Miłosz Kutyła (318427), Jakub Ossowski (318435),

Patryk Jankowicz (318422), Jan Walczak (318456)

Sprawozdanie z realizacji laboratorium KRYCY nr 1

18 grudnia 2023

Spis treści

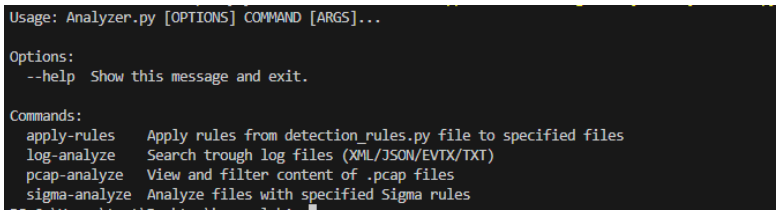
Wstęp

Celem laboratorium jest stworzenie prototypu systemu EDR/XDR w języku Python umożliwiającego skanowanie plików PCAP i logów w poszukiwaniu anomalii. Narzędzie ma umożliwiać zintegrowane zarządzenie źródłami danych z sieci i hostów, zunifikowaną interakcję z OS z poziomu jednej aplikacji oraz detekcję cyberzagrożeń poprzez wyrażenia regularne, reguły jako funkcje w języku Python oraz przez język reguł Sigma.

1. Analizator Cyberzagrożeń (aplikacja CLI)

Podpunkt GEN.MGMT.1

Zgodnie z poleceniem napisaliśmy aplikację z interfejsem CLI wykorzystując bibliotekę `Click`. Na rysunku 1. przedstawione są dostępne metody obsługujące utworzone scenariusze. Każdą z nich można skonfigurować według udostępnionych opcji, co zostało szczegółowo omówione w kolejnych sekcjach sprawozdania.



Rysunek 1: Interfejs CLI aplikacji

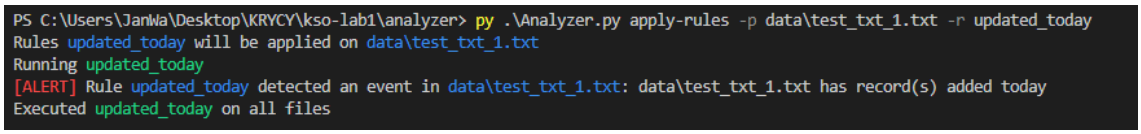
Podpunkt GEN.MGMT.2

Na CLI aplikacji wypisywane są informacje związane z wykonywanymi metodami oraz alerty z wykrytych przez poszczególne reguły zdarzeń. Aplikacja posiada również plik loga `analyzer.log`, w którym zbierane są logi informacyjne – przekazujące informacje analogiczne do tych wypisywanych na CLI. Dokładny opis wyświetlanych i logowanych informacji przedstawiony jest w kolejnych sekcjach sprawozdania.

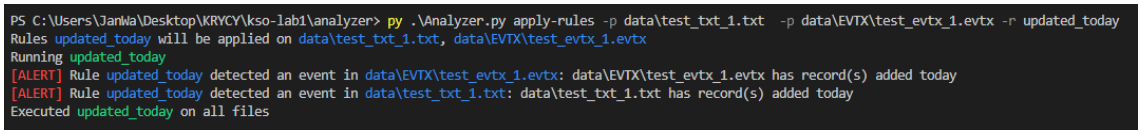
Podpunkt GEN.MGMT.3

Analizator udostępnia możliwość wskazania zarówno pliku jak i folderu do analizy - program rozpoznaje rodzaj atrybutu. W przypadku folderu rekurencyjnie sprawdza jego zawartość, zapisując wszystkie występujące w nim pliki o obsługiwanych rozszerzeniach.

Argumenty możemy przekazywać używając flagi `-p` lub `--path`. W przypadku gdy chcemy przeanalizować kilka plików/folderów, każdy z nich musimy podać z osobną flagą. Sposoby przekazywania plików przedstawiają rysunki 2a i 2b.



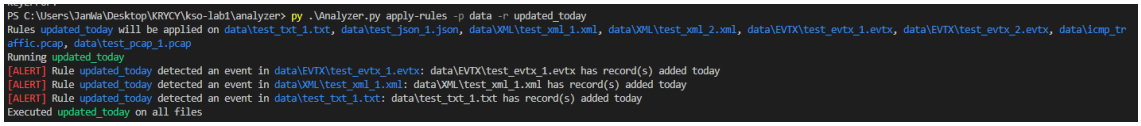
(a) Przekazanie jednego pliku jako argument



(b) Przekazanie dwóch plików o różnych rozszerzeniach jako argumenty

Rysunek 2: Przekazywanie plików do Analizatora Cyberzagrożeń

Sposoby przekazywania folderów przedstawiają rysunki 3a i 3b.



(a) Przekazanie folderu jako argument

```
PS C:\Users\JanWa\Desktop\KRYCY\kso-lab1\analyzer> py .\Analyzer.py apply-rules -p data\EVTX -p data\XML -r updated_today
Rules updated_today will be applied on data\XML\test_xml_1.xml, data\XML\test_xml_2.xml, data\EVTX\test_evtx_1.evtx, data\EVTX\test_evtx_2.evtx
Running updated_today
[ALERT] Rule updated_today detected an event in data\EVTX\test_evtx_1.evtx: data\EVTX\test_evtx_1.evtx has record(s) added today
[ALERT] Rule updated_today detected an event in data\XML\test_xml_1.xml: data\XML\test_xml_1.xml has record(s) added today
Executed updated_today on all files
```

(b) Przekazanie dwóch różnych folderów jako argumenty

Rysunek 3: Przekazywanie folderów do Analizatora Cyberzagrożeń

Podpunkt GEN.MGMT.4

Obsługiwane formaty plików, czyli `.txt`, `.xml`, `.json`, `.pcap` i `.evtx`, przechowujemy w słowniku list. Kluczem w słowniku jest rozszerzenie pliku, a wartością jest lista plików o danym formacie (zgodnym z kluczem). Obsługa plików o danym rozszerzeniu dla danej metody została opisana w dalszych częściach dokumentu.

