

Universität Regensburg  
Fakultät für Wirtschaftswissenschaften  
Lehrstuhl für Wirtschaftsinformatik I - Informationssysteme

## **Angriffssimulation und strukturierte Datenerfassung**



Praxisseminar

Eingereicht bei: Prof. Dr. Günther Pernul  
Betreuung: Marietheres Dietz und Daniel Schlette

Eingereicht am 24. Juli 2020

Eingereicht von:

Johannes Seitz

Matrikelnummer: 2136257

# Abstract

Corona hat der Welt gezeigt, wie anfällig das Wirtschaftssystem ist. Viele Firmen haben erst jetzt den Wert des Einsatzes von IT-Systemen zu schätzen gelernt. Deshalb ist es unerlässlich diese Systeme zu schützen, denn dadurch kann die Wirtschaft auch in Zeiten eines Lockdowns aufrecht erhalten werden. Jedoch hat sich in den letzten Wochen und Monaten gezeigt, dass sich Hacker oder Cyberkriminelle nicht von Pandemien stoppen lassen. Deswegen ist es wichtig, Möglichkeiten zu entwickeln, die es Securityexperten weltweit ermöglichen strukturiert zusammenzuarbeiten.

Im Rahmen dieses Praxisseminars sollte zunächst ein Digitaler Zwilling entwickelt werden, der ein industrielles Setting simuliert. Daraufhin sollte als Nächstes ein Angriff auf das Netzwerk des Digitalen Abbildes erfolgen. Der Netzwerkverkehr der Attacke muss aufgezeichnet und im Anschluss strukturiert für Cyber Threat Intelligence aufgearbeitet werden.

Als erstes wurde mittels des Frameworks MiniCPS, welches auf Basis des Netzwerkemulators Mininet arbeitet, der Digitale Zwilling entwickelt. Als Setting wurde ein Förderbandsystem einer Abfüllanlage für Getränkeflaschen gewählt. Das Förderband wird durch einen Motor angetrieben und verfügt über die Eigenschaften des Motorstatus (Ein/ Aus) und der Geschwindigkeit, die über einen Sensor gemessen wird. Die Daten der Physischen Ebene werden über einen PLC mit der Netzwerkebene kommuniziert. Ein HMI kann die Daten auslesen und gegebenenfalls verändern.

Im nächsten Schritt wurde das Digitale Umfeld des Förderbandsystems mittels zweier Angriffsmethoden attackiert. Zu den Methoden zählte ein Man-In-The-Middle-Angriff und ein Denial-of-Service-Angriff. Die Daten des dabei angefallenen Netzwerkverkehrs wurden mit Hilfe des Programms Wireshark aufgezeichnet und im JSON-Format abgespeichert.

Als letzten Punkt auf der Agenda des Praxisseminars sollten die Informationen aus den Netzwerkmitschnitten der Angriffe strukturiert aufgearbeitet werden. Dazu wurde das Format STIX eingesetzt. STIX gilt als de facto Standard in vielen Organisationen und nutzt JSON als Serialisierungssprache. Im Rahmen des Seminars wurde ein Parser in der Programmiersprache Python geschrieben, um die wichtigsten Informationen aus dem angefallenen Netzwerkverkehr zu extrahieren und mittels eines Python-Producers in STIX-Objekte zu überführen.

Letztlich kann aus dem Praxisseminar geschlossen werden, dass die Kombination aus der Entwicklung von Digitalen Zwillingen für das industrielle Umfeld und der strukturierten Aufarbeitung von Angriffstests, die Securityexperten, wie zu Beginn bereits erwähnt, konstruktiver und schneller zusammenarbeiten können.

# Inhaltsverzeichnis

|   |            |
|---|------------|
| <b>Abbildungsverzeichnis</b>                                  | <b>ii</b>  |
| <b>Listings</b>   | <b>iii</b> |
| <b>Abkürzungsverzeichnis</b>                                  | <b>iv</b>  |
| <b>1 Einleitung</b>   | <b>1</b>   |
| <b>2 Aufbau und Ziele</b>                                     | <b>3</b>   |
| 2.1 Ziele des Praxisseminars . . . . .                        | 3          |
| 2.2 Ablauf des Praxisseminars . . . . .                       | 3          |
| 2.3 Tools . . . . .   | 4          |
| <b>3 Entwicklung des Digitalen Zwillings</b>                  | <b>6</b>   |
| 3.1 Was versteht man unter einem Digitalen Zwilling . . . . . | 6          |
| 3.2 Mininet . . . . .   | 7          |
| 3.3 MiniCPS . . . . .   | 8          |
| 3.4 Aufbau des Digitalen Zwillings . . . . .                  | 9          |
| <b>4 Angriffssimulation</b>                                   | <b>15</b>  |
| 4.1 Entscheidung für einen Angriff . . . . .                  | 15         |
| 4.2 MITM-Attacke . . . . .                                    | 16         |
| 4.3 DoS-Attacke . . . . .                                     | 17         |
| <b>5 Strukturierte Aufarbeitung der Angriffsdaten</b>         | <b>19</b>  |
| 5.1 Aufzeichnung der Daten . . . . .                          | 19         |
| 5.2 STIX . . . . .  | 19         |
| 5.3 Aufarbeitung der Angriffsdaten mit STIX . . . . .         | 20         |
| <b>6 Fazit</b>  | <b>25</b>  |
| <b>Literaturverzeichnis</b>                                   | <b>26</b>  |

# Abbildungsverzeichnis

|     |                                     |    |
|-----|-------------------------------------|----|
| 2.1 | Ablauf des Praxisseminars . . . . . | 3  |
| 3.1 | Befehl CLI . . . . .                | 7  |
| 3.2 | Beispiel MiniCPS . . . . .          | 8  |
| 3.3 | Förderband . . . . .                | 9  |
| 4.1 | Top 10 Bedrohungen . . . . .        | 15 |
| 4.2 | DoS-Angriff . . . . .               | 18 |

# Listings

|     |  |    |
|-----|--|----|
| 3.1 | Mininet Topo . . . . .                 | 7  |
| 3.2 | MiniCPS Befehle . . . . .              | 8  |
| 3.3 | Makefile . . . . .                     | 10 |
| 3.4 | Logger . . . . .                       | 10 |
| 3.5 | Topologie Förderband . . . . .         | 11 |
| 3.6 | Physischer Prozess . . . . .           | 12 |
| 3.7 | PLC . . . . .                          | 12 |
| 3.8 | HMI-Motorstatus . . . . .              | 13 |
| 4.1 | IP Forwarding . . . . .                | 17 |
| 4.2 | Hping . . . . .                        | 18 |
| 5.1 | Parser . . . . .                       | 21 |
| 5.2 | Anpassung Zeitstempel . . . . .        | 22 |
| 5.3 | Verbindungen am Beispiel TCP . . . . . | 23 |
| 5.4 | STIX Beispiel 1 . . . . .              | 24 |
| 5.5 | STIX Beispiel 2 . . . . .              | 24 |

# Abkürzungsverzeichnis

|      |  |
|------|--|
| API  | Application Programming Interface          |
| ARP  | Address Resolution Protocol                |
| CLI  | Command Line Interface                     |
| CPS  | Cyber Physical System                      |
| CTI  | Cyber Threat Intelligence                  |
| DoS  | Denial of Service Attacke                  |
| GUI  | Graphical User Interface                   |
| HMI  | Human Mashine Interface                    |
| ICS  | Industrial Control Systems                 |
| IDE  | Integrated Development Environment         |
| IDPS | Intrusion Detection and Prevention Systems |
| IP   | Internet Protocol                          |
| JSON | JavaScript Object Notation                 |
| MAC  | Media Access Control                       |
| MITM | Man-in-the-Middle-Attacke                  |
| PLC  | Programmable Logic Controller              |
| SDN  | Software-Defined Networking                |
| STIX | Structured Threat Information Expression   |
| SDOs | STIX Domain Objects                        |
| SCOs | STIX Cyber Observable Objects              |
| SROs | STIX Relationship Objects                  |
| TCP  | Transmission Control Protocol              |
| UUID | Universally unique identifiers             |

# Kapitel 1

## Einleitung

Wie sich in den Anfangsmonaten dieses Jahres bereits gezeigt hat, ist das Weltwirtschaftssystem bei Weitem fragiler, als es durch viele Experten vorhergesagt werden konnte. Innerhalb kürzester Zeit versetzte ein kleiner Virus die komplette Weltökonomie in einen ungewollten Ruhemodus. Es zeigte sich zudem, wie vorteilhaft sich der Einsatz von IT-Systemen auf die meisten Branchen auswirkt. Heimarbeit wurde von vielen Unternehmen gefördert, Universitäten konnten von zu Hause aus lehren und sogar das Teamtraining von Profisportlern wurde durch den Einsatz von Videotelefonie ermöglicht.

Dass die Verwendung von Informationssystemen und Internettechnologie nicht immer ohne Sicherheitsrisiken einhergeht ist allerdings nicht zuletzt seit der "Corona-Krise" bekannt. Immer wieder gelingt es Angreifern an hochsensible Daten zu kommen. Beispielsweise mussten Patienten eines tschechischen Krankenhauses verlegt werden, da die Uniklinik Brno gehackt wurde und somit der Betrieb der Klinik lahmgelegt wurde [Len20] [Mar20] [Pet20]. Aber auch in der Industrie und bei Behörden kommt es immer öfter zu Angriffsversuchen. Deshalb ist es für die jeweiligen IT-Abteilungen eine ständige Herausforderung die betriebseigenen Systeme sowohl proaktiv, als auch präventiv zu schützen.

Um sich vor potentiellen Risiken zu bewahren, ist es durchaus schwierig das laufende System unter Stresstests zu setzen. Zu groß ist die Gefahr einen Ausfall herbeizuführen und den laufenden Betrieb dadurch zu unterbrechen. Als eine hilfreiche Methode hat sich in den letzten Jahren das Simulieren eines sogenannten Digitalen Zwillings herauskristallisiert. Dieser Digital Twin soll es Entwicklern ermöglichen, ein reales System in einer geschützten Simulationsumgebung zu spiegeln und dieses unter beliebigen Bedingungen zu testen.

