Snyk Code Analysis

Marcus Boerger

snyk

Overview

Parsing -> Processing -> Querying -> Presenting

Overview

Parsing -> Processing -> Querying -> Presenting

Easy :-)

Intro

Every piece of code is processed with multiple “engines”.

Here we look at Snyk’s Analysis Engine.

We process and learn from:

- Open Source -> For all our Users.
  - We process whole repositories in pipelines.
  - Specialists analyze Code and create rules to identify Issues.
- Custom Code repositories -> For the respective User.
- We let Users run their own Analysis extensions.

For the analysis all code is transferred into a StarDb and Comp/EventGraph.

Code to Abstract

Let’s consider some C++ code

```
class Logger {
 public:
  void Log(std::string s);
};
```

Code to Abstract

Let’s consider some C++ code

```
class Logger {
 public:
  void Log(std::string s);
};
```

We have got:

- A class “Logger”
  - A method “Logger::Log”
    - With a parameter named “s”
    - Which has type “std::string”

Code to Abstract

Let’s consider some C++ code

```
class Logger {
 public:
  void Log(std::string s);
};
```

```
class Data {
 public:
  std::string ToString();
  void Log() {
    logger_.Log(ToString());
  }
};
```

We have got:

- A class “Logger”
  - A method “Logger::Log”
    - With a parameter named “s”
    - Which has type “std::string”

snyk

Code to Abstract

Let’s consider some C++ code

```
class Logger {
 public:
  void Log(std::string s);
};
```

```
class Data {
 public:
  std::string ToString();
  void Log() {
    logger_.Log(ToString());
  }
};
```

We have got:

- A class “Logger”
  - A method “Logger::Log”
    - With a parameter named “s”
    - Which has type “std::string”
- A class “Data”
  - A method “ToString”
  - A method “Log”
    - Calling “ToString”
    - Calling “logger_.Log”

Code to Abstract

Let’s consider some C++ code
All of this can be expressed in a graph

```
class Logger {
 public:
  void Log(std::string s);
};
```

```
class Data {
 public:
  std::string ToString();
  void Log() {
    logger_.Log(ToString());
  }
};
```

```
ClassDef[Logger] -- HasMethod -> Method[Log]
MethodDef[Log] -- Parameter:0 -> Type[std::string]
MethodDef[Log] -- Parameter:0 -> Name[s]
ClassDef[Data] -- HasMethod -> Method[ToString]
ClassDef[Data] -- HasMethod -> Method[Log]
MethodDef[ToString] -- ReturnType -> Type[std::string]
MethodDef[Log] -- Calls -> Method[ToString]
MethodDef[Log] -- Calls -> Method[this]
MethodDef[Log] -- Calls -> Method[Log]
MethodDef[Log] -- Calls -> Arg:0[Logger]
MethodDef[Log] -- Calls -> Arg:1[temp-arg]
```

Code to Abstract

Let's consider some C++ code
All of this can be expressed in a graph
Too much data

1 = "Logger"
2 = "Log"
3 = "std::string"
4 = "Data"

5 = "ToString"
6 = "this"
7 = temp-arg
8 = "s"

```
class Logger {
 public:
  void Log(std::string s);
};
```

```
class Data {
 public:
  std::string ToString();
  void Log() {
    logger_.Log(ToString());
  }
};
```

ClassDef[1] -- HasMethod -> Method[2]
MethodDef[2] -- Parameter:0 -> Type[3]
MethodDef[2] -- Parameter:0 -> Name[8]
ClassDef[4] -- HasMethod -> Method[5]
ClassDef[4] -- HasMethod -> Method[2]
MethodDef[5] -- ReturnType -> Type[3]
MethodDef[2] -- Calls -> Method[5]
MethodDef[2] -- Calls -> Method[6]
MethodDef[2] -- Calls -> Method[2]
MethodDef[2] -- Calls -> Arg:0[1]
MethodDef[2] -- Calls -> Arg:1[7]

Code to Abstract

Let's consider some C++ code
All of this can be expressed in a graph
Too much data

1 = "Logger"
2 = "Log"
3 = "std::string"
4 = "Data"

5 = "ToString"
6 = "this"
7 = temp-arg
8 = "s"

```
class Logger {
 public:
  void Log(std::string s);
};
```

```
class Data {
 public:
  std::string ToString();
  void Log() {
    logger_.Log(ToString());
  }
};
```

```
ClassDef[1] -- HasMethod -> Method[2] <- "Log"
MethodDef[2] -- Parameter:0 -> Type[3]
MethodDef[2] -- Parameter:0 -> Name[8]
ClassDef[4] -- HasMethod -> Method[5]
ClassDef[4] -- HasMethod -> Method[2]
MethodDef[5] -- ReturnType -> Type[3]
MethodDef[2] -- Calls -> Method[5]
MethodDef[2] -- Calls -> Method[6]
MethodDef[2] -- Calls -> Method[2]
MethodDef[2] -- Calls -> Arg:0[1]
MethodDef[2] -- Calls -> Arg:1[7]
```

Code to Abstract

Let's consider some C++ code
All of this can be expressed in a graph
Too much data

1 = "Logger"
2 = "Log"
3 = "std::string"
4 = "Data"

5 = "ToString"
6 = "this"
7 = temp-arg
8 = "s"

```
class Logger {
 public:
  void Log(std::string s);
};
```

```
class Data {
 public:
  std::string ToString();
  void Log() {
    logger_.Log(ToString());
  }
};
```

```
ClassDef[1] -- HasMethod -> Method[2] <- "Log"
MethodDef[2] -- Parameter:0 -> Type[3]
MethodDef[2] -- Parameter:0 -> Name[s]
ClassDef[4] -- HasMethod -> Method[5]
ClassDef[4] -- HasMethod -> Method[2]  <- Also "Log"
MethodDef[5] -- ReturnType -> Type[3]
MethodDef[2] -- Calls -> Method[5]
MethodDef[2] -- Calls -> Method[6]
MethodDef[2] -- Calls -> Method[2]
MethodDef[2] -- Calls -> Arg:0[1]
MethodDef[2] -- Calls -> Arg:1[7]
```

