Open source is a virus

Prabhu Subramanian
prabhu@appthreat.dev
GitHub: @prabhu

AboutCode

AppThreat

About me

https://aboutcode.org

https://github.com/AppThreat/atom
https://github.com/AppThreat/chen

https://github.com/owasp-dep-scan/blint
https://github.com/owasp-dep-scan/dosai

Sponsors:

nlnet FOUNDATION

LEVO

Open source is a virus. Example: ffmpeg

```
prabhu@mpro /Applications [1]
❯ find . -name "*ffmpeg*"
./LM Studio.app/Contents/Frameworks/Electron Framework.framework/Versions/A/Libraries/libffmpeg.dylib
./VideoProc Converter AI.app/Contents/Frameworks/ffmpeg
./Rancher Desktop.app/Contents/Frameworks/Electron Framework.framework/Versions/A/Libraries/libffmpeg.dylib
./DaVinci Resolve/DaVinci Resolve.app/Contents/Applications/Electron.app/Contents/Frameworks/Electron Framework.framework/Versions/A/Libraries/libffmpeg.dylib
./Bitwarden.app/Contents/Frameworks/Electron Framework.framework/Versions/A/Libraries/libffmpeg.dylib
./Slack.app/Contents/Frameworks/Electron Framework.framework/Versions/A/Libraries/libffmpeg.dylib
./Canva.app/Contents/Frameworks/Electron Framework.framework/Versions/A/Libraries/libffmpeg.dylib
```

- Remember the 3cx 2023 supply-chain attack?
- Should we trust ffmpeg? Which one?

```
diff "./Canva.app/Contents/Frameworks/Electron Framework.framework/Versions/A/Libraries/libffmpeg.dylib"
"./Slack.app/Contents/Frameworks/Electron Framework.framework/Versions/A/Libraries/libffmpeg.dylib"
Binary files ./Canva.app/Contents/Frameworks/Electron Framework.framework/Versions/A/Libraries/libffmpeg.dylib and
./Slack.app/Contents/Frameworks/Electron Framework.framework/Versions/A/Libraries/libffmpeg.dylib differ
```

- Capabilities guess:
  - audio, video, network, gpu, crypto, AI

Identifying malware in oss at scale

- Run strace at scale - OSSF Malicious Packages (https://github.com/ossf/malicious-packages)
- Run lots of YARA rules and pray (Everyone?)
- Compare a bunch of hashes (AV vendors?)
- LLMs (!)

Can we do better? Yes, with abcd!

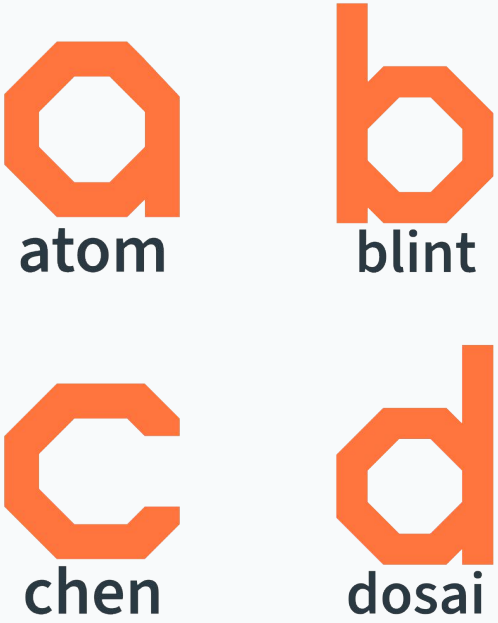
Terminologies

- Static analysis & Program slicing
  - AST, CFG, DDG, PDG
- Software Bill-of-Materials (SBOM)
  - CBOM, OBOM, ML-BOM
- Compilation vs Disassembly vs Decompilation
- Pros & Cons of slicing source and binary
- tree-sitter, capstone/nyxstone, LIEF, Roslyn SDK

TLP:AMBER

- **NO GRANT or PAPER submission**
- Usage in both open-source and commercial applications is allowed
- Recording is allowed

Introducing **pre-computed** slices for ecosystems



atom (MIT) - Intermediate representation (IR) and static slicing tool for source
blint (MIT) - Binary analysis and disassembler with heuristics
chen (Apache-2.0) - Code Hierarchy Exploration Network library and REPL
Dosai (Apache-2.0) - Dotnet Source and Assembly Inspector

