PROJEKT

Multitenantné operačné centrum kybernetickej bezpečnosti riešené ako otvorená cloudová služba s prvkami strojového učenia

V14 funkčná infraštruktúra

KPB5 – Realizácia riešenia

typ – dokumentácia

mesiac odovzdania – M19

Financované Európskou úniou
NextGenerationEU

PLÁN OBNOVY

ÚRAD VLÁDY
SLOVENSKEJ REPUBLIKY

Obsah

1 ELK Stack 6
1.1 Opis systému 6
1.2 Dokumentácia systému 6
1.3 Požiadavky na zdroje 7
1.4 Architektúra nástroja 8
1.5 Inštalácia – variant Docker 9
1.6 Základná konfigurácia 11
1.7 Integrácie 14
2 Shuffle 15
2.1 Opis systému 15
2.2 Dokumentácia systému 16
2.3 Požiadavky na zdroje 16
2.4 Architektúra nástroja 16
2.4.1 Vnútorná architektúra 17
2.5 Inštalácia – variant Docker 18
2.6 Inštalácia – variant Openstack 20
2.7 Základná konfigurácia 21
2.7.1 Pripojenia na externé aplikácie Wazuh a Greenbone 22
2.7.2 Vstupy a výstupy 22
2.8 Integrácie 23
3 Wazuh 25
3.1 Opis systému 25
3.2 Dokumentácia systému 25
3.3 Požiadavky na zdroje 26
3.4 Architektúra nástroja 26
3.5 Inštalácia – variant Quickstart 28
3.6 Základná konfigurácia 28
3.7 Integrácie 31
4 Greenbone Community edition 32
4.1 Opis systému 32
4.2 Dokumentácia systému 33
4.3 Požiadavky na zdroje 33
4.4 Architektúra nástroja 33
4.5 Inštalácia – variant Docker 35
4.6 Základná konfigurácia 40
5 OpenCTI 41
5.1 Opis systému 41
5.2 Dokumentácia systému 42
5.3 Požiadavky na zdroje 42
5.4 Architektúra nástroja 42

5.5 Inštalácia – variant Docker 44
5.6 Inštalácia – variant Openstack 46
5.7 Základná konfigurácia 47
5.7.1 Konfigurácia MITTRE ATTACK 48
5.7.2 Konfigurácia Urlhaus 49
5.7.3 Konfigurácia Ransomwarelive 49
5.7.4 Vstupy a výstupy 50
5.8 Integrácie 51
5.8.1 Integrácia OpenCTI – pozícia CTI v SOC a integračný model 51
6 MISP 53
6.1 Opis systému 53
6.2 Dokumentácia systému 53
6.3 Požiadavky na zdroje 53
6.4 Architektúra nástroja 54
6.5 Integrácie 56
6.5.1 Pozícia MISP v SOC 56
6.5.2 Integrácia MISP s ostatnými SOC komponentmi 56
6.5.3 OpenCTI versus MISP 57
7 GLPI 58
7.1 Opis systému 58
7.2 Dokumentácia systému 58
7.3 Požiadavky na zdroje 59
7.4 Architektúra nástroja 59
7.5 Inštalácia – variant Docker (Compose) 61
7.6 Základná konfigurácia 62
7.7 Integrácie 64
8 NetBox 66
8.1 Opis systému 66
8.2 Dokumentácia 66
8.3 Požiadavky na zdroje 67
8.4 Architektúra nástroja 67
8.4.1 Modelovanie infraštruktúry 68
8.5 Experiment 70
8.5.1 Schéma 71
8.5.2 Inštalácia – variant Docker 71
8.5.3 Inštalácia – variant Openstack 88
8.5.4 Základná konfigurácia 89
8.5.5 Integrácie 90
8.5.6 Integrácia NetBox so SOC komponentmi 91
8.5.7 NetBox versus GLPI 92
9 DFIR IRIS – Ticketing 94

9.1 Opis systému ... 94
9.2 Dokumentácia systému ... 94
9.3 Inštalácia – variant Docker ... 96
9.4 Inštalácia – variant Openstack ... 97
9.5 Základná konfigurácia ... 99
9.6 Integrácie ... 100
10 DFIR IRIS – Evidencia, správa incidentov ... 101
10.1 Opis systému ... 101
10.2 Dokumentácia systému ... 101
10.3 Inštalácia – variant Docker ... 103
10.4 Inštalácia – variant Openstack ... 104
10.5 Základná konfigurácia ... 107
10.6 Integrácie ... 108
11 Kibana (ELK) ... 109
11.1 Opis nástroja ... 109
11.2 Dokumentácia systému ... 109
11.3 Požiadavky na HW ... 110
11.4 Architektúra nástroja ... 110
11.5 Inštalacia – variant Docker ... 111
11.6 Základná konfigurácia ... 112
11.7 Integrácie ... 112
12 Wiki.js ... 113
12.1 Opis systému ... 113
12.2 Dokumentácia systému ... 113
12.3 Požiadavky na zdroje ... 114
12.4 Architektúra nástroja ... 114
12.5 Inštalácia – variant Docker ... 115
12.6 Základná konfigurácia ... 116
12.7 Integrácie ... 117
13 Prostredie nasadenia ... 118
13.1 OpenStack KIS FRI UNIZA ... 118
13.1.1 Kontext CC riešenia ... 118
13.1.2 Architektúra OpenStack ... 118
13.1.3 Cloudové riešenie – OpenStack ako platforma pre INNOSOC ... 120
13.1.4 Infraštruktúrna vrstva prostredia ... 122
13.1.5 Hardvérové komponenty prostredia ... 125
13.1.6 Umiestnenie a izolácia SOC komponentov v prostredí ... 127
13.2 Kubernetes Brainit ... 129
13.2.1 Popis infraštruktúry Kubernetes ... 129
13.2.2 Infraštruktúra virtuálnych strojov – IaaC ... 130
13.2.3 Štruktúra repozitára ... 132

13.2.4 Správa a nasadzovanie Kubernetes aplikácií 133
13.3 Docker Brainit 134
13.3.1 Popis infraštruktúry docker 134
13.3.2 Operačný systém virutálnych serverov 134
13.3.3 Kontajnerizačná vrstva 135
13.3.4 Orchestrácia aplikácii 135
13.3.5 Celková architektúra riešenia 135
Záver 137

1 ELK STACK

Typ nástroja: SIEM/Log management/Security analytics

Názov nástroja: ELK Stack (Elasticsearch + Logstash/Beats + Kibana)

Verzia: ELK Stack 8.x (nasadenie prostredníctvom oficiálnych Docker image)

1.1 Opis systému

ELK Stack poskytuje rozsiahlu podporu pre zber logov a telemetrie prostredníctvom rodiny tzv. *data shipper* nástrojov (napr. Filebeat, Winlogbeat, Auditbeat), ktoré umožňujú zber udalostí z rôznych typov systémov vrátane Windows, Linux, bezpečnostných auditných záznamov a vybraných aplikačných služieb. Tieto komponenty sú navrhnuté tak, aby odľahčili zdrojové systémy a spoľahlivo doručili dáta do spracovateľskej vrstvy alebo priamo do Elasticsearch.

Na úrovni spracovania dát zabezpečuje Logstash parsovanie, normalizáciu a obohatenie údajov (napr. rozklad syslog správ, mapovanie na jednotné polia a zjednotenie formátov), čím pripravuje dáta na efektívnu indexáciu. Elasticsearch následne poskytuje výkonný indexačný a dotazovací engine, ktorý umožňuje rýchle vyhľadávanie, agregácie a časové korelácie nad veľkými objemami udalostí. Kibana nad týmto základom poskytuje používateľské rozhranie pre analýzu, tvorbu dashboardov, filtrovanie a prácu s časovými radmi, čo je v SOC prostredí kľúčové pre triage a vyšetrovanie incidentov.

Z pohľadu detekcie a reakcie je možné nad dátami budovať alerting mechanizmy (pravidlá a notifikácie) tak, aby boli podozrivé udalosti eskalované analytikom alebo nadväzujúcim procesom. Rozsah dostupných funkcií závisí od zvolenej distribúcie a licenčného modelu: v ekosystéme Elastic sú mnohé „core“ možnosti dostupné aj bez platených edícií, zatiaľ čo pokročilejšie analytické a enterprise funkcionality (napr. vybrané ML/UEBA scenáre alebo rozšírené integračné možnosti) môžu byť viazané na vyššie plány. V prípade alternatívnej implementácie postavenej na OpenSearch je alerting typicky riešený cez samostatný alerting modul/plugin.

Z bezpečnostného hľadiska je pri návrhu SOC riešenia dôležité riešiť ochranu komunikácie medzi komponentmi (šifrovanie), autentifikáciu a autorizáciu (riadenie prístupov) a audit prístupov.

1.2 Dokumentácia systému

Oficiálna dokumentácia ELK Stacku je verejne dostupná a poskytuje podrobný popis architektúry, inštalácie, konfigurácie a prevádzky jednotlivých komponentov. Dokumentácia pokrýva aj odporúčané postupy pre návrh ingest pipeline (parsovanie, normalizácia a enrichment), indexačnú stratégiu, vyhľadávanie a agregácie, tvorbu vizualizácií v Kibane, ako aj napojenie dátových zdrojov prostredníctvom data-shipper nástrojov (Beats) alebo jednotného prístupu cez Elastic Agent/Fleet.

Oficiálne zdroje (Elastic Docs):

- Dokumentačný portál Elastic (prehľad všetkých produktov a verzií): https://www.elastic.co/guide/index.html

- Elasticsearch – referenčná dokumentácia: https://www.elastic.co/docs/reference
- Elasticsearch – integrácie: https://www.elastic.co/integrations/data-integrations
- Logstash – referenčná dokumentácia: https://www.elastic.co/docs/reference/logstash
- Kibana – používateľská a referenčná dokumentácia: https://www.elastic.co/guide/en/kibana/index.html

Neoficiálne zdroje:

- Docker compose dokumentácia pre inštaláciu: https://github.com/deviantony/docker-elk

Zber dát (logy/udalosti) – Beats a Elastic Agent:

- Fleet & Elastic Agent (centrálna správa agentov a integrácií): https://www.elastic.co/docs/reference/fleet
- Migrácia z Beats na Elastic Agent (praktický prechodový guide): https://www.elastic.co/docs/reference/fleet/migrate-from-beats-to-elastic-agent
- Filebeat / Winlogbeat / Auditbeat (verzionované manuály sú dostupné cez Elastic docs index): https://www.elastic.co/guide/index.html
- GitHub repozitár Beats (prehľad projektu a odkazy na dokumentáciu): https://github.com/elastic/beats

Poznámka: Pri implementácii v SOC prostredí je vhodné pracovať s dokumentáciou verziovo konzistentne (napr. všetky komponenty v rámci jednej hlavnej verzie 8.x), aby sa predišlo rozdielom v konfigurácii, kompatibilite ingest pipeline a správaní UI v Kibane.

1.3 Požiadavky na zdroje

Hardvérové požiadavky ELK Stacku závisia predovšetkým od objemu spracovávaných dát, počtu zdrojov logov, retenčnej politiky a komplexnosti analytických dotazov a vizualizácií. V prostredí SOC je potrebné počítať s kontinuálnym tokom dát a s požiadavkou na rýchlu odozvu pri vyhľadávaní a korelácii udalostí.

Pre účely Proof of Concept (PoC) a menšie až stredne veľké nasadenie sú odporúčané nasledovné minimálne parametre:

- CPU: 4 vCPU
- RAM: 8–16 GB (kritické najmä pre Elasticsearch)
- Diskový priestor: min. 100 GB (SSD odporúčané, podľa nastavení retencie údajov)
- Disk: lokálny SSD alebo rýchle sieťové úložisko (IOPS sú kľúčové)
- Sieť: stabilná konektivita medzi zdrojmi logov a ingest vrstvou

V produkčnom SOC prostredí je bežné:

- oddeliť ingest vrstvu (Logstash/Beats) od databázovej vrstvy,

- nasadiť Elasticsearch v klastrovanom režime,
- používať samostatné uzly pre hot/warm dáta podľa retenčnej politiky.

Pre účely tejto práce je však ELK Stack nasadený v zjednodušenej PoC konfigurácii, ktorá umožňuje overenie funkčnosti detekcie, analýzy a vizualizácie bez potreby plnohodnotného produkčného clustra.

1.4 Architektúra nástroja

Architektúra nasadenia ELK Stacku v rámci SOC je navrhnutá ako viacvrstvové riešenie, ktoré oddeľuje zber dát, ich spracovanie, ukladanie a analytickú prezentáciu. Tento prístup zvyšuje škálovateľnosť, prehľadnosť a umožňuje postupné rozširovanie riešenia.

Na vstupnej vrstve sa nachádzajú zdroje dát, medzi ktoré patria operačné systémy (Linux, Windows), sieťové zariadenia a bezpečnostné senzory (napr. IDS/IPS). Zber logov je realizovaný prostredníctvom data-shipper nástrojov (Filebeat, Winlogbeat, Auditbeat) alebo prostredníctvom syslog mechanizmov. Ich úlohou je spoľahlivo doručiť udalosti do spracovateľskej vrstvy bez výrazného zaťaženia zdrojových systémov.

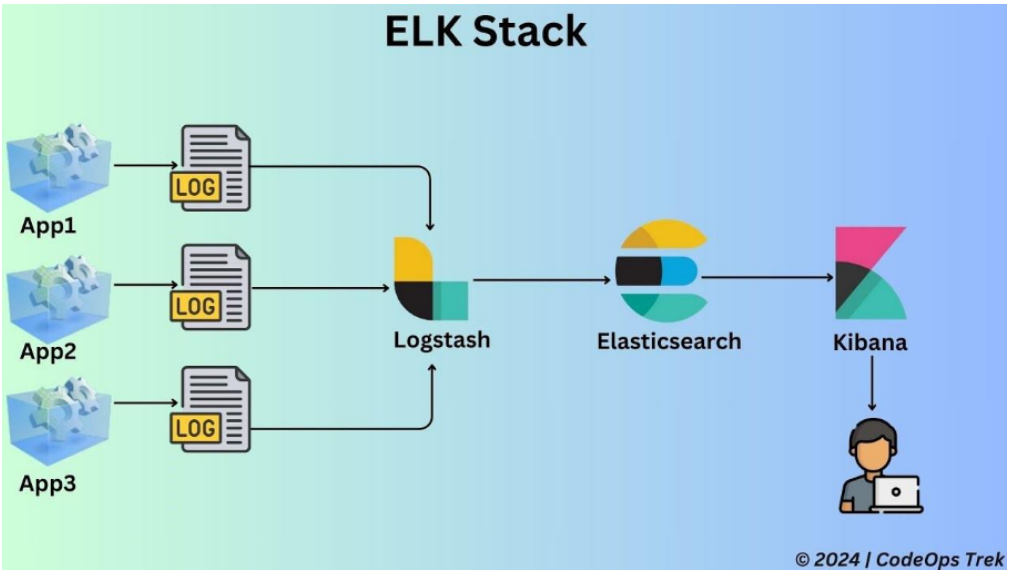
Spracovateľskú vrstvu tvorí Logstash, ktorý zabezpečuje parsovanie, normalizáciu a základný enrichment dát. V tejto fáze sú heterogénne vstupy transformované do jednotnej štruktúry, čo umožňuje efektívnu indexáciu a následnú koreláciu udalostí. Logstash zároveň predstavuje miesto, kde je možné implementovať základnú logiku pre triedenie udalostí podľa ich typu alebo závažnosti.

Dátová a analytická vrstva je reprezentovaná ElasticSearch-om, ktorý uchováva spracované udalosti v indexoch optimalizovaných na časové dotazy a agregácie. Elasticsearch tvorí jadro SIEM funkcionality, keďže umožňuje rýchle vyhľadávanie, koreláciu udalostí v čase a podporu detekčných mechanizmov založených na pravidlách.

Nad touto vrstvou sa nachádza Kibana, ktorá poskytuje používateľské rozhranie pre analytikov SOC. Umožňuje tvorbu dashboardov, vizualizáciu trendov, filtrovanie udalostí a detailnú analýzu incidentov. Kibana je hlavným pracovným nástrojom SOC analytika počas triage a vyšetrovania bezpečnostných udalostí.

Z pohľadu SOC architektúry je ELK Stack doplnený o kontextové zdroje, ako je dokumentačný systém NetBox. Tieto systémy neposkytujú samotnú detekciu, ale dodávajú analytikom nevyhnutný kontext (informácie o aktívach, IP adresácii, lokalite), ktorý zvyšuje presnosť interpretácie udalostí a znižuje počet falošných poplachov.

Schéma architektúry ELK Stack-u




```
[ Zdroje logov / senzory ]
         |
   (Beats / Syslog)
         |
     [ Logstash ]
         |
  [ Elasticsearch ]
         |
     [ Kibana ]
         |
   [ SOC analytik ]
```

1.5 Inštalácia – variant Docker

```
services:
  elasticsearch:
    build:
      context: elasticsearch/
      args:
        ELASTIC_VERSION: ${ELASTIC_VERSION}
    volumes:
      -
./elasticsearch/config/elasticsearch.yml:/usr/share/elasticsearch/config/elasticsearch.yml:
ro,Z
      - elasticsearch:/usr/share/elasticsearch/data:Z
    ports:
      - 9200:9200
      - 9300:9300
    environment:
      node.name: elasticsearch
      ES_JAVA_OPTS: -Xms512m -Xmx512m
      # Bootstrap password.
      # Used to initialize the keystore during the initial startup of
```

```
      # Elasticsearch. Ignored on subsequent runs.
      ELASTIC_PASSWORD: ${ELASTIC_PASSWORD:-}
      # Use single node discovery in order to disable production mode and avoid bootstrap
checks.
      # see:
https://www.elastic.co/guide/en/elasticsearch/reference/current/bootstrap-
checks.html
      discovery.type: single-node
    networks:
      - elk
    restart: unless-stopped
  logstash:
    build:
      context: logstash/
      args:
        ELASTIC_VERSION: ${ELASTIC_VERSION}
    volumes:
      - ./logstash/config/logstash.yml:/usr/share/logstash/config/logstash.yml:ro,Z
      - ./logstash/pipeline:/usr/share/logstash/pipeline:ro,Z
    ports:
      - 5044:5044
#      - 50000:50000/tcp
      - 514:50000/tcp
#      - 50000:50000/udp
      - 514:50000/udp
      - 9600:9600
    environment:
      LS_JAVA_OPTS: -Xms256m -Xmx256m
      LOGSTASH_INTERNAL_PASSWORD: ${LOGSTASH_INTERNAL_PASSWORD:-}
    networks:
      - elk
    depends_on:
      - elasticsearch
    restart: unless-stopped
  kibana:
    build:
      context: kibana/
      args:
        ELASTIC_VERSION: ${ELASTIC_VERSION}
    volumes:
      - ./kibana/config/kibana.yml:/usr/share/kibana/config/kibana.yml:ro,Z
    ports:
      # - 5601:5601
      - 80:5601
    environment:
      KIBANA_SYSTEM_PASSWORD: ${KIBANA_SYSTEM_PASSWORD:-}
      SERVER_PUBLICBASEURL: http://elk.innosoc.brainit.tech
    networks:
      - elk
    depends_on:
      - elasticsearch
    restart: unless-stopped
networks:
  elk:
    driver: bridge
volumes:
  elasticsearch:
```

Zmeny v docker compose: Zmenené niektoré sieťové porty – viď zakomentované riadky vyššie.

1.6 Základná konfigurácia

Konfigurácia nástroja:

ELK Stack, obsahuje konfiguračný skript ktorý správne nastaví adresy a potrebnú autentifikáciu medzi službami. Väčšinu konfigurácie som potom nechali v predvolených nastaveniach. Zvyšok konfigurácie potom už pozostáva len prispôsobeniu požiadaviek alebo pluginy a nastavenie retencie údajov.

Konfigurácia logstash pre syslog z Fortigate 200

`logstash/pipeline/fortigate.conf`

```
input {
  # Listen on the CONTAINER port 50000/udp, mapped from host port 514/udp
  # If you chose TCP on FortiGate, change 'udp' to 'tcp' here.
  udp {
    port => 50000 # <<< This is the CONTAINER port mapped in docker-compose.yml
    codec => plain { charset => "UTF-8" }
    type => "fortigate" # Add a type for easier filtering later
    tags => ["fortigate", "udp"]
  }

  # --- OR if you chose TCP on the FortiGate ---
  # tcp {
  #   port => 50000 # <<< CONTAINER port
  #   codec => line { charset => "UTF-8" } # Use 'line' codec for TCP
  #   type => "fortigate"
  #   tags => ["fortigate", "tcp"]
  # }
}

filter {
  # Only process messages tagged as fortigate AND likely containing key-value pairs
  if "fortigate" in [tags] and [message] =~ /type=/ {

    # Parse the FortiGate key-value format
    kv {
      source => "message"
      field_split => " "
      value_split => "="
      trim_key => "\""
      trim_value => "\""
      # prefix => "fortigate_" # Optional: Uncomment to prefix all fields
    }
```

```
    # Convert known numeric fields
    mutate {
      convert => {
        "rcvdbyte" => "integer"
        "sentbyte" => "integer"
        "duration" => "integer"
        "proto" => "integer" # Protocol number (e.g., 6 for TCP)
        "srcport" => "integer"
        "dstport" => "integer"
        "policyid" => "integer"
        # Add other common numeric fields: devid, vd, sessionid, count, etc.
      }
      # Remove quotes from potentially problematic fields if kv didn't catch them
      gsub => [
        "srcip", '"', '',
        "dstip", '"', '',
        "action", '"', '',
        "service", '"', ''
        # Add others as needed
      ]
    }

    # Parse timestamp from 'date' and 'time' fields
    if [date] and [time] {
      mutate {
        add_field => { "event_timestamp_str" => "%{date} %{time}" }
      }
      date {
        match => [ "event_timestamp_str", "yyyy-MM-dd HH:mm:ss" ] # Verify this format
matches your logs!
        target => "@timestamp"
        # timezone => "Your/Fortigate/Timezone" # IMPORTANT: Set if FortiGate isn't UTC and
doesn't include offset
      }
      mutate {
        remove_field => [ "event_timestamp_str" ]
      }
    }

    # GeoIP Enrichment (Optional, requires plugin install - see below)
    # if [srcip] {
    #   geoip {
    #     source => "srcip"
    #     target => "geoip_src"
    #     # database => "/path/inside/container/to/GeoLite2-City.mmdb" # Needs GeoIP
database mounted or built into image
    #   }
    # }
    # if [dstip] {
    #   geoip {
    #     source => "dstip"
    #     target => "geoip_dst"
    #     # database => "/path/inside/container/to/GeoLite2-City.mmdb"
    #   }
    # }

    # Optional: Rename fields for ECS compatibility if desired
```

```
    # mutate {
    #   rename => { "srcip" => "source.ip" }
    #   rename => { "dstip" => "destination.ip" }
    #   # ... more renames
    # }

  } else {
    # Optionally drop messages that don't look like FortiGate logs if other
    # things might arrive on this port (though dedicated port is better)
    # drop { }
  }
}

output {
  elasticsearch {
    # Use the service name 'elasticsearch' as host because they are on the same 'elk'
docker network
    hosts => ["http://elasticsearch:9200"]
    index => "fortigate-%{+YYYY.MM.dd}" # Daily indices

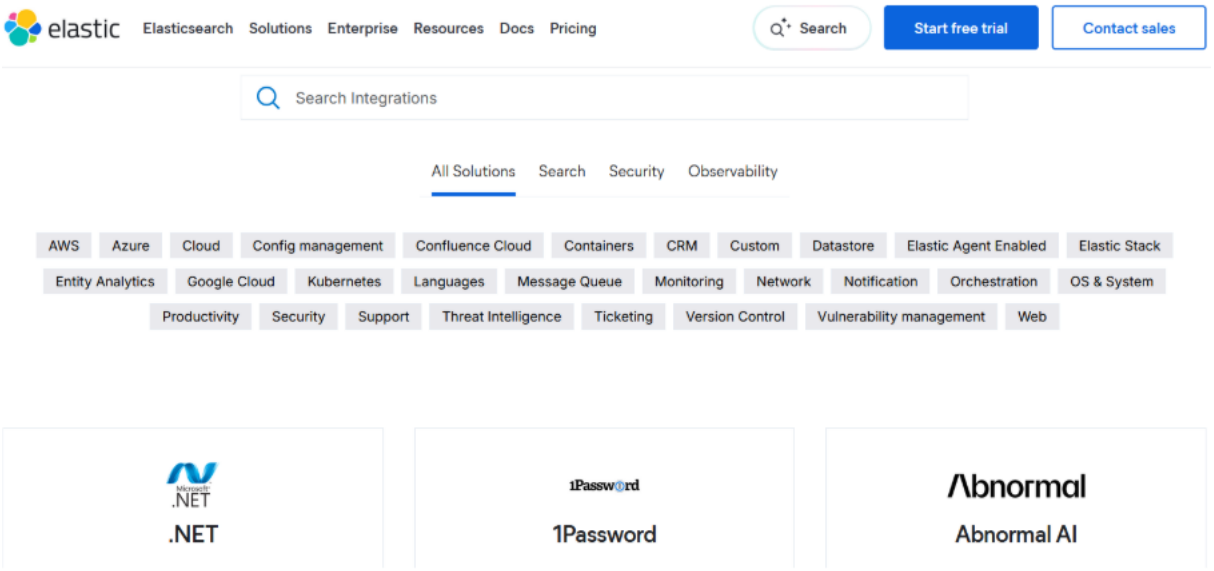
    # === Authentication ===
    # Use the internal user and password defined in your .env file and passed to the
container
    user => "logstash_internal"
    password => "${LOGSTASH_INTERNAL_PASSWORD}" # Logstash reads this from its environment

    # === Index Lifecycle Management (Recommended for Production) ===
    # Setup an ILM policy in Kibana first (e.g., named 'fortigate_policy')
    # ilm_enabled => true
    # ilm_rollover_alias => "fortigate"
    # ilm_policy => "fortigate_policy" # Make sure this policy exists in
Elasticsearch/Kibana
    # ilm_pattern => "000001"
  }

  # Optional: Output to container logs for debugging during setup
  # Remove or comment out for production.
  # stdout {
  #   codec => rubydebug
  # }
}
```

1.7 Integrácie

Elastic má obrovské množstvo a možnosti integrácií:



V stacku je potom taktiež logstash. Logstash je po správnej konfigurácií toho ako má spracovávať konkrétne syslog logy možné integrovať s akýmkoľvek systémom ktorý odosiela syslog. Príkladom takejto integrácie je jednoduchá integrácia ktorú sme si napísali pre prijímanie logov z firewallu Fortigate.

2 SHUFFLE

Typ nástroja: SOAR (Security Orchestration, Automation and Response)

Názov nástroja: Shuffle

Verzia: Release 2.1.0 (open-source verzia (kontajnerizované nasadenie))

2.1 Opis systému

Shuffle je open-source nástroj typu SOAR (Security Orchestration, Automation and Response), určený na orchestráciu a automatizáciu bezpečnostných procesov v prostredí SOC. Jeho hlavnou úlohou je znižovať manuálnu záťaž analytikov prostredníctvom automatizácie opakujúcich sa reakčných a obohacovacích činností, ktoré nadväzujú na detekčné mechanizmy SIEM systému.

Nástroj poskytuje webové používateľské rozhranie, v ktorom je možné vizuálne navrhovať, spravovať a spúšťať automatizované workflowy, tzv. *playbooky*. Shuffle je navrhnutý ako ľahký a flexibilný orchestrátor, ktorý dokáže prepájať bezpečnostné a IT nástroje prostredníctvom ich API rozhraní bez potreby rozsiahlej infraštruktúry alebo komplexných integračných platforiem.

Shuffle nepredstavuje plnohodnotný incident management systém ani náhradu SIEM riešenia. Jeho úlohou nie je detekcia udalostí ani ich dlhodobá analytika, ale rýchla reakcia, automatizované spracovanie udalostí a integrácia existujúcich nástrojov. V architektúre SOC tak vystupuje ako reakčná a automatizačná vrstva, ktorá nadväzuje na detekciu (ELK Stack) a pracuje s kontextom dodaným inými systémami (napr. NetBox).

V navrhovanom riešení je Shuffle využívaný na automatizáciu vybraných reakčných scenárov, obohacovanie incidentov o externé informácie a podporu konzistentného postupu pri spracovaní bezpečnostných udalostí.

Orchestrácia a automatizácia reakcií (Playbooky)

Základnou funkciou nástroja Shuffle je tvorba a vykonávanie automatizovaných playbook-ov, ktoré reprezentujú sekvenciu krokov reagujúcich na bezpečnostnú udalosť. Playbook-y sú vytvárané graficky pomocou drag-and-drop rozhrania, pričom jednotlivé kroky predstavujú volania API, spracovanie dát alebo rozhodovaciu logiku.

Shuffle umožňuje vytvárať:

- jednoduché workflow-y (napr. notifikácia, enrichment IP adresy),
- komplexné workflow-y s podmienkami, vetveniami a spätnými väzbami.

Playbook-y môžu byť spúšťané:

- manuálne (analytikom SOC),
- prostredníctvom webhooku,
- automaticky ako reakcia na udalosť z externého systému (napr. SIEM).

Z pohľadu SOC procesov je Shuffle využiteľný najmä vo fáze Response, kde dokáže:

- automaticky obohatiť incident o externé údaje,
- vykonať preddefinované reakčné kroky,
- zabezpečiť jednotný a opakovateľný postup pri spracovaní udalostí.

2.2 Dokumentácia systému

Oficiálna dokumentácia nástroja Shuffle je verejne dostupná a poskytuje prehľad architektúry, možností integrácie, návrhu playbook-ov a spôsobu nasadenia. Dokumentácia sa zameriava najmä na praktické použitie nástroja v SOC prostredí, vrátane práce s webhook-mi, API konektormi a workflow engine.

Odkazy na oficiálne zdroje:

- Oficiálna dokumentácia: https://shuffler.io/docs/about
- GitHub repozitár projektu: https://github.com/shuffle/shuffle
- Oficiálny Docker image: https://hub.docker.com/r/shuffle/shuffle

2.3 Požiadavky na zdroje

Shuffle má nízke hardvérové nároky, čo umožňuje jeho rýchle nasadenie aj v menších alebo testovacích prostrediach.

Pre testovacie alebo menšie produkčné nasadenie postačuje:

- CPU: 2 jadrá
- RAM: 4 GB
- Diskový priestor: 20–50 GB

Pre väčšie produkčné prostredia sa odporúča:

- CPU: 4–8 jadier
- RAM: 8–16 GB
- Disk: SSD úložisko

Vyššie nároky sa uplatňujú najmä pri prevádzke väčšieho počtu workflowov, integrácií a paralelných reakčných procesov.

2.4 Architektúra nástroja

Architektúra nástroja Shuffle je navrhnutá ako modulárne a ľahké riešenie, ktoré je možné jednoducho integrovať do existujúcej SOC infraštruktúry. Základ architektúry tvorí webová aplikačná vrstva, ktorá poskytuje používateľské rozhranie pre návrh a správu playbook-ov, a backend-ová API vrstva, ktorá zabezpečuje spracovanie požiadaviek a komunikáciu s externými systémami.

Súčasťou riešenia je workflow engine, ktorý vykonáva jednotlivé kroky playbook-ov a riadi logiku ich spracovania. Shuffle komunikuje s integrovanými nástrojmi prostredníctvom API konektorov, čím umožňuje automatizované volanie externých služieb a spracovanie ich odpovedí. Interná databáza slúži na ukladanie konfigurácií, stavov workflow-ov a výsledkov jednotlivých krokov.

V architektúre SOC je Shuffle umiestnený nad detekčnou vrstvou (SIEM) a slúži ako automatizačný komponent, ktorý reaguje na udalosti identifikované inými nástrojmi. Schéma architektúry nástroja Shuffle použitá v tejto práci vychádza z oficiálnej dokumentácie a ilustruje základné komponenty a ich vzťahy.

2.4.1 Vnútorná architektúra

Vnútorná architektúra nástroja Shuffle rozdeľuje jeho funkcionality do samostatných logických komponentov. Architektúra je navrhnutá modulárne, s jasným oddelením používateľského rozhrania, aplikačnej logiky, workflow enginu a integračných komponentov, čo umožňuje flexibilné nasadenie v on-premise aj hybridných prostrediach.

2.4.1.1 Frontend a Backend vrstva

Používateľské rozhranie (Frontend) je realizované ako webová aplikácia, ktorá slúži na návrh, správu a spúšťanie playbookov. Frontend komunikuje s backendovou časťou výhradne prostredníctvom REST API, čím je zabezpečené oddelenie prezentačnej vrstvy od aplikačnej logiky.

Backend je implementovaný v jazyku Go a predstavuje riadiacu časť systému. Zodpovedá za:

- správu playbookov,
- spracovanie požiadaviek z frontendu,
- komunikáciu s workflow engine,
- integráciu s externými systémami a (voliteľne) Shuffle Cloud.

Tento dizajn umožňuje, aby bol frontend plne nahraditeľný alebo automaticky ovládaný externými systémami (napr. SIEM) bez zásahu do jadra aplikácie.

2.4.1.2 Workflow engine, Worker a Orborus

Samotné vykonávanie playbook-ov je oddelené od backend-u a realizované pomocou worker komponentov. Worker je zodpovedný za spúšťanie jednotlivých krokov workflowu a môže vykonávať viacero úloh paralelne. Tento prístup zvyšuje škálovateľnosť a zabraňuje tomu, aby dlhšie bežiace alebo blokujúce operácie ovplyvnili dostupnosť backendu.

Komponent Orborus plní úlohu plánovača a spúšťača workflowov. Čaká na úlohy z backendu a zabezpečuje ich spustenie v príslušnom worker prostredí. Vďaka tomuto rozdeleniu je možné horizontálne škálovať vykonávaciu vrstvu bez zmeny architektúry aplikácie.

2.4.1.3 Aplikačné konektory (Apps)

Integrácie s externými nástrojmi sú realizované prostredníctvom tzv. Apps, ktoré sú implementované prevažne v jazyku Python a využívajú štandardizované API rozhrania (napr. OpenAPI). Apps predstavujú konkrétne akcie, ktoré môže playbook vykonávať, napríklad volanie REST API, spracovanie dát alebo reakčné operácie.

Oddelenie Apps od jadra systému umožňuje:

- jednoduché rozširovanie funkcionality,
- izoláciu integračnej logiky,
- bezpečnejšie vykonávanie akcií bez zásahu do core komponentov.

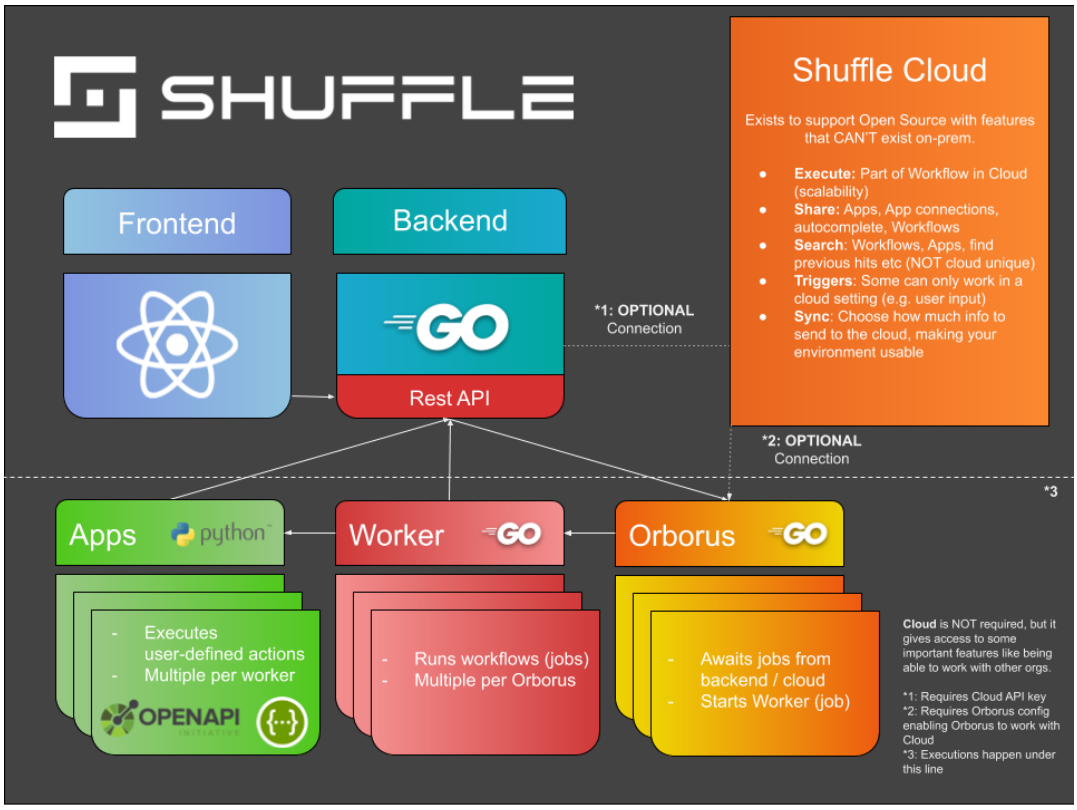


Schéma nástroja Shuffle z oficiálnej webovej stránky

2.5 Inštalácia – variant Docker

```
sudo apt update
sudo apt install -y ca-certificates curl gnupg lsb-release

sudo mkdir -p /etc/apt/keyrings
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o
/etc/apt/keyrings/docker.gpg

echo \
"deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.gpg] \
https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable" \
| sudo tee /etc/apt/sources.list.d/docker.list > /dev/null

sudo apt update
sudo apt install -y docker-ce docker-ce-cli containerd.io docker-compose-plugin
```

```
docker --version
docker compose version

mkdir /opt/shuffle
cd /opt/shuffle

YAML súbor

version: "3.8"
services:
  shuffle:
    image: shuffle/shuffle:latest
    container_name: shuffle
    restart: unless-stopped
    ports:
      - "3001:3001"
    environment:
      - SHUFFLE_HOST=0.0.0.0
      - SHUFFLE_PORT=3001
    volumes:
      - shuffle_data:/shuffle
    depends_on:
      - shuffle-db

  shuffle-db:
    image: postgres:14
    container_name: shuffle-db
    restart: unless-stopped
    environment:
      POSTGRES_USER=shuffle
      POSTGRES_PASSWORD=shufflepassword
      POSTGRES_DB=shuffle
    volumes:
      - shuffle_db:/var/lib/postgresql/data

volumes:
  shuffle_data:
  shuffle_db:

docker compose up -d
```

Webové rozhranie Shuffle je dostupné na adrese: http://<IP_adresa_servera>:3001

Pri prvom prihlásení je potrebné:

- vytvoriť administrátorský účet
- nastaviť organizáciu
- nakonfigurovať základné API prístupy

Reštart systému – upozornenie:

Príkaz: *docker compose down -v* **odstráni aj perzistentné úložiská**, čo vedie k strate konfigurácie a workflowov. Na bežný reštart sa odporúča používať iba:

```
docker compose down
docker compose up -d
```

2.6 Inštalácia – variant Openstack

Pri variante Openstack, bez využitia kontajnerizácie, bude nástroj Shuffle nainštalovaný na virtuálny stroj (Virtual Machine - VM) so systémom Ubuntu 22.04.

1. Aktualizácia systému

```
sudo apt update && sudo apt upgrade –y
```

2. Inštalácia závislostí

```
sudo apt install -y \
git \
python3 \
python3-pip \
python3-venv \
nodejs \
npm \
postgresql \
postgresql-contrib
```

3. Nastavenie PosgreSQL databázy

```
sudo -u postgres psql

CREATE DATABASE shuffle;
CREATE USER shuffle WITH PASSWORD 'shufflepassword';
GRANT ALL PRIVILEGES ON DATABASE shuffle TO shuffle;
\q
```

4. Stiahnutie Shuffle zo zdrojového kódu

```
cd /opt
sudo git clone https://github.com/shuffle/shuffle.git
sudo chown -R $USER:$USER shuffle
cd shuffle
```

5. Backend – Python časť

```
python3 -m venv venv
source venv/bin/activate
pip install --upgrade pip
```

```
pip install -r requirements.txt
```

Nastavenie databázového pripojenia v konfiguračnom súbore (podľa dokumentácie Shuffle):

```
export DATABASE_HOST=localhost
export DATABASE_NAME=shuffle
export DATABASE_USER=shuffle
export DATABASE_PASSWORD=shufflepassword
```

Spustenie backendu:

```
python3 app.py
```

6. Frontend – Webové rozhranie

```
cd frontend
npm install
npm run build
npm run start
```

7. Prístup do aplikácie

Po úspešnom spustení backendu a frontendu je Shuffle dostupný cez webový prehliadač na definovanom porte (štandardne 3001).

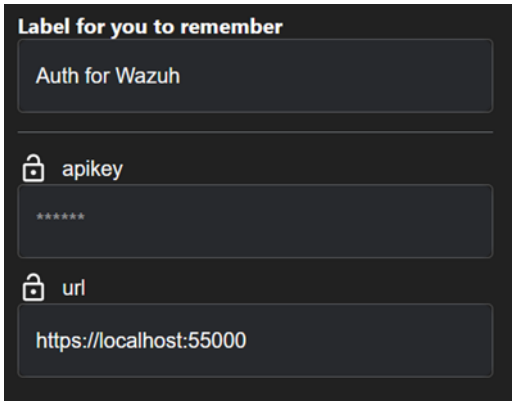
2.7 Základná konfigurácia

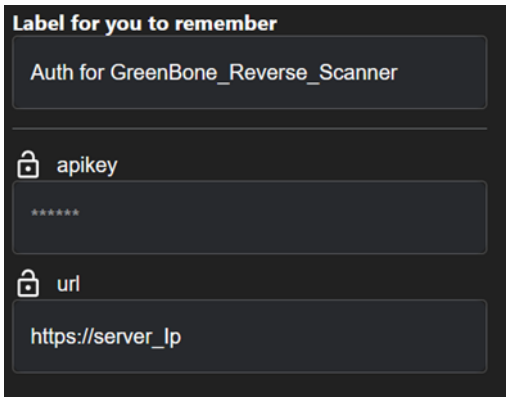
Shuffle je navrhnutý ako flexibilný orchestrátor, ktorého konfigurácia prebieha prevažne prostredníctvom webového rozhrania. Základná konfigurácia sa vykonáva bezprostredne po inštalácii a zahŕňa nastavenia používateľov, bezpečnostných mechanizmov, integrácií a samotnej automatizačnej logiky.

Po prvom prihlásení je potrebné vytvoriť administrátorský účet a definovať organizačnú štruktúru. Následne sa nastavujú používateľské roly a oprávnenia, ktoré určujú, kto môže vytvárať a upravovať workflowy, spravovať integrácie alebo spúšťať playbooky. Shuffle poskytuje základný model riadenia prístupov, ktorý je postačujúci pre menšie a stredne veľké tímy.

Dôležitou časťou konfigurácie je správa aplikácií a konektorov. Každá integrovaná služba je v Shuffle reprezentovaná ako aplikácia, ku ktorej sa konfigurujú autentifikačné údaje, najčastejšie vo forme API kľúčov, tokenov alebo OAuth mechanizmov. Tieto údaje sú uložené centrálne a následne používané v jednotlivých workflowoch, čo znižuje potrebu opakovaného zadávania citlivých informácií.

2.7.1 Pripojenia na externé aplikácie Wazuh a Greenbone





Konfigurácia workflowov prebieha vo vizuálnom editore, kde sa definujú jednotlivé kroky automatizácie. Každý krok predstavuje konkrétnu akciu, napríklad volanie externého API, spracovanie odpovede, podmienené rozhodovanie alebo odoslanie notifikácie. Shuffle umožňuje pracovať s premennými, ktoré sa prenášajú medzi krokmi workflowu, čo umožňuje dynamické spracovanie dát a tvorbu komplexných reakčných scenárov.

Workflow na posielanie notifikácií do komunikačnej platformy Discord.



2.7.2 Vstupy a výstupy

Vstupy

Vstupom do Shuffle sú predovšetkým udalosti a dáta prichádzajúce z externých systémov. Najčastejším spôsobom vstupu je využitie webhookov alebo REST API, prostredníctvom ktorých môžu bezpečnostné nástroje odosielať alerty, logy alebo incidentné informácie. Typickými zdrojmi vstupných dát sú SIEM systémy, log manažment nástroje, EDR riešenia alebo iné bezpečnostné platformy.

Okrem automatických vstupov je možné workflow spúšťať aj manuálne, napríklad analytikom pri vyšetrovaní incidentu. Vstupné dáta môžu obsahovať rôzne typy informácií, ako sú IP adresy, domény, URL adresy, hash hodnoty súborov, identifikátory incidentov alebo celé JSON štruktúry s detailným popisom udalosti.

