4 scripts Python d'enrichissement d'information pour TheHive :

ip_malicious_analyzer.py :

```
#!/usr/bin/env python3
# encoding: utf-8

import requests
from cortexutils.analyzer import Analyzer

class IPMaliciousAnalyzer(Analyzer):
    def __init__(self):
        Analyzer.__init__(self)

        # **Récupération de l'IP à analyser et de son type (observable)**
        self.data_to_analyze = self.get_param("data", None, "No IP provided.")
        self.data_type = self.get_param("dataType", None, "")

        # **Récupération de la clé API définie dans la configuration**
        self.api_key = self.get_param("config.api_key", None, "Missing API key in configuration.")

        # **Définition de l'URL de base pour l'API VirusTotal**
        self.api_url = "https://www.virustotal.com/api/v3/ip_addresses/"

    def run(self):
        report = None

        # **On vérifie que le type d'observable est bien une adresse IP**
        if self.data_type == "ip":
            try:
                # **Préparation des en-têtes pour l'appel API**
                headers = {
                    "x-apikey": self.api_key,
                    "Accept": "application/json"
                }
```

```
            # **Construction de l'URL finale pour l'appel à VirusTotal**
            url = f"{self.api_url}{self.data_to_analyze}"

            # **Envoi de la requête GET à VirusTotal**
            response = requests.get(url, headers=headers)

            # **Traitement de la réponse selon le code HTTP reçu**
            if response.status_code == 200:
                report = response.json()
            elif response.status_code == 400:
                self.error("Bad request. (Invalid IP or parameters)")
            elif response.status_code == 401:
                self.error("Unauthorized: invalid API key.")
            elif response.status_code == 429:
                self.error("Rate limit exceeded. (Too many requests)")
            else:
                self.error(f"VirusTotal API error: {response.status_code} - {response.text}")

        except Exception as e:
            # **Gestion des erreurs inattendues**
            self.unexpectedError(e)

        # **Envoi du rapport d'analyse à TheHive**
        self.report(report)

    else:
        # **Si l'observable n'est pas une IP, on retourne une erreur**
        self.notSupported("This analyzer only supports IP addresses.")

def artifacts(self, raw):
    """
    **Construction des artifacts à envoyer à TheHive**
    Permet d'enrichir l'analyse avec des éléments contextuels (ex: pays d'origine)
    """
    artifacts = []
```

```
        if raw and "data" in raw and isinstance(raw["data"], dict):
            attributes = raw["data"].get("attributes", {})
            country = attributes.get("country")
            if country:
                artifacts.append(
                    self.build_artifact(
                        "location",
                        country,
                        tags=["ip_malicious_analyzer"],
                        message="Country of the IP (via VirusTotal)"
                    )
                )
        return artifacts

    def summary(self, raw):
        """
        **Création de taxonomies pour TheHive**
        Sert à générer un tag visible dans l'interface avec un score de réputation.
        """
        taxonomies = []
        level = "info"
        namespace = "IPMaliciousAnalyzer"
        predicate = "VirusTotalReputationScore"

        if raw and "data" in raw and isinstance(raw["data"], dict):
            attributes = raw["data"].get("attributes", {})
            score = attributes.get("reputation", "unknown")
            taxonomies.append(
                self.build_taxonomy(level, namespace, predicate, str(score))
            )

        return {"taxonomies": taxonomies}

# **Point d'entrée du script – lance l'analyse**
if __name__ == "__main__":
```

```
IPMaliciousAnalyzer().run()
```

