striga

Od transformera do ich serwera

Agentic AI w wykrywaniu podatności w open-source
(CVE w httpd, n8n, Ollamie, axios i Tomcacie)

Bartłomiej Dmitruk

The Hack Summit 2026

50+ CVE w 2026

Bartłomiej Dmitruk

- Założyciel Striga
- Offensive security i machine learning
- bartekdmitruk.com

50+ CVE w 2026 w oprogramowaniu używanym przez setki milionów ludzi

Jedna linia przesunięta za daleko

- Łatka na padding oracle (CVE-2026-29146) przeniosła jedno wywołanie za catch
- Fail-closed stał się fail-open: unauth RCE na klastrze (CVE-2026-34486)
- Tomcat jest w OSS-Fuzz od 2022. Żaden z 5 harnessów nie dotyka miejsca podatności

Czy jakikolwiek automat miał szansę wykryć ten błąd?

EncryptInterceptor.java

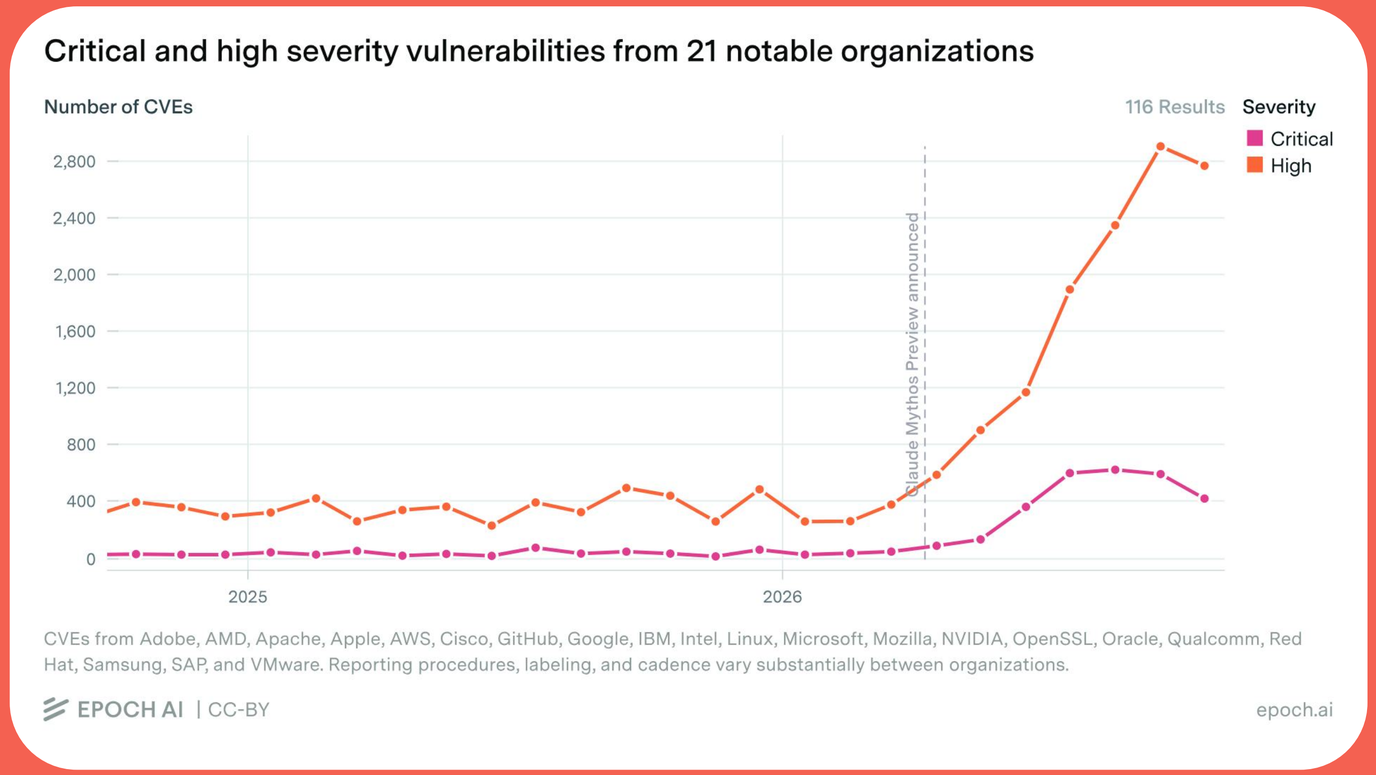
```
  void messageReceived(ChannelMessage msg) {
      try {
          data = encryptionManager.decrypt(data);
          xbb.clear(); xbb.append(data, …);
-         super.messageReceived(msg);
      } catch (GeneralSecurityException gse) {
          log.error("decrypt.failed");
      }
+     super.messageReceived(msg);
  }
```



