

Лабораторная работа №5

Тема: Автоматизация сбора и формирования правил Suricata из внешних источников

Теория

Threat Intelligence и публичные источники данных

Threat Intelligence (TI) — это информация о киберугрозах, индикаторах компрометации (IoC), тактиках, техниках и процедурах (TTP) злоумышленников. Публичные источники threat intelligence позволяют организациям:

- Получать актуальные списки вредоносных IP-адресов, доменов, URL
- Загружать готовые правила детектирования для IDS/IPS
- Автоматизировать обновление защитных механизмов
- Снижать время реакции на новые угрозы

Типы источников threat intelligence

1. Готовые наборы правил Suricata

Профессионально разработанные правила для детектирования атак:

- **PT Security Rules** (<https://rules.ptsecurity.com/>) — правила от PT Expert Security Center, фокус на APT и продвинутые угрозы
- **Emerging Threats Open** (<https://rules.emergingthreats.net/open/suricata/>) — один из крупнейших открытых наборов правил

2. IP Blocklists (черные списки IP-адресов)

Списки IP-адресов, связанных с вредоносной активностью:

- **Feodo Tracker** (от abuse.ch) — botnet C&C серверы (Dridex, Emotet, TrickBot, QakBot, BazarLoader)
- **SSLBL** (от abuse.ch) — IP-адреса с вредоносными SSL-сертификатами
- **URLhaus** (от abuse.ch) — URL и IP-адреса для распространения malware
- **Botvrij.eu** — IoC список от голландской инициативы по борьбе с ботнетами

3. Списки IP-адресов сервисов

Официальные диапазоны IP-адресов облачных провайдеров и популярных сервисов:

- **Cloud Providers** — Google Cloud, AWS, Azure, Cloudflare, DigitalOcean, Facebook, Twitter
- Источники: <https://github.com/lord-alfred/ipranges>, <https://cloud-ip-ranges.com>, <https://github.com/rezmoss/cloud-provider-ip-addresses>

4. Специфичные для России источники

- **Antifilter** (<https://antifilter.download>, <https://antifilter.network>) — списки IP из реестра РКН
- **Zapret-info** (<https://github.com/zapret-info/z-i>) — реестр запрещенных ресурсов РФ
- **Роскомсвобода** (<https://reestr.rublacklist.net/>) — есть API ❤️

Управление правилами в Suricata

В предыдущих лабораторных работах правила вносились вручную в файл `local.rules` — это удобно для обучения, но неприменимо в реальных средах. В промышленной эксплуатации управление ruleset'ом выносится в отдельный процесс, а правила становятся артефактом, который проходит этапы проверки и тестирования перед применением.

Инструмент `suricata-update`

`suricata-update` — стандартный инструмент экосистемы Suricata для загрузки и обновления правил из различных источников. Инструмент умеет скачивать правила, управлять категориями, применять локальные модификации и формировать итоговый файл `suricata.rules`.

DevOps-подход к управлению правилами

Вместо ручного редактирования правил DevOps-инженеры строят pipeline:

- **fetch**: загрузка внешних фидов и правил
- **generate**: генерация собственных `.rules` файлов из IoC-списков
- **test**: проверка конфигурации через `suricata -T`
- **deploy**: применение правил при успешном teste

Этот подход гарантирует, что невалидные правила никогда не попадут в production, а процесс обновления полностью автоматизирован и воспроизводим.

Практическая часть

1. Подготовка лабораторного стенда

Используется хост Ubuntu, на котором установлена Suricata в IPS/IDS режиме по инструкции из ЛР1.

