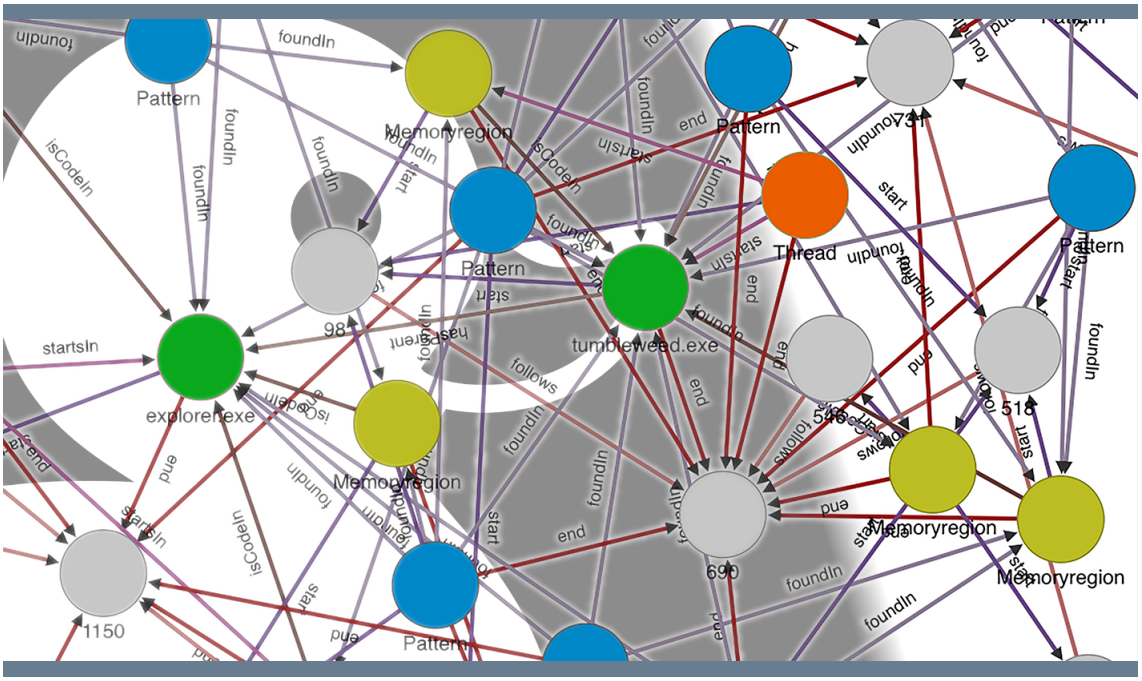


Clusteranalyse von Malware basierend auf Installationsgraphen

Malwareverhalten mit Graphen vergleichen

Bachelorthesis

Studiengang:	BSc Informatik
Autoren:	XXXXXX, XXXXXX
Betreuer:	XXXXXX, XXXXXX
Auftraggeber:	XXXXXX
Experte:	XXXXXX
Datum:	19.01.2017

Versionen

Version	Datum	Status	Bemerkungen
0.1	22.09.2016	Entwurf	Dokumentinitialisierung
0.2	01.12.2016	Entwurf	Arbeitsdokumentation
0.3	23.12.2016	Entwurf	Kapitel Methodik
0.4	30.12.2016	Entwurf	Kapitel Hintergrund
0.5	09.01.2017	Entwurf	Erstes Korrekturlesen
0.6	10.01.2017	Entwurf	Kapitel Schlussfolgerungen und Abstract
0.7	16.01.2017	Entwurf	Verbesserungen nach Feedback
1.0	19.01.2017	Abgabe	Abgabe zur IP-Prüfung

Abstract

Im Jahr 2015 wurden 431 Millionen neue Malware-Varianten entdeckt. Da die manuelle Analyse von Malware nicht skaliert, entwickelt die IT-Security-Branche automatisierte Technologien und Prozesse. Die Praxis zeigt, dass sich Malware mit gleichem Ziel ähnlich verhält. Gestützt auf dieser Beobachtung, stellen wir in unserer Arbeit einen Graph-basierten Ansatz vor, der Ähnlichkeiten auf Basis von Attributen berechnet. Dazu verwenden wir sogenannte Installationsgraphen. Installationsgraphen sind gewichtete, attributierte und gelabelte Graphen die ähnlich sind, wenn sie vergleichbare Substrukturen enthalten. Als letzten Schritt haben wir den neu entwickelten Ansatz anhand Übereinstimmungen unserer mit vorgegebenen Gruppierungen bewertet und festgestellt, dass unsere Methode ein Wiedererkennen von Verhaltensmustern ermöglicht. Damit müssen in Zukunft weniger Malware-Samples manuell untersucht werden, was eine Fokussierung des Analyseprozesses neuer Malware bedeutet.

Selbständigkeitserklärung

Wir bestätigen mit unseren Unterschriften, dass wir unsere vorliegende Bachelorthesis selbständig durchgeführt haben. Alle Informationsquellen (Fachliteratur, Besprechungen mit Fachleuten, usw.) und anderen Hilfsmittel, die wesentlich zu unserer Arbeit beigetragen haben, sind in unserem Arbeitsbericht im Anhang vollständig aufgeführt. Sämtliche Inhalte, die nicht von uns stammen, sind mit dem genauen Hinweis auf ihre Quelle gekennzeichnet.

Ort, Datum:	Bern, 19.01.2017	
Namen Vornamen:	XXXXXX	XXXXXX
Unterschriften:	.....	.....

Inhaltsverzeichnis

1. Einleitung

Im Jahr 2015 wurden 431 Millionen neue Malware-Varianten entdeckt [38]. Dies entspricht einem Zuwachs von 36% (317 Millionen) gegenüber 2014. Um die Entwicklung von Malware zu verfolgen, ist die Klassifikation nach deren Zielen besonders interessant. Kennt man die Ziele der Malwarehersteller, können passende Abwehrmechanismen eingesetzt werden. In vielen Fällen sind diese Ziele aber nicht bekannt und müssen erst durch manuelle Analyse eruiert werden. Da die manuelle Analyse aber nicht skaliert, entwickelt die IT-Security-Branche automatisierte Technologien und Prozesse.

Die Praxis zeigt, dass sich Malware mit gleichem Ziel ähnlich *verhält*. Wir definieren Verhalten als die Entitäten und deren Interaktion in einem System. Aus diesem Grund hat die *dynamische Analyse* stark an Wichtigkeit zugenommen [2] [3] [37]. Im Gegensatz zur statischen Analyse, in der Programme anhand des Sourcecodes und des Kompilates untersucht werden, führen dynamische Analysesysteme das Programm aus und beobachten die Interaktion mit dem System. Graphen sind eine mögliche Datenstruktur um solche Beobachtungen zu codieren. Um das charakteristische Laufzeitverhalten von Malware zu beschreiben, wurden bereits Graph-basierte Ansätze entwickelt [10] [17] [24] [25]. Die Idee ist dabei API-Aufrufe aufzuzeichnen, als Graph zu codieren und diese Graphen zu vergleichen. Mit *Illusion* wurde aber bereits ein Angriff auf genau diesen Ansatz vorgestellt [34]. Um die IT-Security-Branche im Wettrüsten mit Malware-Hersteller besser zu positionieren, braucht es Systeme die ein vollumfängliches Bild liefern und mit einer sich stetig verändernden Bedrohungslage umgehen können.

Nahezu alle Informationen die zu einer Rekonstruktion des Verhaltens benötigt werden, sind im physikalischen Speicher abgelegt. Aus diesem Grund hat des Security Engineering Lab (SEL) der Berner Fachhochschule[1] *KAN* entwickelt. KAN überwacht den physikalischen Speicher eines virtuellen Gastsystems und legt bei bestimmten Ereignissen eine Kopie an. Aus jeder Kopie extrahiert KAN danach Informationen über den Zustand des Systems. Eine zeitlich geordnete Sequenz dieser Zustandsinformationen nennen wir *Memory-Trace*. Basierend auf KAN entwickelt das SEL Methoden zur Clusterung von Malware. Aus diesen Anstrengungen ist das System *Deckard* entstanden [36]. Deckard findet Ähnlichkeiten zwischen Malware anhand aus Memory-Traces extrahiertem Code. Mit dem Ziel der Erforschung einer verhaltensbasierten Alternative zu Deckard hat das SEL diese Arbeit in Auftrag gegeben.

Von einem bekannten ausgehend, stellen wir in unserer Arbeit einen neuen Ansatz vor um Malware nach Verhalten zu klassifizieren. Aus Traces erstellen wir gewichtete, attributierte und gelabelte Graphen, sogenannte *Installationsgraphen*. Basierend auf diesen clustern wir danach 125 Traces und prüfen die Hypothese, ob Installationsgraphen geeignet sind um Malware nach Familie zu Clustern.

Wir machen damit folgenden Beitrag:

- Eine formale Beschreibung der Ähnlichkeiten zwischen Malware-Samples basierend auf Verhalten.
- Eine Referenzimplementation des entwickelten Ansatzes.
- Ein Nachweis der Praxistauglichkeit des entwickelten Ansatzes.
- Die Prüfung der Hypothese: Installationsgraphen sind geeignet Malware nach Familie zu clustern.
- Eine Bewertung unserer Cluster anhand der Code-Clusterung von Deckard.

Diese Arbeit ist wie folgt aufgebaut: Kapitel 2 erklärt das für diese Arbeit benötigte Hintergrundwissen. Kapitel 3 stellt unser Vorgehen zur Prüfung der Hypothese vor. Kapitel 4 geht auf implementationsspezifische Themen ein und betrachtet die Praxistauglichkeit unseres Ansatzes. Kapitel 5 stellt die Resultate der im Kapitel 3 vorgestellten Schritte vor. Kapitel 6 beendet die Arbeit, macht Schlussfolgerungen und gibt einen Ausblick auf mögliche zukünftige Arbeiten.

[1] `https://sel.bfh.ch`

2. Hintergrund

Einige in dieser Arbeit verwendeten Begriffe bedürfen einer kurzen Einführung. Dieses Kapitel hat zum Ziel den Leser an diese heranzuführen.

2.1. sql2fluentflow

Als Vorarbeit haben wir *sql2fluentflow* entwickelt. sql2fluentflow bündelt Datenbankabfragen mit *FluentFlow*-Filtern. FluentFlow ist eine auf JavaScript basierende Filtermaschine, die es erlaubt "gefolgt-von"-Relationen zu definieren [18]. sql2fluentflow hat zum Ziel die Datenanalyse zu erleichtern und zu vereinheitlichen. Dazu unterteilt sql2fluentflow den Analysevorgang in zwei Schritte:

1. Daten über SQL strukturieren und auslesen.
2. Daten mit FluentFlow in JavaScript nachbearbeiten.

Eine Kombination aus SQL und FluentFlow-Filter wird in einem *Workspace* gebündelt. Workspaces erlauben zudem eine Parametrisierung der beiden Arbeitsschritte.

2.2. Malware-Familien

An dieser Stelle präsentieren wir einen Überblick der verwendeten Malware-Familien. Für jede Familie ist mindestens ein Merkmal sowie die Verbreitung angegeben:

Fobber
: Neuartige und vielseitig einsetzbare Malware. Versucht sich durch exzessiven Gebrauch von Verschlüsselung zu verstecken. Da noch wenig über die Malware bekannt ist, kann Ihr Ausbreitungsgebiet und Einsatzzweck noch nicht abgesteckt werden. [13]

Darkcomet
: Fernwartungssoftware mit einfach zu bedienender Oberfläche. Kann mit einem breiten Spektrum an Funktionalität paketiert werden. Installiert sich durch direkte Interaktion mit dem Opfer. Wurde unter anderem in Syrien zur Spionage verwendet. [19]

XtremeRAT
: Fernwartungssoftware mit ähnlichem Funktionsprofil und Angriffsvektor wie Darkcomet. Wurde zu Spionagezwecken gegen Energieproduzenten, Finanzinstitute, High-Tech Firmen und einige Staaten eingesetzt. [35]

Dridex
: Bankentrojaner der sich über Microsoft Word-Makros installiert und dezentralisiert kommuniziert. Die Malware wird über Spam global verteilt. [23]

Retefe
: Bankentrojaner der sich darauf beschränkt einen DNS-Eintrag anzulegen und ein Zertifikat zu installieren. Löscht sich nach der Installation vom System. Verbreit in der Schweiz, Österreich, Schweden und Japan. [9]

Cosmicduke
: Ein aus der Cosmu-Familie stammende Malware die zum Stehlen von Informationen verwendet werden kann. Installiert sich über eine Schwachstelle in Adobe Reader oder durch direkte Interaktion mit dem Opfer. Wurde für gezielte Attacken gegen NATO- und Europäische-Staaten eingesetzt. [12]

Zeus
: Ein modularer Bankentrojaner-Werkzeugkasten. Die Malware wird auf das Ziel zugeschnitten und versteckt sich bei der Installation in verschiedenen Prozessen. Kommuniziert wird über eine Server-Client Architektur. Hauseigene Werkzeuge erlauben die einfache Administration der Installationen. Global verbreitet und aktiv. [4]

2.3. Deckard

Neben den Malware-Familien verwenden wir Deckard-Cluster als eine zweite Quelle um unsere Resultate zu validieren. An dieser Stelle ein kurzer Auszug aus der Arbeit die Deckard beschreibt [36]:

> Deckard consists of four parts that are briefly introduced here and will reoccur throughout the document. **Code extraction** is concerned with dynamically extracting code from malware samples, no matter where and when the code appears in a system. This allows us to circumvent many methods of obfuscation and anti-analysis. We use a novel technique called memory tracing, which is able to detect memory changes of a system under inspection, while executing a malware sample. The code extraction part returns the disassembled code of the malware, which is then stored in a database. **Code similarity** introduces a similarity measure between two code functions, providing a necessary basis to evaluate code similarities on a larger scale. The focus hereby lies on performance and reusability. The similarity measure and a multi-staged search algorithm are the core pieces of the **similarity search**, which provides a scalable search engine to identify similar code. Based upon the similarity search, a **clustering** technique is introduced, able to automatically reveal code similarities between a large number of malware samples. By connecting the aforementioned parts, Deckard is able to automatically process a malware sample, discover code similarities and provide analysis thereof in minutes. Deckard is flexible in its usage, ranging from automatically analyzing code functions to visualizing the evolution of a malware family's codebase.

2.4. Graph-Distanzen

Ein Graph ist eine Datenstruktur. Da sich Daten aller Art als Graphen repräsentieren lassen, erhofft sich die Wissenschaft von der Erforschung Graph-basierter Ansätze wiederverwendbare Lösungen für eine Vielzahl von Problemen. Ein Forschungsgebiet beschäftigt sich mit der Ähnlichkeit von Graphen. Die Ähnlichkeit zweier Graphen wird dabei als Distanz d ausgedrückt. Algorithmen die Distanzen zwischen zwei Graphen ermitteln, können in drei Kategorien unterteilt werden [33] [15]:

Eigenschafts-basierte
: ermittelt wohldefinierte Eigenschaften der zu vergleichenden Graphen, zum Beispiel den Durchmesser, die Planarität, Eigenvektoren oder den Zusammenhang. All diese Eigenschaften ergeben pro Graph einen Vektor. Die einzelnen Graph-Distanzen sind dann gleich der Distanzen zwischen diesen Vektoren.

Editdistanz-basierte
: berechnen die Modifikationen um den einen Graph in den anderen zu transformieren. Die Graph-Distanz ist danach gleich der Summe der gewichteten Modifikationen.

Substruktur-basierte
: suchen nach Strukturen die in beiden Graphen vorkommen. Solche treten zum Beispiel auf wenn eine Malware mit dem Ziel Administratorrechte zu erlangen, den Flashplayer verändert und dann startet. Die Graph-Distanzen leiten sich danach von den übereinstimmenden Substrukturen ab.

2.5. Clusteralgorithmen

Clusteralgorithmen bezeichnen Methoden des maschinellen Lernens die zum Ziel haben Daten $\mathcal{X}$ anhand einer Distanz d sinvoll zu unterteilen. Solche Unterteilungen nennen wir *Clusterung* Z. Dabei bezeichnet $Z(x)$ den Teil der Daten der $x \in \mathcal{X}$ enthält, das sogenannte *Cluster* C von x. Die Literatur unterscheidet drei Grundtypen von Clusteralgorithmen [16]:

Partitionierende
: unterteilen $\mathcal{X}$ in exakt k Cluster. Da k bekannt sein muss, setzen partitionierende Clusteralgorithmen grosses Wissen über das Einsatzumfeld voraus. Partitionierende Clusteralgorithmen erstellen eine Startgruppierung und optimieren dann iterativ die Durchschnittsdistanzen zu Clustercentern.

Hierarchische
: unterteilen $\mathcal{X}$ iterativ in immer kleiner werdende Cluster. Dies wird solange wiederholt bis eine Abbruchbedingung eintrifft.

Dichte-basierende bilden in $\mathcal{X}$ dort Cluster wo Punkte gemäss d eng zusammen liegen. Die Dichte ist dabei die durchschnittliche Distanz zwischen allen Punkten in einem Cluster.

DBSCAN (Density-Based Spatial Clustering of Applications with Noise) [11] ist einer der meist zitierten dichte-basierten Algorithmen [21]. Da die Dichte der gefunden Cluster aber direkt von einem Parameter abhängt, ist DBSCAN nicht geeignet Cluster mit unterschiedlichen Dichten zu finden. Aus diesem Grund wurde DBSCAN zu *HDBSCAN* (Hierarchical Density-Based Spatial Clustering of Applications with Noise) [6] weiterentwickelt.

2.5.1. HDBSCAN

HDBSCAN ist eine Kombination aus hierarchischem und dichte-basierten Clusteralgorithmus. Da eine gut lesbare Beschreibung des Algorithmus der offiziellen Dokumentation[2] beiliegt, erklären wir an dieser Stelle nur vereinfacht die beiden wichtigen Konzepte der *Stabilität* und des *Noise*.

Der Noise einer Clusterung leitet sich von der minimalen Clustergrösse ab. Dieser Parameter von HDBSCAN definiert die minimale Anzahl Punkte die ein Cluster ausmachen. Alle Punkte in einem kleineren Cluster sind Noise. Dies bedeutet, dass Noise-Punkte zu wenig nahe Nachbarn haben um ein eigenständiges Cluster zu bilden. Für alle nicht Noise-Punkte, berechnet HDBSCAN eine Stabilität. Die Stabilität gibt an wie gut der Punkt die Anderen im Cluster repräsentiert. Die Stabilität nimmt dabei Werte im Intervall $[0, 1]$ an, wobei 1 einen perfekten Repräsentanten auszeichnet.

Wir verwenden in unserer Arbeit die folgende formale Definition von HDBSCAN:

Definition 1. *$\mathbb{X}$ sei die Menge aller von HDBSCAN clusterbaren Datenpunkte, $\mathcal{X} \subseteq \mathbb{X}$ die Menge aller Datenpunkte der Clusterung und $D \in \mathbb{R}^{|\mathcal{X}|\times|\mathcal{X}|}$ eine Matrix die eine Distanz zwischen allen Punkten repräsentiert, dann ist $HDBSCAN_D$ eine Funktion $HDBSCAN_D : \mathcal{X} \to \mathbb{N}_0 \cup \{-1\}$ die allen $x \in \mathcal{X}$ in einem Cluster ein eindeutiges Cluster-Label zuordnet. Ist $x \in \mathcal{X}$ Noise, liefert $HDBSCAN_D$ den Wert -1 zurück.*

$$HDBSCAN_D(x) = \begin{cases} clusterlabel \in \mathbb{N}_0, & x \text{ clustert} \\ -1, & x \text{ ist Noise} \end{cases} \tag{2.1}$$

Weiter berechnet der Algorithmus 1 mit HDBSCAN Label für eine Menge von reellwertigen Datenpunken $\mathcal{X} \subseteq \mathbb{R}$. Die resultierenden Label sind dabei gemäss Bedingung 2.2 geordnet.

$$\forall_{x_1\in\mathcal{X}}\forall_{x_2\in\mathcal{X}} x_1 \leq x_2 \Leftrightarrow label(x_1, \mathcal{X}) \leq label(x_2, \mathcal{X}) \tag{2.2}$$

Algorithmus 1 Berechne ein Label anhand sortierter HDBSCAN-Cluster

```
1:  procedure label(p, 𝒳)
2:      p ← Der Datenpunkt für den das Label bestimmt werden soll
3:      𝒳 ← Die Datengrundlage der Clusterung
4:      D_ij ← |𝒳_i − 𝒳_j|          ▷ Berechne die Distanmatrix D anhand der Differenz der Punkte
5:      C_l ← {x ∈ 𝒳 | HDBSCAN_D(x) = l}          ▷ 𝒳 in Cluster unterteilen
6:      n ← −1
7:      for x ∈ C_{−1} do          ▷ Jedem Noise-Punkt ein eigenes Label geben
8:          C_n ← {x}
9:          n ← n − 1
10:     C ← sort(C, min)          ▷ Anhand des Minimalwertes der Cluster diese sortieren und neu labeln
11:     return l wobei p ∈ C_l
```

[2] `http://hdbscan.readthedocs.io/en/latest/how_hdbscan_works.html`

2.6. Masszahlen

Wir verwenden einige Masszahlen um die Resultate unserer Clusterung Z zu bewerten. In der Literatur werden Masszahlen zur Qualitätsbewertung von Cluster in zwei Gruppen unterteilt [14]. Externe Masszahlen vergleichen zwei Clusterungen Z_1 Z_2 und interne betrachten lediglich die Struktur der Clusterung Z. Eine externe Masszahl ist der *Rand-Index*. Als interne Masszahlen wurden die Stabilität und der Nicht-Noise-Anteil bereits im Abschnitt 2.5 vorgestellt. Eine weitere interne Masszahl ist der *Silhouettenkoeffizient*.

2.6.1. Rand Index

Der Rand Index [29] misst anhand einer Menge von Datenpunkten $\mathcal{X}$ die Ähnlichkeit von zwei verschiedenen Clusterungen Z_1 und Z_2 gemäss Gleichung 2.3.

$$Rand(Z_1, Z_2) = \frac{|A \cup B|}{|A \cup B \cup C \cup D|} = \frac{|A \cup B|}{\binom{|\mathcal{X}|}{2}} \tag{2.3}$$

Wobei sich die vier Mengen A, B, C und D gemäss Gleichung 2.4 zusammensetzen.

$$A = \{(x_i, x_j) \in (\mathcal{X} \times \mathcal{X}) \mid Z_1(x_i) = Z_1(x_j) \wedge Z_2(x_i) = Z_2(x_j)\} \tag{2.4a}$$
$$B = \{(x_i, x_j) \in (\mathcal{X} \times \mathcal{X}) \mid Z_1(x_i) \neq Z_1(x_j) \wedge Z_2(x_i) \neq Z_2(x_j)\} \tag{2.4b}$$
$$C = \{(x_i, x_j) \in (\mathcal{X} \times \mathcal{X}) \mid Z_1(x_i) = Z_1(x_j) \wedge Z_2(x_i) \neq Z_2(x_j)\} \tag{2.4c}$$
$$D = \{(x_i, x_j) \in (\mathcal{X} \times \mathcal{X}) \mid Z_1(x_i) \neq Z_1(x_j) \wedge Z_2(x_i) = Z_2(x_j)\} \tag{2.4d}$$

Interpretieren wir Z_2 als eine "Wahre" Clusterung, unterteilen diese vier Mengen alle Paare in richtig positive A, richtig negative B, falsch positive C und falsch negative D. Der Wertebereich des Rand Index liegt im Intervall $[0, 1]$. Ein Rand-Index von 0 bedeutet, dass die Clusterungen Z_1 und Z_2 überhaupt nicht übereinstimmt.

2.6.2. Silhouettenkoeffizient

Der Silhouettenkoeffizient $s(Z)$ ist eine von der Anzahl Cluster unabhängige Masszahl [30]. Gemäss Gleichung 2.5 ist $s(Z)$ gleich dem durchschnittlichen *Silhouettenwerte* $s(x, Z)$ einer Clusterung Z.

$$S(Z) = \frac{\sum_{x \in \mathcal{X}} s(x, Z)}{|\mathcal{X}|} \tag{2.5}$$

Ein Silhouettenwert von einem Datenpunkt x berechnen wir in zwei Schritten. Erst ermitteln wir die Durchschnittsdistanz zu Punkten im gleichen Cluster $a(x, Z)$. Danach berechnen wir die minimale Durchschnittsdistanz von x zu allen Punkten, die nicht im gleichen Cluster wie x sind $b(x, Z)$. Dabei ist $b(x, Z) = min(d(x, C_i))$ und $d(x, C_i)$ die Durchschnittsdistanz von x zu allen Datenpunkten in Cluster C_i für $C_i \neq Z(x)$. Abbildung 2.1 veranschaulicht diese Überlegung anhand einer Clusterung bestehen aus drei Clustern (C_0, C_1, C_2).

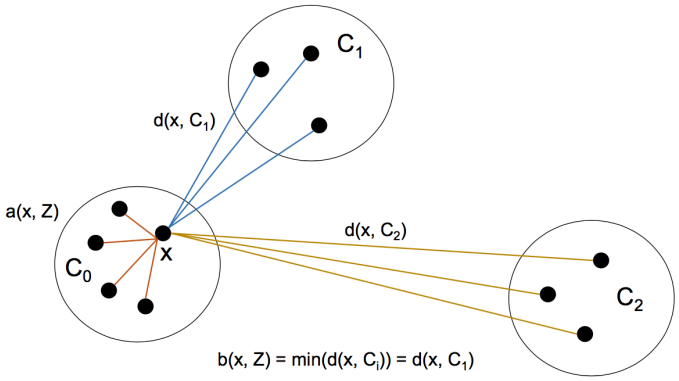


Abbildung 2.1.: Berechnung des Silhouettenwertes für x in Cluster C_0 [30]

Der Silhouettenwert $s(x, Z)$ eins Punktes x ist gemäss Gleichung 2.6 die Differenz von $b(x, Z)$ und $a(x, Z)$ geteilt durch den grösseren der beiden Werte.

$$s(x, Z) = \frac{b(x, Z) - a(x, Z)}{max(a(x, Z), b(x, Z))} \tag{2.6}$$

Der Wertebereich des Silhouettenkoeffizient $S(Z)$ liegt dabei im Intervall $[-1, 1]$. Bei einem Wert von 1 sind die Distanzen innerhalb der Cluster deutlich kleiner als die Distanzen zum nächsten Cluster. Dies bedeutet, dass die Cluster klar getrennt sind. Der Koeffizient wird 0, wenn die Distanzen innerhalb der Cluster ungefähr gleich gross sind wie die Distanz zum nächsten Cluster. In diesem Fall sind die Punkte gleichmässig verteilt. Bei einem Wert von -1 sind die Distanzen innerhalb der Cluster grösser als die Distanz zum Nachbarscluster. Im letzten Fall sind die Cluster also nicht klar getrennt.

2.7. Dimensionsreduktion

Das menschliche Auge ist darauf trainiert Strukturen in maximal drei Dimensionen zu erkennen. Um trotzdem Daten mit mehr als drei Dimensionen visuell interpretieren zu können, müssen wir diese auf zwei oder drei Dimensionen projizieren. Solche Projektionen nennen wir Dimensionsreduktionen. Es existiert eine Vielzahl von Algorithmen, die dies bewerkstelligen [26]. Wir verwenden in unserer Arbeit Multi-dimensionale Skalierung (MDS) um aus Distanzinformationen zweidimensionale Projektionen zu errechnen. MDS-Verfahren versuchen alle einen Abbildungsfehler, den sogenannte *Stress*, zu minimieren.

