Zürcher Hochschule
für Angewandte Wissenschaften

School of Engineering

# Projektarbeit (IT)

# Machine Learning Driven Control of Moving Target Defense (MTD) for Cybersecurity

| | |
|---|---|
| **Autoren** | Levi Cailleret<br>Sascha Kyburz |
| **Hauptbetreuung** | Dr. Gürkan Gür<br>Prof. Dr. Bernhard Tellenbach |
| **Datum** | 19.12.2020 |

# Formular

## Eigenständigkeitserklärung

Mit der Abgabe dieser Projektarbeit versichert der/die Studierende, dass er/sie die Arbeit selbständig und ohne fremde Hilfe verfasst hat. (Bei Gruppenarbeiten gelten die Leistungen der übrigen Gruppenmitglieder nicht als fremde Hilfe.)

Der/die unterzeichnende Studierende erklärt, dass alle zitierten Quellen (auch Internetseiten) im Text oder Anhang korrekt nachgewiesen sind, d.h. dass die Projektarbeit keine Plagiate enthält, also keine Teile, die teilweise oder vollständig aus einem fremden Text oder einer fremden Arbeit unter Vorgabe der eigenen Urheberschaft bzw. ohne Quellenangabe übernommen worden sind.

Bei Verfehlungen aller Art treten die Paragraphen 39 und 40 (Unredlichkeit und Verfahren bei Unredlichkeit) der ZHAW Prüfungsordnung sowie die Bestimmungen der Disziplinarmassnahmen der Hochschulordnung in Kraft.

| | | | |
|---|---|---|---|
| Vorname, Name: | Levi Cailleret | | |
| Ort, Datum: | Bütschwil, 18.12.2020 | Unterschrift: |  |

| | | | |
|---|---|---|---|
| Vorname, Name: | Sascha Kyburz | | |
| Ort, Datum: | Kreuzlingen, 18.12.2020 | Unterschrift: |  |

# Zusammenfassung

Diese Arbeit beschäftigt sich mit Moving Target Defense (MTD) im Zusammenhang mit Reinforcement Learning. Die Asymmetrie zwischen Angreifern und Verteidigern ist ein großes Problem beim Schutz von Informationssystemen. Angreifer müssen nur eine einzige Schwachstelle finden, um erfolgreich zu sein. Verteidiger müssen alle Schwachstellen verschliessen, um sicher zu sein. Moving-Target-Defense (MTD) macht die Identifikation und Ausnutzung solcher Schwachstellen schwierig, da MTD den Angreifern ein sich ständig veränderndes Informationssystem präsentiert. Der deutliche Schwachpunkt der MTD ist die Entscheidung zu fällen, wann welche Aktion zur Abwehr sinnvoll und richtig ist.

Ein möglicher Lösungsansatz zum automatisierten Finden der richtigen MTD Reaktion könnte der Einsatz von Reinforcement Learning und passenden Angreifer- und Infrastrukturmodellen sein.

Dass dies bisher kaum untersucht wurde, zeigt unser erster Beitrag, ein kurzer Überblick über den Stand der Technik und die grundlegende Kategorisierung von MTD Ansätze. Das eine eingehendere Untersuchung angebracht ist, zeigt unser zweiter Beitrag. Mittels eines MTD Versuchsaufbaus aus existierenden Arbeiten vermessen wir die Performance von statischen, und auf Reinforcement Learning basierenden MTD Ansätzen und stellen fest, dass zweitere deutlich besser abschneiden. Unsere Messungen zeigen aber auch, dass dies wahrscheinlich mit einem etwas höheren Ressourcenverbrauch erkauft werden muss. Ob sich ein Einsatz in der Praxis lohnt, kann nicht abschliessend geklärt werden und muss noch weiter untersucht werden.

## Abstract

This thesis is about Moving Target Defense (MTD) in connection with reinforcement learning. The asymmetry between attackers and defenders is a major problem in protecting information systems. Attackers only need to find a single vulnerability to be successful. Defenders need to find any vulnerabilities to be safe. Moving Target Defense (MTD) makes the identification and exploitation of such vulnerabilities difficult, since MTD presents the attacker with a constantly changing information system. The weak point of the MTD is the decision making, the question which action should be taken at which time to defend against an attacker.

A possible solution for the automated selection of the correct MTD reaction could be the use of reinforcement learning and suitable attacker and infrastructure models.

Because this has hardly been investigated so far, our first chapter shows a brief overview of the state of the art and the basic categorization of MTD approaches. Our second chapter shows that a more detailed investigation is appropriate, using an MTD test setup from existing work. We measure the performance of static and reinforcement learning based MTD approaches and find that the latter performs significantly better. However, our measurements also show that this probably has a higher consumption of resources. Whether it is worthwhile to use it in practice cannot be determined and must be investigated further.

# Inhaltsverzeichnis

# 1 Einleitung

## 1.1 Ausgangslage

Bis anhin hatten Angreifer auf ein System einen bedeutenden Vorteil gegenüber den Verteidigern. Dieser Vorteil entsteht daraus, dass ein Sicherheitssystem meist statisch ist. Ein System wird geplant, entwickelt, getestet und irgendwann eingesetzt. Nach dem Einsetzen (dem Deployment) ändert sich das System nur sporadisch, immer dann, wenn Aktualisierungen stattfinden. Die Zeit ist hierbei auf der Seite des Angreifers. Er kann das System untersuchen, Schwachstellen ausmachen und mögliche Angriffe entwickeln, die diese ausnutzen, und schlussendlich zuschlagen. Der Verteidiger hingegen muss einen Angriff erst erkennen, eine Gegenmassname beschliessen, und diese in Form einer Aktualisierung in das angegriffene System bringen. In dieser Zeit kann der Angreifer bereit massive Übergriffe gestartet und sensible Daten entwendet haben.

Um diesem Nachteil entgegen zu wirken, werden Verteidigungsstrategien getestet, die auf Moving Target Defense, hier mit MTD abgekürzt, basieren. Dieser Name beschreibt sinnbildlich auch die Funktionsweise einer MTD. Die Idee ist, dass durch eine Verschiebung von Ressourcen (z.B. eines Services auf ein andere IP-Adresse) oder die Wandlung der Ressource selbst (z.B. des Codes einer Applikation oder der unterliegenden Plattform) die gesammelten Informationen eines Angreifers nur für einen kurzen Zeitraum aktuell sind. Eine potenzielle Schwachstelle kann so «verschoben» oder ganz obsolet werden, so dass ein möglicher Angriff den Angriff erst neu koordinieren muss. Dies gleicht Verteidiger und Angreifer aneinander an, da sie nun beide gezwungen sind dem Faktor Zeit noch mehr Bedeutung zu schenken.

Bei einer MTD stellen sich grundsätzlich drei Fragen. (1) Was alles bewegt werden kann in einem System, (2) wann eine Bewegung stattfinden soll und schliesslich, (3) wie man sich für eine bestimmte Bewegung entscheidet. Bis anhin gibt es noch sehr wenige Ansätze, die versuchen, mit künstlicher Intelligenz diese Aspekte anzugehen. Unser Ziel ist es, Lösungen für diese Fragen mit Machine Learning zu finden und zu optimieren.

## 1.2 Zielsetzung

Das Ziel dieser Arbeit ist es, einen grundlegenden Überblick über das Thema Moving Target Defense zu geben und zu untersuchen, wie Machine Learning in Kombination mit MTD eingesetzt werden kann, um ein Netzwerk zu verteidigen. Auf den Grundlagen der theoretischen Recherche wird ein eigener Ansatz entworfen, getestet, und ausgewertet. Für das Testen und Auswerten werden geeignete Frameworks und Tools eingesetzt.

Unser Ansatz soll nach Sicherheits- und Leistungsaspekten bewertet werden. Dazu werden als Vergleich ein statisches System und eine naive MTD entworfen. Unser Ansatz soll sich von der naiven MTD in dem Aspekt unterscheiden, wann eine Bewegung stattfinden soll und wie man sich für eine bestimmte Bewegung entscheiden soll. Schlussendlich wird ein Vergleich zwischen statischen MTD Ansatz und dem Reinforcement Learning basierten Ansatz gezogen.

# 2 Theoretische Grundlage

## 2.1 Grundlegendes und Klassifikation des Modells

### 2.1.1 Grundlegende Idee von MTD

Das Ziel von Moving Target Defense (MTD) ist es, kontinuierlich Komponenten (Zum Beispiel Ports) eines Systems in einer zufälligen Art so zu bewegen, dass die Angriffsfläche (Attack Surface) minimiert oder in einem Zustand der ständigen Wandlung gehalten wird. Dies stellt sicher, dass gesammelte Informationen eines Angreifers über ein verteidigtes System seltener genutzt werden können. Im Vergleich zu statischen Verteidigungen sind MTDs flexibel und können mit Hilfe einer entsprechenden Policy während der Laufzeit mehr Einfluss auf das Geschehen nehmen als statische Verteidigungssysteme (siehe Abbildung 1). Es kostet den Angreifer so deutlich mehr Ressourcen, schlussendlich in ein System einzudringen [1]. MTDs stellen grundsätzlich drei Fragen: (1) Was alles bewegt werden kann in einem System "What to move", (2) wann eine Bewegung stattfinden soll "When to move" und schliesslich, (3) wie man bewegt "How to move" [2].

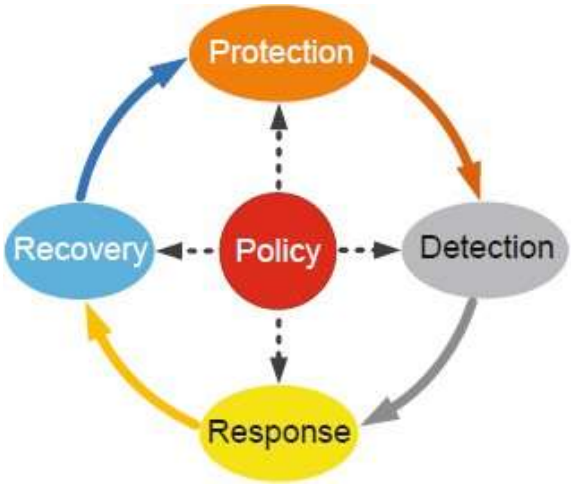



*Abbildung 1: Funktionsweise MTD Abwehr [3]*

### 2.1.2 Klassifikationen von MTD Modellen

Das Forschungsfeld MTD hat einige Klassifikationen von MTD-Modellen hervorgebracht. Viele dieser Klassifikation nutzen ihre eigenen Terme. Trotzdem gibt es einige Gemeinsamkeiten, die darauf schliessen lassen, dass wichtige Charakteristiken von MTDs immer dieselben sind. Daraus lässt sich ableiten, was zu einer guten MTD gehört. Die folgenden Unterpunkte sind Cai et all entnommen [1].

**Multi-Candidate:** Multi-Candidate meint das Prinzip, nach dem jede MTD einen gewissen Spielraum hat, welche Komponenten sie verschiebt/auswechselt. Beim IP-Shuffling meint das beispielsweise die verschiedenen Werte, die eine IP-Adresse oder ein Port annehmen kann. Die MTD braucht also ein Set von Komponenten, die sie nutzen kann, um Angreifer in die Irre zu führen. Hier unterscheiden sich die verschiedenen MTDs dadurch welche Art von Multi-Candidate sie verwenden. Es kann sich wie beschrieben um Ports, IP-Adressen, aber auch VMs und ganze Betriebssysteme handeln ("What to move").

**Diversity:** Diversity nutzt Multi-Candidate als Basis. Diversity meint, dass die Komponenten, die eine MTD verschieben kann, möglichst verschiedene Implementationen besitzen müssen, um dieselbe Aufgabe zu erfüllen. Je verschiedener die Implementationen zweier Dienste sind, die verschoben werden, desto besser. Hier unterscheiden sich die verschiedenen MTD-Methoden dadurch, wie weit voneinander entfernt Implementationen für einen Dienst sind ("What to move").

**Randomness:** Ein wichtiges Ziel für eine MTD ist es, möglichst unvorhersehbar handeln zu können, damit ein Angreifer ihre Schritte nicht voraussehen kann. Um dies zu erreichen, setzt man meist auf Pseudozufälligkeit. Der Unterschied zwischen verschiedenen MTDs ist nicht das Einsetzen von Zufälligkeit, sondern wie sie diese erreichen wollen ("How to move").

**Limited Timliness:** Limited Timeliness meint das Prinzip, wonach gesammelte Informationen, die ein Angreifer erlangt hat, nur für eine bestimmte Zeit einen Nutzen haben. Dies ist einer der Punkte, die MTDs effektiv machen. Sobald die Möglichkeit besteht, dass Informationen über ein Netzwerk ihren Wert verlieren, steht der Angreifer unter Zeitdruck. Hier unterscheiden sich die MTD-Techniken danach wie und was für ein Zeitintervall festgelegt wird ("When to move").

### 2.1.3 Angriffsoberfläche

Die Angriffsoberfläche (Attack Surface) wird in der Fachliteratur oftmals nicht präzise definiert, da eine Definition, was Komponenten von MTD-Systemen angeht, schwer ist. Das Forschungsfeld der Moving Target Defense befindet sich durch noch laufende Forschung im Wandel, so dass Definitionen und Herangehensweisen ständigen neu betrachtet werden müssen. In den meisten Publikationen meint die Angriffsoberfläche die Komponenten eines Systems, auf die von ausserhalb des Systems zugegriffen werden kann und damit angreifbar sind. Diese Schwächen sind sowohl Angreifern als auch Verteidigern bekannt und können im Verlauf eines Angriffs vom Verteidiger verschoben werden, indem dieser zum Beispiel einzelne Komponenten aber nicht die Abstraktion ändert (siehe Abbildung 2). Das Attack Surface könnte beispielsweise ein Codefragment beinhalten, welches mittels einer Funktion einen Userinput annimmt. Das spezielle Verhalten dieser Funktion innerhalb seiner Implementation kann angegriffen werden. Die verteidigende MTD kann diese Funktion schützen, indem sie laufend die Implementieren dieser Funktion auswechselt.

Den Versuch einer Definition leisten Pratyusa Manadhata und Jeannette M. Wing in «An Attack Surface Metric» [3]. Ein Angreifer wird immer versuchen, einen Angriff auf ein System über Aktionen jenes Systems oder über seine Ressourcen zu beginnen. Grundsätzlich kann jede Aktion eines Systems und jeder Ressource zu der Angriffsoberfläche dazugezählt werden. Es macht aber wenig Sinn sämtliche Ressourcen oder Aktionen eines Systems bei der Definition der Angriffsoberfläche gleichwertig zu berücksichtigen. Auswertungen von Angriffen auf Linux- und Windows-Systeme über eine längere Zeitspanne zeigen häufigste Sicherheitsrisiken und Angriffsmuster. Es macht also Sinn Aktionen und Ressourcen zu gewichten. System – Aktionen und -Ressourcen, gewichtet nach ihrer Angreifbarkeit und Wichtigkeit für das System, stellen zusammengefasst die Angriffsoberfläche nach Manadhata und Wing dar [3].

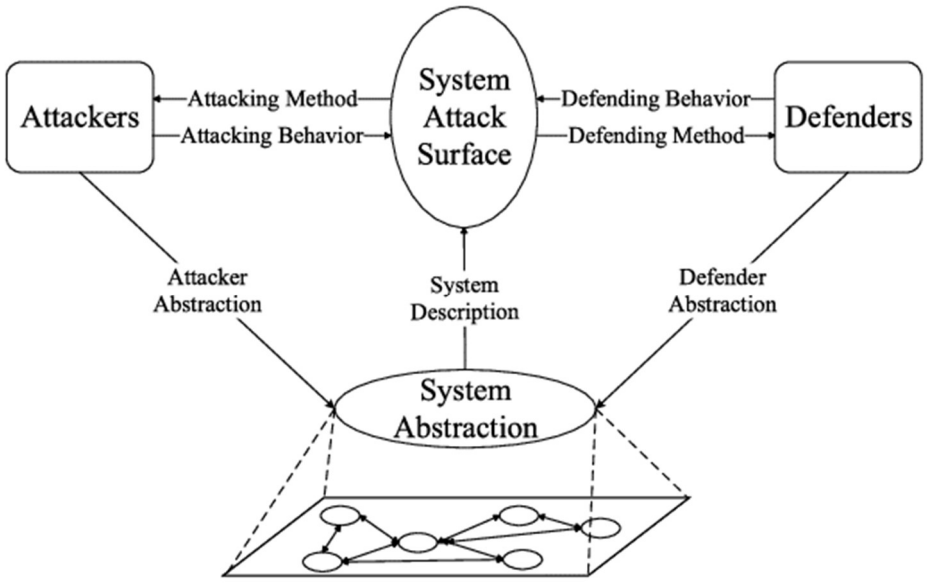



*Abbildung 2: Attack Surface zwischen Angreifer und Verteidiger [6]*

### 2.1.4 Aspekte von MTDs

Cho et al. und Cai et al. definieren in "Toward Proactive, Adaptive Defense: A Survey on Moving Target Defense" [4] und "Moving target defense: state of the art and characteristics" [1] verschiedene Unterkategorien von MTDs. Die vorgeschlagene Kategorisierung setzt sich zusammen aus der Zeitkomponente "Wann soll eine Bewegung stattfinden" (When to move), der Entscheidungskomponente "Wie soll bewegt werden?" (How to move) und dem Gegenstand "Was soll bewegt werden? " (What to move).

**Zeitkomponente (When to move):**

**Time-Based:** Diese Art der MTD löst eine Änderung im System in einem bestimmten, vordefinierten Zeitintervall aus. Dabei gilt es vor allem auf das dafür genutzte Intervall zu achten. Sollte die Intervallzeit zu lang sein, ist es für den Angreifer eventuell möglich das Fenster auszunutzen und eine vollständige Attacke zwischen zwei Intervallen auszuführen. Sollte die Intervallzeit zu kurz gesetzt sein, arbeitet die MTD, ohne dabei einen Angriff abzuwehren und verbraucht unnötig Ressourcen, was ihre Effizienz schwächt. Hier hat das Zeitintervall einen signifikanten Einfluss darauf, wie effektiv die MTD ist.

Event-Based: Diese Art der MTD löst eine Änderung im System aus, sollte ein bestimmtes Ereignis eintreten. Für gewöhnlich soll das System reagieren, wenn ein Angriff oder Zeichen eines Angriffs registriert wurden, oder aber das System genau registrieren kann, was der Angreifer versucht. Der Nachteil an dieser Variante der MTD ist, dass Angriffe erst korrekt erkannt werden müssen. Für die Erkennung könnte in Zukunft Machine Learning mit Game Theory und Control Theory wichtig werden.

**Hybrid:** Unter diese Art der MTD fallen alle möglichen Mischarten vom MTDs, die sowohl eine zeitliche als auch eine eventbasierte Komponente nutzen.

**Entscheidungskomponente (How to move):**

**Shuffling:** Diese Technik bewegt oder randomisiert Systemkonfigurationen. Dazu zählen zum Beispiel IP-Adressen auf dem TCP/IP-Layer und die Anpassung der Migrationszeit auf eine VM. Diese Technik dient vor allem dazu, den Angreifer zu verwirren und die Erkennung von lohnenswerten Zielen innerhalb des Systems hinauszuzögern. Damit verbraucht der Angreifer Ressourcen ohne grossen Gewinn und wird länger aufgehalten. Dies erhöht die Chance, dass das System auf einen Angriff entsprechend reagieren kann bevor eine Sicherheitslücke ausgenutzt wird.