Podpunkt GEN.MGMT.5

W module Analizatora dodaliśmy możliwość komunikacji ze Zdalnym Kolektorem Zdarzeń. Analizator po wykryciu zdarzenia i wygenerowaniu alertu wysyła wiadomość HTTP do Kolektora zawierającą obiekt zdarzenia i informuje użytkownika o statusie odpowiedzi (powodzenie lub niepowodzenie wysłania) na interfejsie CLI.

Podpunkt GEN.MGMT.5.1

Komunikacja odbywa się za pomocą REST API. Zdarzenia są generowane w postaci obiektu, a następnie są konwertowane do formatu `json` i przesyłane w ciele wiadomości `HTTP POST` do interfejsu API Kolektora Zdarzeń. Na rysunku 4. zostały przedstawione informacje na interfejsie CLI Analizatora informujące o poprawnym przesłaniu wiadomości do Kolektora zdarzeń (otrzymanie odpowiedzi od Kolektora z kodem HTTP 200). Rysunek 5. przedstawia DTO generowanego Eventu wraz z metodą służącą do jego wysyłania.

```
> python .\analyzer\Analyzer.py apply-rules  -p .\analyzer\data\icmp_traffic.pcap
E:\Politechnika\Semestr 5\KRYCY\krycy-lab1\analyzer\Analyzer.py:222: SyntaxWarning: invalid escape sequence '\d'
  RULES_SOURCE = 'analyzer\detection_rules.py'
Rules icmp_scan, example, updated_today will be applied on .\analyzer\data\icmp_traffic.pcap
Running icmp_scan
[ALERT] Rule icmp_scan detected an event in .\analyzer\data\icmp_traffic.pcap: Possible scanning activity detected
! 192.168.200.8 sent ICMP requests to the following hosts: 10.0.0.0-10.0.0.255
Event successfully uploaded to Event Collector
[ALERT] Rule icmp_scan detected an event in .\analyzer\data\icmp_traffic.pcap: Possible scanning activity detected
! 192.168.200.8 sent ICMP requests to the following hosts: 192.168.0.0-192.168.0.255
Event successfully uploaded to Event Collector
[ALERT] Rule icmp_scan detected an event in .\analyzer\data\icmp_traffic.pcap: Possible scanning activity detected
! 192.168.200.8 sent ICMP requests to the following hosts: 198.211.99.0-198.211.99.255
Event successfully uploaded to Event Collector
Executed icmp_scan on all files

Running example
[ALERT] Rule example detected an event in .\analyzer\data\icmp_traffic.pcap: Test event for .\analyzer\data\icmp_t
raffic.pcap
Executed example on all files

Running updated_today
Executed updated_today on all files
```

Rysunek 4: CLI: Komunikacja Analizatora z interfejsem API Kolektora Zdarzeń

```
69  class Event:
70
71      def __init__(self, rule_name: str, source_file: str, description: str) -> None:
72          self.rule_name = rule_name
73          self.source_file = source_file
74          self.description = description
75
76  def send_data_to_collector(event: Event):
77      url = "http://localhost:8000/upload"
78
79      if requests.post(url, data=json.dumps(event.__dict__)).status_code == 200:
80          print("Event successfully uploaded to Event Collector")
81      else:
82          print("Event upload failed - error on Server")
83
```

Rysunek 5: DTO Eventu oraz metoda do wysyłania zdarzeń do Kolektora

2. Zdalny Kolektor Zdarzeń

Podpunkt GEN.LOG.1

Przygotowaliśmy aplikację CLI opartą na `Click`, która odbiera wiadomości od Analizatora przy pomocy udostępnianego REST API. Analizator po wygenerowaniu alertu przesyła wiadomość do modułu Kolektora Zdarzeń. Kolektor, jeśli działa w trybie odbierania wiadomości, odbiera przesłaną wiadomość, wypisuje treść zdarzenia na CLI oraz zapisuje do plikowej bazy danych `SQLite`. Na rysunku 6. została przedstawiona wiadomość help modułu Kolektora Zdarzeń zawierająca informację o dwóch możliwych trybach działania.

```
> python .\EventCollector.py --help
Usage: EventCollector.py [OPTIONS] COMMAND [ARGS]...

Options:
  --help  Show this message and exit.

Commands:
  filedump  REST API server start and continous CLI view
  sqlview   View event history with filtering options
```

Rysunek 6: Wiadomość help aplikacji Kolektora Zdarzeń

Podpunkt GEN.LOG.2

Kolektor posiada dwa tryby działania.

1. Odbierane wiadomości ze zdarzeniami i wypisywanie ich w trybie ciągłym na CLI oraz zapisywane do plikowej bazy danych `sqlite` – tryb `filedump`.
2. Odczytywanie historii zdarzeń z plikowej bazy danych `sqlite` z możliwością filtrowania – tryb `sqlview`.

