Deep Agent architektúrák a biztonsági logelemzésben

Pataki Gréta, Orbán Gábor

2026. április 21.

Agenda

1. Projekt bemutatás
2. Előzetes elemzés
3. Workflow tervezés
4. Prompt tervezés
5. Trade-offs
6. Benchmarkok kialakítása, értékelés

Projekt bemutatása

Az ügyfél

- Az Egyesült Államok egyik legnagyobb telekommunikációs szolgáltatója – internet, mobil, kábel-TV –, közel 7 millió ügyféllel
- Több száz Kubernetes klasztert és több ezer microservice-t üzemeltet multi-cloud környezetben (GCP, AWS)
- Dedikált biztonsági csapat felügyeli az infrastruktúrát, amelynek napi munkáját **HashiCorp Vault audit logok, Kubernetes Gatekeeper policy-riasztások és AWS GuardDuty alertek** támogatják.

A probléma

- **Napi ~1,5 millió biztonsági log érkezik**: ennek nagyobbik része Kubernetes Gatekeeper esemény
- **Alert fatigue**: a biztonsági elemzők kapacitása véges; a riasztások zömét a rutin, alacsony kockázatú események teszik ki, amelyek elfedik a valóban kritikus jelzéseket
- A jelenlegi eszközök (SIEM, dashboardok) aggregálnak és vizualizálnak, de nem végeznek tartalmi elemzést: **nem ismernek fel cross-resource mintázatokat, nem priorizálnak** üzleti kontextus alapján, és nem generálnak cselekvésre kész ajánlásokat
- **Reaktív működési modell**: az elemzők akkor kezdenek vizsgálódni, amikor egy incidens már megtörtént – ahelyett, hogy a logokból kiolvasható korai jelekből proaktívan megerősítenék az infrastruktúrát

A célkitűzés: AI alapú támogatás

- **Napi ütemezett futás:** a rendszer automatikusan felolvassa, előszűri és elemzi a logokat, majd priorizált, cselekvésre kész finding-ok formájában e-mailben küld napi riportot a biztonsági csapatnak
- A cél az SOC csapat képességeik kiterjesztése: a másfél millió nyers logból **kiemelni azt a 10–15 valóban fontos megállapítást**, amely azonnali figyelmet érdemel
- A megoldás **deep agent architektúrára** épül: többlépéses, autonóm reasoning, code interpreter alapú logelemzés, kontextusfüggő priorizálás és napi trendkövetés

Előzetes elemzés – logtípusok vizsgálata

KUBERNETES GATEKEEPER LOGOK

- A Gatekeeper egy Kubernetes-natív policy engine (OPA alapú), amely az admission controller rétegben ellenőrzi, hogy a klaszterbe érkező erőforrás-kérések megfelelnek-e a szervezet biztonsági szabályainak.
- Minden egyes pod létrehozáskor, módosításkor lefut a policy kiértékelés, és az eredmény – warn, deny vagy dryrun – logba kerül.

Jellemzők

- **Nyers formátum:** JSON, 30+ mező/record
- **Napi mennyiség:** a ~1,5M log nagyobbik része, rendkívül zajos adatforrás
- **10 különböző constraint típus (policy szabály)**
- **Több mint 100 egyedi klaszter és namespace, prod és non-prod vegyes környezet**

HASHICORP VAULT AUDIT LOGOK

- A HashiCorp Vault egy titkoskulcs- és hozzáférés-kezelő rendszer (secrets management), amely minden API-hívásról részletes audit logot ír
- A logok tartalmazzák a hitelesítési kísérleteket, secret-olvasásokat, konfigurációs műveleteket és hibákat

Jellemzők

- **Nyers formátum:** JSON, mélyen beágyazott struktúra (40+ mező)
- **Napi mennyiség:** kisebb volumen, de magasabb biztonsági relevancia, rendkívül magas zajszint

Előzetes elemzés – priorizálás és előszűrés

Előszűrési stratégia

1. deny kiszűrése (warn logok megtartása ~82% - szűrés után jelentős volumencsökkenés)
2. dryrun kiszűrése (nem aktív szabály)
3. CronJob deduplikáció, ismétlődő konfigurációs lekérdezések deduplikálása
4. sandbox környezetek kiszűrése