2. Установка и настройка suricata-update

Проверьте версию suricata-update:

```
suricata-update --version
```

Обновите индекс источников правил:

```
sudo suricata-update update-sources
```

Просмотрите доступные источники:

```
suricata-update list-sources | head -30
```

Вы увидите множество источников, включая:

- et/open — Emerging Threats Open
- abuse.ch/feodotracker — Feodo Tracker Botnet C2 IP ruleset
- abuse.ch/sslbl-blacklist — SSL Blacklist
- abuse.ch/sslbl-ja3 — JA3 fingerprints

Убедитесь, что в `/etc/suricata/suricata.yaml` в секции `rule-files` указаны базовые файлы:

```
rule-files:  
  - /var/lib/suricata/rules/suricata.rules  
  - /etc/suricata/rules/local.rules
```

3. Подключение Emerging Threats Open

Проверьте, включен ли ET Open:

```
sudo suricata-update list-enabled-sources
```

Если не включен, включите:

```
sudo suricata-update enable-source et/open
```

Загрузите правила:

```
sudo suricata-update
```

Проверьте результат:

```
ls -lh /var/lib/suricata/rules/suricata.rules  
wc -l /var/lib/suricata/rules/suricata.rules
```

Должно быть загружено ~62,000+ правил.

Проверьте конфигурацию:

```
sudo suricata -T -c /etc/suricata/suricata.yaml
```

Если тест успешен, перезапустите Suricata:

```
sudo systemctl restart suricata  
sudo systemctl status suricata --no-pager
```

4. Скрипт автоматизации обновления правил

Создайте каталог и скрипт:

```
sudo mkdir -p /opt/suricata-tools  
sudo nano /opt/suricata-tools/update_rules.sh
```

Содержимое скрипта:

```
#!/usr/bin/env bash  
set -euo pipefail  
  
LOG_FILE="/var/log/suricata/update_rules.log"  
DATE_NOW="$(date -Iseconds)"  
  
echo "[${DATE_NOW}] === Starting Suricata rules update ===" | tee -a  
"${LOG_FILE}"  
  
echo "[${DATE_NOW}] [1/3] Fetch + merge rules via suricata-update..." | tee -a
```

```

"$LOG_FILE"
if ! suricata-update >>"$LOG_FILE" 2>&1; then
    echo "[${DATE_NOW}] ERROR: suricata-update failed, aborting." | tee -a
"$LOG_FILE"
    exit 1
fi

echo "[${DATE_NOW}] [2/3] Test Suricata configuration..." | tee -a "$LOG_FILE"
if ! suricata -T -c /etc/suricata/suricata.yaml >>"$LOG_FILE" 2>&1; then
    echo "[${DATE_NOW}] ERROR: suricata -T failed, NOT reloading service." | tee
-a "$LOG_FILE"
    exit 1
fi

echo "[${DATE_NOW}] [3/3] Reload Suricata service..." | tee -a "$LOG_FILE"
if systemctl reload suricata 2>>"$LOG_FILE"; then
    echo "[${DATE_NOW}] SUCCESS: Rules updated and Suricata reloaded." | tee -a
"$LOG_FILE"
else
    echo "[${DATE_NOW}] WARNING: reload failed, trying restart..." | tee -a
"$LOG_FILE"
    systemctl restart suricata
fi

```

Сделайте исполняемым и запустите:

```

sudo chmod +x /opt/suricata-tools/update_rules.sh
sudo /opt/suricata-tools/update_rules.sh
sudo tail -n 20 /var/log/suricata/update_rules.log

```

5. Сбор IoC-фидов и генерация custom_ioc.rules

Создайте рабочий каталог:

```

mkdir -p ~/suricata-ioc/feeds
cd ~/suricata-ioc

```

Создайте скрипт загрузки fetch_feeds.sh :

```

nano fetch_feeds.sh

```

Содержимое:

```

#!/usr/bin/env bash
set -euo pipefail

FEEDS_DIR=$(dirname "$0")/feeds"
mkdir -p "$FEEDS_DIR"

echo "[*] Downloading Feodo Tracker IP blocklist..."
curl -sS "https://feodotracker.abuse.ch/downloads/ipblocklist_recommended.txt" \
-o "${FEEDS_DIR}/feodo_ips.txt"

echo "[*] Downloading URLhaus IP blocklist..."
curl -sS "https://urlhaus.abuse.ch/downloads/text_ips/" \
-o "${FEEDS_DIR}/urlhaus_ips.txt"

echo "[*] Downloading Botvrij.eu Destination IPs..."
curl -sS "https://www.botvrij.eu/data/ioclist.ip-dst.raw" \
-o "${FEEDS_DIR}/botvrij_ips.txt"

echo "[*] Done. Feeds saved to: ${FEEDS_DIR}"
ls -lh "${FEEDS_DIR}"