Tryb działania wybierany jest za pomocą przekazania odpowiedniej komendy podczas uruchamiania aplikacji. Na rysunku 7. została przedstawiona wiadomość help aplikacji dla obu trybów działania, która zawiera informacje o możliwych do przekazania argumentach.

```
> python .\EventCollector.py filedump --help
Usage: EventCollector.py filedump [OPTIONS]

  REST API server start and continous CLI view

Options:
  --host TEXT     Hostname of API server (Default: localhost)
  --port INTEGER  Port of API server (Default: 8000)
  --help          Show this message and exit.
```

(a) Wiadomość help trybu `filedump` Kolektora Zdarzeń

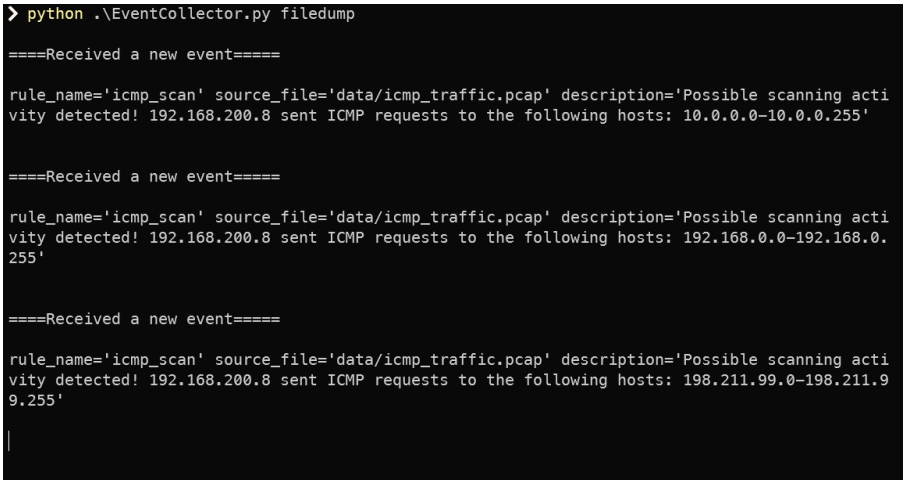
```
> python .\EventCollector.py sqlview --help
Usage: EventCollector.py sqlview [OPTIONS] [FILTER_STRING]

  View event history with filtering options

Options:
  -f, --filter [rule_name|source_file|date]
                                  Select the type of filter to use
  -s, --start_date INTEGER        Unix timestamp of earliest date
  -e, --end_date INTEGER          Unix timestamp of latest date
  --help                          Show this message and exit.
```

(b) Wiadomość help trybu `sqlview` Kolektora Zdarzeń

Rysunek 7: Wiadomości help dla obu trybów działania

Podpunkt GEN.LOG.2.1

Pierwszy tryb działania pozwala na odbieranie zdarzeń od Analizatora za pomocą interfejsu REST API oraz ciągłe wyświetlanie odebranych zdarzeń na interfejsie CLI. Na rysunku 8. został przedstawiony przykład działania ciągłego wyświetlania wiadomości po odebraniu zdarzeń przez Kolektor.



Rysunek 8: Przykład działania Kolektora w trybie `filedump`

Podpunkt GEN.LOG.2.2

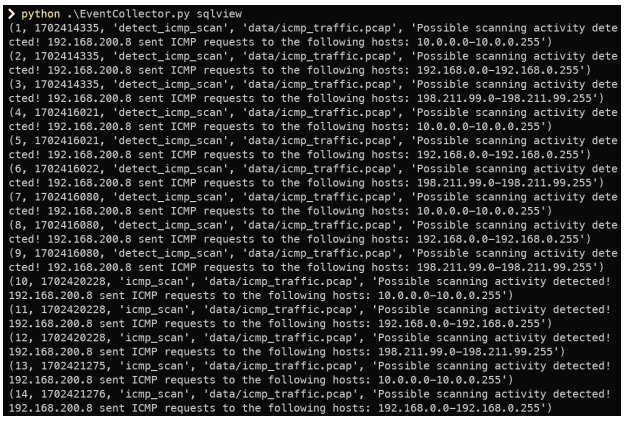
Drugi tryb działania pozwala na przeszukiwanie zapisanych wcześniej zdarzeń w bazie `SQLite` wraz z możliwością filtrowania otrzymywanych wyników. Tabela w bazie danych, w której przechowujemy zdarzenia, oprócz kolumny `id` (klucz główny) zawiera:

- `timestamp` – Unixowy timestamp zapisujący czas, w którym Kolektor odebrał zdarzenie,
- `rule_name` – nazwa reguły, która wygenerowała zdarzenie,
- `source_file` – ścieżka do pliku, który był analizowany przez regułę,
- `description` – szczegółowy opis zdarzenia.

Zdecydowaliśmy się na filtrowanie wpisów bazy danych na podstawie wyszukiwania wartości trzech z wyżej wymienionych kolumn:

- filtr `rule_name` – umożliwia wyszukiwanie zdarzeń wygenerowanych przez regułę, której nazwa została przekazana w argumencie.
- filtr `source_file` – umożliwia wyszukiwanie zdarzeń wygenerowanych przez plik, którego nazwa została przekazana w argumencie.
- filtr `date` – umożliwia wyszukiwanie danych wygenerowanych w przekazanym za pomocą flag `-s` (start date) oraz `-e` (end date) przedziałów czasowych.

Na rysunku 9. przedstawione zostały przykłady działania poszczególnych trybów filtrowania. W opisach dla filtrowania za pomocą daty został przedstawiony jedynie czas z dokładnością do sekundy, ponieważ wszystkie testowane zdarzenia odebrane zostały tego samego dnia.



(a) Wszystkie zdarzenia

(b) Zdarzenia z reguły, której nazwa zaczyna się od "detect"

(c) Zdarzenia z pliku, który miał rozszerzenie ".pcap"

(d) Zdarzenia odebrane później niż 22:20:22

(e) Zdarzenia odebrane wcześniej niż 22:20:22

(f) Zdarzenia odebrane pomiędzy 22:20:22, a 23:47:56

Rysunek 9: Działanie modułu Kolektora Zdarzeń w trybie wyświetlania historii zdarzeń

3. Scenariusz 1: Analiza plików offline - PCAP

Podpunkt OFF.PCAP.1

W Analizatorze zaimplementowaliśmy możliwość wyświetlania zawartości pakietów wczytanych z plików `pcap`. Do obsłużenia plików `pcap` użyliśmy biblioteki `pyshark`. Na rysunku 10. została przedstawiona wiadomość help dla metody `pcap-analyze`.