Code to Abstract

So far we have all of the code

```
1 = ClassDef
2 = Method
3 = MethodDef
4 = Type
5 = Name
6 = Arg:0
```

```
1[1] -- 1 -> 2[2]
2[2] -- 2 -> 4[3]
3[2] -- 2 -> 5[8]
1[4] -- 1 -> 2[5]
1[4] -- 1 -> 2[2]
3[5] -- 3 -> 4[3]
3[2] -- 4 -> 2[5]
3[2] -- 4 -> 2[6]
3[2] -- 4 -> 2[2]
3[2] -- 4 -> 6[1]
3[2] -- 4 -> 6[7]
```

```
1 = “Logger”
2 = “Log”
3 = “std::string”
4 = “Data”
5 = “ToString”
6 = “this”
7 = temp-arg
8 = “s”
```

```
1 = HasMethod
2 = Parameter:0
3 = ReturnType
4 = Calls
```

Let’s make this usable

So far we have the code
Let’s add some “meta” information

```
class Logger {
 public:
  void Log(std::string s);
};
```

logger.h

```
class Data {
 public:
  std::string ToString();
  void Log() {
    logger_.Log(ToString());
  }
};
```

data.h

snyk

Let’s make this usable

So far we have the code
Let’s add some “meta” information

```
class Logger {
 public:
  void Log(std::string s);
};
```

InFile -> File[logger.h]

```
class Data {
 public:
  std::string ToString();
  void Log() {
    logger_.Log(ToString());
  }
};
```

InFile -> File[data.h]

snyk

Let’s make this usable

So far we have the code & some “meta” information

```
1 = ClassDef
2 = Method
3 = MethodDef
4 = Type
5 = Name
6 = Arg:0
7 = File
```

```
1[1] -- 1 -> 2[2]
2[2] -- 2 -> 4[3]
3[2] -- 2 -> 5[3]
1[4] -- 1 -> 2[5]
1[4] -- 1 -> 2[2]
3[5] -- 3 -> 4[3]
3[2] -- 4 -> 2[5]
3[2] -- 4 -> 2[6]
3[2] -- 4 -> 2[2]
3[2] -- 4 -> 6[1]
3[2] -- 4 -> 6[7]
```

```
1 = “Logger”
2 = “Log”
3 = “std::string”
4 = “Data”
5 = “ToString”
6 = “this”
7 = temp-arg
8 = “s”
```

```
1 = HasMethod
2 = Parameter:0
3 = ReturnType
4 = Calls
```

Let’s make this usable

(A): ClassDef[Logger] HasMethod Method[Log]

1 = ClassDef
2 = Method
3 = MethodDef
4 = Type
5 = Name
6 = Arg:0
7 = File

1[1] -- 1 -> 2[2] A
2[2] -- 2 -> 4[3]
3[2] -- 2 -> 5[8]
1[4] -- 1 -> 2[5]
1[4] -- 1 -> 2[2]
3[5] -- 3 -> 4[3]
3[2] -- 4 -> 2[5]
3[2] -- 4 -> 2[6]
3[2] -- 4 -> 2[2]
3[2] -- 4 -> 6[1]
3[2] -- 4 -> 6[7]

1 = “Logger”
2 = “Log”
3 = “std::string”
4 = “Data”
5 = “ToString”
6 = “this”
7 = temp-arg
8 = “s”

1 = HasMethod
2 = Parameter:0
3 = ReturnType
4 = Calls

Let’s make this usable

(A): ClassDef[Logger] HasMethod Method[Log] InFile File[logger.h]

1 = ClassDef
2 = Method
3 = MethodDef
4 = Type
5 = Name
6 = Arg:0
7 = File

1[1] -- 1 -> 2[2] A
2[2] -- 2 -> 4[3]
3[2] -- 2 -> 5[8]
1[4] -- 1 -> 2[5]
1[4] -- 1 -> 2[2]
3[5] -- 3 -> 4[3]
3[2] -- 4 -> 2[5]
3[2] -- 4 -> 2[6]
3[2] -- 4 -> 2[2]
3[2] -- 4 -> 6[1]
3[2] -- 4 -> 6[7]
1[1] -- 5 -> 7[9] A

1 = “Logger”
2 = “Log”
3 = “std::string”
4 = “Data”
5 = “ToString”
6 = “this”
7 = temp-arg
8 = “s”
9 = “logger.h”

1 = HasMethod
2 = Parameter:0
3 = ReturnType
4 = Calls
5 = InFile

Let’s make this usable

(B): ClassDef[Data] HasMethod Medthod[Log]

1 = ClassDef
2 = Method
3 = MethodDef
4 = Type
5 = Name
6 = Arg:0
7 = File

1[1] -- 1 -> 2[2] A
2[2] -- 2 -> 4[3]
3[2] -- 2 -> 5[8]
1[4] -- 1 -> 2[5]
1[4] -- 1 -> 2[2] B
3[5] -- 3 -> 4[3]
3[2] -- 4 -> 2[5]
3[2] -- 4 -> 2[6]
3[2] -- 4 -> 2[2]
3[2] -- 4 -> 6[1]
3[2] -- 4 -> 6[7]
1[1] -- 5 -> 7[9] A

1 = “Logger”
2 = “Log”
3 = “std::string”
4 = “Data”
5 = “ToString”
6 = “this”
7 = temp-arg
8 = “s”
9 = “logger.h”

1 = HasMethod
2 = Parameter:0
3 = ReturnType
4 = Calls
5 = InFile

Let’s make this usable

(B): ClassDef[Data] HasMethod Medthod[Log] InFile File[data.h]

1 = ClassDef
2 = Method
3 = MethodDef
4 = Type
5 = Name
6 = Arg:0
7 = File

1[1] -- 1 -> 2[2] A
2[2] -- 2 -> 4[3]
3[2] -- 2 -> 5[8]
1[4] -- 1 -> 2[5]
1[4] -- 1 -> 2[2] B
3[5] -- 3 -> 4[3]
3[2] -- 4 -> 2[5]
3[2] -- 4 -> 2[6]
3[2] -- 4 -> 2[2]
3[2] -- 4 -> 6[1]
3[2] -- 4 -> 6[7]
1[1] -- 5 -> 7[9] A
1[4] -- 5 -> 7[10] B