Výstupy

Výstupom zo Shuffle sú vykonané automatizované akcie alebo generované výsledky, ktoré sú odosielané späť do externých systémov. Medzi typické výstupy patrí vytváranie tiketov v nástrojoch na správu incidentov, odosielanie notifikácií prostredníctvom komunikačných platforiem, aktualizácia záznamov v bezpečnostných systémoch alebo spúšťanie reakčných opatrení, ako je blokovanie IP adries alebo deaktivácia používateľských účtov.

Výstupy môžu mať podobu API volaní, správ, logov alebo štruktúrovaných dát, ktoré sú následne využívané ďalšími nástrojmi v bezpečnostnom ekosystéme. Shuffle tak plní úlohu centrálneho orchestration bodu, ktorý prepája vstupné udalosti s automatizovanými reakciami.

2.8 Integrácie

Shuffle podporuje integráciu s širokým spektrom bezpečnostných a IT nástrojov prostredníctvom aplikačných konektorov (Apps) a otvoreného REST API. Integrácie sú navrhnuté tak, aby Shuffle fungoval ako centrálna orchestračná a automatizačná vrstva, ktorá prepája existujúce komponenty SOC bez potreby ich priamej úpravy.

Medzi typické integračné scenáre patria:

- SIEM nástroje (napr. Elastic Stack, Graylog) ako zdroj udalostí a triggerov,
- kontextové systémy (napr. NetBox, OpenCTI) pre obohacovanie incidentov,
- externé služby (VirusTotal, WHOIS, DNS služby) pre enrichment,
- komunikačné a ticketovacie nástroje (Slack, Jira, e-mail),
- repozitáre a úložiská (GitHub, Git) na uchovávanie výstupov a auditných artefaktov.

Vďaka otvorenému API a modularite konektorov je možné vytvárať aj vlastné integrácie, čo umožňuje prispôsobiť Shuffle špecifickým potrebám organizácie alebo experimentálneho PoC prostredia.

Pozícia Shuffle v SOC

Z pohľadu SOC architektúry táto vnútorná štruktúra umožňuje, aby Shuffle fungoval ako samostatná reakčná a automatizačná vrstva, ktorá:

- prijíma udalosti z detekčných systémov (napr. SIEM),
- vykonáva obohacovanie a rozhodovaciu logiku,
- spúšťa reakčné kroky prostredníctvom externých nástrojov.

Oddelenie riadenia (backend) od vykonávania (workers) znižuje riziko, že automatizácia negatívne ovplyvní stabilitu SOC infraštruktúry, a podporuje bezpečné a

kontrolované nasadenie kde ELK Stack deteguje, NetBox poskytuje kontext a Shuffle zabezpečuje automatizovanú reakciu.

```
[ Zdroje udalostí ]
   | (syslog / beats / IDS)
   v
[ ELK Stack ]
  - Elasticsearch (indexy)
  - Kibana (analýza/alert)
        |
        | (webhook / alert action / manuálne spustenie)
        v
[ Shuffle (SOAR) ]
  - Web UI (playbooky)
  - Workflow engine (vykonanie krokov)
  - Interná DB (stav, výsledky)
        |
        |----> [ Kontext / Enrichment ]
        |        - NetBox (asset/IP/topológia) cez API
        |        - OSINT/WHOIS/DNS (voliteľne)
        |
        |----> [ Response / Actions ]
                 - Notifikácia (email/chat)
                 - Ticketing (Jira/GLPI/…)
                 - Network action (FW/ACL) (voliteľne)
                 - Git/artefakty (log výstupov) (voliteľne)
```

Kde zamýšľaný tok spracovania udalosti je:

1. Detection (ELK): Udalosti (napr. syslog, IDS alert, Windows event) sa zbierajú do ELK. Analytik alebo pravidlo identifikuje zaujímavú udalosť.
2. Trigger (Shuffle): Shuffle sa spúšťa buď:
   - manuálne (analytik vyberie incident a spustí playbook),
   - alebo automaticky webhookom/alert action (PoC režim).
3. Enrichment (kontext): Playbook doplní incident o kontext:
   - NetBox: „komu patrí IP“, „aké zariadenie“, „lokalita“, „segment/VLAN“,
   - voliteľne DNS/WHOIS/GeoIP, reputácia.
4. Response (akcia): Na základe výsledkov (podmienky v playbooku):
   - pošle notifikáciu,
   - otvorí ticket,
   - alebo vykoná jednoduchú reakciu (napr. blokovať IP na FW).
5. Audit/Output: Shuffle uloží stav workflow-u a výsledky krokov (záznamy sú dohľadateľné pre audit a spätnú analýzu).

3 WAZUH

Typ nástroja: Endpoint security/SIEM/EDR/FIM

Názov nástroja: Wazuh

Verzia: Wazuh 4.14 (open-source, kontajnerizované nasadenie)

3.1 Opis systému

Wazuh je open-source bezpečnostná platforma zameraná na monitorovanie koncových bodov, detekciu bezpečnostných udalostí a základnú reakciu, pričom v prostredí SOC plní úlohu kombinácie EDR, SIEM a FIM riešenia. Jeho hlavným účelom je zber, korelácia a vyhodnocovanie bezpečnostne relevantných dát z operačných systémov, aplikácií a infraštruktúrnych komponentov a ich následné poskytovanie do centrálneho SIEM prostredia.

Zber dát je realizovaný predovšetkým prostredníctvom Wazuh agentov, ktoré sú inštalované na koncových systémoch (Linux, Windows, macOS). Tieto agenty umožňujú:

- zber systémových a aplikačných logov,
- monitorovanie integrity súborov (File Integrity Monitoring – FIM),
- audit systémových udalostí,
- základnú detekciu anomálií a hrozieb na úrovni endpointu.

Okrem agentového zberu dokáže Wazuh prijímať údaje aj prostredníctvom Syslog, čo umožňuje centralizovaný zber logov zo sieťových zariadení, bezpečnostných prvkov alebo iných systémov, kde nie je možné alebo vhodné nasadiť agenta. Tento prístup rozširuje použiteľnosť Wazuhu aj na pasívne alebo embedded zariadenia.

Wazuh obsahuje vlastný korelačný a detekčný engine, ktorý využíva rozsiahlu sadu pravidiel založených na známkach kompromitácie, behaviorálnych vzoroch a mapovaní na MITRE ATT&CK techniky. Pravidlá sú pravidelne aktualizované komunitou a poskytujú okamžitú použiteľnosť bez potreby manuálnej tvorby detekčných pravidiel.

Webové používateľské rozhranie Wazuhu podporuje role-based access control (RBAC), správu agentov, prehľad udalostí a integráciu s vizualizačnou vrstvou (Kibana). V navrhovanej SOC architektúre Wazuh vystupuje ako zdroj detekčných udalostí na úrovni endpointov, ktoré sú následne posielané do centrálneho SIEM (ELK Stack) na ďalšiu koreláciu a analýzu.

Jeho hlavnou silou je kombinácia EDR vlastností s prehľadným SIEM prístupom, čo z neho robí vhodné riešenie najmä pre menšie a stredne veľké SOC tímy, ktoré potrebujú pokryť endpoint security aj základnú koreláciu bez nasadenia viacerých samostatných nástrojov.

3.2 Dokumentácia systému

Oficiálna dokumentácia platformy Wazuh je verejne dostupná a poskytuje komplexný prehľad architektúry systému, možností zberu dát, detekčných mechanizmov a integračných scenárov. Dokumentácia zahŕňa návody na inštaláciu agentov, konfiguráciu serverových komponentov, správu pravidiel a integráciu s externými SIEM nástrojmi.

Dokumentácia je orientovaná prakticky a pokrýva aj oblasti ako:

- konfigurácia FIM a auditných modulov,
- správa agentov a ich životného cyklu,
- práca s MITRE ATT&CK mapovaním,
- integrácia s Elasticsearch a Kibana.

Odkazy na oficiálne zdroje:

- Oficiálna dokumentácia Wazuh: https://documentation.wazuh.com/
- Oficiálna stránka projektu: https://wazuh.com/
- GitHub repozitár: https://github.com/wazuh/wazuh
- Oficiálne Docker image: https://hub.docker.com/u/wazuh

3.3 Požiadavky na zdroje

Hardvérové požiadavky Wazuhu závisia od počtu monitorovaných endpointov, objemu generovaných udalostí a miery zapojenia korelačných pravidiel. Keďže Wazuh kombinuje viacero bezpečnostných funkcií, jeho nároky sú vyššie ako pri jednoduchých logových kolektoroch.

Minimálne požiadavky (PoC / menšie prostredie):

- CPU: 2–4 jadrá
- RAM: 4–8 GB
- Disk: 50 GB (logy a indexy)
- Počet agentov: desiatky

Odporúčané požiadavky (produkčné prostredie):

- CPU: 8+ jadier
- RAM: 16–32 GB
- Disk: SSD (v závislosti od retencie dát)
- Počet agentov: stovky a viac

Zvýšené nároky vznikajú najmä pri:

- intenzívnom FIM monitoringu,
- vysokom objeme auditných udalostí,
- dlhodobej retencii dát v SIEM.

3.4 Architektúra nástroja

Architektúra Wazuhu je navrhnutá ako viacvrstvové distribuované riešenie, ktoré oddeľuje zber dát, detekciu a prezentáciu výsledkov. Základ architektúry tvoria tri hlavné komponenty: agenti, serverová časť a vizualizačná vrstva.

Agenti sú nasadené na koncových systémoch a zodpovedajú za zber bezpečnostných dát. Serverová časť Wazuhu spracúva prijaté udalosti, aplikuje korelačné pravidlá a generuje bezpečnostné alerty. Výsledné udalosti sú následne odosielané do Elasticsearch, kde sú indexované a sprístupnené pre ďalšiu analýzu.

Vizualizačná vrstva je realizovaná prostredníctvom webového rozhrania a integrácie s Kibana dashboardmi, ktoré umožňujú analytikom sledovať stav endpointov, detekcie a trendy v čase. V SOC architektúre je Wazuh umiestnený na rozhraní medzi endpointmi a SIEM vrstvou:

- na jednej strane zabezpečuje detailnú detekciu na úrovni koncových bodov,
- na druhej strane poskytuje normalizované a korelované udalosti do centrálneho ELK Stacku.

Týmto spôsobom Wazuh výrazne rozširuje detekčné schopnosti SOC bez potreby nasadzovať samostatné EDR a SIEM riešenia pre základné použitie.

Schéma

Architektúra Wazuhu je založená na distribuovanom modeli, v ktorom sú jasne oddelené komponenty zberu dát, detekcie, korelácie a vizualizácie. Tento prístup umožňuje škálovanie riešenia podľa počtu endpointov a objemu bezpečnostných udalostí a zároveň udržiava prehľadnú integračnú väzbu na SIEM vrstvu.

Základ architektúry tvoria:

- Wazuh agenti, nasadení na koncových systémoch,
- Wazuh server (manager), ktorý vykonáva koreláciu a detekciu,
- indexačná a analytická vrstva (Elasticsearch),
- vizualizačná vrstva (Kibana / Wazuh UI).

Agenti zbierajú bezpečnostne relevantné dáta (logy, audit, FIM udalosti) a odosielajú ich do Wazuh servera. Server aplikuje korelačné pravidlá a mapovanie na MITRE ATT&CK a výsledné alerty odosiela do SIEM vrstvy. Vizualizačná časť umožňuje analytikom sledovať stav endpointov a detekcií v reálnom čase.

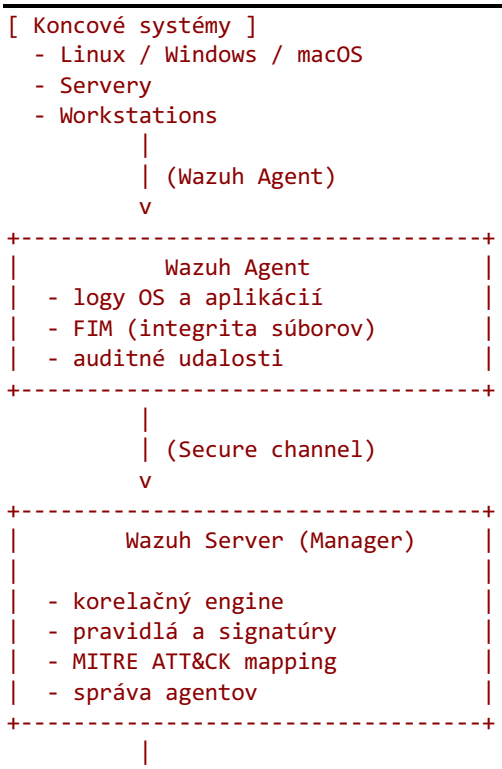




Schéma ilustruje, že Wazuh je **primárnym zdrojom endpointových detekcií**, ktoré sú:

- spracované a korelované na úrovni Wazuh servera,
- odoslané do centrálneho SIEM (ELK Stack),
- vizualizované v Kibane,
- využité pre reakciu (SOAR) a incident management.

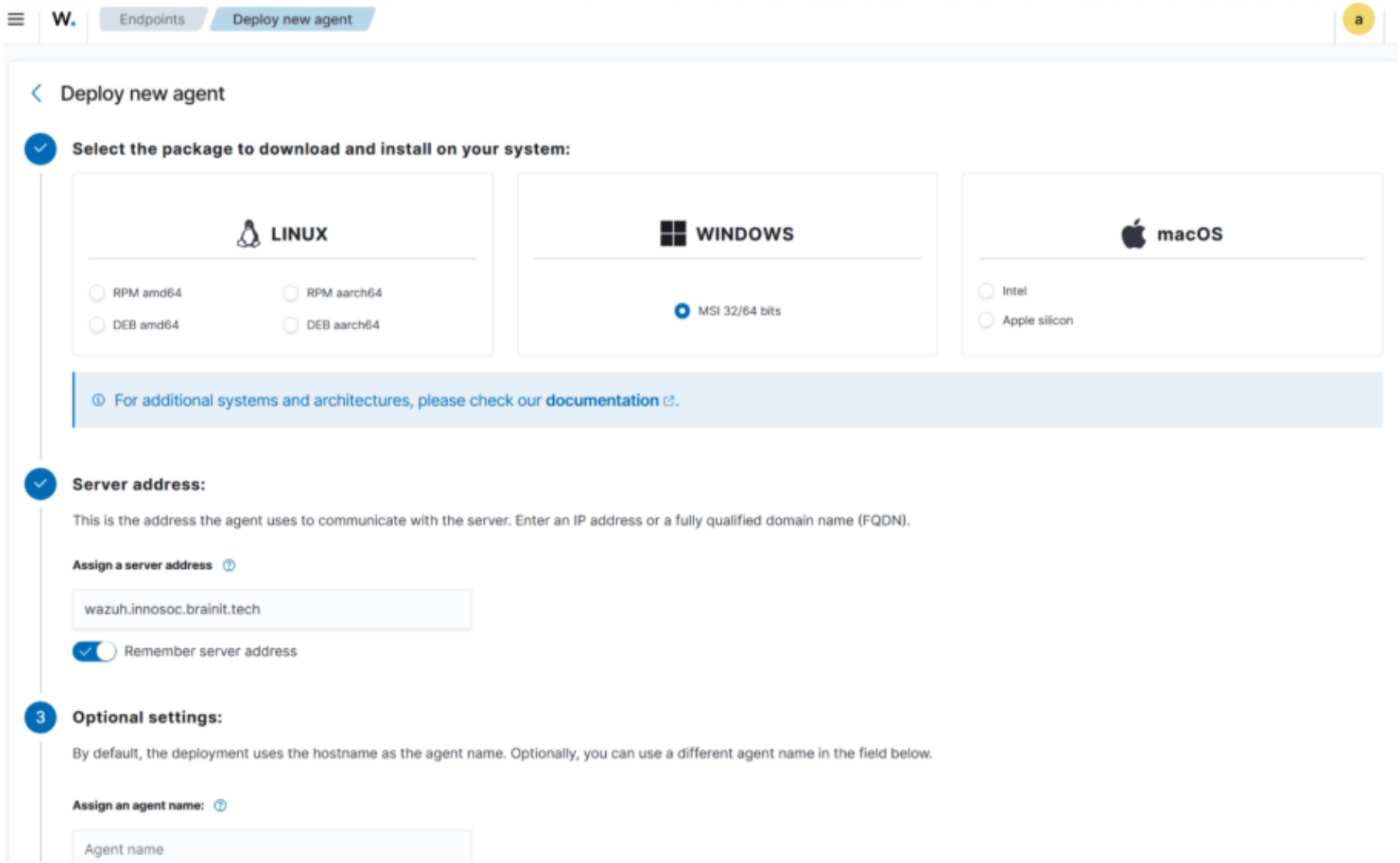
Wazuh tak v SOC architektúre vypĺňa **medzeru medzi surovým zberom logov a vyššou koreláciou**, pričom poskytuje detailný pohľad na stav a správanie koncových bodov.

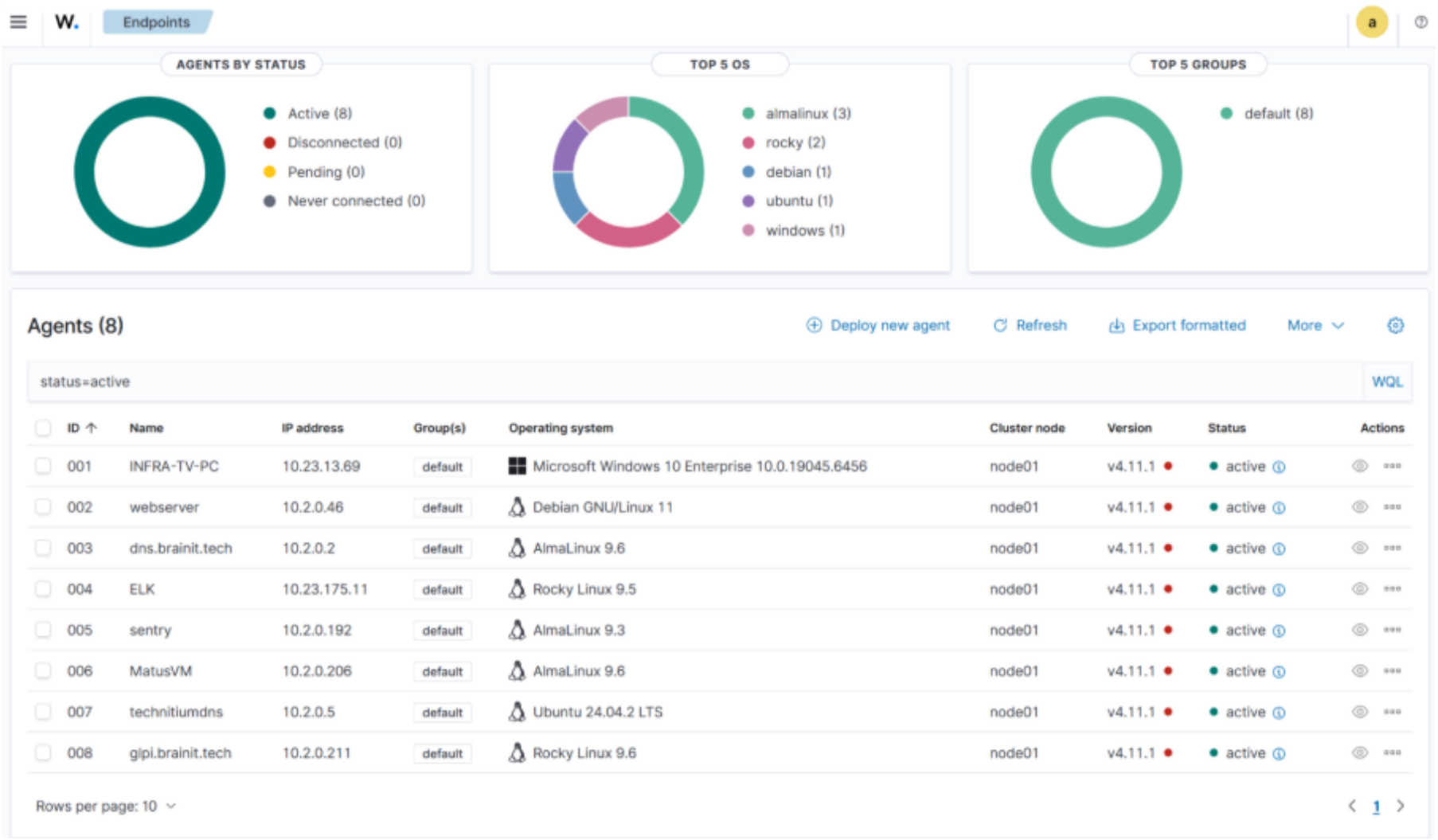
3.5 Inštalácia – variant Quickstart

Nainštalovaný na rocky linux 9 VM podľa oficiálnej quickstart dokumentácie https://documentation.wazuh.com/current/quickstart.html

3.6 Základná konfigurácia

Wazuh má vstavaný systém na poskytovanie inštalátora wazuh agent a onboarding klientskych staníc:


W.
Endpoints
Deploy new agent
a
Deploy new agent
Select the package to download and install on your system:
LINUX
RPM amd64
RPM aarch64
DEB amd64
DEB aarch64
WINDOWS
MSI 32/64 bits
macOS
Intel
Apple silicon
For additional systems and architectures, please check our documentation.
Server address:
This is the address the agent uses to communicate with the server. Enter an IP address or a fully qualified domain name (FQDN).
Assign a server address
wazuh.innosoc.brainit.tech
Remember server address
3
Optional settings:
By default, the deployment uses the hostname as the agent name. Optionally, you can use a different agent name in the field below.
Assign an agent name:
Agent name


Endpoints
AGENTS BY STATUS
Active (8)
Disconnected (0)
Pending (0)
Never connected (0)
TOP 5 OS
almalinux (3)
rocky (2)
debian (1)
ubuntu (1)
windows (1)
TOP 5 GROUPS
default (8)
Agents (8)
Deploy new agent
Refresh
Export formatted
More
status=active
WQL
ID
Name
IP address
Group(s)
Operating system
Cluster node
Version
Status
Actions
001 INFRA-TV-PC 10.23.13.69 default Microsoft Windows 10 Enterprise 10.0.19045.6456 node01 v4.11.1 active
002 webserver 10.2.0.46 default Debian GNU/Linux 11 node01 v4.11.1 active
003 dns.brainit.tech 10.2.0.2 default AlmaLinux 9.6 node01 v4.11.1 active
004 ELK 10.23.175.11 default Rocky Linux 9.5 node01 v4.11.1 active
005 sentry 10.2.0.192 default AlmaLinux 9.3 node01 v4.11.1 active
006 MatusVM 10.2.0.206 default AlmaLinux 9.6 node01 v4.11.1 active
007 technitiumdns 10.2.0.5 default Ubuntu 24.04.2 LTS node01 v4.11.1 active
008 glpi.brainit.tech 10.2.0.211 default Rocky Linux 9.6 node01 v4.11.1 active
Rows per page: 10
1

3.7 Integrácie

Wazuh je možné integrovať s rôznymi SIEM systémami vrátane elastic search, bol preto integrovaný do ELK stacku:

- https://documentation.wazuh.com/current/integrations-guide/index.html
- https://documentation.wazuh.com/current/integrations-guide/elastic-stack/index.html

4 GREENBONE COMMUNITY EDITION

Typ nástroja: Vulnerability Management/Vulnerability Assessment

Názov nástroja: Greenbone Community Edition (GCE)

Verzia: Version 24.10 Open-source edícia (OpenVAS/Greenbone Vulnerability Management)

4.1 Opis systému

Greenbone Community Edition (GCE) je open-source nástroj určený na pravidelné skenovanie infraštruktúry a služieb a reporting zraniteľností, ktorý v prostredí SOC plní úlohu základného komponentu pre riadenie zraniteľností (Vulnerability Management). Slúži na identifikáciu technických slabín systémov ešte pred ich potenciálnym zneužitím a poskytuje podklad pre preventívne bezpečnostné opatrenia.

GCE je bezplatná open-source verzia riešenia od spoločnosti Greenbone Networks, založená na nástroji OpenVAS (Open Vulnerability Assessment System). Poskytuje vysokokapacitné skenovanie s využitím rozsiahlej databázy zraniteľností dostupnej prostredníctvom komunitného GVMD feedu, ktorý obsahuje pravidelne aktualizované testy zraniteľností (NVT). Tieto testy pokrývajú známe zraniteľnosti, nesprávne konfigurácie, slabé kryptografické nastavenia a zastarané služby.

Systém podporuje plánované a opakované skeny, čím umožňuje dlhodobé sledovanie bezpečnostného stavu infraštruktúry a identifikáciu trendov v oblasti zraniteľností. Výsledky skenovania sú spracované v modernej webovej aplikácii Greenbone Security Assistant (GSA), ktorá umožňuje filtrovanie, triedenie a vyhodnocovanie nálezov podľa závažnosti (CVSS), aktíva alebo typu zraniteľnosti. Výstupy je možné generovať vo viacerých formátoch, najmä CSV, XML a PDF, čo umožňuje ich ďalšie spracovanie, archiváciu alebo prezentáciu manažmentu.

Greenbone Community Edition podporuje štandardy ako CVE a CPE, čím zabezpečuje kompatibilitu s ostatnými bezpečnostnými nástrojmi a metodikami hodnotenia rizík. Z pohľadu integrácie ponúka viacero mechanizmov, vrátane GMP (Greenbone Management Protocol), ako aj podpory pre SSH, LDAP, RADIUS a ďalšie autentifikačné a integračné rozhrania. Tieto možnosti umožňujú prepojenie s existujúcimi systémami správy identít, automatizačnými nástrojmi alebo ticketing systémami.

Pre rozsiahle alebo distribuované prostredia GCE podporuje aj master–sensor architektúru, v ktorej je centrálna riadiaca inštancia doplnená o viaceré skenovacie senzory. Tento prístup umožňuje škálovanie skenovania, zníženie sieťovej záťaže a pokrytie oddelených segmentov infraštruktúry bez nutnosti centrálneho skenovania cez bezpečnostné hranice.

Z pohľadu SOC Greenbone Community Edition nevykonáva detekciu v reálnom čase ani reakciu na incidenty. Jeho úlohou je poskytovať systematický a opakovateľný pohľad na úroveň zraniteľností aktív, ktorý dopĺňa detekčné nástroje (SIEM/EDR) o preventívny bezpečnostný kontext. Výsledky skenov slúžia ako vstup pre analýzu rizík, plánovanie nápravných opatrení a podporu rozhodovania v oblasti bezpečnostnej politiky organizácie.

4.2 Dokumentácia systému

Greenbone Community Edition má dostupnú oficiálnu dokumentáciu, ktorá popisuje architektúru riešenia, spôsob nasadenia, konfiguráciu skenov a správu výsledkov. Dokumentácia sa primárne zameriava na technickú stránku nástroja a jeho komponenty.

Oficiálne zdroje:

- Dokumentácia Greenbone Community Edition: https://greenbone.github.io/docs/
- GitHub repozitár projektu: https://github.com/greenbone
- OpenVAS / GVM dokumentácia: https://www.openvas.org/
- DockerHub repozitáre: https://hub.docker.com/u/greenbone

4.3 Požiadavky na zdroje

Hardvérové nároky Greenbone závisia od:

- počtu skenovaných aktív,
- rozsahu skenov,
- frekvencie spúšťania.

Minimálne požiadavky (PoC / menšie prostredie):

- CPU: 2–4 jadrá
- RAM: 4–8 GB
- Disk: 40–80 GB (databázy, NVT feedy, reporty)

Odporúčané pre väčšie prostredie:

- CPU: 4–8 jadier
- RAM: 8–16 GB
- Disk: SSD úložisko (100 GB a viac)

Vyššie nároky sa uplatňujú najmä pri paralelnom skenovaní viacerých rozsahov a pri uchovávaní historických výsledkov.

4.4 Architektúra nástroja

Architektúra Greenbone Community Edition je založená na modulárnom modeli, kde sú oddelené komponenty pre skenovanie, správu výsledkov a používateľské rozhranie. Tento model umožňuje škálovanie a integráciu do existujúcej SOC infraštruktúry.

Základné komponenty architektúry:

- Scanner (OpenVAS Scanner) – vykonáva samotné testy zraniteľností,
- Manager (gvmd) – riadi skeny, spracúva výsledky a uchováva ich v databáze,
- Databáza (PostgreSQL) – ukladá výsledky skenov a metadáta,
- Web UI (Greenbone Security Assistant – GSA) – rozhranie pre správu a analýzu.

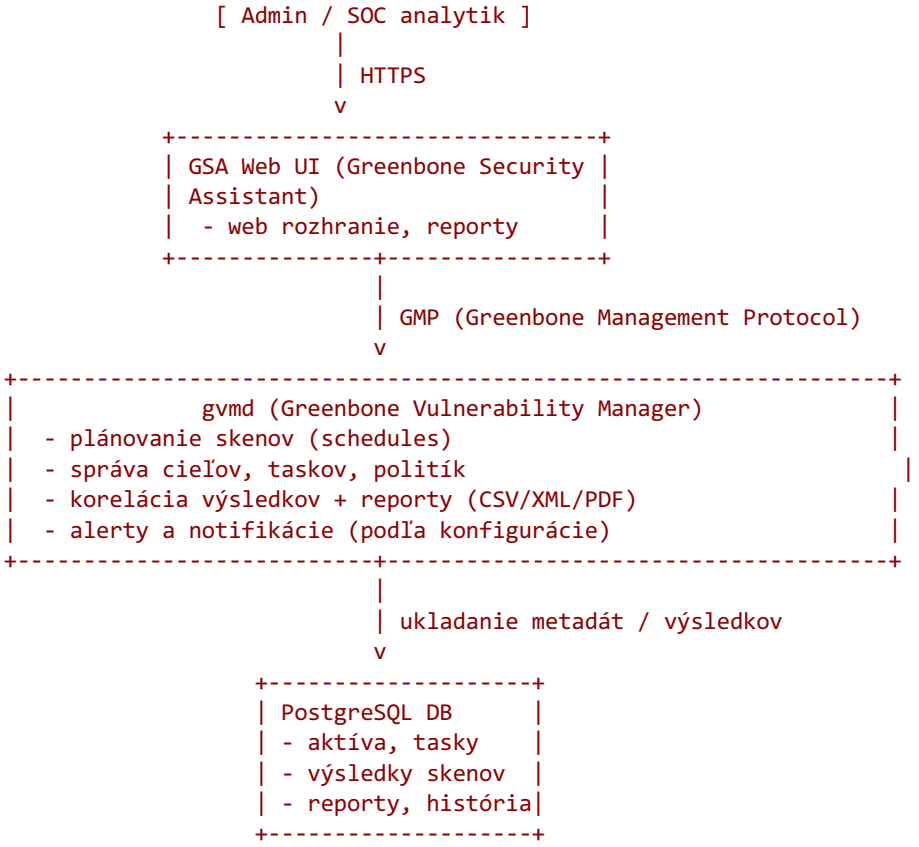
Greenbone komunikuje s cieľovými systémami aktívne prostredníctvom siete (TCP/UDP), pričom skeny môžu byť:

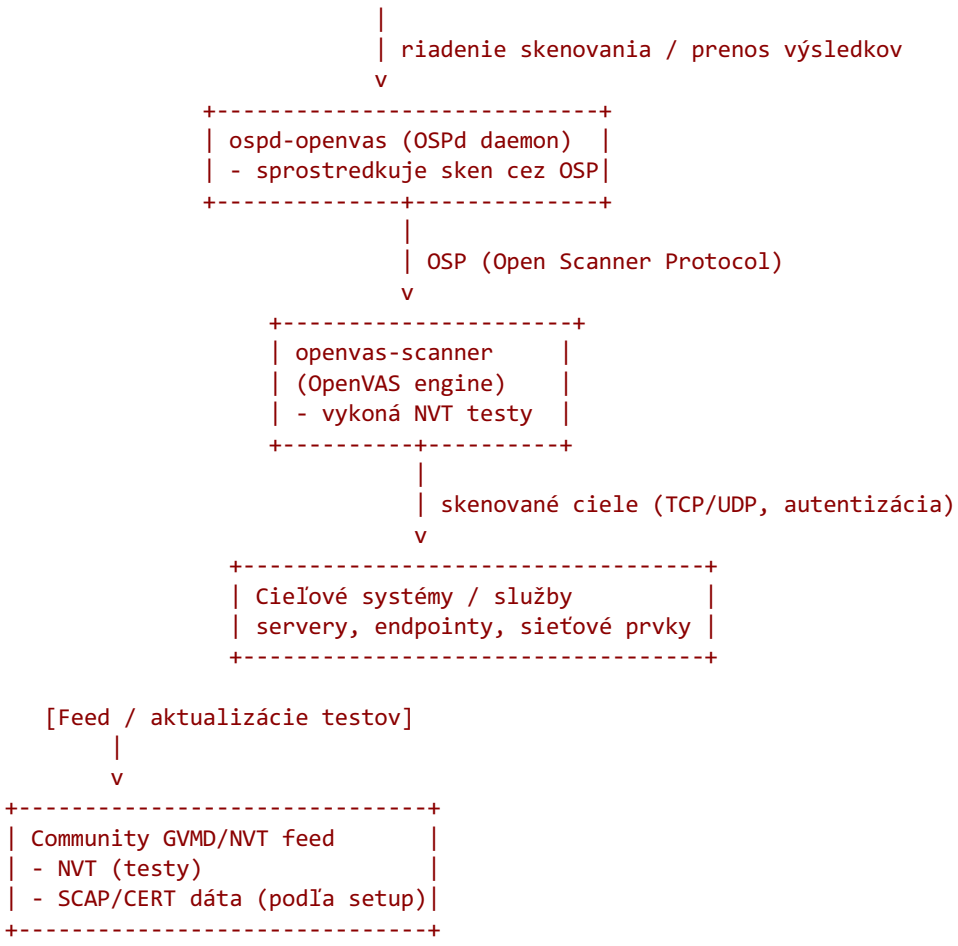
- neautentifikované (black-box),
- autentifikované (white-box, vyššia presnosť).

V SOC architektúre je Greenbone umiestnený mimo produkčného dátového toku, aby neovplyvňoval dostupnosť služieb, a jeho výstupy sú využívané ako doplnkový bezpečnostný kontext.

Schéma

Schéma znázorňuje referenčnú architektúru Greenbone Community Edition (GVM/OpenVAS), ktorá je založená na oddelení riadiacej, skenovacej a prezentačnej vrstvy. Webové rozhranie GSA slúži ako vstupný bod pre analytikov SOC a komunikuje s komponentom gvmd, ktorý zabezpečuje plánovanie skenov, správu cieľov a vyhodnocovanie výsledkov. Samotné skenovanie je realizované komponentom openvas-scanner, ktorý je riadený prostredníctvom ospd-openvas a vykonáva testy zraniteľností nad cieľovými systémami. Výsledky skenovania sú ukladané do databázy PostgreSQL a prezentované vo forme reportov a alertov. Architektúra podporuje pravidelné a opakovateľné skenovanie infraštruktúry, čím poskytuje SOC preventívny pohľad na bezpečnostné riziká aktív.





4.5 Inštalácia – variant Docker

Použitý docker compose súbor:

```
services:
  vulnerability-tests:
    image: registry.community.greenbone.net/community/vulnerability-tests
    environment:
      FEED_RELEASE: "24.10"
      KEEP_ALIVE: 1
    volumes:
      - vt_data_vol:/mnt

  notus-data:
    image: registry.community.greenbone.net/community/notus-data
    environment:
      KEEP_ALIVE: 1
    volumes:
      - notus_data_vol:/mnt

  scap-data:
    image: registry.community.greenbone.net/community/scap-data
```

```
  environment:
    KEEP_ALIVE: 1
  volumes:
    - scap_data_vol:/mnt

cert-bund-data:
  image: registry.community.greenbone.net/community/cert-bund-data
  environment:
    KEEP_ALIVE: 1
  volumes:
    - cert_data_vol:/mnt

dfn-cert-data:
  image: registry.community.greenbone.net/community/dfn-cert-data
  environment:
    KEEP_ALIVE: 1
  volumes:
    - cert_data_vol:/mnt
  depends_on:
    cert-bund-data:
      condition: service_healthy

data-objects:
  image: registry.community.greenbone.net/community/data-objects
  environment:
    FEED_RELEASE: "24.10"
    KEEP_ALIVE: 1
  volumes:
    - data_objects_vol:/mnt

report-formats:
  image: registry.community.greenbone.net/community/report-formats
  environment:
    FEED_RELEASE: "24.10"
    KEEP_ALIVE: 1
  volumes:
    - data_objects_vol:/mnt
  depends_on:
    data-objects:
      condition: service_healthy

gpg-data:
  image: registry.community.greenbone.net/community/gpg-data
  volumes:
    - gpg_data_vol:/mnt

redis-server:
  image: registry.community.greenbone.net/community/redis-server
  restart: on-failure
  volumes:
    - redis_socket_vol:/run/redis/

pg-gvm:
  image: registry.community.greenbone.net/community/pg-gvm:stable
  restart: on-failure
  volumes:
    - psql_data_vol:/var/lib/postgresql
```

```
      - psql_socket_vol:/var/run/postgresql

  gvmd:
    image: registry.community.greenbone.net/community/gvmd:stable
    restart: on-failure
    volumes:
      - gvmd_data_vol:/var/lib/gvm
      - scap_data_vol:/var/lib/gvm/scap-data/
      - cert_data_vol:/var/lib/gvm/cert-data
      - data_objects_vol:/var/lib/gvm/data-objects/gvmd
      - vt_data_vol:/var/lib/openvas/plugins
      - psql_data_vol:/var/lib/postgresql
      - gvmd_socket_vol:/run/gvmd
      - ospd_openvas_socket_vol:/run/ospd
      - psql_socket_vol:/var/run/postgresql
    depends_on:
      pg-gvm:
        condition: service_started
      scap-data:
        condition: service_healthy
      cert-bund-data:
        condition: service_healthy
      dfn-cert-data:
        condition: service_healthy
      data-objects:
        condition: service_healthy
      report-formats:
        condition: service_healthy

  gsa:
    image: registry.community.greenbone.net/community/gsa:stable
    restart: on-failure
    ports:
      - 127.0.0.1:9392:80
    volumes:
      - gvmd_socket_vol:/run/gvmd
    depends_on:
      gvmd:
        condition: service_started
  # Sets log level of openvas to the set LOG_LEVEL within the env
  # and changes log output to /var/log/openvas instead /var/log/gvm
  # to reduce likelyhood of unwanted log interferences
  configure-openvas:
    image: registry.community.greenbone.net/community/openvas-scanner:stable
    volumes:
      - openvas_data_vol:/mnt
      - openvas_log_data_vol:/var/log/openvas
    command:
      - /bin/sh
      - -c
      - |
        printf "table_driven_lsc = yes\nopenvasd_server = http://openvasd:80\n" >
/mnt/openvas.conf
        sed "s/127/128/" /etc/openvas/openvas_log.conf | sed 's/gvm/openvas/' >
/mnt/openvas_log.conf
        chmod 644 /mnt/openvas.conf
        chmod 644 /mnt/openvas_log.conf
```

```
      touch /var/log/openvas/openvas.log
      chmod 666 /var/log/openvas/openvas.log

# shows logs of openvas
openvas:
  image: registry.community.greenbone.net/community/openvas-scanner:stable
  restart: on-failure
  volumes:
    - openvas_data_vol:/etc/openvas
    - openvas_log_data_vol:/var/log/openvas
  command:
    - /bin/sh
    - -c
    - |
      cat /etc/openvas/openvas.conf
      tail -f /var/log/openvas/openvas.log
  depends_on:
    configure-openvas:
      condition: service_completed_successfully

openvasd:
  image: registry.community.greenbone.net/community/openvas-scanner:stable
  restart: on-failure
  environment:
    # `service_notus` is set to disable everything but notus,
    # if you want to utilize openvasd directly, remove `OPENVASD_MODE`
    OPENVASD_MODE: service_notus
    GNUPGHOME: /etc/openvas/gnupg
    LISTENING: 0.0.0.0:80
  volumes:
    - openvas_data_vol:/etc/openvas
    - openvas_log_data_vol:/var/log/openvas
    - gpg_data_vol:/etc/openvas/gnupg
    - notus_data_vol:/var/lib/notus
  # enable port forwarding when you want to use the http api from your host machine
  # ports:
  #   - 127.0.0.1:3000:80
  depends_on:
    vulnerability-tests:
      condition: service_healthy
    notus-data:
      condition: service_healthy
    configure-openvas:
      condition: service_completed_successfully
    gpg-data:
      condition: service_completed_successfully
  networks:
    default:
      aliases:
        - openvasd

ospd-openvas:
  image: registry.community.greenbone.net/community/ospd-openvas:stable
  restart: on-failure
  hostname: ospd-openvas.local
  cap_add:
    - NET_ADMIN # for capturing packages in promiscuous mode
```

```
      - NET_RAW # for raw sockets e.g. used for the boreas alive detection
    security_opt:
      - seccomp=unconfined
      - apparmor=unconfined
    command:
      [
        "ospd-openvas",
        "-f",
        "--config",
        "/etc/gvm/ospd-openvas.conf",
        "--notus-feed-dir",
        "/var/lib/notus/advisories",
        "-m",
        "666",
      ]
    volumes:
      - gpg_data_vol:/etc/openvas/gnupg
      - vt_data_vol:/var/lib/openvas/plugins
      - notus_data_vol:/var/lib/notus
      - ospd_openvas_socket_vol:/run/ospd
      - redis_socket_vol:/run/redis/
      - openvas_data_vol:/etc/openvas/
      - openvas_log_data_vol:/var/log/openvas
    depends_on:
      redis-server:
        condition: service_started
      gpg-data:
        condition: service_completed_successfully
      configure-openvas:
        condition: service_completed_successfully
      vulnerability-tests:
        condition: service_healthy
      notus-data:
        condition: service_healthy

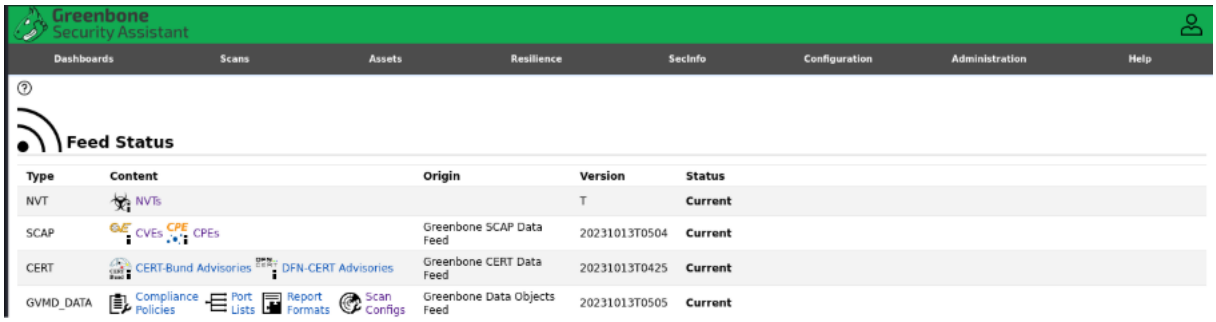
  gvm-tools:
    image: registry.community.greenbone.net/community/gvm-tools
    volumes:
      - gvmd_socket_vol:/run/gvmd
      - ospd_openvas_socket_vol:/run/ospd
    depends_on:
      - gvmd
      - ospd-openvas

volumes:
  gpg_data_vol:
  scap_data_vol:
  cert_data_vol:
  data_objects_vol:
  gvmd_data_vol:
  psql_data_vol:
  vt_data_vol:
  notus_data_vol:
  psql_socket_vol:
  gvmd_socket_vol:
  ospd_openvas_socket_vol:
  redis_socket_vol:
```

```
openvas_data_vol:
openvas_log_data_vol:
```

4.6 Základná konfigurácia

Aktualizácia databáz CVE, pridanie hostov.



