THOMAS MORE

Fortinet firewall automated compliance testing

Realisatiedocument

Quinten Van der Wildt
Student Bachelor in de Elektronica-ICT – Cloud & Cyber Security

2025 - 2026

Inhoudsopgave

1. Inleiding

Dit verslag beschrijft mijn stageopdracht bij Cyfora van 23 februari tot 29 mei 2025, waarin ik aan het project Fortinet Firewall Automated Framework Compliance Testing heb gewerkt. In de eerste fase van mijn stage werkte ik aan Cyfora's bestaande Palo Alto firewall analyser, waar ik verschillende security compliance checks implementeerde. Deze inleiding was zeer waardevol, omdat ik hierdoor kon begrijpen hoe het analyseplatform van Cyfora functioneert, hoe gegevens door het systeem stromen, en welke architectuurpatronen reeds bewezen zijn. Met deze kennis kon ik vervolgens aanzienlijk efficiënter aan het Fortinet-project beginnen.

Het Fortinet-project richtte zich op het ontwerpen van een framework dat firewall-configuraties automatisch kan valideren. Fortinet-firewalls bevatten immers honderden configuratieparameters, en handmatige controle op naleving van best practices is inefficiënt en foutgevoelig. Het doel was een end-to-end oplossing te creëren die Fortinet YAML-configuraties inleest, deze valideert tegen 41 security best practices (beheerbeleid, netwerkzones, IPS/AV-profielen, vergrendeling policies, enzovoort), en een interactief rapport genereert. Dit framework is inmiddels als onderdeel van Cyfora's platform geïmplementeerd en helpt hun klanten firewall-configuraties automatisch te valideren.

2. Analyse

Alvorens de daadwerkelijke implementatie van het Fortinet framework begon, was een grondige analyse van het probleem, de beschikbare tools en technologieën nodig. Deze analyse richt zich op het begrijpen van Cyfora's analyseplatform, het onderzoeken van Fortinet configuraties, en het bepalen van de best passende tools en frameworks.

2.1. Platform architectuur begrijpen

Alvorens aan het Fortinet-project te beginnen, werkte ik eerst aan Cyfora's bestaande Palo Alto firewall analyser. Dit bleek een uitstekende inleiding op mijn stage, hierdoor kon ik de codestructuur van het platform doorgronden. Gedurende de eerste weken implementeerde ik diverse security compliance checks voor Palo Alto:

De eerste opdrachten bestonden uit het toevoegen van kleine securitychecks binnen de Palo Alto analyser. Dit omvatte onder andere:

- Admin lockout policy controles
- Idle timeout configuraties
- Password policy validaties
- Login banner controles
- NTP server configuraties
- High Availability monitoring checks

Door aan het Palo Alto-project te werken, verkreeg ik diepgaand inzicht in de interne werking van Cyfora's platform. Dit was fundamenteel voor het vervolgwerk. Door deze implementaties kreeg ik inzicht in hoe een analyser opgebouwd wordt:

1. Configuratie inlezen
2. Gegevens normaliseren
3. Compliance logica uitvoeren
4. Scores genereren
5. Evidence toevoegen
6. Rapportering opbouwen

Deze ervaring vormde de basis voor de verdere ontwikkeling van het Fortinet framework.

2.2. Analyse Voor Fortinet-technologie

Na de analyse van Fortinet-technologie bleek het volgende.

2.2.1. Configuratie Export Opties

Er zijn twee mogelijkheden om de configuratie uit te lezen. Bestanden of de API

De bestanden hebben 2 verschillende bestandstypes:

1. **YAML Export**: Gestructureerd YAML-formaat dat parseerbaar is door standaard YAML-parsers. Dit maakt geautomatiseerde verwerking eenvoudig.
2. **.conf Export**: Fortinet's proprietaire CLI-tekstformaat. Dit is human-readable en gebruikt voor CLI-based configuratie.

Voor mijn project koos ik YAML omdat Python reeds over bibliotheken beschikt voor het verwerken van YAML-bestanden.
De API reageert in een JSON-formaat dat gewoon zonder library kan behandeld worden in python.

2.2.2. Configuratie Structuur: 2 modes

2.2.2.1. Configuratie Structuur 1: Multi Vdom

TWEE HOOFDNIVEAUS

Fortinet-firewalls organiseren hun configuratie in twee primaire hiërarchische niveaus:

LEVEL 1: GLOBAL CONFIGURATION (SYSTEEM-NIVEAU)

Global Configuration bevat alle systeem-brede instellingen die bepalen hoe de firewall als geheel functioneert:

- System Settings: Timezone, hostname, DNS servers, NTP synchronization
- Admin Access: Administrator accounts, SSH/HTTPS ports, TLS versions, certificates
- Admin Policies: Login attempt limits, lockout duration, password requirements, 2FA configuration
- Management: SNMP configuration, HA (High Availability) mode, session pickup
- Remote Access: SSL-VPN settings, IPsec tunnel defaults
- Security Services: IPS (Intrusion Prevention System) probestands, antivirus policies, web filtering

Specifieke voorbeelden van Global Configuration parameters:

```YAML
system:
  setting:
    timezone: 0                # UTC = 0, andere zones hebben ander nummer
    hostname: "FortiGate-60F"
    admin-lockout-threshold: 3   # Login pogingen
    admin-lockout-duration: 60   # Minuten
ntp-server:
  - server: "0.fortios.pool.ntp.org"
  - server: "1.fortios.pool.ntp.org"
```

LEVEL 2: VDOM CONFIGURATION (VIRTUELE DOMEIN-NIVEAU)

