KOREA UNIVERSITY

CSSA Center for Software Security and Assurance

CCSLAB Computer & Communication Security Laboratory

SSP LAB Software Security & Privacy Laboratory

CNEPS: A Precise Approach for Examining Dependencies among Third-Party C/C++ Open-Source Components

Yoonjong Na, Seunghoon Woo

Joomyeong Lee, Heejo Lee

Korea University

nooryyaa@korea.ac.kr

Apr. 19, 2024

Motivation

What is software dependency?

- Open-source software (OSS) reuse became popular in development

- *We will briefly call reused software as **component** in this presentation*

Motivation

What is software dependency?

- Open-source software (OSS) reuse became popular in development

- **Dependencies** refer to a relationship where a component **requires** another component

Motivation

What is software dependency?

- Open-source software (OSS) reuse became popular in development
- **Dependencies** refer to a relationship where a component **requires** another component
- Tracking components dependencies also became important because…

Motivation

What is software dependency?

- Open-source software (OSS) reuse became popular in development

- **Dependencies** refer to a relationship where a component **requires** another component

- Tracking components dependencies also became important because…

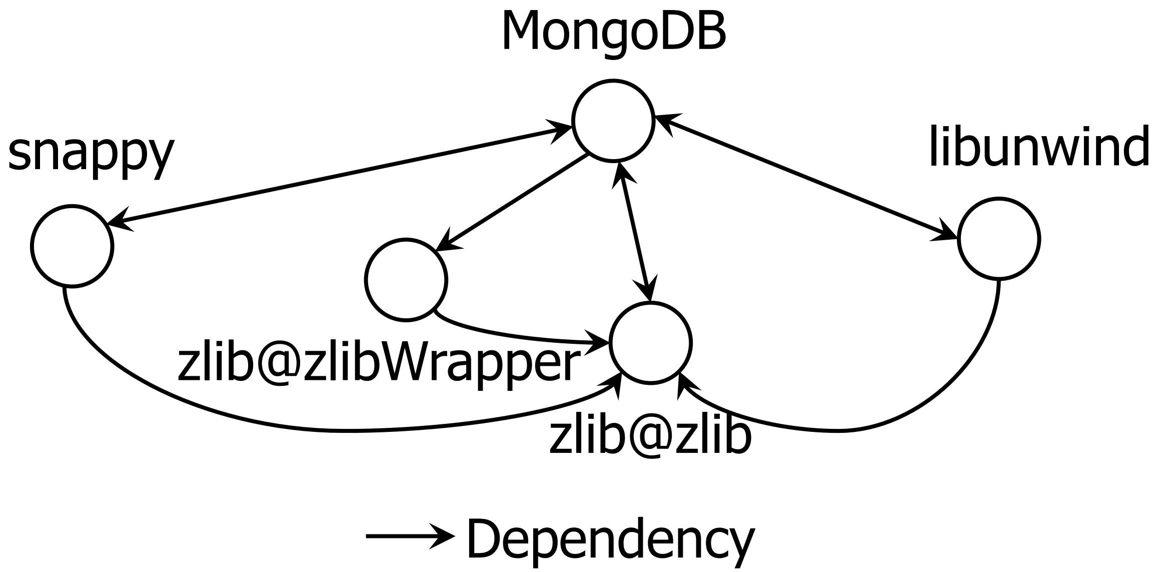
(1) Dependency can be used for **security threats management** by exploitability triage

(2) Precise dependency can be used to provide **supply chain transparency**

Motivation

What is software dependency?

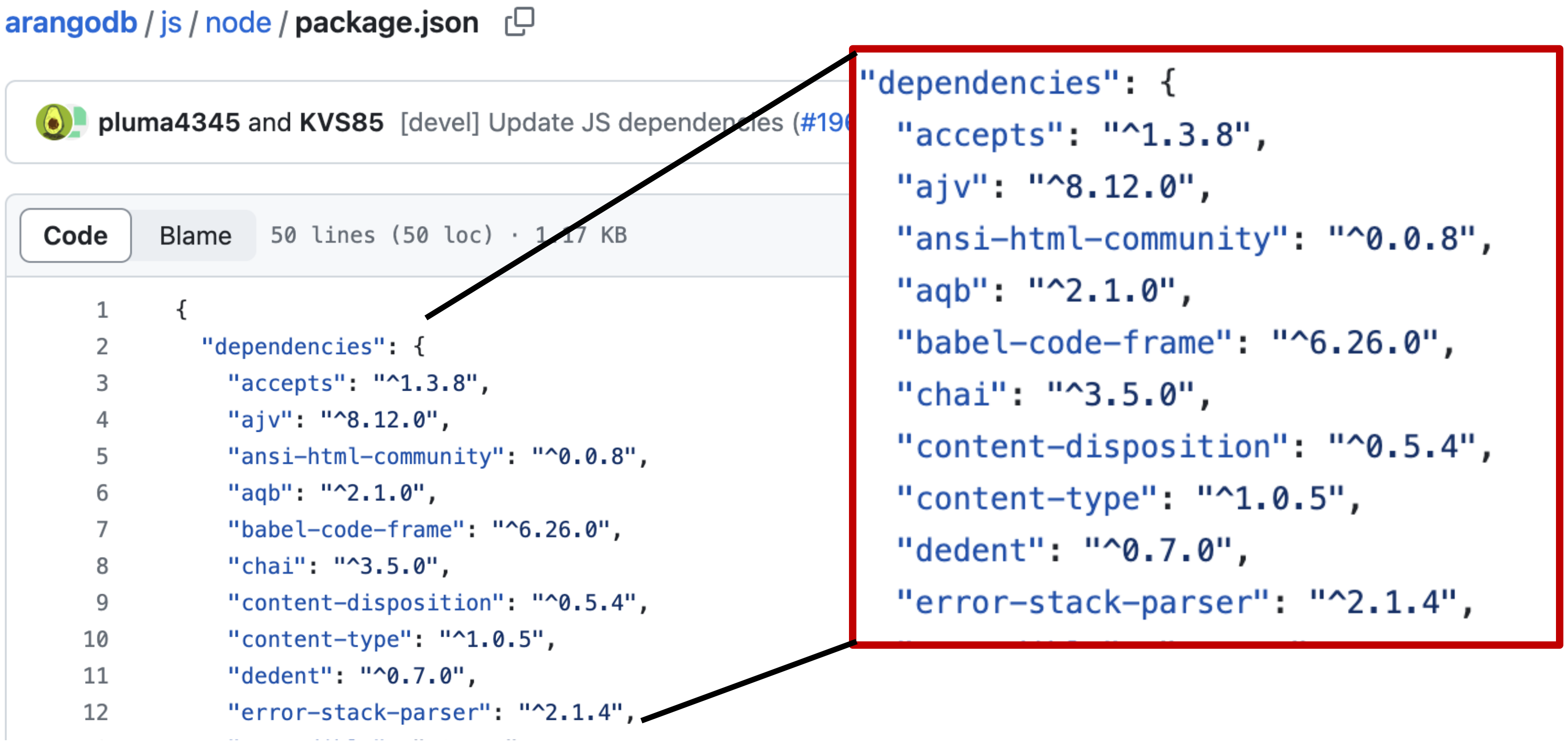
- Open-source software (OSS) reuse became popular in development
- **Dependencies** refer to a relationship where a software requires other **reused** software
- Tracking component dependencies also became important because of…
  - (1) Dependency can be used for **security threats management** by exploitability triage
  - (2) Precise dependency can be used to provide **supply chain transparency**



Motivation

Why it is difficult?