5 OPENCTI

Typ nástroja: Threat Intelligence Platform (TIP)

Názov nástroja: OpenCTI

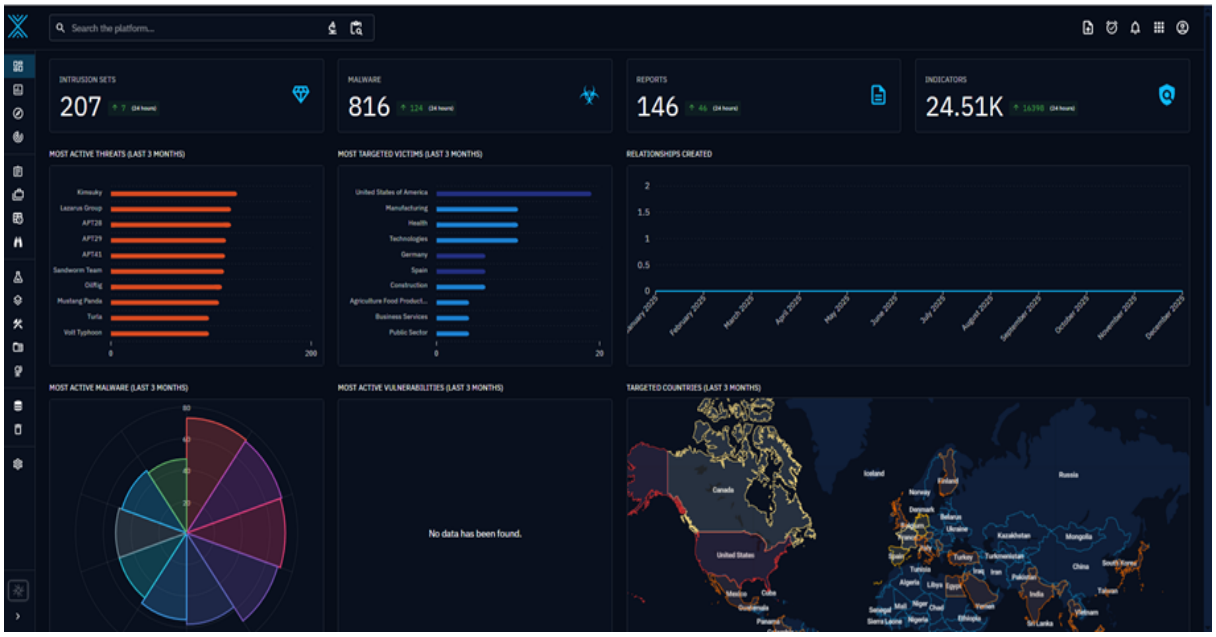
Verzia: Version 6.9.0 – Open-source verzia (kontajnerizované nasadenie)

5.1 Opis systému

OpenCTI je open-source platforma určená na centralizovanú správu Cyber Threat Intelligence. Slúži ako centrálne miesto, kde organizácie zhromažďujú informácie o kybernetických útokoch, útočníkoch, malvéri, zraniteľnostiach a ďalších technických indikátoroch, ktoré môžu súvisieť s bezpečnostnými incidentmi. Hlavnou myšlienkou OpenCTI je nielen zbierať dáta, ale najmä ich prepájať do zmysluplných vzťahov, aby bezpečnostní analytici videli širší kontext hrozieb a dokázal lepšie pochopiť, kto útočí, aké nástroje používa a aké ciele sleduje.

Platforma podporuje štandardy STIX/TAXII a integráciu s množstvom externých databáz a bezpečnostných nástrojov. To znamená, že informácie o hrozbách sú štruktúrované jednotným a zdieľateľným spôsobom. OpenCTI tieto údaje zobrazuje vo forme grafu, kde sú jasne viditeľné vzťahy medzi útočníckymi skupinami, malvérom, IP adresami, doménami či zraniteľnosťami typu CVE.

Z technického pohľadu je OpenCTI postavený najmä na Pythone a využíva ElasticSearch na ukladanie a vyhľadávanie dát. Keďže ide o open-source riešenie, je dostupný zadarmo a organizácie si ho môžu prispôsobiť vlastným potrebám, či už ide o rozšírenia, konektory alebo spôsob nasadenia.



5.2 Dokumentácia systému

Oficiálna dokumentácia platformy OpenCTI je verejne dostupná a priebežne aktualizovaná vývojárskym tímom a komunitou projektu. Dokumentácia poskytuje podrobný prehľad architektúry systému, objektového a dátového modelu založeného na štandarde STIX, možností integrácie prostredníctvom konektorov, ako aj odporúčané postupy pre nasadenie a prevádzku v SOC prostredí.

Súčasťou dokumentácie sú detailné návody na kontajnerizované nasadenie pomocou Dockeru, konfiguráciu jednotlivých komponentov, správu používateľov a oprávnení, ako aj opis interných mechanizmov spracovania threat intelligence dát. Dokumentácia zároveň zahŕňa referenčné príklady integrácií s externými zdrojmi informácií o hrozbách a bezpečnostnými nástrojmi.

Odkazy na oficiálne zdroje:

- Oficiálna dokumentácia OpenCTI: https://docs.opencti.io/latest/
- GitHub repozitár projektu: https://github.com/OpenCTI-Platform/opencti
- Oficiálne Docker image (OpenCTI a konektory): https://hub.docker.com/u/opencti

5.3 Požiadavky na zdroje

Hardvérové požiadavky platformy OpenCTI závisia od rozsahu spracovávanej threat intelligence, počtu aktívnych konektorov a intenzity práce s grafovým dátovým modelom. Keďže OpenCTI pracuje so štruktúrovanými dátami a vzťahmi medzi objektmi, nároky na výpočtové zdroje sú vyššie ako pri jednoduchých databázových aplikáciách, no stále primerané pre nasadenie v SOC prostredí.

Pre testovacie alebo PoC nasadenie postačuje:

- CPU: 2–4 jadrá
- RAM: 8 GB
- Diskový priestor: 50–100 GB (SSD odporúčané)

Pre produkčné alebo rozsiahlejšie SOC prostredie sa odporúča:

- CPU: 4–8 jadier
- RAM: 16–32 GB
- Disk: SSD úložisko (najmä pre databázovú a indexačnú vrstvu)

Vyššie nároky sú dané najmä počtom ingestovaných TI feedov, veľkosťou dátového modelu (STIX objekty a vzťahy), frekvenciou aktualizácií a počtom paralelne bežiacich konektorov. V praxi je OpenCTI často nasadzovaný kontajnerizovane, čo umožňuje flexibilné škálovanie jednotlivých komponentov podľa aktuálnej záťaže.

5.4 Architektúra nástroja

Architektúra platformy OpenCTI je navrhnutá ako modulárny a distribuovaný systém, ktorý umožňuje efektívne spracovanie, obohacovanie a koreláciu threat intelligence dát z rôznych zdrojov. Základ architektúry tvorí webová aplikačná vrstva, aplikačné API a množina samostatných konektorov, ktoré zabezpečujú import, export a spracovanie dát.

Centrálnym komponentom je OpenCTI platforma (backend + frontend), ktorá poskytuje webové používateľské rozhranie a REST/GraphQL API. Prostredníctvom tohto rozhrania môžu analytici prehliadať, analyzovať a spravovať threat intelligence objekty, ako sú aktéri hrozieb, kampane, malvér, techniky útokov a indikátory kompromitácie. Backend zároveň zabezpečuje aplikačnú logiku, kontrolu prístupu a správu vzťahov medzi jednotlivými objektmi podľa STIX dátového modelu.

Perzistentná dátová vrstva je realizovaná pomocou databázových a indexačných mechanizmov, ktoré umožňujú ukladanie veľkého množstva štruktúrovaných threat intelligence dát a ich efektívne vyhľadávanie. Architektúra je optimalizovaná na prácu s grafovými vzťahmi medzi objektmi, čo je kľúčové pre pochopenie kontextu hrozieb.

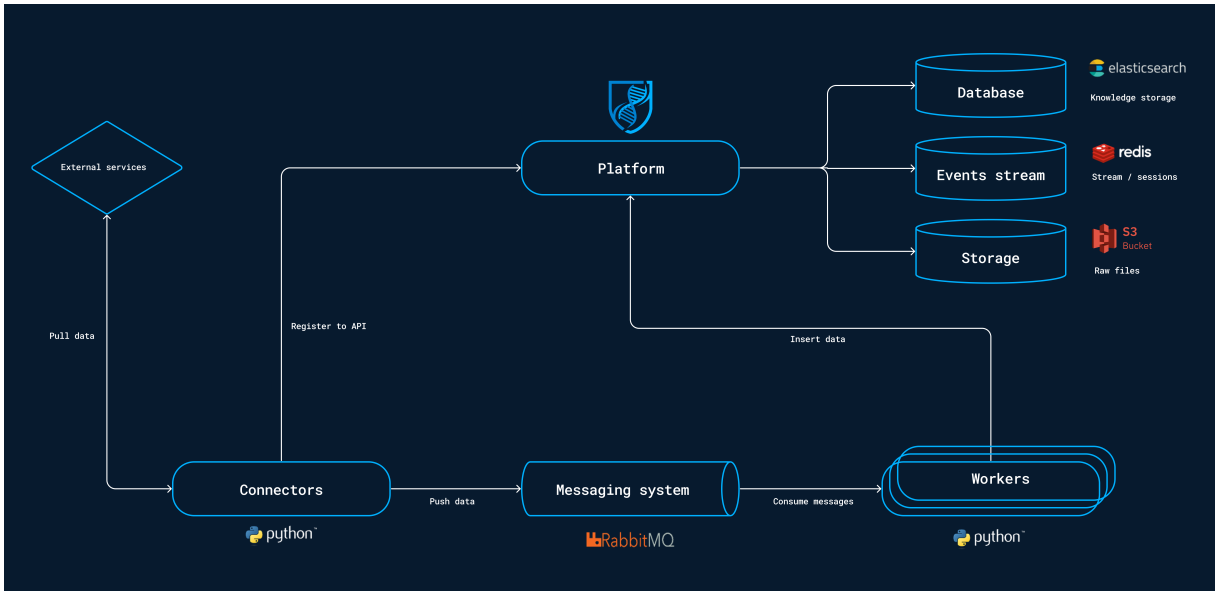
Dôležitou súčasťou OpenCTI sú konektory (connectors), ktoré fungujú ako samostatné procesy alebo kontajnery. Konektory zabezpečujú:

- import dát z externých zdrojov (OSINT, komerčné TI feedy),
- export dát do externých systémov (SIEM, SOAR),
- obohacovanie existujúcich objektov o ďalšie informácie.

Tento prístup umožňuje rozširovať funkcionalitu OpenCTI bez zásahu do jadra aplikácie a podporuje bezpečné oddelenie integračnej logiky od hlavnej platformy.

Z pohľadu SOC architektúry je OpenCTI umiestnené ako inteligenčná vrstva medzi detekciou a reakciou. Poskytuje kontext pre detekčné nástroje (napr. SIEM) a vstupy pre automatizované reakčné mechanizmy (SOAR). Samotné OpenCTI nevykonáva detekciu ani reakciu, ale zvyšuje kvalitu rozhodovania tým, že prepája technické indikátory s taktickým a strategickým pohľadom na hrozby.

Schéma architektúry OpenCTI (použitá v dokumente) vychádza z oficiálnej dokumentácie projektu a ilustruje tok threat intelligence dát medzi platformou, konektormi a externými systémami v rámci SOC infraštruktúry.



5.5 Inštalácia – variant Docker

Inštalácia bola vykonaná podľa oficiálnej OpenCTI dokumentácie s využitím kontajnerizácie, ktorá bola nasadená na zariadenie so systémom Linux Ubuntu 22.04.

Kroky inštalácie:

1. Inštalácia docker-conpose na zariadení

```
sudo apt install docker-compose
```

2. Vytvorenie nového priečinka pre inštalačný balíček a presun do tohto priečinka

```
mkdir -p /path/to/your/app && cd /path/to/your/ap
```

3. Stiahnutie inštalačného balíčka z platformy GitHub

```
git clone https://github.com/OpenCTI-Platform/docker.git
```

4. Úprava vzorového súboru .env sample, z ktorého docker-compose používa premenné pri inštalácií, podľa požadovaných preferencií používateľa

root@opencti-s: /opt/docker

```
 GNU nano 6.2                                    .env.sample
############################
# DEPENDENCIES             #
############################

MINIO_ROOT_USER=opencti
MINIO_ROOT_PASSWORD=changeme
RABBITMQ_DEFAULT_USER=opencti
RABBITMQ_DEFAULT_PASS=changeme
SMTP_HOSTNAME=localhost
OPENSEARCH_ADMIN_PASSWORD=changeme
ELASTIC_MEMORY_SIZE=4G

############################
# COMMON                   #
############################

XTM_COMPOSER_ID=8215614c-7139-422e-b825-b20fd2a13a23
COMPOSE_PROJECT_NAME=xtm

############################
# OPENCTI                  #
############################

OPENCTI_HOST=localhost
OPENCTI_PORT=8080
OPENCTI_EXTERNAL_SCHEME=http
OPENCTI_ADMIN_EMAIL=admin@opencti.io
OPENCTI_ADMIN_PASSWORD=changeme
OPENCTI_ADMIN_TOKEN=ChangeMe_UUIDv4
OPENCTI_HEALTHCHECK_ACCESS_KEY=changeme

############################
# OPENCTI CONNECTORS       #
############################

CONNECTOR_EXPORT_FILE_STIX_ID=dd817c8b-abae-460a-9ebc-97b1551e70e6
CONNECTOR_EXPORT_FILE_CSV_ID=7ba187fb-fde8-4063-92b5-c3da34060dd7
CONNECTOR_EXPORT_FILE_TXT_ID=ca715d9c-bd64-4351-91db-33a8d728a58b
CONNECTOR_IMPORT_FILE_STIX_ID=72327164-0b35-482b-b5d6-a5a3f76b845f
CONNECTOR_IMPORT_DOCUMENT_ID=c3970f8a-ce4b-4497-a381-20b7256f56f0
CONNECTOR_IMPORT_FILE_YARA_ID=7eb45b60-069b-4f7f-83a2-df4d6891d5ec
```

5. Po úprave .env súboru je spustená služba docker a sú vytvorené jednotlivé kontajnery so softvérom OpenCTI a jeho súčasťami

```
sudo systemctl start docker.service
docker-compose up -d
```

6. Otvorenie aplikácie a prihlásenie do účtu

```
http://ip_adresa:8080
```

5.6 Inštalácia – variant Openstack

Pri manuálnej inštalácii OpenCti na platforme OpenStack sa rovnako postupovalo podľa oficiálnej webovej dokumentácie. V OpenStacku bol vytvorený virtuálny stroj (Virtual Machine VM) so systémom Linux Ubuntu 22.04, ako tomu bolo aj pri inštalácii pomocou kontajnerizácie.

Kroky inštalácie

Prerekvizity:

1. **Inštalácia závislostí**
   Pred samotnou inštaláciou softvérových balíčkov OpenCTI je metódou “dobrej praxe” doinštalovať potrebné závislosti pre beh hlavnej aplikácie a jednotlivých súvislostí.

```
sudo apt-get update
sudo apt-get install build-essential nodejs npm python3 python3-pip python3-dev
```

2. **Stiahnutie aplikačných súborov**
   Najskôr je potrebné stiahnuť poslednú vydanú verziu aplikácie (resp. aplikačného balíčka). Pre systémy Linux (Ubuntu, Debian,...) je potrebné nainštalovať aktuálnu verziu aplikácie s použitím príkazu:

```
opencti-release_{RELEASE_VERSION}.tar.gz
```

Inštalácia hlavnej platformy:

3. **Konfigurácia aplikácie**
   Hlavná aplikácia obsahuje jeden konfiguračný JSON súbor, ktorý je potrebné nastaviť podľa preferencií používateľa, (resp. Je potrebné súbor upraviť tak, aby odpovedal konfigurácií ElasticSearch) a niekoľko Python modulov na inštaláciu.

```
cd opencti
cp config/default.json config/production.json
```

4. **Inštalácia Python modulov**

```
cd src/python
pip3 install -r requirements.txt
cd ../..
```

5. **Spustenie aplikácie**
   Aplikácia je iba NodeJS proces, vytvorenie schémy databázy a migrácia sa vykoná pri spustení.

```
npm install -g corepack
```

Po skontrolovaní naištalovaných verzií NodeJS je možné aplikáciu spustiť vo vnútri **opencti** priečinka.

```
yarn install
yarn build
yarn serv
```

Predvolené používateľské meno a heslo pre aplikáciu sú používateľom definované v

```
config/production.json
```

Inštalácia workerov

6. Konfigurácia workera

```
cd worker
pip3 install -r requirements.txt
cp config.yml.sample config.yml
```

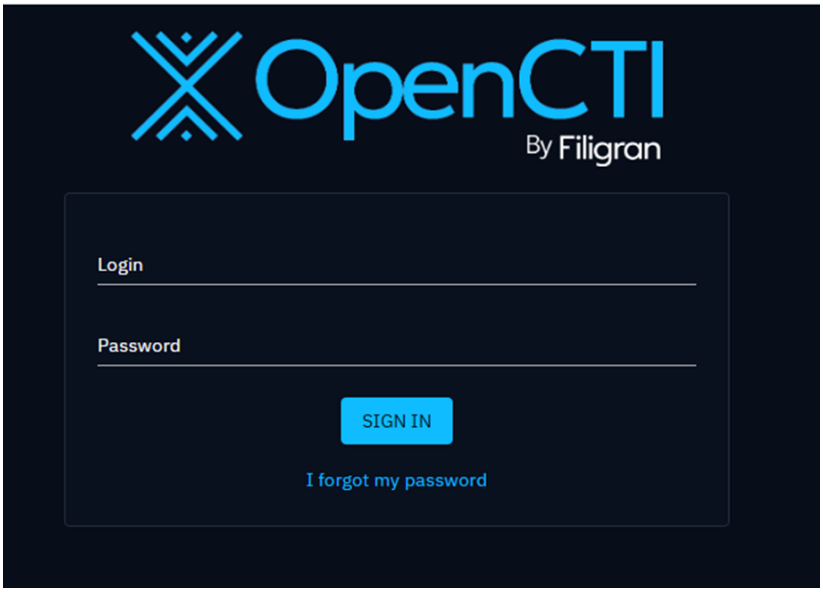
7. Spustenie workera

V tomto kroku je možnéspustiť toľko workerov, koľko je potrebné

```
python3 worker.py &
python3 worker.py &
```

8. Otvorenie aplikácie a prihlásenie do účtu

```
http://ip_adresa:8080
```



5.7 Základná konfigurácia

Základná konfigurácia OpenCTI sa najčastejšie realizuje pomocou Docker Compose a pozostáva z niekoľkých kľúčových komponentov, ktoré spolu tvoria funkčný ekosystém pre

správu kybernetických hrozieb. Jadrom je samotná aplikačná časť OpenCTI, ktorá poskytuje webové rozhranie a API. Táto aplikácia komunikuje s databázovou vrstvou, ktorú tvorí ElasticSearch určený na ukladanie a rýchle vyhľadávanie STIX objektov, a s Redisom, ktorý slúži na správu front úloh a asynchrónne spracovanie dát. Súčasťou základnej konfigurácie je aj MinIO, objektové úložisko používané na ukladanie súborových príloh, reportov a binárnych dát.

Po úvodnej inštalácii je potrebné nastaviť základné prevádzkové parametre, ako je administrátorský účet, organizácia, používateľské roly a úrovne prístupu. Konfigurujú sa tiež retenčné politiky dát, výkonové limity a spôsob autentifikácie, ktorý môže byť lokálny alebo napojený na externý identitný systém.

Dôležitou súčasťou konfigurácie je aj nastavenie konektorov, ktoré zabezpečujú prísun dát do OpenCTI alebo naopak export informácií do iných bezpečnostných nástrojov. V aktuálne nasadenej verzii aplikácie sú v OpenCTI aktívne 3 konektory na databázy MITTRE ATT&CK, Urlhuas(abuse.ch) a Ransomwarelive.

Všetky nové konektory sme pridávali do docker-compose.yml pričom sme pridali nasledovnú konfiguráciu pre jednotlivé konektory:

5.7.1 Konfigurácia MITTRE ATTACK

```
connector-mitre:
  image: opencti/connector-mitre:6.9.4
  environment:
    - OPENCTI_URL=http://opencti:8080
    - OPENCTI_TOKEN=${OPENCTI_ADMIN_TOKEN}
    - CONNECTOR_ID=${CONNECTOR_MITRE_ID}
    - CONNECTOR_AUTO_CREATE_SERVICE_ACCOUNT=true
    - CONNECTOR_AUTO_CREATE_SERVICE_ACCOUNT_CONFIDENCE_LEVEL=75
  restart: always
  depends_on:
    opencti:
      condition: service_healthy
```

5.7.2 Konfigurácia Urlhaus

```
connector-urlhaus:
  image: opencti/connector-urlhaus:latest
  environment:
    - OPENCTI_URL=http://opencti:8080
    - OPENCTI_TOKEN=${OPENCTI_ADMIN_TOKEN}
    - CONNECTOR_ID=df05d65f-b4d9-4438-9988-231e7c99f994
    - CONNECTOR_NAME=Abuse.ch URLhaus
    - CONNECTOR_SCOPE=urlhaus
    - CONNECTOR_LOG_LEVEL=error
    - URLHAUS_INTERVAL=3
    - URLHAUS_CSV_URL=https://urlhaus.abuse.ch/downloads/csv_recent/
    - URLHAUS_DEFAULT_X_OPENCTI_SCORE=80
    - URLHAUS_IMPORT_OFFLINE=true
    - URLHAUS_THREATS_FROM_LABELS=true
  restart: always
  depends_on:
    opencti:
      condition: service_healthy
```

5.7.3 Konfigurácia Ransomwarelive

```
connector-ransomware:
  image: opencti/connector-ransomwarelive:latest
  container_name: ransomware-connector
  environment:
    - OPENCTI_URL=http://opencti:8080
    - OPENCTI_TOKEN=${OPENCTI_ADMIN_TOKEN}
  restart: always
  depends_on:
    opencti:
      condition: service_healthy
```

Všetkým konektorom bolo potrebné priradiť administrátorský token platformy OpenCTI. V prípade konektorov MITRE ATT&CK a Urlhaus bolo zároveň nevyhnutné vygenerovať jedinečný identifikátor typu UUID4, ktorý bol použitý ako hodnota parametra *connector_id*. Na záver bola do konfigurácie doplnená podmienka zabezpečujúca, že jednotlivé konektory sa budú inicializovať až po úspešnom spustení služby OpenCTI.

Po nakonfigurovaní konektorov bol Docker kontajnerový stack zastavený pomocou príkazu *docker-compose down* a následne opätovne spustený príkazom *docker-compose up -d*, aby sa nové nastavenia korektne aplikovali. Je potrebné zdôrazniť, že príkaz *docker-compose down -v* by nemal byť použitý, nakoľko by došlo k odstráneniu perzistentných úložísk a tým aj k strate už stiahnutých CTI dát.

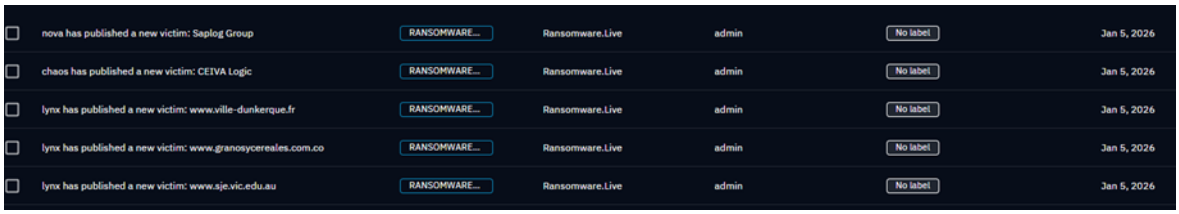
Po úspešnom spustení konektorov sa databáza platformy OpenCTI začala postupne plniť dátami z oblasti kybernetických hrozieb.

5.7.4 Vstupy a výstupy

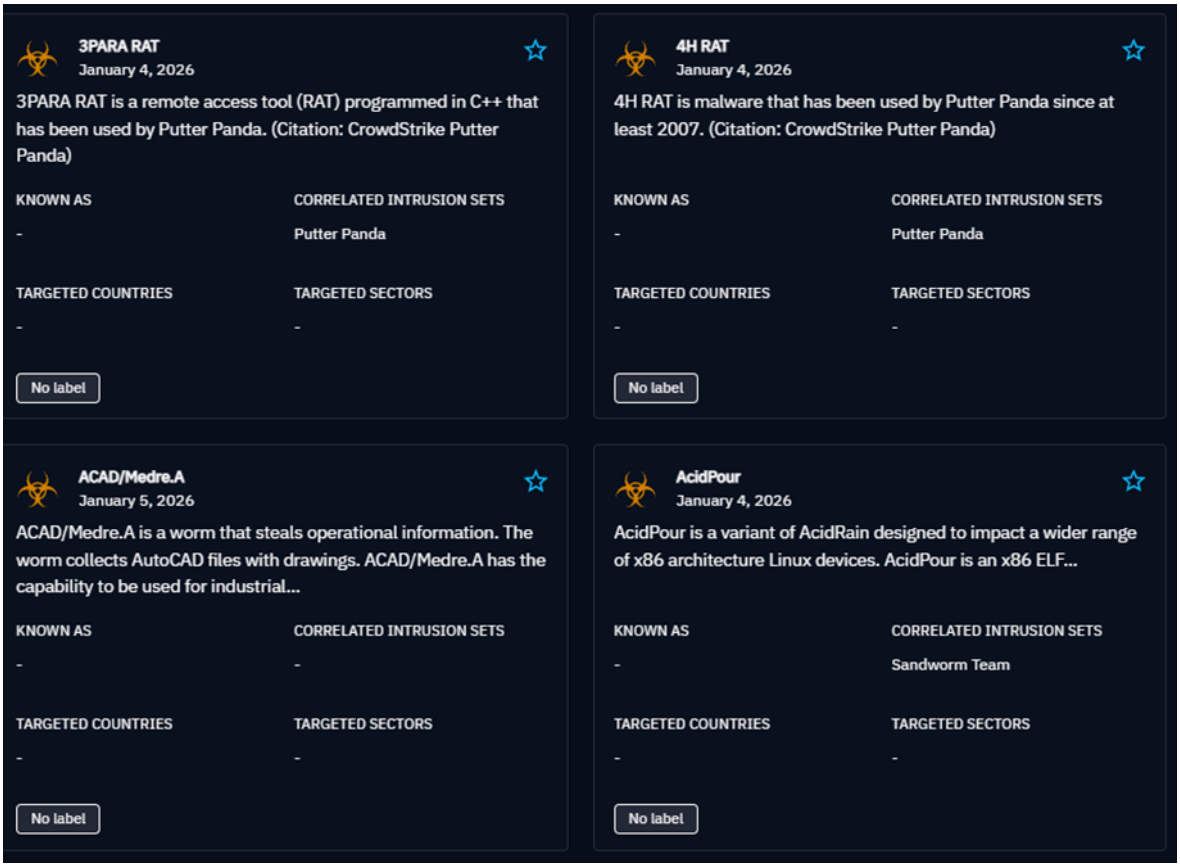
Vstupom do OpenCTI sú informácie o kybernetických hrozbách, ktoré pochádzajú z externých zdrojov prostredníctvom konektorov alebo manuálneho zadania. Ide najmä o údaje o malvéri, útočníckych skupinách, zraniteľnostiach, IP adresách, doménach a ďalších indikátoroch kompromitácie, napríklad zo systémov ako MISP alebo VirusTotal.

Výstupom z OpenCTI sú spracované a kontextualizované threat intelligence dáta, ktoré sú dostupné vo webovom rozhraní, cez API alebo prostredníctvom exportných konektorov. Tieto výstupy sa využívajú v ďalších bezpečnostných nástrojoch, ako sú SIEM, SOAR, EDR alebo firewally, a slúžia na detekciu, prevenciu a reakciu na kybernetické hrozby.

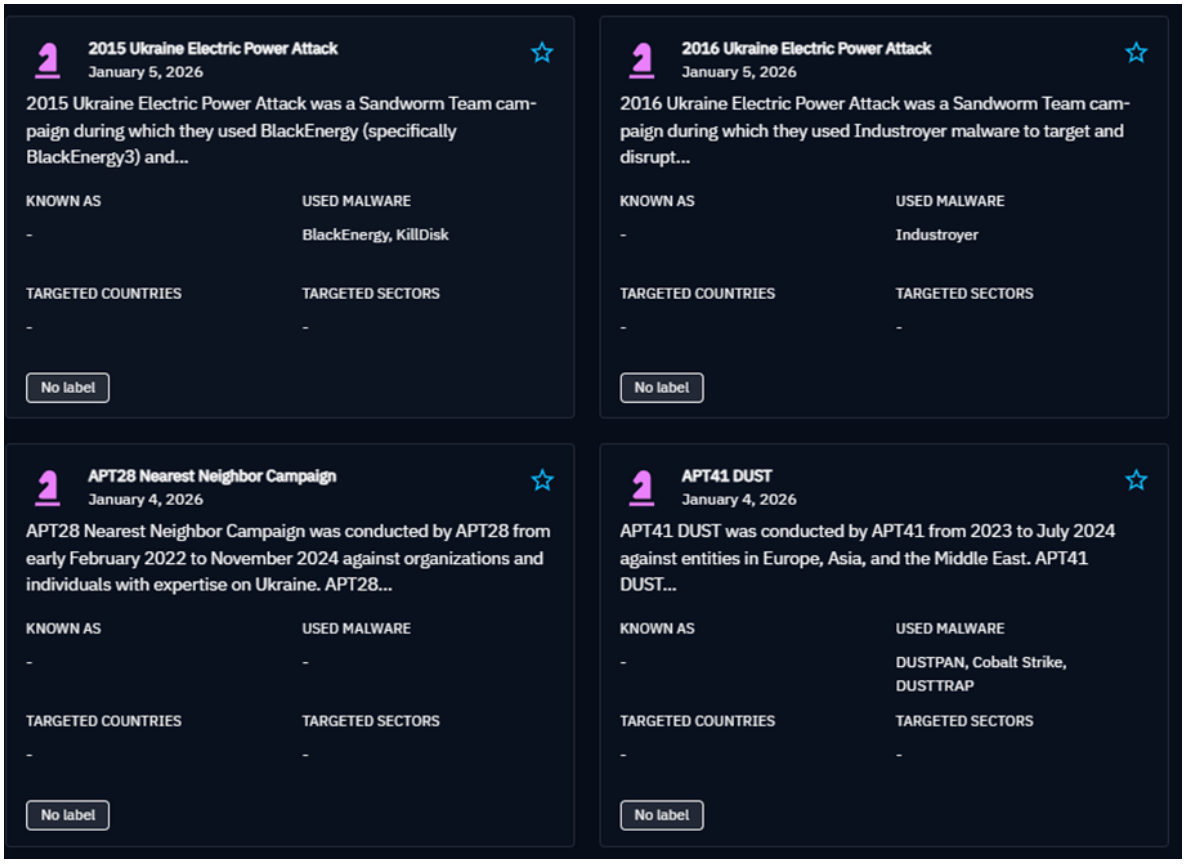
Informácie o nových obetiach v OpenCTI



Informácie o známych málvéroch v OpenCTI



Informácie o prebiehajúcich kampaniach



5.8 Integrácie

Platforma je navrhnutá tak, aby spolupracovala so SIEM a SOAR nástrojmi, kam môže poskytovať overené indikátory alebo kontextové informácie o hrozbách. V opačnom smere vie prijímať dáta z incidentov a obohacovať ich o threat intelligence kontext. Integrácie sú dostupné aj pre systémy typu EDR alebo firewall, kde OpenCTI slúži ako centrálna databáza hrozieb, z ktorej sa následne generujú blokovacie alebo detekčné pravidlá.

Vďaka otvorenému API a podpore štandardu STIX/TAXII je OpenCTI veľmi flexibilný a umožňuje vytvárať vlastné integrácie podľa potrieb organizácie. To znamená, že okrem hotových konektorov je možné jednoducho doprogramovať vlastný zdroj dát alebo exportný mechanizmus, čo robí z OpenCTI silný centrálny nástroj pre riadenie kybernetických hrozieb v modernom bezpečnostnom prostredí.

5.8.1 Integrácia OpenCTI – pozícia CTI v SOC a integračný model

5.8.1.1 Pozícia OpenCTI v SOC

OpenCTI plní v architektúre SOC úlohu centrálnej platformy pre threat intelligence (CTI), ktorá prepája strategické, takticko-operatívne a technické informácie o hrozbách. Na

rozdiel od SIEM alebo SOAR nástrojov OpenCTI nespracúva logy v reálnom čase a nevykonáva reakčné akcie, ale poskytuje kontext, ktorý zvyšuje kvalitu detekcie a reakcie.

Z pohľadu SOC procesov je OpenCTI zaradené:

- pred detekciou – ako zdroj indikátorov a poznatkov o hrozbách,
- počas analýzy – ako kontextualizačný nástroj pri vyšetrovaní incidentov,
- pred reakciou – ako vstup pre rozhodovanie o primeraných opatreniach.

OpenCTI umožňuje prepojiť jednotlivé technické indikátory (IP adresy, domény, hashe) s vyšším kontextom, ako sú kampane, aktéri hrozieb alebo techniky MITRE ATT&CK, čím podporuje analytické rozhodovanie v SOC.

5.8.1.2 Integrácia OpenCTI s ostatnými SOC komponentmi

OpenCTI je navrhnuté ako integračná platforma, pričom základným mechanizmom integrácie sú samostatné konektory a API rozhrania. Integrácia prebieha obojsmerne – OpenCTI môže údaje prijímať aj poskytovať.

Integrácia so SIEM (ELK Stack):

- OpenCTI poskytuje indikátory kompromitácie a kontext (napr. reputácia IP, väzba na kampaň),
- SIEM môže využívať tieto informácie pri korelácii udalostí a znižovaní falošných poplachov,
- analytik počas vyšetrovania incidentu využíva OpenCTI na lepšie pochopenie pôvodu a významu detekcie.

Integrácia so SOAR (Shuffle):

- OpenCTI slúži ako zdroj threat intelligence pre automatizované playbooky,
- Shuffle môže v rámci enrichment fázy dotazovať OpenCTI na kontext indikátora,
- výsledky dotazu ovplyvňujú rozhodovaciu logiku playbooku (napr. eskalácia, blokovanie).

Integrácia s externými zdrojmi TI:

- OpenCTI ingestuje dáta z OSINT alebo komerčných TI feedov prostredníctvom konektorov,
- tieto dáta sú normalizované do jednotného STIX modelu,
- zabezpečuje sa jednotný a auditovateľný pohľad na informácie o hrozbách.

5.8.1.3 Zhrnutie integračnej logiky

V navrhovanej SOC architektúre platí nasledovná deľba úloh:

- ELK Stack – detekcia a analýza udalostí,
- NetBox – kontext aktív a infraštruktúry,
- OpenCTI – kontext hrozieb a znalostná vrstva,
- Shuffle – automatizovaná reakcia a orchestrácia.

OpenCTI tak vystupuje ako znalostný uzol SOC, ktorý prepája technické indikátory s vyšším kontextom a zvyšuje presnosť aj efektivitu detekčných a reakčných procesov bez toho, aby priamo zasahoval do infraštruktúry.

6 MISP

Typ nástroja: Threat Intelligence Platform (TIP)/Information Sharing

Názov nástroja: MISP (Malware Information Sharing Platform)

Verzia: Open-source verzia (kontajnerizované nasadenie) 2.5.28

6.1 Opis systému

MISP (Malware Information Sharing Platform) je open-source platforma určená na zber, správu, koreláciu a zdieľanie hrozbových informácií medzi bezpečnostnými tímami a organizáciami. Primárnym cieľom MISP je umožniť efektívnu výmenu indikátorov kompromitácie (IoC), informácií o incidentoch a poznatkov o hrozbách v štandardizovanom a štruktúrovanom formáte.

MISP je navrhnutý ako operatívna CTI platforma, ktorá sa zameriava najmä na technickú úroveň threat intelligence. Pracuje s objektmi ako IP adresy, domény, URL, hashe, e-mailové artefakty a ďalšie indikátory, pričom umožňuje ich koreláciu naprieč udalosťami a zdrojmi. V porovnaní s OpenCTI, ktoré kladie dôraz na modelovanie hrozieb a ich vzťahov, MISP poskytuje rýchly a praktický mechanizmus výmeny IoC medzi systémami a organizáciami.

V architektúre SOC vystupuje MISP ako zdroj a distribučný uzol hrozbových indikátorov, ktoré sú využívané pri detekcii a automatizovaných reakciách. MISP samotný nevykonáva detekciu ani reakciu, ale poskytuje aktuálne a overené vstupy pre SIEM a SOAR nástroje, čím zvyšuje presnosť detekčných pravidiel a podporuje rýchlu reakciu na známe hrozby.

V navrhovanom riešení je MISP využívaný ako praktický TI feed backend, ktorý dopĺňa interné a externé zdroje hrozbových informácií a umožňuje ich integráciu do SOC workflow-ov.

6.2 Dokumentácia systému

Oficiálna dokumentácia platformy MISP je verejne dostupná a poskytuje detailný popis architektúry, dátového modelu, API rozhraní a možností integrácie. Dokumentácia sa zameriava najmä na praktické aspekty používania MISP v bezpečnostných tímoch, vrátane správy udalostí, atribútov, korelácií a zdieľania dát medzi inštanciami.

Odkazy na oficiálne zdroje:

- Oficiálna dokumentácia MISP: https://www.misp-project.org/documentation/
- GitHub repozitár projektu: https://github.com/MISP/MISP
- Oficiálne Docker image: https://hub.docker.com/u/misp

6.3 Požiadavky na zdroje

Hardvérové požiadavky MISP sú relatívne mierne, keďže platforma je optimalizovaná na prácu so štruktúrovanými TI dátami a koreláciami.

Pre testovacie alebo PoC nasadenie postačuje:

- CPU: 2 jadrá
- RAM: 4–8 GB
- Diskový priestor: 50 GB

Pre produkčné alebo intenzívnejšie používanie sa odporúča:

- CPU: 4–8 jadier
- RAM: 8–16 GB
- Disk: SSD úložisko

Vyššie nároky môžu vzniknúť pri veľkom počte korelácií, dlhšej retencii dát a pri synchronizácii s viacerými externými MISP inštanciami.

6.4 Architektúra nástroja

Architektúra platformy MISP je navrhnutá ako centralizovaný webový systém určený na správu, koreláciu a zdieľanie informácií o hrozbách. Základ riešenia tvorí webová aplikačná vrstva, ktorá poskytuje používateľské rozhranie pre správu udalostí, atribútov a zdieľacích politík, a backendová aplikačná logika, ktorá zabezpečuje spracovanie dát a korelačné mechanizmy.

Perzistentná dátová vrstva je realizovaná prostredníctvom relačnej databázy, v ktorej sú ukladané udalosti, indikátory kompromitácie (atribúty), ich metaúdaje a vzájomné korelačné väzby. MISP automaticky vykonáva koreláciu nových indikátorov s existujúcimi dátami, čím umožňuje rýchlu identifikáciu opakujúcich sa hrozieb alebo známych vzorcov útokov naprieč viacerými incidentmi.

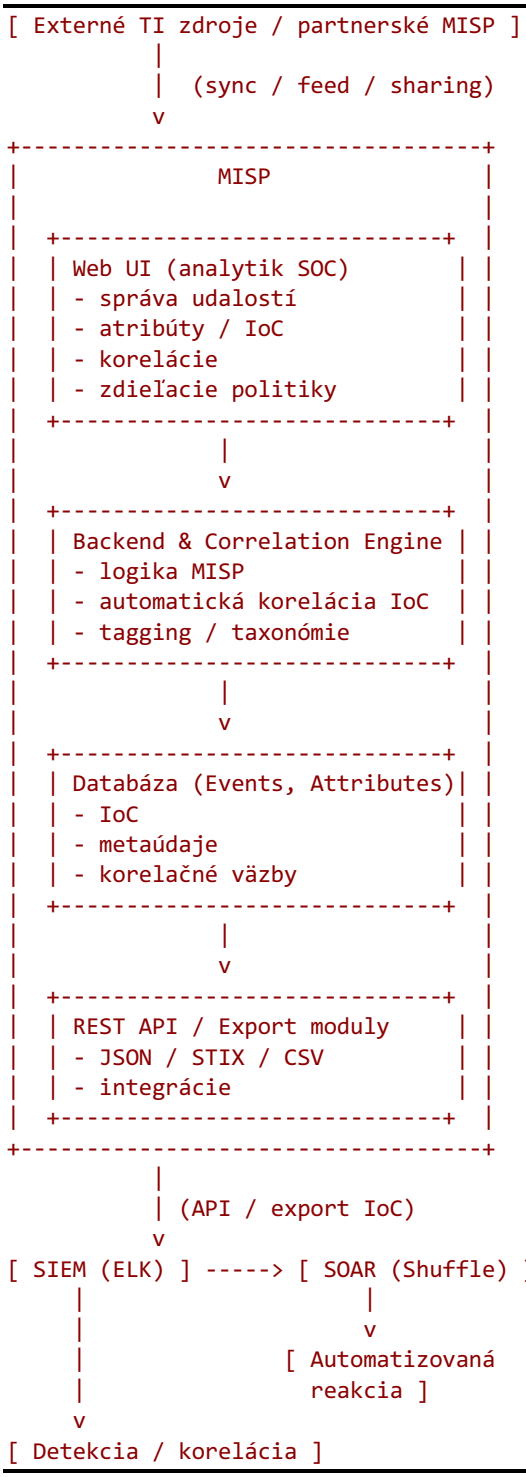
Dôležitým prvkom architektúry je synchronizačný mechanizmus medzi MISP inštanciami, ktorý umožňuje bezpečné zdieľanie udalostí a indikátorov medzi organizáciami. Synchronizácia prebieha na základe definovaných pravidiel dôvery, filtrov a zdieľacích politík, čo umožňuje selektívne poskytovanie dát bez straty kontroly nad citlivými informáciami.

MISP poskytuje REST API, ktoré je kľúčovým integračným rozhraním pre externé systémy. Prostredníctvom API je možné:

- importovať nové indikátory a udalosti,
- exportovať IoC v rôznych formátoch,
- synchronizovať dáta s inými platformami (SIEM, SOAR, TIP).

Architektúra MISP podporuje aj rozšírenia vo forme modulov, ktoré umožňujú automatizované obohacovanie dát (enrichment) alebo ich export do externých nástrojov. Tieto moduly sú oddelené od jadra systému, čo zvyšuje flexibilitu riešenia a umožňuje jeho prispôsobenie konkrétnym potrebám SOC.

Z pohľadu SOC architektúry je MISP umiestnený ako operatívny CTI uzol, ktorý stojí medzi externými zdrojmi hrozbových informácií a detekčnými nástrojmi. Zabezpečuje normalizáciu, koreláciu a distribúciu indikátorov kompromitácie, čím vytvára spoľahlivý a auditovateľný zdroj dát pre detekčné a reakčné procesy.


[Externé TI zdroje / partnerské MISP]
(sync / feed / sharing)
MISP
Web UI (analytik SOC)
- správa udalostí
- atribúty / IoC
- korelácie
- zdieľacie politiky
Backend & Correlation Engine
- logika MISP
- automatická korelácia IoC
- tagging / taxonómie
Databáza (Events, Attributes)
- IoC
- metaúdaje
- korelačné väzby
REST API / Export moduly
- JSON / STIX / CSV
- integrácie
(API / export IoC)
[SIEM (ELK)]
[SOAR (Shuffle)]
[Automatizovaná reakcia]
[Detekcia / korelácia]

6.5 Integrácie

6.5.1 Pozícia MISP v SOC

MISP plní v architektúre SOC úlohu operatívnej CTI platformy zameranej na praktickú prácu s indikátormi hrozieb (IoC) a ich rýchlu distribúciu. Na rozdiel od SIEM alebo SOAR nástrojov MISP nespracúva logy v reálnom čase a nevykonáva reakčné akcie, ale poskytuje overené indikátory a metaúdaje, ktoré zvyšujú presnosť detekcie a podporujú rýchle rozhodovanie pri incidente.

Z pohľadu SOC procesov je MISP zaradený:

- pred detekciou – ako zdroj IoC a feedov pre detekčné pravidlá (korelácia),
- počas analýzy – ako referenčný register známych indikátorov (tagy, scoring, väzby na udalosti),
- pred reakciou – ako vstup pre automatizované rozhodovanie (napr. eskalácia, blokovanie, notifikácia).

MISP je preto vhodný najmä tam, kde SOC potrebuje rýchlo pracovať s IoC (IP/domény/URL/hashe) a mať kontrolu nad tým, čo sa zdieľa, komu a v akej kvalite.

6.5.2 Integrácia MISP s ostatnými SOC komponentmi

MISP je navrhnutý ako integračný uzol pre threat intelligence, pričom základným mechanizmom integrácie je REST API, exportné formáty a synchronizačné mechanizmy medzi MISP inštanciami. Integrácia prebieha obojsmerne – MISP môže dáta poskytovať aj prijímať (napr. IoC z incidentov, externé feedy).

Integrácia so SIEM (ELK Stack):

- MISP poskytuje IoC a ich metaúdaje (tagy, typ, závažnosť, časová platnosť),
- SIEM využíva tieto údaje pri korelácii udalostí a enrichment alertov (napr. „indikátor sa zhoduje s TI“),
- výstupy vyšetrovania (nové IoC z incidentu) môžu byť spätne vložené do MISP ako interná TI.

