Hochschule Karlsruhe für Technik und Wirtschaft:

University of Applied Sciences

Informatik und Wirtschaftsinformatik (IWI)

Frühwarnsystem für Cybergefahren

Projektarbeit von:

Selim Karali, Adrian Graf, Maja Magschok

Betreuende Professoren: Prof. Dr. Dirk W. Hoffmann, Prof. Dr. Thomas Fuchß

Inhaltsverzeichnis

Einleitung (Maja Magschok)

Diese Projektarbeit beschäftigt sich mit der Entstehung eines Frühwarnsystems in Form eines Prototyps. Von den ersten Vorstellungen und Ideen, bis hin zu der Entwicklung. Zunächst soll das Problem beleuchtet werden und warum unser Projekt die EnBW unterstützen würde. Anschließend wird genauer darauf eingegangen, was wir uns unter einem Frühwarnsystem für Cybergefahren vorstellen und wie wir uns die Umsetzung als Ganzes vorstellen.

EnBW

Die EnBW ist eines der größten Energieversorgungsunternehmen in Deutschland. Dadurch ist sie Angriffsziel vieler Cyberattacken oder mitbetroffen, wenn beispielsweise eine Anwendung betroffen ist, welche von dem Unternehmen verwendet wird. Unser Frühwarnsystem soll am Ende die EnBW dabei unterstützen potenzielle Gefahren zu erkennen und frühzeitig präventive Handlungen auszuführen. Dementsprechend orientiert sich unser Lösungsansatz stark an der Arbeitsweise unseres Projektbetreuers.

Aktuelle Arbeitslage

Da es sich bei der EnBW um ein sehr großes Unternehmen handelt, welches bereits viele Programme nutzt und Mitarbeiter beschäftigt, gibt es durchaus bereits Vorkehrungen gegen Cybergefahren. Es ist im Interesse der Firma, diese so früh wie möglich zu erkennen, sodass Gegenmaßnahmen ergriffen werden können.

Das bisherige Vorgehen bei EnBW beruht auf der manuellen Suche nach Sicherheitslücken und Gefahren. Dies nimmt jedoch viel Arbeitszeit in Anspruch und ist nicht so gründlich wie ein Programm mit entsprechender Datengrundlage.

Zielsetzung im Rahmen der Projektarbeit

Diese Projektarbeit soll verständnisvoll erklären, wie wir von der Aufgabe zu dem fertigen Projekt gelangen. Dies soll für den Leser verständnisvoll erklärt werden. Der Fokus liegt dabei nicht nur auf dem Projekt selbst, sondern auch auf der Organisation und auf den Hindernissen, welche das Projekt erschwert haben. Letztendlich soll die Projektarbeit die Erstellung des Prototyps zusammenfassend darstellen und die Einzelschritte und Entscheidungen nachvollziehbar erklären.

Grundlagen (Maja Magschok)

Im folgenden Kapitel wird erläutert, was genau für ein Problem existiert und wie dieses bereits vermieden wird. Außerdem wird erklärt, mit welchem Lösungsvorschlag sich diese Projektarbeit beschäftigt. Bevor es in den nächsten Kapiteln dann um die Umsetzung geht, wird genauer auf die grundsätzliche Organisation des Projektes eingegangen.

Cybergefahren

Grundsätzlich versteht man unter Cybergefahren (1) ein gezielter Angriff auf ein oder mehrere informationstechnische Systeme. Diese werden von Einzelpersonen oder Gruppen ausgeführt und haben das Ziel dem System zu schaden. Bei großen Firmen sind es häufig geheime Firmeninformationen, welche das Ziel des Angriffs sind. Cyberangriffe sind keine Seltenheit und Firmen wie die EnBW sind so groß, dass sie durchaus häufiger Ziel einer solchen Attacke werden. Auch verwendet die Firma viele Systeme, welche selbst ein Angriffsziel sein könnten.

Grundidee des Frühwarnsystems

Ein Frühwarnsystem soll an erster Stelle (wie der Name schon sagt) vor Angriffen warnen. Je früher man eine Information über einen möglichen Angriff erhält, desto früher kann man Gegenmaßnahmen ergreifen. Daher ist es das übergeordnete Ziel des Frühwarnsystems, die Information zu übergeben, welche Systeme derzeit von Cyberangriffen bedroht sind. Das System soll vor allem den Mitarbeitern der EnBW dienen, welche dadurch früh an das Wissen über Angriffe gelangen können.

Problem

Derzeit muss ein EnBW-Mitarbeiter viele verschiedene Websites und Newsletter durchforsten, um sich ein Bild davon zu machen, welche Systeme, die auch die EnBW betreffen, von Cyberangriffen betroffen sind.

Dies frisst enorm viel Arbeitszeit und ist nicht gerade übersichtlich oder strukturiert. Dadurch kann es auch leicht zu Fehlern kommen. Beispielsweise fällt es nur bedingt auf, dass ein System häufiger angegriffen wird als sonst.

Bisherige Lösungsansätze

Es gibt bereits recht viele Websites, die solche Informationen angeben. Manche übersichtlicher als andere. Dies sind jedoch sehr viele und sich durch all diese zu klicken, frisst viel Zeit und Mühen. Im nächsten Kapitel, wenn es genauer um die ersten Umsetzungsversuche geht, wird auch genauer darauf eingegangen welche Vor- und Nachteile die bisherigen Websites bieten und wie wir uns diese im Projekt zunutze machen können.

Lösungsvorschlag Frühwarnsystem

Die Lösung für das Problem scheint zunächst einleuchtend zu sein. Es braucht eine Website, bei der alle Informationen von verschiedenen Quellen gefiltert werden können. Dies soll mit der EnBW abgesprochen werden, sodass klar wird, welche Informationen wichtig sind und welche Funktionen die Website überhaupt braucht.

Motivation

Zusätzlich zu den Wünschen der EnBW, kommen einige Fakten, die die Erstellung eines Frühwarnsystems für Cybergefahren noch mehr motivieren.

So wurden im Jahr 2023 mehr als 58 deutsche Unternehmen Opfer einer Cyberattacke. Natürlich sind auch Einzelpersonen oder gleich ganze Universitäten betroffen. Auch unsere Hochschule wurde nicht verschont und viele Websites und Informationssysteme der HKA waren das ganze Semester über nicht oder nur begrenzt verfügbar.

Grundsätzlich kann man sagen, dass die entstandenen Schäden durch Cyberattacken oft sehr schwerwiegend sind und nicht nur die HKA ist monatelang von ihnen betroffen.

Sollte das Unternehmen zudem zentral für die Wirtschaft sein, reichen die Folgen noch weiter und betreffen ihrerseits mehr Unternehmen.

Dies kann auch große finanzielle Folgen haben. Unternehmen müssen viel Zeit und Kosten dafür verwenden, ihre Systeme zu aktualisieren und sich sicherer gegen Attacken zu schützen.

Dabei ist jedoch klar, dass die Mitarbeiter selbst das größte Sicherheitsrisiko sind. Durch Schulungen können zwar einige Mitarbeiter gebildet werden und so die Gefahr verringert werden, dass sie beispielsweise auf einen ominösen Phishing-Link klicken, doch auch die Angreifer werden immer besser

und passen sich an neue Systeme und Abwehrmechanismen an. Es braucht nur einen Fehlgriff eines naiven Mitarbeiters, um ein ganzes Firmensystem zu überlisten.

Daher ist es essenziell, dass diese frühestmöglich gewarnt werden und somit achtsamer agieren können.

Allerdings macht es einen Unterschied, ob das Frühwarnsystem von allen Mitarbeitern verstanden und genutzt werden sollte, oder nur von wenigen übergeordneten Einzelpersonen, die hinterher die Informationen an alle Mitarbeiter weitergeben.

Organisation

Essenziell für das Projekt ist der Austausch mit der EnBW. Das primäre Ziel ist schließlich, ihnen den Arbeitsalltag zu erleichtern. Auch ein wöchentlicher Austausch mit der Hochschule hilft dabei, dass die Dimensionen des Projekts im Rahmen bleiben, sodass es während des Semesters fertiggestellt werden kann.

Konzept (Maja Magschok)

Bevor es an die Entwicklung gehen kann, ist es wichtig, dass wir die Aufgabenstellung richtig interpretieren. Schließlich wollen wir nicht an etwas arbeiten, das am Ende weder der EnBW etwas nützt noch den Anforderungen der Hochschule genügt.

Daher wird in diesem Kapitel zunächst genauer auf die Aufgabenstellung eingegangen. Danach werden die bereits vorgegebenen Websites genauer untersucht und bewertet.

Häufige Cybergefahren

Um uns gegen die Cybergefahren zu schützen, hilft es zu wissen, welche von ihnen häufiger vorkommen. (2)

1. **Phishing-Angriffe**

   Hier sind die Mitarbeiter der Firma diejenigen, die vorsichtig vorgehen müssen. Bei Phishing-Angriffen nutzen die Kriminellen oft gefälschte Emails oder Websites. Klickt der Mitarbeiter auf einen Link in beispielsweise einer solchen E-Mail, gibt er so zum Beispiel seine Nutzungsdaten frei.

   Verfasste Mails oder erstellte Websites sehen oft täuschend echt aus. Mitarbeiter sollten vorsichtig sein und ihren Verstand nutzen.

   Die EnBW hat 2022 noch 26980 Mitarbeiter (3) beschäftigt. Diese Zahl ist im letzten Jahr vermutlich noch mehr gestiegen. All diese Mitarbeiter bieten ein natürliches Ziel für Phishing-Angriffe und es ist daher sehr wichtig, dass sie über diese gut in Kenntnis gesetzt werden, sodass ihnen die Gefahr bewusst ist.

2. **Ransomware-Angriffe**

   Dabei werden zunächst verschiedene Methoden (so auch Phishing) genutzt, um Ransomware zu verbreiten. Auch bei einer ungewöhnlichen Werbung auf einer Website könnte es sich um Ransomware handeln.

   Einmal auf dem System, verschlüsselt Ransomware jegliche Daten und zeigt anschließend dem Opfer eine Lösegeldforderung an. Diese existiert häufig in Verbindung mit einem Countdown, um so den Druck auf das Opfer zu erhöhen.

Gegen Ransomware schützt man sich, indem man häufig Backups macht, um so im Notfall wieder auf den Ursprungszustand zurücksetzen zu können.

Es hilft auch, die Mitarbeiter zu bilden und so beispielsweise die Gefahr eines Phishing-Angriffs zu reduzieren. Auch Schutzmaßnahmen wie Antiviren-Filter oder E-Mail-Filter sind ein wirksames Mittel gegen solche Angriffe.

Auch Ransomware müssen wir in unserem Frühwarnsystem beachten. Die EnBW hat viele Mitarbeiter und dies zielt erneut auf sie ab.

3. **DDoS-Angriffe**

   Distributed Denial of Service (DDoS)-Angriffe sind darauf ausgerichtet, bestimmte Websites, Dienste oder Netzwerke durch eine Überflutung mit unnötigem Traffic unzugänglich zu machen. Traffic bedeutet in dem Fall, dass die Website sehr oft aufgerufen wird oder bei vielen Nutzern viele verschiedene komplizierte Ladevorgänge gleichzeitig machen muss. Dazu werden oft komplizierte Botnetze, also Netzwerke aus kompromittierten Computern oder anderen Geräten benutzt, um kontinuierlich Datenpakete an das Angriffsziel zu senden. Dies führt erst zu einer Überlastung der Netzwerkressourcen und schließlich zu einer Dienstunterbrechung.

   Unternehmen können spezielle DDoS-Schutzmaßnahmen nutzen, um sich gegen solche Angriffe zu schützen.

Es gibt noch viele weitere Methoden (wie zum Beispiel SQL-Injektion, Cross-Site Scripting oder APTs). Alle zielen auf unterschiedliche Schwachpunkte ab, können aber meistens durch das Nutzen von Antiviren-Filtern oder spezieller Software vermieden werden. Wichtig ist es, dass die Mitarbeiter über die Gefahr in Kenntnis gesetzt werden.

Natürlich ist es gut, so früh wie möglich vor Angriffen zu wissen, sodass man sich baldmöglichst schützen kann. Da hilft nun ein Frühwarnsystem.

