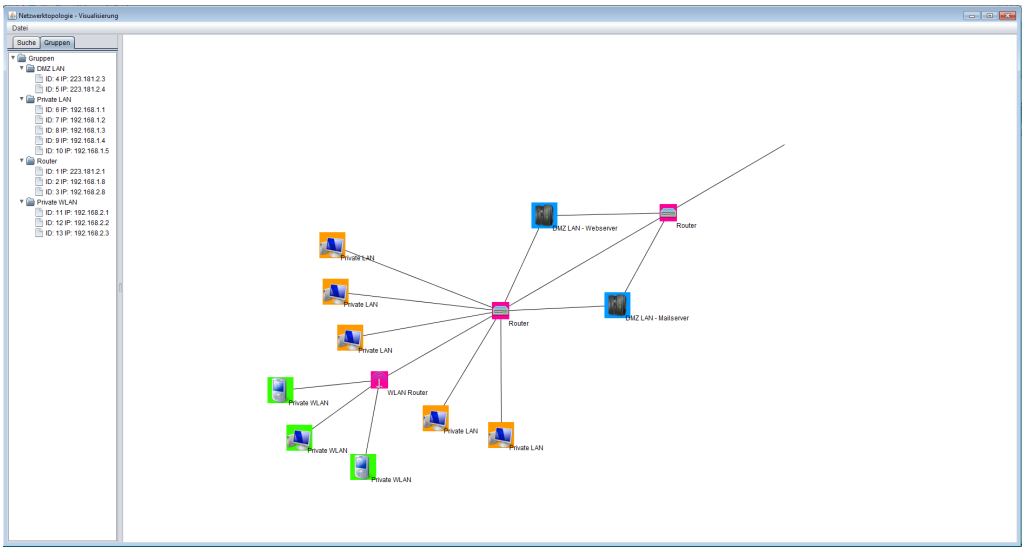


Abbildung 4.2.: Editor Screenshot

Auswertung und Visualisierung

Für die Auswertung und Visualisierung werden teilweise existierende Programme verwendet. Diese werden durch entsprechende Schnittstellen an das Simulationssystem angebunden.

4.2.3. Agentenkommunikation

Da für den Entwurf und die Implementierung - wie in Kapitel 3 erwähnt - ein agentenbasierter Ansatz gewählt wird, und Agenten eigenständig handeln und kommunizieren können, muss diesem Umstand innerhalb des Agentensystems besondere Beachtung geschenkt werden. Um die Anforderungen FA-17-FA-19 zu realisieren, wird unter Berücksichtigung der Anforderung FA-5 (Differenzierbarkeit der Systeme) ein Kommunikationsoverlay (Abb. 4.3) implementiert.

Hierbei werden Netzknoten - dies sind sowohl Endgeräte als auch Netzwerkhardware - auf einzelne, differenzierbare Agenten abgebildet. Diese erhalten Wissen über den gesamten Netzgraphen, um routingfähige Kommunikation zu ermöglichen. Der Netzgraph kann dabei ungerichtet sein und keine Kantengewichte aufweisen, da, wie unter Absatz 4.1.1 erwähnt, auf ausgefeiltes Routing verzichtet werden soll. Aufwändige Routing-Algorithmen sind hierdurch und durch die globalen Kenntnisse nicht nötig. Die Route kann anhand von Shortest-Path-Algorithmen ermittelt werden. Entlang des ermittelten Pfades können dann die Nachrichten gesendet werden. Dadurch, dass auf einem Kommunikationspfad mehrere Agenten liegen können und diese die zu vermittelnden Nachrichten auch entsprechend behandeln müssen,

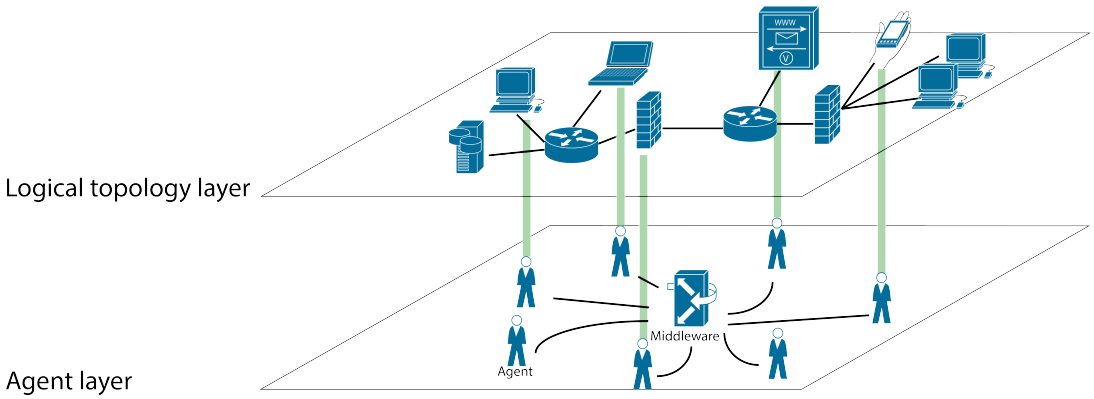


Abbildung 4.3.: Kommunikationsoverlay über typischem Agentenframework

ergibt sich die Möglichkeit, die Anforderung nach Nachrichtenfiltern (FA-19) zu erfüllen. Das Overlay wird flexibel und dynamisch gestaltet, so dass auch die Anforderung FA-20 und FA-13 nach sich ändernden Netzen und Ortsveränderung berücksichtigt wird.

4.2.4. Abbildung von Systemeigenschaften

Erfasste Systemeigenschaften und -verhalten werden auf Agenten abgebildet. Dazu gehören auch installierte Dienste, die offeriert werden (z.B. Netzwerkshares [FA-12]). Dies geschieht, wie in Abschnitt 2.2.1 erläutert, durch Behaviours. Die Gesamtheit von Eigenschaften, Verhaltensweisen und Fähigkeiten definiert einen individuellen Agenten auf einer gegebenen Grundlage (Abb. 4.4). Dieser Umstand lässt sich im Sinne eines Frameworks nutzen und begünstigt Modularität. Somit entsteht ein Baukasten für Agenten, mit dem sich diese frei definieren lassen.

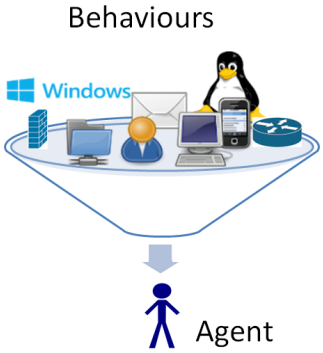


Abbildung 4.4.: Zusammensetzung von Agenten durch Systemeigenschaften

4.2.5. Benutzer und Aktivitäten

Da Benutzer das Verhalten eines Systems beeinflussen, werden die Benutzer selber als Verhalten implementiert, welches einem Agenten, der ein System repräsentiert, angehängt wird. Das Verhalten des Benutzers hat somit die grundsätzliche Möglichkeit das Gesamtverhalten des Systems zu beeinflussen. Das Benutzerverhalten ist dabei durch eine Statemachine definiert, welche je nach Benutzertyp anders ausfallen kann. Somit ist es möglich, durch die Implementierung der Statemachine, das unterschiedliche Verhalten von verschiedenartigen Facharbeitern und deren Handlungen zu modellieren (FA-14, FA-15). Da bei der prototypischen Implementierung kein GIS erforderlich ist (FA-16), aber dennoch Mobilität von Benutzern und Systemen modellierbar sein sollen, sind die Benutzerverhalten dynamisch änderbar, so dass ein Entfernen oder Installieren des Benutzerverhaltens möglich ist.

4.2.6. Agentendefinition

Zu den bereits genannten Punkten Systemeigenschaften und Benutzerverhalten wird ein Agent, welcher ein System repräsentiert, durch seine Beziehung zu anderen Agenten beschrieben. Diese bestimmt, mit welchen anderen Agenten ein Agent direkt kommunizieren kann, um seine Fähigkeiten anbieten und einsetzen zu können. Dies wird durch den in Abschnitt 4.2.3 beschriebenen Ansatz notwendig. Um zudem noch die Anforderung nach filterbaren Nachrichten zu erfüllen, erhält ein Agent die Information welche Arten von Nachrichten gefiltert werden. Formal lässt sich ein Agent wie folgt definieren:

$$A = (B, V, U, F, N_0) \tag{4.1}$$

Wobei:

- B Basisdefinition eines Agenten
- V Menge von Verhaltensweisen/Fähigkeiten
- U Benutzerverhalten
- F Menge von Nachrichtenfiltern
- N_0 bestimmt die Nachbarschaft zum Beginn der Simulation innerhalb des Netzwerks; definiert durch eine Menge von Agenten, wobei $A \notin N_0$

Das Benutzerverhalten U ist optional und wird nur Agenten zugeordnet, die Endnutzergeräte darstellen sowie Interaktionsmöglichkeiten bieten und von einem Benutzer aktuell verwendet werden. Für alle anderen Agenten ist U durch $\emptyset$ definiert. Um eine hohe Flexibilität der einzelnen Verhaltensweisen zu erzeugen und die Modularität zu stärken sind Elemente von V als Tripel (V_k, P, E) spezifiziert. V_k ist dabei ein konkretes Verhalten, P eine Menge von Parametern, die das Verhalten beeinflussen und variieren können. E beschreibt ausnutzbare Schwachstellen[1], deren Gesamtheit die Verwundbarkeit des Systems definieren.

4.2.7. Infektion von Systemen

Eine Infektion eines Systems findet durch Interaktion statt. Dies ist in der Regel eine technische Interaktion, welche - je nach Typ der Malware - durch technische Systeme automatisch oder auch manuell durch einen Benutzer ausgelöst wird. Damit eine Infektion stattfinden kann, müssen bestimmte Voraussetzungen gegeben sein. Diese sind abhängig von der jeweiligen Schadsoftware und den Systemeigenschaften bzw. die hierdurch entstehenden Vulnerabilitäten (FA-11). Da eine Schadsoftware das Systemverhalten verändern oder beeinträchtigen kann, werden diese durch Verhaltensweisen eines Systems beschrieben. Diese besonderen Verhaltensweisen bestehen aus der Beschreibung ihrer Handlungsweise und einer Menge von Vorbedingungen:

$$M_b = (H, P) \tag{4.2}$$

Wobei:

- M_b Malware-Behaviour
- H Handlungsweise/Behaviour
- P Prämisse zur Infektion als Menge von Vulnerabilitäten

Eine Infektion - also ein Installieren des Verhaltens - passiert genau dann, wenn eine entsprechende vorhergehende Aktion dies ermöglicht (z.B. das Öffnen einer infizierten Datei), und

$$P \subset E \tag{4.3}$$

$$H \notin V \tag{4.4}$$

[1] Exploitable Vulnerabilities

gilt.

4.3. Entwurf

4.3.1. Komponenten

Nach der Zusammenstellung von fachlichen und technischen Anforderungen an Simulationsmodell und -anwendung, folgt der Entwurf der Applikation. Hierbei soll das Modell in überschaubare Module aufgeteilt werden, so dass die Anforderung TA-8 erfüllt und der Charakter eines Frameworks hergestellt wird. Hierbei wird sich an den Konzepten und Motiven von Quasar orientiert. Quasar ist die Kurzform von „Qualitätssoftwarearchitektur“, welche das Denken in Komponenten und Schnittstellen in den Mittelpunkt rückt (Siedersleben, 2004), und somit das Erstellen von Modulen mit loser Kopplung und hoher Kohäsion unterstützt.

Abbildung 4.5 zeigt das Komponentendiagramm des Entwurfs auf einem hohen Abstraktionslevel inklusive der ausgegliederten Komponenten. Der Komponentenschnitt ergibt sich dabei durch eine Aufteilung, die Bezug auf die fachlichen Funktionen nimmt. Er geht aus dem angestrebten Workflow, der in 4.2.1 beschrieben ist, und seinen einzelnen Teilschritten hervor. So ergibt sich eine Teilung zwischen für das Information Gathering, das Bearbeiten von Daten, die Simulation sowie die Auswertung und Visualierung von Ergebnissen zuständigen Komponenten. Diese sind durch Schnittstellen miteinander verbunden und werden folgend detaillierter erläutert.

Persistenz

Die Persistenz - in Abb. 4.5 grün gekennzeichnet - bekommt eine besondere Rolle zugeteilt. Da die Arbeitsschritte (siehe 4.2.1) zeitlich unabhängig voneinander ausgeführt werden können, und zwischen den einzelnen Schritten beliebig viel Zeit vergehen kann, werden die Ergebnisse der Teilprozesse persistiert. So stehen sie den anderen Komponenten zur weiteren Verarbeitung zur Verfügung.

Vor allem die Komponente „TopologyPersistence“ nimmt hier eine zentrale Rolle ein, da sie die Arbeitsschritte „Information Gathering“, „Prüfen + Bearbeiten/Erstellen von Informationen“ und „Simulation“ über das gemeinsame Interface „ITopologyInformation“ miteinander verbindet. So schreibt die Komponente „InformationGathering“ ihre Ergebnisse in die „TopologyPersistence“, welche von der „Editor“-Komponente zum Bearbeiten gelesen und gespeichert werden können. Ist die zu testende Topologie erfasst, greift der Simulationskern auf diese Persistenz zu, um die Simulationsumgebung anhand der gespeicherten Daten zu erstellen.

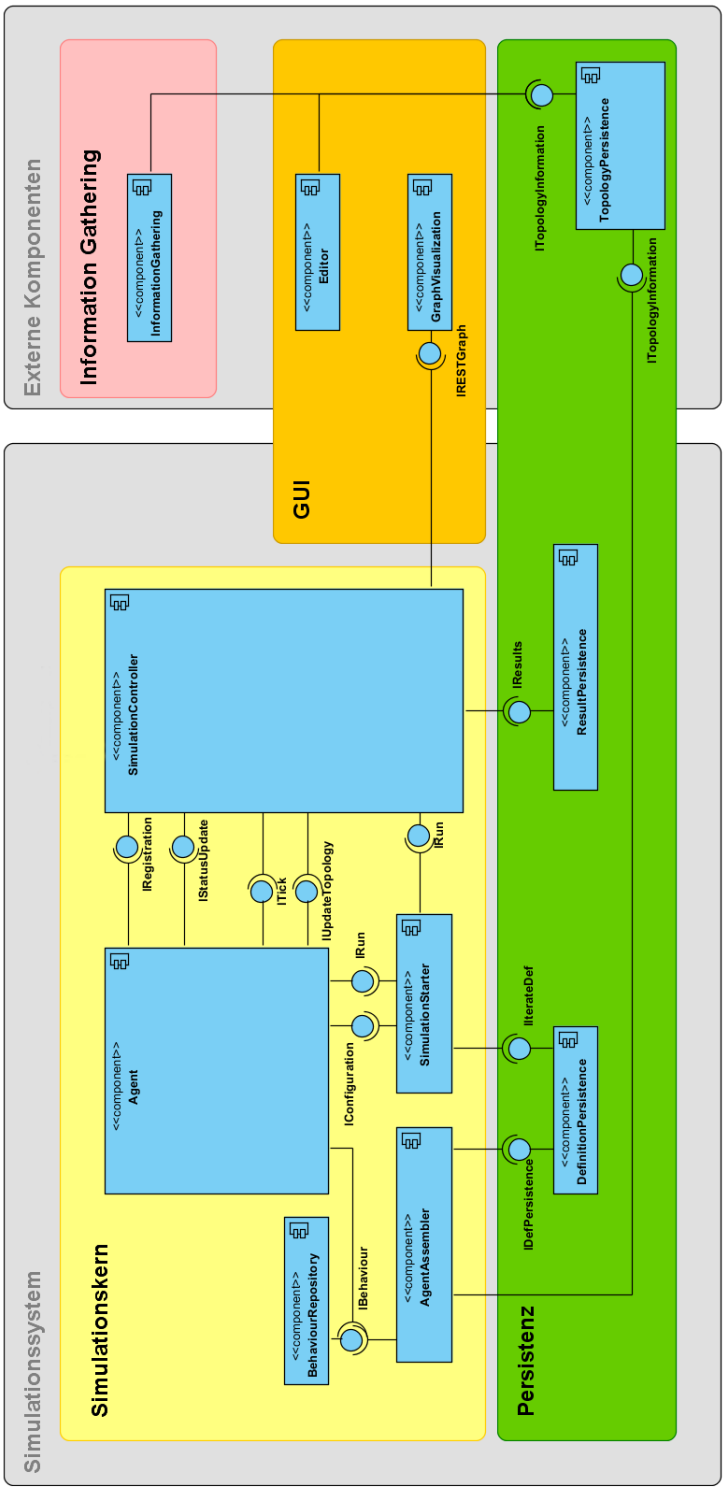


Abbildung 4.5.: Komponentendiagramm des Simulators inkl. Fremdkomponenten

Außerdem existiert die Komponente „DefinitionPersistence“, welche aus der „TopologyPersistence“ generierte Agentenvorlagen speichert, um den Vorgang des Generierens bei mehrmaliger Simulation nur einmal notwendig zu machen. Abschließend ist die Komponente „ResultPersistence“ dafür zuständig, über das Interface „ IResults“ Ergebnisse vom Simulationskern entgegen zu nehmen und zur späteren Beurteilung zu persistieren.

Information Gathering

Das Information Gathering ist durch die gleichnamige Komponente manifestiert, welche - wie bereits erwähnt - durch Robert Krauß entwickelt wurde (Krauß, 2012). Sie sammelt strukturelle Netzwerkinformationen und Systemeigenschaften, welche durch Scans ermittelt werden, und bereitet diese auf. Die Ergebnisse werden mittels der Schnittstelle „ITopologyInformation“ persistiert und stehen anderen Komponenten zur weiteren Verarbeitung zur Verfügung.

GUI

Die GUI-Schicht ist in zwei verschiedene Komponenten unterteilt: Zum einen die „Editor“-Komponente, welche das Bearbeiten und Erstellen von Topologie-Informationen ermöglicht und diese mittels der Schnittstelle „ITopologyInformation“ speichert. Zum anderen existiert die „GraphVisualization“-Komponente, welche eine Visualisierung einer laufenden Simulation und ihrer Ergebnisse ermöglicht. Diese veranschaulicht Kennzahlen, die sich dynamisch ändern können, und ist damit ein Werkzeug zum besseren Verständnis und zur Auswertung von Simulationsergebnissen. Wie noch in Abschnitt 4.4.1 erläutert wird, handelt es sich hierbei um eine Fremdkomponente, die wichtige Visualisierungsoptionen unterstützt und daher nicht selbst entwickelt werden muss. Insbesondere da es sich um die optimierte Darstellung komplexer Graphen handelt, bietet sich das Verwenden einer externen Komponente an. Zum Anbinden an den eigentlichen Simulationskern existiert die Schnittstelle „IRESTGraph“ und ein passender Adapter im Simulationskern.

Anwendungskern

Im Anwendungskern, welcher in Abb. 4.5 gelb hinterlegt ist, sind die Kernkomponenten zusammengefasst, die die eigentliche Simulation vorbereiten und durchführen.

Die Komponente „BehaviourRepository“ stellt eine Liste von zur Verfügung stehenden Verhaltensweisen - genau die, die implementiert sind - für Agenten bereit.

Der „AgentAssembler“ bekommt über die Schnittstelle „ITopologyInformation“ die gesammelten und persistierten Daten zur zu prüfenden Infrastrukur. Diese Informationen werden

zu formalen Definitionen von Agenten konvertiert. Diese Definitionen werden anschließend mittels des Interfaces „IDefPersistence“ persistiert.