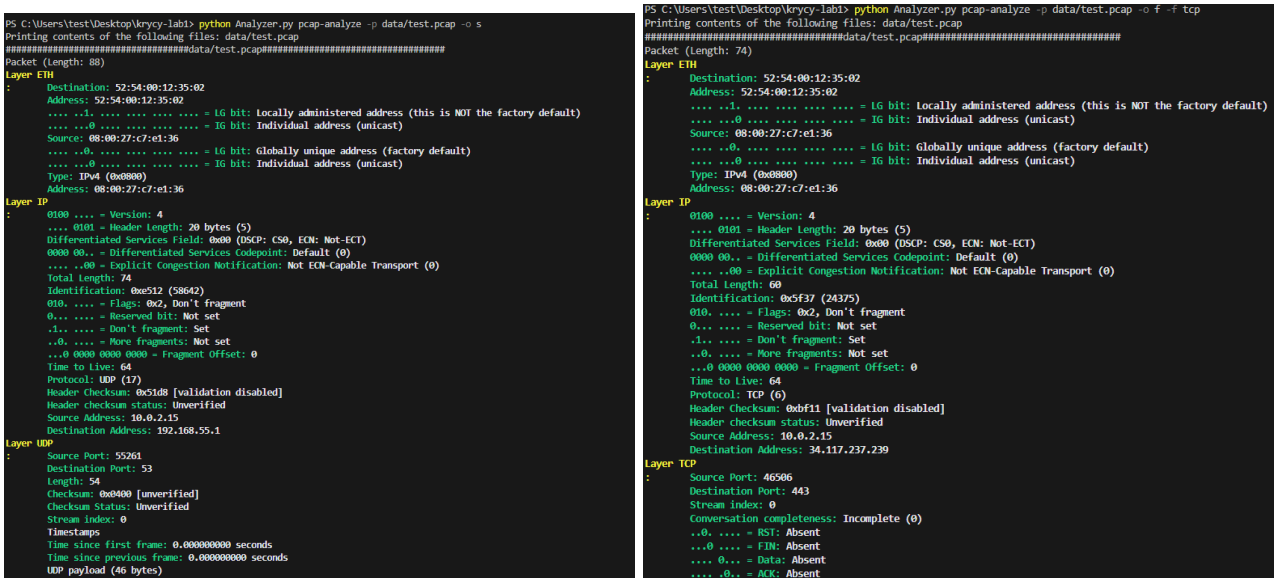
```
Usage: Analyzer.py pcap-analyze [OPTIONS]

  View and filter content of .pcap files

Options:
  -p, --path TEXT     Enter file path
  -o, --option [s|f]  Available commands: s - show | f - filter
  -f, --filter TEXT   Enter by what value you want to filter the pcap file
  --help              Show this message and exit.
```

Rysunek 10: Wiadomość help metody `pcap-analyze`

W celu zachowania czytelności wyświetlania, pakiety wypisywane są w grupach po 10 pakietów. Co każde 10 pakietów użytkownik ma możliwość zatrzymania (wprowadzając znak 'Q' lub 'q') lub kontynuowania (wprowadzając inny znak) wypisywania następnych pakietów. Wynik wypisywania przedstawia rysunek 11a.

Podpunkt OFF.PCAP.2

Analizator udostępnia również możliwość przekazania filtru zgodnego z formatem BPF. Wynik przykładowego filtrowanego wypisywania przedstawia rysunek 11b.



(a) Rezultat wypisania pliku .pcap

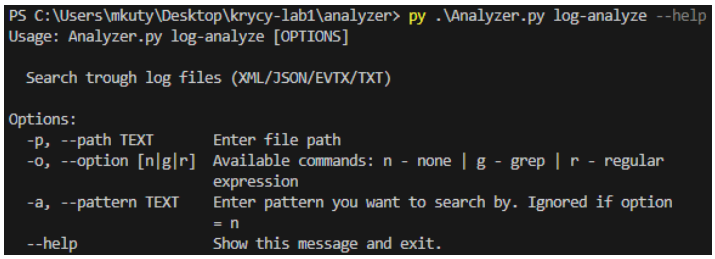
(b) Rezultat filtrowania pliku .pcap

Rysunek 11: Analiza plików offline

4. Scenariusz 2: Analiza plików offline - TXT / Logi

Podpunkt OFF.LOG.1

Analizator udostępnia funkcjonalność polecenia systemowego `grep`, którą wywołujemy przy pomocy metody `log-analyze`. Wiadomość help dla metody `log-analyze` przedstawia rysunek 12.