Anforderungen

Die Anforderungen an unser Endprodukt waren zu Beginn der Projektarbeit recht ungenau. Unter dem Begriff Frühwarnsystem kann einiges verstanden werden und konkrete Ziele waren bei der Aufgabenbeschreibung nicht genannt. Dies sorgte in unserem Team für Verwirrung und wir wussten nicht so recht, wo wir anfangen sollen. Nach mehreren Besprechungen mit dem Projektbetreuer seitens EnBW haben wir folgende Anforderungen ausgearbeitet:

Der Prototyp soll:

- eine möglichst große Datengrundlage haben
- aktuelle CVEs anzeigen und z.B. mit einer Trendanalyse ein weiteres Kriterium bieten, um den Risikofaktor der CVEs beurteilen zu können
- anzeigen zu welchen CVEs ein Exploit existiert und weitere Informationen zu dem Exploit anzeigen
- eine Suchfunktion für Exploits und CVEs besitzen

Analyse der Datenquelle

Nach der Besprechung der Anforderungen erhielten wir eine Liste von Seiten die als potenzielle Datenquellen dienen könnten. Daraufhin untersuchten wir die einzelnen Seiten, ob sie geeignet wären die Daten zu liefern die wir zur Umsetzung der Anforderungen brauchen.

Kaspersky

https://cybermap.kaspersky.com/de/stats

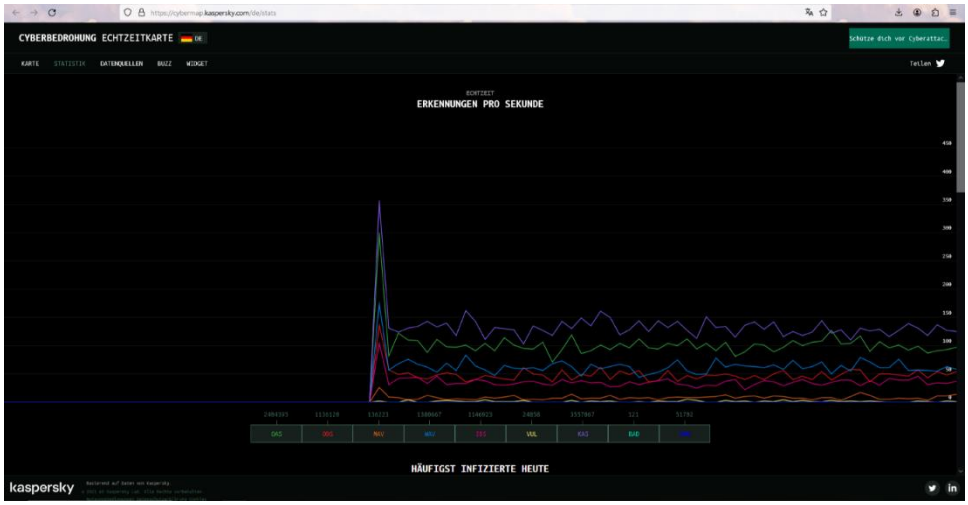


Abbildung 1: Kaspersky Internetseite

Wie man anhand Abbildung 1 sehen kann, zeigt die Kaspersky-Website die erkannten Cyberangriffe in verschiedenen Ländern vergleichend da. In Abbildung 1 ist dies von Deutschland dargestellt.

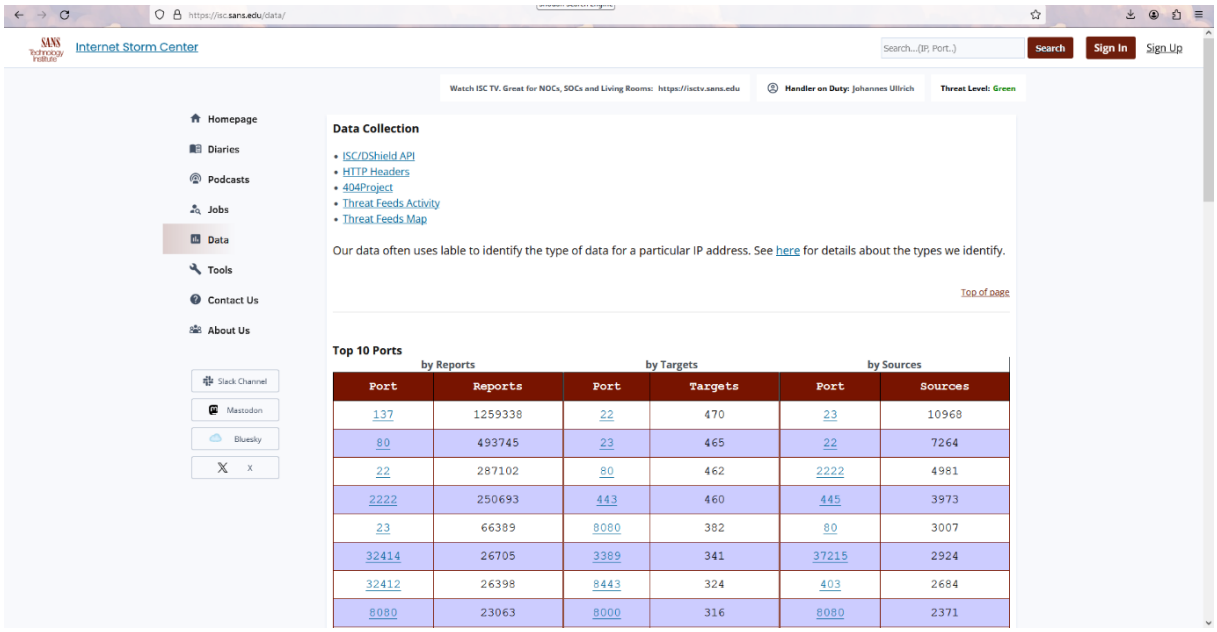
Um dies zu filtern, vergleicht Kaspersky verschiedene Scan-Algorithmen und stellt deren Funde hier vergleichsweise dar. Diese kann man dann aktivieren oder auch deaktivieren.

Der Fokus von Kaspersky liegt auf dem Vergleich der Attacken-Zahl auf verschiedene Länder der Welt. Die Attacken werden dabei nicht detailliert dargestellt, sondern vereinheitlicht.

Für unser Frühwarnsystem ist ein Vergleich der Angriffe mit anderen Ländern zu oberflächlich und somit irrelevant.

SANS Internet Storm Center

https://isc.sans.edu/data/threatfeed.html



Port	Reports	Port	Targets	Port	Sources
137	1259338	22	470	23	10968
80	493745	23	465	22	7264
22	287102	80	462	2222	4981
2222	250693	443	460	445	3973
23	66389	8080	382	80	3007
32414	26705	3389	341	37215	2924
32412	26398	8443	324	403	2684
8080	23063	8000	316	8080	2371

Abbildung 2: SANS Internetseite

Das Internet Storm Center überwacht die Anzahl der schädlichen Aktivitäten im Netz. Dabei liegt der focus auf Internetverbindungen insbesondere spezielle Ports. Da wir keine Liste der Ports von EnBW bekommen, nützt uns auch diese Datenquelle nichts.

Shodan

https://www.shodan.io/

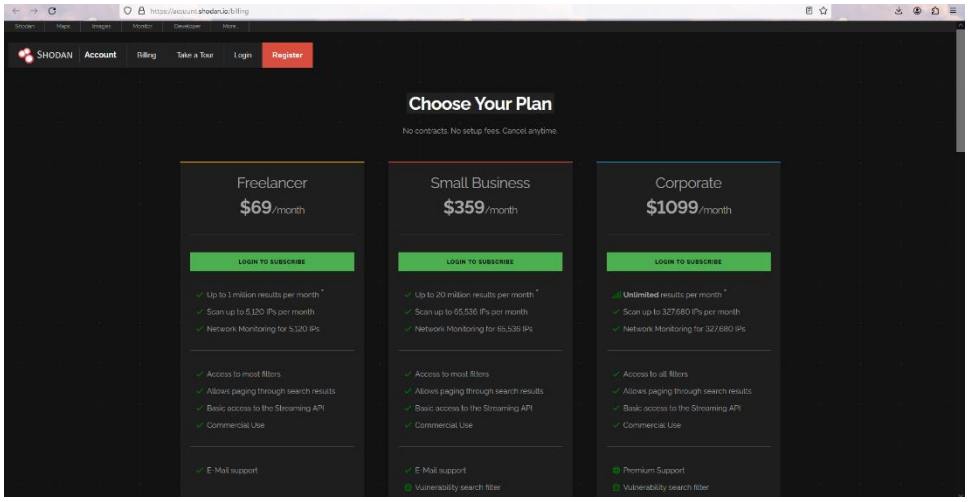


Abbildung 3: Shodan Internetseite

Wie man sehen kann, ist unser Problem bei der Seite von Shodan ein ganz anderes. Bevor wir uns auf eine kostenpflichtige Informationsquelle festlegen, ziehen wir es vor, erst einmal alle anderen kostenlosen Alternativen durchzugehen.

Warnmeldungen des LKA BW

https://lka.polizei-bw.de/warnmeldungen/

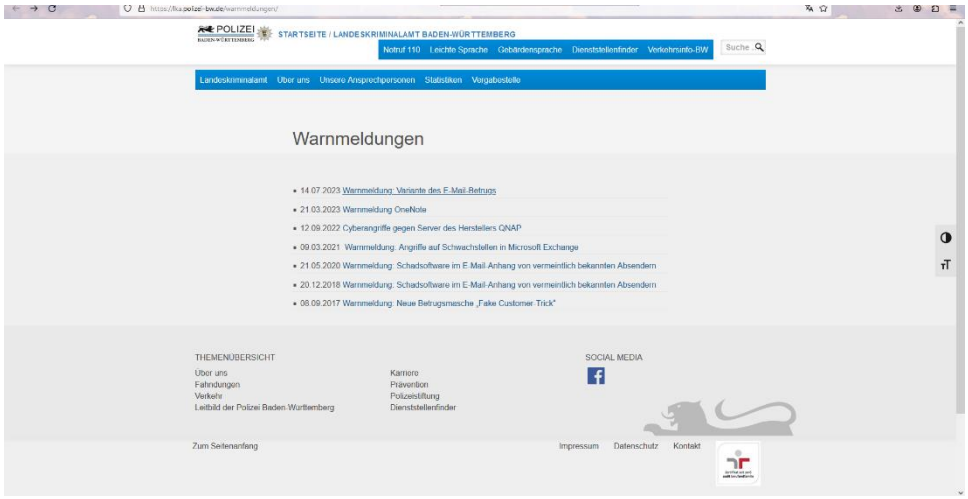


Abbildung 4: LKA Internetseite

Die Polizei stellt auf ihrer Warnmeldung-Seite die aktuellen Betrugsfälle in Berichtform dar, sodass jeder verstehen kann, worum es dabei geht. Das Problem liegt hier zunächst in der Anzahl der Berichte. In den vergangenen sieben Jahren kam es auch nur zu sieben Berichten. Das rechtfertigt schlichtweg gar nicht erst den Aufwand, die Website in unser Frühwarnsystem mit einzubeziehen. Auch sind die Berichte dort dazu gedacht, eine Sicherheitslücke verständlich der breiten Masse zu erklären. Somit sind die relevanten Informationen gar nicht erst direkt ersichtlich.

Metasploit

https://www.metasploit.com/

Metasploit ist für uns tatsächlich ein sehr hilfreiches Werkzeug. Das Metasploit Framework beinhaltet sehr viele Exploits mit detaillierten Informationen. Diese können wir nutzen, um herauszufinden, ob es eine Exploit für ein CVE gibt. Leider bieten sie keinen kostenlosen Dienst an die Daten über die Webseite zu beziehen. Stattdessen werden wir die Daten lokal über das Open-Source Metasploit-Framework beziehen.

Virustotal

https://docs.virustotal.com/reference/overview

Virustotal ist ein Dienst von Google. Dort können beliebige Dateien hochgeladen werden, die dann von über 70 Virenscannern überprüft und mit diversen IT-Sicherheitsdiensten und Antiviren-Software-Herstellenden geteilt werden.

Da wir keine einzelnen Dateien auf Viren testen müssen, ist die Seite für uns nicht relevant.

BSI (Bundesamt für Sicherheit in der Informationstechnik)

https://wid.cert-bund.de/portal/wid/kurzinformationen
Die Seite des BSI liefert uns genau die Informationen, die wir brauchen. Jeden Tag werden die wichtigsten Informationen zu neuen CVEs hochladen. Somit haben wir eine tagesaktuelle Liste der zurzeit relevanten CVEs.