Definition 2. *$\mathcal{X}$ sei die Menge aller Punkte, $d(x, y)$ die wahre Distanz zweier Punkte und $\hat{d}(x, y)$ die euklidische Distanz der beiden Punkte in der errechneten Abbildung. Dann ist der Stress $stress$ gegeben durch Gleichung 2.7.*

$$stress = \sum_{(x,y)\in(\mathcal{X}\times\mathcal{X})} (d(x, y) - \hat{d}(x, y))^2 \tag{2.7}$$

3. Methodik

Wir haben drei Schritte ausgearbeitet zur Prüfung der im Kapitel 1 vorgestellten Hypothese. Erst transformieren wir die Memory-Traces zu Graphen. Danach vergleichen wir diese anhand einer selbst entwickelten Distanz. Distanzen zwischen den Graphen sind die Grundlage einer Clusterung. Anhand des Rand-Index und des Nicht-Noise-Anteils führen wir zum Schluss einen statistischen Test durch. Dieses Kapitel stellt die einzelnen Schritte vor.

3.1. Installationsgraphen erstellen

Erst transformieren wir Traces mit *Workspaces* zu *Installationsgraphen*.

Definition 3. *T sei die Menge aller Traces, $\mathbb{I}$ die Menge aller Installationsgraphen, dann ist ein Workspace w eine Funktion $w : T \to \mathbb{I}$ die aus einem Trace einen Installationsgraphen erstellt.*

Definition 4. *L sei die Menge aller Knoten- und Kantenlabels, $\mathcal{L}$ die Menge aller Attributlabels und $A \subseteq (\mathcal{L} \times \{a_1, a_2, \dots\})$ die Menge aller gelabelten Attribute. Dann ist ein Installationsgraph $\mathcal{I} = (\mathcal{V}, \mathcal{E}, \omega)$ ein Tupel bestehend aus:*

- *Einer Menge von gelabelten und attributierten Knoten $\mathcal{V} \subseteq (L \times \{attrs \mid attrs \subseteq A\})$*
- *Einer Menge von gelabelten Kanten $\mathcal{E} \subseteq (L \times \mathcal{V} \times \mathcal{V})$*
- *Einer Funktion ω die jedem Knoten und jeder Kante ein Gewicht zuordnet, $\omega : (\mathcal{V} \cup \mathcal{E}) \to \mathbb{N}$*

Workspaces erlauben eine flexible Sicht auf die Traces. Zum Beispiel könnte ein Workspace das im Kapitel 1 vorgestellte Überwachen von API-Aufrufen implementieren. Aus der Veröffentlichung von Illusion [34] lernen wir aber, dass eine robuste Clusterung viele verschiedene Sichten auf die Daten in Betracht ziehen muss. So haben wir gemäss Tabelle 3.1 gleich fünf verschiedene Workspaces definiert, die im Anhang A detailliert vorgestellt werden.

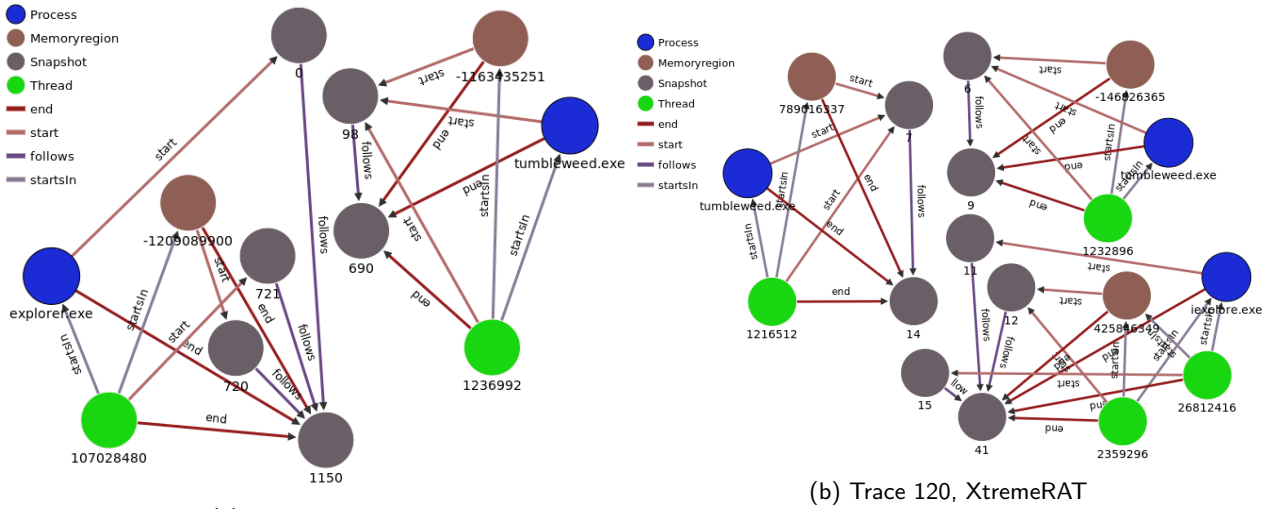
ID	Der Workspace extrahiert aus Traces ...
7414	Prozesse und deren Hierarchie
f9c9	Memoryregionen mit Code
9ab4	Ähnlichkeiten zwischen Memoryregionen unterschiedlicher Prozesse
0c7e	Das Laufzeitverhalten sowie den Ursprung der Threads
3449	Modifikationen von API-Funktionen in legitimen Modulen, sogenannte *Hooks*

Tabelle 3.1.: Die in dieser Arbeit verwendeten Workspaces im Überblick

Mit diesen Workspaces können wir aus jedem beliebigen Trace fünf verschiedene Typen von Installationsgraphen erstellen.

3.1.1. Beispiel

In Abbildung 3.1 haben wir Beispielhaft zwei Installationsgraphen aufgezeichnet, die mit Workspace 0c7e extrahiert wurden. startsIn-Kanten zeigen an von wem der Thread gestartet wurde. start-, end- und follows-Kanten codieren wann und wie lange die einzelnen Entitäten auf dem System liefen. Die Malware befindet sich im Prozess tumbleweed.exe.



(a) Trace 50, Fobber

(b) Trace 120, XtremeRAT

Abbildung 3.1.: Installationsgraphen des Workspace 0c7e

Die Codierung als Installationsgraph scheint geeignet da wir bereits durch diese einfache Visualisierung gewisse Unterscheide zwischen den Malware-Familien erkennen können. So sehen wir, dass tumbleweed.exe in Fobber viel länger läuft (Snapshot 98 bis 690) als der gleiche Prozess in XtremeRAT (Snapshot 6 bis 9 und 7 bis 14). Weiter unterscheiden sich die beiden Familien in der Anzahl nicht verbundenen Teilgraphen. Solche Beobachtungen deuten stark auf Unterschiede im Verhalten der entsprechenden Malware-Samples hin.

3.2. Clusteranalyse

3.2.1. Distanzen ermitteln

Im nächsten Schritt werden alle Distanzen zwischen den erstellten Installationsgraphen ermittelt. Dazu verwenden wir einen strukturbasierten Ansatz der nachfolgend motiviert wird.

Editdistanz-basierte und eigenschafts-basierte Algorithmen verlangen eine auf das zu lösende Problem zugeschnittene Gewichtung der Graphmodifikationen oder einen passenden Eigenschaftsvektor. Das Berechnen von Distanzen zwischen Installationsgraphen ist ein neuartiges Problem, somit können wir nicht auf Literaturwerte für diese Parameter zurückgreifen. Weiter macht die abstrakte Natur der beiden Herangehensweisen ein Abschätzen dieser Parameter schwierig. Aus diesem Grund wählen wir einen strukturbasierten Algorithmus. Strukturbasierte Algorithmen sind zwar auch nicht frei von Parametern, doch können diese mit Fachkenntnis sinnvoll abgeschätzt werden.

Die Literatur kennt bereits einige strukturbasierte Algorithmen, die aber alle nur in nichtdeterministischer polynomialer Zeit berechnet werden können [5]. Bis zum heutigen Zeitpunkt ist keine effiziente Implementation solcher Algorithmen bekannt. Dies bedeutet, dass strukturbasierte Ansätze mit der stetig wachsenden Zahl an Malware-Varianten nicht mithalten können. Lösungen, die auf solchen Algorithmen basieren, ersetzen die kritischen Stellen durch Heuristiken. Anders als die Algorithmen, versprechen diese in akzeptabler Zeit eine Lösung zu finden. Die in dieser Arbeit verwendete Heuristik basiert auf der Beobachtung, dass die meisten strukturbasierten Ansätze beliebige Graphen vergleichen. Dies ist aber gar nicht nötig, da die Problemdomäne in der Praxis bekannt ist. So motiviert, haben wir eine praxistaugliche Distanz für Installationsgraphen entwickelt.

Attribut-basierter Vergleich von Installationsgraphen

Installationsgraphen sind gelabelte Graphen. Aus diesem Grund erweitert unser Ansatz eine bekannte Distanzmetrik für gelabelte Graphen [7]. Analog zum bekannten Ansatz, definieren wir die Distanz zweier Intallationsgraphen über das Verhältnis der Schnittmenge zur Vereinigungsmenge.

Definition 5. *f sei eine monoton wachsende Funktion hinsichtlich der Vereinigung, dann definieren wir eine Distanz $\delta : (\mathbb{I} \times \mathbb{I}) \to \mathbb{R}$:*

$$\delta(\mathcal{I}_1, \mathcal{I}_2) = 1 - \frac{f(\mathcal{I}_1 \sqcap_M \mathcal{I}_2)}{f(\mathcal{I}_1 \cup \mathcal{I}_2)} \tag{3.1}$$

Wobei in unserem Ansatz die Funktion $f : \mathbb{I} \to \mathbb{N}$ die Summe der Gewichte aller Knoten und Kanten ist.

$$f(\mathcal{V}, \mathcal{E}, \omega) = \sum_{g \in (\mathcal{V} \cup \mathcal{E})} \omega(g) \tag{3.2}$$

Und sowohl die Vereinigung $\cup : (\mathbb{I} \times \mathbb{I}) \to \mathbb{I}$,

$$\begin{aligned} \mathcal{I}_1 \cup \mathcal{I}_2 = (& \\ & \mathcal{V}_1 \cup \mathcal{V}_2, \\ & \mathcal{E}_1 \cup \mathcal{E}_2, \\ & \omega(g) : \begin{cases} \omega_1(g), & g \in (\mathcal{V}_1 \cup \mathcal{E}_1) \wedge g \notin (\mathcal{V}_2 \cup \mathcal{E}_2) \\ \omega_2(g), & g \notin (\mathcal{V}_1 \cup \mathcal{E}_1) \wedge g \in (\mathcal{V}_2 \cup \mathcal{E}_2) \\ \omega_1(g) + \omega_2(g), & g \in (\mathcal{V}_1 \cup \mathcal{E}_1) \wedge g \in (\mathcal{V}_2 \cup \mathcal{E}_2) \end{cases} \\ ) & \end{aligned} \tag{3.3}$$

als auch die Schnittmenge bezüglich eines Mappings $\sqcap_M : (\mathbb{I} \times \mathbb{I}) \to \mathbb{I}$ ein Installationsgraph ist.

$$\begin{aligned} \mathcal{I}_1 \sqcap_M \mathcal{I}_2 = (& \\ & \{(l, v_1) \in \mathcal{V}_1 \mid \exists_{(l,v_2) \in \mathcal{V}_2} (v_1, v_2) \in M_l\} \\ & \cup \{(l, v_2) \in \mathcal{V}_2 \mid \exists_{(l,v_1) \in \mathcal{V}_1} (v_2, v_1) \in M_l\}, \\ & \{(l, v_1, v_1') \in \mathcal{E}_1 \mid \exists_{(l,v_2,v_2') \in \mathcal{E}_2} (v_1, v_2) \in M_l \wedge (v_1', v_2') \in M_l\} \\ & \cup \{(l, v_2, v_2') \in \mathcal{E}_2 \mid \exists_{(l,v_1,v_1') \in \mathcal{E}_1} (v_2, v_1) \in M_l \wedge (v_2', v_1') \in M_l\}, \\ & \omega(g) : \begin{cases} \omega_1(g), & g \in (\mathcal{V}_1 \cup \mathcal{E}_1) \wedge g \notin (\mathcal{V}_2 \cup \mathcal{E}_2) \\ \omega_2(g), & g \notin (\mathcal{V}_1 \cup \mathcal{E}_1) \wedge g \in (\mathcal{V}_2 \cup \mathcal{E}_2) \\ \omega_1(g) + \omega_2(g), & g \in (\mathcal{V}_1 \cup \mathcal{E}_1) \wedge g \in (\mathcal{V}_2 \cup \mathcal{E}_2) \end{cases} \\ ) & \end{aligned} \tag{3.4}$$

Dabei fügen wir der Schnittmenge Knoten des einen Installationsgraphen hinzu, wenn wir Knoten im anderen finden, sodass die beiden gemäss M in Relation stehen. Kanten fügen wir der Schnittmenge hinzu wenn wir Kanten im anderen Installationsgraphen finden, sodass die beiden Start- sowie Endknoten der Kanten gemäss M in Relation stehen.

Unser Ansatz unterscheidet sich im wesentlichen von [7] in der Berechnung des Mappings M. So wird im bekannten Ansatz das optimale Mapping zweier Installationsgraphen M mit Hilfe eines Greedy-Algorithmus in der Menge aller möglichen Mappings gesucht. Unser Ansatz entstammt der Beobachtung, dass in der Praxis über die Menge $\mathcal{A}_{l\ell}$ einiges bekannt ist.

Definition 6. *$\mathcal{A}_{l\ell}$ ist die Menge aller Attribute eines Installationsgraphen $\mathcal{I}$ mit gleichem Knoten- $l \in L$ sowie Attribut-Label $\ell \in \mathcal{L}$.*

$$\mathcal{A}_{l\ell} = \{x \mid \exists_{(l,attrs) \in \mathcal{V}} \exists_{(\ell,a) \in attrs} x = a\} \tag{3.5}$$

So wissen wir zum Beispiel, das Prozesse mit Namen svchost.exe Systemprozesse sind. Oder aber, dass der Windows-Linker main()-Threads mit einen auf 1 MiB begrenzten Stack startet [22]. Anhand dieser Feststellung berechnen wir M_l gemäss Gleichung 3.6.

$$M_l = \{(v_1, v_2) \in (\mathcal{V}_1 \times \mathcal{V}_2) \mid \forall_{(\ell,a_1) \in v_1} \exists_{(\ell,a_2) \in v_2} a_1 \sim_{l\ell} a_2\} \tag{3.6}$$

Dabei ist v_1 auf v_2 mappable wenn jedes Attribut a_1 mit mindestens einem Attribut a_2 gemäss $\sim_{l\ell}$ in Relation steht. Zum Beispiel können wir so ähnliche main()-Threads mit dem Attribut (stackLimit, 1048576) oder Prozesse

mit dem Attribut (name, svchost.exe) aufeinander mappen. Wir stellen fest, dass die einzelnen $\sim_{l\ell}$ eine zentrale Rolle für unseren Ansatz spielen. Da wir diese aber für jedes Attribut- und Knotenlabel wählen können, finden wir leicht auf das Problem zugeschnittene Relationen. In unserem Ansatz unterscheiden wir vier solche, die nachfolgend einzeln vorgestellt werden.

Für je zwei Attribute $a_1 \in \mathcal{A}_{l\ell}$ und $a_2 \in \mathcal{A}'_{l\ell}$ soll $a_1 \sim_{l\ell} a_2$ genau dann gelten, wenn ...

(exaktes vergleichen) ... deren Wert genau der gleiche ist. Wir verwenden diese Art von Vergleich für Namen von Prozessen, Namen von Modulen und Namen sowie die Typen von Hooks. Dies sind alles Attribute die stark identifizierend sind.

Beispiel Für die Installationsgraphen der Abbildung 3.1 finden wir mit dieser Art Vergleich ein Mapping zwischen Prozessen mit dem Namen tumbleweed.exe. In der Abbildung 3.2 ist das Mapping $M_{Process}$ schwarz eingezeichnet.

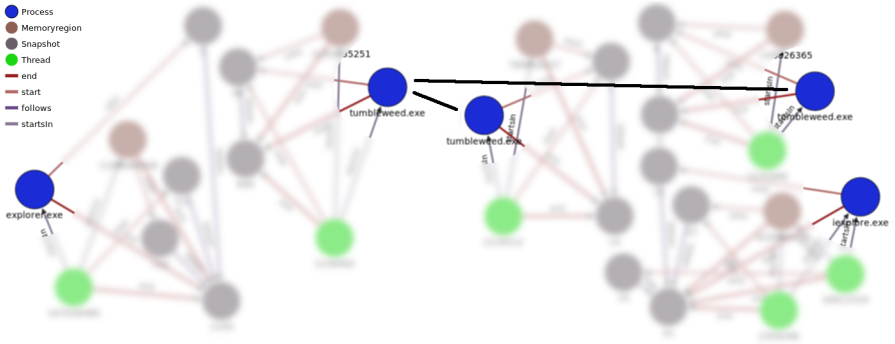


Abbildung 3.2.: Mapping aus exaktem Vergleichen der Attribute

(separiert neu labeln) ... $label(a_1, \mathcal{A}_{l\ell}) = label(a_2, \mathcal{A}'_{l\ell})$. Wobei $label$ für jedes Attribut ein Label anhand Algorithmus 1 berechnet. Wir verwenden diese Art von Vergleich für Zeitreihen, konkret Snapshots-Ids. Die Literatur kennt viele Ansätze um Zeitreihen zu vergleichen [1]. Diese gegeneinander abzuwägen und den passendsten Ansatz zu finden, würde aber den Rahmen dieser Arbeit sprengen. Wir beschränken uns deshalb mit $label$ auf einen simplen Vergleichsmechanismus der zwei Überlegungen berücksichtigt: Erstens betrachten wir der Einfachheit halber nur die zeitliche Abfolge der Snapthots. $label$ löst dies elegant indem es jedem Attribut geordnet ein neues Label zuordnet. Zweitens erwarte wir, dass Snapshots in kurzem Intervall dasselbe Ereignis repräsentieren. Durch den Einsatz von HDBSCAN in $label$ werden automatisch solche Intervalle unter einem Label zusammengefasst.

Beispiel In der Abbildungen 3.3 wurden die von $label(a_1, \mathcal{A}_{l\ell})$ beziehungsweise $label(a_2, \mathcal{A}'_{l\ell})$ berechneten Labels weiss eingezeichnet. Das resultierende Mapping $M_{Snapshot}$ wird durch die schwarzen Striche dargestellt.

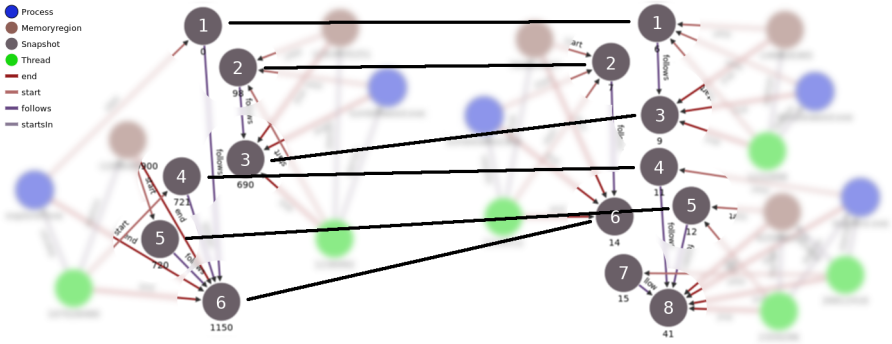


Abbildung 3.3.: Mapping aus separiertem neu Labeln der Attribute

(gemeinsam neu labeln) ... $label(a_1, \mathcal{A}_{l\ell} \cup \mathcal{A}'_{l\ell}) = label(a_2, \mathcal{A}_{l\ell} \cup \mathcal{A}'_{l\ell})$. Wobei $label$ für jedes Attribut eine Label anhand Algorithmus 1 errechnet. Wir verwenden diese Art von Vergleich für Stacklimitierungen von Threads und Ähnlichkeiten sowie Grössen von Pattern. Das Vereinigen der Mengen $\mathcal{A}_{l\ell}$ und $\mathcal{A}'_{l\ell}$ hat zur Folge, dass Attribute das gleiche Label erhalten deren Differenz klein ist. Dieser Ansatz implementiert damit eine Art fuzzy-Vergleich.

Beispiel In der Abbildungen 3.4 wurden die von $label(a_1, \mathcal{A}_{l\ell} \cup \mathcal{A}'_{l\ell})$ berechneten Label weiss eingezeichnet. Da kein Label zweimal vergeben wurde, ist das Mapping M_{Thread} leer.

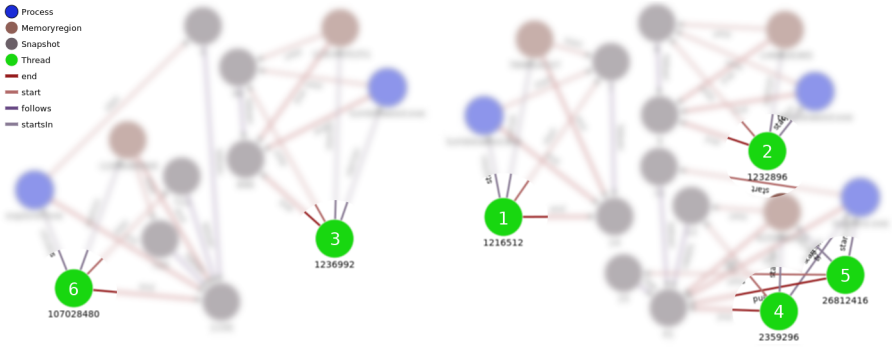


Abbildung 3.4.: Mapping aus gemeinsamen neu Labeln der Attribute

(sdhashes vergleichen) ... gemäss Sdhash-Vergleich diese stark übereinstimmen[3]. Wir verwenden diese Art von Vergleich für Sdhashes von Memoryregionen. Sdhash kommt aus dem gleichen Grund zum Einsatz wie in Workspace 9ab4.

Beispiel In der Tabelle 3.2 haben wir alle Sdhash-Vergleiche der Memoryregionen der Abbildung 3.5 angegeben. Wir stellen fest, dass einzig 789016337 eine hohe Übereinstimmung mit -146826365 aufweist. Da sich diese Memoryregionen aber im selben Installationsgraphen befinden, ist das Mapping $M_{Memoryregion}$ leer.

[3]Dies heisst $sdhash(x, y) \geq 21$ [32]

		Trace 50, Fobber		Trace 120, XtremeRAT		
		-1209089900	-1163435251	789016337	-146826365	425846349
Trace 50, Fobber	-1209089900	100	1	0	1	0
	-1163435251	1	100	0	0	0
Trace 120, XtremeRAT	789016337	0	0	100	83	0
	-146826365	1	0	83	100	1
	425846349	0	0	0	1	100

Tabelle 3.2.: Sdhash-Vergleiche von Memoryregionen der Traces 50 und 120

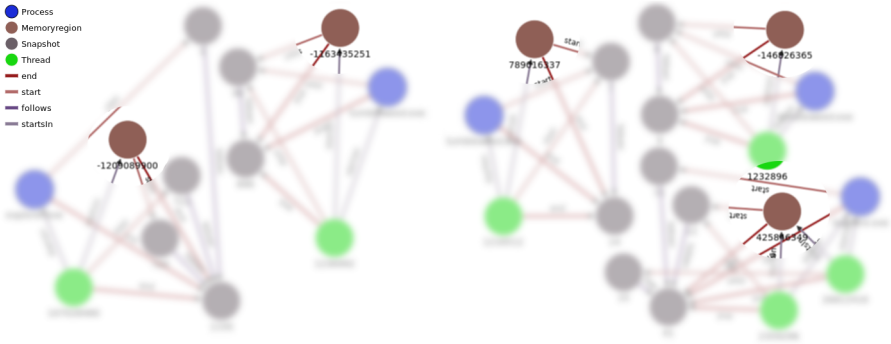


Abbildung 3.5.: Mapping aus sdhashes Vergleichen der Attribute

Die Distanz zwischen Installationsgraphen δ ist die Grundlage der Distanz zweier Traces. Um robust gegenüber Verschleierungstaktiken zu sein, soll unsere Distanzmetrik mehrere Workspaces berücksichtigen. So berechnen wir für alle 31 nichtleeren Workpsace-Kombinationen $\mathcal{W}$ der fünf Workspaces, die Distanzen zweier Traces anhand $d_{\mathcal{W}}$.

Definition 7. *$\mathbb{W}$ sei die Menge aller Workspaces und $t \in T$ ein Trace, dann ist für eine Workspace-Kombination $\mathcal{W} \in \{\mathcal{W} \mid \mathcal{W} \subseteq \mathbb{W} \wedge |\mathcal{W}| > 0\}$ die Distanz zweier Traces $d_{\mathcal{W}} : (T \times T) \to \mathbb{R}$:*

$$d_{\mathcal{W}}(t_1, t_2) = \frac{\sum_{w \in \mathcal{W}} \delta(w(t_1), w(t_2))}{|\mathcal{W}|} \tag{3.7}$$

3.2.2. Clusterbildung

Mit der Distanz $d_{\mathcal{W}}$ können wir für jede Workspace-Kombination eine Distanzmatrix $D_{\mathcal{W}} \in \mathbb{R}^{|T| \times |T|}$ gemäss Gleichung 3.8 aufstellen.

$$D_{\mathcal{W}_{ij}} = d_{\mathcal{W}}(t_i, t_j) \tag{3.8}$$

Anhand der Distanzmatrix $D_{\mathcal{W}}$ clustern wir danach mit einem Clusteralgorithmus die Traces. Da sich die Wahl das Algorithmus stark auf die Clusterbildung auswirkt, motivieren wir an dieser Stelle den verwendeten HDBSCAN.

Für den Einsatz von partitionierende Algorithmen muss vorgängig die Anzahl Cluster bekannt sein. Wir könnten nun die Zahl der Malware-Familien als solchen Parameter annehmen. Da wir dann aber immer gleich viele Cluster finden wie Malware-Familien, bleiben feinere Strukturen verborgen. Aus diesem Grund verwenden wir keine partitionierenden Algorithmen. Für Traces gleicher Malware-Familie erwarten wir, dass diese dicht zusammen liegen. Für unsere Arbeit sind somit dichte-basierte Algorithmen die richtige Wahl. Da sich die Schätzung der erwarteten Dichten als schwierig erweist, ist DBSCAN hingegen nicht geeignet um Traces zu clustern. Der Ansatz diese Schätzung durch optimieren des Silhouettenkoeffizienten zu errechnen, liefert keine befriedigende Resultate. MDS-Projektionen der

Distanzmatrizen bestätigen, dass die Traces Cluster unterschiedlicher Dichten formen. Aus diesem Grund ist das Schätzen einer erwarteten Dichte nicht möglich. Diesen Nachteil von DBSCAN beseitigt HDBSCAN. So errechnet HDBSCAN für Traces stabile Cluster.

Mit HDBSCAN können wir nun für jede Distanzmatrix $D_{\mathcal{W}}$ eine Clusterung $Z_{\mathcal{W}}$ bestimmen. Die Menge der durch kombinieren von Workspaces ermöglichten Clusterungen $\mathcal{Z}$ verwenden wir anschliessend um unsere Hypothese zu verifizieren. Für unseren fünf Workspaces setzt sich dabei die Menge $\mathcal{Z}$ gemäss Gleichung 3.9 zusammen.

$$
\begin{aligned}
\mathcal{Z} = \{ & \\
& Z_{\{7414\}}, Z_{\{f9c9\}}, Z_{\{9ab4\}}, Z_{\{0c7e\}}, Z_{\{3449\}}, \\
& Z_{\{7414,f9c9\}}, Z_{\{7414,9ab4\}}, Z_{\{7414,0c7e\}}, Z_{\{7414,3449\}}, Z_{\{f9c9,9ab4\}}, \\
& Z_{\{f9c9,0c7e\}}, Z_{\{f9c9,3449\}}, Z_{\{9ab4,0c7e\}}, Z_{\{9ab4,3449\}}, Z_{\{0c7e,3449\}}, \\
& Z_{\{7414,f9c9,9ab4\}}, Z_{\{7414,f9c9,0c7e\}}, Z_{\{7414,f9c9,3449\}}, Z_{\{7414,9ab4,0c7e\}}, Z_{\{7414,9ab4,3449\}}, \\
& Z_{\{7414,0c7e,3449\}}, Z_{\{f9c9,9ab4,0c7e\}}, Z_{\{f9c9,9ab4,3449\}}, Z_{\{f9c9,0c7e,3449\}}, Z_{\{9ab4,0c7e,3449\}}, \\
& Z_{\{7414,f9c9,9ab4,0c7e\}}, Z_{\{7414,f9c9,9ab4,3449\}}, Z_{\{7414,f9c9,0c7e,3449\}}, Z_{\{7414,9ab4,0c7e,3449\}}, \\
& Z_{\{f9c9,9ab4,0c7e,3449\}}, \\
& Z_{\{7414,f9c9,9ab4,0c7e,3449\}} \\
\} &
\end{aligned}
\tag{3.9}
$$

3.3. Verifikation der Hypothese

Wir nehmen an, dass ein Workspace alleine das Verhalten einer Malware nicht erfassen kann. So betrachten wir für die Prüfung der Hypothese nur Clusterungen, die eine *Aussage* machen, die nicht von einem Workspace *dominiert* wird.