- Package manager provides useful **meta-data**



arangodb / js / node / package.json

pluma4345 and KVS85 [devel] Update JS dependencies (#19

50 lines (50 loc) · 1.17 KB

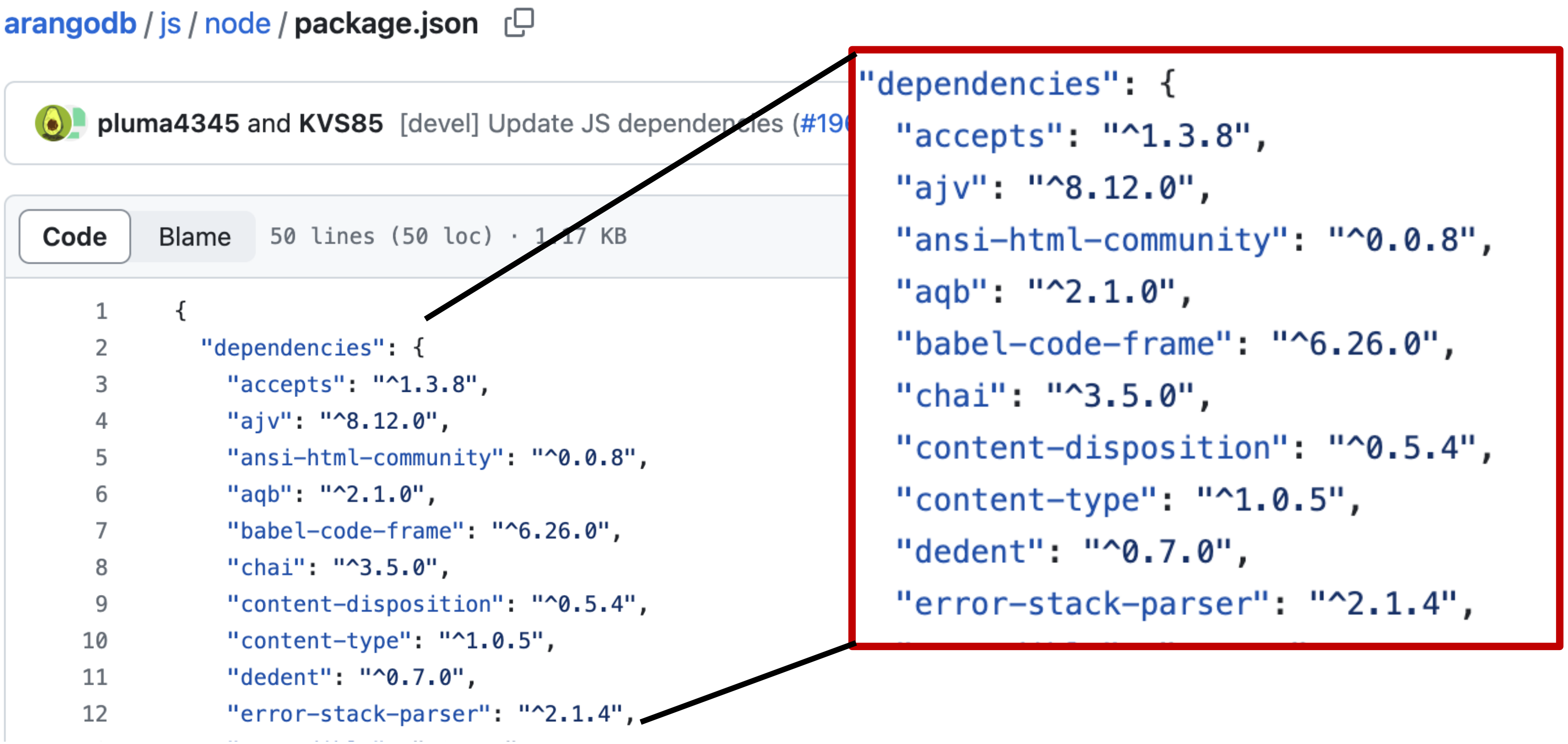
```
{
  "dependencies": {
    "accepts": "^1.3.8",
    "ajv": "^8.12.0",
    "ansi-html-community": "^0.0.8",
    "aqb": "^2.1.0",
    "babel-code-frame": "^6.26.0",
    "chai": "^3.5.0",
    "content-disposition": "^0.5.4",
    "content-type": "^1.0.5",
    "dedent": "^0.7.0",
    "error-stack-parser": "^2.1.4",
```

```
"dependencies": {
  "accepts": "^1.3.8",
  "ajv": "^8.12.0",
  "ansi-html-community": "^0.0.8",
  "aqb": "^2.1.0",
  "babel-code-frame": "^6.26.0",
  "chai": "^3.5.0",
  "content-disposition": "^0.5.4",
  "content-type": "^1.0.5",
  "dedent": "^0.7.0",
  "error-stack-parser": "^2.1.4",
```

Motivation

Why it is difficult?

- Package manager provides useful **meta-data**
- What if **meta-data** does not exist?
- Developers sometimes get components by **code-clone** (copy-paste)



arangodb / js / node / package.json

pluma4345 and KVS85 [devel] Update JS dependencies (#19

50 lines (50 loc) · 1.17 KB

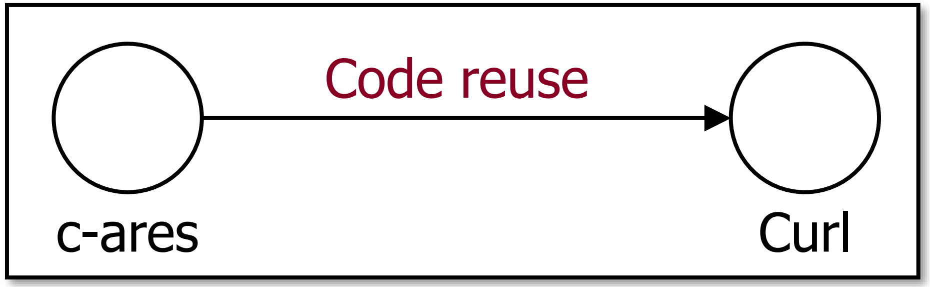
```
{
  "dependencies": {
    "accepts": "^1.3.8",
    "ajv": "^8.12.0",
    "ansi-html-community": "^0.0.8",
    "aqb": "^2.1.0",
    "babel-code-frame": "^6.26.0",
    "chai": "^3.5.0",
    "content-disposition": "^0.5.4",
    "content-type": "^1.0.5",
    "dedent": "^0.7.0",
    "error-stack-parser": "^2.1.4",
```

```
"dependencies": {
  "accepts": "^1.3.8",
  "ajv": "^8.12.0",
  "ansi-html-community": "^0.0.8",
  "aqb": "^2.1.0",
  "babel-code-frame": "^6.26.0",
  "chai": "^3.5.0",
  "content-disposition": "^0.5.4",
  "content-type": "^1.0.5",
  "dedent": "^0.7.0",
  "error-stack-parser": "^2.1.4",
```

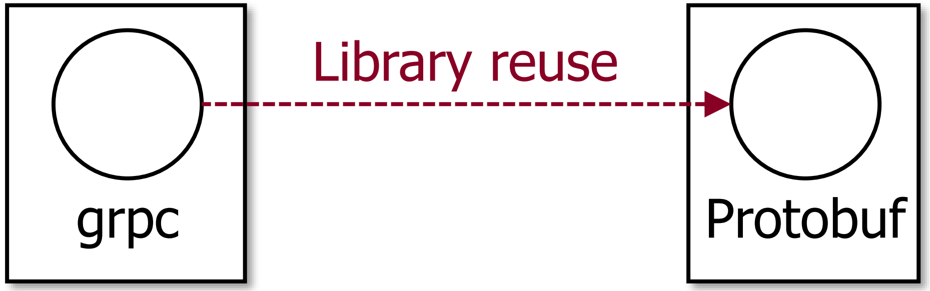
Motivation

- There are two ways to reuse components with the **code-clone method**

1) Developer can clone function directly into developers' file



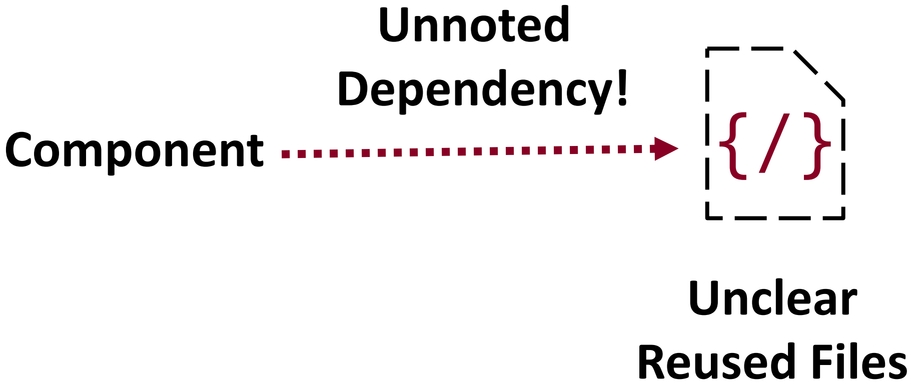
2) Developer can clone part of components and reuse them as a library

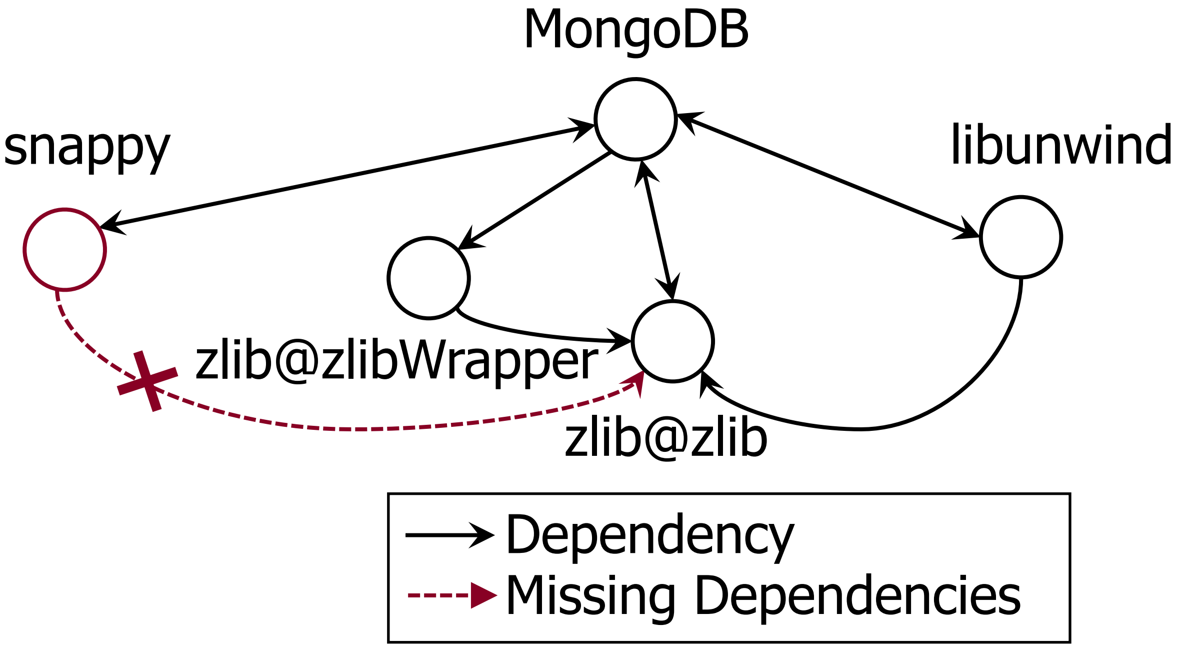


Motivation

Challenge 1: Indistinguishable files

- Files that are unclear whether reused or not
- Unidentified reused files may lead to unidentified dependencies
  - e.g., single-lined function, implementation of the cryptographic function

