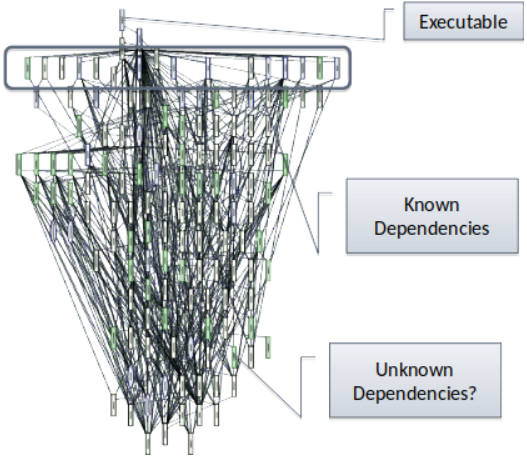
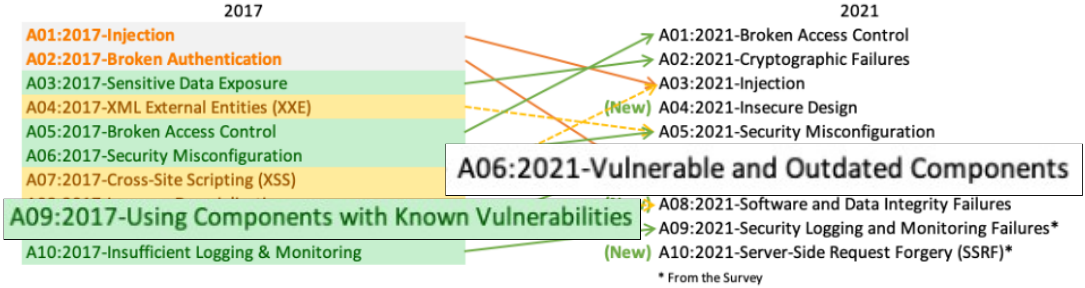
Binary Software Composition Analysis with CodeSentry

Antonio Flores Montoya, Drew DeHaas, Paul Anderson and Vineeth Kashyap
GrammaTech, Inc.
May 17th, 2022

Motivation: Reality of Modern Software Development

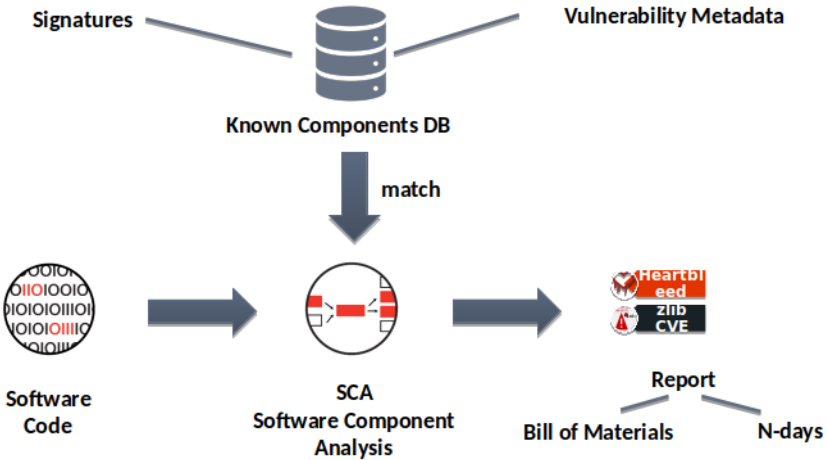


Motivation: Top 10 Security Risks



Source: https://github.com/OWASP/Top10

Solution: Binary Software Composition Analysis (BSCA)



Challenges To BSCA

- Same source code → **Very different** binaries
  - Due to compiler and compiler optimizations
- Check against **hundreds of thousands** of known third-party components
- Need to identify components **and their versions**
- Detect **partial** library inclusions

CodeSentry's Approach

- Multiple LibIds (**Lib**rary **Id**entification) components
  - Each Libid reports library matches and their confidence level
  - Results are combined for final report
  - Highly parallel: LibIds run in parallel, target binaries analyzed in parallel
- **StrLibId**: Use strings as signatures
- **EmbedLibId**: Use procedure embeddings as signatures
- Steps:

  Populate known components signature DB
  Analyze target binary by extracting signatures and querying against DB

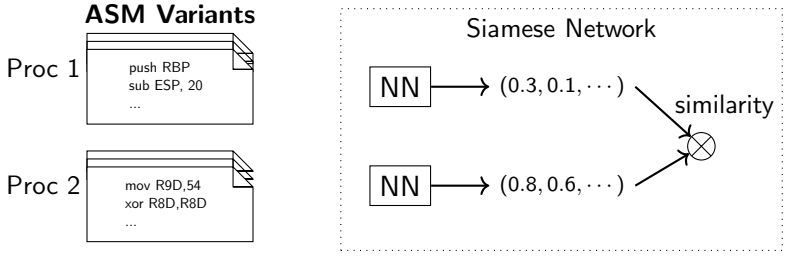
CodeSentry's Approach

- Multiple LibIds (**Lib**rary **Id**entification) components
  - Each Libid reports library matches and their confidence level
  - Results are combined for final report
  - Highly parallel: LibIds run in parallel, target binaries analyzed in parallel
- **StrLibId**: Use strings as signatures
- **EmbedLibId**: Use procedure embeddings as signatures
- Steps:

Train	neural network to produce embeddings (**EmbedLibId only**)
Populate	known components signature DB
Analyze	target binary by extracting signatures and querying against DB

Train (EmbedLibId)

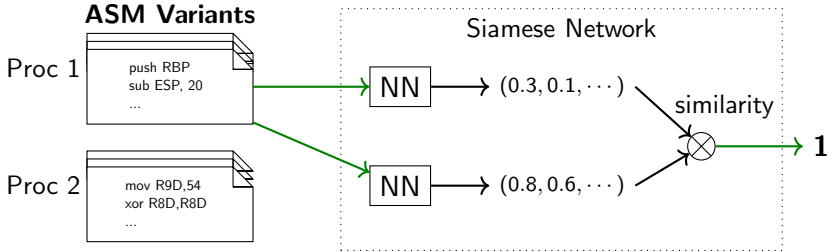
Train siamese neural network (NN) to produce:



Train (EmbedLibId)

Train siamese neural network (NN) to produce:

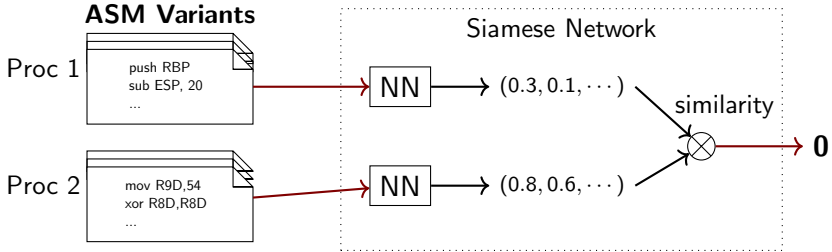
- Similar embeddings for **variants of the same** procedure



Train (EmbedLibId)

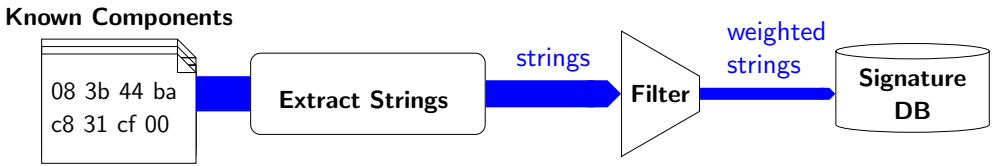
Train siamese neural network (NN) to produce:

- Similar embeddings for **variants of the same** procedure
- Different embeddings for different procedures