KUBERNETES GATEKEEPER LOGOK

Kulcsmezők az elemzéshez

- Milyen policy-t sértettek meg
- Milyen szintű a riasztás
- Mi az érintett erőforrás és hol található
- Melyik klaszterben történt
- Ki vagy mi indította a műveletet (ember, CI/CD, CronJob

HASHICORP VAULT AUDIT LOGOK

Kulcsmezők az elemzéshez

- Milyen típusú művelet
- Melyik Vault útvonalat érintette
- Ki, honnan, milyen jogosultsággal érte el
- A kérés eredete (ismert service vs. ismeretlen forrás)
- Sikertelen műveletek és hibaüzenetek
- Token életciklus és lejárati idők

Referencia – Prisma Cloud

A Wiz / Prisma Cloud referenciaként szolgált:

- Hasonló output-struktúra: priorizált finding-ok, severity, affected resources
- Enrichment szemlélet: a nyers logot kontextussal gazdagítani (mi a resource, miért fontos, mi a javasolt lépés)

Prisma Cloud

- A Prisma Cloud egy teljes CNAPP suite, amelyet a Palo Alto Networks fejleszt. Agent-alapú és agentless képességeket iskombinál. Fő fókusza:
  - Felhős konfiguráció folyamatos monitorozása,
  - Konténer runtime védelem, runtime rule-ok,
  - Image vulnerability scanning (CI/CD pipeline-ba integrálva),
  - Admission controller integráció (hasonló a Gatekeeper-hez),
  - IAM jogok elemzése

Referencia - Wiz

Mit figyel:

- **Felhős konfigurációs hibák** felismerése (pl. nyitott S3 bucket, publikus IP-vel rendelkező VM, titkosítatlan adatbázis)
- **Compliance ellenőrzés** (CIS, SOC2, NIST, PCI-DSS stb.)
- Futó konténerek és VM-ek **sebezhetőségeinek** szkennerje, malware detektálás a workload-okban, runtime viselkedéselemzés
- **Túljogosított** IAM role-ok és service account-ok felismerése, effective permission elemzés (ténylegesen mit ér el egy identitás)
- Klaszter-szintű konfigurációs hibák (RBAC, PSP, network policy), Konténer imagesebezhetőségek, Admission control gap-ek, OS és alkalmazás sebezhetőségek (CVE-k)
- **Prioritizálás** kontextus alapján: nem csak a CVSS score számít, hanem az is, hogy az érintettworkload elérhető-e kívülről, és van-e aktív exploit

Wiz

A **Wiz** egy **agentless** (ügynök nélküli) **felhőbiztonsági platform**, amely API szinten csatlakozik a felhőszolgáltatókhoz (AWS, GCP, Azure), és egyetlen átfogó Security Graph-ot épít az infrastruktúráról.

A Wiz kimenete:

Priorizált finding-ok, amelyek tartalmazzák:

- **Severity** (Critical / High / Medium / Low / Informational)
- **Érintett erőforrás** (resource type, name, account, region)
- **Leírás**: mi a probléma
- **Remediation**: hogyan lehet javítani
- **Context**: milyen attack path-ban szerepel az adott finding
- **Evidence**: konkrét konfiguráció vagy log részlet

Hogyan tervezünk meg egy automatizációs workflow-t?

A legfontosabb elv

Először a folyamatot és a követelményeket kell megérteni, a technológiai megoldásokat csak ezután érdemes hozzátársítani.

Bemenetek

Mekkora és milyen az adat? Mekkora a volumen, mekkora a heterogenitás, mennyire változékony a környezet? Egy stabil, jól strukturált input egészen más megközelítést igényel, mint egy dinamikus, sokszínű adatfolyam.

Lépések

Milyen diszkrét feladatokra bontható a folyamat? Ezek egymás után következnek, vagy párhuzamosan futhatnak? Milyen kontextuális függések vannak köztük, melyik lépés eredményére van szüksége a következőnek?

Kimenet

Mennyire szigorú a struktúra? Egy zárt végű, fix formátumú output és egy nyílt végű, komplex követelményrendszernek megfelelő eredmény teljesen eltérő megoldásokat kíván.