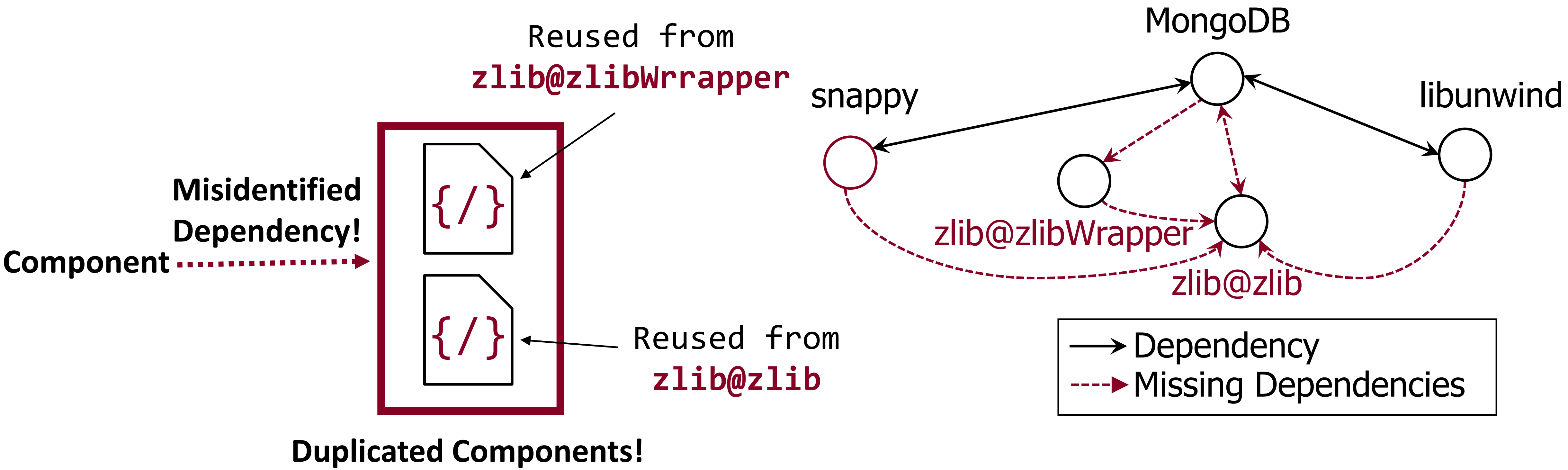




Motivation

Challenge 2: Duplicated component

- The same component can be cloned in the target software **multiple times**
- Incorrectly distinguished components may lead to misidentified dependencies



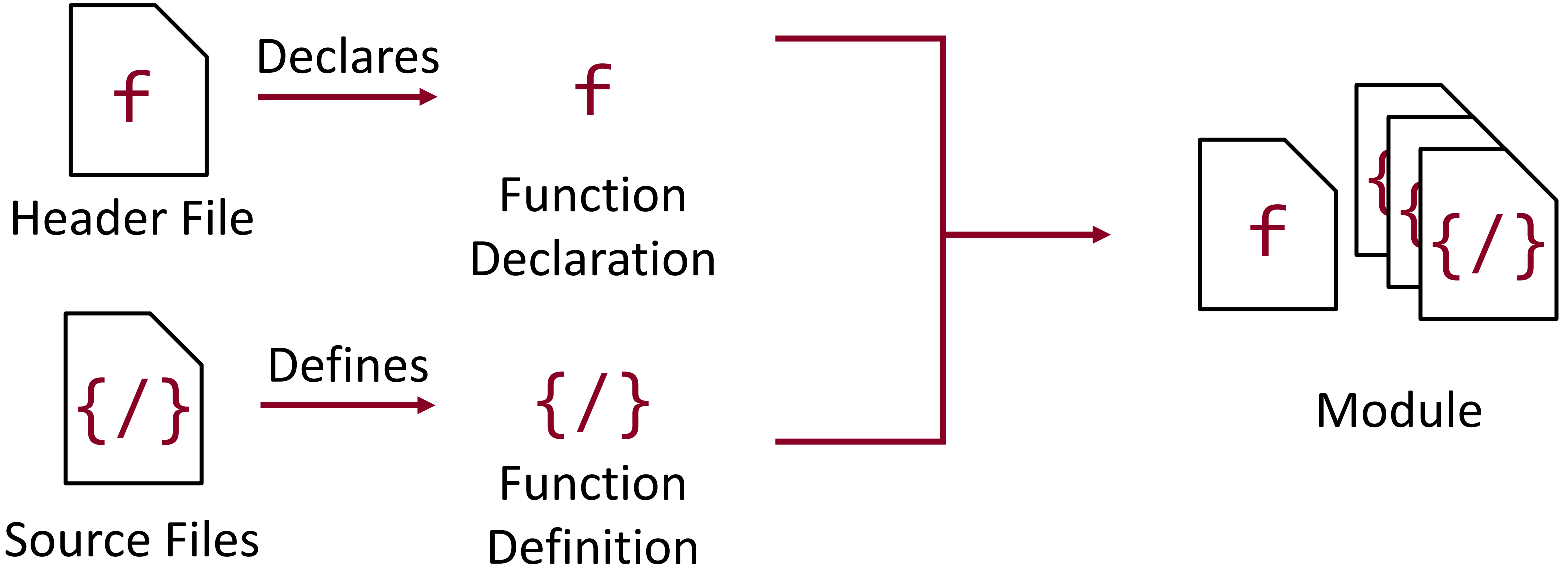
CNEPS: A Precise Approach for Examining Dependencies among Third-Party C/C++ Open-Source Components

CNEPS

CNEPS (<u>C</u>ompo<u>n</u>ent D<u>ep</u>endencies <u>S</u>canner)

- A novel approach to precisely identify dependencies between components

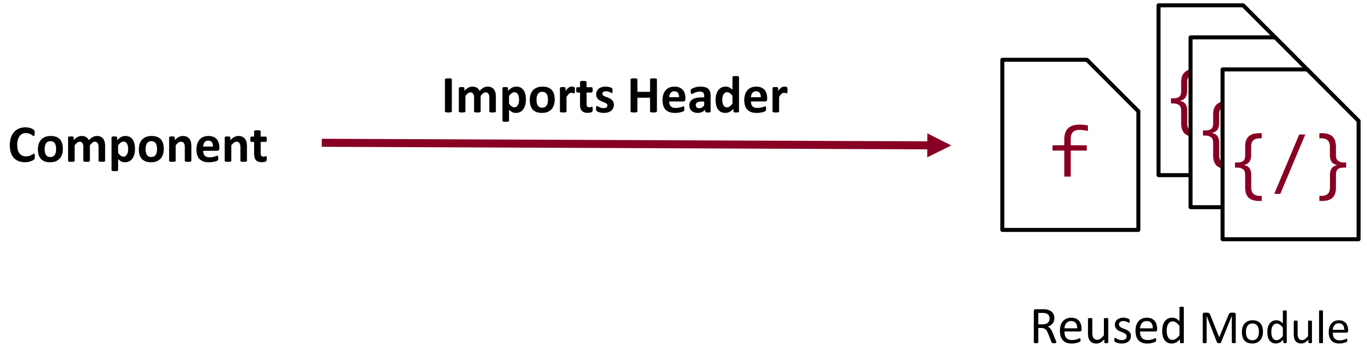
Background: Notations



Module Granularity

Key idea: Module granularity dependency analysis

- Module can also explained as a set of files that are **reusable as a library**
- To reuse a component as a library, these files need to be **cloned together**



Component

Imports Header

Reused Module

Overview of CNEPS

- CNEPS comprises three phases:

1. Module Construction

- Constructs modules for given input source code software

2. Dependency Analysis

- Analyzes dependencies using the module

3. Merging Components

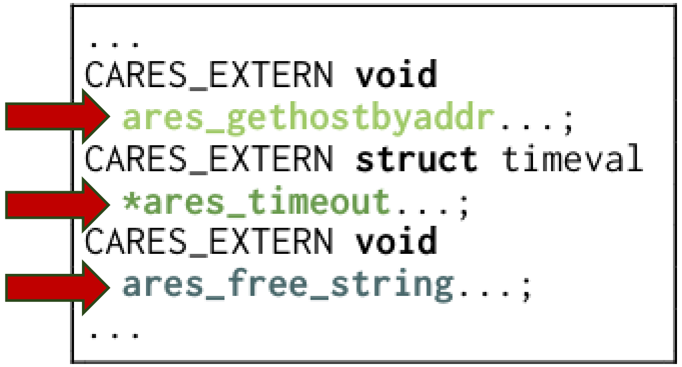
- Merges components that are non-duplicates

P1. Module Construction

Module Constructions

- Parses all **functions declarations** and **definitions** to reconstruct modules