Die Komponente „SimulationStarter“ ist für das Initiieren einer Simulation zuständig. Sie instanziiert die Agenten und die nötigen Behaviours, wie sie in der „DefinitionPersistence“ definiert sind. Hierzu iteriert die Komponente mit Hilfe der Schnittstelle „IIterateDef“ über die persistierten formalen Definitionen und konfiguriert die zu erstellende Agenten-Instanz anhand dieser Informationen über das „IConfiguration“ Interface. Abschließend kann die Instanz über die „IRun“ Schnittstelle gestartet werden.

Eine zentrale Rolle innerhalb des Simulationskerns nimmt der „SimulationController“ ein. Diese Komponente steuert die Simulationsabläufe und ist eine zentrale Instanz, an der sich die teilnehmenden Agenten über das Interface „IRegistration“ anmelden. Dadurch erhält sie globales Wissen über die Agenten. Ändert ein Agent seinen Status, z.B. seine Position innerhalb des Netzes, kann dies dem Controller über das Interface „IStatusUpdate“ mitgeteilt werden. Da Agenten, wie unter 4.2.3 beschrieben, dieses globale Wissen ebenfalls - zumindest in Teilen - erhalten sollen, werden Informationen und Veränderungen, die wichtig für jeden Agenten sind, über die Schnittstelle „ITopologyUpdate“ publiziert. Instantiiert wird die Komponente, wie auch Agenten, durch den „SimulationStarter“ und übernimmt danach die Koordination. Dies umfasst auch die Synchronisation der Agenten. Hierfür erstellt die Komponente diskrete Simulationsschritte, welche den Agenten über das „ITick“-Interface mitgeteilt werden. Hierdurch wird auch die Realisierung von Anforderung FA-15 nach zeitlicher Dauer von Aktionen ermöglicht, da über dieses Interface auch die Simulationszeit kommuniziert wird. Da der Controller globales Wissen über die Simulation und die Zustände von Agenten verfügt, eignet er sich als Datenquelle für die Visualisierung und kann gesammelte Ergebnisse persistieren lassen, indem Snapshots vom globalen Zustand durch das „ IResults“-Interface gespeichert werden. Zum Aktualisieren der grafischen Darstellung verwendet der Controller das „IRESTGraph“ der „GraphVisualization“-Komponente.

Einen weiteren elementaren Teil stellt die „Agent“-Komponente dar. Sie repräsentiert einen Netzknoten, dient als Container für Behaviours und anderen Systemeigenschaften, und verhält sich anhand von Anweisungen und Informationen entsprechend. Somit findet hier die eigentliche Simulation statt. Zustandsänderungen werden dem „SimulationController“ über das Interface „IStatusUpdate“ mitgeteilt. Die Konfiguration, die der Agent erhält, enthält die nötigen Informationen, mit denen der Agent seine Behaviours aus dem „BehaviourRepository“ laden kann.

4.4. Implementierung

Bei der Implementierung wird darauf geachtet, dass das Erweitern des Frameworks möglichst einfach ist. Außerdem wird die geplante Modularität und Plattformunabhängigkeit (TA-1, TA-8) bei der Wahl der Implementierungsmittel berücksichtigt. Da Python plattformunabhängig ist, durch seine dynamische Typisierung die gewollte Modularität effizient umsetzen lässt und sich zudem für Rapid-Prototyping eignet, ist der Simulationskern in Python implementiert.

4.4.1. Werkzeuge

Bei der Entwicklung wird auf verschiedene bestehende Werkzeuge und Bibliotheken zurückgegriffen. Da diese elementare Aufgaben innerhalb des Frameworks übernehmen und die konkrete Implementierung hiervon abhängt, wird auf die wichtigsten folgend eingegangen.

SPADE-Framework

Smart Python multi-Agent Development Environment (SPADE) ist ein in Python geschriebenes Framework zur Entwicklung multiagentenbasierter Software, welches zur Kommunikation auf das Extensible Messaging and Presence Protocol (XMPP) setzt. SPADE erlaubt durch Verwenden von XMPP eine Verteilung auf mehrere Hosts und den Einsatz eines beliebigen Kommunikationsservers. Möchte man eine große Anzahl von Agenten betreiben, die stark kommunikativ sind, bietet sich ein Austausch des von SPADE mitgebrachten XMPP-Servers, durch den in Erlang geschriebenen ejabberd-Server, an. Dieser unterstützt Clustering und lässt sich somit einfach skalieren. Außerdem ist eine P2P-Kommunikation zwischen Agenten möglich, die den Kommunikationsserver nicht weiter belastet.

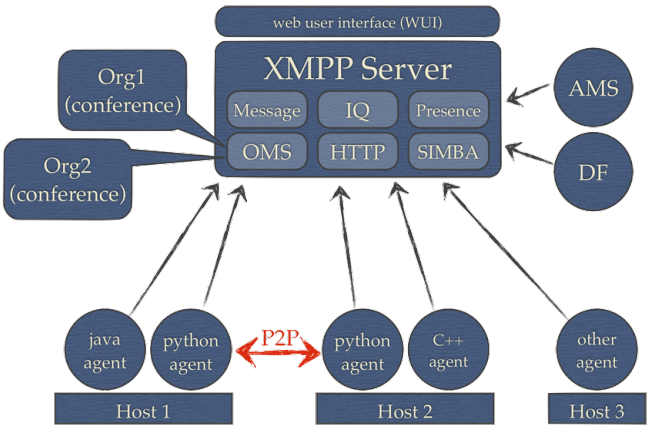


Abbildung 4.6.: Übersicht SPADE Kommunikation und Verteilung (aus SPADE Dokumentation)

Durch die Einhaltung von FIPA-Standards lassen sich neben Python-Agenten auch Agenten in beliebigen anderen Sprachen implementieren (siehe Abb. 4.6). Als Userinterface stellt SPADE eine Weboberfläche mit grundlegenden Funktionen bereit. Hier sind angemeldete Agenten, Dienste und ausgetauschte Nachrichten einsehbar.

SPADE bietet zum Erstellen von Behaviours verschiedene Klassen an, die es ermöglichen unterschiedliche Arten von Verhalten zu erstellen:

- Zyklische und periodische Behaviours: Zum Realisieren sich wiederholender Aufgaben.
- One-Shot und Time-Out Verhalten: Zum Implementieren normaler - nicht wiederkehrender - Aufgaben.
- Finite State Machine: Erlaubt das Erstellen komplexer Behaviours.
- Event Behavior: Reagiert auf bestimmte Ereignisse, die ein Agent wahrnimmt.

Behaviours können dabei flexibel einem Agenten hinzugefügt werden und bestimmen so seine Fähigkeiten und auch Eigenschaften. Behaviours können außerdem Nachrichten erzeugen und empfangen. Der Empfang der Nachrichten geschieht durch einen internen Dispatcher. Somit erreicht - wie Abb. 4.7 zeigt - eine Nachricht nicht nur den korrekten Agenten, sondern außerdem noch entsprechende Verhalten, das auf die Nachricht entsprechend seiner Intention reagieren kann.

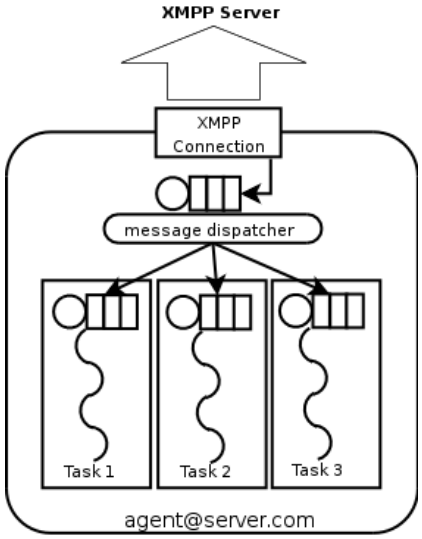


Abbildung 4.7.: SPADE Agent Model (aus Gregori et al. (2006))

Da die zum Zeitpunkt der Erstellung dieser Arbeit existierende SPADE-Version[2] nicht kompatibel mit Python 3 ist, sind auch die implementierten Komponenten nicht zwingend Python 3 kompatibel. Als Interpreter kommt PyPy für Python 2.7 zum Einsatz.

NetworX

Zur Abbildung und Verwaltung des Netzgraphen wird die Python Bibliothek NetworkX[3] verwendet. Diese bietet neben den nötigen Datenstrukturen zum Abbilden von komplexen Graphen auch Messverfahren für gängige Graphenmetriken und die Kalkulation von Pfaden im Graphen. Darüber hinaus besteht die Möglichkeit mit NetworkX erstellte Graphen in andere Datenformate, wie JavaScript Object Notation (JSON), zu exportieren, was zum Datenaustausch und Persistieren genutzt wird.

Gephi

Zur Visualisierung während der Laufzeit wird Gephi[4] (Abb. 4.8) verwendet.

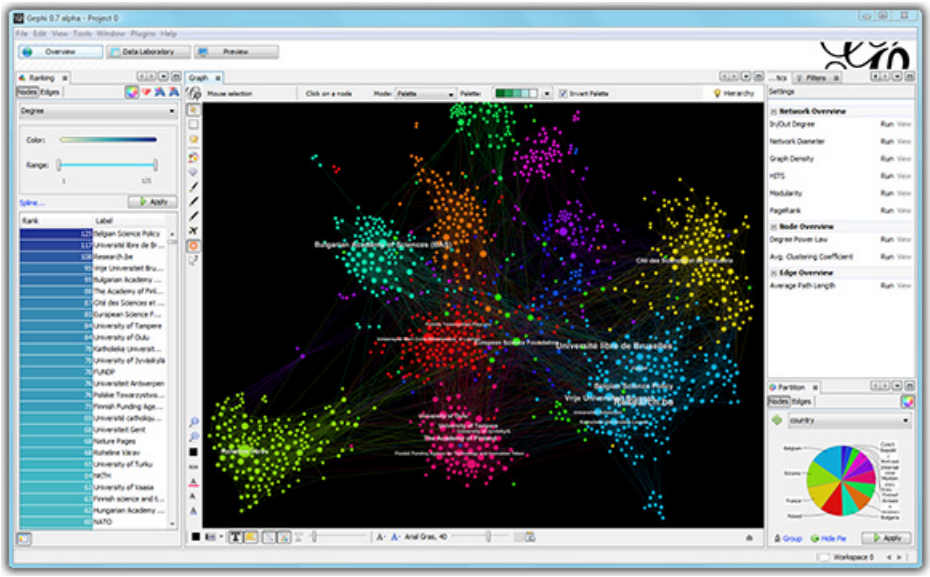


Abbildung 4.8.: Gephi Screenshot (von Gephi Homepage)

[2] https://github.com/javipalanca/spade Version 2.0 (01.10.2013)
[3] http://networkx.github.io/ (02.01.2014)
[4] http://gephi.org/ Version 0.8 (03.01.2013)

Hierbei handelt es sich um ein Open Source Programm zur Visualisierung von Graphen, welches auf NetBeans aufsetzt und in Java implementiert ist. Somit ist auch die Anforderung nach Plattformunabhängigkeit der Visualisierung (TA-4) erfüllt.

Gephi bietet die Möglichkeit eine sogenannte Data-Library zu pflegen, die als Grundlage für die Visualisierung dient. Hier lassen sich neben diversen Graphen-Metriken, die Gephi selber berechnen kann, auch benutzerdefinierte Felder erstellen, welche zum Beispiel den Infektionsstatus eines Systems enthalten können. Somit lassen sich der Infektionsverlauf und andere Kennzahlen zur Laufzeit visualisieren (TA-6, TA-7). Sollte eine gewünschte Metrik in Gephi nicht implementiert sein, so kann diese per Plug-In implementiert werden. Eine weitere Möglichkeit stellt das Importieren der Messwerte aus einer externe Datenquelle - z.B. NetworkX CSV oder JSON - dar.

Als Schnittstelle zu Gephi wird das Graph Streaming Interface, welches in Gephis Graph Streaming Plug-In realisiert ist, verwendet. Diese Schnittstelle wird per HTTP und JSON bedient. Somit ist es möglich, die Schnittstelle über das Netzwerk zu verwenden und die grafische Oberfläche auf einem anderen System laufen zu lassen, als die Simulationsumgebung (TA-5).

4.4.2. Umsetzung von Agenten

Da Agenten zentraler Bestandteil der Simulation sind und unterschiedliche Systeme repräsentieren sollen, legt die Implementierung besonderen Fokus auf Flexibilität und Modularität. Hier wird folgend auf den internen technischen Aufbau und die konkrete Definition eingegangen.

Agenten-Definition

Die Definition von Agenten erfolgt im JSON-Format. Dies ermöglicht ein einfaches Speichern und Laden der Information via Python, da die Python Standard Library ein JSON-Modul zum Umgang mit JSON enthält. Hierbei werden Dictionaries und Listen entsprechend umgewandelt oder erzeugt.

Die Komponente „AgentAssembler“, welche als einfaches Script implementiert ist, erzeugt JSON-Dateien, deren Inhalt vom Agenten interpretiert werden kann. Listing 4.1 zeigt eine exemplarische Definition für einen CIFS[5]-Server im JSON-Format. Da es sich um einen reinen Server handelt, ist kein User-Behaviour angegeben.

```
1 {
2     "agent_class": "basicagent.BasicAgent",
```

[5]Common Internet File System (CIFS)

```
    "behaviours": [
        {
            "class": "cifs",
            "params": {
                "shares": [
                    "plaene",
                    "ablauf",
                    "dokumente"
                ],
                "size": 125829120,
                "uar": [
                    {
                        "access": [
                            "plaene"
                        ],
                        "name": "user1"
                    }
                ]
            }
        }
    ],
    "description": "No IP assigned",
    "filter_rules": [],
    "host": "localhost",
    "name": "20",
    "neighbours": [],
    "secret": "joshua",
    "type": "NAS"
}
```

Listing 4.1: Beispielhafte Agentendefinition

Innerhalb des JSON-Dokuments sind die in 4.2.6 aufgeführten Elemente wiederzufinden. So ist mit *agent_class* die Basisdefinition eines Agenten gegeben. In *behaviours* stehen die Verhaltensweisen mit ihren Einstellungen und *filter_rules* gibt die Filterregeln an. Durch *neighbours* ist N_0 - also die Nachbarschaft - festgelegt. Rein informativen Nutzen hat das Feld *description* und spielt für die eigentliche Definition keine weitere Rolle. Verzichtet wird auf das Definieren von Vulnerabilitäten, da diese statisch im Behaviour hinterlegt und durch dieses mitgeteilt werden. Die technischen Gegebenheiten erfordern es, die Definition um technische Elemente zu erweitern. So sind *host*, *name* und *secret* Verbindungsdaten zum XMPP-Server von SPADE.

Unter Verwendung des „SimulationStarter“ wird die beschriebene Definition geladen und die unter *agent_class* stehende Klasse instantiiert. Da es sich hierbei im Prinzip nur um einen Class-Loader handelt, wird nicht detailliert auf diese Umsetzung dieser Komponente eingegangen.

Agenten-Klasse

Wie in 4.2.4 erwähnt, ist es erwünscht, dass ein Agent sich maßgeblich durch seine Verhalten definiert und diese verschiedene Systemeigenschaften repräsentieren. Die Implementierung berücksichtigt dies und stellt einen möglichst generischen Agenten zur Verfügung, der sich dynamisch erweitern lässt. Teile der konkreten Umsetzung sind im Code-Listing 4.2 abgebildet.

```
1  class BasicAgent(spade.Agent.Agent):
2
3      def __init__(self, config):
4          super(BasicAgent,self).__init__(config["name"]+"@"+config["host"],
               config["secret"])
5          self.config = config
6
7          self.name = config["name"]
8          self.host = config["host"]
9          self.neighbours = config["neighbours"]
10         self.routing_messagequeue = Queue.Queue()
11         self.topology = nx.Graph()
12
13         self._behaviours = []
14         self._userbehaviour = None
15         self._type = config["type"]
16
17         self._status = {} #status information
18         self._malware = None
19
20     def _setup(self):
21
22         #subscribe to events
23         pubsub.subscribe(self, "topo_change", topopresence._BHSubTopoUpdate
               ())
24
25         topology_presence_bhv = behaviours.TopologyPresence.TopologyPresence
               (self.name, self.neighbours)
26         self.addBehaviour(topology_presence_bhv, None)
27
28
29         #add routing ability
```

```
30          self.config["filter_rules"] = []
31          routing_bhv = behaviours.routing.behaviour.Routing(filter_list=self.
                config["filter_rules"])
32          self.addBehaviour(routing_bhv, routing_bhv.getTemplate())
33          routingsender_bhv = behaviours.routing.behaviour.RoutingSender()
34          self.addBehaviour(routingsender_bhv)
35
36          #add traceroute receiver
37          tracert_bhv = behaviours.routing.behaviour.TraceRouteReceiver()
38          self.addBehaviour(tracert_bhv, tracert_bhv.getTemplate())
39
40          #add infection receiver
41          infect_bhv = self._BHInfect()
42          self.addBehaviour(infect_bhv, infect_bhv.getTemplate())
43
44          if self.config.has_key("behaviours"):
45              self._setup_behaviours(self.config["behaviours"])
46
47          if self.config.has_key("malware"):
48              self.infect(set(config["malware"]["exploit"]), config["malware"
                    ]["class"])
49
50          if self.config.has_key("user"):
51              user_bhv = classloader.get_class(malware)
52              self._userbehaviour = user_bhv()
53              self.addBehaviour(self._userbehaviour)
54
55      def _setup_behaviours(self, behaviours):
56
57          for bhv in behaviours:
58              bhv_class = classloader.get_class(bhv['class'])
59              bhv_inst = bhv_class(**bhv['params'])
60              self._behaviours.append(bhv_inst)
61              self.addBehaviour(bhv_inst, bhv_inst.getTemplate())
62
63      ...
64      ...
65
66      def on_topology_received(self, graph):
67          print("topo received")
68          self.topology = graph
```

Listing 4.2: Auszug aus basicagent.py

Die Klasse *BasicAgent* leitet sich von dem von SPADE mitgelieferten Agenten ab. Sie ist als Grundlage für Agenten des erstellten Frameworks gedacht.

Durch dynamisches Erzeugen von Objekten werden die Behaviours „on Demand“ einem Agenten hinzugefügt. Dies geschieht in der Funktion *_setup_behaviours*, die eine Menge von Behaviournamen mit passenden Parametern entgegen nimmt. Diese Menge stammt aus der Agentendefinition, und wird iterativ abgearbeitet. Die darin enthaltenen Elemente werden nacheinander instantiiert und dem Agenten gemäß SPADE-Framework hinzugefügt. Bei der Instantiierung werden außerdem Parameter übergeben, die aus der Agentendefinition stammen, und je nach Behaviour verschiedene Bedeutungen haben können. Hierbei wird sich die Unpacking-Operation von Python zu Nutze gemacht. Diese erlaubt es in diesem Fall, aus einem Dictionary (in anderen Sprachen auch Hash-Map) eine Liste von sogenannten Named Arguments zu erstellen. Diese ermöglichen eine flexiblere Parameterübergabe, bei der die Reihenfolge der Parameter egal ist und eine Zuordnung über den Parameternamen erfolgt. Hierdurch wird die Entwicklung von Behaviours und die Weiterentwicklung des AgentAssemblers vereinfacht, da Reihenfolgen innerhalb des Formats der Agentendefinitionen keine Rolle spielen. Somit können ein Verändern der Reihenfolge oder ein Hinzufügen von Parametern nicht zu Inkompatibilitäten führen.