Open source tools, specifications, and dataset!

pre-computation workflow

1. **High-level Representation:**
   - **atom (source):** compute usages, data-flows, and reachable flows for open-source projects from source.
   - **dosai (source + binary):** compute namespaces, call graphs, and dependencies for dotnet projects.
   - **semantic analysis:** Identify ***capabilities***, sources, sinks, and, attack surface.
2. **Low-level Representation (binary):**
   - **blint:** compute low-level symbols, ***capabilities***, mnemonics categories, and, registries information.
3. **Correlate, store, and publish in purldb**
4. **Analyze with YARA or chennai:**

   chennai: CHEN Not AI is an advanced REPL for atom and slices.

Demo - Analyze ffmpeg binaries with **abcd** tools

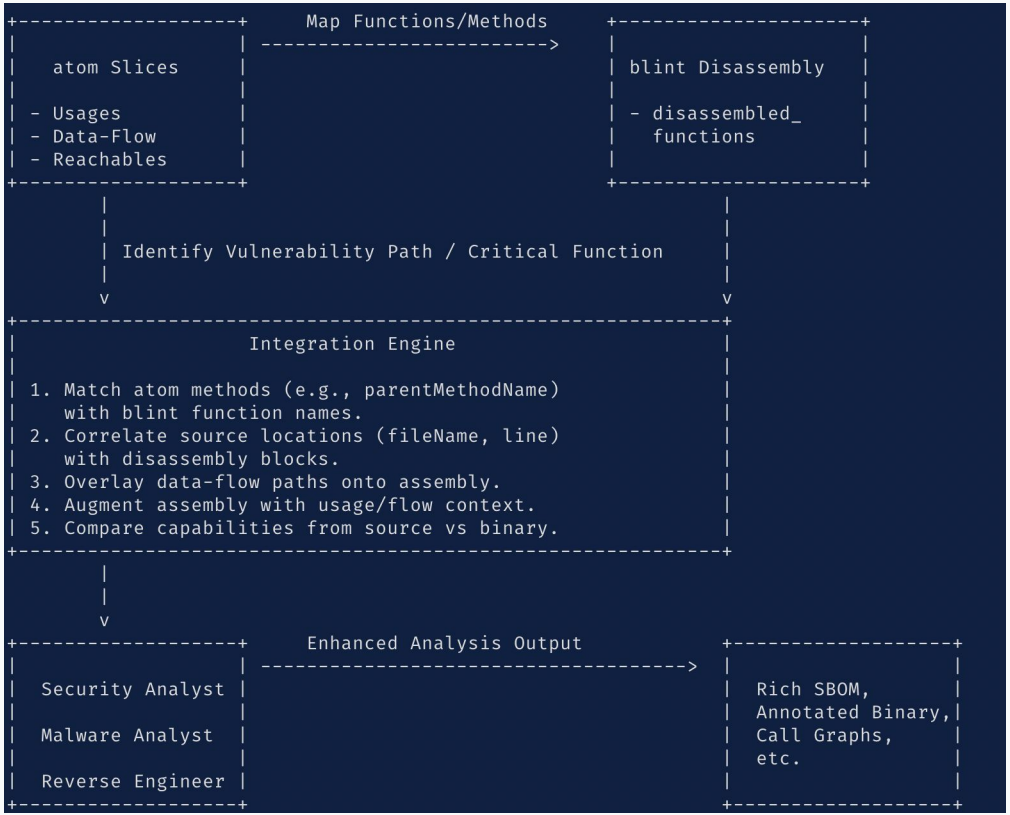
Repo: https://github.com/prabhu/hacklu

Discussion ideas:

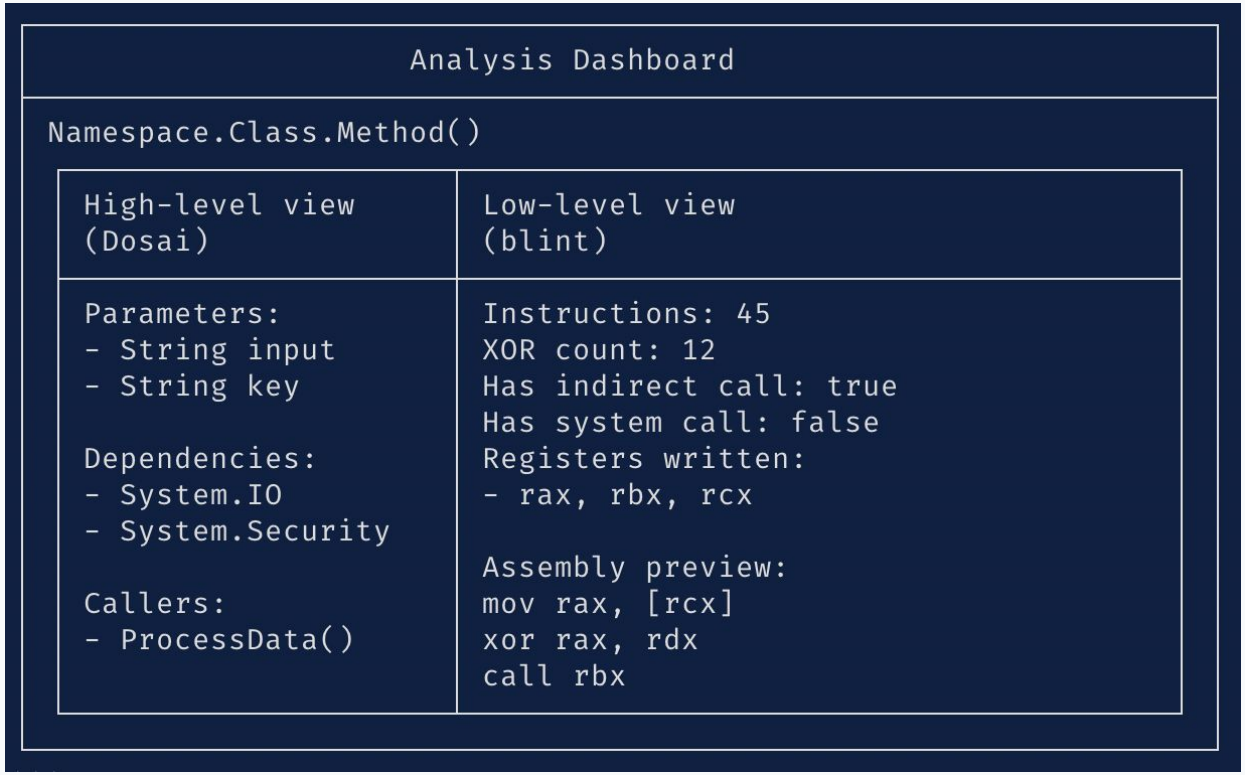
- Capability Discrepancy Detection
- Supply Chain Security - Detecting **Hidden Dependencies (*)** or Modifications
- Vulnerability Verification - Buffer Overflow
- Reverse Engineering - Understanding Third-Party Library Integration
- Malware Analysis - Identifying C2 Communication

* Disclosure currently under review (Not ffmpeg-related)

Correlation workflow - c/c++ (apk*) (atom + blint)



Correlation workflow - .Net (dosai + blint)



Use cases (WIP)

Capability Discrepancy Detection

- **Source:** Generate usages and reachables slices with atom. Generate SBOM with cdxgen (`cdxgen -t c –deep`)
- Identify source capabilities using YARA rules.
  - `Example: network, fileIO`
- **Binary:** Build and disassemble the target binaries with blint (`blint -i <directory> -o <reports dir> –disassemble`)
- Identify binary capabilities based on symbol names and registers information using YARA rules.
  - `Example: http, gpu, SIMD, crypto`

Maybe LLMs could help here in the future?

Use cases (WIP)

Supply Chain Security - Detecting **Hidden Dependencies** or Modifications

- **Source:** Generate usages and reachables slices with atom. Generate SBOM with cdxgen (`cdxgen -t c –deep`)
- **Binary:** Build and disassemble the target binaries with blint (`blint -i <directory> -o <reports dir> –disassemble`)
- Identify symbols and dependencies not present in the source SBOM (*)
- Identify problematic dependencies in the binary using YARA rules.

(*) - Contributions are welcome

CREDITS: AboutCode purldb + BANG

Inspired by a project proposal by Armijn

Binary Analysis Next Generation (BANG)
https://github.com/armijnhemel/binaryanalysis-ng

purldb
https://github.com/aboutcode-org/purldb

AboutCode

Things to ask me

- Do you have any **free** AppThreat t-shirts?
- How is life as a full-time open-source developer?
- How can we sponsor and become part of your community?
- Do you offer in-depth workshops and bespoke training?