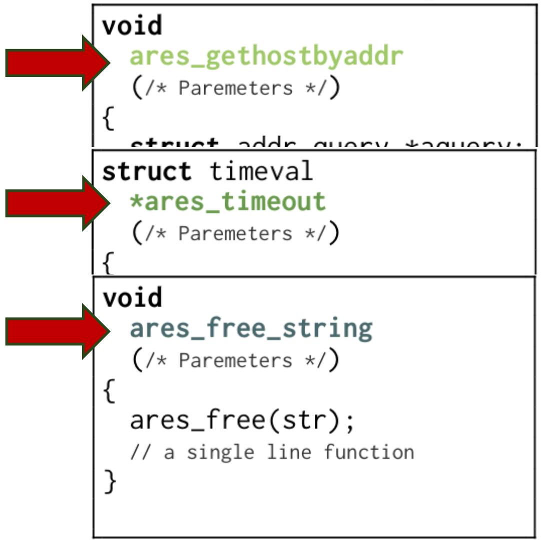
Declarations



```
...
CARES_EXTERN void
 ares_gethostbyaddr...;
CARES_EXTERN struct timeval
 *ares_timeout...;
CARES_EXTERN void
 ares_free_string...;
...
```

ares.h with declarations

Definitions



```
void
  ares_gethostbyaddr
  (/* Paremeters */)
{
  struct addr_query *aquery;
```

```
struct timeval
  *ares_timeout
  (/* Paremeters */)
{
```

```
void
  ares_free_string
  (/* Paremeters */)
{
  ares_free(str);
  // a single line function
}
```

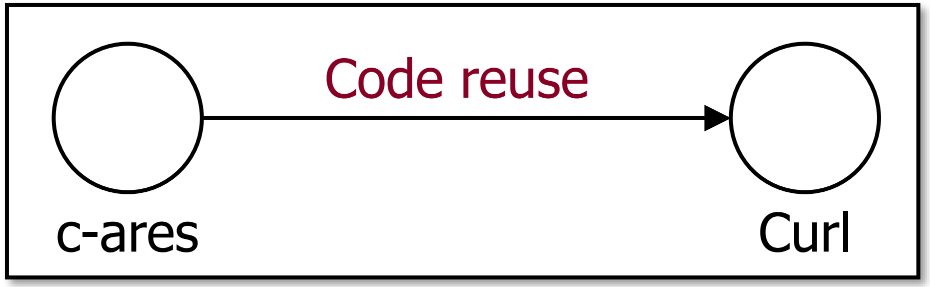
ares_gethostbyaddr.c,
ares_timeout.c
ares_free_string.c

P2. Dependency Analysis

- Analyze dependencies using modules

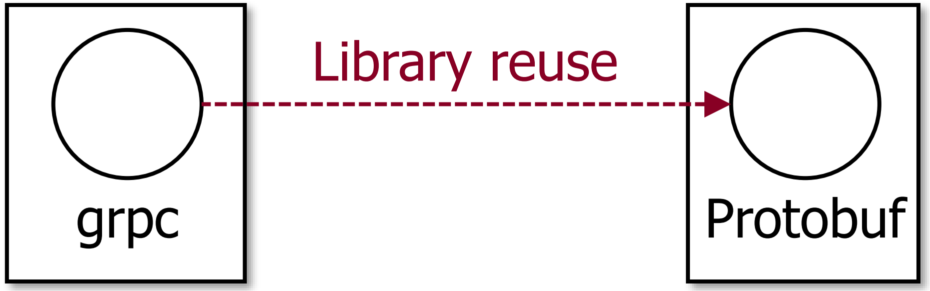
(1) Reused by function cloning (code reuse)

- Examine other components (function) included in the module



(2) Reused by cloning and reused as a library (library reuse)

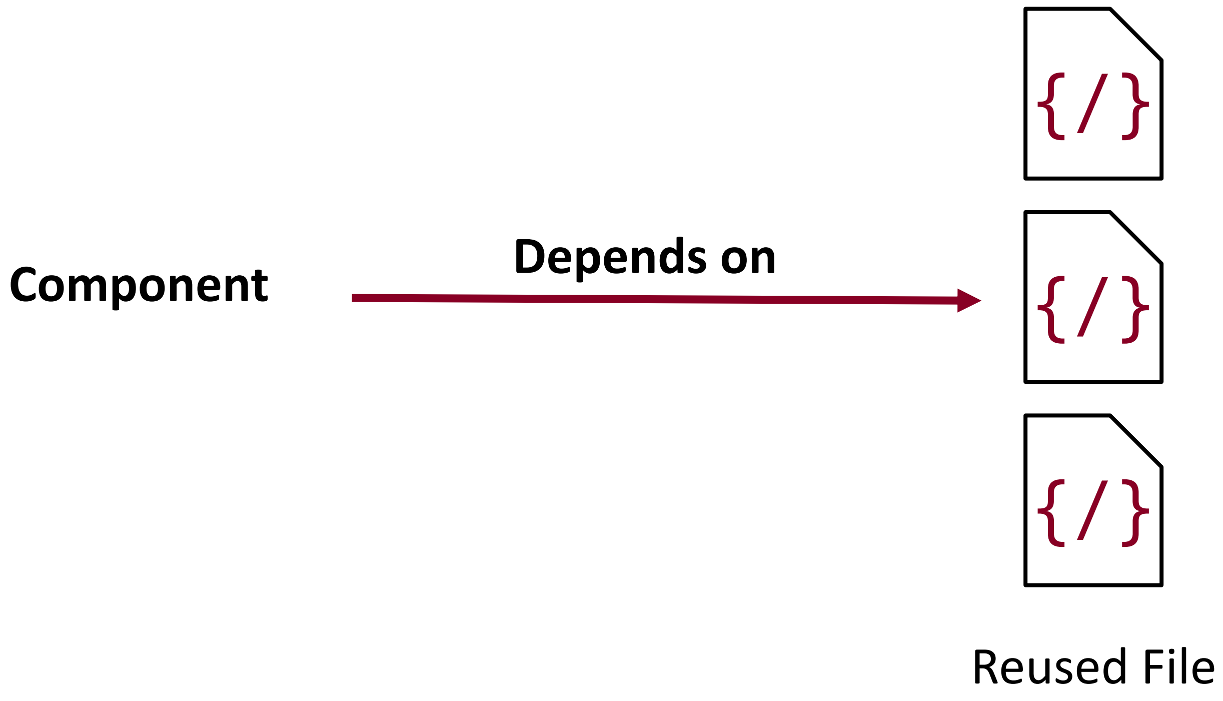
- Examine components that try to reuse other module *(#include directives*)



P2. Dependency Analysis (cont.)

Dependency analysis with other granularities

- Missing dependency can be caused by indistinguishable files



Component

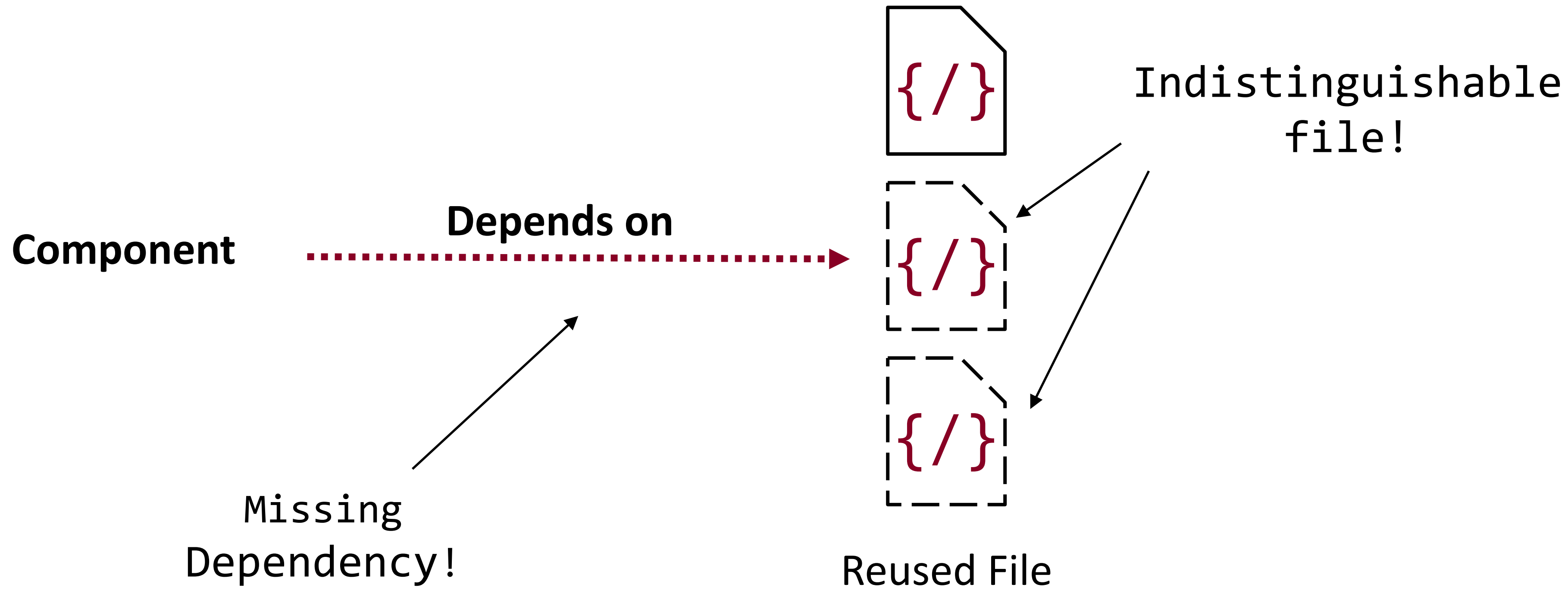
Depends on

Reused File

P2. Dependency Analysis (cont.)