Agenda

1. **Cyberapokalipsa** — ile CVE i dlaczego tyle
2. **Koszmary maintainerów** — co się dzieje ze zgłoszeniem
3. **Detektory** — od reguł do agentów
4. **Case study** — które klasy detektorów wykryją 10 podatności Strigi?
5. **Co z tym zrobić** — model, pipeline, wnioski

Cyberapokalipsa



- W październiku 2026: 17 059 CVE
- ponad jedna trzecia całego 2025 (48 151)
- Powód? **Bugflation**
- Lepsze, łatwiej dostępne modele

Źródło: https://epoch.ai/data/cve?view=graph

Maintainerzy toną w zgłoszeniach



„the continued flood of AI reports has basically made the security list almost entirely unmanageable”

Linus Torvalds, ogłoszenie Linux 7.1-rc4



“libheif and libde265 are maintained by a single independent developer with almost no recurring funding, while 89 security advisories had to be investigated, fixed and released in 2026 alone.”

Dirk Farin, README repozytorium libheif

Źródło: https://lwn.net/Articles/1073193/, https://github.com/strukturag/libheif

Koszmary maintainerów open-source

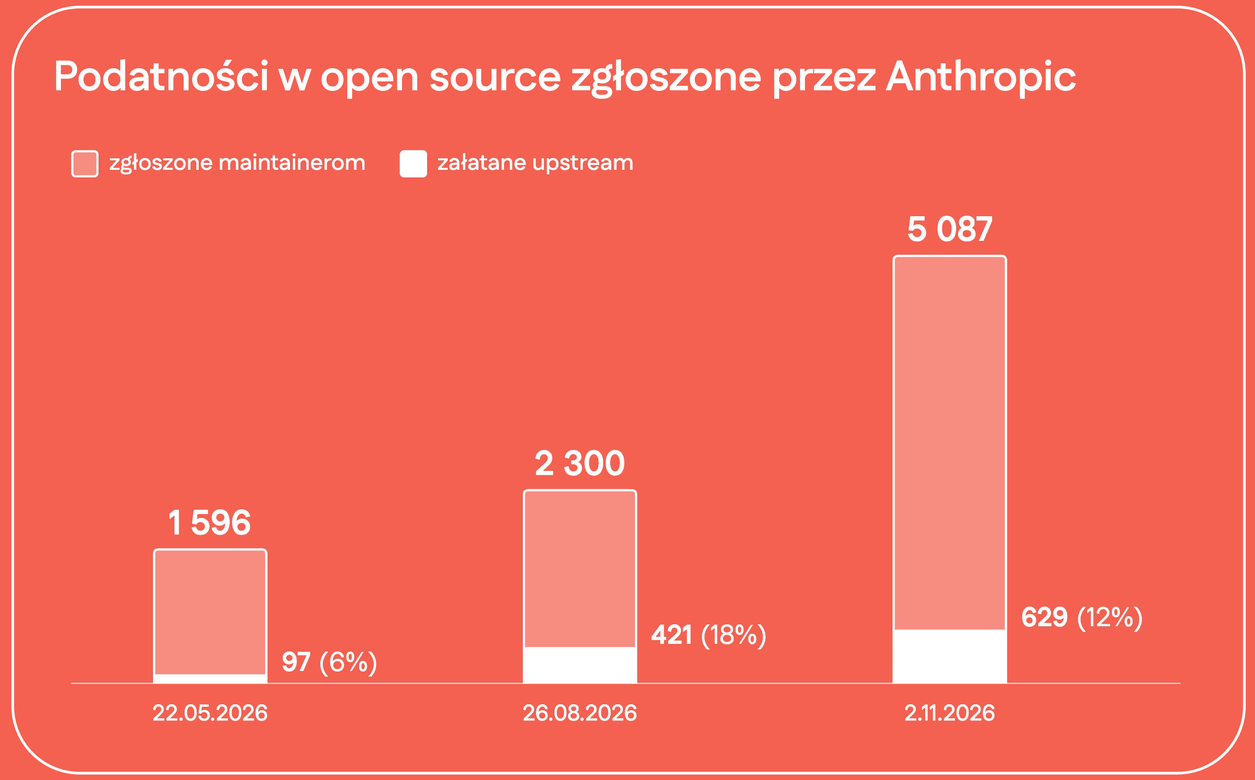
- Każde zgłoszenie = czas niedofinansowanych maintainerów
- Większość zgłoszeń jest słabej jakości
- Brak PoC obniża jakość zgłoszenia
- Nieuruchomiony PoC jest niewiele lepszy od jego braku
- Na co zwrócić uwagę:
  - Osiągalność
  - Dead code
  - Kod testów / kod demo aplikacji
  - Nierealistyczne warunki