Integrácia so SOAR (Shuffle):

- MISP slúži ako TI backend pre automatizované playbooky (lookup / enrichment),
- Shuffle dotazuje MISP na prítomnosť indikátora, jeho tagy alebo scoring,
- výsledok ovplyvňuje rozhodovaciu logiku (eskalácia, ticket, blokovanie, notifikácia).

Integrácia s bezpečnostnými systémami (EDR / IDS / firewall):

- MISP môže slúžiť ako zdroj indikátorov pre generovanie detekčných alebo blokovacích pravidiel,
- typicky ide o export IoC do formátov vhodných pre konkrétne nástroje (napr. blacklisty, signatúry),
- cieľom je TI-driven detekcia/reakcia pri zachovaní auditovateľnosti a kontroly nad distribúciou.

Integrácia s externými zdrojmi TI a komunitou:

- MISP podporuje ingest indikátorov z externých feedov a zdieľanie medzi organizáciami,
- synchronizácia medzi MISP inštanciami umožňuje selektívne zdieľanie podľa dôvery a filtrov,
- takto sa zabezpečuje aktuálnosť a operatívna výmena IoC v SOC ekosystéme.

6.5.3 OpenCTI versus MISP

MISP a OpenCTI predstavujú dve rozdielne, no do značnej miery prekrývajúce sa platformy pre threat intelligence (CTI). Obe riešenia sú schopné plniť úlohu centrálneho nástroja pre prácu s informáciami o hrozbách v prostredí SOC a v určitých scenároch môžu byť používané ako vzájomné alternatívy.

Z funkčného hľadiska je možné MISP považovať za protiľahlý (alternatívny) prístup k OpenCTI, keďže:

- obe platformy umožňujú evidenciu a správu indikátorov kompromitácie (IoC),
- podporujú integráciu so SIEM a SOAR nástrojmi,
- poskytujú API rozhrania na obojsmernú výmenu dát,
- slúžia ako zdroj kontextu pre detekčné a reakčné procesy v SOC.

V rámci projektu boli testované obe platformy – MISP aj OpenCTI – s cieľom overiť ich vhodnosť pre úlohu CTI komponentu v navrhovanej SOC architektúre. Testovanie potvrdilo, že obe riešenia sú schopné plniť základné CTI funkcie samostatne, najmä v oblasti poskytovania kontextu k detekciám a podpory analytického rozhodovania.

Zároveň sa však ukázalo, že MISP a OpenCTI sa prirodzene dopĺňajú, pričom každé z riešení je silnejšie v inej oblasti:

- MISP je vhodnejší pre ***operatívnu threat intelligence***, rýchlu prácu s IoC, ich zdieľanie, synchronizáciu medzi organizáciami a priame napojenie na detekčné a blokovacie mechanizmy.
- OpenCTI je vhodnejší pre ***znalostnú a analytickú vrstvu CTI***, kde poskytuje bohatší kontext hrozieb, vzťahy medzi aktérmi, kampaňami a technikami (MITRE ATT&CK) a podporuje strategickú a taktickú analýzu.

V kombinovanom nasadení môže MISP plniť úlohu rýchleho operatívneho zdroja indikátorov, zatiaľ čo OpenCTI funguje ako centrálna znalostná báza hrozieb, ktorá tieto indikátory obohacuje o vyšší kontext. Takýto model umožňuje rozdeliť zodpovednosti medzi nástroje bez zbytočnej duplicity a zvyšuje flexibilitu SOC architektúry.

Z pohľadu návrhu SOC infraštruktúry je preto možné:

- použiť iba MISP alebo iba OpenCTI (v závislosti od zamerania SOC),
- alebo ich kombinovať, pričom každý nástroj sa venuje inej vrstve threat intelligence.

Tento prístup umožňuje škálovať CTI schopnosti SOC podľa potrieb organizácie bez nutnosti viazať sa na jediné riešenie.

7 GLPI

Typ nástroja: Asset Management/IT Service Management (ITSM)/Ticketing

Názov nástroja: GLPI

Verzia: GLPI 10.0.16 (open-source verzia, kontajnerizované nasadenie)

7.1 Opis systému

GLPI (Gestionnaire Libre de Parc Informatique) je open-source nástroj určený na evidenciu IT aktív, správu konfigurácií (CMDB) a riadenie IT služieb (ITSM). V prostredí SOC plní úlohu centrálneho evidenčného a procesného systému, ktorý zabezpečuje prepojenie technických bezpečnostných udalostí s organizačným kontextom, vlastníkmi aktív a poskytovanými službami.

Hlavným účelom nasadenia GLPI je:

- evidencia aktív (hardvér, softvér, virtuálne systémy, sieťové prvky),
- CMDB/ITSM evidencia vrátane vzťahov medzi aktívami, službami a používateľmi,
- väzba aktív na vlastníkov, zodpovedné osoby a organizačné jednotky,
- správa incidentov, požiadaviek a zmien prostredníctvom ticketing systému.

GLPI umožňuje modelovať IT infraštruktúru a služby z procesného hľadiska – teda odpovedať na otázky *„komu aktívum patrí“, „akú službu podporuje“ a „kto je zodpovedný za jeho prevádzku“*. Tento pohľad dopĺňa technickú dokumentáciu poskytovanú nástrojmi ako NetBox a vytvára most medzi infraštruktúrnou, bezpečnostnou a organizačnou rovinou.

V navrhovanej SOC architektúre je GLPI využívaný ako:

- systém pre formálnu evidenciu bezpečnostných incidentov,
- nástroj pre sledovanie životného cyklu incidentu (od nahlásenia po uzavretie),
- komponent zabezpečujúci auditovateľnosť, zodpovednosť a procesnú konzistenciu riešenia incidentov.

GLPI tak nevystupuje ako detekčný alebo reakčný nástroj, ale ako procesná a evidenčná vrstva SOC, ktorá zabezpečuje, že technické udalosti identifikované SIEM a spracované SOAR nástrojmi sú správne zdokumentované, priradené k aktívam a riešené v súlade s internými postupmi a zodpovednosťami.

7.2 Dokumentácia systému

Oficiálna dokumentácia nástroja GLPI je verejne dostupná a pokrýva všetky hlavné oblasti systému – od inštalácie a konfigurácie až po správu aktív, ticketing, workflow a integrácie prostredníctvom API a pluginov. Dokumentácia reflektuje komunitný vývoj projektu a podporuje rôzne modely nasadenia vrátane kontajnerizovaného prostredia.

Odkazy na oficiálne zdroje:

- Oficiálna dokumentácia: https://glpi-project.org/documentation/
- Oficiálna stránka projektu: https://glpi-project.org/

- GitHub repozitár: https://github.com/glpi-project/glpi
- Oficiálny Docker image: https://hub.docker.com/_/glpi
- Oficiálne doplnky a pluginy: https://glpi-plugins.readthedocs.io/en/latest/

7.3 Požiadavky na zdroje

Hardvérové požiadavky GLPI závisia od počtu evidovaných aktív, používateľov a objemu spracovávaných ticketov. Pre testovacie alebo menšie produkčné nasadenie postačujú nasledovné parametre:

Minimálne požiadavky (PoC / menšie prostredie):

- CPU: 2 jadrá
- RAM: 4 GB
- Diskový priestor: 20–40 GB
- Databáza: MySQL / MariaDB

Odporúčané požiadavky (produkčné SOC prostredie):

- CPU: 4–8 jadier
- RAM: 8–16 GB
- Disk: SSD úložisko
- Databáza: samostatná DB inštancia

Vyššie nároky sa uplatňujú najmä pri intenzívnom používaní ticketing systému, veľkom množstve aktív a rozšíreniach prostredníctvom pluginov.

7.4 Architektúra nástroja

Architektúra nástroja GLPI je navrhnutá ako viacvrstvová webová aplikácia, ktorá oddeľuje prezentačnú, aplikačnú a dátovú vrstvu. Tento architektonický prístup umožňuje jednoduché nasadenie, škálovanie a integráciu do existujúcej IT a SOC infraštruktúry bez potreby špecializovaných komponentov.

Základom riešenia je webová aplikačná vrstva, ktorá poskytuje používateľské rozhranie pre správu aktív, incidentov, zmien a služieb. GLPI je implementované ako PHP aplikácia bežiaca nad webovým serverom (Apache alebo Nginx) a je prístupné prostredníctvom štandardného webového prehliadača. Táto vrstva je určená pre analytikov SOC, IT administrátorov a ďalších používateľov zapojených do riešenia incidentov.

Aplikačná logika systému zabezpečuje:

- správu aktív a CMDB vzťahov,
- spracovanie ticketov (incidenty, požiadavky, zmeny),
- workflow a eskalačné pravidlá,
- notifikácie a históriu zásahov,
- integračné mechanizmy prostredníctvom REST API a pluginov.

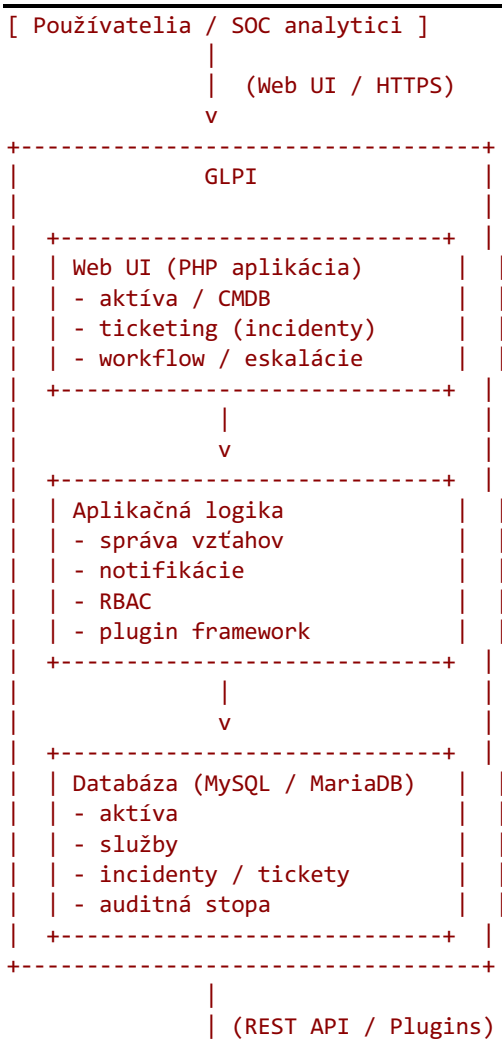
Dátová vrstva je reprezentovaná relačnou databázou (MySQL alebo MariaDB), ktorá uchováva všetky prevádzkové a historické dáta systému, vrátane:

- evidencie aktív a ich vzťahov,
- záznamov o incidentoch a ticketoch,
- používateľských účtov a rolí,
- auditnej stopy zmien a zásahov.

GLPI podporuje pluginový mechanizmus, ktorý umožňuje rozširovať funkcionalitu bez zásahu do jadra aplikácie. Pluginy sú využívané najmä na integráciu s externými nástrojmi (monitoring, inventarizácia, bezpečnostné systémy) alebo na rozšírenie CMDB a ITSM funkcií.

V kontexte SOC architektúry je GLPI umiestnené za detekčnou a reakčnou vrstvou. Nevykonáva aktívnu detekciu ani automatizované reakcie, ale slúži ako procesný a evidenčný systém, ktorý formálne zachytáva bezpečnostné incidenty identifikované inými nástrojmi a zabezpečuje ich riadené spracovanie.

Schéma



```
              v
+----------------------------------+
| Externé systémy                  |
| - ELK Stack (detekcia)           |
| - Shuffle (SOAR)                 |
| - NetBox (SSOT aktív)            |
| - E-mail / notifikácie           |
+----------------------------------+
```

7.5 Inštalácia – variant Docker (Compose)

Docker-compose.yml (zhodný s oficiálnou dokumentáciou, pridaný NGINX ako HTTPS reverse proxy pre SSL/TLS)

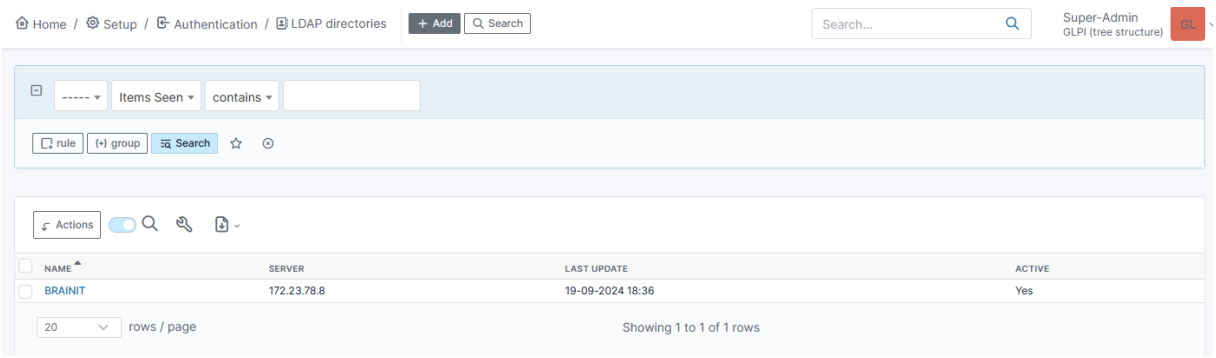
```
services:
  glpi:
    image: "glpi/glpi:latest"
    restart: "unless-stopped"
    volumes:
      - "./storage/glpi:/var/glpi:rw"
    env_file: .env # Pass environment variables from .env file to the container
    depends_on:
      db:
        condition: service_healthy
    ports:
      - "80:80"
  db:
    image: "mysql"
    restart: "unless-stopped"
    volumes:
       - "./storage/mysql:/var/lib/mysql"
    environment:
      MYSQL_RANDOM_ROOT_PASSWORD: "yes"
      MYSQL_DATABASE: ${GLPI_DB_NAME}
      MYSQL_USER: ${GLPI_DB_USER}
      MYSQL_PASSWORD: ${GLPI_DB_PASSWORD}
    healthcheck:
      test: mysqladmin ping -h 127.0.0.1 -u $$MYSQL_USER --password=$$MYSQL_PASSWORD
      start_period: 5s
      interval: 5s
      timeout: 5s
      retries: 10
    expose:
      - "3306"
  nginx-https-proxy:
    image: nginx:latest
    restart: always
    ports:
      - "443:443"
    volumes:
      - ./storage/nginx/conf.d/:/etc/nginx/conf.d/:ro
      - ./storage/nginx/certs/:/etc/nginx/certs/:ro
    environment:
```

```
    - TIMEZONE=Europe/Bratislava
    - NGINX_HOST=glpi.brainit.tech
    - NGINX_PORT=443
```

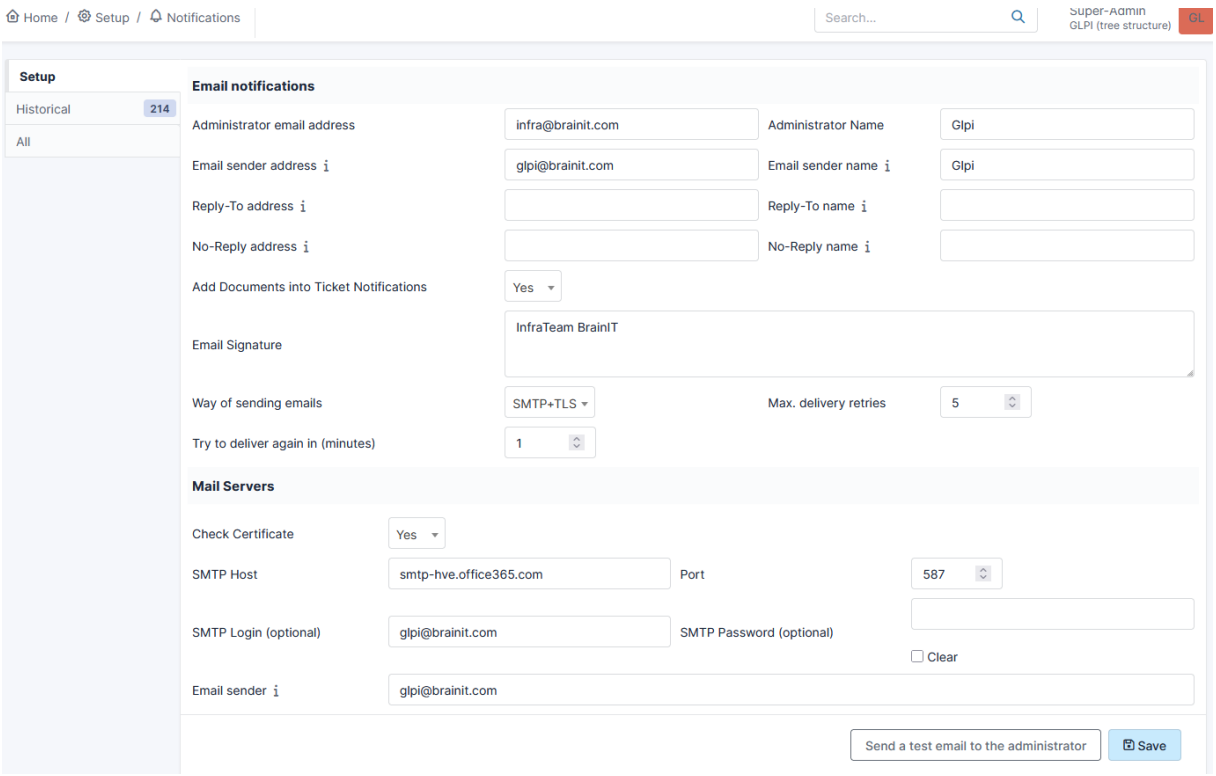
Súbor .env

```
GLPI_DB_HOST=db
GLPI_DB_PORT=3306
GLPI_DB_NAME=glpi
GLPI_DB_USER=glpi
GLPI_DB_PASSWORD=glpi
```

7.6 Základná konfigurácia

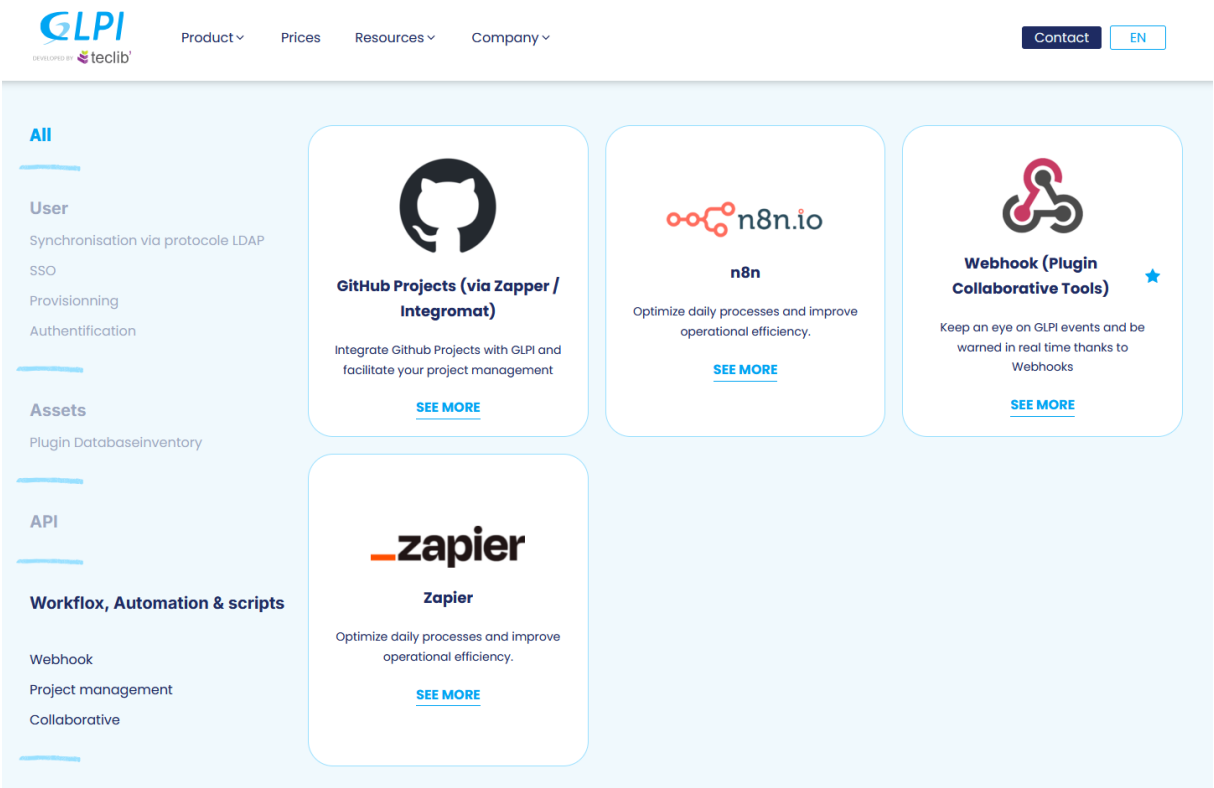


Autentifikácia cez AD/LDAP pre doménu brainit.tech

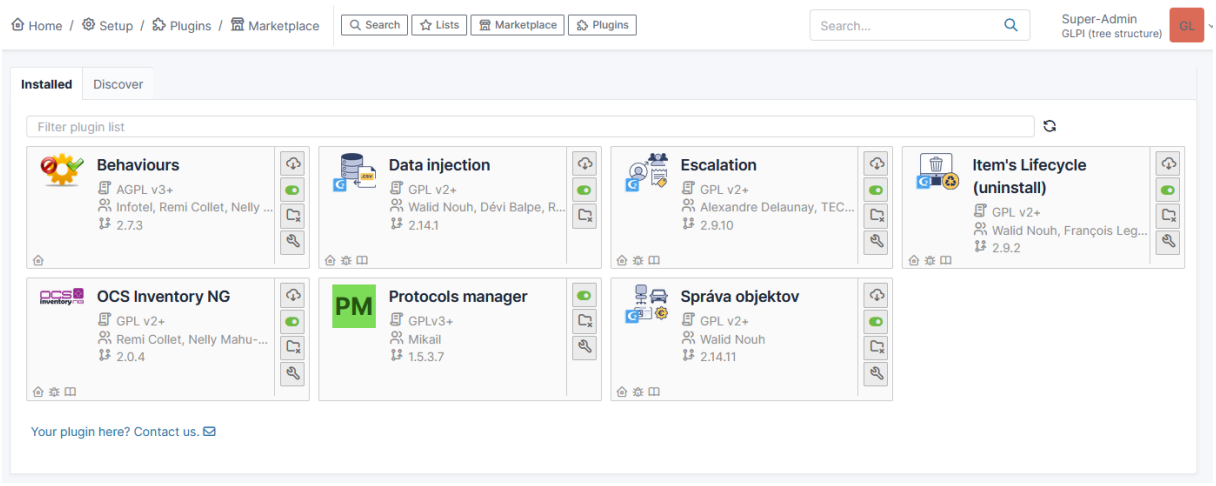


Nastavenie emailových notifikácií ako glpi@brainit.com

7.7 Integrácie



Viac o možnostiach integrácií na: https://www.glpi-project.org/en/integrations/



Zoznam pluginov v našej inštancii:

- pridanie povinných polí, vynucovanie formátovania údajov, https://github.com/InfotelGLPI/behaviors
- ticket management a eskalácia https://github.com/pluginsGLPI/escalade
- pridanie inventárnych stavov pre vyradzovanie

https://github.com/pluginsglpi/uninstall
- pridávanie vlastných typov objektov https://github.com/pluginsGLPI/genericobject
- pre synchronizáciu s „OCS Inventory NG" https://github.com/pluginsGLPI/ocsinventoryng
- import dát vo formáte CSV https://plugins.glpi-project.org/#/plugin/datainjection

Pozícia GLPI v SOC

Z architektonického hľadiska je GLPI procesným jadrom riešenia incidentov, ktoré:

- prijíma technické výstupy z detekčných a reakčných nástrojov,
- mapuje ich na aktíva, vlastníkov a služby,
- zabezpečuje auditovateľné a riadené spracovanie incidentov.

Tento prístup umožňuje oddeliť technickú bezpečnostnú logiku (SIEM/SOAR) od procesného riadenia incidentov (ITSM/CMDB), čo zvyšuje prehľadnosť, škálovateľnosť a dlhodobú udržateľnosť SOC architektúry.

8 NETBOX

Typ nástroja: Asset management/IPAM/DCIM

Názov nástroja: NetBOX

Verzia: NetBox 4.x (nasadenie prostredníctvom oficiálnych Docker image)

8.1 Opis systému

NetBox je open-source nástroj určený na centrálne riadenie a evidenciu sieťovej a infraštruktúrnej dokumentácie, ktorý v prostredí SOC plní úlohu centrálneho zdroja pravdy (Single Source of Truth – SSOT). Slúži na jednotnú evidenciu IP adresných priestorov, sieťových zariadení, fyzických a logických prepojení, lokalít, rackov a ďalších infraštruktúrnych objektov, ktoré sú nevyhnutné pre správnu prevádzku, monitoring a bezpečnostné analýzy.

V rámci navrhovanej infraštruktúry predstavuje NetBox základnú referenčnú vrstvu, z ktorej vychádzajú ďalšie bezpečnostné a prevádzkové nástroje. Umožňuje presnú evidenciu IP rozsahov, VLAN sietí, prefixov a jednotlivých IP adries, čím vytvára konzistentný podklad pre sieťové skeny, skenovanie zraniteľností a koreláciu bezpečnostných udalostí v SIEM nástrojoch či automatizáciu. Zároveň poskytuje detailný prehľad o fyzickej topológii siete, zariadeniach a ich prepojeniach, čo je kľúčové pri analýze incidentov, plánovaní zmien a hodnotení dopadov výpadkov.

NetBox podporuje modelovanie infraštruktúry na viacerých úrovniach – od fyzickej vrstvy (rack, zariadenia, porty, káble, napájanie) až po logickú vrstvu (IPAM, VLAN, VPN, virtuálne stroje). Vďaka tomu je možné v jednom systéme prepojiť technické, prevádzkové aj administratívne informácie o infraštruktúre. Súčasťou riešenia je aj REST API, ktoré umožňuje integráciu s externými nástrojmi, automatizačnými skriptmi a bezpečnostnými platformami.

V kontexte multitenantného SOC riešenia je NetBox využívaný ako stabilný a konzistentný zdroj údajov, ktorý znižuje nejednoznačnosť v interpretácii infraštruktúrnych dát a podporuje automatizáciu procesov. Jeho úloha nie je aktívna detekcia alebo reakcia na incidenty, ale poskytovanie presných a aktuálnych dát, na ktoré sa ostatné komponenty SOC infraštruktúry spoľahlivo odvolávajú.

8.2 Dokumentácia

Oficiálna dokumentácia nástroja NetBox je verejne dostupná a priebežne aktualizovaná komunitou a vývojármi projektu. Dokumentácia pokrýva všetky hlavné oblasti systému, vrátane architektúry, inštalačných postupov, konfiguračných možností, práce s objektovým modelom, rozhrania REST API, ako aj možností rozšírenia prostredníctvom pluginov a vlastných skriptov. Dokumentácia zároveň reflektuje rýchly vývoj projektu a obsahuje odporúčania pre produkčné nasadenie aj prevádzku vo väčších infraštruktúrach.

- Oficiálna dokumentácia: https://github.com/netbox-community/netbox
- Oficiálna stránka projektu NetBox: https://netboxlabs.com/
- GitHub repozitár projektu: https://github.com/netbox-community/netbox
- Oficiálny Docker image (NetBox a súvisiace komponenty): https://hub.docker.com/r/netboxcommunity/netbox

8.3 Požiadavky na zdroje

Hardvérové požiadavky NetBoxu závisia od rozsahu dokumentovanej infraštruktúry a intenzity využívania API a automatizácie. Pre základné až stredne veľké nasadenie sú odporúčané nasledovné minimálne parametre:

- **CPU:** 2 vCPU
- **RAM:** 4 GB
- **Diskový priestor:** 20 GB (SSD odporúčané)
- **Databáza:** PostgreSQL (samostatný kontajner alebo externá inštancia)
- **Cache/Queue:** Redis

Pri väčších nasadeniach alebo intenzívnom využívaní automatizácie a pluginov sa odporúča navýšiť pamäťové a výpočtové zdroje.

8.4 Architektúra nástroja

Architektúra nástroja NetBox je založená na viacvrstvovom modeli webovej aplikácie, v ktorom sú jednotlivé funkčné komponenty oddelené do samostatných logických a prevádzkových vrstiev. Tento prístup umožňuje škálovateľnosť, jednoduchú údržbu a jasné oddelenie aplikačnej logiky od perzistentných dát a pomocných služieb.

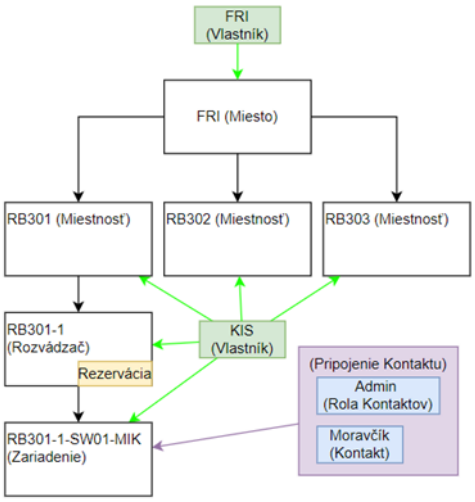
V nasadení využívajúcom kontajnerizačné technológie je NetBox prevádzkovaný ako súbor vzájomne prepojených Docker kontajnerov, pričom každý z nich plní špecifickú úlohu v rámci celkového riešenia. Aplikačnú vrstvu tvorí samotná NetBox aplikácia, ktorá zabezpečuje webové používateľské rozhranie a REST API. Táto vrstva spracúva požiadavky používateľov a externých systémov a vykonáva aplikačnú logiku nad uloženými dátami.

Dátová vrstva je reprezentovaná databázovým systémom PostgreSQL, ktorý uchováva všetky konfiguračné a dokumentačné údaje infraštruktúry. Pre optimalizáciu výkonu a spracovanie úloh na pozadí je použitý systém Redis, ktorý slúži ako cache a zároveň ako fronta pre asynchrónne úlohy. Tieto úlohy sú spracovávané samostatnými worker procesmi, čím sa minimalizuje zaťaženie hlavnej aplikačnej vrstvy.

Komunikácia medzi jednotlivými komponentmi prebieha prostredníctvom definovaných sieťových rozhraní v rámci Docker prostredia, pričom architektúra umožňuje nasadenie všetkých služieb na jednom hoste alebo ich rozdelenie na viaceré uzly. Tento model je v súlade s oficiálnou Docker-based architektúrou NetBoxu a predstavuje odporúčaný spôsob nasadenia pre produkčné prostredia.

Textová schéma architektúry uvedená nižšie ilustruje základné vzťahy medzi jednotlivými komponentmi NetBoxu v kontajnerizovanom prostredí a slúži ako referenčný prehľad architektúry nasadenia.

```
[ User / API client ]
                |
        [ NetBox ]
         /       \
[ PostgreSQL ] [ Redis ]
                    |
              [ RQ Workers ]
```



1 Vzťahy – Organizácia

8.4.1 Modelovanie infraštruktúry

8.4.1.1 Modelovanie vzťahov a objektov v NetBoxe – princípy

NetBox je postavený na objektovom modeli infraštruktúry, v ktorom sú jednotlivé prvky siete reprezentované ako samostatné objekty s definovanými vlastnosťami a vzťahmi. Základným princípom tohto prístupu je oddelenie definície objektov (čo objekt je) od ich vzájomných väzieb (ako spolu súvisia). Vďaka tomu je možné modelovať infraštruktúru spôsobom, ktorý verne odráža jej reálnu fyzickú aj logickú štruktúru.

Vzťahy medzi objektmi v NetBoxe nie sú náhodné, ale vychádzajú z preddefinovaných pravidiel a hierarchie, ktoré určujú, aké objekty môžu existovať samostatne a ktoré sú závislé od iných. Typickým príkladom je väzba zariadenia na lokalitu alebo miestnosť, prípadne väzba IP adries na prefixy a VLAN siete. Tieto väzby zabezpečujú konzistenciu dát a zabraňujú vzniku nejednoznačných alebo neúplných záznamov v dokumentácii.

Modelovanie je založené na princípe postupného spresňovania – najskôr sa definujú všeobecné kontextové objekty (organizácia, lokality, vlastníci), na ktoré sa následne viažu technické objekty infraštruktúry (zariadenia, rozhrania, IP adresy). Takýto prístup umožňuje jednotnú dokumentáciu infraštruktúry naprieč rôznymi úrovňami detailu a podporuje škálovateľnosť modelu pri raste siete.

Dôležitým aspektom modelovania v NetBoxe je aj dôraz na jednoznačný zdroj pravdy. Každý objekt a jeho vzťahy majú byť evidované len na jednom mieste a ostatné systémy sa na tieto dáta odvolávajú prostredníctvom API alebo exportov. V kontexte SOC infraštruktúry to znamená, že NetBox neposkytuje aktívne monitorovanie alebo bezpečnostnú analytiku, ale vytvára presný a konzistentný referenčný rámec, ktorý je nevyhnutný pre správnu interpretáciu dát v ďalších nástrojoch.

Na základe týchto princípov sú v nasledujúcich kapitolách rozpracované konkrétne kategórie objektov a ich vzťahy, so zameraním na organizačné objekty a objekty zariadení, ktoré tvoria základ dokumentácie sieťovej infraštruktúry.

8.4.1.2 Modelovanie objektov organizácie

V NetBoxe sa kategória **Organizácia (Organization)** používa na modelovanie netechnického kontextu infraštruktúry – najmä **vlastníctva**, **umiestnenia** a **kontaktných osôb**. V praxi sa tento

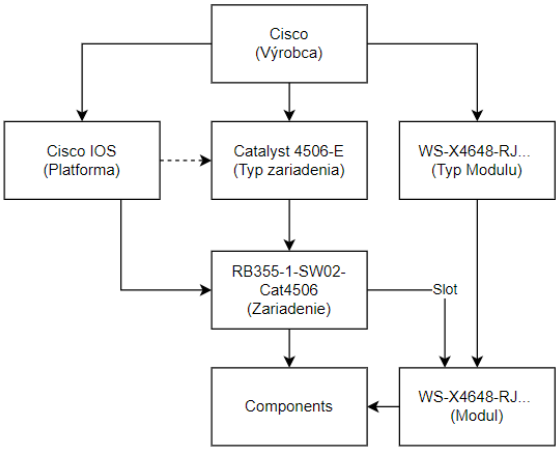
model opiera o základné objekty **Sites (Miesta)** a **Locations (Miestnosti)**, ktoré zabezpečujú, že každé zariadenie má definované, *kde sa fyzicky nachádza*. V prostredí KIS je model zjednodušený na jednu pobočku (Site) reprezentujúcu budovu, napr. **FRI**, pričom jednotlivé učebne, kancelárie a technické priestory sú evidované ako **Locations** viazané na daný Site. Do miestností sa následne umiestňujú rozvádzače (Racks) alebo priamo zariadenia (Devices).

Dôležitou súčasťou organizačného modelu sú **Tenants (Vlastníci)**, ktoré určujú, komu patria objekty ako zariadenia, miesta alebo IP prefixy. V tomto prístupe má každý objekt priradeného najviac jedného vlastníka a vlastníctvo sa používa najmä na administratívne účely a prehľadnosť správy (napr. rozlíšenie FRI/KIS/iný vlastník).

Model dopĺňajú **Contacts (Kontakty)**, **Contact Groups** a **Contact Roles**, ktoré umožňujú evidovať zodpovedné osoby a ich roly pri správe infraštruktúry (napr. administrátor zariadenia). Prepojením kontaktu, jeho roly a konkrétneho objektu vzniká „contact assignment“ – praktická väzba, ktorá podporuje rýchlu eskaláciu a prevádzkovú orientáciu v správe aktív.

Z pohľadu odporúčanej postupnosti tvorby objektov (podľa internej schémy vzťahov) je kľúčové najprv definovať vlastníkov, následne pobočky (Sites), potom miestnosti (Locations) a až následne nadväzujúce objekty infraštruktúry. Táto logika je zdôraznená aj v schéme „Vzťahy kategória organizácia“, ktorá popisuje odporúčanú postupnosť a povinné väzby medzi objektmi.

8.4.1.3 Modelovanie objektov infraštruktúry



Vzťahy – Zariadenia

Kategória Zariadenia (Devices) v nástroji NetBox slúži na modelovanie fyzickej sieťovej infraštruktúry a jej komponentov. Zariadenia predstavujú aktívne aj pasívne prvky siete, ako sú prepínače, smerovače, firewally, bezdrôtové prístupové body, ale aj patch panely či rozvodné káble. Každé zariadenie je viazané na konkrétnu lokalitu (Site) a miestnosť (Location), čím je zabezpečená presná fyzická lokalizácia v rámci infraštruktúry. Pri vytváraní zariadenia je povinné definovať aspoň jeho rolu, typ zariadenia, umiestnenie a stav.

Základom modelovania zariadení sú Typy zariadení (Device Types), ktoré fungujú ako šablóny. Tieto šablóny obsahujú informácie o modeli zariadenia, jeho rozmeroch (U jednotky), rozhraniach, moduloch, napájacích a konzolových portoch. Použitím typov zariadení sa zabezpečuje konzistentnosť dokumentácie a automatické vytváranie všetkých potrebných komponentov pri zakladaní nového zariadenia. Typy zariadení sú spravované centrálne a môžu byť importované aj z verejných repozitárov výrobcu alebo komunity.

Súčasťou kategórie zariadení sú aj Roly zariadení (Device Roles), ktoré vyjadrujú funkciu zariadenia v sieti (napr. prepínač, smerovač, firewall). Roly umožňujú logické triedenie zariadení a ich ďalšie využitie v automatizácii, filtrovaní alebo pri generovaní konfigurácií.

Platformy (Platforms) definujú typ operačného systému zariadenia a slúžia na zjednodušené rozlíšenie technologických platforiem bez potreby evidovať presné verzie softvéru.

NetBox ďalej umožňuje modelovanie pokročilejších konštrukcií, ako sú virtuálne šasi (Virtual Chassis), ktoré reprezentujú viacero fyzických zariadení zdieľajúcich jednu riadiacu rovinu (napr. technológie StackWise alebo vPC). Zariadenia môžu obsahovať moduly, rozhrania, konzolové porty, napájacie porty a ďalšie komponenty, ktoré umožňujú detailné a realistické zobrazenie fyzickej topológie siete. Tieto komponenty sú následne využívané pri dokumentácii prepojení a vizualizácii topológie.

Modelovanie zariadení v NetBoxe vytvára presný a jednotný obraz fyzickej infraštruktúry, ktorý je využiteľný ako podklad pre sieťové mapy, monitoring, automatizáciu a bezpečnostné analýzy v rámci SOC prostredia.

8.5 Experiment

Cieľom experimentálnej časti bolo overiť nasadenie a praktické využitie nástroja NetBox ako centrálneho zdroja pravdy pre dokumentáciu aktív relevantných pre SOC. Proof of Concept (PoC) bol riešený primárne za účelom zavedenia konzistentnej evidencie aktív (zariadenia, lokality, prepojenia, IP adresácia a vybrané služby) tak, aby bolo možné tieto dáta ďalej využívať pri bezpečnostnom monitoringu, skenovaní, korelácii udalostí a pri riešení incidentov. Nešlo teda o cieľ vytvoriť kompletné produkčné riešenie, ale o kontrolované overenie základných princípov dokumentácie a pripravenosti dátového modelu pre potreby SOC.

PoC nadväzuje na interné skúsenosti s dokumentáciou siete a infraštruktúry a na publikované výsledky, ktoré preukazujú, že NetBox je vhodný na nahradenie statickej dokumentácie (wiki, tabuľky) databázovým prístupom a poskytuje stabilný základ pre ďalšie procesy prevádzky a automatizácie. V rámci experimentu bol kladený dôraz na to, aby dokumentácia nevznikala ako izolovaný „inventár“, ale ako referenčná vrstva pre nadväzujúce SOC činnosti – najmä na účely identifikácie a klasifikácie aktív, mapovania topológie a jednoznačného priradenia IP adries a sieťových segmentov ku konkrétnym zariadeniam a lokalitám.

Súčasťou PoC bolo overenie viacerých variantov nasadenia NetBoxu, pričom hlavná pozornosť bola venovaná kontajnerizovanému nasadeniu pomocou Dockeru. Overená bola aj základná konfigurácia systému, rozšíriteľnosť prostredníctvom pluginov a možnosti integrácie s automatizačnými nástrojmi (najmä pre opakovateľné prevádzkové úlohy, ako je správa inventára a zálohovanie konfigurácií). Výstupy experimentu predstavujú technický základ pre ďalšie rozšírenie dokumentácie aktív a jej prepojenie s ostatnými komponentmi SOC infraštruktúry.

8.5.1 Schéma

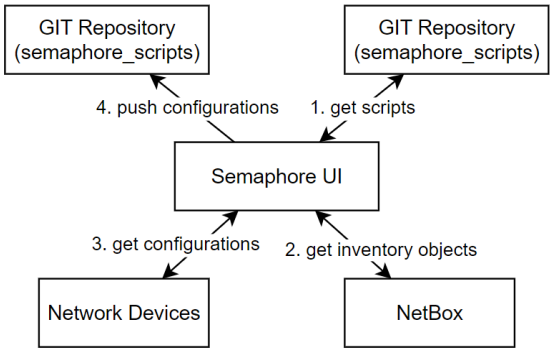


Schéma experimentu

Schéma experimentu znázorňuje PoC workflow, v ktorom je NetBox použitý ako centrálny register aktív (asset inventory / SSOT) pre potreby SOC. Primárnym cieľom je zabezpečiť, aby SOC procesy pracovali s konzistentnými údajmi o tom, aké aktíva existujú, kde sa nachádzajú, aké majú IP adresy/segmenty a aké sú medzi nimi vzťahy. Táto vrstva „pravdy“ je kritická pre korektnú interpretáciu udalostí (logs/alerts), mapovanie incidentov na konkrétne zariadenia a pre presnú segmentačnú orientáciu počas vyšetrovania. V schéme vystupuje Semaphore UI ako orchestrátor automatizačných úloh (Ansible), pričom zdroje informácií sú rozdelené:

1. Git repozitár poskytuje skripty a definície úloh (runbook/automation-as-code),
2. NetBox poskytuje inventárne údaje o aktívach (zariadenia, roly, IP adresácia, vzťahy),
3. Sieťové zariadenia sú cieľom operácií (získanie konfigurácií).

Z pohľadu SOC ide o nasledovnú logiku:

- Asset (inventár): NetBox definuje autoritatívny pohľad na aktíva a ich kontext (kde je zariadenie, aké má rozhrania, IP adresy, segmenty).
- Detection (korektná interpretácia): konzistentné asset dáta znižujú nejednoznačnosť pri mapovaní udalostí na konkrétny objekt (napr. „táto IP adresa patrí tomuto zariadeniu v tejto lokalite“). To podporuje triage a prioritizáciu incidentov, lebo aktívum je možné rýchlo identifikovať a zaradiť do správneho kontextu.
- Response (opakovateľné operácie): Semaphore/Ansible vykonáva štandardizované úlohy (v tomto PoC najmä zálohovanie konfigurácií), čím vytvára prevádzkový artefakt využiteľný pri incidente: referenčný stav konfigurácie pre analýzu zmien, obnovu a forenzné porovnanie. Výstup (konfigurácie) sa verzionuje v Git repozitári, čo podporuje auditovateľnosť a spätnú dohľadateľnosť zmien.

Experiment teda demonštruje, že NetBox ako asset SSOT nie je „detekčný nástroj“, ale poskytuje kľúčový základ pre SOC: správny kontext aktív pre detekciu a opakovateľný mechanizmus podpory reakcie (cez automatizované prevádzkové úlohy a verzované konfigurácie). Zároveň potvrdzuje, že NetBox je vhodný ako referenčná databáza pre automatizované činnosti relevantné pre SOC, bez toho aby sám vykonával aktívne zásahy do infraštruktúry.

8.5.2 Inštalácia – variant Docker

Nasadenie NetBoxu v PoC je realizované kontajnerizovane pomocou **Docker Compose**. V rámci experimentálneho PoC nasadenia bežali nasledovné Docker kontajnery:

1. netbox – hlavná aplikácia (Web UI + REST API)

2. netbox-worker – background worker (RQ)
3. netbox-housekeeping – housekeeping plánované úlohy
4. postgres – databáza (PostgreSQL)
5. redis – queue/cache vrstva (u vás je to Valkey image)
6. redis-cache – separátny cache/redis endpoint (tiež Valkey)

Konfigurácia je rozdelená do viacerých súborov tak, aby bolo možné oddeliť

- definíciu služieb,
- runtime premenné prostredia a
- aplikačnú konfiguráciu NetBoxu vrátane pluginov.