Dependency analysis with other granularities

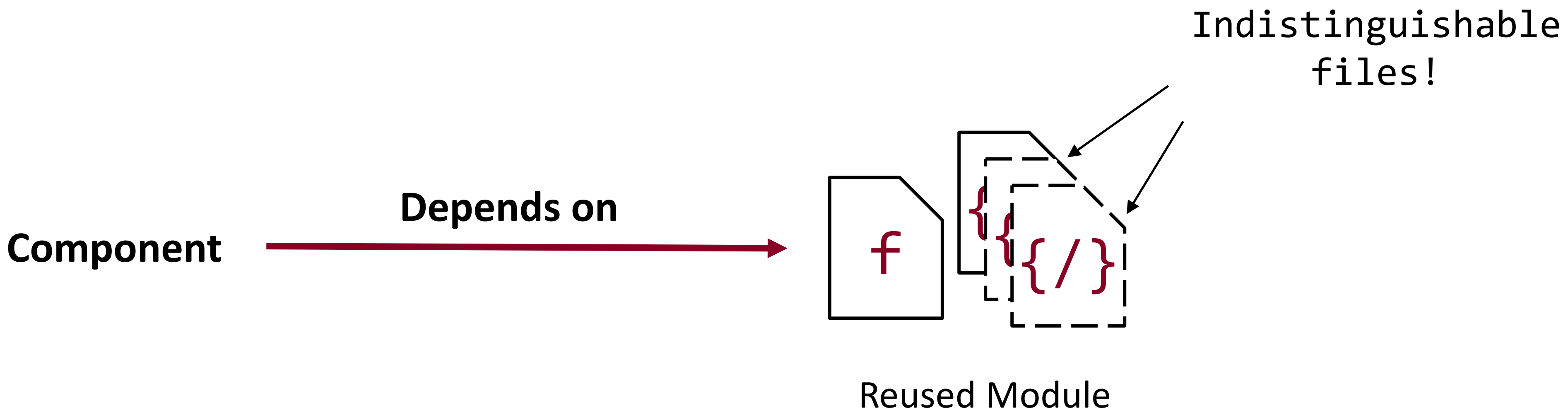
- Missing dependency can be caused by indistinguishable files



P2. Dependency Analysis (cont.)

Module Granularity

- We do not miss dependencies from **reused indistinguishable files**



Component

Depends on

Indistinguishable files!

Reused Module

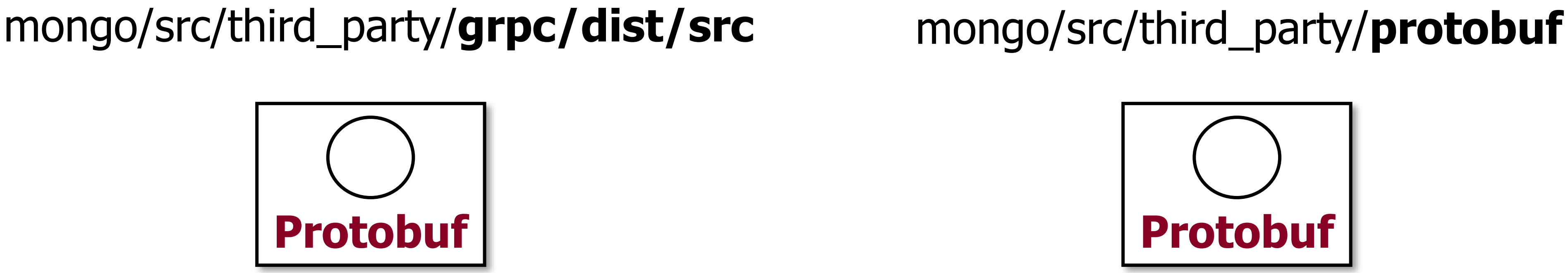
P3. Merging Components

- CNEPS merges components that are cloned from **the same project**

  (1) Which directory is the component cloned at? (cloned path)

  (2) Who includes this component? (parent component)

  (3) Is there a duplicated file? (the existence of the same files)



mongo/src/third_party/**grpc/dist/src**

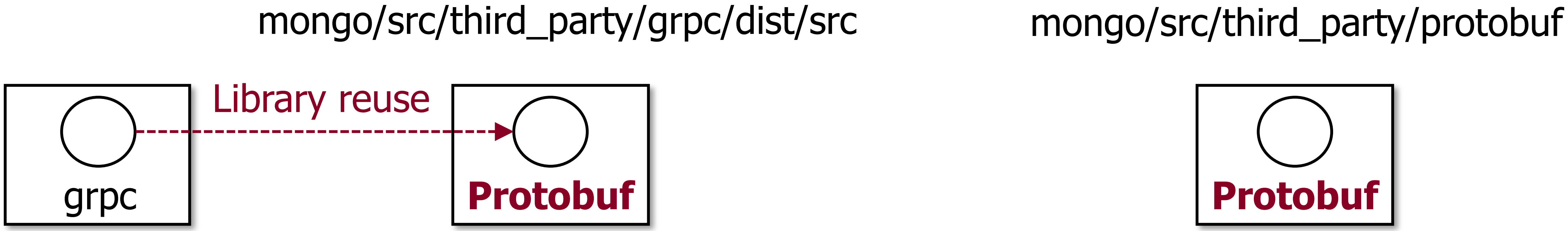
Protobuf

mongo/src/third_party/**protobuf**

Protobuf

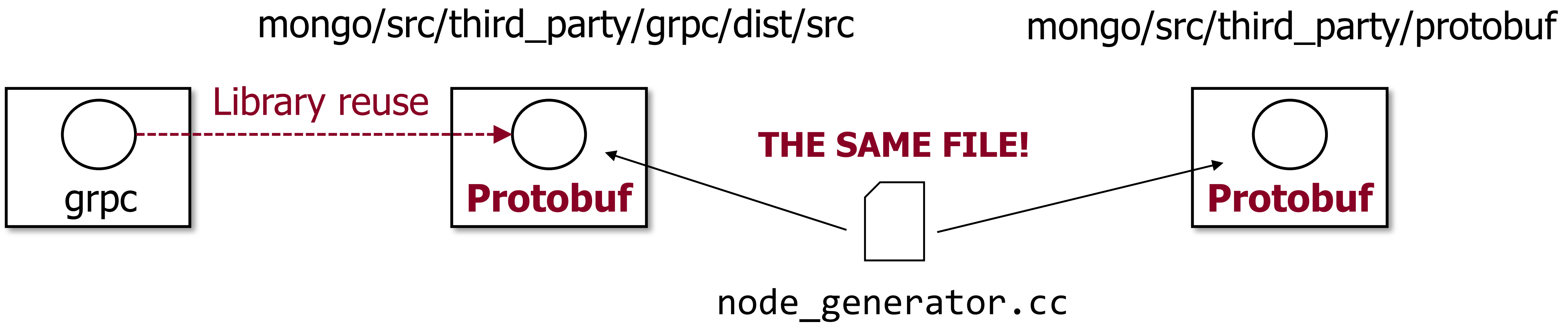
P3. Merging Components

- CNEPS merges components that are cloned from **the same project**
  - (1) Which directory is the component cloned at? (cloned path)
  - **(2) Who includes this component? (parent component)**
  - (3) Is there a duplicated file? (the existence of the same files)