Definition 8. *$\mathcal{Z}$ sei die Menge aller Clusterungen, dann ist gemäss Gleichung 3.10 eine Aussage A_l die Menge aller Clusterungen $Z \in \mathcal{Z}$, die anhand der Distanzmatrix $D_{ij} = 1 - Rand(Z_i, Z_j)$ dem Label l zugeordnet werden.*

$$A_l = \{Z \in \mathcal{Z} \mid l = HDBSCAN_D(Z)\} \tag{3.10}$$

Definition 9. *Die Zellen der Distanzmatrix D nennen wir Rand-Distanz.*

Definition 10. *$\mathbb{A}$ sei die Menge aller Aussagen, $\mathbb{W}$ die Menge aller Workspaces, dann ist gemäss Gleichung 3.11 die Menge aller dominierten Aussagen $\mathcal{D} \subseteq \mathbb{A}$. Wobei alle Clusterungen jeder dominierten Aussagen $Z \in A \in \mathcal{D}$ genau einen Workspace $w \in \mathbb{W}$ gemeinsam verwenden.*

$$\mathcal{D} = \{A \in \mathbb{A} \mid \exists_{w_1 \in \mathbb{W}} \forall_{Z_{\mathcal{W}_1} \in A} w_1 \in \mathcal{W}_1 \forall_{w_2 \in (\mathbb{W} \setminus w_1)} \exists_{Z_{\mathcal{W}_2} \in A} w_2 \notin \mathcal{W}_2\} \tag{3.11}$$

Als nächsten Schritt betrachten wir für die verbliebenen Clusterungen $Z \in A \notin \mathcal{D}$ zwei externe sowie eine interne Masszahl. Als externe Masszahlen verwenden wir Rand-Indizes bezüglich den Malware-Familien sowie der Deckard-Cluster. Als interne Masszahlen bewerten wir den *Nicht-Noise-Anteil*.

Definition 11. *T sei die Menge aller Traces, dann ist gemäss Gleichung 3.12 der Nicht-Noise-Anteil NNA einer Clusterung $Z_{\mathcal{W}}$ das Verhältnis der Traces, die einem Cluster zugeordnet wurden zu der Gesamtzahl der Traces.*

$$NNA(Z_{\mathcal{W}}) = \frac{|\{t \in T \mid HDBSCAN_{D_{\mathcal{W}}}(t) \neq -1\}|}{|T|} \tag{3.12}$$

Basierend auf diesen Masszahlen führen wir danach einen statistischen Test durch.

Für alle Paare von Traces prüfen wir anhand einer vorgegeben Clusterung Z_2 ob diese in unseren Clusterungen Z_1 *richtig* oder *falsch* geclustert wurden. Ein Paar ist richtig geclustert, wenn die beiden Traces in Z_1 und Z_2 im selben oder in unterschiedlichen Clustern sind. Dies entspricht den Mengen A und B des Rand-Index. Falsche Paare sind dazu komplementär, also die Mengen C und D. Durch den richtig/falsch Charakter des Testaufbaus motiviert,

führen wir mit Hilfe einer Binomialverteilung $B(k \mid p, n)$ wobei $n = \lfloor \binom{|T|}{2} \cdot NNA(Z_{\mathcal{W}}) \rfloor$, einen rechtsseitigen Test durch. Für eine Signifikanz von $\alpha = 1\%$ prüfen wir folgende Null- H_0 sowie Alternativhypothese H_1:

$$H_0 : p \leq 0.9 \tag{3.13a}$$

$$H_1 : p > 0.9 \tag{3.13b}$$

Als Nullhypothese H_0 nehmen wir an, dass unsere Methode für weniger oder exakt 90% der Paare eine richtige Aussage macht. In der Alternativhypothese nehmen wir an, dass unsere Methode für mehr als 90% der Paare eine richtige Aussage macht. Die Hypothese dieser Arbeit gilt als bestätigt, wenn für alle Clusterungen mindestens einer Aussage die Alternativhypothese signifikant gilt.

4. Implementation

Die im Kapitel 3 vorgestellten Schritte haben wir in einem System implementiert. Die Abbildung 4.1 ist eine Übersicht der dazu verwendeten Technologien und Skripte.

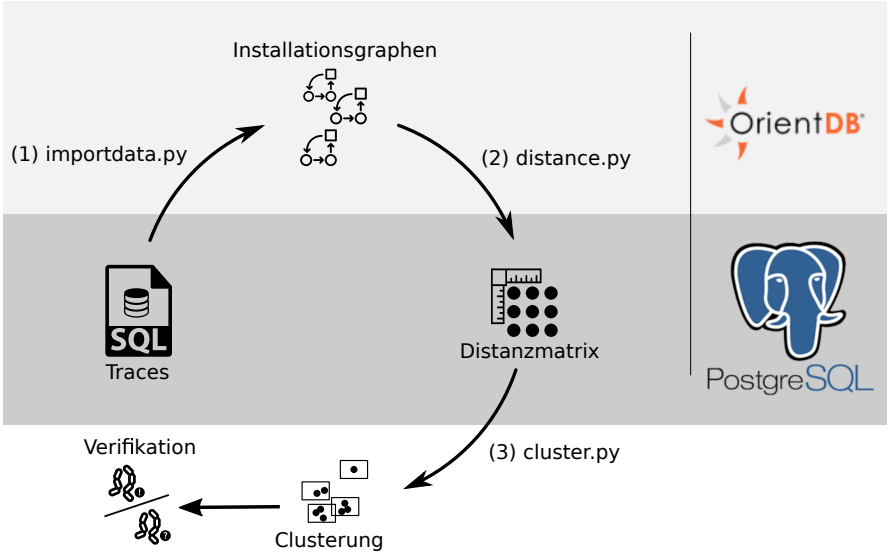


Abbildung 4.1.: Technologie- und Systemübersicht

4.1. Systembeschreibung

Mit Ausnahme von sql2fluentflow haben wir alle Teile dieses Systems als Python-Scripte realisiert.

```
src/
├── importdata.py ..... Generiert mit sql2fluentflow die Installationsgraphen
├── distance.py ..... Berechnet Distanzen zwischen den Installationsgraphen
├── vertices.py ..... Knoten der Installationsgraphen
├── edges.py ..... Kanten der Installationsgraphen
├── cluster.py ..... Erstellt die HDBSCAN Clusterung und generiert Statistiken
├── config.py ..... Globale Konfiguration
├── compare_cluster.py ..... Vergleicht die Clusterungen mit DECKARD
├── measure_time.py ..... Laufzeitanalyse der Distanzberechnung sowie des HDBSCAN
├── utils.py ..... Funktionen zum Erstellen von Grafiken sowie zum Laden der Distanzmatrizen
└── lib/
    ├── graph.py ..... Graph, Vertex, Edge Basisklassen
    ├── persistentdict.py ..... Implementation eines persistenten Dictionaries
    ├── pure.py ..... Implementation des @pure Dekorators
    ├── sdhash.py ..... Implementation des sdhash Vergleichsalgorithmus
    └── statistics.py ..... Implementation des Rand-Index
```

Die folgenden Abschnitte haben zum Ziel den Einsatz der einzelnen Skripte zu erklären.

4.1.1. Installationsgraphen erstellen

Die fünf in Kapitel 3.1 vorgestellten Workspaces sind in sql2fluentflow abgelegt. Jeder solche Workspace lädt mit Hilfe von SQL-Queries die Trace-Informationen aus der KAN-Datenbank und verarbeitet diese danach mit FluentFlow zu einem Installationsgraphen. Die so erstellten Installationsgraphen importieren wir in ein Graph-basiertes Datenbanksystem. Für jeden Workspace existiert eine Datenbank. **importdata.py** legt erst alle Datenbanken an und koordiniert danach den eben beschriebenen Importvorgang. Als Datenbankmanagementsystem setzen wir OrientDB[4] ein. OrientDB ist auf Graph-Datenstrukturen optimiert. So können zum Beispiel, ausgehend von einem Startknoten, Graphen traversiert werden. OrientDB stellt uns zusätzlich ein Tool zur Visualisierung der Installationsgraphen zur Verfügung. Abbildungen 3.1 wurden damit erstellt. Zur Laufzeitoptimierung führen wir die Workspaces parallel aus.

4.1.2. Clusteranalyse

Distanzen ermitteln

Im zweiten Schritt erstellen wir mit **distance.py** alle Distanzmatrizen $D_{\mathcal{W}}$. Dazu laden wir die Installationsgraphen aus OrientDB in eine passende Datenstruktur. Diese ist in **lib/graph.py** implementierte und kann durch Vererbung einfach um neue Kanten (**vertices.py**) und Knoten (**edges.py**) erweitert werden. Die für die Berechnung von δ wichtige Gleichung 3.6 ist mit der Methode mapping() implementiert. Dabei definiert mapping() für jedes Graphelement ob dieses auf einen anderes abgebildet werden kann. Dank Polymorphismus kann dies für alle Kanten und Knoten angepasst werden.

So implementiert zum Beispiel mapping() von Memoryregionen einen sdhash-Vergleich. Aus diesem Grund haben wir in **lib/sdhash.py** die in der Programmiersprache C geschriebene Referenzimplementation[5] für Sdhash-Vergleiche in Python nachprogrammiert. Sdhashes zu vergleichen ist zeitintensiv aber glücklicherweise eine pure Funktion.

Definition 12. *Eine Funktion ist pur wenn gilt:*

- *Die Funktion liefert bei gleich bleibenden Argumenten, das gleiche Resultat.*
- *Das Ausführen der Funktion hat keine Seiteneffekte.*

Mit **lib/pure.py** haben wir einen Python-Dekorator implementiert, der für alle Aufrufe einer Funktion die Rückgabewerte abspeichert. Wird eine Funktion zweimal mit den selben Parametern aufgerufen, erkennt @pure dies und liefert ohne Funktionsaufruf den Rückgabewert. In **lib/sdhash.py** verwenden wir @pure gemäss Algorithmus 2.

Algorithmus 2 Einsatz von @pure in lib/sdhash.py

```
def _compare(sdhashes, pure_state):
    """Compares sdhash tuples."""
    @pure(pure_state)
    def do_compare(sdhash1, sdhash2):
        return Sdhash(sdhash1).compare(Sdhash(sdhash2))
    return do_compare(*sdhashes)
```

Dabei ist das Argument pure_state der interne @pure-Cache. Diesen persistieren wir zwischen Aufrufen von **distance.py** mit **lib/persistentdict.py**. Durch diesen Trick können wir viele Sdhashes schnell vergleichen, was eine deutliche Laufzeitreduktion für δ bedeutet. Die mit δ berechneten Distanzmatrizen $D_{\mathcal{W}}$ werden danach in einer PostgreSQL[6] Datenbank gespeichert.

[4] http://orientdb.com/
[5] https://github.com/sdhash/sdhash
[6] https://www.postgresql.org/

Clusterbildung

Im nächsten Schritt erstellt **cluster.py** aus den Distanzmatrizen $D_{\mathcal{W}}$ eine Clusterung. Erst werden die einzelnen Distanzmatrizen $D_{\mathcal{W}}$ aus der PostgreSQL Datenbank geladen und danach mit HDBSCAN geclustert. Die HDBSCAN-Implementation haben wir aus scikit-learn-contribute[7], eine zu scikit-learn[8] kompatiblen Bibliothek. Aus scikit-learn verwenden wir auch die Implementation zur Berechnung der Silhouettenkoeffizienten.

4.1.3. Verifikation der Hypothese

compare_cluster.py lädt eine in JSON abgespeicherten Clusterung und berechnet den Rand-Index mit **lib/statistics.py**. So vergleichen wir zum Beispiel unsere mit der Deckard-Clusterung. Die Clusterung stellen wir mit in **utils.py** implentierten plot-Funktionen dar. Wir verwenden dazu matplotlib[9]. **utils.py** enthält ebenfalls Funktionen zum Auslesen der Distantmatrix aus der PostgreSQL-Datenbank. Um Zeit zu sparen berechnen wir die Clusterbildung parallelisiert.

config.py enthält die Konfiguration aller vorgestellten Skripte. So können wir zum Beispiel global die Parameter des Clusteralgorithmus definieren, oder aber die verwendeten Traces einschränken.

Die Laufzeitmessungen des folgenden Abschnittes sind in **measure_time.py** implementiert.

4.2. Praxistauglichkeit

Der im Kapitel 3 vorgestellte Ansatz berechnet für n Traces die Distanzmatrix $D_{\mathcal{W}}$ und clustert danach diese mit HDBSCAN. Anhand Messungen machen wir uns in diesem Abschnitt erst ein Bild von den Laufzeiten der beiden Schritte und versuchen danach eine Obergrenze der Laufzeitkomplexität abzuleiten. Die Laufzeiten ΔT_1 und ΔT_2 der Gleichung 4.1 bestimmen wir mit Algorithmus 3.

$$\Delta T_1 = measure\_time(D_{\mathcal{W}}, 1, 20, 10) \tag{4.1a}$$

$$\Delta T_2 = measure\_time(\text{HDBSCAN}, 10^3, 20, 10) \tag{4.1b}$$

Algorithmus 3 Zeitmessung für diverse Funktionen

```
1:  procedure measure_time(f, s, m, r)
2:      f ← Die zu vermessende Funktion
3:      s ← Die Skalierung der Messung
4:      m ← Die Anzahl Messwerte pro Eingabegrösse
5:      r ← Die Anzahl Wiederholungen pro Messwert
6:      i_min ← 1
7:      i_step ← 5
8:      i_max ← 100
9:      for i : i_min --i_step--> i_max do          ▷ Die Messreihen generieren
10:         n ← i · s                               ▷ Skalieren von n
11:         for j : 1 --1--> m do
12:             x ← Generiere Eingabedaten der Grösse n
13:             t_0 ← time()                        ▷ Zeitmessung starten
14:             for k : 1 --1--> r do
15:                 Berechne f mit Eingabedaten x
16:             ΔT_ij ← (time() − t_0) / r          ▷ Zeitmessung stoppen
17:     return ΔT
```

[7] https://github.com/scikit-learn-contrib/scikit-learn-contrib/
[8] http://scikit-learn.org/
[9] http://matplotlib.org/

Die Zeilen der ΔT-Matrizen sind dabei die Messreihen. Anhand dieser Messreihen zeichnen wir in der Abbildung 4.2 und 4.3 die einzelnen Boxplots. Die Länge der Whisker beschränkten wir dabei auf ein 1.5-faches des Interquartilabstandes. Danach legen wir eine Funktion $g : \mathbb{N} \to \mathbb{R}$ mit $g(x) = a + b \cdot x^c$ durch Anpassen der Parameter a, b und c in die Daten.

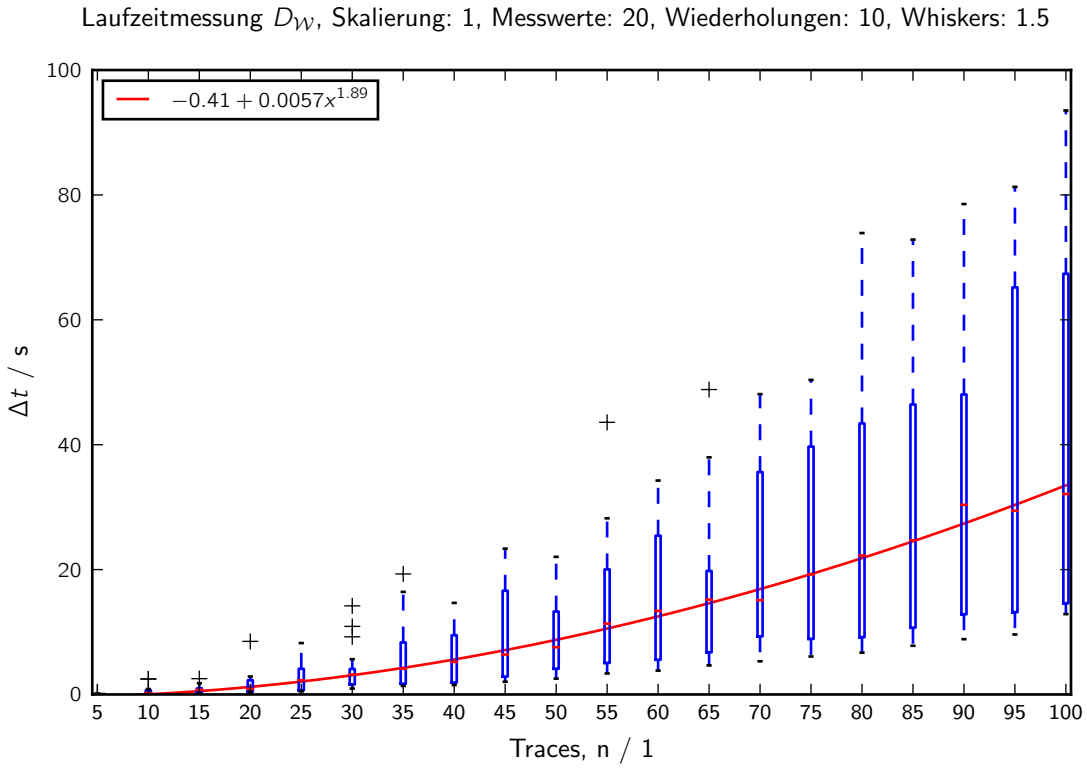


Abbildung 4.2.: Messreihe ΔT_1, Laufzeitverhalten für die Berechnung von $D_{\mathcal{W}}$

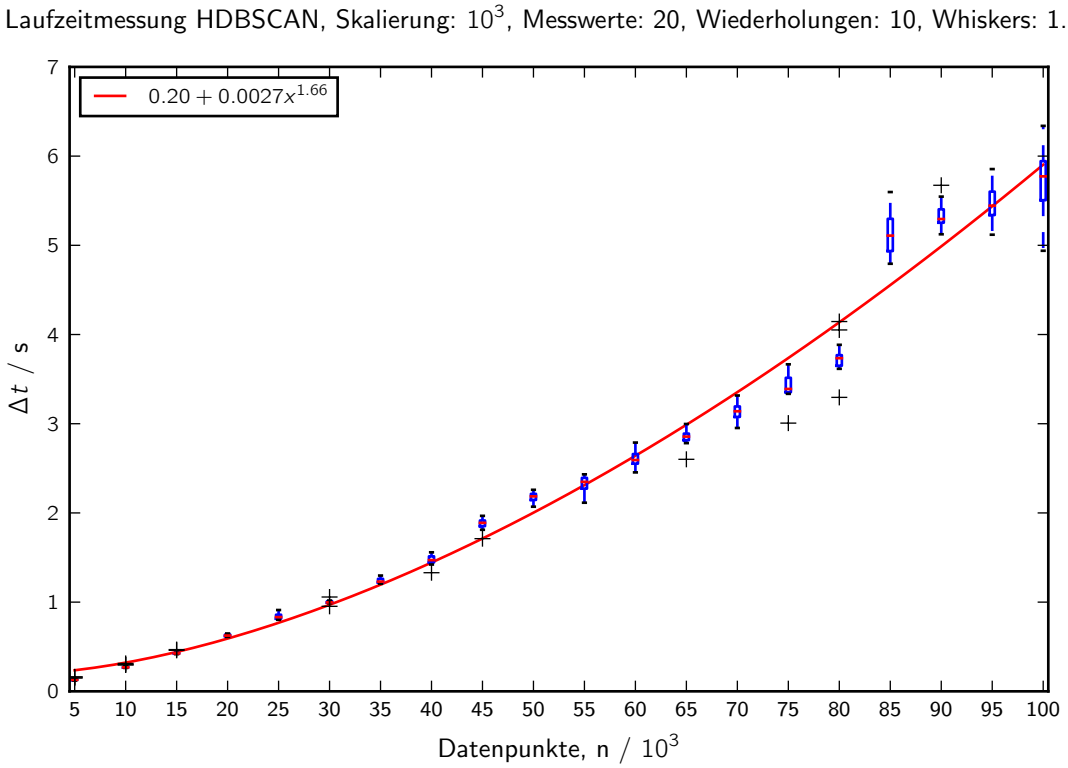


Abbildung 4.3.: Messreihe ΔT_2, Laufzeitverhalten von HDBSCAN

Wir stellen fest, dass die Berechnung der Distanzmatrizen $D_{\mathcal{W}}$ wesentlich länger dauert als die Clusterbildung. Die Laufzeit von HDBSCAN kann also ignoriert werden. Unser Ansatz benötigt um 100 Traces zu analysieren ungefähr 30 Sekunden. Weiter beobachten wir, dass die Boxplots der Abbildung 4.2 mit grösser werdendem n auch immer grösser werden. Dies kommt daher, dass wir unterschiedliche Workspaces-Kombinationen $\mathcal{W}$ messen. So wächst zum Beispiel die Laufzeit für die Berechnung von $D_{7414-f9c9-9ab4-3449-0c7e}$ schneller als die von D_{7414}. Weiter stellen wir in Abbildung 4.3 einen Sprung der Laufzeiten zwischen $n = 80$ und $n = 85$ fest. In nicht dokumentierte Messungen grösserer Intervalle von n haben wir festgestellt, dass solche Sprünge in regelmässigen Abständen auftreten. Wir vermuten, dass dies ein Artefakt der Speicherallokation ist.

Wollen wir eine Laufzeitkomplexität des Ansatzes abschätzen, so müssen wir erst die untere Schranke des Wachstums finden. Gemäss Gleichung 3.8 muss zur Berechnung der Distanzmatrizen $D_{\mathcal{W}}$ mindestens jeder Trace einmal mit jedem anderen verglichen werden. Sei n die Anzahl der zu vergleichenden Traces und f unsere Ansatz, so wächst die Zeitkomplexität von f also mindestens quadratisch $f \in \Omega(n^2)$. Weiter stellen wir fest, dass für die beiden approximierte Funktionen $c < 2$ gilt. Dies ist ein starkes Indiz dafür, dass $f \in \mathcal{O}(n^2)$ und somit unser Ansatz praxistauglich ist.

5. Resultate und Diskussion

Dieses Kapitel dokumentiert und diskutiert die Resultate der im Kapitel 3 vorgestellten Schritte. Dabei betrachten wir erst die Datengrundlage. Danach führen wir einen Top-Down-Analysevorgang durch, ähnlich wie er in der Praxis ablaufen könnte. Top-Down bedeutet, dass wir erst die Hypothese prüfen und danach die Clusterung genauer betrachten.

5.1. Installationsgraphen erstellen

Diese Arbeit verwendet ausschliesslich Traces welche vorgängig vom SEL erstellt und klassifiziert wurden. Die Klassifizierung ergibt sich aus der Familie der analysierten Malware in den Traces. So stehen uns 209 Traces zur Verfügung. Aus verschiedenen Gründen haben wir sechs Traces vorgängig aussortiert. Weiter verwenden wir nur Traces die mit einer ähnlichen KAN-Parametrisierung aufgezeichnet wurden. Als letzten Schritt haben wir alle Traces entfernt, die einer Familie angehören für die uns weniger als vier Samples zur Verfügung stehen. So verwenden wir 125 Traces für die Clusteranalyse. Eine Komplette Auflistung aller aussortierten Traces gruppiert nach Begründung kann im Anhang B.2 gefunden werden.

Die Tabelle 5.1 bietet eine Übersicht der verwendeten Traces gruppiert nach Malware-Familien. Für jede Familie ist die Anzahl sowie relative Häufigkeit der enthaltenen Traces angegeben. Eine komplette Auflistung aller Traces und deren Klassifizierung kann im Anhang B.1 gefunden werden.

Malware-Familie	Anzahl Traces	Relative Häufigkeit
Fobber	14	0.112
Darkcomet	4	0.032
XtremeRAT	23	0.184
Cosmicduke	12	0.096
Dridex	25	0.2
Retefe	24	0.192
Zeus	23	0.184
	125	1.0

Tabelle 5.1.: Übersicht der Datengrundlage gruppiert nach Malware-Familie

5.1.1. Diskussion

In einer ersten Iteration haben wir alle 209 Traces als Datengrundlage verwendet. Mit dem Ergebnis, dass wir Cluster erhalten haben, die mit den unterschiedlichen KAN-Parametrisierungen korrelierten. Da dies keine sinnvollen Beobachtungen zuliess, haben wir in einer zweiten Iteration nur noch Traces verwendet, die gleich parametrisiert aufgezeichnet wurden. Dies hat zu wesentlich interessanteren Resultate geführt. Wir stellen also fest, dass eine gleichbleibende Parametrisierung der Messeinrichtung essentiell für unseren Ansatz ist.

Darkcomet ist mit einer relativen Häufigkeit von 0.032 stark untervertreten. Damit ist die Chance gross, dass gefundene Cluster die Darkcomet Familie schlecht repräsentieren.

5.2. Verifikation der Hypothese

In der Abbildung 5.1 ist die MDS-Projektion der Rand-Distanzen zwischen den Clusterungen angegeben. Da der Stress der Abbildung nur sehr gering ist, ist die Rand-Distanz dabei fast gleich der euklidischen Distanz zwischen den einzelnen Punkten. Die Farben der Punkte repräsentieren die Aussagen. Die Aussage -1 steht für die *Noise-Aussage* und bezeichnet Clusterungen die alleine eine Aussage machen. Die Zahlen neben den Punkte stehen für die jeweiligen Clusterungen, die in der Legende unten angegeben sind. Die Grösse der einzelnen Punkte ist proportional zu der Stabilität des Punktes in der Aussage.