<1 na 20

zgłoszeń w curl w 2025 okazało się prawdziwą podatnością

Źródło: Daniel Stenberg, „The end of the curl bug bounty", 26.01.2026

Zgłoszone ≠ załatane

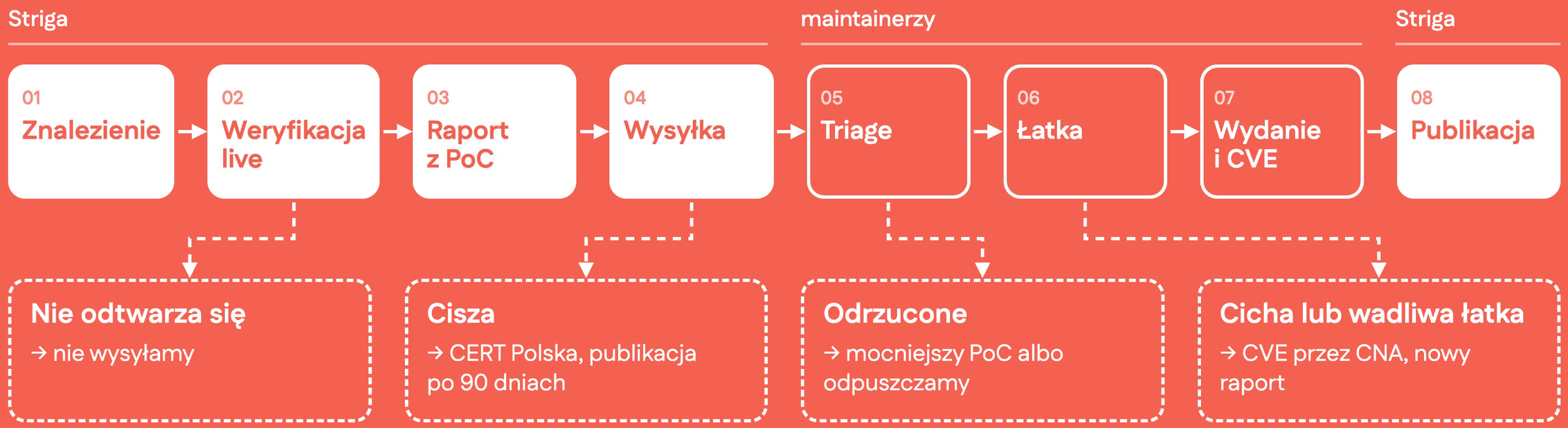


Źródło: Anthropic, Coordinated Vulnerability Disclosure dashboard, red.anthropic.com/2026/cvd

- Znalezienie podatności to nie wszystko
  - Trzeba ją załatać
  - … a w łatce nie może być kolejnej podatności
- Szczególnie problematyczne jest to w złożonych zbiorach kodu

← Cyberapokalipsa…

Proces zgłaszania podatności w open-source



Detektory podatności w kodzie źródłowym

	Detektor / metoda
SAST	Classic SAST
	AI-filtered SAST
	AI-native SAST
DAST	fuzzing z LLM

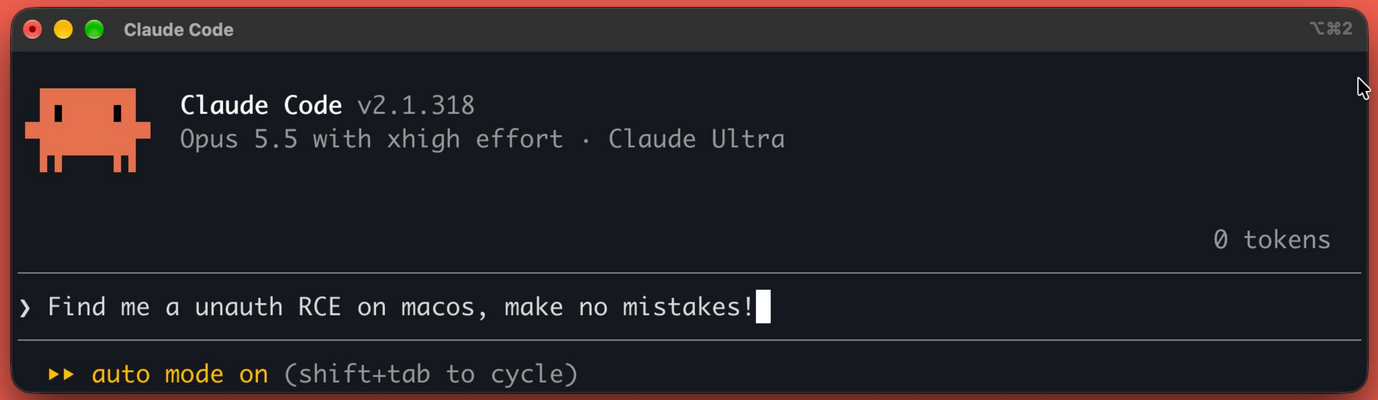
Detektory podatności w kodzie źródłowym

	Detektor / metoda	Precyzja
SAST	Classic SAST	niska
	AI-filtered SAST	średnia / wysoka
	AI-native SAST	średnia / wysoka?
DAST	fuzzing z LLM	wysoka