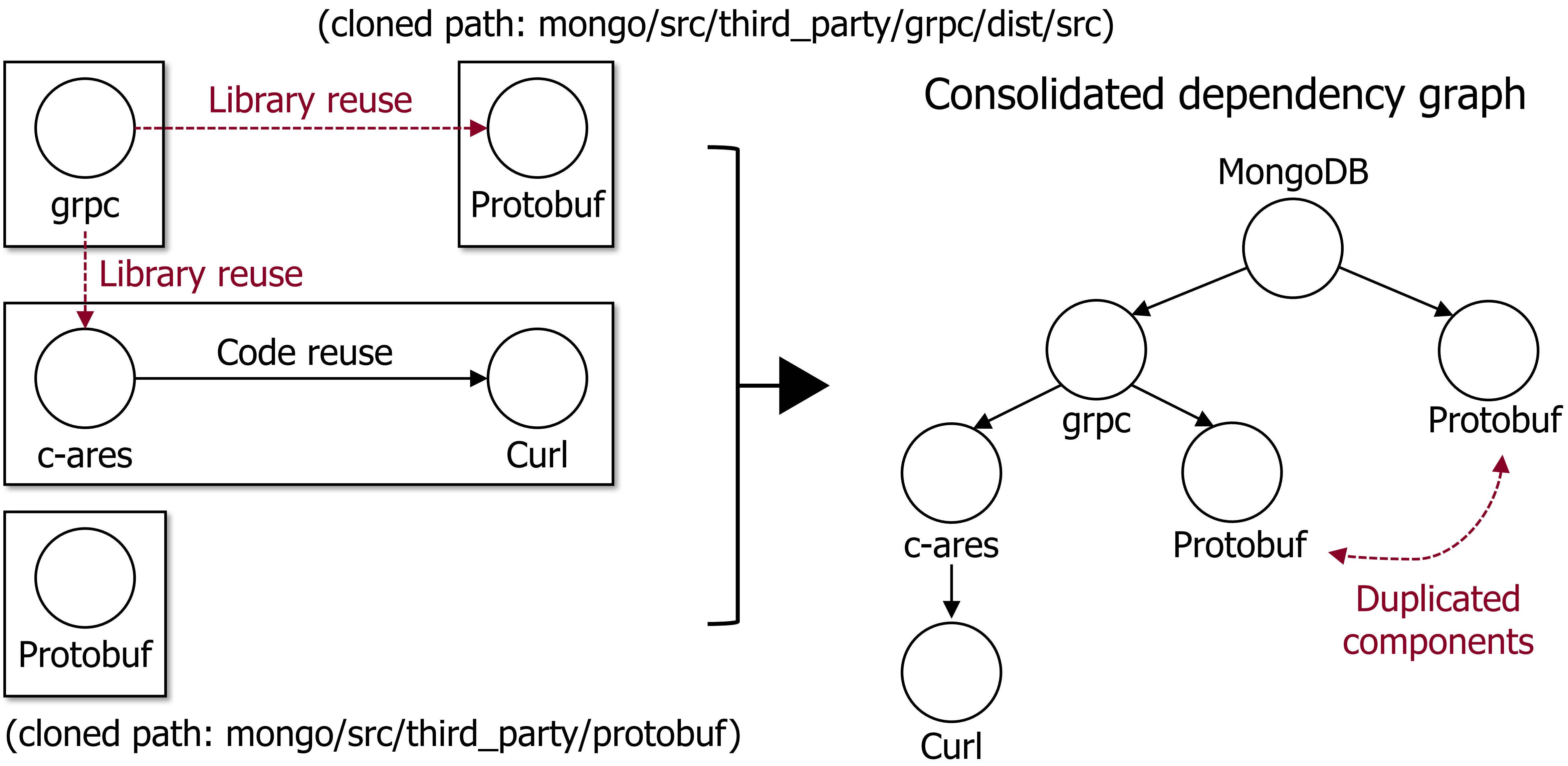

mongo/src/third_party/grpc/dist/src
mongo/src/third_party/protobuf
Library reuse
grpc
Protobuf
Protobuf

P3. Merging Components

- CNEPS merges components that are cloned from **the same project**
  - (1) Which directory is the component cloned at? (cloned path)
  - (2) Who includes this component? (parent component)
  - **(3) Is there a duplicated file? (the existence of the same files)**


mongo/src/third_party/grpc/dist/src
mongo/src/third_party/protobuf
grpc
Library reuse
Protobuf
THE SAME FILE!
Protobuf
node_generator.cc

Graph Output


(cloned path: mongo/src/third_party/grpc/dist/src)
grpc
Library reuse
Protobuf
Library reuse
c-ares
Code reuse
Curl
Protobuf
(cloned path: mongo/src/third_party/protobuf)
Consolidated dependency graph
MongoDB
grpc
Protobuf
c-ares
Protobuf
Curl
Duplicated components

Evaluation

EVALUATION

Q1. Is really CNEPS performing? (Accuracy of CNEPS)

Dataset

- Top 100 C/C++ open-source software from GitHub
  - Collected based on the number of stargazers

Comparison Target

- Centris (ICSE’21), an approach that detects reused components
- We advanced Centris to detect dependency **between** components

Woo, Seunghoon, et al. "CENTRIS: A precise and scalable approach for identifying modified open-source software reuse." 2021 ICSE

Accuracy of CNEPS

- (*Metric*) Examined *Precision*, *Recall* of the **dependencies**

- CNEPS outperformed existing approach with

  89.9% Precision and 93.2% Recall

- Discovered around 2.2 times more correct dependencies

Approaches	Identified Deps	Precision	Recall
Centris	219	63.5%	42.5%
CNEPS	**480**	**89.9%**	**93.2%**

Accuracy of CNEPS

- (1) Accuracy in identifying component of **indistinguishable files**

#identified indistinguishable files	TP	FP	Precision
34,611	31,681	2,930	**91.5%**

- (2) Accuracy in distinguishing **duplicated components**

# All Components	#identified Duplicated Components	TP	FP	Precision
297	40	33	7	**82.5%**

Impact of CNEPS

Q2. Is really CNEPS useful? (Impact of CNEPS)

Dataset

- Collected 1,000 C/C++ OSS based on stargazers
- Examined the number of dependencies discovered when challenges are dealt

Impact of CNEPS

Q2. Is really CNEPS useful? (Impact of CNEPS)

Dataset

- Collected 1,000 C/C++ OSS based on stargazers
- Examined the number of dependencies discovered when challenges are dealt

672 Deps

Non-considered

Impact of CNEPS

Q2. Is really CNEPS useful? (Impact of CNEPS)

Dataset

- Collected 1,000 C/C++ OSS based on stargazers
- Examined the number of dependencies discovered when challenges are dealt

672 Deps

Non-considered

861 Deps, +28%

Indistinguishable files

Impact of CNEPS

Q2. Is really CNEPS useful? (Impact of CNEPS)

Dataset

- Collected 1,000 C/C++ OSS based on stargazers
- Examined the number of dependencies discovered when challenges are dealt

672 Deps

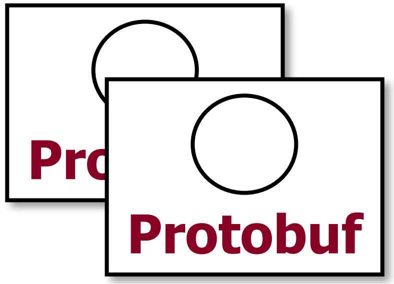
Non-considered

861 Deps, +28%

{/}

Indistinguishable files

919 Deps, +36%



Duplicated Components

Impact of CNEPS

Q2. Is really CNEPS useful? (Impact of CNEPS)

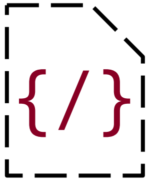
Dataset

- Collected 1,000 C/C++ OSS based on stargazers
- Examined the number of dependencies discovered when challenges are dealt