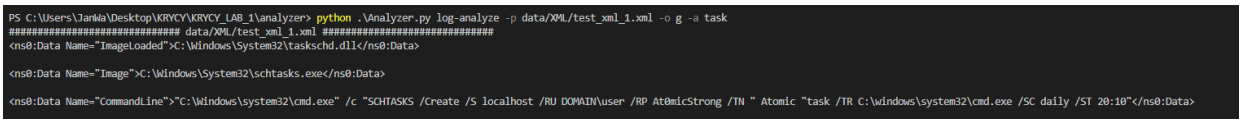


Rysunek 12: Korzystanie z metody `log-analyze`

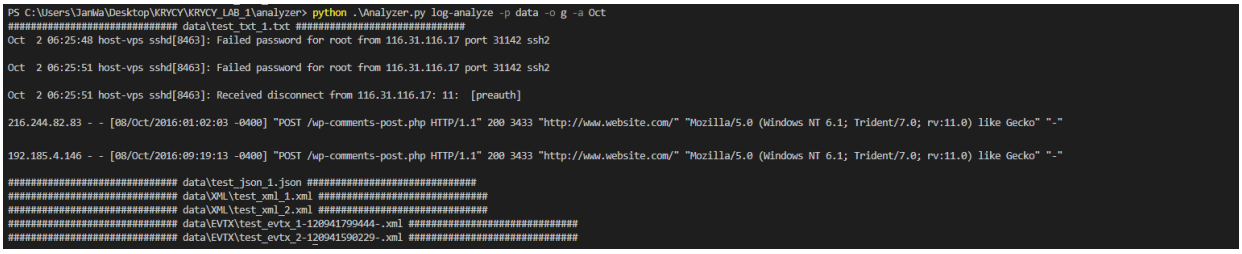
Metoda działa bezpośrednio na plikach tekstowych o rozszerzeniach: `json`, `xml` oraz `txt`. Obsługuje dodatkowo binarny typ pliku EVTX, który przetwarza w następujący sposób:

1. konwertuje go do formatu `xml` (z wykorzystaniem bibliotek `Evtx` oraz `xml`),
2. zapisuje plik w formacie `xml` do pliku o nazwie utworzonej poprzez dopisanie hashu pliku `xml` (obiektu utworzonego z pliku źródłowego `evtx`) do nazwy źródłowego pliku `evtx` (np. dla `test_evtx_2.xml` tworzony jest `test_evtx_2-174223562105.xml`),
3. obsługuje utworzony plik `xml`.

Rysunek 13. przedstawia grepowanie pliku `.evtx` po frazie `task`, natomiast rysunek 14. grepowanie po frazie "Oct" we wszystkich plikach z folderu `data`.



Rysunek 13: Rezultat grepowania pliku `evtx` po frazie `task`



Rysunek 14: Rezultat grepowania plików z folderu `data` po frazie `Oct`

Podpunkt OFF.LOG.2

Do obsługi wyrażeń regularnych wykorzystaliśmy bibliotekę `re`. W celu wykonania filtrowania wyrażeniem regularnym na wskazanych plikach należy w argumencie `-o` podać wartość `r` oraz uzupełnić argument `-a` odpowiednim regexem. Przykładowe wykonanie metody filtrującej przy pomocy regexu przedstawia rysunek 15.

Rysunek 15: Rezultat wyszukiwania adresów IP z zakresu 191.0.0.0. do 191.255.255.255 za pomocą regexu

5. Scenariusz 3: Reguły jako funkcje języka Python

Podpunkt OFF.DETPY.1

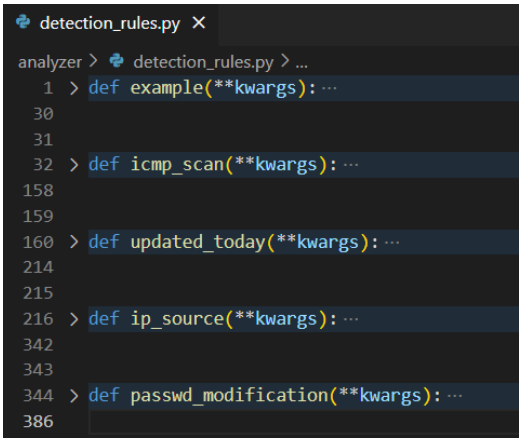
W Analizatorze Cyberzagrożeń zaimplementowaliśmy możliwość załadowania reguł do detekcji zdarzeń. Ładowane reguły są przechowywane w lokalnej dla metody liście. Dzięki temu każdorazowe wywołanie ładowania reguł oznacza usunięcie reguł istniejących w pamięci programu (flush) i załadowanie nowego zestawu reguł.

Podpunkt OFF.DETPY.1.1

Wszystkie reguły detekcji zostały zaimplementowane jako funkcje Pythona w pliku `detection_rules.py`.

Podpunkt OFF.DETPY.1.2

Każda reguła została zdefiniowana jako oddzielna funkcja w języku Python w pliku `detection_rules.py`, zgodnie ze wskazanym w instrukcji formacie. Fragment pliku `detection_rules.py` przedstawia rysunek 16.



Rysunek 16: Reguły (funkcje) w pliku `detection_rules.py`

Podpunkt OFF.DETPY.1.3

Każda reguła zwraca listę słowników opisujących wykryty event. Każdy słownik (event) zawiera trzy klucze:

- `action_alert`, czyli akcja alertująca: `local` (wypisanie alertu na CLI Analizatora oraz do loga) lub `remote` (funkcjonalność jak `local` z dodatkowym wysłaniem alertu do Kolektora Zdarzeń),
- `source`, czyli plik, w którym wykryto event,
- `description`, czyli opis tekstowy wykrytego zdarzenia.

Przykład tworzenia eventu (słownika) przedstawia rysunek 17.

```
event = {
    'action_alert': 'remote',
    'source': pcap,
    'description': f'Possible scanning activity detected! {source} sent ICMP requests to the following hosts: {ip_range}'
    }
events.append(event)
```

Rysunek 17: Tworzenie eventu dla reguły `icmp_scan`

Podpunkt OFF.DETPY.2

Nasz interfejs wywołania reguł, czyli metoda `apply-rules` pliku `Analyzer.py`, umożliwia ich użycie w dwóch trybach opisanych poniżej:

- Wywołanie całego zestawu reguł na wybranym zestawie plików (OFF.DETPY.2.1): poprzez przekazanie do metody jedynie plików do analizy.
- Wywołanie wybranych reguł (poprzez wskazanie ich nazwy tj. nazwy funkcji z `detection_rules.py`) na wybranym zestawie plików przekazanym do reguły (OFF.DETPY.2.2).

