Hauptstudie

Anomalie Detection in Netzwerken

Ives Schneider

Index

1. Management Summary

Anhand der, in der Vorstudie evaluierten Softwarelösungen Splunk und PRTG, wird eine Softwarearchitektur definiert, welche im weiteren vorgehen zu entwickeln ist.
Die Applikation nimmt dabei Anspruch auf die bereitgestellten Funktionalitäten der beiden genannten Lösungen.

In der Hauptstudie soll definiert werden, mit welcher Architektur und mit welchen Fähigkeiten eine Applikation ausgerüstet werden muss, um gewissen Anomalien erkennen zu können. Zusätzlich wurde mithilfe des MITRE ATT&CK Frameworks ermittelt, wie und welche Angriffe normalerweise durchgeführt werden.

Features

Anhand der Architektur bestehen folgende kontinuierlichen Möglichkeiten, Anomalien zu erkennen:

- Erkennung neuer Hosts
- Veränderungen von Netzwerkservices
- Fehlgeschlagene Loginversuche
- Gezielte Überwachung von gewünschten Zielen

Kosten

Da es sich um ein komplett Open-Source Projekt handelt (minus Splunk+PRTG), entstehen ausser der aufgewendeten Arbeitszeit keine zusätzliche Kosten.

Projekt

Im laufe der Hauptstudie wurde entschieden, die Testumgebung (POC-Lan) auf eine virtuelle Umgebung zu verschieben.
Dies ersparrt zusätzlichen aufwand, die Hardwareumgebung nachträglich mit Splunk auszurüsten.

Innerhalb diese Dokuments wird die Applikation fortlaufend mit ihrem Projektnamen (**Nidhogg**) bezeichnet.

2. Änderungen aus der Vorstudie

2.1. POC-LAN

Aufgrund Hardwarefailures muss leider auf die bestehende Infrastruktur versichtet werden.
Daher wurde eine virtuelle Infrastruktur mithilfe Ansible und Terraform erstellt.

Falls das POC-Netzwerk nachgebaut werden will, kann das Ansible Playbook und das Terraform Script hier gefunden werden.

3. Anomalie

3.1. Definition

> Anomalien sind unerwartete Abweichungen von Regeln, im Kontext der Produktion also Abweichungen von "normalen Betriebszuständen". Diese treten meist in einem Fehlerfall auf. Sie können allerdings auch ein Hinweis auf einen Angriff bzw. eine Manipulation innerhalb eines Produktionsnetzwerkes sein. Das gilt insbesondere dann, wenn Ereignisse erstmalig auftreten, Prozesse sich anders verhalten oder Geräte miteinander kommunizieren, die es bisher nicht getan haben.
>
> — BSI, Monitoring und Anomalieerkennung in Produktionsnetzwerken

3.2. Erkennung

Die Erkennung soll anhand eines Algorithmus erfolgen. Dabei soll der Algorithmus mehrere Merkmale analysieren. Grundsätzlich ist die Frage zu klären, wie eine standardmässige Kill Chain aussieht und mit welchen Massnahmen welche Schritte erkennen könnte.

3.3. Kill-Chain

Eine Kill-Chain beschreibt wie ein Angreiffer normalerweise Zugang und Foothold in einem Netzwerk bekommt. Um einen klaren Überblick für die Möglichkeiten zu erhalten, wie eine Anomalie erkannt werden kann, wird auf das Mitre ATT&CK Framework zurückgegriffen, welche die einzelnen Phasen beschreibt.