VDOM staat voor Virtual Domain en is bedoeld voor firewall partitionering. Een enkele fysieke firewall kan meerdere onafhankelijke VDOMs bevatten, elk met:

- Firewall Policies: Allow/deny regels voor traffic tussen zones
- Security Zones: Logische interface grouping (bijv. "trusted", "dmz", "untrusted")
- IPsec Tunnels: Site-to-site VPN configuratie per VDOM
- SSL Inspection Policies: Decryption en inspection van HTTPS traffic
- Security Probestands: AV, IPS, web filter, DNS filter probestands per policy
- Logging: Event logging configuratie (succesvolle/gefaalde connections)

Specifieke voorbeelden van VDOM Configuration parameters:

```YAML
vdom:
  name: "VDOM-A"
  firewall:
    policy:
      - policyid: 1
        name: "Allow-Internal-to-DMZ"
        srcintf: ["internal"]
        dstintf: ["dmz"]
        srcaddr: ["all"]
        dstaddr: ["all"]
        action: "accept"
        av-probestand: "default"
        ips-sensor: "default"
        ssl-ssh-probestand: "deep-inspection"
        logtraffic: "enable"
    addrgrp:
      - name: "internal-servers"
        member: ["web-server", "db-server"]
    address:
      - name: "web-server"
        type: "ipmask"
        subnet: "10.1.1.100/32"
```

2.2.2.2. Configuratie Structuur 2: Non-Vdom

Deze structuur is grotendeels vergelijkbaar met structuur 1, met als verschil dat alle informatie die normaal onder niveau 2 valt rechtstreeks onder niveau 1 wordt geplaatst.

2.2.3. Multi-Tenant Architectuur

Een van de krachtige kenmerken van Fortinet is de native multi-tenant ondersteuning via VDOMs. Dit maakt het volgende mogelijk:

- **Organisatie-isolatie**: Locatie A en B kunnen totaal gescheiden firewall policies hebben op dezelfde fysieke hardware
- **Resource Sharing**: Onderliggende hardware wordt gedeeld, maar policies zijn volledig onafhankelijk
- **Administrator-delegatie**: Verschillende VDOM beheerders kunnen hun eigen policies configureren zonder zicht op andere VDOMs
- **Compliance-scheiding**: Compliance checks kunnen per VDOM worden gevalideerd (nuttig voor multi-tenant klanten)

2.2.4. Test Firewall Specificaties

Voor dit project gebruikte Cyfora een FortiGate 60F firewall. De firewall werd initieel met firmware versie 7.2.12 geleverd. Halverwege het project voerde ik een firmware upgrade uit naar versie 7.4.10, wat waardevolle inzichten opleverde, zoals ook een bug.

2.3. Analyse van bestandsverwerking

Bij een eerste test uitlezing van de YAML-bestand dat het volledige bestand moest weergeven merkte ik op dat de YAML-structuur niet klopte. Na verder onderzoek werden vier verschillende YAML-syntaxfouten geïdentificeerd.

1. Er waren meerdere waarden voor 1 key die geen lijst zijn
2. Er waren fouten door waardes die een string horen te zijn
3. Dit is een variant van error 1: er waren meerdere waarden voor één key waarbij strings en integers werden gecombineerd, waardoor een specifieke verwerking vereist was.
4. Certificate inline errors

Daarnaast zijn er ook nog 2 andere symptomen die ik heb gefixed.

1. Dat er escapes gebeuren in strings met special characters
2. Excessieve lege regels

Verder onderzoek toonde aan dat 4 van de bovenstaande te wijten waren aan een bug en de 2 andere zijn default gedrag.

2.3.1. BUG 976722 Ontdekt

De test-firewall die Cyfora beschikbaar had (een FortiGate 60F met versie 7.2.12) genereerde exports die syntactisch ongeldig waren. Fortinet had hier een bekende bug voor geïdentificeerd: BUG ID 976722, aanwezig in versies 7.2.x tot 7.4.7.

Dit probleem manifesteerde zich op meerdere manieren:

Tabel 1

Symptoom	Omvang	Voorbeeld	Na patch 7.4.8
Excessieve lege regels	5000+ per export	Het **bestand** werd onleesbaar, 90% lege regels	Nog steeds aanwezig
Certificate inline errors	Multiple per config	`"-----BEGIN\nCERT\n-----END"` - ongeldig YAML-syntax	Opgelost
Multiple quoted strings	20+ instances	`member: "a" "b"` - geen list, twee strings na elkaar	Opgelost
Wildcard escaping	90+ instances	`- *.bat:` - unquoted, foutieve YAML-syntax	Nog steeds aanwezig
Invalid escape sequences	10+ instances	`"test\'string"` - escape niet geldig in YAML	Opgelost
Mixed key-value pairs	5+ instances	`"internal1" 26 "wan1" 97` - geen valid structuur	Opgelost

In het hoofdstuk 3.2.1.1 worden ze verder uitgelegd.