1 = “Logger”
2 = “Log”
3 = “std::string”
4 = “Data”
5 = “ToString”
6 = “this”
7 = temp-arg
8 = “s”
9 = “logger.h”
10 = “data.h”

1 = HasMethod
2 = Parameter:0
3 = ReturnType
4 = Calls
5 = InFile

Let’s make this usable

(C): ClassDef[Data] HasMethod Medthod[Log] Calls Method[Log] Arg:0[Logger]

1 = ClassDef
2 = Method
3 = MethodDef
4 = Type
5 = Name
6 = Arg:0
7 = File

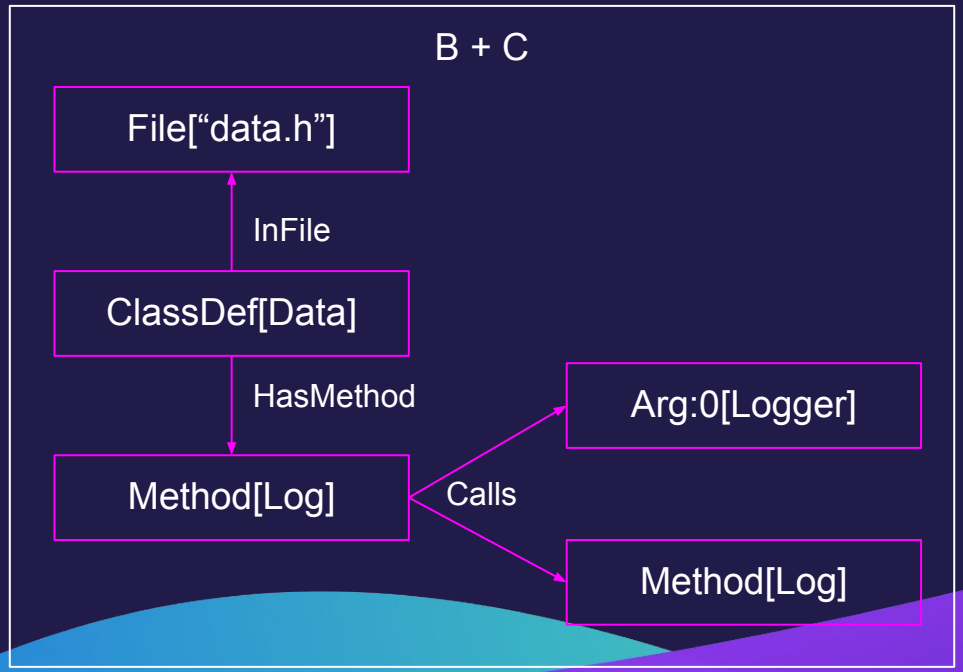
1[1] -- 1 -> 2[2] A
2[2] -- 2 -> 4[3]
3[2] -- 2 -> 5[8]
1[4] -- 1 -> 2[5]
1[4] -- 1 -> 2[2] B
3[5] -- 3 -> 4[3]
3[2] -- 4 -> 2[5]
3[2] -- 4 -> 2[6]
3[2] -- 4 -> 2[2] C
3[2] -- 4 -> 6[1] C
3[2] -- 4 -> 6[7]
1[1] -- 5 -> 7[9] A
1[4] -- 5 -> 7[10] B

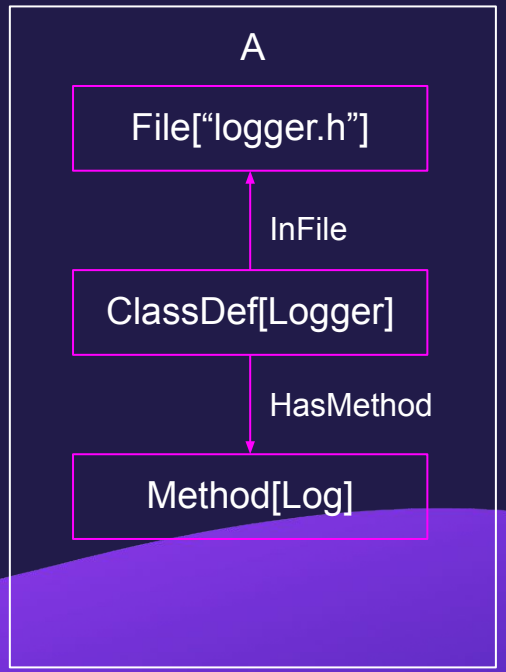
1 = “Logger”
2 = “Log”
3 = “std::string”
4 = “Data”
5 = “ToString”
6 = “this”
7 = temp-arg
8 = “s”
9 = “logger.h”
10 = “data.h”