**Diversity:** Das Ziel dieser Technik ist es, dass weitere Funktionieren des Systems zu gewährleisten, selbst wenn ein Angreifer teilweise Zugang erhalten hat. System-Komponenten sollen verschieden implementiert verfügbar sein, so dass das System eine Implementation einer Komponente gegen eine andere austauschen kann. Dadurch ist es möglich, dass das System weiter funktioniert, jedoch nicht dieselben Angriffsstellen wie die letzte Konfiguration bietet.

**Redundancy:** Das Ziel dieser Technik ist es, Redundanz zu erzeugen, so dass beim Ausfall bestimmter Dienste Replikate dieser Dienste benutzt werden können, um das Funktionieren des Systems sicherzustellen. Zum Beispiel kann man mehrere gleiche Wege zwischen Knoten im Network-Layer erstellen, so dass gewisse Wege ausfallen dürfen, ohne das System zu gefährden [4].

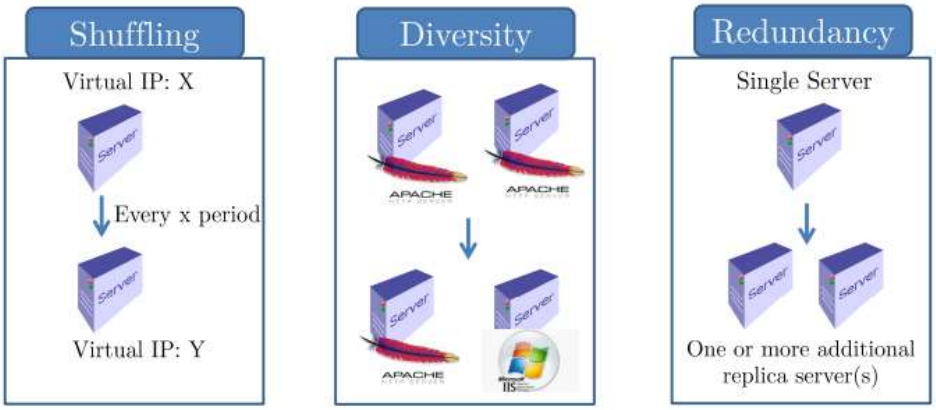



*Abbildung 3: How to move, drei grundlegende Arten [7]*

**Gegenstand (What to move):**

**Software Transformation:** Die Softwaretransformation nutzt die Applikation/die Software als Gegenstand, der verschoben werden soll. Diese wird so verändert, dass der Angreifer zu verschiedenen Zeitpunkten mehreren Implementationen ausgesetzt ist. Dabei kann Diversity eingesetzt werden, um verschiedene Implementationen zu generieren. Hier können ebenfalls verschiedene Methoden eingesetzt werden. Es ist möglich Diversity der Softwaretransformation über eine Zufallsvariable oder auch nach praktischen Kriterien herzustellen.

**Dynamic Platform Techniques**: Die Plattformtechniken nutzen die Ausführungsumgebungen, also ein zugrundeliegendes Betriebssystem oder Datenbanksystem, als Gegenstand, der verschoben wird. Das beinhaltet auch die Laufzeitumgebung und ihre Konfiguration. Um einen Wechsel zwischen zwei Plattformen zu gewährleisten, müssen mehrere Plattformen für das System verfügbar sein, was wiederum dem Prinzip der Redundancy entspricht. Die Unterschiede zwischen den Laufzeitumgebungen können variieren. Es kann sich um verschiedene Versionen desselben Betriebssystems handeln oder auch um ein komplett anderes Betriebssystem, z.B. von Windows zu Ubuntu. Durch das Wechseln der Laufzeitumgebung ist der Angreifer immer wieder vor eine neue Ausgangslage gestellt, so dass er seine Angriffe neu koordinieren muss.

**Network Address Shuffling:** Diese Methode benutzt das Mapping zwischen Netzwerkadressen und entsprechenden Geräten/Komponenten als Gegenstand, der verschoben wird. Netzwerkadressen sind eine Kombination aus IP und Informationen aus dem Transport Layer. Beide dieser Komponenten können für das Vertauschen, das Shuffling, genutzt werden. Das Wechseln der Adressen wird einen Angreifer bei mehrmaligem Zugriff immer wieder an eine andere Komponente senden, so dass er schnell den Überblick verliert und zunehmend Ressourcen bei der Anpassung seiner Angriffe einsetzen muss. Bei der Entscheidung, welche Adressen getauscht werden sollen, und zu welchem Zeitpunkt, gehen die Methoden wieder auseinander. Es gibt unzählige Alternativen für eine genaue Implementation [1].

| *Aspekt* | mögliche Implementationen | | |
|---|---|---|---|
| *When to move* | Time-Based | Event-Based | Hybrid |
| *How to move* | Shuffling | Diversity | Redundancy |
| *What to move* | Software Transformation | Dynamic Platform Techniques | Network Address Shuffling |

*Tabelle 1: Übersicht der Aspekte*

Tabelle 1 zeigt nochmals alle Aspekte und deren mögliche Implementationen auf.

### 2.1.5 Effektivität von MTDs

Es stellt sich die Frage ob und wie MTDs tatsächlich effektiver als herkömmliche Verteidigungsmechanismen sind.

Die Idee Diversity als Verteidigungsstrategie einzusetzen, wie es mit MTDs oft der Fall ist, ist keine, die lediglich im Bereich von MTDs stattfindet. Adressraum-Shuffling, beispielsweise, findet auch in statischen Verteidigungen statt. Die Sicherheit des Netzwerks wird dadurch erhöht und manche Angriffe können erfolgreich abgewehrt werden. Allerdings leiden statische Verteidigungsmethoden, welche mit Diversity arbeiten, unter einer meist zu kleinen Entropie, welche dem statischen Charakter der Verteidigung geschuldet ist. Solche Methoden sind deshalb anfällig auf Brute Force- und Probing-Attacken. [5]

Evans et all. stellen in "Effectiveness of Moving Target Defenses" fest, dass die Effektivität von MTDs im Vergleich zu statischen Verteidigungen, je nach Art der Attacke variiert [5]. Tabelle 2 fasst ihre Erkenntnisse zusammen.

| *Angriffsarten* | MTD Effektivität |
|---|---|
| *Circumvention Attacks* | MTD bietet keine Vorteile. |
| *Deputy Attacks* | MTD bietet keine Vorteile. |
| *Entropy Reducing Attacks* | MTD schafft es die benötigte Zeit bis zu einem erfolgreichen Angriff mindestens zu verdoppeln. |
| *Probing Attacks* | MTD ist effektiver, wenn sie über gute Zufallsfunktionen verfügt. |
| *Incremental Attacks* | MTD ist effektiver. |

*Tabelle 2: Effektivität von MTDs nach Art des Angreifers gegenüber Statischen Verteidigungen*

### 2.1.6 Charakteristiken von Netzwerken und bekannte Angriffe

**Geschäftsnetzwerke**

Da es verschiedenste Arten von Netzwerken gibt, muss es auch verschiedenste angepasste MTD-Techniken für diese Netzwerke geben. Im Idealfall ist eine MTD immer an die Art System angepasst, das sie schützt. Das Netzwerk eines durchschnittlichen Betriebes besitzt eine statische Konfiguration. Dies ermöglicht das Auskundschaften des Systems und das Durchführen von Angriffen ohne Zeitdruck.

**Techniken:** Sollte eine MTD vorhanden sein, verfügt diese meist über Diversity und Shuffling basierte Methoden. Zum Beispiel IP-Shuffling, Proxy-Shuffling und System-Diversity [5].

**Häufigste Attacken:** Übliche Angriffe auf solche Systeme sind DDoS-Attacken [6], Scanning-Attacken und abstrakte Attacken [7], die einem Angriff-Verteidigung Muster folgen (Game Theory).

**Cloudnetzwerke**

Cloudcomputing macht eigene IT-Infrastruktur heute immer mehr obsolet. Eine Auslagerung der IT in die Cloud hat neben positiven Effekten, wie die Nutzung von Skaleneffekten durch das Teilen von Recheninfrastruktur, auch negative Effekte. Durch die Konzentration der Infrastruktur auf wenige Rechenzentren erhöhte sich typischerweise auch der Return-of-Investment (ROI) bei einem erfolgreichen Angriff, weil ein Angreifer unter Umständen somit gleich mehrere Nutzer hacken kann. Weiterhin ist ein solcher Service auch auf möglichst hohe Kompatibilität und Homogenität ausgelegt, damit er jede mögliche Anfrage bedienen kann. Dadurch können erhebliche Sicherheitslücken entstehen. Nämlich dann, wenn dadurch eine zentrale Schwachstelle entsteht [8].

**Techniken:** Die meist genutzten MTD-Techniken zur Verteidigung von Cloudnetzwerken sind Shuffling von Serverkopien (Redundancy), VM-Migration [9] und Platformdiversity [10]. Bisher werden oft nur Methoden eingesetzt, die sich auf das Ändern von Konfigurationen oder Systemkomponenten beschränken.

**Häufigste Attacken:** Die häufigsten Angriffe auf Cloud basierte Dienste sind DDoS-Attacken [6], VM Probing-Attacken und Eavesdropping. Oftmals wird dabei vor allem versucht an Benutzerdaten zu gelangen [11].

## 2.2 Reinforcement Learning

### 2.2.1 Einleitung in Reinforcement Learning

In der Literatur wird mit Reinforcement Learning Agent und Reinforcement Learning Algorithmus meist auf dasselbe verwiesen. In dieser Arbeit wird die Bezeichnung Reinforcement Learning Agent und folgend RL Agent verwendet.

Die Idee des Reinforcement Learning folgt in Grundzügen denselben Prozessen, wie wir Menschen sie als Kinder und auch später als Erwachsene noch erleben. Unsere Umwelt ist in einem bestimmten Zustand, und wir interagieren mit ihr, um diesen Zustand zu verändern. Ein Beispiel: Wir öffnen eine Tür, um den dahinterliegenden Raum zugänglich zu machen. Kleinkinder besitzen das Wissen um Aktion und Reaktion nicht. Sie erlernen die Welt, und wie sie funktioniert, von Grund auf. Ähnlich darf man sich auch das Vorgehen im Reinforcement Learning vorstellen. Es geht darum, Situationen an Aktionen zuweisen. Der Lernende weiss nicht, welche Aktionen er in welcher Situation vollführen muss, um an das Ziel zu gelangen. Es wird ihm auch nicht gesagt. Er hat nur die Möglichkeit via Trial-and-Error selbst einen optimalen Weg zu finden. Dies sind auch die zwei wichtigsten Komponenten des Reinforcement Learning: Trial-and-Error Suche, und die Möglichkeit einer verspäteten Belohnung [12].

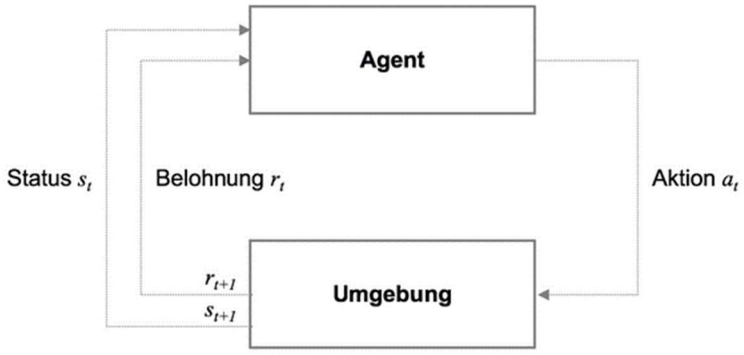



*Abbildung 4: Allgemeine Funktionsweise Reinforcement Learning [17]*

Im standartmässigen Reinforcement Learning Modell ist ein Agent mit einer Umgebung verbunden. Dies kann über Sensoren geschehen, kann aber auch ein rein digitaler Input sein. Er kann in dieser Umgebung bestimmte Aktionen vornehmen und bestimmte Aspekte innerhalb der Umgebung wahrnehmen (siehe Abbildung 4). Der Agent erhält somit einen Input, einige Informationen über den aktuellen Status und kann dann eine Aktion ausführen. Diese Aktion wird den Status der Umgebung wiederum beeinflussen. Zu einem vordefinierten Zeitpunkt erhält der Agent dann eine Rückmeldung, in der er eine Belohnung oder eine Bestrafung dafür erhält, wie gut der Status des Umfelds ist. Der Agent weiss also nicht genau was das Ziel ist, aber ihm wird von Zeit zu Zeit mitgeteilt, wie die aktuelle Situation zu bewerten ist. Durch vielfaches Ausführen und Wiederholen soll der Agent dann selbstständig lernen, welche Aktionen in welcher Situation den grössten Vorteil tragen, und wird so fähig, eine Aufgabe im Trial-and-Error Verfahren selbst zu erlernen [13].

Reinforcement Learning ist eine Herangehensweise an eine bestimmte Aufgabe, die ein Rechner erfüllen soll. Dabei liegt das Ziel darin, dass ein Agent selbstständig den besten Weg findet, eine bestimmte Aufgabe zu lösen. Dies ist offensichtlich insbesondere dann nützlich, wenn wir Menschen keinen optimalen Weg kennen, die Aufgabe zu lösen oder die Ausführung der nötigen Aktionen nicht perfektionieren können. Wenn Reinforcement Learning eingesetzt wird, hoffen wir also, dass ein Agent mit den Ressourcen eines Rechners fähig ist, eine Methode zur Lösung eines Problems zu finden, die wir nicht finden können, oder nicht derart komplett und genau ausführen können, wie der Rechner selbst.

### 2.2.2 Arten von Reinforcement Learning

Im Allgemeinen lassen sich Reinforcement Learning Modelle nur schwer kategorisieren, denn sie sind stark davon abhängig, was der Nutzer von ihnen verlangt. Selten gleicht ein System dem anderen, weshalb die erdachten Modelle oft nur auf ein spezifisches Problem angewandt werden können. Bei dem Erstellen eines Modells gibt es aber dennoch Fragen, die zu beantworten sind (siehe Abbildung 5).

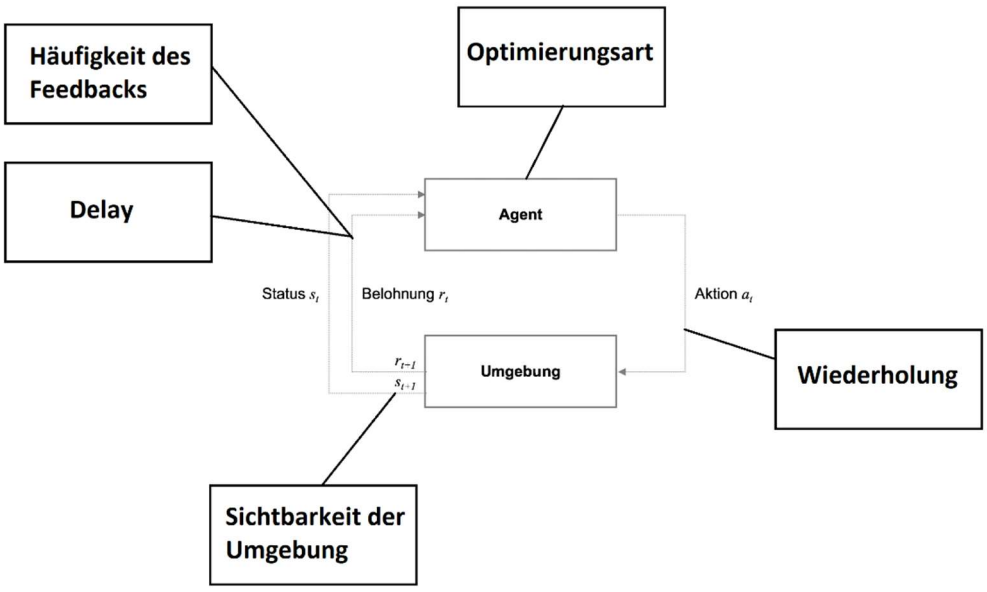



*Abbildung 5: Grundlegende Fragen an das Modell*

**Die Frage nach der Sichtbarkeit der Umgebung:** Was genau der Agent registriert/sehen kann ist eine wichtige Komponente. Je mehr der Agent wahrnimmt, desto mehr kann er auf Feedback reagieren.

**Die Frage nach der Häufigkeit des Feedbacks:** Je öfters der Agent Feedback erhält, desto schneller kann er adaptieren. Die Frage ist hier, ob man nach jedem Zeitschritt Feedback geben kann, oder nur nach einer bestimmten Zeit. In manchen Modellen macht es auch Sinn, erst am Ende eines Programmdurchlaufs Feedback zu geben. Je häufiger Feedback erfolgt, desto schneller kann der Agent lernen.

**Die Frage nach dem Delay:** Da Aktionen häufig nicht sofort, sondern erst verspätet zu einer Belohnung führen könnten, muss der Agent auch eine Möglichkeit haben, die Zukunft zu berücksichtigen, wenn er Entscheidungen trifft. Die Frage stellt sich, wie weit in die Zukunft der Agent schauen können sollte, um eine optimale Belohnung für sich zu erreichen. Je mehr Schritte er voraus denkt, desto genauer wird er. Die Rechenleistung erhöht sich dabei aber auch mit jedem Schritt unter Umständen stark.

**Die Frage nach der Wiederholung:** Wenn ein Modell erstellt ist, muss entschieden werden, wie der Agent darin wiederholt eingesetzt wird. Dabei gibt es zwei mögliche Arten. Zum ersten ist es möglich, ihn in einer Endlosschleife mit einem Programm interagieren zu lassen, um zu lernen. Die Umgebung würde niemals zu einem Ende kommen und der Agent würde wie in einem Sandkasten verschiedene Dinge ausprobieren. Allerdings gibt es auch Umgebungen, die irgendwann ein Ende finden, wie zum Beispiel ein Schachspiel. Eine solche Umgebung muss neu gestartet werden, und der Agent muss von vorne anfangen.

**Die Frage nach der Optimierungsart:** Es gibt grundsätzlich zwei Arten, wie der Schwerpunkt eines Agenten auf die Optimierung sein kann. Er kann Prozess basiert oder Value basiert arbeiten. Der Agent versucht dabei also entweder den Prozess selbst zu verbessern oder einfach eine möglichst hohe Punktzahl bei der Belohnung zu erreichen.

### 2.2.3 Ad-Hoc Techniken

Obwohl es schwer ist, Reinforcement Learning Modelle zu kategorisieren, gibt es einige Ad-Hoc-Techniken, die zum Einsatz kommen. Diese sind sehr simpel, und selten der beste Weg, ein bestimmtes Problem zu bewältigen. Aber sie können als vernünftige, ressourcensparende, und vor allem nachverfolgbare Testalgorithmen angesehen werden. Nach Thrun [14] gibt es folgende Techniken:

**Greedy Strategien:** Der erste Gedanke beim Aussuchen einer Gewichtung ist es, den Agenten immer die Aktion ausführen zu lassen, die die meiste Belohnung mit sich bringt. Das Problem an dieser Methode ist, dass wenn die beste Strategie Aktionen beinhaltet, die zu Beginn wenig beobachtbaren Erfolg bieten, dafür später umso mehr, andere Aktionen bevorzugt werden und somit zwar ein lokales, jedoch nicht ein globales Optimum gefunden werden kann. Wenn eine Greedy Strategie verwendet wird, ist es deshalb zu empfehlen, diese sehr optimistisch zu konfigurieren, so dass die Suche aus einem lokalen Optimum auch ausbrechen kann und somit weitere Wege ausprobiert werden [13].