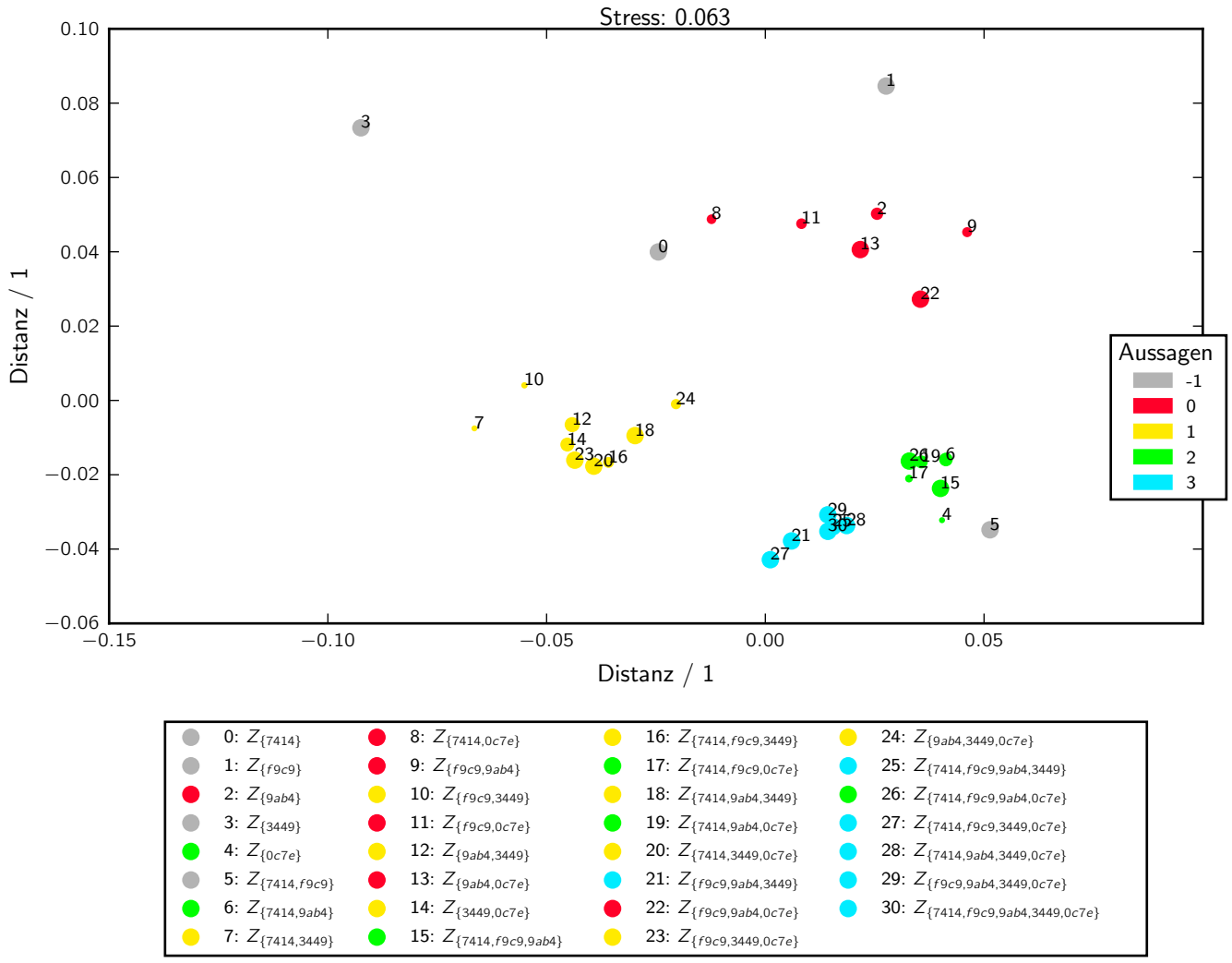


Abbildung 5.1.: MDS-Projektion der Distanzen sowie Clusterung der Aussagen

Wir beobachten vier Aussagen, die aus mehr als einer Workspace-Kombination bestehen und vier Noise-Aussagen. Dabei werden die Aussagen 1 und 3 von dem Workspace 3449 dominiert. Weiter dominieren die Workspaces 7414, f9c9 und 3449 drei Noise-Aussagen.

Zur Prüfung der Hypothese betrachten wir nun für alle nicht dominierten Aussagen die Rand-Indizes sowie den Nicht-Noise-Anteil. Dabei haben wir in der Abbildung 5.2 alle Kombinationen der 125 Traces mit den Malware-Familien verglichen. Da einige Traces mit Deckard nicht analysiert wurden, können wir in der Abbildung 5.3 nur die Kombinationen aus 118 Traces vergleichen. Auf der X-Achse der beiden Abbildungen ist dabei der Nicht-Noise-Anteil angegeben. Auf der Y-Achse der Rand-Index gemäss Vergleich. Die Zahlen neben den Punkte stehen für die jeweiligen Clusterungen, die in der Legende unten angegeben sind. Clusterungen die im Blau hinterlegte Bereich zu liegen kommen, sind dabei gemäss der Testdefinition in 3.3 nicht signifikant.

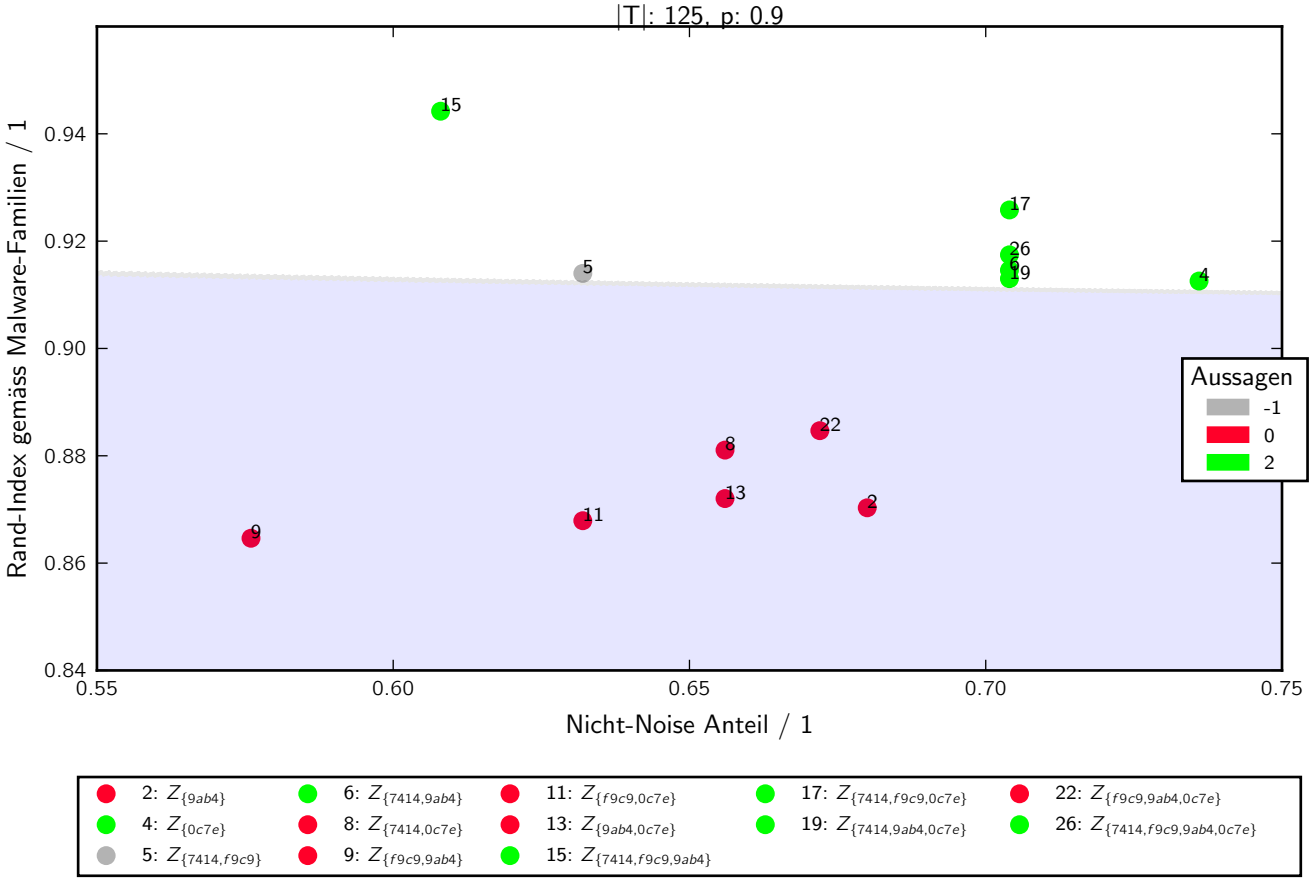


Abbildung 5.2.: Signifikanz gemäss Malware-Familien

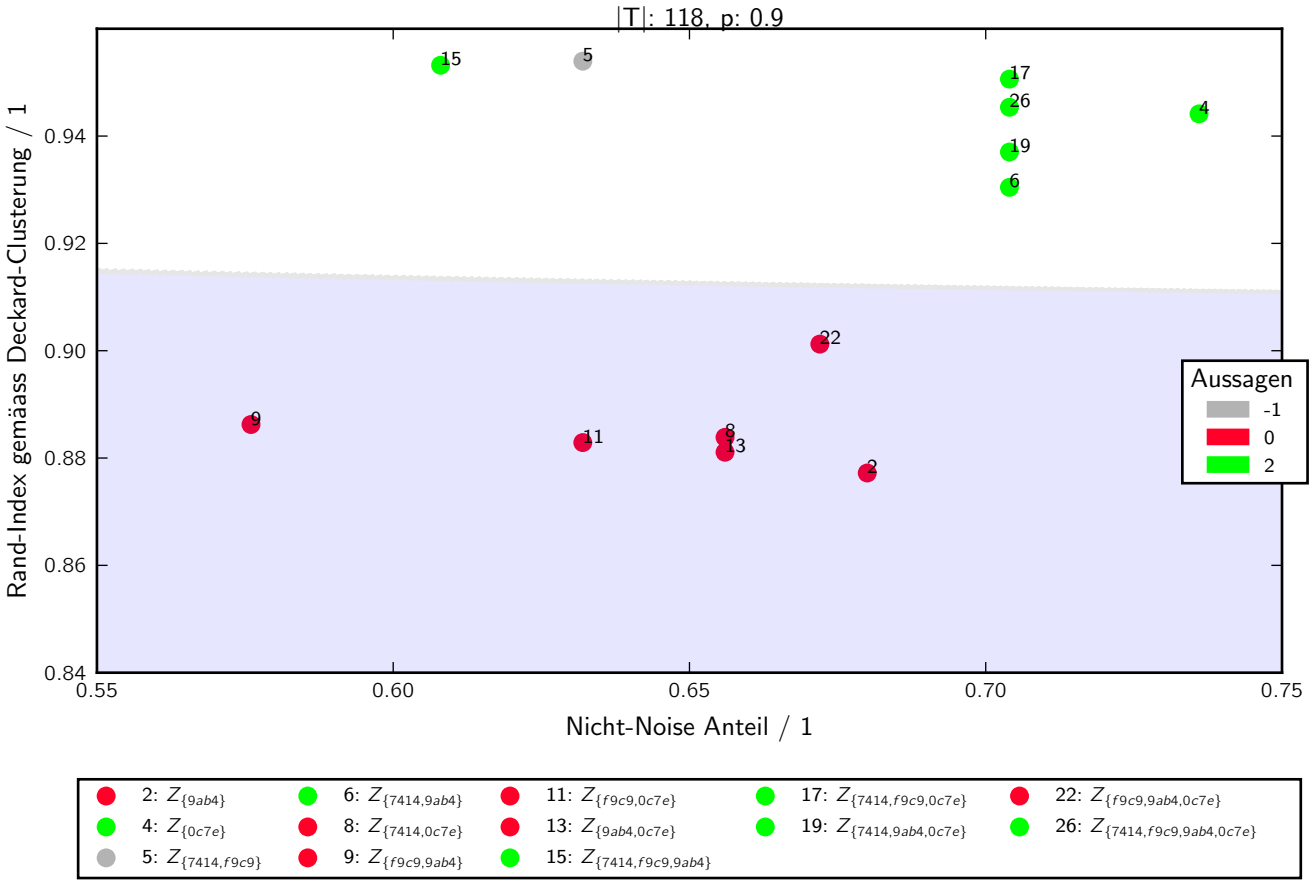


Abbildung 5.3.: Signifikanz gemäss Deckard-Clusterung

Wir beobachten, dass gemessen an den Malware-Familien, wie auch der Deckard-Clusterung, alle Workspace-Kombinationen der Aussage 2 sowie der Noise-Aussage 7414-f9c9 einen signifikant höheren Rand-Index als 0.90 aufweisen. Dies bedeutete, dass die Workspace-Kombinationen für mehr als 90% der Trace-Paare eine richtige Aussage machen und wir gemäss des definierten Tests die Nullhypothese H_0 zugunsten der Alternativhypothese H_1 verwerfen.

5.2.1. Diskussion

Einen Grund für die Dominaz von Workspace 3449 ist der Abbildung C.7 zu entnehmen. Wir sehen darin viele Cluster die Traces aus verschiedenen Malware-Familien enthalten. Es scheint, als ob die einzelnen Malware-Familien oft das gleiche Hook-Verhalten teilen. Dies ist höchst unwahrscheinlich und bestätigt vielmehr die Aussage des SEL, dass die Hookerkennung in KAN noch nicht ausgereift ist. So werden viele Funktionen fälschlicherweise als Hooks erkannt.

Da alle Clusterungen der Aussage 2 gemessen an den Malware-Familien einen signifikant besseren Rand-Index als 0.9 aufweisen, nehmen wir die Hypothese dieser Arbeit an. Die Clusterung $Z_{\{7414,f9c9,0c7e\}}$ sticht dabei besonders heraus da sie unter all den Clusterungen bestehend aus mehr als einem Workspace den grössten Rand-Index sowie den höchsten Nicht-Noise-Anteil aufweist.

5.3. Clusteranalyse

Im folgenden schauen wir uns nun die Clusterung $Z_{\{7414,f9c9,0c7e\}}$ an. Zuerst analysieren wir die Zuteilung der Trace auf die Cluster. Dabei gruppieren wir in Abbildung 5.4 die Traces nach Malware-Familie. Auf der X-Achse haben wir die Malware-Familien und auf der Y-Achse die durch den HDBSCAN erstellte Clusterung abgetragen. Die Werte in den einzelnen Zellen entsprechen der Anzahl Traces in einem Cluster. Die Farben repräsentieren die relative Häufigkeit einer Familie in einem Cluster.

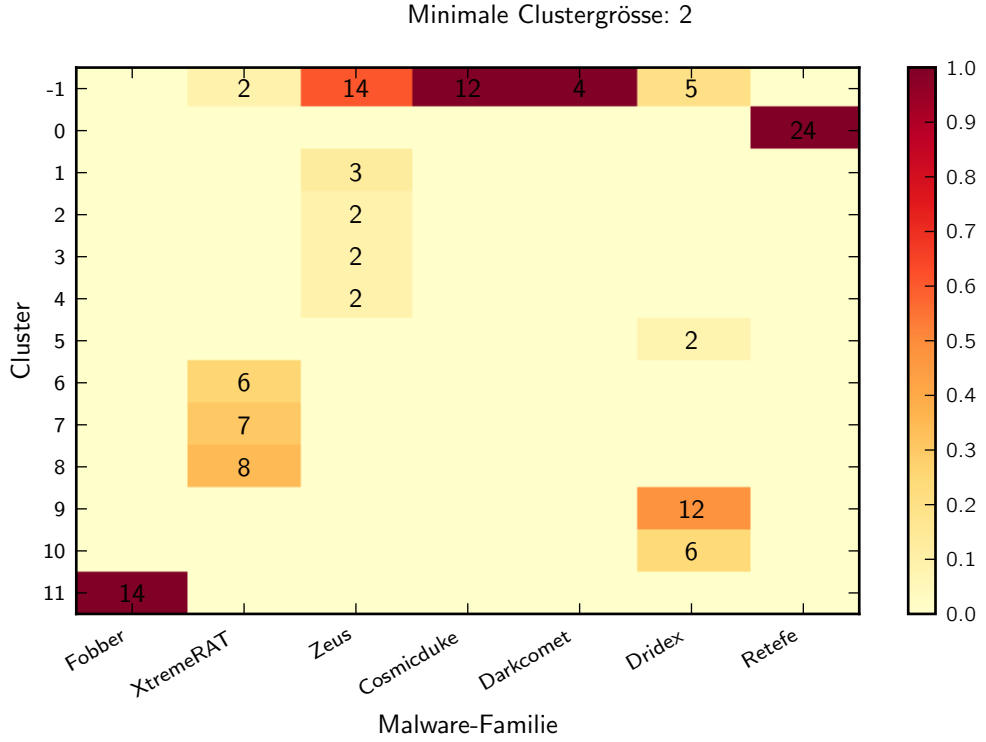


Abbildung 5.4.: Korrelation der Clusterung $Z_{\{7414,f9c9,0c7e\}}$ mit den Malware-Familien

Wir beobachten, dass Traces der Malware-Familie **Fobber** und **Retefe** ohne Noise in einem Cluster (0, 11) gruppiert sind. Weiter sehen wir keine Traces aus anderen Malware-Familien, die in eines dieser beiden Cluster fallen. Traces der Malware-Familie **XtremeRAT** sind in drei Cluster (6, 7, 8) gruppiert. Zwei Traces sind nicht geclustert worden. Alle drei Cluster beinhalten etwa gleich viele Traces. Es gibt keine Überschneidungen mit anderen Malware-Familien. Traces der Malware-Familie **Dridex** sind ebenfalls in drei Cluster (5, 9, 10) aufgeteilt. Zwei davon enthalten deutlich mehr Traces als das Dritte. Auch in diesen drei Cluster gibt es keine Überschneidungen mit anderen Malware-Familien. Fünf Traces sind Noise. Traces der Malware-Familie **Zeus** sind in vier kleine Cluster (1, 2, 3, 4) gruppiert. Die vier Cluster enthalten allerdings jeweils nur zwei oder drei Traces. In den vier Cluster gibt es aber keine Überschneidungen mit anderen Malware-Familien. 14 Traces und somit knapp 40% sind Noise. Die Malware-Familien **Cosmicduke** und **Darkcomet** konnten gar nicht geclustert werden. Sämtliche Traces sind Noise.

Das Resultat der Clusterbildung schauen wir uns genauer an. Dazu haben wir in Abbildung 5.5 für jeden Trace der einem Cluster zugeordnet wurde, die MDS-Projektion von $D_{\{7414,f9c9,0c7e\}}$ angegeben. Damit man die Malware-Familien auseinanderhalten kann, zeichnen wir für jede ein unterschiedliches Symbol. Die Farben sowie die Zahlen stehen für die Cluster. Die Grösse der Symbole ist proportional zu der Stabilität des Traces im Cluster.

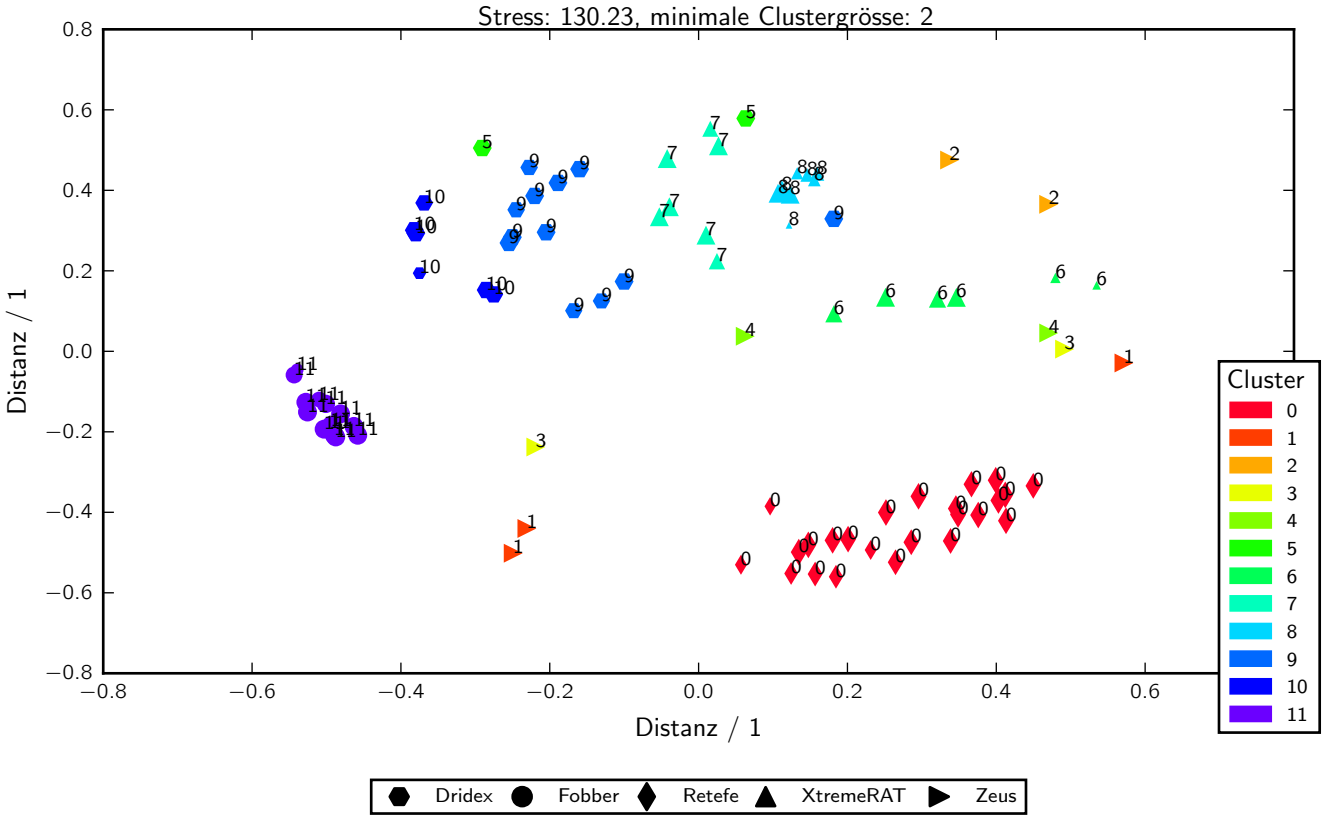


Abbildung 5.5.: MDS-Projektion von $D_{\{7414,f9c9,0c7e\}}$

Wir beobachten, dass in der Malware-Familie **Fobber** zwei Traces (48, 55) leicht weiter entfernt sind von den restlichen. In der Malware-Familie **Retefe** fallen ebenfalls zwei Traces (232, 246) durch eine leicht weitere Entfernung zu den restlichen Traces auf. Relativ zu den Distanzen der anderen Familien sind **Fobber** und **Retefe** weiter entfernt.

Weiter haben wir in Tabelle 5.2 die Silhouettenkoeffizienten pro Familie angegeben.

Malware-Familie	Silhouettenkoeffizient
Fobber	0.699
Retefe	0.577
XtremeRAT	0.268
Dridex	0.119
Darkcomet	0.061
Cosmicduke	0.022
Zeus	-0.069

Tabelle 5.2.: Silhouettenkoeffizient der Clusterung $Z_{\{7414,f9c9,0c7e\}}$ pro Malware-Familie

Wir beobachten, dass die Malware-Familien **Fobber** und **Retefe** einen hohen Silhouettenkoeffizient aufweisen. **XtremeRAT** und **Dridex** haben mittlere Werte. Die Silhouettenkoeffizienten von **Darkcomet**, **Cosmicduke** und **Zeus** sind nahe bei Null.

5.3.1. Diskussion

Im Folgenden diskutieren wir die gemachten Beobachtungen pro Malware-Familie.

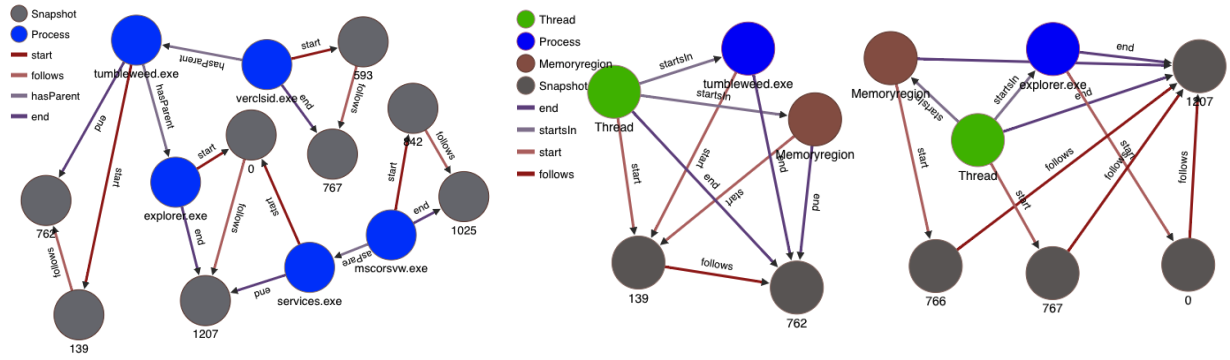
Fobber

Um zu verstehen warum Fobber so gut clustert, schauen wir in Tabelle 5.3 die einzelnen Clusterungen $Z_{\{7414,f9c9,0c7e\}}$, $Z_{\{7414\}}$, $Z_{\{f9c9\}}$ und $Z_{\{0c7e\}}$ genauer an.

Trace	Cluster			
	$Z_{\{7414,f9c9,0c7e\}}$	$Z_{\{7414\}}$	$Z_{\{f9c9\}}$	$Z_{\{0c7e\}}$
42	11	1	9	5
43	11	1	8	5
44	11	1	8	5
45	11	1	8	5
46	11	1	9	5
47	11	1	9	5
48	11	1	10	5
50	11	1	10	5
51	11	1	8	5
52	11	1	9	5
53	11	1	8	5
54	11	1	9	5
55	11	1	8	5
56	11	1	10	5

Tabelle 5.3.: Clusterzuordnung der Malware-Familie Fobber

Sowohl in Clusterung $Z_{\{7414\}}$ wie auch in Clusterung $Z_{\{0c7e\}}$ liegen die Fobber Traces jeweils in einem Cluster. Diese beiden Workspaces sind daher ausschlaggebend dafür, dass Fobber in einem Cluster gruppiert wird. In der Clusterung $Z_{\{7414\}}$ liegen gemäss Abbildung C.2 fast alle Fobber-Traces auf einem Punkt. Zwei Traces (48, 55) sind getrennt von den restlichen. Dies bedeutet, dass die Prozesshierarchie und Thread-Strukturen sehr charakteristisch sind für Fobber. Die Installationsgraphen der Abbildung 5.6 zeigen diese charakteristischen Installationsgraphen.



(a) Workspace 7414

(b) Workspace 0c7e

Abbildung 5.6.: Charakteristische Installationsgraphen für Fobber, Trace: 43

Zusätzlich sehen wir in Abbildung C.3, dass in Clusterung $Z_{\{f9c9\}}$ die Malware-Familie Fobber in drei Cluster aufgeteilt wurde. Daher vermuten wir drei Varianten von Fobber. Da sich der Workspace f9c9 auf Memoryregionen konzentriert, könnte dies nun anhand der Memoryregionen in den Installationsgraphen überprüft werden. In dieser Arbeit verzichten wir aber darauf.

Ein hoher Silhouettenkoeffizient in Tabelle 5.2 sagt uns, dass die Workspace-Kombination 7414-f9c9-0c7e geeignet ist um Malware der Familie Fobber von anderer zu unterscheiden.

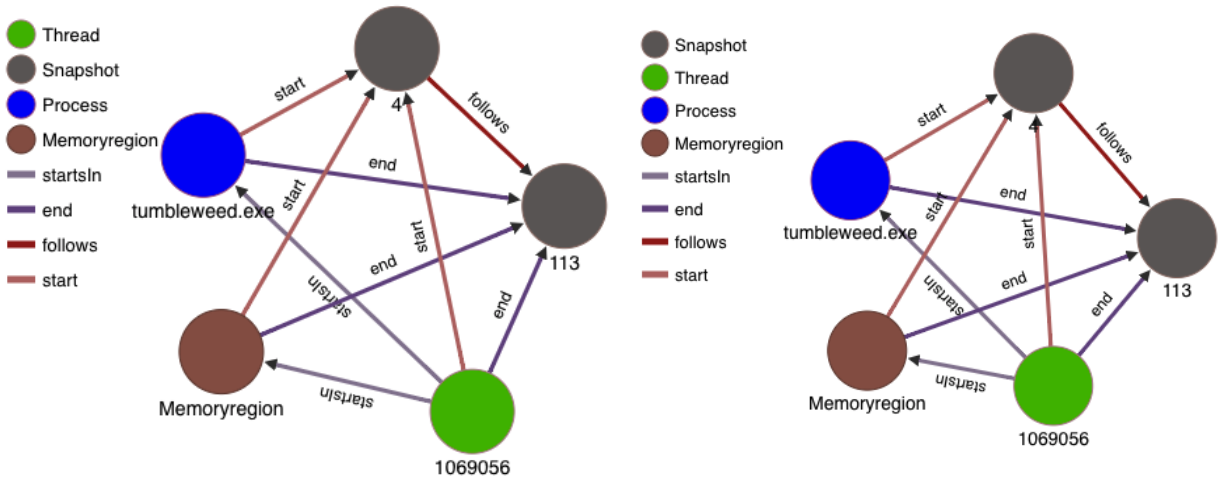
Refete

Wiederum schauen wir uns anhand der Tabelle 5.4 die Aufteilung der Traces in den vier Clusterungen an.