Nasadenie je realizované cez kombináciu docker-compose.yml a docker-compose.override.yml (compose automaticky načíta override súbor, ak je v rovnakom priečinku).

Konfigurácia aplikačných služieb (NetBox)

Základná konfigurácia je postavená na kombinácii:

- netbox.env – runtime premenné,
- configuration.py – NetBox/Django konfigurácia, ktorá tieto premenné mapuje do systému.

Kľúčové nastavenia v PoC (výber):

- **DB pripojenie:** DB_HOST=postgres, DB_NAME=netbox, DB_USER=netbox (heslo je v env).
- **Redis oddelený na „tasks“ a „cache“:**
  - tasks: REDIS_HOST=redis, REDIS_DATABASE=0
  - cache: REDIS_CACHE_HOST=redis-cache, REDIS_CACHE_DATABASE=1
    Toto je priamo reflektované aj v configuration.py, kde sú dve samostatné sekcie REDIS['tasks'] a REDIS['caching'] s odporúčaním používať odlišné DB ID.
- **API a integračné rozšírenia:** GRAPHQL_ENABLED=true, WEBHOOKS_ENABLED=true (pripravenosť na integrácie a automatizáciu).
- **CORS:** CORS_ORIGIN_ALLOW_ALL=True (otvorené CORS v PoC režime).
- **Housekeeping:** HOUSEKEEPING_INTERVAL=86400 (periodicita 24h).

Pluginy (inštalácia + aktivácia)

Pluginy sú riešené dvojkrokom:

- **(a) Inštalácia balíkov do image**
  V plugin_requirements.txt sú definované pluginy:
  - netbox-topology-views
  - netbox-reorder-rack

- netbox-attachments

- **(b) Aktivácia a konfigurácia v NetBoxe**
  V plugins.py sú pluginy aktivované cez PLUGINS a nastavené cez PLUGINS_CONFIG. Konkrétne netbox_topology_views má zapnuté ukladanie súradníc (vizualizácia topológie).

8.5.2.1 Predpoklady spustenia

- Hostiteľský systém: Linux (alebo VM v OpenStack IaaS), sieťový prístup podľa potreby PoC.
- Nainštalované nástroje:
  - **Docker Engine**
  - **Docker Compose v2** (príkaz docker compose …)
- Pripravená adresárová štruktúra podľa PoC (t. j. netbox/ obsahuje docker-compose.yml, docker-compose.override.yml, Dockerfile, configuration/ a env/) **ako je ďalej**.

8.5.2.2 Zoznam súborov (kompozitný compose)

- **docker-compose.yml:** Základný orchestration súbor pre Docker Compose – definuje služby NetBox stacku (NetBox aplikácia, worker procesy, housekeeping, databáza PostgreSQL a Redis vrstvy), ich závislosti, sieťovanie a perzistentné volumes.
- **docker-compose.override.yml:** Override súbor, ktorým sa dopĺňajú alebo prepisujú parametre z docker-compose.yml (typicky lokálne úpravy, mounty konfiguračných súborov, build nastavenia). Vďaka tomu je možné zachovať základnú compose štruktúru a meniť len potrebné časti pre konkrétne prostredie.
- **Dockerfile:** Vlastný build aplikačného image nad oficiálnym NetBox image, použitý na doplnenie pluginov/balíkov. V PoC sa používa spolu s plugin_requirements.txt (inštalácia vybraných NetBox pluginov).
- **plugin_requirements.txt:** Zoznam plugin balíkov, ktoré sa majú doinštalovať do NetBox image. V tomto nasadení sú definované pluginy:
  - netbox-topology-views
  - netbox-reorder-rack
  - netbox-attachments
- **netbox.env:** Centrálne runtime premenné prostredia pre NetBox stack. V PoC sú nastavené kľúčové parametre:
  - DB pripojenie (DB_HOST=postgres, DB_NAME=netbox, DB_USER=netbox)
  - Redis pre tasks (REDIS_HOST=redis, REDIS_DATABASE=0)
  - Redis cache (REDIS_CACHE_HOST=redis-cache, REDIS_CACHE_DATABASE=1)
  - funkčné moduly: GRAPHQL_ENABLED=true, WEBHOOKS_ENABLED=true
  - housekeeping interval: HOUSEKEEPING_INTERVAL=86400
- **redis-cache.env:** Premenné pre cache Redis/Valkey vrstvu (v PoC obsahuje napr. heslo pre cache backend).

- **Configuration.py:** Hlavná aplikačná konfigurácia NetBoxu, ktorá číta hodnoty z netbox.env a mapuje ich do Django/NetBox nastavení. Zásadné je, že konfig nastavuje:
  - databázové pripojenie cez premenné DB_* (PostgreSQL)
  - dve Redis konfigurácie oddelene: tasks a caching (oddelené databázové ID a endpointy)
  - CORS a ďalšie voliteľné nastavenia cez env premenné
- **plugins.py:** Konfigurácia pluginov – explicitne definuje, ktoré pluginy sú aktívne a ich nastavenia. V PoC sú aktivované:
  - netbox_topology_views (s ukladaním súradníc a statickým adresárom obrázkov)
  - netbox_reorder_rack
  - netbox_attachments

8.5.2.2.1 Štruktúra súborov

```
+---netbox
|   |   docker-compose.yml
|   |   docker-compose.override.yml
|   |   Dockerfile
|   |   plugin_requirements.txt
|   |
|   +---configuration
|   |       configuration.py
|   |       plugins.py
|   |
|   \---env
|           netbox.env
|           postgres.env
|           redis-cache.env
|           redis.env
|
\---semaphore
        docker-compose.yml
        requirements.txt
```

8.5.2.3 Funkčný princíp nasadenia (čo zabezpečuje „plnú funkčnosť“)

- Compose zabezpečí spustenie všetkých potrebných služieb (aplikácia + databáza + queue/cache vrstvy + background procesy).
- netbox.env drží runtime parametre (DB/Redis, zapnuté API moduly a housekeeping interval), ktoré sa prenesú do kontajnerov.
- configuration.py je “source of truth” pre aplikačné nastavenia – mapuje env premenné do NetBox konfigurácie a zabezpečí korektné pripojenie na PostgreSQL a oba Redis backendy (tasks + caching).
- Pluginy sú riešené dvojkrokom: (a) inštalácia balíkov (plugin_requirements.txt) a (b) aktivácia + nastavenie (plugins.py).

Netbox\docker-compose.yml

```
services:
  netbox: &netbox
    image: docker.io/netboxcommunity/netbox:${VERSION-v4.2-3.2.0}
    depends_on:
      - postgres
      - redis
      - redis-cache
    env_file: env/netbox.env
    user: "unit:root"
    healthcheck:
      test: curl -f http://localhost:8080/login/ || exit 1
      start_period: 90s
      timeout: 3s
      interval: 15s
    volumes:
      - ./configuration:/etc/netbox/config:z,ro
      - netbox-media-files:/opt/netbox/netbox/media:rw
      - netbox-reports-files:/opt/netbox/netbox/reports:rw
      - netbox-scripts-files:/opt/netbox/netbox/scripts:rw
  netbox-worker:
    <<: *netbox
    depends_on:
      netbox:
        condition: service_healthy
    command:
      - /opt/netbox/venv/bin/python
      - /opt/netbox/netbox/manage.py
      - rqworker
    healthcheck:
      test: ps -aux | grep -v grep | grep -q rqworker || exit 1
      start_period: 20s
      timeout: 3s
      interval: 15s
  netbox-housekeeping:
    <<: *netbox
    depends_on:
      netbox:
        condition: service_healthy
    command:
      - /opt/netbox/housekeeping.sh
    healthcheck:
      test: ps -aux | grep -v grep | grep -q housekeeping || exit 1
      start_period: 20s
      timeout: 3s
      interval: 15s

  # postgres
  postgres:
    image: docker.io/postgres:17-alpine
    healthcheck:
      test: pg_isready -q -t 2 -d $$POSTGRES_DB -U $$POSTGRES_USER
      start_period: 20s
      timeout: 30s
      interval: 10s
      retries: 5
```

```
    env_file: env/postgres.env
    volumes:
      - netbox-postgres-data:/var/lib/postgresql/data

  # redis
  redis:
    image: docker.io/valkey/valkey:8.0-alpine
    command:
      - sh
      - -c # this is to evaluate the $REDIS_PASSWORD from the env
      - valkey-server --appendonly yes --requirepass $$REDIS_PASSWORD ## $$ because of
docker-compose
    healthcheck: &redis-healthcheck
      test: '[ $$(valkey-cli --pass "$${REDIS_PASSWORD}" ping) = ''PONG'' ]'
      start_period: 5s
      timeout: 3s
      interval: 1s
      retries: 5
    env_file: env/redis.env
    volumes:
      - netbox-redis-data:/data
  redis-cache:
    image: docker.io/valkey/valkey:8.0-alpine
    command:
      - sh
      - -c # this is to evaluate the $REDIS_PASSWORD from the env
      - valkey-server --requirepass $$REDIS_PASSWORD ## $$ because of docker-compose
    healthcheck: *redis-healthcheck
    env_file: env/redis-cache.env
    volumes:
      - netbox-redis-cache-data:/data

volumes:
  netbox-media-files:
    driver: local
  netbox-postgres-data:
    driver: local
  netbox-redis-cache-data:
    driver: local
  netbox-redis-data:
    driver: local
  netbox-reports-files:
    driver: local
  netbox-scripts-files:
    driver: local
```

Netbox\docker-compose.override.yml

```
services:
  netbox:
    image: netbox:plugins
    ports:
      - "8000:8080"  # Change the external port
    build:
      context: .
      dockerfile: Dockerfile
```

```
  netbox-worker:
    image: netbox:plugins
    pull_policy: never
  netbox-housekeeping:
    image: netbox:plugins
    pull_policy: never
```

Netbox\Dockerfile

```
FROM netboxcommunity/netbox:latest

COPY ./plugin_requirements.txt /opt/netbox/
RUN /usr/local/bin/uv pip install -r /opt/netbox/plugin_requirements.txt

# These lines are only required if your plugin has its own static files.
COPY configuration/configuration.py /etc/netbox/config/configuration.py
COPY configuration/plugins.py /etc/netbox/config/plugins.py
RUN DEBUG="true" SECRET_KEY="dummydummydummydummydummydummydummydummydummydummy" \
    /opt/netbox/venv/bin/python /opt/netbox/netbox/manage.py collectstatic --no-input
```

Netbox\plugin_requirements.txt

```
netbox-topology-views
netbox-reorder-rack
netbox-attachments
```

Netbox\configuration\configuration.py

```
####
## We recommend to not edit this file.
## Create separate files to overwrite the settings.
## See `extra.py` as an example.
####

import re
from os import environ
from os.path import abspath, dirname, join
from typing import Any, Callable, Tuple

# For reference see https://docs.netbox.dev/en/stable/configuration/
# Based on https://github.com/netbox-
community/netbox/blob/develop/netbox/netbox/configuration_example.py

###
# NetBox-Docker Helper functions
###

# Read secret from file
def _read_secret(secret_name: str, default: str | None = None) -> str | None:
    try:
        f = open('/run/secrets/' + secret_name, 'r', encoding='utf-8')
    except EnvironmentError:
        return default
```

```
    else:
        with f:
            return f.readline().strip()

# If the `map_fn` isn't defined, then the value that is read from the environment (or the
default value if not found) is returned.
# If the `map_fn` is defined, then `map_fn` is invoked and the value (that was read from
the environment or the default value if not found)
# is passed to it as a parameter. The value returned from `map_fn` is then the return value
of this function.
# The `map_fn` is not invoked, if the value (that was read from the environment or the
default value if not found) is None.
def _environ_get_and_map(variable_name: str, default: str | None = None, map_fn:
Callable[[str], Any | None] = None) -> Any | None:
    env_value = environ.get(variable_name, default)

    if env_value == None:
        return env_value

    if not map_fn:
        return env_value

    return map_fn(env_value)

_AS_BOOL = lambda value : value.lower() == 'true'
_AS_INT = lambda value : int(value)
_AS_LIST = lambda value : list(filter(None, value.split(' ')))

_BASE_DIR = dirname(dirname(abspath(__file__)))

#########################
#                       #
#   Required settings   #
#                       #
#########################

# This is a list of valid fully-qualified domain names (FQDNs) for the NetBox server.
NetBox will not permit write
# access to the server via any other hostnames. The first FQDN in the list will be treated
as the preferred name.
#
# Example: ALLOWED_HOSTS = ['netbox.example.com', 'netbox.internal.local']
ALLOWED_HOSTS = environ.get('ALLOWED_HOSTS', '*').split(' ')
# ensure that '*' or 'localhost' is always in ALLOWED_HOSTS (needed for health checks)
if '*' not in ALLOWED_HOSTS and 'localhost' not in ALLOWED_HOSTS:
    ALLOWED_HOSTS.append('localhost')

# PostgreSQL database configuration. See the Django documentation for a complete list of
available parameters:
#   https://docs.djangoproject.com/en/stable/ref/settings/#databases
DATABASE = {
    'NAME': environ.get('DB_NAME', 'netbox'),       # Database name
    'USER': environ.get('DB_USER', ''),             # PostgreSQL username
    'PASSWORD': _read_secret('db_password', environ.get('DB_PASSWORD', '')),
```

```
                                                # PostgreSQL password
    'HOST': environ.get('DB_HOST', 'localhost'),    # Database server
    'PORT': environ.get('DB_PORT', ''),             # Database port (leave blank for
default)
    'OPTIONS': {'sslmode': environ.get('DB_SSLMODE', 'prefer')},
                                                # Database connection SSLMODE
    'CONN_MAX_AGE': _environ_get_and_map('DB_CONN_MAX_AGE', '300', _AS_INT),
                                                # Max database connection age
    'DISABLE_SERVER_SIDE_CURSORS': _environ_get_and_map('DB_DISABLE_SERVER_SIDE_CURSORS',
'False', _AS_BOOL),
                                                # Disable the use of server-side
cursors transaction pooling
}

# Redis database settings. Redis is used for caching and for queuing background tasks such
as webhook events. A separate
# configuration exists for each. Full connection details are required in both sections, and
it is strongly recommended
# to use two separate database IDs.
REDIS = {
    'tasks': {
        'HOST': environ.get('REDIS_HOST', 'localhost'),
        'PORT': _environ_get_and_map('REDIS_PORT', 6379, _AS_INT),
        'SENTINELS': [tuple(uri.split(':')) for uri in
_environ_get_and_map('REDIS_SENTINELS', '', _AS_LIST) if uri != ''],
        'SENTINEL_SERVICE': environ.get('REDIS_SENTINEL_SERVICE', 'default'),
        'SENTINEL_TIMEOUT': _environ_get_and_map('REDIS_SENTINEL_TIMEOUT', 10, _AS_INT),
        'USERNAME': environ.get('REDIS_USERNAME', ''),
        'PASSWORD': _read_secret('redis_password', environ.get('REDIS_PASSWORD', '')),
        'DATABASE': _environ_get_and_map('REDIS_DATABASE', 0, _AS_INT),
        'SSL': _environ_get_and_map('REDIS_SSL', 'False', _AS_BOOL),
        'INSECURE_SKIP_TLS_VERIFY': _environ_get_and_map('REDIS_INSECURE_SKIP_TLS_VERIFY',
'False', _AS_BOOL),
    },
    'caching': {
        'HOST': environ.get('REDIS_CACHE_HOST', environ.get('REDIS_HOST', 'localhost')),
        'PORT': _environ_get_and_map('REDIS_CACHE_PORT', environ.get('REDIS_PORT', '6379'),
_AS_INT),
        'SENTINELS': [tuple(uri.split(':')) for uri in
_environ_get_and_map('REDIS_CACHE_SENTINELS', '', _AS_LIST) if uri != ''],
        'SENTINEL_SERVICE': environ.get('REDIS_CACHE_SENTINEL_SERVICE',
environ.get('REDIS_SENTINEL_SERVICE', 'default')),
        'USERNAME': environ.get('REDIS_CACHE_USERNAME', environ.get('REDIS_USERNAME', '')),
        'PASSWORD': _read_secret('redis_cache_password',
environ.get('REDIS_CACHE_PASSWORD', environ.get('REDIS_PASSWORD', ''))),
        'DATABASE': _environ_get_and_map('REDIS_CACHE_DATABASE', '1', _AS_INT),
        'SSL': _environ_get_and_map('REDIS_CACHE_SSL', environ.get('REDIS_SSL', 'False'),
_AS_BOOL),
        'INSECURE_SKIP_TLS_VERIFY':
_environ_get_and_map('REDIS_CACHE_INSECURE_SKIP_TLS_VERIFY',
environ.get('REDIS_INSECURE_SKIP_TLS_VERIFY', 'False'), _AS_BOOL),
    },
}
```

```
# This key is used for secure generation of random numbers and strings. It must never be
exposed outside of this file.
# For optimal security, SECRET_KEY should be at least 50 characters in length and contain a
mix of letters, numbers, and
# symbols. NetBox will not run without this defined. For more information, see
# https://docs.djangoproject.com/en/stable/ref/settings/#std:setting-SECRET_KEY
SECRET_KEY = _read_secret('secret_key', environ.get('SECRET_KEY', ''))


#########################
#                       #
#   Optional settings   #
#                       #
#########################

# # Specify one or more name and email address tuples representing NetBox administrators.
These people will be notified of
# # application errors (assuming correct email settings are provided).
# ADMINS = [
#    # ['John Doe', 'jdoe@example.com'],
# ]

if 'ALLOWED_URL_SCHEMES' in environ:
    ALLOWED_URL_SCHEMES = _environ_get_and_map('ALLOWED_URL_SCHEMES', None, _AS_LIST)

# Optionally display a persistent banner at the top and/or bottom of every page. HTML is
allowed. To display the same
# content in both banners, define BANNER_TOP and set BANNER_BOTTOM = BANNER_TOP.
if 'BANNER_TOP' in environ:
    BANNER_TOP = environ.get('BANNER_TOP', None)
if 'BANNER_BOTTOM' in environ:
    BANNER_BOTTOM = environ.get('BANNER_BOTTOM', None)

# Text to include on the login page above the login form. HTML is allowed.
if 'BANNER_LOGIN' in environ:
    BANNER_LOGIN = environ.get('BANNER_LOGIN', None)

# Maximum number of days to retain logged changes. Set to 0 to retain changes indefinitely.
(Default: 90)
if 'CHANGELOG_RETENTION' in environ:
    CHANGELOG_RETENTION = _environ_get_and_map('CHANGELOG_RETENTION', None, _AS_INT)

# Maximum number of days to retain job results (scripts and reports). Set to 0 to retain
job results in the database indefinitely. (Default: 90)
if 'JOB_RETENTION' in environ:
    JOB_RETENTION = _environ_get_and_map('JOB_RETENTION', None, _AS_INT)
# JOBRESULT_RETENTION was renamed to JOB_RETENTION in the v3.5.0 release of NetBox. For
backwards compatibility, map JOBRESULT_RETENTION to JOB_RETENTION
elif 'JOBRESULT_RETENTION' in environ:
    JOB_RETENTION = _environ_get_and_map('JOBRESULT_RETENTION', None, _AS_INT)

# API Cross-Origin Resource Sharing (CORS) settings. If CORS_ORIGIN_ALLOW_ALL is set to
True, all origins will be
# allowed. Otherwise, define a list of allowed origins using either CORS_ORIGIN_WHITELIST
                                                                                    or
```

```
# CORS_ORIGIN_REGEX_WHITELIST. For more information, see https://github.com/ottoyiu/django-cors-headers
CORS_ORIGIN_ALLOW_ALL = _environ_get_and_map('CORS_ORIGIN_ALLOW_ALL', 'False', _AS_BOOL)
CORS_ORIGIN_WHITELIST = _environ_get_and_map('CORS_ORIGIN_WHITELIST', 'https://localhost', _AS_LIST)
CORS_ORIGIN_REGEX_WHITELIST = [re.compile(r) for r in _environ_get_and_map('CORS_ORIGIN_REGEX_WHITELIST', '', _AS_LIST)]

# Set to True to enable server debugging. WARNING: Debugging introduces a substantial performance penalty and may reveal
# sensitive information about your installation. Only enable debugging while performing testing.
# Never enable debugging on a production system.
DEBUG = _environ_get_and_map('DEBUG', 'False', _AS_BOOL)

# This parameter serves as a safeguard to prevent some potentially dangerous behavior,
# such as generating new database schema migrations.
# Set this to True only if you are actively developing the NetBox code base.
DEVELOPER = _environ_get_and_map('DEVELOPER', 'False', _AS_BOOL)

# Email settings
EMAIL = {
    'SERVER': environ.get('EMAIL_SERVER', 'localhost'),
    'PORT': _environ_get_and_map('EMAIL_PORT', 25, _AS_INT),
    'USERNAME': environ.get('EMAIL_USERNAME', ''),
    'PASSWORD': _read_secret('email_password', environ.get('EMAIL_PASSWORD', '')),
    'USE_SSL': _environ_get_and_map('EMAIL_USE_SSL', 'False', _AS_BOOL),
    'USE_TLS': _environ_get_and_map('EMAIL_USE_TLS', 'False', _AS_BOOL),
    'SSL_CERTFILE': environ.get('EMAIL_SSL_CERTFILE', ''),
    'SSL_KEYFILE': environ.get('EMAIL_SSL_KEYFILE', ''),
    'TIMEOUT': _environ_get_and_map('EMAIL_TIMEOUT', 10, _AS_INT),  # seconds
    'FROM_EMAIL': environ.get('EMAIL_FROM', ''),
}

# Enforcement of unique IP space can be toggled on a per-VRF basis. To enforce unique IP space within the global table
# (all prefixes and IP addresses not assigned to a VRF), set ENFORCE_GLOBAL_UNIQUE to True.
if 'ENFORCE_GLOBAL_UNIQUE' in environ:
    ENFORCE_GLOBAL_UNIQUE = _environ_get_and_map('ENFORCE_GLOBAL_UNIQUE', None, _AS_BOOL)

# By default, netbox sends census reporting data using a single HTTP request each time a worker starts.
# This data enables the project maintainers to estimate how many NetBox deployments exist and track the adoption of new versions over time.
# The only data reported by this function are the NetBox version, Python version, and a pseudorandom unique identifier.
# To opt out of census reporting, set CENSUS_REPORTING_ENABLED to False.
if 'CENSUS_REPORTING_ENABLED' in environ:
    CENSUS_REPORTING_ENABLED = _environ_get_and_map('CENSUS_REPORTING_ENABLED', None, _AS_BOOL)

# Exempt certain models from the enforcement of view permissions. Models listed here will be viewable by all users and
```

```
# by anonymous users. List models in the form `<app>.<model>`. Add '*' to this list to
exempt all models.
EXEMPT_VIEW_PERMISSIONS = _environ_get_and_map('EXEMPT_VIEW_PERMISSIONS', '', _AS_LIST)

# HTTP proxies NetBox should use when sending outbound HTTP requests (e.g. for webhooks).
HTTP_PROXIES = {
    'http': environ.get('HTTP_PROXY', None),
    'https': environ.get('HTTPS_PROXY', None),
}

# IP addresses recognized as internal to the system. The debugging toolbar will be
available only to clients accessing
# NetBox from an internal IP.
INTERNAL_IPS = _environ_get_and_map('INTERNAL_IPS', '127.0.0.1 ::1', _AS_LIST)

# Enable GraphQL API.
if 'GRAPHQL_ENABLED' in environ:
    GRAPHQL_ENABLED = _environ_get_and_map('GRAPHQL_ENABLED', None, _AS_BOOL)

# # Enable custom logging. Please see the Django documentation for detailed guidance on
configuring custom logs:
# #   https://docs.djangoproject.com/en/stable/topics/logging/
# LOGGING = {}

# Automatically reset the lifetime of a valid session upon each authenticated request.
Enables users to remain
# authenticated to NetBox indefinitely.
LOGIN_PERSISTENCE = _environ_get_and_map('LOGIN_PERSISTENCE', 'False', _AS_BOOL)

# When enabled, only authenticated users are permitted to access any part of NetBox.
# Disabling this will allow unauthenticated users to access most areas of NetBox (but not
make any changes).
LOGIN_REQUIRED = _environ_get_and_map('LOGIN_REQUIRED', 'True', _AS_BOOL)

# The length of time (in seconds) for which a user will remain logged into the web UI
before being prompted to
# re-authenticate. (Default: 1209600 [14 days])
LOGIN_TIMEOUT = _environ_get_and_map('LOGIN_TIMEOUT', 1209600, _AS_INT)

# Setting this to True will display a "maintenance mode" banner at the top of every page.
if 'MAINTENANCE_MODE' in environ:
    MAINTENANCE_MODE = _environ_get_and_map('MAINTENANCE_MODE', None, _AS_BOOL)

# Maps provider
if 'MAPS_URL' in environ:
    MAPS_URL = environ.get('MAPS_URL', None)

# An API consumer can request an arbitrary number of objects =by appending the "limit"
parameter to the URL (e.g.
# "?limit=1000"). This setting defines the maximum limit. Setting it to 0 or None will
allow an API consumer to request
# all objects by specifying "?limit=0".
if 'MAX_PAGE_SIZE' in environ:
    MAX_PAGE_SIZE = _environ_get_and_map('MAX_PAGE_SIZE', None, _AS_INT)
```

```
# The file path where uploaded media such as image attachments are stored. A trailing slash
is not needed. Note that
# the default value of this setting is derived from the installed location.
MEDIA_ROOT = environ.get('MEDIA_ROOT', join(_BASE_DIR, 'media'))

# Expose Prometheus monitoring metrics at the HTTP endpoint '/metrics'
METRICS_ENABLED = _environ_get_and_map('METRICS_ENABLED', 'False', _AS_BOOL)

# Determine how many objects to display per page within a list. (Default: 50)
if 'PAGINATE_COUNT' in environ:
    PAGINATE_COUNT = _environ_get_and_map('PAGINATE_COUNT', None, _AS_INT)

# # Enable installed plugins. Add the name of each plugin to the list.
# PLUGINS = []

# # Plugins configuration settings. These settings are used by various plugins that the
user may have installed.
# # Each key in the dictionary is the name of an installed plugin and its value is a
dictionary of settings.
# PLUGINS_CONFIG = {
# }

# When determining the primary IP address for a device, IPv6 is preferred over IPv4 by
default. Set this to True to
# prefer IPv4 instead.
if 'PREFER_IPV4' in environ:
    PREFER_IPV4 = _environ_get_and_map('PREFER_IPV4', None, _AS_BOOL)

# The default value for the amperage field when creating new power feeds.
if 'POWERFEED_DEFAULT_AMPERAGE' in environ:
    POWERFEED_DEFAULT_AMPERAGE = _environ_get_and_map('POWERFEED_DEFAULT_AMPERAGE', None,
_AS_INT)

# The default value (percentage) for the max_utilization field when creating new power
feeds.
if 'POWERFEED_DEFAULT_MAX_UTILIZATION' in environ:
    POWERFEED_DEFAULT_MAX_UTILIZATION =
_environ_get_and_map('POWERFEED_DEFAULT_MAX_UTILIZATION', None, _AS_INT)

# The default value for the voltage field when creating new power feeds.
if 'POWERFEED_DEFAULT_VOLTAGE' in environ:
    POWERFEED_DEFAULT_VOLTAGE = _environ_get_and_map('POWERFEED_DEFAULT_VOLTAGE', None,
_AS_INT)

# Rack elevation size defaults, in pixels. For best results, the ratio of width to height
should be roughly 10:1.
if 'RACK_ELEVATION_DEFAULT_UNIT_HEIGHT' in environ:
    RACK_ELEVATION_DEFAULT_UNIT_HEIGHT =
_environ_get_and_map('RACK_ELEVATION_DEFAULT_UNIT_HEIGHT', None, _AS_INT)
if 'RACK_ELEVATION_DEFAULT_UNIT_WIDTH' in environ:
    RACK_ELEVATION_DEFAULT_UNIT_WIDTH =
_environ_get_and_map('RACK_ELEVATION_DEFAULT_UNIT_WIDTH', None, _AS_INT)

# Remote authentication support
```

```
REMOTE_AUTH_AUTO_CREATE_GROUPS = _environ_get_and_map('REMOTE_AUTH_AUTO_CREATE_GROUPS',
'False', _AS_BOOL)
REMOTE_AUTH_AUTO_CREATE_USER = _environ_get_and_map('REMOTE_AUTH_AUTO_CREATE_USER',
'False', _AS_BOOL)
REMOTE_AUTH_BACKEND = _environ_get_and_map('REMOTE_AUTH_BACKEND',
'netbox.authentication.RemoteUserBackend', _AS_LIST)
REMOTE_AUTH_DEFAULT_GROUPS = _environ_get_and_map('REMOTE_AUTH_DEFAULT_GROUPS', '',
_AS_LIST)
# REMOTE_AUTH_DEFAULT_PERMISSIONS = {} # dicts can't be configured via environment
variables. See extra.py instead.
REMOTE_AUTH_ENABLED = _environ_get_and_map('REMOTE_AUTH_ENABLED', 'False', _AS_BOOL)
REMOTE_AUTH_GROUP_HEADER = _environ_get_and_map('REMOTE_AUTH_GROUP_HEADER',
'HTTP_REMOTE_USER_GROUP')
REMOTE_AUTH_GROUP_SEPARATOR = _environ_get_and_map('REMOTE_AUTH_GROUP_SEPARATOR', '|')
REMOTE_AUTH_GROUP_SYNC_ENABLED = _environ_get_and_map('REMOTE_AUTH_GROUP_SYNC_ENABLED',
'False', _AS_BOOL)
REMOTE_AUTH_HEADER = environ.get('REMOTE_AUTH_HEADER', 'HTTP_REMOTE_USER')
REMOTE_AUTH_USER_EMAIL = environ.get('REMOTE_AUTH_USER_EMAIL', 'HTTP_REMOTE_USER_EMAIL')
REMOTE_AUTH_USER_FIRST_NAME = environ.get('REMOTE_AUTH_USER_FIRST_NAME',
'HTTP_REMOTE_USER_FIRST_NAME')
REMOTE_AUTH_USER_LAST_NAME = environ.get('REMOTE_AUTH_USER_LAST_NAME',
'HTTP_REMOTE_USER_LAST_NAME')
REMOTE_AUTH_SUPERUSER_GROUPS = _environ_get_and_map('REMOTE_AUTH_SUPERUSER_GROUPS', '',
_AS_LIST)
REMOTE_AUTH_SUPERUSERS = _environ_get_and_map('REMOTE_AUTH_SUPERUSERS', '', _AS_LIST)
REMOTE_AUTH_STAFF_GROUPS = _environ_get_and_map('REMOTE_AUTH_STAFF_GROUPS', '', _AS_LIST)
REMOTE_AUTH_STAFF_USERS = _environ_get_and_map('REMOTE_AUTH_STAFF_USERS', '', _AS_LIST)

# This repository is used to check whether there is a new release of NetBox available. Set
to None to disable the
# version check or use the URL below to check for release in the official NetBox
repository.
RELEASE_CHECK_URL = environ.get('RELEASE_CHECK_URL', None)
# RELEASE_CHECK_URL = 'https://api.github.com/repos/netbox-community/netbox/releases'

# Maximum execution time for background tasks, in seconds.
RQ_DEFAULT_TIMEOUT = _environ_get_and_map('RQ_DEFAULT_TIMEOUT', 300, _AS_INT)

# The name to use for the csrf token cookie.
CSRF_COOKIE_NAME = environ.get('CSRF_COOKIE_NAME', 'csrftoken')

# Cross-Site-Request-Forgery-Attack settings. If Netbox is sitting behind a reverse proxy,
you might need to set the CSRF_TRUSTED_ORIGINS flag.
# Django 4.0 requires to specify the URL Scheme in this setting. An example environment
variable could be specified like:
# CSRF_TRUSTED_ORIGINS=https://demo.netbox.dev http://demo.netbox.dev
CSRF_TRUSTED_ORIGINS = _environ_get_and_map('CSRF_TRUSTED_ORIGINS', '', _AS_LIST)

# The name to use for the session cookie.
SESSION_COOKIE_NAME = environ.get('SESSION_COOKIE_NAME', 'sessionid')

# If true, the `includeSubDomains` directive will be included in the HTTP Strict Transport
Security (HSTS) header.
```

```
# This directive instructs the browser to apply the HSTS policy to all subdomains of the
current domain.
SECURE_HSTS_INCLUDE_SUBDOMAINS = _environ_get_and_map('SECURE_HSTS_INCLUDE_SUBDOMAINS',
'False', _AS_BOOL)

# If true, the `preload` directive will be included in the HTTP Strict Transport Security
(HSTS) header.
# This directive instructs the browser to preload the site in HTTPS. Browsers that use the
HSTS preload list will force the
# site to be accessed via HTTPS even if the user types HTTP in the address bar.
SECURE_HSTS_PRELOAD = _environ_get_and_map('SECURE_HSTS_PRELOAD', 'False', _AS_BOOL)

# If set to a non-zero integer value, the SecurityMiddleware sets the HTTP Strict Transport
Security (HSTS) header on all
# responses that do not already have it. This will instruct the browser that the website
must be accessed via HTTPS,
# blocking any HTTP request.
SECURE_HSTS_SECONDS = _environ_get_and_map('SECURE_HSTS_SECONDS', 0, _AS_INT)

# If true, all non-HTTPS requests will be automatically redirected to use HTTPS.
SECURE_SSL_REDIRECT = _environ_get_and_map('SECURE_SSL_REDIRECT', 'False', _AS_BOOL)

# By default, NetBox will store session data in the database. Alternatively, a file path
can be specified here to use
# local file storage instead. (This can be useful for enabling authentication on a standby
instance with read-only
# database access.) Note that the user as which NetBox runs must have read and write
permissions to this path.
SESSION_FILE_PATH = environ.get('SESSION_FILE_PATH', environ.get('SESSIONS_ROOT', None))

# Time zone (default: UTC)
TIME_ZONE = environ.get('TIME_ZONE', 'UTC')

# If true disables miscellaneous functionality which depends on access to the Internet.
ISOLATED_DEPLOYMENT = _environ_get_and_map('ISOLATED_DEPLOYMENT', 'False', _AS_BOOL)
```

Netbox\configuration\plugins.py

```
# Add your plugins and plugin settings here.
# Of course uncomment this file out.

# To learn how to build images with your required plugins
# See https://github.com/netbox-community/netbox-docker/wiki/Using-Netbox-Plugins

# PLUGINS = ["netbox_bgp"]

# PLUGINS_CONFIG = {
#   "netbox_bgp": {
#     ADD YOUR SETTINGS HERE
#   }
# }
PLUGINS = [
    "netbox_topology_views",
```

```
    "netbox_reorder_rack",
    "netbox_attachments",
]

PLUGINS_CONFIG = {
    'netbox_topology_views': {
        'static_image_directory': 'netbox_topology_views/img',
        'allow_coordinates_saving': True,
        'always_save_coordinates': True
    },
    "netbox_reorder_rack": {},
    "netbox_attachments": {},
}
```

Netbox\env\netbox.env

```
CORS_ORIGIN_ALLOW_ALL=True
DB_HOST=postgres
DB_NAME=netbox
DB_PASSWORD=PASS
DB_USER=netbox
EMAIL_FROM=netbox@bar.com
EMAIL_PASSWORD=
EMAIL_PORT=25
EMAIL_SERVER=localhost
EMAIL_SSL_CERTFILE=
EMAIL_SSL_KEYFILE=
EMAIL_TIMEOUT=5
EMAIL_USERNAME=netbox
# EMAIL_USE_SSL and EMAIL_USE_TLS are mutually exclusive, i.e. they can't both be `true`!
EMAIL_USE_SSL=false
EMAIL_USE_TLS=false
GRAPHQL_ENABLED=true
HOUSEKEEPING_INTERVAL=86400
MEDIA_ROOT=/opt/netbox/netbox/media
METRICS_ENABLED=false
REDIS_CACHE_DATABASE=1
REDIS_CACHE_HOST=redis-cache
REDIS_CACHE_INSECURE_SKIP_TLS_VERIFY=false
REDIS_CACHE_PASSWORD=PASS
REDIS_CACHE_SSL=false
REDIS_DATABASE=0
REDIS_HOST=redis
REDIS_INSECURE_SKIP_TLS_VERIFY=false
REDIS_PASSWORD=PASS
REDIS_SSL=false
RELEASE_CHECK_URL=https://api.github.com/repos/netbox-community/netbox/releases
SECRET_KEY=KEY'
SKIP_SUPERUSER=true
WEBHOOKS_ENABLED=true
```

Netbox\env\postgres.env

```
POSTGRES_DB=netbox
POSTGRES_PASSWORD=PASS
POSTGRES_USER=netbox
```

Netbox\env\redis-cache.env

```
REDIS_PASSWORD=PASS
```

Netbox\env\redis.env

```
REDIS_PASSWORD=PASS
```

Semaphore\docker-compose.yml

```
services:
  postgres:
    restart: unless-stopped
    image: postgres:17
    hostname: postgres
    volumes:
     - ./data/semaphore-postgres:/var/lib/postgresql/data
    environment:
     POSTGRES_USER: semaphore
     POSTGRES_PASSWORD: semaphore
     POSTGRES_DB: semaphore
  semaphore:
    restart: unless-stopped
    ports:
      - 3000:3000
    image: semaphoreui/semaphore:latest
    build:
      context: .
      dockerfile: Dockerfile
    environment:
      SEMAPHORE_DB_USER: semaphore
      SEMAPHORE_DB_PASS: semaphore
      SEMAPHORE_DB_HOST: postgres # for postgres, change to: postgres
      SEMAPHORE_DB_PORT: 5432 # change to 5432 for postgres
      SEMAPHORE_DB_DIALECT: postgres # for postgres, change to: postgres
      SEMAPHORE_DB: semaphore
      SEMAPHORE_PLAYBOOK_PATH: /tmp/semaphore/
      SEMAPHORE_ADMIN_PASSWORD: 'admin'
      SEMAPHORE_ADMIN_NAME: admin
      SEMAPHORE_ADMIN_EMAIL: admin@localhost
      SEMAPHORE_ADMIN: admin
      SEMAPHORE_ACCESS_KEY_ENCRYPTION: gs72mPntFATGJs9qK0pQ0rKtfidlexiMjYCH9gWKhTU=
      SEMAPHORE_LDAP_ACTIVATED: 'no' # if you wish to use ldap, set to: 'yes'
      SEMAPHORE_LDAP_HOST: dc01.local.example.com
      SEMAPHORE_LDAP_PORT: '636'
      SEMAPHORE_LDAP_NEEDTLS: 'yes'
      SEMAPHORE_LDAP_DN_BIND:
'uid=bind_user,cn=users,cn=accounts,dc=local,dc=shiftsystems,dc=net'
      SEMAPHORE_LDAP_PASSWORD: 'ldap_bind_account_password'
      SEMAPHORE_LDAP_DN_SEARCH: 'dc=local,dc=example,dc=com'
```

```
      SEMAPHORE_LDAP_SEARCH_FILTER:
"(\u0026(uid=%s)(memberOf=cn=ipausers,cn=groups,cn=accounts,dc=local,dc=example,dc=com))"
      TZ: UTC
    depends_on:
      - postgres # for postgres, change to: postgres
    volumes:
      - ./requirements.txt:/etc/semaphore/requirements.txt
      - ./data/playbooks:/playbooks
```

Semaphore\requirements.txt

```
paramiko
pytz
```

8.5.2.4 Spustenie

V koreňovom priečinku **netbox/**:

```
docker compose build
docker compose up -d
docker compose ps
docker compose logs -f netbox
```

Po štarte je web rozhranie dostupné na URL (podľa override portu):

```
http://<IP_alebo_hostname_hosta>:8000/
```

Keďže v PoC je nastavené SKIP_SUPERUSER=true, prvý administrátorský účet sa vytvorí explicitne:

```
docker compose exec netbox /opt/netbox/venv/bin/python /opt/netbox/netbox/manage.py
createsuperuser
```

Overenie funkčnosti:

- v prehliadači sa načíta prihlasovacia stránka,
- docker compose ps zobrazuje služby ako healthy (najmä netbox).

8.5.3 Inštalácia – variant Openstack

V tomto variante je NetBox nasadený ako virtuálny server (VM) v prostredí IaaS cloudovej infraštruktúry OpenStack. OpenStack zabezpečuje virtualizačnú vrstvu vrátane výpočtových zdrojov, úložiska a sieťovej konektivity, zatiaľ čo samotná aplikačná logika NetBoxu je prevádzkovaná kontajnerizovane pomocou Dockeru priamo vo vnútri virtuálneho stroja.

Virtuálny stroj slúži ako izolované aplikačné prostredie, v ktorom je nainštalovaný Docker engine a Docker Compose. Všetky komponenty NetBox stacku (aplikačná služba, databáza PostgreSQL, Redis vrstvy a pomocné procesy) sú následne nasadené identicky ako v Docker variante, čo zabezpečuje konzistentnosť konfigurácie naprieč prostrediami. Tento

prístup umožňuje jednoduchý presun riešenia medzi testovacím a produkčným prostredím bez potreby zásadných zmien v aplikačnej konfigurácii.

Nasadenie NetBoxu ako VM v OpenStacke je vhodné najmä v prostrediach, kde je cloudová infraštruktúra štandardom a kde je požadovaná integrácia s existujúcimi mechanizmami správy virtuálnych strojov, sieťovej segmentácie a bezpečnostných politík na úrovni IaaS.

8.5.4 Základná konfigurácia

8.5.4.1 Konfigurácia nástroja

Základná konfigurácia NetBoxu v rámci PoC bola zameraná na vytvorenie stabilného a konzistentného prostredia pre dokumentáciu aktív relevantných pre SOC a na prípravu systému pre ďalšie integračné a automatizačné scenáre. Konfigurácia je realizovaná kombináciou premenných prostredia a aplikačných nastavení definovaných v konfiguračných súboroch, čím je zabezpečená oddeliteľnosť konfigurácie od samotnej aplikácie.

Na aplikačnej úrovni je NetBox nakonfigurovaný na používanie databázového systému PostgreSQL ako perzistentného úložiska dát a dvoch samostatných Redis backendov – jedného pre spracovanie úloh na pozadí (tasks) a druhého pre aplikačnú cache. Toto oddelenie zodpovedá odporúčanej architektúre NetBoxu a zvyšuje stabilitu a predvídateľnosť správania systému aj pri vyššej záťaži.

V rámci PoC sú aktivované rozhrania a funkcie podporujúce integrácie, najmä:

- REST API ako primárne rozhranie pre prístup k dátam,
- GraphQL API pre flexibilnejšie dotazovanie nad objektovým modelom,
- Webhooks ako mechanizmus pre notifikácie o zmenách v dokumentovaných aktívach.

Z konfiguračného hľadiska je kladený dôraz na to, aby NetBox vystupoval ako autoritatívny zdroj pravdy (SSOT), pričom všetky zmeny v infraštruktúrnej dokumentácii sú vykonávané centrálne a následne sprístupňované ostatným systémom prostredníctvom API rozhraní.

8.5.4.2 Výstupy / Výstupy

Hlavným výstupom základnej konfigurácie NetBoxu je štruktúrovaná databáza aktív, ktorá obsahuje:

- evidenciu sieťových zariadení a ich rolí,
- fyzickú lokalizáciu (lokality, miestnosti, racky),
- IP adresné priestory, prefixy a priradenia IP adries,
- základné topologické vzťahy medzi zariadeniami.

Tieto výstupy slúžia ako referenčný kontext pre SOC činnosti, najmä pre mapovanie bezpečnostných udalostí na konkrétne aktíva, rýchlu identifikáciu dotknutých segmentov a podporu analýzy incidentov. NetBox v tomto modeli neposkytuje detekciu ani reakciu, ale vytvára dátový základ, bez ktorého by boli tieto procesy nejednoznačné alebo neefektívne.

8.5.4.3 Rozšírenie funkcionality NetBox pomocou pluginov (prečo a za čo)

V PoC nasadení je NetBox rozšírený o vybrané pluginy, aby bol systém použiteľnejší pre praktickú dokumentáciu aktív a ich prevádzkový kontext v SOC. Zámerom nie je “pridávať

funkcie navyše", ale doplniť chýbajúce prvky, ktoré zvyšujú kvalitu inventára a jeho využiteľnosť pre analýzu a prevádzku (topológia, fyzické usporiadanie, prílohy k aktívam).

Pluginy sú do nasadenia zaradené dvojkrokom:

1. Inštalácia balíkov do vlastného image cez plugin_requirements.txt (build podľa Dockerfile).
2. Aktivácia a konfigurácia v plugins.py (PLUGINS a PLUGINS_CONFIG).