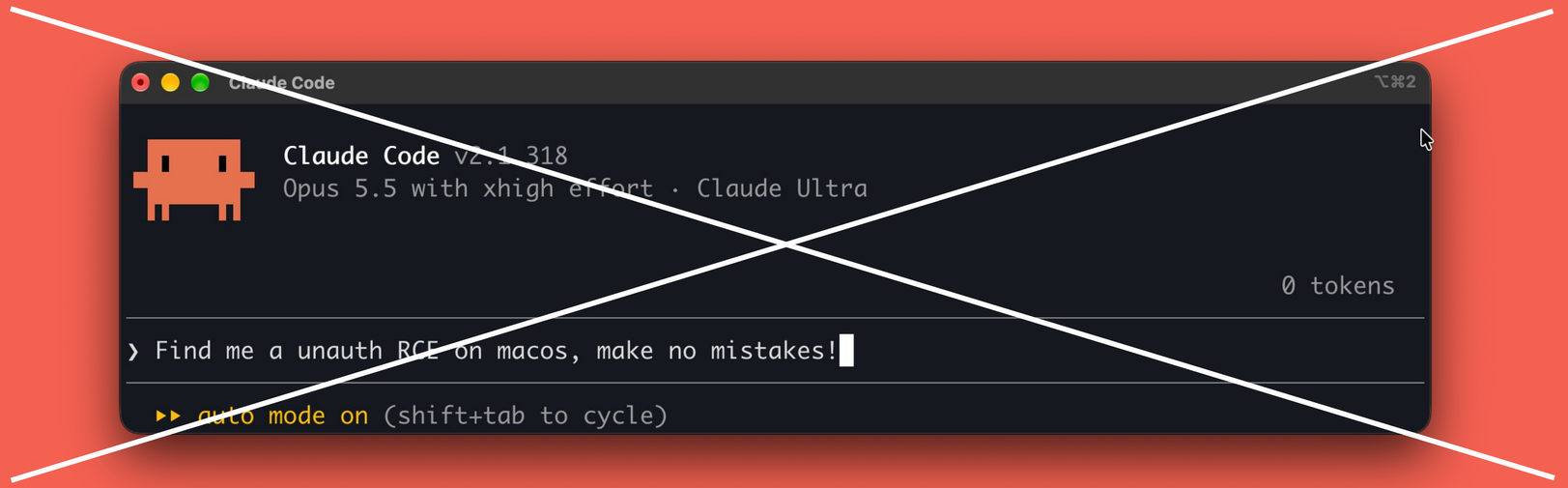
Detektory podatności w kodzie źródłowym

	Detektor / metoda	Precyzja	Recall
SAST	Classic SAST	niska	niski
	AI-filtered SAST	średnia / wysoka	niski
	AI-native SAST	średnia / wysoka?	wysoki?
DAST	fuzzing z LLM	wysoka	średni?