Eine weitere Herausforderung stellt die Präsentation von möglichen Angriffsvektoren dar. Die generierten Daten sind meistens nur schwer zu entziffern und daher nicht für den alltäglichen Umgang mit Bedrohungen im eigenen Netzwerk zu gebrauchen. Um dieses Problem zu lösen bietet es sich an, einen Standard zu verwenden, der bereits von vielen Akteuren der IT-Sicherheitsbranche genutzt wird. Ein derartiger Standard ist STIX (Structured Threat Information eXpression).

Im Rahmen eines Praxisseminars mit dem Titel 'Angriffssimulation und strukturierte

Datenerfassung' sollte ein industrielles Setting abgebildet werden und der Netzwerkverkehr eines simulierten Angriffs mitgeschnitten werden. Die erfassten Daten sollten dann mittels STIX 2.x strukturiert für Cyber Threat Intelligence aufgearbeitet werden.



## Kapitel 2

# Aufbau und Ziele

Dieses Kapitel beschäftigt sich mit den Zielen des Praxisseminars und wie dabei vorgegangen wurde diese zu erreichen. Zudem wird in einem weiteren Unterkapitel erläutert welche Tools für die Erarbeitung der Ziele genutzt wurden.

## 2.1 Ziele des Praxisseminars

Laut Aufgabenstellung durch den Lehrstuhl haben sich vor allem drei Hauptziele für die Erstellung dieses Praxisseminars herauskristallisiert:

- Aufbau eines Digitalen Zwillings als Abbild eines realistischen Industriesettings
- Angriffssimulation auf den Digitalen Zwilling und Aufzeichnen des angefallenen Netzwerkverkehrs
- Strukturierte Aufarbeitung der Daten des aufgezeichneten Netzwerkverkehrs

## 2.2 Ablauf des Praxisseminars

Nachdem dieses Semester von den Auswirkungen der Corona-Pandemie geprägt war, galt es zunächst eine Struktur für das Praxisseminar zu finden. Daher war es unter anderem wichtig sich einen Ablaufplan für die Bearbeitung des Seminars zu erarbeiten. In der folgenden Grafik kann man die ursprüngliche Zeitplanung anhand eines Gantt-Diagramms beobachten.

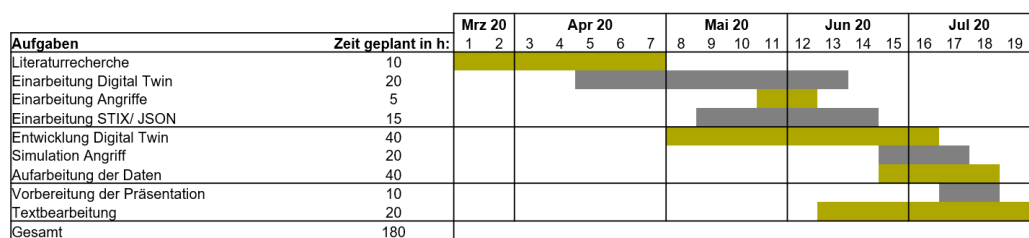


Abbildung 2.1: Ablauf des Praxisseminars

In der Grafik sieht man die Auflistung der Aufgaben die zur Strukturierung des Praxisseminars dienen. Zur zeitlichen Planung diente zudem die Orientierung an den ECTS-Punkten, die durch dieses Modul erreicht werden können. Nach erfolgreichem Bestehen des Moduls Praxisseminar erhält ein Student insgesamt sechs Credit-Points. Nach der üblichen Aufteilung von Arbeitsstunden pro ECTS, ergibt das Modul einen Workload von 180 Stunden Arbeit (30 Arbeitsstunden pro ECTS)[Hoc20]. In der obigen Darstellung kann man neben der Aufzählung der Arbeitsaufgaben, die für jede Aufgabe geplante Zeit und die ungefähre zeitliche Einteilung im Sinne eines Gantt-Diagramms erkennen. Nach Zusage des Themas durch den Lehrstuhl wurde bereits Mitte März 2020 mit der Literaturrecherche begonnen. Das Praxisseminar endete mit der Abgabe der schriftlichen Arbeit am 24.07.2020. Wie man der Grafik entnehmen kann, umfasste das Seminar ca. 19 Wochen.

In einem ersten Arbeitsschritt galt es sich in die Themen des Seminars einzulesen und einzuarbeiten. Hierbei wurde beispielsweise auf Quellen zurückgegriffen, die durch die Betreuer zur Verfügung gestellt wurden [AT][Bre20]. Des Weiteren wurde eine Literaturrecherche durchgeführt, um sich vor Allem in die Themen Digital Twin und Structured Threat Information Expression (STIX) einzulesen. Aufgrund der begrenzten Vorkenntnisse nahm bereits diese Phase der Einarbeitung und Recherche viel Arbeitszeit in Anspruch. Nach der intensiven Einarbeitungsphase, stand als Erstes die Entwicklung eines Digitalen Zwillings auf der Agenda. Hierbei muss erwähnt werden, dass sich auch während der Erarbeitung des Digital Twins oft Fragen ergaben und somit keine klare Trennung zwischen Einarbeitungsphase und Entwicklungsphase existierte. Des Weiteren musste nach Fertigstellung des Digitalen Zwillings laut Aufgabenstellung mindestens ein Cyberangriff auf den Digital Twin erfolgen. Als letzter Punkt auf der Agenda sollten die Daten des Angriffs mittels STIX strukturiert aufgearbeitet werden.

Zuletzt mussten die Daten des Praxisseminars in Form einer Live-Präsentation und des hier vorliegenden Textes strukturiert dargestellt werden.

## 2.3 Tools

Für diese Arbeit wurde auf einige Tools zurückgegriffen, die im Folgenden kurz aufgezählt und erläutert werden.

Zur Versionierung wurde die Plattform Github verwendet. Durch dieses Format konnten die Betreuer den Fortschritt des Codes permanent verfolgen. Vor allem der Code des Digital Twins wurde hierüber geteilt. Zudem konnte Github dazu genutzt werden den aktuellsten Code in die Virtuelle Maschine einzupflegen.

Für die Programmierung wurde die Entwicklungsumgebung PyCharm verwendet. PyCharm verfügt über eine Github-Schnittstelle und somit konnte der Code über wenige Kommandozeilenbefehle über ein Remote-Repository auf die Versionierungsplattform hochgeladen werden. PyCharm wurde als Integrated Development Environment (IDE) ausgewählt, da für dieses Praxisseminar hauptsächlich mit der Programmiersprache Python gearbeitet

wurde und PyCharm eine frei verfügbare IDE für diese Sprache ist. Außerdem wurde der Texteditor Notepad++ verwendet, um verschiedene Skripte einzusehen (bspw. JSON-Code).

Als Virtualisierungsumgebung für den Digitalen Zwilling wurde auf eine virtuelle Maschine auf Basis des Programms Virtualbox von Oracle mit einem Betriebssystem Ubuntu 18.04 zurückgegriffen.

## Kapitel 3

# Entwicklung des Digitalen Zwillings

Im Folgenden wird kurz erläutert, was unter dem Begriff eines Digitalen Zwillings zu verstehen ist, welche Frameworks verwendet wurden, um den Digital Twin zu gestalten und wie dieser im Rahmen des Praxisseminars entwickelt wurde.

### 3.1 Was versteht man unter einem Digitalen Zwilling

Unter einem Digitalen Zwilling (auch Digital Twin) versteht man im Allgemeinen den digitalen Gegenpart eines Realwelt-Systems. Diese Systeme beschreiben meist industrielle Settings [MBG19, 1]. Der digitale Zwilling wird optimalerweise so gestaltet, dass er sein reales Gegenüber in allen Funktionsweisen spiegelt. Dadurch können Systeme überwacht, simuliert, getestet oder sogar gegebenenfalls optimiert werden, ohne direkten Einfluss auf das reale Setting zu nehmen. Allerdings existierten laut Olaf Sauer vom Fraunhofer-Institut im Jahr 2018 für weniger als ein Prozent der heute ausgelieferten Maschinen ein derartiges digitales Abbild [Ola18]. Für das Konzept der Industrie 4.0 ist die weitere Entwicklung und Erforschung von Digitalen Zwillingen essentiell. Die digitale Spiegelung ist aber sehr anwendungsspezifisch und muss für jedes Unternehmen maßgeschneidert angepasst werden.

Für die Erarbeitung des Digital Twins im Rahmen dieses Praxisseminars musste zunächst entschieden werden, welches reale Ebenbild gespiegelt werden soll. Zur Auswahl standen die Abbildung eines Industrial Control Systems (ICS), wie beispielsweise das Setting einer Wasseraufbereitungsanlage, oder die Gestaltung eines rein Hostbasierten-Netzwerks, wie es beispielsweise bei einer Universität oder einem Krankenhaus der Fall ist. Zur Einschränkung der Komplexität und der Tatsache, dass ein physisches System bildlich besser dargestellt werden kann, fiel die Auswahl auf ein ICS.

Daraufhin war es entscheidend, das richtige Framework für die Erstellung des Industrial Control Systems zu erörtern. Dazu wurde von den Betreuern des Seminars ein URL-Link der Plattform Github zur Verfügung gestellt, der eine Übersicht an verschiedenen Frameworks und Tools zur Erstellung eines Digital Twins bietet (<https://github.com/hslatman/awesome-industrial-control-system-security>). Letztendlich ist die Entscheidung auf das Framework MiniCPS gefallen. Zu MiniCPS existiert ein Paper der Entwickler, welches das System

anhand von Beispielen konkretisiert [AT]. Zudem basiert das Framework auf der Programmiersprache Python, eine Sprache die sehr weit verbreitet und gegenüber anderen Programmiersprachen einfacher zu verstehen ist. Dadurch konnte die Einarbeitungszeit verkürzt werden und die Konzentration auf die Entwicklung des Digitalen Zwillings gelegt werden. MiniCPS basiert auf dem Netzwerk Emulator Mininet, der im folgenden Kapitel beschrieben wird.