Trace	Cluster			
	$Z_{\{7414,f9c9,0c7e\}}$	$Z_{\{7414\}}$	$Z_{\{f9c9\}}$	$Z_{\{0c7e\}}$
223	0	8	0	1
225	0	8	-1	1
226	0	8	0	1
227	0	-1	3	1
228	0	10	2	1
229	0	9	4	1
230	0	-1	2	1
231	0	9	2	1
232	0	8	-1	1
233	0	10	-1	1
234	0	8	2	1
235	0	9	3	1
236	0	7	4	1
237	0	7	4	1
238	0	7	1	1
239	0	10	1	1
240	0	8	2	1
241	0	10	1	1
242	0	9	2	1
243	0	8	4	1
244	0	-1	3	1
245	0	8	4	1
246	0	8	-1	1
247	0	9	0	1

Tabelle 5.4.: Clusterzuordnung der Malware-Familie Retefe

Für die Clusterung in $Z_{\{7414,f9c9,0c7e\}}$ ist Workspace 0c7e ausschlaggebend. In Abbildung C.5 sehen wir, dass sämtliche Retefe Traces auf einem Punkt liegen. Die Installationsgraphen sind also identisch. Abbildung 5.7 zeigt dies exemplarisch für zwei unterschiedliche Traces.



(a) Trace 230

(b) Trace 231

Abbildung 5.7.: Charakteristische Installationsgraphen für Retefe, Workspace 0c7e

Die Traces (232, 246) sind Ausreisser, da sie im Cluster $Z_{\{f9c9\}}$ als Noise eingestuft wurden. Da sich der Workspace f9c9 mit Memoryregionen auseinandersetzt, vermuten wir, dass diese einige unterschiedliche solche Memoryregionen enthalten. Auf eine genaue Analyse verzichten wir in dieser Arbeit.

Ein hoher Silhouettenkoeffizient in Tabelle 5.2 sagt uns, dass die Workspace-Kombination 7414-f9c9-0c7e geeignet ist um Malware der Familie Retefe von anderer zu unterscheiden.

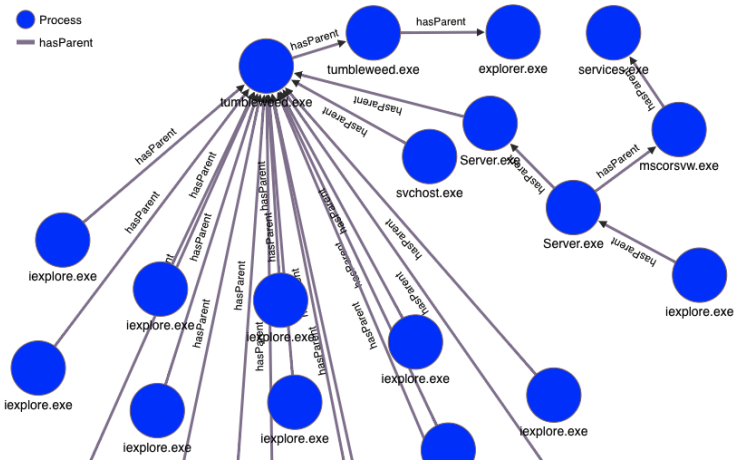
XtremeRAT

Da XtremeRAT in drei Cluster gruppiert wurde, wollen wir anhand der Tabelle 5.5 herausfinden welche Workspaces ausschlaggebend für die einzelnen Cluster sind.

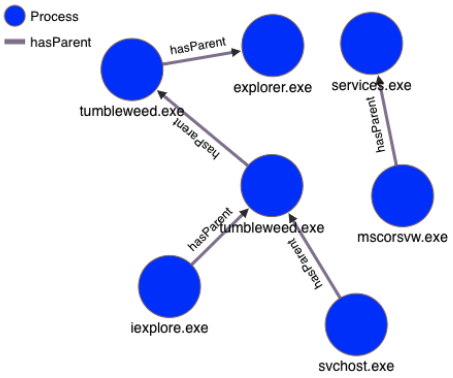
Trace	Cluster			
	$Z_{\{7414,f9c9,0c7e\}}$	$Z_{\{7414\}}$	$Z_{\{f9c9\}}$	$Z_{\{0c7e\}}$
112	6	-1	-1	3
114	6	11	14	3
119	6	11	14	3
121	6	11	14	3
126	6	11	14	3
132	6	11	-1	3
113	7	2	15	-1
117	7	2	-1	10
120	7	2	12	10
127	7	2	15	13
130	7	2	15	10
133	7	2	15	10
134	7	2	15	11
111	8	2	12	13
115	8	2	12	13
118	8	2	12	13
122	8	2	12	13
124	8	2	12	13
125	8	2	12	13
128	8	2	12	13
129	8	2	12	13

Tabelle 5.5.: Clusterzuordnung der Malware-Familie XtremeRAT

Wir sehen, dass das Cluster 6 der Clusterung $Z_{\{7414,f9c9,0c7e\}}$ von Cluster 7 und 8 durch den Workspace 7414 separiert wird. Um dies zu verstehen schauen wir uns exemplarisch in Abbildung 5.8 einen Installationsgraphen aus Cluster 11 und einen aus Cluster 2 an.



(a) Cluster 11, Trace 119



(b) Cluster 2, Trace 120

Abbildung 5.8.: Charakteristische Installationsgraphen für XtremeRAT, Workspace 7414

Wir sehen deutlich, dass Malware in Cluster 11 viel mehr iexplore.exe Prozesse startet. Dafür können wir uns drei mögliche Gründe vorstellen:

- Ein Aufzeichnungsfehler von KAN erkennt zu viele iexplore.exe Prozesse.
- Die installierte iexplore.exe Version in der KAN Umgebung ist nicht dieselbe.
- Wir sehen unterschiedliche XtremeRAT Versionen.

Cluster 8 in der Clusterung $Z_{\{7414,f9c9,0c7e\}}$ entsteht wegen Cluster 12 in $Z_{\{f9c9\}}$ und Cluster 13 in $Z_{\{0c7e\}}$.

Wir kommen zum Schluss, dass die Workspace-Kombination 7474-f9c9-0c7e geeignet ist, um XtremeRAT zu identifizieren. Ein zu Fobber und Retefe tiefere Silhouettenkoeffizient erklären wir durch die Aufteilung in drei Cluster.

Dridex

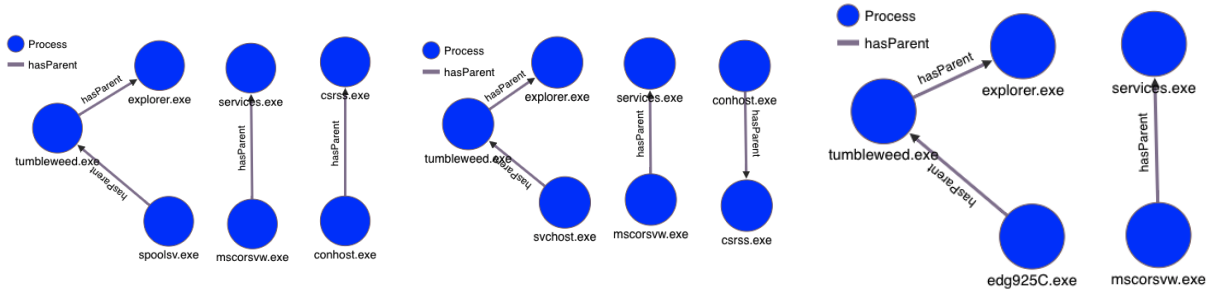
In Tabelle 5.6 schauen wir wiederum zuerst die Clusterzuordnung der drei Dridex-Cluster an.

Trace	Cluster			
	$Z_{\{7414,f9c9,0c7e\}}$	$Z_{\{7414\}}$	$Z_{\{f9c9\}}$	$Z_{\{0c7e\}}$
207	5	14	7	0
221	5	14	7	0
203	9	13	15	7
204	9	13	15	7
209	9	13	5	7
210	9	13	15	7
214	9	13	15	9
215	9	13	-1	-1
216	9	13	15	7
217	9	13	5	7
219	9	13	-1	7
222	9	13	15	-1
202	10	12	13	9
205	10	12	-1	8
206	10	12	13	8
213	10	12	13	8
218	10	12	-1	6

Tabelle 5.6.: Clusterzuordnung der Malware-Familie Dridex

Wir stellen fest, dass die Clusterung $Z_{\{7414\}}$ mit $Z_{\{7414,f9c9,0c7e\}}$ korreliert. Dabei separieren sich die Traces (207, 221) in allen Clusterungen deutlich von den anderen.

In der Abbildung 5.9 schauen wir uns exemplarisch je einen Installationsgraphen aus Workspace 7414 für die Cluster 12, 13 und 14 an.



(a) Cluster 12, Trace 206

(b) Cluster 13, Trace 204

(c) Cluster 14, Trace 207

Abbildung 5.9.: Charakteristische Installationsgraphen für Dridex, Workspace 7414

In Cluster 12 sehen wir, dass die Malware einen Prozess mit Namen spoolsrv.exe startet. In Cluster 13 heisst der gestartete Prozess svchost.exe. Im Gegensatz dazu scheint in Cluster 14 der von der Malware gestartete Prozess einen zufällig generierten Namen zu haben. Zudem fehlt in Cluster 14 die Substruktur bestehend aus conhost.exe und csrss.exe. Wir vermuten daher drei verschiedene Variationen von Dridex zu sehen.

Wir kommen zum Schluss, dass die Workspace-Kombination 7474-f9c9-0c7e geeignet ist, um Dridex zu identifizieren. Ein zu Fobber und Retefe tiefere Silhouettenkoeffizient erklären wir durch die Aufteilung in drei Cluster.

Zeus, Darkcomet, Cosmicduke

Für Darkcomet und Cosmicduke haben wir keine Cluster gefunden. Der Silhouettenkoeffizient bestätigt, dass keine Clusterung möglich ist.

Die beobachteten Zeus-Cluster sind zu klein und der Noise-Anteil zu hoch. Deshalb können wir keine Aussage darüber machen ob die Workspace-Kombination 7474-f9c9-0c7e geeignet ist, diese Familien zu erkennen. Gemäss Aussage SEL ist Zeus allerdings anhand einiger charakteristischen Hooks sehr gut erkennbar. Wie bereits in Abschnitt 5.2.1 festgestellt, ist die Hook-Erkennnung in KAN allerdings noch nicht geeignet für Installationsgraphen. Von einer Weiterentwicklung dieser Funktionalität erhoffen wir uns eine bessere Erkennungsrate der Malware-Familie Zeus.

5.3.2. Zusammenfassung

Folgende Erkenntnisse haben wir während dieser Arbeit gemacht:

- Um Installationsgraphen zu clustern, ist es wichtig, dass alle Traces mit gleicher KAN-Parametrisierung aufgezeichnet werden.
- Da die Hookerkennung in KAN noch viele Funktionen fälschlicherweise als Hooks erkennt, wirkt sich der Workspace 3449 dominant auf alle Aussagen aus.
- Die Workspace-Kombination 7474-f9c9-0c7e ist geeignet um Fobber, Retefe, XtremeRAT und Dridex zu erkennen.
- Die Workspace-Kombination 7474-f9c9-0c7e sieht vielversprechend aus um Zeus zu erkennen.
- Die Malware-Familien Darkcomet und Zeus müssen mit mehr Traces untersucht werden.
- Die Workspace-Kombination 7474-f9c9-0c7e kann Cosmicduke nicht erkennen.
- Wir haben Substrukturen in der Malware-Familie Fobber gefunden.
- Für die Malware-Familien XtremeRAT und Dridex haben wir unterschiedliche Cluster gefunden. Diese können Varianten der Malware-Familie repräsentieren oder aber auf Fehler beim Erzeugen der Traces hinweisen.

6. Schlussfolgerungen

Die in Kapitel 5 vorgestellten Resultate zeichnen ein nicht eindeutiges Bild. So können wir einige Malware-Familien sehr gut erkennen, andere gehen aber im Noise unter. Wir konnten nicht abschliessend klären warum dies so ist, vermuten aber, dass die gewählten Workspaces noch nicht ausreichend sind um Malware aller Art zu trennen. Den vorgestellten Ansatz im Kapitel 3 halten wir aber für vielversprechend, da er einfach zu verstehen ist und erweitert werden kann.

Dieses Kapitel gibt erst einen Ausblick auf mögliche zukünftige Arbeiten und beendet danach unsere Arbeit.

6.1. Ausblick

Beim erarbeiten des Kapitel 5 stellten wir fest, dass wir präzisere Aussagen hätten machen können wenn uns mehr Möglichkeiten für die Visualisierung der $\mathcal{A}_{l\ell}$ und der Schnittmengen zur Verfügung gestanden hätten. So sehen wir, dass für jede im Abschnitt 3.2.1 vorgestellte Äquivalenzrelation $\sim_{l\ell}$ eine eigenen Visualisierung gesucht werden muss. Die Graph-Visualisierung von OrientDB ist dabei nur für das exakte Vergleichen geeignet. Mögliche solche neuen Visualisierungen sind zum Beispiel Histogramme, Zeitreihenvisualisierungen oder Graphvisualisierungen für zwei oder mehrere Installationsgraphen inklusive Mapping.

Weiter verwenden wir in dieser Arbeit die Graph-Datenbank lediglich als Zwischenspeicher. Graph-Datenbanken sind aber dazu optimiert, Graphen über Abfragen schnell zu traversieren. Nicht dokumentierte Untersuchungen solcher Abfragen haben aber den subjektiven Eindruck vermittelt, dass die Abfragesprache in OrientDB nicht geeignet ist für diesen Schritt. Im Allgemeinen konnte uns OrientDB nicht überzeugen. Um in einer zukünftigen Lösung unabhängig vom gewählten Datenbankmanagementsystem zu bleiben, scheint Apache TinkerPop[10] eine vielversprechende Alternative zu sein.

sql2fluentflow erachten wir als eine geeignete Plattform für zukünftige Implementationsanstrengungen. Die bereits heute in einem Workspace gebündelten Schritte könnte man wie folgt erweitern:

1. Struktur und Attribute des Installationsgraphen definieren.
2. Vergleichsmethoden der Attribute selektieren.
3. Daten über SQL strukturieren und auslesen.
4. Daten mit FluentFlow in JavaScript nachbearbeiten.

Die fünf in dieser Arbeit eingesetzten Workspaces wurden ohne Expertenwissen über die aufgezeichneten Malware-Samples erstellt. Eine um die vorgestellten Punkte erweitertes sql2fluentflow würde aber einem Experten ein mächtiges Werkzeug in die Hand legen um auf Malware zugeschnittene Workspaces zu entwickeln. Weiter sehen wir, dass auf Basis von sql2fluentflow auch der im Abschnitt 5.2 durchgeführte Analyseprozess vereinheitlicht werden könnte.

Die Hookerkennung und System-Call Aufzeichnung in KAN sind zurzeit unter aktiver Entwicklung. In naher Zukunft hoffen wir, dass damit ein Installationsgraph, ähnlich zu den in dem Kapitel 1 vorgestellten Ansätzen, generiert und verglichen werden kann.

Um unsere Resultate zu verifizieren/falsifizieren schlagen wir weiter eine Studie vor, welche die Hypothese mit gutartiger Software untersucht. Da die Ziele von gutartiger Software bekannt sind, fällt es leicht eine passende Klassifikation der Samples zu finden. Erfahrungsgemäss ist gutartige Software besser dokumentiert als Malware. Aus diesem Grund erhoffen wir uns, dass Resultate einfacher zu interpretieren wären. Als letzten Schritte einer zukünftigen Untersuchung würden wir gutartige Software zusammen mit Malware-Samples clustern. Davon erhoffen wir uns weitere Rückschlüsse auf die Ziele von Malware.

[10] `https://tinkerpop.apache.org`

6.2. Schluss

In dieser Arbeit haben wir einen Ansatz präsentiert der verhaltensbasiert Traces clustert. Workspaces wurden dabei als flexibles Werkzeug vorgestellt um schnell neue Sichtweisen auf Traces zu implementieren. Der vorgestellte Test erlaubt eine Beurteilung der Qualität neuer Workspaces. Anhand von zwei vorgegeben Clusterungen konnten wir mit diesem Test signifikant zeigen, dass unser Ansatz fähig ist valide Strukturen in Daten zu finden. Mit Messungen unserer Referenzimplementation haben wir eine höchstens quadratisch wachsende Laufzeitkomplexität des Ansatzes nachgewiesen.

Um neues Verhalten zu verstehen, müssen in Zukunft nur noch wenige Malware-Samples genauer untersucht werden. Diese Fokussierung bedeutete, dass der Analyseprozess besser skaliert. Damit haben wir einen Beitrag geleistet zur Bewältigung der stetig wachsenden Zahl neuer Malware-Varianten.

Literaturverzeichnis

[1] C. M. Antunes and A. L. Oliveira, "Temporal data mining: An overview," in *KDD workshop on temporal data mining*, vol. 1, 2001, p. 13.

[2] M. Bailey, J. Oberheide, J. Andersen, Z. M. Mao, F. Jahanian, and J. Nazario, "Automated classification and analysis of internet malware," in *International Workshop on Recent Advances in Intrusion Detection*. Springer, 2007, pp. 178–197.

[3] U. Bayer, P. M. Comparetti, C. Hlauschek, C. Kruegel, and E. Kirda, "Scalable, behavior-based malware clustering." in *NDSS*, vol. 9. Citeseer, 2009, pp. 8–11.

[4] H. Binsalleeh, T. Ormerod, A. Boukhtouta, P. Sinha, A. Youssef, M. Debbabi, and L. Wang, "On the analysis of the zeus botnet crimeware toolkit," in *Privacy Security and Trust (PST), 2010 Eighth Annual International Conference on*. IEEE, 2010, pp. 31–38.

[5] H. Bunke and K. Shearer, "A graph distance metric based on the maximal common subgraph," *Pattern recognition letters*, vol. 19, no. 3, pp. 255–259, 1998.

[6] R. J. Campello, D. Moulavi, A. Zimek, and J. Sander, "Hierarchical density estimates for data clustering, visualization, and outlier detection," *ACM Transactions on Knowledge Discovery from Data (TKDD)*, vol. 10, no. 1, p. 5, 2015.

[7] P.-A. Champin and C. Solnon, "Measuring the similarity of labeled graphs," in *International Conference on Case-Based Reasoning*. Springer, 2003, pp. 80–95.

[8] P. P.-S. Chen, "The entity-relationship model toward a unified view of data," *ACM Transactions on Database Systems (TODS)*, vol. 1, no. 1, pp. 9–36, 1976.

[9] S. Droz, S. Greminger, F. Herberg, D. Stirnimann, M. Fuchs, M. Hausding, and Y. Bovard, "Retefe bankentrojaner," switch. [Online]. Available: https://securityblog.switch.ch/2014/07/22/retefe-bankentrojaner/

[10] A. A. E. Elhadi, M. A. Maarof, and A. H. Osman, "Malware detection based on hybrid signature behaviour application programming interface call graph," *American Journal of Applied Sciences*, vol. 9, no. 3, p. 283, 2012.

[11] M. Ester, H.-P. Kriegel, J. Sander, X. Xu *et al.*, "A density-based algorithm for discovering clusters in large spatial databases with noise." in *Kdd*, vol. 96, no. 34, 1996, pp. 226–231.

[12] F-Secure, "Cosmicduke: Cosmu with a twist of miniduke," F-Secure. [Online]. Available: https://www.f-secure.com/documents/996508/1030745/cosmicduke_whitepaper.pdf

[13] GovCERT.ch, "Fobber analysis," GovCERT.ch, Tech. Rep., 09 2015.

[14] M. Halkidi, Y. Batistakis, and M. Vazirgiannis, "On clustering validation techniques," *Journal of intelligent information systems*, vol. 17, no. 2-3, pp. 107–145, 2001.

[15] D. Hidović and M. Pelillo, "Metrics for attributed graphs based on the maximal similarity common subgraph," *International Journal of Pattern Recognition and Artificial Intelligence*, vol. 18, no. 03, pp. 299–313, 2004.

[16] L. Kaufman and P. J. Rousseeuw, *Finding groups in data: an introduction to cluster analysis*. John Wiley & Sons, 2009, vol. 344.

[17] C. Kolbitsch, P. M. Comparetti, C. Kruegel, E. Kirda, X.-y. Zhou, and X. Wang, "Effective and efficient malware detection at the end host." in *USENIX security symposium*, 2009, pp. 351–366.

[18] T. Lang, "Fluentflow," Github. [Online]. Available: https://github.com/t-moe/FluentFlow

[19] W. R. Marczak, J. Scott-Railton, M. Marquis-Boire, and V. Paxson, "When governments hack opponents: A look at actors and technology," in *23rd USENIX Security Symposium (USENIX Security 14)*, 2014, pp.

511–525.

[20] Massachusetts Institute of Technology - MIT, "Lsh algorithm and implementation." [Online]. Available: http://www.mit.edu/~andoni/LSH/

[21] Microsoft, "Microsoft academic." [Online]. Available: http://academic.research.microsoft.com/

[22] ——, "Thread stack size." [Online]. Available: https://msdn.microsoft.com/en-us/library/windows/desktop/ms686774(v=vs.85).aspx

[23] A. Moore, "Research note-banking malware explained: The case of dridex," Tech. Rep., 2016.

[24] Y. Park, D. Reeves, V. Mulukutla, and B. Sundaravel, "Fast malware classification by automated behavioral graph matching," in *Proceedings of the Sixth Annual Workshop on Cyber Security and Information Intelligence Research*. ACM, 2010, p. 45.

[25] Y. Park, D. S. Reeves, and M. Stamp, "Deriving common malware behavior through graph clustering," *Computers & Security*, vol. 39, pp. 419–430, 2013.

[26] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[27] M. Pietrek, "Peering inside the pe: A tour of the win32 portable executable file format." [Online]. Available: https://msdn.microsoft.com/en-us/library/ms809762.aspx

[28] C. Quates, "sdhash," Github. [Online]. Available: https://github.com/sdhash/sdhash

[29] W. M. Rand, "Objective criteria for the evaluation of clustering methods," *Journal of the American Statistical association*, vol. 66, no. 336, pp. 846–850, 1971.

[30] P. J. Rousseeuw, "Silhouettes: a graphical aid to the interpretation and validation of cluster analysis," *Journal of computational and applied mathematics*, vol. 20, pp. 53–65, 1987.

[31] V. Roussev, "Data fingerprinting with similarity digests," in *IFIP International Conference on Digital Forensics*. Springer, 2010, pp. 207–226.

[32] V. Roussev and C. Quates, "sdhash - quick start - result interpretation." [Online]. Available: http://roussev.net/sdhash/tutorial/03-quick.html#result-interpretation

[33] A. Sanfeliu and K.-S. Fu, "A distance measure between attributed relational graphs for pattern recognition," *IEEE transactions on systems, man, and cybernetics*, no. 3, pp. 353–362, 1983.

[34] A. Srivastava, A. Lanzi, J. Giffin, and D. Balzarotti, "Operating system interface obfuscation and the revealing of hidden operations," in *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*. Springer, 2011, pp. 214–233.

[35] N. Villeneuve and J. Bennett T., "Xtremerat: Nuisance or threat?" FireEye. [Online]. Available: https://www.fireeye.com/blog/threat-research/2014/02/xtremerat-nuisance-or-threat.html

[36] J. Wagner, "Automated discovery and analysis of code similarities in malware," Master's thesis, Engineering and Information Technology, Bern University of Applied Sciences, 2016.

[37] C. Willems, T. Holz, and F. Freiling, "Toward automated dynamic malware analysis using cwsandbox," *IEEE Security and Privacy*, vol. 5, no. 2, pp. 32–39, 2007.

[38] P. Wood, B. Nahorney, K. Chandrasekar, S. Wallace, and K. Haley, "Symantec global internet security threat report," *White Paper, Symantec Enterprise Security*, vol. 21, 2016.

Abbildungsverzeichnis

Tabellenverzeichnis

Algorithmenverzeichnis

A. Workspaces

Als Dokumentationsmittel der aus den Workspaces generierten Installationsgraphen verwenden wir Chen notierte[11] Entity-Relationship-Modelle.

A.1. Process hasParent Process

Dieser Workspace extrahiert alle Prozesse welche in Vater-Kind Beziehung stehen. Eine solche Beziehung besteht wenn ein Prozess von einem anderen gestartet wurde. Für jede so extrahierten Entität wird ein Prozess-Knoten angelegt. Die Vater-Kind-Beziehung zwischen zwei Prozesse wird mit einer hasParent-Kante notiert.

Für Start sowie Ende jedes extrahierten Prozesses wird ein Snapshot-Knoten angelegt und mit entsprechender start- sowie end-Kante versehen. Snapshot-Knoten, die so in Relation stehen werden, durch eine follows-Kante verbunden.

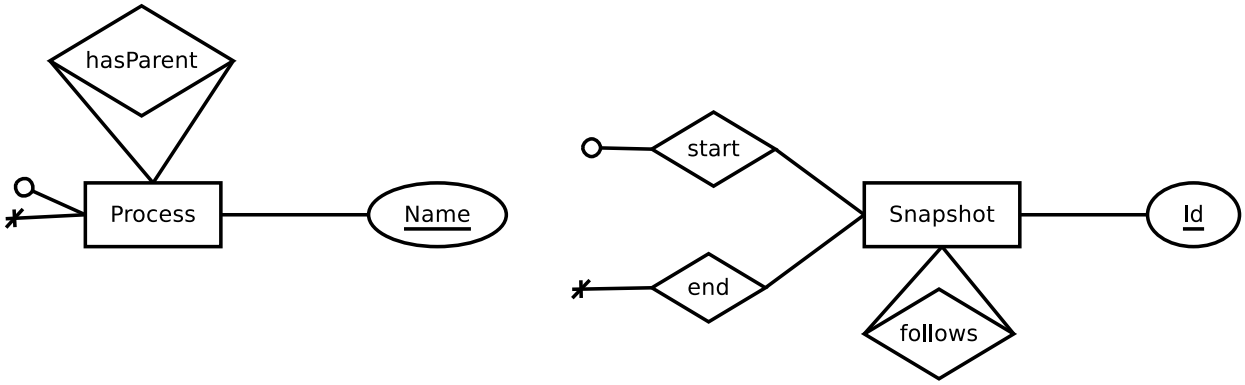


Abbildung A.1.: Workspace: 741493df-6d05-46dd-8892-29824db41fc9, kurz: 7414

A.2. Process hasCodeIn Memoryregion

Dieser Workspace extrahiert alle Prozesse und dazugehörige Speicherbereiche welche möglicherweise Programmcode enthalten. Programmcode wird von KAN über verschiedene Heuristiken erkannt. Die nachfolgende Liste führt einige Beobachtungen auf, die zu solchen Heuristiken führen:

- *Portable Executable*-Header, kurz: PE-Header, findet man im Speicher wenn ganze ausführbare Dateien geladen werden. PE-Header können über eine für sie typischen magische Zahl sowie weitere Strukturen erkannt werden. [27]
- Ausführbarer Code verwendet oft Konstanten, wie zum Beispiel Dateinamen oder Format-Strings, die einfach erkannt werden können.
- Ausführbarer Code hat oft einen spezifischen Fussabdruck bezüglich der Shannon-Entropie.

Alle möglichen Code-Speicherbereiche und deren Prozesse werden als Knoten angelegt. Eine isCodeIn-Kante setzt diese in Relation.

[11] Gemäss [8]

Für Start sowie Ende jedes extrahierten Prozesses, sowie Allokation und Freigabe jedes Speicherbereiches wird ein Snapshot-Knoten angelegt und mit entsprechender start- sowie end-Kante versehen. Snapshot-Knoten, die so in Relation stehen, werden durch eine follows-Kante verbunden.