V PoC sú použité tieto pluginy a ich prínos:

- **Topology Views (netbox-topology-views):** Slúži na vizualizáciu sieťovej topológie a prepojení. V PoC je zapnuté ukladanie súradníc (allow_coordinates_saving a always_save_coordinates), čo umožňuje udržiavať konzistentný layout topológie. Prínos pre SOC je najmä rýchlejšia orientácia pri incidente (vizuálne mapovanie dotknutého segmentu a väzieb medzi zariadeniami).
- **Reorder Rack (netbox-reorder-rack):** Umožňuje praktickú správu a vizuálne usporiadanie zariadení v racku (drag&drop). Prínosom je presnejšia fyzická dokumentácia, čo uľahčuje zásahy a eskaláciu pri fyzických incidentoch (napr. identifikácia zariadenia v rozvádzači).
- **Attachments (netbox-attachments)**: Umožňuje pripájať súbory k objektom (napr. schémy, konfigurácie, prevádzkové dokumenty, runbooky). Pre SOC je to prínosné tým, že "kontext aktíva" (návody, výnimky, diagramy) je uložený priamo pri objekte a je dostupný pri triage a reakcii.

8.5.5 Integrácie

8.5.5.1 API a integračné mechanizmy

NetBox je v rámci PoC integrovaný do prostredia SOC prostredníctvom štandardizovaných rozhraní, ktoré umožňujú jeho využitie ako backendového systému pre ďalšie nástroje a procesy.

Základným integračným mechanizmom je REST API, ktoré sprístupňuje všetky kľúčové objekty NetBoxu (zariadenia, rozhrania, IP adresy, lokality, vzťahy). Toto rozhranie je určené pre:

- automatizačné nástroje,
- skripty a integračné komponenty,
- externé systémy vyžadujúce aktuálny asset kontext.

Doplňujúcim rozhraním je GraphQL API, ktoré umožňuje efektívnejšie a cielenejšie dotazy nad dátovým modelom, čo je vhodné najmä pri komplexnejších integračných scenároch alebo analytických úlohách.

Pre udalosťami riadené integrácie sú aktivované Webhooks, ktoré umožňujú generovať notifikácie pri zmenách v dokumentovaných objektoch (napr. zmena IP adresácie, pridanie zariadenia). Tento mechanizmus vytvára základ pre budúce prepojenie NetBoxu s automatizovanými reakčnými alebo synchronizačnými procesmi.

Z pohľadu riadenia prístupov je NetBox pripravený na role-based access control (RBAC), ktorý umožňuje obmedziť prístup k objektom a funkciám podľa používateľských rolí. V PoC je RBAC chápaný ako nevyhnutný predpoklad pre bezpečné využitie dokumentačného systému v SOC prostredí, najmä pri zapojení viacerých tímov alebo používateľov s rôznymi oprávneniami.

Výsledkom tejto fázy PoC je funkčné nasadenie NetBoxu ako dokumentačného a integračného jadra pre SOC, pripraveného na napojenie na detekčné, monitorovacie a automatizačné nástroje. Integrácie sú v tejto fáze využívané najmä na čítanie dát a synchronizáciu kontextu, čo zodpovedá úlohe NetBoxu ako podporného, nie aktívneho bezpečnostného nástroja.

8.5.5.2 Integrácia Semaphore (PoC) – token → Semaphore → playbook → Git

1. Vygenerovanie API tokenu v NetBoxe
   V NetBox UI vytvor token pre používateľa (API Token) a skopíruj ho.
2. Vloženie tokenu do Semaphore
   V Semaphore UI vytvor:
3. Inventory/Environment alebo premennú (napr. NETBOX_TOKEN) a vlož tam token,
4. prípadne nastav aj URL NetBoxu (napr. NETBOX_URL=http://netbox:8080 alebo externú URL podľa siete).
5. Spustenie automatizačnej úlohy (playbook)
   V Semaphore nastav projekt tak, aby používal playbooky uložené v pripojenom repozitári/volume (./data/playbooks:/playbooks v compose). Následne spusti job (template), ktorý:
6. načíta inventory z NetBoxu (cez API),
7. vykoná požadovaný use-case (napr. backup konfigurácií).
8. Výstup do Git (verzionovanie)
   Výstup experimentu je uložený do Git repozitára (konfigurácie/artefakty), čím vzniká auditovateľná história zmien. V PoC tak Git slúži ako úložisko a verzovací mechanizmus výsledkov automatizačnej úlohy.

8.5.6 Integrácia NetBox so SOC komponentmi

8.5.6.1 Integrácia so SIEM (ELK Stack)

NetBox poskytuje SIEM nástrojom kontext k detegovaným udalostiam:

- mapovanie IP adresy na konkrétne zariadenie,
- identifikácia lokality, segmentu alebo VLAN,
- určenie roly zariadenia (core, access, firewall, server).

V praxi to znamená, že bezpečnostná udalosť z ELK Stacku môže byť korektne interpretovaná v kontexte infraštruktúry (*„ktoré zariadenie, kde, aký význam“*), čo výrazne znižuje nejednoznačnosť pri triage incidente.

8.5.6.2 Integrácia so SOAR (Shuffle)

NetBox je využívaný v enrichment fáze automatizovaných playbookov:

- Shuffle dotazuje NetBox API na informácie o IP adrese alebo zariadení,
- získaný kontext ovplyvňuje rozhodovaciu logiku playbooku (priorita, eskalácia, reakcia).

NetBox tak podporuje automatizáciu bez toho, aby sám vykonával akékoľvek aktívne zásahy.

8.5.6.3 Integrácia s CTI platformami (OpenCTI / MISP)

V kombinácii s CTI nástrojmi NetBox poskytuje **infraštruktúrny kontext hrozby**:

- CTI odpovedá na otázku *„čo je to za hrozbu“*,
- NetBox odpovedá na otázku *„kde sa to v infraštruktúre nachádza“*.

Táto kombinácia je kľúčová pre správne vyhodnotenie dopadu incidentu.

8.5.7 NetBox versus GLPI

NetBox a GLPI predstavujú dva rozdielne, no čiastočne sa prekrývajúce nástroje určené na evidenciu aktív a podporu prevádzkových procesov v IT prostredí. Obe riešenia sú schopné plniť úlohu centrálneho registra aktív v prostredí SOC, avšak každé z nich pristupuje k tejto úlohe z odlišného pohľadu a s iným primárnym zameraním.

Z funkčného hľadiska je možné GLPI považovať za procesne orientovanú alternatívu k NetBoxu, keďže:

- oba nástroje umožňujú evidenciu IT aktív,
- podporujú väzby aktív na vlastníkov a organizačný kontext,
- poskytujú API rozhrania pre integráciu s externými systémami,
- môžu slúžiť ako zdroj kontextu pre riešenie bezpečnostných incidentov v SOC.

V rámci projektu boli testované oba nástroje – NetBox aj GLPI – s cieľom overiť ich vhodnosť pre úlohu evidencie aktív v navrhovanej SOC architektúre. Testovanie potvrdilo, že obe riešenia sú použiteľné ako podporný systém pre SOC, avšak s rozdielnou mierou presnosti a technickej detailnosti poskytovaných dát.

Zároveň sa ukázalo, že NetBox a GLPI sa prirodzene dopĺňajú, pričom každý z nástrojov je silnejší v inej oblasti:

- NetBox je vhodnejší pre technickú a infraštruktúrnu dokumentáciu, detailné modelovanie siete, IP adresácie, topológie a vzťahov medzi zariadeniami. Poskytuje presný technický kontext, ktorý je kľúčový pre správnu interpretáciu bezpečnostných udalostí a podporu automatizácie.
- GLPI je vhodnejší pre procesnú a organizačnú evidenciu, správu incidentov, ticketing, CMDB a väzby aktív na služby, používateľov a zodpovednosti. Podporuje

auditovateľnosť a riadené spracovanie incidentov, avšak s menším dôrazom na detailnú sieťovú topológiu.

V kombinovanom nasadení môže NetBox plniť úlohu technického zdroja pravdy (SSOT) pre infraštruktúru, zatiaľ čo GLPI funguje ako procesná a evidenčná vrstva pre riadenie incidentov a IT služieb. Takýto model umožňuje rozdeliť zodpovednosti medzi nástroje bez zbytočnej duplicity dát a zvyšuje prehľadnosť SOC architektúry.

Z pohľadu návrhu SOC infraštruktúry je preto možné:

- použiť iba NetBox ako primárny zdroj technického kontextu (v technicky orientovanom SOC),
- použiť iba GLPI v prostrediach s dôrazom na ITSM procesy,
- alebo ich kombinovať, pričom NetBox zabezpečuje technickú presnosť a GLPI procesné riadenie incidentov.

Tento prístup umožňuje škálovať evidenciu aktív a procesné schopnosti SOC podľa potrieb organizácie, pričom NetBox poskytuje detailný infraštruktúrny pohľad a GLPI zabezpečuje formálnu správu a auditovateľnosť riešenia incidentov.

9 DFIR IRIS – TICKETING

Typ nástroja: Ticketing

Názov nástroja: DFIR IRIS

Verzia: verzia 2.4.20

9.1 Opis systému

DFIR-IRIS (Digital Forensics and Incident Response – Incident Response Investigation System) je open-source platforma určená na evidenciu, správu a dokumentáciu kybernetických bezpečnostných incidentov. Hoci sa nejedná o klasický helpdesk nástroj, z pohľadu SOC plní úlohu ticketingového systému tým, že umožňuje centralizovanú správu incidentov, úloh a rozhodnutí počas celého životného cyklu incidentu.

DFIR-IRIS je navrhnutý primárne pre potreby Incident Response a digitálnej forenznej analýzy. Základným objektom systému nie je tiket, ale incidentový prípad (Case), ktorý v sebe združuje všetky informácie súvisiace s riešením incidentu. Tento prístup eliminuje fragmentáciu dát typickú pre tradičné ticketingové systémy a poskytuje ucelený pohľad na incident v jednom rozhraní.

Účelom DFIR-IRIS v oblasti ticketingu je poskytnúť SOC tímu nástroj na systematickú evidenciu a riadenie bezpečnostných incidentov s dôrazom na technický kontext a dokumentáciu vyšetrovania. Nástroj je vhodný najmä pre SOC a CSIRT tímy, ktoré potrebujú prepojiť procesy ticketingu s technickou analýzou, prácou s IOC, dôkazmi a reportovaním incidentov.

Správa ticketov a úloh SOC (triage, eskalácie, SLA)

Ticketingová funkcionalita DFIR-IRIS je realizovaná prostredníctvom incidentových prípadov a úloh (Tasks). Každý incidentový prípad plní úlohu hlavného „ticketu“, v rámci ktorého prebieha triage, klasifikácia, eskalácia a riešenie incidentu. Systém umožňuje priradiť incident konkrétnym analytikom, definovať jeho závažnosť, prioritu a stav riešenia.

Úlohy v rámci prípadu slúžia na rozdelenie práce medzi členov SOC tímu a nahrádzajú klasické podtiketové mechanizmy. Každá úloha má vlastný stav, zodpovednú osobu a je zaznamenaná v časovej osi prípadu. Hoci DFIR-IRIS neimplementuje natívny SLA modul typický pre ITSM nástroje, podporuje sledovanie časových údajov a auditnú stopu, čo umožňuje vyhodnocovať dodržiavanie interných procesov a reakčných časov.

9.2 Dokumentácia systému

Oficialna dokumentácia:

https://docs.dfir-iris.org/latest/

Oficiálny GitHub projekt:

https://github.com/dfir-iris/iris-web

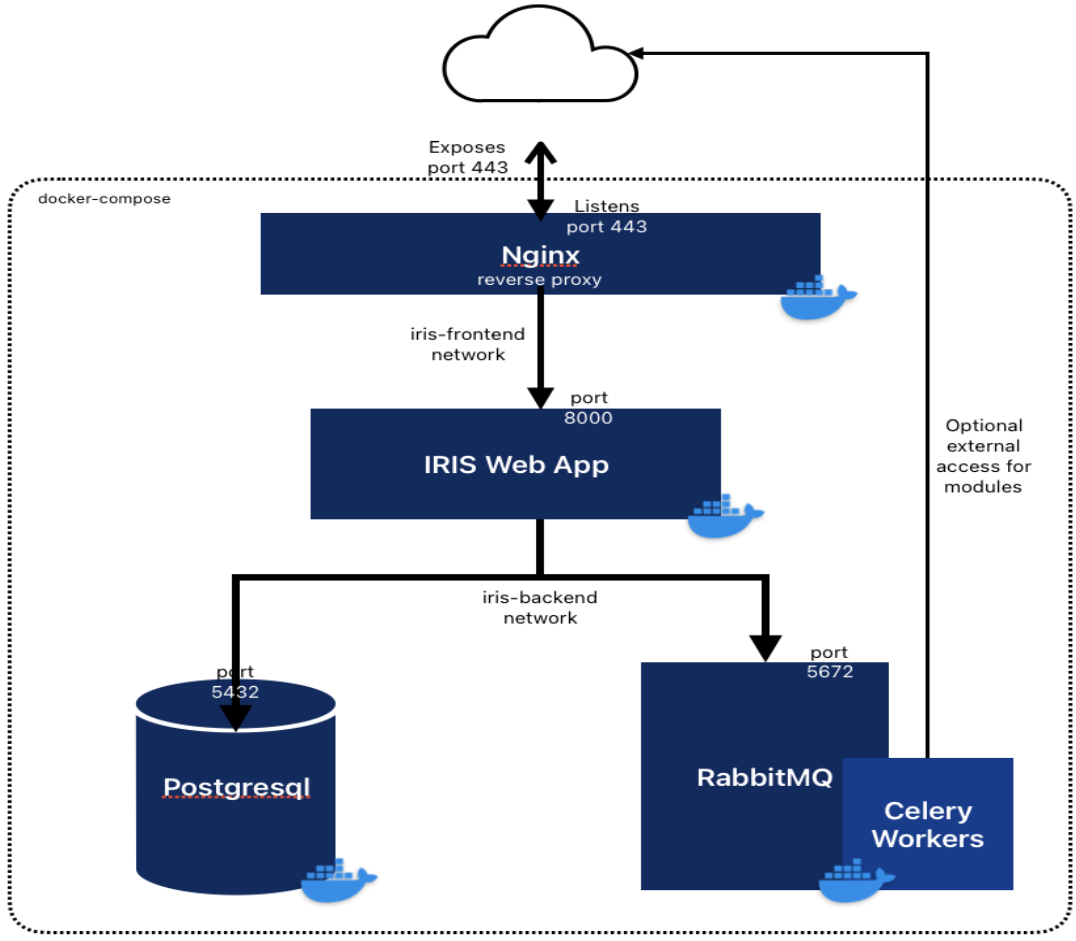
Hardvérové požiadavky

DFIR-IRIS má stredné hardvérové nároky. Pre testovacie alebo menšie produkčné nasadenie postačuje virtuálny server s minimálne 2–4 CPU jadrami, 8 GB RAM a približne 40 GB diskového priestoru. Pre väčšie SOC prostredia s viacerými analytikmi a dlhodobou archiváciou incidentov sa odporúča 4–8 CPU jadier, 16 GB RAM a rýchle SSD úložisko. Nároky rastú najmä pri práci s prílohami, dôkazmi a generovaní reportov.

Architektúra a schémy

Architektúra DFIR-IRIS je modulárna a pozostáva z webovej aplikácie, databázovej vrstvy a asynchrónnych spracovateľských komponentov. Webové rozhranie slúži ako hlavný bod interakcie používateľa so systémom. Backend komunikuje s databázou PostgreSQL, zatiaľ čo časovo náročné operácie sú spracúvané asynchrónne prostredníctvom frontu správ (RabbitMQ) a workerov. Súčasťou architektúry je REST API, ktoré umožňuje integráciu s ďalšími SOC komponentmi.

Schéma architektúry DFIR IRIS z oficiálnej webovej stránky



9.3 Inštalácia – variant Docker

1. Aktualizácia systému

```
sudo apt update && sudo apt upgrade -y
```

2. Inštalácia potrebných balíkov

```
sudo apt install -y ca-certificates curl gnupg lsb-release git
```

3. Pridanie Docker GPG kľúča

```
sudo mkdir -p /etc/apt/keyrings
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | \
sudo gpg --dearmor -o /etc/apt/keyrings/docker.gpg
```

4. Pridanie Docker repozitára

```
echo \
"deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.gpg] \
https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable" | \
sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
```

5. Inštalácia Docker Engine a Docker Compose

```
sudo apt update
sudo apt install -y docker-ce docker-ce-cli containerd.io docker-compose-plugin
```

6. Povolenie Docker služby

```
sudo systemctl enable docker
sudo systemctl start docker
```

7. Overenie Dockeru

```
docker --version
docker compose version
```

8. Stiahnutie DFIR-IRIS

```
cd /opt
git clone https://github.com/dfir-iris/iris-web.git
cd iris-web
```

9. Spustenie DFIR-IRIS

```
docker compose up -d
```

10. Overenie bežiacich kontajnerov

```
docker compose ps
```

11. Zistenie admin hesla (prvý login)

```
docker compose logs app | grep create_safe_admin
```

12. Prístup do webového rozhrania

```
https://<IP_adresa_servera>
```

9.4 Inštalácia – variant Openstack

1. Aktualizácia systému

```
sudo apt update && sudo apt upgrade -y
```

2. Inštalácia systémových závislostí

```
sudo apt install -y \
       git curl build-essential \
       python3 python3-venv python3-pip \
       postgresql postgresql-contrib \
       rabbitmq-server \
       nodejs npm \
       nginx
```

Overenie verzií:

```
python3 --version
       psql --version
       node --version
```

3. Nastavenie PostgreSQL databázy

Spustenie databázy

```
sudo systemctl enable postgresql
sudo systemctl start postgresql
```

Vytvorenie databázy a používateľa

```
sudo -u postgres psql

CREATE DATABASE iris;
CREATE USER irisuser WITH PASSWORD 'StrongPassword123';
GRANT ALL PRIVILEGES ON DATABASE iris TO irisuser;
ALTER DATABASE iris OWNER TO irisuser;
\q
```

4. Nastavenie RabbitMQ

```
sudo systemctl enable rabbitmq-server
sudo systemctl start rabbitmq-server
```

Vytvorenie používateľa:

```
sudo rabbitmqctl add_user iris irispassword
sudo rabbitmqctl add_vhost iris_vhost
sudo rabbitmqctl set_permissions -p iris_vhost iris ".*" ".*" ".*"
```

5. Stiahnutie DFIR-IRIS zdrojového kódu

```
cd /opt
       sudo git clone https://github.com/dfir-iris/iris-web.git
       sudo chown -R $USER:$USER iris-web
       cd iris-web
```

6. Backend – Python aplikácia

Vytvorenie virtuálneho prostredia

```
python3 -m venv venv
source venv/bin/activate
```

Inštalácia Python balíkov

```
pip install --upgrade pip
pip install -r requirements.txt
```

7. Nastavenie backend konfigurácie

Vytvorenie konfiguračných premenných:

```
export IRIS_ENV=production
export DATABASE_URL=postgresql://irisuser:StrongPassword123@localhost/iris
export RABBITMQ_URL=amqp://iris:irispassword@localhost:5672/iris_vhost
export SECRET_KEY=$(openssl rand -hex 32)
```

Inicializácia databázy:

```
python manage.py db upgrade
python manage.py create_admin
```

8. Spustenie Celery workerov

```
celery -A app.celery worker --loglevel=info &
```

9. Frontend – webové rozhranie

```
cd frontend
npm install
npm run build
```

10. Nginx – reverzný proxy

Konfigurácia

```
sudo nano /etc/nginx/sites-available/iris
```

```
server {
    listen 80;
    server_name _;

    location / {
        proxy_pass http://127.0.0.1:8000;
        proxy_set_header Host $host;
        proxy_set_header X-Forwarded-For $remote_addr;
    }
}
```

Aktivácia konfigurácie:

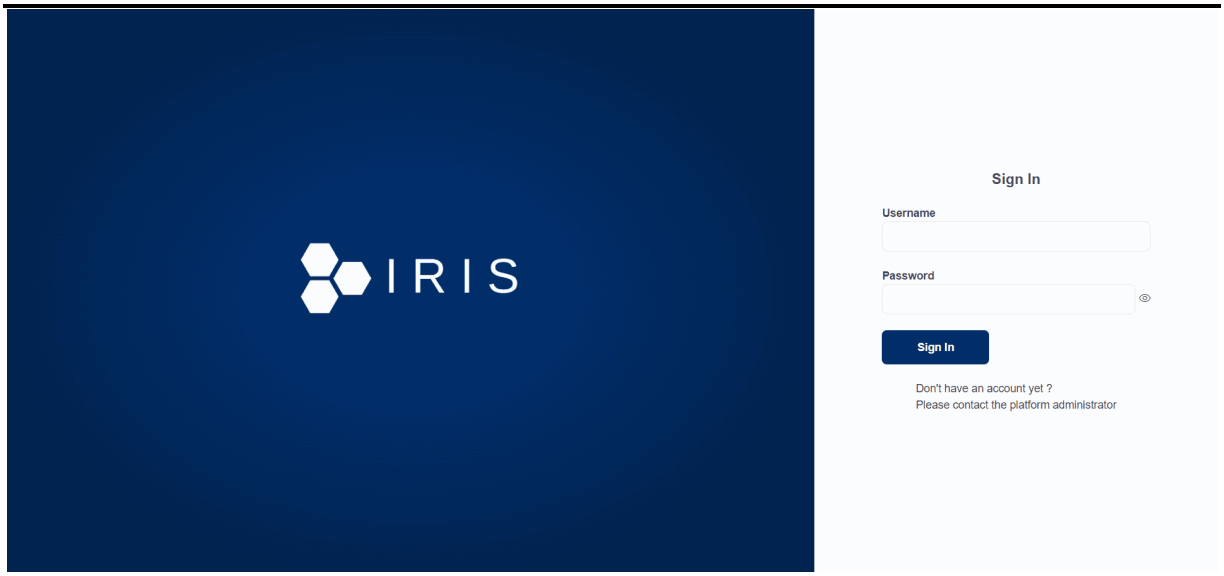
```
sudo ln -s /etc/nginx/sites-available/iris /etc/nginx/sites-enabled/
sudo nginx -t
sudo systemctl restart nginx
```

11. Spustenie DFIR-IRIS backendu

```
cd /opt/iris-web
source venv/bin/activate
gunicorn -w 4 -b 127.0.0.1:8000 app:app
```

12. Prístup do aplikácie

```
http://<IP_address_VM>
```



9.5 Základná konfigurácia

Konfigurácia nástroja.

Vstupy/Výstupy

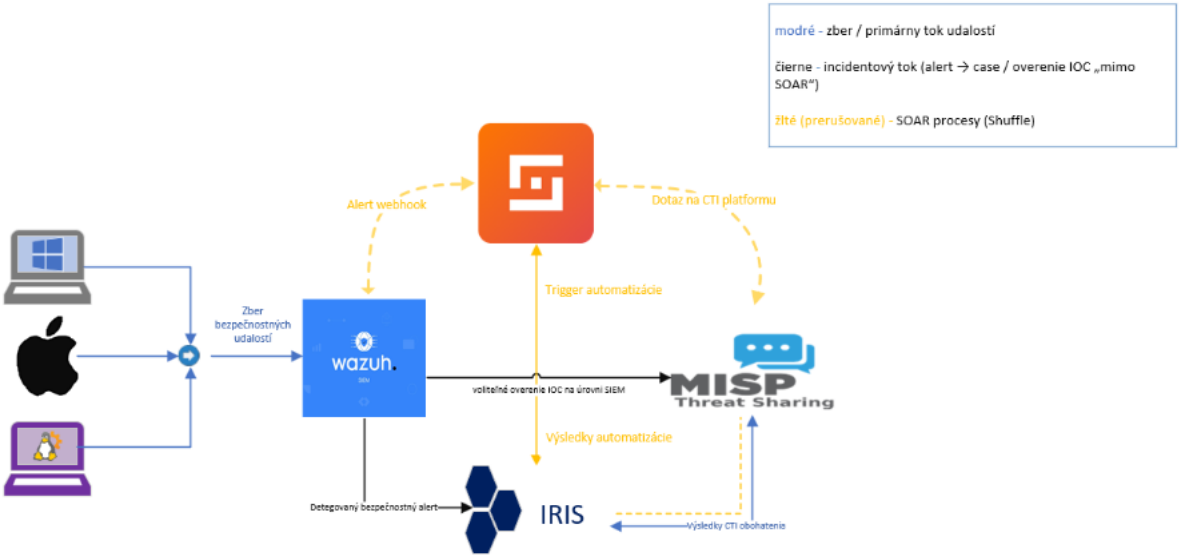
Vstupom do DFIR-IRIS sú bezpečnostné alerty, hlásenia incidentov alebo manuálne vytvorené prípady. Dáta môžu pochádzať zo SIEM systémov, CTI platforiem alebo od

analytikov. Výstupom sú spracované incidentové prípady, auditné záznamy, reporty a štruktúrované dáta, ktoré je možné exportovať alebo zdieľať s ďalšími systémami.

9.6 Integrácie

DFIR-IRIS podporuje integráciu s nástrojmi používanými v SOC prostredí, najmä so SIEM systémami, CTI platformami a SOAR riešeniami. Integrácie umožňujú automatizovaný príjem alertov, obohacovanie incidentov o kontext hrozieb a spätné zapisovanie výsledkov vyšetrovania.

Návrh prepojenia komponentov v rámci SOC prostredia



10 DFIR IRIS – EVIDENCIA, SPRÁVA INCIDENTOV

Typ nástroja: Evidencia, správa incidentov

Názov nástroja: DFIR IRIS

Verzia: v2.4.20

10.1 Opis systému

Nástroj DFIR-IRIS je navrhnutý ako centralizovaná platforma pre evidenciu a správu kybernetických bezpečnostných incidentov počas celého ich životného cyklu. Systém poskytuje jednotné prostredie, v ktorom sú sústredené všetky informácie súvisiace s riešením incidentu, vrátane jeho klasifikácie, priebehu vyšetrovania, prijatých opatrení a výsledného vyhodnotenia.

Základným prvkom evidencie incidentov v DFIR-IRIS je incidentový prípad (Case), ktorý reprezentuje ucelený záznam konkrétneho bezpečnostného incidentu. Každý prípad obsahuje základné identifikačné údaje, ako je názov incidentu, jeho popis, kategória, závažnosť a priradená organizácia. Okrem týchto metadát systém umožňuje priradenie analytikov, evidenciu úloh, zaznamenávanie poznámok a automatické vytváranie časovej osi riešenia incidentu.

DFIR-IRIS podporuje systematický prístup k správe incidentov tým, že umožňuje priebežné dopĺňanie technických informácií, ako sú indikátory kompromitácie, artefakty, dôkazové súbory a výsledky analýzy. Všetky vykonané kroky sú zaznamenávané auditovateľným spôsobom, čo zabezpečuje spätnú dohľadateľnosť rozhodnutí a podporuje potreby interných auditov alebo následného reportovania.

V porovnaní s tradičnými ticketovacími systémami poskytuje DFIR-IRIS širší kontext správy incidentov, keďže nielen eviduje samotnú udalosť, ale zároveň podporuje celý proces Incident Response. Tento prístup umožňuje analytikom pracovať s incidentom ako s jedným uceleným objektom, čím sa znižuje fragmentácia informácií a zvyšuje prehľadnosť riešenia incidentov v prostredí bezpečnostného operačného centra.

10.2 Dokumentácia systému

Oficialna dokumentácia:

https://docs.dfir-iris.org/latest/

Oficiálny GitHub projekt:

https://github.com/dfir-iris/iris-web

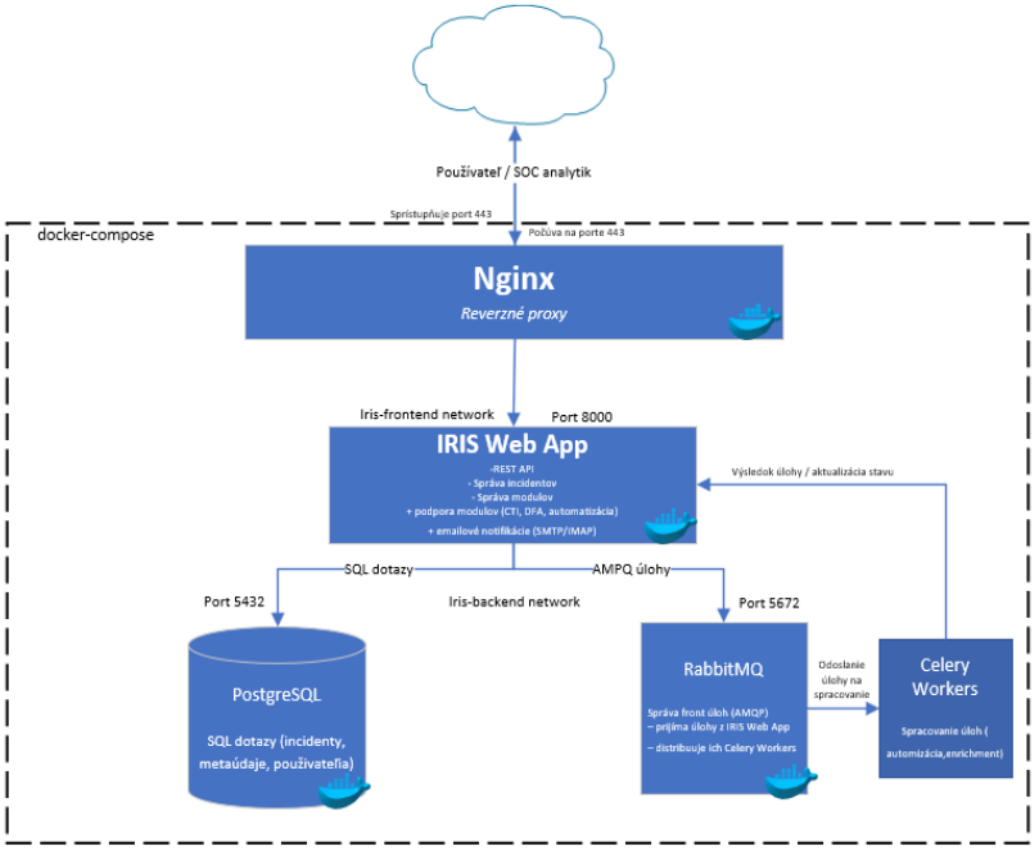
Hardvérové požiadavky

DFIR-IRIS má stredné hardvérové nároky. Pre testovacie alebo menšie produkčné nasadenie postačuje virtuálny server s minimálne 2–4 CPU jadrami, 8 GB RAM a približne 40 GB diskového priestoru. Pre väčšie SOC prostredia s viacerými analytikmi a dlhodobou archiváciou incidentov sa odporúča 4–8 CPU jadier, 16 GB RAM a rýchle SSD úložisko. Nároky rastú najmä pri práci s prílohami, dôkazmi a generovaní reportov.

Architektúra a schémy

Architektúra DFIR-IRIS je modulárna a pozostáva z webovej aplikácie, databázovej vrstvy a asynchrónnych spracovateľských komponentov. Webové rozhranie slúži ako hlavný bod interakcie používateľa so systémom. Backend komunikuje s databázou PostgreSQL, zatiaľ čo časovo náročné operácie sú spracúvané asynchrónne prostredníctvom frontu správ (RabbitMQ) a workerov. Súčasťou architektúry je REST API, ktoré umožňuje integráciu s ďalšími SOC komponentmi.

Schéma architektúry DFIR IRIS



10.3 Inštalácia – variant Docker

1. Aktualizácia systému

```
sudo apt update && sudo apt upgrade -y
```

2. Inštalácia potrebných balíkov

```
sudo apt install -y ca-certificates curl gnupg lsb-release git
```

3. Pridanie Docker GPG kľúča

```
sudo mkdir -p /etc/apt/keyrings
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | \
sudo gpg --dearmor -o /etc/apt/keyrings/docker.gpg
```

4. Pridanie Docker repozitára

```
echo \
"deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.gpg] \
https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable" | \
sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
```

5. Inštalácia Docker Engine a Docker Compose

```
sudo apt update
sudo apt install -y docker-ce docker-ce-cli containerd.io docker-compose-plugin
```

6. Povolenie Docker služby

```
sudo systemctl enable docker
sudo systemctl start docker
```

7. Overenie Dockeru

```
docker --version
docker compose version
```

8. Stiahnutie DFIR-IRIS

```
cd /opt
git clone https://github.com/dfir-iris/iris-web.git
cd iris-web
```

9. Spustenie DFIR-IRIS

```
docker compose up -d
```

10. Overenie bežiacich kontajnerov

```
docker compose ps
```

11. Zistenie admin hesla (prvý login)

```
docker compose logs app | grep create_safe_admin
```

12. Prístup do webového rozhrania

```
https://<IP_adresa_servera>
```

10.4 Inštalácia – variant Openstack

1. Aktualizácia systému

```
sudo apt update && sudo apt upgrade -y
```

2. Inštalácia systémových závislostí

```
sudo apt install -y \
       git curl build-essential \
       python3 python3-venv python3-pip \
       postgresql postgresql-contrib \
       rabbitmq-server \
       nodejs npm \
       nginx
```

Overenie verzií:

```
python3 --version
       psql --version
       node --version
```

3. Nastavenie PostgreSQL databázy

Spustenie databázy

```
sudo systemctl enable postgresql
sudo systemctl start postgresql
```

Vytvorenie databázy a používateľa

```
sudo -u postgres psql

CREATE DATABASE iris;
CREATE USER irisuser WITH PASSWORD 'StrongPassword123';
GRANT ALL PRIVILEGES ON DATABASE iris TO irisuser;
ALTER DATABASE iris OWNER TO irisuser;
\q
```

4. Nastavenie RabbitMQ

```
sudo systemctl enable rabbitmq-server
sudo systemctl start rabbitmq-server
```

Vytvorenie používateľa:

```
sudo rabbitmqctl add_user iris irispassword
sudo rabbitmqctl add_vhost iris_vhost
sudo rabbitmqctl set_permissions -p iris_vhost iris ".*" ".*" ".*"
```

5. Stiahnutie DFIR-IRIS zdrojového kódu

```
cd /opt
       sudo git clone https://github.com/dfir-iris/iris-web.git
       sudo chown -R $USER:$USER iris-web
       cd iris-web
```

6. Backend – Python aplikácia

Vytvorenie virtuálneho prostredia

```
python3 -m venv venv
source venv/bin/activate
```

Inštalácia Python balíkov

```
pip install --upgrade pip
pip install -r requirements.txt
```

7. Nastavenie backend konfigurácie

Vytvorenie konfiguračných premenných:

```
export IRIS_ENV=production
export DATABASE_URL=postgresql://irisuser:StrongPassword123@localhost/iris
export RABBITMQ_URL=amqp://iris:irispassword@localhost:5672/iris_vhost
export SECRET_KEY=$(openssl rand -hex 32)
```

Inicializácia databázy:

```
python manage.py db upgrade
python manage.py create_admin
```

8. Spustenie Celery workerov

```
celery -A app.celery worker --loglevel=info &
```

9. Frontend – webové rozhranie

```
cd frontend
npm install
npm run build
```

10. Nginx – reverzný proxy

Konfigurácia

```
sudo nano /etc/nginx/sites-available/iris

server {
    listen 80;
    server_name _;

    location / {
        proxy_pass http://127.0.0.1:8000;
        proxy_set_header Host $host;
        proxy_set_header X-Forwarded-For $remote_addr;
    }
}
```

Aktivácia konfigurácie:

```
sudo ln -s /etc/nginx/sites-available/iris /etc/nginx/sites-enabled/
sudo nginx -t
sudo systemctl restart nginx
```

11. Spustenie DFIR-IRIS backendu

```
cd /opt/iris-web
source venv/bin/activate
gunicorn -w 4 -b 127.0.0.1:8000 app:app
```

12. Prístup do aplikácie

```
http://<IP_address_VM>
```



10.5 Základná konfigurácia

V rámci webového rozhrania nástroja DFIR-IRIS bol realizovaný test manuálneho vytvorenia nového incidentového prípadu. Systém umožňuje zadanie základných údajov o incidente, ako sú názov incidentu, jeho popis, priradená organizácia a identifikátor externého SOC tiketu. Proces vytvorenia incidentového prípadu prebehol bez technických problémov, čím bola overená základná funkčnosť nástroja v oblasti evidencie incidentov.

Počas ďalšej práce so systémom bolo identifikované, že v predvolenej konfigurácii nástroja chýbajú incidentové šablóny pre konkrétne typy kybernetických bezpečnostných udalostí. Neexistovali predpripravené šablóny napríklad pre phishingové útoky, malvérové incidenty, kompromitáciu používateľských účtov alebo ransomvérové útoky. V dôsledku toho by každý incident musel byť vytváraný manuálne, bez jednotnej štruktúry a bez preddefinovaných krokov riešenia.

Takýto spôsob práce by v reálnom prevádzkovom prostredí viedol k zníženej efektivite riešenia incidentov a k absencii konzistentného, štandardizovaného postupu v rámci SOC tímu. Z uvedeného dôvodu bolo potrebné overiť existenciu dostupných incidentových šablón, ktoré by bolo možné do nástroja DFIR-IRIS doplniť.

Pri tomto overovaní boli identifikované viaceré komunitné incidentové šablóny určené pre staršie verzie nástroja DFIR-IRIS. Následné testy ich importu však preukázali nekompatibilitu s aktuálnou verziou systému, keďže DFIR-IRIS už nepodporuje pôvodný formát šablón založený na JSON štruktúre.

Na základe týchto zistení bolo rozhodnuté existujúce šablóny upraviť a prispôsobiť ich aktuálnemu formátu nástroja, aby ich bolo možné úspešne importovať a následne využívať pri evidencii a správe incidentov.

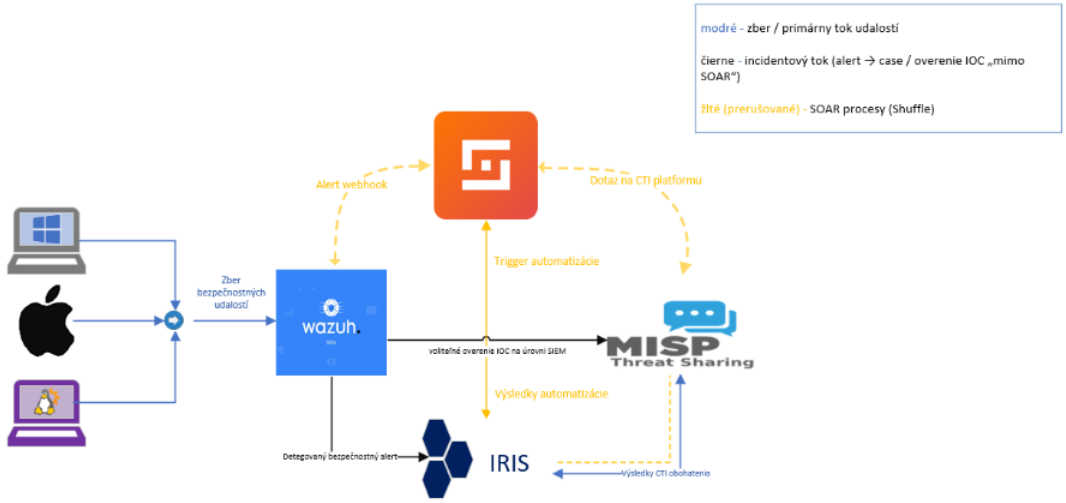
Vstupy/Výstupy

Vstupom do DFIR-IRIS sú bezpečnostné alerty, hlásenia incidentov alebo manuálne vytvorené prípady. Dáta môžu pochádzať zo SIEM systémov, CTI platforiem alebo od analytikov. Výstupom sú spracované incidentové prípady, auditné záznamy, reporty a štruktúrované dáta, ktoré je možné exportovať alebo zdieľať s ďalšími systémami.

10.6 Integrácie

DFIR-IRIS podporuje integráciu s nástrojmi používanými v SOC prostredí, najmä so SIEM systémami, CTI platformami a SOAR riešeniami. Integrácie umožňujú automatizovaný príjem alertov, obohacovanie incidentov o kontext hrozieb a spätné zapisovanie výsledkov vyšetrovania.

Návrh prepojenia komponentov v rámci SOC prostredia



11 KIBANA (ELK)

Typ nástroja: Vizualizačný a analytický nástroj (SIEM front-end/Data Visualization)

Názov nástroja: Kibana

Verzia: Kibana 9.x (open-source/Elastic Stack)

11.1 Opis nástroja

Kibana je webový vizualizačný a analytický nástroj, ktorý tvorí prezentačnú a analytickú vrstvu Elastic Stacku. V prostredí SOC je jej hlavným účelom sprístupniť bezpečnostné a prevádzkové dáta vo forme prehľadných vizualizácií, dashboard-ov a reportov, ktoré podporujú operatívne rozhodovanie analytikov aj manažérsky dohľad.

Základnou funkciou Kibany je umožniť interaktívnu prácu s dátami uloženými v Elasticsearch, a to bez potreby priamej manipulácie s databázou. Používateľom poskytuje nástroje na:

- vizualizáciu bezpečnostných udalostí a logov,
- analýzu trendov a anomálií,
- koreláciu udalostí naprieč zdrojmi,
- tvorbu vlastných dashboardov pre SOC analytikov,
- generovanie reportov pre manažment a audit.

V navrhovanej SOC architektúre plní Kibana úlohu centrálneho grfického rozhrania pre prácu s detekčnými dátami, pričom umožňuje rýchlu orientáciu v bezpečnostnej situácii, sledovanie stavu infraštruktúry a podporu vyšetrovania incidentov.

Z pohľadu manažmentu slúži Kibana ako reportingový a prezentačný nástroj, ktorý poskytuje agregovaný pohľad na bezpečnostné ukazovatele (počty incidentov, trendy, typy hrozieb) bez potreby technických znalostí detailov infraštruktúry.

11.2 Dokumentácia systému

Oficiálna dokumentácia nástroja Kibana je súčasťou dokumentácie Elastic Stacku a poskytuje detailný popis architektúry, možností vizualizácie, práce s dátami a integrácie s ostatnými komponentmi stacku. Dokumentácia pokrýva aj bezpečnostné mechanizmy, reporting a správu používateľských práv.

Odkazy na oficiálne zdroje:

- Oficiálna dokumentácia Kibana: https://www.elastic.co/guide/en/kibana/current/index.html
- Oficiálna stránka Elastic Stack: https://www.elastic.co/elastic-stack
- GitHub repozitár: https://github.com/elastic/kibana
- Oficiálny Docker image: https://hub.docker.com/_/kibana
- Neoficiálna docker compose dokumentácia: https://github.com/deviantony/docker-elk

11.3 Požiadavky na HW

Hardvérové požiadavky Kibany závisia od objemu spracovávaných dát, počtu používateľov a komplexnosti dashboardov. Keďže Kibana pracuje výhradne nad dátami uloženými v Elasticsearch, jej nároky sú nižšie ako pri indexačnej vrstve, no významne ovplyvnené interaktívnym používaním.

Minimálne požiadavky (PoC / menšie SOC):

- CPU: 2 jadrá
- RAM: 2–4 GB
- Disk: minimálny (bez perzistencie dát)

Odporúčané požiadavky (produkčné SOC):

- CPU: 4–8 jadier
- RAM: 8 GB a viac
- Disk: SSD (logy a cache)

Vyššie nároky sa uplatňujú pri:

- veľkom počte súčasne pripojených používateľov,
- komplexných vizualizáciách a koreláciách,
- generovaní reportov a dlhodobých dashboardov.

11.4 Architektúra nástroja

Architektúra Kibany je navrhnutá ako klient-server webová aplikácia, ktorá funguje výhradne v úzkej väzbe na Elasticsearch. Kibana neukladá dáta samostatne, ale pracuje s indexmi uloženými v Elasticsearch, nad ktorými vykonáva dotazy a vizualizácie.

Základ architektúry tvorí:

- webová aplikačná vrstva, ktorá poskytuje používateľské rozhranie,
- aplikačná logika, ktorá spracúva dotazy, filtrovanie a vizualizácie,
- komunikačná vrstva, ktorá zabezpečuje interakciu s Elasticsearch API.

V kontexte SOC architektúry je Kibana umiestnená nad detekčnou vrstvou (Elasticsearch) a slúži ako hlavné rozhranie pre analytikov a manažment. Umožňuje sledovať aktuálny stav bezpečnosti, analyzovať historické dáta a podporovať rozhodovanie bez priameho zásahu do dátovej vrstvy.