---

file_malicious_analyzer.py :

```
#!/usr/bin/env python3
# encoding: utf-8

import requests
from cortexutils.analyzer import Analyzer

class VirusTotalFileAnalyzer(Analyzer):
    def __init__(self):
        Analyzer.__init__(self)

        # **Récupération de la donnée à analyser (hash, fichier ou nom de fichier)**
        self.data_to_analyze = self.get_param("data", None, "Aucun hash fourni.")

        # **Type d'observable reçu**
        self.data_type = self.get_param("dataType", None, "")

        # **Récupération de la clé API dans la configuration**
        self.api_key = self.get_param("config.api_key", None, "Clé API manquante.")

        # **URL de l'API VirusTotal pour les fichiers**
        self.api_url = "https://www.virustotal.com/api/v3/files/"

    def run(self):
        report = None

        # **Vérifie si le type de donnée est accepté**
        if self.data_type in ["file", "hash", "filename"]:
            try:
                # **Préparation des en-têtes pour la requête HTTP**
                headers = {
```

```
            "accept": "application/json",
            "x-apikey": self.api_key
        }

        # **Construction de l'URL complète avec le hash à analyser**
        url = f"{self.api_url}{self.data_to_analyze}"

        # **Affiche dans la console l'URL utilisée et le hash analysé (utile pour le debug)**
        print(f"Envoi de la requête à {url} avec le hash {self.data_to_analyze}")

        # **Envoi de la requête GET à l'API VirusTotal**
        response = requests.get(url, headers=headers)

        # **Affichage du code de réponse et du contenu (debug)**
        print(f"Réponse de l'API: {response.status_code}")
        print(f"Contenu de la réponse: {response.text}")

        # **Traitement de la réponse selon le code HTTP**
        if response.status_code == 200:
            report = response.json()
        elif response.status_code == 404:
            self.error("Hash introuvable sur VirusTotal.")
        elif response.status_code == 403:
            self.error("Clé API invalide ou quota dépassé.")
        else:
            self.error(f"Erreur API VirusTotal: {response.status_code} - {response.text}")

    except Exception as e:
        # **Gestion des erreurs inattendues**
        self.unexpectedError(e)

    # **Retourne le rapport à TheHive**
    self.report(report)

else:
```

```
        # **Lance une erreur si le type d'observable n'est pas reconnu**
        self.notSupported("Cet analyseur ne supporte que les types de données fichier, hash ou nom de fichier.")

# **Point d'entrée du script**
if __name__ == "__main__":
    VirusTotalFileAnalyzer().run()
```

---

ldap_machine_info.py :

```
#!/usr/bin/env python3

import json
import ldap
from cortexutils.analyzer import Analyzer

class LDAPMachineInfo(Analyzer):
    def __init__(self):
        Analyzer.__init__(self)

        # **Définition de l'URL du serveur LDAP**
        self.ldap_url = "ldap://127.0.0.1:389"

        # **Récupération des identifiants de connexion au serveur LDAP depuis la config**
        self.ldap_bind_dn = self.get_param("config.ldap_bind_dn", None, "Missing LDAP bind DN")
        self.ldap_bind_password = self.get_param("config.ldap_bind_password", None, "Missing LDAP bind password")

        # **Nom de la machine à rechercher (observable envoyé depuis TheHive)**
        self.machine_name = self.get_param("data", None, "Missing machine name")

    def decode_ldap_entry(self, entry):
        """
        **Convertit les valeurs LDAP en chaînes lisibles**
        LDAP retourne des objets en bytes, donc on les décode proprement en UTF-8.
```

```
    """
    if isinstance(entry, dict):
        return {
            k.decode('utf-8') if isinstance(k, bytes) else k:
            [v.decode('utf-8') if isinstance(v, bytes) else v for v in vals]
            for k, vals in entry.items()
        }
    return entry


def run(self):
    # **Affiche le nom de la machine analysée (utile pour le debug)**
    print(f"🚀 Début du script avec machine_name: {self.machine_name}")
    try:
        # **Initialisation et authentification sur le serveur LDAP**
        conn = ldap.initialize(self.ldap_url)
        conn.simple_bind_s(self.ldap_bind_dn, self.ldap_bind_password)
        print("✅ Connexion LDAP réussie")

        # **Création du filtre de recherche LDAP avec le nom de la machine**
        search_filter = f"(uid={self.machine_name})"

        # **Recherche dans la base LDAP (portée sur tout le sous-arbre)**
        result = conn.search_s("dc=echelon,dc=local", ldap.SCOPE_SUBTREE, search_filter)
        print(f"🔍 Résultat LDAP : {result}")

        if result:
            # **On décode les données de la machine trouvée pour les rendre lisibles**
            machine_info = self.decode_ldap_entry(result[0][1])

            # **On envoie le résultat à TheHive**
            self.report(machine_info)
            print(f"📊 Machine trouvée : {json.dumps(machine_info, indent=2)}")
        else:
            # **Erreur si aucune machine ne correspond au filtre**
            self.error(f"Aucune machine trouvée : {self.machine_name}")
```

```
            # **On ferme proprement la connexion LDAP**
            conn.unbind_s()

        except ldap.LDAPError as e:
            # **Gestion des erreurs spécifiques à LDAP**
            print(f"❌ Erreur LDAP : {e}")
            self.error(f"LDAP query failed: {e}")

        except Exception as ex:
            # **Gestion des erreurs inattendues**
            print(f"⚠️ Erreur Inattendue : {ex}")
            self.error(f"Unexpected Error: {ex}")

# **Point d'entrée du script : exécution principale**
if __name__ == "__main__":
    LDAPMachineInfo().run()
```