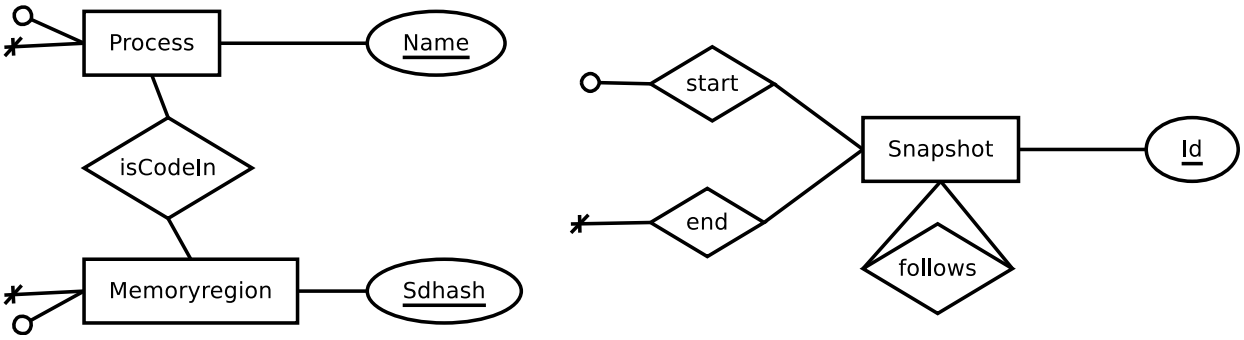


Abbildung A.2.: Workspace: f9c98d2b-899a-4204-849c-4ed284eb7a9e, kurz: f9c9

A.3. Pattern foundIn Memoryregion, Process

Dieser Workspace extrahiert alle Prozesse und dazugehörige Speicherbereiche, die mindestens ein Muster enthalten das ähnlich zu einem Muster in einem anderen Speicherbereich ist. Da gleiche Muster in unterschiedlichen Memoryregionen anders abgelegt werden können, macht der exakte Vergleich von Memoryregionen nur begrenzt Sinn. Aus diesem Grund werden Ähnlichkeiten zwischen Mustern von KAN über Locality-sensitive hashing ermittelt. Solche Hashes haben zum Ziel, das Nachbarschaftsproblem in hochdimensionalen Daten zu lösen [20]. Die verwendete Implementation ist sdhash[12]. Es werden nur Muster extrahiert mit einer starken Übereinstimmung[13]. Entitäten die so in Relation stehen, werden mit einer foundIn-Kante verbunden.

Für Start sowie Ende jedes extrahierten Prozesses, Allokation und Freigabe jedes Speicherbereiches, sowie erstes und letztes Auftreten jedes Musters wird ein Snapshot-Knoten angelegt und mit entsprechender start- sowie end-Kante versehen. Snapshot-Knoten, die so in Relation stehen, werden durch eine follows-Kante verbunden.

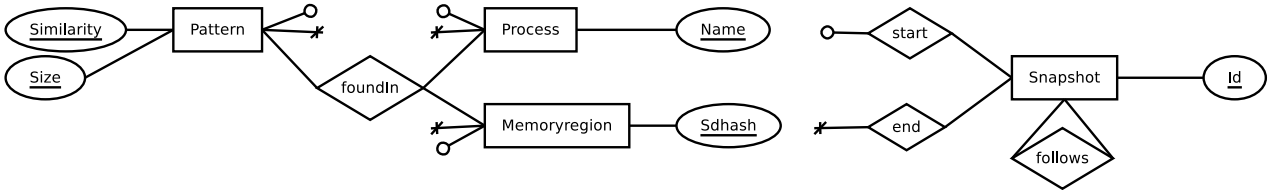


Abbildung A.3.: Workspace: 9ab41dc4-2f1e-4216-9ec4-0efc18085310, kurz: 9ab4

A.4. Thread startsIn Memoryregion, Process

Dieser Workspace extrahiert alle Prozesse und dazugehörige Speicherbereiche in denen mindestens ein Thread gestartet wurde. Diese Information extrahiert KAN aus internen Strukturen des Betriebssystems. Threads und Prozesse die so in Beziehung stehen werden mit einer startsIn-Kante verbunden.

Für Start sowie Ende jedes extrahierten Prozesses, Allokation und Freigabe jedes Speicherbereiches, sowie Start und Ende jedes Threads wird ein Snapshot-Knoten angelegt und mit entsprechender start- sowie end-Kante versehen. Snapshot-Knoten, die so in Relation stehen, werden durch eine follows-Kante verbunden.

[12]Weitere Informationen zu sdhash kann in [31] und [28] gefunden werden.
[13]Dies heisst $sdhash(x, y) \geq 21$ [32]

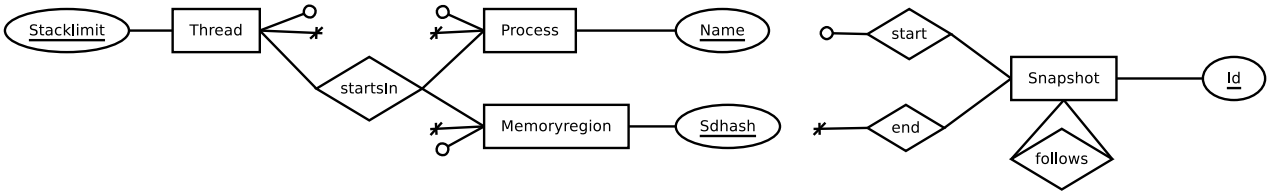


Abbildung A.4.: Workspace: 0c7e8d78-1ec6-4837-bf93-67ca18b8c889, kurz: 0c7e

A.5. Hook detectedIn Module, Process

Dieser Workspace extrahiert alle Prozesse und dazugehörige Speicherbereiche die mindestens ein *Hook* enthalten. Mit Hooks werden Techniken bezeichnet die den normalen Programmablauf ändern. Hooks werden von Schadsoftware missbraucht um Daten abzufangen. Für die Erkennung von Hooks überwacht KAN charakteristische Datenstrukturren aller als gutartig gekennzeichnete Module. Bei einer Veränderung werden alle betroffenen Funktionen, sowie der Datenstrukturtyp extrahiert. Hooks und Module, die so in Beziehung stehen, werden mit einer detectedIn-Kante verbunden.

Für Start sowie Ende jedes extrahierten Prozesses, Laden und Entladen jedes extrahierten Moduls sowie Allokation und Freigabe des Speicherbereiches in dem sich der Hook befindet, wird ein Snapshot-Knoten angelegt und mit entsprechender start- sowie end-Kante versehen. Snapshot-Knoten, die so in Relation stehen, werden durch eine follows-Kante verbunden.

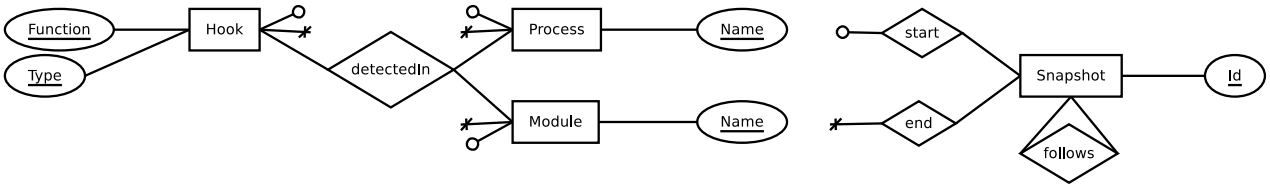


Abbildung A.5.: Workspace: 34494b4f-8546-4361-9cb0-aa003e1bce64, kurz:3449

B. Traces

B.1. Verwendete Traces

ID	UUID	Sample Hash	Kommentar
42	bc977679-d484-754d-a211-7f56d955ae50	d9bac445de8537c483bc60dd1ebdba00c1a2649c66197aa554a2d3b5e19468c8	fobber, 2min, bl=explorer
43	6fb86414-8f22-0447-a5c1-e995ec315458	ab6e4dba7dcd24f139c33eef87e7e3c025a5be56511a1de5a37f1235dd749687	fobber, 2min, bl=explorer
44	20d0ea7e-b20b-f44a-9b5c-09fa1be3e0a3	33776404e6463e2093f07e2a8daf881d86b4a052d26102b29e7c28bb3fa20b64	fobber, 2min, bl=explorer
45	87174d67-82ad-084e-9e3f-b57cd09886bc	191586f91b8057eed118f9e0cc66ef1400534fe29bf4ab5e60d4754a706ed7fa	fobber, 2min, bl=explorer
46	af65013e-faa4-e649-8d3c-aa0bbbf78c3b	8212ebe1311b41d539a76cfb27d7e73137974bf656df3d7a6e45893aaffb65d1	fobber, 2min, bl=explorer
47	31c9f900-863c-6549-af2b-3fb533f802fd	8135ad8a608c9d841cea14fe0151978c5d2b8adb7fa006f31e579df22e342cbd	fobber, 2min, bl=explorer
48	4843c494-15f0-6c4b-ba65-cd866424ab3c	6145a3f86acff5c37c00fc59e5daaadd47e7d4b179f77cf7ddf5354d5203fbd2	fobber, 2min, bl=explorer
50	afd723af-5011-924d-8350-fb10ae5a1987	825f80a8dc2bd203c0087748d8cc5d44ffec2289f75e351980f95d0bf5249db6	fobber, 2min, bl=explorer
51	d34182b4-b32e-e34f-a1f7-d7c5d26b47ce	64a20918e9cce528bf2d8c7ea49337176893b404b724d069f989bc166c32a27b	fobber, 2min, bl=explorer
52	58b3e79d-ec9d-4f4b-b7e0-bcaf2c65ba3a	38ec54d51e6844f226987c4bca0f2269b25156aab4622b0d31c80ace10f23dc2	fobber, 2min, bl=explorer
53	96dd70bc-c58a-5541-862a-aac1f5260ec9	20f2d01f9fd163cc8ffb8ed7e4ce153c2f5cd063ea1cea67f77707369fbb6503	fobber, 2min, bl=explorer
54	2e621058-963c-4644-a812-2617441921e5	8ebc2335b4fbc5f56ce0cd8b4e7b4df5cc9dc9ab9eb7e3c466d330b84fb8b289	fobber, 2min, bl=explorer
55	c48375b0-1df5-ce40-a376-47b6731e9b46	5d0d798cf8ef16cc57ff57999f70f25d02c7c69a1dc67114dd5766c990771469	fobber, 2min, bl=explorer
56	4dfa4e27-c621-f64b-8230-dd8f3de81f73	1d3a00bb72ebed2fee1dd7f6ea3ecd6f232f25bcf80b1895b079567fc22340ec	fobber, 2min, bl=explorer
107	a857ae07-7960-c24e-9cfc-d15fb805a242	b78e50b73389197d8ffe4a2a50e95f1d472cb2ef5c5a7d4eca2ebb9dbf92e80e	darkcomet, 3min, bl=explorer, slim
108	a151bd6d-72ae-9b48-b6b4-655cf29837f9	ad7302bc95da3e72d9206c733f8e6229af851810bc3a631572c3e39e1266a8a9	darkcomet, 3min, bl=explorer, slim
109	ccbfbe09-6618-d142-86bf-099c9178faaf	74482cc037b0b169aff6a66e955347fc68e2cb3a3a4218827ab8d92cbddb3c10	darkcomet, 3min, bl=explorer, slim
110	016af640-44be-5a43-ad48-a81171c2eb4f	43a600b42d48b683b633fa5318c99d955b2edbaaee6fc064ed904b6f8db26a5c	darkcomet, 3min, bl=explorer, slim
111	6ce3583d-3257-684d-8921-869907f2bf83	9adde38963dc352edc555f2adb4cee8e7e979a5954c406d9c52015255da1e7d8	xtreme_120_lw
112	b8e25ec2-9c14-634f-8ba6-362a0270fa9c	7fd4206cfcae7bfbd6ce11c651e49b6661e782c15600456f019c66df121d554a	xtreme_120_lw
113	b80b64ad-cb01-c243-8538-f7fb6c1cabba	6a9cdfda65d52463843066d13f3d9b98db3fe9613c3fa7c8ed53f8145eebe4d6	xtreme_120_lw
114	f90da66c-949a-ba48-8caf-b55baa548c0f	5f883cadc4b7a0bd08c95f24725c1c28ea2d2e4596f2b3c6f5512762f0996319	xtreme_120_lw
115	e07089f7-87b0-3947-891d-e1bef6f75509	3dbbc08b1d019a38ad6ce9bb33518771baba08610371e50fb221dc6a73b1ec2e	xtreme_120_lw
117	beac8c20-02a5-dd4b-9f94-21f4dbd48d92	1c8de1cd96c3053d86c176b8bb2abfc13eb4691615943d8bd2b0d1e9204cf754	xtreme_120_lw
118	136e5678-944f-b14c-bc29-f5b318f4574a	03bbfcc623b38a9080d6073832a58f38202ae70d462ce504deabddc438071fc5	xtreme_120_lw
119	68328ab0-7245-f54a-a7eb-d205f5696890	041025af7e54777e8c1ad062930d212873cc9f85166de1e17a5b19f1be8a251f	xtreme_120_lw
120	4831983b-ee0d-6a46-9f41-1738e58b992a	a558ede6fbe3d60f88d57373306224c44f4304d1ed5be8d05294d4458f2c4974	xtreme_120_lw
121	710622eb-0df0-e446-9ffb-c6edc15c9072	041025af7e54777e8c1ad062930d212873cc9f85166de1e17a5b19f1be8a251f	xtreme_120_lw
122	63e81db1-5352-f749-b564-580450eaf528	72b79f302f8bc4c47b277210ef53c26bc1d6f6ff6e34f997d97e6215e14c859d	xtreme_120_lw
123	1f55cda3-bae2-ab48-8da7-a7dd5f2e8e98	9ff0bfea87849e6250dacea547fb8b3f0350e130fcef03987e1abffffe50584e	xtreme_120_lw

124	e0fe06b7-cdd6-f140-9c0f-44c88ae5e8c4	ec8b33b7972535e2b9a688b734f74b3907c6211221e9e39f28c9e23cb3d96334	xtreme_120_lw
125	ae3825bf-b1c1-b14d-a529-3b62c0570ed9	eb2bbe13c8f115260b3afa3756dbe0381f4ca7802611035c8676afb7a37ebc39	xtreme_120_lw
126	0d27ea30-dfde-774c-8c46-8ce0c587be6c	e46cbed7747902cbf1bc0f26dbc847549d4c626facea329f3e165117ff28ed7e	xtreme_120_lw
127	15451ee1-4856-4041-b2df-8613def8acf8	bce68fe35a87c36ae1c7a6699987bef4af3288a3676fc4f82f25faadd1efaf83	xtreme_120_lw
128	2d669195-55cd-6146-8120-8000bac5bfe7	b37bb9f0d0e3a1de397a2aa35d62b3ec8ffb35e51185ff73394729ff993eb9a1	xtreme_120_lw
129	39b5cb5a-b13b-b94f-986c-60605e096385	adb8456c69c76756ab9ee94a49c6c56ed980091d51a19e56a6dae8fc98472587	xtreme_120_lw
130	ab204dc3-81e1-df42-8e18-2203bd9df850	9892dd1005d21cf3f1d9b0e64936bf577c2a12884625ce22ffc569824e4c51f2	xtreme_120_lw
131	88e95e50-99f2-284c-9d09-d3e8ffae5ebf	9298ac79d15e8246d430e0ab689525f85136fbdbbb722b4af3fb31fa252ef356	xtreme_120_lw
132	ae873c52-5837-c44e-a1ca-d2710eb09064	952f7ad678d8803d1ce527b653dc92d8fbd14fab2fdf5fd3abd391205c10bac0	xtreme_120_lw
133	382b2bbd-d587-3e4c-9adf-03e845b2f172	75ff6bca3f2ed2a9d0762a254a9838caa4cbebda73d08d5b7009dc70a78e2786	xtreme_120_lw
134	4caa270d-b405-764b-8f99-9359b2e88a88	c05858dd1b14919f9ea42c2f68f44e3ca0fdedef67963c684401769458aa5a1c	xtreme_120_lw
156	7c706f89-71bf-8b40-93c2-a1a3132a3529	f21794d0b0938643e2aabe9f2ed762528e631a2ebda76020d0b59ce91fb51e41	cosmicduke, 15min, bl=explorer
157	da0d0213-629d-da49-8de7-93d1f77d6a5c	f6c62f9f846b3d100d60b1f2ae57a71c91dd8dc215dce652e2c85dff60c0197f	cosmicduke, 15min, bl=explorer
160	2385a864-5197-6e45-8077-223ff192b5c6	05637ef950feaeb0944d9fccca38eeff38e366c24a137ef08c9f1442aeb6afb7	cosmicduke, 15min, bl=explorer
161	90f58811-2626-5d4c-82ee-e9d509ba0eec	2146da9bc0e27d7eb10983b7dd89f250fa0015ce284dde8f0bb6a79626d34a2a	cosmicduke, 15min, bl=explorer
162	e3f16a9b-5ae5-6545-8784-6a6f8f9e70cd	187b1cc7264c04c3158f835546cad0be74e6411bb50cb8899179a71018f0b4b9	cosmicduke, 15min, bl=explorer
163	9296b1d9-5634-9544-a238-1c3b764af95d	4fc0bbb90aeecd3229aa932437273ba59f887a6eac569b56693602b957e205e2	cosmicduke, 15min, bl=explorer
164	feb6f9d4-63ce-c549-8834-05dbc8d19bde	3c5d2fcacafc21d9f43c595ddf03bec801ccb958b8641018612c21bc741800d0	cosmicduke, 15min, bl=explorer
165	11eed4ec-864f-744c-9d34-2f2e7b4ef86a	2eafc64769c500d635b7225c9b1411db8f50db8618e4d5807e1640b641a2f5ee	cosmicduke, 15min, bl=explorer
166	70223c0b-f4e4-694f-9335-d218bd83cb2b	1c86bcc74684c2533026a8b4d9463ad4b5a1f30f6915ca19197b41e0cb893b77	cosmicduke, 15min, bl=explorer
167	b022730e-c0cf-5044-8b60-208a66ffe4b5	0dc70c0f2ed18c813a89c59686f375787ba683b549b1e6bb9aee6ca33be64bfb	cosmicduke, 15min, bl=explorer
170	9bdd4b4e-5e64-0247-bf71-347c61ddc4f4	ccd3c69710977360459c0d2539d5e7e7defce097bcfee3ae62e564de7c938f17	cosmicduke, 15min, bl=explorer
171	7f99b8df-0bd5-ad40-b46a-1f174dc3af04	8290b324f5cdb5c3ea17fa48a74bc11c856f0da0b049d07d9316d161f71f26a5	cosmicduke, 15min, bl=explorer
174	ea659fbe-d54f-fe4d-9cb8-5e395de54f30	f6a2e324b2bfe8c718d630a7c266a5589e95285ad54fd0526965599ba27eba08	zeus, 2min, bl=explorer.exe, lw
175	ac63b0d5-8f98-0b41-bb1a-ed13f55bee45	f2f08ef6fae983e75a5b7b6432e29b29c5d057970bf0bef59c3fbcda6108b78a	zeus, 2min, bl=explorer.exe, lw
176	36fe5a3a-51c2-8a45-97de-d2216fdc7b80	eb255cddd82822a70d53dbaf3662bd88fea983ea86c6f83d810af09a68069df1	zeus, 2min, bl=explorer.exe, lw
177	70ae2f8d-4fbe-d74c-925b-aeba988a3df4	e653ad2c6897bcbd22650bb70f9bc9499bc4305ae25920f907b6c3cf106f359e	zeus, 2min, bl=explorer.exe, lw
178	01f1aaa9-6322-0244-adca-8d22f6075db0	dd6a89fb877e9a8c4a5769c03443f291535d8ed6e5a8cd308b5bf326d7e87710	zeus, 2min, bl=explorer.exe, lw
179	9ff841c9-43b3-3d4f-b107-3f63c13f2da8	cce3be2b1ebd0f27a539c9c1814dc04fb8b21c5b13e1fa8dcc3710fc8fc17fa7	zeus, 2min, bl=explorer.exe, lw
180	10535f9b-f985-c94b-928f-84befe87976f	bedfea0f87f465643b47b994619578e99a0deaff9728a02426cb1a9e5cd03b93	zeus, 2min, bl=explorer.exe, lw
181	4faabb33-b0d1-6c4c-a1b6-a6bae520e33e	afb91080d4a0521551de3a94e045c8aa3b23d14a6fd2122ef5fc6e8ff0f3b900	zeus, 2min, bl=explorer.exe, lw
182	985759a1-a215-3542-ac6e-1b922e2ce488	af6480b2bf8d3d792151b44fc60c49efd57e2c1de50394e276f873f06ba869bb	zeus, 2min, bl=explorer.exe, lw
183	8ba53e88-5a60-6344-bc59-27119e4e7a49	a27e4563dafafc96d09d8465226a55572338b69a219a74daed8fda5223916658	zeus, 2min, bl=explorer.exe, lw

184	4b00e39a-5f88-994e-98f1-22f9b99c2cd9	715446fcece80c11b49580b5073d06bfe03d3e0f9d890d16c9fb0936948a1e31	zeus, 2min, bl=explorer.exe, lw
185	17f98248-9902-474a-970f-d0f8770d999a	9134f9ba06cbf96333a759d967b9c5041aa661d8b48dd7a7cfdcc6b05de59d68	zeus, 2min, bl=explorer.exe, lw
186	4685481f-c2d9-dd4a-8ee2-859799578226	7585e6477fc41fcd2d2a47b5d4738d7230d14ce67a66459eb7bbe28def269b6b	zeus, 2min, bl=explorer.exe, lw
187	c8cd1f14-92fc-1942-99c3-23950bc6e521	988c8b201ad55031350df7fc750e6b5ff26bbb9b5098f66d9c976ce64e85eae7	zeus, 2min, bl=explorer.exe, lw
189	e37d44c0-e2c5-b74c-956b-1ba12a6941de	34ca1099cdb6d4bacf6bc36562b1171b60d28b79907e399efa61eb574eaa4338	zeus, 2min, bl=explorer.exe, lw
190	3a594669-5241-5e46-a6fb-25b2bfbdd36a	28c6320db338b03b7c38b165eed32295eb250720194f9b2924a202142b48bcc0	zeus, 2min, bl=explorer.exe, lw
191	4ddfc846-1ef0-1d44-97ec-52c7e3473706	28a4ec6f1fa0938dab6e2a8b7203627f1cca60d485cf71543d2fbfba73b72a36	zeus, 2min, bl=explorer.exe, lw
192	a810826b-4dae-7041-855e-35d3c6e7fcb3	9dad31cb544d0b6076d7f31307f98f1698f56f227fddcc9d53989cf17d5ee8d9	zeus, 2min, bl=explorer.exe, lw
193	77f8831c-dcbf-f743-b884-f4e6c0762349	8ee2d646b98da4983b28879bf51fcda9ddacd3901bf0176203fb0744eaca6344	zeus, 2min, bl=explorer.exe, lw
194	2b3e0f91-a6af-3e44-b8b0-5c86c72d9abd	8a1337da470226434308d465ffb45fd8c0c17ce0b065f0e753b882ee118d054d	zeus, 2min, bl=explorer.exe, lw
195	abf97775-2d89-d347-8d11-2a32731e687b	7ff5caf2c80bb5095fc1e7be1b853f3b299337ee20798674474e14b7fc9f24be	zeus, 2min, bl=explorer.exe, lw
196	e5d99deb-a38b-8d42-85f7-e0f82ec57aab	3ea58d9f68cb9e96d982fa051a91869d5179e603f26bb79f942713864a7dba91	zeus, 2min, bl=explorer.exe, lw
197	d0b1fdfd-e45c-9042-b7e6-46758324715c	1c947821aab3a2ff234e10a122749439a9fcd784981409e9f419a943f235ace5	zeus, 2min, bl=explorer.exe, lw
198	8bde5b3e-13c5-5b47-aff7-082abc6858f2	ffb687de64f7f7449037a402840dca5671d0c4dac7e7b3cdc48eba5601a0f5b8	dridex, 2min, bl=explorer.exe, lw
199	57c41096-0d60-f94b-9f8c-432cadccedf6	fa0d7cfb1ae1d770ce6202a574d23a878354fadfce4101cad53f062dd9d77f03	dridex, 2min, bl=explorer.exe, lw
200	c761afd2-4def-6a42-b4e5-0cc2970a0ae6	e655a65f39ff241a6f76ee97bb7d2695a96647d3e807c53682b5abebee32d9c1	dridex, 2min, bl=explorer.exe, lw
201	98d12c25-e178-5143-ab00-2eda153df030	c0119eb1855f46ea316acbd7aff2f808f6693c62fb2a38e430d23b0814aa2e39	dridex, 2min, bl=explorer.exe, lw
202	5553ab13-78f4-b34f-aa6c-3ac3062638cf	bd2cab8ad18af280f66a0e9b109678ae3fbf9ed0bb80b16f221c2110808b006a	dridex, 2min, bl=explorer.exe, lw
203	e2be7b8c-d138-c347-af8e-1eb999b247b2	bcaa57c93dc973aabd419b65dcdc4e9ae68bcae5ddfe920070cc2b2ae9dbaf3f	dridex, 2min, bl=explorer.exe, lw
204	a80cff53-73d7-784c-aada-d08e988a6a48	b62b13739dbad2bf40326a1a157252c94cd998a4d0fe17a5b3ba8da107c7372e	dridex, 2min, bl=explorer.exe, lw
205	88751e15-88ee-2a42-a360-a4bd61449bff	ae3a734078dd34f6ec7733bc08ee1119d4998f210e3b4aeb33c346901096fead	dridex, 2min, bl=explorer.exe, lw
206	e844d5da-045f-3a47-a00a-74f0a1c0031a	348088a8b914ec504bf3eef9028fa46f2ad205ca474c007a13db6004fda0dd8f	dridex, 2min, bl=explorer.exe, lw
207	401d8fbc-4bbd-dd4b-bf30-dfb5a402a6f0	257827b720515337f59df2d938a685fb22181c3b5bae8eb9680c291065495867	dridex, 2min, bl=explorer.exe, lw
208	df92c656-0498-6e4e-8ae1-388044048a60	079117d057a539ef9572fee0746920833d966d31115b0fdf2ac6f70e2d6d6c01	dridex, 2min, bl=explorer.exe, lw
209	530d0ad7-90cf-6e4d-9ad8-f76f807b4e9e	75717e7acf4f41de953e0c6f57986844bc21dcda546d5a37371ad8d5a7952782	dridex, 2min, bl=explorer.exe, lw
210	0902113d-72a4-a34e-b896-f932865aa832	8844abc943965caea0d0d43c1a5462c061154afdd702dbe5e52ec9850b42fe1f	dridex, 2min, bl=explorer.exe, lw
211	a4d3d83b-5a26-f844-b063-780cc4b7950d	874a4148d0e6c0319a0d3fc64ca382ea4ed8422dad9228a4663a1e3e2c079fc5	dridex, 2min, bl=explorer.exe, lw
212	ab6a28ad-c453-d546-b9a8-73e283294673	707cec6955c39f7e952a2440069b0cfc699e56b92f8b63c2d4713c897ba29095	dridex, 2min, bl=explorer.exe, lw
213	d27d698b-34c6-5f48-9e62-299f1247bb5d	663c54ba5481b8e900f056de12b6e0b6b87c0a617f8ed4de1021a9eb9e0fa5e1	dridex, 2min, bl=explorer.exe, lw
214	ab09a699-5e26-5843-90e8-6726ed6b44fe	382d2de27f8279ae4652786fd30ec74d7ea5a6ee8508c266c444934a76802d6e	dridex, 2min, bl=explorer.exe, lw
215	b5d4a25d-da8e-a74f-9d9d-a80c8f0084f9	346fda05d7a59fc4bd3eef27ba26df8eba96422c29523007d1560413f3fda00b	dridex, 2min, bl=explorer.exe, lw
216	49f92738-8e6b-d141-bd76-41234b1ae755	207b4ae38b3a5c51614aacd6b9d09bff242b23fab777446e9f752eefde57bac8	dridex, 2min, bl=explorer.exe, lw
217	0af265f9-a9d7-e743-b3b7-b43541dc0f01	38f3693fbc29345aadb2c46941ad3c6b5905e47b9011696d55ca550f23f30402	dridex, 2min, bl=explorer.exe, lw