```

Сделайте исполняемым и запустите:

```

chmod +x fetch_feeds.sh
./fetch_feeds.sh

```

Создайте скрипт генерации правил generate_custom_ioc_rules.py :

```

nano generate_custom_ioc_rules.py

```

Содержимое:

```

#!/usr/bin/env python3
"""

Генерация custom_ioc.rules из IoC-источников
"""

from pathlib import Path
from datetime import datetime
import sys

# Конфигурация диапазонов SID для разных источников

```

```
SOURCES = {
    "feodo": {
        "file": "feodo_ips.txt",
        "base_sid": 9000000,
        "msg_prefix": "[IPS] Feodo Tracker C&C",
        "classtype": "trojan-activity"
    },
    "urlhaus": {
        "file": "urlhaus_ips.txt",
        "base_sid": 9100000,
        "msg_prefix": "[IPS] URLhaus Malicious IP",
        "classtype": "trojan-activity"
    },
    "botvrij": {
        "file": "botvrij_ips.txt",
        "base_sid": 9200000,
        "msg_prefix": "[IPS] Botvrij.eu IoC",
        "classtype": "trojan-activity"
    }
}

def load_ips(path: Path):
    """Загрузка IP-адресов из текстового файла"""
    ips = []
    if not path.exists():
        print(f"[!] WARNING: File {path} not found, skipping...")
        return ips

    with path.open() as f:
        for line in f:
            line = line.strip()
            if not line or line.startswith("#"):
                continue
            # Извлекаем только IP-адрес (первое поле)
            ip = line.split(',') [0] if ',' in line else line.split()[0]
            # Базовая валидация IPv4
            if '.' in ip:
                parts = ip.split('.')
                if len(parts) == 4:
                    try:
                        if all(0 <= int(p) <= 255 for p in parts):
                            ips.append(ip)
                    except ValueError:
                        continue
    return ips
```

```
def generate_drop_rules(source_name, config, ips):
    """Генерация drop-правил для списка IP"""
    rules = []
    sid = config["base_sid"]

    # Ограничиваем количество правил для практической работы
    max_rules = 1000
    for ip in ips[:max_rules]:
        rule = (
            f"drop ip {ip} any -> $HOME_NET any "
            f'({msg}:{config["msg_prefix"]}{ip}); '
            f'classtype:{config["classtype"]}; '
            f'sid:{sid}; rev:1;)\n'
        )
        rules.append(rule)
        sid += 1

    return rules, sid

def add_to_yaml_config(rules_file: Path):
    """Добавление custom_ioc.rules в suricata.yaml если его там нет"""
    yaml_file = Path("/etc/suricata/suricata.yaml")

    with yaml_file.open() as f:
        content = f.read()

    if "custom_ioc.rules" in content:
        print("[*] custom_ioc.rules already in suricata.yaml")
        return

    # Ищем секцию rule-files и добавляем наш файл
    lines = content.split('\n')
    new_lines = []

    for i, line in enumerate(lines):
        new_lines.append(line)
        if 'rule-files:' in line:
            # Смотрим на следующие строки для определения отступа
            for j in range(i+1, min(i+5, len(lines))):
                if lines[j].strip().startswith('-'):
                    indent = len(lines[j]) - len(lines[j].lstrip())
                    new_lines.append(' ' * indent + f"- {rules_file}")
                    break

    with yaml_file.open('w') as f:
        f.write('\n'.join(new_lines))
```

```
print(f"[+] Added {rules_file} to suricata.yaml")

def main():
    feeds_dir = Path(__file__).parent / "feeds"
    out_file = Path("/etc/suricata/rules/custom_ioc.rules")

    all_rules = []
    stats = {}

    print("[*] Starting IoC rules generation...\n")

    # Генерация правил для каждого источника
    for source_name, config in SOURCES.items():
        source_file = feeds_dir / config["file"]
        ips = load_ips(source_file)

        if not ips:
            stats[source_name] = 0
            print(f"[!] {source_name.upper()}: No IPs loaded")
            continue

        rules, last_sid = generate_drop_rules(source_name, config, ips)
        all_rules.extend(rules)
        stats[source_name] = len(rules)

        print(f"[+] {source_name.upper()}: Generated {len(rules)} rules "
              f"(SID: {config['base_sid']}-{last_sid-1})")

    if len(all_rules) == 0:
        print("\n[!] ERROR: No rules generated. Check if feeds were "
              "downloaded correctly.")
        print("[!] Run ./fetch_feeds.sh first to download IoC feeds.")
        sys.exit(1)

    # Запись всех правил в один файл
    out_file.parent.mkdir(parents=True, exist_ok=True)
    with out_file.open("w") as f:
        f.write(f"# Autogenerated IoC-based rules from multiple sources\n")
        f.write(f"# Generated: {datetime.now().isoformat()}\n")
        f.write(f"# Total rules: {len(all_rules)}\n")
        f.write(f"# Sources: {', '.join(SOURCES.keys())}\n")
        f.write(f"\n")
        for source, count in stats.items():
            f.write(f"# - {source}: {count} rules\n")
    f.write(f"\n")
```

```
        for rule in all_rules:
            f.write(rule)

    print(f"\n[+] Total: Generated {len(all_rules)} rules into {out_file}")

    # Добавляем файл в конфигурацию Suricata
    add_to_yaml_config(out_file)

    print("\n[*] Done! Run 'sudo /opt/suricata-tools/update_rules.sh' to
apply changes.")

if __name__ == "__main__":
    main()
```