AI-native SAST



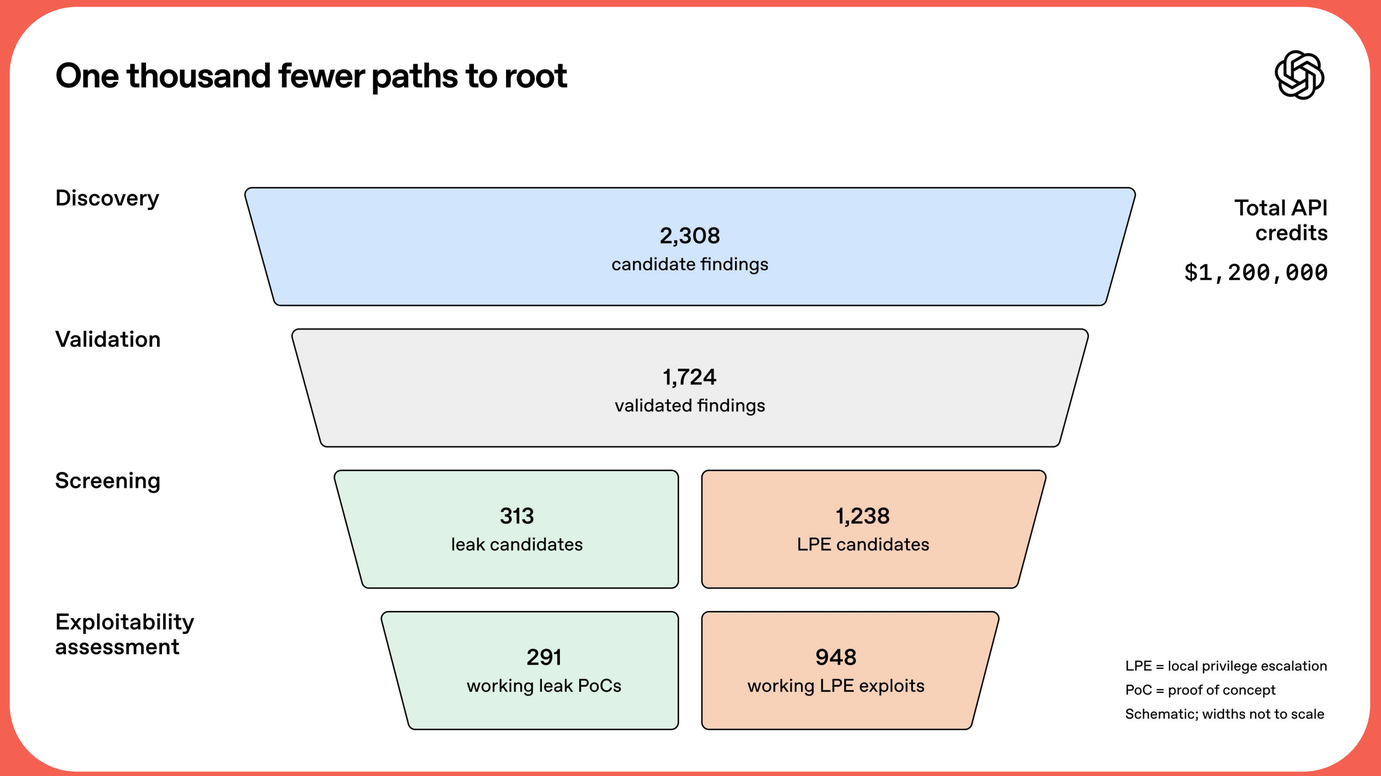
AI-native SAST



Generyczny schemat działania harnessu wykrywającego podatności w kodzie

Rozpoznanie → Detekcja → Deduplikacja → Ewaluacja → Raportowanie

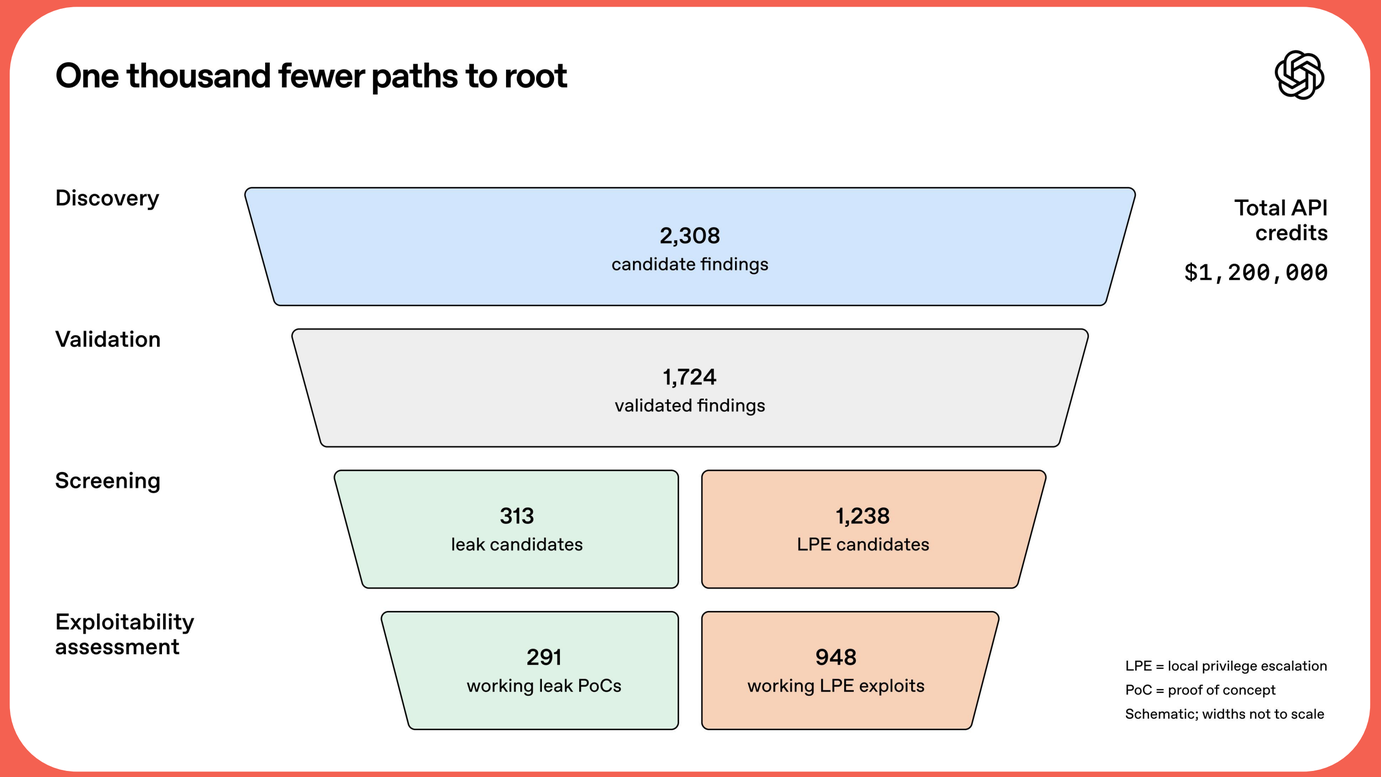
Nawet duże laby odsiewają połowę



- Audyt bezpieczeństwa kernela Linuxa
- 2 308 kandydatów
- 1 239 działających PoC
- $1,2 mln

Źródło: https://openai.com/index/a-thousand-fewer-paths-to-root/

Nawet duże laby odsiewają połowę



- Audyt bezpieczeństwa kernela Linuxa
- 2 308 kandydatów
- 1 239 działających PoC
- $1,2 mln

Detekcja bazująca na LLM działa, ale jest jeszcze nieekonomiczna.

Źródło: https://openai.com/index/a-thousand-fewer-paths-to-root/

Case study

- Które klasy detektorów wykryją 10 podatności znalezionych przez Strigę?
  - axios, Tomcat, Ollama, n8n, httpd, ...
- Każda podatność w wersji podatnej i załatanej
- Klasy detektorów:
  - Classic SAST: Semgrep CE, Opengrep, CodeQL, gosec, cppcheck
  - AI-filtered SAST: LLM ocenia alerty SAST, RAPTOR
  - AI-native SAST: claude-code-security-review, deepsec, Metis, open-kritt, OpenHack
- Modele:
  - DeepSeek V4.1 Flash, GLM 5.3, GLM 5.3 Flash, Opus 5.5

Case study

- Warunki wykrycia:
    1. Alert w podatnej funkcji
    2. Klasa pasuje do klasy bugu
    3. Alert znika po łatce
- Ograniczenia:
    - W case study harnessy mają podany gotowy zakres
    - W prawdziwym skanie to harness musi go znaleźć
    - Model DeepSeek V4.1 Flash jest mały...

Macierz wyników

CVE	Classic SAST 5 narzędzi
axios (DoS)	○
Tomcat	○
Ollama	◐
n8n	○
httpd	○
Koszt	$0

Classic SAST: Semgrep

```