Services

1. DATA INGESTION

- **Üzleti funkció: „A napi ~1,5M logból kiszűrni a releváns jeleket"**
- A nyers logokat az ügyfél S3 bucket-jeiből olvassuk fel
- Elvégezzük a szükséges szűréseket és átadjuk az agent-nek az utolsó napi adatokat.
- Eredmény: a másfél millió nyers logból egy tisztított, kezelhető méretű adatforrás, amit az AI-elemzés érdemben fel tud dolgozni

2. ANALYST AGENT

- **Üzleti funkció: „Megtalálni mindent, ami biztonsági szempontból figyelmet érdemel"**
- A megtisztított logokat Code Interpreter segítségével elemzi:
- Először felépíti a „normális" képet (baseline), majd az ettől való eltéréseket keresi, nem statikus szabályokkal, hanem kontextusfüggő elemzéssel
- Minden gyanús mintázatot egy-egy specializált sub-agentnek delegál, amely mélyelemzést végez és strukturált finding-ot készít
- A finding tartalmazza: mi a probléma, miért fontos, és hogyan lehet kivizsgálni ezek az ügyfél egyik legfontosabb elvárásai voltak
- Eredmény: 15+ finding, amelyek lefedik a teljes logkészletet

3. PRIORITIZATION AGENT

- **Üzleti funkció: „A biztonsági csapat reggelente ne 15 finding-ot lásson, hanem a legkritikusabb 5-öt"**
- Az ügyfél elvárása egyértelmű volt: a napi riportban a TOP 5 finding legyen kiemelve – amit reggel az e-mail megnyitásakor azonnal át tudnak tekinteni
- A Trend Analyzer sub-agent a korábbi napok finding-jait veti össze az aktuálisakkal - ha egy probléma napok óta fennáll és nem javítják, az prioritást kap
- **A priorizálás hat szempont mentén történik:**
  - A finding súlyossága
  - Production vagy non-prod környezet érintettsége
  - Trend: visszatérő, növekvő, vagy új probléma?
  - Blast radius: hány erőforrást érint?
  - Exploitability: mennyire könnyű kihasználni?
  - Észlelhetőség: észrevennék-e időben, ha megtörténik?

4. REPORT GENERATION

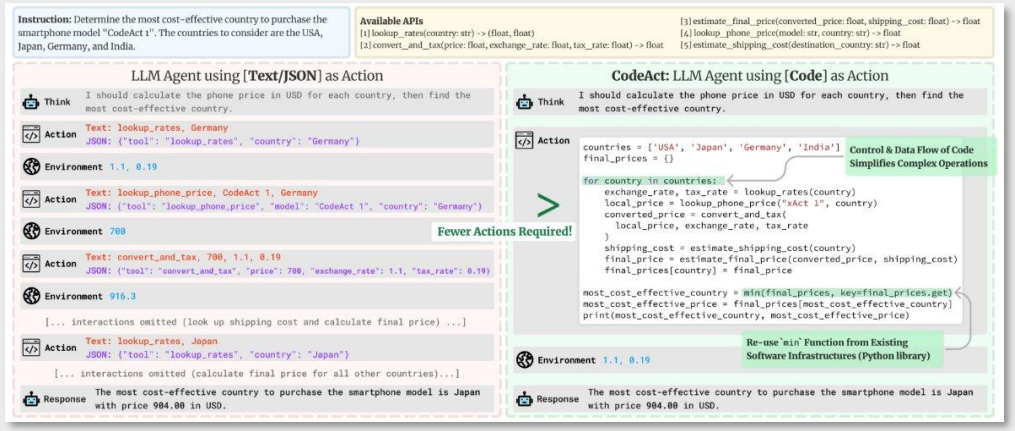
- **Üzleti funkció: „A napi riportot úgy tálalni, hogy a biztonsági csapat azonnal cselekedni tudjon"**
- Az ügyfél elvárása egy olyan e-mail riport volt, amely:
- Tartalmaz egy összefoglaló dashboardot (hány issue, hány critical, hány log elemezve, időintervallum),
- Illetve a TOP 5 priorizált finding-ot kártyaszerűen, kinyitható részletekkel – cím + severity egy sor, kattintásra/kinyitásra látható a részletes leírás és a vizsgálati lépések
- Az investigation step-ek konkrétak: „nyisd meg a GCP console-t, keresd meg az X pod-ot az Y namespace-ben, ellenőrizd a Z beállítást, nem általánosságok
- Severity szerinti rendezés (CRITICAL - HIGH - MEDIUM - LOW)
- Az ügyfél visszajelzése alapján a remediation mellett a vizsgálati lépések kaptak elsőbbséget

CodeAct - Miért nem csak NLP?