**Randomized Strategien:** Eine andere einfache Strategie für den Agenten ist es, die Aktion mit der höchstmöglichen Belohnung als Standard zu nehmen, aber ihn immer mit einer gewissen Wahrscheinlichkeit p eine zufällige Aktion aussuchen zu lassen. Manche Versionen dieser Strategie starten mit einem hohen Wert p, welcher mit der Zeit schrumpft. So wird sichergestellt, dass der Agent anfänglich viele verschiedene Wege testet [13].

### 2.2.4 Reinforcement Learning und MTDs

Traditionelle, statische Verteidigungsmechanismen für Netzwerke sind erwiesenermassen zu langsam und starr, um mit den Bedrohungen aus der heute hochdynamischen Bedrohungslandschaft (Threat-Landscape) mitzuhalten [15].

Moving Target Defense Strategien scheinen in verschiedenster Hinsicht ein Fortschritt gegenüber statischen Verteidigungsmethoden zu sein. Allerdings braucht diese Art der Verteidigung in grösseren Netzwerken eine grosse Menge an Ressourcen, um effizient zu arbeiten. Diese Schwäche hindert MTDs daran, grossflächig eingesetzt zu werden. Aktuelle Forschung versucht den Einsatz von Ressourcen dadurch zu minimieren, dass Prozesse beschleunigt und an spezifische Netzwerke angepasst werden. Dies führt aber wieder dazu, dass Implementationen von MTDs zu spezifisch werden und erneut nicht grossflächig eingesetzt werden können [16].

Um dieses Problem zu beheben schlagen bereits Chai et al. in «DQ-MOTAG: Deep Reinforcement Learning-based Moving Target Defense Against DDoS Attacks» vor, auf Reinforcement Learning zurück zu greifen, um einen optimalen Ressourcenverbraucht und eine optimale Verteidigungsstrategie für eine MTD festzulegen. Anstatt eines statischen MTD-Algorithmus, der die Verteidigung diktiert, soll ein RL Agent die effektivste und günstigste Art der Verteidigung erlernen [17].

# 3 Methodik

## 3.1 Modell 1.0

Um den Einsatz eines RL Agenten innerhalb einer Moving Target Defense zu testen und Vergleiche anstellen zu können, braucht es eine IT-Infrastruktur respektive ein Modell von dieser. Auf diese Infrastruktur werden dann Angriffe simuliert, die der Agent abwehren muss. Das Modell der IT-Infrastruktur wird nachfolgend aufgezeigt (siehe Abbildung 6). Die möglichen Aktionen des Agenten werden in Abschnitt 3.3.1 genauer erläutert.

Beim Aufbau des Modells dient als Orientierung der Aufbau von Zhuang et al. in "Investigating the Application of Moving Target Defenses to Network Security" [18]. Hier wird ein konservativer Angriffsgraph (CAG) vorgeschlagen, welcher zur Simulation und Evaluation dient. Dieser CAG besitzt sechs Zustände und arbeitet mit fixen Wahrscheinlichkeiten als Übergänge. Diese Übergänge simulieren den Erfolg des Angreifers pro Zeitschritt. Diese Abstraktion erlaubt es, das schwierige Wechselspiel zwischen Angreifer und Ziel auf eine anpassbare Zahl zu projizieren. Diese Spezifikation führt dazu, dass der Angriffsgraph wie eine state machine aufgebaut ist.

Bei der Verteidigung mit MTDs verliert ein Angreifer erlangte Informationen über das Netzwerk wieder. Normalerweise verändern sich diese Informationen nicht, sodass während einem Angriff nicht mehrmals nach bestimmten Eigenschaften geforscht werden muss. Dies bricht mit der üblichen Annahme, dass Angriffe monoton stattfinden [19].

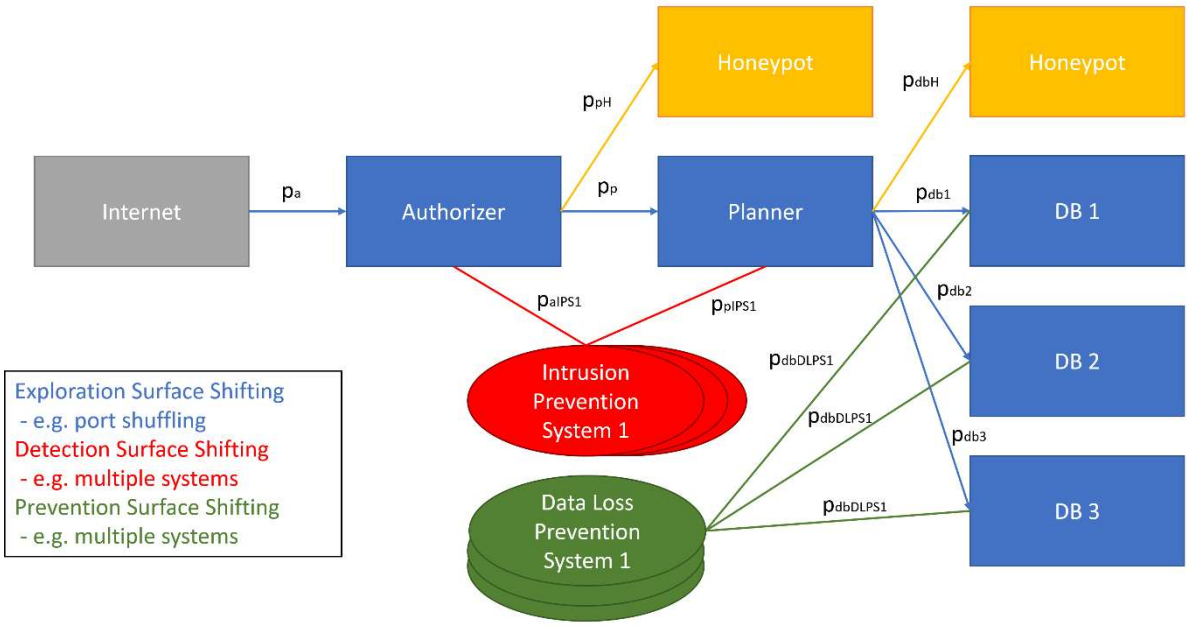



*Abbildung 6: Modell 1.0*

Im Laufe der Entwicklung des Modells ist es wichtig, die Möglichkeit zu haben, verschiedenste Parameter dynamisch anzupassen. Aus diesem Grund fällt die Entscheidung auf den simplen Ansatz aus Zhang et al. [18]. Version 1.0 des Modells kennt 8 verschiedene Zustände (siehe Abbildung 6). In diesem Modell stellen die Knoten Services dar, während die Übergänge Wahrscheinlichkeiten sind, mit denen der Angreifer in den nächsten Service eindringen kann. Die verwendeten Kürzel im Modell werden in Tabelle 3 ausführlicher beschrieben.

**Legende der Wahrscheinlichkeiten für Modell 1.0**

| *Kürzel* | Formulierung |
|---|---|
| *pa* | Angreifer erlangt Kontrolle über den Authorizer |
| *pb* | Angreifer erlangt Kontrolle über den Planner. |
| *pdb1* | Angreifer erlangt Kontrolle über Datenbank 1. |
| *pdb2* | Angreifer erlangt Kontrolle über Datenbank 2. |
| *pdb3* | Angreifer erlangt Kontrolle über Datenbank 3. |
| *ppH* | Angreifer tritt in den Honeypot anstatt in den Planner. |
| *pdbH* | Angreifer tritt in den Honeypot anstatt in eine der Datenbanken. |
| *paIPS1* | IPS 1 erkennt einen Angreifer auf dem Authorizer und wirft ihn raus. |
| *ppIPS1* | IPS 1 erkennt einen Angreifer auf dem Planner und wirft ihn raus. |
| *pdbDLPS1* | DLPS 1 erkennt einen Angreifer auf einer Datenbank und wirft ihn raus. |

*Tabelle 3: Legender der Wahrscheinlichkeiten des Modells 1.0*

Der Angreifer startet im Zustand "Internet" und hat dort pro Zeitschritt die Möglichkeit, mit einer bestimmten Wahrscheinlichkeit in den Zustand "Authorizer" zu gelangen. So geht es weiter von "Authorizer" zu "Planner" und dann in eine der drei möglichen "DB"s. Zusätzlich zum zugrundeliegenden Modell wurden weitere Zustände in das eigene Modell eingefügt. Die Zustände "Honeypot" simulieren den Einsatz von Honeypots als Verteidigungsstrategie. Sie sollen einen Angreifer anlocken, und ihn so auf einen falschen Weg lenken. Im besten Fall glaubt er Angreifer, wenn er den Honeypot betritt, dass er sein Ziel erreicht hat und entwendet dort Daten, die er für wertvoll hält.

Weitere Ergänzungen sind das Intrusion Prevention System, folgend IPS, und das Data Loss Prevention System, folgend DLPS. Sie stellen die Verteidigung des Netzwerkes dar. Im Folgenden wird mit "Präventionssystem" gleichermassen auf das IPS und auf das DLPS verwiesen. Das IPS ist in diesem Modell eine Mischung von Intrusion Detection System und Intrusion Prevention System. Bei einem entdeckten Angriff wird der Verursacher aus dem Netzwerk geworfen. Dies konkret zu implementieren, oder genauer zu erläutern, liegt nicht im Rahmen dieser Arbeit. Auch das DLPS kickt den Verursacher aus dem Netzwerk, sollte es eine Datenextraktion erkennen. Auch das Problem, dass normale Nutzer von einem IDS, IPS oder DLPS als Angreifer klassifiziert werden können, wird nicht weiter aufgegriffen. Bei schon existierende Produkten und MTDs ist diese Problematik bekannt und würde hier erst bei einer konkreten Implementation relevant sein.

## 3.2 Hilfsmittel und Frameworks

Die Version 1.0 des Modells wird als Entwurf genutzt, um die Ziele der Arbeit genauer zu definieren. Zudem können damit die geeigneten Frameworks für die RL Agenten erörtert werden. Das Gespräch mit einem Fachmann auf dem Gebiet des Reinforcement Learning, Dr. Volker Ziebart, zeigt auf, welche Werkzeuge für gewöhnlich verwendet werden, um RL Agenten zu trainieren, und deren Lern-Umgebung aufzubauen. Aus den Erkenntnissen des Gesprächs ist der Entscheid auf Gym von OpenAI und Stable Baselines 3 gefallen.

**OpenAI Gym:** Dies ist ein Werkzeug, welches für die Entwicklung und den Vergleich von RL Agenten genutzt wird. Gym ist eine Open Source Bibliothek, welche Zugriff auf eine standardisierte Umgebung ermöglicht. Mit dieser Bibliothek ist es möglich, eine standardisierte Test-Umgebung für RL Agenten zu definieren [20]. Zusätzlich kann die selbst definierte Test-Umgebung auf Kompatibilität mit RL Agenten von Stable Baselines 3 geprüft werden.

**Stable Baselines 3:** Dies ist eine Erweiterung für OpenAI Gym. Es handelt sich um ein Set von verbesserten Implementationen von RL Agenten, auf denen man wiederum eigene Projekte aufbauen kann. Wie dem Namen Stable Baselines zu entnehmen ist, sind diese Agenten von Anfang an "stabil" im Lernen, das heisst, die Hyperparameter müssen von Benutzern in der Regel nicht mehr angepasst werden, um erste nützliche Resultate zu erhalten. Für das Modell dieser Arbeit konnten die Hyperparameter direkt von Stable Baselines übernommen werden und sind nicht mehr angepasst worden. Nur in einem Aspekt dieser Arbeit wäre eine Überarbeitung sinnvoll, mehr dazu im Kapitel 4.1.4.

Diese verbesserten Agenten decken eine Vielzahl verschiedenen Ansätze ab und öffnen damit das Spektrum und die Einsetzbarkeit von Gym. So wird es einfacher für die Nutzer, und auch die Industrie, funktionierende Techniken zu kopieren, abzuändern, zu ersetzen, und zu verbessern. Stable Baselines wird verwendet um vordefinierte RL Agenten auf besagtem Modell zu trainieren [21].

Da Gym lediglich für die Sprache Python entwickelt wurde, muss auch das Modell mit Python modelliert werden.

## 3.3 Modell 2.0

Die gewählten Werkzeuge Gym und Stable Baselines ermöglichen es, das Modell zu präzisieren. Durch verschiedene Vorgaben ergibt sich auch die Notwendigkeit, das ursprüngliche Modell 1.0 der Umgebung für den RL Agenten anzupassen.

Das Modell 2.0 (siehe Abbildung 7) folgt denselben Prinzipien wie Modell 1.0. Die Übergänge sind als Wahrscheinlichkeiten geplant und die Zustände sind weiterhin dieselben. In dieser Version des Modells werden die Präventionssysteme genauer definiert. Das Erkennen eines Angreifers findet nun in den drei Übergängen von Service zu Service statt. Die Prävention arbeitet nun auf der Verknüpfung zu einem neuen Zustand "Extract Data". Dieser Zustand ist die letzte Hürde für den Angreifer, und stellt die letzte Verteidigung des Systems dar. Gut zu sehen ist ebenfalls dass die Übergänge nun bidirektional eingezeichnet sind. Das heisst, bei einem Neustart eines Services wird der Angreifer eine Stufe zurückversetzt, also zum Beispiel vom Planner zurück zum Authorizer.

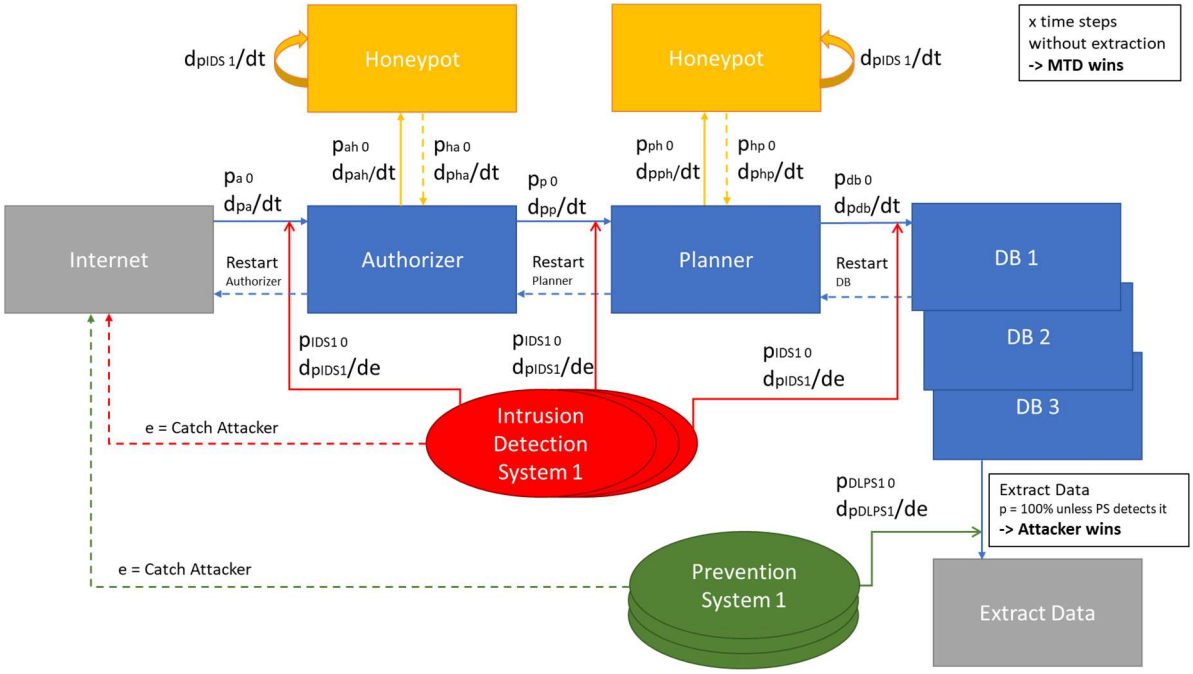



*Abbildung 7: Modell 2.0*

### 3.3.1 Angreifer

Der Angreifer wird nur simuliert und hat in diesem Sinne keinen "freien Willen". Ist der Angreifer z.B. im Authorizer, werden als erstes die Wahrscheinlichkeiten der Übergange gelesen, also die zum Honeypot und die zum Planner (siehe Tabelle 4). Dann wird mit einem Zufallsgenerator der nächste Zustand des Angreifers entschieden. Kommt der Angreifer nicht zum nächsten Service, "lernt" er. Das heisst, die Wahrscheinlichkeit, dass er im nächsten Zeitschritt zu diesem Service gelangt, erhöht sich. Gelingt es dem Angreifer vom Internet in den Honeypot einzudringen, kann er auch in zukünftigen Schritten wieder direkt eindringen, ausser der Authorizer wird neugestartet.

Werden die Aktionen des Angreifers von einem Präventionssystem erkannt, wird er sofort aus dem Netzwerk geworfen. Allerdings ist das Präventionssystem nun weniger effektiv, da auch hier der Angreifer "lernt". Das heisst die Wahrscheinlichkeit, dass das Präventionssystem den Angreifer nochmals erkennt, wird verringert.

Dies alles wiederholt sich nun für jeden Zeitschritt, bis der Angreifer entweder im Zustand "Extracted Data" landet, was bedeutet, dass der Angreifer gewinnt, oder eine gewisse Anzahl Zeitschritte vergangen sind, was bedeutet, der Verteidiger gewinnt.

**Legende mit Wahrscheinlichkeiten**

| *Kürzel* | Formulierung | Wert |
|---|---|---|
| *pa 0* | Angreifer erlangt Kontrolle über den Authorizer. | 0.6 [18] |
| *dpa/dt* | Veränderung von pa nach einem Zeitschritt. | 0.1 |
| *pp 0* | Angreifer erlangt Kontrolle über den Planner. | 0.6 [18] |
| *dpp/dt* | Veränderung von pp nach einem Zeitschritt. | 0.1 |
| *pdb1 0*[1] | Angreifer erlangt Kontrolle über Datenbank 1. | 0.4 [18] |
| *dpdb1/dt* | Veränderung von pdb1 nach einem Zeitschritt. | 0.1 |
| *pdb2 0* | Angreifer erlangt Kontrolle über Datenbank 2. | 0.4 [18] |
| *dpdb2/dt* | Veränderung von pdb2 nach einem Zeitschritt. | 0.1 |
| *pdb3 0* | Angreifer erlangt Kontrolle über Datenbank 3. | 0.4 [18] |
| *dpdb3/dt* | Veränderung von pdb3 nach einem Zeitschritt. | 0.1 |
| *pah 0* | Angreifer tritt vom Authorizer in den Honeypot. | 0.7 |
| *dpah/dt* | Veränderung von pah nach einem Zeitschritt. | -0.1 |
| *pha 0* | Angreifer gelangt vom Honeypot zurück zum Authorizer. | 0.0 |
| *dpha/dt* | Veränderung von pha nach einem Zeitschritt. | 0.1 |
| *pph 0* | Angreifer tritt vom Planner in den Honeypot. | 0.7 |
| *dpph/dt* | Veränderung von pph nach einem Zeitschritt. | -0.1 |
| *php 0* | Angreifer gelangt vom Honeypot zurück zum Planner. | 0.0 |
| *dphp/dt* | Veränderung von php nach einem Zeitschritt. | 0.1 |
| *pIPS1 0* | IPS 1 erkennt Angreifer vor dem Authorizer, Planer oder einer Datenbank. | 0.9911 [22] |
| *dpIPS1/de* | Veränderung von pIPS1 nach erkennen und Herauswerfen des Angreifers. | 0.5 |
| *pDLPS1 0* | DLPS1 erkennt Angreifer beim Daten Extrahieren. | 0.5 |
| *dpDLPS1/de* | Veränderung von pDLPS1 nach Erkennen und Herauswerfen des Angreifers. | 0.5 |

[1] Der Kürzel "pdb 0" in der Abbildung 7 steht sinnbildlich für alle Verbindungen vom Planner zu den Datenbanken. Aus Gründen der Übersichtlichkeit wird in der Abbildung des Modells auf ausführliche Beschriftung verzichtet.