Neben den nötigen Mechanismen zur flexiblen Erweiterbarkeit enthält er auch andere wichtige Funktionalitäten, die es dem Agenten ermöglichen an einer Simulation teilzunehmen. Hierzu zählen alle Funktionen, die Verbindung zum Simulationscontroller aufnehmen. Diese sind als Agent-Behaviours implementiert und werden innerhalb der *_setup*-Methode installiert (Listing 4.2 23-26). Diese registrieren sich beim Simulationscontroller und melden sich beim Event zum Empfangen von Änderungen des Netzes an, um aktuelles und vollständiges Wissen über die Umgebung zu erlangen. Dieses wird über einen Publish-Subscribe-Kanal verteilt. Der gesamte Anmeldevorgang und Informationsaustausch erfolgt so, wie in Abb. 4.9 schematisch veranschaulicht.

Der Agent fügt sich dabei selber das *TopologyPresence*-Behaviour hinzu, welches für das Versenden des Agentenzustandes - wie z.B. seine Nachbarschaft zu anderen Agenten aussieht - zuständig ist. Daraufhin - und auch wenn ein anderer Agent seinen Zustand an den Simulationscontroller übermittelt, erhält der Agent das globale Wissen über die simulierte Umgebung vom Simulationscontroller über den Publish-Subscribe-Mechanismus.

Außerdem umfasst der Agent den Routing-Algorithmus, der wie in 4.2.3 beschrieben umgesetzt ist. Folgend wird auf diesen genauer eingegangen.

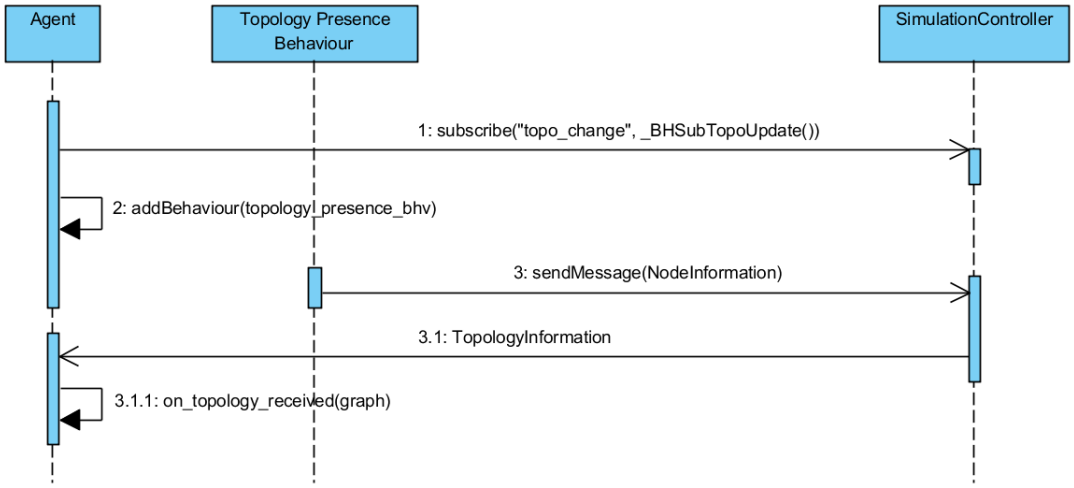


Abbildung 4.9.: Agent-Subscription Sequenzdiagramm

Routing

Da das Routing von Nachrichten eine besondere Rolle bei der Betrachtung der Topologie spielt, wird die konkrete Implementierung hier genauer beschrieben. Die Umsetzung des unter 4.2.3 beschriebenen Ansatzes findet im Modul *behaviours.routing.behaviour* und *behaviours.routing.util* der Agenten statt. Hier werden neben der nötigen Funktionalität auch Hilfsmittel implementiert, die zur Nutzung des Routings für andere Behaviours notwendig sind.

Für das Routing wird die eigentlich zu übermittelnde Nachricht - die Payload - in eine Routing-Nachricht gekapselt. Diese wird mit einem Routing-Header versehen, der eine Liste der geplanten Hops enthält. Diese wird mittels des globalen Wissens des sendenden Agenten und dem Dijkstra-Algorithmus erstellt, und enthält somit einen Shortest-Path zum Ziel innerhalb des simulierten Netzwerks.

Bei jedem Hop, wird ein Element der Liste entfernt und die Routing-Nachricht an den nächsten Knoten weiter geschickt. Der letzte Knoten wandelt - wie in 4.10 illustriert - die Payload dann in ein passendes Paket um, so dass der Dispatcher-Mechanismus von SPADE (siehe 4.4.1) die Nachricht beim Empfänger korrekt weiter verarbeiten kann.

Neben dem eigentlichen Routing sind in den bereits erwähnten Modulen eine Klasse zum Erstellen der Routing-Messages und Behaviours zum Testen des Routings enthalten, welche ein Traceroute ermöglichen. Hiermit kann abgefragt werden, welche Knoten eine Nachricht durchlaufen würde und es werden die benötigten Zeiten zum Zustellen einer Nachricht ermittelt.

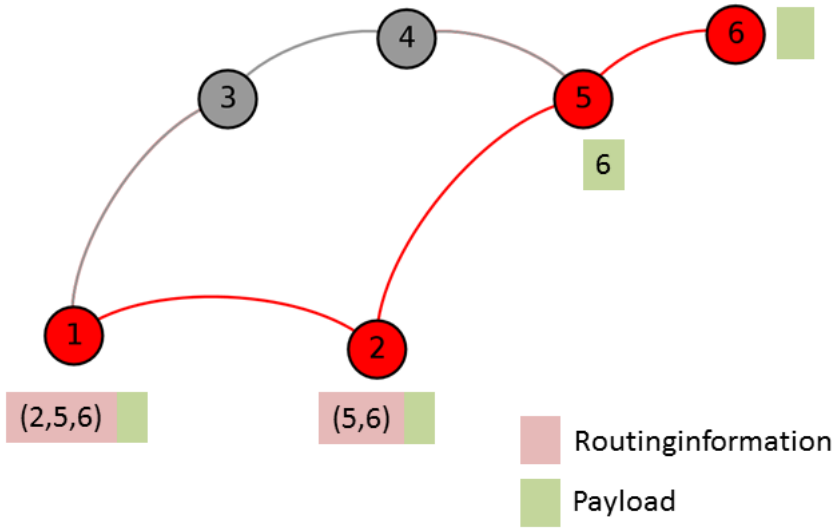


Abbildung 4.10.: Routing, Beispiel

Die Traceroute-Funktionalität ist dem *BasicAgent* schon beigefügt (siehe Code-Listing 4.2 Zeile 37-38).

Zusätzlich zu seiner Hauptaufgabe, dem Routing, kann das Routing-Behaviour durch Callback-Funktionen erweitert werden. Die Methode *addPostProcessReceiveHandler* nimmt ein Funktions-Objekt entgegen, welches einen Parameter entgegennimmt. Nach dem Empfang einer Routing-Nachricht wird jede der Callback-Funktionen aufgerufen und ihm der Inhalt der Routing-Nachricht übergeben. Diese Funktionalität ist für das Implementieren von Messfunktionen gedacht. Diese können z.B. Messwerte erfassen, die etwas über die tatsächliche Nutzung des Knotens oder seiner Beteiligung bei der Schadsoftwareausbreitung aussagen; und sind somit unabhängig von theoretischen Graphenmetriken.

4.4.3. Persistenzen

Die Persistenz wird innerhalb des gesamten Prozesses häufig als Schnittstelle verwendet. Daher, und weil verschiedene Arten von Persistierung genutzt werden, wird hier kurz auf die konkrete Implementierung der einzelnen Persistenzkomponenten eingegangen.

TopologyPersistence

Die Persistenz, die dem Simulationskern als Schnittstelle zum InformationGatherer und Editor dient, ist durch eine SQLite Datenbank realisiert. Die Daten werden relational gespeichert und

können über die Vielzahl an verfügbaren Bibliotheken für unterschiedliche Programmiersprachen per SQL abgerufen werden. Das verwendete Datenmodell wird von Krauß (2012) in seiner Bachelorarbeit beschrieben.

DefinitionPersistence

Die Definitionen, die vom AgentAssembler erstellt werden, werden als Plaintext Dateien im Dateisystem abgelegt. Für jede Agenten-Definition liegt somit eine JSON-Datei innerhalb eines Ordners im Dateisystem. Über diese kann mit gängigen Systemfunktionen iteriert werden, so dass die Definitionen sequenziell geladen werden können.

ResultPersistence

Um Ergebnisse noch nach einer laufenden Simulation abrufbar zu machen, werden Snapshots des Gesamtzustands des simulierten Netzes erstellt. Da dieser innerhalb der dafür zuständigen Komponente (*SimulationController*) vorhanden ist, und in einem Graph-Objekt vorgehalten wird, welches sich einfach - wie bereits in Abschnitt 4.4.1 erwähnt - in ein JSON-Dokument umwandeln lässt, sowie aus Gründen der Verteilbarkeit, wird zum Speichern der Snapshots eine dokumentenorientierte Datenbank verwendet. Hierfür wird CouchDB genutzt, weil diese ebenfalls mit JSON arbeitet und unnötiges Konvertieren so vermieden wird. Ein weiterer Vorteil, der durch die Verwendung einer dokumentenorientierten Datenbank entsteht, ist das leichte Ändern - insbesondere das Erweitern - der gespeicherten Datenstrukturen, da dokumentenorientierte Datenbanken keinem starren Datenschema unterliegen.

4.4.4. SimulationController

Als zentrale Komponente, die die Koordination der Agenten übernimmt und auch die zentrale Einheit zum Austausch und Persistieren von Simulationsdaten darstellt, dient der Simulationscontroller. Dieser ist selbst als SPADE-Agent implementiert und stellt eine Sonderform dar. Im Gegensatz zu der unter 4.4.2 vorgestellten Implementierung von Agenten, die Systeme abbilden, fehlen dem SimulationController Routing-Möglichkeiten und er enthält keine Informationen zu Nachbarschaften. Allerdings wird auch dieser SPADE-Agent mittels der bereits vorgestellten formale Definition durch ein JSON-Dokument (siehe Listing 4.3) beschrieben und kann daher ebenfalls durch den „SimulationStarter“ instantiiert werden.

```
1 {
2     "agent_class": "simulation_controller.SimulationController",
3     "agent_list": [
4         "node1",
```

```
        "node2",
        "node3",
        "node4",
        "node5",
        "node6",
        "node7",
        "node8",
        "node9"
    ],
    "couch_access": {
        "database": "simresults",
        "url": "http://localhost:5984/"
    },
    "host": "localhost",
    "name": "topo_man",
    "secret": "joshua",
    "start_time": 1391238000,
    "time_resolution": 60
}
```

Listing 4.3: Beispielhafte Definition für den Simulationscontroller

Damit andere Agenten wissen, wie diese zentrale Kontrollinstanz zu erreichen ist, um seine Dienste nutzen zu können, trägt sich der SimulationController im Verzeichnisdienst der SPADE-Middleware ein. Hierüber kann die zur Kommunikation nötige Adresse ermittelt werden.

Simulationsstart

Der Zeitpunkt für den Simulationsstart wird automatisch ermittelt. Dies geschieht durch einen Vergleich, mit den in der Controller-Konfiguration (siehe Listing 4.3) unter *agent_list* angekündigten Agenten, mit der Menge an tatsächlich angemeldeten Agenten. Sind beide Mengen identisch, wird das Timeslicing gestartet und die Simulation läuft.

Simulationsschritte

Der Simulationscontroller erzeugt diskrete Zeitschritte (Timeslicing), welche die Simulationszeit bilden, um die Aktionen der anderen Agenten zu synchronisieren. Die Länge dieser Zeitschritte lässt sich über den Parameter *time_resolution* einstellen, welche in Sekunden angegeben wird. Ein niedriger Wert erhöht die zeitliche Auflösung, führt aber auch zu mehr Einzelschritten, was die Simulationsdauer und den Overhead erhöht. Zu Beginn eines Schrittes

werden alle Agenten über das Fortführen der Simulation informiert. Hierfür wird den Agenten eine Nachricht mit der aktuellen Simulationszeit geschickt. Das zeitliche Delta zwischen zwei Simulationsschritten kann von den Agenten genutzt werden, um das Ausführen von Aktionen - wie z.B. das Lesen von E-Mails - zeitlich korrekt zu simulieren. Nach dem Ausführen der möglichen Aktion, melden alle Agenten ihren aktuellen Status (z.B. über Infektionen) dem Simulationscontroller. Ist dies von allen Agenten durchgeführt worden, erzeugt der Controller den nächsten Timeslice. Nach jedem Simulationsschritt wird der Zustand des simulierten Netzes persistiert, um an die Simulation anschließend die Daten auswerten zu können.

4.4.5. Infektion von Systemen

Bei der Simulation von Schadsoftwareausbreitung spielt das eigentliche Infizieren eines Systems eine elementare Rolle. Daher wird hier auf auf die konkrete Umsetzung zur Systeminfektion eingegangen, welche sich auf den in Abschnitt 4.2.7 erstellten Entwurf stützt. Hiernach soll das Verhalten der Schadsoftware als Agent-Behaviour realisiert werden und eine Infektion unter Ausnutzung vorhandener Schwachstellen möglich sein. Die Umsetzung sieht vor, dass ein Verhalten seine Verwundbarkeiten mittels der Methode *get_vulnerabilities* zur Verfügung stellt. Diese werden durch CVE-IDs[6] identifiziert und als Menge (*set*) bereitgehalten. Das Iterieren über alle Behaviours eines Agenten und die Vereinigung aller Mengen ergibt die Gesamtmenge an Vulnerabilitäten eines Systems. Hat eine Aktion eine potentielle Infektion zur Folge, zum Beispiel das Öffnen infizierter Dateien auf einem Netzwerkshare, so wird dem Agenten die potentielle Infektion mitgeteilt. Dies geschieht, indem ihm eine Nachricht geschickt wird, die die ausgenutzten Vulnerabilitäten in Form von CVE-IDs und den Namen, der im Erfolgsfall zu installierende Schadsoftwareverhaltens, enthält. Die Auswertung der Bedingungen zu einer Infektion sowie das Installieren des Behaviours geschehen dann im Agenten (siehe Listing 4.4). Die Überprüfung ist durch einen Vergleich der CVE-ID-Mengen realisiert, welche testet, ob die zur Infektion nötige Menge CVE-IDs eine echte Teilmenge der Vulnerabilitäten des Agenten ist.

```
1 ...
2
3     def infect(self, exploit, malware):
4         #collect vulnerabilities
5         vulnerabilities = set()
6         for bhv in self._behaviours:
7             #check if behaviour has vulnerabilities
8             if callable(getattr(bhv, "get_vulnerabilities", None)):
```

[6] Common Vulnerabilities and Exposures (CVE)

```
9                   vulnerabilities = vulnerabilities.union(bhv.
                        get_vulnerabilities()) #add them to existing ones
10
11          #if agent is vulnerable to all CVEs, we can assume it as infected
12          #=> vuln is strict subset of system vulnerabilities
13          if vulnerabilites > exploit:
14              self._status["infected"] = True
15
16              mal_class = classloader.get_class(malware)
17              self._malware = mal_bhv()
18              self.addBehaviour(self._malware)
19
20  ...
```

Listing 4.4: Infektionsrelevanter Auszug aus basicagent.py

Ist dies der Fall, wird das angegebene Verhalten der Schadsoftware dynamisch geladen und dem Agenten hinzugefügt. Hierfür wird die bestehende Funktionalität verwendet, die auch schon für das Installieren von Behaviours beim Starten des Agenten benutzt wird. Das Malware-Behaviour kann dabei die Funktionalität des Agenten beliebig beeinflussen. Dies geschieht durch Monkey Patching [7]. Hiermit lässt sich der infizierte Agent beliebig beeinflussen und erweitern. Eine Überprüfung, ob die Einflussnahme die Möglichkeiten, welche in der Realität durch die verwendeten Exploits eröffnet werden, übersteigt, findet nicht statt. Diese Verantwortung liegt beim Entwickler des Malware-Behaviours.

[7] auch bekannt als Duck Punching

5. Fallstudie

Zur Beantwortung der Eingangs gestellten Fragen, ob ein technisch relevanter Knoten auch eine hohe Relevanz bei der Ausbreitung von Schadsoftware hat, wird ein exemplarisches Experiment mittels des vorgestellten Simulations-Frameworks durchgeführt. Hierfür dient ein definiertes Szenario als Grundlage, nach dem sich die konkrete Konfiguration richtet. Folgend wird dieses Szenario kurz erläutert und die Konfiguration - die spezifische Umsetzung - beschrieben.

5.1. Szenariobeschreibung

Das Szenario, welches als Fallbeispiel dienen soll und anhand dessen Möglichkeiten zum Treffen von Aussagen gefunden werden sollen, basiert auf einem Angriff mittels einer APT. Die simulierte Infrastruktur bildet ein einfaches fiktives Netzwerk eines mittelständischen Unternehmens, oder auch einer mittelgroßen Organisation, nach. Es wird davon ausgegangen, dass die Organisation über zwei verschiedene Standorte verteilt ist und diese durch ein Virtual Private Network (VPN) miteinander verbunden sind. An einem dieser Standorte befindet sich ein Industrial Control System (ICS), welches abgetrennt vom restlichen Netz ist, und durch mobile Endgeräte (Laptops) programmiert wird. Sobald diese infiziert sind, kann das ICS kompromittiert werden[1]. Es wird für das Szenario angenommen, dass es sich bei dem ICS um eine Steueranlage zur Fertigung von Werkstücken handelt. Diese wird durch Einspielen von CAD-Vorlagen (Konstruktionspläne) programmiert, welche durch eine Entwicklungsabteilung erstellt wird und von einer Fertigungsabteilung abgerufen werden. Zusätzlich existiert noch eine Verwaltung. Hieraus ergeben sich drei unterschiedliche Nutzergruppen mit unterschiedlichem Verhalten, die jeweils auf einen der Teilnetze des VPNs verteilt sind. Ein Austausch der Vorlagen findet über Netzwerkshares mittels CIFS statt, auf den die Verwaltung keinen Zugriff hat, was zusätzlich zur Zugriffskontrolle über eine Firewall sichergestellt wird. Andere Informationen werden per E-Mail ausgetauscht. Somit ergibt sich ein Netzwerk, wie es in Abbildung 5.1 zu sehen ist.

[1] FA-24

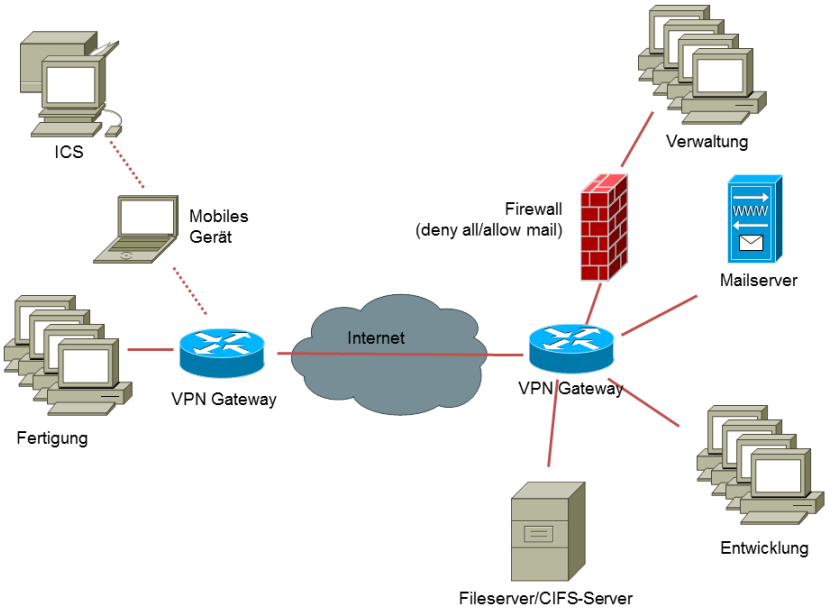


Abbildung 5.1.: Netzdiagramm, Übersicht

Es wird weiterhin davon ausgegangen, dass 147 Endgeräte existieren, die auf die drei genannten Bereiche verteilt sind und von denen eines mobil ist. Zusätzlich existieren 13 weitere Geräte, die zur Vermittlung (z.B. Router) genutzt werden oder Dienste zur Verfügung stellen.