## 3.2 Mininet

Mininet ist um das Paradigma des Software-definierten Netzwerks aufgebaut. Zudem wird durch den Einsatz von Mininet die Entwicklung und Erforschung von Software-Defined Networking (SDN) erleichtert [AT, 2]. Mininet nutzt Linux Container zur Virtualisierung. Man kann Mininet als alleinige virtuelle Maschine nutzen und beispielsweise mittels Virtualbox starten. In der Praxis hat sich allerdings gezeigt, dass Mininet in einer virtuellen Maschine mit dem Betriebssystem Ubuntu 18.04 am stabilsten läuft. Die Einarbeitung in das System erfolgte über die ausführliche Erläuterung auf der Webseite von Mininet [Min18]. Dort wird erklärt wie man Mininet installiert und mittels Python-Code ein einfaches Netzwerk simulieren kann.

```
1 from mininet.net import Mininet
2 from mininet.topolib import TreeTopo
3 tree4 = TreeTopo(depth=2, fanout=2)
4 net = Mininet(topo=tree4)
5 net.start()
6 h1, h4 = net.hosts[0], net.hosts[3]
7 print h1.cmd('ping -c1 %s' % h4.IP())
8 net.stop()
```

**Listing 3.1:** Mininet Topo

Der Code aus 3.1 zeigt wie mittels einer Funktion 'TreeTopo(depth=2, fanout=2)' eine Baumtopologie mit vier Hosts und drei Switches erstellt werden kann. Mit der Funktion '.start()' wird das Netzwerk gestartet und mit '.stop()' wird die Topologie wieder heruntergefahren. Mininet verfügt über ein Command Line Interface (CLI). Über diese Schnittstelle können verschiedene Befehle ausgeführt werden. Beispielsweise können neue Terminals aufgerufen werden, auf denen auf einzelne Knotenpunkte zugegriffen werden kann. Mit dem Befehl aus Abbildung 3.1 kann man die Verbindung zwischen zwei Hosts austesten.

```
mininet> h2 ping h3
```

**Abbildung 3.1:** Befehl CLI

Die Python Application Programming Interface (API) von Mininet ermöglicht es dem Entwickler das Netzwerk beliebig zu erweitern [AT, 3]. Im nächsten Abschnitt wird auf MiniCPS genauer eingegangen.

### 3.3 MiniCPS

Das nun zu beschreibende Framework bietet eine Reihe von Python-Werkzeugen, die einen realistischen Aufbau eines ICS ermöglichen. Dabei steht vor allem der Echtzeitaustausch von Netzwerkverkehr der Cyber Physical System (CPS) im Fokus. Dazu werden Python Skripte geschrieben, die beispielsweise sogenannte Programmable Logic Controller (PLC) emulieren und dadurch das Entschlüsseln von physischen Signalen steuern [AT, 3].

Laut den Entwicklern von MiniCPS verfolgt das Framework folgende Ziel [AT, 3]:

- Kosteneffektivität
- Kompatibilität
- Realistische Simulation von industriellem Netzwerkaustausch
- Nutzerfreundlichkeit: Die Werkzeuge sollten einfach zu konfigurieren sein
- Reproduzierbarkeit

Es wird in dem Paper allerdings klar gestellt, dass MiniCPS nicht als Performanz-Simulator oder als Werkzeug zur Optimierung eingesetzt werden sollte. Zusätzlich zum Paper existiert eine Webseite auf der man sich mit den Eigenschaften von MiniCPS vertraut machen und auch ein erstes Beispiel eines minimalen ICS ausprobieren kann. Das Beispiel stellt hierbei eine Wasseraufbereitungsanlage dar [Ant17].

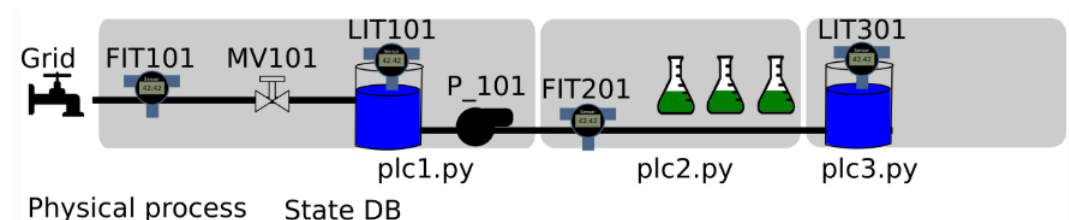


Abbildung 3.2: Beispiel MiniCPS

Das in Abbildung 3.2 dargestellte Schema besteht physisch gesehen aus vier Sensoren und zwei Aktuatoren. Die physischen Komponenten sind mit jeweils einem PLC verbunden. Diese wiederum kommunizieren mit dem Netzwerk und interagieren miteinander. Im Hintergrund werden die Daten in einer SQLite Datenbank gespeichert. Um MiniCPS zu installieren sollte bereits ein Ordner mit dem Mininet-Paket in der virtuellen Maschine vorhanden sein. Danach kann das Repository von MiniCPS in den Mininet-Ordner über einfache Git-Befehle geklont werden.

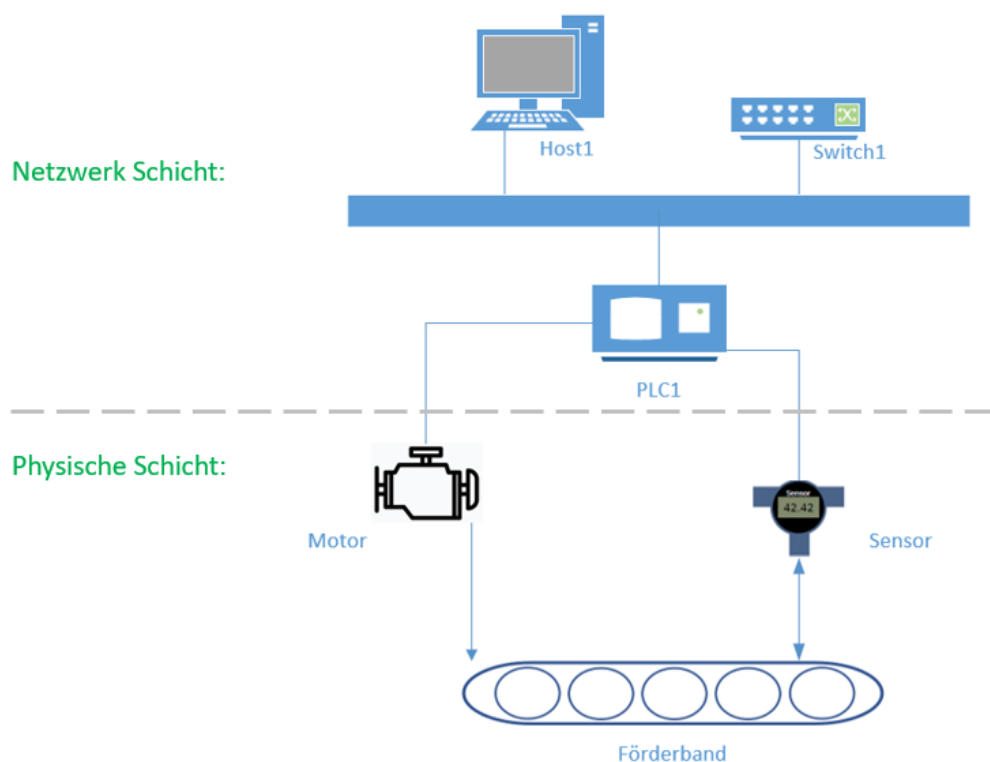
```
1 git clone https://github.com/scy-phy/minicps
2
3 ln -s ~/minicps/minicps /usr/lib/python2.7/minicps
```

Listing 3.2: MiniCPS Befehle

Zudem muss der Python Pfad für MiniCPS über den Befehl in Zeile drei im Code in 3.2 angepasst werden. Danach kann man über einen Terminal-Befehl in den MiniCPS Ordner navigieren und einen ersten Test über den Befehl 'make tests' ausführen. Nach einem Abbruch der Simulation ist es empfehlenswert den Befehl 'sudo mn -c' auszuführen, um alle aktiven Controller und Prozesse abzurechnen. Im folgenden Unterkapitel wird die Erarbeitung des Digitalen Zwillings für das Praxisseminar erläutert.

### 3.4 Aufbau des Digitalen Zwillings

Nachdem die Entscheidung auf die Entwicklung eines ICS als Digitalen Zwilling fiel, wurde erörtert welche Systeme gestaltet werden können. Das bereits vorgestellte Paper beschreibt eine Wasseraufbereitungsanlage. Ein weiteres Beispiel eines komplexen Industrial Control Systems wäre eine Abfüllanlage für Getränkeflaschen. Da im Hinblick auf den Arbeitsumfang des Seminars ein solches System nicht realisiert werden konnte, wurde ein Teilsystem ausgewählt. Dieses Teilsystem soll den Betrieb eines Förderbandes darstellen.



**Abbildung 3.3:** Förderband

Wie man in der Grafik in 3.3 erkennen kann, besteht das System aus einer Netzwerk- und einer Physischen Schicht. Die Physische Schicht umfasst den Prozess des Förderbandes, der durch einen Motor angetrieben wird und dessen Geschwindigkeit über einen Sensor überwacht wird. Sensor und Motor haben jeweils eine Verbindung mit einem PLC. Dieser PLC befindet sich in der Netzwerkschicht und stellt den Vermittler zwischen physischer Schicht und Netzwerkschicht dar. Der PLC ist mit dem Switch verbunden, der

wiederum auch mit einem Host in Verbindung steht. Für den Betrieb des CPS besteht ähnlich zu dem vorher beschriebenen SWAT-Tutorial Beispiel eine SQLite Datenbank [Ant17]. Zudem wurden vier Python Skripte für das Praxisseminar erstellt bzw. für die Gegebenheiten des Förderbandexempels angepasst. Hinsichtlich der Programmierung des Förderband-ICS wurden zunächst ein Teil der Python Skripte des SWAT-Tutorials übernommen ( /minicps/examples/swat-s1). Zudem wurde ein Teil des Makefiles im Hauptordner des MiniCPS Repositories angepasst.

```

1 #Praxisseminar {{{1
2
3 praxisseminar:
4     cd Praxisseminar; $(PYTHON) $(PYTHON_OPTS) run.py; cd ..
5
6 praxisseminar-init:
7     cd Praxisseminar; $(PYTHON) $(PYTHON_OPTS) init.py; cd ..