Obsługę metody `apply-rules` przedstawia rysunek 18.

```
PS C:\Users\mkuty\Desktop\krycy-lab1\analyzer> py .\Analyzer.py apply-rules --help
Usage: Analyzer.py apply-rules [OPTIONS]

Options:
  -p, --path TEXT         Path to file to apply the rules on
  -r, --rules_names TEXT  Rules to apply on passed files
  --help                  Show this message and exit.
```

Rysunek 18: Obsługa metody `apply-rules`

Podpunkt OFF.DETPY.2.1

Rysunek 19. przedstawia wykonanie wszystkich reguł na wskazanych plikach.

```
PS C:\Users\mkuty\Desktop\krycy-lab1\analyzer> py .\Analyzer.py apply-rules -p data/
Rule(s) icmp_scan, updated_today, ip_source, passwd_modification, example will be applied on data/\audit_passwd.txt, data/\test_txt_1.txt, data/\test_json_1.json, data/XML\test_xml_1.xml, data/XML\test_xml_2.
xml, data/EVTX\test_evtx_1.evtx, data/EVTX\test_evtx_2.evtx, data/\icmp_traffic.pcap, data/\test_pcap_1.pcap
Running icmp_scan
[ALERT] Rule icmp_scan detected an event in data/\icmp_traffic.pcap: Possible scanning activity detected! 192.168.200.8 sent ICMP requests to the following hosts: 10.0.0.0-10.0.0.255
Event successfully uploaded to Event Collector
[ALERT] Rule icmp_scan detected an event in data/\icmp_traffic.pcap: Possible scanning activity detected! 192.168.200.8 sent ICMP requests to the following hosts: 192.168.0.0-192.168.0.255
Event successfully uploaded to Event Collector
[ALERT] Rule icmp_scan detected an event in data/\icmp_traffic.pcap: Possible scanning activity detected! 192.168.200.8 sent ICMP requests to the following hosts: 198.211.99.0-198.211.99.255
Event successfully uploaded to Event Collector
Executed icmp_scan on all files

Running updated_today
[ALERT] Rule updated_today detected an event in data/EVTX\test_evtx_1.evtx: data/EVTX\test_evtx_1.evtx has record(s) added today
Executed updated_today on all files

Running ip_source
```

Rysunek 19: Analiza plików przy pomocy wszystkich reguł

Podpunkt OFF.DETPY.2.2

Rysunek 20. przedstawia wykonanie wskazanych reguł na wskazanym pliku.

```
PS C:\Users\mkuty\Desktop\krycy-lab1\analyzer> py .\Analyzer.py apply-rules -p data/icmp_traffic.pcap -r example -r icmp_scan
Rule(s) icmp_scan, example will be applied on data/icmp_traffic.pcap
Running icmp_scan
[ALERT] Rule icmp_scan detected an event in data/icmp_traffic.pcap: Possible scanning activity detected! 192.168.200.8 sent ICMP requests to the following hosts: 10.0.0.0-10.0.0.255
Event successfully uploaded to Event Collector
[ALERT] Rule icmp_scan detected an event in data/icmp_traffic.pcap: Possible scanning activity detected! 192.168.200.8 sent ICMP requests to the following hosts: 192.168.0.0-192.168.0.255
Event successfully uploaded to Event Collector
[ALERT] Rule icmp_scan detected an event in data/icmp_traffic.pcap: Possible scanning activity detected! 192.168.200.8 sent ICMP requests to the following hosts: 198.211.99.0-198.211.99.255
Event successfully uploaded to Event Collector
Executed icmp_scan on all files

Running example
[ALERT] Rule example detected an event in data/icmp_traffic.pcap: Test event for data/icmp_traffic.pcap
Executed example on all files
```

Rysunek 20: Analiza pliku przy pomocy wskazanych reguł

Warto zauważyć zróżnicowaną obsługę różnych trybów alertów:

- `example` zwraca alert, który powinien zostać obsłużony lokalnie. Możemy zauważyć poinformowanie użytkownika poprzez CLI o wykryciu testowego eventu.

- `icmp_scan` zwraca alert, który powinien zostać wysłany do Kolektora. Możemy zauważyć poinformowanie użytkownika poprzez CLI o wykryciu testowego eventu oraz informację o wysłaniu eventu do Kolektora.

Wyniki wykonania obu metod są odpowiednio logowane w pliku `analyzer.log`, co przedstawia rysunek 21.

```
[2023-12-13 21:11:26,164] INFO [9336] - Executed Analyzer.apply_rules ( path = ('data/icmp_traffic.pcap',), rules_names = ('example', 'icmp_scan') )
[2023-12-13 21:11:26,166] INFO [9336] - Analyzer.apply_rules(): rules 'icmp_scan', 'example' will be applied on 'data/icmp_traffic.pcap'
[2023-12-13 21:11:30,791] INFO [9336] - Analyzer.apply_rules(): Rule 'icmp_scan' detected an event in 'data/icmp_traffic.pcap': 'Possible scanning activity detected!
[2023-12-13 21:11:30,804] INFO [9336] - Analyzer.apply_rules(): sent event (detected by 'icmp_scan' in 'data/icmp_traffic.pcap') to Event Collector'
[2023-12-13 21:11:30,805] INFO [9336] - Analyzer.apply_rules(): Rule 'icmp_scan' detected an event in 'data/icmp_traffic.pcap': 'Possible scanning activity detected!
[2023-12-13 21:11:30,814] INFO [9336] - Analyzer.apply_rules(): sent event (detected by 'icmp_scan' in 'data/icmp_traffic.pcap') to Event Collector'
[2023-12-13 21:11:30,814] INFO [9336] - Analyzer.apply_rules(): Rule 'icmp_scan' detected an event in 'data/icmp_traffic.pcap': 'Possible scanning activity detected!
[2023-12-13 21:11:30,822] INFO [9336] - Analyzer.apply_rules(): sent event (detected by 'icmp_scan' in 'data/icmp_traffic.pcap') to Event Collector'
[2023-12-13 21:11:30,823] INFO [9336] - Analyzer.apply_rules(): Rule 'example' detected an event in 'data/icmp_traffic.pcap': 'Test event for data/icmp_traffic.pcap'
```