Entwicklung – Voraussetzungen (Maja Magschok)

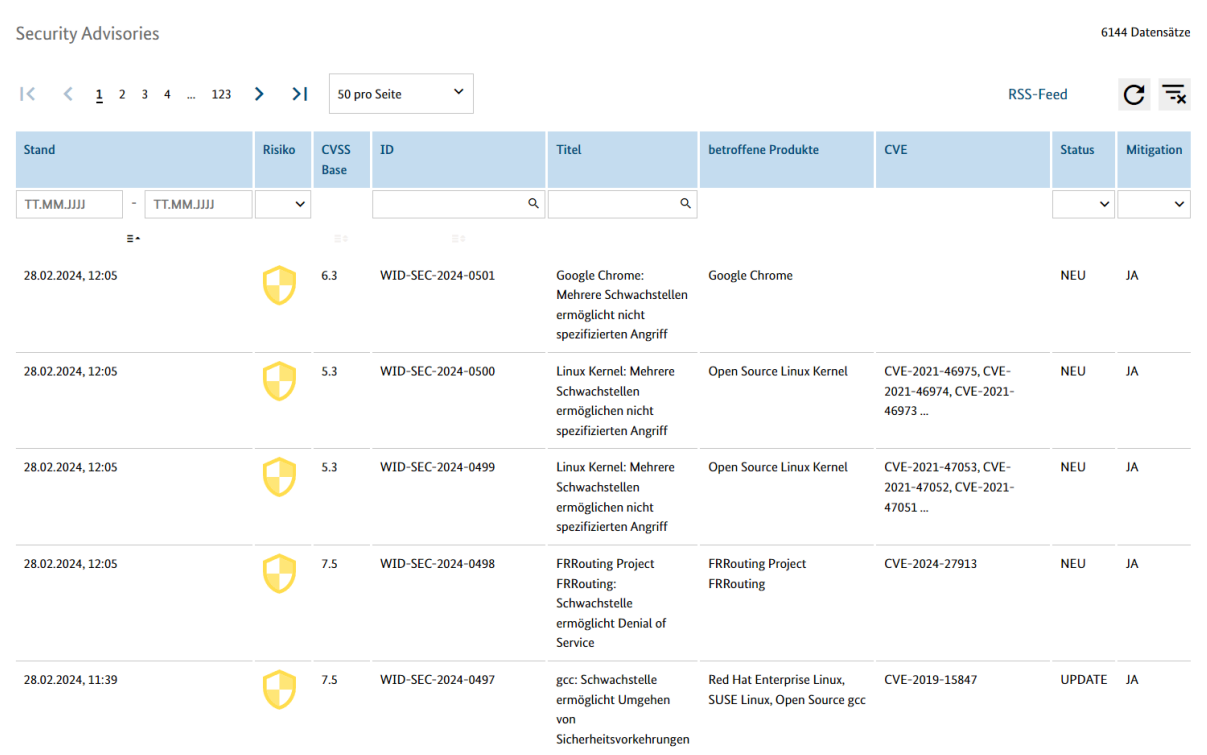
In den nächsten Kapiteln geht es darum, die grundsätzlichen Schritte zu erklären, auf denen die Entwicklung basiert. Als erstes geht es um die Website, von der die Informationen für das Frühwarnsystem gefiltert werden, dann wird genauer darauf eingegangen, was man unter Metasploit versteht und wie wir an die Informationen gelangen wollen.

Was ist das BSI?

Das Bundesamt für Sicherheit in der Informationstechnik (BSI) befasst sich mit vielen Fragen rund um die IT-Sicherheit in der Informationsgesellschaft. Ziel von ihm ist die präemptive Sicherheit vor Cybergefahren in der Gesellschaft. Dabei hilft es, so viele Menschen wie möglich, so früh wie möglich vor den Gefahren zu warnen und zu erklären, wie sie sich besser dagegen schützen können.

BSI-Website

https://wid.cert-bund.de/portal/wid/kurzinformationen



Security Advisories

6144 Datensätze

1 2 3 4 ... 123 | 50 pro Seite | RSS-Feed

Stand	Risiko	CVSS Base	ID	Titel	betroffene Produkte	CVE	Status	Mitigation
TT.MM.JJJJ - TT.MM.JJJJ								
28.02.2024, 12:05		6.3	WID-SEC-2024-0501	Google Chrome: Mehrere Schwachstellen ermöglicht nicht spezifizierten Angriff	Google Chrome		NEU	JA
28.02.2024, 12:05		5.3	WID-SEC-2024-0500	Linux Kernel: Mehrere Schwachstellen ermöglichen nicht spezifizierten Angriff	Open Source Linux Kernel	CVE-2021-46975, CVE-2021-46974, CVE-2021-46973 ...	NEU	JA
28.02.2024, 12:05		5.3	WID-SEC-2024-0499	Linux Kernel: Mehrere Schwachstellen ermöglichen nicht spezifizierten Angriff	Open Source Linux Kernel	CVE-2021-47053, CVE-2021-47052, CVE-2021-47051 ...	NEU	JA
28.02.2024, 12:05		7.5	WID-SEC-2024-0498	FRRouting Project FRRouting: Schwachstelle ermöglicht Denial of Service	FRRouting Project FRRouting	CVE-2024-27913	NEU	JA
28.02.2024, 11:39		7.5	WID-SEC-2024-0497	gcc: Schwachstelle ermöglicht Umgehen von Sicherheitsvorkehrungen	Red Hat Enterprise Linux, SUSE Linux, Open Source gcc	CVE-2019-15847	UPDATE	JA

Abbildung 5: BSI Internetseite

Auf der Website des BSI sind die Informationen über verschiedene Cybergefahren sehr übersichtlich zusammengefasst. Es gibt die Möglichkeit nach Datum und verschiedenen Zeiträumen zu sortieren. Dies reicht jahrelang zurück. Doch es werden auch sehr aktuelle Gefahren schnellstmöglich angezeigt.

Schließlich wird die Gefahr auf einer Risikoskala einsortiert und dementsprechend auch noch mit einem farbigen Wappen ausgezeichnet (wie man in Abbildung 7 sehen kann).

Dies hilft dabei, direkt zu sehen, bei welchen Gefahren es sich um kritische Sicherheitslücken handelt und welche so unbedeutend sind, dass man sie auch ignorieren kann.

Zudem erhält jede Sicherheitslücke eine eigene ID, gibt an welche Systeme betroffen sind und gibt die eigene CVE an.

Die Website legt den Fokus auf die schnelle und übersichtliche Information von Cybergefahren und ist daher optimal für unser Projekt.

Was sind eigentlich CVEs?

CVEs, kurz für Common Vulnerabilities and Exposures, also häufige Schwachstellen und Risiken, ist ein Standard für öffentlichen Sicherheitsschwachstellen in Computersystemen. Jede Schwachstelle erhält dabei ihre eigene CVE-Nummer und ist dadurch eindeutig identifizierbar.

Wichtig ist, dass es sich bei CVEs immer um öffentlich bekannte Sicherheitslücken handelt.

Diese CVE kann dann von verschiedenen Anbietern und Informationssystemen benutzt werden. Dadurch können Schwachstellen leichter identifiziert, kategorisiert und priorisiert werden.

Um eine CVE-Nummer zu erhalten, muss eine Schwachstelle mehrere Kriterien erfüllen:

1. Unabhängig behebbar

   Die Schwachstelle kann unabhängig von anderen Bugs behoben werden.

2. Vom betroffenen Anbieter bestätigt oder dokumentiert

   Der von der Schwachstelle betroffene Anbieter bestätigt, dass die Schwachstelle existiert und die Sicherheit beeinflusst. Auch muss sie die Sicherheitslinien des betroffenen Systems verletzen.

3. Auswirkung auf eine Codebasis

   Schwachstellen, die mehr als ein Produkt beeinträchtigen, erhalten separate CVE-Nummern. Bei geteilten Bibliotheken, Protokollen oder Standards erhält die Schwachstelle nur dann eine gemeinsame CVE-Nummer, wenn die Verwendung des geteilten Codes in allen Fällen dieselbe Schwachstelle verursacht. Ansonsten erhält jede betroffene Codebase oder jedes Produkt eine separate CVE-Nummer.

Aufbau CVE

Abbildung 6: Aufbau CVE

Jede CVE besteht aus einem Präfix, einer Jahreszahl, sowie der fortlaufenden Nummerierung.

Das Präfix ist also CVE. Die Jahreszahl entspricht dem Jahr, in dem die Schwachstelle identifiziert und öffentlich bekannt gemacht wurde. Dabei kann es sein, dass eine Lücke zwar schon entdeckt, aber noch nicht mitgeteilt wurde. In dem Fall gibt es auch noch keine CVE für die Schwachstelle. Die Nummer gibt an, die wievielte Sicherheitslücke sie in diesem Jahr ist.

Außerdem enthält jeder CVE-Eintrag zusätzlich noch die Common Weakness Enumeration (CWE)-ID, sowie Referenzen und eine Beschreibung der Sicherheitslücke.

Bei der CWE handelt es sich um eine einheitliche Benennung zur Ermittlung und Definition von Ursachen der identifizierten Sicherheitslücke. Für jeden Schwachstellentyp existiert eine spezifische CWE.

BSI– Datenbezug (Adrian Graf)

Leider stellt das BSI keine API zur Verfügung, sondern nur einen RSS-Feed der die Daten der letzten sieben Tage beinhaltet. Da uns diese Datenlage nicht genügt, war die erste Herausforderung die Daten der BSI-Seite zu erlangen. Im Laufe der Projektarbeit beschäftigten wir uns mit mehreren Ansätzen, die wir verbessert und ausgebaut haben.

V1 - Pures Webscraping

Die erste Idee war die Daten bei dem Start des Programms jedes Mal per Webscraper abzufragen und daraufhin in einem Klassenweiten Array oder Dictionary zu speichern und alle weiteren Operationen mit dem Array durchzuführen. Die Elemente deren Werte wir abfragen wollten haben wir dafür über Xpath ausfindig gemacht und mit einer for schleife mit entsprechendem Index über alle Einträge einer Seite iteriert. Da eine Seite 50 Einträge beinhaltet, wird mit den Indexen 1-50 auf jeden Eintrag zugegriffen und die Daten in einer Variable gespeichert.

Ursprünglich lief dieser Ansatz gut und erfüllte alle Anforderungen, wobei eine Initialisierungszeit von in etwa einer Minute bei jedem Programmstart uns etwas gestört hat.

```
for row_number in range(1, 51):
    xpath_expression_date=f'tr[{row_number}]/td[1]/div/div'
    xpath_expression_title = f'tr[{row_number}]/td[5]/div/div'
    xpath_expression_cvss =f'tr[{row_number}]/td[3]/div/div'
    xpath_expression_programs = f'tr[{row_number}]/td[6]/div/div'
    xpath_expression_cve = f'tr[{row_number}]/td[7]/div/div'
    xpath_expression_status = f'tr[{row_number}]/td[8]/div/div'
    xpath_expression_id = f'tr[{row_number}]/td[4]/div/div'
    date=element.find_element(By.XPATH, xpath_expression_date).get_attribute("textContent").strip()
    if row_number == 1:
        date_first_element=date
    status=element.find_element(By.XPATH, xpath_expression_status).get_attribute("textContent").strip()
    title = element.find_element(By.XPATH, xpath_expression_title).get_attribute("textContent").strip()
    cvss = element.find_element(By.XPATH, xpath_expression_cvss).get_attribute("textContent").strip()
    programs = element.find_element(By.XPATH, xpath_expression_programs).get_attribute("textContent").strip()
    id = element.find_element(By.XPATH, xpath_expression_id).get_attribute("textContent").strip()
```

Abbildung 7: Hauptteil der Scraping Methode

V2 – Webscraping & JSON

Nachdem die Basisfunktionalität abgedeckt war, hatten wir ein weiteres Meeting mit unserem Projektbetreuer. Dabei kam die Idee auf eine Suche über einen längeren Zeitraum zu ermöglichen.