---

ldap_user_info.py :

```
#!/usr/bin/env python3

import json
import ldap
from cortexutils.analyzer import Analyzer

class LDAPUserInfo(Analyzer):
    def __init__(self):
        Analyzer.__init__(self)

        # **Adresse du serveur LDAP local**
        self.ldap_url = "ldap://127.0.0.1:389"
```

```
    # **DN (Distinguished Name) utilisé pour l'authentification LDAP**
    self.ldap_bind_dn = self.get_param("config.ldap_bind_dn", None, "Missing LDAP bind DN")

    # **Mot de passe associé au DN pour se connecter à LDAP**
    self.ldap_bind_password = self.get_param("config.ldap_bind_password", None, "Missing LDAP bind password")

    # **Nom d'utilisateur reçu comme observable depuis TheHive**
    self.username = self.get_param("data", None, "Missing username")

def decode_ldap_entry(self, entry):
    """
    **Fonction utilitaire pour convertir les données LDAP en format lisible**
    (bytes → str), nécessaire pour générer un JSON valide pour TheHive.
    """
    if isinstance(entry, dict):
        return {
            k.decode('utf-8') if isinstance(k, bytes) else k:
            [v.decode('utf-8') if isinstance(v, bytes) else v for v in vals]
            for k, vals in entry.items()
        }
    return entry

def run(self):
    # **Début de l'analyse : affichage de l'utilisateur concerné**
    print(f"🚀 DÉMARRAGE du script avec username: {self.username}")
    try:
        # **Connexion au serveur LDAP**
        conn = ldap.initialize(self.ldap_url)
        conn.simple_bind_s(self.ldap_bind_dn, self.ldap_bind_password)
        print("✅ Connexion LDAP réussie")

        # **Filtre de recherche basé sur l'attribut uid (nom d'utilisateur)**
        search_filter = f"(uid={self.username})"

        # **Recherche dans le domaine LDAP**
```

```
            result = conn.search_s("dc=echelon,dc=local", ldap.SCOPE_SUBTREE, search_filter)
            print(f"🔍 Résultat LDAP : {result}")

            if result:
                # **Conversion des données brutes LDAP en format lisible**
                user_info = self.decode_ldap_entry(result[0][1])

                # **Transmission du résultat à TheHive**
                self.report(user_info)
                print(f"📊 Utilisateur trouvé : {json.dumps(user_info, indent=2)}")
            else:
                # **Gestion du cas où aucun utilisateur correspondant n'est trouvé**
                self.error(f"Aucun utilisateur trouvé : {self.username}")

            # **Fermeture propre de la session LDAP**
            conn.unbind_s()

        except ldap.LDAPError as e:
            # **Gestion des erreurs LDAP spécifiques**
            print(f"❌ Erreur LDAP : {e}")
            self.error(f"LDAP query failed: {e}")

        except Exception as ex:
            # **Gestion des erreurs générales**
            print(f"⚠️ Erreur Inattendue : {ex}")
            self.error(f"Unexpected Error: {ex}")

# **Point d'entrée du script — déclenche l'analyse**
if __name__ == "__main__":
    LDAPUserInfo().run()
```