*Tabelle 4: Legende und Beispielswahrscheinlichkeiten des Modell 2.0*

**Begründung der Werte**

Die Werte für den Authorizer (0.6), den Planner (0.6) und die DBs (0.4) stammen von der Arbeit von Zhuang et al. [18]. Der Wert für die Veränderung nach einem Angriff (0.1 pro Schritt) ist so gesetzt, dass der Angreifer eine bessere Chance hat, den Service im nächsten Schritt unter Kontrolle zu bringen. Trotzdem ist so nicht garantiert, dass der Angreifer im nächsten Schritt den Service wirklich unter Kontrolle bringt, was auch realistisch ist. Als Beispiel: Beim Zeitschritt x hat der Angreifer eine Chance von 0.6, den Authorizer unter Kontrolle zu bringen. Der Zufallsgenerator entscheidet nun, dass der Angreifer nicht eindringen kann. Der Angreifer ist beim Zeitschritt x+1 also immer noch vor dem Authorizer, allerdings ist seine Chance nun 0.7, um eine Stufe weiter ins Netzwerk einzudringen.

Die Werte für die Honeypots (0.7) sind etwas höher als die des Authorizers und die des Planners, weil ein Honeypot attraktiver sein soll wie das eigentliche Ziel. Ist der Angreifer einmal im Honeypot, bleibt dieser für mindestens einen Schritt darin (0.0). Das widerspiegelt das Ziel eines Honeypots, wie ein echtes Ziel zu wirken (das heisst der Angreifer will da gar nicht weg). Je länger er aber sich darin befindet (0.1 pro Schritt), desto eher fliegt die Täuschung auf, und er gelangt zurück.

Das "verwendete" IPS ist sehr effektiv, einen Angriff zu erkennen (0.9911). Das wird von der Arbeit von S. Dhaliwal et al. ermittelt. [22] Trotzdem ist das Modell so aufgebaut, dass sich die der Angreifer lernen kann und ein statisches IPS schnell umgehen werden kann. Nach Erkennung eines Angriffes wird die Chance, den Angreifer nochmals zu erwischen, halbiert (0.5). Wenn nun aber das IPS ausgewechselt wird, gilt wieder die ursprüngliche Wahrscheinlichkeit (0.9911). Was es allgemein bei IPS zu bedenken gibt, ist, dass zum Teil auch legitime Benutzer des Netzwerkes als Angreifer bezeichnet werden, und vom Netzwerk gekickt werden. Das muss aber unabhängig gelöst werden, ob die Verteidigung nun MTD implementiert oder nicht. Für das DLPS sind effektive Zahlen selten, womit hier angenommen werden muss, dass es die Hälfte der Datenextraktionen erkennt und ebenfalls ineffektiver wird nach Verhinderung einer Datenextraktion.

### 3.3.2 Verteidiger

Die RL Agenten bilden die Verteidiger in der Simulation. Diese können nun aktiv bei jedem Zeitschritt ihre Aktion wählen (siehe Abbildung 8). Im normalen Modus können sie pro Zeitschritt einen Service neustarten und ein Präventionssystem auswechseln. Dies führt bei beiden dazu, dass die Übergangswahrscheinlichkeit wieder zurückgesetzt wird, d.h. alles was der Angreifer über diese Systeme gerade eben erlernte, ist wieder schon wieder ungültig und kann von ihm nicht mehr verwendet werden.

Als Vergleich zu den beiden RL Agenten wird eine zufällige Verteidigung (herkömmliche MTDs), eine statische Verteidigung (kein MTD) und eine eigendefinierte Verteidigung, der Defender2000, simuliert. Genauere Angaben zu den Agenten und zur eigendefinierten Verteidigung sind im Kapitel 3.4 respektive 3.5 zu finden.

**Ein Zeitschritt aus Sicht des Agenten**

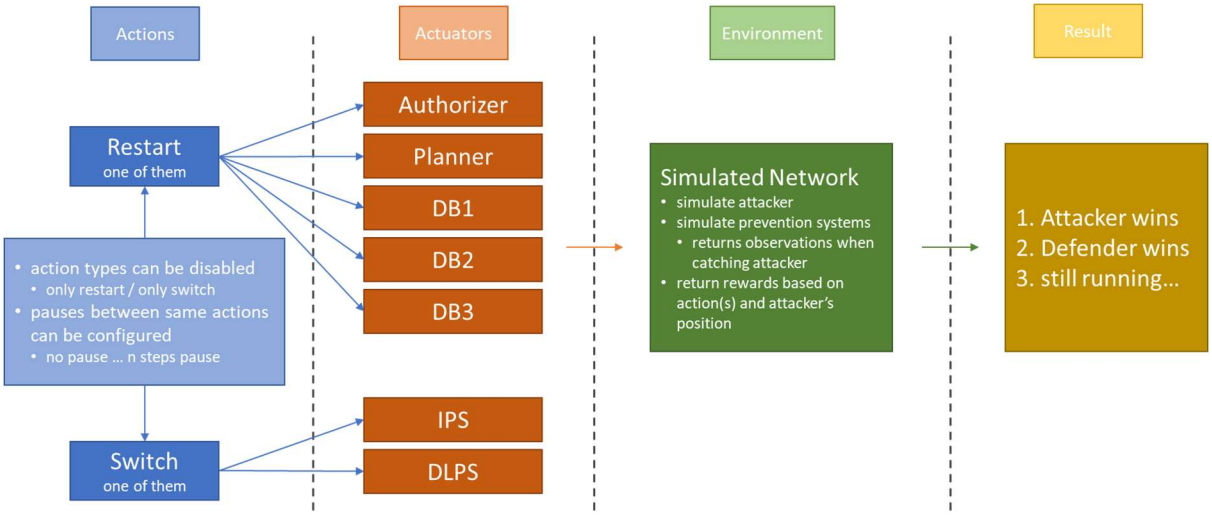



*Abbildung 8: Ein Zeitschritt aus Sicht des Agenten*

### 3.3.3 Zeitschritt t

Die Simulation ist zeitlich diskretisiert, das heisst, der Angreifer sowie die Agenten können nur alle x Sekunden mit dem Modell interagieren. Der Angreifer und die Agenten sind somit unabhängig von einem konkreten Zeitschritt. Die Dauer des Zeitschrittes wirkt sich nur auf die Wahrscheinlichkeiten aus. Hat der Angreifer eine Minute, steigen seine Chancen, im Vergleich dazu, wenn er nur 5 Sekunden hätte pro MTD Aktion. Für die verwendeten Wahrscheinlichkeiten wird ein Zeitschritt von 10 Sekunden angenommen.

Bei einem kleineren Zeitschritt wäre das Modell für normale Benutzer nicht mehr brauchbar, weil das Modell nur noch mit Service Neustarten oder Präventionssystem Auswechseln beschäftigt wäre. Bei einem grösseren Zeitschritt hat der Angreifer bessere Chancen, das Modell unter Kontrolle zu bringen. Ein Angriff kann durchaus weniger als 10 Sekunden in Anspruch nehmen. Allerdings hat der Angreifer in seinem besten Falle 10 Sekunden Zeit, den Service zu untersuchen und die Antwort des Service zu analysieren, bevor diese Informationen wieder nutzlos werden.

### 3.3.4 Aktionen und Auswirkungen

In der Tabelle 5 werden nochmals die möglichen Aktionen der Agenten aufgelistet. In der Tabelle 6 auf der nächsten Seite werden die Kosten der Aktionen und Zuständen aufgelistet. Diese werden benötigt, damit die Aktionen der Agenten bewertet werden können, sich die Agenten somit in ihren Aktionen verbessern können, und schlussendlich das Netzwerk besser verteidigen.

**Legende der möglichen Aktionen:**

| ***Aktion*** | **Auswirkung** |
|---|---|
| *Authorizer neustarten* | Kickt den Angreifer aus dem Authorizer, setzt pa auf pa 0. |
| *Planner neustarten* | Kickt den Angreifer aus dem Planner, setzt pp auf pp 0. |
| *DB neustarten* | Kickt den Angreifer aus der DB, setzt pdb auf pdb 0. |
| *IPS auswechseln* | Setzt pIPS zurück auf pIPS 0. |
| *DLPS auswechseln* | Setzt pDLPS zurück auf pDLPS 0. |
| *Angreifer entdecken* | Kickt den Angreifer aus dem Netzwerk. |
| *Datenextraktion* | Angreifer gewinnt. |
| *X Zeitschritte ohne Datenextraktion* | MTD gewinnt. |

*Tabelle 5: Legender der möglichen Aktionen und Zustände*

**Begründung der Auswirkungen eines Service-Neustarts**

Das Neustarten eines Service hat die Konsequenz, dass der Angreifer eine Stufe zurück versetzt wird Netzwerk und seine erlangten Informationen unnütz zu machen. Das heisst, der Angreifer muss wieder von Anfang an lernen (z.B. pa 0), um in den Service zu gelangen. Diese Arbeit spezifiziert dabei nicht die exakte Funktionsweise, als Beispiele können aber folgende drei Aspekte genannt werden:

**Buffer Overflow Angriff:** Mit "Address Space Layout Optimization" (ASLR) werden die Adressen, die ein Angreifer für einen erfolgreichen Angriff aufrufen muss, bei jedem Start des Programms neu ausgelegt. Somit werden bei jedem Start jegliche Informationen aus vorherigen Angriffen ungültig.

**Session Key Brute Force Angriff**: Bei jedem Neustart werden die alten Session Keys ungültig und werden neu generiert. Dabei wird der Fortschritts des Angreifers ebenfalls vernichtet. Als Ausnahme gilt wiederum, wenn der Neustart zu spät kommt, und der Angreifer schon einen Weg in den Service gefunden hat.

**Reverse Shell**: Bei einem erfolgreichen Angriff wird sehr häufig eine sogenannte Reverse Shell erstellt. Somit kann der Angreifer den Service (oder zumindest einen Teil vom Service) unter seine Kontrolle bringen. Wird nach einem Neustart zudem eine andere Software oder gar ein anderes OS benutzt, ist der Command zur Erstellung der Reverse Shell meist ungültig. Zudem wird der Prozess, auf dem die Reverse Shell läuft, bei einem Neustart des Service unterbrochen. Somit wird der Angreifer eine Stufe zurückversetzt im Netzwerk, also auf den vorhergehenden Service.

**Begründung der Auswirkungen einer Präventionssystem-Auswechslung**

Nach dem Auswechseln eines Präventionssystems weiss der Angreifer nicht mehr, welche Angriffe unentdeckt bleiben und welche erfolgreich sein werden. Daher muss er wieder von Vorne beginnen. Was in dem Modell nicht berücksichtigt wird, ist, dass der Angreifer nach einer gewissen Zeit alle Präventionssysteme kennt und raten könnte, welcher Angriff erfolgreich sein wird. Dem wird entgegengewirkt, indem die Effektivität des Präventionssystems nach jedem erfolglosen Angriff halbiert wird.

Wenn nun alle verwendeten Präventionssysteme dieselbe Schwachstelle haben, ist das Auswechseln nutzlos. Dies ist jedoch eine allgemein bekannte Herausforderung im Bereich der MTD und wird in diesem Modell nicht genauer analysiert [2].

**Kosten der MTD-Aktionen und MTD-Zuständen**

| *Aktion* | Kosten |
|---|---|
| *Authorizer neustarten* | 5 |
| *Planner neustarten* | 5 |
| *DB neustarten* | 5 |
| *IPS auswechseln* | 2 |
| *DLPS auswechseln* | 2 |
| *Progression*[2] | 50 |
| *Invalide Aktion*[3] | 50 |
| *Datenextraktion* | 100 |

*Tabelle 6: Kosten der MTD-Aktionen*

Wie schon aufgezeigt, wird der Angreifer nur simuliert. Das bedeutet, dass alle Aktionen nur vom Verteidiger stammen können und somit kann nur der Verteidiger belohnt, respektive bestraft werden (siehe Tabelle 6).

Eine Progression des Angreifers, oder gar der Sieg des Angreifers kann als Folge von "nicht optimalen" Aktionen der Verteidigers interpretiert werden. Deshalb wird der Verteidiger in solchen Fällen bestraft. Optimalerweise, jedenfalls nach Dr. V. Ziebart, sollte erörtert werden, welche Aktionen zu einem Sieg des Angreifers führen, und dann die Bestrafung über diese Aktionen verteilen. Allerdings sind die RL Agenten von Stable Baselines so gut ausgewogen, dass diese Agenten die vergangenen und zukünftigen Belohnungen respektive Bestrafungen von allein sinnvoll miteinbeziehen.

[2] Authorizer = **1***50, Planner = **2***50, DBs = **3***50, Extract Data = **4***50

[3] Invalide Aktionen können im Modus "Pause" entstehen, mehr Details siehe Kapitel 4.1.4

## 3.4 Reinforcement Learning Agenten

Es gibt grundsätzlich zwei verschiedene Arten von Agenten, jene mit Modell, und jene ohne. In der Literatur wird hier mit "Modell" beschrieben, ob der Agent sich ein Modell aufbauen kann, oder ihm sogar schon ein Modell zur Verfügung steht, dass den nächsten Zustand des Netzwerkes exakt antizipieren kann. Ein bekanntes Beispiel für ein Agent mit gegebenem Modell, ist der, der 2016 den Champion des Spiels "GO" besiegt hat [23]. Auch bei Schach kann der nächste Zustand des Spiels mithilfe von Heuristiken und der damit verbundenen Entscheidungsbäumen antizipiert und bewertet werden. Das gezeigte Modell 2.0 unterstützt dies nicht. Die Agenten sind nur in der Lage, ihre Aktionen kritisch zu hinterfragen, und sinnvolle Aktionen auf gewisse Zustände durchzuführen. Ein Modell im Sinne des RL ist dies aber noch nicht. Zudem unterstützt Stable Baselines 3 nur Modell-freie Agenten, somit fallen einige Agenten weg. Die Modell-freien Agenten unterscheiden sich noch einmal mit Policy Optimization und Q-Learning.

**Policy Optimization**

Bei der Policy Optimization wird für jeden Zustand des Systems eine Aktion zugewiesen und versucht, die beste Aktion für einen gegebenen Zustand zu finden. In gezeigtem Netzwerk könnte das folgendes bedeuten: Der Agent erhält Feedback, dass der Angreifer z.B. auf dem Planner vom IPS gesichtet wurde. Das beschreibt den Zustand des Systems. Nun soll die beste Aktion für diesen Zustand gefunden werden.

**Q-Learning**

Beim Q-Learning wird eine Tabelle angelegt und festgehalten, wie gut z.B. die Aktion a im Zustand z war. Auch hier wird schlussendlich die optimale Aktion für einen gegebenen Zustand gesucht. [23]

**RL Agenten Taxonomie**

In folgender Abbildung 9 werden oben beschriebene Kategorien nochmals aufgezeigt mit den bekanntesten Agenten.

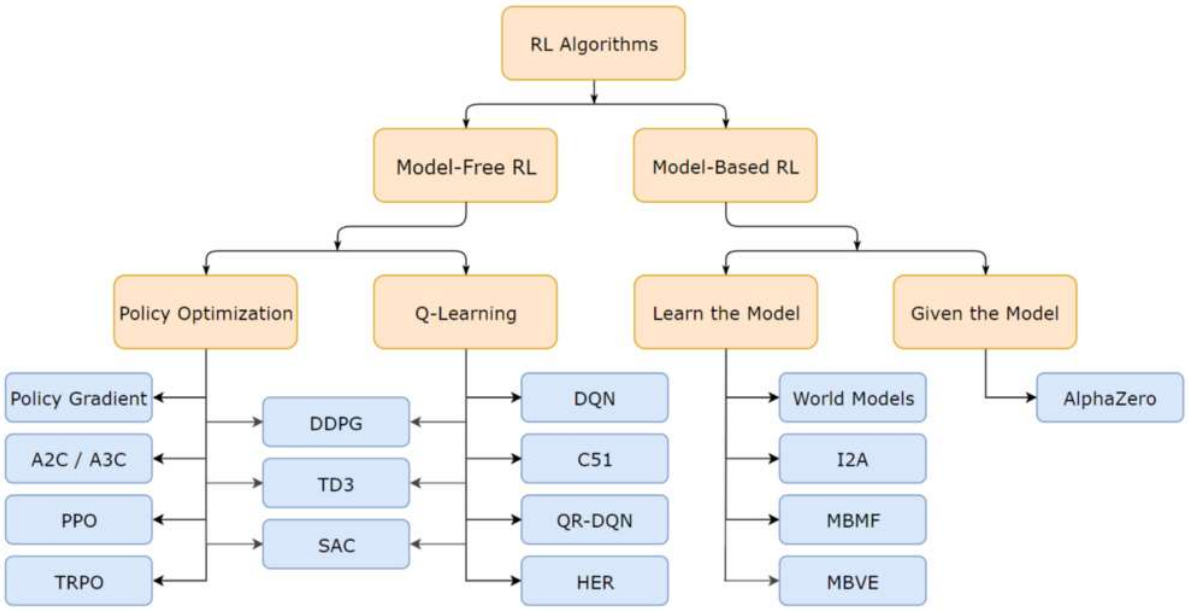



*Abbildung 9: RL Agenten Taxonomie [23]*

Stable Baselines 3 stellt im Vergleich zu den in Abbildung 9 gezeigten Agenten eine geringere Menge an RL Agenten bereit. Die von Stable Baselines 3 unterstützten Agenten sind in Abbildung 10 zu sehen.