2.4. Vergelijking van oplossingsalternatieven

Nadat het probleem duidelijk was vastgesteld, diende ik een aanpak te bepalen. Voor het YAML-reparatieprobleem bestonden meerdere oplossingsrichtingen:

YAML-REPAIRER - MOGELIJKHEDEN:

Table 1

Optie	Voordelen	Nadelen	Score
Regex-based text rewriting	Snel, minimale dependencies	Fragiel, moeilijk te onderhouden	4/10
YAML-parser + Abstract Syntax Tree (AST) fix	Semantisch correct, robuust	Complex, performance overhead	6/10
Custom line-by-line rewriter	Eenvoudig te debuggen, predictable	Moet alle bug-types afhandelen	9/10

OPTIE 1: REGEX-BASED TEXT REWRITING

- Idee: Gebruik regex patterns om fouten op te sporen en te vervangen
- Voordelen:
    - Snelle implementatie (regex patterns geschreven in 1-2 uur)
    - Minimale dependencies (alleen PyYAML nodig)
    - Goed voor eenvoudige find-and-replace operaties
- Nadelen:
    - Fragiel: Regex patterns breken bij kleine variaties in syntax
    - Onderhoud: Elke nieuwe bug-variant vereist nieuwe regex
    - Debugging moeilijk: Regex output is hard te begrijpen
    - Edge cases: Zal veel false positives hebben
- Gebruik case: Eenmalige quick-fixes, niet geschikt voor production

OPTIE 2: YAML-PARSER + AST FIX

- Idee: YAML omzetten naar een AST, correcties uitvoeren op AST-niveau en het resultaat terugschrijven naar YAML.
- Voordelen:
    - Semantisch correct: de toegepaste correcties houden rekening met de onderliggende YAML-structuur en betekenis van de configuratie.
    - Robuust: Werkt met alle variaties van incorrecte YAML
    - Elegant: Transformeert data op het juiste niveau
- Nadelen:
    - Complex: AST-manipulatie moeilijk om correct te doen
    - Performance: AST parsing is trager voor grote bestandsen (2.1 MB)
    - Dependencies: Vereist een goede YAML-parser library
    - Debugging: AST errors zijn moeilijk om af te leiden naar originele regels
- Gebruik case: Voor production-grade systematische reparatie, maar overkill voor dit project

OPTIE 3: CUSTOM LINE-BY-LINE REWRITER (GEKOZEN)

- Idee: Sequentieel door het **bestand**, fix regels met concrete regels (niet regex)
- Voordelen:
    - Directe controle: Exact begrijpen wat er per regel gebeurt
    - Debuggen: regelnummers kunnen nauwkeurig worden gebruikt om de exacte locatie van fouten te traceren.
    - Predictable: Output is altijd wat je verwacht (geen verrassingen)
    - Efficiënt: Lineair doorgang (O(n) complexity)
    - Onderhoudbaarheid: Toekomstige bugs makkelijk toe te voegen
    - Geen parsing overhead: Niet volledig YAML parsen
- Nadelen:
    - Moet alle bug-types expliciet afhandelen
    - Iets meer code dan regex approach

Gekozen: Custom line-by-line rewriter vanwege directe controle over problematische regels, eenvoudig debuggen per bug-type, goede prestaties, en perfecte balans tussen snelheid en robuustheid. Dit wordt gedetailleerd beschreven in sectie 3.3.

2.5. Tools & Dependencies

De volgende tools en libraries werden geselecteerd:

Core Libraries
- PyYAML (3.13+): YAML-parsing
- pydantic (v1/v2): Data validation
- pytest: Unit testing framework

Development Tools
- VS Code: Editor
- Git: Version control
- Python venv: Virtual environment

Quality Assurance & Type Safety
- pytest: 41 unit tests voor analyser coverage
- mypy: Static type checking voor code semantics en type validation

Context: Cyfora's Bestaande Platform

Bij aankomst bij Cyfora ontdekte ik dat het bedrijf al een Python-gebaseerde analyserframework had gebouwd voor hun Palo Alto firewall compliance checks. Dit was enorm waardevol omdat:

1. Analyser Architecture: Het bedrijf had al een bewezen architectuur voor compliance checking:
    a. Inlezen van firewallconfiguraties (JSON/XML)
    b. Runs security best-practicechecks
    c. Genereert gestructureerde scores en bewijsmateriaal
    d. Genereert HTML rapporten

2. Bestaande Tests: De analyser had al een test suite met unit tests, wat een goed voorbeeld was voor hoe ik mijn Fortinet tests zou structureren.

3. Type Checking met mypy: Cyfora gebruikte mypy voor static type checking, wat betekent dat:
    a. Alle TypedDict models strict type-checked waren
    b. IDE autocomplete en error detection werkte goed
    c. Code refactoring was veilig (mypy catch breaking changes)
    d. Type errors werden opgespoord voordat code in production ging

Dit maakte het voor mij aanzienlijk gemakkelijker om:
- Dezelfde patterns toe te passen voor Fortinet
- Consistent code style te handhaven
- Vertrouwen te hebben in de typeveiligheid van de code

Ik kon derhalve rechtstreeks voortbouwen op deze reeds aanwezige architectuur, waardoor mijn ontwikkelingstijd aanzienlijk werd verkort. In plaats van alles vanaf nul uit te werken, kon ik mij concentreren op de Fortinet-specifieke logica (checks, validaties) en het YAML-reparatieframework.

3. REALISATIE & IMPLEMENTATIE

Dit hoofdstuk beschrijft de technische implementatie van het ontwikkelde Fortinet compliance-analyseframework. De architectuur is ontworpen volgens een modulair en source-onafhankelijk principe, waarbij zowel YAML-configuratiebestanden als live REST API-data verwerkt kunnen worden tot één uniform datamodel. Hierdoor blijft de analyse-engine identiek ongeacht de gebruikte databron.