Rysunek 21: Logowanie wykrytych eventów

Wykryte eventy o obsłudze `remote` są odbierane przez Kolektor, co przedstawia rysunek 22.

```
====Received a new event=====

rule_name='icmp_scan' source_file='data/icmp_traffic.pcap' description='Possible scanning activity detected! 192.168.200.8 sent ICMP requests to the following hosts: 10.0.0.0-10.0.0.255'


====Received a new event=====

rule_name='icmp_scan' source_file='data/icmp_traffic.pcap' description='Possible scanning activity detected! 192.168.200.8 sent ICMP requests to the following hosts: 192.168.0.0-192.168.0.255'


====Received a new event=====

rule_name='icmp_scan' source_file='data/icmp_traffic.pcap' description='Possible scanning activity detected! 192.168.200.8 sent ICMP requests to the following hosts: 198.211.99.0-198.211.99.255'
```

Rysunek 22: Odebranie wykrytych eventów przez Kolektor Zdarzeń

6. Scenariusz 4: Wykorzystanie uniwersalnego formatu reguł - SIGMA

Podpunkt REG.DET.1-REG.DET.1.1

Z uwagi na ograniczoną dokumentację projektu Zircolite, nie zdecydowaliśmy się na zaimportowanie projektu jako biblioteki i używanie go bezpośrednio w kodzie. Zdecydowaliśmy wywoływać polecenia Zircolite poprzez metodę `system()` z biblioteki `os`. Integracja projektu `Zircolite` z naszym rozwiązaniem polegała na dodaniu pliku `zircolite.py` wraz z plikiem konfiguracyjnym i regułami sigma z repozytorium projektu `https://github.com/wagga40/Zircolite/tree/master`.

Podpunkt REG.DET.1.2

Umożliwiliśmy podawanie kilku plików i reguł do metody `sigma-analyze`, odpowiedzialnej za testowanie plików pod kątem zdarzeń opisywanych przez reguły Sigma. Generowane alerty wyświetlane są na konsoli, użyte polecenia wraz z alertami są logowane w pliku `analyzer.log`, a wynik detekcji (nazwa użytej reguły, skanowany plik, wynik skanowania) jest wysyłany po REST API do Kolektora.

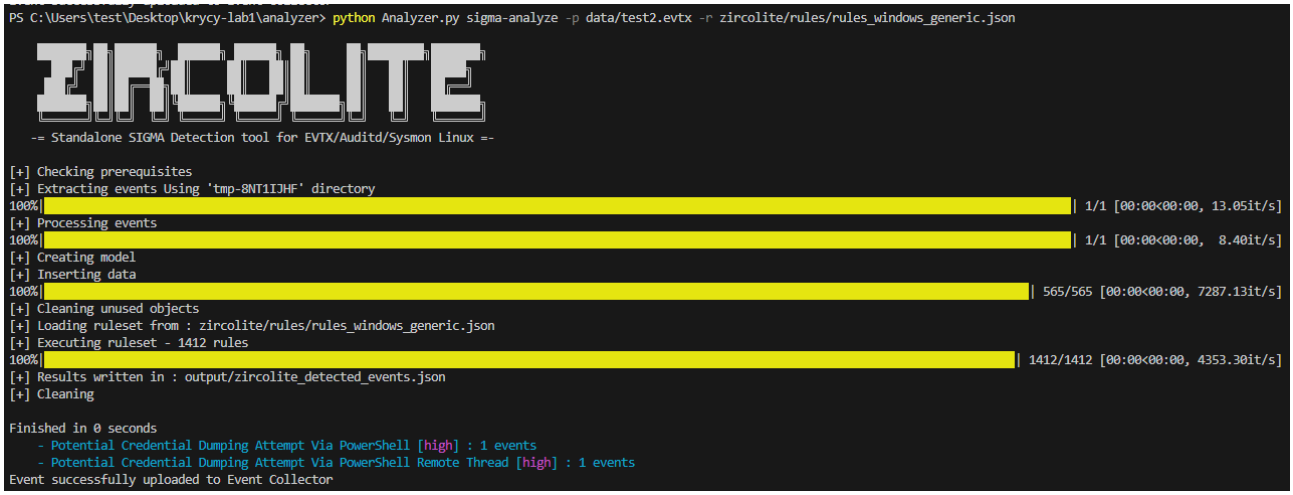
Podpunkt REG.DET.2-REG.DET.2.1

Przetestowaliśmy zintegrowane rozwiązanie dla reguł SIGMA. Wiadomość help dla metody `sigma-analyze` przedstawia rysunek 23.

```
Usage: Analyzer.py sigma-analyze [OPTIONS]

  "Analyze files with specified Sigma rules

Options:
  -p, --path TEXT  Enter file path
  -r, --rule TEXT  Enter a rule or rules to run
  --help           Show this message and exit.
```

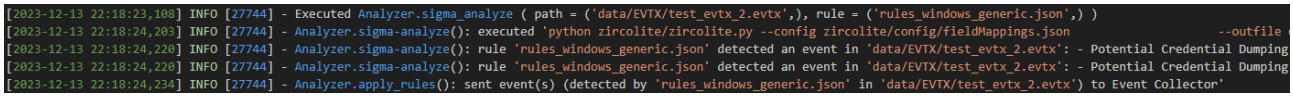
Rysunek 23: Wiadomość help dla metody `sigma-analyze`

Rezultat wywołania polecenia ze wskazaniem jednej reguły przedstawia rysunek 24.