Nach der Implementierung der Suchlogik und Erhöhung des Zeitraums unserer Daten von 30 auf 365 Tagen, erkannten wir das wir ein großes Problem mit der Laufzeit haben. Statt der anfänglichen Minute musste man nun mindestens fünf Minuten warten, bis das Programm die Initialisierung abgeschlossen hatte. Daher kam mir die Idee die Daten des BSI in einer JSON lokal zu speichern. Optional hätte man dafür auch eine lokale Datenbank aufsetzen können, jedoch wäre dies für so simple und einfach strukturierte Daten unverhältnismäßig. In der JSON wurde also Titel, Datum, CVSS-Score, betroffene Programme, CVE-Nummern, URL und Status gespeichert. Somit musste bei jedem Start nur der neueste Eintrag in der JSON mit dem neuesten Eintrag auf der BSI-Seite verglichen werden und gegeben falls nur alle neuen Einträge von der BSI-Seite bezogen werden. Dies verbesserte die Laufzeit drastisch. Bei dem ersten Start des Programms beträgt die Initialisierungszeit zwar immer noch um die fünf Minuten,

da die Daten der letzten 365 Tage bezogen werden, dafür dauert die Initialisierungsphase bei folgenden Starts nur wenige Sekunden. Diese Verbesserung ist nicht nur sehr hilfreich für den Nutzer, sondern auch für uns als Entwickler, da wir viel effizienter testen konnten.

V3 – CVE-Nummern über Webscraping

Im nächsten Schritt kam dann die Idee auf die CVE-Nummer von der BSI-Seite mit den CVE-Nummern der Metasploit Exploits zu matchen. Dafür schrieb ich die Methode get_all_exploits_matching_with_CVE_BSI, die jede CVE-Nummer von jedem BSI-Eintrag mit jeder CVE-Nummer von jedem Metasploit Eintrag abgleicht und die Übereinstimmungen als neue Datenstruktur, die alle wichtigen Informationen beinhaltet, um ein CVE eindeutig mit einem Exploit zu paaren, in einem Array speichert und dieses zurückgibt.

```python
def get_all_exploits_matching_with_CVE_BSI():
    matching_exploits = []
    existing_data = []
    existing_exploits = []
    file_path = "data.json"
    try:
        with open(file_path, "r") as file:
            existing_data = json.load(file)
    except FileNotFoundError:
        return
    file_path = "metasploit.json"
    try:
        with open(file_path, "r") as file:
            existing_exploits = json.load(file)
    except FileNotFoundError:
        return
    for data in existing_data:
        cves = data["cve"].split(", ")
        for exploit in existing_exploits:          # Es wird durch die Exploits iteriert und geprüft, ob ein Exploit zu einer CVE-Nummer der BSI-Seite passt
            for cve in cves:
                if exploit["CVE"] == cve:
                    summary = [data["title"], data["date"], exploit["name"], exploit["disclosure date"], exploit["CVE"], data["programs"], exploit["url"], exploit["description"]]
                    matching_exploits.append(summary)
                    break
    return matching_exploits
```

Abbildung 8: get_all_exploits_matching_with_CVE_BSI) Methode

Nach mehreren Tests und manuellem Abgleichen ist uns aufgefallen, dass die Methode nicht alle Matches zurückgibt. Bei der Schrittweisen Analyse des Programmablaufs stellten wir dann fest, dass uns für verschiedene Einträge der BSI-Seite teilweise CVE-Nummern fehlen. Dies liegt daran, dass auf der BSI-Seite die Spalte mit den CVE-Nummern bei mehr als drei Nummern die Folgenden mit "..." abgekürzt werden. Über das Webscraping haben wir entsprechend nicht die vollständige Auflistung der Nummern bekommen, sondern den literarischen Wert des Feldes, sprich die drei CVE-Nummern gefolgt von "...". Dies stellte ein großes Problem dar, da die CVE-Nummern nur auf einer weiterfolgenden Seite vollständig aufgelistet werden.

Als einzige Lösung blieb nur die zusätzliche Seite für jede abgeschnittene Auflistung zu öffnen.

```python
if "..." in cve:
    cve = ""
    driver.execute_script("window.open('');")
    driver.switch_to.window(driver.window_handles[-1])
    driver.get(url)
    xpath_expression_button=f'/html/body/portal-app/portal/div/portal-module/portal-content-wrapper/main/portal-page/portal-page-sectio
    button = wait.until(EC.presence_of_element_located((By.XPATH, xpath_expression_button)))
    driver.execute_script("arguments[0].scrollIntoView();", button)                    # Es wird zum Button gescrollt
    driver.execute_script("arguments[0].click();", button)                             # Der Button wird geklickt
    xpath_expression_cve = '/html/body/portal-app/portal/div/portal-module/portal-content-wrapper/main/portal-page/portal-page-section/
    cves = wait.until(EC.presence_of_element_located( # Es wird gewartet, bis eie Tabelle mit allen CVEs geladen wurde
        (By.XPATH, xpath_expression_cve)))
    cve=cves.get_attribute("textContent").strip()
    driver.close()
    driver.switch_to.window(driver.window_handles[0])
```

Abbildung 9: IF-Block zur Abfrage der CVE-Nummern

Über JavaScript code, den man den driver ausführen lassen kann, wird das entsprechende Fenster auf dem sich die CVEs befinden geöffnet, der Button gedrückt, um die CVE-Auflistung aufzuklappen und der Inhalt der CVE-Auflistung ausgelesen. Das zusätzliche Öffnen der weiteren Website wirkt sich auf die Laufzeit aus. Nach dieser Ergänzung braucht das Programm nun in etwa 10 Minuten zur Initialisierung, je nach Internetverbindung. Diese Laufzeiterhöhung ist es uns jedoch Wert, da nun die Daten vollständig sind und alle Übereinstimmungen zwischen CVE-Nummern von der BSI-Seite mit den CVE-Nummern der Metasploit Exploits korrekt erkannt werden.

V4 – Die Häufigkeitsanalyse

Nachdem nun das Beziehen der Daten erfolgreich war, kam bei weiterem Brainstorming die Idee auf eine Häufigkeitsanalyse auf den Daten zu erstellen. Dabei haben wir uns auf vier Kategorien festgelegt: fallend, steigend, Dauerbrenner und Newcomer. Die Kategorisierung basiert auf den folgenden Regeln: Ein CVE wird als fallend kategorisiert, wenn das Vorkommen in einem bestimmten Zeitraum zu einem bestimmen Prozentual Satz geringer ist als in einem bestimmten Zeitraum davor. Ein CVE wird als steigend kategorisiert, wenn das Vorkommen in einem bestimmten Zeitraum zu einem bestimmen Prozentual Satz höher ist als in einem bestimmten Zeitraum davor. Ein CVE wird als Dauerbrenner kategorisiert, wenn es in einem bestimmten Zeitraum mindestens zu einer bestimmen Anzahl aufgetreten ist. Ein CVE wird als Newcomer kategorisiert, wenn es in einem bestimmten Zeitraum noch kein einziges Mal aufgetreten ist.

Für die Implementierung dieser Logik habe ich dann die Methode determineTrend geschrieben.

```
for data_row in existing_data:
    if data_row["status"] == "UPDATE":
        continue
    gegebenes_datum = datetime.strptime(data_row["date"], datum_format)
    aktuelles_datum_zeit = datetime.now()
    differenz = aktuelles_datum_zeit - gegebenes_datum
    title = data_row["title"].split(":")[0].strip()
    if differenz.days < tage:
        if title in categories_7_days:
            categories_7_days[title] = categories_7_days[title] + 1
        else:
            categories_7_days[title] = 1
    elif differenz.days < tage * 4:
        if title in categories_28_days:
            categories_28_days[title] = categories_28_days[title] + 1
        else:
            categories_28_days[title] = 1
    else:
        break
```

Abbildung 10: Code zur Häufikgeitsbestimmung

Als Parameter wird der zu analysierende Zeitraum anhand einer Anzahl von Tagen übergeben. Daraufhin wird zuerst über jeden existierenden Eintrag der Daten aus der JSON-Datei iteriert. Dabei wird gefiltert, ob der Eintrag ein Update ist oder tatsächlich ein neu auftretendes CVE. Ist der Eintrag ein neu auftretendes CVE so wird aus dem Titel anhand des ”:” der tatsächliche Name extrahiert. Sprich aus ”Google Chrome: Mehrere Schwachstellen” wird ”Google Chrome” oder aus ”Linux Kernel: Schwachstelle ermöglicht Denial of Service” wird ”Linux Kernel”. Dies ist wichtig damit die Häufigkeit korrekt bestimmt werden kann. Um diese zu berechnen, wird dann die Differenz aus dem heutigen Datum und dem Datum des CVE bestimmt. Liegt dieses in dem übergebenen Zeitraum so wird in einem Dictionary für den Namen als Key das Vorkommen als Value um 1 erhöht oder falls es noch kein Vorkommen gab, der Wert auf 1 gesetzt. Liegt das Datum des CVE nicht in dem übergebenen Zeitraum, so wird überprüft, ob das Datum in dem vierfachen übergebenen Zeitraum liegt. Dies dient dazu, dass das Vorkommen immer anhand eines 1 zu 3 Verhältnis bestimmt wird. Sprich das Auftreten der CVEs

wird mit einem drei Mal längeren Zeitraum verglichen, um bestimmen zu können, wie sich der Trend entwickelt. Liegt das Datum in dem vierfachen Zeitraum, so wird in einem weiteren Dictionary für den Namen als Key das Vorkommen als Value um 1 erhöht oder falls es noch kein Vorkommen gab, der Wert auf 1 gesetzt. Liegt das Datum auch nicht in diesem Zeitraum so sind wir bei einem Datum angekommen, das außerhalb des Auswertungszeitraums liegt und die for-Schleife wird beendet.

```
for key in categories_7_days:
    trend = ""
    if key in categories_28_days:
        if categories_7_days[key] / categories_28_days[key] <= 0.33:
            trend = "fallend"
            fallend = fallend + 1
        elif categories_7_days[key] / categories_28_days[key] >= 1:
            trend = "steigend"
            steigend = steigend + 1
        else:
            gleich = gleich + 1
```

Abbildung 11: Code zur Kategorisierung fallend und steigend

Als nächstes wird für jeden Namen das Verhältnis aus dem Wert aus dem ersten Dictionary geteilt zu dem Wert aus dem zweiten Dictionary berechnet. Ist die Anzahl der Vorkommen in dem übergebenen Zeitraum unter 1/3 so hoch wie bei dem Vergleichszeitraum so wird der aktuelle Eintrag als “fallend” kategorisiert. Ist die Anzahl der Vorkommen in dem übergebenen Zeitraum mindestens genau so hoch wie bei dem Vergleichszeitraum so wird der aktuelle Eintrag als “steigend” kategorisiert.

Diese Schwellwerte haben wir durch ausführliches Testen mit verschiedenen Werten und manuellem Abgleichen, als die am besten passenden ermittelt.

```
count = 0
for entry in existing_data:
    gegebenes_datum = datetime.strptime(entry["date"], datum_format)
    aktuelles_datum_zeit = datetime.now()
    differenz = aktuelles_datum_zeit - gegebenes_datum
    if differenz.days < 180:
        title = entry["title"].split(":")[0]
        if key == title:
            count += 1
            if count > 1:
                break
    else:
        break
if count == 1:
    trend = "Newcomer"
    neu = neu + 1
```

Abbildung 12: Code zur Kategorisierung Newcomer

Im nächsten Schritt wird für den aktuellen Namen, sprich key abgeglichen, ob dieser in den letzten 180 Tagen schon mehr als 1-mal aufgetreten ist. Ist sein aktuelles Vorkommen das erste der letzten 180 Tage so wird sein Trend überschrieben mit “Newcomer”. Ist er schon häufiger aufgetreten so behält er seinen vorherigen Trend.

```python
for data_row in existing_data:
    if key in data_row["title"]:
        sum = 0
        if key in categories_28_days:
            sum += categories_28_days[key]
        if key in categories_7_days:
            sum += categories_7_days[key]
        if sum > tage:
            trend = "Dauerbrenner"
        data_row["trend"] = trend
```

Abbildung 13: Code zur Kategorisierung Dauerbrenner

Zuletzt wird überprüft, ob der aktuelle Name, sprich key, im übergebenen und im Vergleichszeitraum insgesamt häufiger als die Anzahl der übergebenen Tage aufgetreten ist. Ist dies der Fall so wird der Trend auf “Dauerbrenner” gesetzt. Anschließend ist die Analyse abgeschlossen und der Trend wird in die Daten geschrieben und lokal in der JSON persistiert.