```

**Listing 3.3:** Makefile

Mit den Befehlen 'make praxisseminar-init' wird das Förderband initialisiert und mit 'make praxisseminar' wird das Beispiel über die Kommandozeile gestartet. Mit dem Python Skript 'init.py' wird das Beispiel initialisiert, wobei durch Aufruf des Skripts lediglich eine SQLite Tabelle eingerichtet wird. Die Initialisierungsdaten für die Datenbank befinden sich einem weiteren Skript 'utils.py'. Neben den Initialisierungsdaten, Netzwerkdaten und Kontrolldaten für die Geschwindigkeit, wird hier ein Logger definiert, der über einen Filehandler, Logging-Daten in eine Datei im Unterordner 'logs/' abspeichert (Listing 3.4).

```

1 # Eigener Logger
2
3 Praxisseminar_test_logger = logging.getLogger(__name__)
4 Praxisseminar_test_logger.setLevel(logging.DEBUG)
5
6 fh = logging.FileHandler('logs/file.log')
7 fh.setLevel(logging.DEBUG)
8
9 formatter = logging.Formatter('%(asctime)s - %(name)s - %(levelname)s - %(
    message)s')
10 fh.setFormatter(formatter)
11
12 Praxisseminar_test_logger.addHandler(fh)

```

**Listing 3.4:** Logger

Zur Beschreibung des Netzwerks, wird das Python-Modul 'mininet.topo' verwendet. Dazu wurde eine Klasse CbTopo erstellt, welche aus einem Switch und 3 Hosts besteht. Mit der Funktion 'self.addSwitch('s1')' wird ein Switch generiert. Hosts können mit der Funktion 'self.addHost('plc1')' initialisiert und mit einer Funktion 'self.addLink(host1, switch)' kann eine Verbindung zu einem Switch hergestellt werden (Listing 3.5).



```

1 class CbTopo(Topo):
2
3     """Foerderband 1 plc + 1 Host mit HMI + 1 Angreifer + 1 Switch"""
4
5     def build(self):
6
7         switch = self.addSwitch('s1')
8
9         plc1 = self.addHost(
10             'plc1',
11             ip=IP['plc1'] + NETMASK,
12             mac=MAC['plc1'])
13         self.addLink(plc1, switch)
14
15         host1 = self.addHost(
16             'host1',
17             ip=IP['host1'] + NETMASK,
18             mac=MAC['host1'])
19         self.addLink(host1, switch)
20
21         attacker = self.addHost(
22             'attacker',
23             ip=IP['attacker'] + NETMASK,
24             mac=MAC['attacker'])
25         self.addLink(attacker, switch)
26         Praxisseminar_test_logger.info('PLC1, Host1 und Attacker')

```

**Listing 3.5:** Topologie Förderband

Zusätzlich zur Initialisierung über die 'build'-Funktion in Listing 3.5 werden jeweils die IP- und MAC- Adressen definiert. Die Daten hierfür werden aus dem Modul 'utils.py' übergeben. Somit lässt sich beim Start der Topologie bereits diese drei angegebenen Hosts erreichen. Allerdings benötigt das Beispiel noch weitere Python-Skripte, sodass beispielsweise der automatisierte physische Prozess zum Laufen gebracht werden kann. Dazu muss der physische Prozess des Förderbandes zunächst über das entsprechende Skript in einem Terminal des Switch gestartet werden. Der folgende Codeausschnitt zeigt, was beim Start des physischen Prozesses abläuft (Listing 3.6). Wie man anhand des Code erkennen kann, befindet sich das Programm nach Start in einer Dauerschleife. Über eine get-Funktion wird zunächst der aktuelle Status des Motors gelesen. Danach wird mittels einer If-Abfrage abgeprüft, ob der Status des Motors ein (=1) oder aus (=0) ist. Falls der Motor ein ist, wird mit einer weiteren get-Funktion die aktuelle Geschwindigkeit, die über den Sensor gemessen wurde, gelesen. Bei einem ausgeschalteten Motor wird die Geschwindigkeit auf den Mindestwert '0.0' gesetzt. Wie man daran erkennen kann, ist der eigentliche physische Ablauf des Förderbandes sehr simpel gehalten. Kurz gesagt handelt es sich, um einen permanenten Datenbankaustausch, der über get- und set-Funktionen gesteuert wird.

```

1 while True:
2
3     # ueberpruefe ob Motor an ist
4     # wenn ja dann setze standarmaessig Anfangsgeschwindigkeit
5
6     motor = self.get(MOTOR)
7     Praxisseminar_test_logger.info('Motor: ' + str(motor))
8
9     if int(motor) == 1:
10
11         new_velocity = self.get(SENSOR)
12         Praxisseminar_test_logger.info('Sensor: ' + str(new_velocity))
13
14         self.set(SENSOR, new_velocity)
15         Praxisseminar_test_logger.info('Sensordaten wurden veraendert: ' + str(
16             new_velocity))
17
18     # DEBUG 'Standarmaessige Motorgeschwindigkeit
19     else:
20         self.set(SENSOR, MOTOR_VEL['MIN'])
21         Praxisseminar_test_logger.info('Sensordaten wurden veraendert: ' + str(
22             MOTOR_VEL['MIN']))

```

Listing 3.6: Physischer Prozess

Wie bereits zu Beginn des Kapitels erläutert wurde, werden die Rohdaten von Motorstatus und Geschwindigkeitssensor über einen PLC mit dem Netzwerk vermittelt. Dieser PLC wird über ein weiteres Skript auf dem PLC1 Host gestartet.

```

1 while True:
2     rec_m11 = int(self.receive(MOTOR, PLC1_ADDR))
3     self.set(MOTOR, rec_m11)
4     Praxisseminar_test_logger.info("Motor erhaelt von PLC1_ADDR: %s" % str(
5         rec_m11))
6     self.send(MOTOR, rec_m11, PLC1_ADDR)
7     Praxisseminar_test_logger.info("Motor sendet an PLC1_ADDR: %s" % str(rec_m11))
8
9     rec_s11 = float(self.receive(SENSOR, PLC1_ADDR))
10    # Programmabbruch bei zu hoher Geschwindigkeit
11
12    if rec_s11 > MOTOR_VEL['MAX']:
13        Praxisseminar_test_logger.warning('PLC1 shutdown, die Geschwindigkeit hat
14            das Maximum ueberschritten ')
15        self.send(SENSOR, 0.0, PLC1_ADDR)
16        self.set(SENSOR, 0.0)
17        self.send(MOTOR, 0, PLC1_ADDR)
18        self.set(MOTOR, 0.0)
19        break
20    else:
21        if '0.0' in str(rec_s11) and str(rec_m11) == '1':
22            self.set(SENSOR, MOTOR_VEL['STD'])
23            self.send(SENSOR, MOTOR_VEL['STD'], PLC1_ADDR)
24        else:
25            self.set(SENSOR, rec_s11)
26            self.send(SENSOR, rec_s11, PLC1_ADDR)

```

Listing 3.7: PLC

Das Programm des PLCs läuft so lange die Höchstgeschwindigkeit nicht überschritten wird. Als Vermittler zwischen Netzwerk- und physischer Ebene beinhaltet der Programmcode Funktionen aus beiden Bereichen. Die receive- und send- Funktionen verschicken jeweils Netzwerkpakete an die im Code angegebenen Adressen. Zudem werden die erhaltenen Daten über get- und set-Funktionen mit der physischen Ebene kommuniziert. Zunächst erhält der PLC über das Netzwerk den Status des Motors und übergibt diesen wiederum an den physischen Prozess. Danach wird der Geschwindigkeitssensor nach neuen Daten abgefragt. Falls die übergebene Geschwindigkeit nicht die Höchstgeschwindigkeit des Förderbandes überschreitet, werden die Sensordaten wiederum an den Sensor auf physischer Ebene übermittelt und die Geschwindigkeit erhöht oder gesenkt. Sollte die Geschwindigkeit allerdings höher als die angegebene Höchstgeschwindigkeit sein, setzt der PLC den Sensor auf '0.0', schaltet den Motor aus und bricht das PLC-Programm ab. Zusätzlich kann man als Benutzer des Förderbandes über einen Host im Netzwerk ein Human Machine Interface (HMI)-Programm aufrufen, um den Status des Prozesses zu überwachen und gegebenenfalls zu verändern. Beispielsweise kann der Status des Motors ausgelesen werden, wie man anhand von Codeausschnitt 3.8 erkennen kann.

```

1 while True:
2     Praxisseminar_test_logger.debug("Die HMI mit Adresse " + str(HMI_ADDR) + "
        befindet sich in der main loop")
3     print "Sie haben folgende Optionen: "
4     print
5     eingabe = int(raw_input("Auslesen Status: Taste 1/ Geschwindigkeit einstellen:
        Taste 2/ Ein-/Ausschalten: Taste 3/ Programm beenden: Taste 99 "))
6     print 'DEBUG: eingabe = %s' % eingabe
7     Praxisseminar_test_logger.debug("Der User hat folgendes eingegeben: %s" % str(
        eingabe))
8
9     # Status abfragen
10    if eingabe == 1:
11        Praxisseminar_test_logger.debug("User befindet sich in der ersten if-Abfrage
            ")
12        motor = self.receive(MOTOR, PLC1_ADDR)
13        print "DEBUG plc1 erhaelt motor: " + motor
14        Praxisseminar_test_logger.info('Motor erhaelt von PLC1_ADDR: ' + motor)
15
16        if motor == '1':
17            print 'DEBUG plc1 motor: An'
18            Praxisseminar_test_logger.info("Der Motor ist An")
19        elif motor == '0':
20            print 'DEBUG plc1 motor: Aus'
21            Praxisseminar_test_logger.info("Der Motor ist Aus")
22    """ weiterer Code """

```

**Listing 3.8:** HMI-Motorstatus

Aus Platzgründen wurde in Listing 3.8 auf die ausführliche Darstellung des Programmcodes der HMI verzichtet. Das komplette HMI-Programm agiert in einer Dauerschleife, die zunächst dem Benutzer insgesamt vier Optionen gewährt: Auslesen des Motorstatus mit Tastatureingabe 1; Geschwindigkeit einstellen mit Tastatureingabe 2; Ein- und Ausschalten des Förderbandes über Tastatureingabe 3; Programm beenden