5.1.1. Systemlandschaft

Sämtliche verwendeten Endgeräte nutzen Windows 7 in einer 32-Bit Version als Betriebssystem, welches eine ausreichend hohe Marktdurchdringung hat[2], und durch das Auslaufen des erweiterten Supports im Jahre 2020 auch bis hierhin noch Bedeutung haben dürfte. Installiert ist weiterhin das Service Pack 1. Neben dem Betriebssystem werden zwei weitere wichtige Produkte verwendet: Adobe Reader in Version 11.0.01 zum Anzeigen von PDF-Dokumenten und AutoCAD 2010 zum Erstellen und Anschauen von technischen Zeichnungen. Letzteres ist auf den Endsystemen der Verwaltung nicht installiert.

5.1.2. Schadsoftware und Propagationsvektoren

Die simulierte Schadsoftware, macht sich die in Tabelle 5.1 vermerkten Lücken in der genannten Software zu Nutze.

[2] 55% der Desktopsysteme, Stand Dezember 2013; Quelle: statcounter.com

CVE	Kurzbeschreibung	Anwendung	Ziel
CVE-2013-3660	Privilege Escalation	Windows	Erlangen höherer Privilegien zum Installieren eines Treibers, der Systemfunktionen manipuliert
CVE-2013-2718	Arbitrary Code Execution	Adobe Reader	Initialer Angriffsvektor zum Installieren eines schadhaften Treibers
CVE-2010-5241	DLL Hijacking	AutoCAD 2010	Ausführen beliebigen Codes zum Installieren eines schadhaften Treibers zur Propagation

Tabelle 5.1.: Von simulierter Malware ausgenutzte Schwachstellen

Im Rahmen des Szenarios wird von einer initialen Infektion mittels einer Spear-Phishing Attacke per E-Mail und manipuliertem PDF-Dokument ausgegangen. Zur eigentlichen Verbreitung werden Netzwerkfreigaben genutzt, die bei Dateisystemaktionen durch den Treiber (siehe Tabelle 5.1) erkannt werden. Dieser kopiert dann eine schadhafte DLL auf das Netzlaufwerk, welche die angesprochene DLL Hijacking Lücke ausnutzt, um den erwähnten Treiber wiederum auf dem nächsten verwundbaren Knoten zu installieren. Um möglichst wenig aufzufallen, wie es eine echte APT ebenfalls machen würde, breitet sich die Schadsoftware von einem befallenen Endsystem nur drei mal aus und propagiert nicht weiter[3].

Durch diese Grundannahmen - vor allem in den Ausbreitungsvektoren - ergibt sich ein ähnliches Szenario, wie es bei Stuxnet vorhanden war Symantec Corporation (2011), was die Relevanz eines solchen Szenarios unterstreicht.

5.2. Umsetzung und Parametrisierung

Um dieses konkrete Szenario umzusetzen, sind unterschiedliche Verhalten für das in Kapitel 4 erläuterte Framework notwendig, die das Szenario abstrahieren. Diese werden in den folgenden Abschnitten kurz erläutert. Hierbei werden nur die für den Sachverhalt nötigen umgesetzt, um die Komplexität möglichst gering zu halten, und die Nachvollziehbarkeit zwecks einer Plausibilisierung der Ergebnisse zu erhöhen. Somit wird zum Beispiel auf das Implementieren eines Anti-Virus-Verhaltens verzichtet, da es sich um eine APT handelt, die keine bekannte

[3] Somit werden FA-22 und FA-23 erfüllt

Signatur aufweist und ein Anti-Virus-Programm keine Wirkung zeigen würde. Da von zwei geschlossenen Firmennetzwerken ausgegangen wird, welche durch ein VPN zu einem Netzwerk verbunden werden, findet eine Betrachtung externer Systeme nicht statt. Das VPN wird dabei durch vermittelnde Knoten realisiert. Hieraus ergibt sich in Bezug auf Abbildung 5.1 das in Abbildung 5.2 gezeigte simulierte Netzwerk, in dem ein Network Attached Storage (NAS) die Rolle des Speichers für das Netzlaufwerk übernimmt.

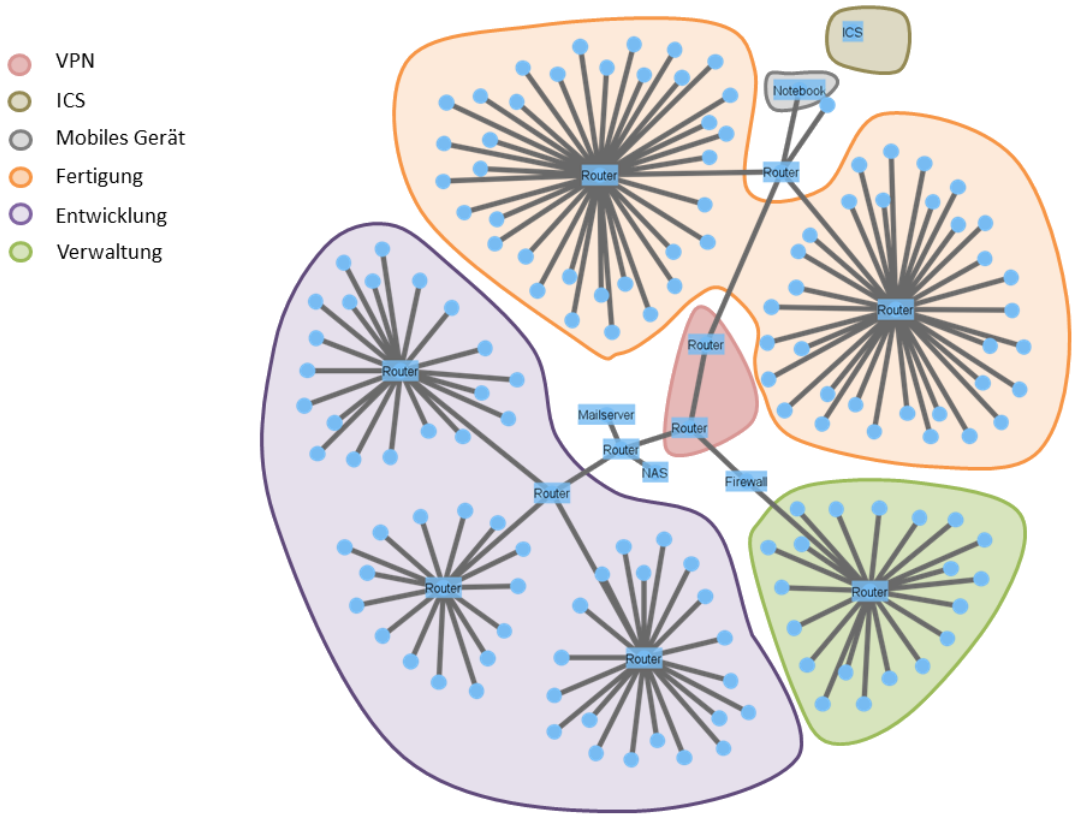


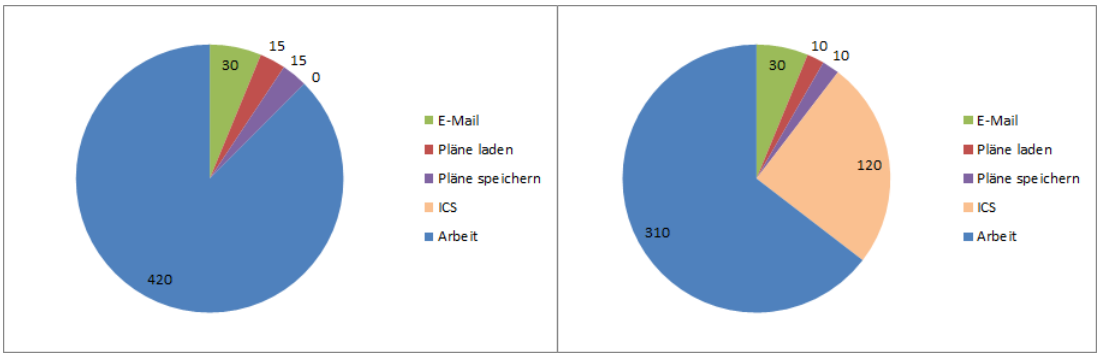
Abbildung 5.2.: Netzgraphvisualisierung in Gephi

Auch wenn es vom Simulationsframework unterstützt wird, so wird keine Unterscheidung zwischen Tag und Nacht sowie Wochenenden getroffen. Vielmehr wird von einer kontinuierlich fortlaufenden Arbeitszeit ausgegangen, in der die beteiligten Systeme und Benutzer Aktionen ausführen.

5.2.1. Benutzer

Für das gewählte Szenario sind unterschiedliche Benutzer nötig, die voneinander abweichendes Verhalten aufweisen. Es wird zwischen normalen Desktop-PC Benutzern, folgend Stan-

dardbenutzer genannt, und mobilen Nutzern unterschieden, die ihre Position, und die des Arbeitsgeräts/Systems, ändern können. Sie werden wie in 4.2.5 erwähnt auf Statemachines abgebildet. Zur Herleitung der verschiedenen Zustände wird hier die verwendete Aufteilung und Parametrisierung beschrieben. Es wird davon ausgegangen, dass die Aufteilung eines Arbeitstages der beiden Benutzergruppen, im Mittel wie in Abbildung 5.3 erfolgt.



(a) Standardbenutzer (b) Mobiler Nutzer

Abbildung 5.3.: Zeitaufteilung von Standard- und Mobil-Nutzer in Minuten

Hierbei entfallen bei beiden Nutzertypen insgesamt 30 Minuten auf das Lesen von E-Mails und jeweils 15 Minuten auf das Öffnen und Speichern von Bauplänen, worunter auch die Suche nach entsprechenden Dateien fällt. Beim Standardbenutzer bestehen 420 Minuten der Arbeitszeit aus sonstigen Arbeiten. Dies variiert beim mobilen Benutzer, der nur 310 Minuten des Arbeitstages mit sonstigen Arbeiten beschäftigt ist, dafür aber im Mittel 120 Minuten seines Arbeitstages mit dem Programmieren der ICS und der damit verbundenen mobilen Tätigkeit verbringt. Es wird von einem achtstündigen Arbeitstag ausgegangen, was eine Gesamtdauer der Tätigkeiten von 480 Minuten ergibt.

Da die einzelnen Tätigkeiten eine gewisse Zeit beanspruchen (siehe Abschnitt 4.1.1), wird diese Dauer hier ebenfalls parametrisiert, wodurch auch die genaue Zusammensetzung der Statemachine und ihre Übergänge definiert werden. Hierbei werden die Übergangswahrscheinlichkeiten der einzelnen Zustände durch die Dauer der Tätigkeiten in Bezug auf ihren Gesamtanteil auf einen achtstündigen Arbeitstag ermittelt. Dies ist den Tabellen 5.2 und 5.3 zu entnehmen. Die Eintrittswahrscheinlichkeit eines Tasks P ergibt sich dabei aus:

$$P_{Task} = \frac{\frac{T_{Task}}{D_{Task}}}{\sum_{i=1}^{Tasks} \frac{T_i}{D_i}} \tag{5.1}$$

Wobei T die Gesamtzeit ist, die für einen Task aufgebracht wird und D die Dauer einer Ausführung des Tasks beschreibt.

	Zeit ges.	Tägl. Anteil	Dauer (Min.)	Task	Häufigkeit pro Tag	Eintritts-wahrscheinlichkeit
E-Mail	30	6,25%		5	6	12%
Pläne laden	15	3,13%		1	15	30%
Pläne speichern	15	3,13%		1	15	30%
ICS	0	0,00%		0	0	0%
Arbeit	420	87,50%		30	14	28%
Gesamt	480	100%		37	50	100%

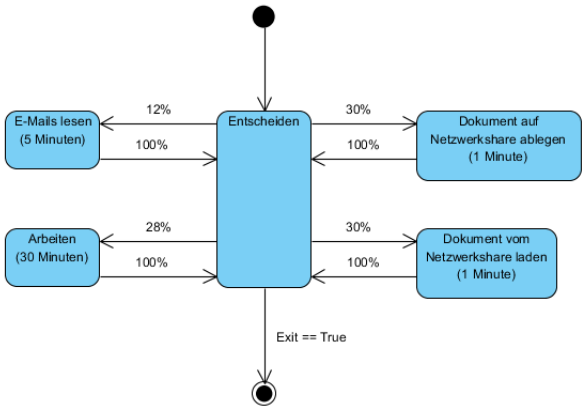
Tabelle 5.2.: Zeitaufteilung und Eintrittswahrscheinlichkeiten für Standardbenutzer

	Zeit ges.	Tägl. Anteil	Dauer (Min.)	Task	Häufigkeit pro Tag	Eintritts-wahrscheinlichkeit
E-Mail	30	6,25%		5	6	15,65%
Pläne laden	10	2,08%		1	10	26,09%
Pläne speichern	10	2,08%		1	10	26,09%
ICS	120	25,00%		60	2	5,22%
Arbeit	310	64,58%		30	10,33	26,96%
Gesamt	480	100,00%		97	38,33	100%

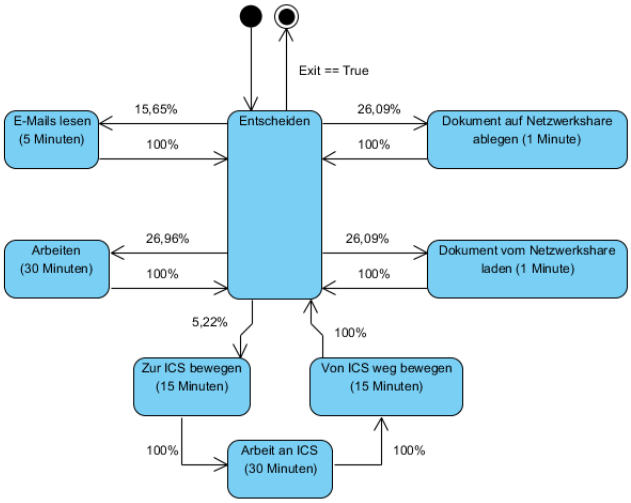
Tabelle 5.3.: Zeitaufteilung und Eintrittswahrscheinlichkeiten für mobilen Nutzer

Die ermittelten Eintrittswahrscheinlichkeiten sind die Voraussetzungen für die Übergänge zwischen den einzelnen Zuständen. Grafisch lassen sich die zwei resultierenden Statemachines wie in Abbildung 5.4 darstellen.

Das Arbeiten an der ICS wird beim mobilen Benutzer in drei Zustände unterteilt, um die Bewegung zur ICS und von dieser wieder weg zu modellieren. Die Gesamtzeit dieser drei Zustände beträgt die in der Tabelle aufgeführten 60 Minuten. Umgesetzt sind die Statemachines - und damit auch die User-Behaviours - als die von SPADE angebotenen Finite State



(a) Standard Benutzer



(b) Mobiler Nutzer

Abbildung 5.4.: Statemachines der Nutzerverhalten

Machine Behaviours. Da die kleinste Einheit der Tätigkeiten eine Minute ist, wird dies auch als Zeitauflösung der Simulation gewählt.

5.2.2. CIFS

Das CIFS-Behaviour, welches für das NAS nötig ist, ist als einfache Liste umgesetzt, die Elemente mit Dateiattributen enthält. Das CIFS-Behaviour akzeptiert dabei Anfragen nach Dateien, welche dann zufällig gewählt werden. Für das Szenario wird angenommen, dass es 200 bestehende Baupläne gibt, welche modifiziert (geladen und gespeichert) werden können. Die individuelle Selektion von Dateien durch die Clients/Benutzer ist nicht vorgesehen, so

dass beim Öffnen oder Speichern von Dateien per Zufall entschieden wird, welches Objekt für die Aktion verwendet wird. Eine Neuanlage ist ebenfals nicht vorgesehen. Somit beträgt die Wahrscheinlichkeit, beim Laden eine bestimmte Datei zu öffnen, $\frac{1}{200}$.

5.2.3. E-Mail

Das Behaviour für den Mail-Server sieht für das Szenario nur ein Empfangen, also Lesen, von E-Mails vor. Es enthält ein Dictionary, welches die Benutzer-ID als Schlüssel und eine Liste von Nachrichten als Value verwendet. Mit der Benutzer-ID, die im Szenario der System-/Knoten-ID entspricht, kann auf die vorliegenden ungelesenen Mails zugegriffen werden.

Da das Szeanrio beim initialen Angriffsvektor von einer Spear-Phishing-Attacke ausgeht, welche durch das gezielte Zustellen einer infizierten Mail geschieht, befindet sich zum Start der Simulation eine infizierte Mail in der Mailbox eines einzelnen Benutzers. Das Öffnen dieser Mail initiiert den Infektionsprozess durch den Aufruf, der im Abschnitt 4.4.5 vorgestellten Funktion mit *exploits = {CVE-2013-2718, CVE-2013-3660}* und der verwendeten simulierten Malware als Parameter. Dieser Aufruf wird vom E-Mail-Behaviour getätigt.

5.2.4. Malware

Zur Implementierung der Malware ist anzumerken, dass sie als abstraktes Konstrukt verstanden werden muss, welches sich zwar an reale Rahmenbedingungen zur Infektion hält, aber keine exakte Kopie einer echten Schadsoftware darstellt. Somit implementiert das Malware-Behaviour zwar seine Ausbreitungsmöglichkeiten, verzichtet aber auf technische Korrektheit.

Da für das Szenario kein Betriebssystem nachgebildet wird, welches normalerweise die Dateioperationen behandelt, manipuliert die simulierte Malware mittels Monkey Patching den Teil des Benutzerverhaltens, der für das Speichern von Dateien zuständig ist, so dass im Falle eines Befalls schadhafte Dateien auf dem Netzwerkspeicher abgelegt werden. Hierbei wird durch einen internen Zähler berücksichtigt, dass sich die Malware maximal drei mal reproduziert. Da kein Betriebssystem simuliert wird, ist eine Nachbildung des DLL-Hijackings durch die Malware notwendig. Dies geschieht beim Netzwerkspeicher. Dieser würde in der Realität nicht manipuliert werden, durch das Fehlen eines komplexen Betriebssystem-Layers wird hier aber so verfahren. Die Malware greift hier in die Abfrageroutine für Dateien ein. Wird auf einen infizierten Ordner zugegriffen, sorgt das veränderte NAS-Behaviour für einen entsprechenden Infektionsversuch beim zugreifenden System. Ist dieses unter Anwendung der Exploits CVE-2013-2718 und CVE-2013-5241 verwundbar, wird das System erfolgreich infiziert.