A probléma: skála és zaj

A vizsgált logfájlok százezres nagyságrendű bejegyzést tartalmaznak. Ha ezeket közvetlenül az LLM kontextusába töltjük, két súlyos problémával szembesülünk:

- Kontextusszennyezés: a rengeteg irreleváns bejegyzés elveszi a modell figyelmét a valóban fontos eseményektől
- Minőségi összefüggések elvesznek: a zaj elfedi a statisztikailag szignifikáns mintázatokat



A megoldás: Code Interpreter Tool

Iteratív folyamat:

01 A logok **CSV formátumba** konvertálódnak

02 Az agent **Python kódot ír és futtat.** Pandas segítségével statisztikailag feltárja az adatokat

03 A **kód outputja** kerül vissza a kontextusba

04 A modell ezt az eredményt **természetes nyelvvel értelmezi és dönt a tovább haladásról**

Orkesztráció – Deep Agent architekrúra

Egy egyszerű agent egy **reason–act loop**: gondolkodik, eszközt hív, megkapja az eredményt, folytatja. Ez sok feladatra elegendő, de komplex, hosszan futó folyamatoknál hamar falba ütközik.

Mi tesz egy agentet "deep"-pé?

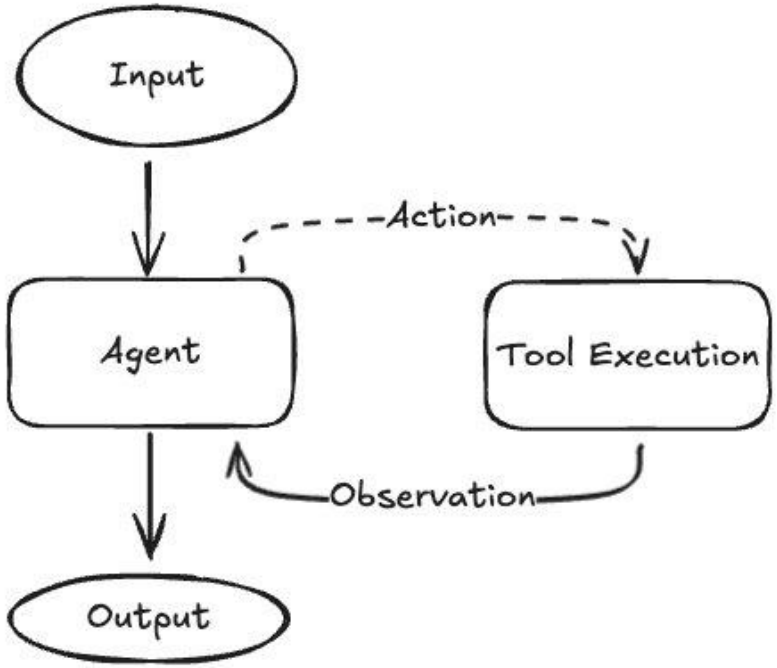
A Deep Agent ugyanarra az alapalgoritmusra épül (reason-act loop) de fel van szerelve azokkal a képességekkel, **amelyek komplex, hosszú futású feladatokhoz** szükségesek:

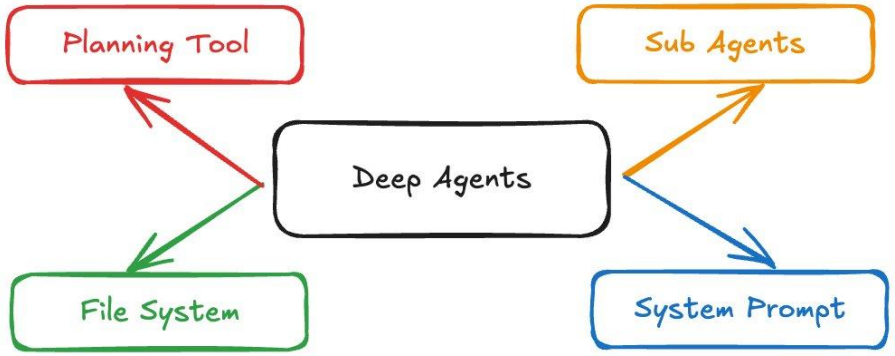
Tervezés és haladáskövetés: a feladatot részfeladatokra bontja, nyomon követi az előrehaladást, és menet közben korrigálja a tervet

Subagentek: egyes részfeladatokat dedikált subagenteknek delegál, amelyek izolált kontextusban, fókuszáltan dolgoznak

Fájlrendszer mint memória: az útközben szerzett információkat, részeredményeket fájlként menti ki, és amikor szükséges, visszaolvassa őket

Hosszú távú memória: a releváns kontextust megőrzi és kezeli a teljes futás során, nem csak az aktuális lépés szintjén