1 = HasMethod
2 = Parameter:0
3 = ReturnType
4 = Calls
5 = InFile

Example: Dependencies

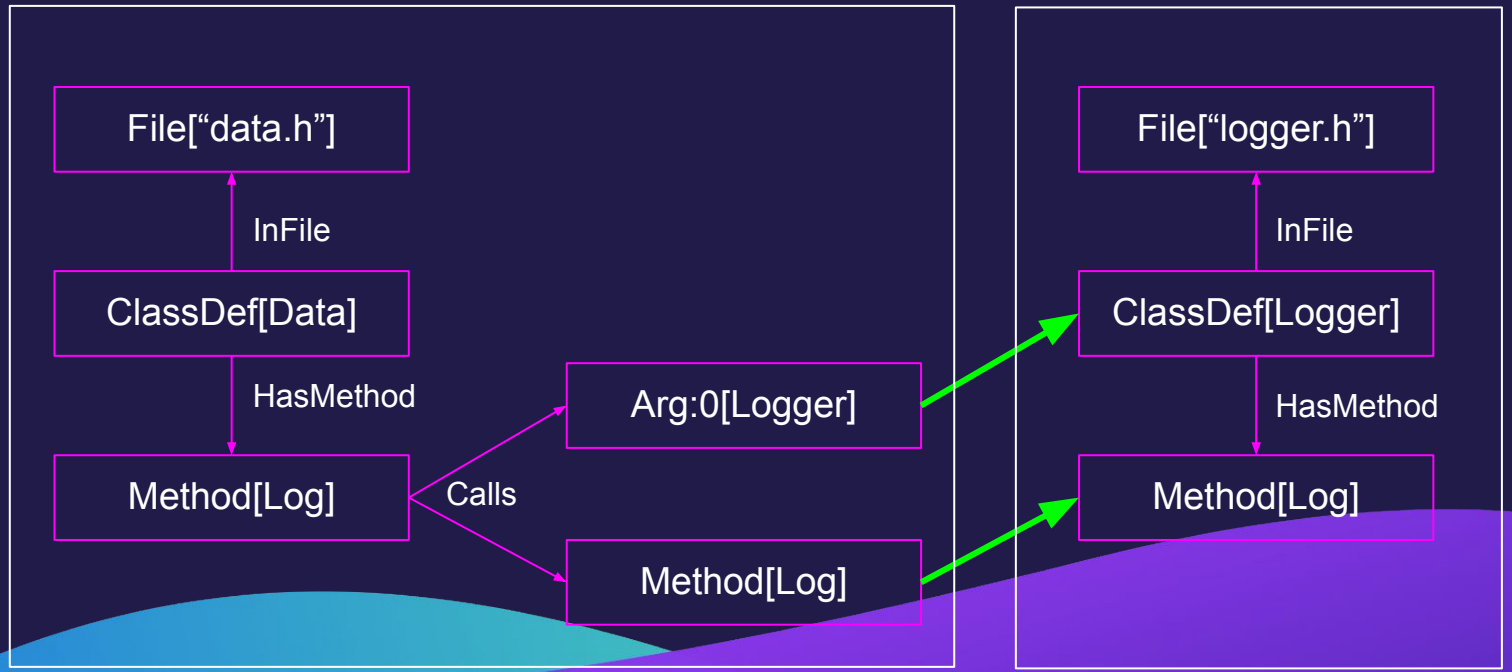
(A): ClassDef[Logger] HasMethod Method[Log] InFile File[logger.h]
(B): ClassDef[Data] HasMethod Method[Log] InFile File[data.h]
(C): ClassDef[Data] HasMethod Method[Log] Calls Method[Log] Arg:0[Logger]





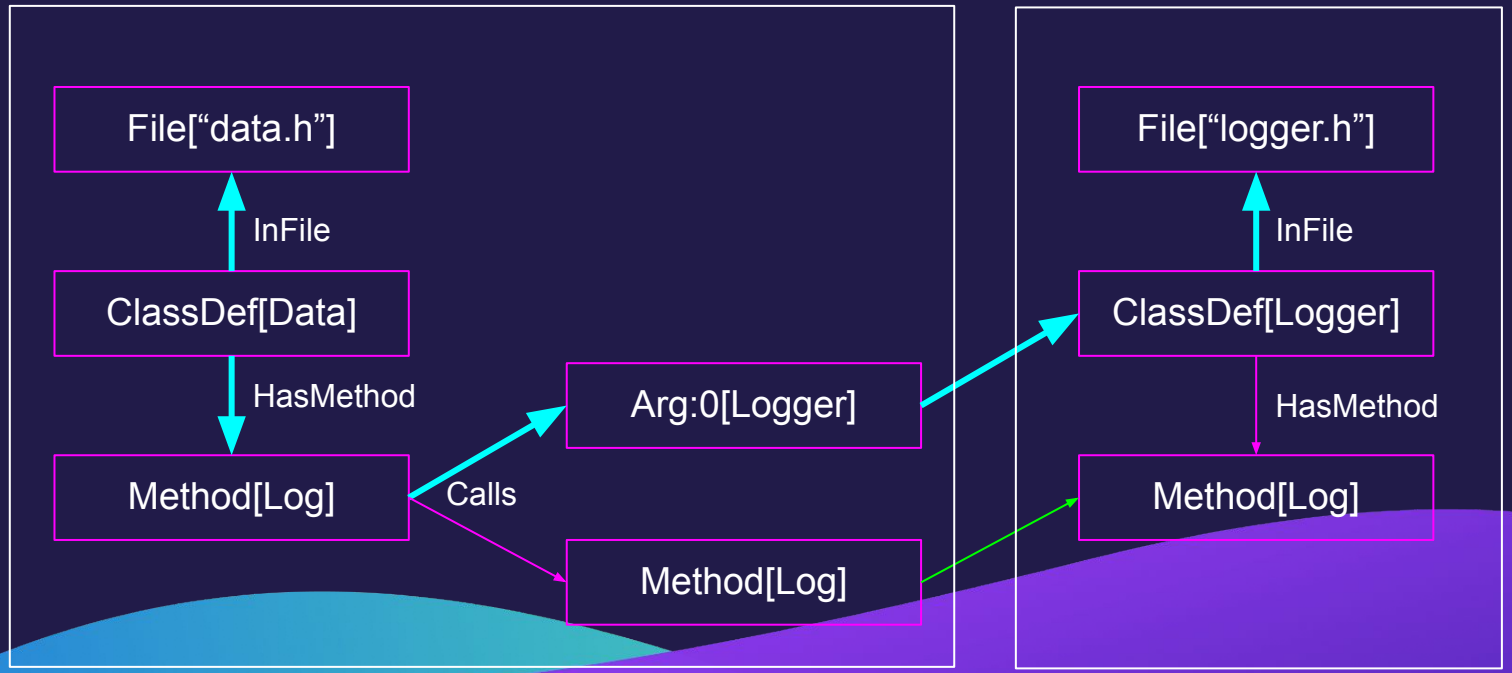
Example: Dependencies

We can *“of course”* match the class args and method calls.