6. Ergebnisse

Das in Kapitel 5 vorgestellte Fallbeispiel ist zum Ermitteln der Ergebnisse mehrfach - konkret sieben mal - simuliert worden. Die Simulationsergebnisse werden nun präsentiert und auf ihre Validität hin überprüft, um sie anschließend in Kapitel 7 zu diskutieren. Die nötigen Kennzahlen werden hierfür aus den Snapshots, die während der Simulation persistiert werden, gewonnen. Die Snapshots des Systemzustandes sind in JSON-Dokumenten innerhalb von CouchDB hinterlegt, welche den kompletten Graphen mit den Eigenschaften der Knoten, des Timeslices, etc. enthalten (siehe Listing 6.1).

```
{
   "graph": [
       [
           "timeslice",
           0
       ]
   ],
   "nodes": [
       {
           "infected": 1,
           "infection_route": [7,6],
            ...
           "id": "133"
       },
        ...
   ],
   "links": [
       {
           "source": 137,
           "target": 138
       }
        ...
   ],
   "multigraph": false
}
```

Listing 6.1: Auszug aus Graph-Dokument in JSON

Zum Selektieren der Ergebnisse werden zwei Views verwendet, die hier kurz vorgestellt werden, um die Resultate nachvollziehen zu können. Die View aus Listing 6.2 wird verwendet, um direkt auf Dokumente über die Timeslice-ID zugreifen zu können. Sie wird benötigt, um zu einem bestimmten Zeitpunkt den Infektionsgraphen bestimmen zu können, indem die nötigen Informationen aus einem Dokument extrahiert werden.

```
1 //map function
2 function(doc) {
3       if(doc.graph[0][1])
4           emit(doc.graph[0][1], doc);
5 }
```

Listing 6.2: View mit Timeslice-ID als Key zum direkten Zugriff nach Timeslice

Listing 6.3 zeigt eine weitere View, die alle zu einem bestimmten Zeitpunkt, welcher durch die Timeslice-ID identifiziert wird, infizierten Knoten innerhalb des Netzgraphen aufsummiert. Dies geschieht durch die gezeigte Reduce-Funktion, die über alle Knoten iteriert und im Falle einer Infektion den Zähler inkrementiert. Somit kann ermittelt werden, wie viele Systeme zu einem gegebenen Messzeitpunkt infiziert sind.

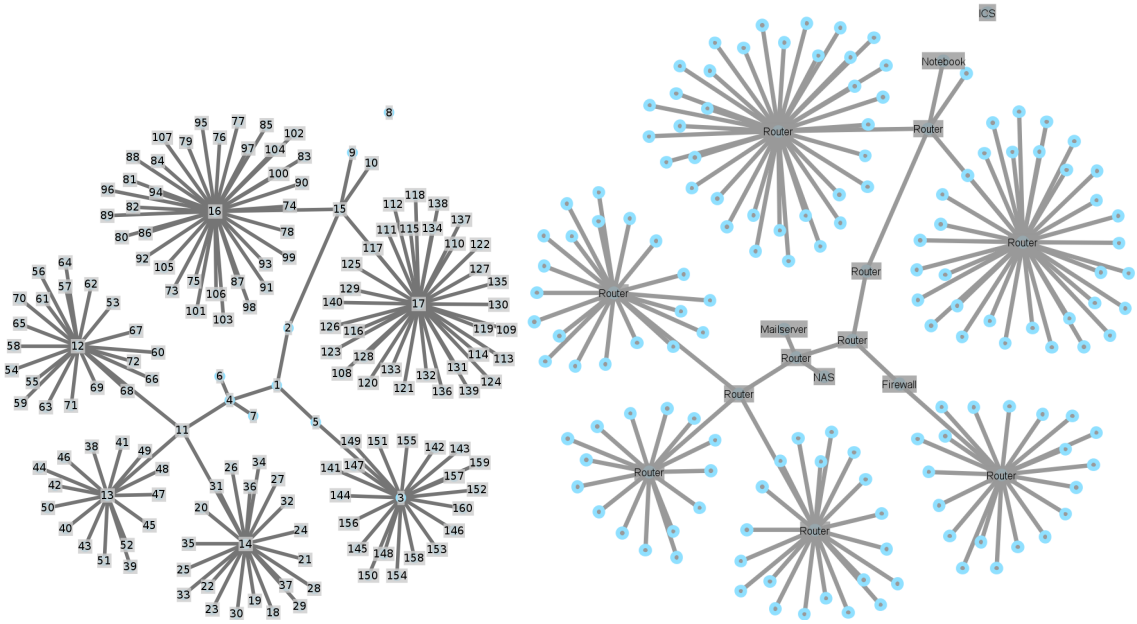
```
1  //map function
2  function(doc) {
3        emit([doc.graph[0][1], doc.graph[1][1], doc.nodes);
4  }
5
6
7  //reduce function
8  function(keys, values) {
9    var sum = 0;
10   values[0].forEach(function(node) {
11         if(node.infected == 1)
12                 sum += 1;
13   });
14   return sum;
15 }
```

Listing 6.3: View zum Zählen aller infizierten Knoten

Wichtig ist, dass bei der Betrachtung der Daten schon dann von einer Infektion ausgegangen wird, wenn der betroffene Netzknoten weitere Infektionen hervorrufen kann. Auch wenn z.B. in der Fallstudie das NAS technisch nicht infiziert ist, so hält es unter Umständen dennoch schadhafte Daten vor, die weitere Systeme befallen können und gilt daher in den Auswertungen und Visualisierungen als infiziert.

6.1. Präsentation

Wie eingangs in Abschnitt 1.2 erwähnt, sollen die Simulationsergebnisse dafür verwendet werden, um herauszufinden, ob topologische Eigenschaften des untersuchten Netzes die Ausbreitung von Schadsoftware maßgeblich beeinflussen, oder ob andere Kriterien ein höheres Gewicht haben. Hierfür werden die im Grundlagenkapitel (2.2.2) vorgestellten Graphenmetriken des simulierten Netzes mit der gemessenen Ausbreitung der Schadsoftware verglichen. Um dies zu erleichtern, haben alle Netzknoten eine eindeutige Kennziffer, deren Position innerhalb des Netzes der Abbildung 6.1 entnommen werden kann.



(a) Knoten und IDs (b) Übersicht

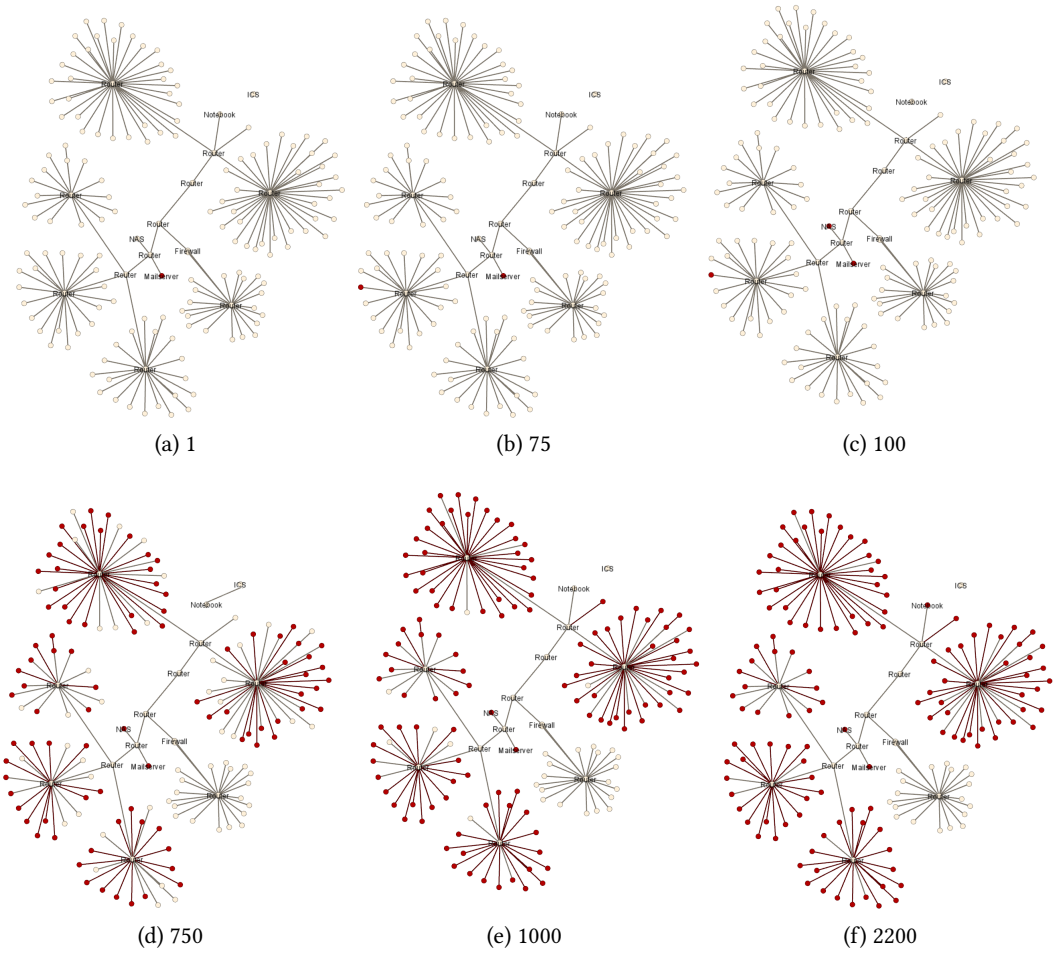
Abbildung 6.1.: ID-Zuordnung zu Netzknoten und Übersicht

Dabei ist der Benutzer, der das System mit der ID 70 bedient, das Opfer der erwähnten Spear-Phishing Attacke.

6.1.1. Infektionsverlauf

Spatialer Infektionsverlauf

Die Identität des Spear-Phishing Opfers wird auch bei der Visualisierung eines Simulationslaufes sichtbar. Dieser ist exemplarisch in Darstellung 6.2 illustriert, um ein grundlegendes Verständnis für die Ausbreitung der Malware innerhalb eines Simulationslaufes.



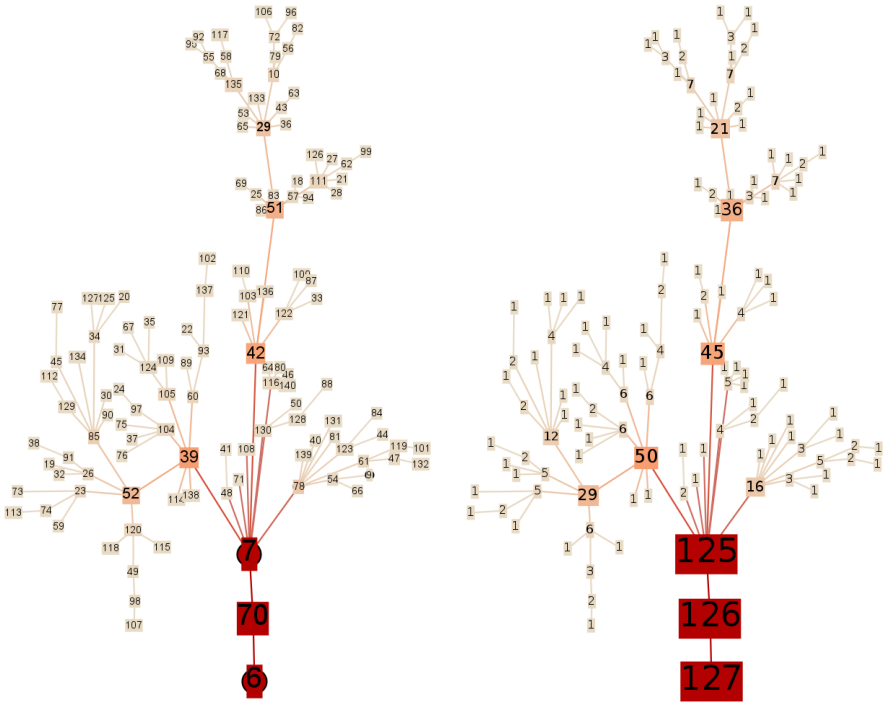
(a) 1 (b) 75 (c) 100

(d) 750 (e) 1000 (f) 2200

Abbildung 6.2.: Visualisierter Infektionsverlauf (rot) innerhalb einer Simulation zu unterschiedlichen Zeitpunkten (Angabe in Simulationsschritten)

Hierbei ist zu erkennen, dass der mobile Knoten (ID 9), welcher zum Programmieren der ICS verwendet wird, bestehende Kanten lösen und neue herstellen kann, um Mobilität zu simulieren. Dieser Knoten wird in der Darstellung erst in der fortgeschrittenen Simulation infiziert. Dieses Verhalten konnte in sechs von sieben Fällen beobachtet werden. Somit kann

aus den einzelnen Ergebnissen abgeleitet werden, dass das ICS eines der letzten Systeme ist, die befallen werden. Außerdem ist zu erkennen, dass der Mailserver (ID 6) zum initialen Zeitpunkt die infizierte Nachricht vorhält und damit als infizierter Knoten gezählt wird. Auffällig ist, dass Systeme eines ganzen Teilnetzes bis zum Ende der Simulation nicht infiziert werden. Dies liegt an der Firewall, die ein Nutzen des Netzwerkspeichers, welcher als Hauptausbreitungsvektor verwendet wird, unterbindet. Somit kommt es zu keinem Befall dieses Teilnetzes.



(a) Infektionsgraph mit Knoten-ID

(b) Vorkommen auf Infektionspfaden; $\widehat{=}$ *Anzahl von Infektionen ausgehend von Knoten* $+1$

Abbildung 6.3.: Infektionsgraph und Infektionshäufigkeiten

Neben der Übersicht des Infektionsverlaufes innerhalb des Netzes, kann aus den gesammelten Daten auch rekonstruiert werden, welche Systeme ein befallener Knoten angesteckt hat. Durch eine Aggregation dieser Informationen kann auch bestimmt werden, wie viele Folgeinfektionen ein Knoten ausgelöst hat. Zur grafischen Darstellung dieser Daten dienen Infektionsgraphen, die in Abbildung 6.3 zu sehen sind. Die Grafik zeigt einen exemplarischen Graphen eines Simulationslaufes; gravierende Unterschiede zu anderen Ausführungen konnten nicht gefunden werden. Außerdem sind Zyklen, die durch Reinfektion entstehen, entfernt worden. Erkennbar

ist eindeutig, dass es eine initiale Infektion gab, die für das weitere Propagieren verantwortlich ist.

Eine andere Darstellung der Folgeinfektionen zeigt Abbildung 6.4. Diese verzichtet auf die Visualisierung des Wegs der Schadsoftwareausbreitung, zeigt aber, welcher Knoten innerhalb des Netzgraphen mehr oder weniger an der Ausbreitung beteiligt war und kennzeichnet somit die Hauptinfektionsherde.

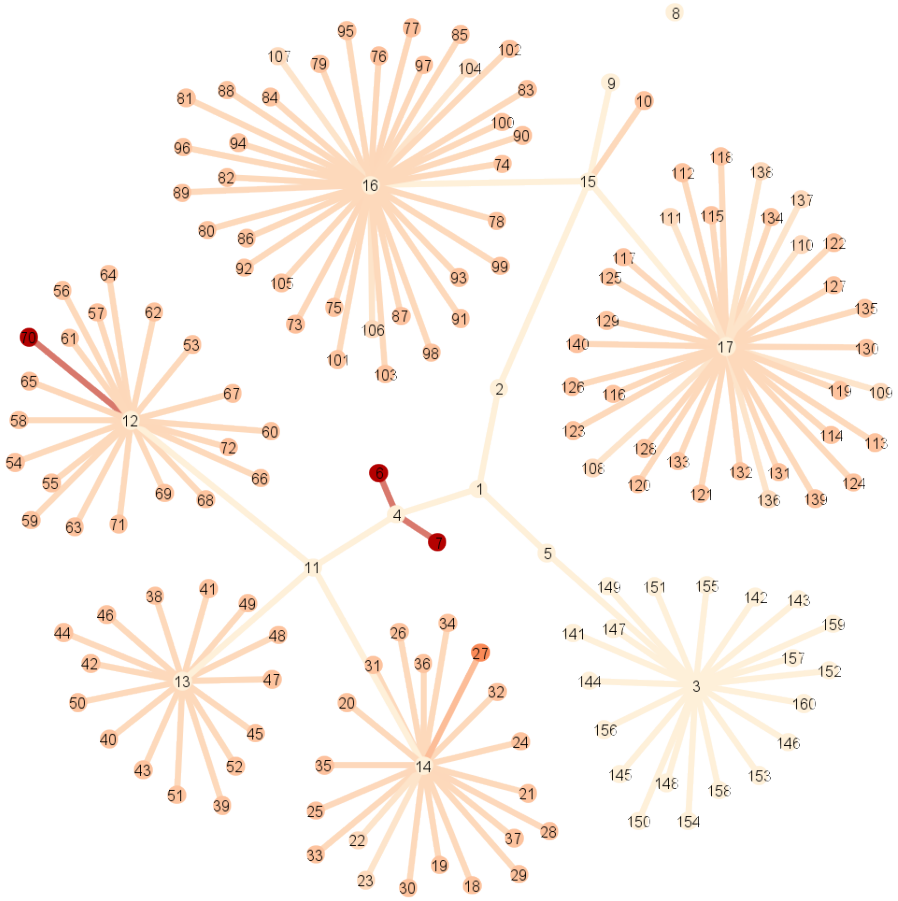
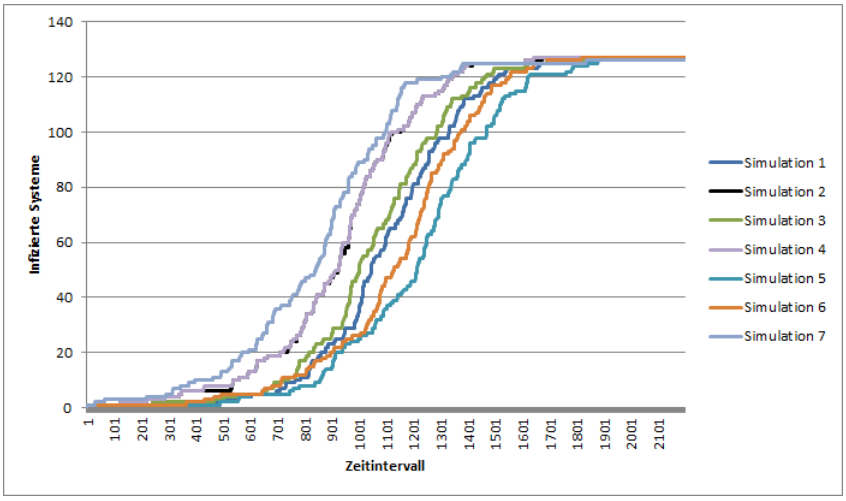


Abbildung 6.4.: Infektionsherde

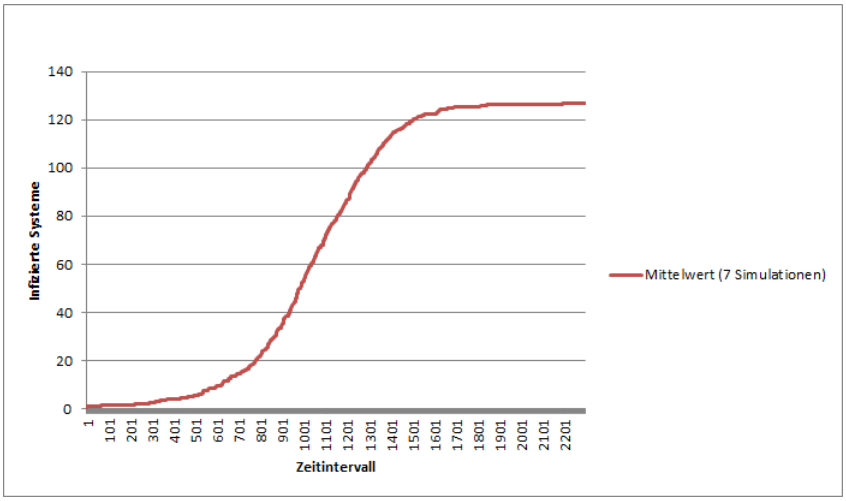
Temporaler Infektionsverlauf

Eine Zeitmessung des Infektionsverlaufs, der in Abbildung 6.5 grafisch dargestellt ist, ergibt im Mittel, dass die Höchstzahl an Infektionen nach 30 Stunden und 46 Minuten (Arbeitszeit)

erreicht wird. Somit wird innerhalb dieses Zeitraums im Mittel ca. alle 14,5 Minuten ein System infiziert.