Schéma

Schéma znázorňuje pozíciu Kibany ako prezentačnej a analytickej vrstvy nad Elasticsearch v rámci SOC architektúry. Kibana nespracúva dáta priamo, ale umožňuje ich vizualizáciu, analýzu a reporting pre potreby SOC analytikov a manažmentu na základe dát indexovaných v Elasticsearch.

```
[ Zdroje udalostí ]
  - servery (Linux / Windows)
  - sieťové zariadenia
```

```
  - aplikácie
  - bezpečnostné systémy
          |
          | (Beats / Logstash / Ingest pipelines)
          v
+-----------------------------------+
|          Elasticsearch            |
|                                   |
|  - indexy logov                   |
|  - bezpečnostné udalosti          |
|  - metadáta / časové rady         |
+-----------------------------------+
          |
          | (REST API / Query DSL)
          v
+-----------------------------------+
|              Kibana               |
|                                   |
|  +-----------------------------+  |
|  | Web UI                      |  |
|  | - dashboards (SOC)          |  |
|  | - vizualizácie              |  |
|  | - time-series analýza       |  |
|  | - reporting                 |  |
|  +-----------------------------+  |
|                                   |
|  +-----------------------------+  |
|  | Analytická logika           |  |
|  | - filtre / korelácie        |  |
|  | - alerting (rules)          |  |
|  | - saved searches            |  |
|  +-----------------------------+  |
+-----------------------------------+
          |
          | (vizualizácie / reporty)
          v
+-----------------------------------+
| Používatelia                      |
| - SOC analytici                   |
| - bezpečnostní operátori          |
| - manažment                       |
+-----------------------------------+
```

11.5 Inštalacia – variant Docker

```
services:
  kibana:
    build:
      context: kibana/
      args:
        ELASTIC_VERSION: ${ELASTIC_VERSION}
    volumes:
      - ./kibana/config/kibana.yml:/usr/share/kibana/config/kibana.yml:ro,Z
    ports:
      # - 5601:5601
```

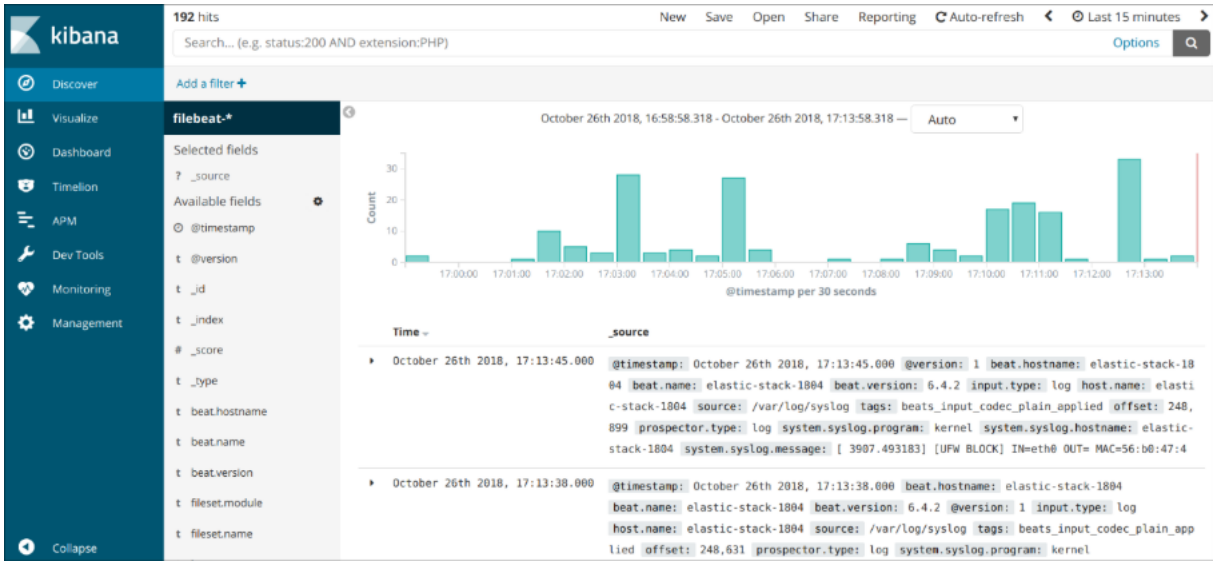
```
      - 80:5601
    environment:
      KIBANA_SYSTEM_PASSWORD: ${KIBANA_SYSTEM_PASSWORD:-}
      SERVER_PUBLICBASEURL: http://elk.innosoc.brainit.tech
    networks:
      - elk
    depends_on:
      - elasticsearch
    restart: unless-stopped
networks:
  elk:
    driver: bridge
volumes:
  elasticsearch:
```

Zmeny v docker compose:

- Toto je samotná Kibana extrahovaná z celého ELK stacku (1. ELK Stack)
- Zmenené niektoré sieťové porty – viď zakomentované riadky vyššie.

11.6 Základná konfigurácia

V prípade že je súčasťou ELK Stacku, obsahuje konfiguračný skript ktorý správne nastaví adresy a potrebnú autentifikáciu medzi službami. V opačnom prípade je len potrebné nastaviť adresy a autentifikáciu manuálne podľa oficiálnej dokumentácie.



11.7 Integrácie

Kibana je len frontend ktorý sa spravidla integruje najmä len s elasticsearch. Elasticsearch má potom ďalšie široké možnosti ďalšej integrácie.

12 WIKI.JS

Typ nástroja: Knowledge Management/Interná dokumentácia

Názov nástroja: Wiki.js

Verzia: Wiki.js verzie 2.5 (open-source, kontajnerizované nasadenie)

12.1 Opis systému

Wiki.js je open-source nástroj určený na správu znalostnej bázy a internej dokumentácie. V prostredí SOC slúži ako centrálny znalostný repozitár, ktorý zhromažďuje prevádzkové postupy, runbooky, playbooky, interné smernice a dokumentáciu systémov využívaných pri zabezpečení infraštruktúry.

Hlavným účelom nasadenia Wiki.js je:

- tvorba a správa runbookov a operačných postupov SOC,
- dokumentácia bezpečnostných procesov a playbookov,
- podpora onboardingu nových analytikov a operátorov,
- centrálna evidencia interných systémov, architektúr a konfigurácií.

Wiki.js umožňuje uchovávať a organizovať znalosti spôsobom, ktorý je dlhodobo udržateľný a nezávislý od konkrétnych osôb. Dokumentácia je dostupná prostredníctvom webového rozhrania a podporuje verzovanie, štruktúrovanie obsahu a kontrolu prístupov.

V navrhovanej SOC architektúre Wiki.js nevystupuje ako technický bezpečnostný nástroj, ale ako znalostná a metodická vrstva, ktorá podporuje konzistentný postup analytikov pri riešení incidentov a zabezpečuje, že know-how organizácie je systematicky zaznamenané a dostupné.

12.2 Dokumentácia systému

Oficiálna dokumentácia nástroja Wiki.js je verejne dostupná a poskytuje podrobný návod na inštaláciu, konfiguráciu, správu obsahu a integráciu s externými systémami. Dokumentácia sa zameriava najmä na praktické používanie nástroja ako internej znalostnej bázy.

Odkazy na oficiálne zdroje:

- Oficiálna dokumentácia: https://docs.requarks.io/
- Oficiálna stránka projektu: https://js.wiki/
- GitHub repozitár: https://github.com/requarks/wiki
- Oficiálny Docker image: https://hub.docker.com/r/requarks/wiki
- Oficálne možnosti integrácií https://js.wiki/modules

12.3 Požiadavky na zdroje

Hardvérové požiadavky Wiki.js sú nízke a umožňujú jej nasadenie aj v menších prostrediach alebo ako doplnkový komponent SOC infraštruktúry.

Minimálne požiadavky (PoC / menšie prostredie):

- CPU: 1–2 jadrá
- RAM: 2–4 GB
- Diskový priestor: 10–20 GB
- Databáza: PostgreSQL / MySQL / MariaDB

Odporúčané požiadavky (produkčné prostredie):

- CPU: 2–4 jadrá
- RAM: 4–8 GB
- Disk: SSD (v závislosti od objemu dokumentácie)

Vyššie nároky môžu vzniknúť pri veľkom počte používateľov alebo rozsiahlej histórii verzií dokumentácie.

12.4 Architektúra nástroja

Architektúra Wiki.js je navrhnutá ako ľahká viacvrstvová webová aplikácia, ktorá pozostáva z prezentačnej, aplikačnej a dátovej vrstvy. Tento model umožňuje jednoduché nasadenie, údržbu a integráciu s ostatnými systémami v SOC prostredí.

Prezentačnú vrstvu tvorí webové používateľské rozhranie, ktoré umožňuje:

- prehliadanie a úpravu dokumentácie,
- správu štruktúry znalostnej bázy,
- vyhľadávanie v obsahu,
- správu prístupových práv.

Aplikačná vrstva zabezpečuje spracovanie obsahu, verzionovanie dokumentov, workflow publikovania a autentifikáciu používateľov. Wiki.js podporuje integráciu s externými autentifikačnými systémami (napr. LDAP, OAuth, SSO), čo je výhodné v organizovanom SOC prostredí.

Dátová vrstva je tvorená relačnou databázou, v ktorej sú uložené:

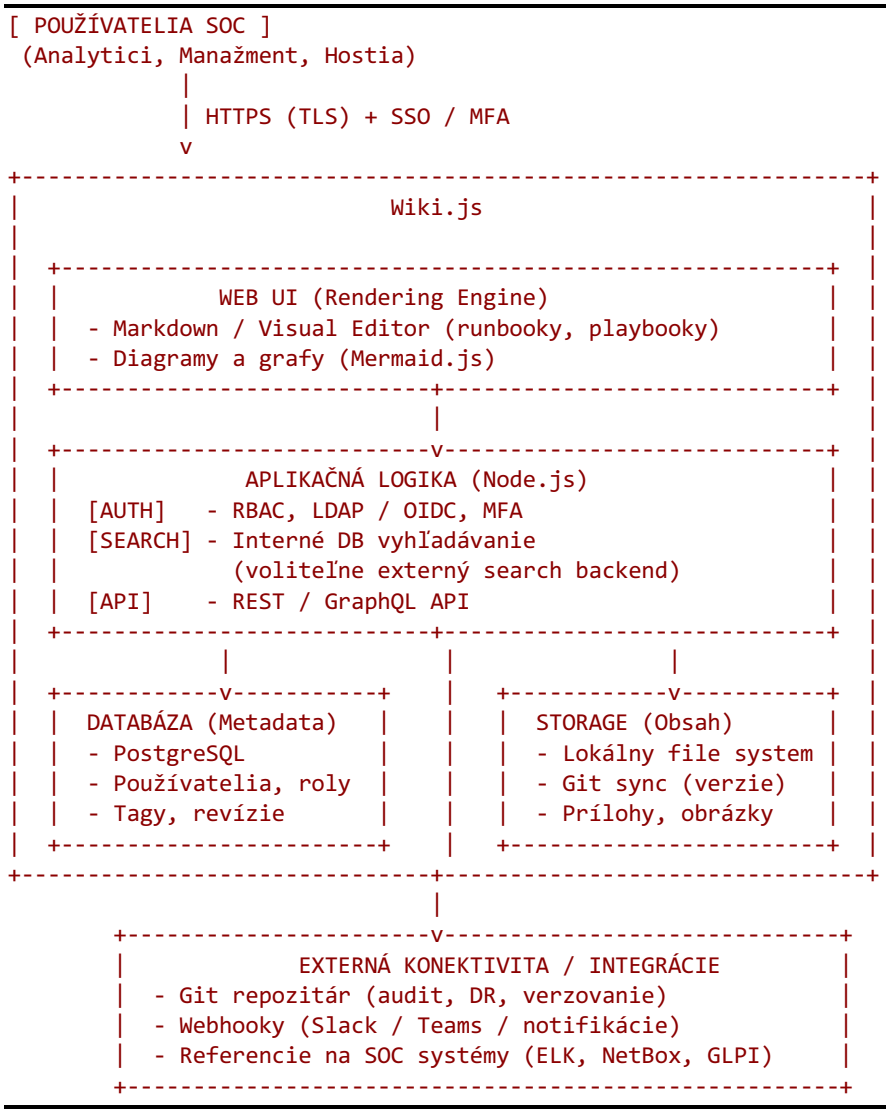
- samotné dokumenty,
- verzionovaná história zmien,
- používateľské účty a oprávnenia,
- metadáta dokumentácie.

V kontexte SOC architektúry je Wiki.js umiestnená mimo detekčnej a reakčnej vrstvy a slúži ako podporný systém, ktorý zabezpečuje, že postupy, rozhodnutia a prevádzkové znalosti sú jednotne zdokumentované a dostupné pre všetkých členov SOC tímu.

Schéma

Schéma znázorňuje pozíciu Wiki.js ako znalostnej a dokumentačnej vrstvy v SOC architektúre. Wiki.js centralizuje runbooky, operačné postupy a playbooky, ktoré sú využívané analytikmi počas riešenia incidentov. Prostredníctvom odkazov a referencií prepája procesnú,

technickú a analytickú vrstvu SOC bez priameho zásahu do detekčných alebo reakčných systémov.

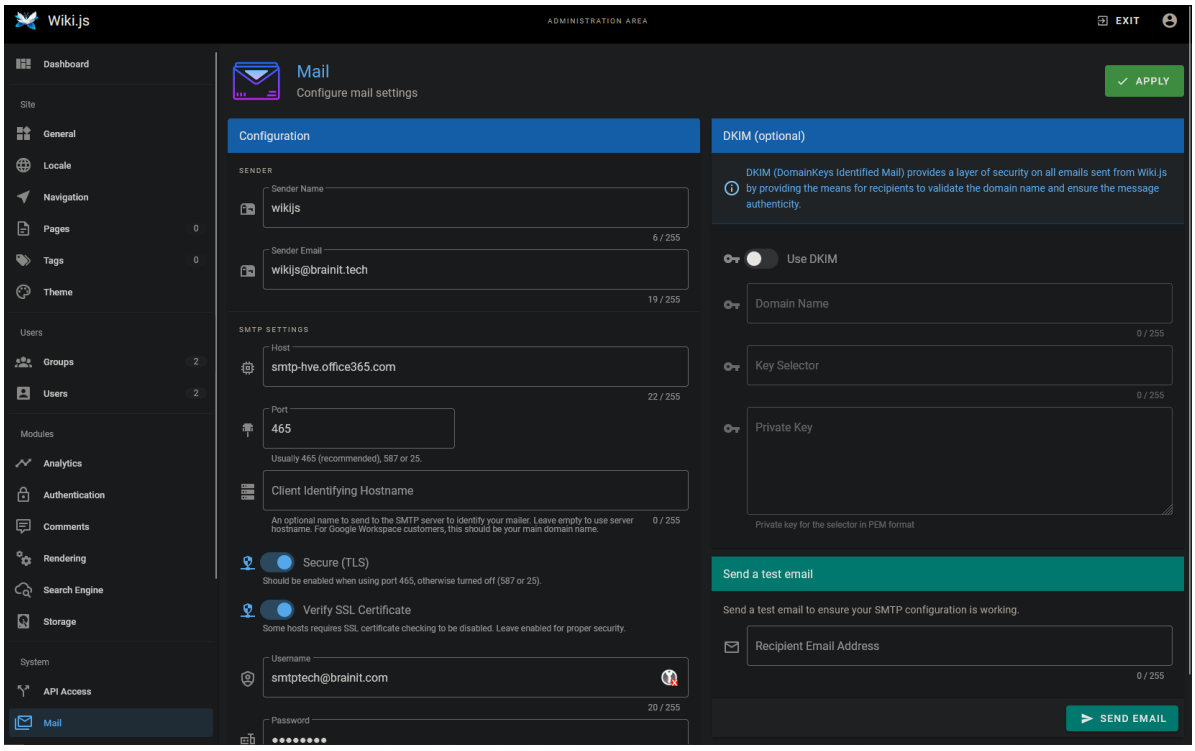


12.5 Inštalácia – variant Docker

```
Docker-compose.yml
services:
  db:
    image: postgres:15-alpine
    environment:
      POSTGRES_DB: wiki
      POSTGRES_PASSWORD: wikijsrocks
      POSTGRES_USER: wikijs
    logging:
      driver: none
    restart: unless-stopped
```

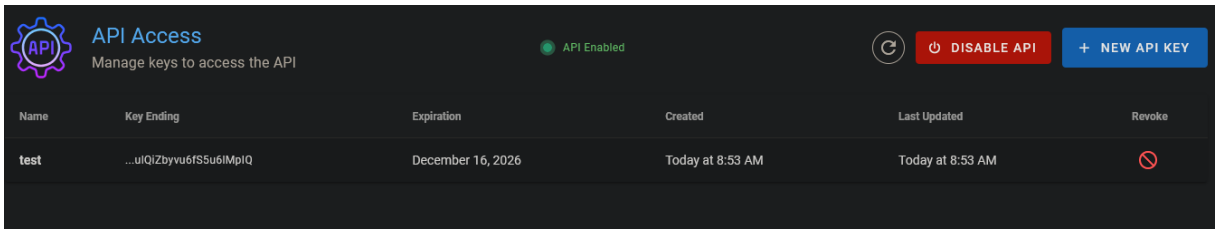
```
    volumes:
      - db-data:/var/lib/postgresql/data
  wiki:
    image: ghcr.io/requarks/wiki:2
    depends_on:
      - db
    environment:
      DB_TYPE: postgres
      DB_HOST: db
      DB_PORT: 5432
      DB_USER: wikijs
      DB_PASS: wikijsrocks
      DB_NAME: wiki
    restart: unless-stopped
    ports:
      - "80:3000"
volumes:
  db-data:
```

12.6 Základná konfigurácia

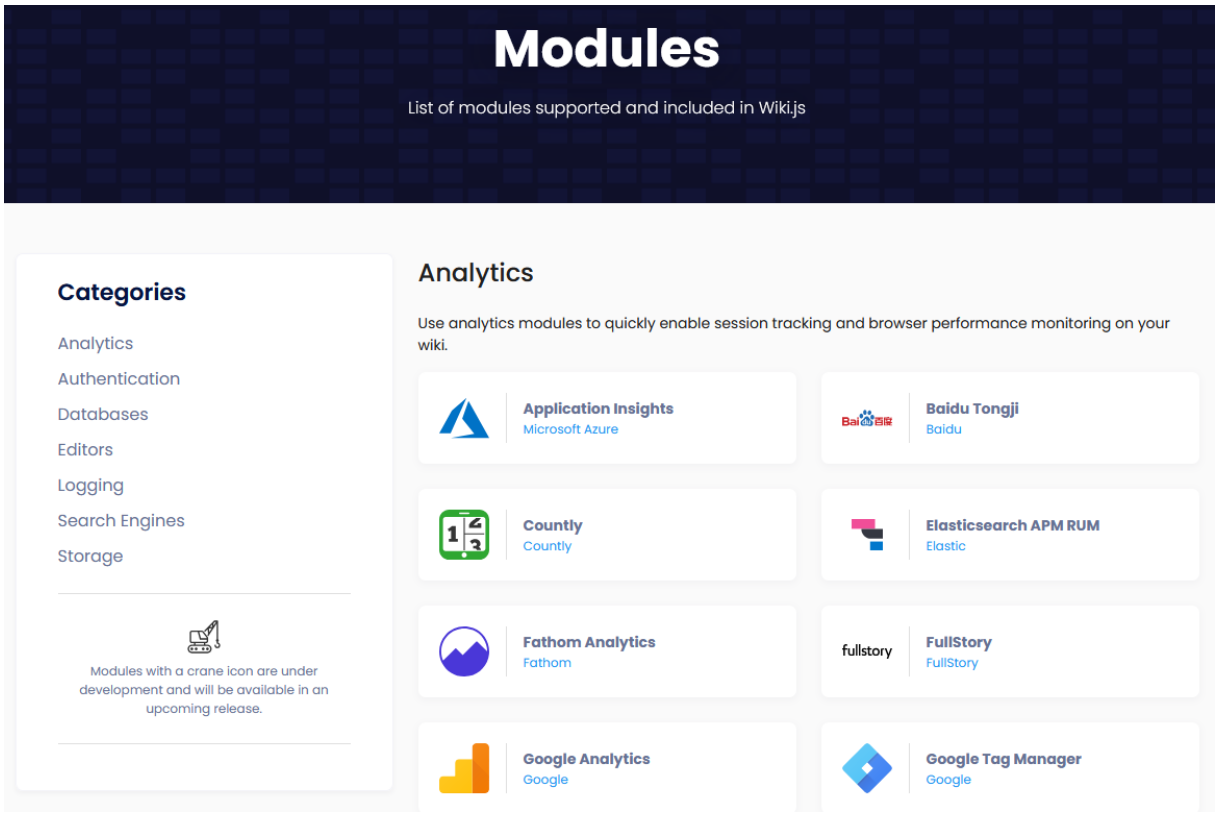


Konfigurácia SMTP

12.7 Integrácie



Možnosť integrácie cez API kľúče



Iné možnosti integrácií dostupné na webe: https://js.wiki/modules

13 PROSTREDIE NASADENIA

13.1 OpenStack KIS FRI UNIZA

13.1.1 Kontext CC riešenia

Prostredie nasadenia navrhovaného riešenia je založené na privátnom cloudovom prostredí OpenStack, ktoré je prevádzkované na Katedre informačných sietí (KIS), Fakulty riadenia a informatiky, Univerzity Žilina (FRI UNIZA). Toto prostredie je dlhodobo využívané na podporu výučby, výskumu a experimentálneho nasadzovania informačných a sieťových technológií. Cloud KIS FRI UNIZA je koncipovaný ako IaaS (Infrastructure as a Service) platforma, ktorá poskytuje virtualizované výpočtové zdroje, sieťové služby a úložiská. Použitie OpenStacku umožňuje vytváranie izolovaných virtuálnych prostredí, ktoré sú vhodné na testovanie komplexných architektúr bez vplyvu na produkčné služby fakulty alebo univerzity.

Z pohľadu projektu je prostredie KIS charakterizované ako kontrolované akademické cloudové prostredie, ktoré kombinuje vlastnosti produkčného cloudu s flexibilitou experimentálneho nasadenia. Tento charakter umožňuje realizáciu Proof of Concept (PoC) riešení, overovanie architektonických návrhov a testovanie integrácie bezpečnostných nástrojov v realistickom prevádzkovom kontexte. OpenStack cloud KIS podporuje vytváranie viacerých virtuálnych sietí, bezpečnostných skupín a samostatných tenantov, čo umožňuje logickú separáciu jednotlivých experimentov a služieb. Táto vlastnosť je kľúčová pre nasadenie SOC architektúry, kde je potrebné oddeliť detekčné, analytické a reakčné komponenty a zároveň zachovať možnosť ich vzájomnej integrácie.

Nasadenie navrhovaného riešenia v prostredí KIS FRI UNIZA nesleduje cieľ vytvoriť plnohodnotné produkčné bezpečnostné operačné centrum, ale slúži na overenie funkčnosti, integračných väzieb a prevádzkových vlastností jednotlivých komponentov SOC architektúry. Prostredie je preto považované za referenčné testovacie a výskumné prostredie, ktoré umožňuje replikovateľnosť experimentov a objektívne hodnotenie navrhovaného riešenia.

13.1.2 Architektúra OpenStack

Opis architektúry OpenStack

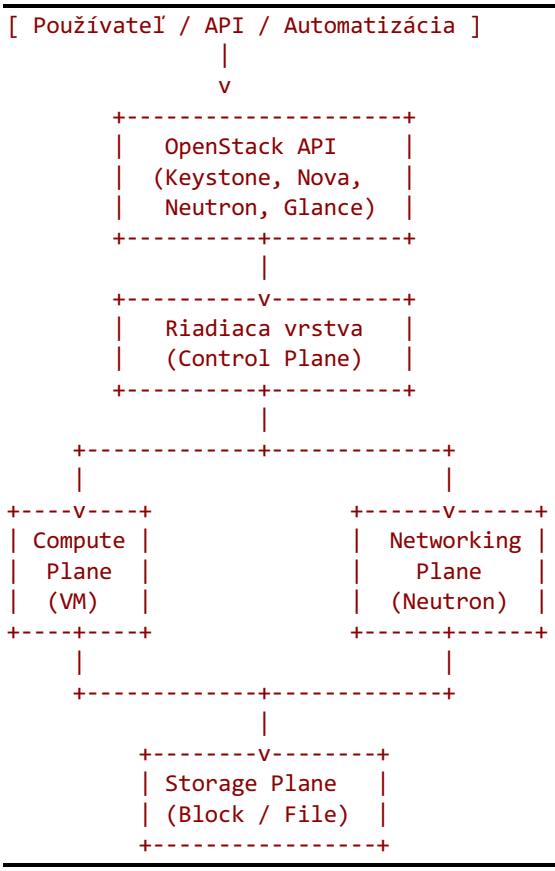
OpenStack je modulárna open-source cloudová platforma typu Infrastructure as a Service (IaaS), navrhnutá na správu a orchestrace výpočtových, sieťových a úložných zdrojov v dátových centrách a privátnych cloudových prostrediach. Architektúra OpenStacku je založená na súbore samostatných, no vzájomne prepojených služieb, ktoré spolu vytvárajú jednotné cloudové rozhranie.

Z architektonického hľadiska OpenStack funguje ako riadiaca a abstrahujúca vrstva medzi fyzickou infraštruktúrou a aplikačnými službami. Používateľ alebo nadriadené systémy (napr. automatizačné nástroje) nepracujú priamo s hardvérom, ale s logickými objektmi – virtuálnymi strojmi, sieťami, obrazmi a úložiskami – ktoré OpenStack dynamicky mapuje na fyzické zdroje.

Základné architektonické vrstvy OpenStack

Z logického pohľadu je architektúru OpenStack možné rozdeliť do nasledujúcich vrstiev:

- **Riadiaca vrstva (Control Plane):** Zodpovedá za autentifikáciu, plánovanie zdrojov, správu API a koordináciu jednotlivých služieb. Túto vrstvu tvoria najmä služby Keystone, Nova (control), Neutron (control) a Glance.
- **Výpočtová vrstva (Compute Plane):** Zahŕňa výpočtové uzly, na ktorých bežia virtuálne stroje. Nova zabezpečuje plánovanie VM a ich mapovanie na fyzické zdroje.
- **Sieťová vrstva (Networking Plane):** Realizovaná prostredníctvom služby Neutron, ktorá umožňuje vytváranie virtuálnych sietí, podsietí, smerovania a bezpečnostných pravidiel. Sieťová vrstva je kľúčová pre izoláciu tenantov a bezpečné prepojenie SOC komponentov.
- **Úložná vrstva (Storage Plane):** Poskytuje perzistentné aj dočasné úložisko pre virtuálne stroje a aplikácie. OpenStack umožňuje integráciu rôznych typov storage backendov podľa potrieb prostredia.



Táto schéma zobrazuje OpenStack ako sprostredkovateľa medzi infraštruktúrnou vrstvou a aplikačnými službami SOC, ktoré sú prevádzkované vo virtuálnych strojoch.

Dokumentácia OpenStack

OpenStack má rozsiahlu a detailnú oficiálnu dokumentáciu, ktorá pokrýva architektúru systému, jednotlivé služby, možnosti nasadenia, bezpečnostné aspekty a odporúčané postupy pre

prevádzku cloudových prostredí. Dokumentácia je udržiavaná komunitou a reflektuje reálne produkčné nasadenia v akademických aj komerčných prostrediach.

Oficiálne zdroje dokumentácie:

- Oficiálna dokumentácia OpenStack: https://docs.openstack.org/
- OpenStack Architecture Design Guide: https://docs.openstack.org/arch-design/
- OpenStack Project Documentation (Git): https://opendev.org/openstack

Tieto zdroje slúžia ako referenčný základ pre návrh architektúry cloudového prostredia použitého v tejto práci a pre zaradenie OpenStacku ako infraštruktúrnej platformy pre experimentálne SOC riešenie.

13.1.3 Cloudové riešenie – OpenStack ako platforma pre INNOSOC

Architektúra cloudového prostredia KIS FRI UNIZA je založená práve na platforme OpenStack, ktorá poskytuje modulárny súbor služieb určených na budovanie privátneho cloudového riešenia typu IaaS. V tejto kapitole sú uvedené len tie komponenty OpenStacku, ktoré sú **funkčne nevyhnutné** pre nasadenie a prevádzku experimentálneho SOC riešenia a ktoré sú zároveň všeobecne používané v OpenStack prostrediach.

Použité komponenty OpenStack architektúry pre CC INNOSOC

OpenStack je tvorený viacerými samostatnými službami, z ktorých každá plní presne definovanú úlohu. Pre potreby tohto riešenia sú relevantné najmä nasledovné komponenty:

- **Nova (Compute):** Zabezpečuje správu životného cyklu virtuálnych strojov. Umožňuje vytváranie, spúšťanie, zastavovanie a odstraňovanie VM, ktoré tvoria základné nosné prostredie pre jednotlivé SOC komponenty (SIEM, SOAR, CTI a pod.).
- **Neutron (Networking):** Poskytuje sieťovú konektivitu medzi virtuálnymi strojmi a umožňuje vytváranie virtuálnych sietí, podsietí a bezpečnostných pravidiel. Neutron je kľúčový pre logickú izoláciu SOC komponentov a oddelenie testovacích a experimentálnych segmentov.
- **Glance (Image Service):** Slúži na správu obrazov operačných systémov, z ktorých sú vytvárané virtuálne stroje. Použitie Glance umožňuje opakovateľné nasadenie rovnakých VM šablón, čo je dôležité pre replikovateľnosť PoC experimentov.

Tieto tri služby tvoria minimálny a všeobecne akceptovaný základ OpenStack IaaS riešenia a sú dostatočné na realizáciu experimentálneho nasadenia SOC architektúry. Okrem základných komponentov využíva OpenStack aj služby, ktoré zabezpečujú riadenie, autentifikáciu a používateľský prístup:

- **Keystone (Identity Service):** Zodpovedá za autentifikáciu a autorizáciu používateľov a služieb. Umožňuje správu tenantov a prístupových práv, čo je dôležité pre oddelenie jednotlivých projektov a experimentov v akademickom prostredí.
- **Horizon (Web Dashboard):** Poskytuje webové rozhranie pre správu cloudových zdrojov. Slúži ako administračné rozhranie pre vytváranie a správu virtuálnych strojov a sietí bez potreby priamej práce s API.

Tieto ďalšie komponenty umožňujú bezpečnú a prehľadnú správu cloudového prostredia bez zásahu do samotnej infraštruktúry.

Vzťah OpenStack v SOC architektúre INNOSOC

OpenStack v navrhovanom riešení nevystupuje ako bezpečnostný nástroj a priamo sa nezapája do detekcie, korelácie ani reakcie na incidenty. Jeho úlohou je poskytovať infraštruktúrnu platformu typu IaaS, ktorá umožňuje nasadzovať a prevádzkovať jednotlivé SOC komponenty ako izolované služby vo virtuálnych strojoch. V prostredí INNOSOC tým OpenStack plní rolu abstrahovanej infraštruktúrnej vrstvy, nad ktorou je možné pružne budovať a meniť experimentálne architektúry bez zásahu do fyzických zdrojov a bez ovplyvnenia ostatných služieb fakulty.

Z praktického hľadiska OpenStack umožňuje vytvárať samostatné virtuálne prostredia (projekty/tenanty), v ktorých sú SOC komponenty oddelené do vlastných inštancií. To podporuje logickú separáciu detekčných, analytických a reakčných častí riešenia a zároveň umožňuje riadiť ich životný cyklus nezávisle. V PoC režime je dôležitá najmä schopnosť rýchlo vytvárať a rušiť testovacie prostredia, upravovať parametre virtuálnych strojov podľa aktuálnych potrieb (CPU, RAM, disk, sieťové pripojenia) a opakovane nasadzovať rovnaké šablóny systémov, čo zvyšuje replikovateľnosť experimentov.

Oddelenie infraštruktúrnej logiky od bezpečnostnej architektúry má aj metodický prínos: umožňuje hodnotiť navrhované SOC riešenie primárne cez integračné väzby a prevádzkové vlastnosti jednotlivých nástrojov (SIEM, SOAR, CTI, asset kontext a pod.), nie cez špecifiká konkrétneho hardvéru. OpenStack tak tvorí stabilnú prevádzkovú bázu, ktorá poskytuje kontrolované a opakovateľné podmienky pre experimentálne nasadenie SOC architektúry.

Automatizované nasadenie a správa OpenStack platformy

Cloudové prostredie OpenStack KIS FRI UNIZA je budované a spravované s dôrazom na automatizáciu nasadenia a konzistentnú správu infraštruktúry. Základným cieľom tohto prístupu je minimalizovať manuálne zásahy do platformy, zjednodušiť aktualizácie jednotlivých vrstiev a zabezpečiť opakovateľnosť nasadenia v súlade s princípmi infraštruktúry ako kódu (Infrastructure as Code).

Automatizované nasadenie platformy je realizované prostredníctvom nástrojov spoločnosti Canonical, ktoré sú bežne využívané pri prevádzke OpenStack prostredí. Fyzická infraštruktúra je spravovaná pomocou nástroja MAAS (Metal as a Service), ktorý umožňuje automatizované nasadenie operačných systémov na bare-metal servery prostredníctvom PXE bootovania. MAAS zabezpečuje evidenciu hardvérových zdrojov, ich inicializáciu a prípravu pre ďalšie vrstvy cloudovej platformy.

Nad fyzickou vrstvou je využívaný nástroj Juju, ktorý slúži ako hlavný orchestračný mechanizmus OpenStack platformy. Juju umožňuje modelovať jednotlivé služby ako samostatné aplikačné entity a riadiť ich životný cyklus prostredníctvom tzv. charms. Charms definujú spôsob inštalácie, konfigurácie, prepojenia a aktualizácie jednotlivých komponentov a sú distribuované prostredníctvom repozitára Charmhub, ktorý v tomto kontexte plní obdobnú úlohu ako Docker Hub pre kontajnerové aplikácie.

Vďaka tomuto prístupu je možné nasadzovať OpenStack služby konzistentne naprieč rôznymi typmi prostredí, či už ide o virtuálne stroje, kontajnery (LXD) alebo priamo bare-metal

servery. Automatizácia zároveň umožňuje efektívnu správu aktualizácií platformy a znižuje riziko nekonzistentných konfigurácií medzi jednotlivými uzlami.

Okrem základných služieb využívaných v PoC scenároch sú v prostredí nasadené aj ďalšie OpenStack komponenty, ktoré rozširujú funkcionalitu cloudovej platformy a podporujú pokročilejšie prevádzkové scenáre.

Medzi kľúčové nasadené služby patria:

- **Nova** pre správu výpočtových zdrojov,
- **Neutron** v kombinácii s **OVN** pre softvérovo definovanú sieťovú architektúru,
- **Glance** pre správu obrazov virtuálnych strojov,
- **Horizon** ako webové administračné rozhranie,
- **Keystone** pre autentifikáciu a správu identít,
- **Cinder** pre blokové úložiská,
- **Heat** pre orchestráciu a deklaratívne nasadzovanie zdrojov,
- **Vault** pre bezpečnú správu tajomstiev a kryptografických materiálov.

Úložná vrstva je realizovaná prostredníctvom **Ceph clusteru**, ktorý poskytuje distribuované a vysoko dostupné úložisko pre potreby OpenStack služieb aj aplikačných workloadov. Integrácia Cephu umožňuje oddelenie výpočtovej a úložnej vrstvy, horizontálne škálovanie kapacity a zvýšenú odolnosť voči výpadkom jednotlivých uzlov, čo je výhodné najmä pre dátovo náročné SOC komponenty.

Takto navrhnutý automatizovaný model nasadenia umožňuje udržiavať cloudovú platformu v konzistentnom stave, zjednodušuje správu jej životného cyklu a vytvára stabilný základ pre experimentálne aj dlhodobejšie nasadenie SOC architektúry v prostredí INNOSOC.

13.1.4 Infraštruktúrna vrstva prostredia

Infraštruktúrna vrstva prostredia OpenStack KIS FRI UNIZA predstavuje technický základ univerzitného cloudu, na ktorom sú prevádzkované výučbové, výskumné aj experimentálne služby. V rámci tejto práce je infraštruktúra chápaná ako podporná platforma pre nasadenie a testovanie komponentov navrhovaného SOC riešenia. Nejde teda o detailný popis prevádzkovej topológie, ale o vysvetlenie princípov, ktoré sú dôležité pre stabilitu, izoláciu a replikovateľnosť PoC nasadenia.

Základ infraštruktúry tvoria fyzické výpočtové uzly, sieťové prvky a úložné systémy. Nad nimi je prevádzkovaná virtualizačná vrstva a riadiaca vrstva OpenStacku, ktorá zabezpečuje alokáciu zdrojov, správu virtuálnych sietí a kontrolu prístupov. V tejto vrstve je možné vytvárať viacužívateľské (multi-tenant) prostredie s logickou separáciou projektov, čo je vhodné pre paralelné experimenty a zároveň pre oddelenie služieb s rôznymi nárokmi na bezpečnosť a dostupnosť. Z architektonického hľadiska je infraštruktúra navrhnutá tak, aby:

- umožňovala oddelenie výpočtových, sieťových a úložných zdrojov,
- podporovala viacpoužívateľské (multi-tenant) prostredie,

- poskytovala izoláciu jednotlivých projektov a experimentov,
- umožňovala flexibilné prideľovanie zdrojov podľa potrieb služieb.

Pre potreby tejto práce je infraštruktúra využívaná na prevádzku virtuálnych strojov, v ktorých sú nasadené jednotlivé komponenty SOC (SIEM, SOAR, CTI, podporné nástroje). Tieto komponenty sú rozdelené do samostatných inštancií, čo umožňuje:

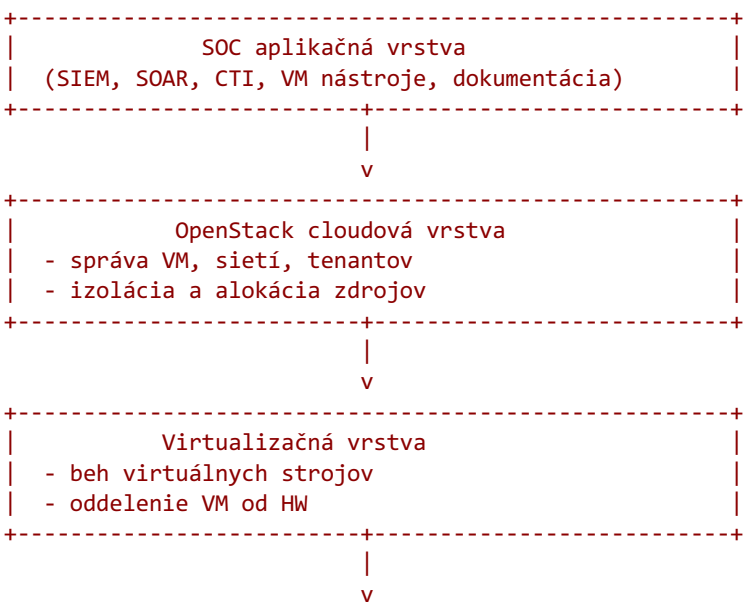
- izoláciu jednotlivých funkčných blokov,
- samostatnú správu a obnovu služieb,
- experimentovanie s konfiguráciou bez dopadu na ostatné systémy.

Z pohľadu prevádzky je infraštruktúra navrhnutá ako dlhodobo dostupná a stabilná platforma, ktorá umožňuje nepretržitú prevádzku cloudových služieb a zároveň podporuje zmeny konfigurácie, rozširovanie kapacity a testovanie nových riešení. Detaily implementácie sú zvolené tak, aby boli v súlade s bežnými architektonickými princípmi OpenStack prostredí používaných v akademických inštitúciách.

Architektúra infraštruktúrnej vrstvy

Architektúra infraštruktúrnej vrstvy vychádza zo štandardného referenčného modelu OpenStack prostredí, v ktorom sú jasne oddelené jednotlivé vrstvy infraštruktúry a ich zodpovednosti. Architektúra infraštruktúrnej vrstvy ako viacúrovňový model umožňuje nezávislý rozvoj infraštruktúry a nadstavbových služieb. Základné vrstvy architektúry sú:

- Fyzická vrstva – výpočtové uzly, sieťové prepínače a úložné systémy,
- Virtualizačná vrstva – zabezpečuje beh virtuálnych strojov a izoláciu zdrojov,
- Cloudová riadiaca vrstva (OpenStack) – riadi alokáciu zdrojov, sieťové segmenty a prístupové práva,
- Aplikačná vrstva – SOC komponenty prevádzkované nad OpenStackom.



```
+-----------------------------------------------------+
|          Fyzická infraštruktúra                     |
|  - výpočtové uzly                                   |
|  - sieťové prvky                                    |
|  - úložiská                                         |
+-----------------------------------------------------+
```

Z hľadiska návrhu je infraštruktúra dimenzovaná tak, aby podporovala:

- oddelenie riadiacej a dátovej roviny,
- škálovanie výpočtových uzlov bez zásahu do aplikačnej vrstvy,
- oddelené sieťové segmenty pre manažment, služby a dátovú prevádzku.

Sieťová architektúra a odporúčania

Aj keď konkrétna sieťová topológia a adresný plán nie sú predmetom tejto práce, pre bezpečné a stabilné fungovanie cloudového prostredia aj SOC komponentov je vhodné dodržať základné princípy segmentácie. Z architektonického hľadiska sa odporúča oddeliť minimálne tri typy prevádzky: manažment, tenant (prevádzka virtuálnych strojov) a storage komunikáciu. Takéto rozdelenie znižuje riziko bočného pohybu v prípade kompromitácie jednej služby, zjednodušuje monitoring a umožňuje presnejšie aplikovať bezpečnostné politiky.

Z architektonického hľadiska sa odporúča:

- vyhradená VLAN pre manažment OpenStack komponentov,
- samostatná VLAN pre tenant traffic (VM siete),
- oddelená VLAN pre storage prevádzku, najmä ak je využívané distribuované alebo sieťové úložisko,
- obmedzenie prístupu medzi VLAN prostredníctvom firewall pravidiel.

Takéto rozdelenie zvyšuje bezpečnosť, zjednodušuje monitoring a znižuje riziko bočného pohybu útočníka v prípade kompromitácie služby.

Úložná architektúra a odporúčania

Úložná vrstva infraštruktúry je významná najmä pre SOC komponenty spracúvajúce veľké objemy dát a vyžadujúce stabilnú perzistenciu, ako sú logové úložiská, indexačné databázy alebo výstupy zo skenovania zraniteľností. OpenStack umožňuje abstrahovať rôzne typy úložísk do jednotného rozhrania, čo zjednodušuje správu a umožňuje prispôsobiť úložisko typu služby.

Z architektonického hľadiska je vhodné rozlišovať systémové úložisko pre operačné systémy virtuálnych strojov, dátové úložisko pre perzistentné dáta SOC nástrojov a dočasné úložisko pre spracovanie a cache. Pre databázové a indexačné služby je preferované rýchle úložisko (ideálne SSD), keďže výkon diskového subsystému zásadne ovplyvňuje rýchlosť vyhľadávania, korelácie a tvorby reportov. Zároveň je potrebné myslieť na pravidelné zálohovanie kritických dát a obmedziť priamy prístup k storage sieti len na komponenty, ktoré ho skutočne potrebujú

V prostrediach založených na OpenStacke je ako vhodná technológia úložnej vrstvy často využívaný distribuovaný úložný systém **Ceph**, ktorý poskytuje objektové, blokové aj súborové úložisko v rámci jednotnej platformy. Ceph je navrhnutý ako škálovateľné a vysoko dostupné riešenie, ktoré umožňuje rozloženie dát naprieč viacerými uzlami a tým zvyšuje odolnosť voči výpadkom jednotlivých komponentov. V kombinácii s OpenStackom je Ceph bežne využívaný ako backend pre blokové úložisko virtuálnych strojov alebo ako dátové úložisko pre aplikácie s vysokými nárokmi na kapacitu a dostupnosť.

Pre SOC workloady, ako sú SIEM systémy alebo zraniteľnostné skenery, je výhodou Ceph najmä schopnosť horizontálneho škálovania a oddelenia výpočtovej a úložnej kapacity. Zároveň je však potrebné zohľadniť výkonové charakteristiky – databázové a indexačné služby môžu výrazne profitovať z rýchleho úložiska (napr. SSD), zatiaľ čo menej kritické alebo archivačné dáta je možné umiestniť na kapacitne orientované úložisko.

Z hľadiska odporúčaní je vhodné:

- oddeliť systémové a dátové disky virtuálnych strojov,
- preferovať rýchle úložisko pre databázové a indexačné služby,
- zabezpečiť pravidelné zálohovanie kritických dát,
- obmedziť priamy prístup k storage sieti len na oprávnené komponenty infraštruktúry.