Das Metasploit-Projekt (Selim Karali)

Die Idee eines Frühwarnsystems für Cybergefahren ist es vor möglichen Gefahren, die durch Sicherheitslücken zustande kommen zu warnen, bevor diese von offiziellen Institutionen bestätigt wurden. Wurde eine Sicherheitslücke, die möglicherweise eine größere Gefahr darstellt, von Seiten einer offiziellen Institution bestätigt, kann man davon ausgehen, dass die Gefahr bereits seit längerem bekannt und Zeit verstrichen ist, bis diese bestätigt und bekanntgegeben wurde und möglicherweise wurde sie bis zur Bekanntgabe bereits ausgenutzt. Somit kann man das Ausnutzen einer Sicherheitslücke und die damit einhergehende Gefahr als betroffenes Unternehmen oder Person frühzeitig verhindern, um so früher man davon erfährt. Um offiziellen Institutionen einen Schritt voraus zu sein haben wir uns im Rahmen der Projektarbeit das Metasploit-Framework zunutze gemacht.

Beim Metasploit-Projekt handelt es sich um ein Werkzeug, welches Informationen über Sicherheitslücken bereitstellt. Metasploit gehört zum Unternehmen Rapid7. Rapid7 stellt im Rahmen des Metasploit-Projektes das Metasploit-Framework zur Verfügung, welches ein Open-Source Tool darstellt. Der eigentliche Zweck des Frameworks ist das sogenannte ethische Hacken, bei dem versucht wird, Sicherheitslücken im hauseigenen System zu entdecken und zu beheben. Das ausfindig machen von Sicherheitslücken ist beim Framework möglich, da sogenannte “Exploits” bereitgestellt werden (7).

Metasploit-Framework

Exploits

Bei Exploits handelt es sich um Code, der sich Zugriff auf ein bestimmtes System verschafft, durch das Ausnutzen einer Sicherheitslücke (8). Metasploit-Framework stellt eine Datenbank, mit mehr als 2000 solcher Exploits zur Verfügung. Um Metasploit-Framework hat sich eine Community gebildet, welche neue Exploits entdeckt und Metasploit-Nutzern zugänglich macht, dadurch kann man mithilfe von Metasploit frühzeitig von neuen Sicherheitslücken, die möglicherweise eine Gefahr für das eigene System darstellen, erfahren. Neue Exploits kann jeder Nutzer dem Metasploit-Framework im Rahmen des GitHub-Projektes hinzufügen (9).

Nutzung des Metasploit-Frameworks

Metasploit-Framework ist ein Command-Line Tool, welches die Möglichkeit bietet, alle möglichen Exploits und die dazugehörigen Informationen in einer lokalen Datenbank einzusehen. Die Exploits können auch mithilfe des Frameworks ausgeführt werden.

Die für die Projektarbeit relevante Funktion des Frameworks war jedoch das Abrufen aller verfügbaren Exploits. Einem Exploit ist ein Veröffentlichungsdatum, eine Beschreibung der ausgenutzten Sicherheitslücke, und in vielen Fällen sogar ein CVE, zugeordnet.

Einbindung des Metasploit-Frameworks in die Projektarbeit

Die Problemstellung während den Anfängen der Projektarbeit war, wie anfangs des Kapitels schon erwähnt:
Wie können wir bewerten, ob ein CVE wirklich bedrohlich ist und können wir möglicherweise sogar vor der Veröffentlichung eines CVEs über eine Sicherheitslücke berichten?

Diese Problemstellung konnte durch das Framework gelöst werden, indem zum einen überprüft wurde, ob es zu den vom BSI veröffentlichten CVEs einen Exploit gibt, wenn ja stellte dieses CVE eine größere Gefahr dar. Zum anderen konnten wir über bereits in Programmen entdeckte Sicherheitslücken berichten, bevor ein CVE dazu veröffentlicht wurde.

Einbindung des Metasploit-Frameworks in unser Programm

Die bereits erwähnte lokale Datenbank des Metasploit-Frameworks und die darin enthaltenen Exploits werden aktualisiert, durch das manuelle Updaten des Frameworks. Es gibt jedoch eine kostenpflichtige Metasploit Pro Version, die weitere Funktionen anbietet, u.a. kann man jederzeit die aktuellen Exploits in einer von Metasploit gehosteten Datenbank einsehen und es wird eine API mit erweiterten Funktionen bereitgestellt, um die Datenbank und die dort enthaltenen Informationen in einem Programm zu nutzen. Dies war auf den ersten Blick mit der kostenlosen Version nicht möglich.

Die Funktionen, die wir manuell realisieren mussten, sind:

1. Das Updaten der lokalen Metasploit-Datenbank, sodass immer die neusten Exploits zur Verfügung stehen und dem Nutzer präsentiert werden können.
2. Das Abrufen und aufbereiten aller Exploits und der dazugehörigen Informationen

Das Updaten des Metasploit-Frameworks

Zuallererst beschäftigte ich mich damit, wie das Metasploit-Framework automatisiert aktualisiert werden kann, sobald ein Update zur Verfügung steht. Ohne automatisiertes Updaten, wäre das Metasploit-Framework nutzlos, da der Nutzer und unser Programm, zu jederzeit die neuesten Exploits benötigte. Das manuelle Aktualisieren wird durchgeführt, indem man in der Kommandozeile im Ordner metasploit-framework/bin den Befehl “msfupdate” ausführt. Nun wird ein minutenlanger Prozess gestartet, der überprüft, ob eine neue Version zur Verfügung steht und ein Update startet, falls dies zutrifft. Die Überprüfung auf ein neues Update dauert jedoch sehr lange und es ist nicht zumutbar, den Nutzer bei jedem Start unseres Programms diese Überprüfung abzuwarten.

Deshalb suchte ich nach einer Möglichkeit, “msfupdate” nur dann auszuführen, sobald ein Update zur Verfügung steht. Die aktuelle Version des installierten Frameworks konnte man anhand des Befehls “msfconsole -v” herausfinden. Die aktuell neuesten verfügbaren Versionen des Frameworks konnte man auf folgender Seite einsehen und herunterladen: https://windows.metasploit.com.

Die Beschreibung der Versionen beinhaltet die Versionsnummer und einen Zeitstempel. Mit dem Link https://windows.metasploit.com/LATEST kann man eine Seite ansteuern, welche auf einer sehr simple gehaltenen HTML-Seite die neuste Version dokumentiert. Diesen Inhalt kann man per Python-Skript durch Nutzung des Paketes “requests” abrufen und mit dem Paket “BeautifulSoup” parsen.

Somit konnte man einen String extrahieren, welcher so aussah:

metasploit-framework-6.3.58+20240225113416-1rapid7-1-x64.msi

Der gesamte Update-Prozess beinhaltete somit diese Schritte:

1. Herausfinden welches die neueste verfügbare Version ist
2. Herausfinden welche Version aktuell installiert ist
3. Starten eines Updates, falls die aktuell installierte Version veraltet, ist

Schritt 1:

```python
@staticmethod
def get_latest_version():
    """
    Die Methode get_latest_version() gibt die neueste Version von Metasploit zurück.
    """
    url = 'https://windows.metasploit.com/LATEST'
    response = requests.get(url)
    html = BeautifulSoup(response.content, 'html.parser')  # Die Version wird aus dem HTML-Code extrahiert
    version_text = html.get_text()
    version_text = version_text.split('-')[2]
    version_text = version_text.split('+')[0]      You, 2 months ago • parallelität dies das
    return version_text
```

Abbildung 14: Abrufen der Versionsnummer der neuesten verfügbaren Metasploit-Framework Version

In der Methode ”get_latest_version()” (Abb. 14) wurde der beschriebene Vorgang zum in Erfahrung bringen der neuesten Version umgesetzt.

Es wurde ein String wie ”metasploit-framework-6.3.58+20240225113416-1rapid7-1-x64.msi” extrahiert und z.B. die Versionsnummer “6.3.58” dem String entnommen und zurückgegeben.

Schritt 2:

```python
@staticmethod
def get_current_version():
    """
    Die Methode get_current_version() gibt die aktuelle Version von Metasploit zurück.
    """
    process = subprocess.Popen("msfconsole -v", stdout=subprocess.PIPE, stderr=subprocess.PIPE, shell=True)
    stdout, stderr = process.communicate()
    lines = stderr.decode("utf-8").splitlines()
    pattern = r'Framework Version: (\d+\.\d+\.\d+)'      You, seconds ago • Uncommitted changes
    version_number = "Version not found"
    for line in lines:
        match = re.match(pattern, line)
        if match:
            version_number = match.group(1)
            break
        else:
            version_number = "Version not found"
    return version_number
```

Abbildung 15: Abrufen der Versionsnummer des installierten Metasploit-Frameworks

In der Methode ”get_current_version” (Abb.15) wird in der Konsole der Befehl ”msfconsole -v” ausgeführt.

Zugriff auf die Konsole kann man in Python mit dem Package “subprocess” bekommen

Das Ergebnis ist folgender String:

“C:/metasploit-framework/embedded/lib/ruby/gems/3.0.0/gems/rex-core-0.1.31/lib/rex/compat.rb:381: warning: Win32API is deprecated after Ruby 1.9.1; use fiddle directly instead

Framework Version: 6.3.58-dev-3759346f10a6b41b2e527be2b3f31f9997ec58d0"

Nach “Framework Version:” befindet sich die Versionsnummer des aktuell installierten Frameworks, welche in der Methode “get_current_version” extrahiert wurde.

Schritt 3:

```python
def update_metasploit_available():
    """
    Die Methode update_metasploit_available() gibt True zurück, wenn ein Update für Metasploit verfügbar ist, ansonsten False.
    """
    current_version = Metasploit.get_current_version()
    latest_version = Metasploit.get_latest_version()      You, 2 months ago • parallelität dies das
    if current_version < latest_version:                                    # Es wird geprüft, ob ein Update verfügbar ist
        return True
    print("No update available")
    return False

@staticmethod
def update_metasploit():
    """
    Die Methode update_metasploit() startet den Update-Prozess von Metasploit.
    Durch ein Update kommen neue Exploits hinzu und bestehende Exploits werden verbessert.
    """
    subprocess.Popen("msfupdate", stdout=subprocess.PIPE, stderr=subprocess.PIPE, shell=True) # Der Update-Prozess wird in der Konsole gestartet
    if os.path.exists('metasploit.json'):
        os.remove('metasploit.json')
```

Abbildung 16: Updaten des Metasploit-Frameworks

In der Methode ”update_metasploit_available()” (Abb.16) wird dann letztendlich überprüft, ob die aktuell installierte Version kleiner ist als die neueste zur Verfügung stehende Version, was bedeuten würde das ein Update zur Verfügung steht. Wenn ein Update zur Verfügung steht, wird dann in ”update_metasploit()” über die Kommandozeile mit dem Befehl ”msfupdate” das Aktualisieren gestartet.

Abrufen der Exploits und aufbereiten der Informationen

Das Abrufen aller Exploits gestaltete sich anfangs schwieriger, da wir nicht die Möglichkeit hatten die kostenpflichtige API zu nutzen, die diese Funktionen bereitstellte.

Doch nach ausführlicher Recherche bin ich, auf das Python-Package “pymetasploit3” gestoßen. Dieses erlaubt sich mit einem lokal gestarteten Metasploit-Server zu verbinden, um dann vom Package bereitgestellte Funktionen nutzen zu können. Die Funktionen sind u.a. das Abrufen und Ausführen von Exploits Um einen Metasploit-Server lokal zu starten, muss man in der Konsole den Befehl “msfrpcd -P Passwort” ausführen. Da dies nicht vom Nutzer der App erwartet werden kann, wurde dies im Programm automatisiert.

```python
@staticmethod
def start_local_metasploit_server():
    """
    Die Methode start_local_metasploit_server() startet einen lokalen Metasploit-Server.
    """
    subprocess.Popen("msfrpcd -P Passwort", shell=True)
    connected = False
    while not connected:
        global client
        try:
            client = MsfRpcClient('Passwort', ssl=True)
            connected = True
        except:
            sleep(1)
            continue
```

Abbildung 17: Methode zum Starten des lokalen Metasploit-Servers

In der Methode “start_local_metasploit_server()” (Abb.17) wird über die Konsole ein Server gestartet und ein Client erzeugt, der sich verbindet. Die Client-Funktionalität wird vom “pymetasploit3”- Package bereitgestellt.