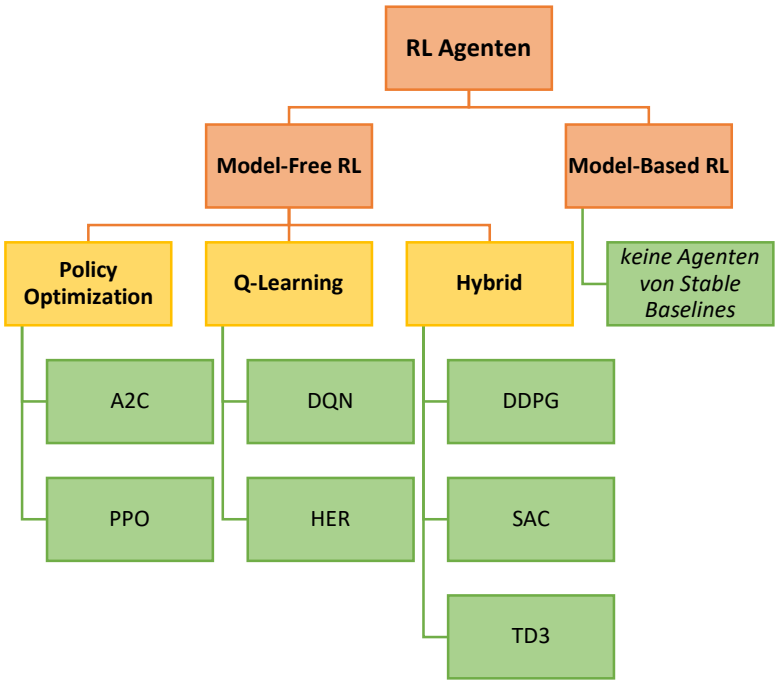



*Abbildung 10: Stable Baselines RL Agenten Übersicht [21]*

Diese Agenten unterstützen wiederum nur bestimmte Modelle. Hier bezieht sich "Modell" wieder auf das Netzwerk vom Kapitel 3.3, Modell 2.0. Das Modell 2.0 bietet den Agenten nur diskrete Aktionen an. Dies kann damit gezeigt werden, dass der Agent nur den Service 1, 2, … neustarten kann. Ein Service 2.43 existiert ja gar nicht. Zusätzlich können Services und Präventionssysteme im gleichen Zeitschritt neugestartet, respektive ausgewechselt werden. Somit ist ein Multi-Diskreter-Aktionsraum erforderlich. In diesem Aktionsraum kann der Agent (in diesem Falle) zwei Aktionen pro Zeitschritt ausführen. Dieser Aktionsraum wird nur noch von den Agenten A2C und PPO unterstützt. DQN und HER unterstützen den diskreten Aktionsraum, allerdings entstehen so Probleme beim Implementieren. Im folgenden Absatz werden diese Probleme geschildert.

Ein Diskreter-Aktionsraum, im Vergleich zu einem Multi-Diskreten-, bedeuten, dass pro Zeitschritt entweder ein Service oder ein Präventionssystem angesteuert werden kann, aber nicht beide gleichzeitig. Eine Ausnahme gäbe es, um diese Einschränkung zu umgehen, aber dadurch verdoppeln sich die möglichen Aktionen und der Agent hat es viel schwieriger, eine Lösung zu finden. Abgesehen von diesen Herausforderungen lässt sich DQN implementieren. Allerdings haben erste Tests mit dem DQN Agenten sehr schlechte Resultate ergeben im Vergleich zu A2C und PPO. Somit wird dieser Ansatz nicht weiterverfolgt. Beim HER Agenten kommt ein zusätzliches Implementationsproblem dazu.

Die Umgebung, in welcher der HER Agent trainieren kann, also das Modell 2.0, ist nicht kompatibel mit dem Agenten. Das Modell soll für HER vektorisiert sein. Stable Baselines bietet eine Funktion an um ein Modell zu vektorisieren, das funktioniert aber nicht bei diesem Modell. Eine eigene Implementation ist voraussichtlich zu aufwändig. Zusätzlich wird erwartet, dass HER ebenfalls schlechte Resultate liefert, da der Agent von derselben Kategorie wie DQN ist.

Schlussendlich bedeutet das, dass nur Agenten vom Typ Policy Optimization implementiert sind. Nochmal zur Repetition, dieser Typ von Agent versucht grundsätzlich, auf einen gegebenen Zustand des Modells die passendste und beste Aktion zu lernen. [23] In den folgenden zwei Unterkapiteln wird genauer auf die Funktionsweise der verwendeten Agenten eingegangen.

### 3.4.1 A2C Algorithmus

A2C ist ausgeschrieben "Advantage Actor-Critic", was sich wie folgt verstehen lässt:

**Advantage**
Der Agent lernt von den Belohnungen, die er vom Modell erhält. War eine Aktion gut, wird das so notiert, war sie nicht so gut, wird das ebenfalls festgehalten. Bevor jedoch der Agent die Belohnungen erhält, werden diese noch leicht angepasst, sodass zukünftige Belohnungen relativiert werden können [23].

**Actor-Critic**
Dies kombiniert Vorteile von Policy Optimization und Q-Learning. Der Agent schätzt die Belohnung seiner nächsten Aktion mithilfe einer Value-Funktion und einer Policy. Die Value Funktion bestimmt dabei, wie gut es für den Agenten ist, in einem bestimmten Zustand zu sein, und die Policy bestimmt für einen Zustand die beste Aktion [23].

### 3.4.2 PPO Algorithmus

PPO steht abgekürzt für "Proximal Policy Optimization". Das Hauptziel von diesem Agenten ist die Optimierung seiner Policy, also die optimale Aktion für einen gegebenen Zustand zu finden. Im Gegensatz zu A2C setzt PPO zusätzlich den Fokus darauf, seine Policy zu verbessern, ohne schlechtere Resultate insgesamt zu liefern [23].

## 3.5 Eigene MTD – Defender2000

Der Defender2000 Agent soll sinnbildlich eine Verbindung zwischen dem Random Agent und den RL Agenten herstellen. Der Gedanke dabei ist: Wenn Reinforcement Learning eine gute Lösung findet im Vergleich zu Random, kann eine selbst definierte Logik dies replizieren?

Der Random Agent hat in diesem Sinne keine Intelligenz, und die RL Agenten "erlernen" ihre Intelligenz von allein. Mit dem Defender2000 kann nun abgeschätzt werden, wie intelligent die RL Agenten wirklich sind, indem man die Resultate vergleicht. Kurz zusammengefasst ist der Defender2000 ein Stück besser als der Random Agent, überraschenderweise ist er aber bei weitem nicht so gut wie die RL Agenten. Eine ausführlichere Analyse folgt im Kapitel 4.3.

Die Logik, mit der der Defender2000 versucht die Angreifer abzuwehren, ist wie folgt:

**Präventionssystem meldet Angriff**

Entdeckt er einen Angreifer auf einem bestimmten System, wechselt er als erstes das Präventionssystem aus, damit ein Angreifer nicht herleiten kann, welchen Angriff das Präventionssystem nicht erkennt. Gleichzeitig wird der Service, auf dem ein Angreifer erkannt wurde, neugestartet. Somit soll derselbe Angriff den Service nicht mehr unter Kontrolle bringen, dies kann aber nicht garantiert werden. Zudem werden alle vorherigen Services neugestartet, ausgehend vom geknackten Service. Als Beispiel: Wird ein Angreifer auf dem Planner entdeckt, wird sofort das IPS ausgewechselt und der Planner neugestartet. Im nächsten Schritt wird der Vorgänger des Planners neugestartet, also der Authorizer. Als Ausnahme gilt hier, falls ein Angreifer schon wieder auf dem Planner oder sogar in der DB entdeckt wird, wird der hier beschriebene Ablauf sofort von diesen Service ausgeführt, also vom Planner oder der DB aus.

**Kein Angriff registriert**

Sind alle Aufgaben abgearbeitet, also nach einem Angriff die Services neugestartet und das Präventionssystem ausgetauscht, und es wurde kein Angriff mehr registriert, dann startet der Defender2000 einen zufälligen Service neu und wechselt ein zufälliges Präventionssystem aus. Somit soll versucht werden, auch unentdeckte Angriffe zu verhindern oder zumindest erschweren. Diese Logik mit dem zufälligen Austauschen widerspiegelt die Funktionsweise des Random Agenten.

Der Defender2000 Agent agiert somit unvorhersehbar wie der Random Agent, kann aber mithilfe vom Feedback der Präventionssysteme auch Logik und mehr Effektivität in seine Verteidigung einbauen.

# 4 Resultate

## 4.1 Erläuterung verschiedener Modi

Für die Simulationen sind vier verschiedene Modi definiert worden, um erstens mehr Fälle der Industrie abzudecken, und zweitens die Lernfähigkeit der RL Agenten und die Effektivität aller Agenten in verschiedenen Umgebungen zu testen. In allen Simulationen wird nur ein Angreifer simuliert. Dies hat aber keinen Einfluss auf die Agenten, da diese unabhängig von der Anzahl Angreifer operieren.

### 4.1.1 Normal

Der erste Modus beschreibt ein normales Modell. Das heisst, an jedem Zeitschritt kann ein beliebiger Service neugestartet und ein beliebiges Präventionssystem ausgewechselt werden. Für einen Einsatz in echter Umgebung muss natürlich gewährleistet sein, dass das Netzwerk selbst diese Aktionen alle x Sekunden unterstützt, also dass zum Beispiel alle 10 Sekunden ein Service neugestartet werden kann. Eine andere Möglichkeit wäre, den Zeitschritt z.B. auf 60 Sekunden zu setzen. Somit haben Angreifer viel mehr Zeit, die Systeme auszukundschaften, und Schwachstellen ausfindig zu machen, allerdings ist das immer noch viel besser als keine MTD.

### 4.1.2 Nur Service Neustarts

Wie zu erwarten ist unterstützt dieser Modus nur Service Neustarts. Dieser Modus kann angewendet werden, wenn ein Netzwerk noch keine Präventionssysteme implementiert hat, oder das bestehende nicht auswechselt. Wie unsere Versuche gezeigt haben, ist der Unterschied zwischen einem statischen Präventionssystem und keinem Präventionssystem nur klein. Dazu wird angenommen, dass der Angreifer eine Schwachstelle im Präventionssystem findet, und diese folglich immer ausnützen kann.

### 4.1.3 Nur Präventionssystem Auswechslungen

Wie im letzten Modus ist der Name selbsterklärend. Hier können nur die Präventionssysteme ausgewechselt werden, jedoch nicht die Services neugestartet werden.

### 4.1.4 Pause

In diesem Modus können die Agenten für eine gewisse Anzahl Schritte nicht dieselbe Aktion ausführen. Somit kann zum Beispiel definiert werden, dass ein Service nur alle sechs Zeitschritte neugestartet werden kann, und ein Präventionssysteme nur alle drei Zeitschritte ausgewechselt werden kann. Dies soll vor allem die Lernfähigkeit der RL Agenten in erschwerten Umgebungen austesten. Zusätzlich kann das Verhältnis der Service-Neustarts und der Präventionssysteme-Auswechslungen an der Schwerfälligkeit angepasst werden. In diesem Beispiel würde dies bedeuten, dass ein Service Neustart etwa zwei Mal so schwerfällig ist wie eine Präventionssystem Auswechslung.

Theoretisch ist die gezeigte Konfiguration mit einem Zeitschritt von 10 Sekunden und Pausen von 6, respektive 3 Schritten gleichbedeutend, wie eine Konfiguration mit einem Zeitschritt von 30 Sekunden und Pausen von 2, respektive einem Schritt. In den Versuchen zeigt sich allerdings, dass die RL Agenten mit solch speziellen Auflagen nur noch sehr schwerfällig und instabil lernen. Wenn nun eine Pause eingebaut werden soll, sollte man den Zeitschritt so zu wählen, dass zumindest eine Aktion (also Neustart oder Auswechslung) ohne Pause möglich ist. Die Pause der anderen Aktion lässt sich dann vom Verhältnis der Schwerfälligkeiten zwischen den Aktionen herleiten.

Während dem Lernen erhalten die RL Agenten einen Malus, sollten Sie die Pause nicht berücksichtigen. Zudem wird in diesem Falle die Aktion nicht ausgeführt. Auf diese Weise sollen die Agenten lernen, die Pause zu beachten. Andere Methoden wären für die Agenten zu kompliziert zu erlernen [21]. Während dem Simulieren fällt der Malus weg. Eine Aktion, die eine Pause missachtet wird, trotzdem nicht ausgeführt.

## 4.2 Übersicht

### 4.2.1 Einleitung

Für jede der fünf verschiedenen Versuchsreihen sind fünf Tabellen mit Daten zusammengetragen worden. An diesen Tabellen kann abgelesen werden, wie effektiv die eingesetzten Algorithmen unter bestimmten Bedingungen sind. Die Daten in jeder Zelle stellen die durchschnittliche Resultate nach 200 Simulationen dar. Eine Simulation kann maximal 5000 Schritte dauern. Werden 5000 Schritte erreicht, wird angenommen, der Agent kann das Netzwerk unendlich lange verteidigen.

In den Tabellen sind folgende Merkmale aufgeführt, in dieser Reihenfolge:

- Die verwendeten Agenten.
- Die Lernschritte über welche die Agenten insgesamt gelernt haben.
- Die Zeit, die dafür benötigt wurde.
- Die durchschnittlichen Simulationsschritte, bis der Angreifer gewinnt, wobei "5000" bedeutet, dass der RL-Agent immer gewinnt.
- Die durchschnittliche Anzahl Schritte, in welcher der Agent keine Aktion tätigt. Die erste Zahl stellt dabei die Service-Neustarts dar, die zweite Zahl die Präventionssystem-Auswechslungen.
- Die durchschnittliche Belohnung, die der Agent pro Schritt erhält.

Der Defender2000 Agent, der Random Agent und der Static Agent weisen weder Lernschritte noch eine Lernzeit auf, weil diese drei Agenten nicht lernen und ihre interne Logik statisch ist. Deshalb werden diese Agenten nicht besser oder schlechter über die Zeit, und somit können die Resultate auf eine Zeile zusammengefasst werden.

### 4.2.2 Normal

Bei dieser Versuchsreihe sind die Service Neustarts und die Präventionssystem Auswechslungen aktiviert. Es gibt keine Pause. Die Resultate der Simulation sind in Tabelle 7 aufgelistet.

| Agent | Lern-schritte | Lernzeit | ø Sim. Schritte | ø Null Action | | ø Belohnung |
|---|---|---|---|---|---|---|
| A2C | 10^3 | 0.8 s | 122.6 | 0.14 | 0.34 | 29.5 |
| | 10^4 | 8.4 s | 2498.6 | 0.16 | 0.06 | 46.7 |
| | 10^5 | 74.8 s | 4981.0 | 0.14 | 0.02 | 52.5 |
| | 10^6 | 747.4 s | 5000.0 | 0.00 | 0.00 | 52.5 |
| | 10^7 | 8236.3 s | 5000.0 | 0.00 | 0.00 | 52.6 |
| PPO | 10^3 | 1.6 s | 140.6 | 0.12 | 0.31 | 30.2 |
| | 10^4 | 15.1 s | 2835.7 | 0.14 | 0.12 | 46.9 |
| | 10^5 | 147.5 s | 4975.7 | 0.33 | 0.00 | 54.1 |
| | 10^6 | 1494.6 s | 4981.2 | 1.00 | 0.00 | 57.5 |
| | 10^7 | 14228.5 s | 4990.2 | 1.00 | 0.00 | 57.6 |
| Defender2000 | 0 | 0.0 s | 173.7 | 0.10 | 0.28 | 34.5 |
| Random | 0 | 0.0 s | 117.7 | 0.13 | 0.33 | 28.9 |
| Static | 0 | 0.0 s | 22.2 | 1.00 | 1.00 | 7.7 |

*Tabelle 7: Resultate "Normal"*

Wie zu sehen ist, sind die RL Agenten um Magnituden besser als die nicht RL Agenten im Verteidigen des Netzwerkes. Bemerkenswert ist, dass der PPO Agent gelernt hat, das Netzwerk nur mit Präventionssystem Auswechslungen zu verteidigen. Das zeigt sich daran, dass die durchschnittliche Null Aktion bzgl. Service Neustarts auf 1.0 geht (also nie eine Aktion auf alle Zeitschritte) und die Null Aktionen der Präventionssysteme auf 0.0 geht (also immer eine Aktion pro Zeitschritt). Der A2C Agent dagegen benötigt beide Aktionen, um den Angreifer fernzuhalten.

Allerdings ist beim A2C fast garantiert, dass der Angreifer nicht Daten extrahieren kann, da in 200 Simulationen immer 5000 Schritte erreicht wurden. Der PPO Agent kann in einer oder zwei Simulationen den Angreifer nicht von seinem Ziel abhalten. Für sehr kritische Systeme ist eher A2C empfehlenswert. Für eine immer noch sehr gute Verteidigung, die keine Service Neustarts benutzen soll, ist der PPO der gesuchte Agent.

### 4.2.3 Nur Service Neustarts

Bei dieser Versuchsreihe sind nur die Service Neustarts aktiviert. Es gibt keine Pause. In Tabelle 8 sind die Resultate der Simulation abgebildet.

| Agent | Lern-schritte | Lernzeit | ø Sim. Schritte | ø Null Action | | ø Belohnung |
|---|---|---|---|---|---|---|
| A2C | 10^3 | 1.3 s | 28.4 | 0.13 | 1.00 | 7.9 |
| | 10^4 | 16.1 s | 29.3 | 0.17 | 1.00 | 8.4 |
| | 10^5 | 105.6 s | 4963.8 | 0.00 | 1.00 | 20.1 |
| | 10^6 | 835.9 s | 5000.0 | 0.00 | 1.00 | 20.1 |
| | 10^7 | 7992.8 s | 5000.0 | 0.00 | 1.00 | 20.1 |
| PPO | 10^3 | 2.8 s | 27.6 | 0.11 | 1.00 | 6.7 |
| | 10^4 | 20.9 s | 46.6 | 0.07 | 1.00 | 11.3 |
| | 10^5 | 233.6 s | 244.9 | 0.03 | 1.00 | 3.5 |
| | 10^6 | 1371.8 s | 5000.0 | 0.00 | 1.00 | 20.1 |
| | 10^7 | 17023.3 s | 4989.0 | 0.00 | 1.00 | 20.1 |
| Defender2000 | 0 | 0.0 s | 35.0 | 0.08 | 1.00 | 11.3 |
| Random | 0 | 0.0 s | 28.6 | 0.12 | 1.00 | 7.8 |
| Static | 0 | 0.0 s | 21.7 | 1.00 | 1.00 | 7.4 |

*Tabelle 8: Resultate "Nur Services"*

Überraschenderweise hat der Agent A2C in diesem Modus nach 10^4 Lernschritten immer noch grosse Mühe, den Angreifer abzuwehren, der Defender2000 ist etwa 20% effektiver. Im normalen Modus ist der A2C nach 10^4 Lernschritten um mehr als das 10 fache besser als der Defender2000. Nach 10^5 Lernschritte ist der A2C Agent fast gleich effektiv wie im normalen Modus und kann den Angreifer fast immer abhalten.

Auch der PPO Agent hat grosse Mühe zu lernen im Vergleich zum normalen Modus. Er ist auch nach 10^5 Schritten bei weitem noch nicht bei seiner Höchstleistung. Erst nach 10^6 Lernschritte bekommt er den Angreifer unter Kontrolle. Wahrscheinlich ist es ein Zufall, dass dann bei 10^7 Lernschritten die durchschnittlichen Simulationsschritte wieder weniger werden als bei 10^6 Lernschritten.

Aber auch hier sind die RL Agenten mit genügend Lernzeit viel besser als die nicht RL Agenten. In diesem Modus empfiehlt sich allerdings immer, den A2C Agent einzusetzen, er ist in allen Bereichen besser als der PPO Agent. Bemerkenswert ist, dass der Random Agent und Defender2000 Agent viel ineffektiver wurden im Vergleich zum normalen Modus.