3.1. Best Practices Framework Design

De 41 security best practices zijn gebaseerd op Fortinet's eigen aanbevelingen en industry standards (NIST, CIS):

Table 2

Categorie	Number of checks	Voorbeelden
Device Configuration	3	Timezone UTC, NTP, Firmware updates
Admin Access Configuration	2	HTTPS/SSH ports, TLS versies
Admin User Configuration	7	Default admin disabled, 2FA, password policy, lockout policy
Management Configuration	6	SNMP v3, HA mode, HA monitoring, HA session pickup
Remote Access Configuration	1	SSL-VPN disabled
VDOM Configuration	22	Firewall zones, deep inspection, logging, IPsec, DDoS protection, SD-WAN zones, interface security

SCORING METHODOLOGIE:

Voor elke best practice geldt:
- **Binary score**: 0 = niet geconfigureerd/niet-compliant, 100 = correct geconfigureerd/compliant
- **Percentage score**: Score = (compliant_items / total_items) * 100 voor checks met meerdere configuratie-opties
- **Evidence**: Elke score wordt ondersteund met gedetailleerde data uit de configuratie

VOORBEELDEN VAN VERSCHILLENDE SCORING-MODELLEN:

1. **Binary (0 of 100)**
   - Timezone UTC: 100 als timezone=0, anders 0
   - SSL-VPN disabled: 100 als uitgeschakeld, 0 als ingeschakeld
2. **Percentage (0-100)**
   - Password policy: (enabled + min_length + mixed_case + special_char + number) / 5 * 100
   - DoS baseline actions: (tcp_block + udp_block + icmp_block) / 3 * 100
   - Firewall policies: (policies_with_correct_probestands / total_policies) * 100
3. **Partial (50 punten)**
   - NTP configuration: 100 als 2+ servers, 50 als 1 server, 0 als 0 servers
   - Admin users: 100 als 2+ super admins, 0 anders

3.2. Architectuur Overzicht

Het systeem ondersteunt twee databronnen en bestaat uit componenten in dual-pipeline format. Beide bronnen leiden tot dezelfde data model, waardoor de analyser engine onveranderlijk blijft:

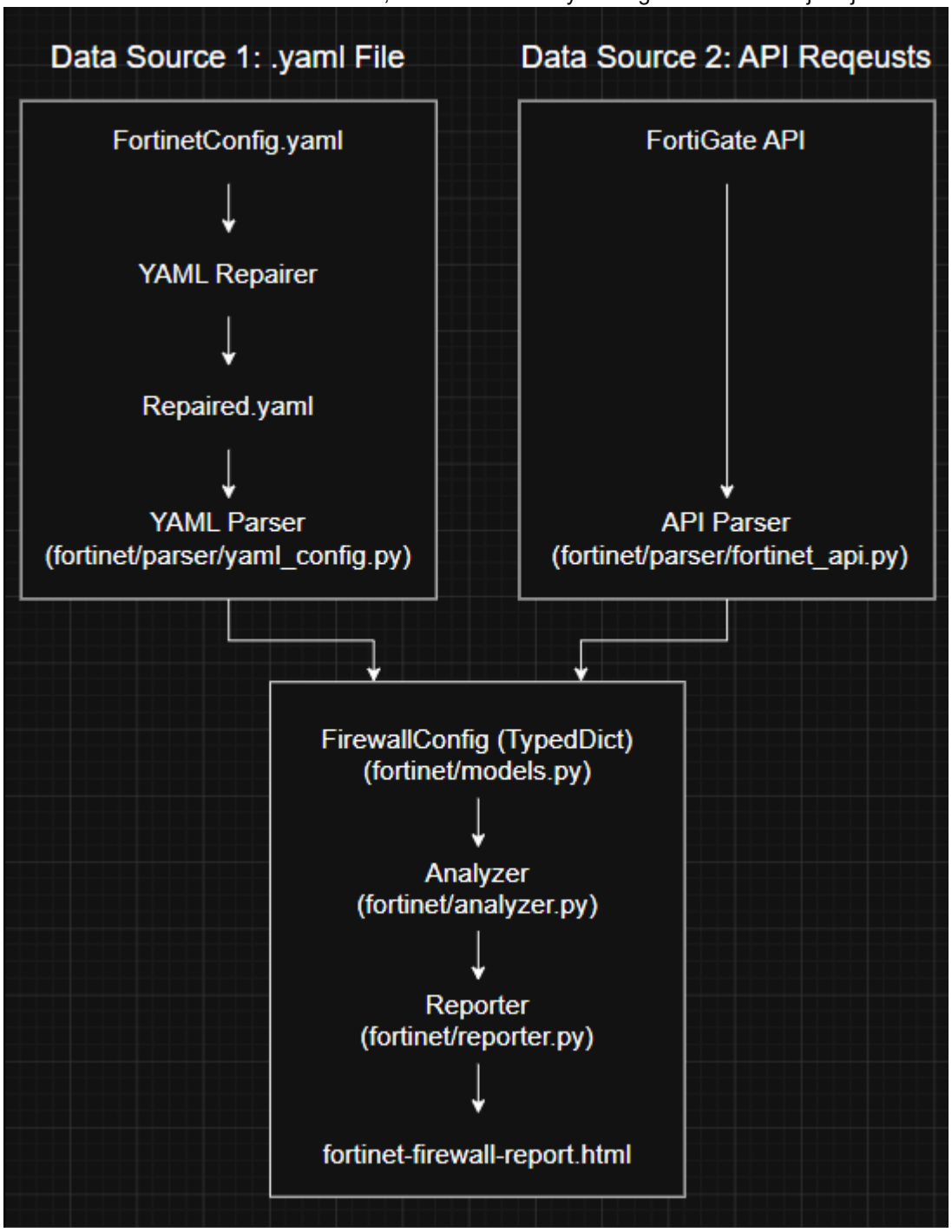

Figure 1: Data Flow Diagram(DFD) Fortinet Technologie

3.2.1. Databron 1: YAML-configuratie:

We starten met een geëxporteerd YAML-bestand van een Fortinet-firewall.