(a) Infektionsverläufe



(b) Gemittelter Infektionsfverlauf

Abbildung 6.5.: Infektionsverläufe

6.1.2. Zentralitäten

Zur Feststellung, ob die Ausbreitung in Korrelation mit einer der in Abschnitt 2.2.2 vorgestellten Graphen-Metriken steht, sind die Messwerte zu diesen ermittelt und grafisch aufbereitet worden. Hierzu wurde Gephi verwendet. Dabei sind die Werte normiert und mit einer vorgegebenen Farbskala abgeglichen worden, so dass ein visueller Vergleich möglich ist. Die

resultierenden Graphen sind folgend aufgeführt, wobei ein hellerer Farbton einen niedrigen Wert angibt und ein dunkler einen hohen Messwert.

Grad-Zentralität

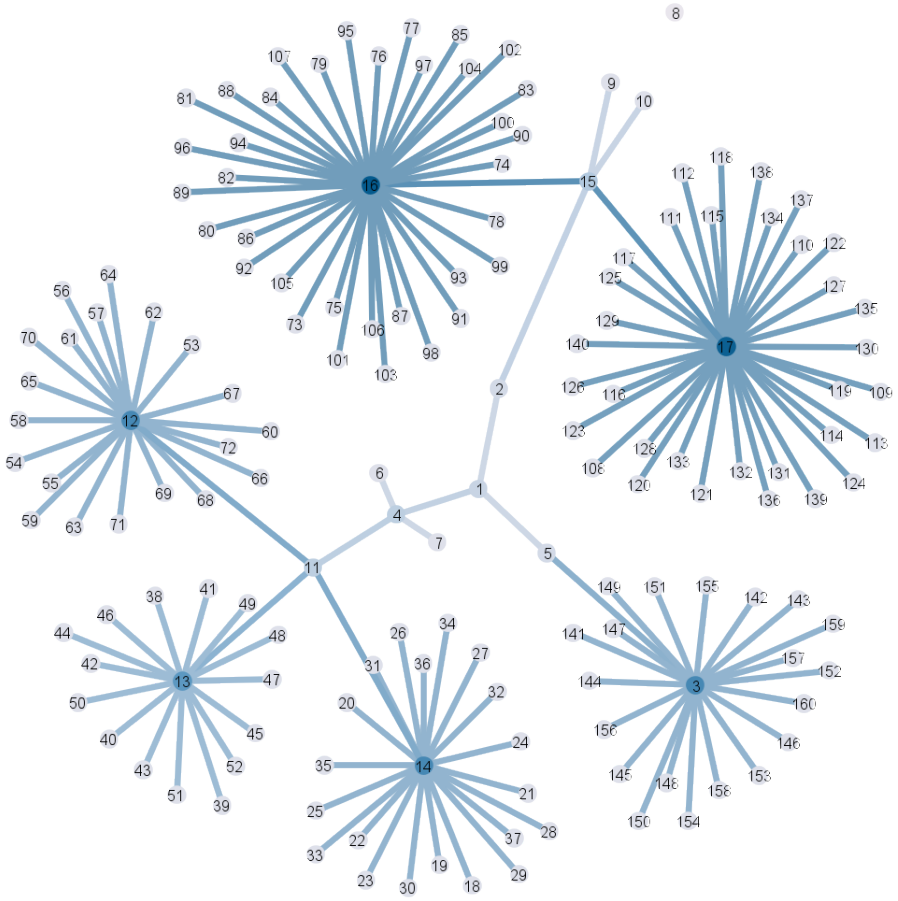


Abbildung 6.6.: Grad-Zentralität des simulierten Netzes

Der gefärbte Graph in Abbildung 6.6 zeigt die ermittelte Grad-Zentralität der Knoten des simulierten Netzes. Klar zu erkennen ist, dass die Router der Teilnetze als Konzentrationsknoten eine hohe Grad-Zentralität aufweisen, da sie viele Kanten zu anderen Knoten aufweisen. Auf ihre benachbarten Knoten hat dies keinen Einfluss.

Eigenvektor-Zentralität

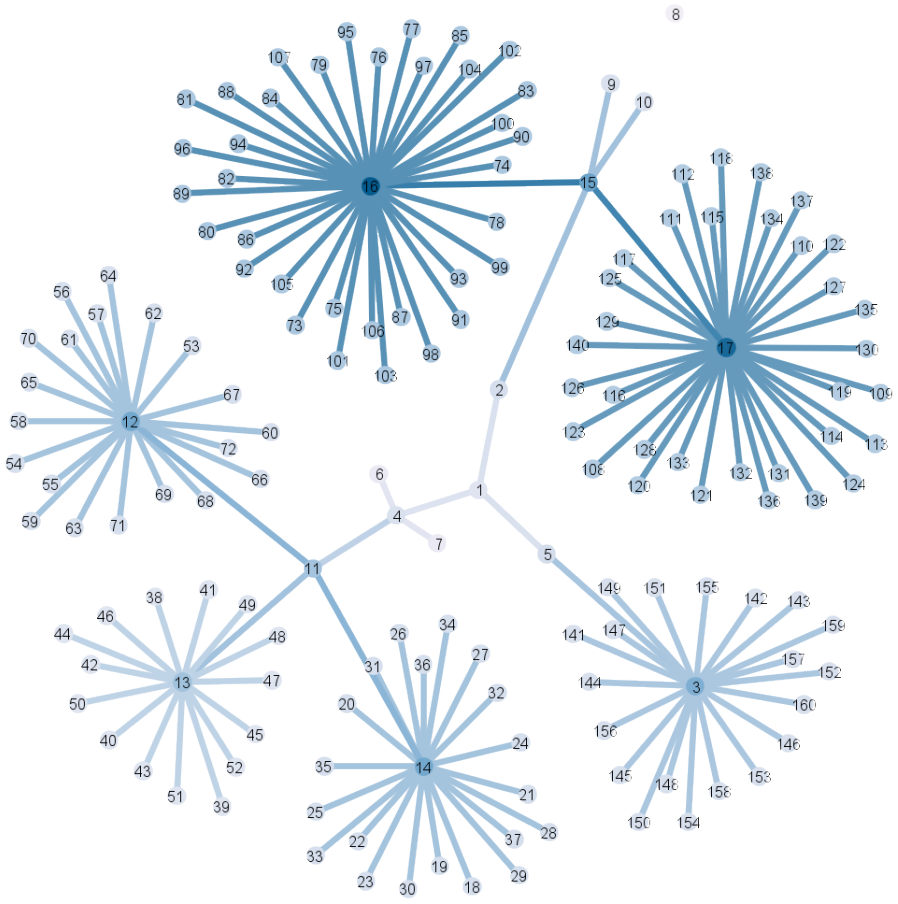


Abbildung 6.7.: Eigenvektor-Zentralität des simulierten Netzes

Die Eigenvektor-Zentralität[1], welche in Abbildung 6.7 dargestellt wird, zeigt, dass im vorliegenden Netz die Knoten, welche viele Verbindungen konzentrieren, wichtig sind. Diese Relevanz setzt sich in den folgenden Knoten, welche diese Konzentrationspunkte mit dem restlichen Netz verbinden, fort. Auch die Relevanz der mit dem konzentrierenden Knoten verbundenen Endsysteme steigt.

[1] ermittelt mit Gephi; Standardparameter verwendet: 100 Iterationen; gleiche Startrelevanz

Betweeness-Zentralität

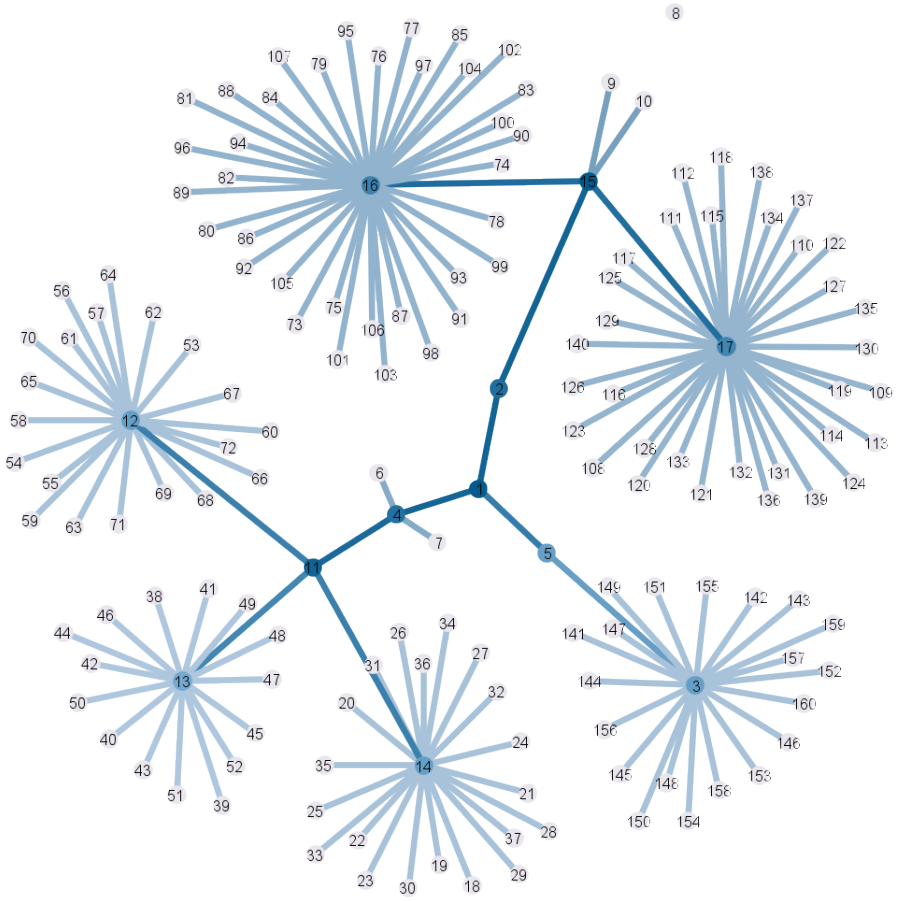


Abbildung 6.8.: Betweenness-Zentralität des simulierten Netzes

Die Betweenness-Zentralität (Abb. 6.8), welche die Knoten hervorhebt, die am häufigsten auf dem kürzesten Pfad zwischen zwei Knoten liegen, zeigt, dass im Wesentlichen die Knoten, die die Teilnetze miteinander verbinden, in diesem Kontext eine starke Bedeutung haben. Hierzu gehören auch die Knoten, welche die VPN-Gateways darstellen und wahrscheinlich auch erhöhten Kommunikationsverkehr zu erwarten haben.

6.1.3. Netzauslastung

Neben den bereits vorgestellten Messergebnissen sind zusätzlich noch Daten erhoben worden, die etwas über die Nutzung des Netzes aussagen. So zeigt Abbildung 6.9, welche Knoten

innerhalb einer Simulation viele Nachrichten routen mussten. Erkennbar wird, dass dies - wie zu erwarten ist - tatsächlich auch die Router sind. Die Betrachtung dieser Messwerte erlaubt eine Einschätzung über die tatsächliche Wichtigkeit - im Bezug auf Kommunikation - eines Knotens innerhalb des Netzes.

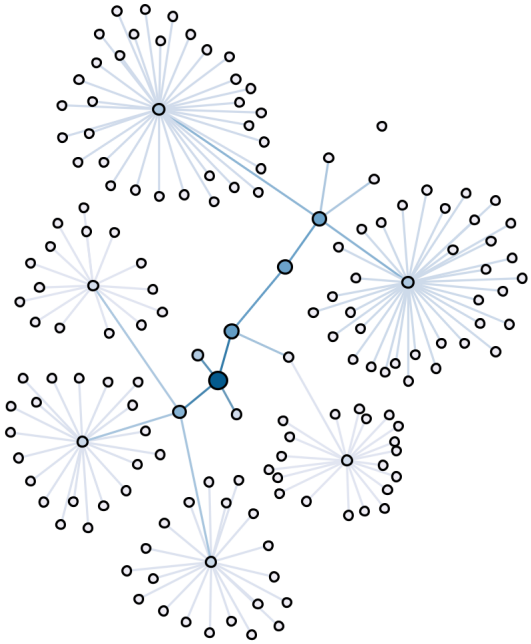


Abbildung 6.9.: Kommunikationsstärke von Knoten

Der Graph in Abbildung 6.10 zeigt die Anzahl an Anfragen, die ein Knoten während der Simulation erhalten hat. Hiermit sind Dienstaufrufe oder auch Antworten darauf gemeint.

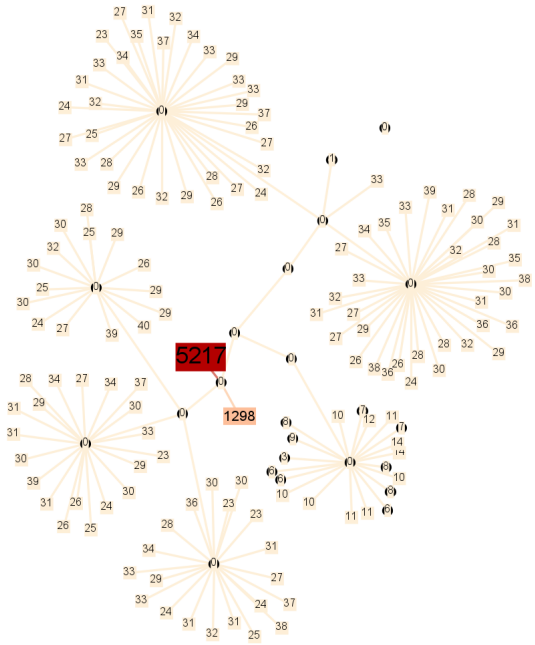


Abbildung 6.10.: Gezählte Aufrufe von Knotendiensten

Zu sehen ist, dass Router, die keine Dienste anbieten, auch keine Aufrufe erhalten. Endsysteme hingegen erhalten mehr Aufrufe, da diese von direkten Antworten ausgelöst werden. Am häufigsten werden Knoten, welche zentrale Dienste anbieten, angefragt.

6.2. Plausibilisierung

Eine Verifizierung der Simulationsergebnisse ist aufgrund der Komplexität nicht möglich. Daher wird hier überprüft, ob sie plausibel sind. Hierfür werden zwei Referenzen zum Vergleich herangezogen, um die Faktoren Ausbreitungsstruktur und Ausbreitungsverlauf, welche für die Ergebnisbewertung substantiell sind, zu vergleichen und auf ihre Plausibilität zu prüfen.

Da für die Fallstudie eine Malware verwendet wurde, die ähnliche Ausbreitungswege wie Stuxnet nutzt, und ähnliche Einschränkungen in der Replikationshäufigkeit unterliegt, liegt ein Vergleich mit vorliegenden Daten zu Stuxnet nahe. In einem Dossier der Symantec Corporation (2011) über Stuxnet wird neben technischen Details zur eigentlichen Schadsoftware auch die Verbreitung der Malware behandelt. Unter anderem wird auch auf die spatialen Eigenschaften der Ausbreitung eingegangen. Hier wird festgestellt, dass der Ausbreitungsgraph, welcher den Weg der Infektion beschreibt, primär aus linearen Ästen besteht und kaum Verzweigungen

aufweist (siehe Abbildung 6.11). Dies wird zum einen auf die limitierenden Ausbreitungsfaktoren zurückgeführt (ein Device repliziert die Malware maximal drei mal). Zum anderen wird davon ausgegangen, dass die Anzahl der zur Untersuchung wiederhergestellten Systeme zu beschränkt war. Symantec geht davon aus, dass bei einer größeren Anzahl untersuchter Systeme mehr Verzweigungen erkennbar gewesen wären.

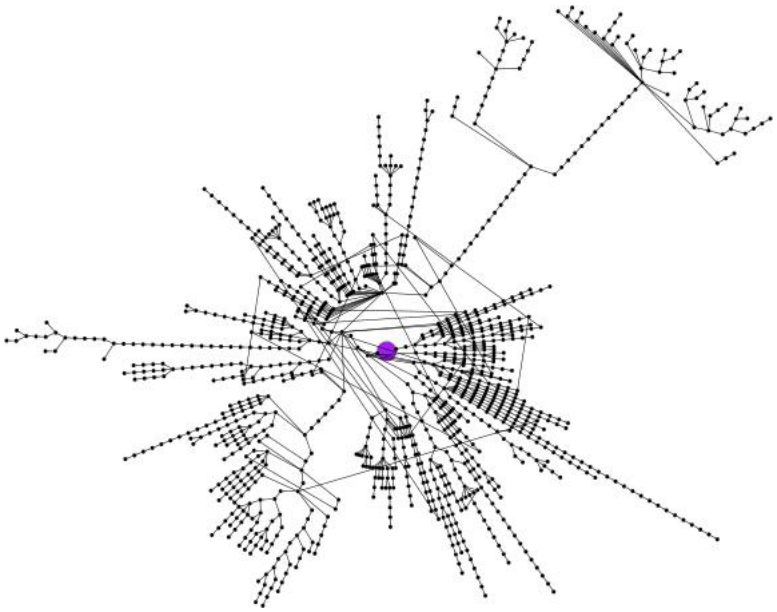


Abbildung 6.11.: Infektionsgraph Stuxnet, eine Domäne (aus Symantec Corporation (2011))

Ein Vergleich dieser Ergebnisse mit denen, die aus der durchgeführten Simulation gewonnen wurden, kann Aufschluss über einen plausiblen Infektionsweg geben. Hierzu wurden Messdaten erfasst, aus denen ein Ausbreitungsgraph generiert werden kann. Da eine Mittelung über die sieben durchgeführten Simulationen beim Erstellen eines Graphen nicht möglich ist, wird hier exemplarisch nur auf einen eingegangen[2]. Der in Abbildung 6.12 gezeigte Graph ist das Resultat eines Simulationslaufs, wobei Zyklen entfernt wurden. Er weist ähnliche Charakteristika auf, wie der von Symantec Corporation (2011) beschriebene Graph der Stuxnet-Ausbreitung. Zu erkennen sind im Kern lineare Äste, die wenige Abzweigungen aufweisen. Unter Anbetracht der ähnlichen Eigenschaften der Ausbreitungsvektoren zu Stuxnet, scheint die Simulation der modellierten Schadsoftware plausibel zu sein.

[2]Eine relevante Abweichung zu den Ergebnissen anderer Simulationsläufe konnte nicht festgestellt werden.