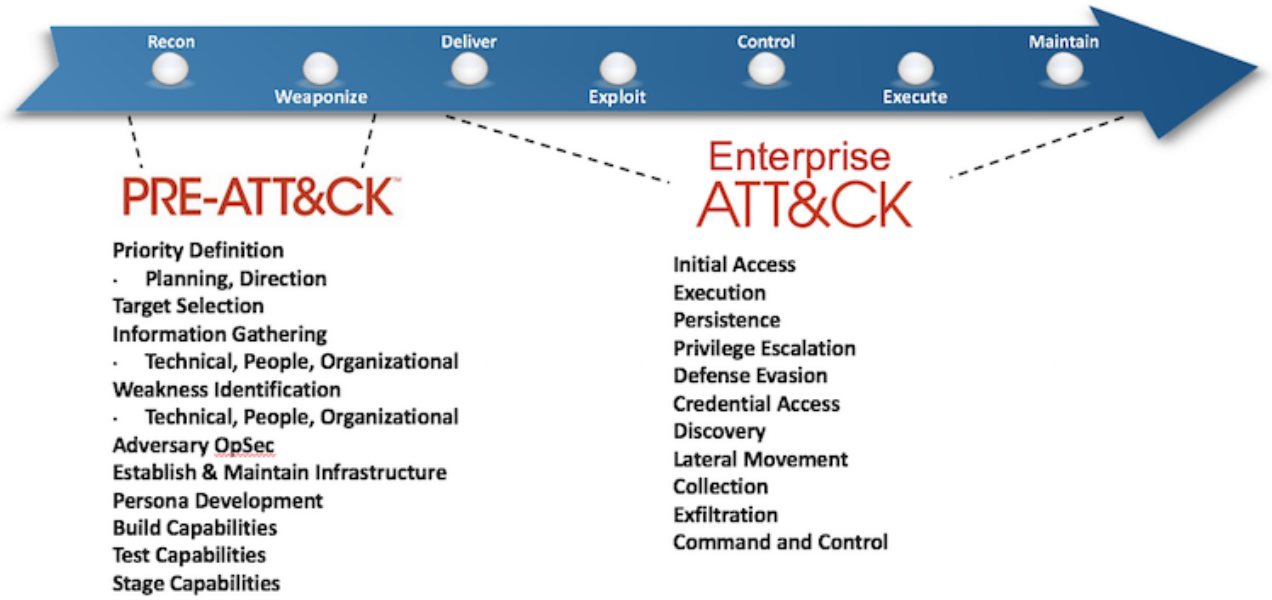

Figure 1. https://attack.mitre.org/ - Killchain

3.3.1. Reconnaissance

Beschreibung

In dieser Phase werden weitere Informationen über das Ziel beschaft.
Während dieser Phase, befindet sich der Angreiffer meist noch ausserhalb des zu angreifenden Netzwerkes. Allerdings ist Reconnaissance eine wiederholende Phase, welche während des gesamten Angriffs vortbeständig durchgeführt wird.

Erkennung

Solange sich der Angreiffer ausserhalb des Netzwerkes befindet, gibt es nur eine limitierte Anzahl an Erkennungsmassnahmen.
Da der Scope von Nidhogg eher auf eine interne Erkennung liegt, wird dies auch nicht weiter verfolgt.

Intern hingegen, wird anhand eines passiven/aktiven ARP-Request Scannings das Netzwerk nach neuen Hosts durchsucht. Sollte ein neuer Host entdeckt werden, wird der Administrator via Mail auf die neue MAC Adresse aufmerksam gemacht.

3.3.2. Weaponization

Beschreibung

Der Angreiffer beschafft sich einen Exploit, welcher für die gefunden Schwachstelle in Phase I zugeschnitten ist. Die Art des Exploits spielt dabei keine Rolle.

Erkennung

Es gibt mehrere Möglichkeiten einen Exploit zu erkennen. Ein relativ neuer Ansatz ist mit sogenannten YARA Regeln.
Hierbei wird anhand bestimmter Regeln der Datenfluss analysiert und bei einer gewissen String-Abfolge entschieden, ob sich die Datei innerhalb der Known Malware Liste befindet.

Für Nidhogg wurde bewusst entschieden, auf eine zu intrusiven Netzwerkerkennung zu verzichten. Daher kann diese Phase nicht erkannt werden.

3.3.3. Delivery

Beschreibung

Der Exploit aus Phase II wird zum Ziel übermittelt. (Beispielsweise via E-Mail, Website etc.) Diese Phase ist einer der Keypunkte, einen Angriff erfolgreich zu verhindern.
Da die Phasen I und II sich ausserhalb des eigenen Netzwerkes befindet, besteht kein Kontakt mit dem Angreiffer. Ab Phase III wird die Kommunikation mit mindestens einem Teil der Organisation aufgenommen.

Erkennung

Da die Kommunikation bewusst nicht überwacht wird, besteht hier keine Erkennungschance.

INPORTANT: Die Kommunikation und Host Überwachung wird auf Splunk + PRTG ausgelagert.