Daarna Gaat deze in Component 1 de YAML-repairer

3.2.1.1. Component 1: YAML-Repairer

Bestand : `Fortinet/parser/Fortinet_YAML_parser.py`

Werking & herstelstrategiestrategie

De YAML-repairer implementeert de zes bug-fixes uit sectie 2.2 (Bug 1-6: lege regels, certificates, quoted strings, wildcards, escape sequences en key-value pairs) via een geautomatiseerde line-by-line rewriter aanpak.

Deze component is essentieel omdat Fortinet YAML exports in FortiOS-versies 7.2.0 tot en met 7.4.7 syntactisch ongeldig kunnen zijn. Zonder reparatie kunnen standaard YAML-parsers de configuratie niet correct verwerken.

Implementatie stappen

1. Lege regels verwijderen (Type 6): filtert alle volledig lege regels uit het bestand.
2. Certificate formatting (Type 5): converteert inline certificaten naar correcte multiline YAML-syntax met |.
3. Syntax fixes (Types 1-4): repareert quoted strings, key-value patronen, wildcards en escape sequences.
4. Spacing toevoegen: voegt extra lege regels toe tussen configuratiesecties voor betere leesbaarheid.

HOOFD-FIX TYPES

1. Invalid Escape Sequences

Fortinet exporteert soms backslash-escaped single quotes binnen double-quoted YAML strings. Deze escape-sequences zijn niet geldig volgens de YAML-specificatie. De repairer verwijdert daarom onnodige backslashes zodat de string correct geïnterpreteerd kan worden door de YAML-parser.

```python
def fix_invalid_escape_sequences(line: str) -> Tuple[str, bool]:
    """Fix: \' → ' (single quotes don't need escaping in YAML double-quoted strings)"""
    if "\\'" in line:
        line = line.replace("\\'", "'")
        return line, True
    return line, False
```

2. Unquoted Wildcard Patterns

Firewallregels met wildcard patronen zoals *.bat worden door YAML geïnterpreteerd als aliassen in plaats van gewone strings. De repairer detecteert deze patronen automatisch en plaatst ze tussen quotes zodat ze correct als tekstwaarden verwerkt worden.

```python
def fix_unquoted_patterns(line: str) -> Tuple[str, bool]:
    """Fix: - *.bat: → - "*.bat":"""
    if re.search(r'^\s*-\s+\*\S+:', line):
        # Quote the wildcard pattern
        return modified_line, True
    return line, False
```

3. Mixed Key-Value Patterns

Bepaalde configuratieregels bevatten opeenvolgende key-value combinaties zonder duidelijke dictionary-structuur. De repairer herstructureert deze regels naar een correcte YAML dictionary-notatie zodat parsing mogelijk wordt.

```python
def fix_mixed_key_value_patterns(line: str) -> Tuple[str, bool]:
    """Fix: "key1" val1 "key2" val2 → {"key1": val1, "key2": val2}"""
    if re.search(r':\s*"[^"]*"\s+\d+', line):
        # Convert to dict format
        return converted_dict, True
    return line, False
```

4. Certificate Inline Formatting

Inline certificaten met embedded newline-karakters veroorzaken parsingproblemen in YAML. De repairer converteert deze certificaten naar correcte multiline YAML blocks met de | syntax, waardoor certificaatdata veilig behouden blijft.

```python
def fix_certificate_inline(line: str) -> Tuple[str, bool]:
    """Fix: Converteer inline certificates naar multiline format"""
    if "-----BEGIN CERTIFICATE-----" in line and "\n" in line:
        # Converteer naar proper YAML multiline string met |
        return converted_multiline, True
return line, False
```

Voorbeeld:
"-----BEGIN\nCERT\n-----END" → correcte multiline YAML representatie.

READABILITY FIX TYPES

5. Lege Regels Verwijderen

Fortinet YAML exports bevatten vaak duizenden volledig lege regels. In de testconfiguratie bestond meer dan 90% van het bestand uit lege regels. De repairer verwijdert deze regels volledig om bestandsgrootte en parsing-complexiteit te reduceren.

```python
def remove_blank_lines(lines: list[str]) -> list[str]:
    """Fix: Verwijder alle volledig lege regels (5,000+ → 0)"""
    # Filter alle regels die alleen whitespace bevatten
    return [line for line in lines if line.strip()]
```

6. Multiple Quoted Strings

Sommige configuraties bevatten meerdere quoted strings zonder lijststructuur of scheidingstekens. Hierdoor ontstaat ongeldige YAML-syntax. De repairer converteert dergelijke patronen automatisch naar geldige YAML-lijsten.

```python
def fix_multiple_quoted_strings(line: str) -> Tuple[str, bool]:
    """Fix: member: "a" "b" → member: ["a", "b"]"""
    if re.search(r':\s*"[^"]*"\s+"[^"]*"', line):
        # Convert to list format
        return converted_line, True
    return line, False
```

TWEE REPAIR WORKFLOWS (VERSIE-AFHANKELIJK)

Halverwege het project werd een firmware-upgrade uitgevoerd op de test-firewall van FortiOS 7.2.12 naar 7.4.10. Hierdoor werd ontdekt dat bepaalde YAML-fouten nog steeds aanwezig waren, ondanks Fortinet's patch in versie 7.4.8+.