### 4.2.4 Nur Präventionssystem Auswechslungen

Bei dieser Versuchsreihe sind nur die Präventionssystem Auswechslungen aktiviert. Es gibt keine Pause. In folgender Tabelle 9 sind die Resultate dieser Simulation zu sehen.

| Agent | Lern-schritte | Lernzeit | ø Sim. Schritte | ø Null Action | | ø Belohnung |
|---|---|---|---|---|---|---|
| A2C | 10^3 | 0.8 s | 95.1 | 1.00 | 0.26 | 34.3 |
| | 10^4 | 8.3 s | 1007.5 | 1.00 | 0.16 | 49.3 |
| | 10^5 | 77.4 s | 5000.0 | 1.00 | 0.00 | 57.5 |
| | 10^6 | 737.0 s | 5000.0 | 1.00 | 0.00 | 57.6 |
| | 10^7 | 8218.6 s | 5000.0 | 1.00 | 0.00 | 57.6 |
| PPO | 10^3 | 1.5 s | 78.2 | 1.00 | 0.29 | 31.9 |
| | 10^4 | 16.4 s | 1781.5 | 1.00 | 0.12 | 51.3 |
| | 10^5 | 146.1 s | 4993.7 | 1.00 | 0.01 | 57.1 |
| | 10^6 | 1517.9 s | 4989.3 | 1.00 | 0.00 | 57.5 |
| | 10^7 | 17455.0 s | 5000.0 | 1.00 | 0.00 | 57.5 |
| Defender2000 | 0 | 0.0 s | 77.6 | 1.00 | 0.26 | 33.4 |
| Random | 0 | 0.0 s | 55.2 | 1.00 | 0.33 | 26.3 |
| Static | 0 | 0.0 s | 21.2 | 1.00 | 1.00 | 6.2 |

*Tabelle 9: Resultate "Nur Präventionssysteme"*

Im vorherigen Modus konnte der A2C Agent den PPO Agenten in allen Bereichen ausstechen. In diesem Modus jedoch erweist sich der PPO Agent als etwas effektiver. Im Vergleich zum normalen Modus sind beide RL Agenten etwas langsamer im Lernen, allerdings hat der Agent A2C schlussendlich die höhere Belohnung pro Schritt als im normalen Modus, bei immer noch 5000 Simulationsschritten. Das heisst der Agent ist gleich effektiv wie im normalen Modus, ist zusätzlich aber noch ein wenig effizienter. Wenn dieser Agent also etwas besser geführt ist, kann er noch bessere Resultate erzielen.

Der PPO Agent liefert exakt dieselben Resultate wie im normalen Modus. Somit kann davon ausgegangen werden, dass in beiden Modi dieselbe Lösung gefunden wird. Wieder wenig überraschend sind die nicht RL Agenten bei weitem nicht so effektiv das Netzwerk vor Angriffen zu schützen. Und zusätzlich sind sie in diesem Modus nur etwa halb so effizient wie im normalen Modus.

### 4.2.5 Pause

Bei dieser Versuchsreihe sind wieder Service Neustarts und Präventionssystem Auswechslungen aktiviert. Allerdings darf derselbe Service nur alle 6 Schritte neugestartet werden. Dasselbe Präventionssystem (IPS oder DLPS) kann alle 3 Schritte ausgewechselt werden. Die Resultate davon sind in nachstehender Tabelle 10 aufgeführt.

| Agent | Lern-schritte | Lernzeit | ø Sim. Schritte | ø Null Action | | ø Belohnung |
|---|---|---|---|---|---|---|
| A2C | 10^3 | 1.0 s | 321.6 | 0.47 | 0.58 | 32.3 |
| | 10^4 | 7.7 s | 251.0 | 0.56 | 0.62 | 32.2 |
| | 10^5 | 78.4 s | 1797.7 | 0.92 | 0.66 | 35.1 |
| | 10^6 | 690.5 s | 2028.9 | 1.00 | 0.69 | 34.8 |
| | 10^7 | 6908.7 s | 214.2 | 1.00 | 0.70 | 31.6 |
| PPO | 10^3 | 1.6 s | 79.6 | 0.46 | 0.61 | 24.1 |
| | 10^4 | 16.7 s | 507.0 | 0.77 | 0.63 | 34.1 |
| | 10^5 | 149.5 s | 1620.8 | 0.93 | 0.69 | 35.2 |
| | 10^6 | 1363.7 s | 131.3 | 1.00 | 0.73 | 29.3 |
| | 10^7 | 14167.0 s | 998.0 | 0.97 | 0.69 | 34.7 |
| Defender2000 | 0 | 0.0 s | 207.8 | 0.21 | 0.50 | 31.4 |
| Random | 0 | 0.0 s | 128.4 | 0.24 | 0.54 | 27.2 |
| Static | 0 | 0.0 s | 22.2 | 1.00 | 1.00 | 6.8 |

*Tabelle 10: Resultate "Pause"*

In diesem Modus haben die RL Agenten nun die grösste Mühe, eine gute Lösung zu finden. Sie lernen im Vergleich zu allen anderen Modi ein Stück langsamer. Noch viel aussergewöhnlicher ist jedoch, dass Agenten nach mehr Zeit nicht nur besser werden, sondern zum Teil auch schlechter. Auf diesen Aspekt wird im Kapitel 5 nochmals genauer eingegangen.

Interessant ist zudem, dass der Defender2000 Agent und der Random Agent etwas besser sind als im normalen Modus. Das könnte daher stammen, dass die Aktionen mehr verteilt werden, weil die Pause beachtet werden muss. Es kann nicht mehr dieselbe Aktion nacheinander ausgeführt werden, was diesen zwei Agenten anscheinend etwas auf die Sprünge hilft. Trotzdem sind sie immer noch etwa 10 Mal schlechter als die RL Agenten, weshalb man besser die RL Agenten genauer analysieren soll.

### 4.2.6 Falsches Lernen

In folgendem Versuch wird mit einem simplerem Angriff gelernt und simuliert (einfach), und danach wird auf die gelernten Agenten nochmals einen effektiveren Angriff gestartet (schwer). Alle Wahrscheinlichkeiten im simpleren Angriff sind exakt die Hälfte im Vergleich zu den Wahrscheinlichkeiten des effektiveren Angriffs. In diesem Versuch sind wiederum nur die Service Neustarts aktiviert, ansonsten wären die Auswirkungen nur sehr gering. Zudem werden nur die RL Agenten simuliert, die anderen Agenten können nicht lernen. In Tabelle 11 sind die Resultate dargestellt.

| Agent | Lern-schritte | Lernzeit | ø Sim. Schritte | ø Null Action | | ø Belohnung |
|---|---|---|---|---|---|---|
| A2C - einfach | 10^6 | 755.5 s | 5000.0 | 0.00 | 1.00 | 37.5 |
| PPO - einfach | 10^6 | 1626.4 s | 5000.0 | 0.00 | 1.00 | 37.5 |
| A2C - schwer | 0 | 0.0 s | 5000.0 | 0.00 | 1.00 | 20.1 |
| PPO - schwer | 0 | 0.0 s | 4850.2 | 0.00 | 1.00 | 20.1 |

*Tabelle 11: Resultate "Falsches Lernen"*

Im Vergleich zum "Nur Service Neustarts" Modus, wo mit einem schwieriger Angriff gelernt und danach simuliert wird, sind beide Agenten hier im einfachen Modus viel effektiver. Sie starten zwar auch bei jedem Schritt einen Service neu (das heisst sind gleich effizient), aber ihre Belohnung ist fast doppelt so hoch. Das bedeutet, dass der Angreifer im Schnitt weniger weit ins Netzwerk eingedrungen ist, also sind sie effektiver.

Überraschend ist, dass auch wenn die Agenten mit einem vereinfachten Angriff lernen und danach schwierigere Angriffe stattfinden, die Agenten exakt dieselben Resultate liefern, wie wenn sie ursprünglich mit den schwierigeren Angriffen lernten. Somit kann gezeigt werden, dass die Wahrscheinlichkeiten des Angriffs beim Lernen keine grosse Rolle spielen, solange sie keine zu grossen Differenzen aufzeigen. In diesem Versuch ist z.B. die eine Wahrscheinlichkeit beim simplen Angriff bei 0.3, beim effektiveren bei 0.6. Fatale Folgen hätte es voraussichtlich, wenn hier die Wahrscheinlichkeit des simpleren Angriffs bei 0.1 wäre. Dies müsste aber auch wieder bestätigt werden.

## 4.3 Evaluation der Resultate

Auf den folgenden Seiten sind nochmals die wichtigsten Erkenntnisse in Bezug auf die fünf Versuche in Form von Graphen festgehalten.

### 4.3.1 Agenten Performance bzgl. Lernzeit

Als erstes wird die Effektivität der RL Agenten im Vergleich zu den nicht RL Agenten visualisiert. Wie in Abbildung 11 zu sehen ist, sind die RL Agenten viel effektiver darin, den Angreifer abzuwehren. Aus diesem Grund ist die y-Achse logarithmisch dargestellt, ansonsten wären die Unterschiede zwischen den nicht RL Agenten nicht mehr sichtbar.

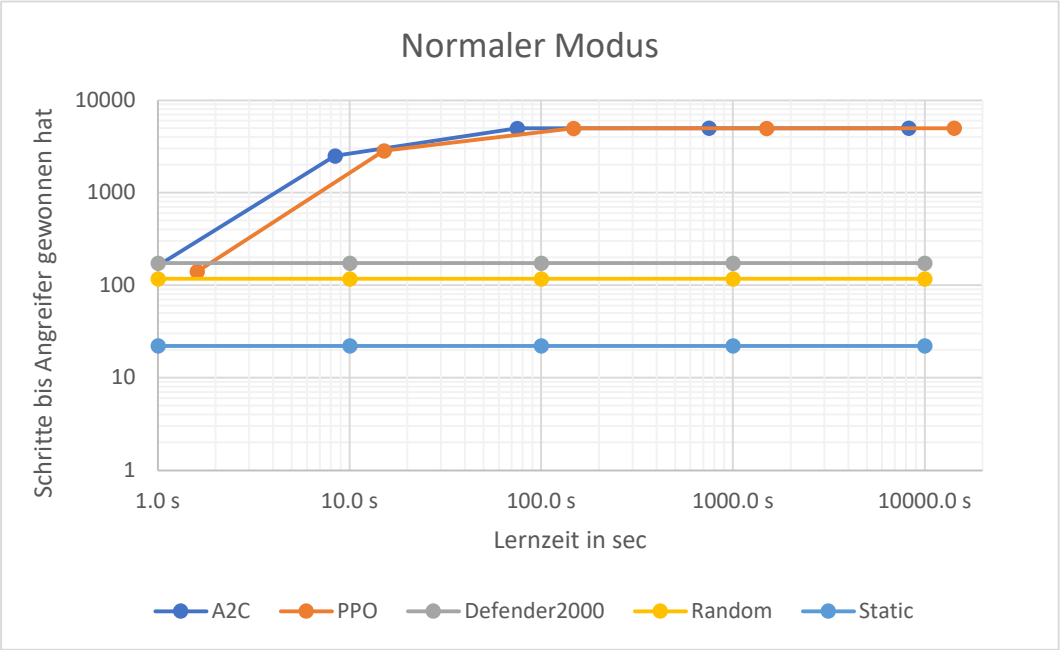



*Abbildung 11: Schritte bis der Angreifer gewinnt im Vergleich zu den Lernschritten*

Bemerkenswert ist, dass der A2C Agent schon nach einer Sekunde Lernzeit bessere Resultate liefert als der Random Agent, und etwa dieselben wie der Defender2000. Auch der PPO Agent ist nach nur 2 Sekunden Lernzeit effektiver als der Defender2000.

Bei der Beobachtung der Lernzeit fällt auf, dass PPO im Durchschnitt etwa doppelt so viel Zeit benötigt wie A2C, um die erforderlichen Lernschritte zu absolvieren. Trotz der längeren Zeit die PPO benötigt, um den Agenten zu trainieren, liefern die Agenten ungefähr dieselben Resultate.

### 4.3.2 Agenten Performance bzgl. Modus

In dieser Graphik werden die durchschnittlichen Schritte abgebildet, die der Angreifer braucht, um Daten zu extrahieren. Was nicht sichtbar ist, ist, dass die beiden RL Agenten je nach Modus 10^5 oder 10^6 Lernschritte brauchen, um diese Leistungen zu erreichen.

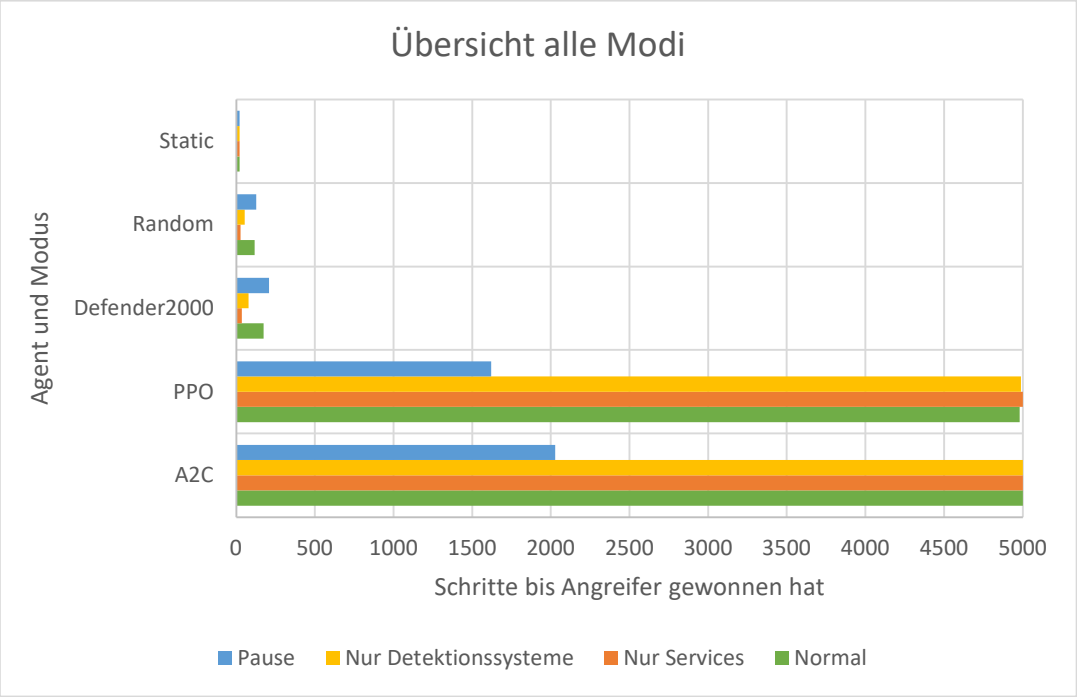



*Abbildung 12: Schritte bis der Angreifer gewinnt in allen Modi*

In Abbildung 12 ist ersichtlich, dass die Agenten Static, Random und Defender2000 in allen vier getesteten Modi grosse Probleme damit haben, den Angreifer über 5000 Schritte abzuwehren. Während der Defender 2000 doch ein wenig bessere Resultate liefert als der Random Agent, ist er zusammen mit ihm aber doch weit weniger wirksam als die RL Agenten. Der Static Agent, also keine MTD, ist trotzdem in jedem Fall nochmals weniger effektiv.

Die RL Agenten PPO und A2C schnitten in allen vier getesteten Modi sehr gut ab. Sobald die Anzahl Lernschritte grösser wurde, konnten sie das Netzwerk aktiv verteidigen. Nach 10^5 respektive 10^6 Lernschritten ist schliesslich der Erfolg bei beiden RL Agenten in den Modi "Normal", "Nur Services" und "Nur Präventionssysteme" garantiert. Lediglich im Pausenmodus konnten A2C und PPO den Angreifer nicht unter Kontrolle halten. Trotzdem sind sie auch in diesem Modus etwa zehn Mal effektiver als die nicht RL Agenten.

### 4.3.3 Performance bzgl. falschem Lernen

In den folgenden zwei Abbildungen werden nochmals die Auswirkungen visualisiert, wenn die RL Agenten mit falschen Wahrscheinlichkeiten trainiert werden. Als Beispiel: Der Angreifer hat in der Simulation eine Chance von 0.6 zum nächsten Service zu gelangen, aber die Agenten haben mit dem Wert 0.3 trainiert. In Abbildung 13 wird die Belohnung dargestellt, die die Agenten im Durchschnitt erhalten haben. Abbildung 14 wird gezeigt, wie lange die Agenten den Angreifer abwehren können.

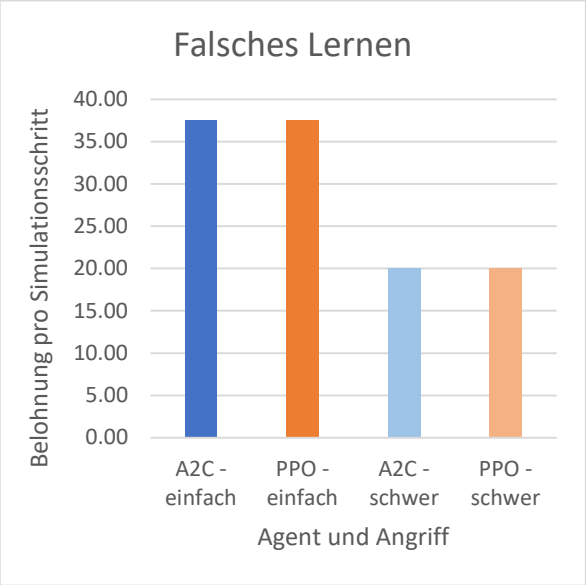



*Abbildung 14: Belohnung pro Schritt bei falschem Lernen*

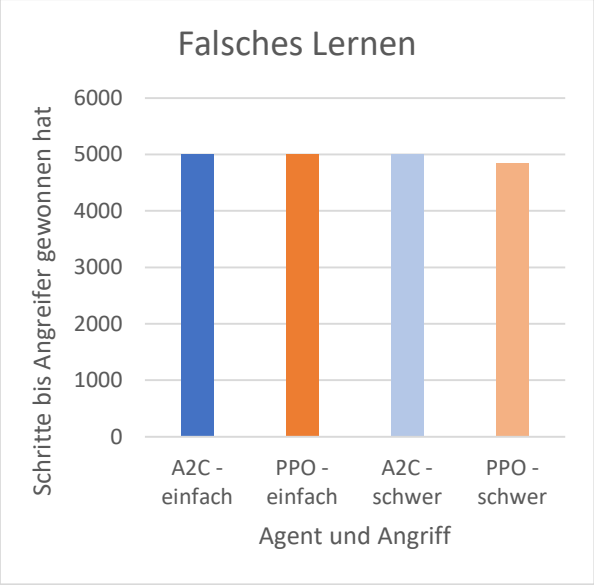



*Abbildung 14: Schritte bis der Angreifer gewinnt bei falschem Lernen*

Wie sich zeigt, kommt der Angreifer auch mit verbesserten Chancen nicht wirklich zu seinem Ziel. Nur beim PPO kann der Angreifer mit verbesserten Chancen in sehr wenigen der 200 Simulationen Daten extrahieren. Das kann allerdings auch Zufall sein. Der A2C wäre in solchen Situationen trotzdem die bessere Wahl mit diesen ersten Resultaten.