3.3.4. Exploitation

Beschreibung

Malware wird ausgeführt und folgt der eingebauten Logik ab.

Erkennung

Angriffe welche gezielt durchgeführt werden, setzen meist auf einen der folgende vorgehensweisen:

1. E-Mail Attachement
2. Dropper
3. Download additional Malware
4. Foothold

1. Exploit
2. Reverse shell
3. Foothold

1. Exploit
2. Binding shell
3. Foothold

Nidhogg spezialisiert sich eher auf Erkennung und nicht auf Verhinderung.
Allerdings wird mithilfe des Portscanners würde eine Binding shell erkannt werden.

CAUTION	Dropper + Reverse Shell können leider nicht erkannt werden.

3.3.5. Installation

Beschreibung

Zusätzlicher Backdoor wird auf dem Zielsystem installiert (foothold).
Dies ermöglicht dem Angreiffer neue Verbindungen und weitere Kommandos der Malware zu senden.

Erkennung

Es gibt viele möglichkeiten wie man ein Backdoor einrichten kann. Nidhogg soll die Möglichkeit haben, mindestens eine davon zu erkennen.

Table 1. Erkennung

Technik	Wird erkannt
Reverse Shell	-
Binding Shell	x
DNS backdoor	-
CnC Server	-

3.3.6. Command and Control

Beschreibung

CnC Server sendet Malware neue Instruktionen und ermöglicht dem Angreiffer, Informationen aus dem Netzwerk zu ziehen.

Erkennung

Siehe Phase "Installation"

3.3.7. Actions on Objective

Beschreibung

Schritte zur Erfüllung des Ziels des Angreiffers werden durchgeführt.
Dies kann von Vernichtung von Daten beinahlten (selten) bis hin zu Datendiebstahl.

Erkennung

Grundsätzlich kann anhand der PRTG Auswertungen erkannt werden, ob zusätzliche Daten zu ungewöhnlichen Zeiten versendet werden.

3.4. Einstufung

Je nach Art des Alerts, soll eine gewisse Abfolge innerhalb von Nidhogg durchgeführt werden. Da es um eine Anomalie-Erkennung gehen wird, muss schlussendlich ein Administrator selbst entscheiden, wie schwerwiegend die gemeldete Informationen der Unternehmung schaden können.

3.4.1. Event

Events sind normale Meldungen welche nicht auf schwerwiegende Anomalien hindeuten.
z.Bsp. Hoher Netzwerkspike ohne zusätzliche Anomalien. Beispiel:

NOTE	High Bandwith: {IP}

3.4.2. Alert

Meldungen welche auf Downtime oder neue Geräte hinweisen. Allerdings ohne zusätzliche Informationen
Beispiel:

CAUTION	New device found: {IP} {MAC}

3.4.3. Incident

Anomalien welche auf lateral movement hinweisen könnten. Beispiel:

WARNING	New Port: {PORT} on {IP}

3.5. Algorithmen

Für die Anomalie Erkennung wird auf mehrere Algorithmen zurückgegriffen welche im Hintergrund laufen sollen.

3.5.1. New Host

Neue Hosts können auf einen Angreiffer hindeuten, welcher einen freien Netzwerkport gefunden hat.
Leider ist diese Variante für grössere Netzwerke, nicht einfach maintable, da die Anzahl Hosts massiv höher ist.

Nidhogg soll in der lage sein, Administratoren über neue Hosts innerhalb eines definierten Netzwerkabschnitts zu informieren.

Somit kann die visibility massiv erhöht werden, welcher Administratoren einfacheren durchblick über die Netzwerkinfrastruktur erbringt.
Da im POC Netzwerk die Hostanzahl begrenzt ist, wird auf eine PDO Verbindung verzichtet.

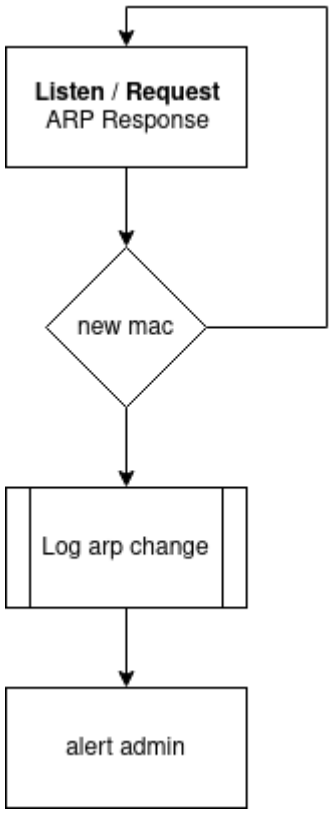

Figure 2. ARP-Change

IMPORTANT	Die Geschwindigkeit der Erkennung ist netzwerkgrössen abhängig.