Dit resulteerde in twee afzonderlijke repair workflows.

WORKFLOW A: FORTIOS 7.2.0 – 7.4.7 (VOLLEDIGE REPAIR)

Deze workflow voert alle zes de reparatiestappen uit:

1. Escape sequences herstellen
2. Wildcards quoten
3. Multiple quoted strings repareren
4. Mixed key-value patronen converteren
5. Certificate formatting corrigeren
6. Lege regels verwijderen
7. YAML-validatie uitvoeren

WORKFLOW B: FORTIOS 7.4.8+ (LICHTGEWICHT REPAIR)

Vanaf versie 7.4.8 zijn bepaalde bugs opgelost door Fortinet. Enkel de resterende structurele problemen worden nog hersteld:

1. Wildcard patterns quoten
2. Escape sequences herstellen
3. Lege regels verwijderen
4. YAML-validatie uitvoeren

BELANGRIJK INZICHT

De firmware-upgrade leverde drie belangrijke inzichten op:

- Types 1-3 (certificates, multiple strings en key-value patterns) zijn opgelost in FortiOS 7.4.8+.
- Types 4-6 (wildcards, escape sequences en lege regels) blijken fundamentele eigenschappen van Fortinet's YAML-export en komen nog steeds voor in alle versies.

REPAIRER USAGE:

```python
from Fortinet.parser.Fortinet_YAML_parser import normalize_Fortinet_YAML,
normalize_Fortinet_YAML_non_bug

# Voor BUG 976722-aangetaste versies (7.2.0 - 7.4.7):
repaired_YAML = normalize_Fortinet_YAML(input_bestand_path)

# Voor versies 7.4.8+ (lichtgewicht repair):
repaired_YAML = normalize_Fortinet_YAML_non_bug(input_bestand_path)

# Parse nu het gerepareerde bestand
import YAML
data = YAML.safe_load(repaired_YAML.read_text())
```

De reparatie genereert een artifact van het gerepareerde bestand, dat vervolgens wordt gebruikt door Component 2 (data-extractie).

3.2.1.2. Component 2: Yaml Configuration Parser

Bestand: `Fortinet/parser/YAML_config.py`

```python
class YAMLConfigParser(BaseConfigParser):
    """Parse a standalone Fortinet.conf.yml bestand."""
    def __init__(self, config_bestand: Path) -> None:
        self.config_bestand = config_bestand

    def parse(self) -> FirewallConfig:
        """Parse YAML-bestand into FirewallConfig TypedDict"""
        raw_text = self.config_bestand.read_text(encoding="utf-8")

        # Detect version and apply appropriate normalization
        config_version = _get_config_version(raw_text)

        if _is_buggy_YAML_version(config_version):
            # Apply full repair for FortiOS 7.2.0 - 7.4.7
            repaired_bestand = normalize_Fortinet_YAML(self.config_bestand)
            cleaned_text = repaired_bestand.read_text(encoding="utf-8")
        else:
            # Apply lightweight normalization for FortiOS 7.4.8+
            repaired_bestand = normalize_Fortinet_YAML_non_bug(self.config_bestand)
            cleaned_text = repaired_bestand.read_text(encoding="utf-8")

        # Parse cleaned YAML
        data = YAML.safe_load(cleaned_text) or {}

        # Extract global and VDOM configurations
        return self._parse_configuration(data, config_version)
```

VDOM EXTRACTION LOGIC:

1. Parse YAML document
2. Zoek "vdom:" sectie met alle VDOM-namen
3. Voor elke VDOM:
    a. Vind corresponderende vdom config block
    b. Extract firewall policies, interfaces, etc.
    c. Converteer naar FirewallConfig schema
4. Return {vdom_name: {policy, interface, ...}, ...}

Voordelen YAML-approach:

- Offline processing - geen network nodig
- Sneller - direct bestandsverwerking (50-100ms)
- Versie-detectie (FortiOS 7.2 - 7.4+)
- Automatische BUG 976722 reparatie voor kwetsbare versies
- VDOM extraction uit YAML met `extract_vdoms(YAML_text: str)` functie
- Ondersteunt batch-processing van meerdere configuraties
- Geen credentials nodig

Nadelen YAML-approach:

- YAML-syntax errors (BUG 976722) vereisen reparatie
- Bestandsbeheer nodig (upload/download)
- Enkel snapshot van bepaald moment

3.2.2. Databron 2: REST API-parser

Bestand: `Fortinet/parser/Fortinet_api.py`

```python
class FortinetAPIConfigParser(BaseConfigParser):
    """Parse raw API JSON into the common FirewallConfig model."""
    def __init__(self, fortigate_api_url: Any, fortigate_api_token: Any) -> None:
        self.fortigate_api_url = fortigate_api_url
        self.fortigate_api_token = fortigate_api_token

    def parse(self) -> FirewallConfig:
        fortigate_api_url = self.fortigate_api_url
        fortigate_api_token = self.fortigate_api_token

        # 1. Extract hostname and version
        url = f"{fortigate_api_url}api/v2/monitor/system/status"
        headers = {"Authorization": f"Bearer {fortigate_api_token}"}
        response = requests.get(url, headers=headers, verify=False, timeout=10)
        response.raise_for_status()

        data = response.json()
        hostname = data["results"]["hostname"]
        config_version = data["version"]

        # 2. Extract global configuration
        global_system_congfiguration = _parse_global_system_congfiguration(
            fortigate_api_token, fortigate_api_url
        )
        # Other extractions

        # 3. Extract VDOM configurations
        vdoms = extract_vdoms(fortigate_api_token, fortigate_api_url)
        vdom_configuration: list[VdomConfiguration] = []
        if vdoms is not None:
            for vdom_name, vdom_conf in vdoms.items():
                vdom_config_iter = _parse_vdoms_api(
                    vdom_name, fortigate_api_token, fortigate_api_url
                )
                vdom_configuration.append(vdom_config_iter)


        # 4. Return complete FirewallConfig
        return FirewallConfig(
            hostname=hostname,
            config_version=config_version,
            global_system_congfiguration=global_system_congfiguration,
            # Other firewall data
        )
```