218	3f210947-1fe1-6441-a4c6-1b643969ab1a	32a06219046989082deaa37c3fbfd330573f12748939bc866c1bbc0b2a36f167	dridex, 2min, bl=explorer.exe, lw
219	bea10ef4-a6e6-0443-96c4-820f51dbc568	9d50f55479404abcd4faca8afc3b2ba50d0a3846937ca937aff4c458339e2e10	dridex, 2min, bl=explorer.exe, lw
220	fc9ef7f4-a313-5c46-9874-bea001e1d6d1	4f56d394bd62fb2d6cd087f969b7097bf1bf6c72cfc893e0d3f7196ff1639007	dridex, 2min, bl=explorer.exe, lw
221	769e9262-d93a-5f41-9cc6-982c72ee28c9	3c2c6959cfce88ff06a6aa201293f91e01b350fd0502076cbb5f22a8bcd4729a	dridex, 2min, bl=explorer.exe, lw
222	d837f6d8-6368-4641-b44e-92747bb852ae	2d9c2edc8d1cfb2b5691b0f6a938d17d5adf1e7797ab401dfa12bd29df79af44	dridex, 2min, bl=explorer.exe, lw
223	90c0d58d-8fb6-d548-bbaa-0eb3c8c1f4df	fee523f69b21202264d8628c2d2d78f2a426c94b1ec5e915e1b664c51a7d21a0	retefe, 2min, bl=explorer.exe, lw
225	074018d0-5dab-e745-8547-fc5e700bcbe5	eeafc7c730890ef6ac1c5014a0611f845731b4a750f62e65ca6f19559258d745	retefe, 2min, bl=explorer.exe, lw
226	7c3e7f66-222e-3942-b34f-342edd09f7da	e12eda7ec6abcdef1afe625ed87963f5739a9a168e52e87a4075bbc7719e46b8	retefe, 2min, bl=explorer.exe, lw
227	2ae85f96-6a73-9343-a5c2-727fac84e9d7	e8c373b65a53503f2c8e44380a4f6e94e01edbb22df5600b827bac8cb644b7b0	retefe, 2min, bl=explorer.exe, lw
228	74aa2943-27b5-8640-971a-fd506427c72f	c1f7082709467457ac258b3db6dc4b4cbc9b23413f1391d912d25b1cf6141625	retefe, 2min, bl=explorer.exe, lw
229	6fb5f08f-60e4-2e47-94fd-f628cf91f3e9	b1efa88efef1b0872cbdd5f8c1e1e62c2b6fdbe3dca893c1e903466b90c8c3d5	retefe, 2min, bl=explorer.exe, lw
230	a643a698-3185-a440-9137-f267accab4c9	acbdcf4d2f6b5aaa3f25b8d6763c35180ff40d8c954bf3635e6230ba5c886f7f	retefe, 2min, bl=explorer.exe, lw
231	4e674f57-35b1-fd43-aa37-e87329a3d00e	a20cb81975634387b99fdb1f5f1206bbd565aded38a66c650f602c6ab94571da	retefe, 2min, bl=explorer.exe, lw
232	e4f0cc5d-4557-ea49-8f1e-dafa309b51dc	10692414c521984093780e28ef65af684102f9b199e3e94355840ecddc36b566	retefe, 2min, bl=explorer.exe, lw
233	859c43e7-e628-7b45-b62d-ad27ff8de63a	3616c3949b4277a09fc741cfd91d27503d81d3e7e8c85219c74562a3b83e5ab1	retefe, 2min, bl=explorer.exe, lw
234	7cbb91e1-7a5f-244f-acd3-9b1218d804a2	2959a6f9a9df4ec06f5a2026f20fadb7747b8bf8127a9b99027faf1149bca48f	retefe, 2min, bl=explorer.exe, lw
235	2239c8bf-9b08-cb4b-827b-2640d3c10755	93d8d2bb56b3c3613ceeca97307701406b6cd10280f4b0eb549beff3f5e67606	retefe, 2min, bl=explorer.exe, lw
236	b4ce13ce-e958-9d42-bf83-347af8e3073b	92fd69ddbf988612657f0d4456b128b13385b6a605a58ac37731ac14c69180d7	retefe, 2min, bl=explorer.exe, lw
237	29718323-0b38-be42-91a2-10baabee4fd3	76d7bca7f6e0c1482a3da636d5908bd94cb8b6a05453b41d0dd9055590adb917	retefe, 2min, bl=explorer.exe, lw
238	186ae042-cdf4-4b4c-937c-11fe334f373e	75d3ed8866ef6d69f76e7898d325ca36f49bd7fbadbd3e6619e8a1a1af5fa2d1	retefe, 2min, bl=explorer.exe, lw
239	3423a029-6926-0848-b33f-7f6a82c31002	51beffd8fcf546eec940298c029d4afefc30d88ece1c944cfea79a631e62c736	retefe, 2min, bl=explorer.exe, lw
240	1bb3e65e-7cf6-b641-a14f-3397821af59d	11c52fab3f2bd283f9e980729155f6f970b37b931bd8a6ac8815d8dfa51e88c0	retefe, 2min, bl=explorer.exe, lw
241	9f739608-0e4f-d144-ad7d-058158957691	8f15828e94eacd6c4cf254ba8c1248aeb7184819aef01dddea0861d4bd7425d8	retefe, 2min, bl=explorer.exe, lw
242	e9e40a1e-890d-a74f-81da-0b20fa6f4368	6b0f9e0abf0e6e13c048ab299618c4139e590f70b4b4b93357f493645d0da18a	retefe, 2min, bl=explorer.exe, lw
243	f253c688-8237-d742-b325-212549c31a52	4ed3dc079f1009f6bc40b6c520f104b5c50e6996c2fc4462a1d813a8edaeeb50	retefe, 2min, bl=explorer.exe, lw
244	404dce5e-6c3d-7f44-9fc9-f0982428c352	3da46d4a1f904643ebcb8e38212912a632a1657a1d2c74e1b463c0d4e016d0b7	retefe, 2min, bl=explorer.exe, lw
245	d0d5949a-8bb2-8f4e-9e98-d8949e75880a	2b77febe4b92990ba58df6c25bb4a432d0566afb906ed1b6ff432af2e3ea7f7d	retefe, 2min, bl=explorer.exe, lw
246	f0489bdf-ec4a-3648-8799-3aec0fa8e1bc	2ad436187ce3458bd72384035277932f16f970a555363f83fbf8e242d2a698b5	retefe, 2min, bl=explorer.exe, lw
247	bacd3cc6-0c0f-7643-bbb1-bc338a2a57ca	1dff72c8c2ce0f7ec92f6ccc8310ac8adb2ff227c2988f93d92e354688957dc1	retefe, 2min, bl=explorer.exe, lw

Tabelle B.1.: Übersicht der verwendeten Traces

B.2. Nicht verwendete Traces

ID	UUID	Sample Hash	Kommentar
		Möglicherweise falsche Zuordnung zu Familie gemäss SEL	
5	3428c8cc-53df-fc42-9d01-619f90b038f1	6d34ded00c0da9887ba752872093f59c649de72a1f629a32014f5ed8be509363	darkcomet 180s bl=explorer.exe
		Fehlerhafte Aufnahme gemäss SEL	
7	22f64677-eac7-8040-b9d2-993a44917e1c	03bbfcc623b38a9080d6073832a58f38202ae70d462ce504deabddc438071fc5	xtremrat 180s bl=explorer.exe
49	f4b8502b-cca2-d346-9798-bdeb8dbc8711	4837eede5275f3833f5c1ef78afa79a22da3130117c68a4a1eedbd30887d8e37	fobber, 2min, bl=explorer
		Ungewöhnlich kleiner Installationsgraph. Aufzeichnungsfehler wahrscheinlich.	
116	f189c0d7-2cc8-ab4e-beb2-566411bc071a	2c92f0668a30fcbdc02672fa1e258a1c7f167c6253883219be048fd391d990c3	xtreme_120_lw
		Kein tumbleweed.exe-Prozess	
224	92542834-eca1-484c-904e-cc3529c42d5e	eeb68f753e421b2e8a505d06fb658d13298d2cf4686012f63a21e701fbdb2261	retefe, 2min, bl=explorer.exe, lw
		Zu wenig Samples für Familien: PeerToPeer-Zeus, Zeroaccess, Citadel und Ice IX	
9	2d786370-b2cf-9c49-ae11-d1ef3a1e941a	2b30f023b93247401c193f1d79d840ed598a9c659237a6937e75c3427dd6c974	zeroaccess
30	2365f0eb-d71a-9f4c-8a49-26f5f9e80929	51abef033b87dbf08f9f5c126f2cf2066e974171d259c61516102e0fd50db396	citadel (antivm)
31	c18fcbb6-68db-3140-b227-ec76121d7d88	a4462aa277b26a44ca5099a89735a9f7706cc3d5483582aada20d83823731e16	p2pzeus
32	fbf4e070-0288-8d4a-83a3-9a8d4a763604	d18364a9b891af0f80338baa1f899f8a876caca2568f113b6ceabb432a48e623	p2pzeus
35	d5dcc6a7-abd6-1a40-8035-22add4cdaedf	2b30f023b93247401c193f1d79d840ed598a9c659237a6937e75c3427dd6c974	zeroaccess, 2min, bl=explorer
71	5a799bbc-2051-0943-a8d5-b96bae541140	e603f4a8f1bf63a6545d2f03ac125370bae3428a484e19447ac0e9185e77e7ef	iceix, 2min, bl=explorer.exe
72	085f5833-b2c0-0c44-a4d2-1da6494e41c1	94f903d29abddc594837f5d1e73e8bacb7fd34212dc5baa235a9fb7c44574038	citadel, 2min, bl=explorer.exe
77	b7b2df87-e9bc-0645-b74e-eb493ee0af2d	51abef033b87dbf08f9f5c126f2cf2066e974171d259c61516102e0fd50db396	citadel, 2min, bl=explorer, slim
78	57d91a7f-6eb7-2e4e-9260-9b66dba3b99c	94f903d29abddc594837f5d1e73e8bacb7fd34212dc5baa235a9fb7c44574038	citadel, 2min, bl=explorer, slim
81	749c7602-0818-8b41-9555-efce6e5c0212	e603f4a8f1bf63a6545d2f03ac125370bae3428a484e19447ac0e9185e77e7ef	iceix, 2min, bl=explorer, slim
82	d627cc44-f415-b04c-b093-389b2b5ad5d7	2b30f023b93247401c193f1d79d840ed598a9c659237a6937e75c3427dd6c974	zeroaccess, 2min, bl=explorer, readfile
155	e6dc6a58-1b6d-404a-b5b1-0587100d6f2b	2b30f023b93247401c193f1d79d840ed598a9c659237a6937e75c3427dd6c974	zeroaccess, 2min, persistence
169	7a7b207b-b0bc-fd41-8871-53b5d7793efa	2b30f023b93247401c193f1d79d840ed598a9c659237a6937e75c3427dd6c974	zeroaccess, 2min, persistence
172	42d551c2-bf5d-4b42-86c5-1849caeb4c4a	2b30f023b93247401c193f1d79d840ed598a9c659237a6937e75c3427dd6c974	zeroaccess, 2min, bl=explorer,services, fat
173	509a0afa-cf84-8346-89ea-8c6e236fd1d1	2b30f023b93247401c193f1d79d840ed598a9c659237a6937e75c3427dd6c974	zeroaccess, 2min, bl=explorer,services, fat
		Zu wenig anderen Traces mit vergleichbarer Parametrisierung	
4	0aed9f8c-b466-e749-83e5-31262c60e02c	1d3a00bb72ebed2fee1dd7f6ea3ecd6f232f25bcf80b1895b079567fc22340ec	fobber (5min timeout)
10	c7bac113-4d4a-374e-8f19-0ba5eefda5ac	144871280381361e9fb444db491e101f81f6bde44bf3c0275aa589d639616acd	zeus, 3min
63	dfd5ac70-f939-544b-9a54-e2681a83916f	72d2eefac006b20901e49947e33d6a13394751f4cf2b404ccfd317d44e861d92	zeus, 2min, bl=explorer
69	7490feaa-d847-384f-86ed-1295399b66ed	84b7bf3db3e5cd432ca4eb09471493667ce08345a69bcbb7d608b77f21ddc81f	zeus, 2min, bl=explorer

79	f0efb0c9-9741-5e44-9ce4-1c3c5d16549e	72d2eefac006b20901e49947e33d6a13394751f4cf2b404ccfd317d44e861d92	zeus, 2min, bl=explorer, slim
80	12bd2c7d-1b3f-e64c-baba-252bad85c04a	84b7bf3db3e5cd432ca4eb09471493667ce08345a69bcbb7d608b77f21ddc81f	zeus, 2min, bl=explorer, slim
		Sample in Trace mit unterschiedlicher Parametrisierung bereits verwendet	
11	569cb787-29eb-494e-a331-7a931084d85f	1d3a00bb72ebed2fee1dd7f6ea3ecd6f232f25bcf80b1895b079567fc22340ec	fobber, 10min
12	4596eec6-8928-c947-8060-3acf907a1563	d9bac445de8537c483bc60dd1ebdba00c1a2649c66197aa554a2d3b5e19468c8	fobber, 2min
13	6ec91c52-10b6-5f4c-be3b-f5149e79b6f9	ab6e4dba7dcd24f139c33eef87e7e3c025a5be56511a1de5a37f1235dd749687	fobber, 2min
14	91ca00a2-c19c-1e4e-8dd9-f25a6a307bfa	33776404e6463e2093f07e2a8daf881d86b4a052d26102b29e7c28bb3fa20b64	fobber, 2min
15	cdfb4831-cb12-8149-acc1-926b794ae894	191586f91b8057eed118f9e0cc66ef1400534fe29bf4ab5e60d4754a706ed7fa	fobber, 2min
16	207cdbb3-8c5a-cb45-a56c-6c066ec77062	8212ebe1311b41d539a76cfb27d7e73137974bf656df3d7a6e45893aaffb65d1	fobber, 2min
17	e2ca753f-5e5e-c642-a512-afd78658abe0	8135ad8a608c9d841cea14fe0151978c5d2b8adb7fa006f31e579df22e342cbd	fobber, 2min
18	f1787194-1934-8f4e-a735-b2705e351e24	6145a3f86acff5c37c00fc59e5daaadd47e7d4b179f77cf7ddf5354d5203fbd2	fobber, 2min
19	0daeab98-abfb-3948-bc89-bd1b279ae16d	4837eede5275f3833f5c1ef78afa79a22da3130117c68a4a1eedbd30887d8e37	fobber, 2min
20	74cc8e78-478a-5f4f-8d47-c0870eab6d8d	825f80a8dc2bd203c0087748d8cc5d44ffec2289f75e351980f95d0bf5249db6	fobber, 2min
21	f39c3d31-a887-7145-adf9-0e81472977fd	64a20918e9cce528bf2d8c7ea49337176893b404b724d069f989bc166c32a27b	fobber, 2min
22	8c06c5b7-eb8d-be4e-9f8d-44127c512e1f	38ec54d51e6844f226987c4bca0f2269b25156aab4622b0d31c80ace10f23dc2	fobber, 2min
23	bead2926-c72f-3049-a468-c5321e2ab150	20f2d01f9fd163cc8ffb8ed7e4ce153c2f5cd063ea1cea67f77707369fbb6503	fobber, 2min
25	fdb279cb-bc39-aa40-bc75-a80c5bd11259	5d0d798cf8ef16cc57ff57999f70f25d02c7c69a1dc67114dd5766c990771469	fobber, 2min
26	d2b60328-5085-3b49-bfeb-93d249245e1c	1d3a00bb72ebed2fee1dd7f6ea3ecd6f232f25bcf80b1895b079567fc22340ec	fobber, 2min
33	53574de3-d8ea-2d45-9588-1eae93aca67d	8ebc2335b4fbc5f56ce0cd8b4e7b4df5cc9dc9ab9eb7e3c466d330b84fb8b289	fobber 2min
34	61531a18-f4cd-e440-9596-9389f45dba0d	1d3a00bb72ebed2fee1dd7f6ea3ecd6f232f25bcf80b1895b079567fc22340ec	fobber, 10min
36	f05756a1-604f-754f-824d-29ee4091a614	1d3a00bb72ebed2fee1dd7f6ea3ecd6f232f25bcf80b1895b079567fc22340ec	fobber, 10min, bl=explorer
57	60dbcc2b-35f1-aa47-ab41-7c41bcf163ba	43a600b42d48b683b633fa5318c99d955b2edbaaee6fc064ed904b6f8db26a5c	DC from VT 180s
58	e2302aa0-a0f3-a845-8efd-189296f5294d	74482cc037b0b169aff6a66e955347fc68e2cb3a3a4218827ab8d92cbddb3c10	DC from VT 180s
59	5fc25216-013b-a749-8f48-e754eeec2005	ad7302bc95da3e72d9206c733f8e6229af851810bc3a631572c3e39e1266a8a9	DC from VT 180s
60	fb3c38b4-a603-6844-b0b6-b3743fab790b	b78e50b73389197d8ffe4a2a50e95f1d472cb2ef5c5a7d4eca2ebb9dbf92e80e	DC from VT 180s
88	51e4f3e2-b250-c649-af4d-a62f4189604a	1d3a00bb72ebed2fee1dd7f6ea3ecd6f232f25bcf80b1895b079567fc22340ec	fobber, 3min, bl=explorer, slim
89	c678fa1b-bc35-464a-917c-1ddcc45b407c	d9bac445de8537c483bc60dd1ebdba00c1a2649c66197aa554a2d3b5e19468c8	fobber, 3min, bl=explorer, slim
90	5ad28991-2fb9-d646-afb0-2b4d79168752	ab6e4dba7dcd24f139c33eef87e7e3c025a5be56511a1de5a37f1235dd749687	fobber, 3min, bl=explorer, slim
91	cc1b6a98-3cf4-4c42-bd47-9ddbef26c5d6	33776404e6463e2093f07e2a8daf881d86b4a052d26102b29e7c28bb3fa20b64	fobber, 3min, bl=explorer, slim
92	a5d09c0d-9512-2a4c-a16b-8f2cf6f7cdeb	191586f91b8057eed118f9e0cc66ef1400534fe29bf4ab5e60d4754a706ed7fa	fobber, 3min, bl=explorer, slim
93	a5cad8fb-1350-c146-a061-9e2b0a905420	8212ebe1311b41d539a76cfb27d7e73137974bf656df3d7a6e45893aaffb65d1	fobber, 3min, bl=explorer, slim
94	c5350733-c4dc-8745-af6b-013141f31968	8135ad8a608c9d841cea14fe0151978c5d2b8adb7fa006f31e579df22e342cbd	fobber, 3min, bl=explorer, slim
95	7f9c1c90-a044-5c44-813c-36bb7d27a1a5	6145a3f86acff5c37c00fc59e5daaadd47e7d4b179f77cf7ddf5354d5203fbd2	fobber, 3min, bl=explorer, slim

96	4a945fc6-4be5-c842-b93a-26aa13f8d030	4837eede5275f3833f5c1ef78afa79a22da3130117c68a4a1eedbd30887d8e37	fobber, 3min, bl=explorer, slim
97	678be56a-337a-0d46-a73f-c9c057fab8cc	825f80a8dc2bd203c0087748d8cc5d44ffec2289f75e351980f95d0bf5249db6	fobber, 3min, bl=explorer, slim
98	39222049-8fce-f24b-b0ca-8f0aaee237a2	64a20918e9cce528bf2d8c7ea49337176893b404b724d069f989bc166c32a27b	fobber, 3min, bl=explorer, slim
99	00daa6c3-e8bb-a64e-91c3-fd782a4c970e	38ec54d51e6844f226987c4bca0f2269b25156aab4622b0d31c80ace10f23dc2	fobber, 3min, bl=explorer, slim
100	2b02ae4f-9deb-c145-8ac5-307331664c39	20f2d01f9fd163cc8ffb8ed7e4ce153c2f5cd063ea1cea67f77707369fbb6503	fobber, 3min, bl=explorer, slim
101	44a4be00-4077-f44f-bde8-b8112b198546	8ebc2335b4fbc5f56ce0cd8b4e7b4df5cc9dc9ab9eb7e3c466d330b84fb8b289	fobber, 3min, bl=explorer, slim
102	90685a2f-1a8b-3946-810d-8d4f2149a1d9	5d0d798cf8ef16cc57ff57999f70f25d02c7c69a1dc67114dd5766c990771469	fobber, 3min, bl=explorer, slim
103	13112ec7-d776-cb4e-99c7-0d11980a6eb1	1d3a00bb72ebed2fee1dd7f6ea3ecd6f232f25bcf80b1895b079567fc22340ec	fobber, 3min, bl=explorer, slim
135	e8328ac0-0b7c-d348-a8a6-7870e52a4dd2	03bbfcc623b38a9080d6073832a58f38202ae70d462ce504deabddc438071fc5	xtreme_180_lw
136	ff6df560-0f89-2e41-bd7a-962fd6be25c8	041025af7e54777e8c1ad062930d212873cc9f85166de1e17a5b19f1be8a251f	xtreme_180_lw
137	2271cc51-f42c-424e-96d7-9ecf3d06252d	18759a8afef4c0984282681afa2aea3ca09ae050295e9430499d3fff31b5d3c2	xtreme_180_lw
138	4a18f13a-d4e1-de40-877a-6a2f7d54c0dd	ec8b33b7972535e2b9a688b734f74b3907c6211221e9e39f28c9e23cb3d96334	xtreme_180_lw
139	b5a2266b-292d-2a4c-a74a-e84b68c48a59	eb2bbe13c8f115260b3afa3756dbe0381f4ca7802611035c8676afb7a37ebc39	xtreme_180_lw
140	e0dc7be0-5a48-9849-a094-1d1831ac1ec1	e46cbed7747902cbf1bc0f26dbc847549d4c626facea329f3e165117ff28ed7e	xtreme_180_lw
141	2eb86274-1fbe-d74c-8899-fc22bb4a197b	da14fb1ce631bfe825dc3285b37571e023a8ef3976c6b6a1b59f3d4d3e7abb68	xtreme_180_lw
142	2a154b25-d911-6a40-a7f6-26ca8e811b5e	c05858dd1b14919f9ea42c2f68f44e3ca0fdedef67963c684401769458aa5a1c	xtreme_180_lw
143	5ee08ae6-72da-204d-acd1-70b8449a8dd5	bce68fe35a87c36ae1c7a6699987bef4af3288a3676fc4f82f25faadd1efaf83	xtreme_180_lw
144	f5e463b8-cc2f-b94c-a9c9-bf384a32f218	b37bb9f0d0e3a1de397a2aa35d62b3ec8ffb35e51185ff73394729ff993eb9a1	xtreme_180_lw
145	706a99b3-903b-3e4c-95ab-dbb2163b6b7b	adb8456c69c76756ab9ee94a49c6c56ed980091d51a19e56a6dae8fc98472587	xtreme_180_lw
146	6bb93b12-9f10-b84e-9daa-3e33193c248e	a558ede6fbe3d60f88d57373306224c44f4304d1ed5be8d05294d4458f2c4974	xtreme_180_lw
147	ac8b5062-24e8-974f-b73e-4899bb750f80	041025af7e54777e8c1ad062930d212873cc9f85166de1e17a5b19f1be8a251f	xtreme_180_lw
148	19dd59ed-71c2-1e4d-ad8a-28c17cc34a46	18759a8afef4c0984282681afa2aea3ca09ae050295e9430499d3fff31b5d3c2	xtreme_180_lw
149	046449d5-5824-d240-9bfd-f88000075271	9892dd1005d21cf3f1d9b0e64936bf577c2a12884625ce22ffc569824e4c51f2	xtreme_180_lw
150	2b353198-218b-db4b-8363-dfd052e7345f	9298ac79d15e8246d430e0ab689525f85136fbdbbb722b4af3fb31fa252ef356	xtreme_180_lw
151	0421e263-2f86-9444-8d7d-6d2446d3a6e4	952f7ad678d8803d1ce527b653dc92d8fbd14fab2fdf5fd3abd391205c10bac0	xtreme_180_lw
152	8f555c2d-6906-b149-a3a7-1f8a6e2308e0	75ff6bca3f2ed2a9d0762a254a9838caa4cbebda73d08d5b7009dc70a78e2786	xtreme_180_lw
153	2768e364-6305-cb41-8d17-c21813e42355	72b79f302f8bc4c47b277210ef53c26bc1d6f6ff6e34f997d97e6215e14c859d	xtreme_180_lw
154	b7bc03ce-db2f-f844-ad85-9918f466d249	9ff0bfea87849e6250dacea547fb8b3f0350e130fcef03987e1abffffe50584e	xtreme_180_lw

Tabelle B.2.: Übersicht der nicht verwendeten Traces

C. Clusteranalyse

C.1. Workspace 7414

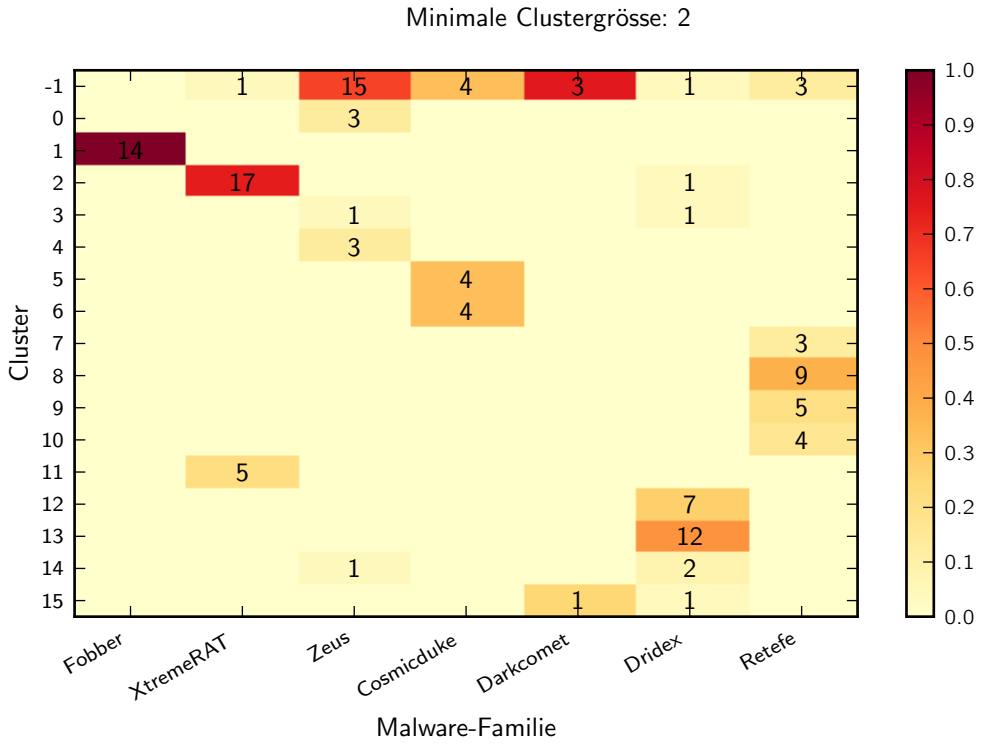


Abbildung C.1.: Korrelation der Clusterung $Z_{\{7414\}}$ mit den Malware-Familien