3.5.2. Portzustand

Durch die Konfiguration sollen offene Ports anhand einer Whitelist definiert werden.
Falls sich der Status des Hosts ändern sollte, wird der Workflow ausgeführt.

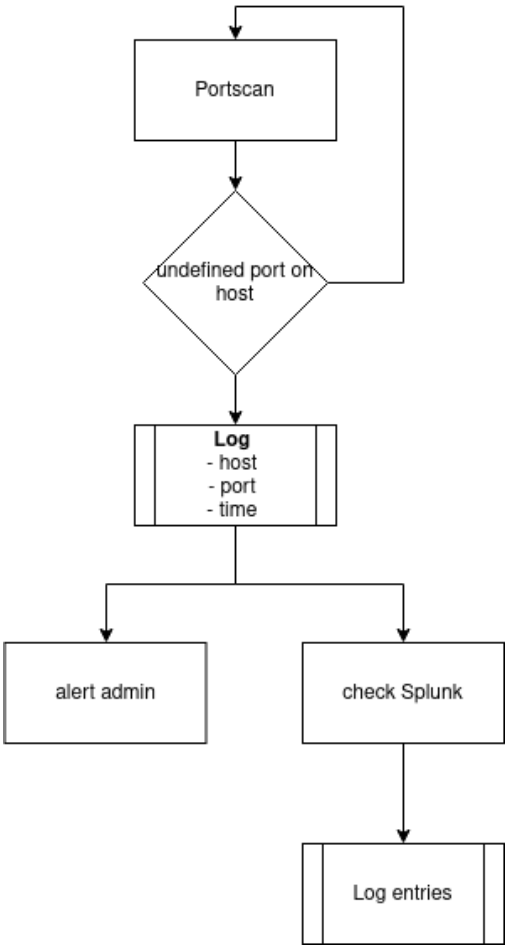

Figure 3. Änderung eines Portzustandes

3.5.3. Splunk failures

Da Splunk in der Vorstudie definiert wurde, soll Splunk via seinem HTTP-API angefragt werden können, ob bestimmte Änderungen geloggt wurden.
Die Checks sollen auf Reaktion eines anderen Alarms getätigt werden und somit die Benachrichtigungen verfeinern.

3.5.4. PRTG Meldung

PRTG besitzt die möglichkeit, über sogenannte Notification Gruppen, Nachrichten an einen HTTP-Endpoint zu senden.
Nidhogg übernimmt hier eine passive Rolle und wartet auf calls an den REST-Endpoint.

Die ermöglicht es, nicht nur für PRTG verfügbar zu sein, sondern könnte auch via Icinga oder andere NMS angesprochen werden.