VDOM EXTRACTION:

```python
def extract_vdoms(fortigate_api_token: Any, fortigate_api_url: Any) -> dict[str, Any]:
    """Extract all VDOM names and their config blocks from FortiGate API."""
    vdom_config_dict: dict[str, Any] = {}

    vdom_url = f"{fortigate_api_url}api/v2/cmdb/system/vdom"
    headers = {"Authorization": f"Bearer {fortigate_api_token}"}

    response = requests.get(vdom_url, headers=headers, verify=False, timeout=10)
    response.raise_for_status()

    vdom_data = response.json()
    vdoms = vdom_data.get("results", []) if "results" in vdom_data else vdom_data

    if vdoms is not None:
        for vdom in vdoms:
            vdom_name = vdom.get("name")
            if vdom_name is not None:
                vdom_config_dict[vdom_name] = {
                    "name": vdom_name,
                    "status": vdom.get("status"),
                }

    return vdom_config_dict
```

Voordelen REST API-approach:

- Real-time data extraction van live firewall (15 – 30 seconden)
- Geen YAML-syntax errors - native JSON structured data
- Automatische versie-detectie via API
- Bearer token authentication (OAuth 2.0)
- HTTPS support met firewall certificate validation optie
- VDOM extraction via `extract_vdoms(token, url)` functie
- Ondersteunt geautomatiseerde monitoring en dashboards

Nadelen REST API-approach:

- Vereist network access naar firewall
- Vereist API credentials/token
- Langzamer door network latency
- Firewall moet online en bereikbaar zijn
- Firewall-specifieke rate limiting

3.2.3. Vergelijking .YAML-parser and API parsing

Table 3

Criterium	YAML-Parsing	REST API
Snelheid	Snel (5 – 7seconden)	Langzaam (15 – 30seconden)
Real-time	Snapshot	Live
Betrouwbaarheid	BUG 976722 reparatie nodig	Altijd correct
Offline support	Ja	Nee
Network nodig	Nee	Ja (TCP/443)
Credentials	Niet nodig	API token vereist
Bulk processing	Ja (veel bestanden)	1 firewall tegelijk
Use case	Compliance audits	Live monitoring

Selectie Richtlijn:

- **YAML-parser:** Voorkeur voor offline compliance audits, periodieke scans, gearchiveerde configs
- **API-parser:** Voorkeur voor live monitoring, real-time dashboards, geautomatiseerde checks
- **Hybride:** Productiefirewall (API) + backup/export (YAML) voor disaster recovery

Gekozen aanpak (default):

- **YAML-approach als primary:** Voor compliance audits en periodic scans (productie-gebruik)
- **API-approach als optie:** Voor real-time monitoring en live dashboards

De implementatie ondersteunt beide, zodat klanten kunnen kiezen op basis van hun requirements.

Conclusie:
Beide parsers delen dezelfde onderliggende data models (`FirewallConfig`, `VdomConfiguration`, etc.) van **algemene component 1 van H3.3.1**, waardoor de Analyser engine onveranderlijk kan blijven ongeacht de data source.

3.3. Algemene Componenten blok

Dit deel gaat over de componenten die niet data-afhankelijk zijn zoals de data models en de scoring testen.

3.3.1. Algemene Component 1: Data Models

Bestand: `Fortinet/models.py`

De configuratie is gemodelleerd met Python `TypedDict` voor type-safety en IDE-autocomplete:

```python
class GlobalSystemConfiguration(TypedDict):
    timezone: int  # 0 = UTC
    hostname: str
    admin_lockout_threshold: int
    admin_lockout_duration: int

class VdomFirewallPolicy(TypedDict):
    policyid: int
    name: str
    srcintf: list[str]
    dstintf: list[str]
    action: str  # "accept" / "deny"
    av_probestand: str
```

Hieronder bevindt zich het algemene model waar alle data bij wordt ingevoegd en doorgestuurd naar Algemene Component 2(Analyser)

```python
class FirewallConfig(TypedDict):
    """Common normalized model -- output of any parser."""

    hostname: str  # Mandatory: from     global:system_global:hostname
    config_version: str  # From      config-version=      , "" if absent
    global_system_congfiguration: GlobalSystemCongfiguration
    system_ntp_configuration: SystemNtpConfiguration
    system_fortigaurd_configuration: SystemFortigaurdConfiguration
    system_admin_configuration: list[SystemAdminConfiguration]
    system_password_policy: SystemPasswordPolicy
    system_snmp_configuration: SystemSnmpConfiguration
    system_ssl_vpn_configuration: SystemSslVpnConfiguration
    ips_global: IpsGlobal
    system_high_availibility: SystemHighAvailibility
    system_interface_configuration: list[SystemInterfaceConfiguration]
    vdom_configuration: list[VdomConfiguration]
'''
```

Voordelen:

- Type-safe (mypy validation)
- IDE autocomplete & self-documenting
- Validation-ready (Pydantic integratie)
- Makkelijk serialiseerbaar naar JSON/YAML

3.3.2. Algemene Component 2: Analyser

Bestand: `Fortinet/analyser.py

De analyser voert alle 41 security best-practicechecks uit en genereert scores (0, 50, of 100) met evidence:

BP_DEFINITIONS REGISTRY:

De 41 checks zijn gedefinieerd in `BP_DEFINITIONS: list[BestPractice]` met volgende structuur per check:

```python
BestPractice(
    name="BP_FORTIFW_BOOL_TIMEZONE",
    title="Descriptive title",
    description="Why this matters for security",
    technical_implementation="What to check in configuration",
    scoring_implementation="How to score (binary/percentage)",
    category="Category name",
    vendor="FORTINET_FIREWALL"
)
```

CHECK REGISTRIES:

Twee registries voor conditionele uitvoering:

2. **_BP_CHECK_REGISTRY** (20 checks) - Global/Admin configuration
    - Timezone, NTP, Lockout policy, HTTPS/SSH, 2FA, Password policy, SNMP, IPS, SSL-VPN, HA modes, etc.

3. **_BP_CHECK_REGISTRY_VDOM** (21 checks) - VDOM/Network configuration
    - NGFW mode, any-any rules, SSL-SSH inspection, security probestands, logging, zones, IPsec, SD-WAN, DoS, etc.

ANALYSER OUTPUT:

```python
class FortinetAnalyser(BaseAnalyser):
    def analyse(self, config: FirewallConfig) -> FortinetAnalysis:
        # Returns scores + evidence for all 41 checks
```

3.3.3. Algemene Component 3: Reporter

Dit bestand heeft als doel om alles wat we hiervoor van data analyse hebben gedaan om te vormen naar een html bestand zodat dit gemakkelijk kan gelezen worden.

4. Besluit

Dit document beschrijft de uitwerking van mijn stageopdracht bij Cyfora rond het Fortinet Firewall Automated Framework Compliance Testing-project. Tijdens deze stage heb ik niet alleen gewerkt aan de technische realisatie van het framework, maar ook aan het begrijpen van de bestaande platformarchitectuur, de onderliggende datastromen en de manier waarop Cyfora compliance-analyse structureert. De eerste ervaring met de Palo Alto-analyser bleek daarbij een belangrijke basis, omdat ik daardoor snel vertrouwd raakte met de bestaande code, de manier van werken en de kwaliteitsnormen binnen het bedrijf.

Uit de analysefase bleek dat Fortinet-firewallconfiguraties in de praktijk niet altijd direct bruikbaar zijn voor automatische verwerking. Vooral de YAML-export kon syntactisch ongeldige elementen bevatten, onder andere door bekende exportproblemen zoals BUG 976722. Dat maakte duidelijk dat een robuuste oplossing nodig was die zowel fouten in configuratiebestanden kan herstellen als verschillende configuratiestructuren correct kan interpreteren. Op basis daarvan koos ik voor een line-by-line repair-aanpak en een modulaire architectuur die zowel YAML-export als API-data kan verwerken.

De implementatiefase resulteerde in een volledig framework dat 41 security best practices kan controleren en de resultaten kan omzetten naar duidelijke scores en evidence. Door het gebruik van gedeelde datamodellen, parsers, analyzers en rapportering blijft de oplossing aansluitend op Cyfora's bestaande platform en blijft verdere uitbreiding mogelijk. De keuze om zowel een YAML- als een API-ingang te ondersteunen maakt het systeem flexibel voor verschillende gebruikssituaties, zoals offline compliance-audits en live monitoring.

Het belangrijkste resultaat van dit project is dat de oplossing klaar is voor gebruik binnen het platform en waarde toevoegt voor klanten die firewallconfiguraties automatisch willen laten controleren. Voor mij persoonlijk was deze stage bijzonder leerzaam omdat ik heb ervaren hoe een technisch project binnen een bedrijf effectief wordt opgezet, ontwikkeld en gevalideerd. Ik heb niet alleen mijn kennis van beveiliging, Python en configuratieverwerking verdiept, maar ook geleerd hoe belangrijk onderhoudbaarheid, documentatie en testen zijn bij professionele softwareontwikkeling. Hierdoor kijk ik terug op een stage die zowel technisch uitdagend als inhoudelijk zeer waardevol was.

Ondanks de succesvolle creatie blijven enkele beperkingen bestaan. De YAML-parser blijft afhankelijk van de huidige exportstructuur van FortiOS, waardoor toekomstige firmwarewijzigingen mogelijk bijkomende aanpassingen vereisen. Daarnaast werd het framework ontwikkeld en getest op een beperkt aantal FortiGate-platformen, waardoor verdere validatie op grotere modellen mogelijks nodig is.

LITERATUURLIJST

Fortinet. Fortinet documentation. https://docs.fortinet.com/

Fortinet. Fortinet. https://www.fortinet.com/

Fortinet. Fortinet community. https://community.fortinet.com/

Palo Alto Networks. Palo Alto Networks documentation. https://docs.paloaltonetworks.com/

Palo Alto Networks. Palo Alto Networks. https://www.paloaltonetworks.com/

PyYAML Project. PyYAML. https://pyyaml.org/

Python Software Foundation. Python documentation. https://docs.python.org/3/

Pytest developers. pytest documentation. https://docs.pytest.org/

Bijlagen

fortinet-firewall-report.html

Deze kan u vinden op mijn portfolio website onder evidences op de volgende url: https://quinten-vdw.be/internship.html