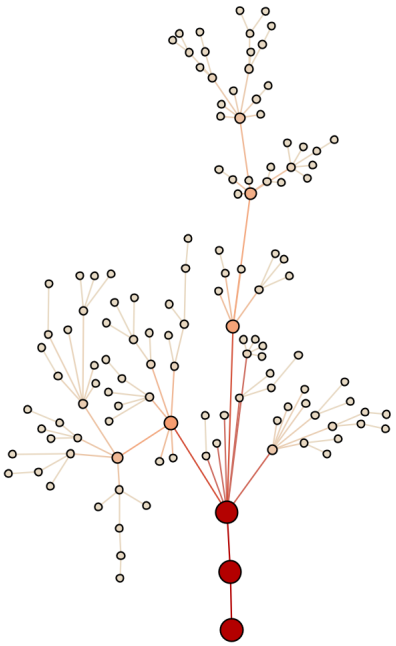


Abbildung 6.12.: Infektionsgraph aus Simulationsergebnissen

Um die Ausbreitungsgeschwindigkeit zu überprüfen, wird das SI-Modell herangezogen. Dies entspricht dem schon im Abschnitt 2.1.5 vorgestellten SIS-Modell, mit einer null prozentigen Desinfektionswahrscheinlichkeit, von der auch das Szenario ausgeht. Als Parameter für das SI-Modell wird dabei von folgenden Werten ausgegangen:

$$
\begin{aligned}
N &= 160 \\
S_0 &= 159 \\
I_0 &= 1 \\
K &= 0,0313 * 0,0625 * \frac{3}{200} \\
q &= 0,925 \\
\alpha &= K * q
\end{aligned}
$$

N ist die Summe aller beteiligten Netzknoten. S_0 ist die Anzahl der nicht infizierten Netzknoten zum Zeitpunkt 0; I_0 die Anzahl der infizierten Systeme. Da das Szenario eine Spear-Phishing Attacke vorsieht, die genau auf einen Benutzer zum Ziel hat, ergibt sich der Wert 1 für den Zeitpunkt 0. Die Kontaktwahrscheinlichkeit K setzt sich aus für die Infektion relevanten Teile

des Tagesablaufs der Standardbenutzer, die über 99% der Nutzer ausmachen, zusammen. Hierzu gehört die 3,13%ige Wahrscheinlichkeit E-Mails abzurufen und die 6,25%ige Wahrscheinlichkeit eine Datei vom Netzwerkshare zu öffnen. Da sich aber 200 Dateien auf dem Netzwerkspeicher befinden und ein infizierter Knoten seine Schadsoftware auf dem Netzwerkspeicher nur drei mal reproduziert, verringert sich die Kontaktwahrscheinlichkeit um den Faktor $\frac{3}{200}$. Der Kontagionsindex q wird mit 0,925 angenommen, da die verwendete Systemkonfiguration auf allen Geräten identisch ist und eine generelle Anfälligkeit durch die APT gegeben ist. Dies bedeutet, dass, wenn es zu einem Kontakt kommt, auch eine Infektion stattfindet. Da dies aber für die Router und Firewalls nicht der Fall ist, werden diese hier ausgenommen. Somit ergibt sich der Kontagionsindex von $\frac{148}{160}$.

In Abbildung 6.13 ist der gemittelte Infektionsverlauf der durchgeführten Simulationen dem, der mit dem SI-Modell ermittelt wurde, gegenübergestellt.

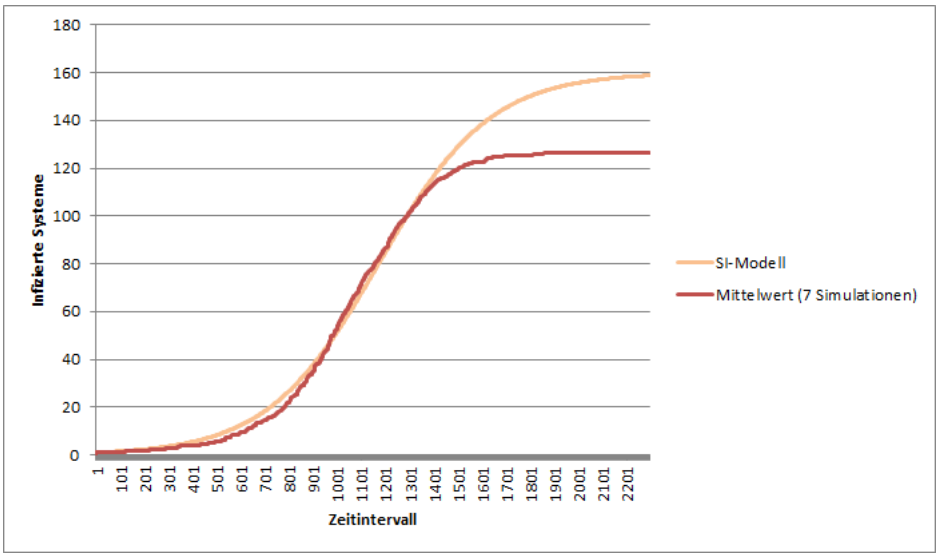


Abbildung 6.13.: Gegenüberstellung des Infektionsverlaufs nach SI-Modell mit vorgestelltem Modell

Zu erkennen ist, dass beide Modelle einen ähnlichen Kurvenverlauf aufweisen und somit die Ausbreitungsgeschwindigkeit bei beiden Modellen nur geringfügig voneinander abweicht. Eine vollständige Durchdringung ist der Schadsoftware im SI-Modell nach 4,8 Arbeitstagen[3] gelungen, was 38 Stunden und 20 Minuten entspricht. Im vorgestellten Modell ist dies im Mittel bereits nach 3,85 Arbeitstagen der Fall, was 30 Stunden und 46 Minuten entspricht. Auffällig ist, dass beim SI-Modell offensichtlich mehr Systeme infiziert werden, als beim

[3] ausgehend von 8 Stunden pro Arbeitstag

hier vorgestellten Modell. Dies liegt daran, dass das SI-Modell von einer homogenen Menge an Systemen ausgeht und somit individuelle System- oder Infrastruktureigenschaften nicht berücksichtigt. Im vorgestellten Szenario wird beim Agenten-System berücksichtigt, dass die Firewall ein Benutzen des Netzwerkspeichers für eine Menge an Systemen unterbindet und somit Infektionen verhindert. Dadurch lassen sich die Ergebnisse beider Modelle nicht einfach vergleichen. Dennoch erhält man eine Einschätzung über die Zuverlässigkeit der Ergebnisse.

Da ein Vergleich der Ausbreitungsgraphen von Stuxnet und der simulierten Malware ähnliche Charakteristika aufweisen, und dies ebenfalls für den zeitlichen Ausbreitungsverlauf im Vergleich mit dem SI-Modell gilt, kann davon ausgegangen werden, dass sowohl die spatialen als auch die temporalen Eigenschaften des vorgestellten Modells plausibel sind, und als Grundlage für weitere Aussagen dienen können.

7. Diskussion

7.1. Implementierung

Die Implementierung des Simulationsframeworks zeigt, dass eine Multi-Agenten-Simulation im Kontext der Schadsoftwareausbreitung machbar ist. Durch den gewählten Ansatz, eine hohe Modularität zu unterstützen, lassen sich grundlegende Bestandteile für unterschiedliche Szenarien weiter verwenden. Die Idee, bestehende Netze im Rahmen des Information Gatherings zu kartographieren und für eine Simulation zu virtualisieren, hat bei der durchgeführten Fallstudie keine Anwendung gefunden, wird aber dennoch als sinnvoll erachtet. Durch die Implementierung und die folgende Anwendung dieser, konnten sowohl technische als auch fachliche Einschränkungen beobachtet werden, welche folgend genauer aufgezeigt werden.

7.1.1. Technische Einschränkungen

Die festgestellten technischen Einschränkungen betreffen hauptsächlich das Implementieren von Tests. Durch die Verteilung des Systems und das Baukastenprinzip, welches den modularen Aufbau von Agenten ermöglicht, sind Unit-Tests nicht vernünftig umsetzbar. Auch SPADE liefert hier keine adäquate Lösung, die ein Testen der verteilten SPADE-Agenten maßgeblich erleichtern würde. Daher musste auf das Implementieren von Tests verzichtet werden, was einen erheblichen Mehraufwand bei der Weiterentwicklung bedeutet. Weiterhin ist das Auffinden von Fehlern durch das Baukastenprinzip relativ mühsam. Bei der Zusammenstellungen oder auch der Entwicklung von Behaviours muss darauf geachtet werden, dass keine Wechselwirkungen während der Laufzeit auftreten können.

Auch wenn keine ausgiebigen Performancetests durchgeführt wurden, so konnte doch festgestellt werden, dass der von SPADE mitgelieferte XMPP-Server unabhängig von der verwendeten Hardware bei ungefähr 500 angemeldeten Agenten merklich langsamer wird. Somit wird der Nachrichtenaustausch verzögert und die Simulationszeit erhöht. Möchte man also größere Netzwerke simulieren ist der Einsatz eines anderen XMPP-Servers unverzichtbar.

7.1.2. Fachliche Einschränkungen

Das Modellieren komplexer Szenarien stellt einen nicht zu unterschätzenden Aufwand dar. Dies gilt sowohl im Falle eines manuellen Aufbaus als auch im Falle eines vorhergehenden Information Gatherings. Im letzteren Fall müssen unter Umständen erhebliche Nacharbeiten durchgeführt werden. Mindestens eine Verteilung entsprechender Benutzerverhalten an Endsysteme muss durchgeführt werden, da diese im Information Gathering Prozess nicht erfasst werden können. Darüber hinaus ist der Aufbau eines Szenarios durch die Implementierung der einzelnen Behaviours beschränkt. Für die in Kapitel 5 durchgeführte Fallstudie wurde zum Beispiel explizit auf die Erstellung eines Betriebssystems-Behaviours verzichtet. Ist eine genaue Betrachtung dieser Ebene allerdings erwünscht, müssen neben den fachlichen Randbedingungen auch die technischen angepasst werden. Dies hat zur Folge, dass für weitere Experimente wahrscheinlich häufig die Behaviours neu erstellt oder angepasst werden müssen.

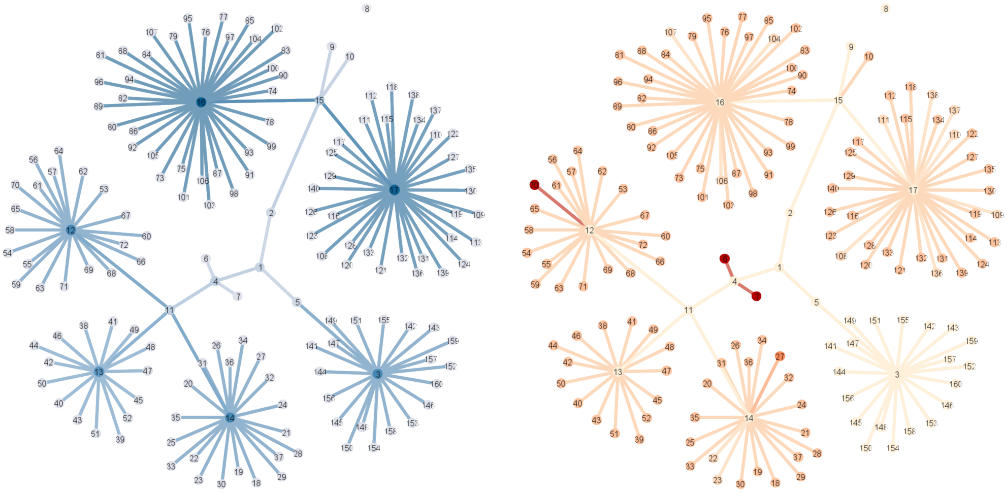
7.2. Ergebnisse

7.2.1. Topologische Einflüsse

Um Gemeinsamkeiten oder auch Unterschiede zwischen der Ausbreitungsbeteiligung und gemessenen Graphenmetriken im vorgestellten Szenario feststellen zu können, werden diese hier visuell miteinander verglichen. Dazu werden die aus Kapitel 6 schon bekannten Darstellungen verwendet und gegenübergestellt.

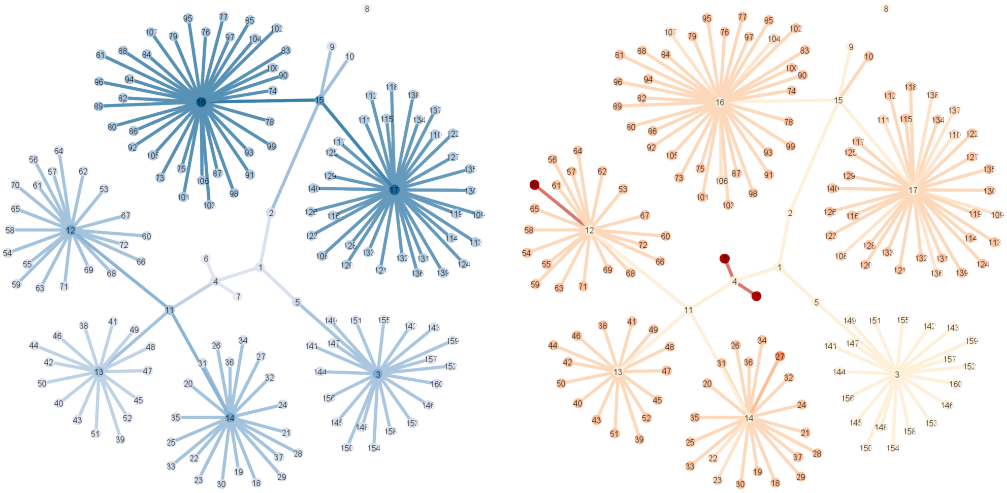
Bei einem Vergleich der Grad-Zentralität mit den Infektionsherden (Abb. 7.1) ist ersichtlich, dass keine Gemeinsamkeiten erkennbar sind. Auffällig ist, dass Knoten mit hoher Grad-Zentralität, dies sind hier vorwiegend Router, keine Infektionen ausgelöst haben. Dies liegt daran, dass sich die Schadsoftware in der konkreten Fallstudie über Endsysteme und Lücken in der Applikationssicherheit ausbreitet.

Schädlinge, die sich auf Ausbreitung über Infrastrukturknoten (z.B. Router) spezialisieren, könnten hier eine andere Wirkung haben. Ein Beispiel für solch eine Malware ist „TheMoon“, welche sich selbstständig auf bestimmte Linksys Router und W-LAN Access Points ausbreitet und von dort weiter propagiert (Linksys, 2014). Eine generelle Aussage über die Ausbreitung lässt sich somit von der Grad-Zentralität nicht ableiten.



(a) Grad-Zentralität des simulierten Netzes (b) Infektionsherde (dunkler = mehr Infektionen)

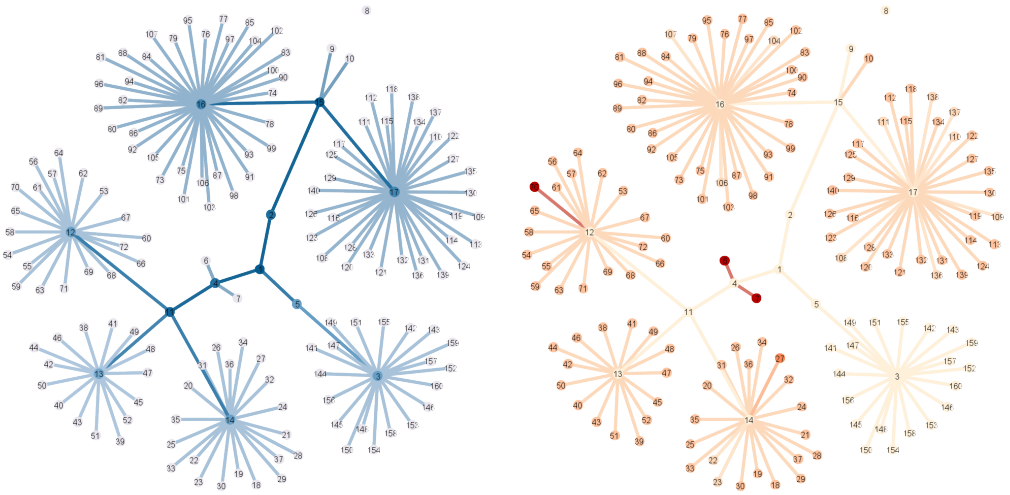
Abbildung 7.1.: Grad-Zentralität verglichen mit Infektionsherden



(a) Eigenvektor-Zentralität des simulierten Netzes (b) Infektionsherde (dunkler = mehr Infektionen)

Abbildung 7.2.: Eigenvektor-Zentralität verglichen mit Infektionsherden

Die ermittelte Eigenvektor-Zentralität, die nicht wesentlich von der Grad-Zentralität abweicht, und die Bedeutung der Nachbarn von - im Sinne der Eigenvektor-Zentralität - wichtigen Knoten verdeutlicht, weist im direkten Vergleich (Abb. 7.2) mit den Ausbreitungsherden ebenfalls keine Parallelen auf. Somit ist auch dieses Maß nicht dazu geeignet, generelle Aussagen zur Schadsoftwareausbreitung in Bezug auf die Netztopologie zu machen.



(a) Betweenness-Zentralität des simulierten Netzes (b) Infektionsherde (dunkler = mehr Infektionen)

Abbildung 7.3.: Betweenness-Zentralität verglichen mit Infektionsherden

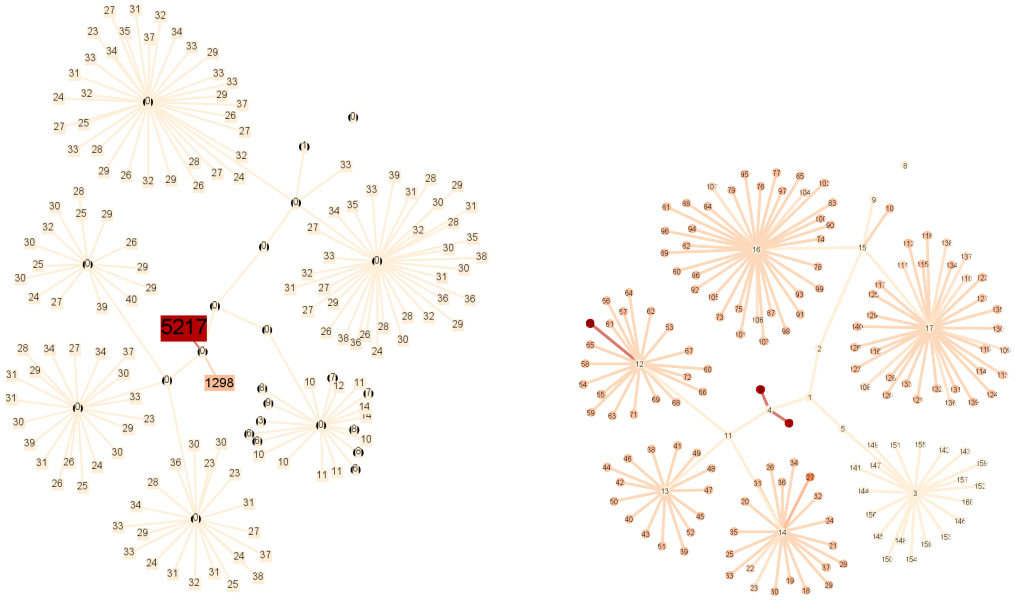
Die Betweeness-Zentralität, welche die Knoten hervorhebt, welche sich am häufigsten auf den kürzesten Wegen zwischen zwei Knoten befinden, lässt im direkten Vergleich (Abb. 7.3) mit Infektionsherden ebenfalls keine Rückschlüsse auf die Ausbreitungsherde zu.

Zusammenfassend kann gesagt werden, dass aus theoretischen Graphenmetriken bezüglich der Netztopologie nicht zwingend ein Rückschluss auf die Ausbreitungsbeteiligung oder -verantwortung gezogen werden kann. Allerdings konnte beobachtet werden, dass der mobile Knoten, welcher eine geringe Kontaktwahrscheinlichkeit aufweist, da er oft nicht mit der restlichen Infrastruktur verbunden ist, meist erst sehr spät infiziert wird.

Einschränkend sei hinzugefügt, dass sich die Messergebnisse auf das spezifische Szenario beziehen und somit ein Gegenbeispiel für die generelle Aussagekraft gefunden wurde. Bei der Untersuchung wurde eine Schadsoftware simuliert, die sich hauptsächlich auf Applikationsebene verbreitet. Malware mit anderen Ausbreitungsvektoren, die sich vor allem auf Infrastruktursysteme (Router, Accesspoints, ...) stützen, kann durchaus andere Verhaltensweisen aufweisen.