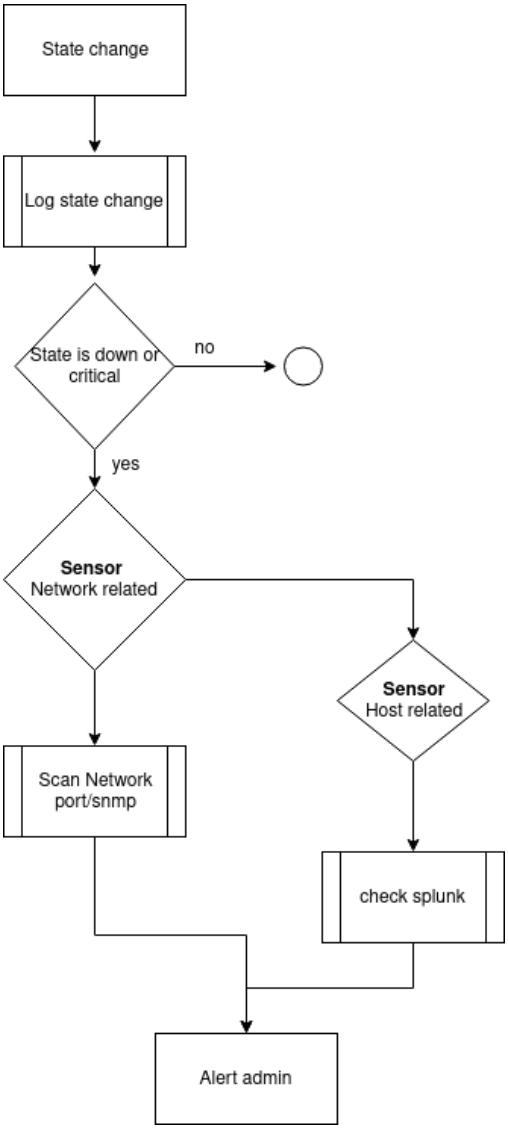

Figure 4. PRTG Meldungen

3.6. Netzwerk

Das POC-Netzwerk wird bewusst etwas kleiner gehalten.
Bestehend aus folgenden Services:

- PRTG
- Splunk
- NGINX (DVWA)
- Nidhogg

Wird eine kleine Infrastruktur simuliert, welche leicht anzugreiffen ist.

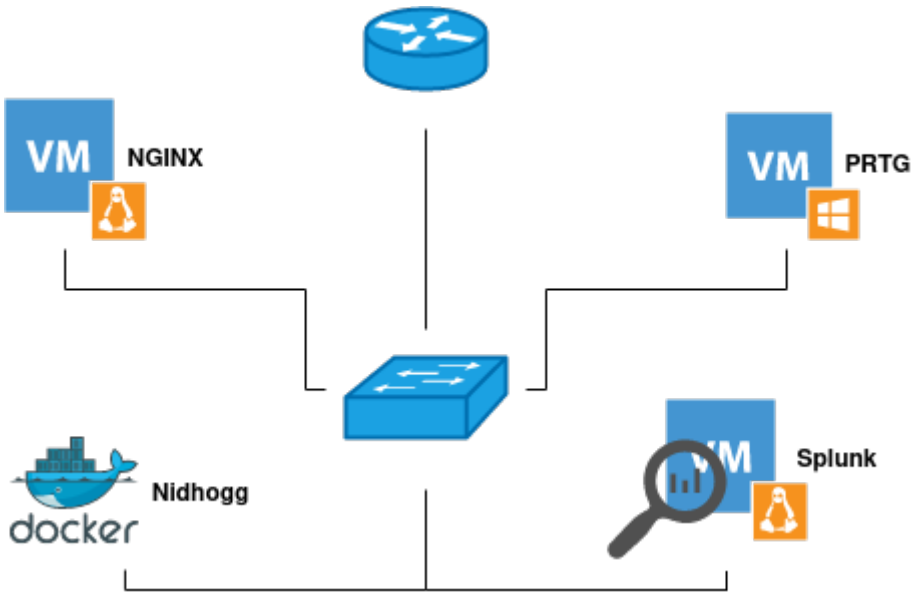

Figure 5. Netzwerkdiagram

3.7. Load

Load wird mithilfe Nping generiert.

4. Applikation

Nidhogg soll als Anlaufstelle für Anomalieerkennungen dienen.
Anhand diversen Merkmalen, soll erkannt werden, ob eine gemeldete Abweichung sich um eine Anomalie handelt welche genauer untersucht werden soll, oder aber um eine Abweichung, welche nicht weiterverfolgt werden muss.

Zugegriffen wird dabei auf folgende Möglichkeiten mit den Umsystemen zu kommunizieren.

Protokolle

- ICMP
- SNMP
- HTTP
- ARP

Es wird versucht den Code möglichst low-level zu halten um die Performance der Umsysteme möglichs wenig zu beeinträchtigen.

4.1. Technology

Nidhogg wird in Rust geschrieben. Dies ermöglicht es, sicheren Quellcode zu schreiben ohne dabei Geschwindigkeit zu verlieren.
Von grossem belangen wird hierbei der Borrowchecker, lifetimes sowie das Secure Memory Management um Nidhogg möglichst erweiterbar und ressourcen schonend zu schreiben.

4.2. Architektur

Nidhogg wird in einer einfachen 3-Tier Architektur entwickelt.
Die einzelnen Layers beziehen sich auf die Abschnitte des Programms.