Example: Dependencies

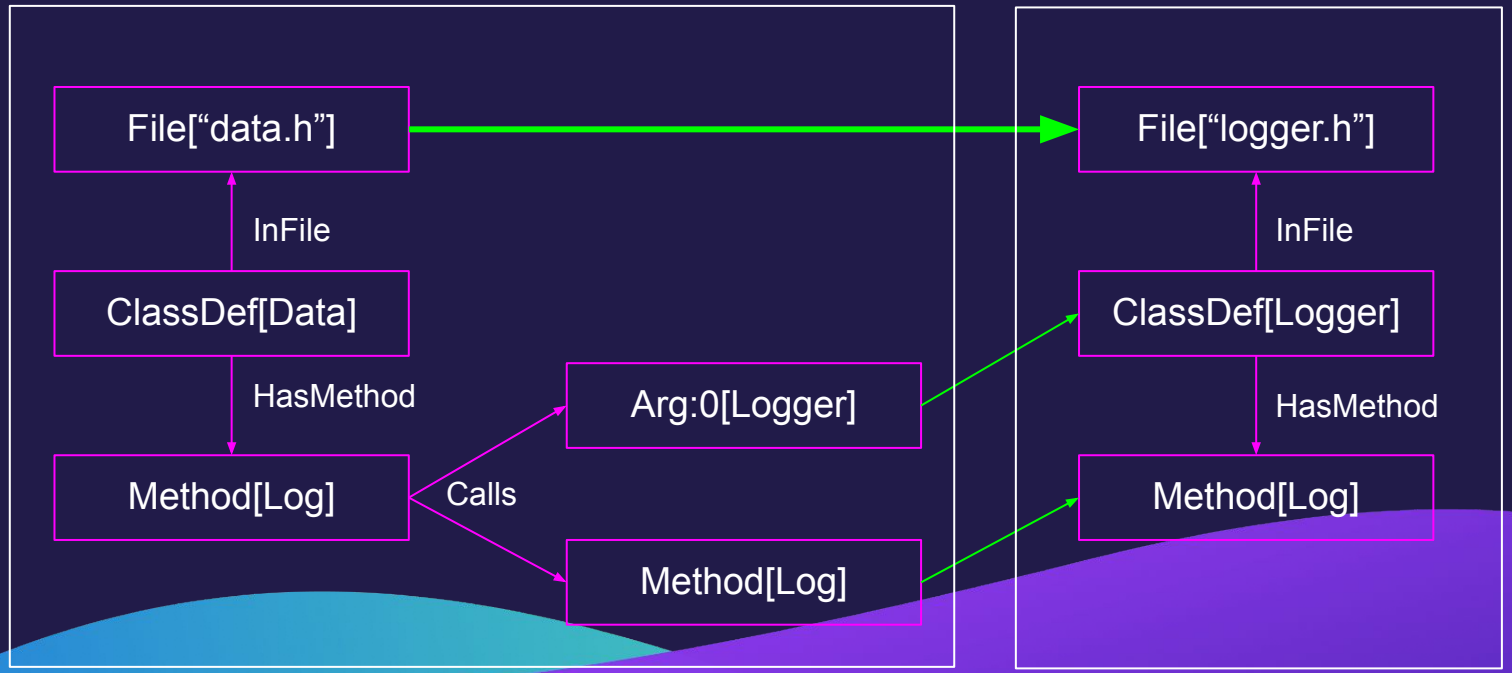
We can *“of course”* match the class args and method calls.
And now there is a path from data.h to logger.h.



Example: Dependencies

So there we go:

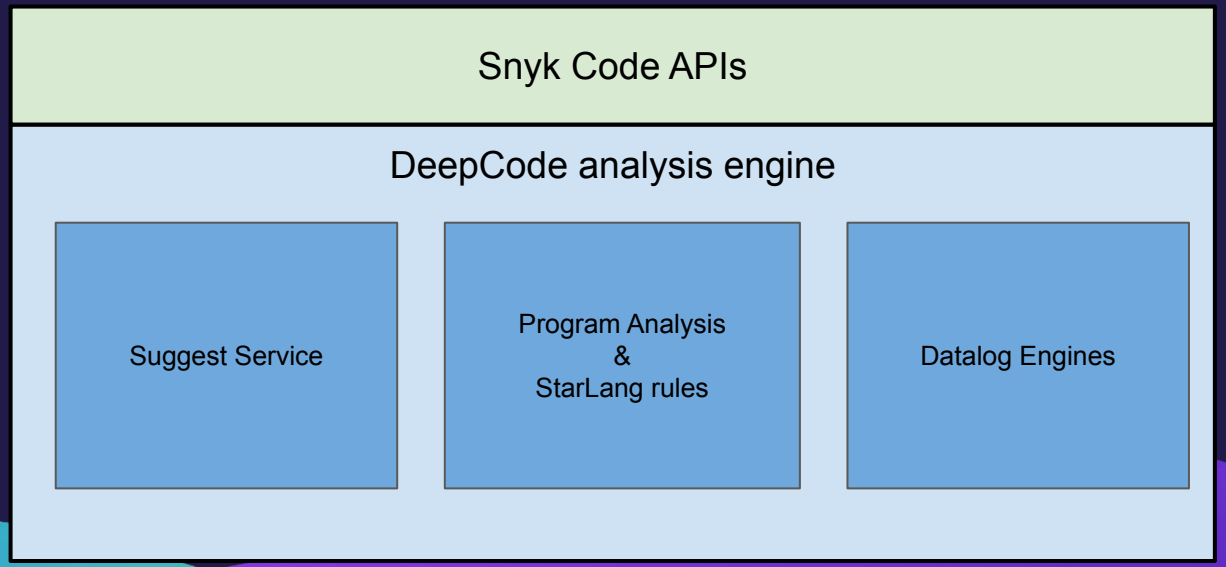
File data.h depends on file logger.h



snyk

Analysis

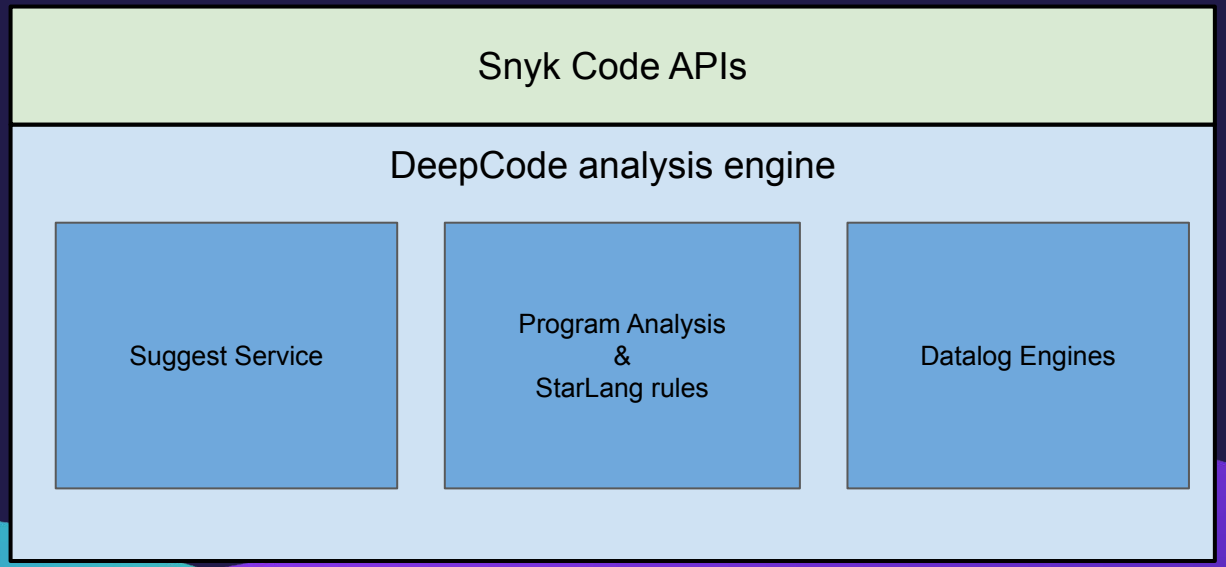
We have an abstract schema that supports several languages.
We use global data and rules via Datalog. Similar to the prior example.
Given a repository or repo and file-set, we can run all our analysis under an API.



Analysis

We have an abstract schema that supports several languages.
We use global data and rules via Datalog. Similar to the prior example.
Given a repository or repo and file-set, we can run code analysis under an API.

IDE checks
PR checks
Repository check



What we have so far

Parsing -> Processing -> Querying -> Presenting

What we have so far

Parsing -> Processing -> Querying -> Presenting

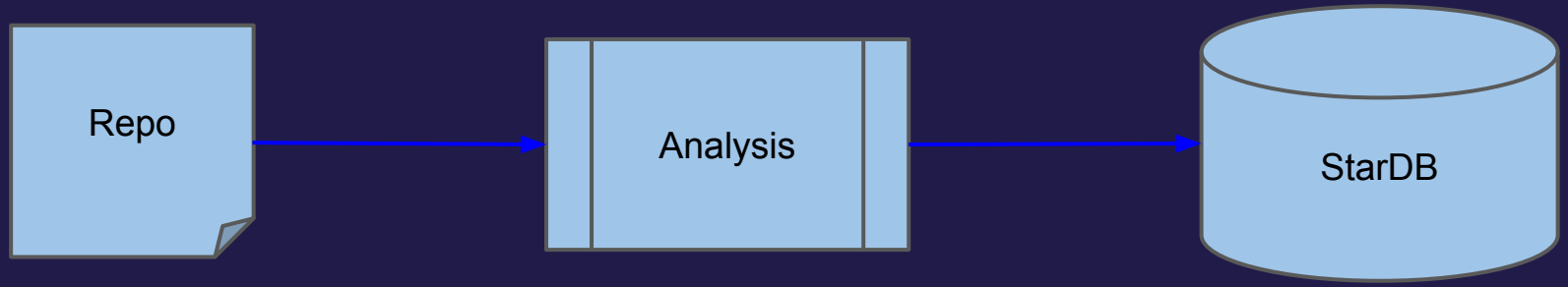
Remember: Easy :-)

And now some AI

Parsing -> Processing -> Querying -> Presenting

AI and Code Analysis

We can understand code - and shrink it down to tiny bits and pieces.
We can run analysis on massive amounts of repositories.