Запустите генерацию:

```
sudo python3 generate_custom_ioc_rules.py
```

Проверьте результат:

```
sudo wc -l /etc/suricata/rules/custom_ioc.rules
sudo head -20 /etc/suricata/rules/custom_ioc.rules
```

Проверьте, что файл добавлен в конфигурацию:

```
grep "custom_ioc.rules" /etc/suricata/suricata.yaml
```

Обновите правила:

```
sudo /opt/suricata-tools/update_rules.sh
```

6. Объединение в Makefile

Установите Make:

```
sudo apt install make
```

Создайте Makefile:

```
nano Makefile
```

Содержимое:

```
.PHONY: all fetch generate test deploy clean status

all: fetch generate test deploy status

fetch:
    @echo "[*] Stage 1/4: Fetching external feeds..."
    ./fetch_feeds.sh

generate:
    @echo "[*] Stage 2/4: Generating custom_ioc.rules..."
    @sudo python3 generate_custom_ioc_rules.py

test:
    @echo "[*] Stage 3/4: Testing Suricata configuration..."
    @sudo suricata -T -c /etc/suricata/suricata.yaml

deploy:
    @echo "[*] Stage 4/4: Deploying rules (reload Suricata)..."
    @sudo systemctl reload suricata || sudo systemctl restart suricata
    @echo "[+] Pipeline completed successfully!"

status:
    @echo "[*] Checking Suricata status..."
    @sudo systemctl status suricata --no-pager | head -10

clean:
    @echo "[*] Cleaning feeds directory..."
    @rm -rf feeds/
```

Запустите полный цикл:

```
make all
```

7. Проверка работы правил

Проверьте логи Suricata через EveBox

Задание к работе

1. Автоматизируйте загрузку и использование актуальных правил от Positive Technologies

2. Используя списки IP-адресов заблокируйте несколько любых публичных сервисов
3. Включите в блокировку ресурсы которые запрещены в РФ (проверку можно сделать с использованием proxy или vpn)