über Tastatureingabe 99. Daraufhin wird in einer If-Abfrage abgeprüft, welche Eingabe der Benutzer getätigt hat. Da es sich bei diesem Beispiel um ein akademisches Projekt handelt, wurde auf den Einbau eines Error-Handlers verzichtet, der eine Falscheingabe des Benutzers abfangen würde. Beim Auslesen des Geschwindigkeitsstatus wird über eine receive-Funktion zunächst einmal der aktuelle Status des Motors über das Netzwerk ausgelesen und im Anschluss dem Benutzer über eine Terminal-Ausgabe kommuniziert. Um den Motor ein- oder auszuschalten, kann der Benutzer mit der Eingabe von drei zu einer weiteren Abfrage navigieren. Hierbei wird wieder der aktuelle Status über das Netzwerk kommuniziert. Mit der entsprechenden Terminaleingabe kann der Benutzer den Status des Motors beibehalten oder verändern. Des Weiteren kann der User über die Eingabe von zwei, die Geschwindigkeit des Motors verändern. Dabei wird zunächst abgeprüft, ob der Motor bereits an ist. Sollte der Motor aus sein, wird dem Nutzer über das Terminal ausgegeben, dass sich der Motor im Status aus befindet. Anschließend springt die Schleife wieder zum Ausgangspunkt. Ist der Motor an, wird der Benutzer gefragt, ob er die Geschwindigkeit verändern möchte. Wenn der Benutzer die Geschwindigkeit verändern möchte, kann er dies über eine Float-Eingabe vollziehen. Die Eingabe wird wiederum über einen receive-Befehl über das Netzwerk an den PLC kommuniziert. Insofern die Eingabedaten nicht die Maximalgeschwindigkeit überschreiten, wird die Änderung vom PLC an den Sensor über den set-Befehl weitergegeben.

Um zu gewährleisten, dass der digitale Zwilling stabil läuft, muss beim Starten folgende Reihenfolge eingehalten werden:

- Mininet Prozesse über Befehl 'sudo mn -c' bereinigen
- Befehl 'make praxisseminar' im Ordner 'Praxisseminar-SS2020' über das Terminal ausführen
- Über 'xterm plc1 host1 s1' Terminals starten
- Im s1-Terminal das Skript des Physischen Prozesses starten
- Im plc1-Terminal das Python-Skript 'plc1.py' ausführen
- Zuletzt im host1-Terminal das Skript der HMI starten

Nachdem der Digitale Zwilling funktionsfähig entwickelt wurde, galt es als nächsten Agendapunkt, einen Angriff auf diesen zu simulieren.

## Kapitel 4

# Angriffssimulation

Vor dem Start eines Angriffs musste zunächst eruiert werden, welche Attacke für den Digitalen Zwilling in Betracht gezogen werden kann. Diese Entscheidung wird im nächsten Unterkapitel erläutert. Im Anschluss werden zwei Angriffe genauer beschrieben.

### 4.1 Entscheidung für einen Angriff

Immer mehr Industrieanlagen agieren über vernetzte Systeme. Und dadurch werden diese Anlagen auch immer öfter als Ziel von Cyberattacken ausgewählt. Unternehmen werden gezielt angegriffen, um beispielsweise Lösegeldforderungen auszusprechen oder dem Konkurrenten zu schaden. Laut einem Bericht des Bundesamtes für Sicherheit in der Informationstechnik zählen folgende Angriffe zu den Top zehn Bedrohungen [BSI19, 2]:











| Top 10 Bedrohungen  | Trend seit 2016   |
|---|---|
| Einschleusen von Schadsoftware über Wechseldatenträger und externe Hardware |  |
| Infektion mit Schadsoftware über Internet und Intranet                      |  |
| Menschliches Fehlverhalten und Sabotage                                     |  |
| Kompromittierung von Extranet und Cloud-Komponenten                         |  |
| Social Engineering und Phishing   |  |
| (D)DoS Angriffe   |  |
| Internet-verbundene Steuerungskomponenten                                   |  |
| Einbruch über Fernwartungszugänge   |  |
| Technisches Fehlverhalten und höhere Gewalt                                 |  |
| Kompromittierung von Smartphones im Produktionsumfeld                       |  |

Abbildung 4.1: Top 10 Bedrohungen

Wie man anhand von Abbildung 4.1 erkennen kann, zählen unter anderem menschliches Fehlverhalten und DoS-Angriffe zu den Top-Bedrohungen im Industrieumfeld. Zudem verzeichnen diese beiden Angriffstypen seit 2016 einen starken Zuwachs. Da die Komplexität des entwickelten Digitalen Zwillings eingeschränkt ist, konnten einige der aufgelisteten Top-Bedrohungen nicht simuliert werden. Um das Angriffsgeschehen dennoch realistisch darstellen zu können, wurden einige Annahmen vor dem Start der

Simulation definiert.

Bei den Angriffen auf den Digitalen Zwilling wird von einem Innentäter als Threat Actor ausgegangen. Dieser Innentäter verfügt über Detailwissen bezüglich des ICS, weiß über die Schwachstellen des Systems Bescheid und kann gegebenenfalls Hindernisse (z.B.: Firewall) umgehen. Der Angreifer verfügt über Expertenwissen und weiß wie er sich unerkannt im Netzwerk aufhalten kann. Motivation für einen derartigen Angriff durch einen Innentäter könnte eventuell eine anstehende Entlassung oder die Bestechung durch eine konkurrierende Firma sein. In den folgenden beiden Unterkapitel wird nun auf die ausgewählten Angriffe näher eingegangen und das jeweilige Vorgehen im Rahmen des Praxisseminars genauer erläutert.

## 4.2 MITM-Attacke

Bei einem Man-in-the-Middle-Attacke (MITM) handelt es sich um einen häufig zitierten Angriffstypus. Dabei geht man davon aus, dass sich ein Threat Actor im Netzwerk befindet und die Kommunikation von zwei oder mehr Kommunikationspartner mithört [MIT19b]. MITM können einem Angreifer dazu verhelfen an Passwörter zu gelangen oder den Datenaustausch mitzulesen. Dieses Vorgehen kann durch das Nutzen von vorgefertigten Graphical User Interface (GUI)-Programmen vereinfacht werden. Für den simulierten Angriff kommen die Programme Wireshark und Ettercap zum Einsatz. Wireshark ist ein sogenannter Packet-Sniffer, der dem Nutzer die Möglichkeit gibt über die Auswahl eines Interfaces und die Eingabe von Filtern Netzwerkverkehr zur Laufzeit mitzulesen, aufzuzeichnen und in verschiedenen Formaten abzuspeichern. Das Programm Ettercap ist speziell für MITM entwickelt worden und bietet mehrere Werkzeuge, um ein Netzwerk zu kompromittieren. Um beide Programme zu nutzen, wurden diese über die entsprechenden Kommandozeilen-Befehle in der Ubuntu-Maschine installiert.

Für die Simulation des Angriffs, wurde zunächst die Umgebung des Digital Twins gestartet (wie in Kapitel 3 beschrieben). Zusätzlich läuft nach dem Start eines weiteren Terminals auf dem Switch s1 das Programm Wireshark, um den kompletten Netzwerkverkehr des Angriffsvorgehens mitschneiden zu können. Als Interfaces werden die Schnittstellen aller vorhandenen Netzwerkteilnehmer in 'Capture Options' ausgewählt. Außerdem wurden zwei Terminals für den Angreifer geöffnet. Auf dem ersten Terminal wird über den Kommandozeilen-Befehl 'ettercap -G' die graphische Oberfläche von Ettercap geöffnet. Unter dem Tab 'Sniff' wird 'Unified sniffing...' ausgewählt und anschließend das attacker-Interface als Netzwerkschnittstelle gewählt. Daraufhin öffnet sich Ettercap in einer neuen Ansicht und man kann nun unter dem Reiter 'Hosts' das Netzwerk mit der Option 'Scan for hosts' nach potentiellen Angriffszielen absuchen. Unter 'Hosts>Hosts list' kann man die gescannten Hosts einsehen. Über einen linken Mausklick auf den jeweiligen Host und den Befehl 'Add to Target ...' unterhalb der Host list, kann man die Ziele für einen Angriff auswählen. Im Anschluss kann über den Reiter 'Mitm' und der Auswahl von 'ARP-Poisoning' und dem Setzen eines Hakens bei 'Sniff remote connections' die

ARP-Tabellen verändert werden. Mit dem Vertauschen der MAC-Adressen werden die Pakete die normalerweise über den Switch laufen über den Angreifer umgelenkt. Nun kann der Threat Actor die Netzwerkdaten der beiden Kommunikationspartner komplett mitlesen. Um dies auch visuell zu verdeutlichen, wird das zweite Terminal des Angreifers verwendet, um die graphische Oberfläche von Wireshark zu starten. Bevor man allerdings Wireshark öffnet, sollte zunächst überprüft werden, ob IP-Forwarding bereits eingeschaltet ist. Sollte dies nicht der Fall sein, kann man dies über die Kommandozeilenbefehle in Listing 4.1 umstellen.

```
1 cat /proc/sys/net/ipv4/ip_forward
2
3 echo 1 > /proc/sys/net/ipv4/ip_forward
```

**Listing 4.1:** IP Forwarding

Diese Einstellung dient dazu, dass alle Pakete die eigentlich nicht für den Host des Angreifers bestimmt sind, trotzdem über diesen Host weitergeleitet werden können.