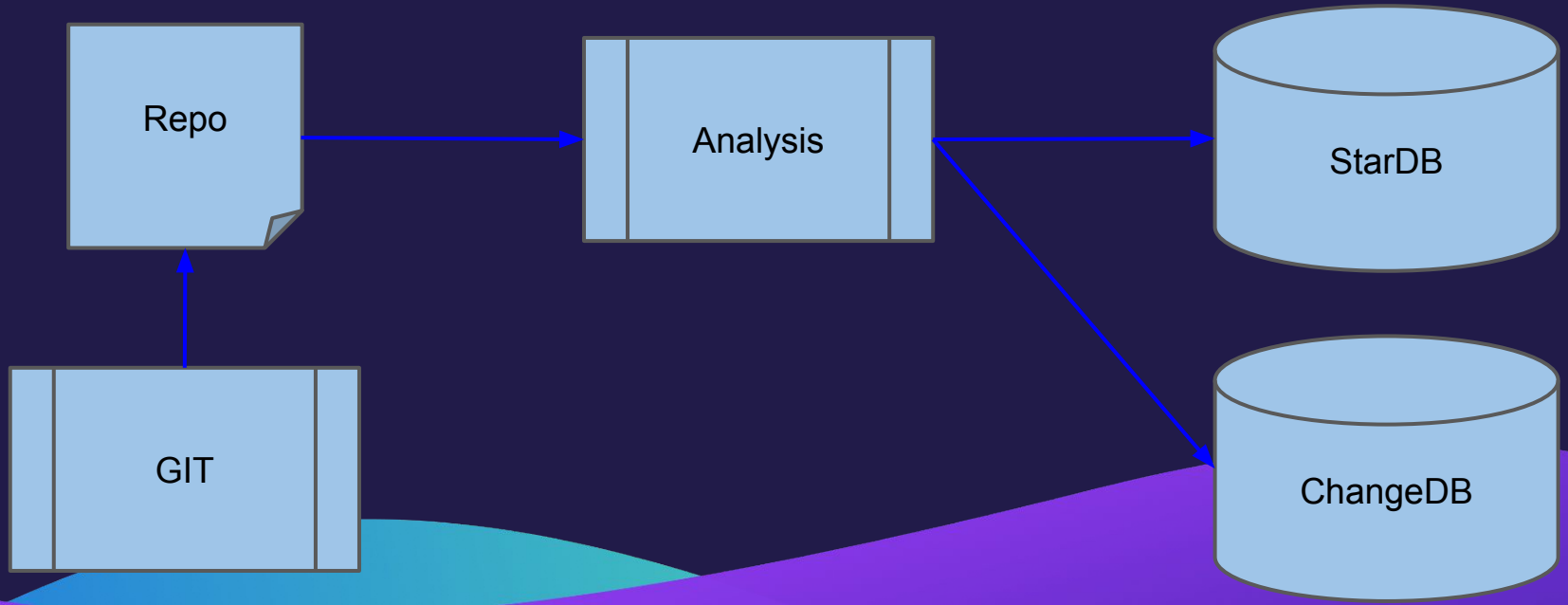


AI and Code Analysis

We can check for changes and identify fixes:

for vulnerabilities through PRs referencing a CVE.

because the vulnerable code can no longer be reached.



snyk

AI and Code Analysis

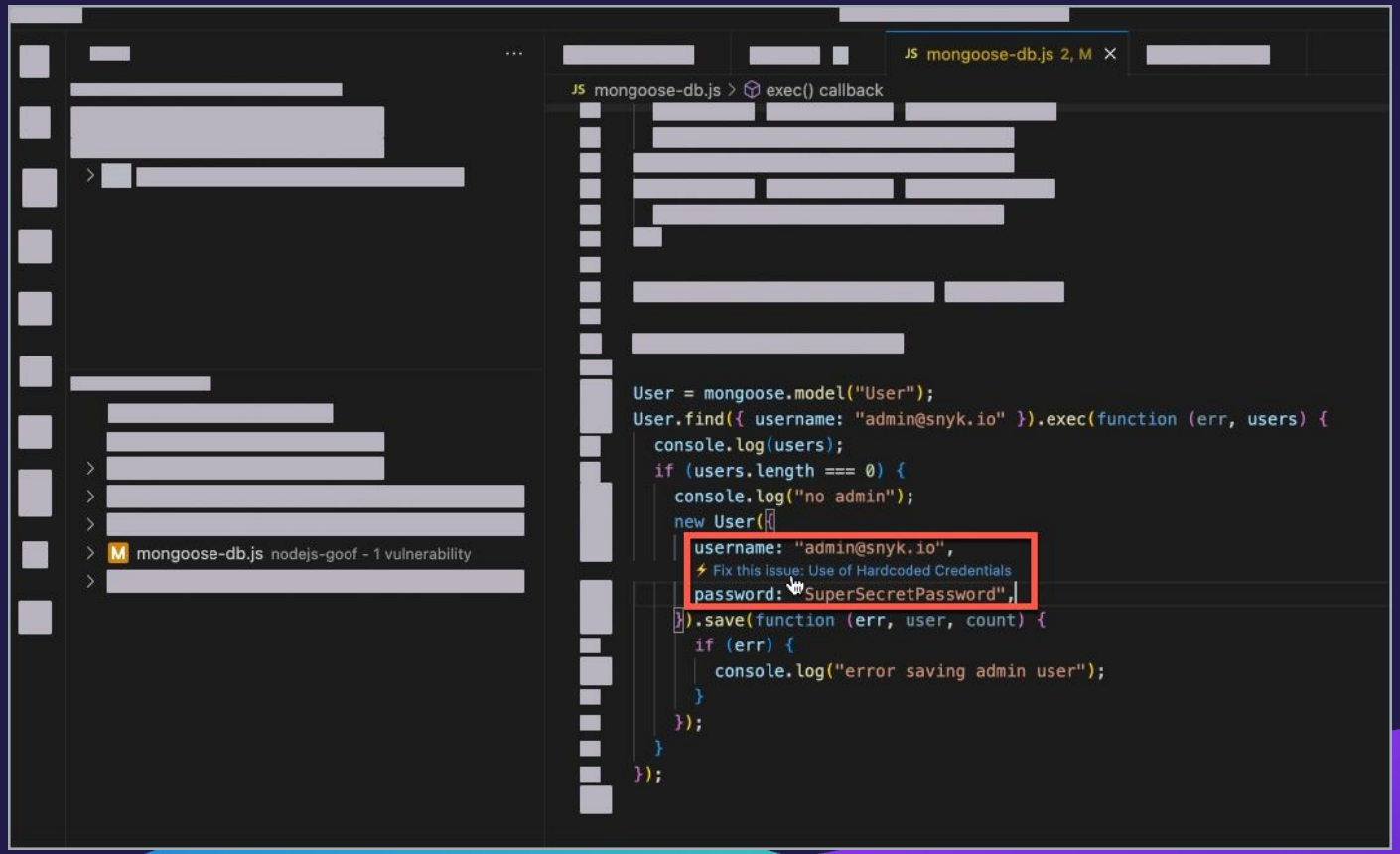
We can identify code that triggers vulnerabilities.
We can match that code against the identified fixes.
Apply the identified fix.

AI and Code Analysis

We can identify code that triggers vulnerabilities.
We can match that code against the identified fixes.
Apply the identified fix - automatically.