672 Deps

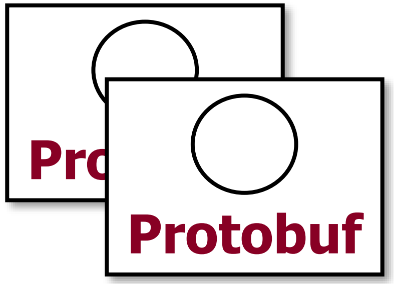
Non-considered

861 Deps, +28%



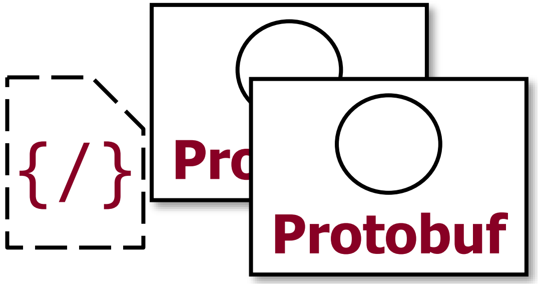
Indistinguishable Files

919 Deps, +36%



Duplicated Components

1179 Deps, +75%

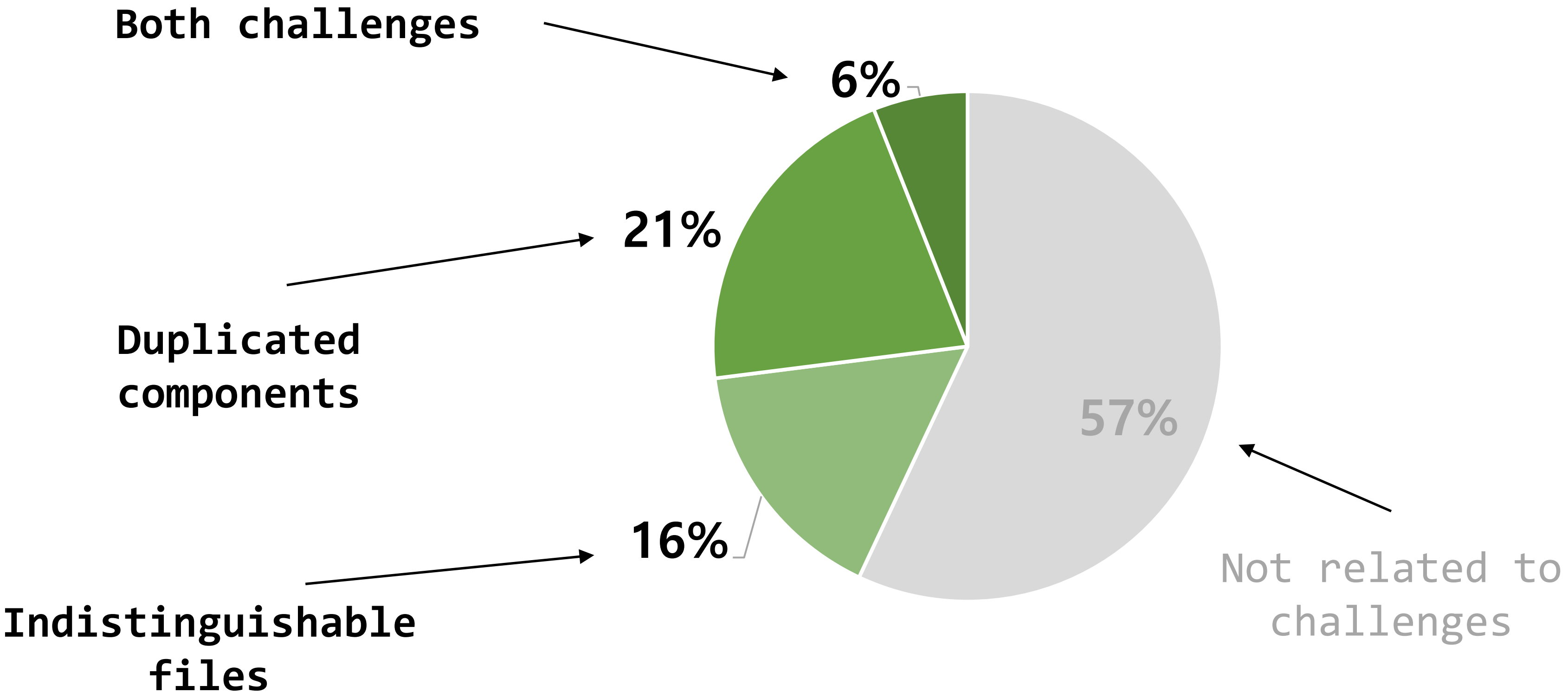


Both

Impact of CNEPS

Distribution of dependencies

- CNEPS was able to examine **75%** more dependencies!



Conclusion

- We present CNEPS, a precise approach for dependency analysis with accuracy of **89.9% precision** and **93.2% recall**

- CNEPS was able to examine **75%** more dependencies by dealing with **indistinguishable files** and **duplicated components**

- Equipped with CNEPS, developers can …

  1) provide more precise software transparency (e.g., SBOM)

  2) examine exploitability of vulnerabilities

Q&A

CNEPS Source code

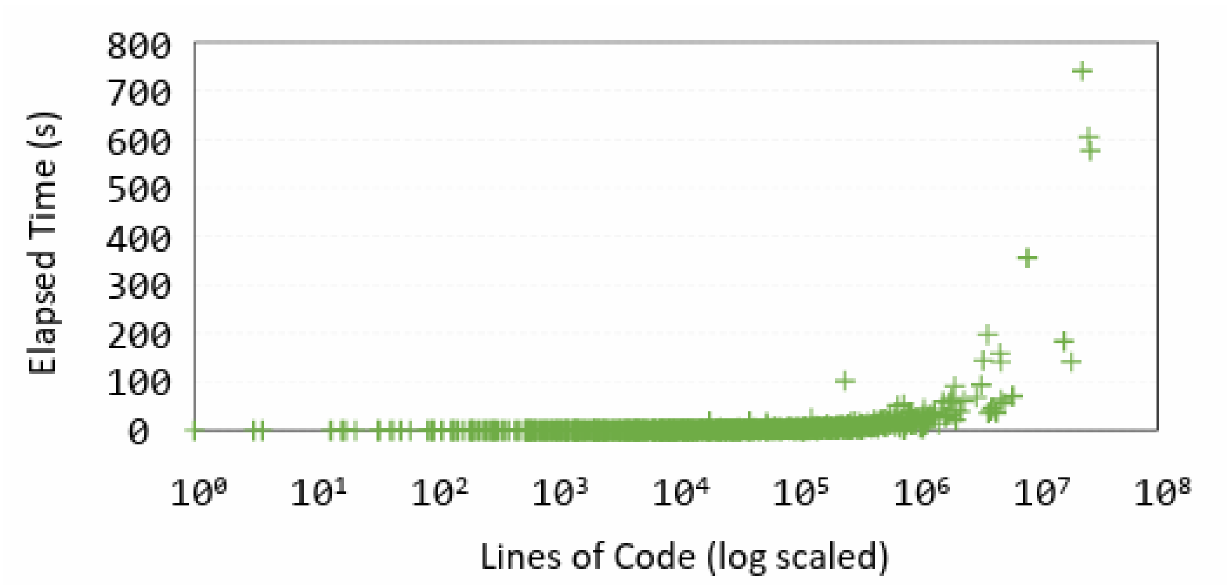
- CNEPS repository: https://github.com/sodium49/CNEPS-public

Contact

- Email: nooryyaa@korea.ac.kr
- Computer & Communication Security Lab (https://ccs.korea.ac.kr)
- Software Security and Privacy Laboratory (https://ssp.korea.ac.kr)
- Center for Software Security and Assurance (https://cssa.korea.ac.kr)

Appendix – scalability

- Elapsed time – lines of code
- Average 8.22s



Appendix – FPs and FNs

- FP: Indistinguishable file problem
- Because we generate module using Name of the function and declaration, sometimes error occurs
- e.g., function with same name (memcpy)

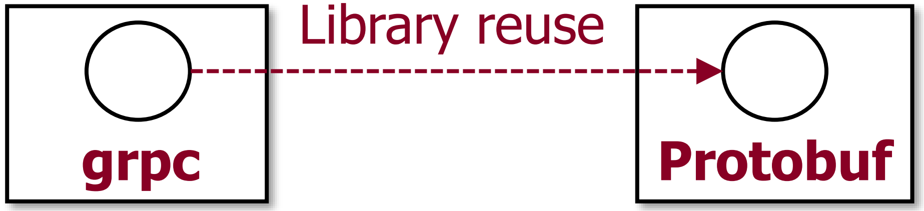
- FP: Determining the exact module when there is a header with the same name as a system library
- *#include <string.h>*

- FN: multiple header with the same name
- e.g., 10 headers with name <foo.h> within same path distance

Appendix – Advanced Centris

- Implemented with verification method Centris (ICSE’21) used
- In the following example, *grpc* depends on *Protobuf*

mongo/src/third_party/**grpc**/dist/src

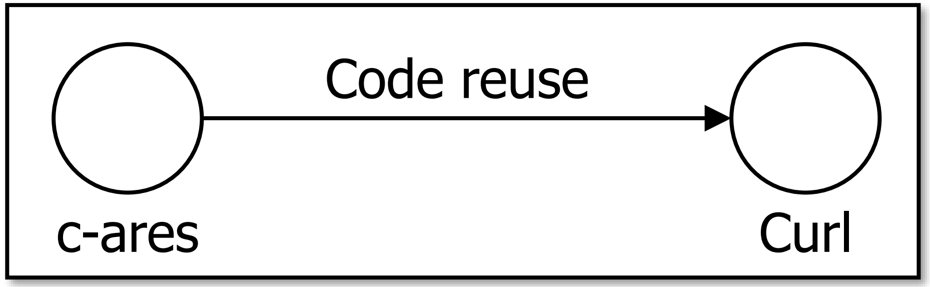


Appendix – Code Reuse Analysis

(1) Internal Reuse Analysis

- Count number of included components

- For example, in *MongoDB*, *ares.h* module contains…
  - **23 *c-ares***
  - 7 *curl*
  - 34 *indistinguishable files*



➔This module is cloned from *c-ares*

- Indistinguishable files are also cloned from *c-ares*

Appendix – Library reuse analysis

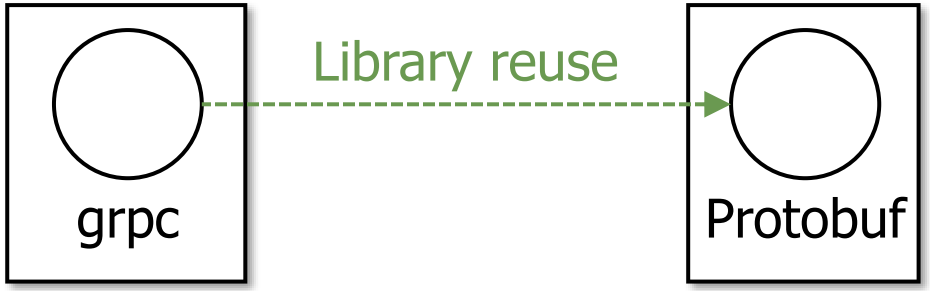
(2) Library reuse analysis

- Examine functions reusing another component by **importing header**
    - **“#include” directive**