7.2.2. Andere Faktoren

Da aus den reinen Graphenmetriken keine generelle Aussage über den Einfluss der Topologie auf das Ausbreitungsverhalten von Schadsoftware, und somit keine Aussage über die Resilienz auf dieser Basis getroffen werden konnte, sollen hier andere Faktoren ergründet werden.
Bei einer sachlichen Betrachtung des Szenarios ist erkennbar, dass eine hohe Ausbreitungsbeteiligung des Netzwerkspeichers dem Szenario inhärent ist. Dies ist vor allem durch die Ausbreitungsvektoren der Malware vorherbestimmt. Diese sind aber ohne Interaktion wirkungslos und somit nur durch das Benutzerverhalten wirksam. Somit kann gesagt werden, dass eine Kombination aus den beiden Faktoren Benutzerverhalten und Ausbreitungsvektoren in diesem Szenario ausschlaggebend für die Ausbreitung ist. Da es aber unterschiedliche Arten von Schadsoftware gibt, die unterschiedliche Ausbreitungsvektoren verwenden, teilweise auch ohne Benutzerinteraktion, lassen sich generalisierte Aussagen nicht treffen.
Allerdings können Gefahrenpotentiale erkannt werden. Stark frequentierte Stellen, die Malware verbreiten können, sind dabei ein solches Gefahrenpotential, da die Kontaktwahrscheinlichkeit zur Schadsoftware erhöht wird. Dies zeigt auch Abbildung 7.4, die die Infektionsherde und die Aufrufe von Diensten gegenüberstellt.



(a) Aufrufe von Knotendiensten (b) Infektionsherde (dunkler = mehr Infektionen)

Abbildung 7.4.: Vergleich Dienstanfragen/Infektionsherde

Zudem bestehen Parallelen zu Waterhole-Attacken, die eine vermeintlich vertrauenswürdige Quelle, die von einer großen Benutzergruppe genutzt wird, infizieren und zur Ausbreitung nutzen (Security, 2012).

Schlussfolgernd hieraus kann gesagt werden, dass im Falle einer Schadsoftware, wie sie für die Fallstudie genutzt wurde - also zum Beispiel Stuxnet -, der Benutzer weitaus relevanter als der Aufbau des Netzwerks ist. Somit ist die Betrachtung des Benutzerverhaltens wichtiger, als die der topologischen Gegebenheiten.

7.2.3. Empfehlungen

Mit der erlangten Sichtweise lässt sich für die Resilience-Analyse und -Förderung von komplexen Computernetzwerken - z.B. innerhalb von Unternehmen - sagen, dass es sinnvoll ist, sich über die Arbeitsweise der Benutzer zu informieren, um potentielle zentrale Ausbreitungspunkte zu identifizieren. Um die Widerstandsfähigkeit gegen Schadsoftware zu erhöhen, kann mit klassischen Konzepten, wie Seperation of Duties (SoD) und dem damit verbundenen Identity and Access Management (IAM) gearbeitet werden, um Zugriffe zu beschränken. Dies kann die Ausbreitung einschränken, wie auch in der Fallstudie gezeigt wurde: Durch die Zugriffsbeschränkung auf den Netzwerkspeicher, konnte die Infektion eines Teilbereiches verhindert werden, da die Kontaktwahrscheinlichkeit auf 0% gesenkt wurde.

Neben den Möglichkeiten, die technischen Systeme dem Benutzerverhalten anzupassen, existiert die Möglichkeit das Verhalten der Benutzer zu ändern und diese für Gefahrenquellen zu sensibilisieren. Solche Awareness-Maßnahmen können dazu beitragen, dass die Nutzung bestimmter Techniken, wie zum Beispiel Cloud- oder Netzwerkspeicher, bewusster geschieht und deren Einsatz möglicherweise vom Benutzer selbst eingeschränkt wird.

8. Zusammenfassung und Ausblick

Schadsoftware und ihr Ausbreitungsverhalten spielen bei Angriffen auf Informationssysteme eine wichtige Rolle. Die von Malware ausgehende Gefahr ist durch die gestiegene Mobilität und die Nutzung von Cloud-Speichern in den letzten Jahren noch gestiegen. Die Risiken schon frühzeitig zu erkennen, kann bei der Vermeidung von Infektionen helfen. Dazu gehört auch das Verständnis über die Ausbreitung von Schadsoftware. Um dieses zu erlangen wurden Ausbreitungssimulationen als probates Mittel identifiziert.
Im Rahmen der Arbeit konnte ein Simulationsframework geschaffen werden, das einen agentenbasierten Ansatz verwendet, um Aspekte wie Mobilität und Systemkonfigurationen in eine Ausbreitungssimulation einbeziehen zu können. Hierbei wurde auf starke Modularität geachtet, um die Diversität von Systemlandschaften berücksichtigen zu können.

Die Simulation eines Szenarios innerhalb einer Fallstudie wurde dazu genutzt, um Messwerte zu erlangen. Diese wurden dazu verwendet, einen Bezug zwischen der Ausbreitung von Schadsoftware und der verwendeten Netzwerktopologie herstellen oder auch ausschließen zu können. Die Betrachtung der Ergebnisse zeigt, dass ein Bezug zwischen der Ausbreitung und der Topologie nicht zwingend bestehen muss. Der Bezug zur Topologie kann je nach Schadsoftware anders ausfallen. Für die untersuchte Malware konnte zumindest ein solcher nicht festgestellt werden.
Dafür stellt sich heraus, dass das Verhalten der Benutzer und ihre Arbeitsweise eine Ausbreitung maßgeblich vorantreiben können, was nicht überraschend ist. Daher wäre eine tiefere Untersuchung des Nutzerverhaltens für fortführende Arbeiten sicher interessant. Das Verhalten könnte mit BDI-Agenten wesentlich natürlicher gestaltet werden, als es hier mit Zustandsautomaten gemacht wurde. Zusätzlich müssten weitere Indikatoren identifiziert werden, die bei der Auswertung des Nutzerverhaltens helfen. Wahrscheinlich ist bei komplexeren Verhaltensweisen eine Rekonstruktion einer weiteren Schicht sinnvoll, die das „soziale“ Verhalten der Benutzer und deren Umgang mit den Systemen abbildet. Hiermit könnte untersucht werden, ob sich Erkenntnisse aus der Sozialen Netzwerkanalyse ebenfalls auf die Nutzungsstatistiken anwenden und Rückschlüsse auf die Ausbreitungsbegünstigung von Schadsoftware ziehen

lassen.

Weiterhin wären für fortführende Arbeiten auch Performanceuntersuchungen und -verbesserungen sowie das Entwickeln einer Testmethodik für die verteilten Agenten sicherlich interessant, um das Framework auch für größere Szenarien verwenden zu können und die Entwicklung dieser zu vereinfachen.
Es wird auf jeden Fall davon ausgegangen, dass der eingeschlagene Weg, die Systemumgebung komplett virtuell zu modellieren und dabei Systeme individuell zu betrachten, auch für weitere Arbeiten sinnvoll sein kann. Hiermit wird ermöglicht sowohl temporale als auch spatiale Eigenschaften der Ausbreitung detailliert nachzuvollziehen.

A. Anforderungen

A.1. Fachliche Anforderungen

A.1.1. Grundlegend

FA-1 Bestehende Infrastrukturen sollen weitgehend automatisch erfasst werden können.

FA-2 Manuelles Datenerfassen soll möglich sein.

FA-3 Manuelles Nachbearbeiten der Daten soll möglich sein.

FA-4 Eine manuelles Bearbeitung der Daten soll automatisch erfasste Daten nicht überschreiben.

FA-5 Systeme sollen einzeln und differenzierbar betrachtet werden können.

A.1.2. Systemeigenschaften und Vulnerabilität

FA-6 Installierte Software auf den Systemen soll möglichst automatisch erfasst werden.

FA-7 Laufende Dienste sollen automatisch erfasst werden können.

FA-8 Versionsnummern der installierten Software und Dienste sollen ermittelt werden.

FA-9 Die unter FA-6 bis FA-8 genannten Eigenschaften sollen nach einer automatischen Erfassung gesichtet und bearbeitet werden können.

FA-10 Die unter FA-6 bis FA-8 genannten Eigenschaften sollen auch ohne Automatismus manuell erfassbar sein.

FA-11 Vulnerabilitäten sollen sich aus den Eigenschaften ableiten lassen können.

A.1.3. Zentralität und Mobilität

FA-12 Zentrale Speicherorte für Dateien sollen berücksichtigt werden.

FA-13 Die Mobilität von Wechseldatenträgern und Endgeräten soll beachtet werden.

A.1.4. Benutzer und Aktivitäten

FA-14 Benutzer sollen ein Arbeitsverhalten aufweisen.

FA-15 Tätigkeiten von Benutzern sollen sich in ihrer Dauer und Auswirkung an der Realität orientieren.

FA-16 Geolokale Angaben müssen nicht berücksichtigt werden.

A.2. Kommunikationswege

FA-17 Kommunikation zwischen den Agenten soll unter Berücksichtigung realer technischer Einschränkungen geschehen. Hierfür soll OSI-Layer 3 betrachtet werden.

FA-18 Für das Routing soll ein Shortest-Path-Algorithmus verwendet werden.

FA-19 Das Filtern von Nachrichten muss ermöglicht werden.

FA-20 Kommunikationspfade müssen zur Simulationszeit dynamisch änderbar sein.

A.2.1. Topologie

FA-21 Die Vernetzung von Systemen soll automatisch möglichst vollständig bis OSI-Layer 3 erfasst werden.

A.3. Schadsoftware

FA-22 Die simulierte Schadsoftware soll eine APT nachstellen.

FA-23 Die simulierte Schadsoftware soll sich via Netzwerkspeicher ausbreiten.

FA-24 Portable Speicher/Geräte sollen einen weiteren Ausbreitungsweg der Schadsoftware darstellen.

A.4. Technische Anforderungen

A.4.1. Allgemein

TA-1 Das Simulationsframework soll plattformunabhängig implementiert sein.

TA-2 Auf einem System sollen mehrere CPU-Kerne - wenn vorhanden - ausgenutzt werden.

TA-3 Eine Verteilung auf mehrere Systeme soll möglich sein.

A.4.2. Visualisierung

TA-4 Auch die Visualisierung soll plattformunabhängig realisiert werden.

TA-5 Die Verteilung auf mehrere Systeme (TA-3) soll auch bei der Visualisierung berücksichtigt werden.

TA-6 Eine Visualisierung des Infektionsverlaufs zur Laufzeit soll möglich sein.

TA-7 Eine grafisch unterstützte Darstellung von Kennzahlen soll realisiert werden.

A.4.3. Erweiterbarkeit

TA-8 Durch den Wunsch nach einem Framework ist besonders auf Modularität und Erweiterbarkeit zu achten.

B. Inhalt der DVD

Dieser Arbeit liegt in ihrer analogen Fassung eine DVD bei. Diese beinhaltet in den einzelnen Ordnern:

- **Framework/core:** Den Frameworkkern mit Basis-Agent, Routing-Klasse, etc.
- **Framework/binutil:** Den Topologie-Editor, AgentAssembler und ein Replayscript (nicht universell verwendbar)
- **Gephi:** Das Graphenvisualisierungs-Tool, welches für diese Arbeit genutzt wurde
- **Szenario:** Die Stammdaten für das Szenario
- **Thesis:** Dieses Dokument in digitaler Form
- **Thesis/bib:** Ein Großteil der referenzierten Arbeiten in digitaler Form

Literaturverzeichnis

[BSI 2011] BSI: *Cyber-Sicherheitsstrategie für Deutschland.* Februar 2011

[Cheng et al. 2011] Cheng, Shin-Ming ; Ao, Weng-Chon ; Chen, Pin-Yu ; Chen, Kwang-Cheng: On Modeling Malware Propagation in Generalized Social Networks. In: *Communications Letters, IEEE* 15 (2011), Nr. 1, S. 25–27. http://dx.doi.org/10.1109/LCOMM.2010.01.100830. – DOI 10.1109/LCOMM.2010.01.100830. – ISSN 1089–7798

[Cohen 1984] Cohen, Fred: *Computer Viruses - Theory and Experiments*, Diss., 1984

[Deloitte 2013] Deloitte: Blurring the lines: 2013 TMT Global Security Study / Deloitte Touche Tohmatsu Limited. 2013. – Forschungsbericht

[Gephi] Gephi: *Gephi Homepage and Documentation.* https://gephi.org/

[Gregori et al. 2006] Gregori, Miguel E. ; Cámara, Javier P. ; Bada, Gustavo A.: A jabber-based multi-agent system platform. In: *Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems.* New York, NY, USA : ACM, 2006 (AAMAS '06). – ISBN 1–59593–303–4, 1282–1284

[John J. Kelly 2008] John J. Kelly, Lauri A.: The botnet peril / Hoover Institution. Version: 2008. http://www.hoover.org/publications/policy-review/article/5662. 2008. – Forschungsbericht. – abgerufen 10.01.2014

[Kaspersky Lab 2012] Kaspersky Lab: Kaspersky Lab Identifies Operation "Red October", an Advanced Cyber-Espionage Campaign Targeting Diplomatic and Government Institutions Worldwide / Kaspersky Lab. Version: 2012. http://www.securelist.com/en/blog/785/The_Red_October_Campaign_An_Advanced_Cyber_Espionage_Network_Targeting_Diplomatic_and_Government_Agencies. 2012. – Forschungsbericht. – Abgerufen 17.02.2013

[Kaspersky Lab 2014] Kaspersky Lab: Unveiling "Careto" - The Masked APT. 2014. – Forschungsbericht

[Kephart u. White 1991] Kephart, J.O. ; White, S.R.: Directed-graph epidemiological models of computer viruses. In: *Research in Security and Privacy, 1991. Proceedings., 1991 IEEE Computer Society Symposium on*, 1991, S. 343–359

[Krauß 2012] Krauss, Robert: Analyse und Aufbereitung sicherheitsrelevanter Informationen von Netzwerkknoten zur Simulation von IT-Angriffen. 2012. – Forschungsbericht

[Leszczyna et al. 2008] Leszczyna, Rafał; Fovino, Igor N. ; Masera, Marcelo: MAISim: mobile agent malware simulator. In: *Proceedings of the 1st international conference on Simulation tools and techniques for communications, networks and systems & workshops*. ICST, Brussels, Belgium, Belgium : ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2008 (Simutools '08). – ISBN 978–963–9799–20–2, 35:1–35:6

[Linksys 2014] Linksys: *How to prevent your Linksys router from getting The Moon malware.* `http://kb.linksys.com/Linksys/ukp.aspx?vw=1&articleid=29259`. Version: 2014. – abgerufen 28.02.2014

[Lora Billings 2002] Lora Billings, Ira B. S. William M. Spears S. William M. Spears: A unified prediction of computer virus spread in connected networks. In: *Physics Letters A* 297 (2002), S. 261–266

[Mandiant 2013] Mandiant: APT1 - Exposing One of China's Cyber Espionage Units. 2013. – Forschungsbericht

[Security 2012] Security, RSA: Lions at the Watering Hole – The "VOHO" Affair. Version: Juli 2012. `http://blogs.rsa.com/lions-at-the-watering-hole-the-voho-affair/`. 2012. – Forschungsbericht. – abgerufen 01.03.2014

[Siedersleben 2004] Siedersleben, Johannes: *Moderne Software-Architektur: Umsichtig planen, robust bauen mit Quasar*. d.punkt, 2004

[Song et al. 2008] Song, Yurong ; Jiang, Guo ping ; Gu, Yiran: Modeling malware propagation in complex networks based on cellular automata. In: *Circuits and Systems, 2008. APCCAS 2008. IEEE Asia Pacific Conference on*, 2008, S. 259–263

[SPADE] SPADE: *SPADE Homepage and Documentation.* `https://github.com/javipalanca/spade`. – abgerufen 20.12.2014

[Symantec Corporation 2011] Symantec Corporation: W32.Stuxnet Dossier / Symantec Corporation. 2011. – Forschungsbericht. – abgerufen 01.04.2011

[Symantec Corporation 2012] Symantec Corporation: W32.Disttrack / Symantech Corporation. Version: 2012. http://www.symantec.com/security_response/writeup.jsp?docid=2012-081608-0202-99. 2012. – Forschungsbericht. – abgerufen 17.02.2013

[Wang u. Stavrou 2010] Wang, Zhaohui ; Stavrou, Angelos: Exploiting smart-phone USB connectivity for fun and profit. In: *Proceedings of the 26th Annual Computer Security Applications Conference*. New York, NY, USA : ACM, 2010 (ACSAC '10). – ISBN 978–1–4503–0133–6, 357–366

[Wooldridge 2002] Wooldridge, Michael: Intelligent Agents: The Key Concepts. In: *Proceedings of the 9th ECCAI-ACAI/EASSS 2001, AEMAS 2001, HoloMAS 2001 on Multi-Agent-Systems and Applications II-Selected Revised Papers*. London, UK, UK : Springer-Verlag, 2002. – ISBN 3–540–43377–5, 3–43

Abkürzungsverzeichnis

APT	Advanced Persistent Threat.
BYOD	Bring your own Device.
CIFS	Common Internet File System.
CVE	Common Vulnerabilities and Exposures.
IAM	Identity and Access Management.
ICS	Industrial Control System.
IDS	Intrusion Detection System.
JSON	JavaScript Object Notation.
NAS	Network Attached Storage.
SoD	Seperation of Duties.
SPADE	Smart Python multi-Agent Development Environment.
VPN	Virtual Private Network.
XMPP	Extensible Messaging and Presence Protocol.

Glossar

BYOD

Der Begriff Bring Your Own Device (BYOD) beschreibt eine Organisationsrichtlinie, die festlegt, wie Mitarbeiter private Geräte zu dienstlichen Zwecken nutzen dürfen .

CIFS

CIFS ist eine erweiterte Version von SMB (Server Message Block) und erlaubt z.B. Datei- und Druckerfreigaben über IP-Netzwerke .

CVE

CVE ist die Kurzform von Common Vulnerabilities and Exposures und beschreibt einen Industriestandard zum Benennen von Sicherheitslücken, um diese eindeutig identifizieren zu können. Hierfür werden CVE-IDs (kurz ebenfalls CVE genannt) verwendet. Eine Datenbank von CVEs ist unter `http://cve.mitre.org/` verfügbar. .

DLL Hijacking

DLL Hijacking - auch als Binary Planting bekannt - bezeichnet das Laden einer DLL aus dem Arbeitsverzeichnis heraus, welches nicht unbedingt durch den Programmierer beabsichtigt ist. Dabei spielt die DLL-Search-Order von Windows eine wichtige Rolle. Durch das Platzieren einer DLL im Arbeitsverzeichnis wird diese einer gleichnamigen DLL in unteren Schritten dieser Reihenfolge vorgezogen. Somit lässt sich beliebiger Code ausführen .

IDS

Ein Intrusion Detection System (IDS) versucht Angriffe auf Computersysteme oder -netze zu erkennen. Dies geschieht meist anhand von Mustern, mit denen das Systemverhalten oder der Netzwerkverkehr verglichen wird .

JSON

JSON ist ein kompaktes, leichtgewichtiges Datenformat, das von Maschinen und Menschen leicht gelesen werden kann. Ursprünglich zum Austausch von JavaScript Objekten gedacht, wird es durch seine Leichtigkeit auch in anderen Sprachen und Kontexten zum Datenaustausch verwendet .

Monkey Patching

Monkey Patching ist auch unter dem Begriff Duck Punching bekannt und beschreibt eine Möglichkeit den Code von dynamischen Programmiersprachen während der Laufzeit zu ändern, ohne den original Code zu modifizieren. Es stellt somit eine Art der Realisierung von sich selbst modifizierendem Code dar .

PyPy

PyPy ist ein in Python Interpreter, der selber in Python geschrieben wurde. Er ist in vielen Fällen schneller als die in C geschriebene Referenzimplementierung CPython .

XMPP

XMPP steht kurz für Extensible Messaging and Presence Protocol. XMPP ist ein durch die IETF standardisiertes Protokoll für Sofortnachrichtendienste, welches neben der Nachrichtenübermittlung diverse weitere Funktionen unterstützt. Es war früher unter dem Namen Jabber bekannt .

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Hamburg, 1. April 2014 André Harms