Nachdem nun die Verbindung hergestellt wurde und wir einen Client haben, der mit dem Server verbunden ist, können nun die Exploits abgerufen werden.

```python
@staticmethod
def make_json():
    """
    Die Methode make_json() erstellt eine JSON-Datei, die alle aktuell vorhandendenen Exploits von Metasploit enthält.
    Die Exploits werden von der Netasploit-Community bereitgestellt und sind nicht immer vollständig und konsistent dokumentiert.
    Dadurch müssen beim parsen der Informationen viele Ausnahmenbehandlungen vorgenommen werden.
    """
    exploits_dictionary = []
    exploits = client.modules.exploits                    # Es werden alle Exploits von Metasploit abgerufen
    exploit_list = list(exploits)
```

Abbildung 18: Methode make_json() welche Exploit-Informationen abruft und verarbeitet

Mit ”client.modules.exploits", können alle vorhandenen Exploits abgerufen werden. Diese werden anschließend in eine Liste gepackt. Nun ist es möglich, durch die Liste mit den Exploits zu iterieren und von jedem Exploit die ihm zugeordneten Informationen abzurufen. In der Abbildung 19, wird z.B. die URL abgerufen, die dem Exploit hinzugefügt ist.

```
for exploit in exploit_list:
    try:
        info = client.modules.use('exploit',exploit)
    except TypeError as e:
        continue
    url = "No URL available"
    cve = "No CVE assigned"
    if len(info.references)>1:
        if info.references[1][1].startswith("http"):
            url = info.references[1][1]
        else:
            for reference in info.references:
                if reference[0] == "URL":
                    url = reference[1]
                    break
```

Abbildung 19: Filtern der dem Exploit zugeordneten URL

Jedoch muss man sagen, dass die Exploits nicht gut dokumentiert sind, aufgrund dessen, dass die Exploits von Leuten aus der Community zusammengestellt werden. Somit konnte es sein, dass an Stelle einer URL, die CVE-Nummer angegeben war.

Dementsprechend hat die Methode “make_json()” welche alle Exploits abruft und die Informationen des Exploits verarbeitet, sehr viele Ausnahmebehandlungen.

Den Exploits wurden folgende Informationen entnommen:

```
exploits_dictionary.append({
    "name": info.name,
    "CVE": cve,
    "disclosure date": date,
    "ranking": info.rank,
    "url": url,
    "description": info.description
})
```

Abbildung 20: Aufbau eines JSON-Eintrags

Diese Informationen wurden anschließend, lokal in einer JSON-Datei abgelegt um beim Start der App beschleunigt alle Informationen zu den Exploits schnell abrufen zu können, da das Abrufen und Verarbeiten der Exploits vom lokalen Metasploit-Server, viel Zeit in Anspruch nimmt. Die JSON-Datei wird auf dem Computer des Nutzers, bei der ersten Nutzung der App erstellt und gelöscht und neu erstellt, nachdem ein Update des Metasploit-Frameworks stattgefunden hat, da dann neue Exploits hinzukommen und alte aktualisiert werden.

Nachdem die JSON-Datei erstellt wurde, wird der Metasploit-Server gestoppt (Abb. 21).

```python
@staticmethod
def stop_metasploit_server():
    """
    Die Methode stop_metasploit_client() stoppt den lokalen Metasploit-Server.
    """
    process = subprocess.Popen("taskkill /F /IM ruby.exe", shell=True) # Die M
    process.wait()                                                      # Es wi
```

Abbildung 21: Stoppen des lokalen Metasploit-Servers

Wenn man den Server manuell in der Kommandozeile beenden würde, würde man z.B. STRG + C drücken. Da der Server jedoch über den Code gestartet wird, ist dies nicht möglich. Somit habe ich im Task-Manager den Prozess ausfindig gemacht, der für den Metasploit-Server verantwortlich ist. Dies war in diesem Fall der "ruby.exe"- Prozess. Anfangs hatte ich den Server nicht gestoppt, stellte jedoch schnell fest, dass es zu Problemen kommen kann, wenn man unsere Anwendung mehrmals hintereinander startet, da ein Server bereits gestartet war und es nicht möglich war einen zweiten zu starten und sich damit zu verbinden. Mehrere Server laufen zu lassen wäre außerdem keine Ressourcen-schonende Vorangehens weise (Abb.21).

Zwei weitere Funktionen, die in der Anwendung gebraucht wurden und die ich in meiner Metasploit-Klasse realisiert habe sind "search_by_cve_exploits()" und "search_by_term_exploits". Diese Funktionen werden von der GUI aufgerufen, welche auch Benutzereingaben an diese weiterleitet, um Exploits ausfindig zu machen, die einem bestimmten CVE zugeordnet sind oder einen bestimmten Suchbegriff enthalten.

Hauptprogramm mit GUI

Damit der Nutzer unseren Code nicht in einer IDE oder in der Kommandozeile ausführen zu müssen, wurde ein Executable erzeugt. Dafür wurde das Python-Package „pyinstaller" verwendet.

Start des Programms (Selim Karali)

Die Schritte, die im Programmcode ausgeführt werden, sobald der Nutzer „Frühwarnsystem.exe" startet, sind folgende:

1. Die Ausführung des Programms beginnt in der Datei "Data_Table_App.py".
2. Es wird ein "root"- Fenster erzeugt, welches das sogenannte Hauptfenster ist. Der Nutzer bekommt ein Fenster zu sehen, mit der Information das nun auf Updates überprüft wird. (Funktion: __init__(self, root))
3. Zum Nutzen unseres Programms muss, wie anfangs erwähnt, Metasploit-Framework installiert sein und dem Programm muss bekannt sein, wo das Framework im Speicher abgelegt ist, damit es z.B. den Metasploit-Server starten kann. Um den Pfad zum Metasploit-Framework Ordner nicht immer wieder neu abzufragen, wird dieser einmal initial abgefragt und in einer nicht sichtbaren JSON-Datei im Verzeichnis, in der das Executable ausgeführt wurde, abgelegt. Sollte der Pfad sich mal ändern und dadurch der in der JSON-Datei abgelegte Pfad falsch sein, dann wird dies erkannt und der Pfad wird neu abgefragt. (Funktion: set_metasploit_path(self))

4. Nun wird die Methode "start_App()" aufgerufen, welche überprüft, ob es eine neue Metasploit Version gibt und ob neue Informationen auf der BSI-Seite vorhanden sind.

4.1. Dafür werden für Metasploit, als auch für die BSI-Daten, zwei Kindprozesse gestartet (process_metasploit_updates, process_get_data_BSI).
Ich habe das Überprüfen auf Updates parallelisiert, da das serielle Ausführen, enorm viel Zeit in Anspruch genommen hat, zudem ist die GUI eingefroren, da in der Zeit, in der die Funktionen ausgeführt wurden, das Hauptfenster nicht aktualisiert werden konnte. Für das Parallelisieren wurde das "multiprocess" - Package von Python verwendet. Bei Beginn der parallelen Prozesse wird dem Nutzer weiterhin "Überprüfen auf Updates" angezeigt.

4.2. Die parallelen Prozesse, führen Funktionen in der Klasse "Metasploit" und im Modul "webscraping" aus, damit diese frühzeitig mitteilen können (aus einer anderen Klasse heraus), ob ein Update zur Verfügung steht, werden zwei sogenannte "Queues" initialisiert, in die die Prozesse Ergebnisse packen können, um diese dann im Vaterprozess abzurufen. In der Zwischenzeit, solange nicht beide Prozesse mitgeteilt haben, ob ein Update zur Verfügung steht oder nicht, wird im Hauptprozess das Hauptfenster aktualisiert, damit dieses nicht einfriert und der Nutzer jederzeit die Anwendung beenden kann. Wird nun von einem der Prozesse mitgeteilt, dass ein Update zur Verfügung steht, bekommt der Nutzer ein Fenster eingeblendet, welches ihm mitteilt, dass nun ein Update durchgeführt wird und dies eine gewisse Zeit in Anspruch nehmen kann.

4.3. Nun wird wieder Hauptfenster aktualisiert, um dem Nutzer Interaktionen zu ermöglichen. Sobald die beiden Prozesse beendet sind und dem Vaterprozess wieder beigetreten sind, hat die Anwendung alle Daten, die sie benötigt, lokal in JSON-Dateien abgelegt und kann nun angezeigt werden.

5. Implementiert wurde zudem eine Methode "before_closing", die durch das Package Tkinter an das Event des Beendens der Anwendung gebunden werden konnte und ausgeführt wird, sobald die Anwendung beendet wird. Sie garantiert, dass alle im Hintergrund laufenden Prozesse vollständig terminieren, u.a. die zwei erwähnten Kindsprozesse, als auch der Metasploit-Server.

Visuelle Darstellung (Maja Magschok)

Zuletzt muss unser Frühwarnsystem noch visuell überzeugen. Dabei ist es vor allem wichtig, dass alles übersichtlich dargestellt wird, sodass alle wichtigen Informationen direkt erfasst werden können.

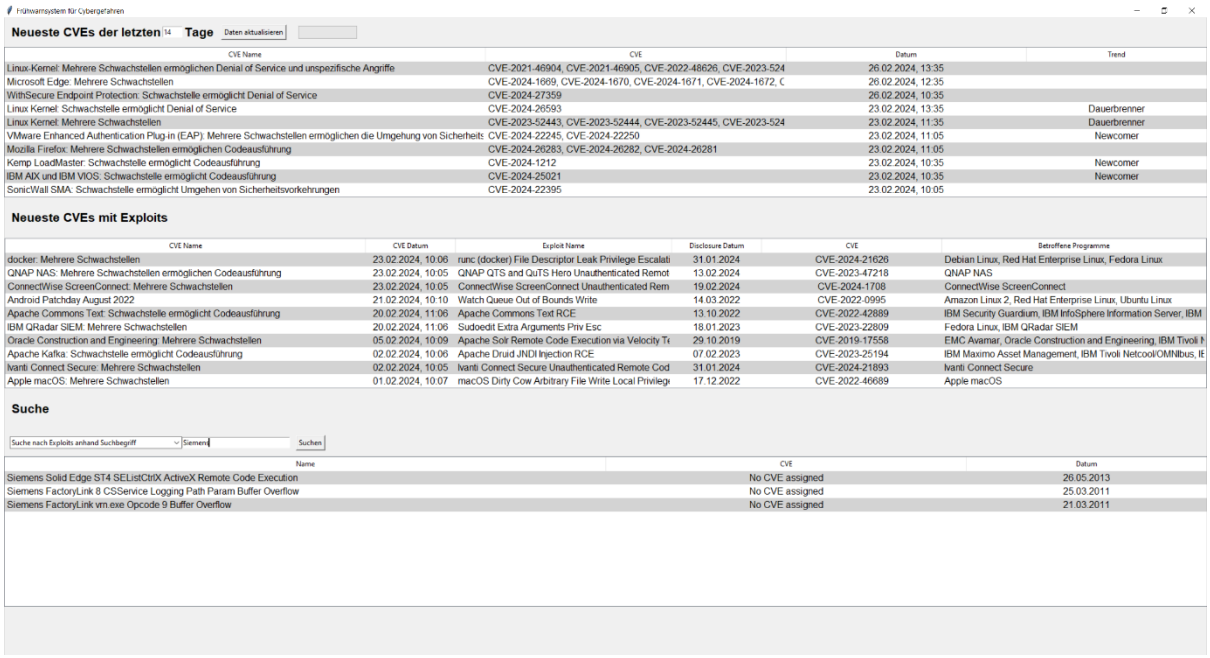


Abbildung 22: Die GUI der Anwendung

Hier dargestellt ist unser fertiges Frühwarnsystem. Das fertige Ergebnis zu sehen, hilft dabei den Code für den Aufbau nachvollziehen zu können.

```
root.title("Frühwarnsystem für Cybergefahren")
root.state("zoomed")
screen_width = root.winfo_screenwidth()
screen_height = root.winfo_screenheight()
cell_frame = tk.Frame(root)
cell_frame.grid(row=0, column=0, padx=10, pady=10, sticky="w")
self.font_size=int(screen_height/1000*11)
self.header_label = tk.Label(cell_frame, text="Neueste CVEs der letzten", font=("Helvetica", 16, "bold"))
self.header_label.grid(row=0,column=0)

self.entry_days = tk.Entry(cell_frame, width=5)
self.entry_days.grid(row=0,column=1)
self.entry_days.insert(0, "14")
self.entry_days.bind('<Return>', lambda event: self.collect_data())
```

Abbildung 23: Grundsätzlicher GUI-Aufbau

Hier ist beispielsweise die Überschrift ”Neuste CVEs der letzten ... Tage” in Code-Format dargestellt, sowie der grundsätzliche Aufbau der Seite.