- For example, in *MongoDB*, a module of a *grpc* was reusing *protobuf*

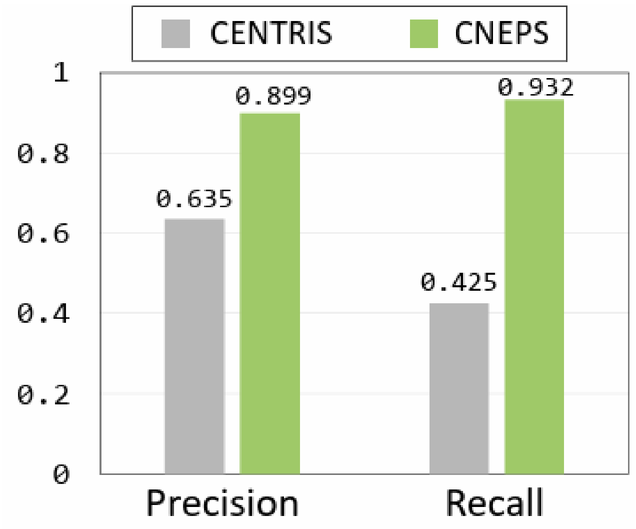
grpc importing *protobuf!*

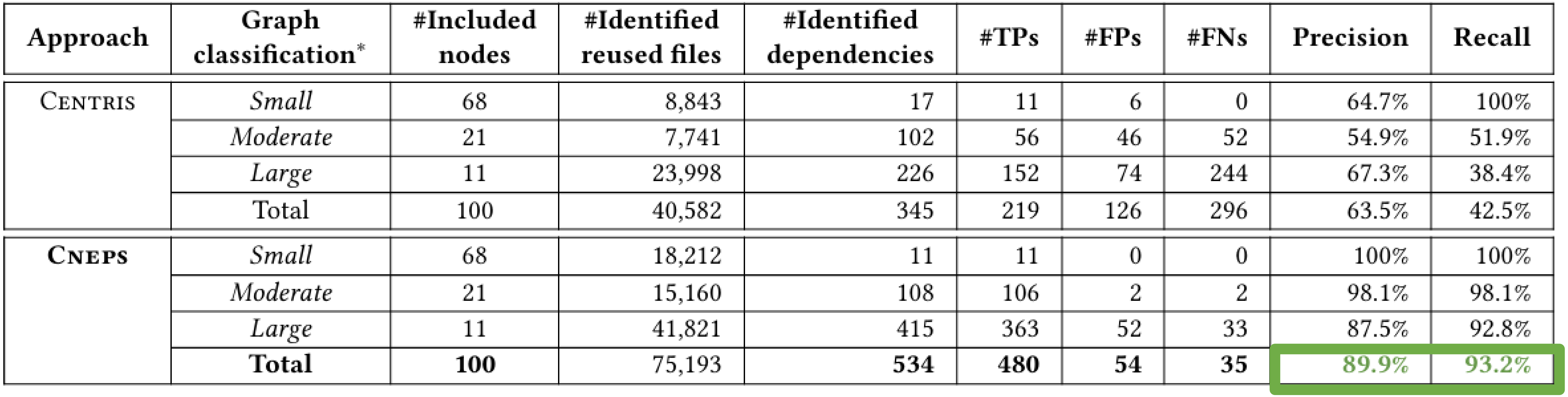
```
// Generates Objective C gRPC service interface out of Protobuf IDL.

#include <memory>
#include <google/protobuf/compiler/objectivec/objectivec_helpers.h>
```



Accuracy

- CNEPS outperformed existing approaches with
  - 89.9% Precision
  - 93.2% Recall
- Discovered around 2.2 times more correct dependencies



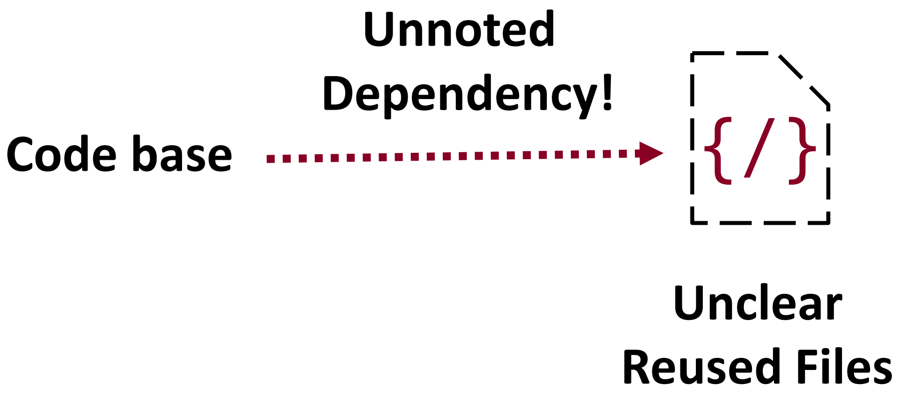


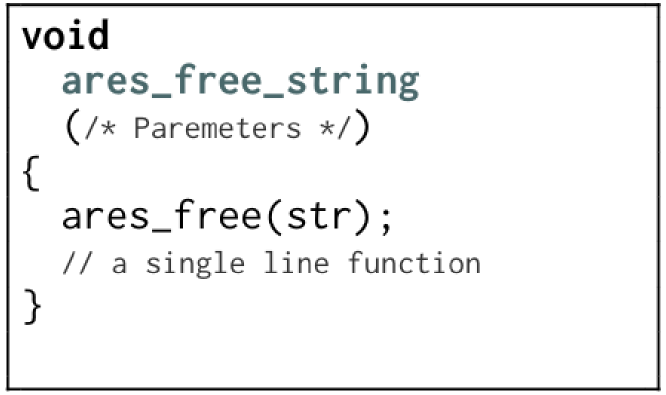
Approach	Graph classification*	#Included nodes	#Identified reused files	#Identified dependencies	#TPs	#FPs	#FNs	Precision	Recall
Centris	*Small*	68	8,843	17	11	6	0	64.7%	100%
	Moderate	21	7,741	102	56	46	52	54.9%	51.9%
	Large	11	23,998	226	152	74	244	67.3%	38.4%
	Total	100	40,582	345	219	126	296	63.5%	42.5%
Cneps	*Small*	68	18,212	11	11	0	0	100%	100%
	Moderate	21	15,160	108	106	2	2	98.1%	98.1%
	Large	11	41,821	415	363	52	33	87.5%	92.8%
	Total	**100**	75,193	**534**	**480**	**54**	**35**	89.9%	93.2%

Appendix – indistinguishable files

Challenge 1: Indistinguishable files

- Files that are unclear whether reused or not
- Unidentified reused files may lead to unidentified dependencies
  - e.g., single-lined function, implementation of the cryptographic function
  - libcrypto, openssl, openssh, …





```
void
  ares_free_string
  (/* Paremeters */)
{
  ares_free(str);
  // a single line function
}
```

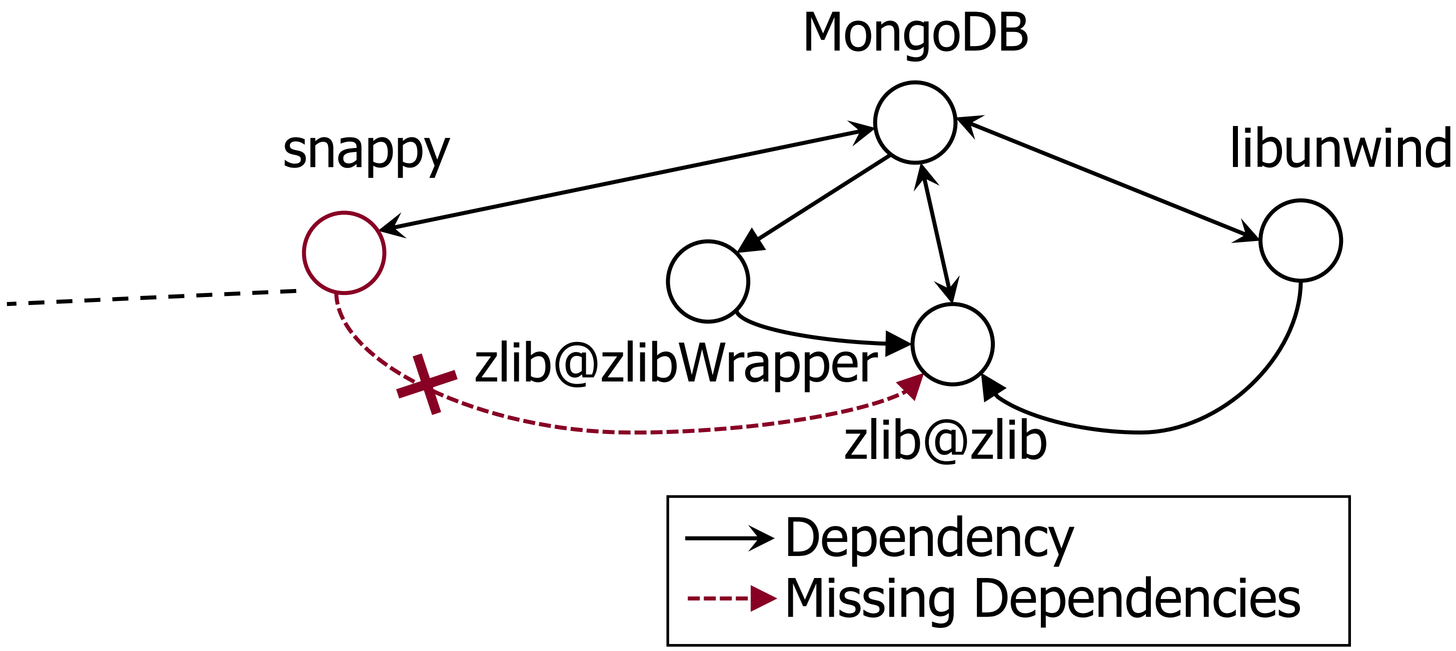
Motivation

Challenge 1: Indistinguishable files

- Files that are unclear whether reused or not
- Unidentified reused files may lead to unidentified dependencies
  - e.g., single-lined function, implementation of the cryptographic function

Unnoted Reused File!

```
#ifdef HAVE_LIBZ
#include "zlib.h"
#endif

#ifdef HAVE_LIBLZO2
#include "lzo/lzo1x.h"
#endif
```


MongoDB
snappy
libunwind
zlib@zlibWrapper
zlib@zlib
Dependency
Missing Dependencies