Als Datenlayer wurde entschieden SQLite zu verwenden.
Natürlich kann argumentiert werden, dass lieber mysql/maria db etc. verwendet werde sollte.
Allerdings besteht bereits ein Programm von welchem ich die Funktionalität wiederverwenden kann.

First Tier

Webinterface

Second Tier

Programm Logik

Third Tier

SQLite

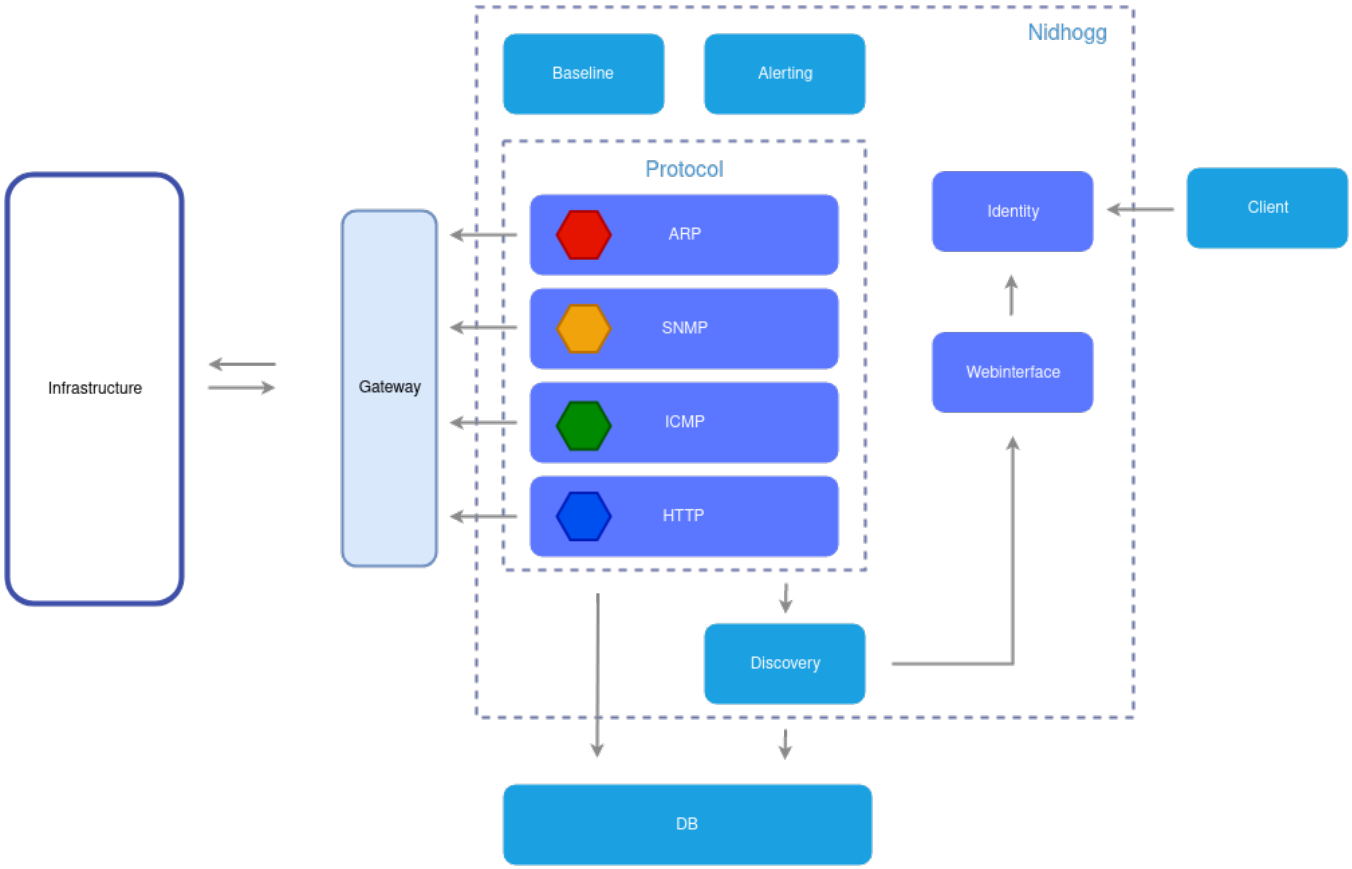

Figure 6. Nidhogg Architektur

5. Abhängigkeiten

Aufgrund des relativ jungen alters von Rust, werden viele Libraries, welche in anderen Sprachen verfügbar wären, nicht existieren.
Dies bedeutet, dass einige Module selbst geschrieben werden müssen oder bereits bestehende Module auf die Funktionalität zugeschnitten werden muss.