Die 14 Tage, die hier als Beispiel gegeben sind, haben nach ein paar Testversuchen für uns stimmige Ergebnisse geliefert, können jedoch vom Nutzer der Seite später auch angepasst werden.

```
# BSI und Metasploit Matching
self.header_matches = tk.Label(root, text="Neueste CVEs mit Exploits", font=("Helvetica", 16, "bold"))
self.header_matches.grid(row=6,column=0,padx=10, pady=20, sticky="w")
self.tree_matches = ttk.Treeview(root, columns=("CVE Name","CVE Datum", "Exploit Name", "Disclosure Datum","CVE","Betroffene Programme","url","description"),
                        show="headings")
self.tree_matches.grid(row=7,column=0,rowspan=5,columnspan=3)
self.tree_matches.heading("Exploit Name", text="Exploit Name")
self.tree_matches.column("Exploit Name", anchor="w", width=int(screen_width*0.18))
self.tree_matches.heading("Disclosure Datum", text="Disclosure Datum")
self.tree_matches.column("Disclosure Datum", anchor="center", width=int(screen_width*0.075))
self.tree_matches.heading("CVE Name", text="CVE Name")
self.tree_matches.column("CVE Name", anchor="w", width=int(screen_width*0.3))
self.tree_matches.heading("CVE Datum", text="CVE Datum")
self.tree_matches.column("CVE Datum", anchor="center", width=int(screen_width*0.075))
self.tree_matches.heading("CVE", text="CVE")
self.tree_matches.column("CVE", anchor="center", width=int(screen_width*0.15))
self.tree_matches.heading("Betroffene Programme", text="Betroffene Programme")
self.tree_matches.column("Betroffene Programme", anchor="w", width=int(screen_width*0.22))
self.tree_matches.heading("url", text="url")
self.tree_matches.column("url", anchor="center", width=0,minwidth=0)
self.tree_matches.heading("description", text="description")
self.tree_matches.column("description", anchor="center", width=0,minwidth=0)
self.y_scrollbar_exploits = ttk.Scrollbar(root, orient="vertical", command=self.tree_matches.yview)
self.tree_matches.configure(yscrollcommand=self.y_scrollbar_exploits.set)
self.tree_matches.bind("<Double-1>", self.double_click_matches)
self.show_data_tree_matches()
```

Abbildung 24: CVE und Exploit Matching GUI

Nachdem wir die CVEs selbst in den Reihen mit den verschiedenen Informationen darstellen, geht es wie in Abbildung 24 gezeigt um das BSI und Metasploit Matching.

Ähnlich wie bei der BSI-Website, stellen wir auch bei uns zum Beispiel die CVE-Nummer und die betroffenen Produkte dar.

In Abbildung 24 sind die verschiedenen Kategorien jeder Schwachstelle dargestellt, wie diese bestimmt werden. Diese werden mithilfe des Treeview-Widgets erstellt, sodass die Daten übersichtlich in tabellarischer Form angezeigt werden können.

Dazu wird zunächst ein Treeview erstellt, welches Tabellendaten anzeigen kann. Anschließend werden Scrollbar und die eben erwähnten Spaltenkategorien hinzugefügt.

Durch einen Doppelklick (Funktion double_click_matches) wird show_data_tree_matches aufgerufen, welches die letztendlichen Daten in den Treeview lädt.

```
# Suche
self.header_search = tk.Label(root, text="Suche", font=("Helvetica", 16, "bold"))
self.header_search.grid(row=12,column=0,padx=10, pady=20, sticky="w")
cell_frame2 = tk.Frame(root)
cell_frame2.grid(row=13, column=0, padx=10, pady=10, sticky="w")
options = ["Suche nach Exploits anhand Suchbegriff","Suche nach Exploits anhand von CVE-Nummer", "Suche nach CVEs anhand Suchbegriff"]
self.selected_option = tk.StringVar()
self.combobox = ttk.Combobox(cell_frame2, textvariable=self.selected_option, values=options, width=45, state="readonly")
self.combobox.grid(row=0,column=0)
self.combobox.set("Suche nach Exploits anhand Suchbegriff")
self.combobox.bind('<<ComboboxSelected>>', self.on_combobox_selected)
self.entry_term = tk.Entry(cell_frame2, width=30,fg="grey")
self.entry_term.insert(0, "Suchbegriff eingeben")
self.entry_term.bind('<FocusIn>', self.clear_entry)
self.entry_term.bind('<FocusOut>', self.insert_placeholder)
self.entry_term.bind('<Return>', lambda event: self.show_data_tree_search(self.entry_term.get()))
self.entry_term.grid(row=0,column=1)
self.search_button = tk.Button(cell_frame2, text="Suchen", command=lambda: self.show_data_tree_search(self.entry_term.get()))
self.search_button.grid(row=0,column=2,padx=10)
self.tree_search = ttk.Treeview(root, columns=("Name", "CVE","Datum","url","description"), show="headings")
self.tree_search.grid(row=14,column=0,rowspan=3,columnspan=3)
```

Abbildung 25: Suche GUI

In Abbildung 25 ist ein Teilcode unserer Search-GUI dargestellt. Nachdem ein Label für die Überschrift erstellt wird, werden Frames für Zellen kreiert. Ein Frame (cell_frame2) wird erstellt und in der GUI an der Position (row=13, column=0) platziert. Dieser Frame wird verwendet, um Widgets (z.B., Combobox, Entry, Button) zu gruppieren.

Um den Suchtyp auszuwählen, wird eine Combox erstellt. Die Optionen werden aus der Liste options genommen. Die Auswahl wird in der Variable self.selected_option gespeichert. Die Combobox wird in der GUI angezeigt und standardmäßig auf die erste Option gesetzt. Ein Ereignis (<<ComboboxSelected>>) wird gebunden, um auf die Auswahl in der Combobox zu reagieren.

Ein Eingabefeld wird erstellt, das einen vordefinierten Text ("Suchbegriff eingeben") als Platzhalter enthält. Es gibt Ereignisbindungen für den Fokus, um den Platzhalter zu entfernen bzw. wieder einzufügen, und für die Eingabetaste (<Return>), um die Suchfunktion aufzurufen.

Anschließend verwenden wir abermals wie bei Abbildung 23 bereits beschrieben, die Treeview um das Ganze in Tabellenform zu bringen.

GUI – Einfügen der Daten (Adrian Graf)

Der letzte Schritt, um das Programm zu vervollständigen ist die Daten in der GUI anzuzeigen. Entsprechend der Aufteilung der GUI in drei Teile, habe ich dafür drei Methoden geschrieben.

show_data_tree_BSI(tage)

Diese Methoden wird von der Methode prepareData() mit den vom Nutzer angegebenen Tagen aufgerufen. PrepareData() überprüft hierbei im Vorfeld ob die angegebenen Tage im erlaubten Zeitraum liegen und ruft dann gegebenenfalls in einem neuen Prozess die Methode determineTrend(tage) aus dem webscraping Modul auf, um die Trends für den neuen Zeitraum neu zu berechnen. Anschließend sind die Vorbereitungen für das Anzeigen der Daten des BSI abgeschlossen und die Methode show_data_tree_BSI(tage) wird aufgerufen.

Diese lädt die Daten aus der JSON-Datei und löscht die bisherigen Einträge aus der GUI-Tabelle.

```
for data_row in existing_data:
    if data_row["status"] == "UPDATE":
        continue
    gegebenes_datum = datetime.strptime(data_row["date"], datum_format)
    differenz = aktuelles_datum_zeit - gegebenes_datum
    if differenz.days < tage:
        if row_id % 2 == 0:
            tags = ('even',)
        else:
            tags = ('odd',)
        if wsc.exploit_for_CVE_exists(data_row["cve"]):
            tags = ('exploit',)
        data_list = [data_row["title"], data_row["cve"], data_row["date"], data_row["trend"],data_row["url"]]
        self.tree_BSI.insert("", "end", values=data_list, tags=tags)
        self.progress_bar["value"] = self.progress_bar["value"] + (1 / len(existing_data))
    else:
        break
    row_id += 1
```

Abbildung 26: Hauptteil show_data_tree_BSI Methode

Daraufhin wird wie in der Abbildung zu sehen über jeden der geladenen Einträge iteriert und zuerst überprüft, ob der aktuelle Eintrag in der übergebenen Zeitspanne liegt. Ist dies der Fall so werden die Tags für die Färbung der Zeile gesetzt. Es wird immer abgewechselt zwischen even(hellgrau) und

odd(weiß). Liefert der Aufruf der Methode exploit_for_CVE_exists() True zurück, sprich es gibt für einen der CVEs des aktuellen Eintrags ein Exploit, so wird der tag auf “exploit” gesetzt und die Zeile somit rot gefärbt.

Zuletzt werden die benötigten Daten des aktuellen Eintrags in einer neuen Datenstruktur gespeichert und diese dann in die “tree_BSI” Tabelle geladen, sowie die progressbar aktualisiert.

show_data_tree_matches()

```
matching_exploits = wsc.get_all_exploits_matching_with_CVE_BSI()
row_id = 0
for exploit_and_data_BSI in matching_exploits:
    if row_id % 2 == 0:
        tags = ('even',)
    else:
        tags = ('odd',)
    data_list = [exploit_and_data_BSI[0], exploit_and_data_BSI[1], exploit_and_data_BSI[2], exploit_and_data_BSI[3],\
    exploit_and_data_BSI[4], exploit_and_data_BSI[5], exploit_and_data_BSI[6], exploit_and_data_BSI[7]]
    self.tree_matches.insert("", "end", values=data_list, tags=tags)
    row_id += 1
font = ("", self.font_size)
self.tree_matches.tag_configure('odd', background='white', font=font)
self.tree_matches.tag_configure('even', background='lightgray', font=font)
```

Abbildung 27: Hauptteil show_data_tree_matches Methode

Die Methode show_data_tree_matches() bekommt zu Beginn die Paare von CVEs und entsprechendem Exploit durch den Aufruf get_all_exploits_Matching_with_CVE_BSI().

Daraufhin wird über jedes Paar iteriert, die Spaltenfarbe über das tag definiert, eine neue Datenstruktur mit den benötigten Daten erstellt und diese Daten in die Tabelle “tree_matches” eingefügt.

show_data_tree_search(search_term)

```
type = self.selected_option.get()
data = []
if type == "Suche nach Exploits anhand Suchbegriff":
    data = msp.search_by_term_exploits(search_term)
elif type == "Suche nach CVEs anhand Suchbegriff":
    data = wsc.search_by_term_CVE(search_term)
else:
    data = msp.search_by_CVE(search_term)
if len(data) == 0:
    messagebox.showwarning("Achtung", "Keine Einträge für diesen Suchbegriff gefunden")
for i in self.tree_search.get_children():
    self.tree_search.delete(i)
row_id = 0
for entry in data:
    if row_id % 2 == 0:
        tags = ('even',)
    else:
        tags = ('odd',)
    if type == "Suche nach Exploits anhand Suchbegriff" or type == "Suche nach Exploits anhand von CVE-Nummer":
        data_list = [entry["name"], entry["CVE"], entry["disclosure date"], entry["url"], entry["description"]]
    else:
        data_list = [entry["title"], entry["cve"], entry["date"], entry["url"]]
    self.tree_search.insert("", "end", values=data_list, tags=tags)
    row_id += 1
font = ("", self.font_size)
```

Abbildung 28: Hauptteil show_data_tree_search Methode

Die Methode show_data_tree_search(search_term) kann die darzustellenden Daten von drei verschiedenen Methoden beziehen, je nach Auswahl des Nutzers. Hat der Nutzer in der GUI eingestellt, dass

Exploits anhand von Suchbegriffen gesucht werden sollen so wird die Methode search_by_term_exploits(search_term) aus dem metasploit Modul aufgerufen. Wurde eingestellt, dass CVEs anhand von Suchbegriffen gesucht werden sollen so wird die Methode search_by_term_CVE(search_term) aus dem webscraping Modul aufgerufen. Andernfalls wird die Methode search_by_CVE(search_term) aus dem metasploit Modul aufgerufen. Daraufhin wird überprüft, ob es entsprechende Suchergebnisse gab. Gibt es keine so wird der Nutzer darauf aufmerksam gemacht. Anschließend werden die Einträge in der GUI-Tabelle der letzten Suchen gelöscht und über jedes Suchergebnis iteriert. Hierbei wird wieder die Farbe der Spalte per tag definiert und abhängig des Such Typs eine neue Datenstruktur definiert. Zuletzt wird jeder Datensatz in die "tree-search" Tabelle eingefügt.