### 4.3 DoS-Attacke

Als zweiter Angriff wurde eine Denial of Service Attacke (DoS) ausgewählt. Bei einer DoS handelt es sich um einen Angriff, der eine bestimmte Aktion unterbindet und somit die Funktionsfähigkeit eines Prozesses temporär einschränkt. Im Gegensatz zu einem DDoS-Angriff bei dem beispielsweise ein große Anzahl an Rechner einen Zielrechner angreifen, wird bei einem DoS-Angriff lediglich ein Ausgangsrechner für den Angriff benötigt. Ziel des Angriffs ist es einen Zielrechner mit sehr vielen Netzwerkpaketen zu fluten, sodass dieser für andere Netzwerkteilnehmer erreichbar wird. Speziell für die Angriffssimulation wurde ein SYN-Flood durchgeführt [MIT19a]. Dieser nutzt den Verbindungsaufbau des Transmission Control Protocol (TCP), wobei als erste Handlung eines Verbindungsaufbaus zwischen Client und Server ein Paket mit SYN-Flag (synchronize) gesendet wird. Im Anschluss sendet der Server eine Bestätigung des Abgleiches an den Client und dieser wiederum sollte dann die erfolgreiche Herstellung der Verbindung bestätigen. Dieser Vorgang wird allerdings für einen DoS mittels SYN-Flood misachtet und es werden so viele SYN-Anfragen in kürzester Zeit an das Angriffsziel versendet, sodass dieser sehr langsam oder gar nicht mehr auf die Flut reagieren kann. Reaktion des Angriffs ist die Überlastung und der Ausfall der jeweiligen Systeme.

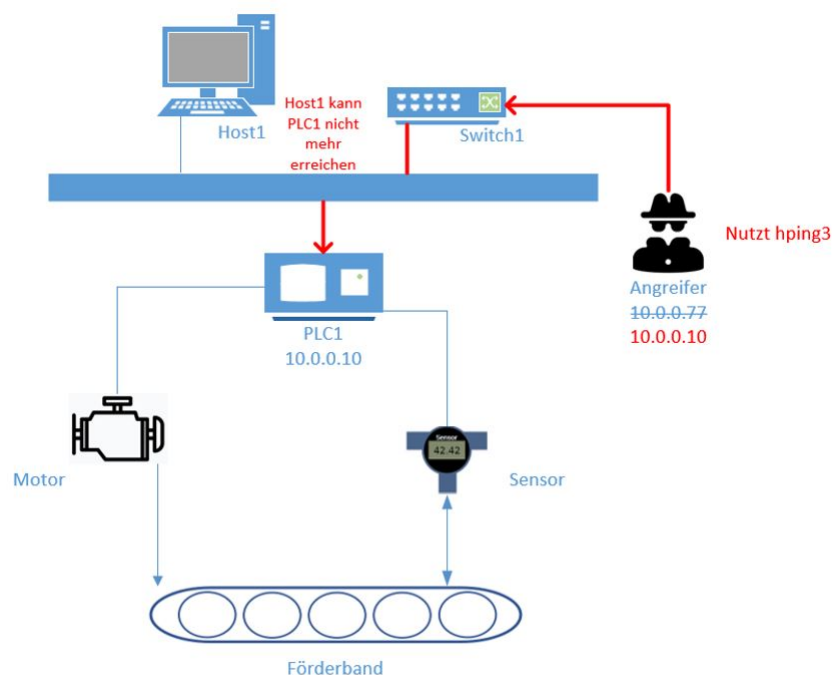
Um einen derartigen Angriff auf den Digitalen Zwilling simulieren zu können, wurde ein weiteres Linux-Programm mit dem Namen 'hping3' über die Kommandozeile in der virtuellen Maschine installiert. Dieses Programm ermöglicht es über einfache Eingabe bestimmter Einstellungen in der Kommandozeile, einen DoS zu starten [San06]. Wie in Kapitel 4.2 bereits beschrieben wird zunächst das komplette Netzwerk gestartet und der Netzwerkverkehr über das Wireshark-Programm, welches am Switch s1 Terminal gestartet wurde, aufgezeichnet. Zudem wurde wiederum ein weiteres Terminal für den

Angreifer hinzugefügt. Nun konnte über den in Listing 4.2 angegebenen Shell-Befehl der DoS-Angriff gestartet werden.

```
1 hping3 -S -a 10.0.0.10 -p 44818 --flood 10.0.0.10
```

**Listing 4.2:** Hping

Wie in Listing 4.2 angegeben, wurden mehrere Optionen für den Angriff verwendet. Die Option '-S' gibt an, dass für die TCP-Pakete das SYN-Flag gesetzt sein soll. '-a 10.0.0.10' sorgt dafür, dass die IP-Adresse der Quelle vorgetäuscht wird und hierbei wird die IP-Adresse des Angriffsziels benutzt. Die Angabe '-p 44818' soll den zu flutenden Port definieren. Der für das Ethernet/IP-Protokoll, einem Industriestandard für CPS, verwendete Port für TCP-Verkehr, lautet 44818 [IAN20].



**Abbildung 4.2:** DoS-Angriff

In Abbildung 4.2 wird noch einmal das Schema des Digital Twins dargestellt und zudem das Vorgehen des Angreifers verdeutlicht. Der Angreifer, welcher über die IP-Adresse '10.0.0.77' verfügt und sich im Netzwerk des Digitalen Zwillings befindet, schickt mittels des Befehls hping3 TCP-Pakete an den PLC, um dessen Funktionalität zu stören. Dabei täuscht er beim Versenden der Pakete eine falsche IP-Adresse vor und gibt an die IP-Adresse des PLCs zu besitzen. Während des Angriffs ist es dem Host1 nicht möglich über die HMI Befehle auszuführen und somit die Daten des Förderbandes auszulesen oder zu verändern. Im nächsten Abschnitt wird die strukturierte Aufarbeitung, der angefallenen Netzwerkmitschnitte der Angriffe thematisiert.



## Kapitel 5

# Strukturierte Aufarbeitung der Angriffsdaten

Als letzte Aufgabe des Praxisseminars stand die strukturierte Aufarbeitung von Angriffsdaten auf der Agenda. Dabei war es zunächst wichtig abzuklären, wie die Daten mitgeschnitten werden, welches Format sich zur strukturierten Darstellung von Angriffen eignet und wie diese Daten letztlich mit dem gewählten Standard aufgearbeitet werden.

### 5.1 Aufzeichnung der Daten

Wie bereits im vorherigen Kapitel beschrieben, wurde zum Aufzeichnen der Daten das Werkzeug Wireshark verwendet. Es gilt als de facto Standard für die Analyse von Netzwerkverkehr. Dabei bietet Wireshark eine GUI und mehrere Filter und Optionen zum Mitschneiden von Netzwerkverkehr [Wir20]. Ähnlich zu dem Beispiel, welches in der Arbeit von Schlette et al. beschrieben wurde, handelt es sich bei dem Förderbandsystem, dass in diesem Praxisseminar thematisiert wurde, um ein fiktives Beispiel [DFMG20, 3]. Es wird angenommen, dass die fiktive Firma bereits über ein Intrusion Detection System verfügt, welches in Bedrohungssituationen in Alarmbereitschaft übergeht und beispielsweise das Aufzeichnen von Netzwerkverkehr automatisch startet. Im Förderbandbeispiel wird daraufhin ein Wireshark Programm auf einem Terminal des Switches ausgeführt. Das Analysetool schneidet nun den kompletten Netzwerkverkehr aller Teilnehmer mit. Dieser Mitschnitt steht dann Experten zur weiteren Analyse und Verarbeitung zur Verfügung. Die Aufgabe besteht darin, die Aufzeichnung sowohl maschinen- als auch menschenlesbar zu bearbeiten.

### 5.2 STIX

Ein Format, dass sich in den letzten Jahren als State-of-the-Art Projekt für halb strukturierte Cyber Threat Intelligence (CTI) Informationen herausgestellt hat, ist STIX [FFG18, 2]. STIX kann dabei auch sogenannte cyber observable Informationen ausdrücken. Durch dieses Projekt wird ein aktives Teilen von CTI miteinander ermöglicht und damit können die

Informationen in der Cyber Security Community besser und schneller untereinander kommuniziert werden, um gegebenenfalls den Schaden von Angriffen deutlich abzumildern oder sogar zu vermeiden [Bre20, 14]. Das semi-strukturierte Format basiert auf JavaScript Object Notation (JSON) als Serialisierungssprache und stellt zwei Haupt-Objektypen zur Verfügung [DFMG20, 3]. Dies sind zum einen STIX Domain Objects (SDOs), welche die Eigenschaften einer Bedrohungslage beschreiben [Bre20, 53]. Beispielsweise stellt ein Threat Actor eine SDO dar. Eine weitere Objektypengruppe wird durch die STIX Relationship Objects (SROs) umfasst. Sie definieren die Beziehungen zwischen den einzelnen SDOs [Bre20, 125]. Als Beispiel kann das Beziehungsobjekt 'targets' genannt werden. Dies verbindet beispielsweise einen Threat Actor der auf eine bestimmte Infrastruktur (SDO: infrastructure) abzielt. Des Weiteren definiert das STIX Format sogenannte STIX Cyber Observable Objects (SCOs), welche dokumentieren, was in einem Netzwerk oder an einem bestimmten Host passiert ist [Bre20, 21]. Darunter zählt beispielsweise der Netzwerkverkehr zwischen zwei IP-Adressen, die während einer Aufzeichnung erfasst wurden. Grundsätzlich stellt STIX ein graphenbasiertes Modell dar, dass aus Knoten und Kanten besteht. Somit stellen SDOs und SCOs die Knotenpunkte und SROs die Kanten zwischen den Knoten dar. Dadurch wird auch eine visuelle Darstellung ermöglicht [Bre20, 20]. Ein beispielhaftes Vorgehen wäre, wenn das Auftreten eines Vorfalls von einem Sicherheitsexperten entdeckt werden würden. Jener diese Daten in einem STIX-Report überführt und diesen im Anschluss auf einer Plattform publiziert. Somit werden die Informationen mit anderen Organisationen und Experten möglichst schnell geteilt und gegebenenfalls weiterer Schaden eingedämmt. Zudem können die geteilten Daten zur Erstellung von Firewallregeln und für Intrusion Detection and Prevention Systems (IDPS) genutzt werden [ASN20, 6]. Neben den bereits genannten Objekten existiert für STIX außerdem ein Bundle Objekt. Dieses Objekt fungiert als Container und beinhaltet beispielsweise alle SDOs und SROs die in einer bestimmten Periode in einem Unternehmen aufgetreten sind. Die Bundles können aber auch nach anderen Kriterien strukturiert sein. STIX arbeitet zur Identifizierung seiner Komponenten mit dem Standard Universally unique identifiers (UUID) [Sal20]. Dabei wird für jedes Element in STIX-Format eine ID angefordert, sodass die Eindeutigkeit gewährleistet ist und Beziehungstypen auf die IDs der jeweiligen Objekte zugreifen können.