Allerdings kann bereits auf einige Abhängigkeiten eingegangen werden, welche benötigt werden:

Table 2. Abhängigkeiten

Abhängigkeit	Version	Grund	URL
Nmap	latest	Portscanning	nmap
libpcap	latest	Arpscanning	libpcap
serde	1.0.102	(de)serializiation	serde
actix_web	1.0.8	Webinterface/RESTful	actix

5.1. Installation

Um die Installation möglichst einfach zu halten, soll Nidhogg einfach über CLI installiert werden können.
Unter Linux wird dies mithilfe des .deb Formates zustandegebracht.
Windows Systeme werden eine portable Binary erhalten.

Die komplette Installationsdokumentation wird in eigenem Dokument mitgeliefert.

5.2. Konfiguration

Einstellungen sollen via eines YAML Files vorgenommen werden können.
Um möglichst flexibel zu bleiben, sollen default Einstellungen mit Nidhogg mitinstalliert.

Wichtig ist es, dass einzelne Funktionalitäten deaktiviert werden können.

Ein erster Draft ist bereits verfügbar:

config_draft.yml

```
webserver:
    address: "0.0.0.0"
    port: "8080"

mail:
    server: "smtp.gmail.com"
    username: "user@gmail.com"
    password: ""
    email: "user@i-401.xyz"
    from: "user@gmail.com"

splunk:
    server: "splunk:8089"
    username: ""
    password: ""
    interval: 500

snmp:
    server: "127.0.0.1"
    community: "my_comm"
    oid: "1.3.6.100.1.2.3.5.1.1.0"

portscan:
    portspec: "~/Repos/nidhogg/portspecs.yml"
    mappings: "~/Repos/nidhogg/mappings.xml"
    timeout: 500

arpscan:
    interface: "eno1"
    db: "/tmp/arp.db"
    timeout: 500
    mac:
      - "00:17:88:28:9f:ca"
      - "00:55:da:50:40:64"
      - "34:7e:5c:31:10:e8"
      - "c8:3c:85:3e:e8:dd"
      - "f4:4d:30:68:9b:d4"
```

Mappings_draft.xml

Da ausserhalb der Hauptkonfiguration, ebenfalls noch Einstellungen für die wiederholende Portscan-Funktion gemacht werden muss, kommen zwei zusätzliche Konfigurationsdateien hinzu.

```
{ "mappings":
    [
        {
            "hostname": "artoria",
            "id": "i-0",
            "ips": ["10.0.0.33"],
            "name": "artoria",
            "portspec": "artoria"
        },
        {
            "hostname": "splunk",
            "id": "i-1",
            "ips": ["10.0.0.36"],
            "name": "splunk",
            "portspec": "splunk"
        }
    ]
}
```

porspec_draft.yml

Die Portspec Datei wird dafür genutzt, Hosts aus dem Mappings.xml, mit den Einstellungen der Ports zu verbinden.

```
portspecs:
  - name: artoria
    ports:
      - id: 22
        state: open
  - name: splunk
    ports:
      - id: 22
        state: open
      - id: 8080
        state: open
```

Der komplette Konfigurationsumfang wird in eigenem Dokument mitgeliefert.

6. Controlling

Die Vollumfänglichkeit der gesetzten Ziele wird innehalb dieses Kapitels getestet und angeschaut.
Tests werden innerhalb des neu definierten POC-Netzwerks in einer kontrollierten Umgebung durchgeführt.

Test und Controlling wird in einem eigenem Dokument mitgeliefert.

7. Kosten

Die Kosten belaufen sich auf die aufgewendete Arbeistzeit.
Da die Entwicklung mit OpenSource Modulen in einer Sprache welche, unter MIT Lizenziert, entwickelt wurde, steht es frei die Applikation für nicht kommerzielle zwecke zu verwenden.

Da das POC-Netzwerk relativ klein gehalten wurde, entstehen auch keine Kosten in Sachen Log-Collector bzw. NMS.