 packages/nodes-base/nodes/Currents/CurrentsTriggerHelpers.ts
❯❯❯ ajinabraham.njsscan.generic.hardcoded_secrets.node_secret
       ❮❮ Blocking ❯❯
       A hardcoded secret is identified. Store it properly in an environment
       variable.
       Details: https://sg.run/Rxpe

        11┆ const WEBHOOK_SECRET_HEADER = 'x-webhook-secret';

 packages/nodes-base/utils/workflowInputsResourceMapping/constants.ts
❯❯❯ ajinabraham.njsscan.generic.hardcoded_secrets.node_password
       ❮❮ Blocking ❯❯
       A hardcoded password in plain text is identified. Store it properly in an
       environment variable.
       Details: https://sg.run/JolW

         7┆ export const PASSTHROUGH = 'passthrough';

 packages/workflow/src/constants.ts
❯❯ ajinabraham.njsscan.generic.hardcoded_secrets.node_username
       ❮❮ Blocking ❯❯
       A hardcoded username in plain text is identified. Store it properly in an
       environment variable.
       Details: https://sg.run/5z6X

       134┆ export const CHAT_WAIT_USER_REPLY = 'waitUserReply';
```

Sekret, hasło, login. Żadne z nich nim nie jest.

Classic SAST: gosec

```
[x/transfer/download.go:297] - G304 (CWE-22): Potential file inclusion via
  variable (Confidence: HIGH, Severity: MEDIUM)
    296:         // Resume — re-hash existing partial data, then append
  > 297:         f, err = os.OpenFile(tmp, os.O_RDWR, 0o644)
    298:         if err != nil {

Autofix: Consider using os.Root to scope file access under a fixed root (Go
  >=1.24). Prefer root.Open/root.Stat over os.Open/os.Stat to prevent directory
  traversal.
```

Dobre miejsce, zły powód. Alert zostaje po łatce.