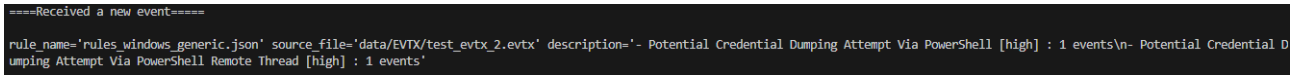
Rysunek 24: Rezultat wywołania polecenia ze wskazaną jedną regułą

Odpowiednie informacje są logowane w pliku `analyzer.log`, co przedstawia rysunek 25.



Rysunek 25: Wpisy dodane do pliku `analyzer.log`

Wynik skanowania jest również wysyłany do Kolektora, co przedstawia rysunek 26.



Rysunek 26: Konsola zdalnego Kolektora

W przypadku podania kilku reguł w jednym zapytaniu, program wywołuje każdą z nich po kolei na wskazanych plikach, co przedstawia rysunek 27.

(a) Rezultat wywołania pierwszej reguły

(b) Rezultat wywołania drugiej reguły

Rysunek 27: Rezultat wywołania kilku reguł

7. Dodatkowe pytania teoretyczne

Skalowanie naszego rozwiązania z punktu widzenia potrzeb rozwiązania EDR/XDR, może zakładać rozszerzenie Kolektora do w pełni funkcjonalnego systemu na styl systemu SIEM oraz dodanie konfigurowalnego systemu / mechanizmu umożliwiającego reakcję na incydent (np. NextGen Firewall dla zagrożeń związanych z ruchem sieciowym). Ma to na celu umożliwienie reaktywnego odpowiadania na zaistniałe w czasie rzeczywistym, wykryte przez Analizator zagrożenia, które po przekazaniu do Kolektora mogą wywołać odpowiednią reakcję w systemie. Zbierając sygnały o wykrytych zagrożeniach z różnych źródeł, moglibyśmy na bieżąco wydawać polecenia mające na celu mitygację zagrożenia.

Zgłaszane zdarzenia powinny mieć wprowadzony parametr informujący o stopniu powagi zdarzenia (*severity/level*, np. o dyskretnych wartościach z zakresu 0-10) nadawany na poziomie reguły detekcji używanej przez Analizator. Ułatwiłoby to analizę i obsługę incydentów poprzez wprowadzenie systemu priorytetu (kolejka z priorytetem).

Dodatkowo możliwość skalowania charakteryzuje się możliwością wygodnego i szybkiego dodawania nowych reguł, które mogą operować na różnych formatach plików. W naszym rozwiązaniu nowo wprowadzane reguły nie zaburzają działania programu, a ich dynamiczne wczytywanie pozwala na ich modyfikacje w trakcie działania Analizatora. W celu ułatwienia zarządzania regułami, możliwe powinno być rozbicie pliku `detection_rules.py` na mniejsze pliki – jest to łatwe do zaimplementowania w naszym rozwiązaniu.

8. Wnioski

Celem laboratorium było utworzenia prototypu systemu EDR/XDR w języku Python umożliwiającego skanowanie plików PCAP i logów w poszukiwaniu anomalii. Należało utworzyć zdalny Kolektor Zdarzeń (udostępniający styk REST API do przesyłania wykrytych zdarzeń), który umożliwia przechowywanie, filtrowanie i przeglądanie zebranych danych. Konfiguracja tej części projektu nie była problematyczna. Ważną decyzją było przechowywanie znaczników czasowych otrzymanych logów – w celu uproszczenia ich przechowywania i transmisji zdecydowaliśmy o zastosowaniu formatu `unix timestamp`.

W kontekście Analizatora pierwsze problemy zaczęły się na poziomie konwersji plików w formacie `EVTX`, których nie można otworzyć bezpośrednio (są binarne), na format `XML`. Sprawy nie ułatwił fakt występowania kilku bibliotek do obsługi tego typu plików różniących się np. wielkością pierwszej litery i działających tylko na konkretnej (niekoniecznie najnowszej) wersji Python'a.

W dalszej części kluczowe było zdecydowanie o sposobie zintegrowania Zircolite'a. Dokumentacja projektu wspomina, że "biblioteki można używać bezpośrednio w Pythonie", ale nie podaje żadnych przykładów jak z niej skorzystać. Dlatego postanowiliśmy wywoływać komendy z poziomu systemu przy pomocy biblioteki `os`. Takie rozwiązanie zadziałało bezproblemowo nawet w przypadku przekazania wielu reguł czy plików.

Ze względu na zunifikowanie obsługi reguł detekcji w jednej aplikacji Analizatora, Analizator można rozwinąć o dodatkowe algorytmy detekcji bazujące np. na uczeniu maszynowym. Po stworzeniu klasy modelu i jego wytrenowaniu, jego obiekt może być serializowany i przechowywany w pliku. Wówczas w razie potrzeby (np. analiza zachowania w ruchu sieciowym) model byłby wczytywany i używany. Dzięki temu moglibyśmy wzbogacić nasze rozwiązanie o dodatkowe mechanizmy detekcyjne, które ciężko byłoby opisać regułami czy funkcjami.

Ostatecznie projekt/laboratorium oceniamy jako ciekawy. Przez swoją złożoność i wieloetapowość był czasochłonny, ale i rozwijający. Podział wymagań na kody identyfikacyjne (GEN.MGMT.2, OFF.DETPY1.3 itd.) pozwolił nam na lepsze zarządzenie czasem i projektem – równy podział zadań między członkami zespołu, wprowadzenie deadlineów, zależności między zadaniami i możliwość łatwego śledzenia postępów (checklista).