Abbildung 13 zeigt, dass beide Agenten nur noch etwa die Hälfte der Belohnung im Durchschnitt pro Zeitschritt erhalten. In Kombination damit, dass die Agenten noch gleich viele Aktionen durchführen wie vorhin, deutet die Halbierung der Belohnung darauf hin, dass der Angreifer weiter ins Netzwerk eindringen kann. Eine Belohnung von 37 stellt sich wie folgend zusammen:

- Pauschal +60 Belohnung pro Schritt (bias)
- -5 weil bei jedem Schritt ein Service neugestartet wird
- -X weil der Angreifer ins System eindringt

Ohne das Eindringen wären es also +55. Das ist eine Differenz von 18 zu +37. Dringt der Angreifer eine Stufe ins Netzwerk, resultiert das in -50. Also dringt der Angreifer im einfachem Angriff pro Zeitschritt im Durchschnitt 18/50 Stufen ins Netzwerk herein. Im schweren Angriff beträgt die Differenz 25, von +55 zu +20. Somit dingt der Angreifer pro Zeitschritt 25/50 Stufen ins Netzwerk hinein. Bei 50/50 würde er pro Zeitschritt eine Stufe eindringen, also hätte er nach 4 Simulationsschritten gewonnen. Falsches Lernen ist somit nicht fatal, wie in Abbildung 14 ersichtlich ist, allerdings werden die Chancen des Angreifers grösser, Daten zu extrahieren.

# 5 Diskussion

**RL MTD Effektivität und weiterführende Forschungsarbeiten**

Wie im vorherigen Kapitel ausführlich hergeleitet und aufgezeigt, sind in die RL Agenten in allen Modi und allen Versuchen markant effektiver als die nicht RL Agenten. Im besten Falle, also im normalen Modus, sind die RL Agenten auch schon nach einer Sekunde Lernzeit besser darin, das Netzwerk zu schützen, als die selbst definierte MTD (Defender2000), oder die rein zufällige MTD (Random). Die Agenten können daher nicht nur exzellent Netzwerke verteidigen, bei denen die Services immer gleichbleiben, sondern können auch "dynamische" Netzwerke sichern. Als Beispiel dafür kann IaaS (Infrastructure as a Service) on Demand genannt werden. Dabei wird, wie zu erwarten ist, auf Anfrage die Infrastruktur bereitgestellt. Im gleichen Moment kann der Agent das neue Netzwerk-Layout erlernen. Das führt dazu, dass eigentlich im selben Moment, wie die neue Infrastruktur bereit ist, auch ein Agent trainiert ist und eine effektive MTD bereitstellt. Dieser Anwendungsfall sollte für künftige Forschungsarbeiten in Betracht gezogen werden.

**Sein oder Schein**

Sind die RL MTD Lösungen nun wirklich viel besser als die nicht RL MTD Lösungen? Nach eigenen Überlegungen ist es wahrscheinlich ziemlich einfach für die RL Agenten, eine sehr gute Lösung für unser MTD Problem zu finden. Diese Lösung entsteht, indem der Agent jede Runde das IPS austauscht. Daraus folgt, dass der Angreifer bei jeden Schritt zu 0.9911 erkannt und vom System gekickt wird. Auch wenn sich das fast zu gut anhört, ist es trotzdem nicht unrealistisch. Denn wenn der Angreifer nie weiss, was er angreifen kann (Attack Surface Shifting) und welche Angriffe unentdeckt bleiben (Prevention Surface Shifting), wie soll er so einen erfolgreichen Angriff starten?

Wenn nun ein Angreifer die Präventionssysteme ausfindig machen kann und einen Angriff findet, der von keinem Präventionssystem entdeckt wird, kann er somit alle Präventionssysteme ausspielen. Dies ist aber ein wohl bekanntes Problem im Bereich MTD (keine gemeinsamen Schwachstellen) [2]. Zudem wirkt sich das auf RL MTDs und "normale" MTDs gleichermassen aus.

Zum wohl interessantesten Versuch, dem Modus "Pause": Im Modus "Pause" wird verhindert, dass bei folgenden Schritten nicht derselbe Service neugestartet und nicht dasselbe Präventionssystem ausgewechselt werden kann. Doch auch mit speziellen Auflagen und Einschränkungen, die schlecht sind für ein effektives Lernen, kann der Angreifer doch erst nach ca. 2000 Schritte Daten extrahieren. Das ist immer noch etwa zehn Mal so effektiv wie die nicht RL Agenten. Zudem zeigt das, dass die zwei verwendeten Agenten von Stable Baselines 3 auch gut ohne Anpassungen der Hyperparameter einigermassen verwendet werden können.

**Agenten und Hyperparameter**

Wie aufgezeigt, sind die RL Agenten im Pausenmodus nicht sehr effektiv, und ihr Lernverhalten lässt bezüglich Stabilität Wünsche offen. Eine schlechte Stabilität im Lernen führt dazu, dass die Agenten mit mehr Lernschritten mal besser und mal schlechter werden im Verteidigen. Hier bräuchte man erheblich mehr Zeit, um optimalere Hyperparameter zu finden, um die Stabilität zu verbessern oder gar die Funktionsweise des Lernens der RL Agenten zu überarbeiten. Diese Hyperparameter steuern zum Beispiel wie schnell ein Agent lernt, also wie schnell er seine Policy überarbeitet. Andere Hyperparameter steuern zum Beispiel, wie "neugierig" die Agenten sind. Das heisst, ob sie immer dasselbe machen, oder auch neue Aktionen ausprobieren. Es können noch viel mehr Eigenschaften des Lernens über diese Hyperparameter gesteuert werden. Allerdings liegt das Erklären, Herleiten und Simulieren mit verschiedenen Hyperparametern nicht im Zeitrahmen dieser Arbeit.

**Ausblick**

Obwohl diese Arbeit klare Vorteile von RL im Bereich MTD aufzeigt, sind noch einige Punkte offen, bevor eine solche MTD wirklich verwendet werden sollte. Nachfolgend sind einige Aspekte aufgelistet, die bei einer Implementation beachtet werden müssen.

- Die Funktionsweise der verwendeten Präventionssysteme in dieser Arbeit muss eventuell angepasst werden, je nach dem, welche Systeme dann in Wirklichkeit verwendet werden.
- Der "Neustart" eines Services soll bei einer Implementation dieselben Auswirkungen haben wie in dieser Arbeit, also den Angreifer einen Service zurückversetzen und die alten Schwachstellen austauschen. Das kann zum Beispiel mit einer OS Rotation verwirklicht werden. Ist die implementierte Funktionsweise nicht dieselbe, muss die Logik des Codes angepasst werden (in graph.py und in node.py).
- Der Austausch eines Präventionssystems soll bei einer Implementation dieselben Auswirkungen haben wie in dieser Arbeit. Das heisst, der Angreifer muss seine Angriffe anpassen, um nicht entdeckt zu werden. Zudem wird der Angreifer aus dem Netzwerk geworfen. Sollten diese zwei Aspekte nicht erfüllt werden, muss die Logik des Codes angepasst werden (in graph.py oder in prevention_system.py). Die Wahrscheinlichkeit, dass der Angriff entdeckt wird, soll auf das verwendete System angepasst werden (in attack_graph.json).

Zusätzlich zu beachten ist, dass diese MTD nur für ein eher simples Netzwerk und einen sehr vorhersehbaren Angriff entwickelt ist. Für weiterfolgende Forschungsarbeiten wären folgende Aspekte sehr relevant

**Threat Modelling:** Die RL MTD soll im Kontext eines gesamten Netzwerkes bewerten werden. Ein ähnliches Netzwerk wie in diese Arbeit, soll aus der Praxis genommen werden und alle Sicherheitsaspekte analysiert werden. Somit kann gezeigt werden, ob, oder wann sich eine RL MTD lohnt und auf welche Aspekte sich diese auswirkt.

**IaaS on Demand:** Die Effektivität von RL MTD soll sich bei ändernden Netzwerken beweisen. Dieser neue Use-Case, der in dieser Arbeit nicht untersucht werden konnte, sollte bei immer mehr Cloud Lösungen von grosser Bedeutung sein.

**Agenten Hyperparameter:** Wie im Modus "Pause" zu erkennen ist, sind die Hyperparameter noch nicht ausgereift. Für bessere Resultate bei komplexeren Netzwerken oder komplexeren Vorgaben (wie z.B. ein Pausenmodus) sind verbesserte Hyperparameter erforderlich. Mit den verwendeten Hyperparameter dieser Arbeit kann der Erfolg für komplexere Netzwerke nicht garantiert werden.

**Verschiedene Bereitschaftsmodi:** In dieser Arbeit ist die MTD immer auf demselben Bereitschaftsniveau. Eine interessante und wohl auch viel effizientere Lösung wären verschiedene Bereitschaftsmodi. Als Beispiel könnte ein Modus "nur" alle 5 Minuten eine Aktion ausführen, wobei ein zweiter Modus alle 5 Sekunden eine Aktion ausführt, sobald ein Angriff entdeckt wird.

**Parallele Präventionssysteme:** Die Präventionssysteme werden in gezeigtem Modell nur einzeln verwendet. Für weiterführende Forschungsarbeiten können die kürzlichen Fortschritte in Multiple Intrusion Detection Systems miteinbezogen werden [24]. Damit werden voraussichtlich noch effizientere Verteidigungen ermöglicht.

# 6 Verzeichnisse

## 6.1 Literaturverzeichnis


[1] G. Cai, B. Wang, W. Hu, und T. Wang, „Moving target defense: state of the art and characteristics“, *Frontiers Inf Technol Electronic Eng*, Bd. 17, Nr. 11, S. 1122–1153, Nov. 2016, doi: 10.1631/FITEE.1601321.

[2] S. Sengupta, A. Chowdhary, A. Sabur, A. Alshamrani, D. Huang, und S. Kambhampati, „A Survey of Moving Target Defenses for Network Security“, *IEEE Communications Surveys Tutorials*, Bd. 22, Nr. 3, S. 1909–1941, thirdquarter 2020, doi: 10.1109/COMST.2020.2982955.

[3] P. Manadhata und J. M. Wing, „An Attack Surface Metric“. Juli 2005.

[4] J.-H. Cho *u. a.*, „Toward Proactive, Adaptive Defense: A Survey on Moving Target Defense“, *arXiv:1909.08092 [cs]*, Sep. 2019, Zugegriffen: Sep. 30, 2020. [Online]. Verfügbar unter: http://arxiv.org/abs/1909.08092.

[5] D. Evans, A. Nguyen-Tuong, und J. Knight, „Effectiveness of Moving Target Defenses“, in *Moving Target Defense*, Bd. 54, S. Jajodia, A. K. Ghosh, V. Swarup, C. Wang, und X. S. Wang, Hrsg. New York, NY: Springer New York, 2011, S. 29–48.

[6] Q. Jia, K. Sun, und A. Stavrou, „MOTAG: Moving Target Defense against Internet Denial of Service Attacks“, Juli 2013, S. 1–9, doi: 10.1109/ICCCN.2013.6614155.

[7] A. Clark, K. Sun, L. Bushnell, und R. Poovendran, „A Game-Theoretic Approach to IP Address Randomization in Decoy-Based Cyber Defense“, Nov. 2015, S. 3–21, doi: 10.1007/978-3-319-25594-1_1.

[8] I. Arce und G. McGraw, „Guest Editors' Introduction: Why Attacking Systems Is a Good Idea“, *IEEE Security Privacy*, Bd. 2, Nr. 4, S. 17–19, Juli 2004, doi: 10.1109/MSP.2004.46.

[9] B. Danev, R. J. Masti, G. O. Karame, und S. Capkun, „Enabling secure VM-vTPM migration in private clouds“, in *Proceedings of the 27th Annual Computer Security Applications Conference on - ACSAC '11*, Orlando, Florida, 2011, S. 187, doi: 10.1145/2076732.2076759.

[10] K. M. Carter, H. Okhravi, und J. Riordan, „Quantitative Analysis of Active Cyber Defenses Based on Temporal Platform Diversity“, *arXiv:1401.8255 [cs]*, Jan. 2014, Zugegriffen: Okt. 22, 2020. [Online]. Verfügbar unter: http://arxiv.org/abs/1401.8255.

[11] W. Peng, F. Li, C. Huang, und X. Zou, „A moving-target defense strategy for Cloud-based services with heterogeneous and dynamic attack surfaces“, in *2014 IEEE International Conference on Communications (ICC)*, Juni 2014, S. 804–809, doi: 10.1109/ICC.2014.6883418.

[12] R. S. Sutton und A. G. Barto, *Reinforcement Learning, second edition: An Introduction*. MIT Press, 2018.

[13] L. P. Kaelbling, M. L. Littman, und A. W. Moore, „Reinforcement Learning: A Survey“, *Journal of Artificial Intelligence Research*, Bd. 4, S. 237–285, Mai 1996, doi: 10.1613/jair.301.

[14] S. B. Thrun, „THE ROLE OF EXPLORATION IN LEARNING CONTROL“, S. 27.

[15] T. Zhang, X. Kuang, Z. Zhou, H. Gao, und C. Xu, „An Intelligent Route Mutation Mechanism against Mixed Attack Based on Security Awareness“, in *2019 IEEE Global Communications Conference (GLOBECOM)*, Dez. 2019, S. 1–6, doi: 10.1109/GLOBECOM38437.2019.9014119.

[16] Z. Zhou, C. Xu, X. Kuang, T. Zhang, und L. Sun, „An Efficient and Agile Spatio-Temporal Route Mutation Moving Target Defense Mechanism“, in *ICC 2019 - 2019 IEEE International Conference on Communications (ICC)*, Mai 2019, S. 1–6, doi: 10.1109/ICC.2019.8761927.

[17] X. Chai, Y. Wang, C. Yan, Y. Zhao, W. Chen, und X. Wang, „DQ-MOTAG: Deep Reinforcement Learning-based Moving Target Defense Against DDoS Attacks“, in *2020 IEEE Fifth International Conference on Data Science in Cyberspace (DSC)*, Juli 2020, S. 375–379, doi: 10.1109/DSC50466.2020.00065.

[18] R. Zhuang, S. Zhang, A. Bardas, S. A. DeLoach, X. Ou, und A. Singhal, „Investigating the application of moving target defenses to network security“, in *2013 6th International Symposium on Resilient Control Systems (ISRCS)*, Aug. 2013, S. 162–169, doi: 10.1109/ISRCS.2013.6623770.

[19] P. Ammann, D. Wijesekera, und S. Kaushik, „Scalable, Graph-Based Network Vulnerability Analysis“, S. 8.
[20] OpenAI, „Gym: A toolkit for developing and comparing reinforcement learning algorithms“. https://gym.openai.com (zugegriffen Nov. 16, 2020).
[21] „Welcome to Stable Baselines docs! - RL Baselines Made Easy — Stable Baselines 2.10.2a0 documentation“. https://stable-baselines.readthedocs.io/en/master/ (zugegriffen Nov. 16, 2020).
[22] S. S. Dhaliwal, A.-A. Nahid, und R. Abbas, „Effective Intrusion Detection System Using XGBoost“, *Information*, Bd. 9, Nr. 7, Art. Nr. 7, Juli 2018, doi: 10.3390/info9070149.
[23] S. AI, „Reinforcement Learning algorithms — an intuitive overview“, *Medium*, Feb. 18, 2019. https://medium.com/@SmartLabAI/reinforcement-learning-algorithms-an-intuitive-overview-904e2dff5bbc (zugegriffen Nov. 03, 2020).
[24] Vrushank M. Shah und A. K. Agarwal, „Reliable Alert Fusion of Multiple Intrusion Detection Systems“, *International Journal of Network Security*, Bd. 19, Nr. 2, März 2017, doi: 10.6633/IJNS.201703.19(2).03.

## 6.2 Abbildungsverzeichnis



## 6.3 Tabellenverzeichnis



## 6.4 Abkürzungsverzeichnis

| ***Abkürzung*** | **Ausgeschrieben** | **Übersetzt / Bedeutung** |
|---|---|---|
| *MTD* | Moving Target Defense | Bewegliches Ziel |
| *IP* | Internet Protocol | - |
| *TCP/IP* | Transmission Control Protocol | - |
| *VM* | Virtual Machine | Virtueller Computer |
| *IPS* | Intrusion Prevention System | Penetrationspräventionssystem |
| *DLPS* | Data Loss Prevention System | Datenverlustpräventionssystem |
| *RL* | Reinforcement Learning | Art von Machine Learning |
| *IaaS* | Infrastructure as a Service | Externe Infrastruktur nutzen |

# 7 Anhang

## 7.1 Projektmanagement

Während der Dauer des Projekts hat sich das Projektteam wöchentlich mit den Betreuern getroffen. An diesen Besprechungen geben die Betreuer Feedback zum Stand des Projekts, offene Aspekte der Arbeit werden abgeklärt und neue Erkenntnisse zum Projekt werden präsentieren.

### 7.1.1 Aufgabenstellung

zhaw School of Engineering — [ ONLINE ADMINISTRATION PRAKTISCHE ARBEITEN ] — [ DEPT. T ADMIN TOOLS ]

zurück Logout

**Projektarbeit 2020 - HS: PA20_tebe_02**

**Allgemeines:**

| | |
|---|---|
| **Titel:** | Machine Learning driven control of Moving Target Defense (MTD) for cybersecurity |
| **Anzahl Studierende:** | 2 |
| **Durchführung in Englisch möglich:** | Ja, die Arbeit kann vollständig in Englisch durchgeführt werden und ist auch für Incomings geeignet. |

**Betreuer:**

**HauptbetreuerIn:** Bernhard Tellenbach, tebe
**NebenbetreuerIn:** Gürkan Gür, gueu

**Zugeteilte Studenten:**

Diese Arbeit ist vereinbart mit:
- Levi Cailleret, cailllev (IT)
- Sascha Kyburz, kybursas (IT)

**Fachgebiet:**

IS Information Security

**Studiengänge:**

IT Informatik

**Zuordnung der Arbeit :**

InIT Institut für angewandte Informationstechnologie

**Infrastruktur:**

benötigt keinen zugeteilten Arbeitsplatz an der ZHAW

**Interne Partner :**

Es wurde kein interner Partner definiert!

**Industriepartner:**

Es wurden keine Industriepartner definiert!

**Beschreibung:**

The asymmetry between attackers and cybersecurity is a big problem since the former has a long time to assess the targets security standing (e.g., vulnerabilities, configuration, potential attack vectors) and perform reconnaissance. The idea of moving-target defense (MTD) is to impose the same asymmetric disadvantage on attackers by making systems dynamic and therefore harder to explore and predict with a constantly changing system and its ever-adapting attack surface. This is fundamentally a control problem which can be addressed via different algorithms including Machine Learning (ML) such as Reinforcement Learning (RL). **The main objective of this BS thesis is to investigate RL as a decision tool for MTD based evasion techniques**.