Macierz wyników

	Classic SAST	AI-filtered SAST	
CVE	5 narzędzi	Triage LLM	RAPTOR
axios (DoS)	○	○	○
Tomcat	○	○	◐
Ollama	◐	◐	○
n8n	○	○	○
httpd	○	○	○
Koszt	$0	$0.26	$1.22

● wykryte ◐ blisko, ale nie ten błąd ○ nic

AI-reviewer (claude-code-security-review), narzędzia AI na DeepSeek V4.1 Flash

striga.ai

The Hack Summit 2026

AI-filtered SAST: Triage LLM

```
reasoning:
  Is blob.Digest attacker-controlled? It's the digest of the content, typically
    validated by the registry.
  […]
  If blob.Digest is not validated, a malicious registry could return a digest
    like "../../etc/foo". But digests are typically parsed and validated by go-
    containerregistry or similar.
  […]
  But we don't see digestToPath.
  […]
  Hmm, but modest risk. I'll answer FP.

response:
  {
    "verdict": "FP",
    "cwe": "CWE-22",
    "reason": "[…] the digest is a validated OCI content digest restricted to
      algorithm/hex characters […]"
  }
```

Rozważył błąd i go odrzucił, bo nie widział funkcji.

Macierz wyników

	Classic SAST	AI-filtered SAST		AI-native SAST
CVE	5 narzędzi	Triage LLM	RAPTOR	AI-reviewer
axios (DoS)	○	○	○	○
Tomcat	○	○	◐	●
Ollama	◐	◐	○	●
n8n	○	○	○	○
httpd	○	○	○	○
Koszt	$0	$0.26	$1.22	$1.55

● wykryte ◐ blisko, ale nie ten błąd ○ nic

AI-reviewer (claude-code-security-review), narzędzia AI na DeepSeek V4.1 Flash

striga.ai

The Hack Summit 2026

AI-native SAST: AI-reviewer

```
{
  "file": "x/transfer/download.go",
  "line": 286,
  "category": "path_traversal",
  "description": "Blob digests originating from registry manifests are used to
    build filesystem paths via digestToPath() without ever calling
    manifest.ValidateDigest/BlobsPath. […] A digest such as
    \"../../../../home/user/x\" (no sha256: prefix even required) escapes the
    blob directory. […]",
  "recommendation": "Validate every Blob.Digest with manifest.ValidateDigest (or
    use manifest.BlobsPath) before it is used […]",
  […]
}
```

Przeczytał funkcję i zaproponował tę samą łatkę co maintainerzy.

Macierz wyników

	Classic SAST	AI-filtered SAST		AI-native SAST				
CVE	5 narzędzi	Triage LLM	RAPTOR	AI-reviewer	deepsec	Metis	open-kritt	OpenHack
axios (DoS)	○	○	○	○	○	○	○	○
Tomcat	○	○	◐	●	●	●	○	○
Ollama	◐	◐	○	●	●	○	○	○
n8n	○	○	○	○	◐	○	○	○
httpd	○	○	○	○	○	○	○	○
Koszt	$0	$0.26	$1.22	$1.55	$7.74	$3.72	$5.99	$6.21