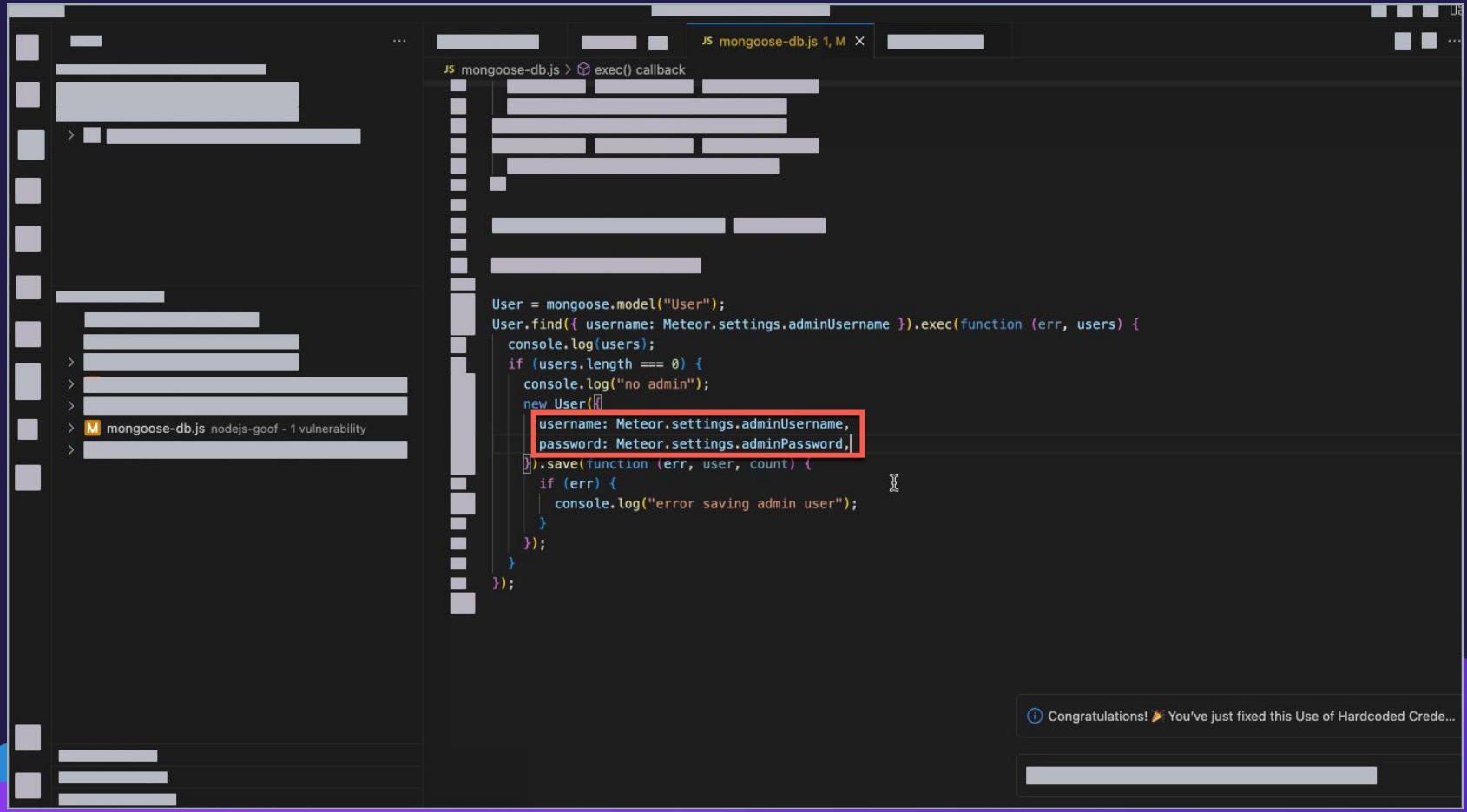
In the IDE the vulnerability will be highlighted and a fix will be offered.

AI and Code Analysis



snyk

AI and Code Analysis



Let’s step back a bit

Let’s step back a bit

Syntactical completion

- If I type a " then I will need another "

Let’s step back a bit

Syntactical completion

- If I type a " then I will need another "
- If I type a ' then I will need another '
- If I type a (then I will need a)
- If I type a [then I will need a]
- If I type a { then I will need a }

Let’s step back a bit

Syntactical completion

- If I type a " then I will need another "
- If I type a ' then I will need another '
- If I type a (then I will need a)
- If I type a [then I will need a]
- If I type a { then I will need a }

You either love it or hate it.

Let’s step back a bit

Contextual Completion

By taking the context into account you can do more.

- You can probably finish a line to some satisfaction.

Let’s step back a bit

Contextual Completion

By taking the context into account you can do more.

- You can probably finish a line to some satisfaction.
- If you apply AI you might get away with providing some strings.

Let’s step back a bit

Contextual Completion

By taking the context into account you can do more.

- You can probably finish a line to some satisfaction.
- If you apply AI you might get away with providing some strings.
  - Might work for simple predictable error message.
  - May even work for some constants.

Let’s step back a bit

Contextual Completion

By taking the context into account you can do more.

- You can probably finish a line to some satisfaction.
- If you apply AI you might get away with providing some strings.
  - Might work for simple predictable error message.
  - May even work for some constants.

You either love it or hate it.

Let’s step back a bit

Predictive Completion

By leveraging more AI you can get more reasonable content.

This is like writing a novel - just with more rules.

Let’s step back a bit

Predictive Completion

By leveraging more AI you can get more reasonable content.

This is like writing a novel - just with more rules.

You either love it or hate it.

Today

Semantic Completion

Put all of the pieces together.

- Apply syntax, context and semantic.

Today

Semantic AI Completion

Put all of the pieces together.

- Apply syntax, context and semantic.
- Provide multiple predictions.
- Rate the completions.

Today

Semantic AI Completion

Put all of the pieces together.

- Apply syntax, context and semantic.
- Provide multiple predictions.
- Rate the completions, using:
  - Parsers,
  - Compilers, and
  - other contextual evaluation.

Today

Semantic AI Completion

Put all of the pieces together.

- Apply syntax, context and semantic.
- Provide multiple predictions.
- Rate the completions, using:
  - Parsers,
  - Compilers, and
  - other contextual evaluation.

Even I like it - If it works.

Thanks

snyk.io/

Fix Code Vulnerabilities Automatically