Laut der Arbeit von Ramsdale et al. lässt sich das STIX 2.x Format vor allem für die Analyse von Malware anwenden [ASN20, 18]. Hinsichtlich des in den vorherigen Kapiteln definierten Beispiels, wurde auf die Anwendung einer Malware verzichtet. Im folgenden Unterkapitel wird die sukzessive Aufarbeitung der gesammelten Daten anhand des Förderbandbeispiels genauer erläutert.

### 5.3 Aufarbeitung der Angriffsdaten mit STIX

Bevor die Daten in das STIX Format übermittelt werden konnten, musste entschieden werden, welche Daten für die Analyse relevant sind. Nachdem ausschließlich der Netz-

werkverkehr mitgeschnitten wurde, ist es naheliegend, die miteinander kommunizierenden IP-Adressen und deren MAC-Adressen aus den Mitschnitten herauszufiltern. Dadurch kann zunächst ein Überblick geschaffen werden, welche Verbindungen im simulierten Netzwerk bestehen und wer daran aktiv teilnimmt. Des Weiteren ist es relevant zu beurteilen, welche Protokolle für die Übertragung genutzt wurden.

Zunächst wurden die Daten der beiden Angriffssimulationen im JSON-Format abgespeichert. Als nächstes wurde beispielhaft eines der beiden JSON-Dateien geöffnet, um die Struktur der Datei zu erkennen. Im nächsten Schritt sollte die Entwicklung eines Parsers in Python dazu genutzt werden, die wichtigsten Informationen aus den JSON-Dateien herauszuarbeiten.

```

1 with open("angriff-MITM.json", "r") as f:
2     data = json.load(f)
3
4 for d in data:
5     protocol = d['_source']['layers']['frame']['frame.protocols']
6
7     if TCP_PROTO in protocol:
8         proto = 'TCP'
9         time = forttime(d['_source']['layers']['frame']['frame.time'])
10        eth_dst = d['_source']['layers']['eth']['eth.dst']
11        eth_src = d['_source']['layers']['eth']['eth.src']
12        ip_src = d['_source']['layers']['ip']['ip.src']
13        ip_dst = d['_source']['layers']['ip']['ip.dst']
14        tcp_src_port = d['_source']['layers']['tcp']['tcp.srcport']
15        tcp_dst_port = d['_source']['layers']['tcp']['tcp.dstport']
16
17        # Daten in eine Liste schreiben
18        liste.append([time, proto, eth_dst, eth_src, ip_src, ip_dst,
19                      tcp_src_port, tcp_dst_port])
20
21    elif ARP_PROTO in protocol:
22        proto = 'ARP'
23        time = forttime(d['_source']['layers']['frame']['frame.time'])
24        eth_dst = d['_source']['layers']['eth']['eth.dst']
25        eth_src = d['_source']['layers']['eth']['eth.src']
26
27        # Daten in eine Liste schreiben
28        liste.append([time, proto, eth_dst, eth_src])
29
30    else:
31        continue

```

**Listing 5.1:** Parser

Im Programmcode des Parsers in Listing 5.1 werden zunächst über einen with-Befehl Daten der Netzwerkpakete in eine Liste eingelesen. Das Übergeben der Daten erfolgte über die JSON-Funktion 'json.load()'. Danach wurde durch eine For-Schleife alle Listenpositionen nach relevanten Informationen geparkt und in jeweilige Variablen abgespeichert. Diese Variablen wurden wiederum in einer Liste hinterlegt. Zudem kann man anhand des Codes erkennen, dass durch eine If-Abfrage nach dem jeweiligen Netzwerkprotokoll unterschieden wurde. Hierbei wurde der Fokus im Rahmen des Praxisseminars

vor allem auf TCP und Address Resolution Protocol (ARP) gelegt. Die Unterscheidung nach anderen Protokollen kann natürlich je nach angefallenen Netzwerkverkehr beliebig erweitert werden. Außerdem ist eine Unterscheidung der beiden Protokolle notwendig, da beispielsweise das ARP gegenüber dem TCP nicht über eine Aufschlüsselung nach Internet Protocol (IP)-Adressen verfügt. Da das STIX-Format über einen bestimmten Zeitstempel verfügt, wurde der Zeitstempel aus der JSON-Capture mittels einer Funktion 'fortime()' angepasst.

```
1 def fortime(zeit):
2     year = zeit[8:12]
3     month = getMon(str(zeit[0:9]))
4     day = zeit[4:6]
5     std = str(int(zeit[13:15]) - 2)
6     m = zeit[16:18]
7     sec = zeit[19:25]
8
9     s = str(year) + "-" + str(month) + "-" + str(day) + "T" + str(std) + ":" +
        str(m) + ":" + str(sec) + "Z"
10
11     return s
```

**Listing 5.2:** Anpassung Zeitstempel

Wie man anhand von Listing 5.2 sieht, empfängt die Funktion einen String und speichert einzelne Elemente des Strings in Variablen ab und setzt diese am Ende wieder zu einem String im Zeitformat von STIX zusammen. Das richtige Format wird im Anschluss wieder an eine String-Variable in der Ursprungsfunktion übergeben.

Nachdem die wichtigsten Informationen aus jedem Paket herausgefiltert wurden, konnte damit begonnen werden die gesammelten Daten zu analysieren. Um beispielsweise die einzelnen redundant abgespeicherten Verbindungen aus der Liste herauszufiltern, wurde eine weitere Funktion 'getCon()' geschrieben, die die ursprüngliche Liste aller Pakete empfängt. In Listing 5.3 wird ein Teil der Funktion 'getCon()' anhand des Beispiels bei TCP-Kommunikationsverbindungen dargestellt. Vorab werden verschiedene Listen initialisiert, die nacheinander mit neu auftretenden Source- bzw. Destinationverbindungen befüllt werden. Daraufhin wird jedes Paketelement nach dem jeweiligen Vorkommen überprüft. Sollte eine Verbindung zwischen zwei Elementen bestehen, werden die Elemente beider Verbindungen in eine weitere Liste aufgenommen. Nun wird jedes Paket nach dem Vorkommen einer bereits bestehenden Verbindung untersucht. Ist die Verbindung in der Liste bereits vorhanden, wird das Paket verworfen. Ist dem nicht der Fall, wird diese Verbindung wiederum der Kombinationsliste angehängt. In einer weiteren For-Schleife werden die Inhalte der Liste der Kombinationen mit der ursprünglichen Liste aller Pakete abgeglichen und mit einer Counter-Variable abgezählt. Der Wert des Zählers wird im Anschluss der jeweiligen Sender-/Empfängerkombination angehängt. Dadurch wird es einem Analysten beispielsweise ermöglicht, abzuprüfen, in welchem Zeitumfang die gleichen Verbindungen gezählt wurden. Dies könnte beispielsweise bei einem vorherrschenden SYN-Flood auf eine DoS hinweisen. Außerdem wird überprüft, ob eine

Media Access Control (MAC)-Adresse in zwei verschiedene IP-Adressen aufgeschlüsselt werden kann. Ist dies der Fall, wird der Liste der Verbindungen eine weitere IP-Adresse angehängt. Dies könnte auf einen ARP-Poisoning Angriff hinweisen und eine mögliche MITM-Attacke implizieren.

```
1 elif str(e[1]) == "TCP":
2     if e[3] in list_Con_TCP_src:
3         if e[2] in list_Con_TCP_dst:
4             for i in list_Con_TCP:
5                 if (str(e[3]) == str(i[3])) and (str(e[2]) == str(i[2])):
6                     if str(e[4]) != str(i[4]):
7                         i.append(e[4])
8                     elif str(e[5]) != str(i[5]):
9                         i.append(e[5])
10                    else:
11                        continue
12                else:
13                    continue
14            else:
15                list_Con_TCP_dst.append(e[2])
16                list_Con_TCP.append(e)
17        else:
18            if e[2] in list_Con_TCP_dst:
19                list_Con_TCP_src.append(e[3])
20                list_Con_TCP.append(e)
21            else:
22                list_Con_TCP_dst.append(e[2])
23                list_Con_TCP_src.append(e[3])
24                list_Con_TCP.append(e)
```

**Listing 5.3:** Verbindungen am Beispiel TCP

Die wichtigsten Informationen werden mit Hilfe dieser ausführlichen Kombinationsliste an eine Variable in Form einer Liste zurückgegeben. In einem weiteren Schritt können diese Daten in eine Datei geschrieben und abgespeichert werden. Zudem können die Daten an eine Funktion übergeben werden, die die Informationen aus der JSON-Datei in STIX-Objekte umwandelt. Ein Beispiel eines solchen Bundle-Creators kann als Python-Producer auf der Webseite von STIX im Bereich 'Examples' gefunden werden [OAS20]. Python hat ein Modul für STIX 2.x und eignet sich daher sehr gut, Daten in dieses Format zu überführen.

Für den Angriff auf das Förderbandbeispiel, wurden drei IP-Adressen und die drei jeweils dazugehörigen MAC-Adressen aus den Angriffsdaten herausgearbeitet. Wenn man bei diesem fiktiven Beispiel davon ausgehen würde, dass ein Netzwerkanalyst darüber Bescheid weiß, welche Adressen für den eigentlichen Prozessablauf verantwortlich sind. Könnte dieser Analyst wiederum darauf schließen, dass die dritte Adresse eventuell für eine Störung des Betriebsablaufs verantwortlich sein könnte. Dieser hat nun durch das Parsen des Netzwerkverkehrs und der daraus übersichtlichen Verbindungsliste die Möglichkeit, die Kommunikationsbeziehungen eines Threat Actors genauer unter die Lupe zu nehmen. In der Verantwortung jenes Netzwerkspezialisten liegt es, die Informationen des Angriffs so in das STIX-Format zu überführen, dass diese Daten wie anfangs beschrieben, sowohl

in betriebseigenen Systemen, als auch in der Cybersecurity-Community geteilt werden können. Auf Basis dieser Informationen können dann gegebenenfalls Schwachstellen schneller identifiziert und behoben werden.