**Goals**

The goals of this thesis are as follows:

- You learn about Reinforcement Learning and how to use related techniques in a network security related case study.
- We have a better understanding of technical challenges regarding MTD and how to control it via RL techniques
- We have a simulated decision module (research grade) that can generate efficient and effective security countermeasures in terms of network configuration after an attack is detected (detection itself is out of scope.)
- We have performance evaluation results from our decision module based on some experimental scenarios

**Tasks**

To reach those goals, you have to complete the following tasks:

- Identify and examine related literature and solutions to understand the State-of-the-Art (SotA) in this field
- Learn about RL and MTD so that you have the knowledgebase to use them to build the thesis deliverables
- Design, implement and evaluate a decision module that provides security countermeasures against network attacks in a simulated environment

zurück Logout

### 7.1.2 Zeitplan

Folgend ist der Zeitplan abgebildet. Dieser wurde in der ersten Projektwoche erstellt und fast alle Termine sind eingehalten worden. Die Methodik benötigte ein wenig mehr Zeit, sodass die Resultate und Diskussion parallel in der dritt letzten Woche erledigt wurden.

| Week | Date | Day | What | Who |
|---|---|---|---|---|
| 38 | 15.09.2020 | Tu | Preparation Kickoff Meeting | PT |
| | 16.09.2020 | We | Kickoff Meeting | PT S |
| 39 | - | - | Further information gathering on MTD, RL, Networksec | PT |
| | 23.09.2020 | We | Weekly Meeting | PT S |
| 40 | - | - | Further information gathering on MTD, RL, Networksec | PT |
| | - | - | Start writing PA, table of contents and structure | PT |
| | 30.09.2020 | We | Weekly Meeting | PT S |
| 41 | - | - | Further information gathering on MTD, RL, Networksec | PT |
| | - | - | Paper: Sota and goal | PT |
| | 07.10.2020 | We | Weekly Meeting | PT S |
| 42 | - | - | Create general Use-Cases | PT |
| | - | - | Paper: Theoretical foundations | PT |
| | 14.10.2020 | We | Weekly Meeting | PT S |
| 43 | - | - | Search and examine tools / frameworks | PT |
| | - | - | Paper: Reviewing & Correcting | PT |
| | 21.10.2020 | We | Weekly Meeting | PT S |
| 44 | - | - | Search and examine tools / frameworks | PT |
| | - | - | Paper: Reviewing & Correcting | PT |
| | 28.10.2020 | We | Weekly Meeting | PT S |
| 45 | - | - | implement experiment | PT |
| | - | - | Paper: Methods | PT |
| | 04.11.2020 | We | Weekly Meeting | PT S |
| 46 | - | - | implement experiment | PT |
| | - | - | Paper: Methods | PT |
| | 11.11.2020 | We | Weekly Meeting | PT S |
| 47 | - | - | implement experiment | PT |
| | - | - | Paper: Results | PT |
| | 18.11.2020 | We | Weekly Meeting | PT S |
| 48 | - | - | experient with tools / frameworks and evaluate outcome / fitnes of methods | PT |
| | - | - | Paper: Discussion | PT |
| | 25.11.2020 | We | Weekly Meeting | PT S |

| | | | | |
|---|---|---|---|---|
| 49 | - | - | documentation of tools & frameworks | PT |
| | - | - | Hand in Paper for Feedback | PT S |
| | - | - | Paper: Reviewing & Correcting | PT |
| | 02.12.2020 | We | Weekly Meeting | PT S |
| 50 | - | - | documentation of tools & frameworks | PT |
| | - | - | Paper: Reviewing & Correcting | PT |
| | 09.12.2020 | We | Weekly Meeting | PT S |
| 51 | - | - | Reserve | |
| | 19.12.2020 | Sa | Abgabe PA | PT |

**Legende**

PT: Project Team
PT S: Project Team with Supervisors

| |
|---|
| Meetings |
| Paper |
| Research |
| Coding |

### 7.1.3 Meeting Notizen

Für das Kickoff Meeting und die erste Besprechung sind ausführliche Notizen erstellt worden. Für die restlichen Besprechungen sind die Notizen in die Protokolle oder direkt in die Arbeit eingeflossen. Die Notizen sind auf den folgenden Seiten ersichtlich.

**KW 24 – Kickoff Meeting**

PA-Thesis Format

- main part in German
- small "summary" of Thesis in English
- Code-Documentation in English
- UI in English

Research

- search "survey MTD"
- use well known journals
- > 60 citations on survey (well known)
- use help from tebe, gueu

Product

- RL for MTD (Network- / Cybersecurity)
- pretty open defined Use-Cases (Network Sec)
- 3 Aspects
    - MTD
    - RL
    - Network Security

Tasks

- kickoff meeting (middle or end of first week)
- what do we do, when do we do it
- small understanding of whole subject

- read up on MTD, RL, Network Sec
- create "general" Use-Cases
- research, read surveys
- play around with tools / frameworks
- ---------- halftime ----------
- decide what surveys are useful
- sort out Use-Cases, choose useful tools (surveys / open-source tools)
- define what our product is really about (what methods for what)
- set up tools, pick specific Use-Cases for further / final work

Target

- at the end of the PA-Thesis, it must be at a state where we or someone else can pick up the work
- at least tested our recommendations (software, learning, ...)

Organization

- weekly short update meeting (show our work)
- pull principle (ask advisors for assistance, not vice versa)

1. Phase

PA – Thesis

2. Phase?

BA – Thesis

**KW 38 – 1. Besprechung**

IT Sec Group -> EU 5G Project -> develop Sec Solution (Networks like 5G)

Gür: Group Leader

MTD for some 5G Network functions -> PA Target

MTD: no disadvantage for defender with attack surface configuration, no scanning with low-cost effort

more difficult to understand the system

2: change ports for network services

3: when to change it

-> Decision Problem

Rule based approaches, simple or complex rules to make decisions

ML for decisions -> Reinforcement Learning (trial and error)

make a model for the problem

most likely python library for RL

PA Goals:

- learn MTD, RL
- implement MTD in simulated environment with research grade
- test it, investigate overhead -> performance

next meeting:

- small presentation, what we found, what would be suitable, what we further look into
- timetable
- repository
- word or latex for PA Report

meeting fixed on Wednesdays, 8:30

surveys & papers for research in MTD from Mr. Gür

### 7.1.4 Protokolle

An jeder Besprechung mit den Betreuern wird ein Protokoll erstellt. Darauf sind die Pendenzen bis zur nächsten Besprechung festgehalten. Im ersten Absatz sind die Pendenzen aufgelistet, die alle betreffen, im folgenden Absatz die Pendenzen für das Projektteam und abschliessend die der Betreuer.

**KW 38 – 1. Besprechung**

All

- meetings will now have a standard timeslot on Wednesday 8.30 - 9.30.

Students

- get a basic overview regarding different MTD approaches and reinforcements learning and "present" them next week.
- create a repository for information exchange between students and supervisors.
- create a time plan that can be discussed next week.
- decide what tool to use regarding the paperwork of the PA (Office, Latex).
- link papers and texts in the repository.
- create meeting protocols and decision protocols.

Supervisors

- look at the repository, papers, texts and provide own material as you see fit.

**KW 39 – 2. Besprechung**

All

- all tasks in the protocols will now be meant to be completed until the next week/the next protocol except a date is stated.
- we decided to use office as tool for the main paperwork of the PA.

Students

- get familiar with Zotero or Mendeley.
- create Teams channel for our paper.
- get a better overview over reinforcement learning.
- work according to plan.

Supervisors

- provide material as you see fit.

**KW 40 – 3. Besprechung**

All

- the report will be in German.

Students

- create a sources-library for easy overview over our sources.
- work according to plan.

Supervisors

- provide material as you see fit.

**KW 41 – 4. Besprechung**

All

- we will use an attack-model to abstract the attack on a network. in this "framework" we will test machine-learning based MTDs.

Students

- get more feedback on the model.
- work according to plan.

Supervisors

- provide material as you see fit.
- provide alternate modelling-idea as you see fit.

**KW 42 – 5. Besprechung**

All

Students

- get more feedback on the model.
- work according to plan.

Supervisors

- provide material as you see fit.
- provide alternate modelling-idea as you see fit.

**KW 43 – 6. Besprechung**

All

Students

- get more feedback on the model.
- work according to plan.

Supervisors

- provide material as you see fit.
- provide alternate modelling-idea as you see fit.

**KW 44 – 7. Besprechung**

All

Students

- get more feedback on the model (talk to outside-experts).
- work according to plan.

Supervisors

- provide material as you see fit.
- provide alternate modelling-idea as you see fit.

**KW 45 – 8. Besprechung**

Besprechung auf Wunsch von Studenten ausgefallen.

**KW 47 – 9. Besprechung**

All

Students

- work according to plan.

Supervisors

- provide material as you see fit.
- provide alternate modelling-idea as you see fit.

**KW 48 – 10. Besprechung**

All

Students

- finalize parts of the report. More tables, more pictures, more differentiated sources, more sub-chapters.
- Begin implementing an abbreviation directory and a documentation/manual for your code.

Supervisors

- provide material as you see fit.
- provide alternate modelling-idea as you see fit.

**KW 49 – 11. Besprechung**

All

Students

- Work according to plan. Results, discussion, abbreviation directory, abstract etc.

Supervisors

- provide material as you see fit.

- provide alternate modelling-idea as you see fit.

- offered grammar correcting if text is sent until 6. of December.

**KW 50 – 12. Besprechung**

All

Students

- work according to plan.
- results, discussion, abbreviation directory, abstract etc.
- finish grammar-checks and most of the text until Friday.

Supervisors

- -provide material as you see fit.
- -provide alternate modelling-idea as you see fit.

**KW 51 – 13. Besprechung**

Es wurde kein Protokoll erstellt.

## 7.2 Weiteres

### 7.2.1 Code Übersicht

Sämtlicher Code inklusive Anleitung und Dokumentation ist auf einem GitHub Repository unter folgendem Link einzusehen:

https://github.zhaw.ch/caillllev/PA_MTD/tree/master/RL_MTD

Auf den folgenden Seiten eine kurze Zusammenfassung und Erklärung des gesamten Codes zu sehen.

**Ordnerstruktur**

Die Ordnerstruktur aller benötigter Dateien ist folgend abgebildet. Ausgeschlossen davon sind die Inhalte des Ordner venv. Darin befinden sich die benötigten externen Python Pakete.

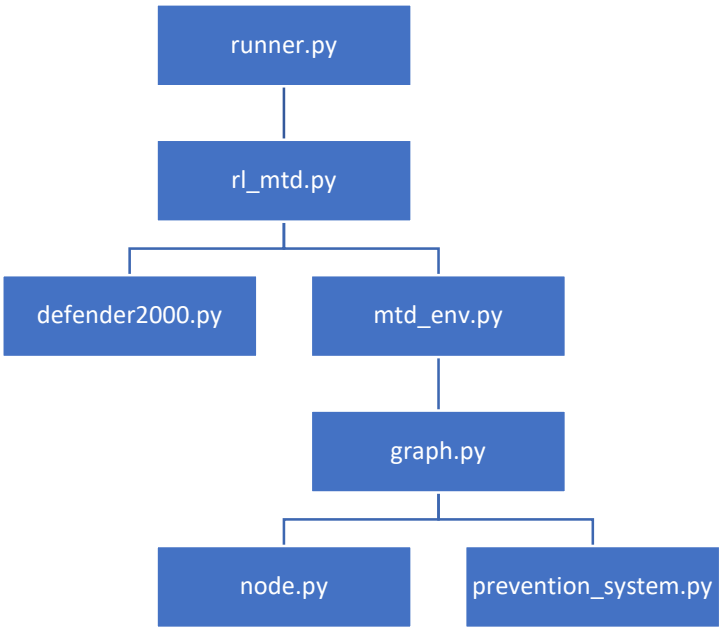
/ **RL_MTD**
- / **config**
  - / attack_graphs.json
- / **src**
  - / env
    - / mtd_env.py
  - / model
    - / detection_system.py
    - / graph.py
    - / node.py
  - / parameters
    - / *diverse Parameter für RL Agenten für jegliche Simulationen*
  - / defender2000.py
  - / rl_mtd.py
  - / runner.py

/ **test**
- / env
  - / test_mtd_env.py
- / model
  - / prevention_system.py
  - / graph.py

/ **venv**

**Klassendiagramm**

Das folgende Klassendiagramm ist nicht komplett vollständig, weil Python abgesehen von Klassen auch Methoden und Variablen von anderen Files importieren kann. Alles darzustellen würde sehr unübersichtlich werden. Die Abbildung ist somit nur eine grobe Zusammenfassung der Abhängigkeiten.




### 7.2.2 Code Base anpassen an eigenes Netzwerk

Sämtliche Anpassungen sind in der Datei attack_graphs.json vorzunehmen. Dort ist das Netzwerk an sich definiert, die einzelnen Services und die Wahrscheinlichkeiten der Angriffe. In dieser Datei können neue Netzwerke oder Angriffe definiert werden oder bestehende angepasst werden. Zusätzlich sind die Klassen lose gekoppelt voneinander, sodass bei einer Anpassung der internen Logik noch alles funktionieren sollte.

Um neue Netzwerke und Angriffe zu simulieren sind in den Dateien rl_mtd.py Linie 451 und Linie 452 oder in runner.py Linie 18 und Linie 19 die Namen des neuen Netzwerks und des neuen Angriffs einzugeben. Alternativ können wie erwähnt auch das bestehende Netzwerk und die bestehenden Angriffe überschrieben werden und die Namen müssen nicht angepasst werden.

Nach dem Anpassen kann verifiziert werden, ob das Modell noch funktioniert, d.h. ob die Anpassungen ungefähr Sinn machen. Dafür ist in graph.py selbst Logik eingebaut, damit zumindest immer die Services und Wahrscheinlichkeiten initialisiert werden können. Kann etwas nicht korrekt initialisiert werden, wirft das Programm einen Fehler mit ausführlicher Erklärung, wieso es nicht funktioniert. Zusätzlich kann in test_graph.py verifiziert werden, dass ein Pfad existiert vom Startservice zum Ziel, also der Angreifer auch wirklich gewinnen könnte.

### 7.2.3 Code Disclaimer

Mit den hier gegebenen Informationen sollte ein Programmierer mit ein wenig Python Erfahrung sich gut zurechtfinden im Projekt. Sämtliche Klassen und Methoden sind dokumentiert. Bei den Klassen defender2000, mtd_env, graph, node und prevention_system sind alle Parameter und Rückgabewerte beschrieben und der Typ der Variable ist gegeben (Type Hint).

Sämtliche Funktionalität ist mit UnitTests sichergestellt. Pro Klasse sind zwischen 5 und 10 unterschiedliche Test Cases erstellt, die alle relevanten Funktionen testen. Ausnahme davon sind rl_mtd und runner, diese sind hauptsächlich per debug getestet. Ein UnitTest würde auch keinen Sinn machen, da viel zu viel Zufälle mitspielen.

Zu beachten ist, dass die meisten Test Cases von der aktuellen Projektkonfiguration abhängig sind. Bei einer Anpassung eines Netzwerkes oder der Angriffe kann die Erfüllung der Test Cases nicht garantiert werden. Wenn die Funktionalität für ein neues Netzwerk verifiziert werden soll, müssen die einzelnen Methoden und Tests verstanden werden und dementsprechend angepasst werden.

### 7.2.4 Code ausführen

Das README.md im GitHub Repository beschreibt dazu alles. Darin sind die Voraussetzungen bezüglich Python Version und den pip installs und eine Beschreibung, was man laufen lassen kann und wie man das tun soll. Nachfolgend wird gezeigt, wie der Code ausgeführt werden kann und wie der default output von rl_mtd.py ausschaut.

**README – running given code**

download this repo (https://github.zhaw.ch/cailllev/PA_MTD/) and go to folder RL_MTD

- run just once (default config includes ~5min of training and then simulates all algos
  - src.rl_mtd => run can be configured at main call at line 447
- run multiple times with different lengths of training sessions (1sec ... 4h):
  - src.runner => runs can be configured at main call at line 15

**default output of src.rl_mtd.main**

```
Config:
Graph Name:                simple_webservice
Attack Name:               professional
Only Nodes:                False
Only Prevention Systems:   False
Nodes Pause:               1
Prevention Systems Pause:  1
*****************************************
Learning A2C, PPO algorithms.
*****************************************
Learning A2C.
No learned parameters found, start from scratch.
100000 steps to learn, with updates to policy each 50 steps.
Start:              Monday 13:31:11.
Estimated finish:   Monday 13:32:41.
...
Actual finish:      Monday 13:32:28.
```

```
****************************************
Learning PPO.
No learned parameters found, start from scratch.
100000 steps to learn, with updates to policy each 50 steps.
Start:              Monday 13:32:28.
Estimated finish:   Monday 13:35:28.
...
Actual finish:      Monday 13:34:56.
****************************************
Prepared to simulate A2C, PPO, Defender2000, Random, Static with 5
simulations.
Start:              Monday 13:34:56.
Estimated finish:   Monday 13:35:24.
...
Start simulating A2C.
5 simulations done.
Start simulating PPO.
5 simulations done.
Start simulating Defender2000.
5 simulations done.
Start simulating Random.
5 simulations done.
Start simulating Static.
5 simulations done.
Actual finish:      Monday 13:35:22.

Printing file content:

**********************************************************************
Starting Simulation Types: A2C, PPO, Defender2000, Random, Static
Simulations per Type:      5
Steps per Simulation:      5000
----------------------------------------------------------------------
Graph Name:                simple_webservice
Attack Name:               professional
Only Nodes:                False
Only Prevention Systems:   False
Nodes Pause:               1
Prevention Systems Pause:  1
**********************************************************************
Simulation Type:          A2C
Learning Time:            77.4 sec
----------------------------------------------------------------------
Avg steps:                5000.0
Avg reward/step:          52.602
Avg null action ratio:    [0.009, 0.0]
Avg invalid actions:      [0.0, 0.0]

Min steps:                5000
Max steps:                5000

Defender wins:            5
Attacker wins:            0

Steps each sim:           [5000, 5000, 5000, 5000, 5000]
Reward each sim:          [262704, 263129, 262702, 263206, 263299]
**********************************************************************
Simulation Type:          PPO
Learning Time:            147.5 sec
----------------------------------------------------------------------
Avg steps:                5000.0
```

```
Avg reward/step:          53.308
Avg null action ratio:    [0.175, 0.004]
Avg invalid actions:      [0.0, 0.0]

Min steps:                5000
Max steps:                5000

Defender wins:            5
Attacker wins:            0

Steps each sim:           [5000, 5000, 5000, 5000, 5000]
Reward each sim:          [267011, 266219, 266464, 266399, 266605]
**********************************************************************
Simulation Type:          Defender2000
Learning Time:            0 sec
----------------------------------------------------------------------
Avg steps:                150.2
Avg reward/step:          34.441
Avg null action ratio:    [0.083, 0.268]
Avg invalid actions:      [0.0, 0.0]

Min steps:                71
Max steps:                316

Defender wins:            0
Attacker wins:            5

Steps each sim:           [82, 71, 316, 126, 156]
Reward each sim:          [2415, 2035, 11524, 4549, 5342]
**********************************************************************
Simulation Type:          Random
Learning Time:            0 sec
----------------------------------------------------------------------
Avg steps:                80.4
Avg reward/step:          27.557
Avg null action ratio:    [0.129, 0.326]
Avg invalid actions:      [0.0, 0.0]

Min steps:                39
Max steps:                172

Defender wins:            0
Attacker wins:            5

Steps each sim:           [90, 59, 42, 39, 172]
Reward each sim:          [2585, 1292, 733, 1010, 5458]
**********************************************************************
Simulation Type:          Static
Learning Time:            0 sec
----------------------------------------------------------------------
Avg steps:                14.0
Avg reward/step:          -7.143
Avg null action ratio:    [1.0, 1.0]
Avg invalid actions:      [0.0, 0.0]

Min steps:                10
Max steps:                20

Defender wins:            0
Attacker wins:            5
```

```
Steps each sim:           [10, 14, 20, 13, 13]
Reward each sim:          [-300, -210, 250, -70, -170]
****************************************************************
A2C is the most effective algorithm with 5000.0 avg steps.
PPO is the most efficient algorithm with 53.308 avg reward/step.


Process finished with exit code 0
```