● wykryte ◐ blisko, ale nie ten błąd ○ nic

AI-reviewer (claude-code-security-review), narzędzia AI na DeepSeek V4.1 Flash

Szum detektorów

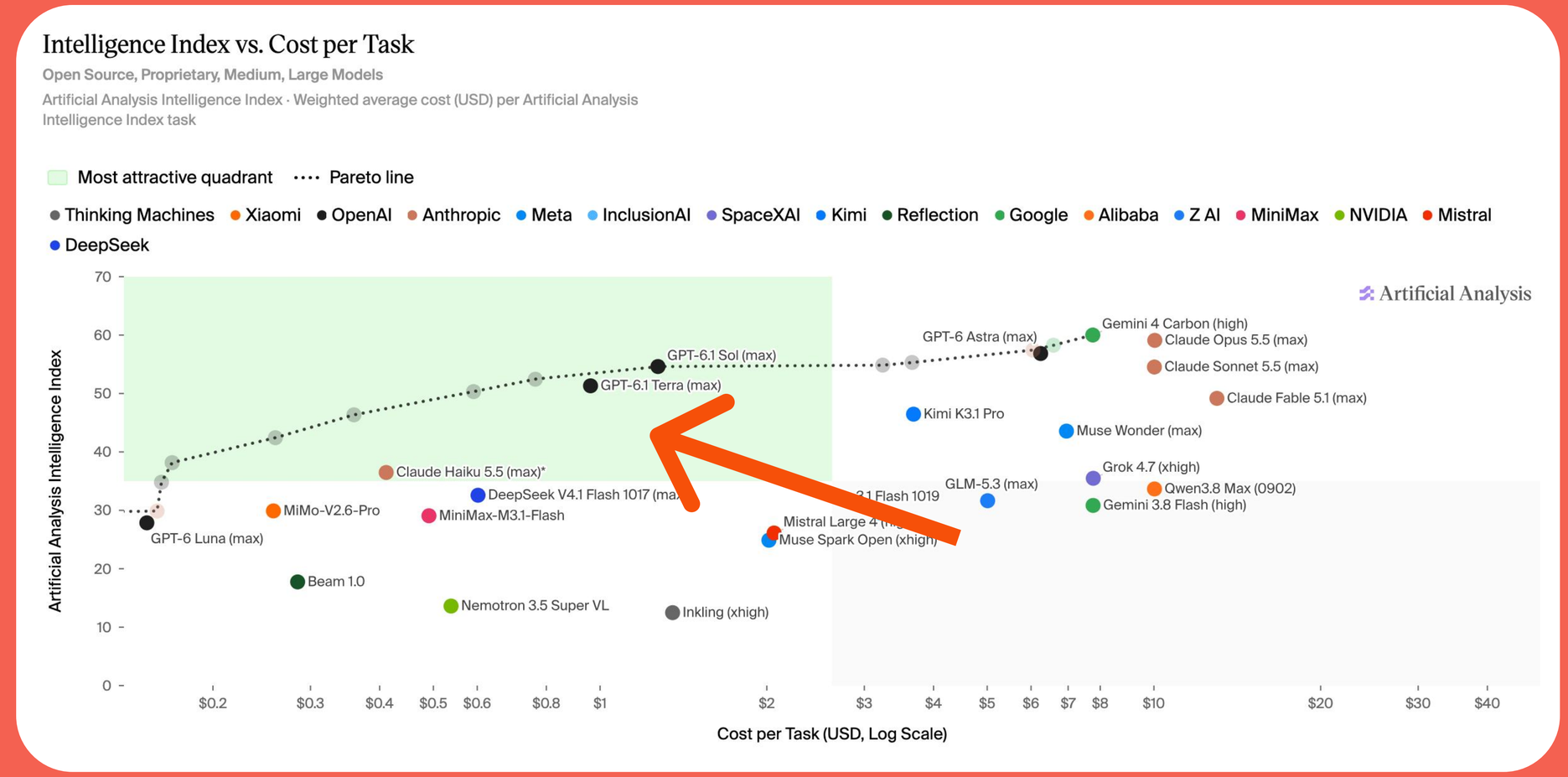
	Narzędzie	Zgłoszenia do przejrzenia	Wykryte CVE
Classic SAST	5 narzędzi	394	—
AI-filtered SAST	Triage LLM	17	—
	RAPTOR	7	—
AI-native SAST	deepsec	181	Tomcat, Ollama
	Metis	84	Tomcat
	AI-reviewer	8	Tomcat, Ollama

5 CVE z tytułu, wersje podatne, modele na DeepSeek V4.1 Flash. Classic SAST i triage LLM liczone w plikach podatnego komponentu, pozostałe narzędzia na szerszym zakresie modułu. Zgłoszenia po wbudowanej walidacji narzędzia.

Jak poprawnie używać detektorów nowej generacji?

- Wybierz dobre modele!
  - Koszt, jakość, szum
- Dobieraj model zależnie od krytyczności modułu i zadania
- Wykluczaj kategorie podatności w świadomy sposób (domyślnie DoS: 4 z 10 bugów niezgłoszonych)
- Sandbox bez sieci i bez wyjścia z repo
- Harness powinien mieć pipeline ewaluacji, nie każdy kandydat okazują się podatnością
- Bierz pod uwagę osiągalność podatności, scenariusz ataku, wpływ
- Zarządzanie podatnością nie kończy się na detekcji

Jaki model wybrać?



Źródło: https://artificialanalysis.ai

Powrót do jednej linii

```
+      super.messageReceived(msg);
```

Wykryte

AI-reviewer (DeepSeek V4.1 Flash, Opus 5.5), deepsec, Metis

Przeoczone

Semgrep CE, Opengrep, CodeQL, OSS-Fuzz, GLM 5.3, GLM 5.3 Flash, RAPTOR, open-kritt, OpenHack

Regresję w przepływie wyjątków widzi model, który czyta kod

Nie każdy model i nie z każdym harnessem

Wnioski

- Dostęp do zdolności cyber jest demokratyzowany
- Osoby odpowiedzialne za utrzymywanie projektów open-source są przytłoczone ilością zgłoszeń

- Mikro: na jednym module o wykryciu decyduje model
- Makro: na całej aplikacji harness decyduje, czy model w ogóle zobaczy podatny kod
  - Optymalizacja harnessu pod: recall / precyzję
- Żadne narzędzie open-source nie odkryło nawet większości podatności znalezionych przez Strigę
  - Klasyczny SAST 0/10, open-source + AI 3/10



Cyberapokalipsa nie nadciąga, ona już tu jest.



bartekdmitruk.com

Dziękuję

Zachęcam do zadawania pytań oraz oceny prelekcji!

Ask a Speaker : 10:25–10:40

striga.ai

The Hack Summit 2026