Takto navrhnutá úložná architektúra poskytuje vyvážený kompromis medzi výkonom, dostupnosťou a flexibilitou, čo je vhodné pre experimentálne aj dlhodobejšie PoC nasadenie SOC riešenia v akademickom cloudovom prostredí.

13.1.5 Hardvérové komponenty prostredia

Hardvérové komponenty prostredia predstavujú fyzický základ cloudovej infraštruktúry OpenStack KIS FRI UNIZA, na ktorom je prevádzkované experimentálne SOC riešenie. Táto vrstva zabezpečuje výpočtový výkon, sieťovú konektivitu a úložnú kapacitu potrebnú na beh virtualizačnej a cloudovej riadiacej vrstvy.

Infraštruktúra je tvorená kombináciou riadiacich a výpočtových serverov, ktoré spolu vytvárajú flexibilnú a škálovateľnú platformu pre cloudové služby. Riadiace servery zabezpečujú prevádzku riadiacich komponentov virtualizačnej a cloudovej vrstvy, zatiaľ čo výpočtové uzly sú určené primárne na beh virtuálnych strojov a aplikačných workloadov, vrátane SOC komponentov.

Výpočtové uzly tvoria hlavný zdroj výpočtovej kapacity cloudového prostredia. Sú vybavené viacjadrovými procesormi, veľkou kapacitou operačnej pamäte a lokálnym diskovým priestorom, ktorý slúži pre systémové disky virtuálnych strojov alebo ako súčasť úložnej architektúry. Takáto konfigurácia umožňuje paralelný beh viacerých virtuálnych strojov s rôznymi nárokmi na výkon, čo je nevyhnutné pre prevádzku heterogénnych SOC komponentov.

Z pohľadu experimentálneho SOC riešenia je dôležité, že výpočtová kapacita umožňuje oddeliť jednotlivé funkčné bloky do samostatných virtuálnych strojov, napríklad SIEM, SOAR, CTI platformy, zraniteľnostné skenery a podporné služby. Tento prístup zvyšuje stabilitu prostredia, uľahčuje správu a umožňuje nezávislé škálovanie alebo reštart jednotlivých komponentov bez dopadu na ostatné časti riešenia.

Aktuálne funkčné HW vybavenie používa:

Riadiace servery:

- 2ks Cisco UCS C220M3: 2x Xeon E5-2667v2 8c/16t, 128GB RAM, 500GB SSD, 600GB HDD, 2x 10GbE, VMWare ESXi, XCP-ng

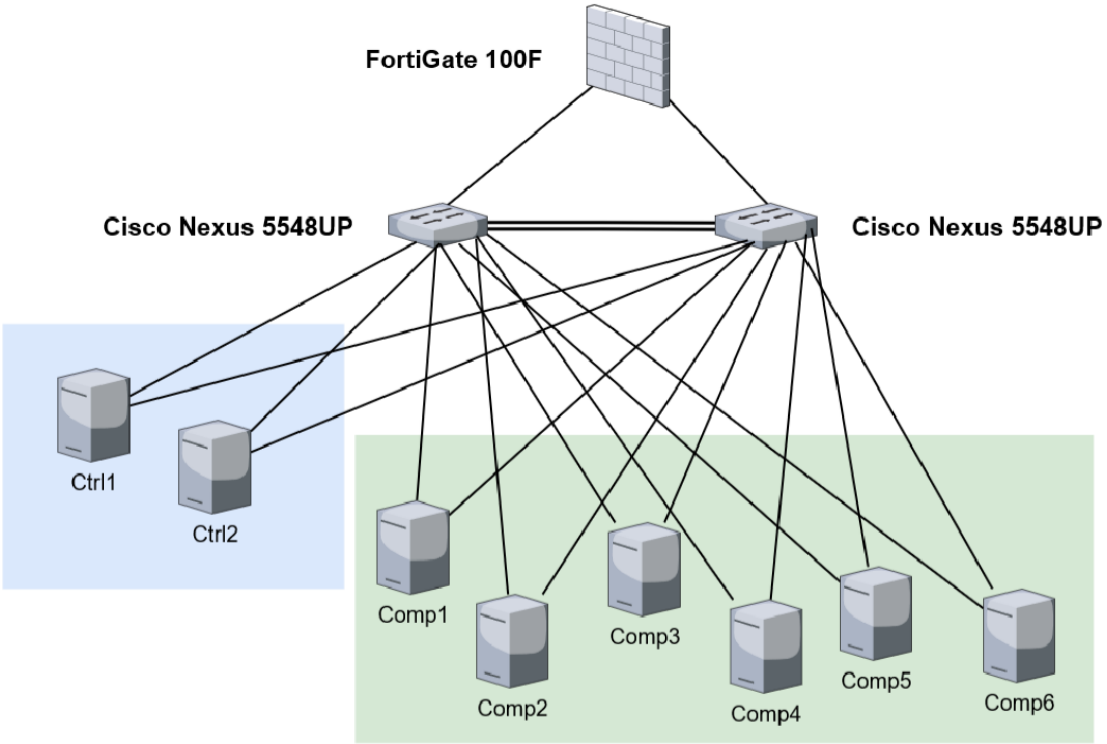
Výpočtové servery:

- 6ks Lenovo x3650M5: 2x Xeon E5-2699v3 18c/36t, (2 servery 2x Xeon E5-2680v414c/28t), 512GB RAM, Systém: 300GB HDD, Storage: 4x 1,2TB HDD, 2x 10GbE

Sieťové prvky a infraštruktúra

Sieťová infraštruktúra zabezpečuje prepojenie výpočtových uzlov, riadiacich serverov a úložných systémov do jednotného funkčného celku. Je navrhnutá tak, aby podporovala vysokú priepustnosť, nízku latenciu a segmentáciu prevádzky, čo je nevyhnutné pre cloudové a bezpečnostné workloady.

Podpora technológií ako VLAN a vysokorýchlostné sieťové rozhrania umožňuje logické oddelenie jednotlivých typov prevádzky, napríklad manažmentu, tenant traffic a storage komunikácie. Z architektonického hľadiska je táto separácia kľúčová pre bezpečnosť a prevádzkovú stabilitu, najmä v prostredí, kde sú paralelne prevádzkované výučbové, výskumné a experimentálne služby.



Úložné systémy

Úložné systémy v CC infraštruktúre poskytujú perzistentný priestor pre operačné systémy virtuálnych strojov aj pre aplikačné dáta. Pre SOC komponenty je úložná vrstva obzvlášť dôležitá, keďže spracúvajú veľké objemy dát a často vyžadujú dlhodobú archiváciu.

Architektúra úložiska je navrhnutá tak, aby umožňovala oddelenie systémových a dátových diskov a podporovala nasadenie distribuovaných úložných riešení. Takýto prístup je kompatibilný s technológiami ako Ceph, ktoré sú bežne využívané v OpenStack prostrediach a umožňujú škálovanie kapacity a zvýšenie odolnosti voči výpadkom. Pre výkonovo citlivé služby, ako sú databázy a indexačné mechanizmy SIEM nástrojov, je preferované rýchle úložisko, zatiaľ čo menej kritické dáta môžu byť umiestnené na kapacitne orientovaných diskoch.

Zhrnutie kapacity pre PoC

Z pohľadu Proof of Concept poskytuje hardvérová infraštruktúra KIS FRI UNIZA dostatočnú výpočtovú, sieťovú aj úložnú kapacitu na overenie funkčnosti navrhovanej SOC architektúry. Prostredie umožňuje paralelný beh viacerých SOC komponentov, experimentovanie s ich konfiguráciou a testovanie integračných väzieb v podmienkach blízkych reálnej prevádzke.

Dôležité je, že infraštruktúra nie je optimalizovaná na maximálny výkon jedného konkrétneho nástroja, ale na flexibilitu a stabilitu celého prostredia. To zodpovedá cieľom práce, ktorými sú overenie architektonického návrhu, integračných scenárov a prevádzkových vlastností SOC riešenia v akademickom cloudovom prostredí.

13.1.6 Umiestnenie a izolácia SOC komponentov v prostredí

Umiestnenie a izolácia jednotlivých komponentov SOC architektúry predstavujú kľúčový aspekt návrhu experimentálneho riešenia. V prostredí KIS FRI UNIZA je cieľom zabezpečiť, aby jednotlivé nástroje mohli byť nasadené, konfigurované a testované nezávisle od seba, bez vzájomného negatívneho ovplyvňovania a bez dopadu na ostatné cloudové služby. Tento prístup zároveň umožňuje realistické overenie architektúry SOC v podmienkach blízkych reálnej prevádzke.

Základným princípom návrhu je použitie **virtuálneho stroja ako primárnej jednotky nasadenia**. Každý hlavný komponent SOC riešenia je prevádzkovaný vo vlastnej virtuálnej inštancii, čím je dosiahnutá jasná izolácia na úrovni operačného systému, sieťových rozhraní a pridelených zdrojov. Tento model zodpovedá bežnej praxi v produkčných SOC prostrediach, kde sú kritické bezpečnostné služby oddelené z dôvodu stability, bezpečnosti a jednoduchej správy.

Virtuálne stroje sú vytvárané a spravované prostredníctvom OpenStack platformy, ktorá zabezpečuje izoláciu tenantov, kontrolu prístupových práv a flexibilné prideľovanie výpočtových a sieťových zdrojov. V PoC režime to umožňuje rýchle vytváranie nových inštancií, zmeny konfigurácie alebo úplné odstránenie experimentálneho prostredia bez zásahu do fyzickej infraštruktúry

Model nasadenia SOC komponentov (VM ako základné jednotky)

V rámci navrhovaného riešenia sú jednotlivé SOC komponenty rozdelené do samostatných virtuálnych strojov podľa ich funkčnej roly. Typicky ide o oddelené inštancie pre detekčné nástroje (napr. SIEM), reakčné a automatizačné nástroje (SOAR), platformy threat intelligence (CTI), nástroje pre správu aktív a kontextu infraštruktúry, ako aj podporné služby, napríklad znalostné bázy alebo zraniteľnostné skenery.

Takéto rozdelenie umožňuje nielen logickú separáciu rolí, ale aj nezávislé dimenzovanie zdrojov podľa charakteru workloadu. Napríklad SIEM komponenty môžu vyžadovať viac pamäte a diskového výkonu, zatiaľ čo SOAR alebo CTI platformy majú nižšie nároky na výpočtový výkon, ale vyššie nároky na dostupnosť a integráciu s externými systémami.

Kontajnerizácia v rámci VM (Docker/Compose) – kde dáva zmysel a prečo

Vo vnútri jednotlivých virtuálnych strojov je v odôvodnených prípadoch využívaná kontajnerizácia prostredníctvom technológií Docker a Docker Compose. Tento prístup umožňuje jednoduchšie nasadenie komplexných aplikácií, zjednodušenie aktualizácií a lepšiu kontrolu nad závislosťami jednotlivých komponentov.

Kontajnerizácia však nie je použitá ako náhrada virtualizácie, ale ako jej doplnok. Virtuálny stroj poskytuje bezpečnostnú a izolačnú hranicu medzi jednotlivými SOC nástrojmi, zatiaľ čo kontajnery slúžia na efektívnu správu aplikačných komponentov v rámci jednej služby. Tento dvojvrstvový model (VM + Docker) je vhodný najmä pre experimentálne a PoC prostredia, kde je potrebné rýchlo meniť konfigurácie a testovať rôzne varianty nasadenia.

Logická separácia rolí (detekcia / reakcia / CTI / knowledge / vuln scanning)

Logická separácia rolí je realizovaná kombináciou virtualizačnej vrstvy, sieťovej segmentácie a aplikačnej architektúry. Detekčné nástroje, reakčné mechanizmy, platformy threat intelligence, systémy správy aktív a znalostné databázy sú oddelené nielen na úrovni virtuálnych strojov, ale aj prostredníctvom samostatných sieťových segmentov a riadených integračných rozhraní.

Takýto prístup znižuje riziko bočného pohybu v prípade kompromitácie jedného komponentu a zároveň podporuje jasné vymedzenie zodpovedností jednotlivých nástrojov. Každý komponent vystupuje v architektúre SOC ako samostatná služba s presne definovanými vstupmi a výstupmi, čo zjednodušuje integráciu, audit a dlhodobú údržbu riešenia.

PoC režim a experimenty (izolácia tenantov/sietí, replikovateľnosť nasadenia)

Nasadenie SOC architektúry v prostredí KIS FRI UNIZA má charakter Proof of Concept, ktorého cieľom nie je zabezpečiť nepretržitú produkčnú prevádzku, ale overiť funkčnosť architektonického návrhu, integračných väzieb a prevádzkových vlastností jednotlivých nástrojov. Tomu je prispôsobený aj spôsob izolácie a správy prostredia.

PoC režim umožňuje paralelné experimenty, testovanie rôznych konfiguračných scenárov a opakované nasadenie rovnakých architektúrnych vzorov. Vďaka využitiu OpenStacku je možné vytvárať oddelené projekty, samostatné virtuálne siete a izolované inštancie služieb, čo podporuje replikovateľnosť experimentov a objektívne porovnávanie výsledkov.

Takto navrhnuté umiestnenie a izolácia SOC komponentov vytvárajú stabilný základ pre experimentálne overenie navrhovaného riešenia a zároveň poskytujú architektonický rámec, ktorý je prenositeľný aj do produkčných cloudových alebo on-premise prostredí.

13.2 Kubernetes Brainit

13.2.1 Popis infraštruktúry Kubernetes

Kubernetes je platforma s otvoreným zdrojovým kódom určená na automatizované nasadzovanie, škálovanie a správu kontajnerizovaných aplikácií - v praxi sa aj preto najčastejšie stretáva s označením, že ide o aplikačný orchestrátor. Na rozdiel od viac tradičných spôsobov nasadenia kontajnerov v rámci jediného výpočtového servera - najčastejšie pomocou nástroja Docker - ide v prípade technológie Kubernetes o zoskupenie viacerých takýchto výpočtových uzlov pod jednu abstraktnú rovinu.

Tieto uzly tvoria klaster, ktorý zabezpečuje beh aplikácií nielen s vysokou dostupnosťou, ale aj napr. s dynamickým škálovaním na základe aktuálnej záťaže danej aplikácie vytvorením viacerých replík. Aj pokiaľ sa používateľ rozhodne nepoužiť funkcionalitu viacerých replík tej istej aplikácie, Kubernetes umožňuje rýchle zotavenie aplikácie aj v prípade, že pri ľubovoľnom uzle klastra príde ku výpadku - v tom prípade kontroléry Kubernetes uzla zodpovedné za jednotlivé časti nasadenia zabezpečia, že jednotlivé aplikácie budú znovu spustené na stále dostupných uzloch klastra. Kubernetes tak vo výsledku rozširuje možnosti kontajnerizačných démonov o viac komplexnú správu kontajnerovej infraštruktúry.

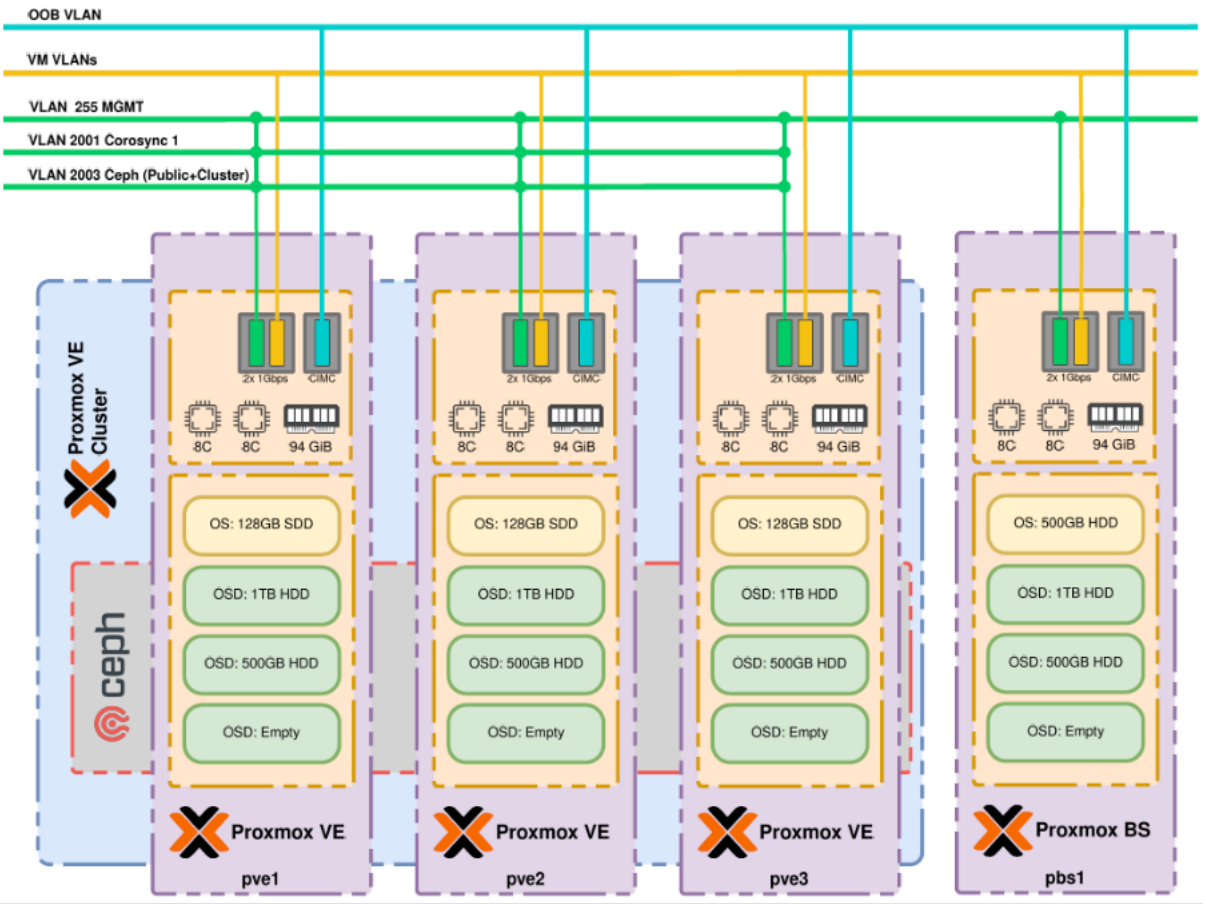
Kubernetes využíva deklaratívny model na definovanie a abstrakciu bežiacich komponentov - túto funkcionalitu zabezpečuje pomocou objektov rôznych typov (v Kubernetes terminológii taktiež \code{kind}), kde každý typ má jasne preddefinovanú štruktúru. Typicky sa tieto objekty definujú v súboroch vo formáte \ac{YAML}, ktoré následne používateľ môže do klastra aplikovať pomocou príkazu \code{kubectl apply --filename file.yaml}.

Ako hypervízor bol zvolený Proxmox Virtual Environment. To si vyžadovalo splnenie hneď niekoľkých úloh:

- aktualizácia firmvéru a ovládačov na všetkých fyzických serveroch a konfigurácia Out-of-band (OOB) manažmentu na vzdialenú správu serverov.
- inštalácia PVE platformy na každý z vybraných serverov
- základná konfigurácia sieťovej konektivity uzlov a príprava jednotlivých podsietí pre potrebné služby:
    - Podsieť určená na manažment PVE serverov.
    - Corosync podsieť (technológia použitá na klastrovanie PVE serverov)
    - Ceph podsieť (technológia zodpovedná za úložisko pre Kubernetes klaster
    - Podsiete určené pre virtuálne stroje (napr. Kubernetes uzly)
- Vytvorenie PVE klastra z vytvorených a nakonfigurovaných serverov.
- Vytvorenie Ceph klastra v rámci PVE uzlov a nasadenie potrebných démonov. Tie zahŕňajú monitorovacie služby (Monitors), riadiace služby (Managers) a jednotlivých úložných démonov (Object Storage Daemon (OSD)), kde každá inštancia OSD je typicky zodpovedná za ukladanie dát na konkrétny priradený disk.
- Nasadenie zálohovacieho systému v podobe Proxmox Backup Server (PBS) na dedikovaný server. PBS je softvérové riešenie určené na efektívne a inkrementálne zálohovanie virtuálnych strojov a kontajnerov bežiacich v PVE prostredí a poskytuje

natívnu integráciu s PVE systémom, prečo aj bolo prirodzenou voľbou pri výbere zálohovacieho systému.

Splnenie všetkých týchto úloh viedlo k vytvoreniu nasledujúcej plne funkčnej infraštruktúry hypervízorov, ktorá mohla byť použitá v ďalších krokoch na implementáciu samotného Kubernetes klastra:



13.2.2 Infraštruktúra virtuálnych strojov – IaaC

Pre nasadenie virtuálnej infraštukrúry strojov, ktoré sú zodpovedné za beh klastra bol zvolený nástroj Terraform od spoločnosti HashiCorp, ktorý umožňuje deklaratívne definovať žiadaný stav infraštruktúry pomocou konfiguračných súborov vo formáte HashiCorp Configuration Language (HCL).

Terraform podporuje široké spektrum poskytovateľov (terminologicky a konfiguračne dostupné ako provider), nad ktorými je schopný žiadané objekty (v tomto prípade primárne virtuálne stroje) nasadiť.

To poskytuje vytvorenému riešeniu jednoduchú rozšíriteľnosť do budúcnosti v prípade, že sa rozhodne pre migráciu z Proxmox-u na iný hypervízor, poprípade o rozšírenie klastra v podobe dodatočných virtuálnych strojov, ktoré nie nutne musia byť súčasťou toho istého klastra hypervízorov.

To taktiež otvára možnosti pre vytvorenie testovacích klastrov aj mimo takéhoto predvoleného riešenia. Pre prácu s nástrojom Terraform bol vytvorený repozitár, v ktorom bol v rámci práce vytvorený vlastný modul (v terminológii Terraform ako module) pre jednoduché vytváranie viacerých replík v podobe rôznych prostredí.

Pre vytvorenie virtuálnych strojov je následne potrebné už iba vyplniť vytvorenú premennú virtual_machines daného modulu, ktorá reprezentuje zoznam špeciálne definovaných objektov, kde každý objekt reprezentuje jeden virtuálny stroj. Pri použití tohto modulu teda konfigurácia nasadenia vyzerá nasledovne:

```
module "infrastructure" {
source = "../../modules/k8s-proxmox"
virtual_machines = [
{
name = "server1.k8s.kis.fri.uniza.sk"
vm_id = 51151
description = "K8S - PROD"
tags = local.tags
node_name = "pve1"
pool_id = local.pool_id
cores = local.server_cpus
memory = local.server_memory
network_bridge = local.network_bridge
boot_order = local.boot_order
cdrom = {
file_id = local.iso
}
disk = [
{
size = local.disk_size
interface = "scsi0"
}
]
datastore_id = "vd"
ipv4_address = "10.23.153.151/24"
ipv4_gateway = local.ipv4_gateway
dns_domain = local.dns_domain
dns_servers = local.dns_servers
},
{
name = "server2.kis.fri.uniza.sk"
```

Kód: Ukážka použitia vytvoreného Terraform modulu

Možno vidieť, že pod sprístupnenou premennou virtual_machines je možné definovať ľubovoľný počet virtuálnych strojov, ktorá zároveň umožňuje definovať ich vlastnosti ako napr. meno (name), unikátny identifikátor v Kubernetes klastri (vm_id), dynamický počet diskov a ich zodpovedajúce veľkosti, sieťové nastavenia a niekoľko ďalších nastavení - podrobný popis je súčasťou dokumentácie daného repozitára.

Pokiaľ ide o premenné local.*, tie boli definované za účelom zníženia duplicitných informácií pri vytváraní viacerých podobných virtuálnych strojov (ako tak tomu pri Kubernetes

klastroch býva). Tie zahŕňajú spoločné informácie pre všetky virtuálne stroje, resp. aj rozdielne informácie na základe úlohy virtuálneho stroja v rámci Kubernetes klastra (control/worker node).

Príkladom je počet pridelenej pamäte alebo virtuálnych procesorových jadier (napr. server_cpus a agent_cpus), avšak tento koncept je aplikovateľný na čokoľvek:

```
locals {
tags = ["terraform", "taloslinux"]
pool_id = "K8S"
server_cpus = 4
server_memory = 8192
agent_cpus = 4
agent_memory = 8192
network_bridge = "vlan51"
ipv4_gateway = "10.23.153.1"
dns_domain = "test.k8s.fri.uniza.sk"
dns_servers = [ "8.8.8.8" ]
iso = "iso:iso/talos-v1.9.5-nocloud-amd64-qemuagent.iso"
disk_size = 50
boot_order = ["scsi0","ide0"]
}
```

Kód: Ukážka použitia vytvoreného Terraform modulu - lokálne premenné

13.2.3 Štruktúra repozitára

Navrhnutá a vytvorená štruktúra repozitára vyzerá nasledovne:

```
terraform/
├─ k8s/
│  ├─ prod/
│  │  ├─ (backend.tf)
│  │  ├─ main.tf
│  │  ├─ provider.tf
│  │  ├─ (terraform.tfvars)
│  │  └─ variables.tf
│  └─ test/
│     └─ ...
├─ modules/k8s-proxmox/
│  ├─ main.tf
│  └─ variables.tf
├─ Makefile
├─ Makefile.vars
└─ README.md
```

Pozn.: Súbory umiestnené v zátvorkách - "()" - nie sú súčasťou životného cyklu Git repozitára, nakoľko obsahujú senzitívne premenné, ktoré je potrebné definovať pri inicializácii lokálneho repozitára.

- modules/ - priečinok obsahujúci moduly. Moduly sú v rámci nástroja Terraform opakovateľne použiteľné zoskupenia viacerých Terraform konfigurácii často reprezentujúce určitú časť infraštruktúry. To umožňuje konfiguráciu definovať raz (taktiež s možnosťou parametrizácie), a následne ju opakovane použiť v rôznych častiach projektu/projektov. Práve na to bol vytvorený modul (priečinok) modules/k8s-proxmox/, ktorý týmto prístupom umožňuje vytvárať rôzne repliky Kubernetes prostredí na testovacie účely.
- k8s/ - obsahuje podpriečinky, ktoré definujú špecifické nasadenie (prostredie) Kubernetes klastra (napr. k8s/prod/ pre produkčné nasadenie alebo k8s-test/ pre testovacie nasadenie).
- k8s/<env>/ - priečinok prostredia (napr. k8s/prod/).
  - backend.tf - definuje to, kde je uchovávaný aktuálny stav infraštruktúry (typicky súbor *.tfstate) 34
  - provider.tf - definuje poskytovateľa (provider), v konkrétnom prípade špecifický Proxmox poskytovateľ
  - variables.tf - deklaruje premenné, ktoré používateľ bude musieť definovať (pomocou súboru terraform.tfvars) po inicializácii lokálneho repozitára. Tieto premenné uchovávajú prihlasovacie údaje do Proxmox klastra, na ktorý majú byť nasadené virtuálne stroje použitého modulu.
  - main.tf - hlavný terraform súbor, ktorý používa vyššie definovaný modul modules/k8s-proxmox

ArgoCD poskytuje prehľadné webové rozhranie (ale aj CLI) na zobrazenie a správu nasadených aplikácií, podporu správy viacerých klastrov naraz, taktiež vďaka čomu je aj často preferovaným nástrojom pri vývoji aplikácií. Disponuje aj množstvom doplnkových modulov poskytujúcich dodatočnú funkcionalitu a napojenie na externé systémy (ako napr. integrácia s HashiCorp Vault pre priame pridanie senzitívnych dát).

FluxCD/**ArgoCD** integrácia - Okrem vyššie spomenutých operátorov je však možné použiť aj rôzne integrácie, ktoré sú zabudované do GitOps nástrojov FluxCD a ArgoCD. ArgoCD ponúka napr. vlastný HashiCorp Vault plugin pre priamu integráciu bez dodatočného operátora, no je možné zapnúť aj dodatočnú podporu pre KSOPS zmenou ArgoCD konfigurácie (nutné nastaviť kustomize.buildOptions s dodatočnými prepínačmi "–enable-alpha-plugins a –enable-exec FluxCD volí naopak vzhľadom aj na jeho architektúru viac minimalistický dizajn a prenecháva túto zodpovednosť zväčša na externé systémy - s jednou výnimkou, nakoľko disponuje natívnou a robustnou podporou pre SOPS.

13.2.4 Správa a nasadzovanie Kubernetes aplikácií

Pre nasadenie dodatočných aplikácií bol ako vhodný zvolený nástroj FluxCD. Pre ten bolo potrebné vytvoriť dodatočný repozitár, ktorý bude obsahovať všetky potrebné YAML manifesty určené pre nasadenie jednotlivých aplikácií. Následne bolo potrebné vytvoriť vhodnú štruktúru, ktorá bude jednoducho rozšíriteľná o ďalšie, nové aplikácie a zároveň umožní jednoduchým a prehľadným spôsobom aplikáciu nielen nasadiť, ale ju aj zo špecifického

klastra vymazať. Rovnako je dôležité aby repozitár umožnil jednoduchú rozšíriteľnosť o nové klastre bez zbytočných duplicít, na čo bol primárne využitý nástroj Kustomize. Cieľom bolo taktiež pri použití tohto nástroja oddeliť verziovanie jednotlivých inštalovaných aplikácii na úrovni klastrov - teda aby bolo možné otestovať aktualizácie jednotlivých aplikácii mimo produkčné nasadenie, a po otestovaní tak bezpečne tieto aktualizácie vykonať aj na produkčnom nasadení.

13.3 Docker Brainit

13.3.1 Popis infraštruktúry docker

Základ infraštruktúry tvorí hypervízor Proxmox VE nasadený priamo na fyzickom serveri (bare-metal). Proxmox využíva kombináciu technológií KVM (Kernel-based Virtual Machine) pre plnú virtualizáciu a LXC pre kontajnerizáciu na úrovni operačného systému. V danom návrhu je použitá výhradne KVM virtualizácia, čím je zabezpečená plná izolácia jednotlivých virtuálnych serverov.

Hypervízor zabezpečuje:

- správu výpočtových zdrojov (CPU, RAM),
- správu diskových úložísk (LVM-thin, ZFS alebo lokálne RAID pole),
- sieťovú virtualizáciu prostredníctvom Linux bridge rozhraní,
- centralizovanú správu a monitoring virtuálnych strojov.

Každý virtuálny server (VM) má pridelené presne definované hardvérové prostriedky, vrátane počtu virtuálnych CPU jadier, veľkosti operačnej pamäte a diskového priestoru. Konfigurácia je navrhnutá tak, aby bolo možné dynamicky škálovať zdroje bez nutnosti reinštalácie operačného systému.

13.3.2 Operačný systém virutálnych serverov

Na jednotlivých virtuálnych serveroch je nasadená distribúcia AlmaLinux, ktorá je binárne kompatibilná s Red Hat Enterprise Linux. Tento operačný systém je zvolený z dôvodu:

- vysokej stability a dlhodobej podpory,
- vhodnosti pre serverové a produkčné nasadenia,
- kompatibility s Docker Engine a modernými DevOps nástrojmi.

Základná konfigurácia AlmaLinux serverov zahŕňa:

- minimal installation (bez grafického rozhrania),
- správu balíkov pomocou DNF,
- centralizované aktualizácie bezpečnostných záplat,
- nastavenie firewallu (firewalld alebo nftables),
- zabezpečený prístup cez SSH s využitím kľúčovej autentifikácie.

Každý virtuálny server je koncipovaný ako aplikačný uzol, na ktorom beží Docker Engine a súvisiace aplikačné kontajnery.

13.3.3 Kontajnerizačná vrstva

Na AlmaLinux virtuálnych serveroch je nainštalovaný Docker Engine, ktorý zabezpečuje izolovaný beh aplikačných komponentov vo forme kontajnerov. Docker poskytuje jednotné prostredie pre nasadenie aplikácií nezávislé od hostiteľského systému.

Docker Engine zabezpečuje:

- správu životného cyklu kontajnerov,
- sieťovanie medzi kontajnermi pomocou virtuálnych Docker sietí,
- perzistenciu dát prostredníctvom Docker volumes,
- izoláciu procesov, súborového systému a sieťových rozhraní.

Kontajnery sú navrhnuté tak, aby každý poskytoval jednu konkrétnu funkciu (napr. databáza, backendová služba, frontendová aplikácia), čím sa dodržiava princíp mikroservisnej architektúry.

13.3.4 Orchestrácia aplikácii

Spúšťanie a konfigurácia kontajnerov je realizovaná pomocou Docker Compose súborov (docker-compose.yml). Docker Compose umožňuje deklaratívne definovať:

- jednotlivé služby (services),
- použité Docker obrazy,
- sieťové prepojenia medzi službami,
- mapovanie portov,
- perzistentné úložiská,
- konfiguračné premenné prostredia.

Docker Compose súbory sú uložené priamo na virtuálnych serveroch a umožňujú:

- jednoduché nasadenie celého aplikačného stacku jedným príkazom,
- reprodukovateľnosť prostredia (test, staging, produkcia),
- jednoduchú údržbu a aktualizáciu služieb.

Každý aplikačný celok je oddelený do samostatného Compose projektu, čím sa minimalizuje riziko konfliktov medzi jednotlivými nástrojmi a službami.

13.3.5 Celková architektúra riešenia

Celá infraštruktúra je navrhnutá ako viacvrstvová architektúra:

- fyzická vrstva (hardvér),
- virtualizačná vrstva (Proxmox VE),
- operačný systém (AlmaLinux VM),
- kontajnerizačná vrstva (Docker Engine),
- aplikačná vrstva (Docker Compose riadené služby).

Takýto prístup zabezpečuje vysokú mieru modularity, škálovateľnosti a jednoduchej správy infraštruktúry. Zároveň umožňuje efektívne oddelenie infraštruktúrnych, systémových a aplikačných zodpovedností, čo je vhodné pre moderné cloud-ready a DevOps orientované prostredia.

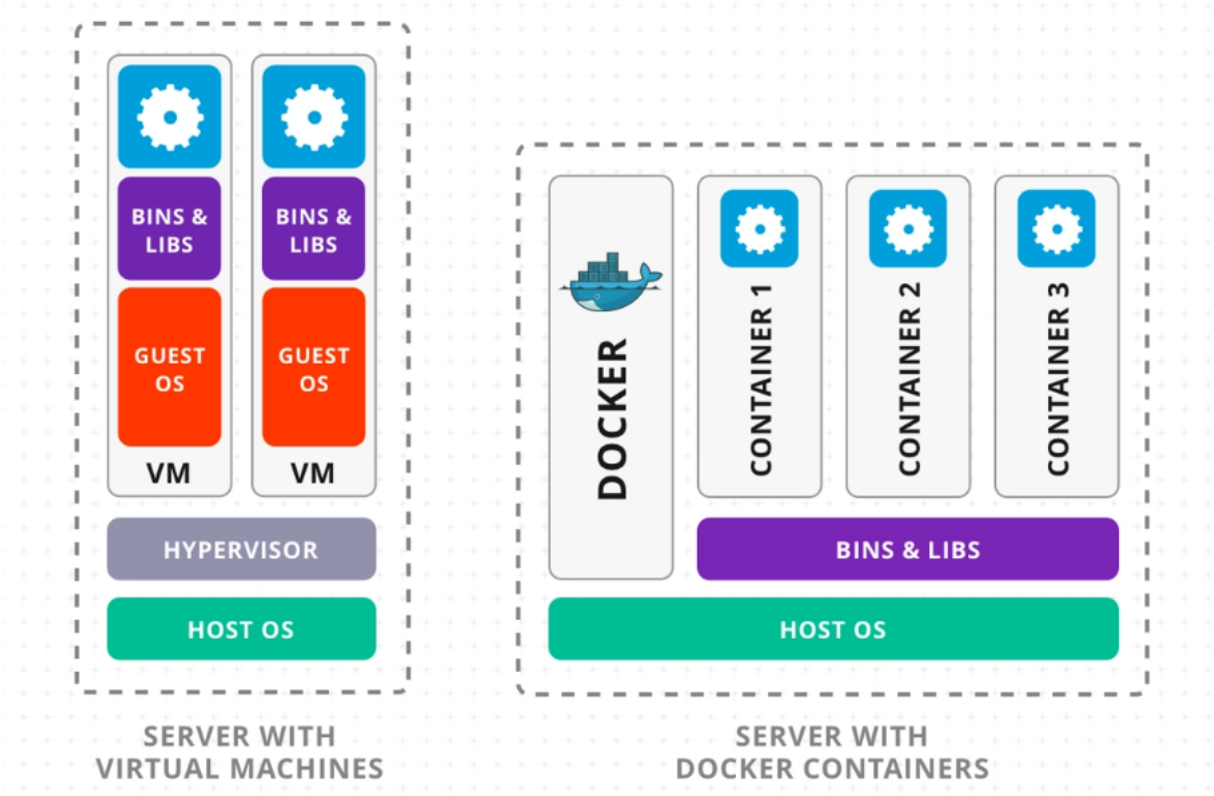


Diagram znázorňujúci topológiu docker infraštruktúry

ZÁVER

Tento dokument predstavuje technickú dokumentáciu funkčného PoC nasadenia multitenantného SOC ekosystému, v ktorom sú jednotlivé bezpečnostné a podporné komponenty nasadené, konfigurované a popísané spôsobom umožňujúcim opakovateľnú reprodukciu prostredia. Štruktúra dokumentu je postavená tak, aby sa pri každom nástroji konzistentne pokryli kľúčové prevádzkové aspekty: opis účelu komponentu v SOC, referencie na dokumentáciu, hardvérové a systémové požiadavky, architektúra so schémou, postup nasadenia (typicky Docker, pri vybraných aj OpenStack), základná konfigurácia a integračné väzby do zvyšku architektúry. Tento prístup zabezpečuje, že výsledkom nie je iba „zoznam nástrojov“, ale ucelený infraštruktúrny popis toho, ako sa z vybraných open-source prvkov skladá prevádzkovateľná SOC vrstva v PoC režime.

Technická línia dokumentu začína kapitolou venovanou ELK Stacku, ktorý v návrhu plní úlohu centrálneho bodu pre zber, ukladanie a analytické spracovanie udalostí. Kapitola je spracovaná vo forme kompletného nasadzovacieho „runbooku“ – od opisu a požiadaviek až po Docker inštaláciu, konfiguráciu a integrácie, čím definuje SIEM/log-management základ, na ktorý nadväzujú ďalšie vrstvy (detekcia, reakcia, evidencia). Na túto vrstvu nadväzuje kapitola o Shuffle, kde je SOC rozšírený o SOAR schopnosti, teda o orchestráciu a automatizáciu reakcií prostredníctvom playbookov. Text explicitne pracuje s tým, že Shuffle je v PoC nasadený vo variantoch Docker aj OpenStack a obsahuje aj časť venovanú pripojeniam na externé aplikácie (napr. Wazuh a Greenbone), definícii vstupov/výstupov a vysvetleniu jeho pozície v SOC.

V ďalšom slede je rozpracovaný Wazuh ako endpointový zdroj detekcií (EDR/SIEM/endpoint security), pričom dokument popisuje architektúru so schémou a spôsob, akým sa Wazuh detekcie integrujú do centrálnej analytickej vrstvy a následne využívajú na reakciu a incident management. Technicky je dôležité, že Wazuh je prezentovaný ako komponent vypĺňajúci priestor medzi surovým zberom logov a vyššou koreláciou – teda ako systematický zdroj endpointových alertov s následným prepojením do ELK/Kibana a na ďalšie SOC nástroje. Paralelne s detekčnou vrstvou dokument pokrýva aj prevenciu cez Greenbone Community Edition, ktorá je spracovaná ako komponent pre vulnerability management – s dôrazom na nasadenie v Dockeri, požiadavky, architektúru a základnú konfiguráciu pre skenovanie a reporting zraniteľností.

Osobitná časť dokumentu je venovaná CTI vrstve, kde sú spracované dva dopĺňajúce sa prístupy. OpenCTI je popísané ako znalostná a kontextová platforma (vrátane konfiguračných krokov ako MITRE ATT&CK a vybrané feedy) s nasadením v Dockeri aj OpenStacku, a so samostatne pomenovanou časťou, ktorá vysvetľuje integračný model CTI v SOC.

Následná kapitola MISP rozpracúva operatívnejšie zdieľanie indikátorov a integračné väzby na ostatné SOC komponenty, pričom dokument obsahuje aj explicitné porovnanie OpenCTI vs. MISP a vysvetlenie, ako môže kombinované nasadenie rozdeliť úlohy medzi „rýchly zdroj indikátorov“ a „centrálnu znalostnú bázu hrozieb“.

V rovine podpory procesov a kontextu infraštruktúry dokument ďalej prechádza na GLPI (asset management/ITSM/ticketing) a NetBox (IPAM/DCIM/asset modelovanie). GLPI je popísané ako procesne orientovaná vrstva, ktorá mapuje technické bezpečnostné výstupy na aktíva, vlastníkov a služby a zabezpečuje auditovateľné a riadené spracovanie incidentov,

pričom sú uvedené aj integračné možnosti a príklady konfigurácií (napr. autentifikácia/oznamovanie, plugin ekosystém). NetBox je spracovaný detailne aj experimentálnou časťou (schémy, nasadenie Docker/OpenStack), pričom sa kladie dôraz na to, že NetBox poskytuje technický zdroj pravdy o infraštruktúre a je využiteľný najmä v enrichment fáze – teda ako zdroj presného kontextu (mapovanie IP → zariadenie, lokalita, segment/VLAN, rola zariadenia), ktorý znižuje nejednoznačnosť pri triage a podporuje automatizáciu bez toho, aby NetBox vykonával aktívne zásahy. Súčasťou je aj technicky formulované porovnanie NetBox vs. GLPI, ktoré dokumentuje, že oba nástroje pokrývajú evidenciu aktív z odlišnej perspektívy (technická topológia vs. procesná/organizačná CMDB/ITSM vrstva) a prirodzene sa dopĺňajú.

Incidentová a evidenčná časť je pokrytá dvoma kapitolami venovanými DFIR-IRIS – najskôr v režime ticketingu a následne v režime evidencie a správy incidentov počas celého životného cyklu. Dokument pri DFIR-IRIS zdôrazňuje case-based prístup (incidentový prípad ako hlavný objekt), prácu s úlohami, auditovateľnosť a možnosť viazať na incident technické artefakty, IOC, poznámky a časovú os vyšetrovania; zároveň sú popísané vstupy/výstupy a integračný charakter voči SIEM/CTI/SOAR nástrojom (automatizovaný príjem alertov, obohatenie incidentu kontextom, export a zdieľanie výsledkov vyšetrovania).

Dokument dopĺňa aj samostatnú kapitolu o Kibane (ELK), ktorá technicky oddeľuje vizualizačnú a analytickú vrstvu od samotného „stacku“ a popisuje jej požiadavky, architektúru, Docker inštaláciu a integračné väzby.

Znalostná a dokumentačná rovina je uzatvorená kapitolou o Wiki.js, ktorá je spracovaná ako interná knowledge base s uvedeným docker-compose nasadením, základnou konfiguráciou (napr. SMTP) a integračnými možnosťami (napr. API kľúče).

Významným uzatvárajúcim prvkom je kapitola Prostredie nasadenia, ktorá dáva celej implementácii infraštruktúrny kontext. Dokument tu technicky rámcuje, že nasadenie v prostredí OpenStack KIS FRI UNIZA má charakter Proof of Concept: cieľom je overenie architektonického návrhu, integračných väzieb a prevádzkových vlastností nástrojov, nie nepretržitá produkčná prevádzka. Súčasne je popísaná logická separácia rolí (detekcia/reakcia/CTI/knowledge/vuln scanning) cez kombináciu virtualizačnej vrstvy, sieťovej segmentácie a riadených integračných rozhraní, pričom OpenStack umožňuje izoláciu tenantov, oddelené siete a replikovateľnosť experimentov. Kapitola zároveň rozširuje pohľad o alternatívne prístupy cez Kubernetes Brainit (orchestrácia kontajnerizovaných aplikácií, infra popis) a Docker Brainit (kontajnerizačná vrstva a compose-orientované nasadenie), čím dokument pokrýva nielen konkrétne PoC nasadenie, ale aj uvažované infraštruktúrne varianty, ktoré majú rovnaký cieľ: izolovane a opakovateľne prevádzkovať SOC komponenty ako samostatné služby.

Ako celok dokument technicky preukazuje, že navrhnutý SOC ekosystém je realizovateľný ako modulárna architektúra, kde sú jasne definované role jednotlivých komponentov a ich integračné hranice. Z pohľadu prevádzky je kľúčové, že je v texte popísaný aj zamýšľaný tok spracovania udalostí naprieč vrstvami (detekcia v ELK, spúšťanie reakcie v Shuffle, enrichment cez NetBox a ďalšie zdroje, následná akcia a auditovateľný výstup), čo spája izolované „návody na inštaláciu“ do jedného prevádzkovo čitateľného SOC scenára.