Populate

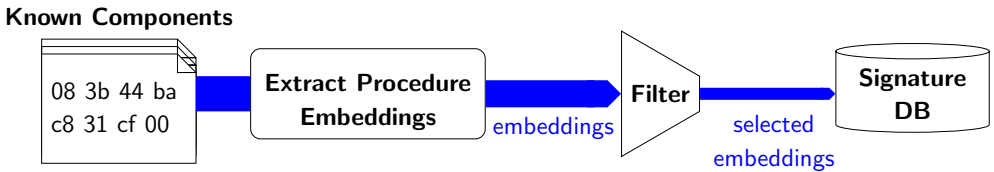
StrLibId



Filter:
TF/IDF: Term Frequency/ Inverse Document Frequency

Populate

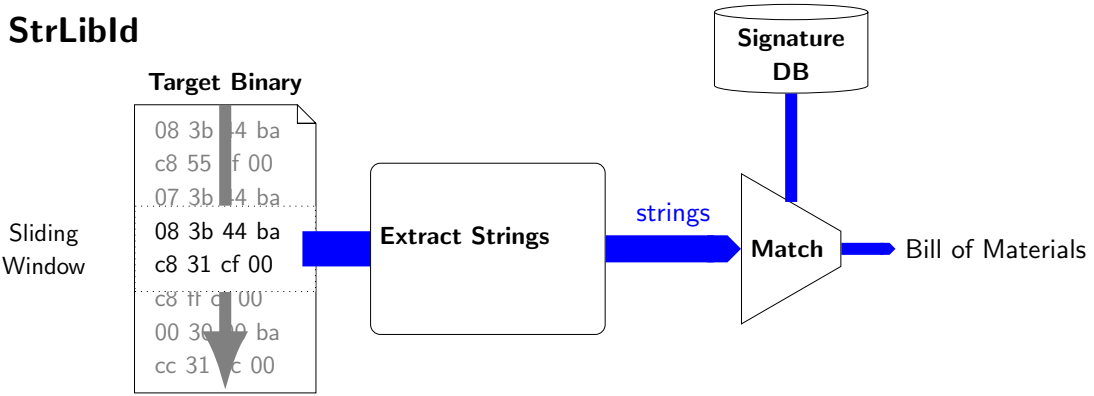
EmbedLibId



Filter:

- Prefer **unique** procedures (dissimilar to procedures in other projects)
- Prefer **stable** procedures (appear in most variants of a project)

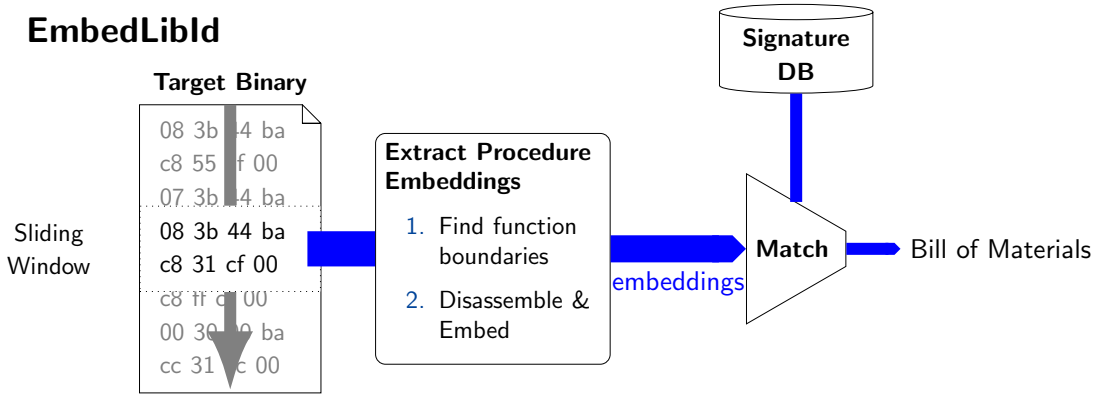
Analyze



Sliding window

- Leverages library locality
- Better find small libraries in big binaries

Analyze



Function boundaries

- Precise methods can be expensive
- Approximate methods are good enough

Conclusion

- BSCA provides reliable **Bill of Materials** and associated **vulnerability** information
- Directly based on the code that gets executed (no intermediaries or chain-of-trust)
- Analysis should be **lightweight** and **robust** to binary software variability
  - Strings provide robust signal (low variability across variants)
  - Use ML to (efficiently) extract signal from binary procedures
  - Strings and procedures provide complementary information