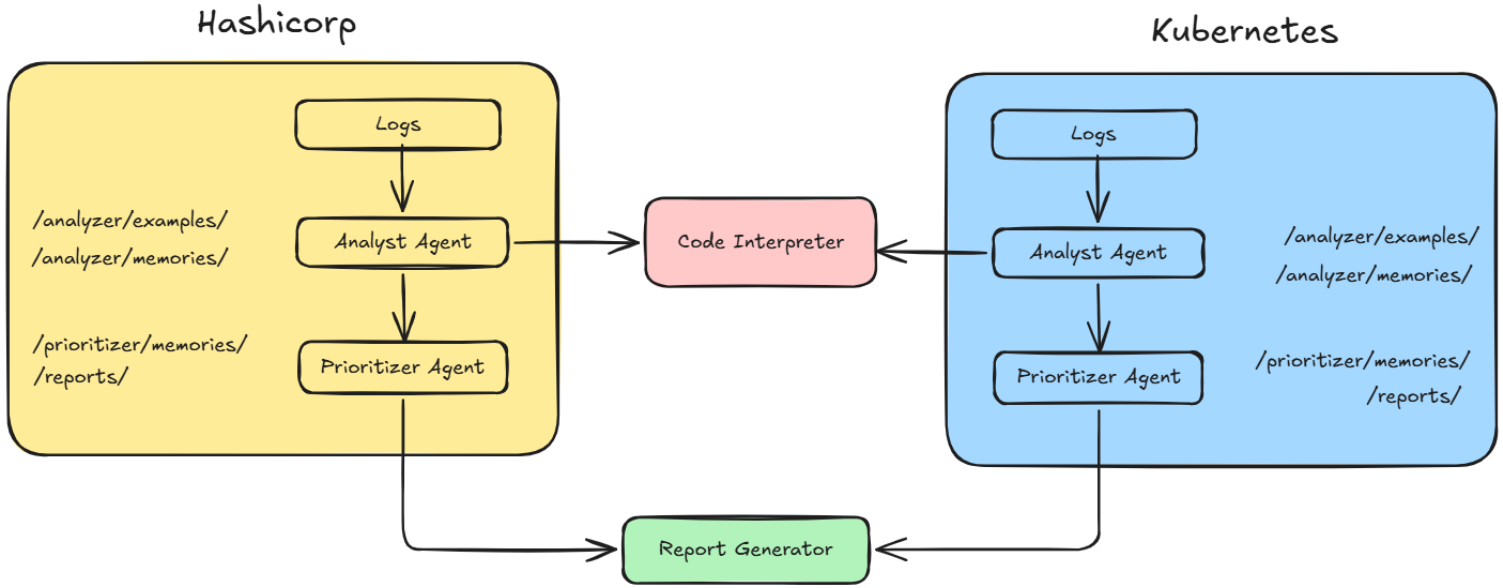




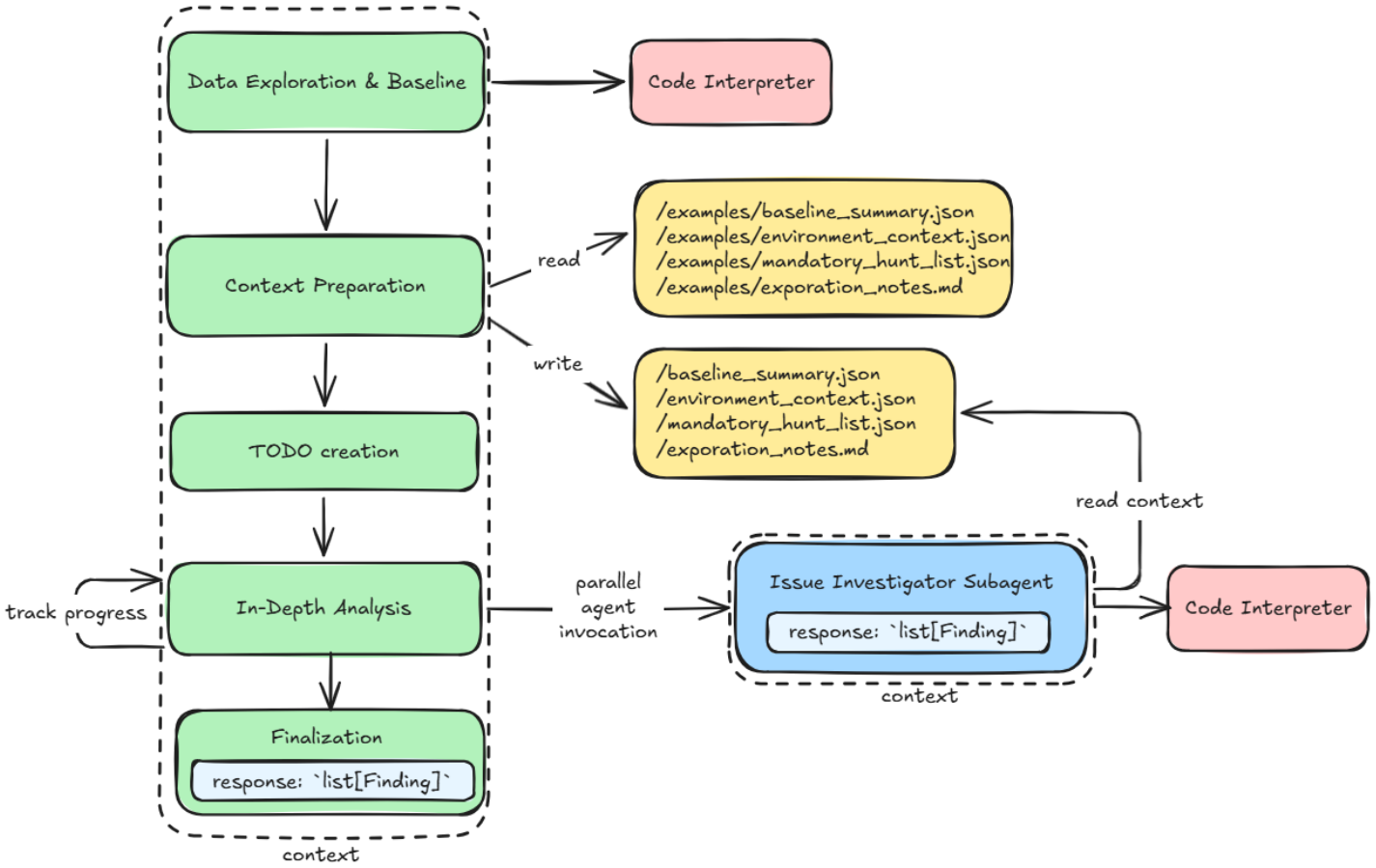
Workflow architektúra

Workflow

- **Moduláris agent nodeok:** az analízis és a priorizáció külön agent nodeon fut, mindegyik node saját deep agenttel rendelkezik. Ez jobb követhetőség és debuggolhatóság érdekében
- **Izolált fájlrendszer per agent:** minden agent saját composite fájlrendszerrel rendelkezik - útvonal prefix alapján külön read/write policy szabályozza, mit olvashat és írhat
- **Techstack-specifikus workflowok**