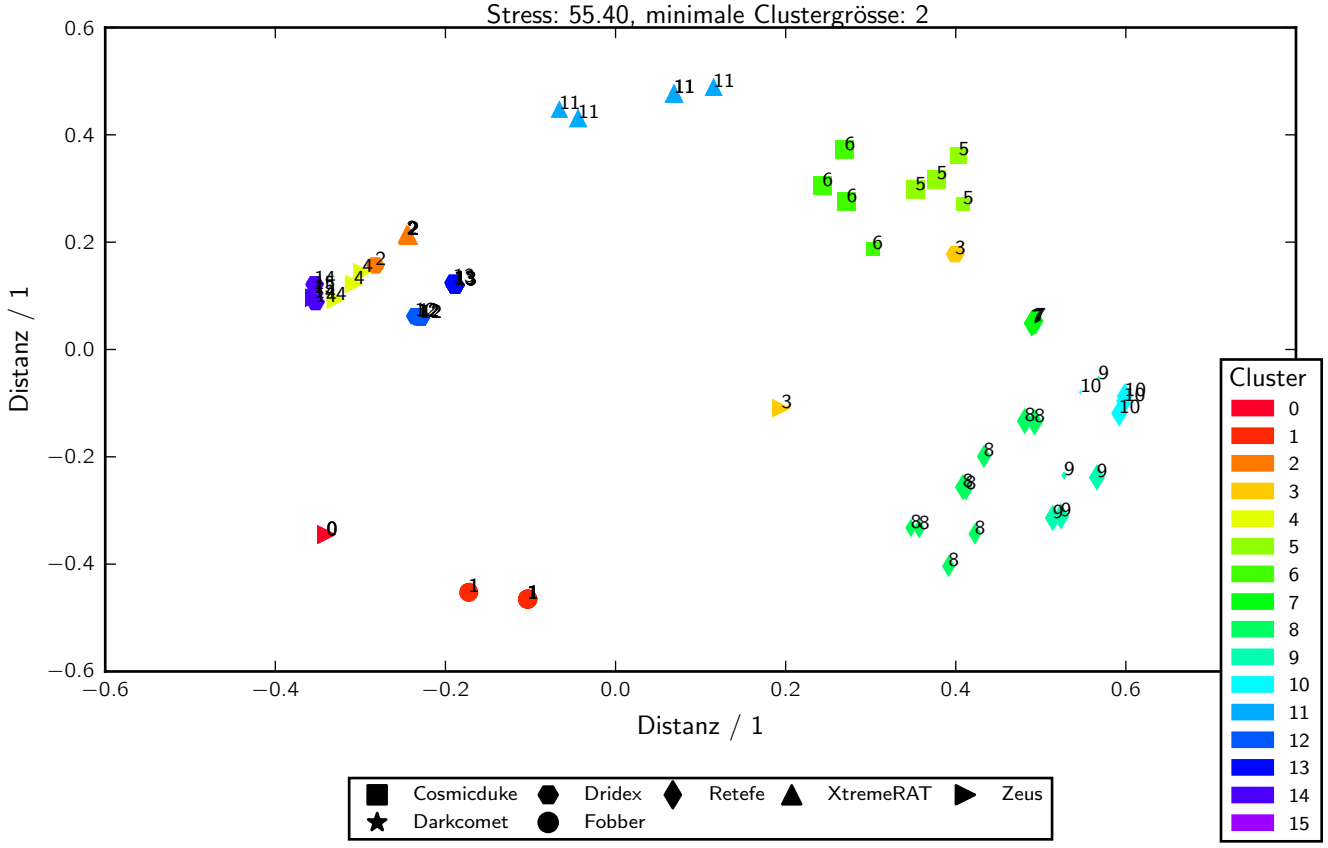


Abbildung C.2.: MDS-Projektion von $D_{\{7414\}}$

C.2. Workspace f9c9

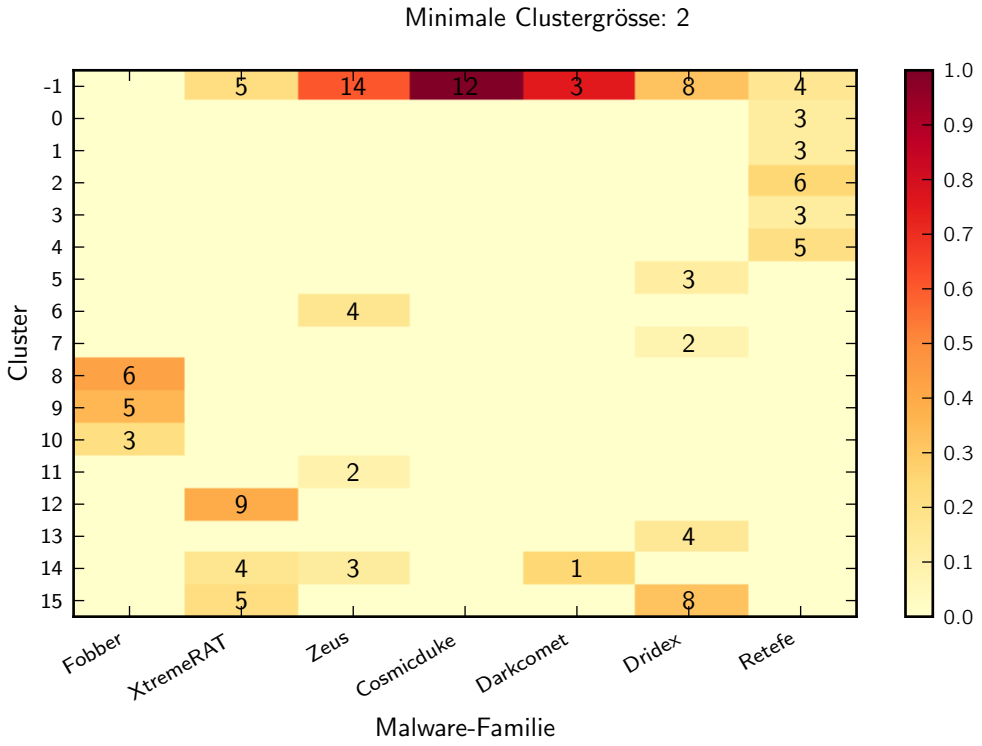


Abbildung C.3.: Korrelation der Clusterung $Z_{\{f9c9\}}$ mit den Malware-Familien

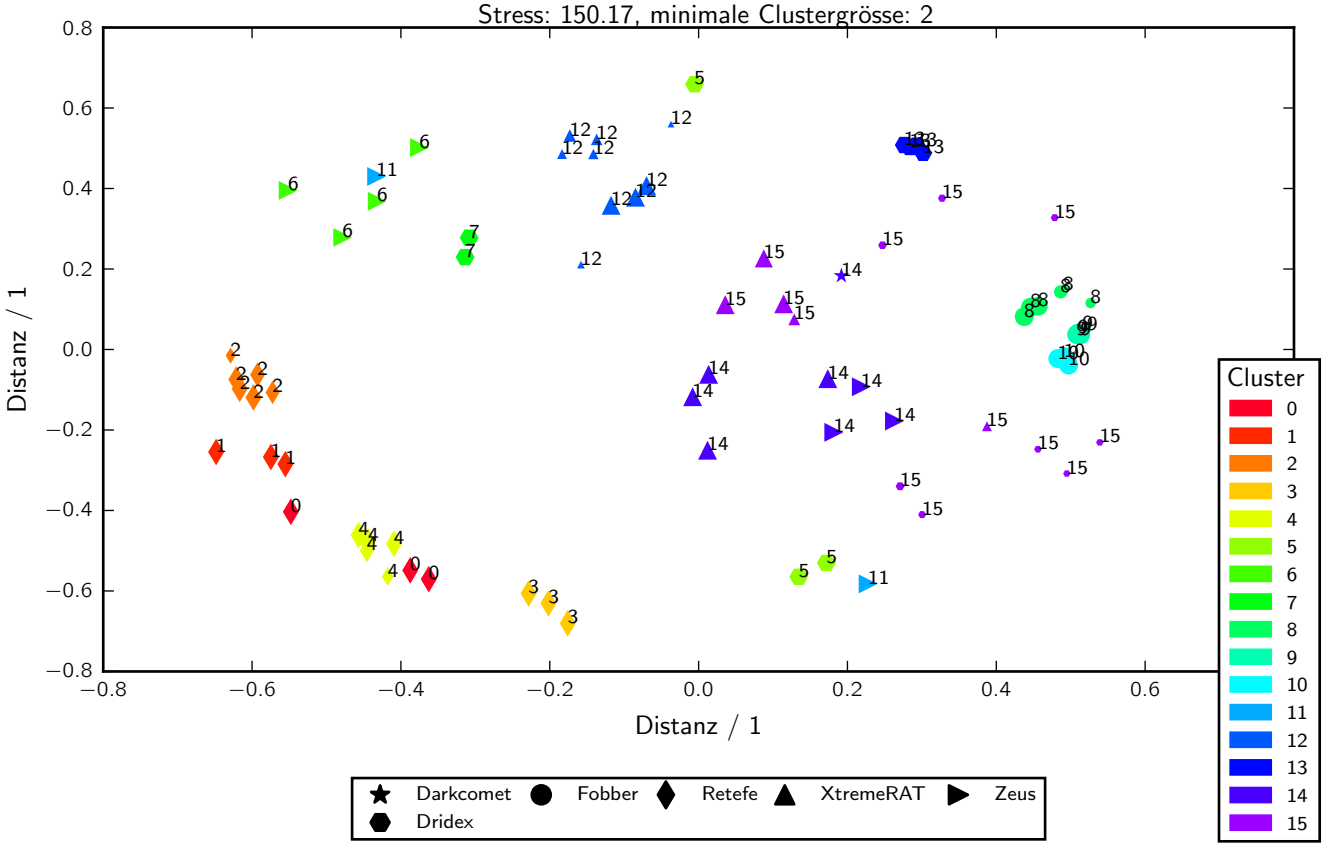


Abbildung C.4.: MDS-Projektion von $D_{\{f9c9\}}$

C.3. Workspace 0c7e

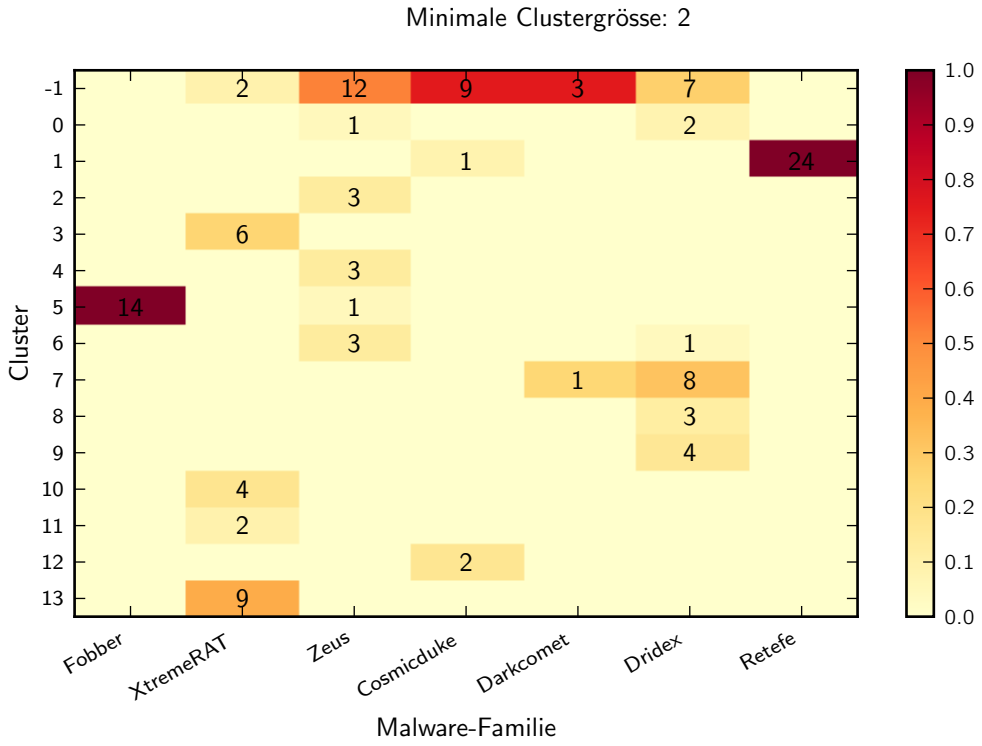


Abbildung C.5.: Korrelation der Clusterung $Z_{\{0c7e\}}$ mit den Malware-Familien

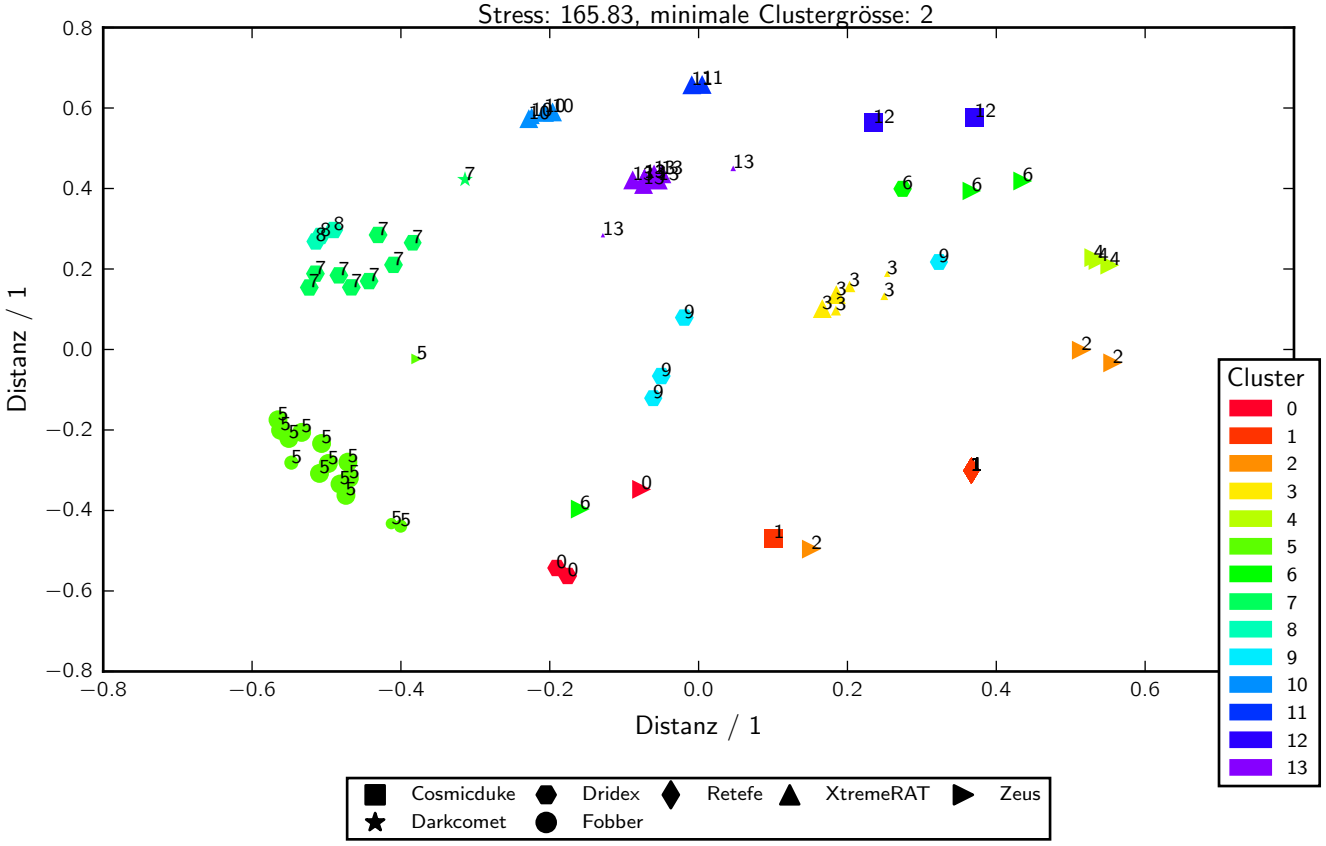


Abbildung C.6.: MDS-Projektion von $D_{\{0c7e\}}$

C.4. Workspace 3449

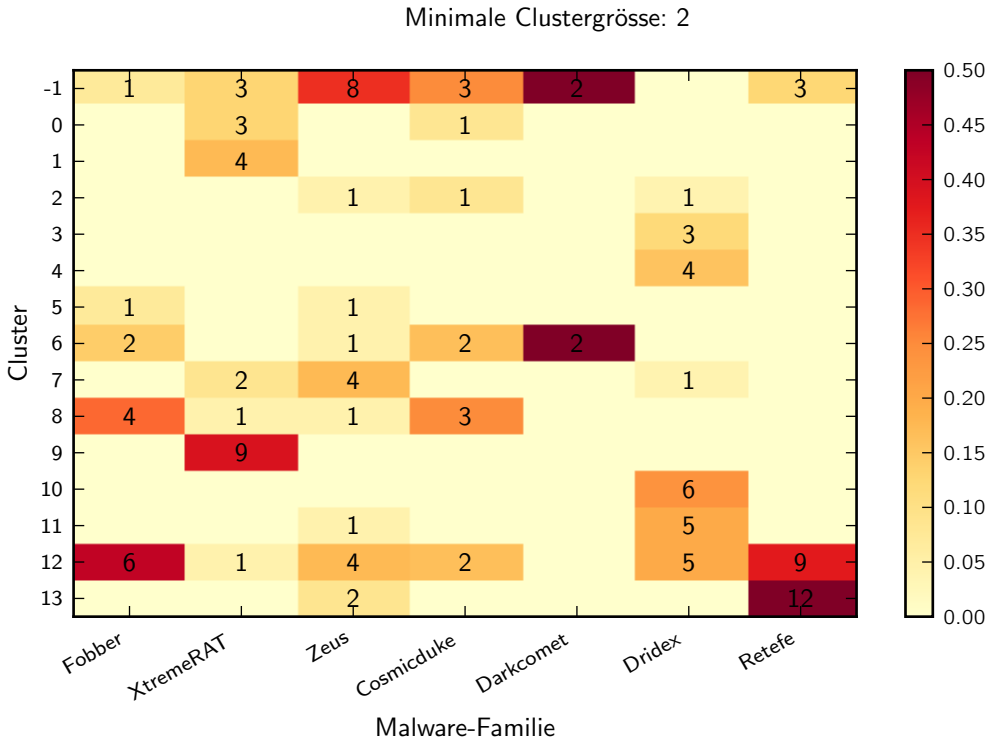


Abbildung C.7.: Korrelation der Clusterung $Z_{\{3449\}}$ mit den Malware-Familien

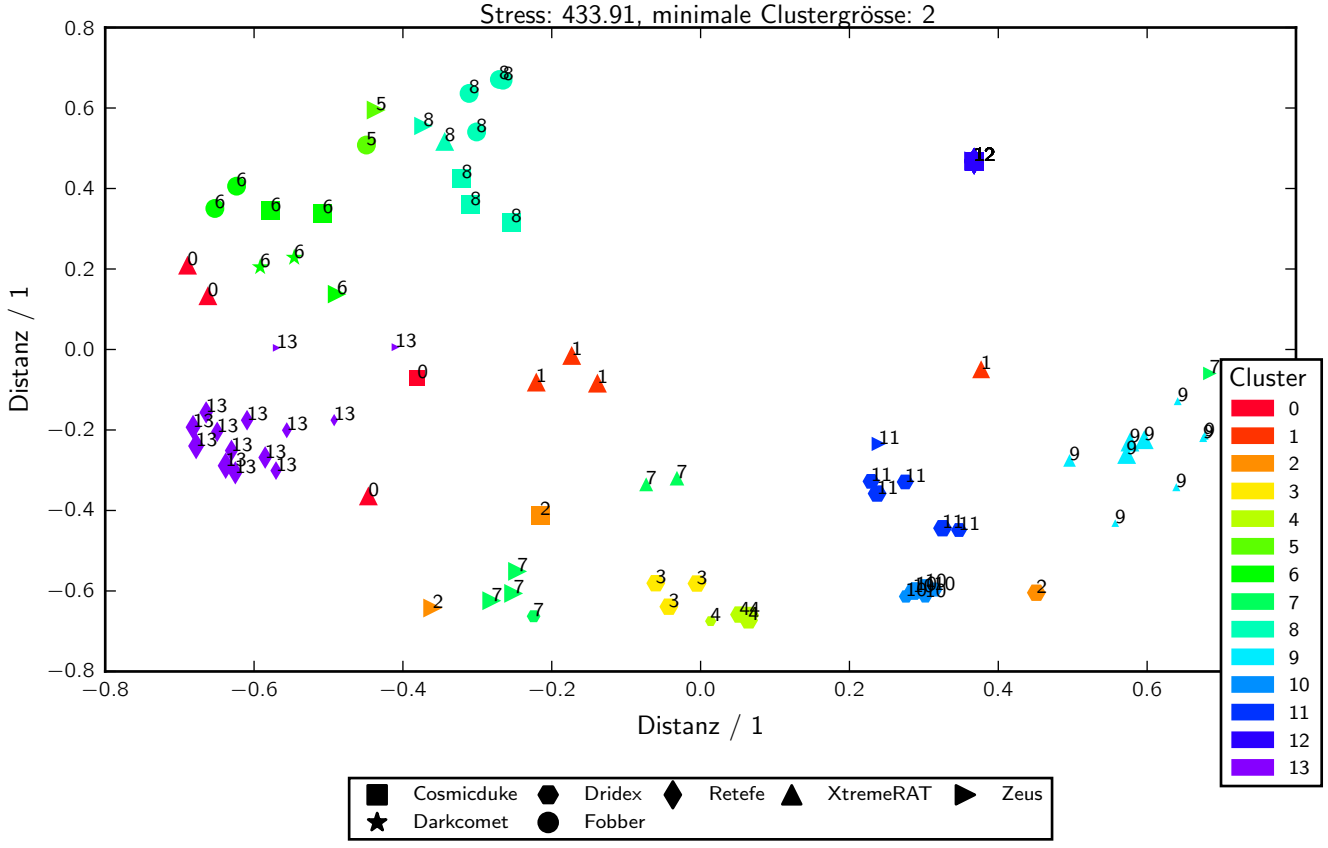


Abbildung C.8.: MDS-Projektion von $D_{\{3449\}}$

C.5. Workspace 9ab4

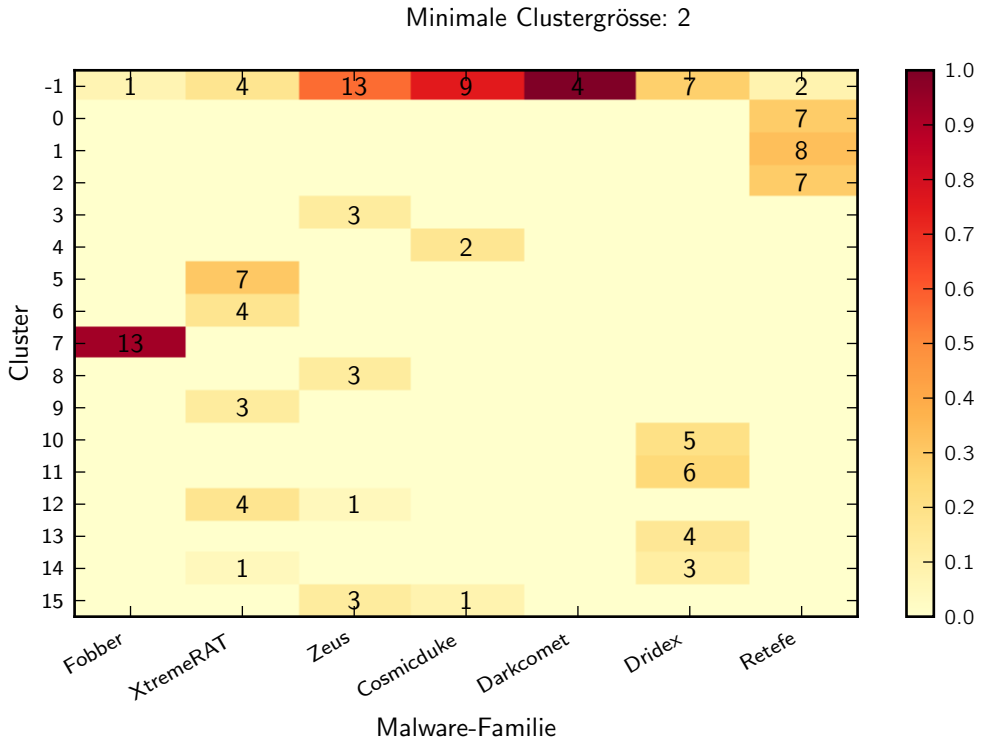


Abbildung C.9.: Korrelation der Clusterung $Z_{\{9ab4\}}$ mit den Malware-Familien

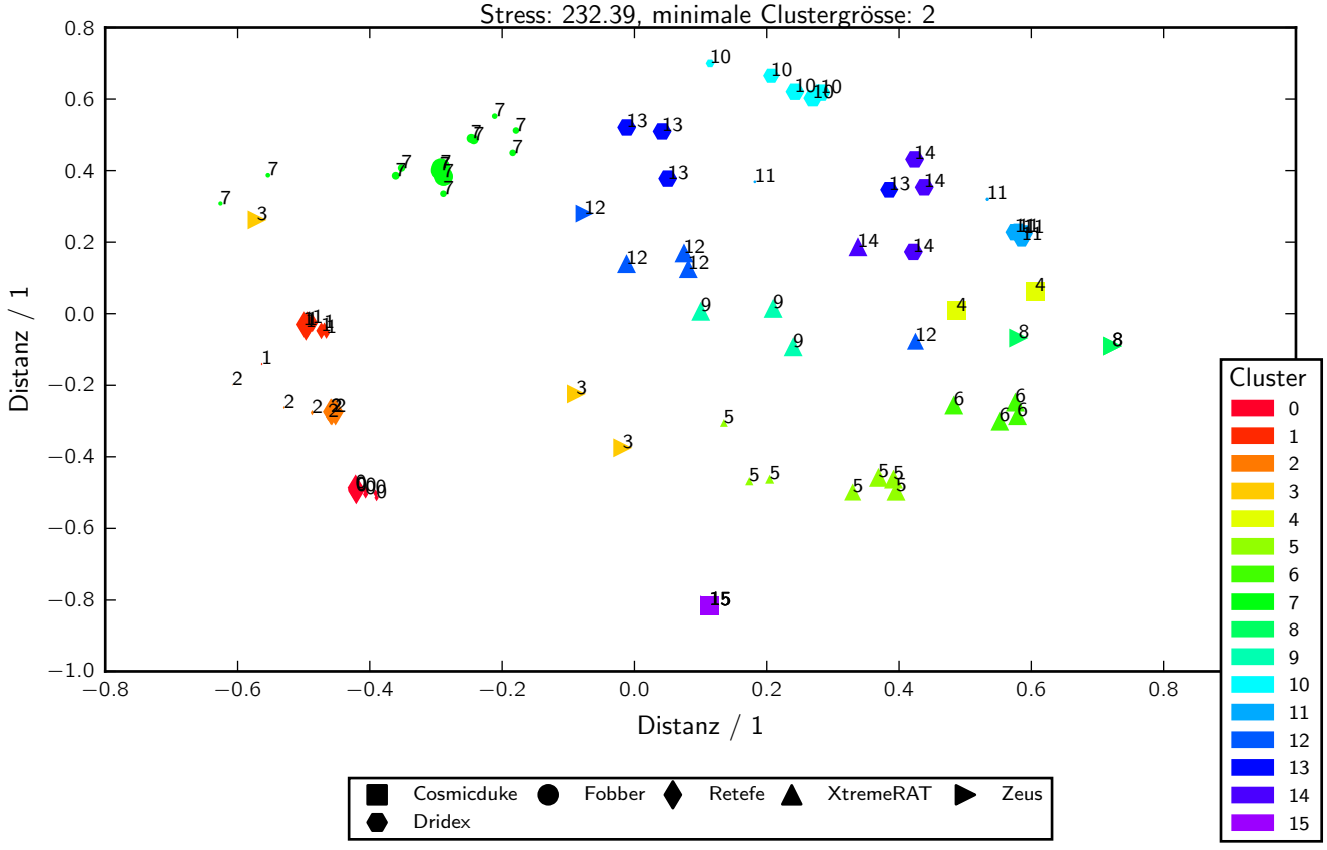


Abbildung C.10.: MDS-Projektion von $D_{\{9ab4\}}$