GUI – Mehr Informationen über Doppelclick(Adrian Graf)

Nachdem die Daten nun dem Nutzer angezeigt werden, habe ich überlegt, dass es praktisch wäre, wenn der Nutzer gerade bei Exploits mit möglichst wenig weiteren Klicks noch mehr Informationen sehen könnte. Die einfachste Variante, die mir dabei einfiel, ist mit Doppelklick auf eine Zeile ein kleines popup-Fenster zu öffnen mit weiteren Informationen.

Dementsprechend habe ich dann auch hier wieder drei Methoden für die drei unterschiedlichen UI-Bereiche geschrieben.

Angefangen mit dem Doppelklick auf eine Zeile der Tabelle mit den BSI-Daten, hatte ich den Einfall, dass dort gar kein neues Fenster in unserer GUI gebraucht wird, sondern man stattdessen einfach die BSI-Seite mit den weiteren Details öffnen kann. Aus dieser Idee entstand die double_click_bsi Methode.

double_click_bsi(event)

```
item = self.tree_BSI.selection()
if item:
    url = self.tree_BSI.item(item, "values")[4]
    webbrowser.open(url, new=0, autoraise=True)
```

Abbildung 29: Hauptteil double_click_bsi Methode

Die Logik hinter dem Doppelklick auf eine Zeile in der Tabelle mit den BSI-Daten ist Recht simpel.

Über tree_BSI.selection() kann einfach ermittelt werden auf welche Zeile geklickt wurde. Von dieser Zeile wird der fünfte value gelesen, sprich die fünfte Zeile, die die URL enthält. Um dies zu ermöglichen habe ich beim Einfügen der Daten die URL in der fünften Spalte eingefügt und diese für den Nutzer unsichtbar gemacht. Somit dient diese nur zur möglichst einfachen Umsetzung der Doppelklick Funktionalität.

Nachdem die URL ausgelesen wurde, wird dann ein neues Browser Fenster mit der entsprechenden Seite geladen.

double_click_matches(event)

```
item = self.tree_matches.selection()
if item:
    url = self.tree_matches.item(item, "values")[6]
    description = self.tree_matches.item(item, "values")[7]
    popup = tk.Toplevel(root)
    popup.title("Exploit Beschreibung")
    popup.geometry("600x300")
    popup.resizable(True, True)
    scroll = tk.Scrollbar(popup)
    scroll.pack(side=tk.RIGHT, fill=tk.Y)
    text_widget = tk.Text(popup, height=10, width=60, font=("Helvetica", 12), yscrollcommand=scroll.set, wrap="word")
    text_widget.pack(padx=10, pady=10,expand=True, fill="both")
    text_widget.insert(tk.END, f"{description}\n\nURL: ")
    if not url == "No URL available":
        text_widget.insert(tk.END, url, 'link')
        text_widget.tag_config('link', foreground="blue", underline=True)
    else:
        text_widget.insert(tk.END, url)

    text_widget.tag_bind('link', '<Button-1>', lambda e: webbrowser.open_new(url))
```

Abbildung 30: Hauptteil double_click_matches Methode

Bei der double_click_matches() Methode, die den Doppelclick auf die Tabelle mit den Paaren von CVEs und Exploits realisiert, ist das Vorgehen ähnlich. Hier werden die URL und die Beschreibung des ausgewählten Exploits aus für den Nutzer unsichtbaren Spalten abgelesen. Daraufhin wird jedoch nicht direkt die URL in einem Browser Fenster geöffnet. Stattdessen wird ein neues GUI-Fenster erstellt. Dieses beinhaltet die Beschreibung sowie die URL. Dadurch kann der Benutzer zuerst die Beschreibung durchlesen und falls er dann immer noch weitere Details sehen möchte, noch auf die URL klicken.

double_click_search(event)

```
url = self.tree_search.item(item, "values")[3]
if self.selected_option.get() == "Suche nach CVEs anhand Suchbegriff":
    webbrowser.open(url, new=0, autoraise=True)
else:
    description = self.tree_search.item(item, "values")[4]
```

Abbildung 31: Hauptteil double_click_search Methode

Der Ablauf der double_click_search() Methode ähnelt der double_click_matches sehr, bis auf die Unterscheidung, die am Anfang getroffen wird.

Wurde nach CVEs gesucht so wird direkt die URL der jeweiligen BSI-Seite geöffnet. Wurde nach Exploits gesucht so wird das Verfahren mit einem extra UI-Fenster durchgeführt.

Weitere GUI-Funktionen (Selim Karali)

Aus Gründen der User-Experience, wurden weitere Funktionen implementiert, um dem Nutzer den Umgang mit der GUI intuitiver zu gestalten und damit zu jederzeit auch die richtigen Informationen eingeblendet werden. Dies war eine sehr große Feinarbeit und als Maßstab dessen, was unsere GUI dem Nutzer bieten soll, wurden eigene Erfahrungen mit Webseiten und Apps herangezogen.

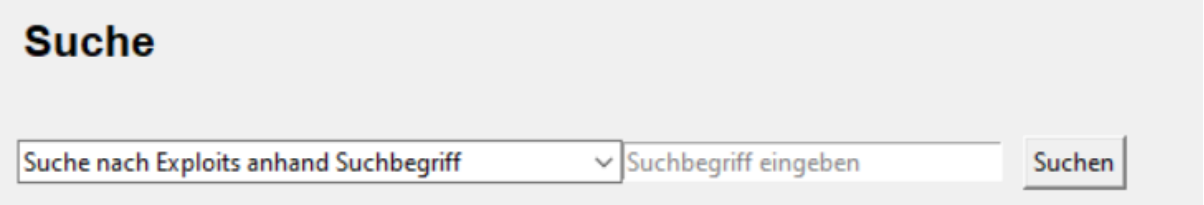


Abbildung 32: Suchleiste

Es wurden mit ”insert_placeholder()” und ”update_placeholder” in Data_Table_App.pz Platzhalter im Suchfeld realisiert. Der Platzhalter ist ein normaler nicht editierbarer und grauer Text. Sobald das Textfeld angeklickt wird, verschwindet der Platzhalter und die Schriftfarbe des Textfeldes ändert sich wieder zu schwarz. (“clear_entry”).

Des Weiteren musste z.B. das Leeren der Suchergebnisse realisiert werden, sobald eine Suche in einer anderen Kategorie begonnen wurde. Dies ist der Fall, sobald der Nutzer im Dropdown-Menü eine andere Auswahl trifft.

Voraussetzungen zur Ausführung der Anwendung (Selim Karali)

1. Der Microsoft-Edge Server muss installiert sein, da dieser für das Webscraping verwendet wird, dies wurde so mit EnBw abgesprochen. Der benötigte Driver, der für das Webscraping benötigt wird, (MSEdgeDriver.Exe) wird durch unseren Code installiert.
2. Es muss das Metasploit-Framework installiert sein (https://docs.rapid7.com/metasploit/installing-the-metasploit-framework/).
   Die auf der Seite genannten Schritte zur Installation sollten befolgt werden, da das Windows Antiviren-Programm eine Installation nur über Umwege zulässt.
3. Es muss eine aktive Internetverbindung bestehen.

Fazit

Nach anfänglichen Problemen dabei die Richtung, in die das Projekt gehen soll zu bestimmen, ist es uns insgesamt gelungen einen Funktionstüchtigen Prototyp zu erstellen, der die Arbeit bei EnBW erleichtert und die definierten Requirements erfüllt. Diesen als Frühwarnsystem zu betiteln kann allerdings irreführend sein. Der Prototyp erleichtert die Suche nach Schwachstellen und hilft zu kategorisieren, wie gefährlich eine Schwachstelle tatsächlich ist. Ein Warnsystem ist das aber in diesem Sinne nicht. Es gibt keine aktiven Meldungen, die den Nutzer darauf hinweisen, dass sein System gerade einer akuten Gefahr ausgesetzt seien könnte. Diese Funktionalität hätte den Zeitrahmen gesprengt, aber ist gut vorstellbar in einer weiteren Projektarbeit.

Abschließend haben wir durch die Projektarbeit einen realistischeren Eindruck über die Projektrealisierung in der Arbeitswelt gewonnen und werden zukünftig zielstrebiger und effizienter mit offenen Aufgabenstellungen umgehen können.

Potenzielle Erweiterungen

Auswertung von News

Aktuelle Top News in Gefahrenbewertung einbeziehen. Dafür analysieren welche Branchen bzw. Arten von Software betroffen sind. Alternativ kann auch für ein CVE anhand des Titels nach News gesucht werden und deren Informationen ausgewertet werden.

KI-Modelle

Ein KI-Modell auf die vorhandenen Daten trainieren, das Prognosen darüber trifft, ob es wahrscheinlich ist das es zu einem CVE ein Exploit geben wird.

Abgrenzungen durch den User

Dem User die Möglichkeit bieten selbst Filter zu definieren, um z.B. bei CVEs allgemein irrelevante Schlagwörter wie z.B. “Python” zu filtern und die Daten somit übersichtlicher zu machen. Unter anderem wäre hier auch möglich den Nutzer priorisieren zu lassen, welche Produkte beispielsweise von großer Relevanz sind.

Benachrichtigung über E-Mail

Vorstellbar ist auch ein Benachrichtigungssystem, das beispielsweise über E-Mail realisiert wird und bei einem neuen Exploit das zu einem CVE passt eine E-Mail mit entsprechender Information sendet oder jeden Montag eine Zusammenfassung der letzten Woche aufbereitet.

Quellen

1. Cybergefahren:
   https://www.myrasecurity.com/de/knowledge-hub/cyberangriff/

   Myra, 2023, aufgerufen am 06.01.2024

2. Häufige Cybergefahren:
   https://www.ingenieur.de/technik/fachbereiche/ittk/die-haeufigsten-varianten-von-cyberangriffen-im-world-wide-web/

   Silvia Hühn, verfasst am 19.10.2023, 14:50 Uhr

3. Häufige Cybergefahren:
   https://de.statista.com/statistik/daten/studie/180915/umfrage/mitarbeiterzahl-von-enbw-seit-2006/

   V. Pawlik, verfasst am 02.01.2024

4. Motivation:
   https://www.csoonline.com/de/a/diese-unternehmen-hat-s-schon-erwischt,3674038

   Redaktion des CSO, verfasst am 19.02.2024

5. Virustotal:
   https://www.luis.uni-hannover.de/de/services/it-sicherheit/praevention/virustotal

   IT-Sicherheits-Team des LUIS, verfasst am 18.07.2023

6. CVE:
   https://www.redhat.com/de/topics/security/what-is-cve

   RedHat, verfasst am 25.11.2021

7. CVE Aufbau:
   https://www.ondeso.com/beitrag/common-vulnerabilities-and-exposures-cve/
   Ondesco, 27.02.2020

8. Metasploit-Framework Docs:
   https://docs.rapid7.com/metasploit/msf-overview/
   aufgerufen am 20.01.2024

9. What is an Exploit:
   https://www.cisco.com/c/en/us/products/security/advanced-malware-protection/what-is-exploit.html
   abgerufen am 02.02.2024

10. Metasploit-Framwork GitHub-Projekt:
    https://github.com/rapid7/metasploit-framework/tree/master
    abgerufen am 20.01.2024

11. CVE-Aufbau:
    https://www.ondeso.com/beitrag/common-vulnerabilities-and-exposures-cve/

    Ondesco, 27.02.2020