Analyst Agent Belülről



Analyst Agent Belülről

- **Progressive disclosure**: a kontextusba az információk just-in-time kerülnek be
- **Write once, read many**: az előzetes feltárás eredményei egyszer kerülnek kiírásra fájlba; a mélységi elemzés során az agent minden alkalommal csak felolvassa őket.
- **TODO lista mint haladáskövetés**: az agent az elemzési pontokat egy TODO listában rögzíti, és végighaladva rajtuk nyomon követi, hol tart a folyamatban
- **Párhuzamos mélységi elemzés subagentekkel**: az agent párhuzamosan hív Issue Investigator subagenteket; ezek izolált kontextusban dolgoznak.
- **Summarization middleware:** kontextus automatikus tömörítése 70% kihasználtság felett.

Prompt tervezés

V1: Egyszerű ReAct prompt (~2000 karakter)

Tartalma röviden:

- „Te egy cybersecurity szakértő vagy.
- Elemezd a warn logokat a meghatározott elvek és szabályok alapján,
- adj fontossági szintet,
- mutasd a top namespace-eket,
- javasolj akciót."

Kimenet:

- ✓ Strukturált, de szabadabb szöveges válasz
- ✓ Nincs konzisztencia-garancia
- ✓ Nincs kontextus-megosztás
- ✓ Output struktúra a promptban

V2: Deep Agent prompt (15 000+ karakter, 5 fázisú)

Kiegészítések:

- Fájlrendszer, eszközök, sub-agentek, memóriák, Jinja templating, iteráció, ellenőrzés, strukturált output

- **5 fázisú workflow**
  - Data Exploration and Baseline,
  - Context Preparation for subagents,
  - TODO Creation: hunt checklist és baseline deviation alapján,
  - In-Depth Analysis and Finding Creation minden TODO elemre,
  - Final structured report

Kimenet:

- 15+ strukturált finding,
- Validált, strukturált output
- Reprodukálhatóbb, ellenőrizhetőbb

Prompt tervezés

Filesystem leírás 1.

- A prompt leírja az agent számára elérhető fájl-rendszert:

/examples/ /memories/ /reports/

- Közös kontextus-megosztás a sub-agenttel

Tool-ok 2.

- execute_code (pandas),
- read_file / write_file, ls (fájl listázás),
- write todo-s,
- task (sub-agent)

Jinja templating 3.

- Dinamikusan paraméterezhető promptok: futásidejű változó-behelyettesítés:

{{ number_of_findings }}

{{ subagent_execution }}

Iteráció 4.

- TODO-lista vezérli a workflow-t: az agent maga hozza létre, nyomon követi, és frissíti a feladat lista állapotát
- /memories/ olvasása: korábbi futásokból tanult lecke beépítése az aktuális elemzésbe

Ellenőrzés 5.

- Mandatory hunt checklist: előre definiált biztonsági ellenőrzőpontok
- Deduplikáció: az összesítéskor azonos issue-k összevonása
- Mandatory checklist lefedettség ellenőrzés

Strukturált output 6.

- Pydantic modell, JSON kimenet
- Minden finding:
  - id, title, category, confidence_level, confidence_reason, what_you_found, why_it_matters, next_steps_of_investigation

Trade-off: Miben erős?

1.	**Komplex környezetek kezelése:** különösen ott teljesít jól, ahol nem csak a feladat végrehajtása, hanem maga a környezet megértése is összetett feladat
2.	**A logika a promptokban él, nem a gráfban:** az üzleti logika promptokban, skillekben és kontextuális fájlokban lakik, nem merev orkesztrációs gráfokban. Ez két előnyt hoz: az agent rugalmasan alkalmazkodik, és az üzleti felhasználók is könnyebben validálják, hogy a workflow az elvárásaiknak megfelel-e
3.	**Kisebb orkesztrációs komplexitás:** nem kell komplex gráfokat fejleszteni és a folyamat változásakor újraírni
4.	**Kisebb prompt redundancia:** egy gráf alapú megközelítésben minden node-hoz külön promptot kell írni, amelyek mindegyike újramagyarázza a kontextust. Deep agent esetén ez elmarad, így kevesebb promptot kell karbantartani
	Cserébe: egy nagyobb, komplexebb promptot használunk. A logika egyetlen átfogó promptban él, ami ugyan nagyobb és összetettebb, de egyben is kezelhető és validálható

Trade-off: Mire kell figyelni?