Im folgenden STIX-Code wird ein Teil, der im Rahmen des Praxisseminars simulierten Angriffsdaten dargestellt. In 5.4 kann man erkennen, wie ein Bundle Objekt definiert wird und dieses Bundle eine ID im UUID-Format erhält. Daraufhin folgen die Objekte des Bundle-Containers. Im ersten Beispiel Abschnitt kann man ein Objekt vom Typ Attack-Pattern erkennen. Dieses verweist mittels einer externer Referenzadresse auf eine Internetseite, die diesen Angriffstyp genauer erläutert.

```

1  "type": "bundle",
2  "id": "bundle--0d08b480-c94a-4a8f-978f-637b60b4326c",
3  "objects": [
4    {
5      "type": "attack-pattern",
6      "spec_version": "2.1",
7      "id": "attack-pattern--1c914568-3b9a-492f-86c5-ef394def6839",
8      "created": "2020-07-21T17:23:20.195Z",
9      "modified": "2020-07-21T17:23:20.195Z",
10     "name": "Man in the Middle Attack",
11     "external_references": [
12       {
13         "source_name": "capec",
14         "url": "https://capec.mitre.org/data/definitions/94.html",
15         "external_id": "CAPEC-94"
16       }
17     ]
18   }

```

**Listing 5.4:** STIX Beispiel 1

In Listing 5.5 wird beispielhaft eine SRO Relationship schematisch dargestellt. Unter dem Schlüssel created wurde die Zeit hinterlegt, wann die Beziehung vermutlich initial aufgetreten ist. Dabei wurde die entsprechende Zeit aus dem Angriffschnitt an das STIX-Objekt übergeben. Zudem kann man in der Beziehung den Verweis auf Quelle und Ziel erkennen. Das komplette Bundle kann im Ordner STIX im Repository unter folgendem Github-Link eingesehen werden: <https://github.com/EnjoyFitness92/Praxisseminar-SS2020>

```

1  {
2    "type": "relationship",
3    "spec_version": "2.1",
4    "id": "relationship--19c57d00-d18e-4187-821b-2c49188bce1b",
5    "created": "2020-07-21T19:53:51.14795Z",
6    "modified": "2020-07-21T19:53:51.14795Z",
7    "relationship_type": "attributed-to",
8    "source_ref": "threat-actor--1d9986b7-63a3-47db-bfea-2a3def1068b1",
9    "target_ref": "identity--35968baa-e4d5-4bbe-a3e5-6c10109a8f4a"
10  }

```

**Listing 5.5:** STIX Beispiel 2

Im nun folgenden letzten Kapitel wird ein Fazit zum Praxisseminar gezogen.

## Kapitel 6

### Fazit

Zum Ende hin muss hervorgehoben werden, dass die beiden Hauptthemen, der Simulation eines Digitalen Zwillings und die strukturierte Aufarbeitung von Angriffsdaten, sehr umfangreich sind und diese Arbeit keinen Anspruch auf Vollständigkeit erhebt. Aufgrund des eingeschränkten zeitlichen Rahmens konnte nur ein Teilbereich jener Themen herausgearbeitet werden. Fakt ist, dass die Entwicklung von Digital Twins im weiteren Verlauf der Entwicklung der Industrie 4.0 weiter voranschreiten wird. Zudem bietet dieses Schema noch sehr viel Spielraum, da viele Konzepte bisher eher theoretischer Natur sind und eigentlich für jede industrielle Anlage ein spezielles Digitales Abbild entwickelt werden muss. Die strukturierte Aufarbeitung von Angriffsdaten, insbesondere mit dem Standard STIX, ermöglicht es Securityexperten weltweit auf einer Ebene miteinander zu arbeiten. Hierbei ist es wichtig, dass das Format von sehr vielen Organisationen in einem einheitlichen Maß gestaltet und genutzt wird. Nur so können immer komplizierter gestaltete Angriffswege verstanden und gegebenenfalls auf Basis der gewonnen Informationen Gegenmaßnahmen getroffen werden.

Abschließend lässt sich erwähnen, dass die Arbeit an dem Praxisseminar zeitweise sehr mühsam war, aber die Anstrengungen, sich in bestimmte Themen intensiv einzuarbeiten, sich in jedem Fall gelohnt haben. Es wurden beispielsweise vor allem Programmierkenntnisse in Python hinzugewonnen, ein besseres Verständnis für Netzwerke und Angriffsvektoren erzielt und der Umgang mit verschiedenen Softwaretools wie zum Beispiel Wireshark vertieft. Ein großer Dank gilt auch den Betreuern Marietheres Dietz und Daniel Schlette.

# Literaturverzeichnis

- [Ant17] ANTONIOLI, Daniele: *minicps*. <https://minicps.readthedocs.io/en/latest/?badge=latest>. Version: 2017
- [ASN20] ANDREW RAMSDALE ; STAVROS SHIAELES ; NICHOLAS KOLOKOTRONIS: A Comparative Analysis of Cyber-Threat Intelligence Sources, Formats and Languages. In: *Electronics* 9 (2020), Nr. 5, S. 824. <http://dx.doi.org/10.3390/electronics9050824>. – DOI 10.3390/electronics9050824
- [AT] ANTONIOLI, Daniele ; TIPPENHAUER, Nils O.: MiniCPS: A toolkit for security research on CPS Networks.
- [Bre20] BRETT JORDAN, RICH PIAZZA, TREY DARLEY: *StixTM version 2.1. Committee Specification 01*. <https://docs.oasis-open.org/cti/stix/v2.1/cs01/stix-v2.1-cs01.html>. Version: 2020
- [BSI19] BSI: *Industrial Control System Security. Top 10 Bedrohungen und Gegenmaßnahmen*. [https://www.allianz-fuer-cybersicherheit.de/ACS/DE/\\_/downloads/BSI-CS/BSI-CS\\_005.pdf?\\_\\_blob=publicationFile&v=12#download=1](https://www.allianz-fuer-cybersicherheit.de/ACS/DE/_/downloads/BSI-CS/BSI-CS_005.pdf?__blob=publicationFile&v=12#download=1). Version: 2019
- [DFMG20] DANIEL SCHLETTE ; FABIAN BÖHM ; MARCO CASELLI ; GÜNTHER PERNUL: Measuring and visualizing cyber threat intelligence quality. In: *International Journal of Information Security* (2020). <http://dx.doi.org/10.1007/s10207-020-00490-y>. – DOI 10.1007/s10207-020-00490-y
- [FFG18] FABIAN BÖHM ; FLORIAN MENGES ; GÜNTHER PERNUL: Graph-based visual analytics for cyber threat intelligence. In: *Cybersecurity* 1 (2018), Nr. 1. <http://dx.doi.org/10.1186/s42400-018-0017-4>. – DOI 10.1186/s42400-018-0017-4
- [Hoc20] HOCHSCHULREKTORENKONFERENZ: *Module, ECTS-Punkte und Workload*. <https://www.hrk-nexus.de/themen/studienqualitaet/ects-und-kreditpunkte/module-ects-punkte-und-workload/>. Version: 2020
- [IAN20] IANA: *Service Name and Transport Protocol Port Number Registry*. <https://www.iana.org/assignments/service-names-port-numbers/service-names-port-numbers.xhtml?&page=132>. Version: 2020



- [Len20] LENA BODEWEIN: *Australien meldet massiven Cyberangriff*. <https://www.tagesschau.de/australien-cyberangriffe-101.html>. Version: 2020
- [Mar20] MARTIN HOLLAND: *Während Coronavirus-Pandemie: Cyberangriff legt tschechisches Krankenhaus lahm*. <https://www.heise.de/security/meldung/Waehrend-Coronavirus-Pandemie-Cyberangriff-legt-tschechisches-Krankenhaus-lahm-4683370.html>. Version: 2020
- [MBG19] MARIETHERES DIETZ ; BENEDIKT PUTZ ; GÜNTHER PERNUL: A Distributed Ledger Approach to Digital Twin Secure Data Sharing. Version: 2019. [http://dx.doi.org/10.1007/978-3-030-22479-0\\_{\\_}15](http://dx.doi.org/10.1007/978-3-030-22479-0_{_}15). In: *Data and Applications Security and Privacy XXXIII*. Springer International Publishing, 2019. – DOI 10.1007/978-3-030-22479-0\_15, S. 281–300
- [Min18] MININET: *Mininet*. <http://mininet.org>. Version: 2018
- [MIT19a] MITRE: *CAPEC-482: TCP Flood*. <https://capec.mitre.org/data/definitions/94.html>. Version: 2019
- [MIT19b] MITRE: *CAPEC-94: Man in the Middle Attack*. <https://capec.mitre.org/data/definitions/94.html>. Version: 2019
- [OAS20] OASIS: *Identifying a Threat Actor Profile*. <https://oasis-open.github.io/cti-documentation/examples/identifying-a-threat-actor-profile>. Version: 2020
- [Ola18] OLAF SAUER: Digitale Zwillinge existieren für die wenigsten Maschinen und Anlagen. In: *Automobil Produktion* (2018), Nr. 4, 6–7. [https://www.iosb.fraunhofer.de/servlet/is/80212/Automobil%20Produktion%20Meinungsmacher%2008\\_2018.pdf?command=downloadContent&filename=Automobil%20Produktion%20Meinungsmacher%2008\\_2018.pdf](https://www.iosb.fraunhofer.de/servlet/is/80212/Automobil%20Produktion%20Meinungsmacher%2008_2018.pdf?command=downloadContent&filename=Automobil%20Produktion%20Meinungsmacher%2008_2018.pdf)
- [Pet20] <https://www.security-insider.de/cyberangriffe-folgen-der-entwicklung-von-covid-19-a-926037/>
- [Sal20] SALZ, P Leach; M Mealing; R.: *A Universally Unique Identifier (UUID) URN Namespace*. <https://tools.ietf.org/html/rfc4122>. Version: 2020
- [San06] SANFILIPPO, Salvatore: *hping*. <http://www.hping.org/>. Version: 2006
- [Wir20] WIRESHARK: *Wireshark*. <https://www.wireshark.org/>. Version: 2020