8. Weiteres Vorgehen

Grundlegende Informationen sind soweit alle Vorhanden um mit der Erstellung von Nidhogg zu beginnen. Da es wesentlich mehr Faktoren benötigt, als anfangs angenommen, wird die Applikation in kleinere Pakete aufgeteilt und via Workspaces zusammengefasst.

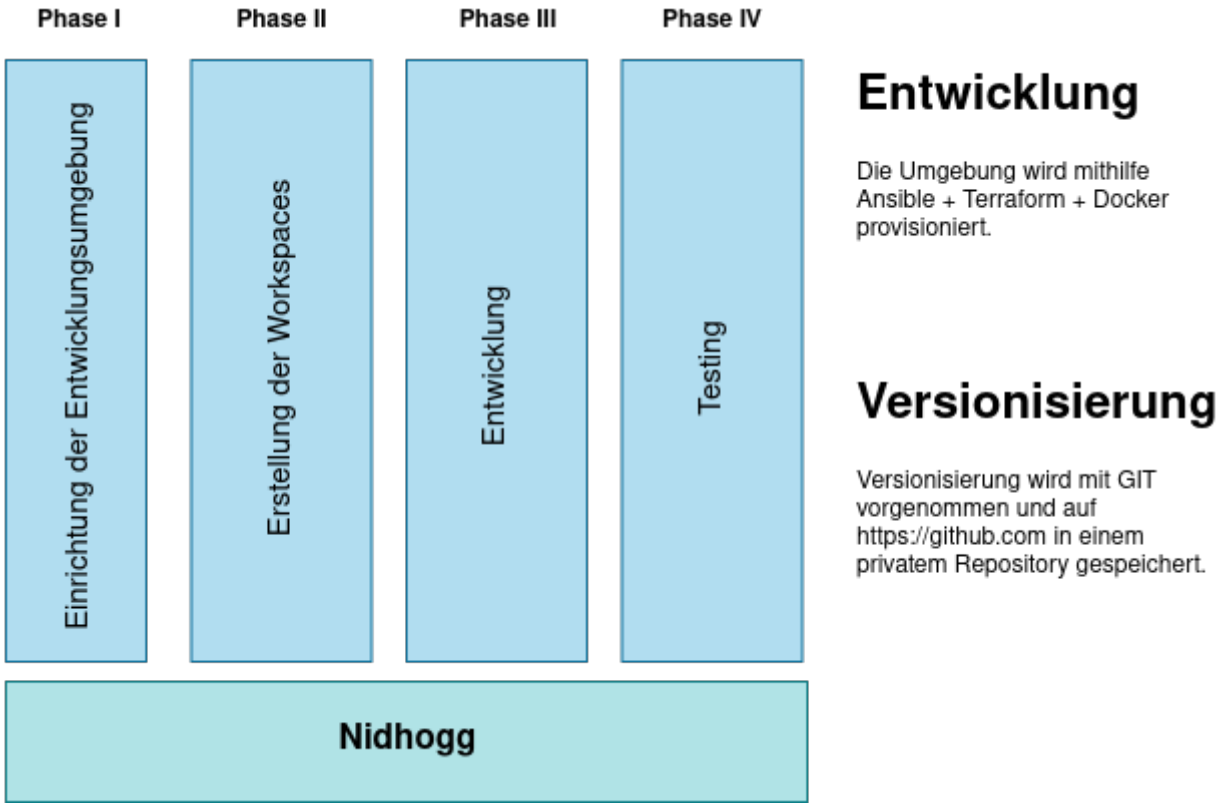

Figure 7. Weiteres Vorgehen

9. Freigabe

Auftraggeber

Technische Berufsschule Zürich
Sihlquai 101
8090 Zürich
admin.hf@tbz.zh.ch

Projektleitung

Ives Schneider
Binzstrasse 19
8712 Stäfa
ives.schneider@i-401.xyz

Experte

Marco Sieber
marco.sieber@tbz.ch

10. Darstellungsverzeichnis

11. Glossar

NMS

Network Monitoring System - Überwachungssystem

POC

Proof of concept - Konzeptbeweise

Kill-Chain

Phasen eines Angriffs

12. Anhang

12.1. Installationsdokumentation

TODO

12.2. Wartungsdokumentation

TODO

12.3. Testing & Controlling

TODO