1.	**Nagyobb szabadságfok kockázatokkal járhat**: az agent autonómiája miatt fontos a guardrailek bevezetése: max LLM call, max tool call, read-only jogosultságok. A rendszer viselkedését folyamatosan ellenőrizni kell.
2.	**Nem konzisztens végrehajtás:** a futás inkább egy keresési folyamat, mint egy determinisztikus pipeline: sok út vezet a célhoz, de nem mindig az optimális. Memória és few-shot példák segítségével az agent idővel javítható.
3.	**Modellválasztás számít:** kisebb modellek (pl. GPT-5+ mini, nano, Claude Haiku) nem feltétlenül képesek a harness kezelésére. Érdemes supervisor szinten erősebb, subagent szinten kisebb modelleket alkalmazni
4.	**Observability kihívás:** a futás nehezebben átlátható, a trace-ek értelmezése komoly feladat. Fejlesztés közben érdemes streamelni és a streameket átnézni; hosszabb trace-ek elemzéséhez egyre inkább nyelvi modell használatát javasolják
5.	**Költség és futásidő:** könnyen elszaladhat; folyamatos monitorozás és biztonsági költségkorlátok bevezetése ajánlott

Benchmark-kialakítás nehézségei

Hogyan mérjük egy AI-agent elemzésének minőségét,
ha nincs előre címkézett „ground truth" adatunk?

Klasszikus ML benchmark (precision, recall, F1) feltételezi, hogy létezik egy annotált referencia-adatkészlet

Cybersecurity log elemzésnél ez ritkán áll rendelkezésre:

- az elvárt finding-ok kontextusfüggőek, az elemző tapasztalatától és a szervezet prioritásaitól függenek
- PoC fázisban nem áll rendelkezésre hónapok historikus adata sem

Benchmarkok

Lehetséges megközelítés	Mire jó?
Szakértői review (Expert Validation)	Precision proxy: A biztonsági csapat értékeli az egyes finding-okat
Coverage-elemzés (Known Issue Coverage)	Recall proxy: Lefedi-e az agent az ismert problémákat? Előre definiált „must-find" lista összevetése az agent outputjával.
Konzisztencia-teszt (Stability)	Ugyanazt a logkészletet többször lefuttatva mennyire stabil az output? Azonos finding-ok, azonos sorrend?
Finding-minőség értékelése (Output Quality Rubric)	Tartalmi ellenőrzés: a finding-ban hivatkozott számok, erőforrások egyeznek-e a nyers logokkal? Előfordul-e hallucináció?
Összehasonlító baseline (Comparative Baseline)	Egy egyszerűbb megoldás (pl. Rule-based aggregáció, vagy egyetlen LLM-hívás) outputjával való összevetés. (Relatív javulási metrika)
Idővonal-alapú validáció (Temporal Consistency)	A napi futások finding-jainak összevetése: egy probléma megjelenik-e újra? Ha javítják, eltűnik-e?

Értékelés

Agent	Ügyfél visszajelzés	Prompt módosítás
Analyst agent	„A noise ratio félreérthető – 98% zaj vagy releváns?"	Executive summary struktúra egyértelműsítése
Analyst agent	„F-001 sérti a saját szabályarendszerét (warn-only finding)"	Új szabály: „a warn-only jelleg nem finding, csak a konkrét kontroll hiány."
Analyst agent	„Investigation step-ek fontosabbak, mint az azonnali remediation"	Új szabály: „ne erőltesd a remediációt, ha előbb vizsgálat szükséges."
Analyst agent	„Egy orosz IP egyszer is finding legyen, akkor is ha nem ismétlődik"	Új szabály: „egyetlen esemény is lehet finding, ha kontroll-rést tár fel."
Prioritization agent	„A non-prod finding-ok is fontosak – a problémákat még prod előtt akarjuk javítani"	Production weight csökkentése a priorizálási szempontrendszerben
Prioritization agent	„A finding severity nem befolyásolja a priorizálást"	„Severity of the finding" bekerült az #1 priorizálási szempontként

Köszönjük a figyelmet!